

Principios de Mecatrónica – SDI-11561

Ingeniería en Mecatrónica

Hugo Rodríguez Cortés

Departamento de Ingeniería Eléctrica y Electrónica

Instituto Tecnológico Autónomo de México

Agosto 2023

- MUL Rd, Rr Multiplicación de números sin signo, el resultado se almacena en R0 byte BAJO y R1 byte ALTO.
- MULS Rd, Rr Multiplicación de números con signo, el resultado se almacena en R0 byte BAJO y R1 byte ALTO.
- MULSU Rd, Rr Multiplicación de números con signo y números sin signo, el resultado se almacena en R0 byte BAJO y R1 byte ALTO.
- Ejemplo

```
LDI    R23, 0x25 ; —
LDI    R24, 0x65 ; —
MUL    R23, R25  ; 0x25 × 0x65 = 0xE99—
                ; R1 = 0x0E, R0 = 0x99—
```

- En 0x310 se encuentra el dato 0xFD, convertirlo a decimal guardando en 0x322, 0x323 y 0x324 los dígitos; en 0x322 el menos significativo.

```

.EQU    HEX_NUM=0x315      ;           —
.EQU    RMND_L=0x322      ;           —
.EQU    RMND_M=0x323      ;           —
.EQU    RMND_H=0x324      ;           —
.DEF    NUM=R20            ;           —
.DEF    DENOMINATOR = R21  ;           —
.DEF    QUOTIENT = R22     ;           —

        LDI    R16, 0xFD   ;           —
        STS    HEX_NUM, R16 ;           —
        LDS    NUM, HEX_NUM ;           —
        LDI    DENOMINATOR, 10 ;       —
        LDI    QUOTIENT, 0 ;           —

L1:     INC    QUOTIENT     ;           —
        SUB    NUM, DENOMINATOR ;       —
        BRCC   L1          ;   salta si C=0—
        DEC    QUOTIENT     ;           —
        ADD    NUM, DENOMINATOR ;       —
        STS    RMND_L, NUM  ;           —
        MOV    NUM, QUOTIENT ;           —
        LDI    QUOTIENT, 0 ;           —

L2:     INC    QUOTIENT     ;           —
        SUB    NUM, DENOMINATOR ;       —
        BRCC   L2          ;           —
        DEC    QUOTIENT     ;           —
        ADD    NUM, DENOMINATOR ;       —
        STS    RMND_M, NUM  ;           —

        STS    RMND_H, QUOTIENT ;       —
    
```

- Para representar el signo de un número con signo se utiliza el bit más significativo

D7	D6	D5	D4	D3	D2	D2	D0
±	-	v	a	l	o	r	-

El rango de números positivos es de 0 a 127, si se requieren valores más grandes se deben utilizar operandos de 16 bits.

- En un número negativo $D7 = 1$ y el valor se representa con el complemento a 2.
- Ejemplo. Representar al número -5

0	0	0	0	0	1	0	1	5 en binario
1	1	1	1	1	0	1	0	complemento a 1
1	1	1	1	1	0	1	1	complemento a 2

The diagram illustrates a set of 16 data points, each represented by an arrow pointing from a central origin. The arrows are color-coded: black for positive values and blue for negative values. Each arrow is labeled with its value in decimal (D:) and binary (BIN) format, followed by a label for its hexadecimal representation (HEX:). The values range from D:120 down to D:-128.

Value (D:)	Binary (BIN)	Hexadecimal (HEX)	Color
120	01111000	78	Black
115	01110111	77	Black
110	01101110	76	Black
105	01100111	75	Black
100	01011100	74	Black
95	01010111	73	Black
90	01001110	72	Black
85	01000111	71	Black
80	00111100	70	Black
75	00110111	6F	Black
70	00101110	6E	Black
65	00100111	6D	Black
60	00011100	6C	Black
55	00010111	6B	Black
50	00001110	6A	Black
45	00000111	69	Black
40	00111100	68	Black
35	00110111	67	Black
30	00101110	66	Black
25	00100111	65	Black
20	00011100	64	Black
15	00010111	63	Black
10	00001110	62	Black
5	00000111	61	Black
0	00000000	60	Black
-5	11111100	FF	Blue
-10	11110111	FE	Blue
-15	11101110	FD	Blue
-20	11100111	FB	Blue
-25	11011100	F8	Blue
-30	11010111	F7	Blue
-35	11001110	F6	Blue
-40	11000111	F5	Blue
-45	10111100	F4	Blue
-50	10110111	F3	Blue
-55	10101110	F2	Blue
-60	10100111	F1	Blue
-65	10011100	F0	Blue
-70	10010111	EF	Blue
-75	10001110	EE	Blue
-80	10000111	ED	Blue
-85	09111100	EC	Blue
-90	09110111	EB	Blue
-95	09101110	EA	Blue
-100	09100111	E9	Blue
-105	09011100	E8	Blue
-110	09010111	E7	Blue
-115	09001110	E6	Blue
-120	09000111	E5	Blue
-125	08111100	E4	Blue
-128	08110111	E3	Blue

- En operaciones con número con signo puede ocurrir un desbordamiento, el CPU indicará esta situación con la bandera V.
- Ejemplo.

```
LDI    R20, 0x60    ;    —  
LDI    R21, 0x46    ;    —  
ADD    R20, R21     ;    —
```

- En operaciones con número con signo puede ocurrir un desbordamiento, el CPU indicará esta situación con la bandera V.
- Ejemplo.

```
LDI    R20, 0x60 ;      —
LDI    R21, 0x46 ;      —
ADD    R20, R21   ;      —
```

$$0x60 + 0x46 = 0110\ 0000 + 0100\ 0110 = 1010\ 0110$$

El CPU indicará la situación con $V=1$. Notar que $N=1$ (resultado negativo).

- El rango de operación para números con signo son los números enteros $[-128, 127]$.

- En operaciones de 8 bits de números con signo la bandera V se asigna igual a 1 si alguna de las condiciones siguientes ocurre
 - ◆ Se lleva uno de D6 a D7 y no se lleva de D7 (C=0).
 - ◆ Se lleva uno de D7 (C=1) y no se lleva de D6 a D7
- Determine el valor de las banderas N y V en el siguiente código

```
LDI    R20, 0x80    ;      —  
LDI    R21, 0xFE    ;      —  
ADD    R20, R21     ;      —
```


- En operaciones de 8 bits de números con signo la bandera V se asigna igual a 1 si alguna de las condiciones siguientes ocurre
 - ◆ Se lleva uno de D6 a D7 y no se lleva de D7 (C=0).
 - ◆ Se lleva uno de D7 (C=1) y no se lleva de D6 a D7
- Determine el valor de las banderas N y V en el siguiente código

```
LDI    R20, 0x80    ;      —  
LDI    R21, 0xFE    ;      —  
ADD    R20, R21     ;      —
```

$N = 0, V = 1$

Examine el valor de las banderas V y N

- Sumar -2 con -5.

LDI	R20, -2	;	—
LDI	R21, -5	;	—
ADD	R20, R21	;	—

Examine el valor de las banderas V y N

■ Sumar -2 con -5.

```
LDI    R20, -2    ;      —
LDI    R21, -5    ;      —
ADD    R20, R21   ;      —
```

1	1	1	1	1	1	1	0	-2	
1	1	1	1	1	0	1	1	-5	, V = 0, N = 1
1	1	1	1	1	0	0	1	-7	

- Sumar 7 con 18.

```
LDI    R20, 7    ;    —  
LDI    R21, 18   ;    —  
ADD    R20, R21  ;    —
```

- Sumar 7 con 18.

```
LDI    R20, 7      ; —
LDI    R21, 18     ; —
ADD    R20, R21    ; —
```

0	0	0	0	0	1	1	1	7
0	0	0	1	0	0	1	0	18
0	0	0	1	1	0	0	1	25

, V = 0, N = 0

- En operaciones sin signo C determina el estado de la operación, mientras que en operaciones con signo lo hace V.

- AND Rd, Rr Aplica el operador lógico AND bit a bit entre los registros Rd y Rr. ANDI Rd, k con k un valor constante.

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Esta operación puede activar las banderas Z, S y N.

LDI	R20, 0x35	;	—
ANDI	R20, 0x0F	;	—

- AND Rd, Rr Aplica el operador lógico AND bit a bit entre los registros Rd y Rr. ANDI Rd, k con k un valor constante.

