Microcontroladores

Microcontrolador ATMEL Atmega16A

Teresa Orrañanos Guerrero

Agosto **2020**

Ciclo **1208**

Contenido

Fusibles del microcontrolador

Para que el microcontrolador funcione de una manera determinada, es posible modificar algunos fusibles (en el software de programación). El ATmega16A cuenta con 16 fusibles, distribuidos en dos bytes (alto y bajo). A continuación, se muestra la tabla que define al fusible alto:

Nombre	No. de Bit	Descripción	Valor por defecto
OCDEN	7	Habilitación OCD	1 (OCD deshabilitado)
JTAGEN	6	Habilitación JTAG	0 (JTAG habilitado)
SPIEN	5	Habilitación de programación serial SPI	0 (prog. SPI habilitado)
CKOPT	4	Opciones del oscilador	1
EESAVE	3	Los datos en la memoria EEPROM son preservados cuando se borra el chip	1 (EEPROM no preservado)
BOOTSZ1	2	Selección del tamaño del Bootloader	0
BOOTSZ0	1	Selección del tamaño del Bootloader	0
BOOTRST	0	Selección del vector de reset	1

A continuación, se muestra la tabla que define el byte de fusibles bajo:

Nombre	No. de Bit	Descripción	Valor por defecto
BODLEVEL	7	Nivel de detección bajo voltaje alim.	1
BODEN	6	Habilitación detector bajo voltaje alim.	1 (BOD deshabilitado)
SUT1	5	Selección de tiempo de arranque	1
SUT0	4	Selección de tiempo de arranque	0
CKSEL3	3	Selección fuente de reloj	0
CKSEL2	2	Selección fuente de reloj	0
CKSEL1	1	Selección fuente de reloj	0
CKSEL0	0	Selección fuente de reloj	1

A pesar de que no se entrará a explicar a detalle cada uno de estos fusibles a continuación, se presenta una breve nota al respecto de cada uno de ellos, con la finalidad de que en caso de requerirse esta información pueda ser utilizada más adelante.

OCDEN: Este fusible habilita o deshabilita la depuración On-Chip del microcontrolador, normalmente esta depuración no se empleará durante el curso POR LO CUAL RESULTA MUY IMPORTANTE DESHABILITAR ESTA OPCIÓN para poder utilizar el puerto C del microcontrolador.

JTAGEN: Este fusible habilita o deshabilita la interfaz JTAG que se encuentra en el Puerto C del ATmega16. Es importante que cuando no se vaya a usar esta interfaz se garantice que este deshabilitada para que haya un funcionamiento normal del Puerto C.

SPIEN: Este fusible habilita o deshabilita la programación serial SPI. Si se está usando este modo de programación, no es posible cambiar este bit.

CKOPT: Este fusible selecciona entre dos modos de amplificador del oscilador. Cuando está programado el reloj es más inmune al ruido y se pueden manejar un rango amplio de frecuencias, aunque con un consumo mayor de potencia.
EESAVE: La habilitación de este fusible genera que los datos en la memoria EEPROM no sean borrados cuando se realice una operación de borrado del microcontrolador.

BOOTSZ1:0: Estos fusibles establecen el tamaño del Bootloader.
BOOTRST: Cuando este fusible es programado, el dispositivo salta a la dirección de reset asignada con los fusibles BOOTSZ1:0. Si no es programado, iniciara en la posición cero.
BODLEVEL: Este fusible es usado para seleccionar entre los dos niveles de voltaje en el cual el microcontrolador se reinicia cuando la alimentación está por debajo del nivel seleccionado. Si no está programado este fusible, el nivel de voltaje es de 2.7V y si está programado el nivel de voltaje es de 4.0V.
BODEN: Este fusible habilita la protección por bajo voltaje. Si este fusible es programado, el microcontrolador se reiniciará de acuerdo al nivel seleccionado con el fusible BODLEVEL.
SUT1:0: Estos fusibles seleccionan entre diferentes retardos para el funcionamiento inicial del dispositivo. El valor seleccionado depende del reloj seleccionado.
CKSEL3:0: Estos fusibles permiten seleccionar entre diferentes fuentes de reloj para el dispositivo.

Modo JTAG

A continuación, se profundizará al respecto del modo JTAG (sobre el cual trata el bit JTAGEN del byte alto de fusibles), el cual resulta muy importante debido a que de ello depende el funcionamiento del puerto C del microcontrolador.
JTAG viene de las iniciales “Joint Test Action Group” que corresponde a un estadar de la IEEE 1149.1. Normalmente ese modo se emplea para la programación y depuración de programas IC (en el dispositivo). En el modo JTAG el microcontrolador ATmega16A comparte cuatro pines del puerto C (PC2, PC3, PC4 y PC5). Por default el modo JTAG viene habilitado en el microcontrolador, y hasta que no se desactive, esos cuatro pines del puerto C no pueden utilizase como puertos de entrada/salida normales.
Existen dos métodos para deshabilitar el modo JTAG:

a) Método de programación
b) Método de fusibles

a) Método de programación

El registro MCUCSR del ATmega16A en el bit 7 corresponde a JTD. Para deshabilitar mediante programación el modo JTAG es necesario escribir por dos veces consecutivas un 1 en este bit. Cabe destacar que este método es temporal, puesto que cada vez que se escribe un nuevo programa es necesario incluir el código de modificación de este bit.

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

El código que se requiere agregar al programa sería

```
IN R__ , MCUCSR        ; Almaceno en R__ el contenido del MCUCSR
ORI R__ , 0b10000000   ; Escribo un 1 en el bit 7 del registro sin modificar los otros bits
OUT MCUCSR, R__        ; Almaceno el registro modificado en MCUCSR
IN R__ , MCUCSR        ; Repito las instrucciones anteriores para que se cambie dos veces el valor
ORI R__ , 0b10000000
OUT MCUCSR, R__
```

b) Método de fusibles

El modo JTAG puede deshabilitarse permanentemente mediante la configuración de dos bits de fusibles OCDEN y JTAGEN, los cuales deberán estar deshabilitados para que el modo JTAG se encuentre también deshabilitado y de esa forma puedan utilizase normalmente los pines del 2 al 5 del puerto C.
A continuación, se muestra el byte alto de fusibles con sus valores por defecto.

Fuse High Byte

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	**0**	**0**	**1**	**1**	**0**	**0**	**1**
OCDEN	JTAGEN	SPIEN	CKOPT	EESAVE	BOOTSZ1	BOOTSZ0	BOOTRST

Para que se deshabiliten los bits correspondientes a OCDEN y JTAGEN únicamente es necesario modificar los bits 7 y 6 escribiendo en ellos un 1 (deshabilitar).

Cabe resaltar que, al utilizar este método, una vez realizado el cambio en los fusibles, no se requiere incluir ningún código especial en el programa, y el cambio realizado permanecerá hasta que se realice alguna otra modificación en los fusibles.

Opciones para el reloj interno del microcontrolador

Para profundizar en el conocimiento de la velocidad del reloj interno del microcontrolador es importante entrar a detalle en el funcionamiento del byte bajo de fusibles.

Fuse Low Byte

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	**1**	**1**	**0**	**0**	**0**	**0**	**1**
BODLEVEL	BODEN	SUT1	SUT0	CKSEL3	CKSEL2	CKSEL1	CKSEL0

En el byte de fusibles bajo nos interesa especialmente identificar con claridad los bits 3..0 que llevan el nombre de CKSEL3..0 puesto que ellos nos permiten controlar la velocidad del reloj interno de microcontrolador.

El microcontrolador ATmega16A tiene la opción de utilizar diferentes fuentes para el reloj, estas fuentes pueden consistir en un cristal externo, un cristal de baja frecuencia, un oscilador RC externo, un oscilador RC interno (que por default viene configurado a 1Mhz (con CKSEL3..0 = 0001) o un reloj externo.

La siguiente tabla muestra los rangos de valores que pueden tomar los bits CKSEL3..0 en el byte de fusibles bajo, según el tipo de reloj que se desee configurar.

Table 8-1. Device Clocking Options Select(1)

Device Clocking Option	CKSEL3:0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

Note: 1. For all fuses "1" means unprogrammed while "0" means programmed.

Por el momento se trabajará con configurar el oscilador RC interno del microcontrolador (opciones entre 0100 – 0001).

