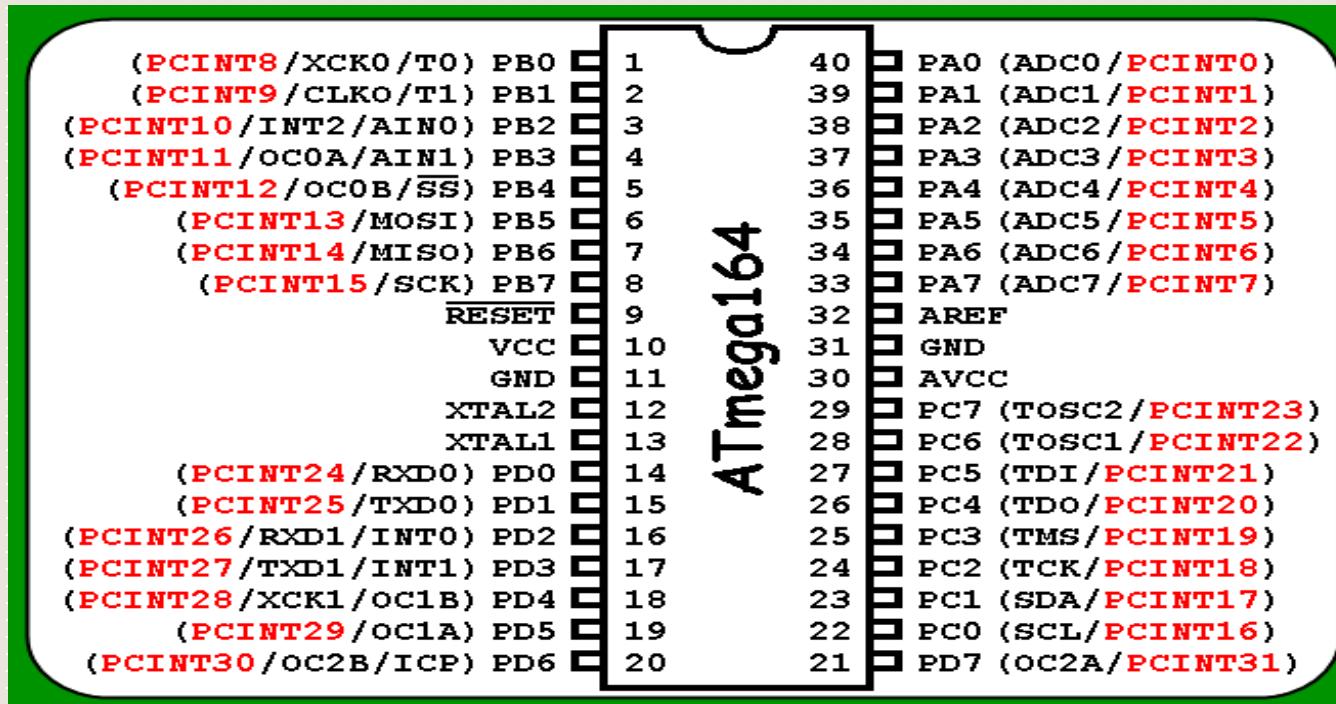


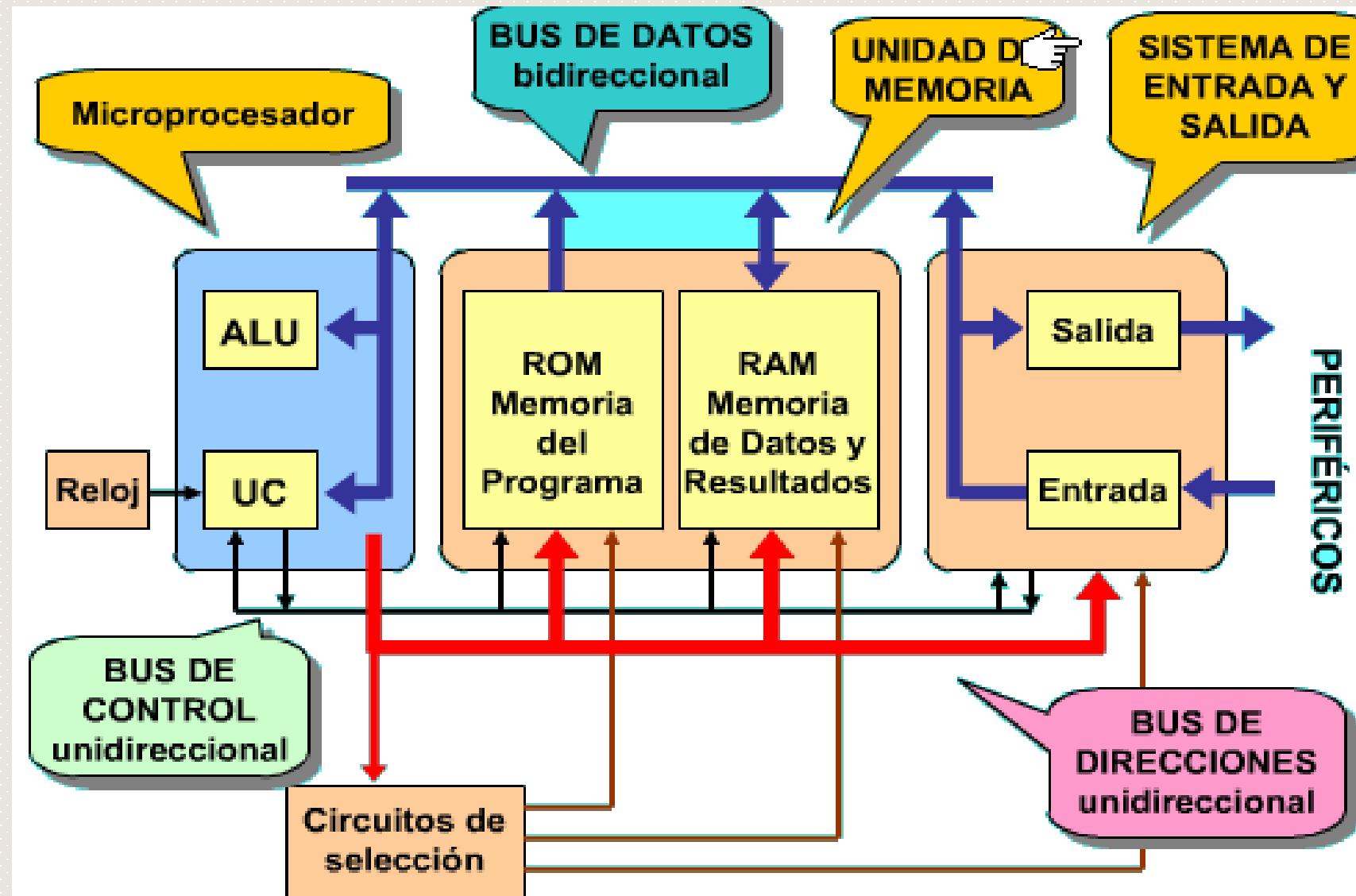
2º Clase

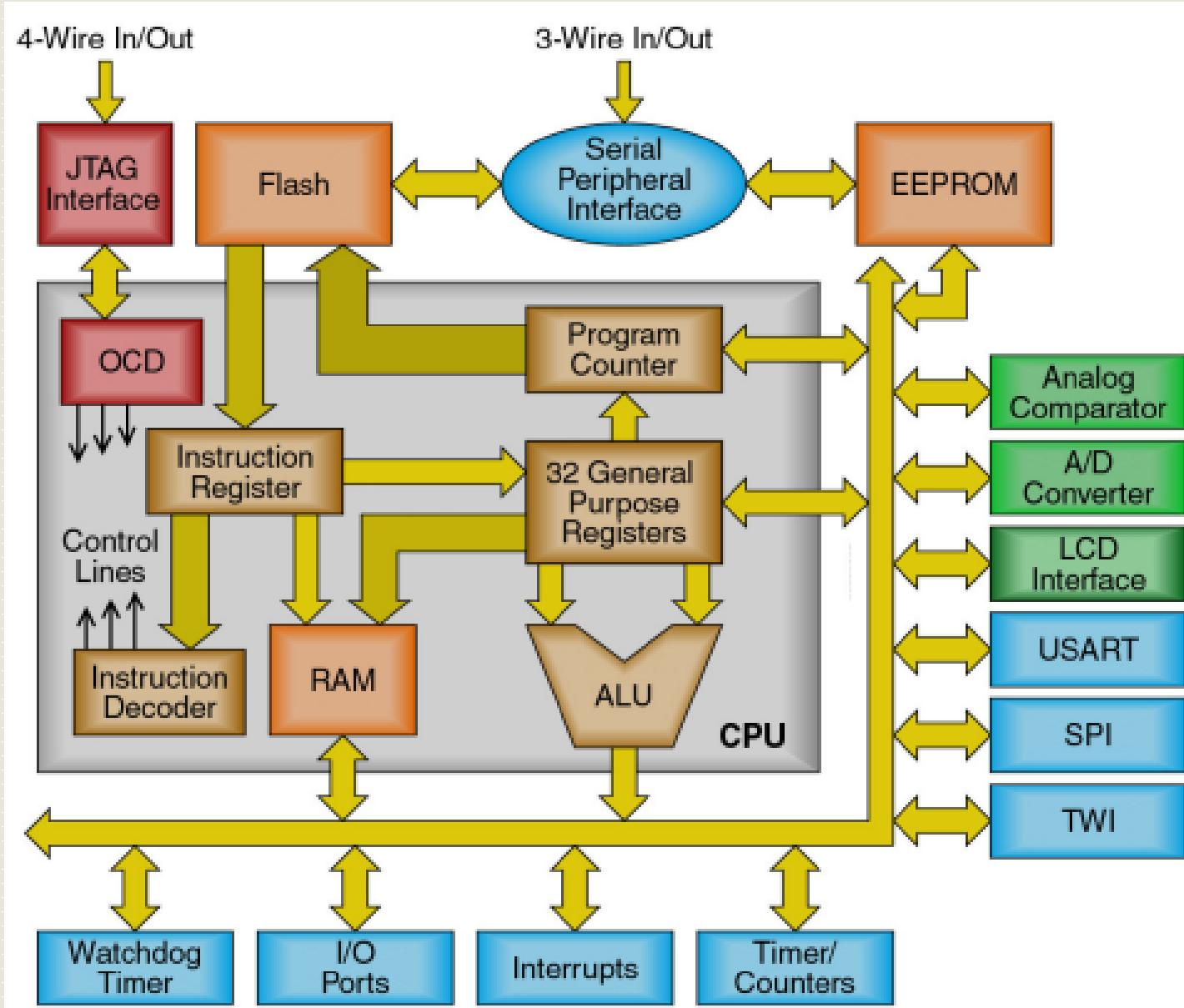
Presentación del micro controlador AVR

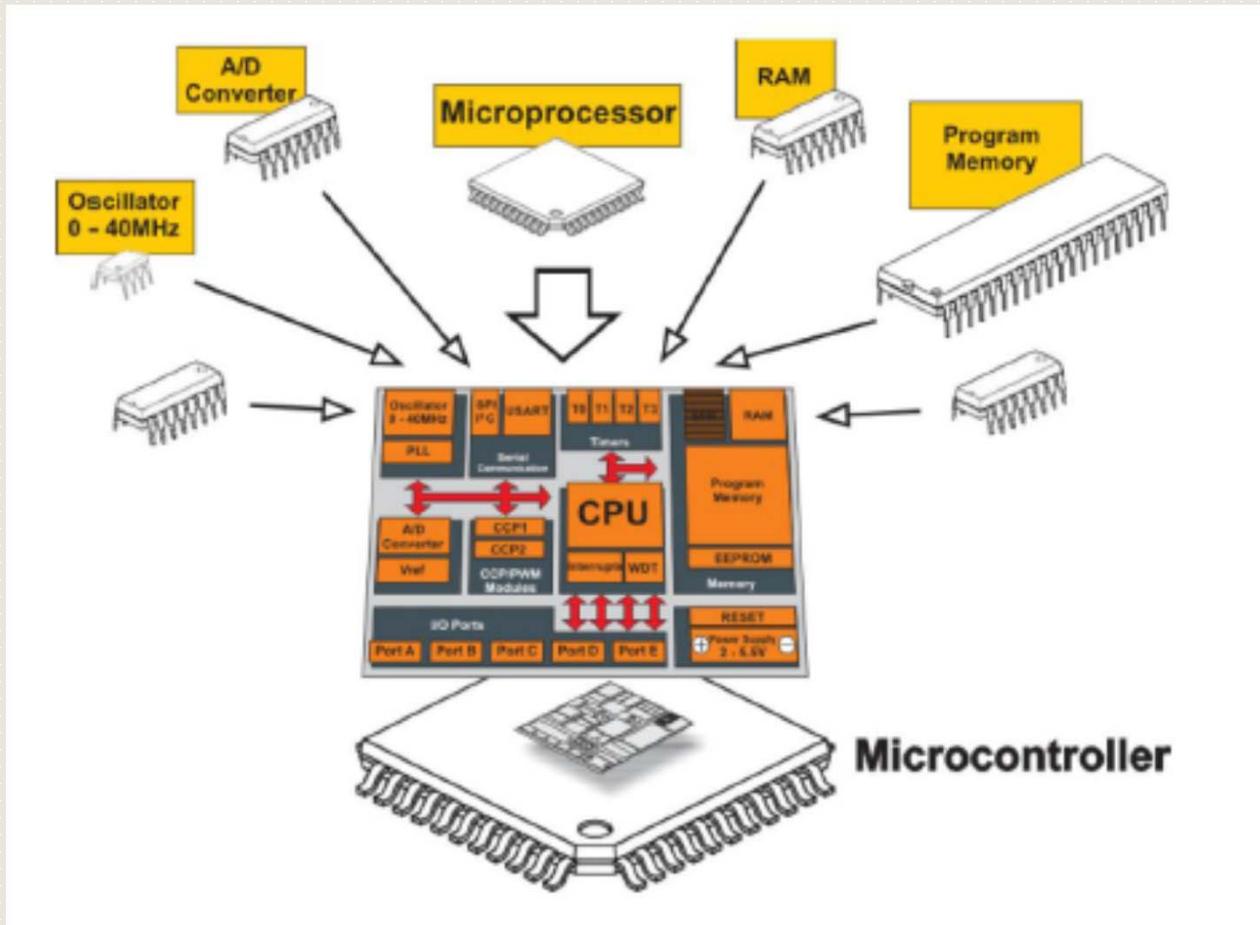


REPASO

Resumen

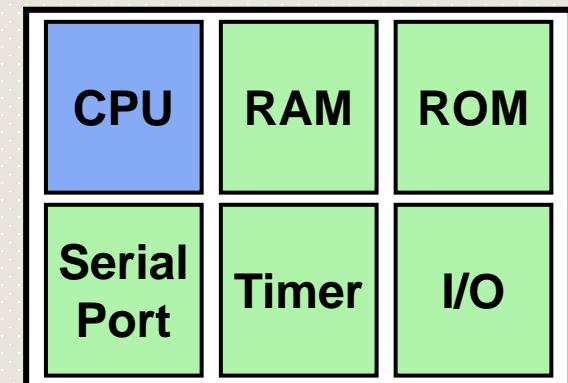






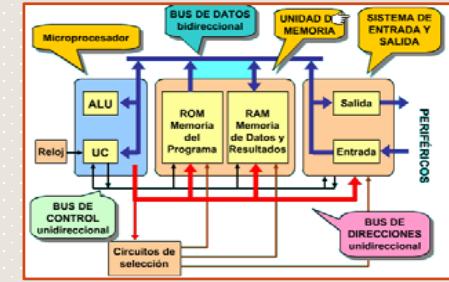
Most common microcontrollers

- 8-bit microcontrollers
 - AVR
 - PIC
 - HCS12
 - 8051
- 32-bit microcontrollers
 - ARM
 - PIC32
 - AVR 32



MICROCONTROLADORES AVR DE ATMEL (AHORA PARTE DE MICROCHIP)

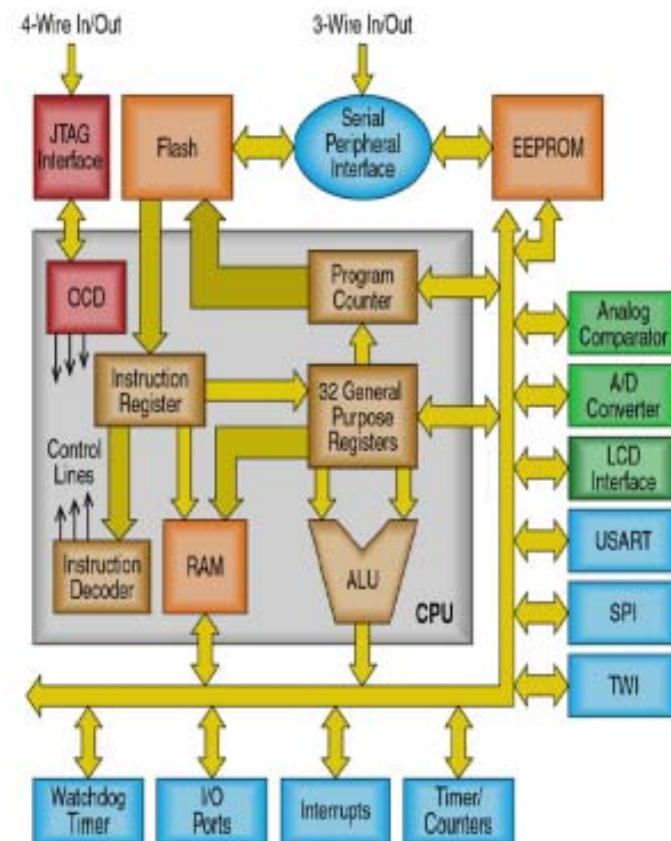
- Arquitetura **RISC** de 8 bits.
- Arquitectura **Harvard**, con memoria Flash para código, SRAM y EEPROM para datos.
- Arquitectura es **Cerrada**.
- Arquitectura es del tipo **Registro-Registro**.
- 32 registros de propósito general de 8 bits
- **Load and Store** Memoria de datos forman un solo espacio de localidades, que se acceden mediante operaciones de carga y de almacenamiento.
- “**Pipeline**” con dos etapas (traer y ejecutar),



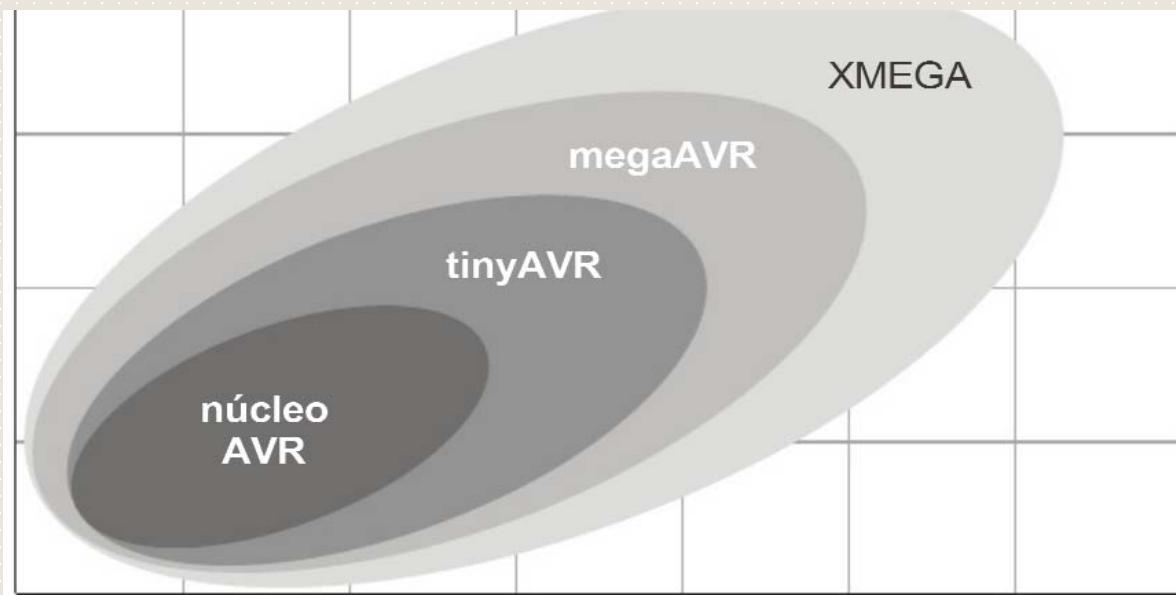
Arquitectura AVR

Arquitectura RISC con conjunto de instrucciones CISC

- Conjunto de Instrucciones poderoso para C y ensamblador
- Escalables
- Mismo Nucleo poderoso en todos los dispositivos AVR
- Ejecucion en un solo ciclo
- Una instruccion por reloj externo
- Consumo de potencia baja
- 32 registros de trabajo
- Todos conectados directamente a la ALU!
- Nucleo (core) muy eficiente
- 20 MIPS @ 20MHz
- Alto nivel de integracion
- Costo del sistema total bajo



los microcontroladores AVR son escalables, es decir, se puede comenzar las aplicaciones con microcontroladores AVR de pequeña capacidad y si se requiere incrementar las funciones de la aplicación, se puede emigrar a un microcontrolador AVR de mayor capacidad sin problema ya que la arquitectura y la forma en que se realiza la configuración y programación del Microcontrolador se mantiene igual.

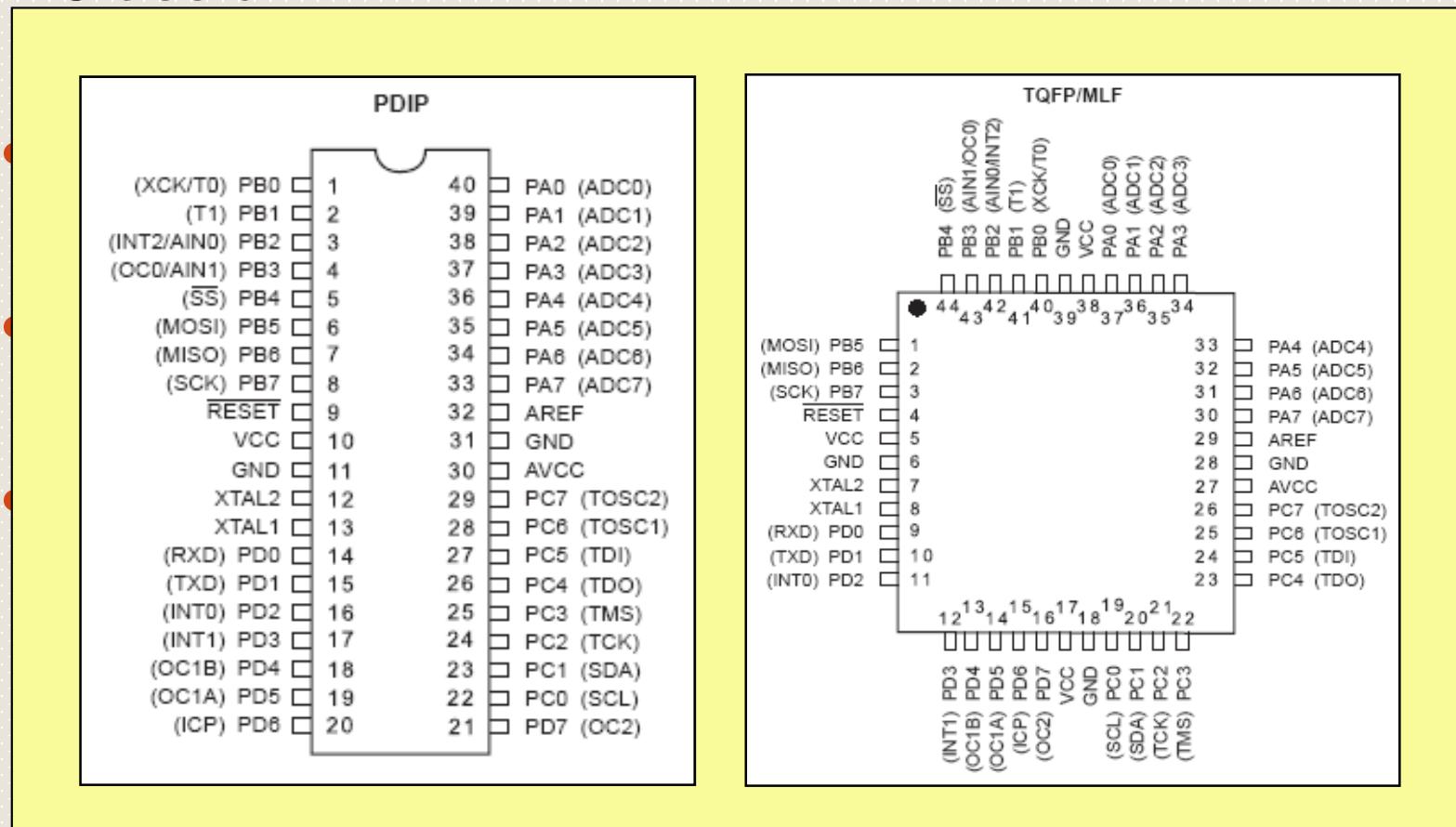


Son una familia de microcontroladores RISC fabricados por Atmel. La arquitectura fue concebida por dos estudiantes en el Norwegian Institute of Technology *; posteriormente refinada en Atmel Norway, la empresa subsidiaria de Atmel fundada por los dos arquitectos del chip.

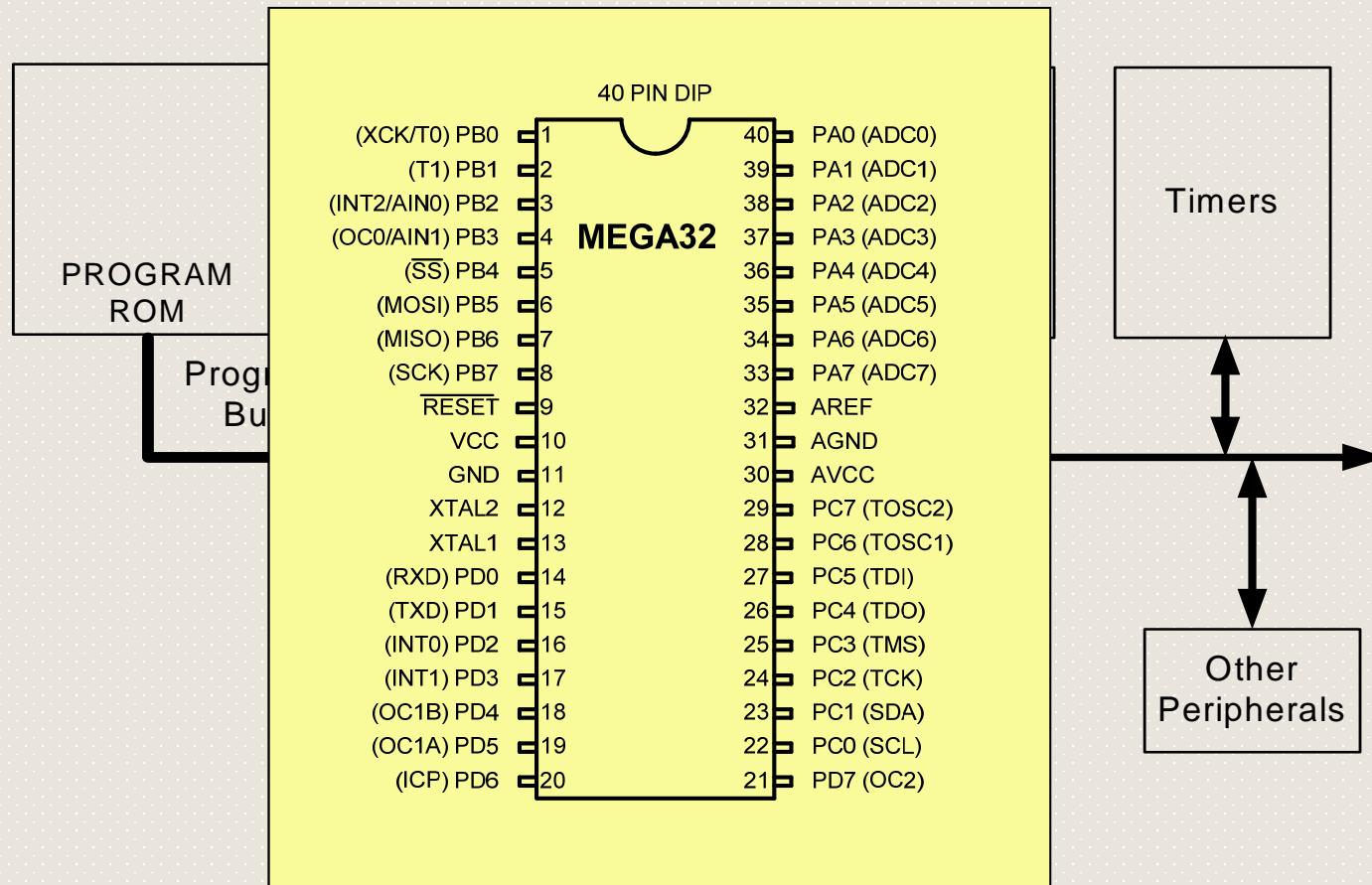
* [Universidad noruega de Ciencia](#)

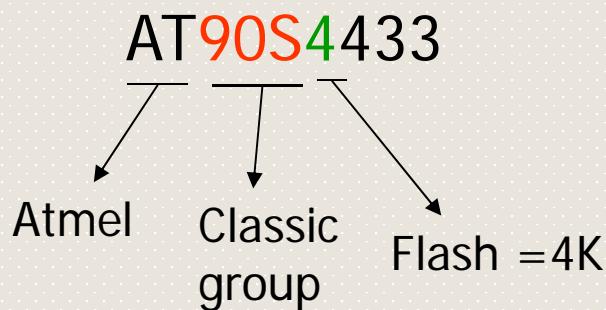
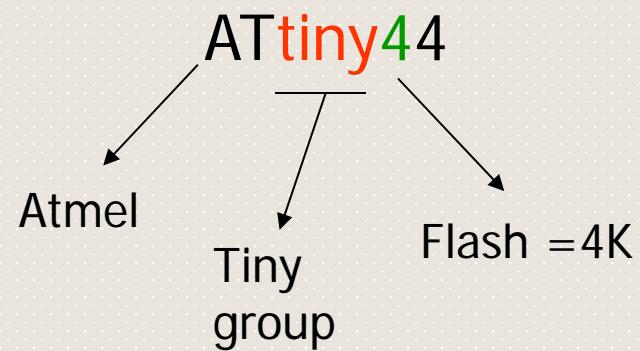
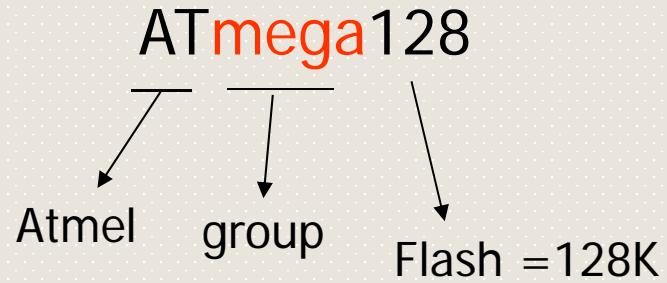
AVR different groups

- Classic AVR



AVR internal architecture





ATmega XXX X - XXX		
ad	Tipo de Empaquetado	Rango de Temperatura
Hz	A = TQFP ¹	C = Comercial
s	P = Plástico DIP ²	I = Industrial
	J = PLCC ³	
	M = MLF ⁴	

ENCAPSULADO PLCC ENCAPSULADO QFP

ENCAPSULADO DIP

ENCAPSULADO SOIC ENCAPSULADO TSOP

⁴ MLF significa Micro Lead Frame

Familia AVR

TINY AVR

Mega AVR

AVR

Con 1K byte Flash	Con 2K byte Flash	Con 4K byte Flash	Con 8K byte Flash	Con 12K byte Flash	Con 16K byte Flash	Con 32K/40K byte Flash	Con 64K byte Flash	Con 128K byte Flash	Con 256K byte Flash
Tiny13	Tiny14	Mega48	Mega8	90VC8544	Mega16	Mega32	Mega64	Mega128	Mega2560
	Tiny25	Tiny45	Mega8515		Mega162	Mega325	Mega645	Mega1280	Mega2561
	Tiny26		Mega8535		Mega169	Mega329	Mega649	Mega1281	
	Tiny2313		Mega88		Mega165	Mega406	Mega644		
					Mega168				

ATMega328

Flash (Kbytes)	32	TWI	Si
EEPROM (bytes)	1 K	ISP	Si
SRAM de propósito general (bytes)	2048	ADC de 10 bits (canales)	8 (6 en encapsulado PDIP)
Max Pines I/O	23	Comparador Analógico	Si
F.max (MHz)	20	Watchdog Timer	Si
Vcc (V)	2.7-5.5	Oscilador Interno	Si
16-bit Timers	1	Multiplicador por Hardware	Si
8-bit Timer	2	Interrupciones	26
PWM (canales)	6	Int. Externas	2
RTC	Si	Int. Por cambios en Pines	3 (una por puerto)
USART	1	Autoprogramación	Si
SPI (Maestro/Eslavo)	1	Debug wire	Si