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Esta operación puede activar las banderas Z, S y N.

```
LDI    R20, 0x35 ; —
ANDI   R20, 0x0F ; —
```

```
0 0 1 1 0 1 0 1 0x35
0 0 0 0 1 1 1 1 0x0F , Z = 0, N = 0
0 0 0 0 0 1 0 1 0x05
```

- OR Rd, Rr Aplica el operador lógico OR bit a bit entre los registros Rd y Rr. ORI Rd, k con k un valor constante.

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

Esta operación puede activar las banderas Z, S y N.

```
LDI  R20, 0x04 ; —
ORI  R20, 0x30 ; —
```


- OR Rd, Rr Aplica el operador lógico OR bit a bit entre los registros Rd y Rr. ORI Rd, k con k un valor constante.

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

Esta operación puede activar las banderas Z, S y N.

```
LDI  R20, 0x04 ; —
ORI  R20, 0x30 ; —
```

```
0 0 0 0 0 1 0 0 0x04
0 0 1 1 0 0 0 0 0x30 , Z = 0, N = 0
0 0 1 1 0 1 0 0 0x34
```

- EXOR Rd, Rr Aplica el operador lógico XOR bit a bit entre los registros Rd y Rr.

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Esta operación puede activar las banderas Z, S y N.

LDI	R20, 0x54	;	—
LDI	R20, 0x78	;	—
EOR	R20, R21	;	—

- EXOR Rd, Rr Aplica el operador lógico XOR bit a bit entre los registros Rd y Rr.

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Esta operación puede activar las banderas Z, S y N.

```
LDI    R20, 0x54 ; —
LDI    R20, 0x78 ; —
EOR    R20, R21  ; —
```

```
0 1 0 1 0 1 0 0 0x54
0 1 1 1 1 0 0 0 0x78 , Z = 0, N = 0
0 0 1 0 1 1 0 0 0x2C
```

- Analice el siguiente código

	LDI	R20, 0xFF	;	—
	OUT	DDRC, R20	;	—
	LDI	R20, 0x00	;	—
	OUT	DDRB, R20	;	—
	OUT	PORTC, R20	;	—
	LDI	R21, 0x45	;	—
HERE:				
	IN	R20, PINB	;	—
	EOR	R20, R21	;	—
	BRNE	HERE	;	—
	LDI	R20, 0x99	;	—
	OUT	PORTC, R20	;	—

- COM Rd Esta instrucción complementa bit a bit el contenido de un registro. Se conoce también como el complemento a uno.
- NEG Rd Esta instrucción calcula el complemento a dos de contenido del registro.
- CP Rd, Rr Esta instrucción es una resta entre Rd y Rr excepto que el valor de Rd no cambia.
- CPI Rd, k comparación con una constante. CP y CPI se unen a operadores de saltos condicionales.

- BCD (binary coded decimal) es la representación binaria de los números del 0 al 9. Se asocia a dos conceptos BCD sin comprimir y BCD comprimido.
- En un BCD sin comprimir el nibble bajo representa al BCD y el nibble alto se llena con ceros. Por ejemplo, $9 = 00001001$, $5 = 00000101$. Se utiliza un byte de memoria para almacenarlo.
- En un BCD comprimido. Los nibbles alto y bajo contienen un número. Por ejemplo, $0x59 = 01011001$. Se requiere la misma cantidad de memoria para almacenarlo, un byte
- En teclados ASCII cuando se activa la tecla '0' se envía $0110000 = 0x30$ al CPU. De forma similar '2' $= 0110010 = 0x32$.

- Para convertir de BCD comprimido a ASCII. Primero el BCD se descomprime y luego se combina con 0x30.

BCD comprimido	BCD sin comprimir		ASCII	
0x29	0x02	0x09	0x32	0x39
0010 1001	0000 0010	0000 1001	0011 0010	0011 1001

- Código

```

LDI    R20, 0x29 ;
MOV    R21, R20  ;
ANDI   R21, 0x0F ;
ORI    R21, 0x30 ;

MOV    R22, R20  ;
SWAP   R22       ; intercambia nibbles—
ANDI   R22, 0x0F ;
ORI    R22, 0x30 ;
    
```

- Para convertir de ASCII a BCD comprimido. Primero el BCD se descomprime y luego se combina para comprimirlo.

Tecla	ASCII	BCD sin comprimir	BCD comprimido
'4'	0x34	00000100	
'7'	0x37	00000111	01000111=0x47

- Código

```

LDI    R21, '4'    ;    —
LDI    R22, '7'    ;    —
ANDI   R21, 0x0F   ;    —
SWAP   R21         ;    —

ANDI   R22, 0x0F   ;    —
OR     R22, R21    ;    —
MOV    R20, R22    ;    —

```


- El CPU acepta operaciones aritméticas del tipo

```
.EQU  ALFA = 50    ;      —  
.EQU  BETA = 40    ;      —  
  
LDI   R23, ALFA    ;      —  
STS   R24, ((ALFA-BETA)*2)+9 ; —
```

- Lo mismo que operaciones lógicas como

```
.EQU  C1 = 0x50    ;      —  
.EQU  C2 = 0x10    ;      —  
.EQU  C3 = 0x04    ;      —  
  
LDI   R21, (C1&C2)|C3 ; —
```

■ Operaciones aritméticas

Símbolo	Acción
+	Adición
—	Resta
*	Multiplicación
/	División
%	Módulo

■ Operaciones lógicas

Símbolo	Acción
&	AND bit a bit
	OR bit a bit
^	XOR bit a bit
~	NOT bit a bite

■ Operadores de corrimiento

Símbolo	Acción	Ejemplo
$X \ll Y$	Recorre hacia la izquierda a X Y número de lugares y viceversa	LDI R20, 0b101<<2 ;R20 = 0b10100
$X \gg Y$	Recorre hacia la derecha a X Y número de lugares y viceversa	LDI R20, 0b100>>1 ;R20 = 0b010

■ Ejemplo

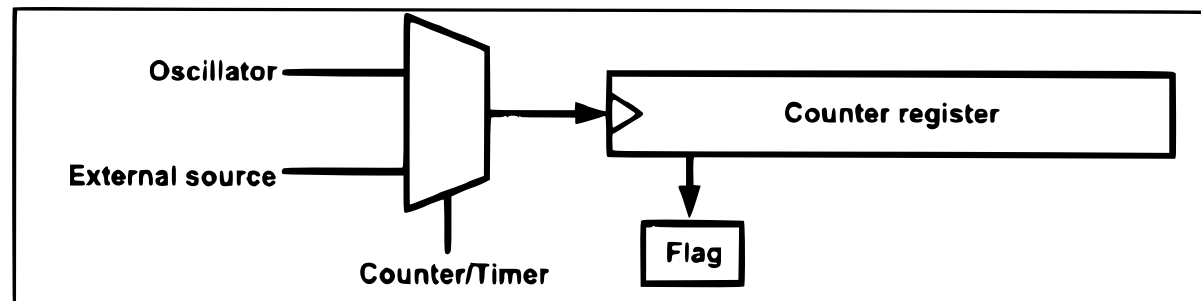
```
LDI R21, 0b00000111<<1 ; R21 = 0b00001110—
```

■ HIGH() y LOW() Obtienen el byte alto y bajo, respectivamente de un dato de 16 bits

```
LDI R21, LOW(0x4455) ; R21 = 0x55—
LDI R22, HIGH(0x4455) ; R22 = 0x44—
```

- Muchas aplicaciones necesitan contar eventos y/o generar tiempos de retardo. El microcontrolador tiene **registros de contadores** para estos propósitos.
- Para contar un evento se conecta la fuente del evento externo al pin del registro del contador. De esta forma, cuando el evento ocurre el contador se incrementa, así, el contenido del contador representa el número de veces que el evento ocurre.
- Para generar tiempo de retardo, se conecta el oscilador al pin del contador. Cada oscilación incrementa el contenido del contador. Como resultado, el contenido del registro del contador representará cuantas oscilaciones han ocurrido desde que limpió el contador.