El reloj interno del microcontrolador puede funcionar a 1.0 Mhz, 2.0 Mhz, 4.0 Mhz o 8 Mhz (estos valores son exactos (±3%) siempre y cuando el voltaje que se aplique al microcontrolador sea de 5V y se encuentre a una temperatura de 25°C). Cuando se configura el reloj interno del microprocesador no es necesario conectar ningún componente (cristal, resistencia o capacitor) adicional al circuito.

A continuación se muestra la tabla que indica los valores que los bits CKSEL pueden tomar de acuerdo con la frecuencia que se desea para el funcionamiento del microcontrolador

Table 8-8. Internal Calibrated RC Oscillator Operating Modes

CKSEL3:0	Nominal Frequency (MHz)
0001(1)	1.0
0010	2.0
0011	4.0
0100	8.0

Note: 1. The device is shipped with this option selected.

Cuando se selecciona este tipo de oscilador (reloj interno) se puede determinar también el tiempo de inicialización del micro, mediante los fusibles SUT localizados en el byte de fusibles bajo. A continuación, se muestra la tabla de los valores que se pueden asignar a los bits SUT.

Table 8-9. Start-up Times for the Internal Calibrated RC Oscillator Clock Selection

SUT1:0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1ms	Fast rising power
10[1]	6 CK	65ms	Slowly rising power
11	Reserved		

Note: 1. The device is shipped with this option selected.

Registros

Un registro es capaz de almacenar 8 bits en la forma en que se muestra a continuación

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0

Es decir que un registro puede almacenar números entre 0000_0000 y 1111_1111 (en binario) o entre 0 y 255 (en decimal) o entre 0x00 y 0xFF.

Las ventajas de los registros, comparadas con otros medios de almacenamiento son:

- Pueden usarse directamente en los comandos de ensamblador.
- Las instrucciones con registros se procesan con más facilidad pues están conectados directamente al acumulador.
- Pueden funcionar como fuente y como destino en las operaciones.

En un AVR ATmega16A se tienen 32 registros (R0…R31), y nos podemos referir a ellos directamente, sin embargo, también es posible asignarles nombres de la siguiente manera:

.DEF MiRegistro = R16

Cuando se quiere introducir un valor en un registro se puede hacer de la siguiente manera:

LDI MiRegistro, 255

Lo cual significa que introducimos de forma inmediata (**LoaD I**mmediate) el valor 255 al registro 16. Si deseamos indicar el número en forma binaria, la instrucción se escribiría:

LDI MiRegistro, 0b1111_1111
LDI MiRegistro, 0b11111111

O bien, si queremos indicar el número en forma hexadecimal, escribiríamos:

LDI MiRegistro, 0xFF
LDI MiRegistro, $FF

El resultad de cualquiera de estas tres formas de expresar la instrucción es exactamente el mismo.

Puede haber momentos en que necesitemos copiar información entre dos registros, para lo cual se utiliza el comando MOV, cabe hacer mención que a pesar de que dicho comando viene de la palabra “move” que en inglés significa mover, la función que realiza en realidad es una “copia” del dato de un registro a otro

.DEF MiRegistro = R16
.DEF OtroRegistro = R18
LDI MiRegistro, 0xFF
MOV OtroRegistro, MiRegistro

El código anterior da por resultado que en R18 (OtroRegistro) quede el dato binario 0b11111111.

MOV Destino, Fuente

Es decir, el primer registro es el lugar en donde queremos copiar los datos, y el segundo registro corresponde a los datos que queremos copiar.

Registros de acceso inmediato

Por lo que sabemos hasta ahora, parecería que el siguiente código es correcto

```
.DEF MiRegistro = R15
LDI MiRegistro, 0xFF
```

Sin embargo, no lo es. Únicamente los registros que van del R16 al R31 tienen la capacidad de guardar datos con el tipo de direccionamiento inmediato, por lo cual, si escribiéramos el código anterior, obtendríamos un error en la 2ª. línea.

Únicamente existe una excepción, con el comando CLR (que equivale a mover un 0x00 al registro). Dicha instrucción puede implementarse con cualquiera de los registros (R0..R31)

```
.DEF MiRegistro = R15
CLR MiRegistro
```

Otras instrucciones que únicamente pueden implementarse en los registros que van del R16 al R31 son:

- ANDI Rx,K
- CBR Rx,M
- CPI Rx,K
- SBCI Rx,K
- SBR Rx,M
- SER Rx
- SUBI Rx,K

Puertos de entrada y salida

Los puertos de un microprocesador son la manera de comunicar la unidad de procesamiento con el hardware externo. Un micro tiene la capacidad tanto de leer los puertos como de escribir en ellos.

Cada puerto tiene asignada una dirección específica, por ejemplo, el puerto B tiene asignada la dirección 0x18, pero no es necesario aprender de memoria la dirección que corresponde a cada uno de los puertos, pues vienen ya definidas en .include <m16Adef.inc>un archivo que se puede incluir al inicio de nuestro programa. La forma de incluir dicho archivo es:

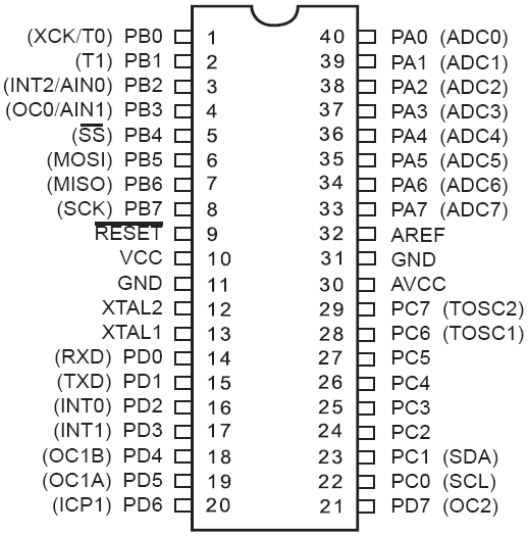
.include <m16Adef.inc>

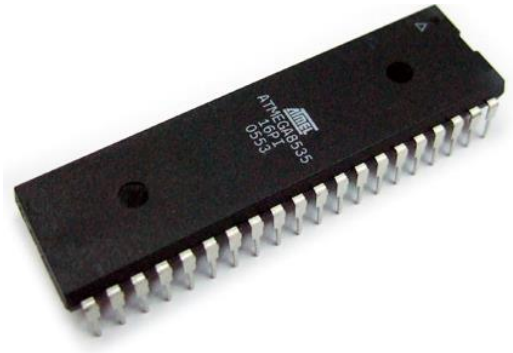
(se sugiere abrirlo mediante un block de notas, para ver cómo es, pero cuidando de no modificar nada).

Este archivo, contiene código similar a:

.EQU PORTB, 0x18

Además, incluye el código necesario para definir los registros, de forma que podamos referirnos a ellos directamente por su nombre.





El AVR ATmega16A incluye 4 puertos (PortA, PortB, PortC PortD), que pueden ser usados como puertos de entrada o de salida. Para poder trabajar con ellos primero es necesario configurarlos.

A manera de ejemplo digamos que se desea trabajar con el puerto A. Si consultamos las características del AVR ATmega16A (datasheet), encontraremos que el puerto A tiene asignadas las siguientes localidades de memoria:

- Port A Data Register – PORTA
- Port A Data Direction Register – DDRA
- Port A Input Pins Ardes – PINA

Definiendo puertos de entrada o salida

Primero es necesario indicarle al microprocesador si deseamos utilizar el puerto de entrada o de salida, eso se hace a través de la dirección PortA. Si enviamos el número 0xFF (0b11111111) al PORTA, estamos indicando que deseamos emplear todo el puerto A como salida, en cambio si enviamos el número 0x00 (0b00000000), le estamos diciendo que queremos utilizarlo como puerto de entrada (Es posible indicar bit por bit cuál es de entrada y cual es de salida).

```
LDI R16, 0xFF          ;Guarda un 255 en R16
OUT DDRA, R16          ;Al mandar 0b11111111 a DDRA quedan definidos todos los bits como de salida
```

Nota: cuando dentro del código utilizamos ";" equivale a indicar que lo que continúa en esa línea es un comentario.