ATMeg8

Flash (Kbytes)	8	TWI	Yes
EEPROM (Kbytes)	0.5	ISP	Yes
SRAM (Bytes)	1024	10-bit A/D (channels)	8
Max I/O Pins	23	Analog Comparator	Yes
F.max (MHz)	16	Brown Out Detector	Yes
Vcc (V)	2.7-5.5	Watchdog	Yes
16-bit Timers	1	On Chip Oscillator	Yes
8-bit Timer	2	Hardware Multiplier	Yes
PWM (channels)	3	Interrupts	18
RTC	Yes	Ext Interrupts	2
SPI	1	Self Program Memory	Yes
UART	1	Packages	MLF 32 PDIP 28 TQFP 32

CARACTERÍSTICAS (1)

DISPOSITIVO	MEMORIA		SRAM Bytes	I/O Max. Pins	FREC. Max. MHz	Vcc Rango V	TIMERS		PWM canales	RTC
	Flash Kbytes	EEPROM Kbytes					de 16-bit	de 8-bit		
ATmega48	4	0.25	512	23	20	1.8-5.5	1	2	6	Si
ATmega48 Automotive	4	0.25	512	23	16	2.7-5.5	1	2	6	Si
ATmega48P	4	0.25	512	23	20	1.8-5.5	1	2	6	Si
ATmega8	8	0.5	1024	23	16	2.7-5.5	1	2	3	Si
ATmega88 Automotive	8	0.5	1024	23	16	2.7-5.5	1	2	6	Si
ATmega88	8	0.5	1024	23	20	1.8-5.5	1	2	6	Si
ATmega88P	8	0.5	1024	23	20	1.8-5.5	1	2	6	Si
ATmega16	16	0.5	1024	32	16	2.7-5.5	1	2	4	Si
ATmega164P Automotive	16	0.5	1024	32	16	2.7-5.5	1	2	6	Si
ATmega164P	16	0.5	1024	32	20	1.8-5.5	1	2	6	Si
ATmega165	16	0.5	1024	54	16	1.8-5.5	1	2	4	Si
ATmega165P	16	0.5	1024	54	16	1.8-5.5	1	2	4	Si
ATmega32	32	1	2048	32	16	2.7-5.5	1	2	4	Si
ATmega324P Automotive	32	1	2048	32	16	2.7-5.5	1	2	6	Si
ATmega324P	32	1	2048	32	20	1.8-5.5	1	2	6	Si
ATmega325	32	1	2048	54	16	1.8-5.5	1	2	4	Si
ATmega64	64	2	4096	54	16	2.7-5.5	2	2	8	Si
ATmega644P Automotive	64	2	4096	32	16	2.7-5.5	1	2	6	Si
ATmega640	64	4	8192	86	16	1.8-5.5	4	2	16	Si
ATmega644	64	2	4096	32	20	1.8-5.5	1	2	6	Si
ATmega128	128	4	4096	53	16	2.7-5.5	2	2	8	Si
ATmega1280	128	4	8192	86	16	1.8-5.5	4	2	16	Si
ATmega1281	128	4	8192	54	16	1.8-5.5	4	2	9	Si
ATmega2560	256	4	8192	86	16	1.8-5.5	4	2	16	Si
ATmega2561	256	4	8192	54	16	1.8-5.5	4	2	9	Si

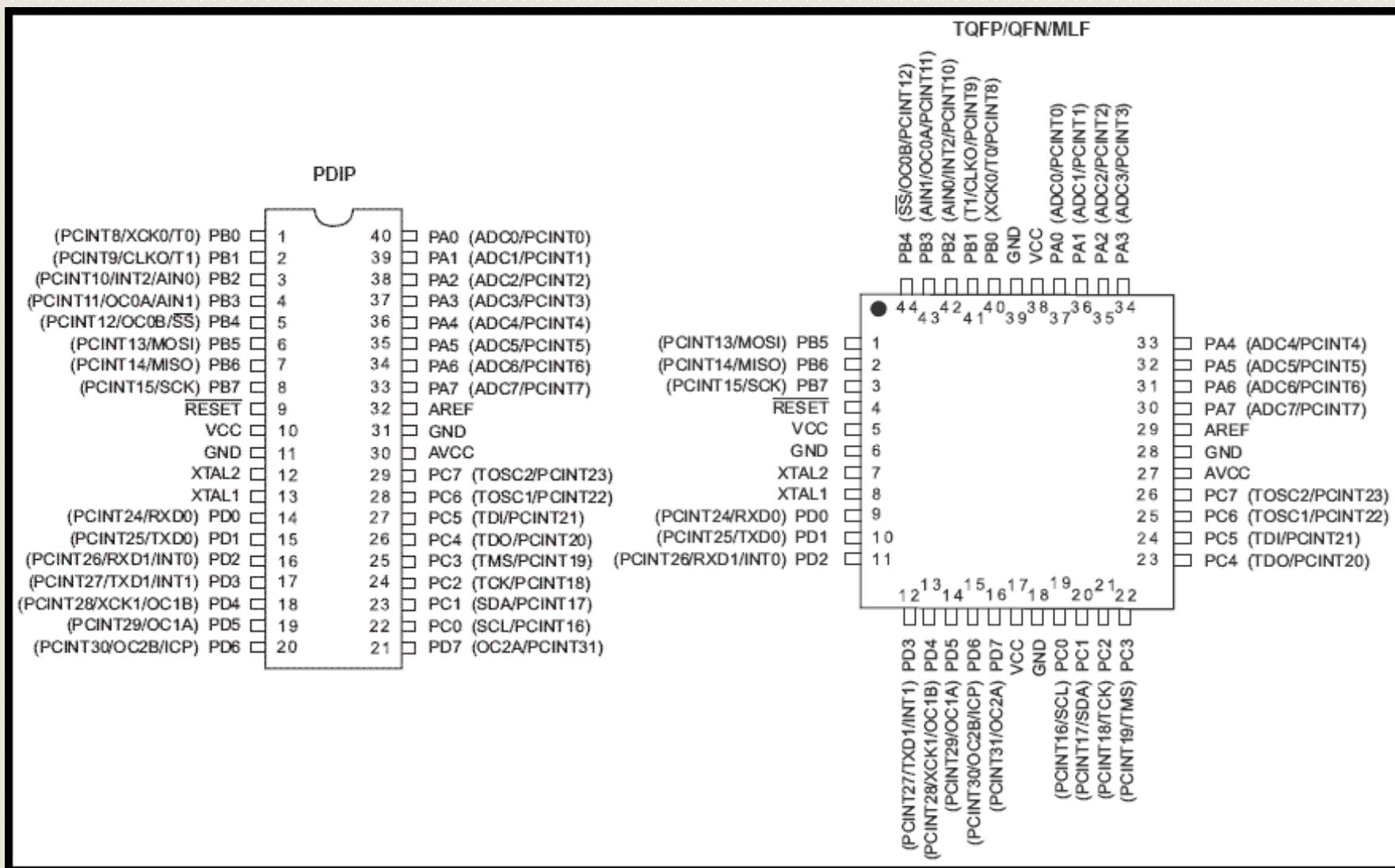
CARACTERÍSTICAS (2)

DISPOSITIVO	SPI	SERIALES			ANALÓGICOS		Detector de pérdida de VCC (Brown Out)	Aviso tipo Perro guardián (Watchdog)
		USART	TWI	ISP	10-bit A/D canales	Compa- rador		
ATmega48	1+USART	1	Si	Si	6 de 8	Si	Si	Si
ATmega48 Automotive	1+USART	1	Si	Si	8	Si	Si	Si
ATmega48P	1+USART	1	Si	Si	8	Si	Si	Si
ATmega8	1	1	Si	Si	6 de 8	Si	Si	Si
ATmega88 Automotive	1+USART	1	Si	Si	8	Si	Si	Si
ATmega88	1+USART	1	Si	Si	6 de 8	Si	Si	Si
ATmega88P	1+USART	1	Si	Si	8	Si	Si	Si
ATmega16	1	1	Si	Si	8	Si	Si	Si
ATmega164P Automotive	1+USART	2	Si	Si	8	Si	Si	Si
ATmega164P	1+USART	2	Si	Si	8	Si	Si	Si
ATmega165	1+USI	1	Si	Si	8	Si	Si	Si
ATmega165P	1+USI	1	Si	Si	8	Si	Si	Si
ATmega32	1	1	Si	Si	8	Si	Si	Si
ATmega324P Automotive	1+USART	2	Si	Si	8	Si	Si	Si
ATmega324P	1+USART	2	Si	Si	8	Si	Si	Si
ATmega325	1+USI	1	Si	Si	8	Si	Si	Si
ATmega64	1	2	Si	Si	8	Si	Si	Si
ATmega644P Automotive	1+USART	2	Si	Si	8	Si	Si	Si
ATmega640	1+USART	4	Si	Si	16	Si	Si	Si
ATmega644	1+USART	1	Si	Si	8	Si	Si	Si
ATmega128	1	2	Si	Si	8	Si	Si	Si
ATmega1280	1+USART	4	Si	Si	16	Si	Si	Si
ATmega1281	1+USART	2	Si	Si	8	Si	Si	Si
ATmega2560	1+USART	4	Si	Si	16	Si	Si	Si
ATmega2561	1+USART	2	Si	Si	8	Si	Si	Si

CARACTERÍSTICAS (3)

DISPOSITIVO	Oscillator en el mismo chip	Multiplicador por Hardware	INTERRUPCIONES		Capacidad de auto programación	Empaquetado
			Total	Externas		
ATmega48	Si	Si	26	26	Si	TQFP 32
ATmega48 Automotive	Si	Si	26	26	Si	QFN 32
ATmega48P	Si	Si	26	26	Si	TQFP 32
ATmega8	Si	Si	18	2	Si	TQFP 32
ATmega88 Automotive	Si	Si	26	26	Si	QFN 32
ATmega88	Si	Si	26	26	Si	TQFP 32
ATmega88P	Si	Si	26	26	Si	TQFP 32
ATmega16	Si	Si	20	3	Si	TQFP 44
ATmega164P Automotive	Si	Si	31	32	Si	QFN 44
ATmega164P	Si	Si	31	32	Si	TQFP 44
ATmega165	Si	Si	23	17	Si	TQFP 64
ATmega165P	Si	Si	23	17	Si	TQFP 64
ATmega32	Si	Si	19	3	Si	TQFP 44
ATmega324P Automotive	Si	Si	31	32	Si	QFN 44
ATmega324P	Si	Si	31	32	Si	TQFP 44
ATmega325	Si	Si	23	17	Si	TQFP 64
ATmega64	Si	Si	34	8	Si	TQFP 64
ATmega644P Automotive	Si	Si	31	32	Si	QFN 44
ATmega640	Si	Si	57	32	Si	TQFP 100 CBGA 100 TQFP 100
ATmega644	Si	Si	31	32	Si	TQFP 44
ATmega128	Si	Si	34	8	Si	TQFP 64
ATmega1280	Si	Si	57	32	Si	TQFP 100 CBGA 100 TQFP 100
ATmega1281	Si	Si	48	17	Si	TQFP 64 MLF 64 TQFP 64
ATmega2560	Si	Si	57	32	Si	CBGA 100 TQFP 100
ATmega2561	Si	Si	48	17	Si	TQFP 64

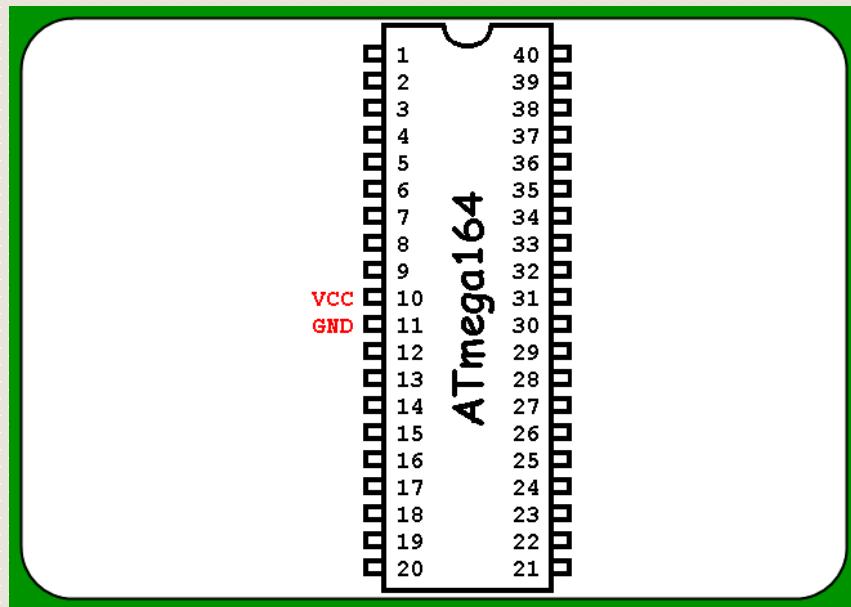
ENCAPSULADOS ATmega164



ATmega16 y el ATmega32

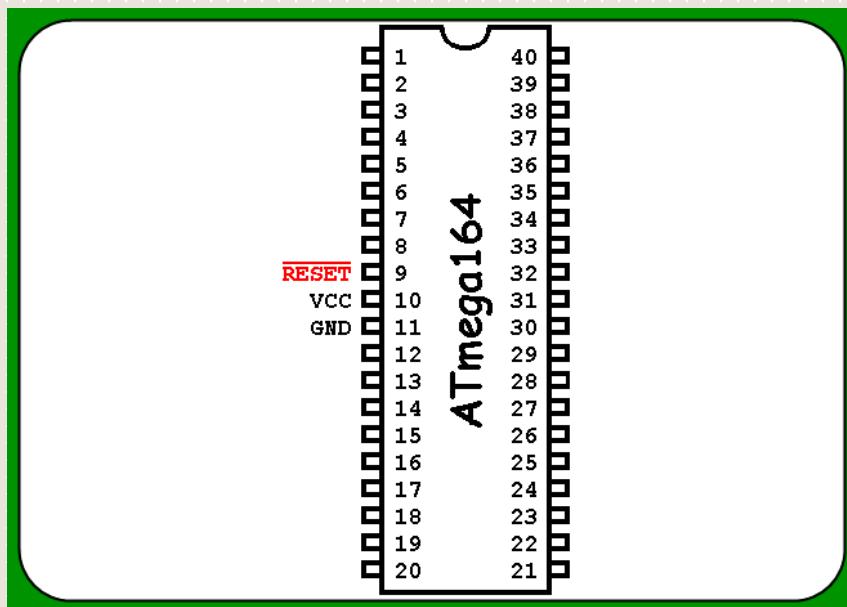
- Son microcontroladores de 8 bits de la familia MEGA
- Bajo consumo de energía
- Arquitectura RISC con 131 instrucciones en ensamblador
- la mayoría se ejecuta en un solo ciclo de instrucción.
- El ATmega 16 puede funcionar a una freq: máxima de 16 MHz.
- Este micro cuenta con 16 KB (32 para el caso del ATmega32) de memoria FLASH (de programa),
- 1 KB de memoria RAM estática
- 513 bytes de memoria EEPROM. Según el datasheet, la memoria FLASH puede ser reprogramada 10,000 vece y la EEPROM 100,000,
- El ATmega16/32 cuenta con 40 pines en empaquetado DIP, los cuales 32 son de E/S de propósito general divididos en 4 puertos de 8 bits cada uno, designados PORTA, PORTB, PORTC y PORTD.

Voltajes y pines de alimentación ATmega164)

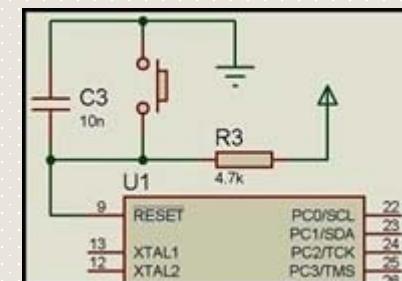


- 2.7 - 5.5V
- (1.8 - 5.5V
(ATmega164PV))
- 200 mA es la corriente máxima en los terminales VCC y GND

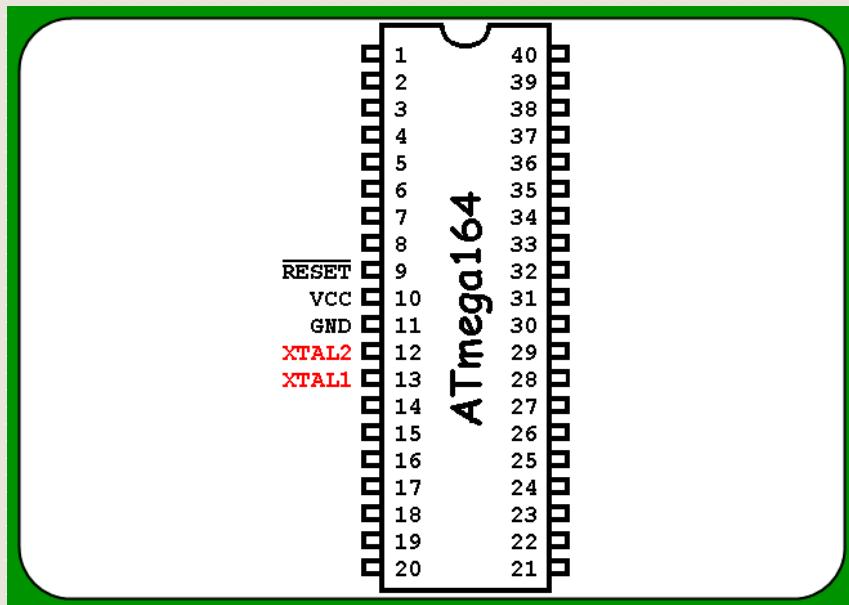
ENTRADA PARA EL RESET



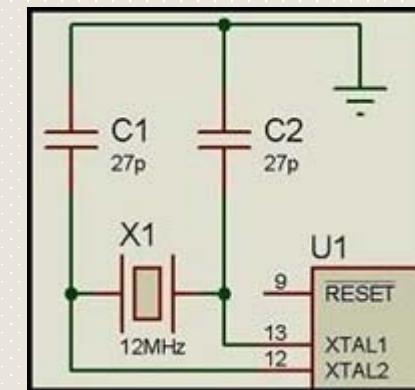
- Reset
- 1. Internos
- 2. Externo



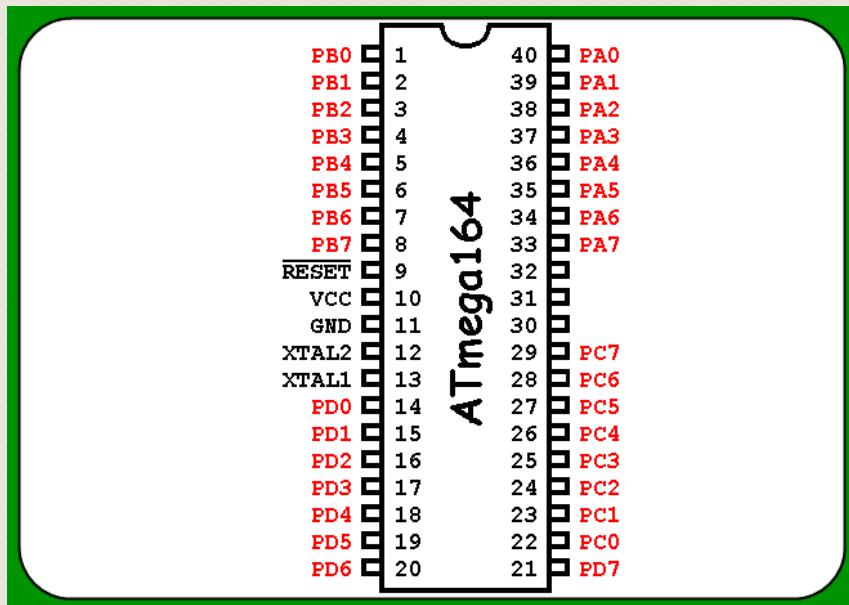
TERMINALES PARA EL CRISTAL



- Rangos de velocidad
 - 0 – 20 MHz
(ATmega164P)
 - 0 – 10 MHz
(ATmega164PV)

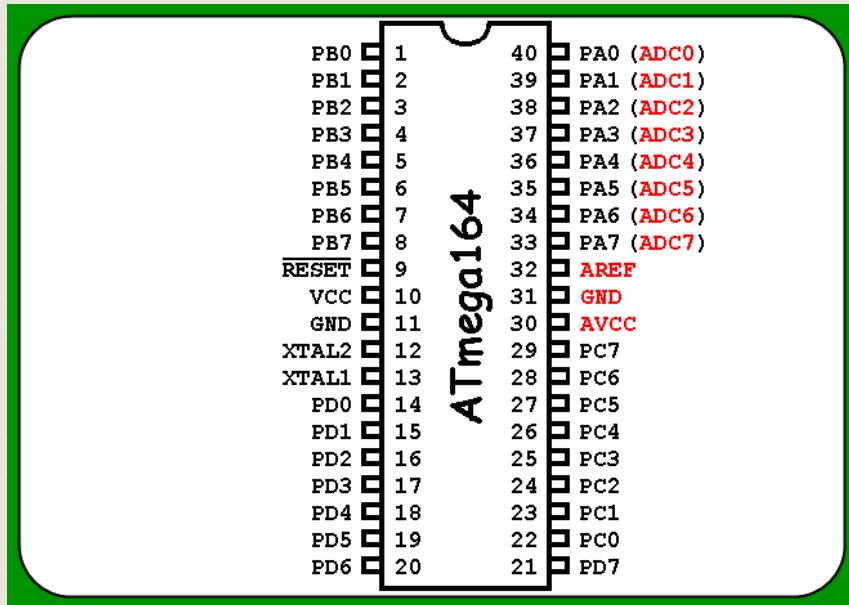


PÓRTICOS DE ENTRADA Y SALIDA PARALELA



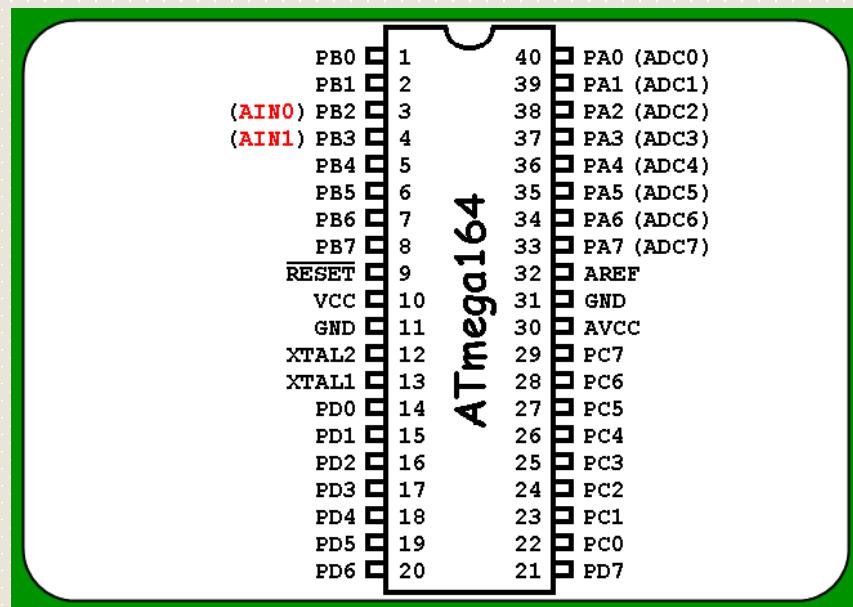
- I/O
 - 32 líneas de E/S programables
 - Puerto A (8 bits)
 - Puerto B (8 bits)
 - Puerto C (8 bits)
 - Puerto D (8 bits)

CONVERSOR DE ANALÓGICO A DIGITAL



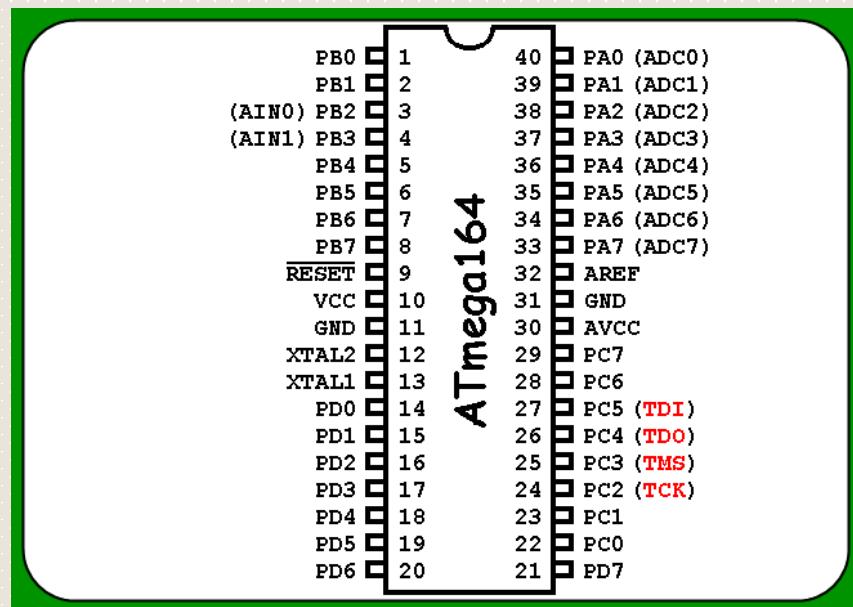
- ADC de 10 bits 8 canales
 - 8 canales de un solo terminal
 - 2 canales diferenciales con ganancia programable de x1, x10 y x200
 - 7 canales diferenciales sólo en el encapsulado TQFP

COMPARADOR ANALÓGICO



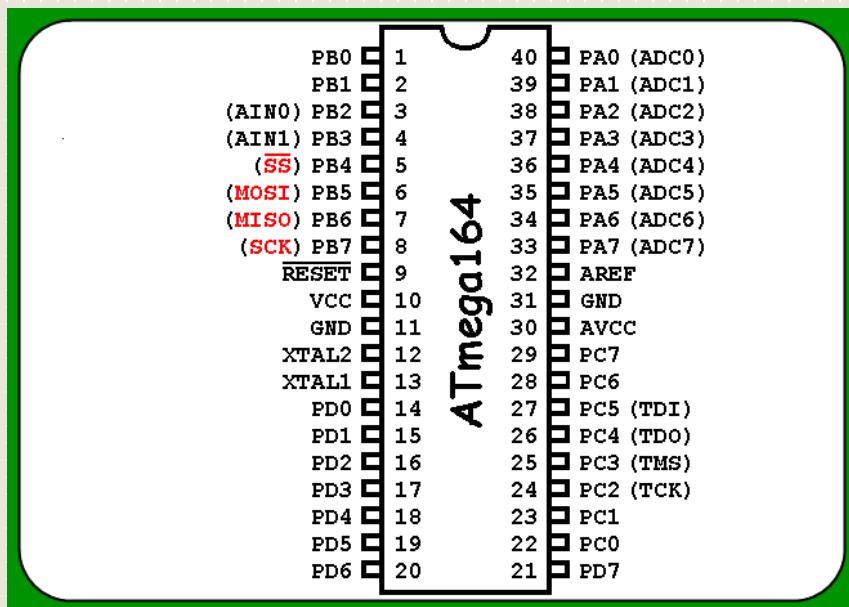
- Incorporado en el mismo chip
 - La entrada positiva es AIN0
 - La negativa es AIN1
 - Se puede reemplazar AIN1 por las entradas analógicas ADC0 .. ADC7

INTERFACE JTAG PARA SISTEMA DE DEPURACIÓN



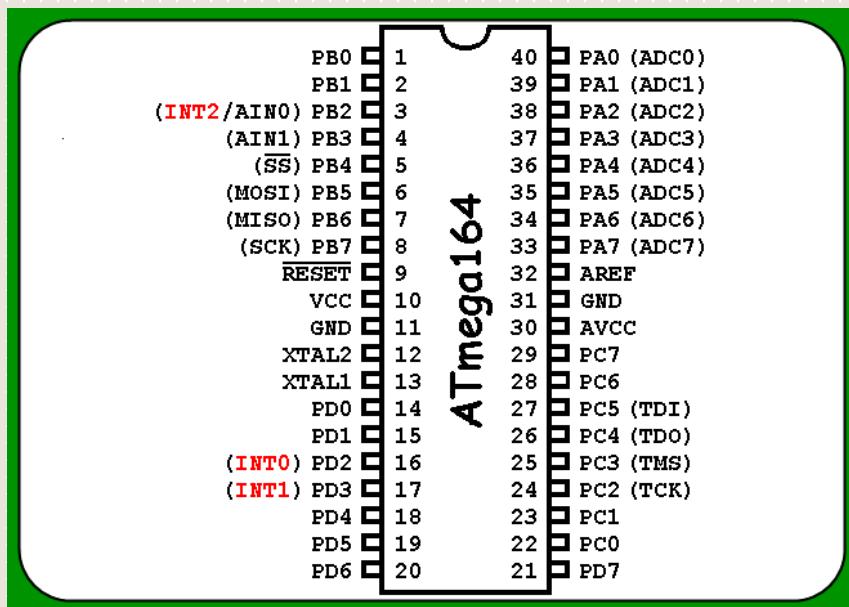
- JTAG (IEEE std1149.1)
 - En la depuración se tiene acceso a todos periféricos
 - Programación de la Flash, EEPROM, Fusibles y Bits de seguridad
 - Depuración soportada por el AVR Studio®

INTERFACE A PERIFÉRICOS SERIALES



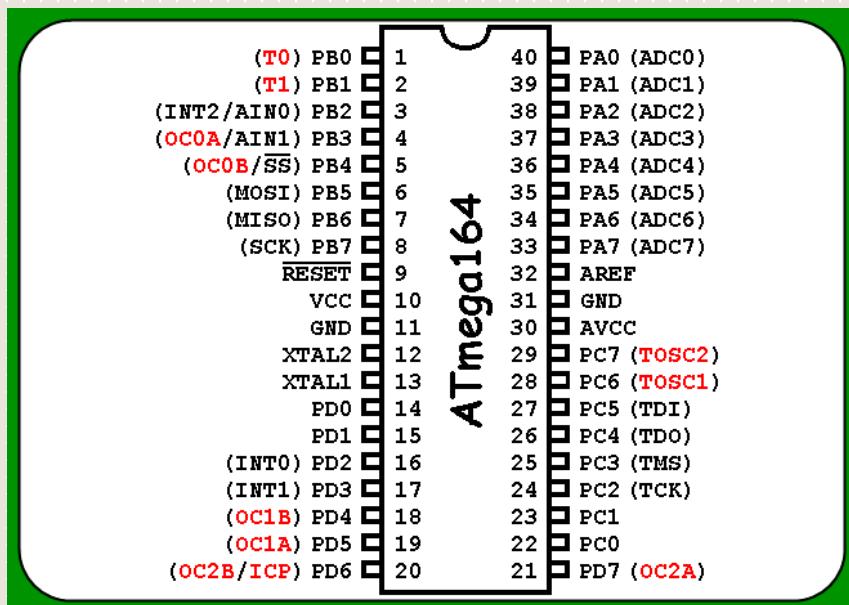
- SPI
 - Full duplex
 - Tres líneas para comunicaciones sincrónicas
- Operación maestro / esclavo
- Siete velocidades programables
- Bandera de fin de la transmisión

INTERRUPCIONES EXTERNAS



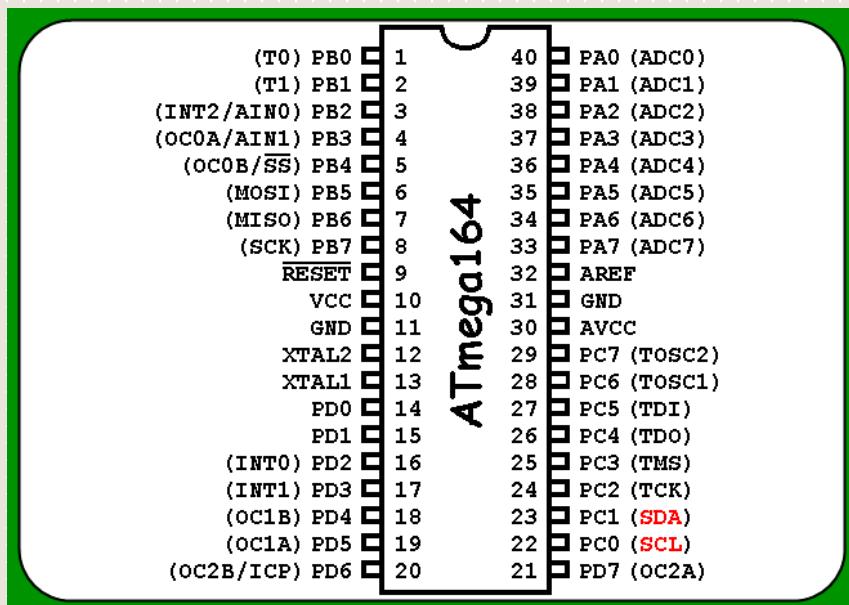
- INT0, INT1 e INT2
 - Pueden activarse por flanco de subida o de bajada, o por nivel de cero lógico
 - También se puede generar por software, si son configurados los terminales como salidas