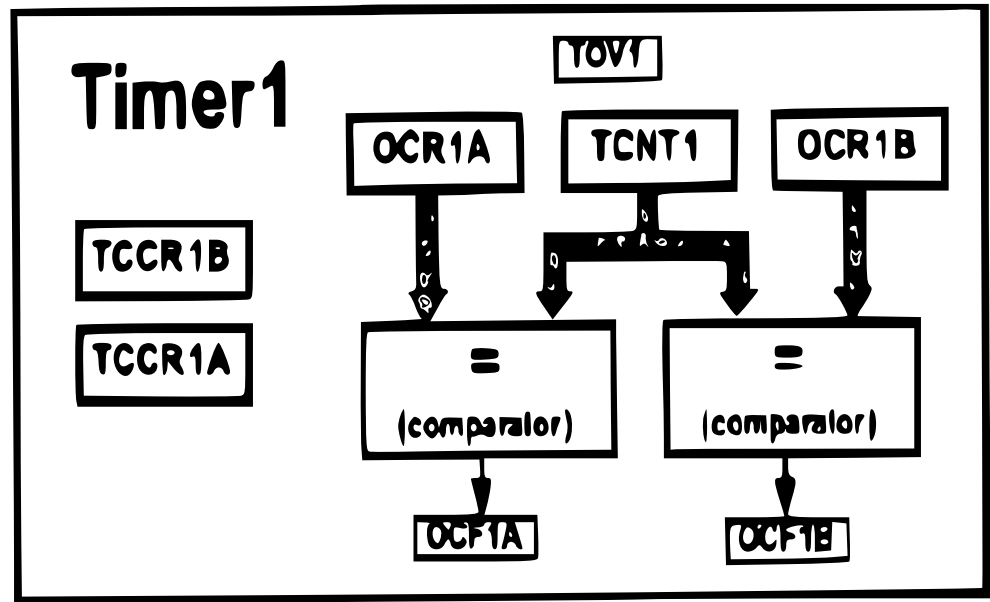
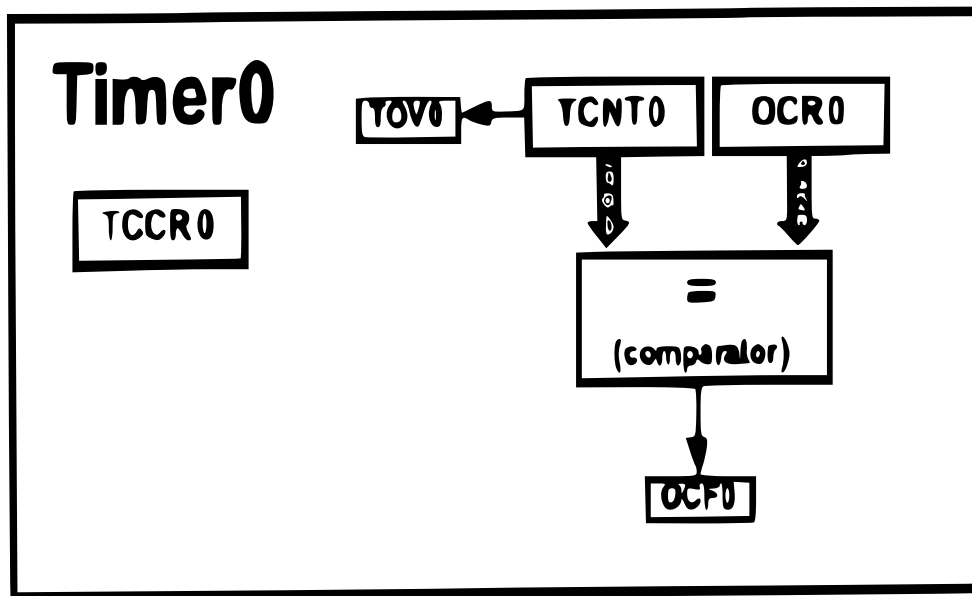
- Ya que la velocidad del oscilador, en un microcontrolador, se conoce, se puede calcular el periodo de la oscilación y del contenido del registro del contador se obtiene el tiempo transcurrido.
- De esta manera para generar un retardo, se borra el contador en el tiempo inicial y se espera hasta que el contador alcance un cierto número.



- Ejemplo, considere un microcontrolador con un oscilador de 1MHz; en el microcontrolador, el contenido del registro del contador incrementa una vez por cada microsegundo. Entonces, si se quiere un retardo de 100 microsegundos, debemos limpiar el contador y esperar hasta que sea igual a 100.
- En el microcontrolador, hay una bandera para cada uno de los contadores.
 - ◆ La bandera se activa cuando el contador se desborda, por software se borra.
 - ◆ El segundo método para generar un retardo es cargar el registro del contador y esperar hasta que el contador se desborde y la bandera se active.

- Ejemplo. En un microcontrolador, con oscilador de 1MHz y registro de contador de 8 bits. Si se requiere un retardo de 3 microsegundos, el registro del contador se carga con 0xFD, la bandera se activará después de tres ciclos. El contenido del registro se incrementará a 0xFE, después a 0xFF y en el tercer ciclo el contenido del registro se desborda a 0x00 y la bandera se activa.
- El AVR tiene de 1 a 6 timers dependiendo del tipo de familia. Ellos son referidos como 0, 1..5. Pueden ser usados como timers para generar retardo o como contadores para contar eventos sucediendo fuera del microcontrolador.
- El ATmega32, tiene 3 timers: Timer0, Timer1 y Timer2. Timer0 y Timer2 son de 8 bits, mientras que el timer T1 es de 16 bits.

- Cada Timer necesita un pulso de reloj para contar. La fuente del pulso puede ser interna o externa.
- Si se usa una fuente interna, entonces la frecuencia del oscilador alimenta al contador. Por lo tanto, al utilizarse para generar retardos se conoce como Timer.
- Si se elige la opción de un pulso externo, es necesario alimentarlo a través de uno de los pines del AVR. En este caso se tiene un contador.



TCNTn (timer/counter) registro del Timer/contador. TOVn (Timer Overflow) bandera de desbordamiento. TCCRn (timer/counter control register) registro para definir modos de operación. OCRn (Output Compare Register). El contenido de OCRn se compara con TCNTn, si son iguales la bandera OCFn (Output Compare Flag) se activa.

- TCNT0 El registro Timer/Contador tiene 8 bits

TCNT0 D7 D6 D5 D4 D3 D2 D1 D0

- TCCR0 Control/Modo de Operación del registro TCNT0

TCCR0	D7	D6	D5	D4	D3	D2	D1	D0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
	W	RW	RW	RW	RW	RW	RW	RW
CI	0	0	0	0	0	0	0	0

FOC0: Se usa para generar ondas. WGM00-WGM01: Modo de Operación 0-0 normal, 0-1 Limpiar el Timer al comparar, 1-0 PWM fase correcta, 1-1 PWM rápido. COM00-COM01: Se utilizan en la generación de ondas. CS02-CS01-CS00: Configuración del reloj, 0-0-0 Timer/Contador detenido, 0-0-1 Reloj clk sin escalador, 0-1-0 clk/8, 0-1-1 clk/64, 1-0-0 clk/256, 1-0-1 clk/1024. 1-1-0 reloj en pin T0 flanco de bajada, 1-1-1 reloj en pin T0 flanco de subida.

- TIFR0 Registro de la bandera de interrupción del registro del Timer/Contador.

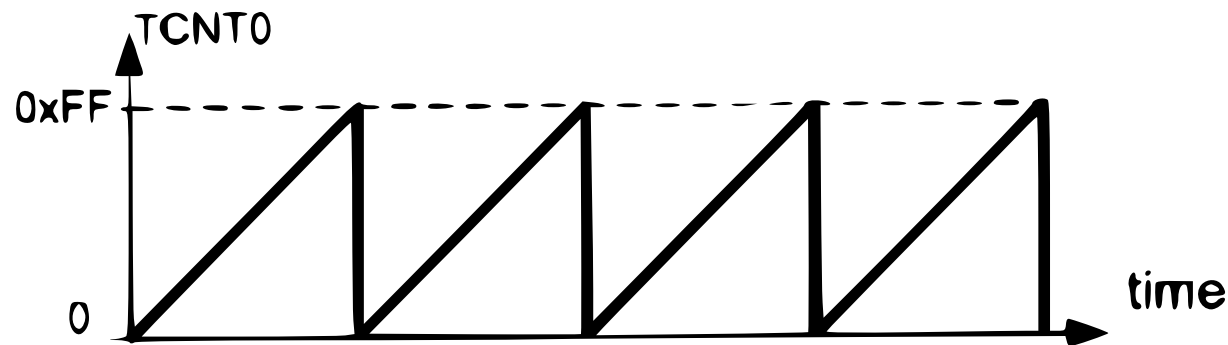
TIFR0	D7	D6	D5	D4	D3	D2	D1	D0
	OFC2	TOV2	ICF1	OFC1A	OFC1B	TOV1	OFC0	TOV0
	RW	RW	RW	RW	RW	RW	RW	RW
CI	0	0	0	0	0	0	0	0

TOV0: 0-1 no desbordamiento/desbordamiento Timer0, 0xFF→0x00. OFC0: 0-1 comparación ocurrió/no ocurrió. TOV1: 0-1 no desbordamiento/desbordamiento Timer1. OFC1B: Bandera de comparación de salida B del Timer1. OFC1A: Bandera de comparación de salida A del Timer1. ICF1: Bandera de captura de entrada. TOV2: Bandera de desbordamiento Timer2. OFC2: Bandera de comparación de salida.