Configurando resistencias de Pull-Up para puertos de entrada

Los AVR incorporan en todos sus puertos transistores a manera de fuente de corriente que en la práctica funcionan como resistencia de pull-up, estos pull-up nos pueden ahorrar el uso de resistencias externas hacia voltaje en los pines que son configurados como entradas. Estas pull-up se podrían equiparar con resistencias de entre 20k y 50k , a partir de dichos valores se podría calcular la corriente que pude fluir a través de ellas si están activas.

El registro SFIOR (Special function IO register), mediante el bit PUD (pull up disable) permite controlar en general todas las resistencias de pull up de los puertos del microcontrolador AVR ATmega16A.

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Cuando PUD=1 entonces todas las resistencias de pull up de los puertos del microcontrolador se encontrarán deshabilitadas, sin importar si a través del registro individual del puerto (PORTx) se indica que estén habilitadas. Por el contrario, cuando PUD=0, entonces las resistencias de pull up pueden ser configuradas para estar o no estar habilitadas, según la configuración individual de cada puerto a través del registro PORTx. El valor por default del pin PUD es 0 (es decir que si se desean emplear las resistencias de pull up, no es necesario modificar este valor).

En caso de que desee cambiar la configuración inicial del microcontrolador, de forma que todas las resistencias de pull up se encuentren deshabilitadas deberán enviarse las siguientes instrucciones

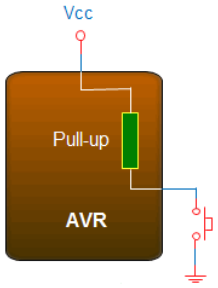
```
ldi r16, 0b00000100
out SFIOR, r16
```

Si se desean regresar a la configuración inicial del microcontrolador, de forma que las resistencias puedan ser habilitadas mediante bits individuales, entonces se envían las instrucciones:

```
ldi r16, 0b00000100
out SFIOR, r16
```

Las resistencias de pull-up se pueden habilitar bit por bit, enviando un 1 al bit donde se quiere que funcionen y un 0 al bit donde no se desea habilitar, al registro PORTx (PortA, PortB,…) . Cabe hacer notar que las resistencias de Pull-Up solamente se pueden configurar cuando se trata de un puerto de entrada, ya que cuando se trata de un puerto de salida, las Pull-Up quedan automáticamente deshabilitadas.

En la siguiente figura se muestra la conexión de un botón al AVR ATmega16A aprovechando la resistencia de pull-up del micro.



Después de haber configurado el puerto como puerto de entrada, se debe utilizar la siguiente instrucción para habilitar (con 1's) o deshabilitar (con 0's) las resistencias de Pull-Up.

```
LDI R16, 0b11110000     ;Ponemos 1's en los pines en que se desea Pull-Up y 0's en los que no
OUT PORTA, R16          ;Se envía el registro al PORT correspondiente.
```

Enviando datos a través de un puerto de salida

Una vez configurado el puerto, si lo hemos definido como puerto de salida, en el momento en que deseemos enviar datos a través de el, se emplea la siguiente instrucción.

```
LDI R16, 0b11110000
OUT PORTA, R16
```

Lo cual resultaría equivale a enviar un 0 por los pines de 0 al 3 y un 1 por los pines del 4 al 7 del puerto A.

Recibiendo datos a través de un puerto de entrada

En cambio, si el puerto ha quedado definido como puerto de entrada, en el momento en que deseemos consultar la información que hay en el puerto, empleamos la siguiente instrucción.

```
IN R16, PINA
```

Que obtendrá los datos presentes en ese momento en el puerto A y los dejará almacenados en el R16.

Códigos de condición – Registro de Estado

El AVR ATmega16A tiene un registro de estado en el que se reflejan las operaciones lógicas y aritméticas del ALU. Este registro recibe el nombre de SREG.

Bit	7	6	5	4	3	2	1	0	
	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 – Global Interrupt Enable (I)

Cuando en el bit 7 del registro SREG se escribe 1 se habilita la posibilidad de que el micro emplee interrupciones (las interrupciones concretas que se desean habilitar deben configurarse también en sus registros correspondientes).

Para poner un 1 en este bit del registro SREG se puede escribir la instrucción

SEI

Cuando en este bit se pone 0, las interrupciones del micro quedan deshabilitadas (sin importar si están habilitadas en forma individual en los registros correspondientes).

Para poner un 0 en este bit del registro SREG se puede escribir la instrucción

CLI

Bit 6 – Bit Copy Storage (T)

Las instrucciones para copiar bits (Bit LoaD – BD y Bit STore – BST) usan el bit T como fuente o destino para el bit de la operación. Con la instrucción BST se puede copiar un bit de un registro de trabajo al bit T y con la instrucción BLD se puede copiar el bit T a un bit de un registro de trabajo.

Bit 5 – Half Carry Flag (H)

La bandera H indica un medio acarreo en algunas operaciones aritméticas. El bit H es muy útil en aritmética BCD.

Bit 4 – Sign Bit (S)

El contenido de este bit es una operación OR EXCLUSIVA entre los registros N y V

$S=N \oplus V$

Bit 3 – Two's complement overflow Flag

Se utiliza en operaciones aritméticas en complemento a dos. Para más detalles ver el Set de instrucciones del AVR ATmega16A.

Bit 2 – Negative Flag (N)

La bandera N indica un resultado negativo en una operación aritmético o lógica

Bit 1 – Zero Flag (Z)

La bandera Z indica que el resultado de una operación aritmética o lógica es cero

Bit 0 – Carry Flag (C)

La bandera C indica que hubo un acarreo en la operación lógica o aritmética

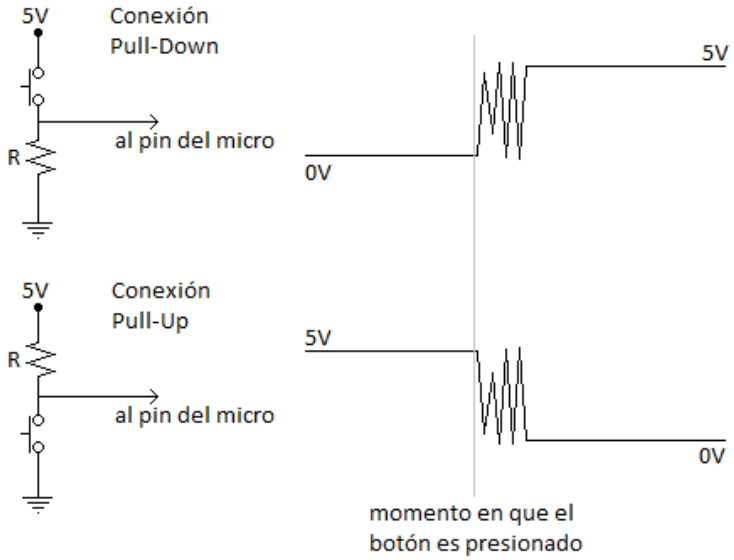
Instrucciones que modifican el registro de estado

La mayoría de los bits del registro SREG se modifican en forma automática después de realizar operaciones o llevar a cabo ciertas instrucciones. A continuación, se tiene una tabla de todas las instrucciones de ensamblador que pueden modificar cada uno de estos bits.

Bit	Aritméticas	Lógicas	Comparación	Bits	Shift	Otras
Z	ADD, ADC, ADIW, DEC, INC, SUB, SUBI, SBC, SBCI, SBIW	AND, ANDI, OR, ORI, EOR, COM, NEG, SBR, CBR	CP, CPC, CPI	BCLR Z, BSET Z, CLZ, SEZ, TST	ASR, LSL, LSR, ROL, ROR	CLR
C	ADD, ADC, ADIW, SUB, SUBI, SBC, SBCI, SBIW	COM, NEG	CP, CPC, CPI	BCLR C, BSET C, CLC, SEC	ASR, LSL, LSR, ROL, ROR	-
N	ADD, ADC, ADIW, DEC, INC, SUB, SUBI, SBC, SBCI, SBIW	AND, ANDI, OR, ORI, EOR, COM, NEG, SBR, CBR	CP, CPC, CPI	BCLR N, BSET N, CLN, SEN, TST	ASR, LSL, LSR, ROL, ROR	CLR
V	ADD, ADC, ADIW, DEC, INC, SUB, SUBI, SBC, SBCI, SBIW	AND, ANDI, OR, ORI, EOR, COM, NEG, SBR, CBR	CP, CPC, CPI	BCLR V, BSET V, CLV, SEV, TST	ASR, LSL, LSR, ROL, ROR	CLR
S	SBIW	-	-	BCLR S, BSET S, CLS, SES	-	-
H	ADD, ADC, SUB, SUBI, SBC, SBCI	NEG	CP, CPC, CPI	BCLR H, BSET H, CLH, SEH	-	-
T	-	-	-	BCLR T, BSET T, BST, CLT, SET	-	-
I	-	-	-	BCLR I, BSET I, CLI, SEI	-	RETI

Utilizando botones

Cuando presionamos un botón o movemos un switch, la señal de voltaje no cambia directamente a su estado estable, sino que dará pequeños rebotes generando ondas irregulares. A continuación, se muestran las imágenes de las señales generadas según el tipo de conexión que se haya empleado para conectar el botón.