TEMPORIZADORES / CONTADORES



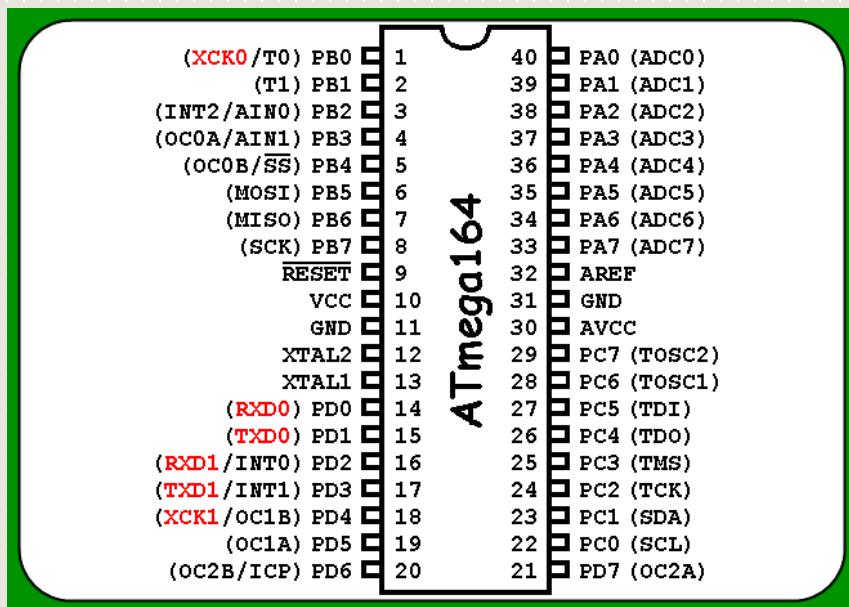
- Timer0 y Timer2 de 8 bits Timer1 de 16 bits
 - Dispone de unidades comparadoras
 - Sirven como Generadores de Frecuencias
- Poseen relojes pre escalables de 10 bits
- Permiten implementar Moduladores por Ancho del Pulso

INTERFACE SERIAL CON DOS LÍNEAS



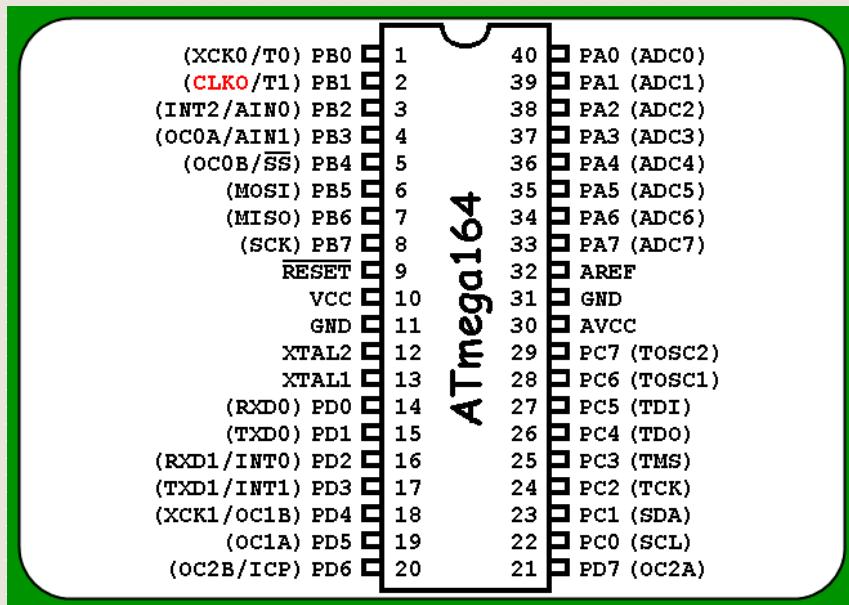
- TWI
 - Operación maestro / esclavo
 - Puede trabajar como transmisor o como receptor
 - Velocidad de transferencia hasta 400 KHz
 - Longitud de la dirección de 7 bits para 127 esclavos

RECEPTORES / TRANSMISORES UNIVERSALES SINCRÓNICOS Y ASINCRÓNICOS



- USART0 y USART1
 - Full duplex
 - Velocidad de alta resolución
 - Tramas de 5, 6, 7, 8 o 9 bits, con 1 o 2 bits de parada
 - Detector de errores de velocidad y en la trama
 - Operación de maestro o esclavo en comunicaciones sincrónicas

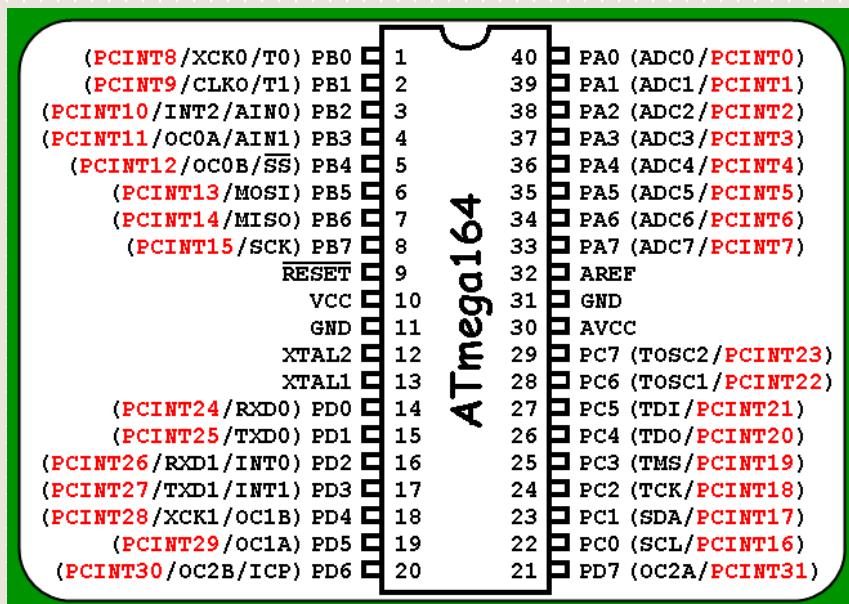
SALIDA DEL RELOJ



• CLOCK

- Habilitación de la señal programando el fusible
- Incluye como fuente al oscilador interno RC
- Se puede utilizar el sistema pre escalable para realizar la división del reloj

INTERRUPCIONES POR CAMBIO DE ESTADO

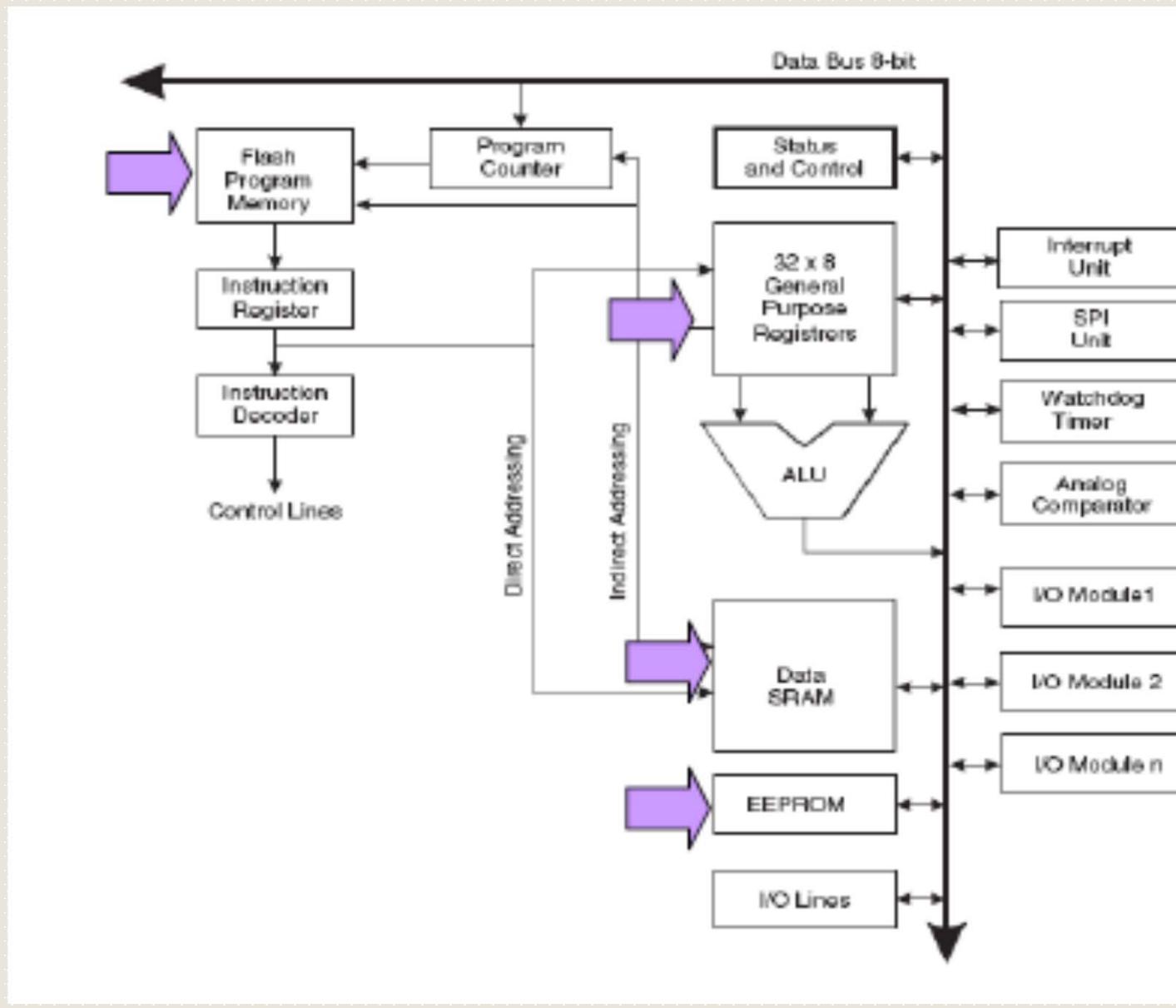


- INTERRUPCIONES EXTERNAS ADICIONALES

- Cambios entre PCINT0 y PCINT7 se registra en PCI0
- Cambios entre PCINT8 y PCINT15 se registra en PCI1
- Cambios entre PCINT16 y PCINT23 se registra en PCI2
- Cambios entre PCINT24 y PCINT31 se registra en PCI3

- 1 .Organización de memoria**
- 2 .Resets**
- 3 .Puertos**

Mapa de Memoria

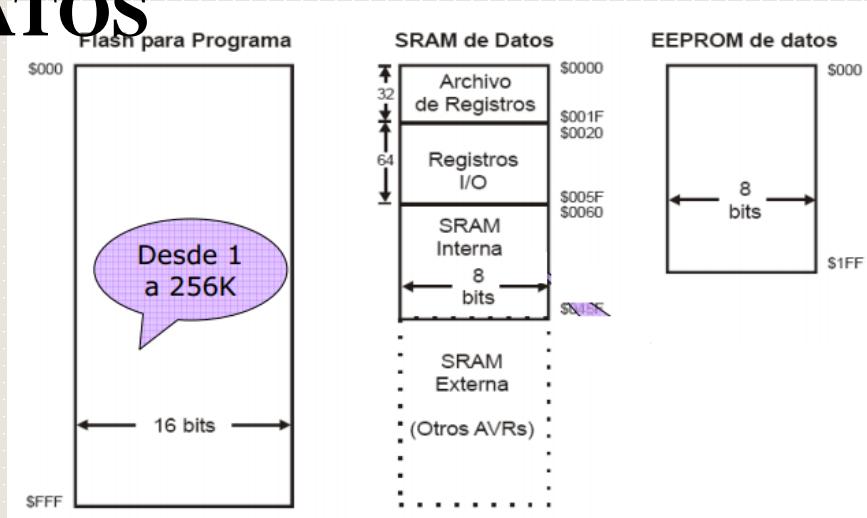


- **1280 BYTES DE MEMORIA RAM ESTÁTICA (SRAM) PARA DATOS**
- **16 KBYTES DE MEMORIA FLASH PARA EL PROGRAMA**
- **512 BYTES DE EEPROM (RAM NO VOLÁTIL), TAMBIÉN PARA DATOS**

MAPA DE MEMORIA

Atmega 164At

37



¿Pensamos en el proyecto?

- ¿Que hicieron mis compañeros?
- 1º
- 2º
- 3º

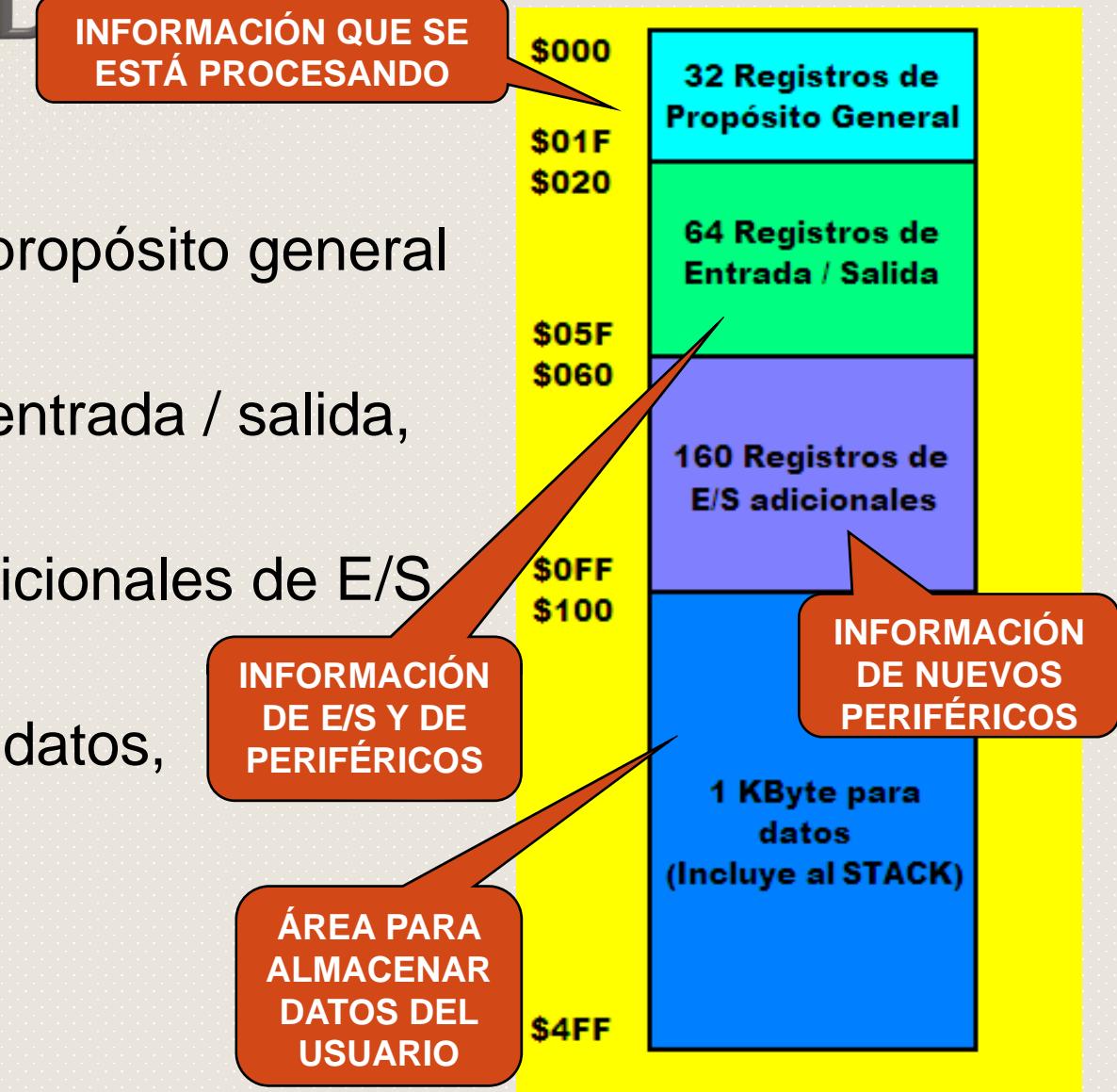


Memoria de datos

MAPA MEMORIA de DATOS 1280 BYTES DE LA SRAM

1. 32 registros de propósito general
2. 64 registros de entrada / salida,
1. 160 registros adicionales de E/S
2. 1024 byte para datos,

\$000 → \$4FF



32 REGISTROS DE PROPÓSITO GENERAL



1. Los registros están identificados entre R0 y R31

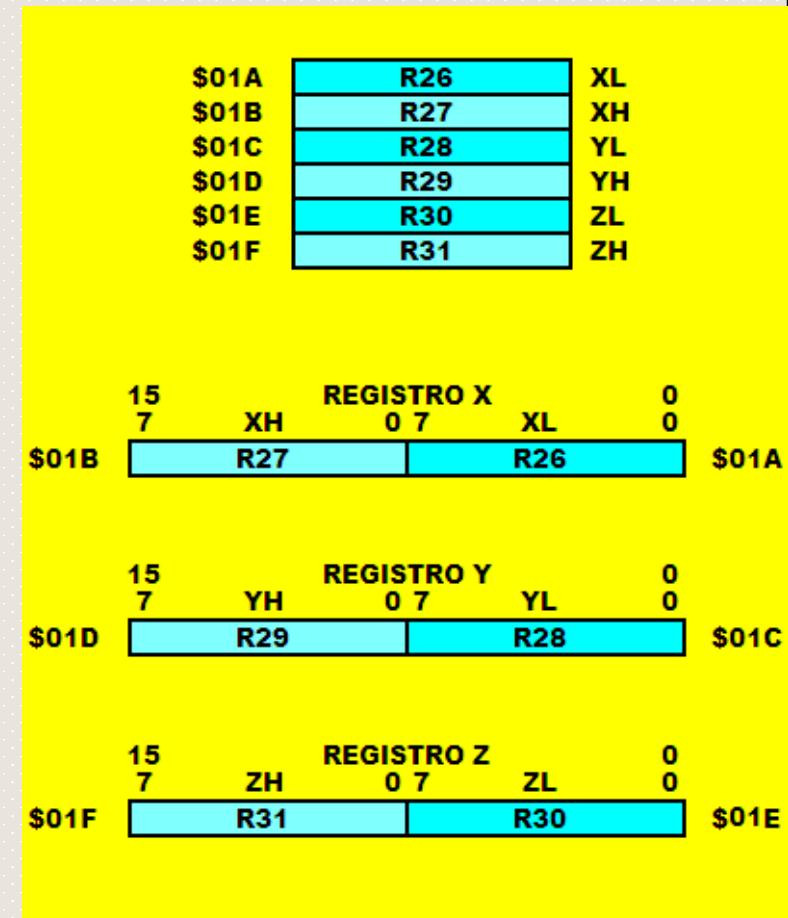
2. Los seis últimos
Pueden ser usados como punteros

\$000	R0
\$001	R1
\$002	R2
\$003	R3
\$004	R4
\$005	R5
\$006	R6
\$007	R7
\$008	R8
\$009	R9
\$00A	R10
\$00B	R11
\$00C	
\$00D	
\$00E	
\$00F	R15
\$010	R16
\$011	
\$012	
\$013	
\$014	
\$015	R25
\$016	R26 ←
\$017	R27 ←
\$018	R28 ←
\$019	R29 ←
\$01A	R30 ←
\$01B	R31 ←

REGISTROS PUNTEROS

Los tres punteros de 16 bits
x, y , z

Permiten acceder a los contenidos de los espacios de memoria utilizando el direccionamiento indirecto



Algunas instrucciones sencillas

Operation:
(i) $Rd \leftarrow Rr$

Syntax:
(i) MOV Rd,Rr

16-bit Opcode:

0010	11rd	dddd	rrrr
------	------	------	------

Program Counter:
 $PC \leftarrow PC + 1$

Operands:
 $0 \leq d \leq 31, 0 \leq r \leq 31$

\$000	R0
\$001	R1
\$002	R2
\$003	R3
\$004	R4
\$005	R5
\$006	R6
\$007	R7
\$008	R8
\$009	R9
\$00A	R10
\$00B	R11
\$00C	
\$00D	
\$00E	
\$00F	R15
\$010	R16
\$011	
\$012	
\$013	
\$014	
\$015	
\$016	
\$017	R25
\$018	R26
\$019	R27
\$01A	R28
\$01B	R29
\$01C	R30
\$01D	R31



Algunas instrucciones sencillas

Example:

```
    mov    r1,r0      ; Copy r0 to r1
    jmp    farplc    ; Unconditional jump
    ...
farplc: nop           ; Jump destination (do nothing)
```

Words: 2 (4 bytes)

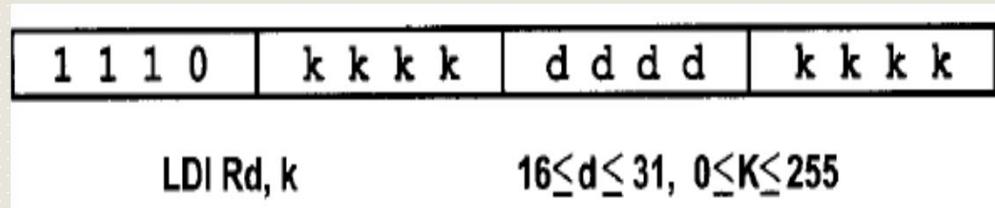
Cycles: 3

Algunas instrucciones sencillas

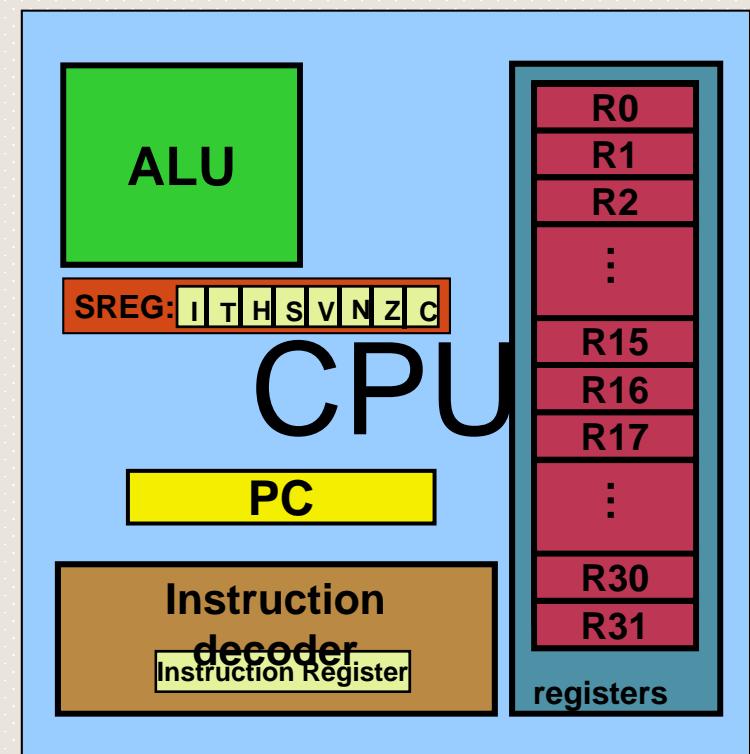
Cargando los valores en los registros de propósito general

LDI (Load Immediate)

- LDI Rd, k



- Ejemplo :
- LDI R16,53 ;(R16) = 53**
- LDI R19,132 ;(R19)= 132**
- LDI R23,0x27 ;(R23) = 0x27**



Algunas instrucciones sencillas . cálculo aritmético

ADD, SUB, MUL, AND, etc.

0 0 0 0	1 1 r d	d d d d	r r r r
ADD Rd,Rr		$0 \leq d \leq 31, 0 \leq r \leq 31$	

- ADD Rd,Rs ; suma de dos Reg.
 - $Rd = Rd + Rs$
 - Ejemplo :
 - ADD R25, R9 ;(R25)= R25 + R9
 - ADD R17,R30 ;(R17) = R17 + R30

Ejemplo Suma

SUMA

$19 + 95$

LDI R16, 19 ; (R16) = 19

LDI R20, 95 ; (R20) = 95

ADD R16, R20 ; (R16) = (R16) + (R20)

\$000	R0
\$001	R1
\$002	R2
\$003	R3
\$004	R4
\$005	R5
\$006	R6
\$007	R7
\$008	R8
\$009	R9
\$00A	R10
\$00B	R11
\$00F	R15
\$010	R16
\$019	R25
\$01A	R26 ←
\$01B	R27 ←
\$01C	R28 ←
\$01D	R29 ←
\$01E	R30 ←
\$01F	R31 ←

programas simples

- Scribe un programa que realice la suma de $19 + 95 + 5$

```
LDI R16, 19 ;(R16) = 19
```

```
LDI R20, 95 ;(R20) = 95
```

```
LDI R21, 5 ;(R21) = 5
```

```
ADD R16, R20 ;(R16) =(R16)+(R20)
```

```
ADD R16, R21 ;(R16) =(R16)+(R21)
```

Ejemplo Resta

- SUB Rd,Rs
 - $Rd = Rd - Rs$
- Example:
 - SUB R25, R9
 - $R25 = R25 - R9$
 - SUB R17,R30
 - $R17 = R17 - R30$

Algunas instrucciones para cálculo aritmético

Operation:
(i) $Rd \leftarrow Rd + 1$ Program Counter:
 $PC \leftarrow PC + 1$

Syntax:
(i) INC Rd Operands:
16-bit Opcode: $0 \leq d \leq 31$

1001	010d	dddd	0011
------	------	------	------

INC Rd

; $(Rd) = Rd + 1$

Ejemplo:

INC R25

; $(R25) = R25 + 1$

Algunas instrucciones para cálculo aritmético

Operation: Program Counter:
(i) $Rd \leftarrow Rd - 1$ $PC \leftarrow PC + 1$

Syntax: Operands:
(i) DEC Rd $0 \leq d \leq 31$

16-bit Opcode:

1001	010d	dddd	1010
------	------	------	------

- **DEC Rd**

$$;(Rd) = (Rd) - 1$$

- Ejemplo:

- **DEC R23**

$$(R23) = R23 - 1$$

64 REGISTROS DE E/S

DIRECCIÓN
COMO E/S



1. Los puertos de entrada / salida y los periféricos están ubicados en esta área.
2. Si se acceden como localidades de memoria, las direcciones son entre \$020 y \$05F.
3. Si se acceden como registros de e/s deben ser entre \$000 y \$03F.



I/O registros

Address		Name
Mem.	I/O	
\$20	\$00	TWBR
\$21	\$01	TWSR
\$22	\$02	TWAR
\$23	\$03	TWDR
\$24	\$04	ADCL
\$25	\$05	ADCH
\$26	\$06	ADCSRA
\$27	\$07	ADMUX
\$28	\$08	ACSR
\$29	\$09	UBRRL
\$2A	\$0A	UCSRB
\$2B	\$0B	UCSRA
\$2C	\$0C	UDR
\$2D	\$0D	SPCR
\$2E	\$0E	SPSR
\$2F	\$0F	SPDR
\$30	\$10	PIND
\$31	\$11	DDRD
\$32	\$12	PORTD
\$33	\$13	PINC
\$34	\$14	DDRC
\$35	\$15	PORTC

Address		Name
Mem.	I/O	
\$36	\$16	PINB
\$37	\$17	DDRB
\$38	\$18	PORTB
\$39	\$19	PINA
\$3A	\$1A	DDRA
\$3B	\$1B	PORTA
\$3C	\$1C	EECR
\$3D	\$1D	EEDR
\$3E	\$1E	EEARL
\$3F	\$1F	EEARH
\$40	\$20	UBRRC
		UBRRH
\$41	\$21	WDTCR
\$42	\$22	ASSR
\$43	\$23	OCR2
\$44	\$24	TCNT2
\$45	\$25	TCCR2
\$46	\$26	ICR1L
\$47	\$27	ICR1H
\$48	\$28	OCR1BL
\$49	\$29	OCR1BH
\$4A	\$2A	OCR1AL

Address		Name
Mem.	I/O	
\$4B	\$2B	OCR1AH
\$4C	\$2C	TCNT1L
\$4D	\$2D	TCNT1H
\$4E	\$2E	TCCR1B
\$4F	\$2F	TCCR1A
\$50	\$30	SFIOR
\$51	\$31	OCDR
		OSCCAL
\$52	\$32	TCNT0
\$53	\$33	TCCR0
\$54	\$34	MCUCSR
\$55	\$35	MCUCR
\$56	\$36	TWCR
\$57	\$37	SPMCR
\$58	\$38	TIFR
\$59	\$39	TIMSK
\$5A	\$3A	GIFR
\$5B	\$3B	GICR
\$5C	\$3C	OCR0
\$5D	\$3D	SPL
\$5E	\$3E	SPH
\$5F	\$3F	SREG



OUT – Store Register to I/O Location

Description:

Stores data from register Rr in the register file to I/O Space
(Ports, Timers, Configuration registers etc.).