- La bandera TOV0 es igual a uno cuando TCNT0 se desborda, decir va de 0xFF a 0x00. Para borrar la bandera se escribe uno en TOV.

```
LDI    R21, 0x01    ;
OUT    TIFR, R21    ;
```

- En este modo, el contenido del timer/contador aumenta cada ciclo de reloj. El contador se incrementa hasta alcanzar su máximo 0xFF, cuando él pasa de 0xFF a 0x00, la bandera TVO0 se activa. Esta bandera se puede leerse.



	LDI	R16, 1<<5	;	—
	SBI	DDRB, 5	;	—
	LDI	R17, 0x00	;	—
	OUT	PORTB, R17	;	—
BEGIN:	RCALL	DELAY	;	—
	EOR	R16, R17	;	—
	OUT	PORTB, R17	;	—
	RJMP	BEGIN	;	—
DELAY:	LDI	R20, 0xF2	;	—
	OUT	TCNT0, R20	;	—
	LDI	R20, 0x01	;	—
	OUT	TCCR0, R20	;	—
AGAIN:	IN	R20, TIFR	;	—
	SBRS	R20, TOV0	;	SBRS Verifica el estatus de un bit, salta si TVO0=1—
	RJMP	AGAIN	;	—
	LDI	R20, 0x00	;	—
	OUT	TCCR0, R20	;	—
	LDI	R20, (1<<TOV0)	;	—
	OUT	TIFR, R20	;	—
	RET		;	—

- ¿Cuál es el valor del retardo generado por el timer asumiendo que el cristal(XTAL) es de 8 MHZ?

- ¿Cuál es el valor del retardo generado por el timer asumiendo que el cristal(XTAL) es de 8 MHZ?
- ◆ La frecuencia del reloj que alimenta al Timer es de 8MHz, entonces cada ciclo de reloj tiene un periodo de $T = 1/(8MHz) = 0.125$ micro segundos. Esto significa que el Timer0 cuenta hacia arriba cada 0.125 microsegundos, esto es,
$$\text{Delay} = \text{número de cuentas} \times (0.125 \text{ microsegundos}).$$
- ◆ El contador inicia en 0xF2 y la bandera TOV0 se activa al llegar a 0x00, entonces, sumaríamos un ciclo extra ($0xFF - 0xF2 = 13 + 1$). Así, tendremos que el retardo es de 1.75 microsegundos.

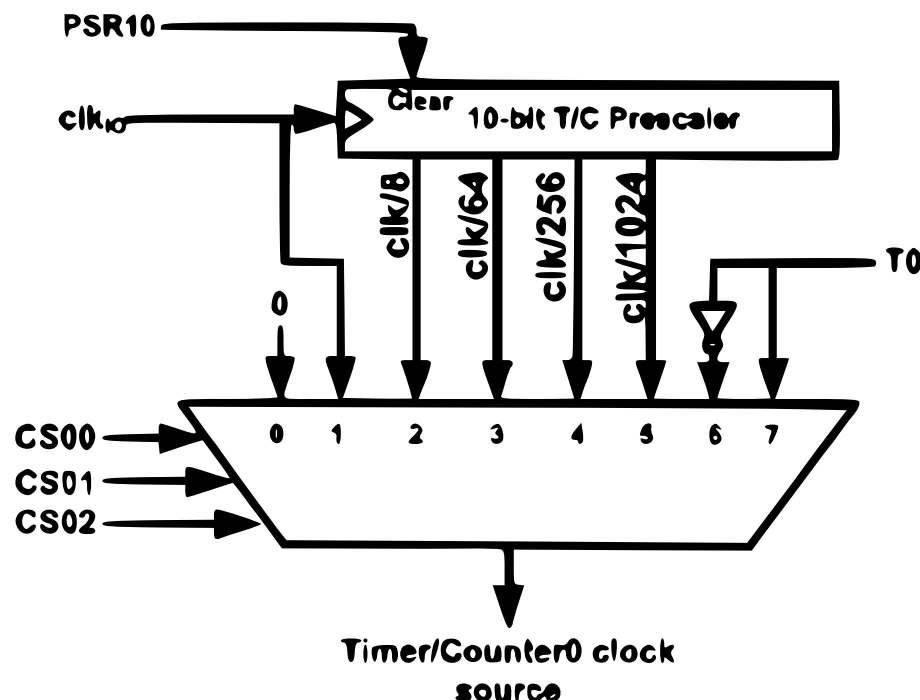
Ejemplo. Suponga que se tiene un cristal de 8MHz. Modificar el programa anterior para generar una onda cuadrada con periodo de $12.5 \mu s$ en el pin 3 del PORTB.

Ejemplo. Suponga que se tiene un cristal de 8MHz. Modificar el programa anterior para generar una onda cuadrada con periodo de $12.5 \mu s$ en el pin 3 del PORTB.

Para generar la onda cuadrada con $T = 12.5 \mu s$ se requiere un retardo de $6.25 \mu s$. Con el cristal de 8MHz se tiene un contador de $0.125 \mu s$. Por lo tanto, se requieren $6.25/0.125$ ciclos. Las modificaciones al programa anterior son las siguientes

	LDI	R16, 0x08	;	—
	SBI	DDRB, 3	;	—
	LDI	R17, 0x00	;	—
	OUT	PORTB, R17	;	—
DELAY:	LDI	R20, 0xCE	;	—

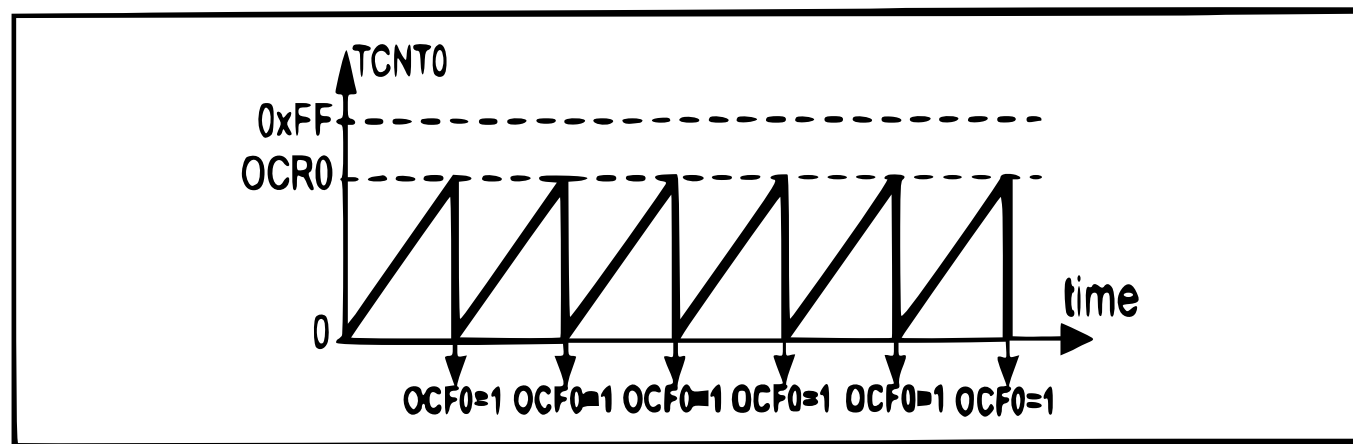
- El control del retardo recae en el registro de 8 bits TNCT0 y la frecuencia del Cristal XTAL.
- El AVR tiene la posibilidad de escalar la frecuencia del cristal por un factor de 8, 64, 256 y 1024.