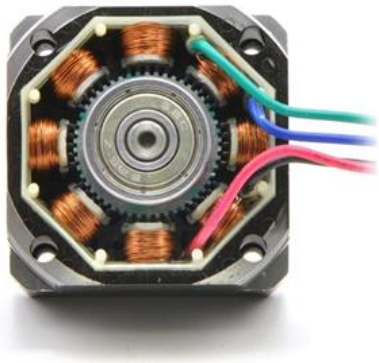
Los rebotes normalmente pueden ser corregidos implementando algún tipo de filtro, sin embargo, cuando se utiliza un microcontrolador resulta más sencillo añadir una rutina anti rebote al programa.

La manera más sencilla de realizar esta rutina anti rebote consiste en poner un retardo, de forma que una vez que se detecta un cambio en el voltaje, se manda el programa a la rutina de retardo para que espere un lapso de tiempo antes de que la señal se estabilice (entre 20 y 100 ms), mientras este tiempo transcurre el microcontrolador estará ocupado y no revisará lo que sucede en el puerto, al terminar este tiempo se hace la rutina deseada para responder al pulso de botón (de la forma que se describe el botón funcionará al momento de ser presionado). Deberá tener cuidado de realizar la programación necesaria para que al presionar el botón una solo vez no incremente la cuenta muchas veces.

Analice el siguiente código que irá decrementando de 255 a cero por 255 veces, es decir en total 65025

```
Retardo:
        ldi Temporal1,0xff
        ciclo1:
        dec Temporal
        breq Salir              ;Brinca cuando la bandera Z (cero) está activa
        ldi Temporal2,0xff
        ciclo2:
                dec Temporal2
                breq ciclo1     ;Brinca cuando la bandera Z (cero) está activa
        rjmp ciclo2
Salir:
        ret
```

Motor de pasos



Un motor paso a paso es un dispositivo electromecánico que convierte una serie de pulsos eléctricos en desplazamientos angulares, lo que significa que es capaz de girar una cantidad de grados (paso o medio paso) dependiendo de sus entradas de control.

Los motores paso a paso son ideales para la construcción de mecanismos en donde se requieren movimientos muy precisos.

Características

Los motores de pasos normalmente se definen por los siguientes parámetros:

Voltaje

Este valor viene impreso en su carcasa o por lo menos se especifica en su hoja de datos. Algunas veces puede ser necesario aplicar un voltaje superior para lograr que un determinado motor cumpla con el torque deseado, pero esto producirá un calentamiento excesivo y/o acortará la vida útil del motor.

Resistencia eléctrica

Esta resistencia determinará la corriente que consumirá el motor, y su valor afecta la curva de torque del motor y su velocidad máxima de operación.

Grados por paso

Es el punto más importante al momento de elegir un motor de pasos. Define la cantidad de grados que girará el eje para cada paso completo. Una operación de medio-paso o semi-paso (half step) del motor duplicará la cantidad de pasos por revolución al reducir la cantidad de grados por paso. Cuando el valor de grados por paso no está indicado en el motor, es posible contar a mano la cantidad de pasos por vuelta, haciendo girar el motor y sintiendo por el tacto cada "diente" magnético. Los grados por paso se calculan dividiendo 360 (una vuelta completa) por la cantidad de pasos que se contaron. Las cantidades más comunes de grados por paso son: 0,72°, 1.8°, 3.6°, 7.5°, 15° y hasta 90°. A este valor se le llama la resolución del motor.

Tipos de Torques

A continuación, se mencionan los diferentes torques que son importantes a considerar para el diseño de nuestro proyecto, es recomendar verificar las especificaciones de cada motor para hacer la comparativa. Los motores chinos es recomendable hacer pruebas ya que posiblemente no nos den lo que indica.

Torque de arranque (pull in torque): Es el torque máximo para vencer la inercia del rotor para empezar a girar a máxima velocidad o la velocidad indicada.

Torque de giro (pull out torque): Es el máximo torque que el motor puede proporcionar sin sufrir pérdida de pasos.

Torque de retención (detent torque): Es el torque máximo aplicado sin provocar la rotación del eje cuando el motor se encuentra sin energizar.

Torque de anclaje (holding torque): Es el torque máximo que puede ser aplicado sin provocar la rotación, ocurre al tener el motor parado y alimentado.

Categorías

Existen principalmente dos categorías de motores de pasos: de imán permanente y de reluctancia variable, aunque también es posible encontrar unos híbridos que unen las mejores características de los anteriores. La principal diferencia en cuanto a funcionamiento entre los de imán permanente y reluctancia variable es que, en los primeros, al estar en reposo el rotor queda fijo, en tanto que en los segundos el rotor puede girar libremente.

Los motores de pasos de imán permanente son los que se encuentran en las impresoras para el control del avance del papel y el cabezal, y se encuentran en tres subcategorías: unipolares, bipolares y multifase. En este curso trabajaremos con motores unipolares, por lo cual se explicará a detalle únicamente este tipo.

Motores de paso unipolares

Son los más simples de controlar. Normalmente tienen 5 o 6 cables dependiendo de su conexión interna. De estos cables 4 se utilizan para los pulsos que indican la secuencia y duración de los pasos, y los restantes sirven como alimentación del motor.

Existen tres secuencias para el manejo de este tipo de motores:

Secuencia normal

En esta secuencia se avanza un paso a la vez, debido a que siempre existen dos bobinas activas.

Con esta secuencia se obtiene un alto torque de paso y retención.

Bobina Paso	A	B	C	D
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

Secuencia de paso completo

En esta secuencia se activa una bobina a la vez.

Con esta secuencia el torque de paso y retención es menor, sin embargo, en algunos motores se produce un funcionamiento más suave.

Bobina Paso	A	B	C	D
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1

Secuencia de medio paso

En esta secuencia se activan las bobinas de manera que se combinan las secuencias anteriores, el resultado es un paso más corto (la mitad que en las secuencias anteriores).

Bobina Paso	A	B	C	D
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Motores de paso bipolares

Normalmente tienen 4 cables y se controlan a través de cambios de dirección en la corriente que circula a través de las bobinas.

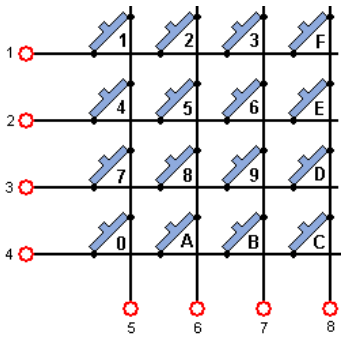
Motor de paso híbrido

La expresión Motor paso a paso híbrido se refiere a un motor eléctrico del tipo paso a paso, cuyo funcionamiento se basa en la combinación de los otros dos tipos de motores paso a paso, el Motor de reluctancia variable y el motor de magnetización permanente.