Operation:
(i) $I/O(A) \leftarrow Rr$

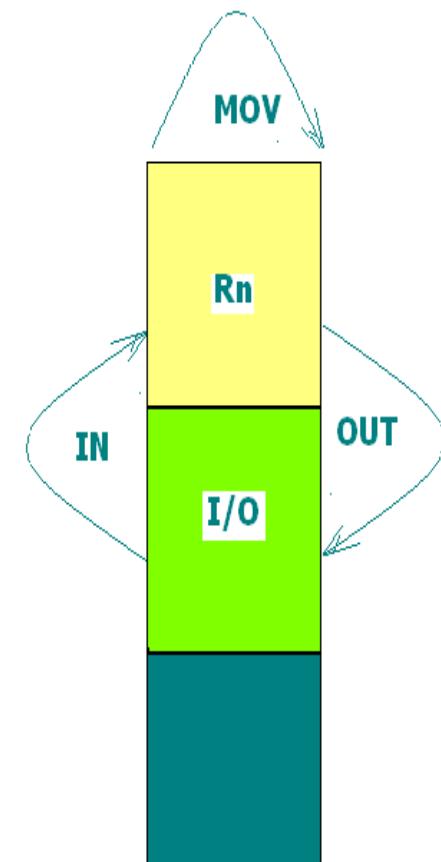
Program Counter:
 $PC \leftarrow PC + 1$

Syntax:
(i) OUT A,Rr

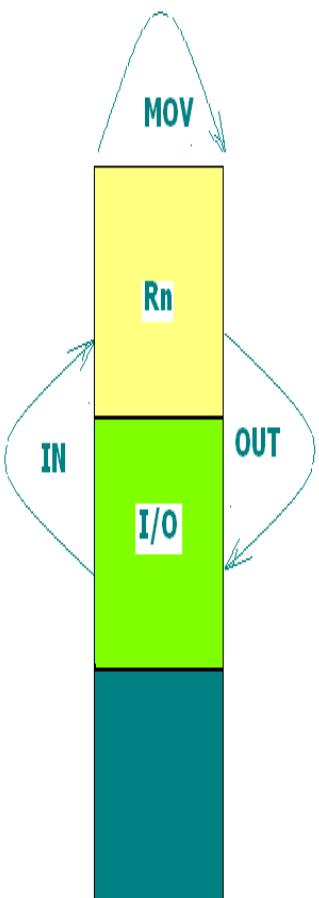
Operands:
 $0 \leq r \leq 31, 0 \leq A \leq 63$

16-bit Opcode:

1011	1AAr	rrrr	AAAA
------	------	------	------



Ejemplo



```
clr r16      ; Clear r16  
ser r17      ; Set r17  
out $18,r16   ; Write zeros to Port B  
nop          ; Wait (do nothing)  
out $18,r17   ; Write ones to Port B
```

Words: 1 (2 bytes) (i)

Operation: $Rd \leftarrow \$FF$ Program Counter:
 $PC \leftarrow PC + 1$

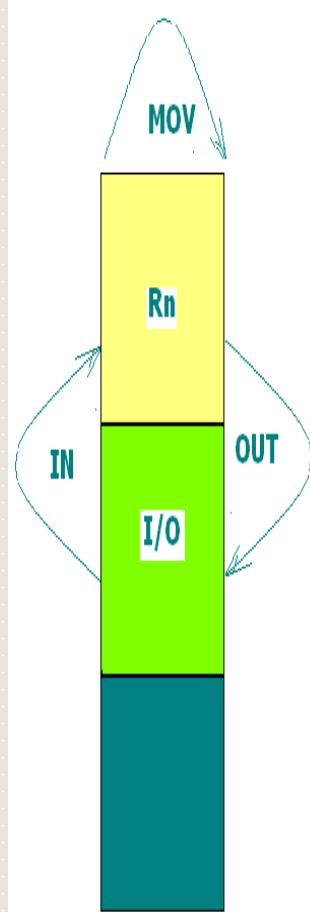
Cycles: 1 (i)

Syntax:
SER Rd

Operands:
16-bit Opcode: $16 \leq d \leq 31$

1110	1111	dddd	1111
------	------	------	------

IN - Load an I/O Location to Register



(i)

Operation:
 $Rd \leftarrow I/O(A)$

(i)

Syntax:
IN Rd,A

16-bit Opcode:

1011	0AA _d	dd _{dd}	A _{AA} A _{AA}
------	------------------	------------------	---------------------------------

Program Counter:
 $PC \leftarrow PC + 1$

Operands:
 $0 \leq d \leq 31, 0 \leq A \leq 63$

Example:

```
in    r25,$16    ; Read Port B  
cpi   r25,4     ; Compare read value to constant  
breq  exit      ; Branch if r25=4  
...  
exit:  nop       ; Branch destination (do nothing)
```

(i) **Operation:**
 Rd - K

Program Counter:
 PC \leftarrow PC + 1

(i) **Syntax:**
 CPI Rd,K

Operands:
 $16 \leq d \leq 31, 0 \leq K \leq 25$

16-bit Opcode:

0011	KKKK	dddd	KKF
------	------	------	-----

(i) **Operation:**
 If Rd = Rr ($Z = 1$) then PC \leftarrow PC + k + 1, else PC \leftarrow PC + 1

(i) **Syntax:** **Program Counter:**
 BREQ k PC \leftarrow PC + k + 1
 PC \leftarrow PC + 1, if condition es falsa

Operands:
 16-bit Opcode: $-64 \leq k \leq +63$

1111	00kk	kkkk	k001
------	------	------	------

REGISTROS DE E/S

- REGISTRO DE ESTADO
- STACK POINTER
- EEPROM
- PUERTOS

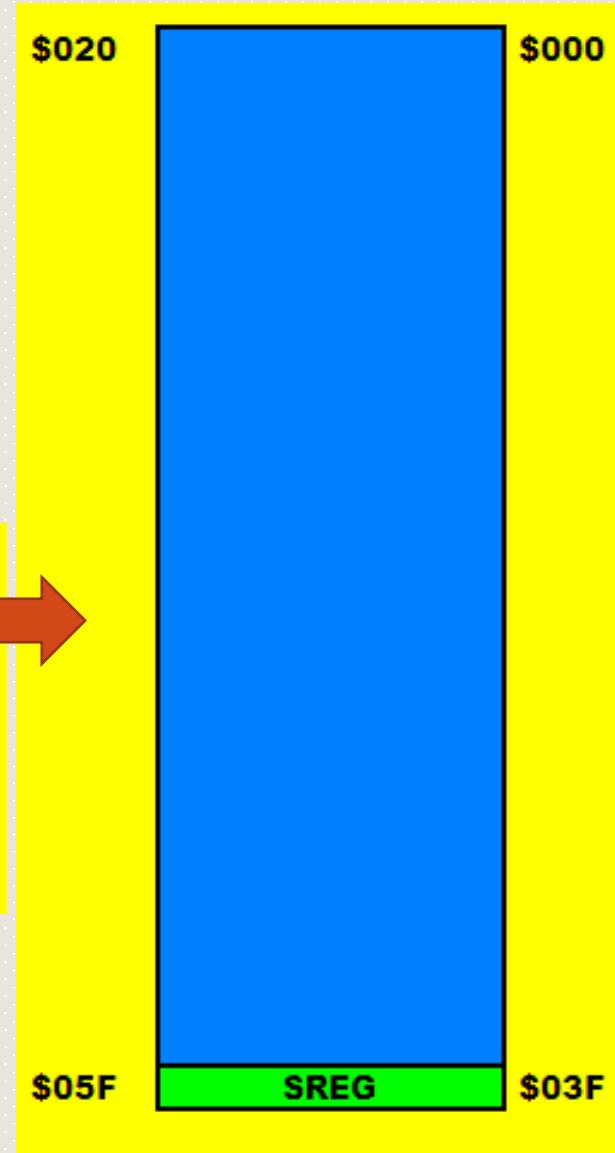
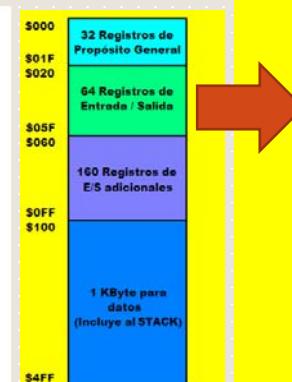
REGISTRO DE ESTADO

EL “status register” o sreg ocupa la última localidad de esta área



Valor inicial 0x00

- **I:** Habilitador global de Interrupciones, con un 1 lógico, las interrupciones son habilitadas.
 - **T:** Para respaldo de un bit, con la instrucción BLD para Carga y BST, para almacenamiento.
 - **H:** Bandera de acarreo de los 4 bits inferiores (Half)
 - **S:** Bit de signo
 - **V:** Bandera de Sobreflujo, en operaciones de complemento a dos.
 - **N:** Bandera de Negativo
 - **Z:** Bandera de Cero
 - **C:** Bandera de Acarreo
- } Se generan con operaciones Aritméticas y lógicas



BANDERAS DEL SREG

REG.	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	I/O	SRAM
SREG	I	T	H	S	V	N	Z	C	0x3F	(0x5F)

BIT 0 = C BANDERA DEL CARRY

BIT 1 = Z BANDERA DE CERO

BIT 2 = N BANDERA DE NEGACIÓN

BIT 3 = V BANDERA DE Overflow

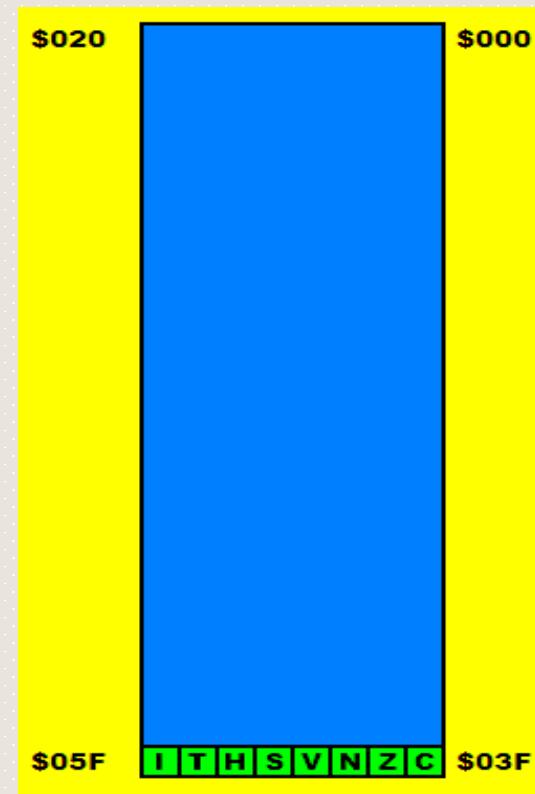
BIT 4 = S BANDERA DEL SIGNO

BIT 5 = H BANDERA DEL CARRY INTERMEDIO

BIT 6 = T BANDERA PARA COPIAR BITS

BIT 7 = I BANDERA PARA HABILITAR GLOBALMENTE LAS INTERRUPCIONES

S: $N \oplus V$, For signed tests.



EL "status"

EL "status register" o sreg

ocupa la última localidad de esta área

ago-18

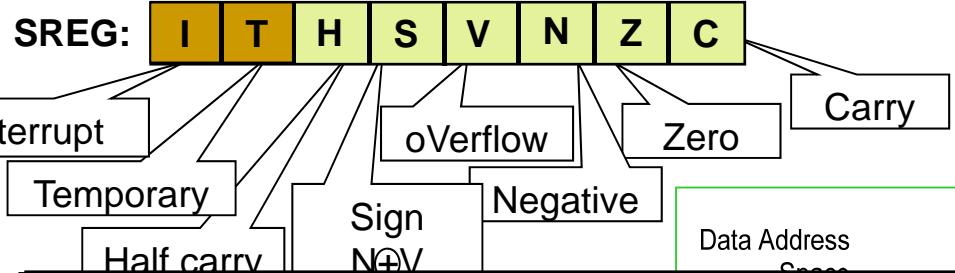
localidad de esta área

Table 2-4: Instructions That Affect Flag Bits

Instruction	C	Z	N	V	S	H
ADD	X	X	X	X	X	X
ADC	X	X	X	X	X	X
ADIW	X	X	X	X	X	
AND		X	X	X	X	
ANDI		X	X	X	X	
CBR		X	X	X	X	
CLR		X	X	X	X	
COM	X	X	X	X	X	
DEC		X	X	X	X	
EOR		X	X	X	X	
FMUL	X	X				
INC		X	X	X	X	
LSL	X	X	X	X		X
LSR	X	X	X	X		
OR		X	X	X	X	
ORI		X	X	X	X	
ROL	X	X	X	X		X
ROR	X	X	X	X		
SEN			1			
SEZ			1			
SUB	X	X	X	X	X	X
SUBI	X	X	X	X	X	X
TST		X	X	X	X	

Note: X can be 0 or 1. (See Chapter 5 for how to use these instructions.)

Status Register (SREG)



Example: Show the status of the C, H, and Z flag after subtraction of 0x9C from 0x9C in the following

LDI R20, 0x9C

LDI R21, 0x9C

SUB R20, R21 ;subtract R21 from R20

Table 2-5: AVR Branch (Jump) Instructions Using Flag Bits

Instruction	Action
BRLO	Branch if C = 1
BRSH	Branch if C = 0
BREQ	Branch if Z = 1
BRNE	Branch if Z = 0
BRMI	Branch if N = 1
BRPL	Branch if N = 0
BRVS	Branch if V = 1
BRVC	Branch if V = 0



SREG:

Solution:

$$\begin{array}{r} \$9C \\ - \$9C \\ \hline \$00 \end{array} \quad \begin{array}{l} 1001\ 1100 \\ 1001\ 1100 \\ \hline 0000\ 0000 \end{array} \quad R20 = \$00$$

C = 0 because R21 is not bigger than R20 and there is no borrow from D8 bit.

Z = 1 because the R20 is zero after the subtraction.

H = 0 because there is no borrow from D4 to D3.

El carry se invierte en restas

SREG – AVR Status Register⁵

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

Non ALU

- Bit 7 – I: Global Interrupt Enable**
The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the `reti` instruction. The I-bit can also be set and cleared by the application with the `sei` and `cli` instructions.
- Bit 6 – T: Bit Copy Storage**
The Bit Copy instructions `bld` (Bit LoAd) and `bst` (Bit STore) use the T-bit as source or destination. A bit from a register can be copied into T ($R_b \rightarrow T$) by the `bst` instruction, and a bit in T can be copied into a bit in a register ($T \rightarrow R_b$) by the `bld` instruction.

ALU

Signed two's complement arithmetic

- Bit 4 – S: Sign Bit, $S = N \oplus V$**
Bit set if answer is negative with no errors or if both numbers were negative and error occurred, zero otherwise.
- Bit 3 – V: Two's Complement Overflow Flag**
Bit set if error occurred as the result of an arithmetic operation, zero otherwise.
- Bit 2 – N: Negative Flag**
Bit set if result is negative, zero otherwise.

Unsigned arithmetic

- Bit 5 – H: Half Carry Flag**
Carry from least significant nibble to most significant nibble. Half Carry is useful in BCD arithmetic.
- Bit 0 – C: Carry Flag**
The Carry Flag C indicates a carry in an arithmetic operation. Bit set if error occurred as the result of an unsigned arithmetic operation, zero otherwise.

Arithmetic and Logical

- Bit 1 – Z: Zero Flag**
The Zero Flag Z indicates a zero result in an arithmetic or logic operation.

⁵ Source: ATmega328P Data Sheet Document 8161 Section 6.3 Status Register

THE SREG OVERFLOW BIT

- The overflow bit indicates if there was an error caused by the addition of two n-bit 2's complement numbers, where the $n-1$ "sign bit" is 1 if the number is negative and 0 if the number is positive. In other words, the sum is outside the range -2^{n-1} to $2^{n-1}-1$.
- Another way to recognize an error in addition is to observe that if you add two numbers of the same sign (positive + positive = negative or negative + negative = positive) then an error has occurred.
- An overflow condition can never result from the addition of two n-bit numbers of opposite sign (positive - negative or negative + positive).
- Here are examples of all four cases for two 8 bit signed numbers.

Case	A	B	C	D
	$0b_6b_5b_4b_3b_2b_1b_0$	$0b_6b_5b_4b_3b_2b_1b_0$	$1b_6b_5b_4b_3b_2b_1b_0$	$1b_6b_5b_4b_3b_2b_1b_0$
	<u>$0b_6b_5b_4b_3b_2b_1b_0$</u>	<u>$1b_6b_5b_4b_3b_2b_1b_0$</u>	<u>$0b_6b_5b_4b_3b_2b_1b_0$</u>	<u>$1b_6b_5b_4b_3b_2b_1b_0$</u>

The variable " b_n " simply indicates some binary value and may be 1 or 0. The index of the carry bit (C_n) is equal to the carry into bit b_n . For example, the carry into b_0 is C_0 and the carry out of an 8-bit register b_7 is C_8 .

- Looking first at Case A, a carry cannot be generated out of the sign bit ($C_{n+1}=0$); therefore, if a carry enters the sign bit ($C_n=1$), the sum will be negative and the answer is wrong.
 - For Case B and Case C no error can occur. Observe that in both case B and C because the numbers are contained in an n-bit ($n=8$) register, we know they are in the range -2^{n-1} to $2^{n-1}-1$ (-128 to 127 for our two 8-bit numbers). Because one number is positive and the other negative, we further know, the answer must be correct.
 - For Case D, a carry will always be generated out of the sign bit $C_{n+1}=1$ (ex. $C_8=1$) with the sign bit itself set to 0; therefore, if a carry does not enter the sign bit $C_n=0$ ($C_7=1$) the sum will be positive and the answer will be wrong.
- Here is what we have discovered translated into a truth-table.
- | C_{n+1} | C_n | V | Case |
|-----------|-------|---|---|
| 0 | 0 | 0 | may occur for cases A, B, C without error |
| 0 | 1 | 1 | A |
| 1 | 0 | 1 | D |
| 1 | 1 | 0 | may occur for cases B, C, D without error |
- Solving for the overflow bit (V) we have, $V = C_{n+1} \oplus C_n$

Bit 4 – S: Bit de signo, $S = N \oplus V$

El bit S siempre es una OR exclusiva entre el flag negativo N y el flag V de desbordamiento de complemento a dos.

$$\begin{array}{r} +96 \quad 0x60 \quad 0110 \quad 0000 \\ + \underline{+70} \quad 0x46 \quad 0100 \quad 0110 \\ + 166 \quad 0xA0 \quad 1010 \quad 0110 \end{array}$$

$$S=N @ V=0$$

$N = \star$ (negative) and $V = \star$

$$\begin{array}{r} -128 \quad 0x80 \quad 1000 \quad 0000 \\ + -2 \quad 0x02 \quad \underline{1111} \quad 1110 \\ - 130 \quad 0x7E \quad 0111 \quad 1110 \end{array}$$

$$S=N @ V=1$$

$N = \star$ (positive) and $V = \star$

$$\begin{array}{r} -2 \quad 1111 \quad 1110 \\ + -5 \quad \underline{1111} \quad 1011 \\ - 7 \quad 1111 \quad 1001 \end{array}$$

$$S=N @ V=1$$

and $V = \star$ and $N = \star$ Sum is negative

$$\begin{array}{r} + 7 \quad 0000 \quad 0111 \\ + \underline{+18} \quad 0001 \quad 0010 \\ +25 \quad 0001 \quad 1001 \end{array}$$

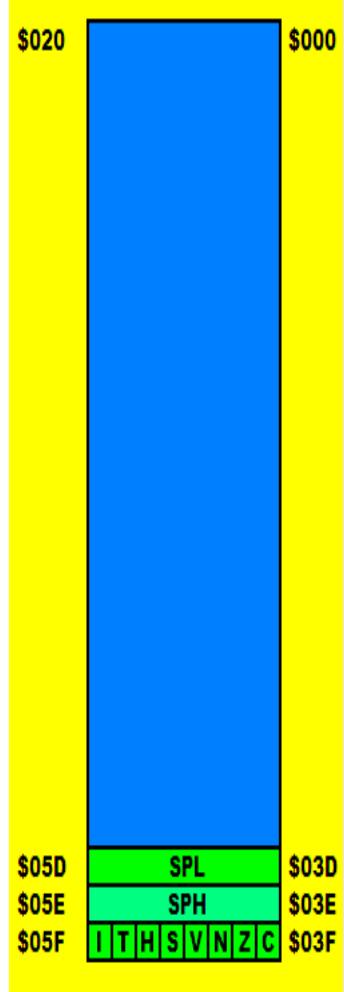
$$S=N @ V=0$$

$N = \star$ (positive 25) and $V = \star$

(STACK POINTER)

- La pila es implementada en el espacio de propósito general ([RAM](#)).
 - Es usada para almacenamiento temporal de variables (instrucciones PUSH y POP) o durante la llamada de subrutinas o el manejo de interrupciones.
 - El [registro SP](#) es el apuntador al tope de la pila. Realmente el SP se compone de 2 registros, para la parte alta (SPH) y para la parte baja (SPL), esto para direccionar al espacio completo de memoria.
 - La pila tiene un [crecimiento hacia abajo](#), es decir, de las direcciones altas de memoria a las direcciones bajas.
 - Despues de un reset, el apuntador de Pila tiene el valor de 0x0000, por lo que [debe ser inicializado](#) dentro del programa (con 0x0460 para ATMega8), para que realice un almacenamiento dentro de un espacio válido.

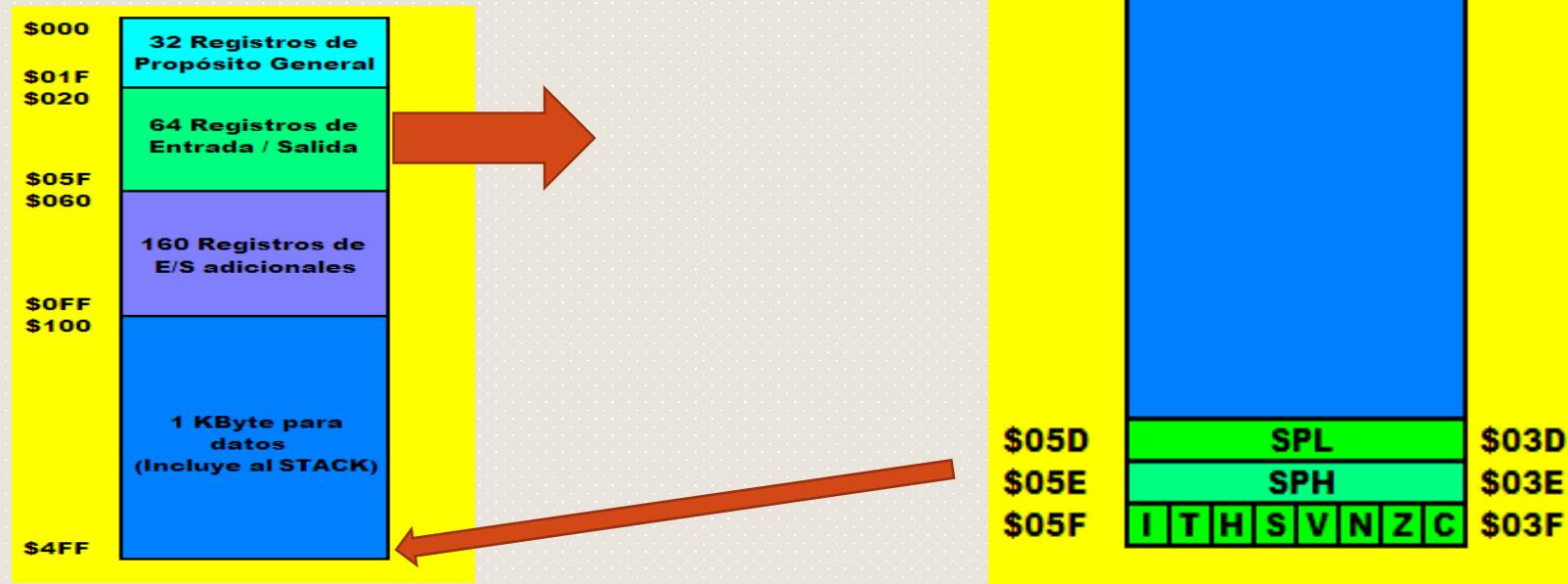
\$3E	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
\$3D	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL



PUNTERO DE LA PILA (STACK POINTER)

Valor inicial 0x00

- LDI R16, low(RAMEND)
- OUT SPL, R16
- LDI R16, high(RAMEND)
- OUT SPH, R16



Ejemplo (RAMEND, PUSH ,POP)

```
.include "tn2313def.inc"

;Set up AVR ATTiny2313 stack
LDI      R16, RAMEND
OUT      SPL, R16

LDI      R16, 0x33
LDI      R17, 0x25
LDI      R18, 0x0A

PUSH    R16
PUSH    R17
POP     R17
PUSH    R18

end:   RJMP  end
```

Ldi r16, HIGH(RAMEND)
Out SPH, r16
Ldi r16, LOW(RAMEND)
Out SPL, r16

AVR ATMEGA

ago-18

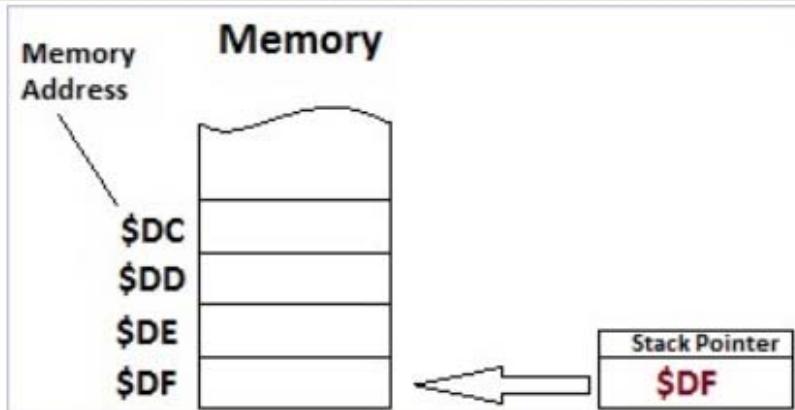


Figure 1 - Initial Stack

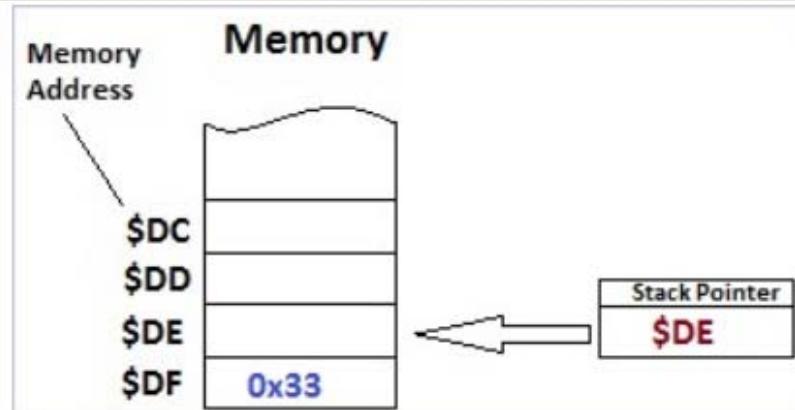


Figure 2 - Stack After First PUSH

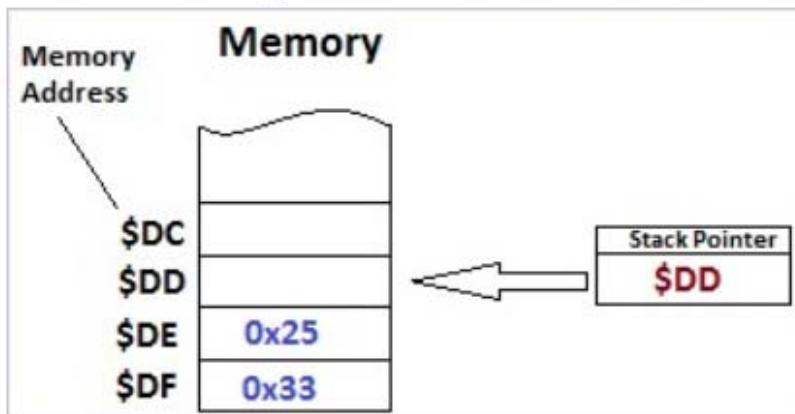


Figure 3 - Stack After Second PUSH

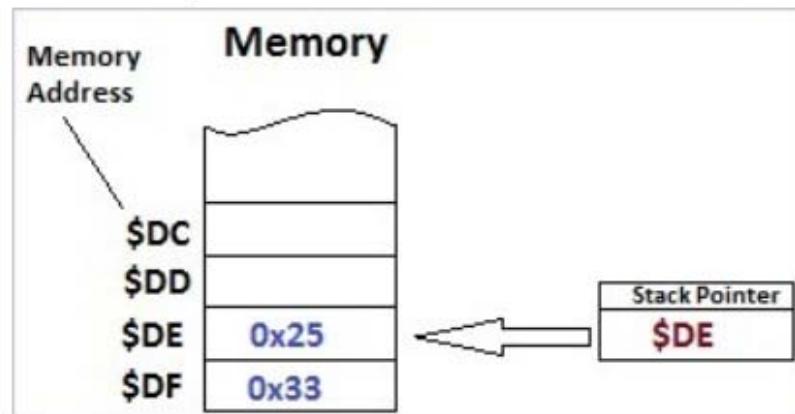


Figure 4 - Stack After POP

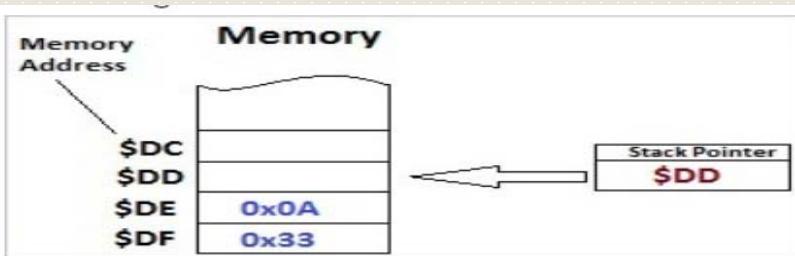
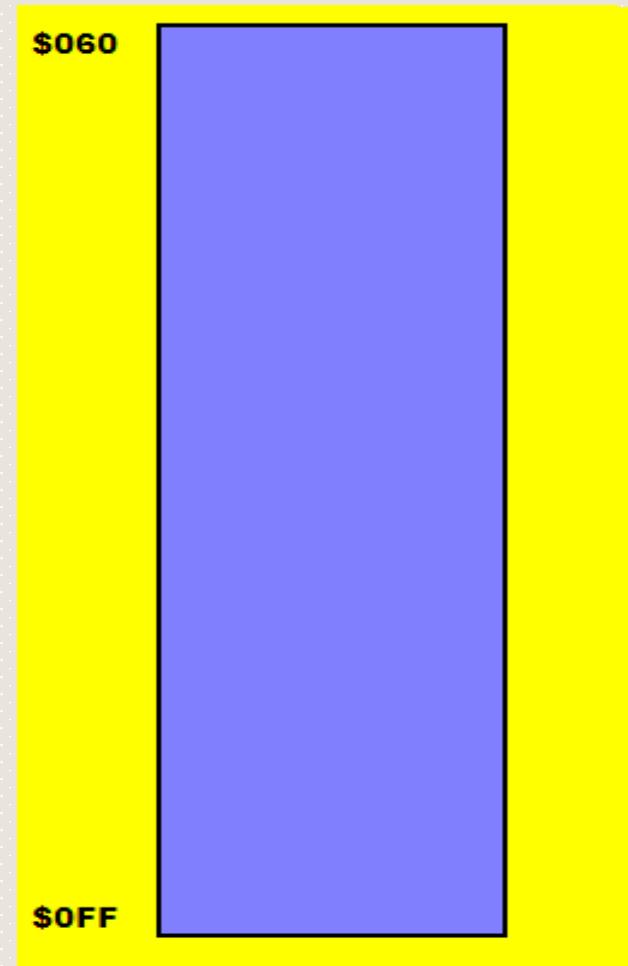
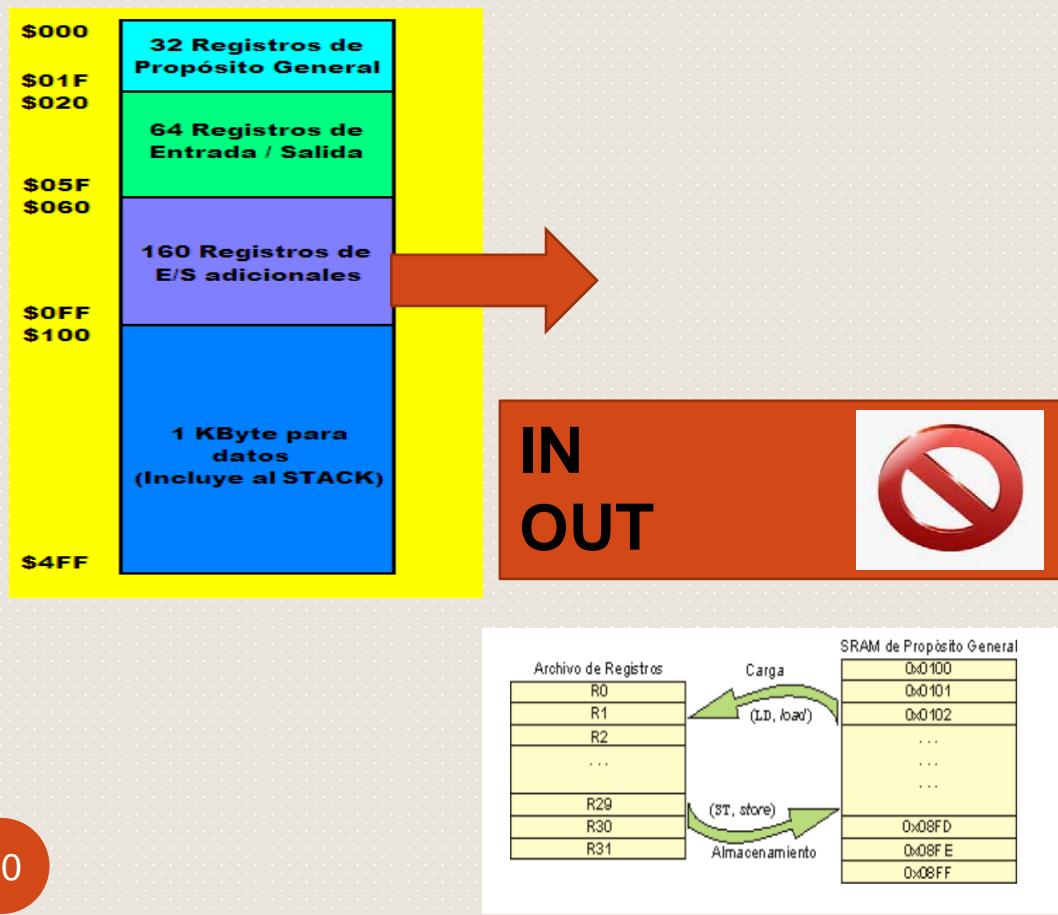


Figure 5 - Stack After Third PUSH

Ejemplo (RAMEND, PUSH, POP)

AVR	Tutorials	Diagram	
LDI	R16, 0x33	PUSH	R16
LDI	R17, 0x25	PUSH	R17
LDI	R18, 0x0A	POP	R17
		PUSH	R18

160 REGISTROS E/S ADICIONALES



Memoria Flash (Men. Prog.)

16 KBYTES DE FLASH

PARA DIRECCIONAR 16 KBYTES
SE NECESITA UN BUS DE
DIRECCIONES DE 14 BITS,
DESDE:

00 0000 0000 0000 = \$0000

HASTA:

11 1111 1111 1111 = \$3FFF

CÓDIGO DE
MAQUINA DE LAS
INSTRUCCIONES

\$0000	1 0 1 0 1 1 1 1
\$0001	1 1 0 0 0 1 1 0
\$0002	1 0 0 1 0 0 1 1
\$0003	1 1 0 0 0 1 1 0
\$0004	0 1 1 1 0 0 1 0
\$0005	0 0 1 1 0 0 1 0
\$0006	0 0 0 1 1 0 1 0
\$0007	1 0 1 0 1 1 1 1
\$0008	0 0 1 1 1 1 1 0
\$0009	0 0 0 0 0 0 0 0
\$000A	0 0 0 1 0 1 1 1
\$000B	0 0 1 1 0 0 1 0
\$3FFC	1 1 1 1 1 1 1 1
\$3FFD	1 1 1 1 1 1 1 1
\$3FFE	1 1 1 1 1 1 1 1
\$3FFF	1 1 1 1 1 1 1 1

ARREGLO DE 8 K x 16 BITS

LAS INSTRUCCIONES Atmega SON DE 16 o 32 BITS.