Calcule el retardo generado por el siguiente programa con XTAL=8MHz

	LDI	R16, 0x08	;	—
	SBI	DDRB, 3	;	—
	LDI	R17, 0x00	;	—
	OUT	PORTB, R17	;	—
BEGIN:	RCALL	DELAY	;	—
	EOR	R16, R17	;	—
	OUT	PORTB, R17	;	—
	RJMP	BEGIN	;	—
DELAY:	LDI	R20, 0x00	;	—
	OUT	TCNT0, R20	;	—
	LDI	R20, 0x05	;	—
	OUT	TCCR0, R20	;	—
AGAIN:	IN	R20, TIFR	;	—
	SBRS	R20, TOV0	;	SBRS Verifica el estatus de un bit, salta si TVO0=1—
	RJMP	AGAIN	;	—
	LDI	R20, 0x00	;	—
	OUT	TCCR0, R20	;	—
	LDI	R20, (1<<TOV0)	;	—
	OUT	TIFR, R20	;	—
	RET		;	—

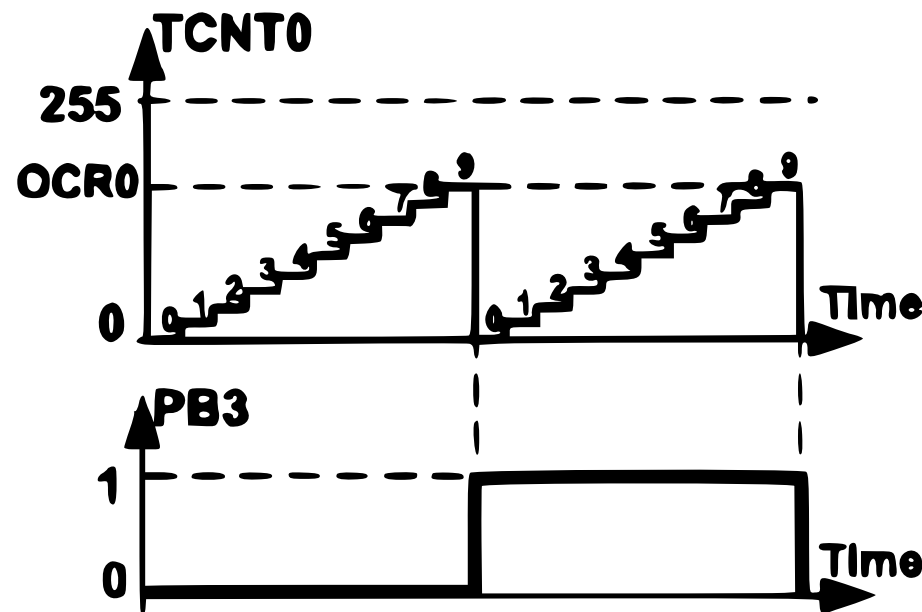
- El registro OCR0 se utiliza con el modo CTC. Al igual que en el modo normal, en el modo CTC, el timer se incrementa con el reloj, pero ahora cuenta hasta que el contenido del registro TCNT0 sea igual al contenido de OCR0.
- En esta punto la bandera OCF0 se activa en el siguiente ciclo de reloj. La bandera OCF0 esta localiza en el registro TIRF.



Determine la salida generada por el siguiente programa

	LDI	R16, 0x08	;	—
	SBI	DDRB, 3	;	—
	LDI	R17, 0x00	;	—
	OUT	PORTB, R17	;	—
BEGIN:	OUT	PORTB, R17	;	—
	RCALL	DELAY, R17	;	—
	EOR	R17, R16	;	—
	RJMP	BEGIN	;	—
DELAY:	LDI	R20, 0x00	;	—
	OUT	TCNT0, R20	;	—
	LDI	R20, 9	;	—
	OUT	OCR0, R20	;	—
	LDI	R20, 0x09	;	—
	OUT	TCCR0, R20	;	—
AGAIN:	IN	R20, TIFR	;	—
	SBRS	R20, OCFR0	;	—
	RJMP	AGAIN	;	—
	LDI	R20, 0x00	;	—
	OUT	TCCR0, R20	;	—
	LDI	R20, 1<<OCF0	;	—
	OUT	TIFR, R20	;	—
	RET		;	—

- OCR0 se carga con 9 y TCNT0 con 0. Después de 9 pulsos de reloj $TCNT0 = OCR0$. En el siguiente ciclo, la bandera OCF0 es igual a 1 y ocurre el reset. Esto significa que TCNT0 se limpia después de 10 ciclos.
- Ya que $XTAL = 8MHz$, el contador cuenta cada $0.125 \mu s$. Por lo tanto, tenemos que $10 \times 0.125 \mu s = 1.25 \mu s$ es el valor del retardo.

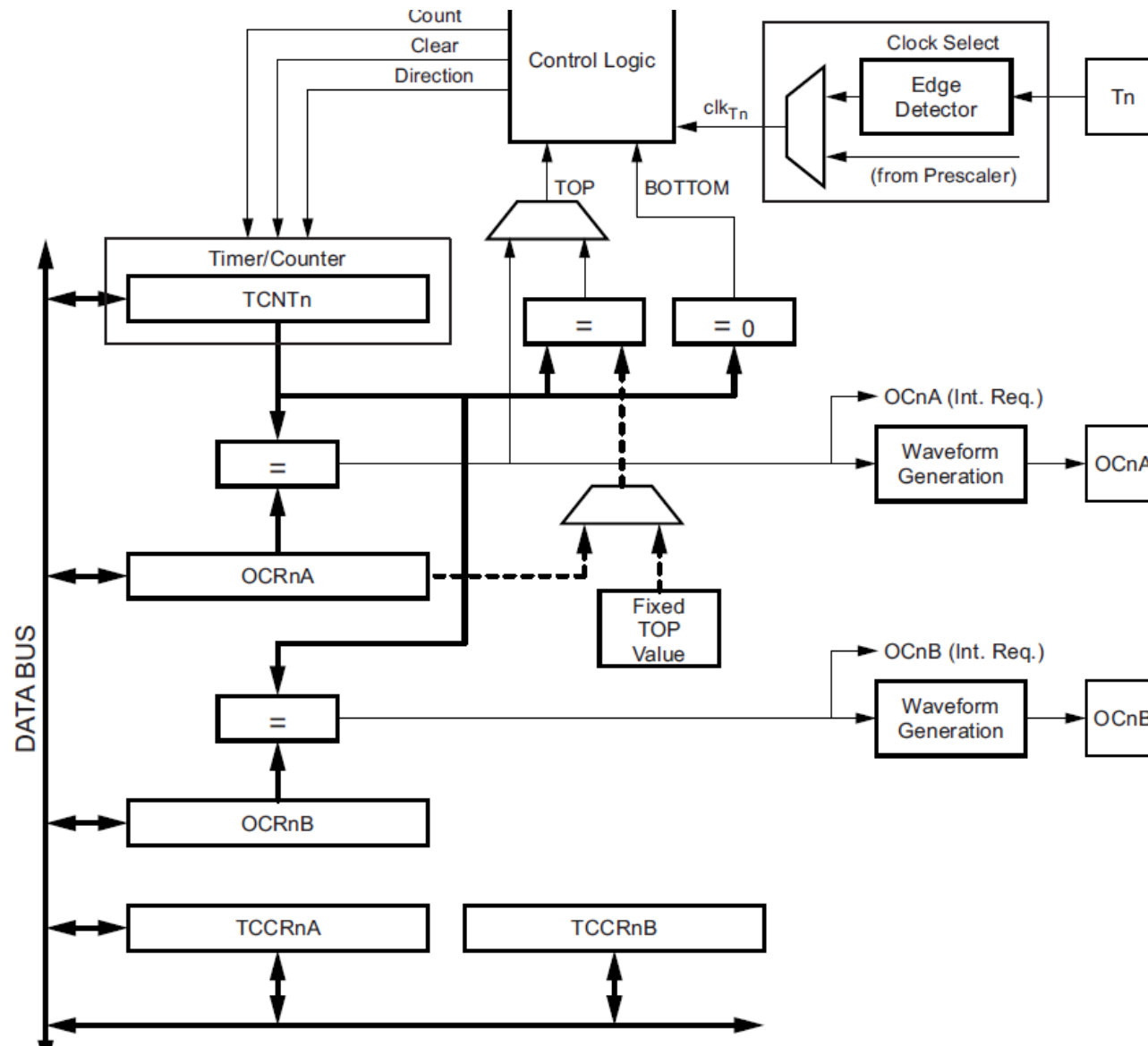


Es siempre recomendable recurrir al manual del microcontrolador, el Timer0 en el AVR ATMega328P es ligeramente diferente, Capítulo 14.

8-bit Timer/Counter0 with PWM

Features

- Two independent output compare units
- Double buffered output compare registers
- Clear timer on compare match (auto reload)
- Glitch free, phase correct pulse width modulator (PWM)
- Variable PWM period
- Frequency generator
- Three independent interrupt sources (TOV0, OCF0A, and OCF0B)



TCNT0 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
0x26 (0x46)	TCNT0[7:0]								TCNT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TIFR0 – Timer/Counter 0 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x15 (0x35)	–	–	–	–	–	OCF0B	OCF0A	TOV0	TIFR0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Aunque los Timer0 y Timer2 son de 8 bits, existen dos diferencias entre ellos.