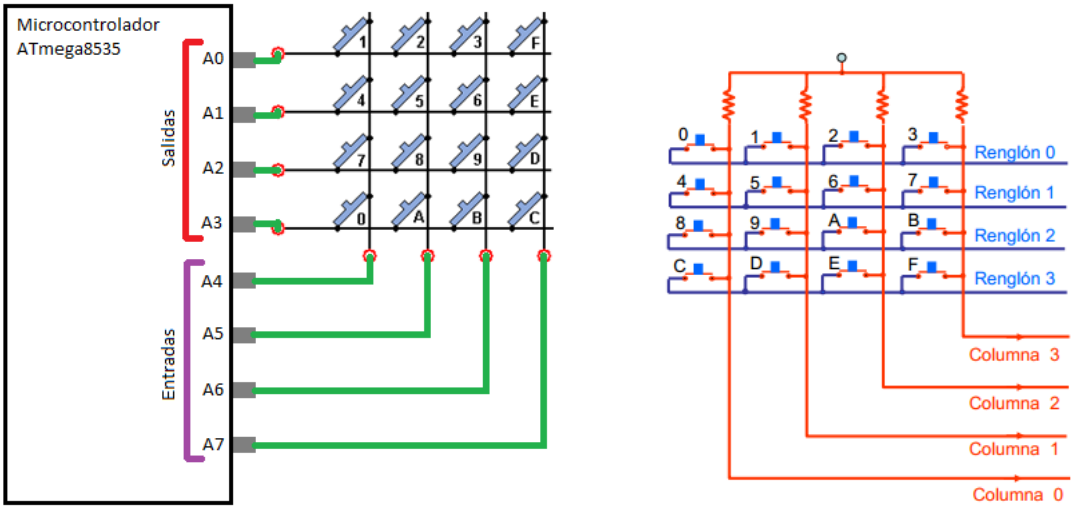
Teclado Matricial

Muchas veces es necesario utilizar un teclado matricial como entrada para nuestro microprocesador. La ventaja de utilizar un teclado matricial, comparado con utilizar únicamente botones independientes, es que se reduce el número de pines de entrada necesarios.

Un teclado matricial está compuesto por filas y columnas de cables, de manera que al pulsar una tecla se pone en contacto una fila con una columna. En la siguiente figura se muestra un diagrama de las conexiones.



Para utilizar un teclado matricial con el microprocesador se conectan las filas a pines de salida y las columnas a pines de entrada o viceversa. Es recomendable utilizar las resistencias de pull-up internas del microcontrolador para no tener que hacer conexiones externas. En la siguiente figura se muestra un ejemplo utilizando el puerto A:



Para detectar la pulsación de una tecla necesitamos escanear el teclado, esto se consigue mediante una rutina que consiste en ir poniendo una a una las líneas de entrada del teclado en nivel bajo. Cada vez que una fila se pone en nivel bajo se hacen cuatro comprobaciones para ver si una de las cuatro columnas ha cambiado de nivel y así saber la tecla pulsada.

Interrupciones Externas

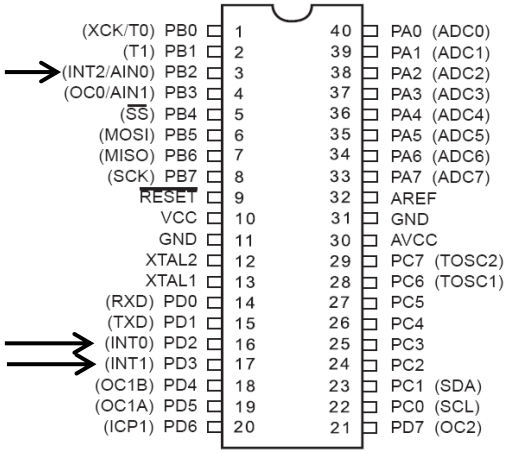
Hay muchas ocasiones en las que es necesario que nuestro programa reaccione ante determinados eventos de hardware, es decir, cuando el microcontrolador detecte un cambio en un pin de entrada. Esto puede hacerse programando un ciclo y monitoreando constantemente el pin deseado para que en el momento que se detecte un cambio, se lleve a cabo el ciclo deseado, sin embargo, este método puede no resultar efectivo cuando el pulso que se desea detectar es muy corto (µs) y el programa podría no alcanzar a pasar por la línea en la que se lee la entrada del pin, es por ello que en dichas ocasiones es conveniente utilizar interrupciones, las cuales pueden detectar cualquier pulso en el momento que se genere, siempre y cuando dicho pulso sea al menos un poco mayor que un periodo de la señal de reloj del microcontrolador.

Cuando se desea utilizar las interrupciones es necesario habilitarlas primero, pues por default todas se encuentran deshabilitadas. Para habilitarlas únicamente se modifican los bits necesarios en el Registro de Estado del microprocesador, correspondientes a las banderas de interrupciones, para que los brincos correspondientes a los vectores de interrupción queden habilitados.

Dentro del Set de instrucciones del ATmega16A, hay una instrucción específica para habilitar las interrupciones, se trata de **sei**

Sei – Set global Interrupt Flag

Para utilizar este comando únicamente es necesario escribirlo después de inicializar el stack pointer, y quitar los puntos y coma del código propuesto para la inicialización del microprocesador puesto que, al momento de generarse una interrupción, el programa irá directamente a esas líneas (0x01, 0x02, 0x12 dependiendo de la interrupción) para ejecutar ese código como resultado de ésta. En esas líneas hemos colocado la instrucción **rjmp** para mandar a la rutina que deseamos ejecutar (el nombre de la rutina puede cambiarse). Al escribir el código para dichas rutinas, en el momento que deseemos que el programa regrese a donde se había quedado cuando se generó la interrupción, se escribe el código **reti – Ret from Interruption**



Los pines que se utilizan para las interrupciones se muestran en la figura de la izquierda.

Las interrupciones INT0 e INT1 pueden ser activadas por flancos de bajada, flancos de subida, o niveles bajos, mientras que INT2 solamente se activa por flancos de subida o bajada.

Los niveles bajos en las interrupciones INT0 e INT1 y los flancos de subida o bajada en INT2 funcionan de manera asíncrona, lo cual implica que dichas interrupciones, configuradas de esta manera pueden funcionar para "despertar" al micro, pero esa función concreta se verá más adelante.

Una vez habilitadas las interrupciones, es necesario configurar algunos otros registros para indicar que utilizaremos las interrupciones externas y especificar la forma en la que queremos que trabajen las mismas.

Registro MCUCR

El **MCU Control Register (MCUCR)** tiene los siguientes bits:

Bit	7	6	5	4	3	2	1	0	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
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	

Mediante este registro se pueden configurar la Interrupción 0 y la Interrupción 1, para elegir si queremos que funcionen por niveles o por flancos.

De los bits de MCUCR es necesario configurar el bit 3 y bit 2 cuando se desea utilizar la interrupción 1 y el bit 1 y bit0 cuando se va a emplear la interrupción 0, a continuación se muestran las tablas tomadas de la hoja de especificación del microcontrolador AVR ATmega16A, en las que se muestran las posibles configuraciones para dichos pines.

Table 35. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

Table 36. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Registro MCUCSR

El registro que nos sirve para configurar la interrupción 2 es el MCUCSR, en su bit 6. El registro tiene la configuración de bits que se muestran a continuación:

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

Si el bit 6 de este registro tiene un cero la interrupción se activa con flancos de bajada, mientras que si tiene un uno se activa con flancos de subida. **Al momento de cambiar el contenido de ISC2 puede ser que se genere una interrupción por lo cual es recomendable realizar los cambios en este registro antes de configurar el registro GICR que se explicará a continuación.**

Registro GICR

Otro de los registros que tiene que ver con las interrupciones es el **General Interrupt Control Register (GICR)** cuya distribución de pines se muestra en seguida:

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Del registro GICR los pines del 7 al 5 indicados como INT1, INT0 e INT2 son los que nos sirven para habilitar las interrupciones, esto se logra poniendo un 1 en el contenido del pin correspondiente a la interrupción deseada, una vez que una interrupción es habilitada en este registro, cualquier actividad en el pin correspondiente a la interrupción generará que se corra la rutina correspondiente, aun cuando el pin de la interrupción esté configurado como puerto de salida.

Si la interrupción que se empleará tiene un 0 en este registro, la interrupción no estará activa y por tanto no funcionará, no importa si se habilitaron las interrupciones en general mediante la instrucción sei, ni si se configuró la interrupción en el registro MCUCR o MCUCSR.

Registro GIFR

Este registro corresponde al **General Interrupt Flag Register (GIFR)**

Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	INTF2	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Los pines INTF0, INTF1 e INTF2 correspondientes al bit7, bit6 y bit5 del registro GIFR cambian su valor de 1 a 0 en forma automática en el momento en que se detecta la interrupción, y regresan a 1 una vez que el microprocesador entra a la rutina que se programó para la misma.