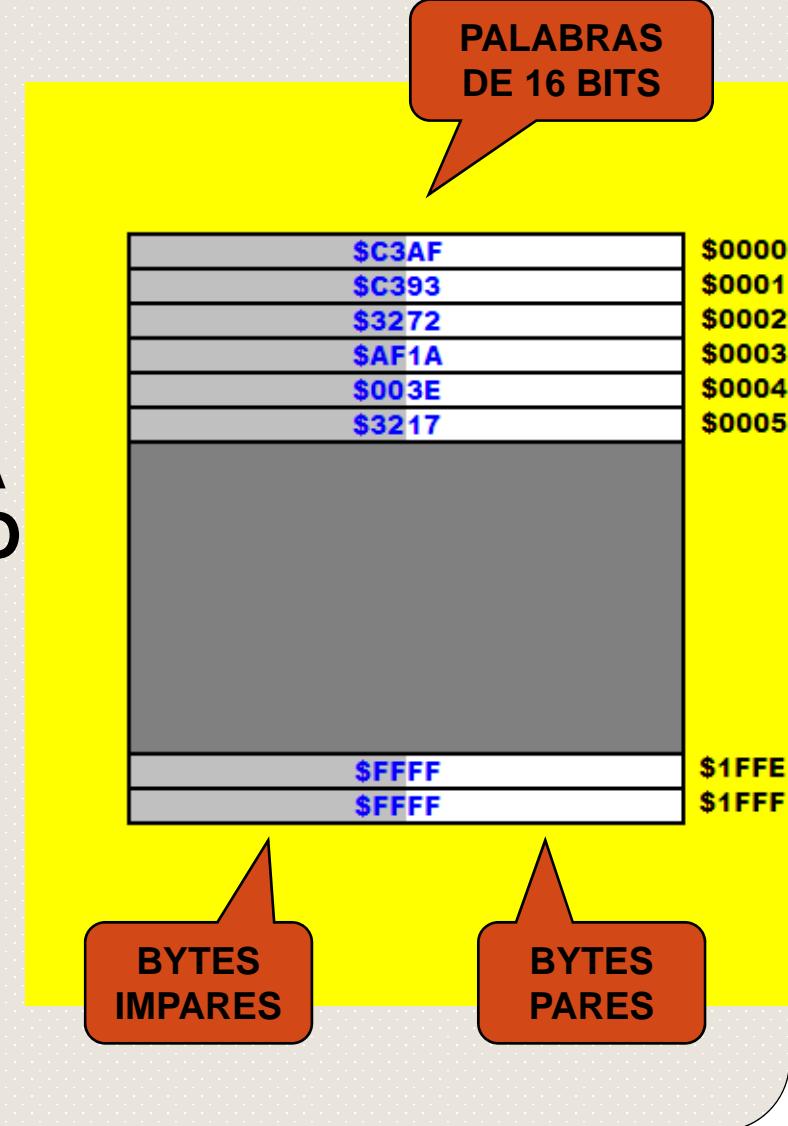
POR ESO, LA FLASH ESTÁ ARREGLADA EN PALABRAS DE 16 BITS

EL CONTADOR DEL PROGRAMA TIENE 13 BITS, DIRECCIONANDO DESDE:

0 0000 0000 0000 = \$0000

HASTA:

1 1111 1111 1111 = \$1FFF



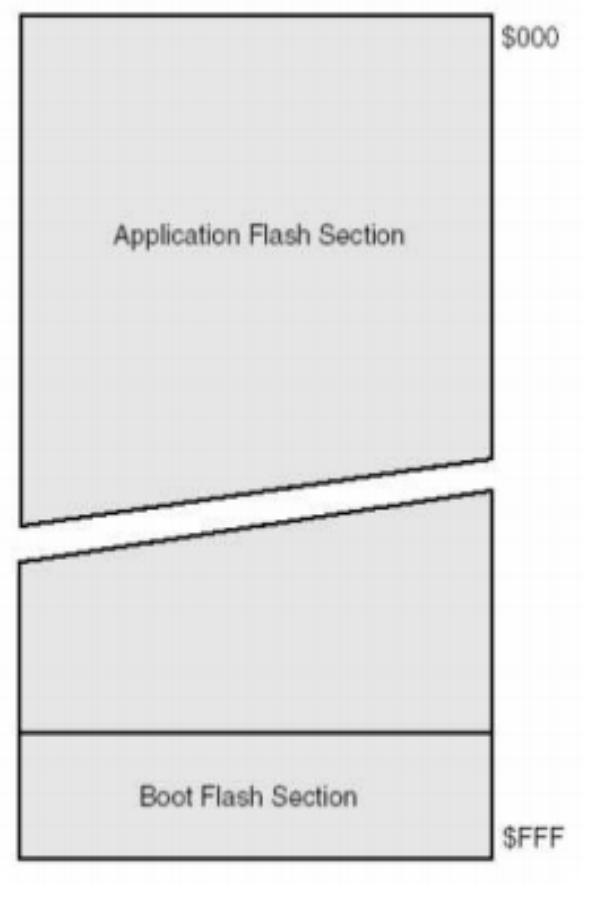
	(Bytes)	(Hex)	Organization
ATtiny25	2K	00000–003FF	1K × 2 bytes
ATmega8	8K	00000–00FFF	4K × 2 bytes
ATmega32	32K	00000–03FFF	16K × 2 bytes
ATmega64	64K	00000–07FFF	32K × 2 bytes
ATmega128	128K	00000–0FFFF	64K × 2 bytes
ATmega256	256K	00000–1FFFF	128K × 2 bytes

The diagram illustrates the memory organization for three AVR microcontrollers:

- Tiny25:** Shows a memory block from address 00000 to 003FF. The width is 2 bytes, indicated by a double-headed arrow above the block. The organization is 1Kx2Bytes.
- Mega16:** Shows a memory block from address 00000 to 01FFF. The width is 2 bytes, indicated by a double-headed arrow above the block. The organization is 8Kx2Bytes.
- Mega64:** Shows a memory block from address 00000 to 07FFF. The width is 2 bytes, indicated by a double-headed arrow above the block. The organization is 32Kx2Bytes.

Mapa de Memoria Programa

- Espacio continuo de memoria Flash cuyo tamaño varia entre procesadores, para el ATMega 8 es de 8 Kbytes, organizados como 4K x 16 bits. Soporta hasta 10,000 ciclos de escritura/borrado.
- La memoria se puede particionar en una sección para aplicación y una sección de arranque, donde podría manejarse un cargador para auto programación
(Boot Loader Support – Read-While-Write Self-programming).



Memoria de Programa: (IRQ's)

- En el espacio de almacenamiento se incluyen a los **Vectores de Interrupciones**, iniciando en la dirección \$000.
- El número de vectores, en la familia AVR, varia de procesador a procesador, en función de los recursos existentes. Un programa debería iniciar en una ubicación, mas allá de estos vectores. ([rjmp Reset](#))

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
1	0x000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, and Watchdog Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	TIMER2 COMP	Timer/Counter2 Compare Match
5	0x004	TIMER2 OVF	Timer/Counter2 Overflow
6	0x005	TIMER1 CAPT	Timer/Counter1 Capture Event
7	0x006	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	0x007	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	0x008	TIMER1 OVF	Timer/Counter1 Overflow

VECTOR de interrupción

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
1	\$000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	\$002	INT0	External Interrupt Request 0
3	\$004	INT1	External Interrupt Request 1
4	\$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	\$008	TIMER2 OVF	Timer/Counter2 Overflow
6	\$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	\$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	\$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	\$010	TIMER1 OVF	Timer/Counter1 Overflow
10	\$012	TIMER0 OVF	Timer/Counter0 Overflow
11	\$014	SPI, STC	Serial Transfer Complete
12	\$016	USART, RXC	USART, Rx Complete
13	\$018	USART, UDRE	USART Data Register Empty
14	\$01A	USART, TXC	USART, Tx Complete
15	\$01C	ADC	ADC Conversion Complete
16	\$01E	EE_RDY	EEPROM Ready
17	\$020	ANA_COMP	Analog Comparator
18	\$022	TWI	Two-wire Serial Interface
19	\$024	INT2	External Interrupt Request 2
20	\$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	\$028	SPM_RDY	Store Program Memory Ready

EJERCICIO PARA PRACTICAR EL USO DE LAS INSTRUCCIONES DE TRANSFERENCIA

LDS - Load Direct from Data Space

Operation:

(i) $Rd \leftarrow (k)$

Program Counter:

$PC \leftarrow PC + 2$

Syntax:

(i) LDS Rd,k

Operands:

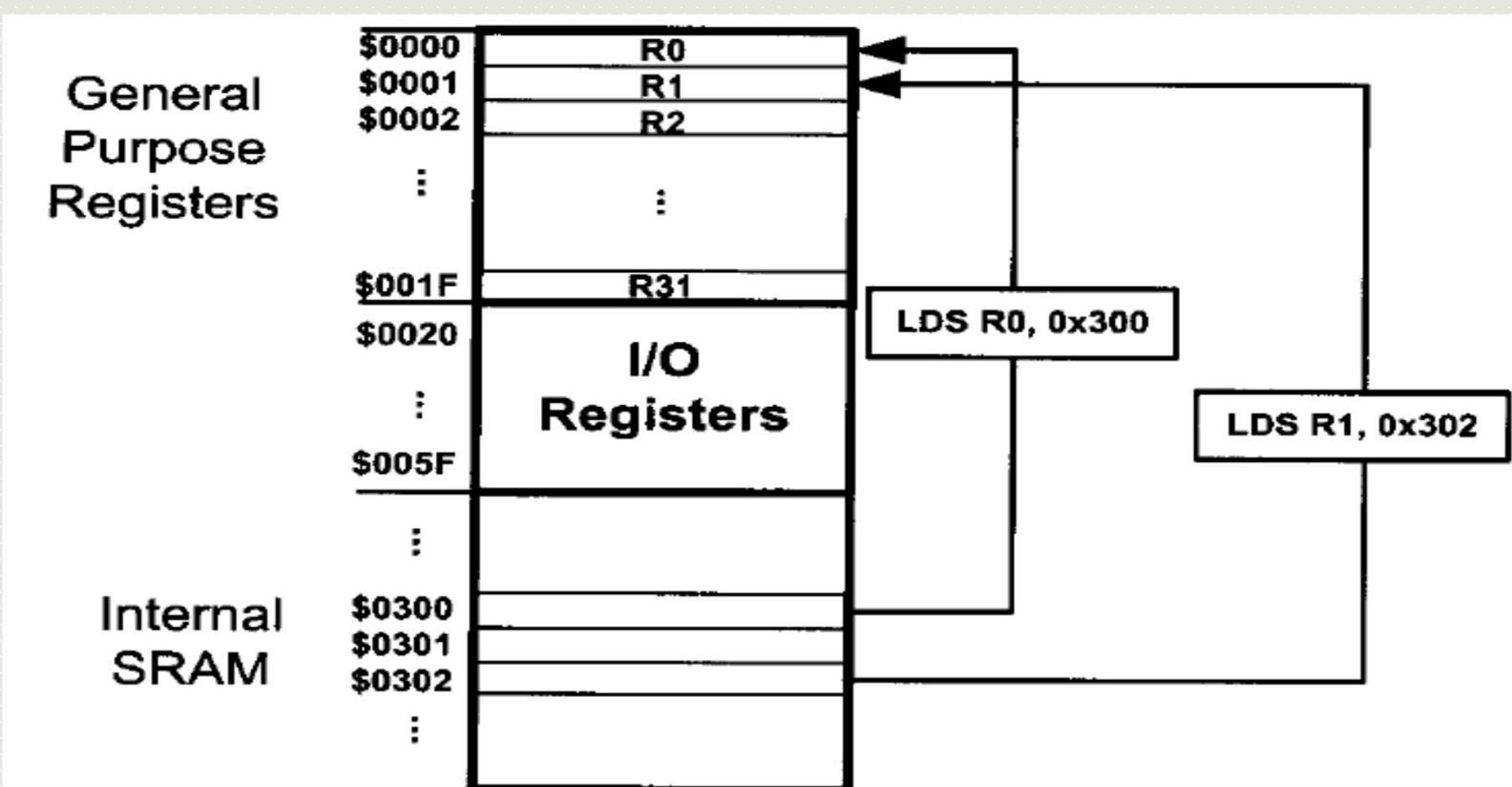
$0 \leq d \leq 31, 0 \leq k \leq 65535$

32-bit Opcode:

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

EJEMPLO

```
LDS R0, 0x300 ;R0 = the contents of location 0x300  
LDS R1, 0x302 ;R1 = the contents of location 0x302  
ADD R1, R0      ;add R0 to R1
```



STS

	Operation: $(k) \leftarrow Rr$	Program Counter: $PC \leftarrow PC + 2$
(i)	Syntax: STS k,Rr	Operands: $0 \leq r \leq 31, 0 \leq k \leq 65535$

32-bit Opcode:

1001	001d	dddd	0000
kkkk	kkkk	kkkk	kkkk

Ejemplo de STS

```

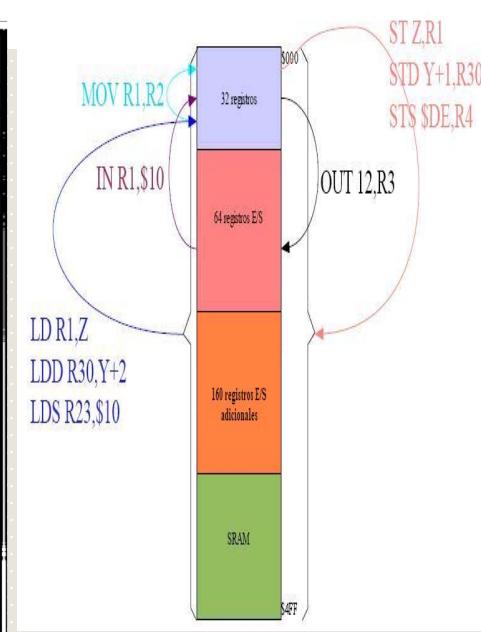
LDI    R16, 0x55      ;R16 = 55 (in hex)
STS    0x38, R16      ;copy R16 to Port B (PORTB = 0x55)
STS    0x35, R16      ;copy R16 to Port C (PORTC = 0x55)
STS    0x32, R16      ;copy R16 to Port D (PORTD = 0x55)

```

Address		Name
Mem.	I/O	
\$20	\$00	TWBR
\$21	\$01	TWSR
\$22	\$02	TWAR
\$23	\$03	TWDR
\$24	\$04	ADCL
\$25	\$05	ADCH
\$26	\$06	ADCSSRA
\$27	\$07	ADMUX
\$28	\$08	ACSR
\$29	\$09	UBRRL
\$2A	\$0A	UCSRB
\$2B	\$0B	UCSRA
\$2C	\$0C	UDR
\$2D	\$0D	SPCR
\$2E	\$0E	SPSR
\$2F	\$0F	SPDR
\$30	\$10	PIND
\$31	\$11	DDRD
\$32	\$12	PORTD
\$33	\$13	PINC
\$34	\$14	DDRC
\$35	\$15	PORTC

Address		Name
Mem.	I/O	
\$36	\$16	PINB
\$37	\$17	DDRB
\$38	\$18	PORTB
\$39	\$19	PINA
\$3A	\$1A	DDRA
\$3B	\$1B	PORTA
\$3C	\$1C	EECR
\$3D	\$1D	EEDR
\$3E	\$1E	EEARL
\$3F	\$1F	EEARH
\$40	\$20	UBRRC
\$41	\$21	WDTCR
\$42	\$22	ASSR
\$43	\$23	OCR2
\$44	\$24	TCNT2
\$45	\$25	TCCR2
\$46	\$26	ICR1L
\$47	\$27	ICR1H
\$48	\$28	OCR1BL
\$49	\$29	OCR1BH
\$4A	\$2A	OCR1AL

Address		Name
Mem.	I/O	
\$4B	\$2B	OCR1AH
\$4C	\$2C	TCNT1L
\$4D	\$2D	TCNT1H
\$4E	\$2E	TCCR1B
\$4F	\$2F	TCCR1A
\$50	\$30	SFIOR
\$51	\$31	OCDR
\$52	\$32	OSCCAL
\$53	\$33	TCNT0
\$54	\$34	TCCR0
\$55	\$35	MCUCSR
\$56	\$36	MCUCR
\$57	\$37	TWCR
\$58	\$38	SPMCR
\$59	\$39	TIFR
\$5A	\$3A	TIMSK
\$5B	\$3B	GIFR
\$5C	\$3C	GICR
\$5D	\$3D	OCR0
\$5E	\$3E	SPL
\$5F	\$3F	SREG



IN vs. LDS

As we mentioned earlier, we can use the LDS instruction to copy the contents of a memory location to a GPR. This means that we can load an I/O register into a GPR, using the LDS instruction. So, what is the advantage of using the IN instruction for reading the contents of I/O registers over using the LDS instruction? The IN instruction has the following advantages:

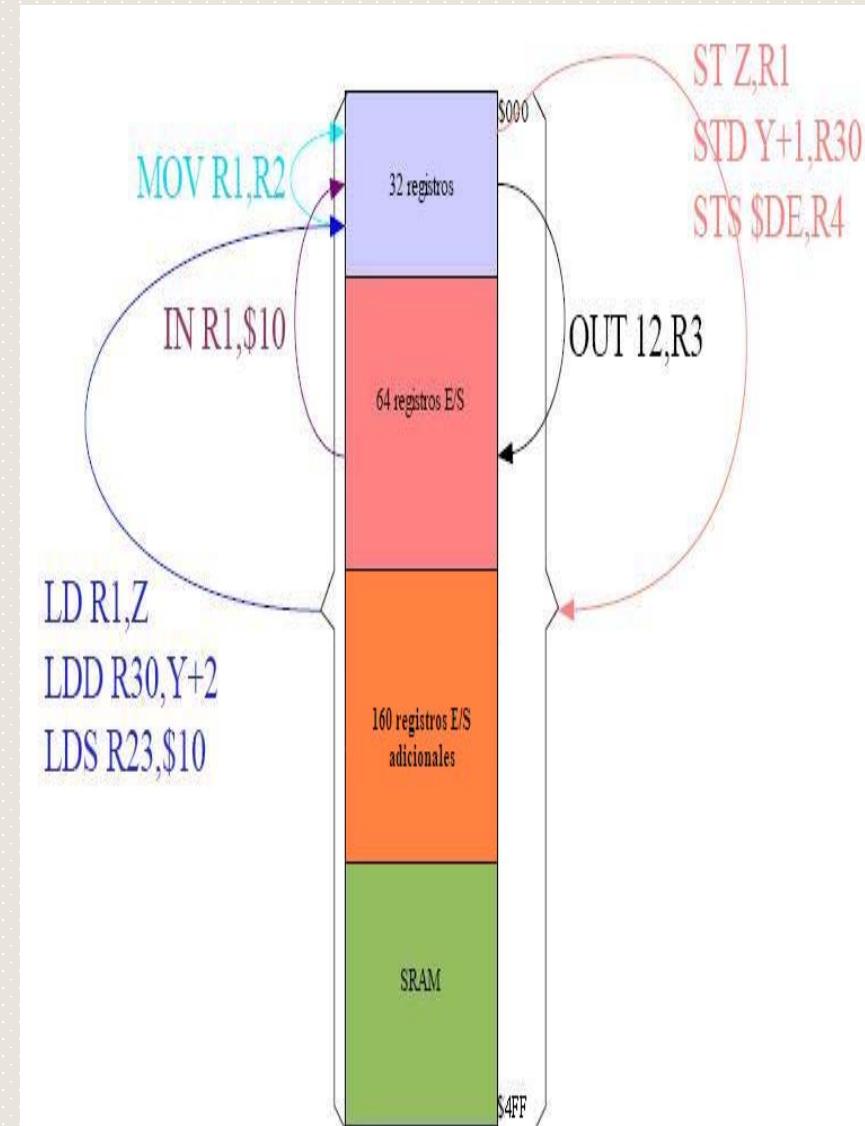
1. The CPU executes the IN instruction faster than LDS. As you will see in Chapter 3, the IN instruction lasts 1 machine cycle, whereas LDS lasts 2 machine cycles.
2. The IN is a 2-byte instruction, whereas LDS is a 4-byte instruction. This means that the IN instruction occupies less code memory.
3. When we use the IN instruction, we can use the names of the I/O registers instead of their addresses.
4. The IN instruction is available in all of the AVR, whereas LDS is not implemented in some of the AVR.

Notice that in using the IN instruction we can access only the standard I/O memory, while we can access all parts of the data memory using the LDS instruction.

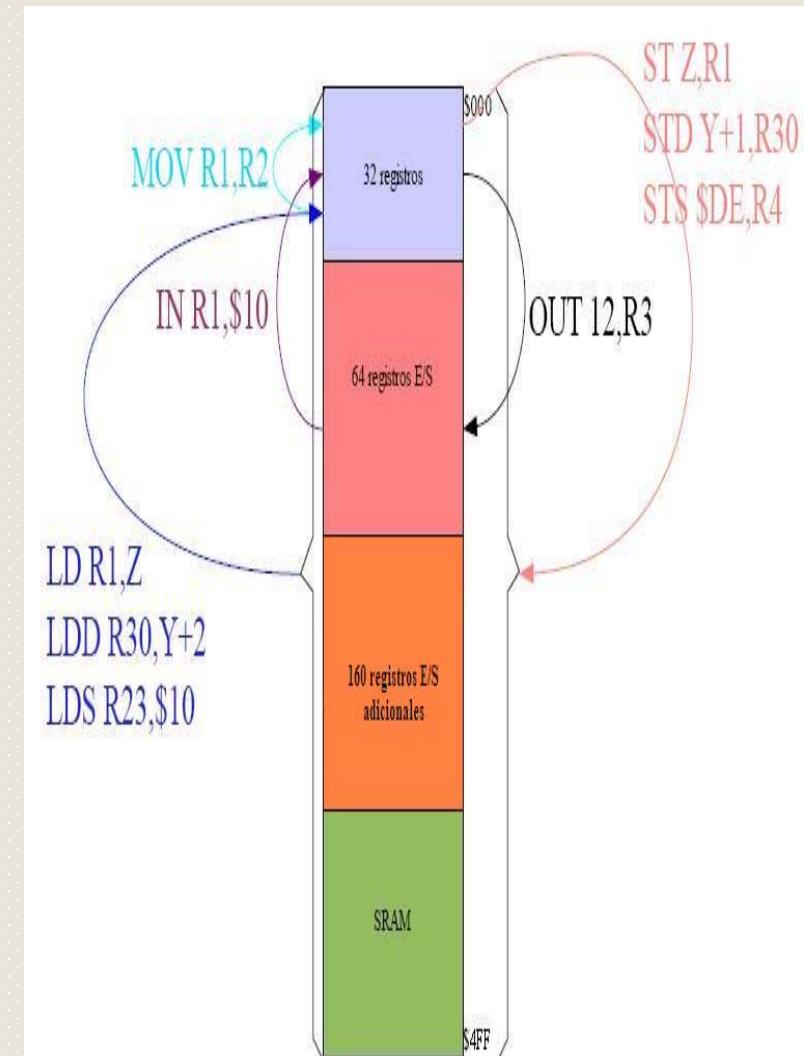
INSTRUCCIONES DE TRANSFERENCIA

SON AQUELLAS QUE PERMITEN MOVER DATOS ENTRE LAS DISTINTAS LOCALIDADES DE LAS MEMORIAS:

- **COPiar REGISTROS (MOV, MOVW)**
- **CARGAR AL REGISTRO UN VALOR (LDI)**
- **CARGAR UN REGISTRO DESDE LA SRAM (LD, LDD, LDS, POP)**

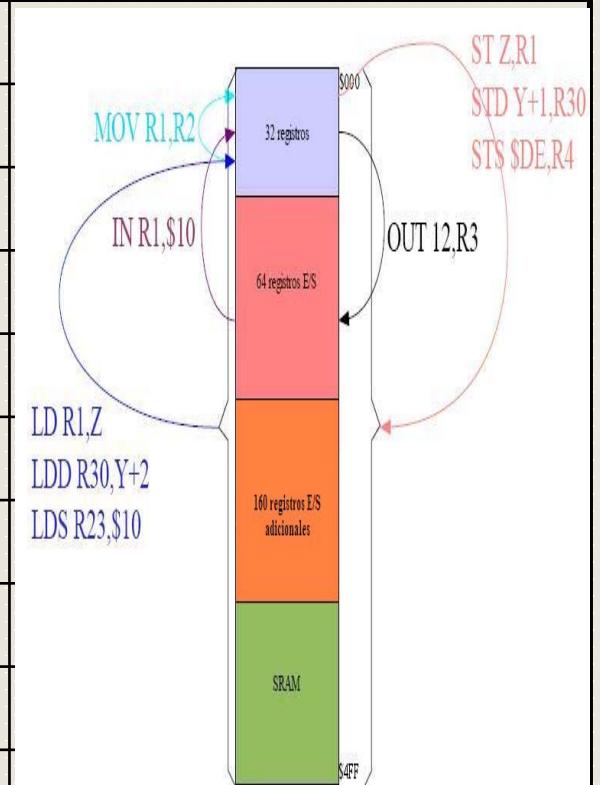


- ALMACENAR EN SRAM UN REGISTRO (ST, STD, STS, PUSH)
- CARGAR UN REGISTRO DESDE LA FLASH (LPM)
- ENTRADAS Y SALIDAS MEDIANTE LOS REGISTROS DE LOS PÓRTICOS (IN, OUT)

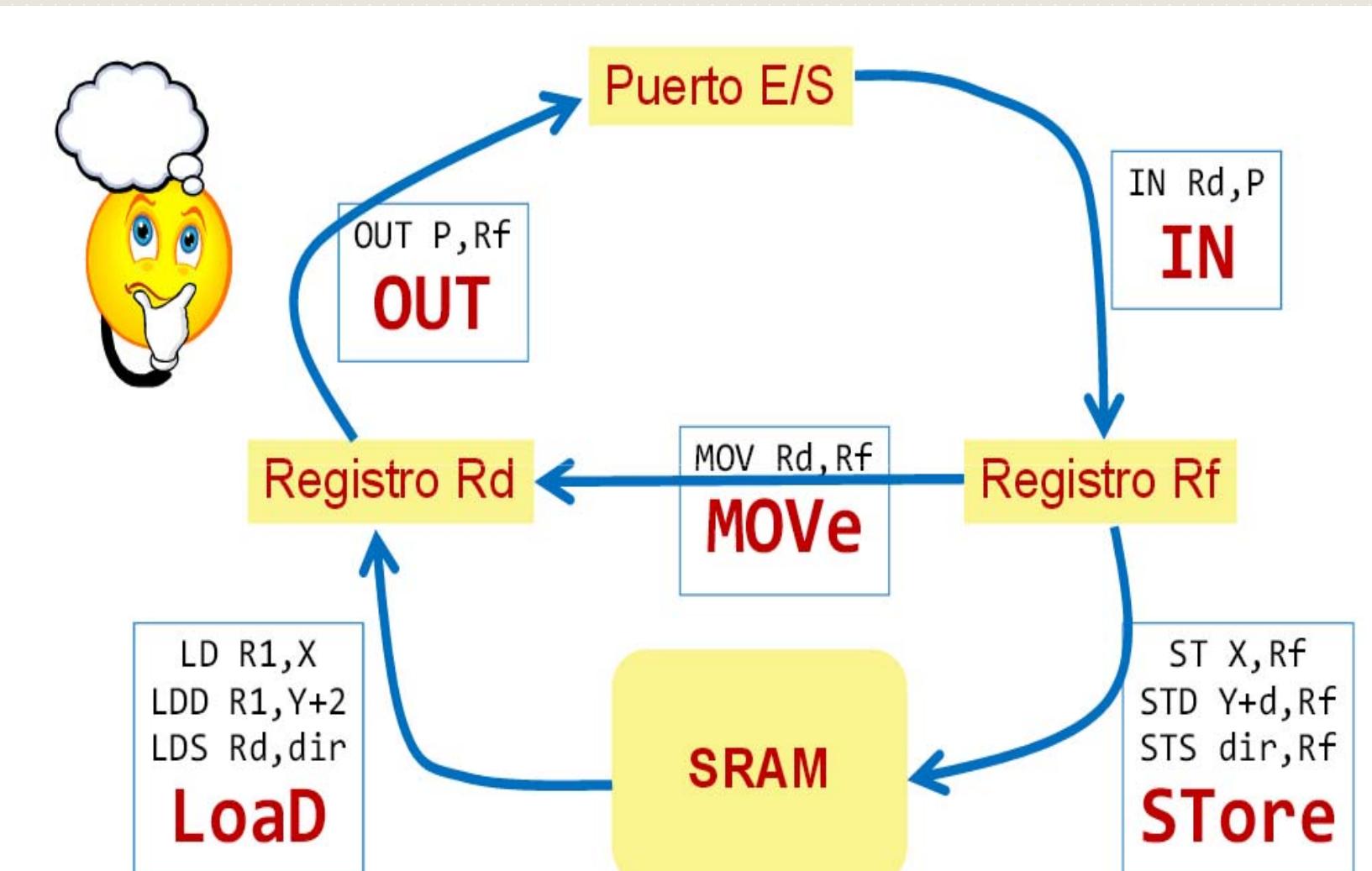


Implementar un programa que permita llenar las 8 primera posiciones de la memoria de datos con los códigos ASCII de las letras H-o-l-a

.cseg			
	LDI	R16,0x48	;letra H → 'H'
	LDI	R17,0x6F	;letra o → 'o'
	LDI	R18,0x6C	letra l → 'l'
	LDI	R19,0x61	letra a → 'a'
	STS	0x100,R16	
	STS	0x101,R17	
	STS	0x102,R18	
	STS	0x103,R19	
	STS	0x104,R16	
	STS	0x105,R17	
	STS	0x104,R18	
	STS	0x105,R19	
	
FIN:	RJMP	FIN	



RESUMEN



Memoria EPROM

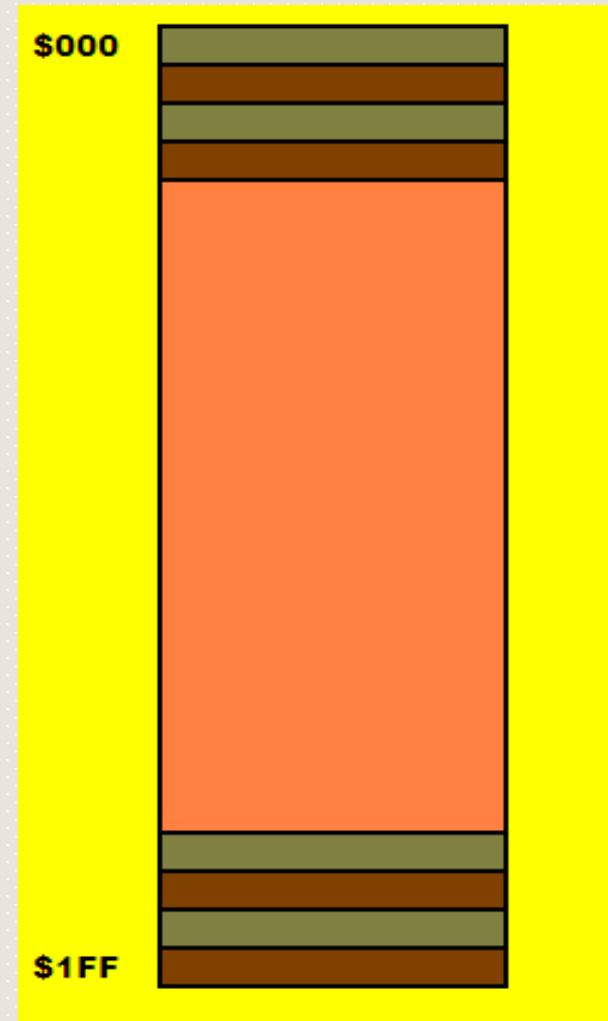
BYTES DE EEPROM

512 BYTES DE EEPROM

rango \$000 → \$1ff

El acceso a esta memoria no se realiza directamente mediante instrucciones, sino a través de registros de e/s

Device	Flash	EEPROM	RAM
ATmega48	4K Bytes	256 Bytes	512 Bytes
ATmega88	8K Bytes	512 Bytes	1K Bytes
ATmega168	16K Bytes	512 Bytes	1K Bytes



Memoria de dato (EEPROM)

- La memoria EEPROM está en un espacio independiente y se requiere del uso de 3 registros I/O para su acceso:

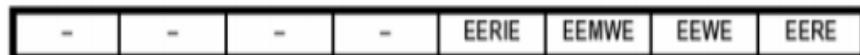
EEAR - Para la dirección (0x1F, 0x1E).