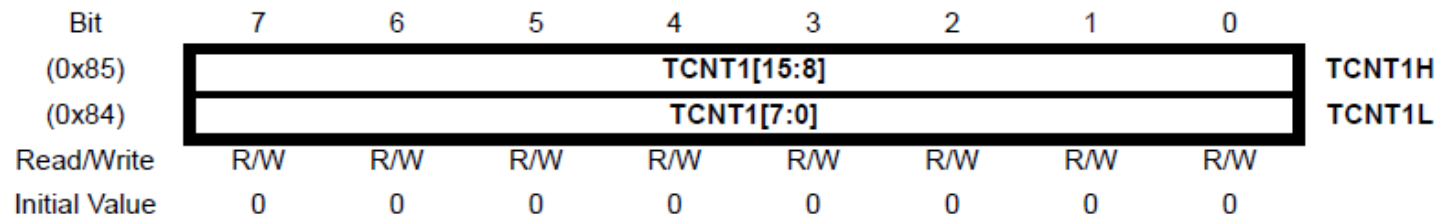
- El Timer2 puede utilizarse como un contador de tiempo real conectando un cristal extra de 32.768KHz en los pines TOSC1 y TOSC2.
- Los bits en el registro TCCR tienen diferente significado.

- El Timer1 es de 16 bits, sus registros se dividen en dos bytes. Para manejar 16 bits, el AVR utiliza una localidad de memoria temporal por lo que deben escribirse o leerse juntos, primero el byte alto.^a

```
LDI    R20, 0x08      ;
LDI    R21, 0x00      ;
STS    TCNT1H, R20    ;
STS    TCNT1L, R21    ;
```

- Registro del Timer1/Contador1 TCNT1L-TCNT1H Bytes bajo y alto.

TCNT1H and TCNT1L – Timer/Counter1



^aEn el ATMega328P los registros TCNT1H-TCNT1L se encuentran en la memoria I/O por lo que debe utilizarse STS para cargar datos.

- TCCR1A-TCCR1B-TCCR1C Registros de control del Timer1/Contador1.

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR1C – Timer/Counter1 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	–	–	–	–	–	–	TCCR1C
Read/Write	R/W	R/W	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

- OCCR1A-OCCR1B. Los registros OCCR están compuestos de dos bytes. Por ejemplo, OCCR1A esta compuesto de OCCR1AH (byte alto) y OCCR1AL(byte bajo).

OCCR1AH and OCCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCCR1A[15:8]								OCCR1AH
(0x88)	OCCR1A[7:0]								OCCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

OCCR1BH and OCCR1BL – Output Compare Register 1 B

Bit	7	6	5	4	3	2	1	0	
(0x8B)	OCCR1B[15:8]								OCCR1BH
(0x8A)	OCCR1B[7:0]								OCCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

■ TIFR1 Registro de banderas de interrupción del Timer1/Contador1

TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Determine la salida generada por el siguiente programa

	LDI	R16, 0x20	;	—
	SBI	DDRB, 5	;	—
	LDI	R17, 0x00	;	—
	OUT	PORTB, R17	;	—
BEGIN:	RCALL	DELAY	;	—
	EOR	R17, R16	;	—
	OUT	PORTB, R17	;	—
	RJMP	BEGIN	;	—
DELAY:	LDI	R20, 0xD8	;	—
	LDI	R21, 0xF0	;	—
	STS	TCNT1H, R20	;	—
	STS	TCNT1L, R21	;	—
	LDI	R20, 0x00	;	—
	LDI	R21, 0x05	;	—
	STS	TCCR1A, R20	;	—
	STS	TCCR1B, R21	;	—
AGAIN:	IN	R20, TIFR1	;	—
	SBRS	R20, TOV1	;	—
	RJMP	AGAIN	;	—
	LDI	R20, 0x00	;	—
	STS	TCCR1A, R20	;	—
	STS	TCCR1B, R20	;	—
	LDI	R20, 0x01	;	—
	OUT	TIFR1, R20	;	—
	RET		;	—

Encontrando valores para ser cargados en el timer

Asumiendo que se conoce el valor del retardo que necesitamos; la pregunta sería: ¿Cómo encontrar los valores necesarios para el registro TCNT0/1? Para calcular los valores a ser cargados en el registro TCNT0/1; podemos seguir los siguientes pasos:

- Calcular el período del reloj usando la formula:

$$T_{clock} = K / (F_{timer})$$

dónde F_{timer} es la frecuencia del reloj usado y K es el factor de escalamiento. Por ejemplo, en modo no escalado $K = 1$, $F_{timer} = F_{frec. oscilador}$. T_{clock} es el periodo en el cual el timer se incrementa.

- Dividir el retardo deseado por T_{clock} . Esto determinará cuantos pulsos de reloj son necesarios.
- Realizar $256/65536 - n$, donde n es el número decimal que se obtuvo del paso anterior.
- Convertir el número decimal del paso anterior a formato hexadecimal, siendo este valor el que se cargará en el registro del timer0/1.

- Los Timers del AVR pueden además utilizarse para contar, detectar y medir el tiempo de eventos pasando fuera del AVR.
- Cuando el Timer se usa para generar retardos, el cristal del AVR es la fuente de frecuencia.
- Cuando se usa como contador, es un pulso externo el que incrementa el registro TCNT_x.
- Recordar que los bits CS02-CS01-CS00 del registro TCCR0B/TCCR1B decide la fuente del reloj para el timer. Si CS02-CS01-CS00 = 001-101 el Timer0/1 obtiene pulsos del cristal oscilador, posiblemente con escalamiento. En contraste, cuando CS02-CS01-CS00 = 110-111 el Timer0/1 se usa como un contador y obtiene sus pulsos de una fuente externa al AVR.

- Por lo tanto, cuando CS02-CS01-CS00 = 110-111 el registro del contador TCNT0/TCNT1 cuenta hacia arriba (pulso subida o de bajada) cada que llegan pulsos al pin T0/T1.
- Ejemplo

```

CBI    DDRB, 0          ;          —
LDI    R20, 0xFF        ;          —
OUT    DDRC, R20        ;          —
LDI    R20, 0X06        ;          —
OUT    TCCR0, R20       ;          —

AGAIN: IN    R20, TCNT0  ;          —
OUT    PORTC, R20       ;          —
IN     R16, TIFR0       ;          —
SBRS   R16, TOV0        ;          —
RJMP   AGAIN           ;          —
LDI    R16, 1<<TOV0     ;          —
OUT    TIFR0, R16       ;          —
RJMP   AGAIN           ;          —

```

- Ejemplo, contar hasta un número de 16 bits.

```

LDI    R19, 0          ;
CBI    DDRB, 0         ;
LDI    R20, 0xFF       ;
OUT    DDRC, R20       ;
OUT    DDRD, R20       ;
LDI    R20, 0x06       ;
OUT    TCCR0, R20      ;

AGAIN: IN    R20, TCNT0 ;
OUT    PORTC, R20      ;
IN     R16, TIFR0      ;
SBRS   R16, TOV0       ;
RJMP   AGAIN          ;
LDI    R16, 1<<TOV0    ;
OUT    TIFR0, R16      ;
INC    R19             ;
OUT    PORTD, R19      ;
RJMP   AGAIN          ;
    
```

- ¿Qué hace el CPU durante la implementación del retardo o del contador?

- Una interrupción es una suspensión inmediata de la ejecución de un programa, enviando al contador del programa a otra acción/subrutina. Esta suspensión se produce, por ejemplo, cuando un periférico requiere la atención del procesador para realizar una operación de I/O.
- Las interrupciones son quizá el mecanismo más importante para la conexión del microcontrolador con el mundo exterior, sincronizando la ejecución de programas con acontecimientos externos/internos.
- Las interrupciones se pueden definir como: eventos que hacen que el microcontrolador deje de ejecutar la tarea que esta realizando para atender dicho acontecimiento y luego regresar a continuar la tarea que estaba realizando.

- Entonces, es un mecanismo que permite ejecutar un bloque de instrucciones, deteniendo la ejecución del programa y luego restablecer su ejecución sin afectarlo.
- En los microcontroladores AVR, para cada interrupción existe una Rutina de Servicio de Interrupción (ISR).
- Cuando se invoca una interrupción, el microcontrolador ejecuta el servicio de rutina de interrupción. En muchos microcontroladores, para cada interrupción hay una localidad fija en memoria que guarda la dirección de su ISR.
- En el AVR, el grupo de ubicaciones de memoria reservado para las direcciones de las ISR's se denomina tabla de vector de interrupciones.