Es importante que estos bits se configuren como 1 antes de habilitar alguna interrupción concreta, para evitar de esta forma que al habilitar las interrupciones se genere en forma inmediata una entrada al código correspondiente.

También vale la pena hacer notar que si en algún momento se desea forzar la ejecución de una interrupción (desde el código), esto se puede hacer cargando un 0 al bit correspondiente de la interrupción en este registro.

Programando las interrupciones externas

La primera línea de programación que es necesario utilizar para configurar las interrupciones es

```
sei     ; nos sirve para habilitar las interrupciones en general
```

Después configuramos el registro MCUCR (para las INT0 e INT1) y el registro MCUCSR (para la INT2) dependiendo de las interrupciones que vayamos a utilizar y de la forma en la que queremos que trabaje cada una de ellas.

Lo primero que requiere es saber el valor binario que cargará a MCUCR y MCUCSR según sea el caso, para generarlo puede ayudarse de la siguientes tablas:

GIFR

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
			0	**0**	**0**	**0**	**0**
INTF1	INTF0	INTF2	-	-	-	-	-
1 = bandera sin activar 0 = bandera activa	1 = bandera sin activar 0 = bandera activa	1 = bandera sin activar 0 = bandera activa					

El código necesario para limpiar las banderas de las interrupciones externas sería:

```
ldi R16, 0b1110_0000        ; se limpian todos los bits correspondientes a las banderas
out GIFR, R16               ; lo mandamos al GIFR
```

MCUCR

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	**0**	**0**	**0**				
SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00
				Ver tabla 35 para elegir el funcionamiento por flancos o niveles de INT1		Ver tabla 36 para elegir el funcionamiento por flancos o niveles de INT0	

MCUCSR

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0		**0**	**0**	**0**	**0**	**0**	**0**
-	ISC2	-	-	WDRF	BORF	EXTRF	PORF
	0 = flancos de bajada. 1 = flancos de subida.			Watch dog reset flag	Brown out reset flag	External reset flag	Power on reset flag

Una vez que se han determinado los valores binarios para el correcto funcionamiento de las interrupciones, es posible general el código correspondiente.

Si se desea utilizar la interrupción 0 o la interrupción 1 el código sería:

```
ldi R16, 0b0000 _ _ _ _      ; para configurar la int0 e int1 con niveles o flancos
out MCUCR, R16                       ; lo mandamos al MCUCR
```

Si se desea utilizar la interrupción 2 el código sería:

```
ldi R16, 0b0 _ 000000        ; Para configurar la int2 con flancos de bajada o de subida
out MCUCSR, R16              ; lo mandamos al MCUCSR
```

Por último, se requiere configurar el registro GICR, para habilitar concretamente las interrupciones deseadas, así que es necesario conocer el valor que se le tiene que cargar a GICR, para ello puede emplear la siguiente tabla:

GICR

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
			0	**0**	**0**	**0**	**0**
INT1	INT0	INT2	-	-	-	IVSEL	IVCE
0 = int. no habilitada 1 = int. habilitada	0 = int. no habilitada 1 = int. habilitada	0 = int. no habilitada 1 = int. habilitada				Interrupt vector select	Interrupt vector change enable

Por lo tanto, el código necesario para habilitar las interrupciones externas sería:

```
ldi R16, 0b _ _ _ 00000        ; se eligen las interrupciones externas que estarán habilitadas
out GICR, R16                  ; lo mandamos al GICR
```

Y de esta forma quedaría listo el programa para que, al momento de generarse una interrupción, se ejecute la rutina que hayamos especificado en los vectores de interrupción al comienzo del programa.

Timer0 (timer de 8 bits)

El Timer0 del ATmega16A consta de 8 bits, es decir que puede contar desde 0 hasta 255 pulsos de reloj. Este timer puede programarse con diferentes formas de operación:

- Normal Mode
- CTC Mode (Clear Timer on Compare Match)
- Fast PWM (Pulse Width Modulation)
- Phase Correct PWM Mode

Para configurar la forma de operación deseada, es necesario determinar el valor que se le cargará al registro TCCR0.

Registro TCCR0 – Timer/Counter Control Register

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

Con el objeto de seleccionar el modo de operación del Timer0 es necesario configurar los bits WGM01 y WGM00 (bits 3 y 6 respectivamente), para seleccionar los valores adecuados se toma como referencia la siguiente tabla:

Table 39. Waveform Generation Mode Bit Description[1]

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Mode of Operation
0	0	0	Normal
1	0	1	PWM, Phase Correct
2	1	0	CTC
3	1	1	Fast PWM

Ahora bien, los bits CS02, CS01 y CS00 normalmente se mantienen en 000 y eso provoca que el timer se encuentre apagado, es decir que no esté contando, cuando se le carga un 001 a estos bits, entonces el timer comenzará a incrementarse en uno por cada ciclo de reloj). Existen también otras posibilidades para poder configurar los bits CS02..00, de acuerdo a la tabla que se muestra a continuación.

Table 43. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

En otras palabras, si configuramos estos bits con un 011, y nuestro reloj del microprocesador está trabajando a 4MHz, entonces (1/4MHz = 0.25µs) cada ciclo de reloj tarda 0.25µs, pero como estamos configurando para clk/64, nuestro timer funcionará a 4MHz/64 = 62.5kHz es decir que ahora el timer se incrementará cada (1/62.5kHz = 16 µs) o visto de otra forma… se multiplica el tiempo que tardaría un ciclo del reloj (s) por el valor del prescaler seleccionado.

Vale la pena aclarar que normalmente el registro TCCR0 se carga hasta después de haber configurado los otros registros necesarios para el adecuado funcionamiento del timer.

TCCR0

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0		**0**	**0**				
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
	Ver tabla 39 (modo de funcionamiento)			Ver tabla 39 (modo de funcionamiento)	Ver tabla 43 (prescaler)	Ver tabla 43 (prescaler)	Ver tabla 43 (prescaler)

Para poder cargar un valor al registro TCCR0 se emplea el siguiente código:

```
Ldi r16, 0b0 _ 0 0 _ _ _ _
Out TCCR0, r16
```

Timer0 en modo normal y CTC

Es importante hacer notar que la única diferencia al momento de realizar la configuración del timer entre el modo normal y el modo CTC radica en los bits WGM00 y WGM01 que se cargan al registro TCCR0 (Ver tabla 39).

A continuación, se explicarán los puertos o registros que es necesario configurar en forma adicional para el funcionamiento del Timer0 tanto en modo normal como en modo CTC, posterior al análisis de estos registros y una vez que estos hayan sido entendidos con claridad, se explicará la diferencia exacta entre estos dos modos de funcionamiento.

REGISTRO TCNT0 – TIMER/COUNTER REGISTER

Son ocho bits, que se irán incrementando cada vez que ocurra un nuevo pulso de reloj. Este registro puede consultarse, o puede escribirse en el momento que se desee a lo largo del programa. Por ejemplo, si queremos cargarle un 0x00 al TCNT0 en algún momento el código sería:

```
Ldi r16, 0x00
Out TCNT0, r16
```

Si lo que queremos es que el contenido del TCNT0 se almacene en r16 la instrucción sería:

```
In r16, TCNT0
```

REGISTRO TIMSK – TIME/COUNTER INTERRUPT MASK REGISTER

De este registro únicamente nos interesan los 2 bits menos significativos.