-	-	-	-	-	-	-	EEAR8
EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0

EEDR - Para el dato (0x1D)



EECR - Para el control (0x1C)



\$020	PINA	\$000
\$021	DDRA	\$001
\$022	PORTA	\$002
\$023	PINB	\$003
\$024	DDRB	\$004
\$025	PORTB	\$005
\$026	PINC	\$006
\$027	DDRC	\$007
\$028	PORTC	\$008
\$029	PIND	\$009
\$02A	DDRD	\$00A
\$02B	PORTD	\$00B
\$03F	EECR	\$01F
\$040	EEDR	\$020
\$041	EEARL	\$021
\$042	EEARH	\$022
\$05D	SPL	\$03D
\$05E	SPH	\$03E
\$05F	I TH SV NZ C	\$03F

Circuito de RESET

- Provoca que el sistema pase a un estado inicial conocido.
- Los registros de E/S toman sus valores por defecto.
-
- Power on RESET (Reset de “encendido del dispositivo”)
- RESET externo
- Reloj Perro guardián (Watchdog Reset)
- Detector de “apagones” (Brown:out detectors)

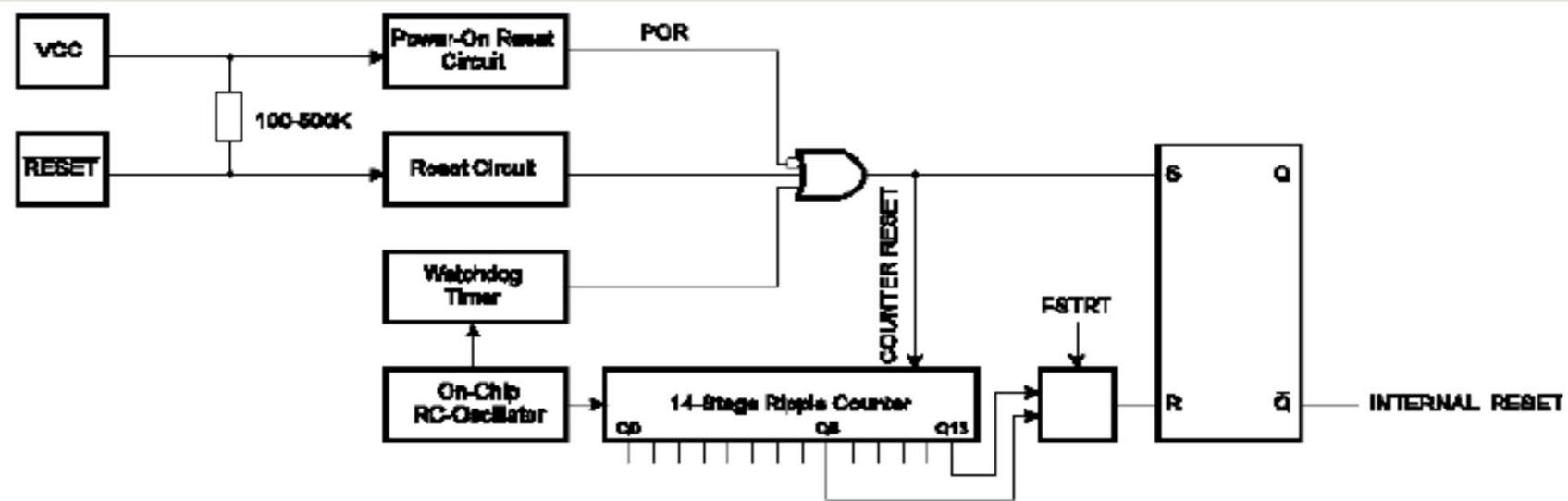
Causas que generan un RESET

El ATmega8 tiene cuatro fuentes de Reset

- **Power-on Reset.** El MCU es inicializado cuando el voltaje de la fuente está por abajo del voltaje de umbral de encendido (V_{POT}).
- **Reset Externo.** El MCU es inicializado cuando un nivel bajo está presente en la terminal RESET por un tiempo mayor que la longitud mínima del pulso.
- **Watchdog Reset.** El MCU es inicializado cuando el Watchdog Timer está habilitado y su periodo termina.
- **Brown-out Reset.** El MCU es inicializado cuando el detector de reducción de voltaje está habilitado y el voltaje VCC de la fuente va por debajo del umbral establecido (V_{BOT}).

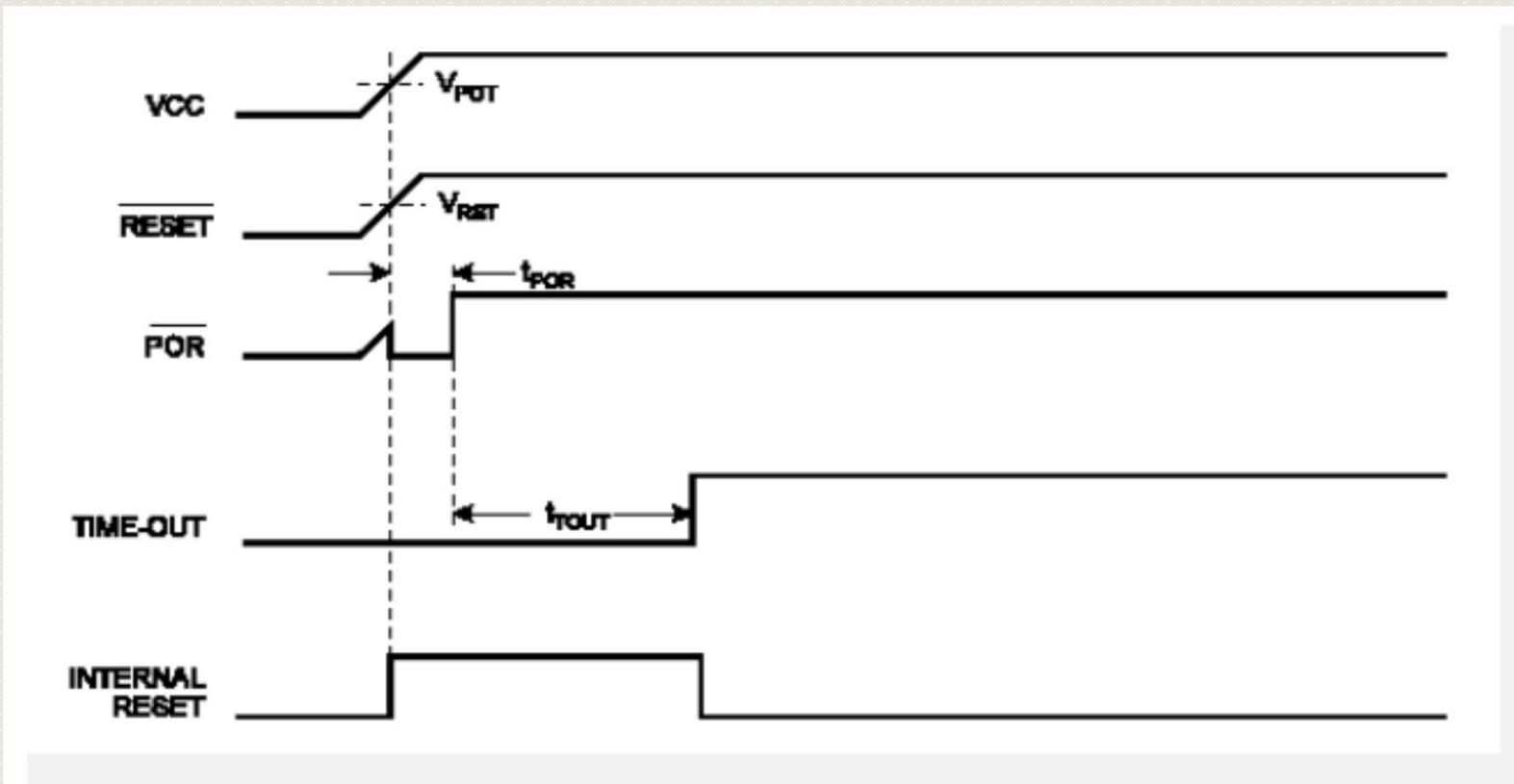
RESET

Una vez que el MCU se recupera de una condición de Reset, espera un tiempo de establecimiento (T_{out} – con un valor típico de 4 ms), antes de recuperar al sistema, para garantizar que los registros tienen su valor inicial.



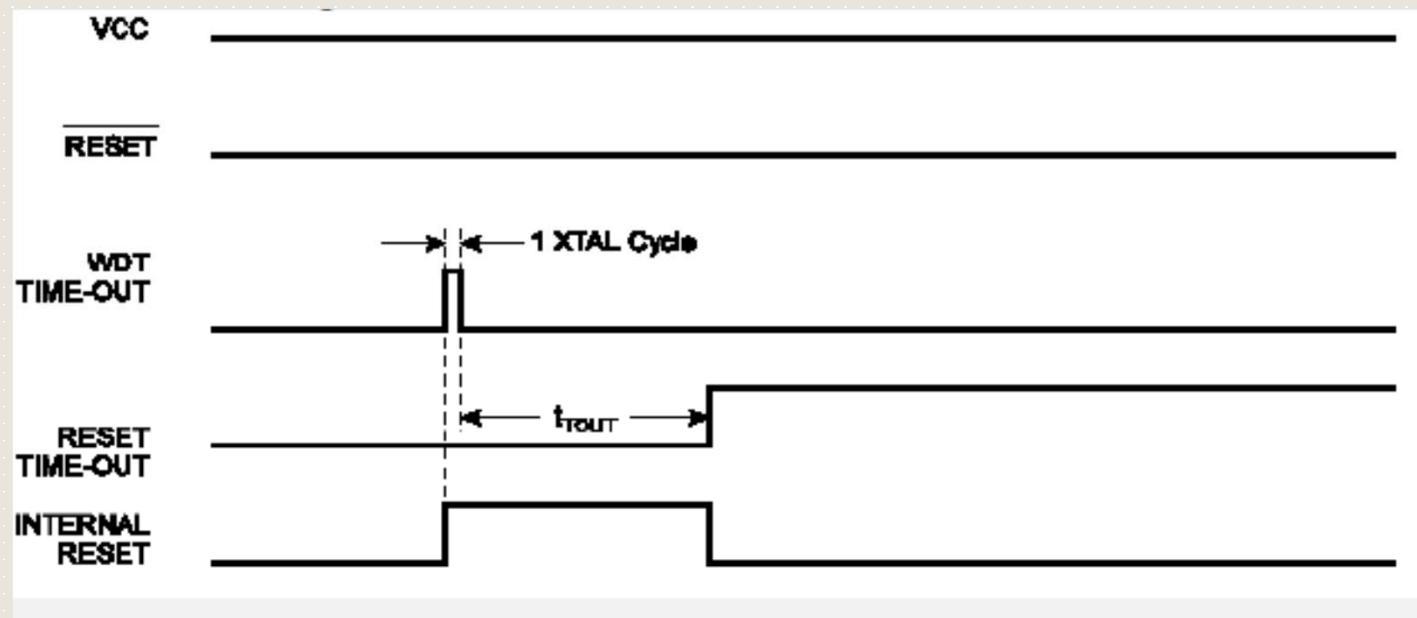
Power-On Reset

- Power-On Reset. Este se activa la primera vez que se da alimentación al microcontrolador, y tiene como objetivo el iniciar y esperar a que la tensión de alimentación alcance un valor adecuado



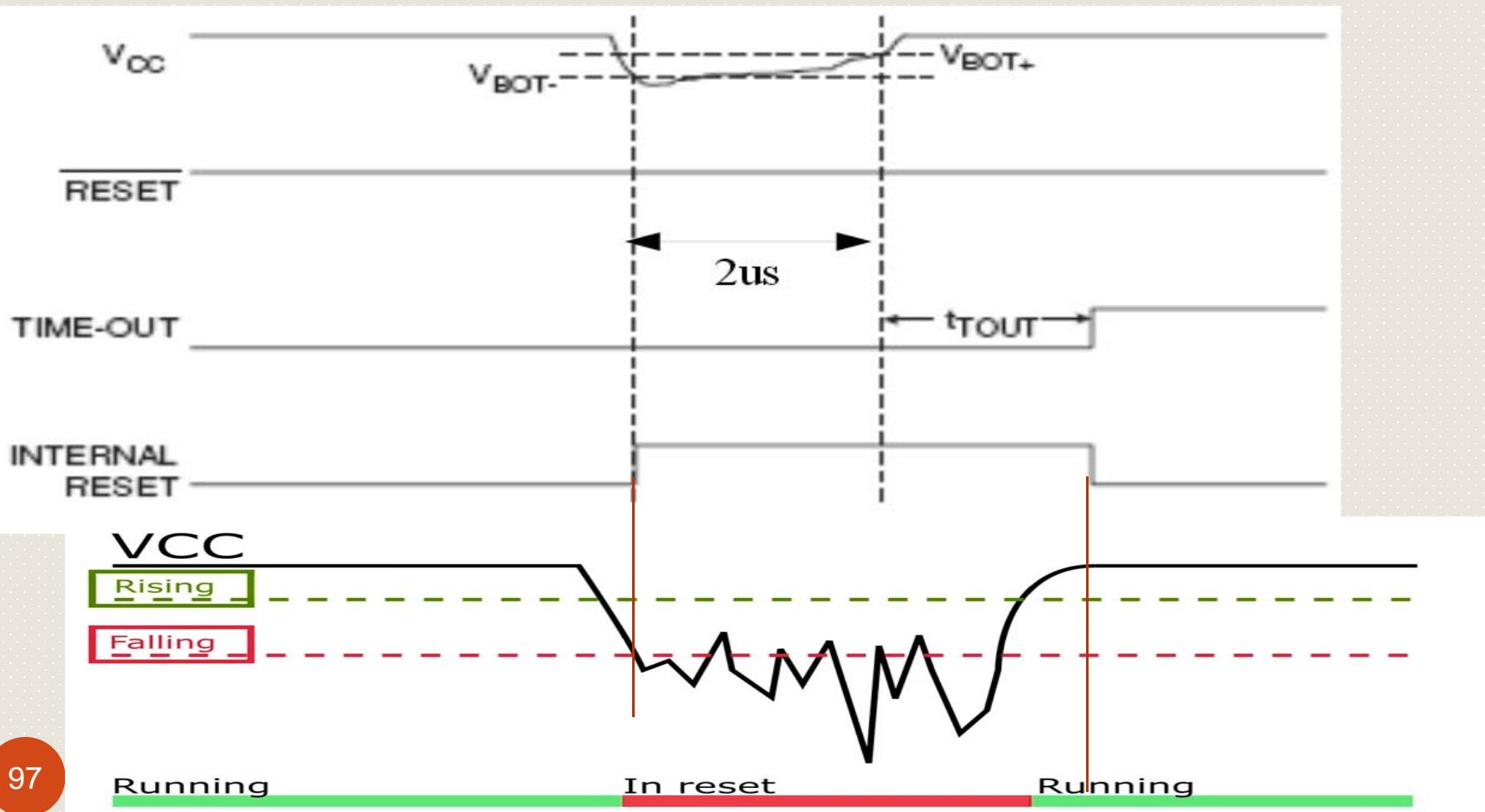
Watchdog Reset

El perro guardián consiste en un contador que cuando alcanza su valor final de cuenta genera un reset interno.



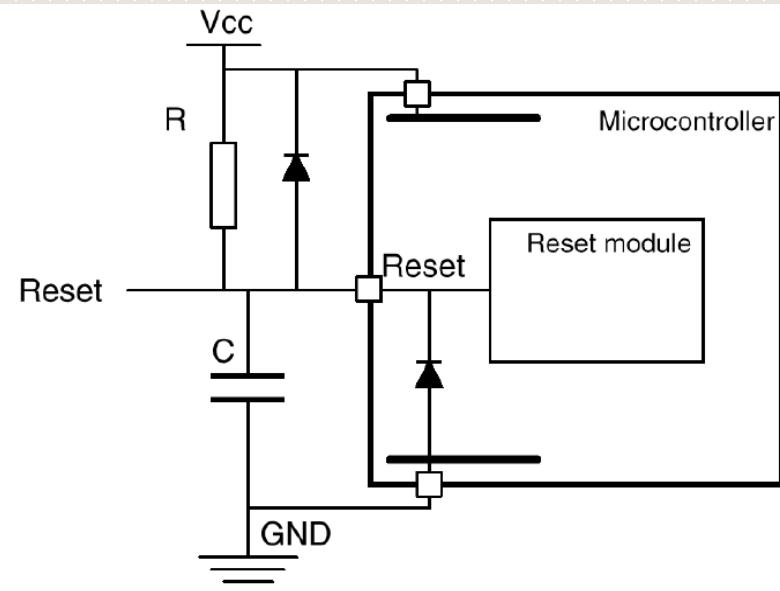
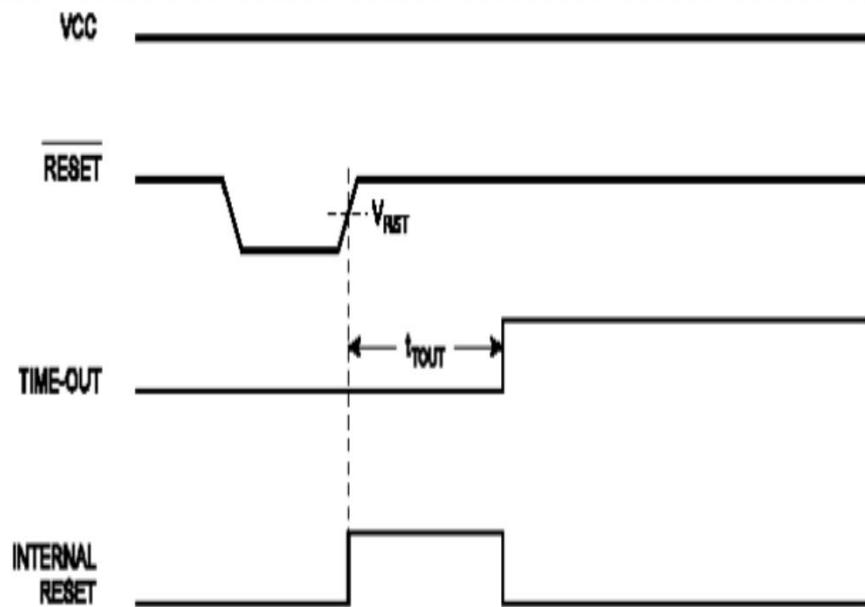
Reset de BROWN_OUT

- Se puede ajustar a 2.7V o 4.0V.
- Vcc debe caer por debajo del nivel V_{BOD-} durante más de 2uS.
- BOD debe habilitar la programación del fusible BOD



Reset Externo

- Inicializa el microcontrolador cuando se activa un pulso en el pin de RESET

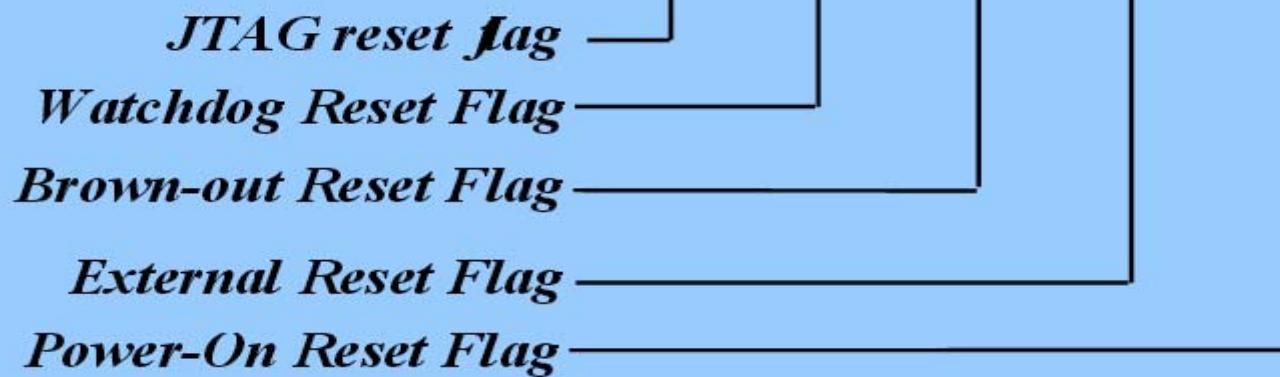


Conexión del
Pin del reset
Reset

ago-18

El Control de MCU y Registro de Estado MCUCSR proporciona información en cuanto al origen del Reset que más recientemente se produjo

Bit	7	6	5	4	3	2	1	0	MCUCSR
Read/Write	JTD	-	-	JTRF	WDRF	BORF	EXTRF	PORF	
Initial Value	0	0	0						



Puertos de I/O Paralelos

4 Puertos A, B, C y D

Cada puerto tiene 3
REGISTROS:

1. DE ENTRADA → PINx

2. SENTIDO DEL DATO → DDRx

3. EL DE SALIDA → PORTx

\$020	PINA	\$000
\$021	DDRA	\$001
\$022	PORTA	\$002
\$023	PINB	\$003
\$024	DDRB	\$004
\$025	PORTB	\$005
\$026	PINC	\$006
\$027	DDRC	\$007
\$028	PORTC	\$008
\$029	PIND	\$009
\$02A	DDRD	\$00A
\$02B	PORTD	\$00B
\$05D	SPL	\$03D
\$05E	SPH	\$03E
\$05F	I T H S V N Z C	\$03F

Puerto Básico

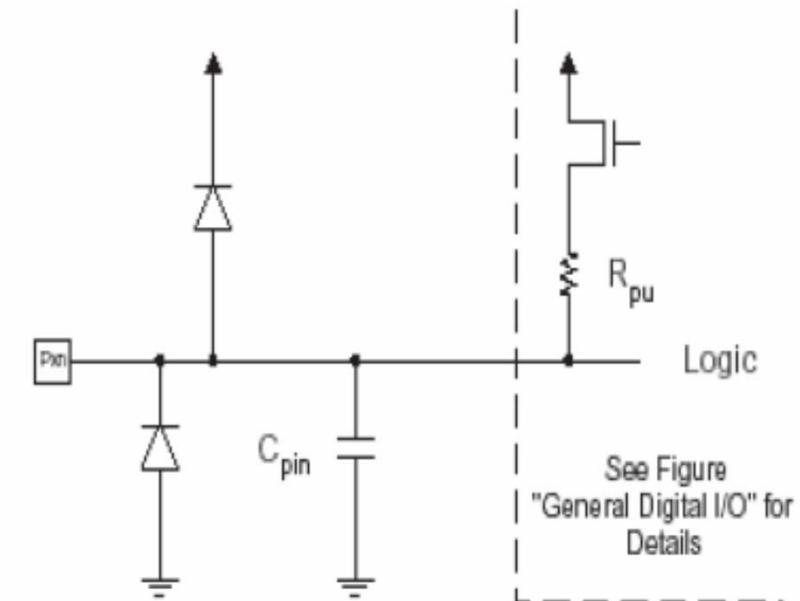
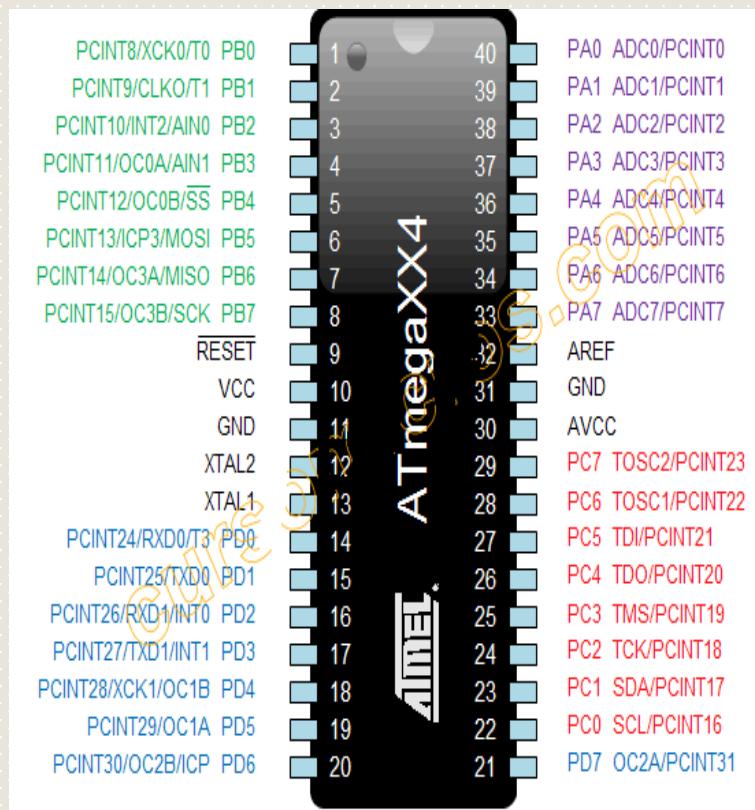
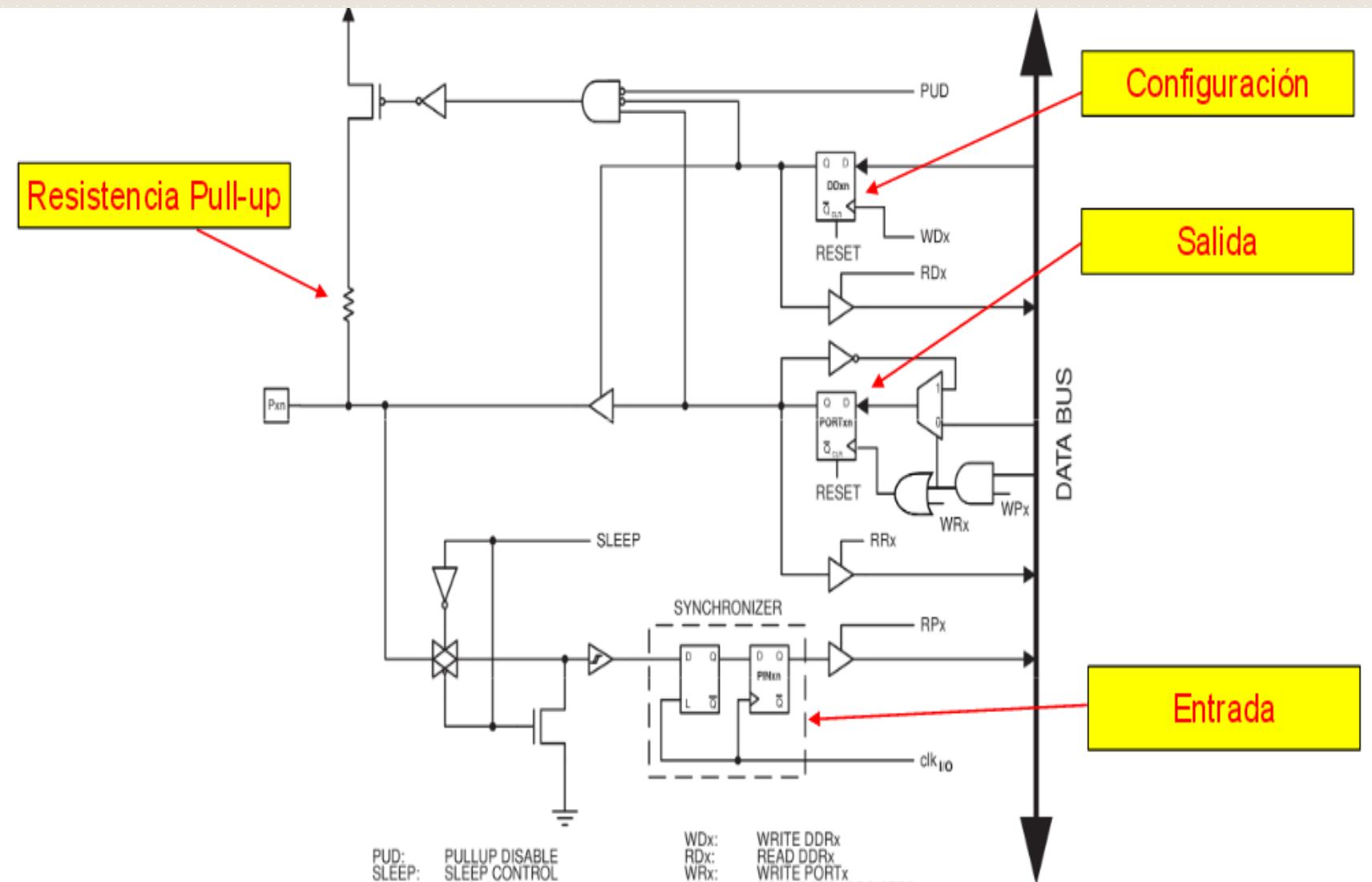


Fig.4.1 Diagrama equivalente de un pin de E/S



Registros de los puertos

- **Registro DDRx.**- El registro DDRx configura la dirección. Escribir un uno a un BIT de este registro, configura la Terminal como una salida; al escribir un cero, se le convierte en entrada.
- **Registro PINx.**- Lee el estado de PORTx, independientemente del estado de DDRx. Básicamente sirve para leer el estado de la Terminal del puerto cuando este se ha configurado como entrada.
- **Registro PORTx.**- Si la Terminal está configurada como salida, escribir un uno o un cero en el bit correspondiente de este registro, ocasiona que la salida en esta Terminal sea uno o cero.
- Si la Terminal está configurada como entrada, escribir un uno en el bit correspondiente de este registro, habilita el resistor de pull-up. Escribir un cero, estando configurado como entrada, deshabilita el resistor de pull-up.

Tensiones y corrientes

- Cuando actúan como salidas, los pines pueden entregar tensiones de hasta Vcc.
- Cuando actúan como entradas pueden manejar niveles de hasta 0.5V por encima de Vcc.
- El diseño de los pines incluye diodos internos de sujeción que les permiten soportar tensiones mucho mayores que Vcc o inferiores que GND, siempre que la corriente no sobrepase del orden de los micro Amperios.
- Cada pin de E/S puede soportar picos de corriente de hasta 40 mA, pero en estado estable cada pin de puerto puede suministrar o recibir hasta 20 mA de corriente cuando $V_{CC} = 5V$
- $V_{CC} = 3V$ hasta 10 mA cuando.
- Esta capacidad no puede estar presente en todos los pines al mismo tiempo.

Tensiones y corrientes

- Para los ATmegaXX8 la distribución de límites es un poco diferente. (Los pines ADC7 y ADC6 no están presentes en encapsulados DIP.)
- La suma de las corrientes suministradas por los pines PC0 - PC5, ADC7, ADC6 no debería exceder los 100 mA.
- La suma de las corrientes suministradas por los pines PB0 - PB5, PD5 - PD7, XTAL1 y XTAL2 no debería exceder los 100 mA.
- La suma de las corrientes suministradas por los pines PD0 - PD4 y RESET no debería exceder los 100 mA.
- La suma de las corrientes recibidas por los pines PC0 - PC5, PD0 - PD4, ADC7 y RESET no debería exceder los 150 mA.
- La suma de las corrientes recibidas por los pines PB0 - PB5, PD5 - PD7, ADC6, XTAL1 y XTAL2 no debería exceder los 150 mA.

Puertos

Data Memory

32 Registers	0x0000
64 I/O Registers	0x0020
160 Ext I/O Reg.	0x0060
Internal SRAM (512/1024/1024/2048 x 8)	0x0100
	0x04FF/

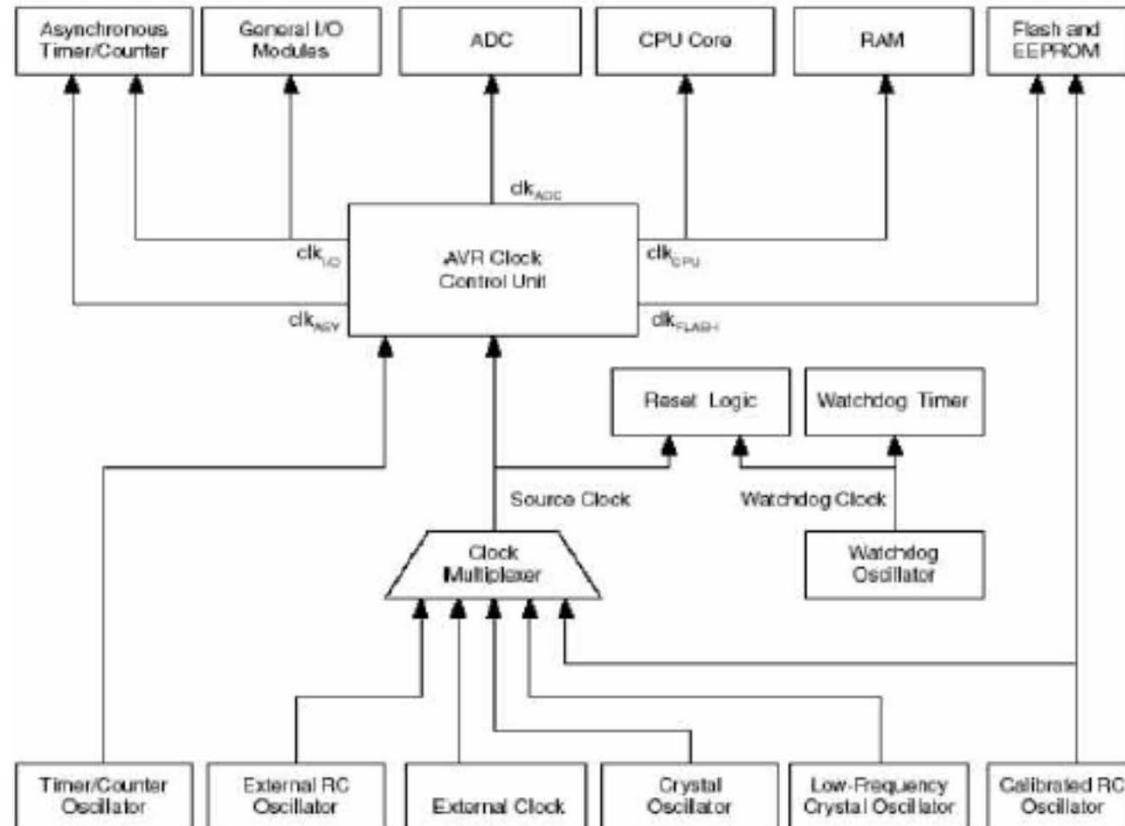
REG.	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	I/O	SRAM
PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	0x1B	(0x3B)
DDRA	DDRA7	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0	0x1A	(0x3A)
PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	0x19	(0x39)
PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	0x18	(0x38)
DDRB	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0	0x17	(0x37)
PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	0x16	(0x36)
PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	0x15	(0x35)
DDRC	DDRC7	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0	0x14	(0x34)
PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	0x13	(0x33)
PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	0x12	(0x32)
DDRD	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0	0x11	(0x31)
PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	0x10	(0x30)

puertos

DDRxn	PORTxn	PUD	E/S	PULL-UP	COMENTARIO
0	0	X	E	NO	ALTA IMPEDANCIA
0	1	0	E	SI	ENTREGA CORRIENTE
0	1	1	E	NO	ALTA IMPEDANCIA
1	0	X	S	NO	V_{OL} (DRENAJE)
1	1	X	S	NO	V_{OH} (FUENTE)

CLOCK

- Se tienen diferentes fuentes para manejar al oscilador interno y a la vez, la señales de reloj se distribuyen por los diferentes módulos:



Assembler Directives

.EQU and .SET

- **.EQU *name* = *value***

- *Example:*

```
.EQU      COUNT  = 0x25
LDI      R21, COUNT           ;R21 = 0x25
LDI      R22, COUNT + 3       ;R22 = 0x28
```

- **.SET *name* = *value***

- *Example:*

```
.SET      COUNT  = 0x25
LDI      R21, COUNT           ;R21 = 0x25
LDI      R22, COUNT + 3       ;R22 = 0x28
.SET      COUNT  = 0x19
LDI      R21, COUNT           ;R21 = 0x19
```

Assembler Directives

.INCLUDE

- **.INCLUDE “filename.ext”**

Table 2-6: Some of the common AVR^s and their include files

MEGA	TINY	Special Purpose
Mega8	m8def.inc	Tiny11_tn11def.inc
M32def.inc		
<pre>.equ SREG = 0x3f .equ SPL = 0x3d .equ SPH = 0x3eequ INT_VECTORS_SIZE = 42 ; size in words</pre>		

Program.asm

```
.INCLUDE "M32DEF.INC"
LDI R20, 10
OUT SPL, R20
```

Assembler Directives

.ORG

- .ORG address

Program.asm

```
.ORG 0
LDI R16, 0x25
.ORG 0x7
LDI R17, 0x34
LDI R18, 0x31
```



00	E205
01	0000
02	0000
03	0000
04	0000
05	0000
06	0000
07	E314
08	E321
09	0000
0A	0000

Segmento de Código

- Esta directiva le dice al ensamblador que el siguiente código o expresiones deberán colocarse en la memoria de programa. Esto es necesario cuando la directiva .dseg se usa para declarar constantes y datos.
 - Sintaxis: **.cseg**

Reserva De Memoria Programa

1. Con esta directiva se podrá colocar valores de las constantes en la memoria de programa en una dirección conocida, por ejemplo, números seriados, cadenas para un menú, tablas. Ellos son tratados como bytes y deberán estar dentro de un rango de 8 bits. Cada directiva .db se colocara al inicio de una nueva palabra en la memoria de programa. Así, dos directivas .db de un solo byte como argumento usaran dos palabras, Mientras que una directiva .db con dos bytes como argumentos usaran solamente una palabra. Ver el ejemplo siguiente:

Sintaxis:

.db expresión1, expresión2, expresión3,...