Tras la activación de una interrupción, el microcontrolador realiza los siguientes pasos:

1. Finaliza la instrucción en ejecución y guarda la dirección de la siguiente instrucción (contador de programa) en el stack.
2. Brinca a una localidad fija en la memoria llamada Interrupted Vector Table. La tabla de vectores de interrupción dirige el microcontrolador a la dirección del ISR.
3. El microcontrolador empieza a ejecutar la “subrutina de servicio de interrupción” hasta que alcanza la última instrucción de la subrutina, la cual debe ser RETI (volver de la interrupción).
4. Al ejecutar la instrucción RETI, el microcontrolador regresa al lugar donde se interrumpió. Primero, obtiene la dirección del contador del programa (PC) y reinicia la ejecución desde esa dirección.

Fuente de interrupciones. Interrupciones para los Timers, Interrupciones externas en pines, Interrupciones de hardware interno, Interrupciones de periféricos (USART, SPI, ADC).

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	RESET	External pin, power-on reset, brown-out reset and watchdog system reset
2	0x0002	INT0	External interrupt request 0
3	0x0004	INT1	External interrupt request 1
4	0x0006	PCINT0	Pin change interrupt request 0
5	0x0008	PCINT1	Pin change interrupt request 1
6	0x000A	PCINT2	Pin change interrupt request 2
7	0x000C	WDT	Watchdog time-out interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 compare match A
9	0x0010	TIMER2 COMPB	Timer/Counter2 compare match B
10	0x0012	TIMER2 OVF	Timer/Counter2 overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 capture event
12	0x0016	TIMER1 COMPA	Timer/Counter1 compare match A

13	0x0018	TIMER1 COMPB	Timer/Counter1 compare match B
14	0x001A	TIMER1 OVF	Timer/Counter1 overflow
15	0x001C	TIMER0 COMPA	Timer/Counter0 compare match A
16	0x001E	TIMER0 COMPB	Timer/Counter0 compare match B
17	0x0020	TIMER0 OVF	Timer/Counter0 overflow
18	0x0022	SPI, STC	SPI serial transfer complete
19	0x0024	USART, RX	USART Rx complete
20	0x0026	USART, UDRE	USART, data register empty
21	0x0028	USART, TX	USART, Tx complete
22	0x002A	ADC	ADC conversion complete
23	0x002C	EE READY	EEPROM ready
24	0x002E	ANALOG COMP	Analog comparator
25	0x0030	TWI	2-wire serial interface
26	0x0032	SPM READY	Store program memory ready

- Al resetear todas las interrupciones están deshabilitadas (enmascaradas) por lo que el microcontrolador no responderá a ellas. El bit D7 es el responsable de desenmascarar (habilitar) ó deshabilitar a todas las interrupciones.
- Pasos para habilitar una interrupción
 - ◆ Poner uno en D7 del SREG. La instrucción SEI lo realiza.

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

- **Bit 7 – I: Global Interrupt Enable**

con I=1 en SREG, cada interrupción se habilita poniendo uno en su bandera de interrupción correspondiente.

■ Ejemplo, interrupciones de Timer0

TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

cuando $TOIE0 = 1$ e $I = 1$ la interrupción de desbordamiento para el Timer/Counter0 se habilita.

■ Ejemplo, interrupciones en el periférico USART

UCSRnA – USART Control and Status Register n A

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREN	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

La bandera UDREN indica si el buffer de transmisión (UDRn) esta listo para recibir información. Si UDREN=1, el buffer esta vacío y por lo tanto listo para recibir información. La bandera UDREN puede generar una interrupción al escribir uno en UDRIEn. La interrupción queda habilitada con I = 1.

- Tras la activación de la interrupción, el propio AVR borra el bit I para asegurarse de que otra interrupción no pueda activarse. Al final de la ISR, la instrucción RETI fijará $I = 1$ permitiendo que pueda entrar otra interrupción.

Como se sabe, la bandera de desbordamiento (TOV0) del timer0 se activa cuando el timer0 se desborda. Para detectar el desbordamiento se monitorea a la bandera del timer0 con la instrucción SBRS sobre TOV0. El problema con este método es que el microcontrolador queda atado al monitoreo de TOV0 sin poder hacer nada más.

Si la interrupción de desbordamiento TOIE0 del registro TIMSK0 se habilita, cuando el desbordamiento ocurra el microcontrolador saltará a la tabla de vectores de interrupción para dar servicio al ISR. De esta manera, el microcontrolador puede hacer otras cosas hasta que se le notifique que el temporizador se ha desbordado.

- Ejemplo, transferir datos de PORTC a PORTD y generar una onda cuadrada en PORTB.5.

```

.ORG      0x00                ;           —
        JMP      MAIN          ;           —
.ORG      0x0020              ;           —
        JMP      T0_OV_ISR     ;           —
.ORG      0x100               ;           —
MAIN:     LDI      R20, HIGH(RAMEND) ;           —
        OUT      SPH, R20      ;           —
        LDI      R20, LOW(RAMEND) ;           —
        OUT      SPL, R20      ;           —
        SBI      DDRB, 5       ;           —
        LDI      R20, (1<<TOIEO0) ;           —
        OUT      TIMSK0, R20    ;           —
        SEI                      ;           —
        LDI      R20, -32       ;           —
        OUT      TCNT0, R20     ;           —
        LDI      R20, 0x01      ;           —
        OUT      TCCR0, R20     ;           —
        LDI      R20, 0x00      ;           —
        OUT      DDRC, R20      ;           —
        LDI      R20, 0xFF      ;           —
        OUT      DDRD, R20      ;           —
HERE:     IN       R20, PINC      ;           —
        OUT      PORTD, R20     ;           —
        JMP      HERE          ;           —
.ORG      0x200               ;           —
T0_OV_ISR:: IN      R16, PORTB    ;           —
        LDI      R17, 0x20      ;           —
        EOR      R16, R17       ;           —
        OUT      PORTB, R16     ;           —
        LDI      R16, -32       ;           —
        OUT      TCNT0, R16     ;           —
        RETI                    ;           —