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
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	

Si activamos el bit marcado como TOIE0 estamos indicando que deseamos que el Timer0 genere una interrupción cada vez que se dé un overflow, es decir cuando TCNT0 tenga un valor de 255 y entre el próximo ciclo de reloj. La interrupción que se genera viene indicada en el vector de inicialización del micro como:

```
rjmp TIM0_OVF          ; Para el manejo del Timer0 Overflow
```

Si activamos el bit marcado como OCIE0 configuramos el Timer0 para que genere una interrupción cada vez que el valor de TCNT0 y el valor del registro OCR0 sean iguales. La interrupción que se generaría en este caso viene indicada como:

```
rjmp TIM0_COMP      ; Para el manejo del Timer0 Compare
```

TIMSK

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	**0**	**0**	**0**	**0**	**0**		
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
						1 = Habilita interrupción por comparación (cuando TCNT0=OCR0)	1 = Habilita interrupción por overflow (cuando TCNT0 sea 255 y llegue el siguiente ciclo de reloj)

Supongamos el caso en que se desean activar alguna de las interrupciones, la manera de configurar el TIMSK sería:

Ldi r16, 0b000000 _ _
Out TIMSK, r16

REGISTRO OCR0 – OUTPUT COMPARE REGISTER

Este registro se emplea cuando queremos utilizar las interrupciones por comparación. Es decir, si cargamos en OCR0 un valor, por ejemplo un 75 y activamos el bit OCIE0 del TIMSK, cada vez que el valor de TCNT0 llegue a 75 se generará una interrupción del Comparador del Timer0

Para cargar un valor en OCR0, por ejemplo el 75 (decimal) el código sería

Ldi r16, 75
Out OCR0, r16

REGISTRO TIFR – TIMER/COUNTER INTERRUPTION FLAG REGISTER

Bit	7	6	5	4	3	2	1	0	
	OCF2	**TOV2**	**ICF1**	**OCF1A**	**OCF1B**	**TOV1**	**OCF0**	**TOV0**	**TIFR**
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	

En este registro se tienen las banderas que nos indican cuando una interrupción del timer ha sido activada. Normalmente no es necesario que nosotros las modifiquemos, pues automáticamente una vez que se atiende la interrupción, la bandera vuelve a su estado de 0.

Son 2 los bits que nos interesan de este registro. El bit indicado como TOV0 indica cuando se activó una interrupción debida al overflow, en cambio el bit indicado como OCF0 nos indica que se activó una interrupción debida a la comparación del contenido de OCR0 con TCNT0

Si por alguna razón deseáramos modificar el contenido de TIFR, para "quitar" una interrupción que haya sido generada, hay que escribirle UNOS a los bits de las interrupciones que queremos quitar.

TIFR

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	**0**	**0**	**0**	**0**	**0**		
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
						1 = Bandera de interrupción por comparación desactivada 0 = Bandera activa	1 = Bandera de interrupción por overlow desactivada 0 = Bandera activa

Diferencia entre modo CTC Y modo NORMAL

La principal diferencia del modo de operación CTC con respecto al modo de operación normal, consiste en que en el modo CTC cuando se ha configurado el timer para que genere una interrupción por comparación (cuando el registro contador TCNT0 llegue al valor que le cargamos a OCR0), el registro TCNT0 se limpiará en forma automática y volverá a 0x00, contrario a lo que sucede cuando el timer ha sido configurado en modo normal en donde una vez que TCNT0 alcanza el valor del OCR0, aún cuando se emplee la interrupción por comparación, el TCNT0 seguirá contando hasta llegar al overflow.

Dicho de otra forma, configurando en modo CTC, y habilitando la interrupción por comparación, podemos hacer que el microprocesador entre a la interrupción cada determinado tiempo (con exactitud), además de que el contador del Timer0 se limpia y vuelve a ser 0x00 cada vez que alcanza el valor de la comparación.

Utilizando este modo de operación (CTC), la interrupción será generada cada:

$$\text{tiempo} = \text{Periodo}_{clk}(N)(1+\text{OCRn})$$

En donde:

Tiempo = tiempo desde que se habilita el timer hasta que entra a la interrupción por comparación.

Periodo_{clk} = El inverso de la frecuencia a la cual trabaja el reloj del microprocesador (medido en s)

N = El valor con el que se ha configurado el prescaler (1, 8, 64, 256 o 1024) en los tres últimos pines del registro TCCR0.

OCRn = Es el valor con que se haya configurado el registro OCR0

O bien, si lo vemos de otra manera y configuramos la interrupción de comparación en modo CTC de forma que cada vez que entre a ella se cambie el valor lógico de uno de los pines de salida del micro, la frecuencia que generaríamos estaría dada por:

$$\text{frecuencia} = \frac{\text{frecuencia}_{clk}}{2N(1+\text{OCRn})}$$

En donde:

Frecuencia = frecuencia que tendría la señal generada por nuestro microcontrolador.

Frecuencia_{clk} = frecuencia a la que está trabajando el reloj del microcontrolador.

N = El valor con el que se ha configurado el prescaler (1, 8, 64, 256 o 1024) en los tres últimos pines del registro TCCR0.

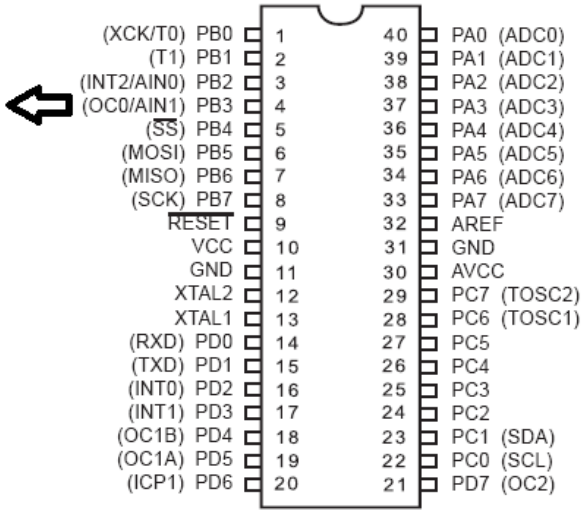
OCRn = Es el valor con que se haya configurado el registro OCR0

Si se desea que el Timer 0 quede configurado en modo normal, pero que los periodos de tiempo por interrupciones de comparación se generen en estos mismos periodos que ya se explicaron, es necesario "limpiar" manualmente el valor del TCNT0 al momento de entrar a la interrupción por comparación, sin embargo ello resulta en un gasto innecesario de ciclos de reloj, puesto que se evita al usar el modo CTC.

Por otra parte, es bueno hacer mención de que, si en alguna aplicación específica se requieren interrupciones tanto por comparación como por overflow, es necesario utilizar el modo NORMAL, pues en el modo CTC nunca se generará una interrupción por overflow.

Salida por el pin OC0 del Timer0 modo Normal o CTC

Tanto para el modo Normal como para el modo CTC, el Timer0 tiene la posibilidad de configurarse de forma de que al momento que el valor de TCNT0 sea igual que OCR0 se genere un cambio en un pin de salida OC0 (Puerto B pin 3).



Esta configuración se realiza en los bits COM01 y COM00 del registro TCCR0, de acuerdo a la siguiente tabla:

When OC0 is connected to the pin, the function of the COM01:0 bits depends on the WGM01:0 bit setting. Table 40 shows the COM01:0 bit functionality when the WGM01:0 bits are set to a normal or CTC mode (non-PWM).

Table 40. Compare Output Mode, non-PWM Mode

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on Compare Match
1	0	Clear OC0 on Compare Match
1	1	Set OC0 on Compare Match

Si COM01..00 = 00 entonces el pin OC0 (PB3) permanece desconectado y no habrá ninguna salida automática.
Si COM01..00 = 01 entonces cada vez que TCNT0=OCR0 el contenido del pin OC0 (PB3) automáticamente se invierte
Si COM01..00 = 10 entonces cada vez que TCNT0=OCR0 el contenido del pin OC0 (PB3) automáticamente se pone en 0
Si COM01..00 = 11 entonces cada vez que TCNT0=OCR0 el contenido del pin OC0 (PB3) automáticamente se pone en 1
Entonces, si se desea que haya una salida que tenga algún comportamiento especial en forma automática al momento que TCNT0 sea igual que OCR0, es necesario configurar los bits COM01 y COM00 del registro TCCR0.

TCCR0

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0							
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
	Ver tabla 39 .WGM01..00 = 10 CTC = 00 Normal	Ver tabla 40 (cuando se desee salida OC0)	Ver tabla 40 (cuando se desee salida OC0)	Ver tabla 39 .WGM01..00 = 10 CTC = 00 Normal	Ver tabla 43 (prescaler)	Ver tabla 43 (prescaler)	Ver tabla 43 (prescaler)

PWM (Pulse Width Modulation)

La modulación por ancho de pulsos (pulse width modulation) es una técnica que puede ser empleada para transmitir información a través de un canal de comunicaciones o bien para controlar la cantidad de voltaje que se desea hacer llegar a una carga (por ejemplo a un motor DC, un LED, etc.). Consiste en modificar el ciclo de trabajo de una señal periódica. (En el caso del microcontrolador, se modifica el ciclo de trabajo de una señal periódica cuadrada).