Ejemplos:

Las cadenas pueden ser colocadas en la memoria de programa con solo la directiva .db:

.db "Hello World!"

Es equivalente a 6 palabras de datos o 12 bytes. Si deseas que a la cadena se le añada

un 0 al final de la misma deberás colocarla así:

.db "Hello World!", 0

.db "Hello\n" // is equivalent to:

-Palabra de Datos

- La directiva .dw trabaja como la directiva .db, pero se usará una palabra para cada valor.
- Sintaxis: **.dw expresión1**
 - **.dw `Ab'**

Directivas de la Memoria

Datos

SRAM

.Segmento de Datos

Le dice al ensamblador que el siguiente texto es usado para establecer la SRAM.

Para cambiarse al segmento de código de nuevo, use .cseg.

.dseg

- Reserva de Memoria Volátil (.byte)

Reserva un número de bytes en el espacio de la SRAM para una etiqueta.

Esta directiva es solamente permitida en segmentos de datos.

Sintaxis: **.byte tamaño**

Ambas directivas, dseg y byte, se complementan con .ORG para establecer el contador de la

localidad SRAM a un valor específico dentro de .dseg. o con .byte puedes definir las localidades de la SRAM en direcciones específicas con un tamaño específico.

Directivas de Código

.MACRO

.endm / .endmacro

"Fin de la Macro"; Esta le dice al ensamblador que una macro previamente comenzó y finalice aquí. Solamente se usa cuando al inicio se tenga la directiva .macro.
.macro
Esta inicia un código de macro.

LSP R16 , RAMEND

.MACRO	LSP
	LDI @0, low(@1)
	OUT SPL, @0
	LDI @0, high(@1)
	OUT SPH, @0
.ENDMACRO	

Directivas de Código

• .list

El ensamblador por default crea un archivo de lista (una combinación de código fuente, código de operación y constantes). Aunada con .nolist puedes especificar que partes del archivo se mostraran en el archivo de lista.

• .listmac

Esta directiva creara una macro expansión del archivo lista. Por default, solamente se ve que macro es llamada y que argumentos se usan. Para poder ver que esta pasando durante la ejecución, para propósitos de debugging.

• .nolist

Directiva que detiene la generación del archivo de lista.

RESUMEN

Some AVR Assembler Directives

Directives	Description
BYTE	Reserve a Byte for a Variable
CSEG	Define the Code Segment Section
DB	Define Constant Byte(s)
DEF	Define a Symbolic Name
DEVICE	Define which Microcontroller to Assemble for
DSEG	Define the Data Segment Section
DW	Define Constant Word(s)
ENDMACRO	Signifies the End of a Macro
EQU	Set a Symbol Equal to an Expression
ESEG	Define the EEPROM Section
Exit	Exit from a File
INCLUDE	Read Source Code from a File
LIST	Turn List Generation ON
LISTMAC	Turn Macro Expression ON
MACRO	Signifies the Beginning of a Macro
NOLIST	Turn List Generation OFF
ORG	Set Program Origin
SET	Set Symbol to an Expression

Ejemplo II

```
.include "m8515def.inc"
.org $00

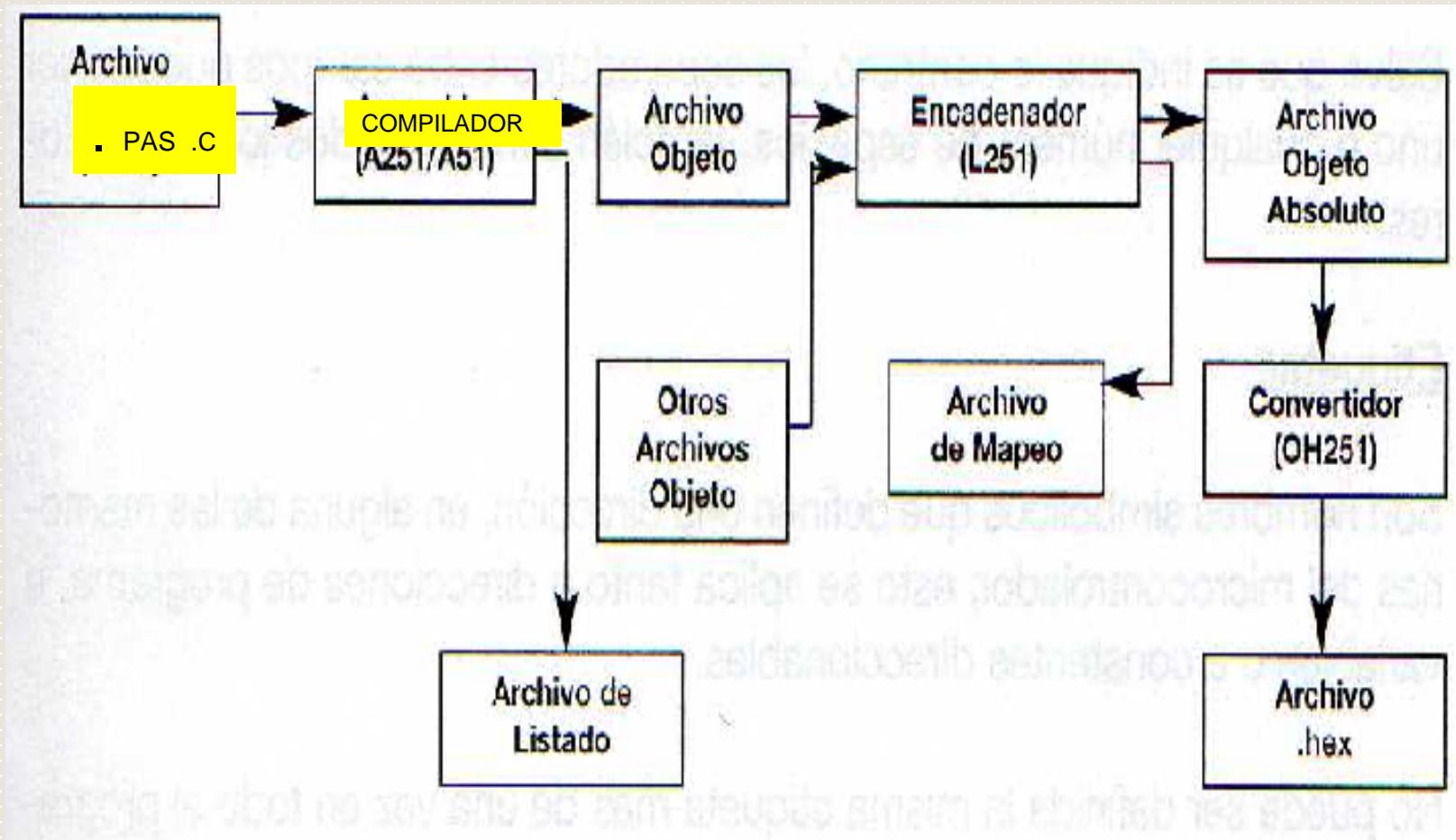
;Initialize the microcontroller stack pointer
LDI    R16, low(RAMEND)
OUT    SPL, R16
LDI    R16, high(RAMEND)
OUT    SPH, R16

;Configure portB as an output put
LDI    R16, 0xFF
OUT    DDRB, R16

;Toggle the pins of portB
LDI    R16, 0xFF
OUT    PORTB, R16
RCALL Delay
LDI    R16, 0x00
OUT    PORTB, R16
RCALL Delay

;Delay subroutine
Delay: LDI    R17, 0xFF
loop:   DEC    R17
        BRNE  loop
        RET
```

Ensamblado de un programa fuente



Traducción en dos pasadas

Primera pasada:

- Contador de Ubicación
inicio =0

Directivas DB ,DW inc. PC

- Crea tabla de símbolos
etiquetas
ctes.

Segunda pasada:

- Traduce cada sentencia del lenguaje ensamblador al combinar los equivalentes numéricos de los códigos de operación, especificadores de registros y rótulos de la tabla de símbolos en una instrucción legal.

Traducción en dos pasadas

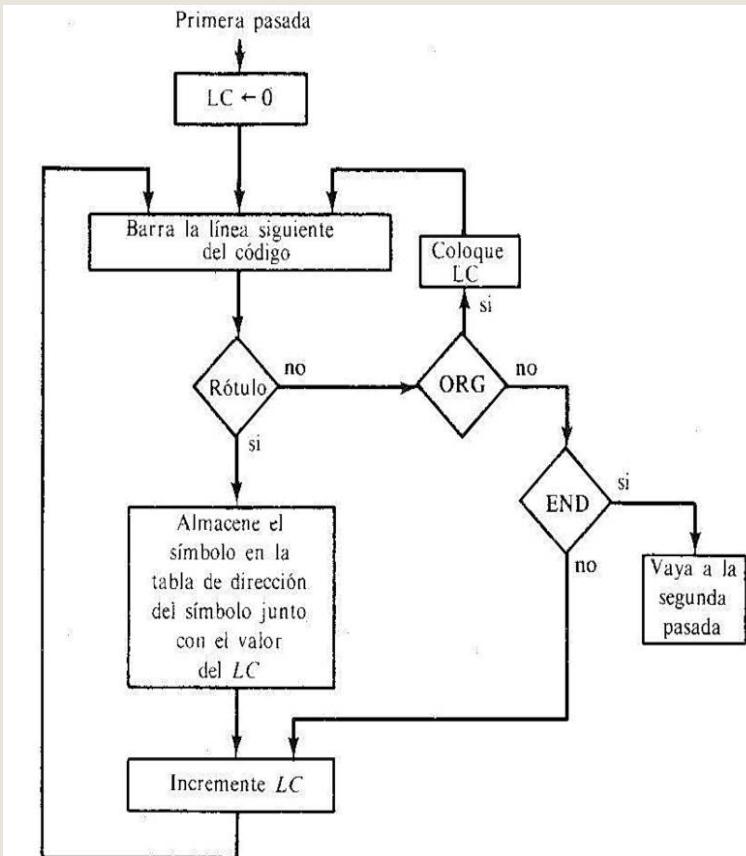


Figura 6-1 Diagrama de flujo para la primera pasada del ensamblador.

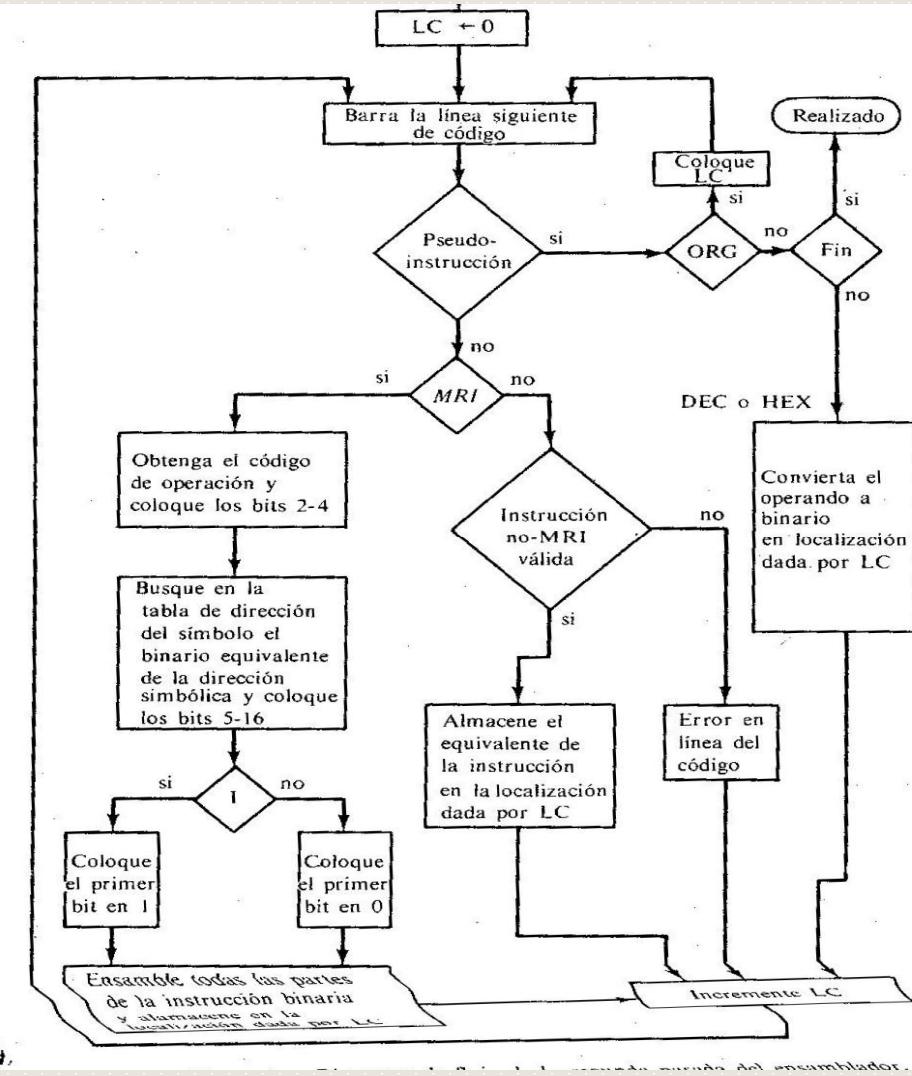
Primera pasada:

- Contador de Ubicación
inicio =0

Directivas DB ,DW inc.
PC

- Crea tabla de símbolos
etiquetas
ctes.

Traducción en dos pasadas



Segunda pasada:

- Traduce cada sentencia del lenguaje ensamblador al combinar los equivalentes numéricos de los códigos de operación, especificadores de registros y rótulos de la tabla de símbolos en una instrucción legal.

Formato Intel Hex

: 03 0030 00 02337A 1E

Marca de reg.	Tamaño de reg.	Direc. de inicio	Tipo de registro	Bytes de datos	Byte checksum
:	##	aaaa	tt	dd....dd	Cc

- : Es el carácter de inicio de registro.
- ## Es un valor ASCII hexadecimal de dos dígitos que indica el tamaño del registro. Es decir, el número de bytes de datos que lo compone (máx. 256). Si el registro es del tipo 01, este campo debe valer 00.
- aaaa Cuatro dígitos hexadecimales expresados en ASCII que corresponden con la dirección inicial de memoria donde deben almacenarse, de forma consecutiva, los bytes de datos. Si el registro es del tipo 01, este campo debe valer 0000.
- tt Es un valor ASCII hexadecimal de dos dígitos que representan el tipo de registro.
- dd...dd Son dos caracteres ASCII por byte de datos. Habrá tantos bytes por registro como indique el campo ##. Si el tipo de registro es 01 (último registro) no habrá ningún byte de datos.
- cc Son dos caracteres ASCII hexadecimal que representan el byte del checksum. Este byte se obtiene de forma tal que, al sumar todos los bytes de los distintos campos de un registro comenzando desde ## y finalizando con el propio cc, se obtenga siempre el valor 00.

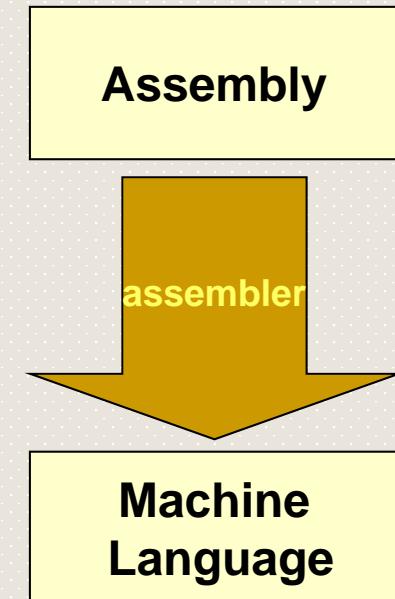
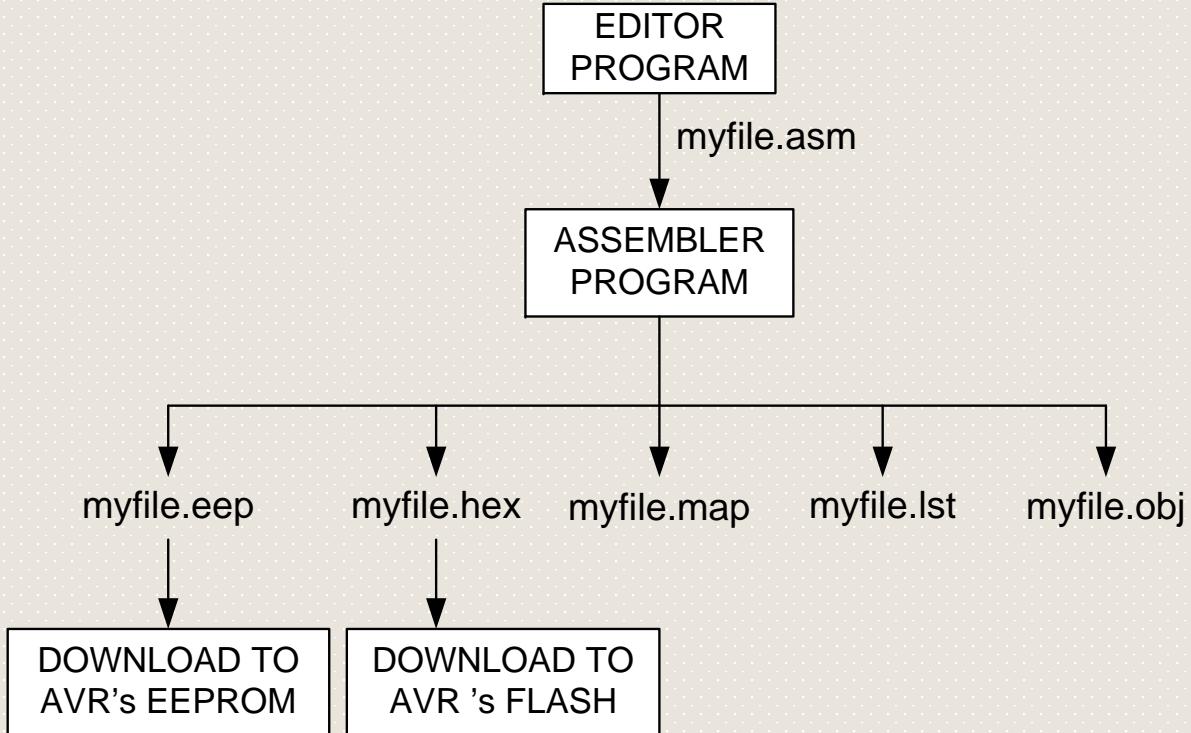
Formato Intel Hex

- :0300300002337A1E
 - **Record Length:** 03 (3 bytes of data)
 - **Address:** 0030 (the 3 bytes will be stored at 0030, 0031, and 0032)
 - **Record Type:** 00 (normal data)
 - **Data:** 02, 33, 7A
 - **Checksum:** 1E

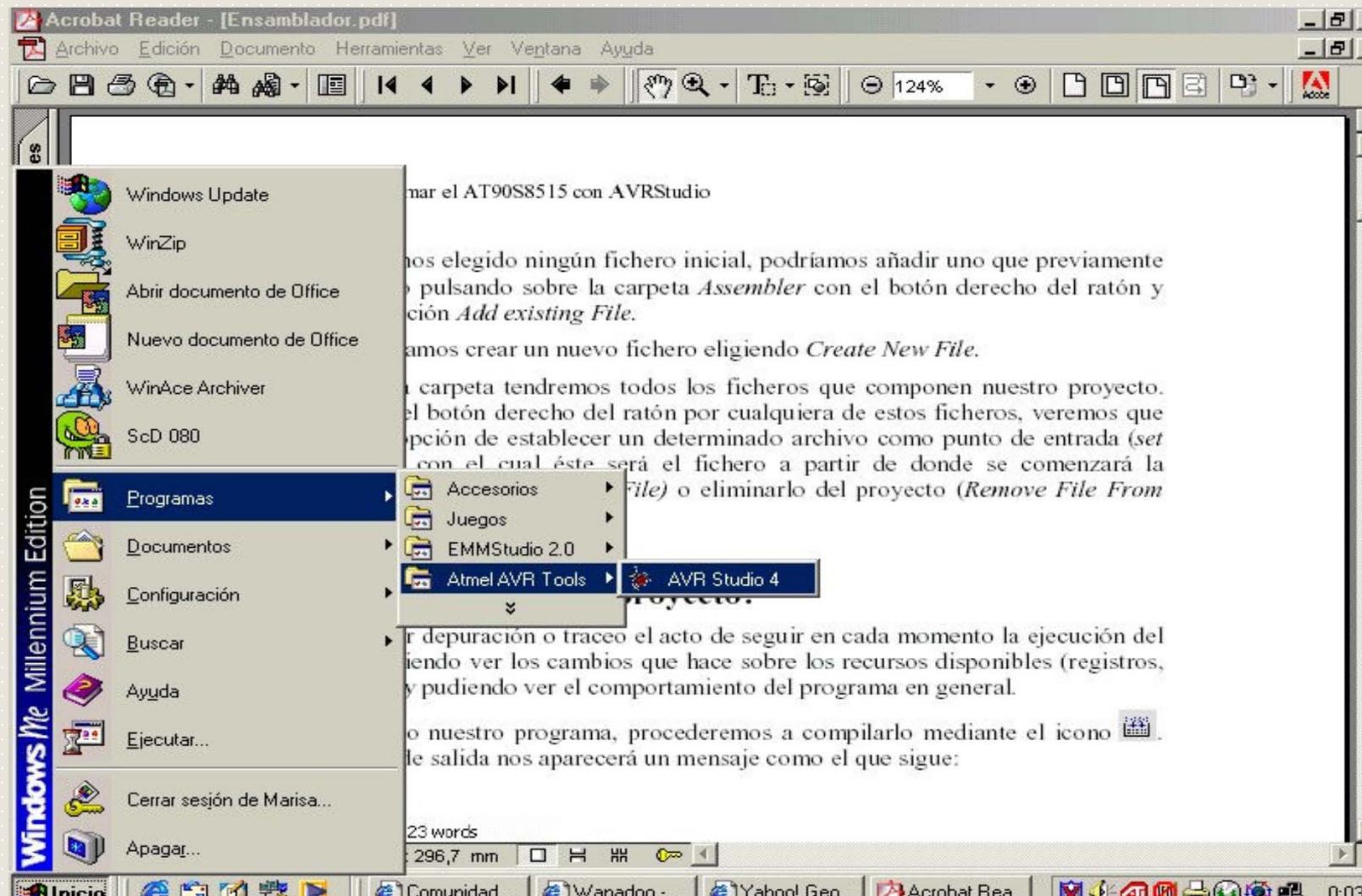
Formato Intel Hex

- :10010000214601360121470136007EFE09D2190140
- :100110002146017EB7C20001FF5F16002148011988
- :10012000194E79234623965778239EDA3F01B2CAA7
- :100130003F0156702B5E712B722B732146013421C7
- :00000001FF ## 00h
 aaaa 0000h
 tt 01h (último registro)
 3er registro
 cc FFh (00h+00h+00h+01h+FFh = 00)