```

- Ejemplo, transferir datos de PORTC a PORTD y generar una onda cuadrada en PORTB.5 con Timer0 y PORTB.7 con Timer1.

```

.ORG    0x00                ;           —
        JMP      MAIN        ;           —
.ORG    0x001A              ;           —
        JMP      T1_OV_ISR   ;           —
.ORG    0x0020              ;           —
        JMP      T0_OV_ISR   ;           —
.ORG    0x100
MAIN:    LDI      R20, HIGH(RAMEND) ;           —
        OUT      SPH, R20        ;           —
        LDI      R20, LOW(RAMEND) ;           —
        OUT      SPL, R20        ;           —
        SBI      DDRB, 5         ;           —
        SBI      DDRB, 7         ;           —
        LDI      R20, (1<<TOIEO0) ;           —
        OUT      TIMSK0, R20      ;           —
        LDI      R20, (1<<TOIEO1) ;           —
        STS      TIMSK1, R20      ;           —
        SEI                        ;           —
        LDI      R20, -160        ;           —
        OUT      TCNT0, R20       ;           —
        LDI      R20, 0x01        ;           —
        OUT      TCCR0, R20       ;           —
        LDI      R20, HIGH(-640)  ;           —
        LDI      R21, LOW(-640)   ;           —
        STS      TCNT1H, R20      ;           —
        STS      TCNT1L, R21      ;           —
        LDI      R20, 0x00        ;           —
        LDI      R21, 0x01        ;           —
        STS      TCCR1A, R20      ;           —
        STS      TCCR1B, R21      ;           —

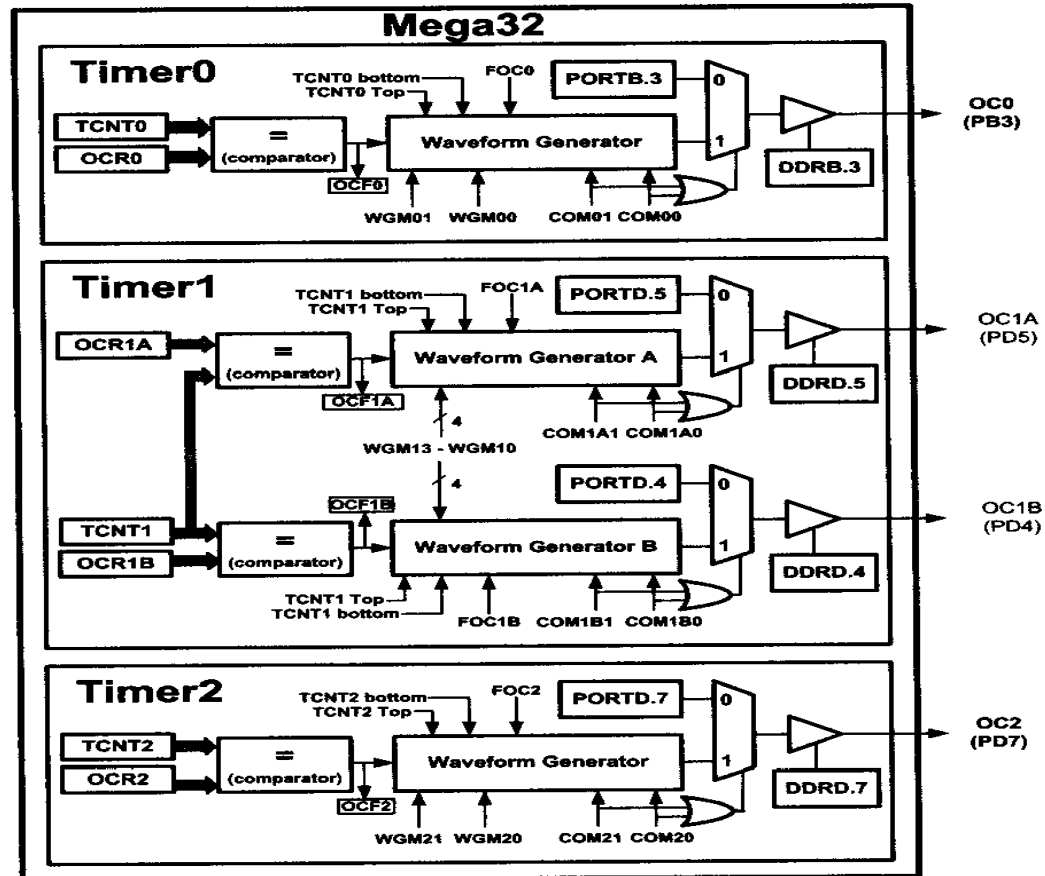
```

```

LDI    R20, 0x00      ;      —
LDI    R21, 0x01      ;      —
STS    TCCR1A, R20    ;      —
STS    TCCR1B, R21    ;      —
LDI    R20, 0x00      ;      —
OUT    DDRC, R20      ;      —
LDI    R21, 0xFF      ;      —
OUT    DDRD, R21      ;      —
HERE:  IN    R20, PINC  ;      —
      OUT    PORTD, R20 ;      —
      JMP    HERE      ;      —
.ORG    0x200          ;      —
T0_OV_ISR:: LDI    R16, -160 ;      —
      OUT    TCNT0, R16 ;      —
      IN     R16, PORTB ;      —
      LDI    R17, 0x02 ;      —
      EOR    R16, R17 ;      —
      OUT    PORTB, R16 ;      —
      RETI   ;      —
.ORG    0x300          ;      —
T1_OV_ISR:: LDI    R18, HIGH(-640) ;      —
      LDI    R19, LOW(-640) ;      —
      STS    TCNT1H, R18 ;      —
      STS    TCNT1L, R19 ;      —
      IN     R18, PORTB ;      —
      LDI    R19, 0x80 ;      —
      EOR    R18, R19 ;      —
      OUT    PORTB, R18 ;      —
      RETI   ;      —

```


La familia de microcontroladores ATmega32 viene con al menos 3 timers, los cuales pueden ser usados como generadores de onda como se muestra en la siguiente figura:



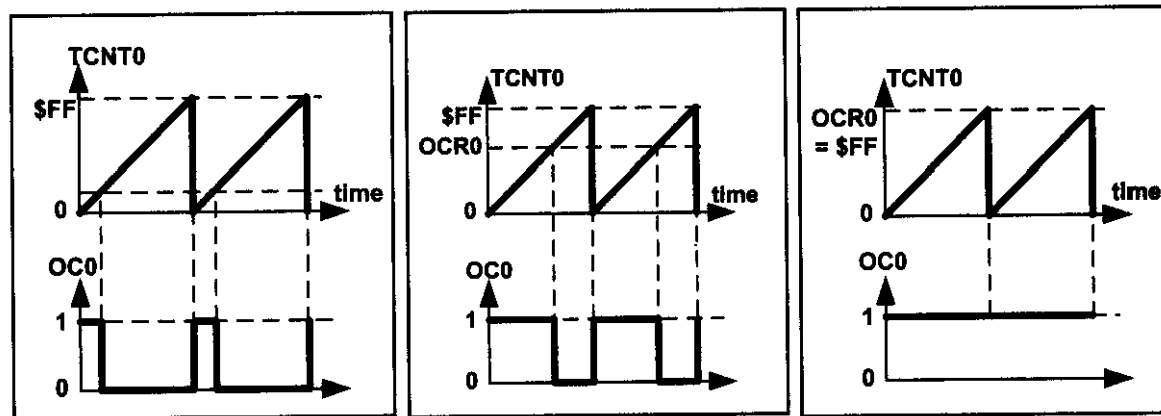
La ventaja de utilizar el PWM incorporado del AVR es que nos da la opción de programar el período y el ciclo de trabajo.

Modo PWM rápido. En el PWM rápido, el contador cuenta como el modo normal.

En la siguiente figura se puede ver la reacción del generador de onda cuando el comparador de emparejamiento ocurre mientras el timer esta en modo rápido PWM.

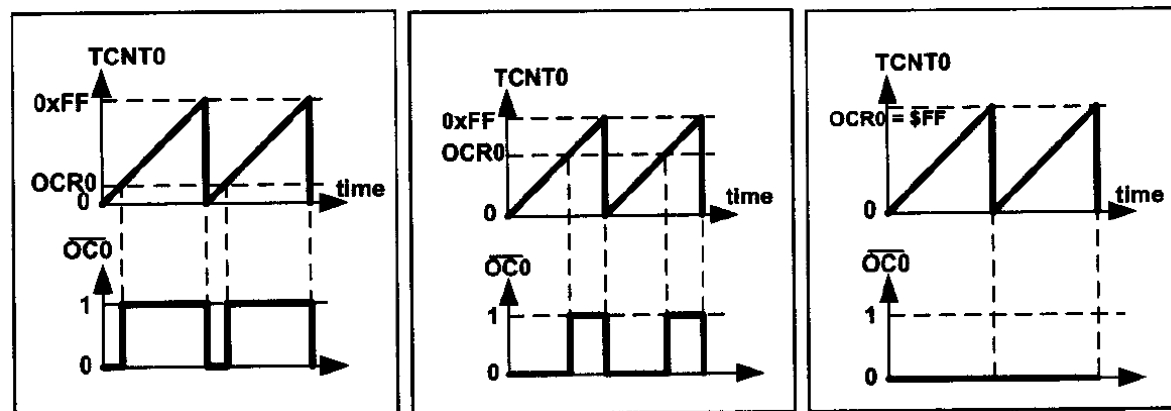
Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0
FOC0	D7							
WGM01:00	D3D6	Timer0 mode selector bit						
	0 0	Normal						
	0 1	PWM, Phase correct						
	1 0	CTC (Clear Timer on Compare match)						
	1 1	Fast PWM						
COM01:00	D5 D4	Compare Output Mode when Timer0 is in Fast PWM mode						
COM01	COM00	Mode Name	Description					
0	0	Disconnected	Normal port operation, OC0 disconnected					
0	1	Reserved	Reserved					
1	0	Non-inverted	Clear OC0 on compare match, set OC0 at TOP					

Cuando $COM01:00=10$, el generador de forma de onda borra el pin OC0 cada vez que se produce una comparación, por lo que deja en alto OC0 hasta antes de que la comparación sea el mismo valor, después, OC0 es cero hasta llegar al conteo máximo que es cuando vuelve a poner en 1 OC0. Este modo es llamado *PWM no invertido*.



Como se puede observar en las figuras, en el PWM no invertido, el ciclo de trabajo del generador de ondas incrementa cuando el valor de OCR0 incrementa.

Cuando COM01:00=11, el generador de forma de onda pone en 1 el pin OC0 cada vez que se producen coincidencias, antes de eso se encontrará en cero. Al llegar al limite el contador lo vuelve a borrar.



Podemos ver que en el modo de PWM invertido cuando el valor de OCR0 se incrementa, el ciclo de trabajo del generador de onda se decrementa.