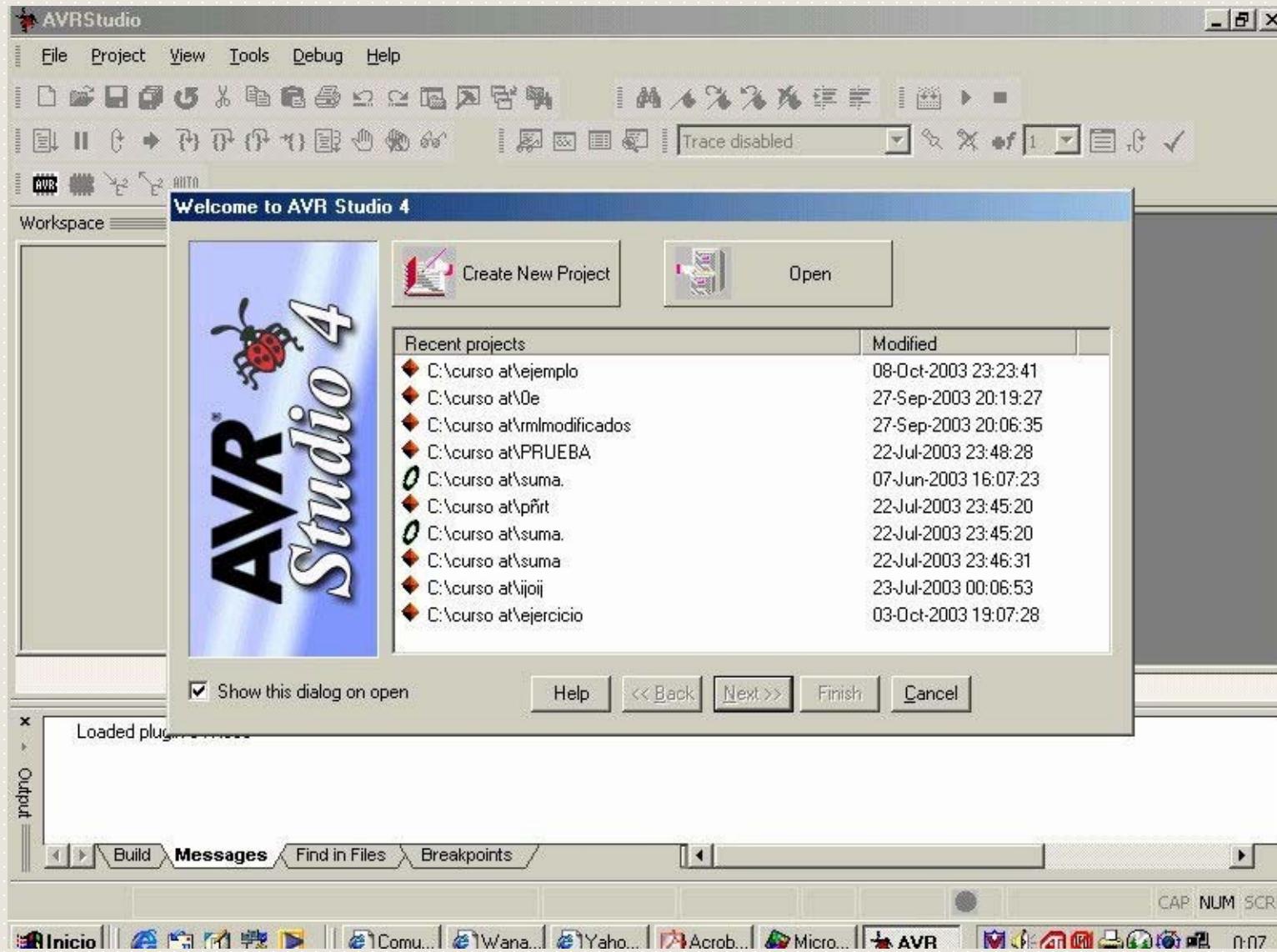
Assembler



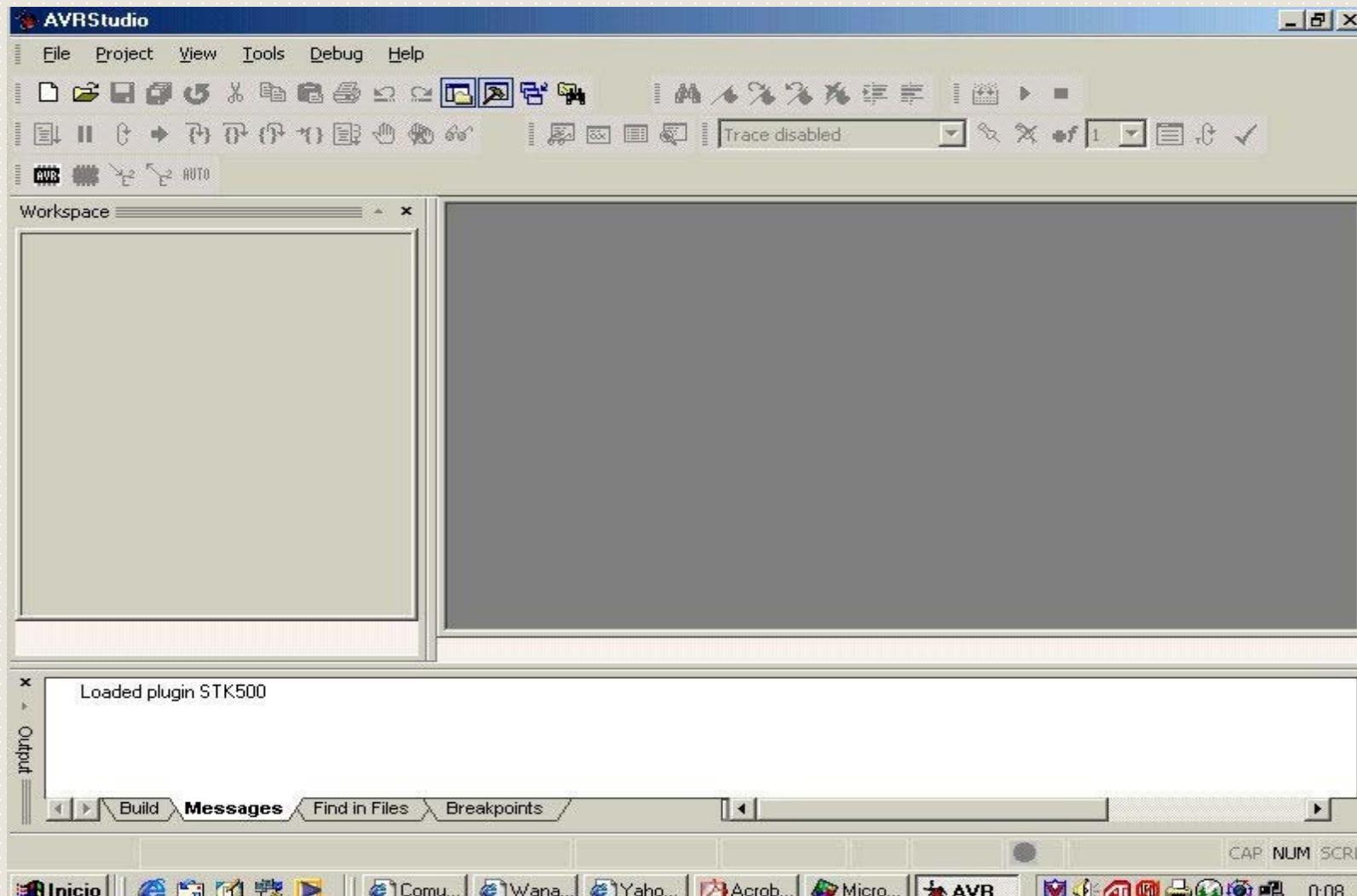
AVR Studio 4



AVR Studio 4



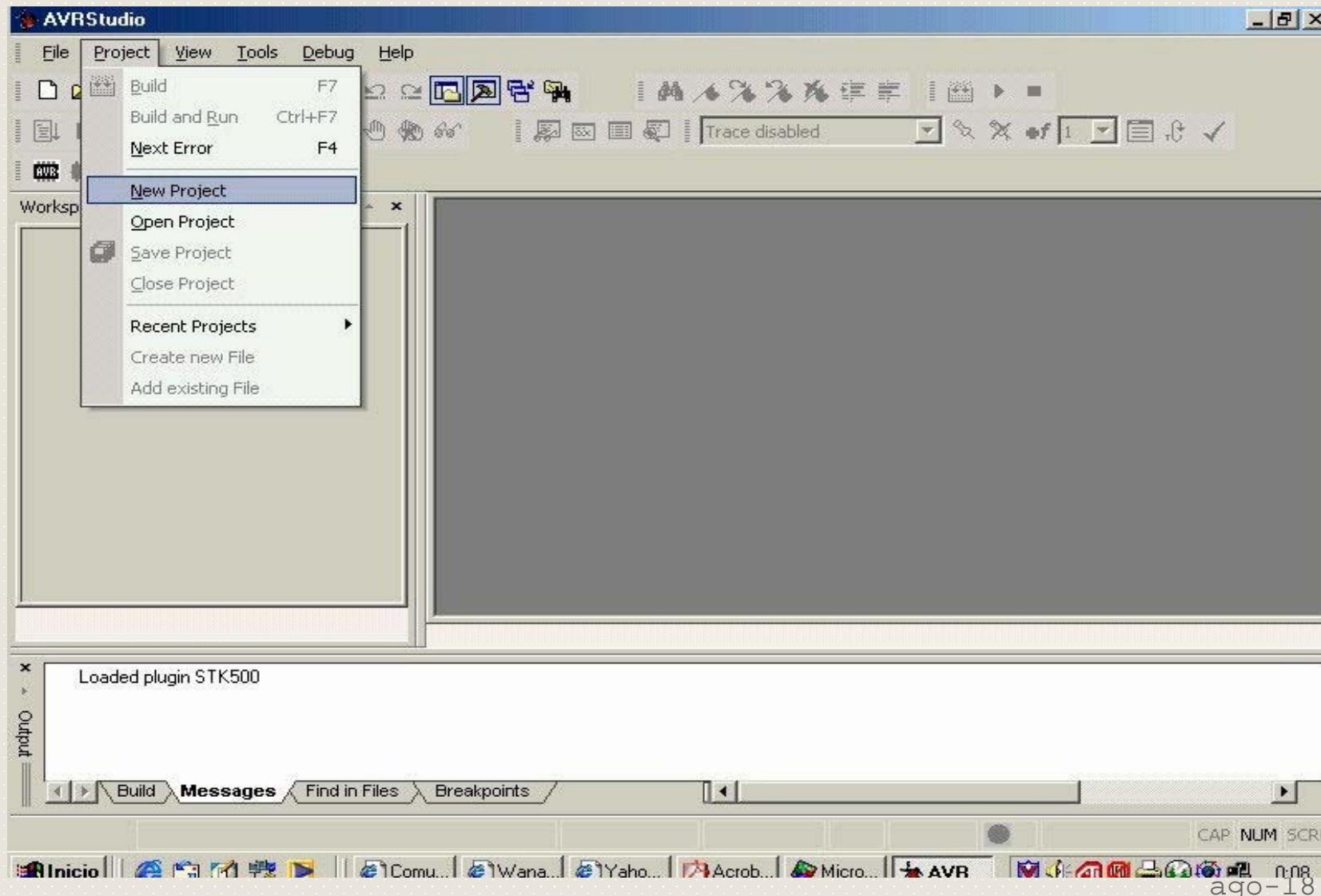
AVR Studio 4



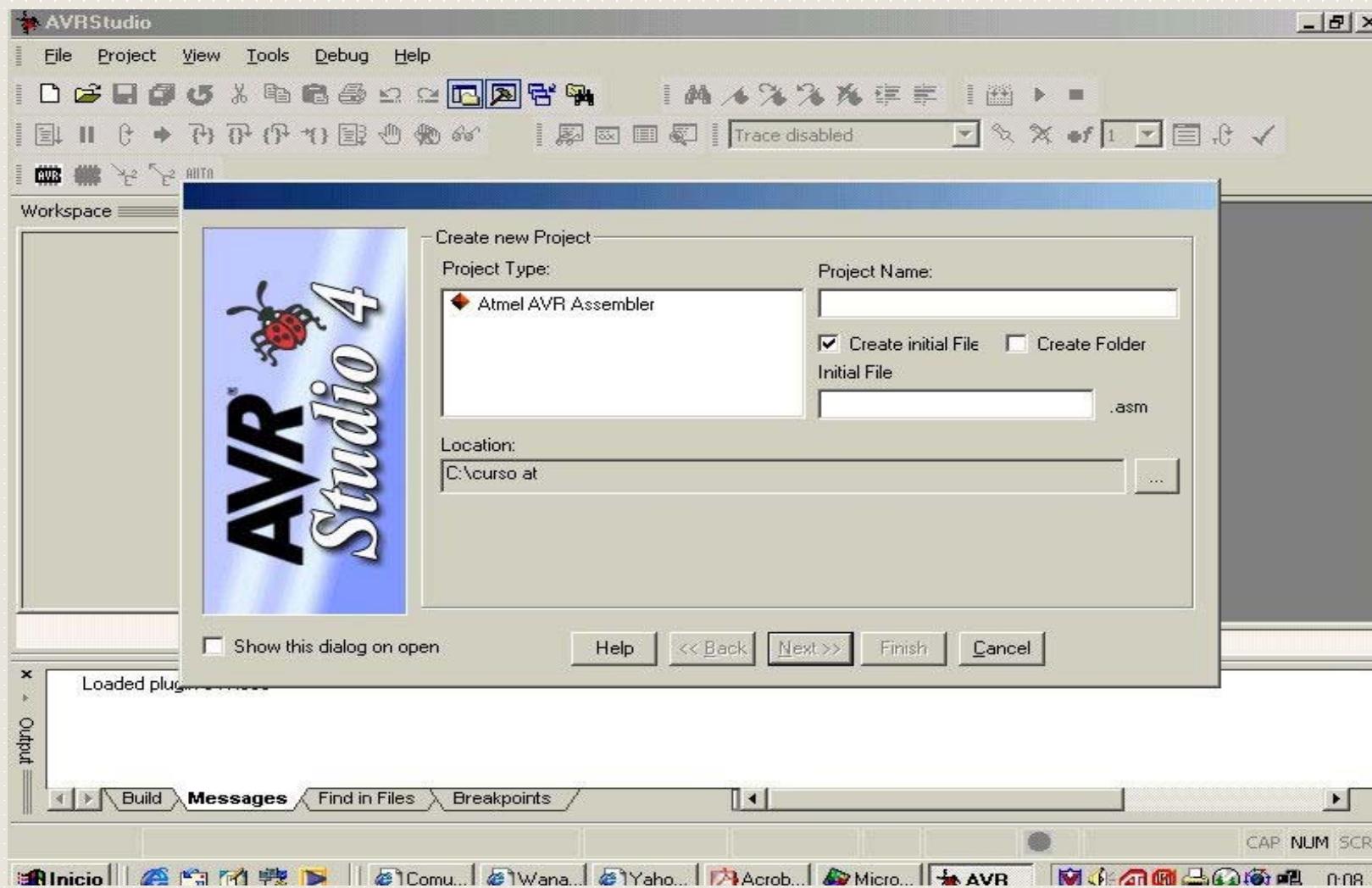
131

seleccionar "Project" -> "New project" ago-18

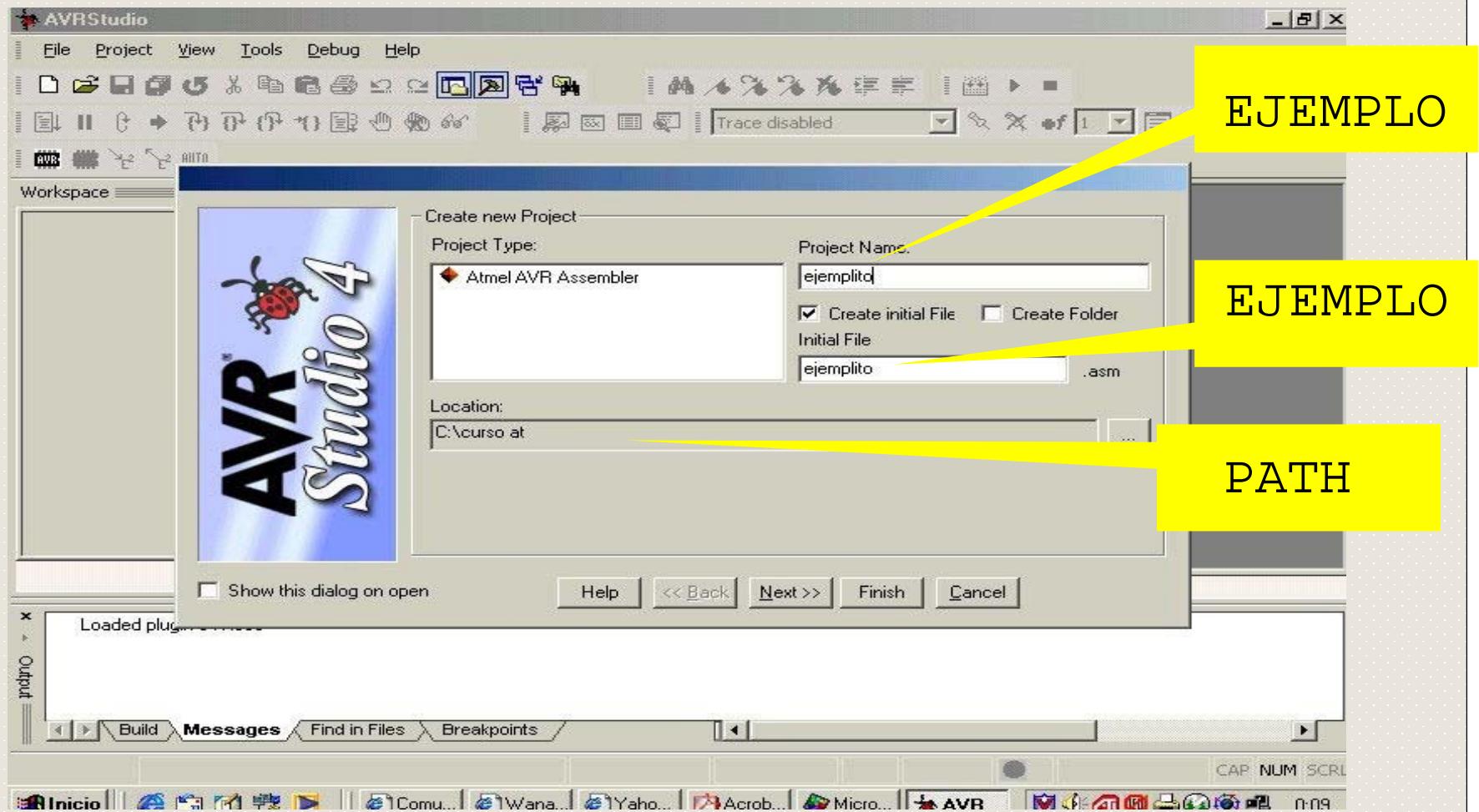
AVR Studio 4



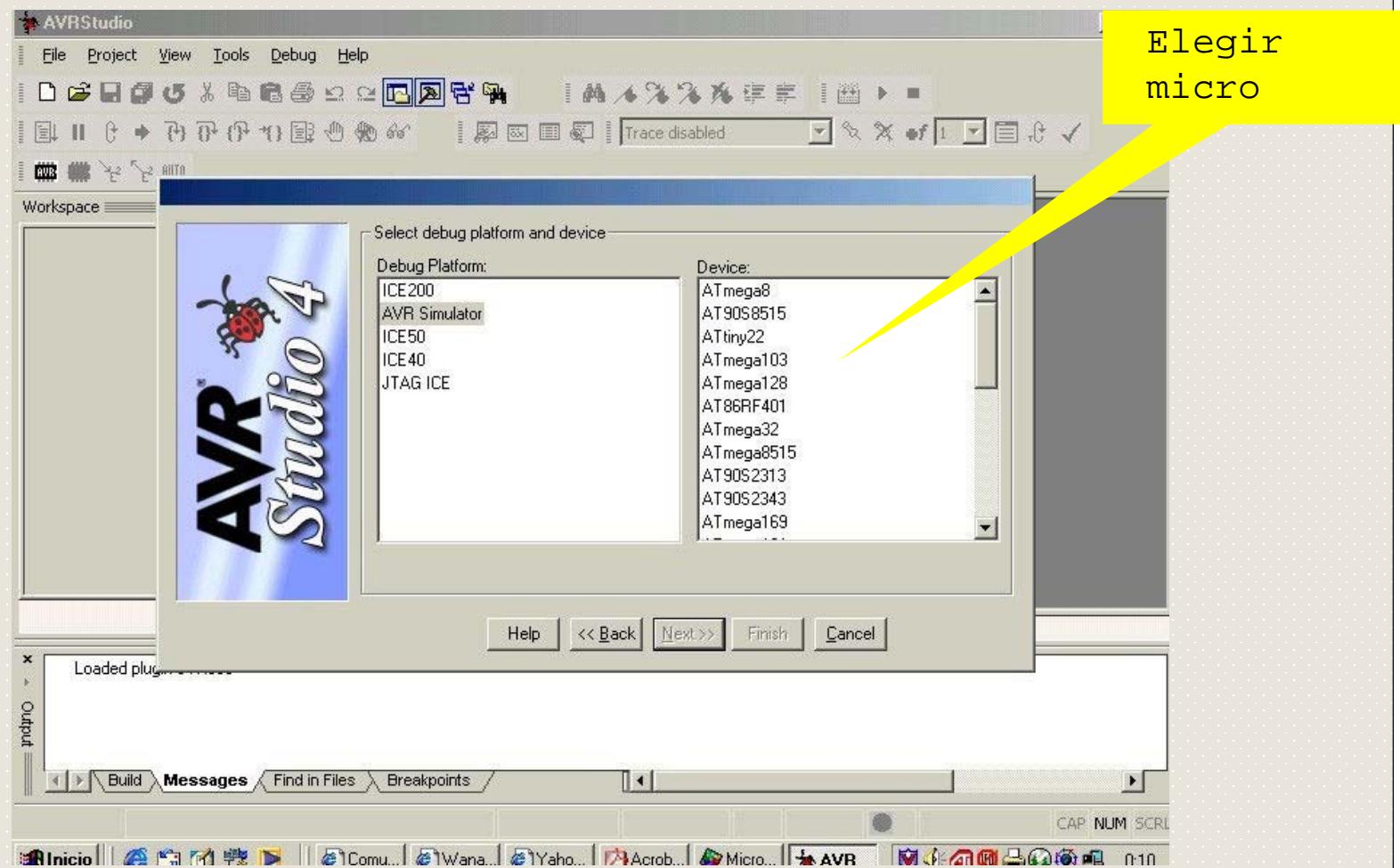
AVR Studio 4



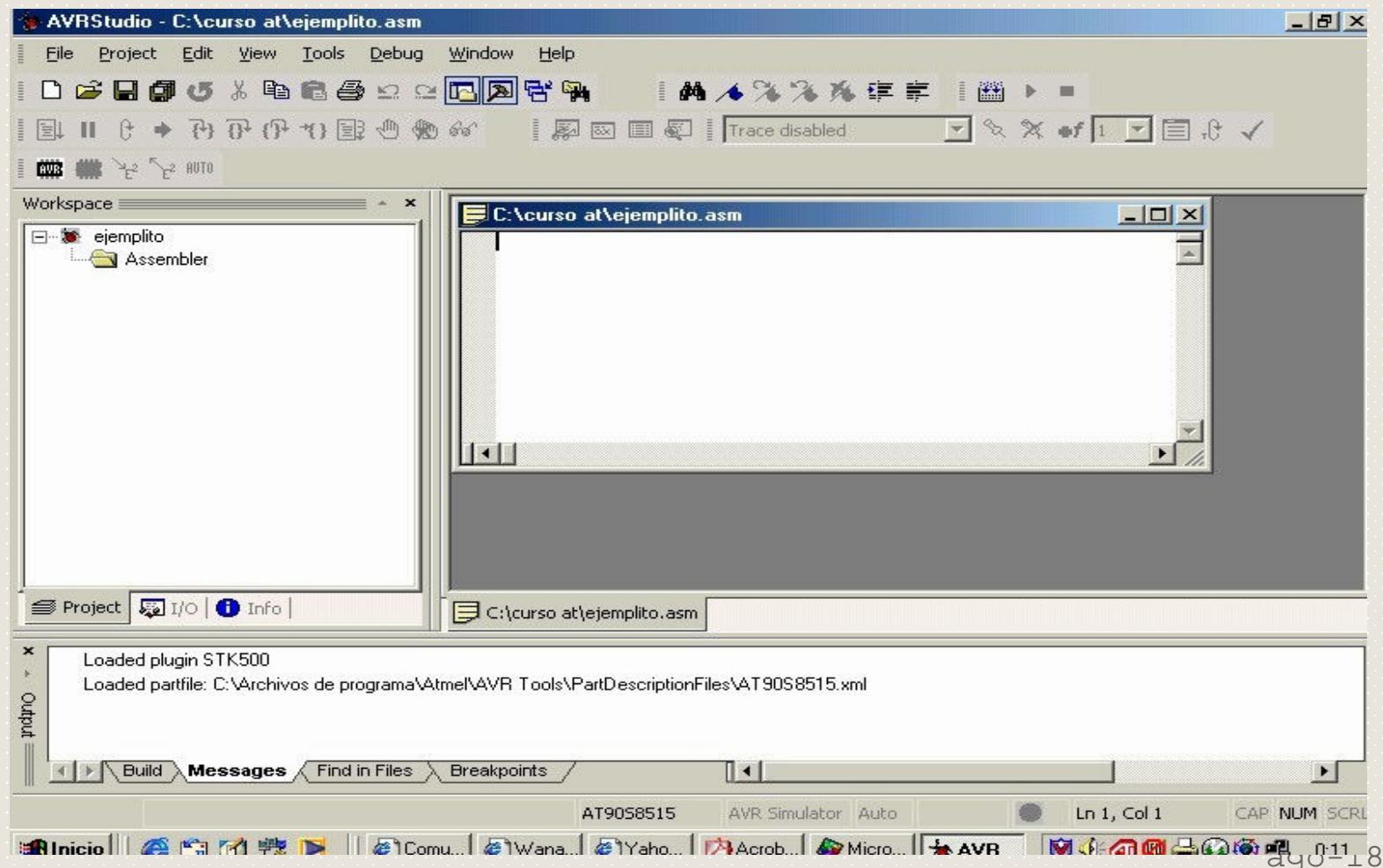
AVR Studio 4



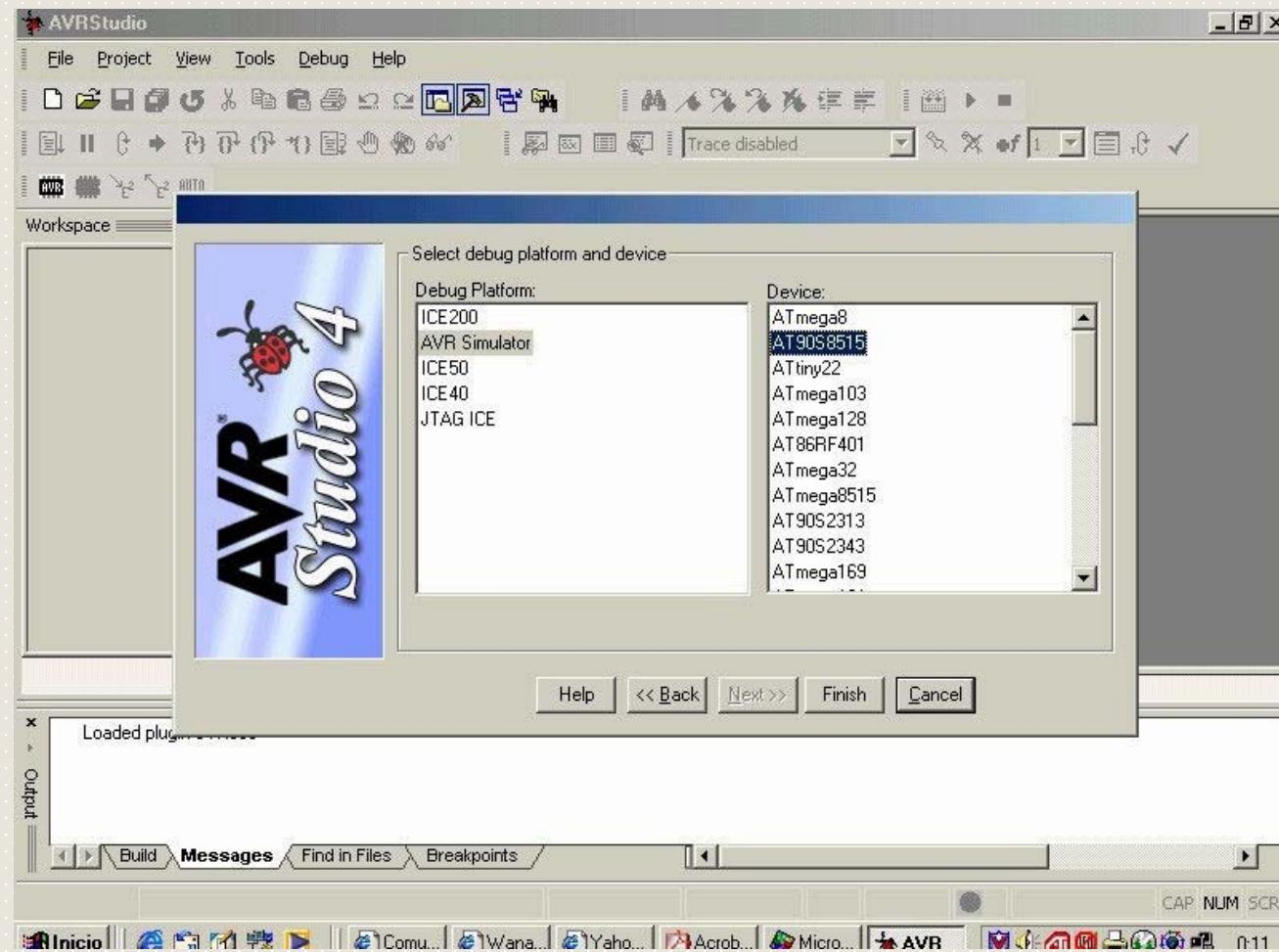
AVR Studio 4



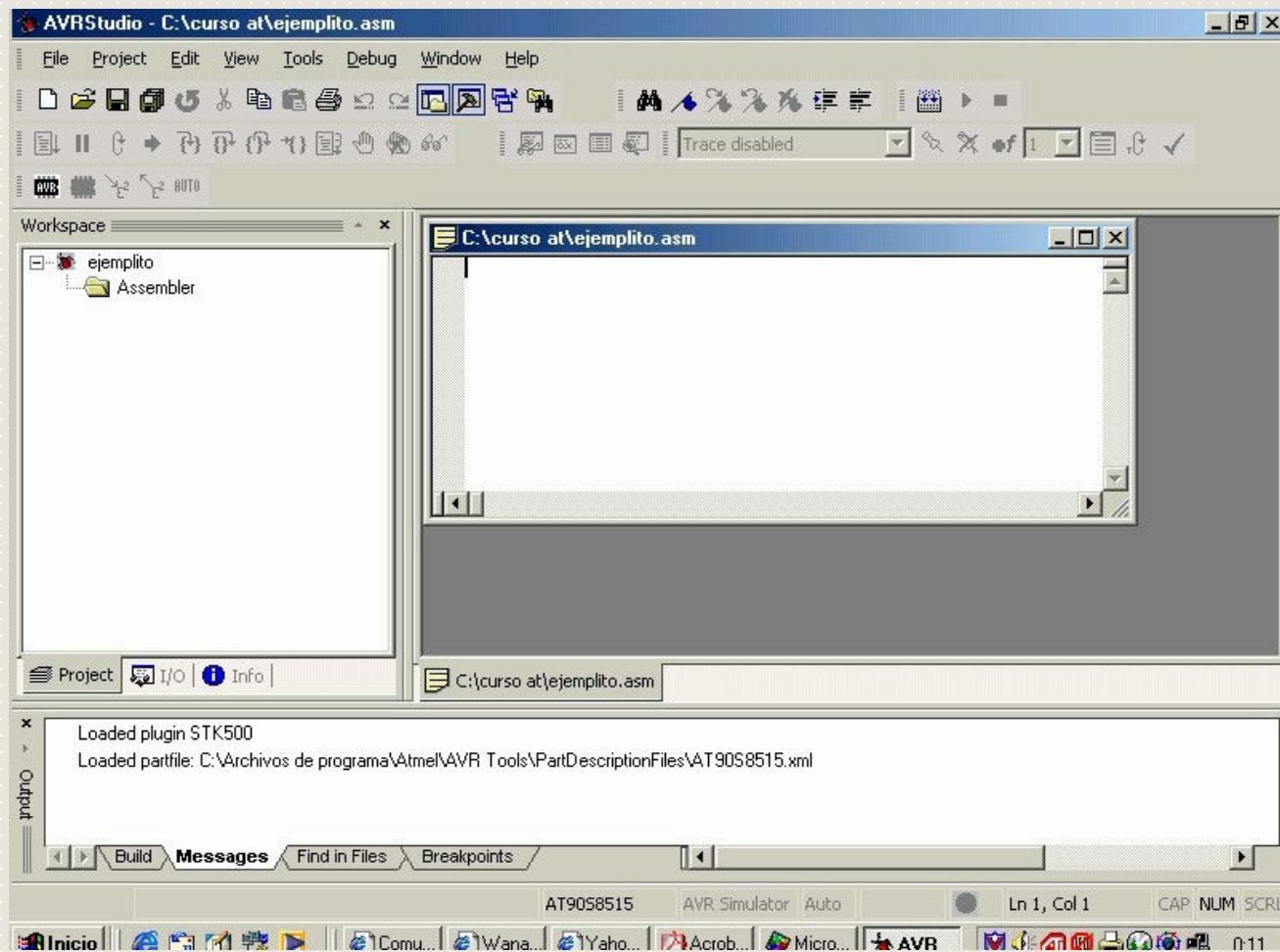
AVR Studio 4



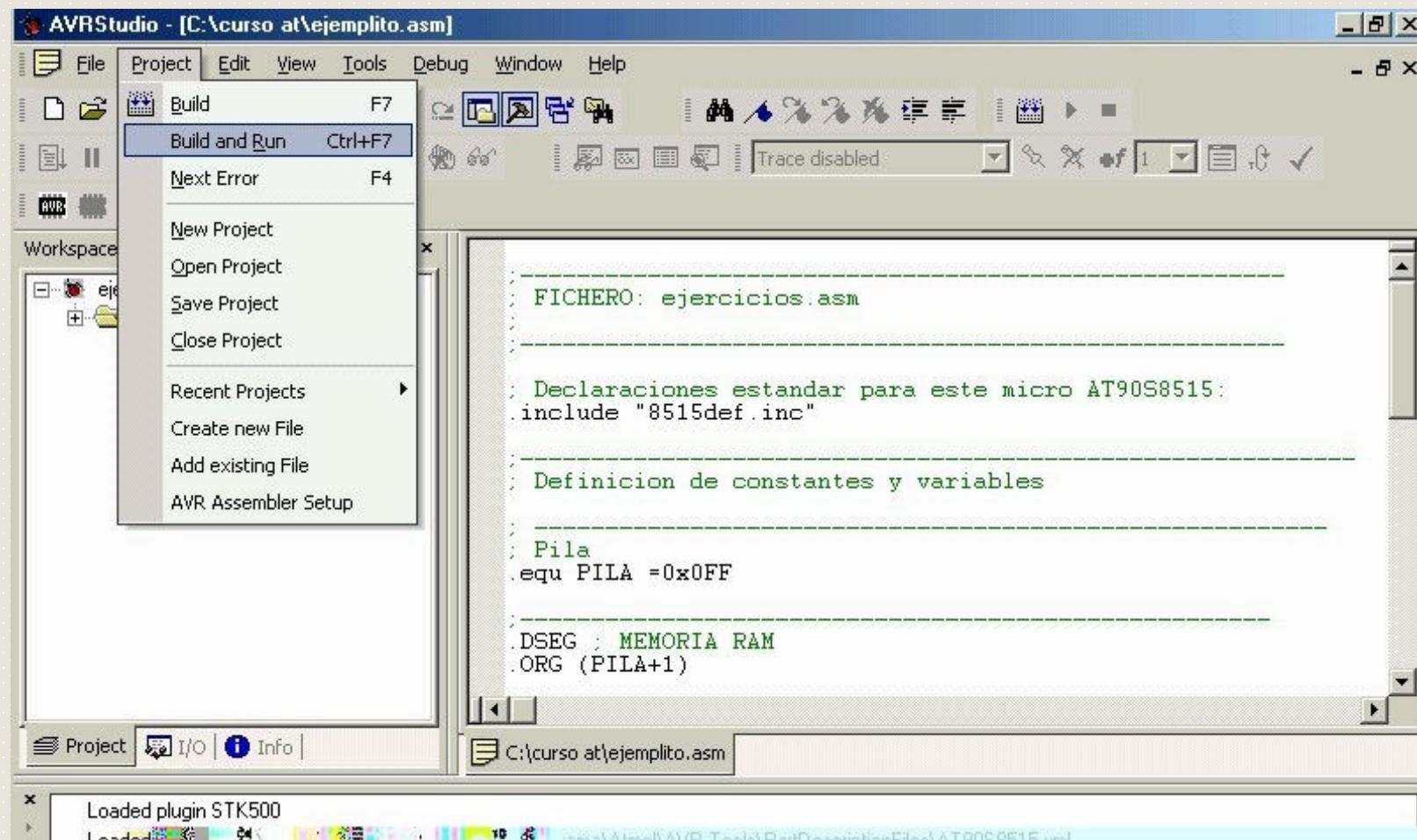
AVR Studio 4



AVR Studio 4

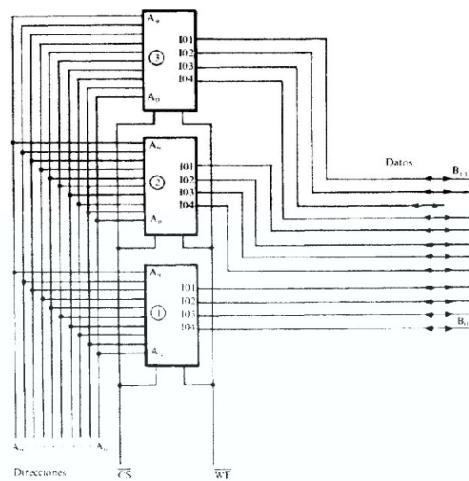


AVR Studio 4



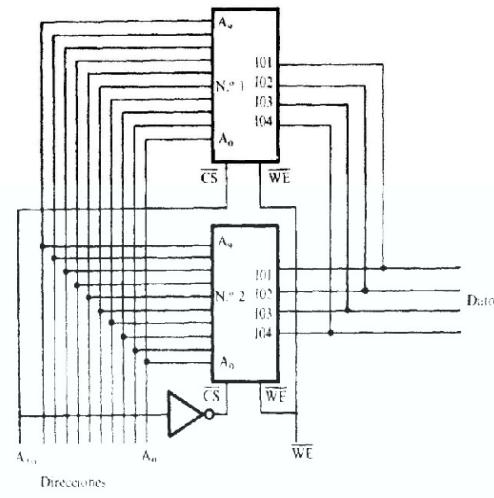
EXPANSIÓN DE MEMORIAS (1)

- **Expansión del tamaño de palabra**
 - Memoria de 1Kx12 con memorias de 1Kx4



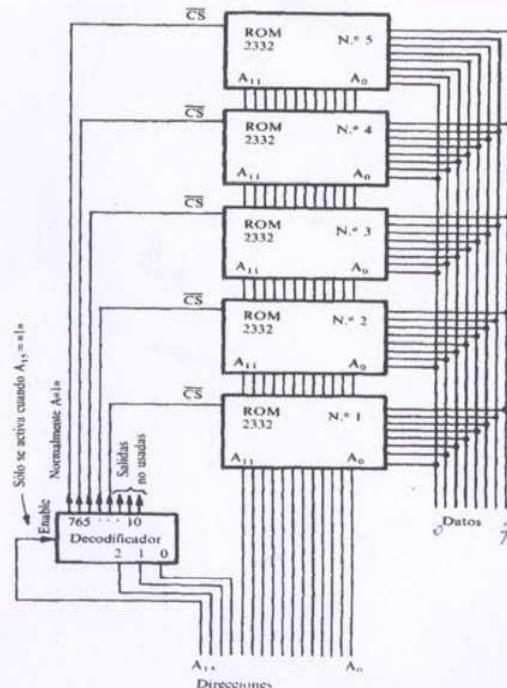
EXPANSIÓN DE MEMORIAS (2)

- **Expansión del número de palabras**
 - Memoria de 2Kx4 con memorias de 1Kx4 usando puertas simples



EXPANSIÓN DE MEMORIAS (3)

- **Expansión del número de palabras (continuación)**
 - Memoria de 20Kx8 con memorias de 4Kx8 usando un decodificador



EXPANSIÓN DE MEMORIAS (4)

- **Expansión del tamaño de palabra y del número de palabras**
 - Memoria de 4Kx8 con memorias de 1Kx4