Digamos que tenemos un LED, al cual se le aplica una señal de voltaje continuo de 5 Volts, sabemos que dicho LED encenderá con intensidad máxima, si dicho voltaje lo cambiamos por 0 Volts, entonces el led permanecerá apagado (intensidad mínima). Ahora bien, si alimentamos al LED con una señal de 2.5 Volts, éste prenderá con una intensidad intermedia, proporcional al valor de voltaje que se le aplicó.

Como nosotros sabemos, el microprocesador únicamente tiene la capacidad de generar voltajes de salida de 0 Volts (que representa un 0 lógico) o de 5 Volts (que representa un 1 lógico), sin embargo, si queremos prender un LED con una intensidad media, podríamos mantener el voltaje en alto y en bajo por frecuencias cortas de tiempo, de tal forma que no se apreciara el "parpadeo" del LED, y que en promedio recibiera el voltaje deseado.

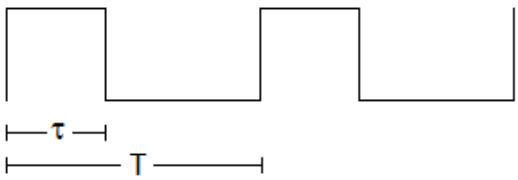
Para generar este tipo de señales resulta mucho más sencillo emplear el PWM generado mediante un Timer del microcontrolador, ya que con una sencilla configuración puede obtenerse el resultado deseado.

Ciclo de trabajo

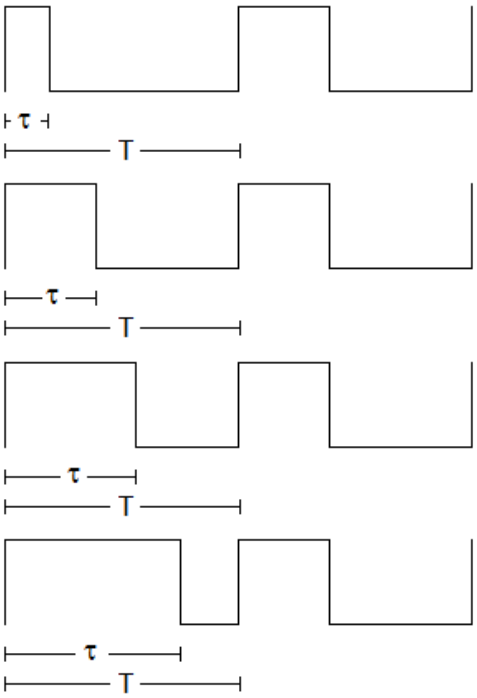
El ciclo de trabajo (duty cycle) de una señal periódica cuadrada es el tiempo que la señal se mantiene en alto en relación con el periodo de la señal.

Es decir: $$D = \frac{\tau}{T}$$

Donde D = Ciclo de trabajo
T = tiempo que la señal permanece en alto
T = periodo de la señal



A continuación, se muestran algunos ejemplos de señales de PWM, todas ellas utilizando la misma frecuencia y variando únicamente el ciclo de trabajo



Timer0 en modo FastPWM

Hemos ocupado ya el Timer0 en el modo de operación normal y CTC, ahora se explicará el funcionamiento del Timer0 en modo FastPWM

Recordemos que para que el Timer0 funcione en modo FastPWM es necesario cargar los Bits WGM01 y WGM00 del registro TCCR0 con los valores necesarios (1 y 1 respectivamente).

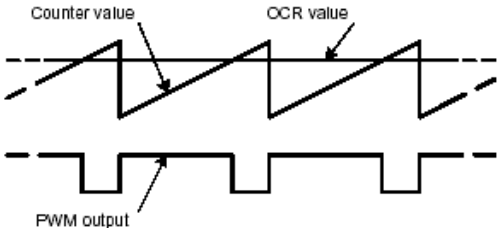
Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 39. Waveform Generation Mode Bit Description[1]

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Mode of Operation
0	0	0	Normal
1	0	1	PWM, Phase Correct
2	1	0	CTC
3	1	1	Fast PWM

Es decir que configuraríamos el TCCR0 enviándole un 0b01cc1xxx (en donde las xxx representan cualquier valor según se quiera configurar el prescaler, y cc representa el valor de los bits COM00 y COM01)

Para generar una señal de PWM el Timer0 comenzará a contar de 0 a 255 con el registro TCNT0, y producirá un cambio en la señal de salida al momento en que TCNT0 sea igual a OCR0, y también cuando TCNT0 alcance su máximo y sea reiniciada



En el diagrama anterior se muestra un ejemplo, digamos que la línea indicada como “OCR Value” equivale a un 200 (decimal), y que “Counter value” es el valor de TCNT0 que se va incrementado con cada ciclo de reloj. En el momento en que el Timer0 haya contado hasta 200, la señal de salida del PWM automáticamente bajará a 0 y permanecerá así hasta el momento en que el contador llegue a 255 y se reinicie, entonces volverá a subir a un 1 lógico… y este ciclo se repetirá constantemente.

La señal de que se genere tendrá una frecuencia que puede ser calculada mediante:

$$\text{frecuencia}_{PWM} = \frac{\text{frecuencia}_{clk}}{256 \cdot N}$$

Donde N representa el valor que se haya cargado en el prescaler.

$F_{microcontrolador}$	$T_{microcontrolador}$(s)	N	T_{pwm}(s)	F_{pwm}(Hz)
1Mhz	0.000001	1	0.000256	3906.25
		8	0.002048	488.2813
		64	0.016384	61.03516
		256	0.065536	15.25879
		1024	0.262144	3.814697
2Mhz	0.0000005	1	0.000128	7812.5
		8	0.001024	976.5625
		64	0.008192	122.0703
		256	0.032768	30.51758
		1024	0.131072	7.629395
4Mhz	0.00000025	1	0.000064	15625
		8	0.000512	1953.125
		64	0.004096	244.1406
		256	0.016384	61.03516
		1024	0.065536	15.25879
8Mhz	0.000000125	1	0.000032	31250
		8	0.000256	3906.25
		64	0.002048	488.2813
		256	0.008192	122.0703
		1024	0.032768	30.51758

Cabe hacer notar que la frecuencia del PWM dependerá únicamente de la frecuencia del reloj del microprocesador, así como del valor del prescaler cargado en el registro TCCR0. El valor que cargamos en el registro OCR0, será el que defina qué tanto tiempo permanecerá el pulso en niveles alto y bajo, y puede irse modificando durante el programa de forma que vaya variando.

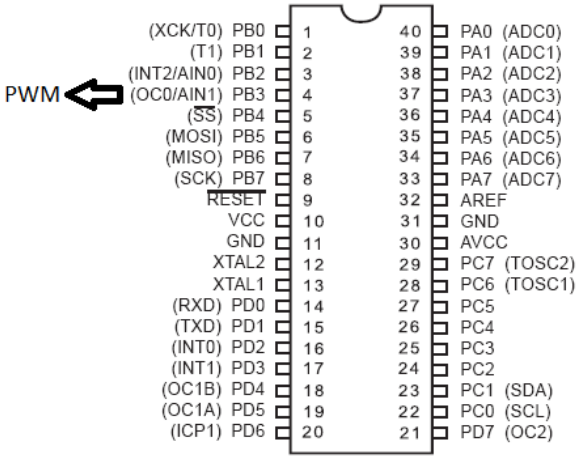
Otros bits que son importantes para la configuración del "Fast PWM" el COM01 y el COM00, los cuales se encargan de indicar si la señal que se generará será normal, o invertida.

Table 41. Compare Output Mode, Fast PWM Mode(1)

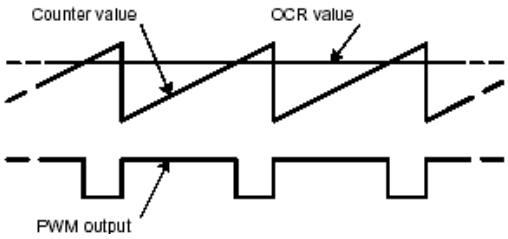
COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on Compare Match, set OC0 at TOP (Non-Inverting).
1	1	Set OC0 on Compare Match, clear OC0 at TOP (Inverting)

Cuando queremos que la señal de PWM salga por el pin OC0 del microprocesador, debemos configurar los pines COM01:0 con un valor diferente a 00, puesto que de tener 00 el pin OC0 estaría desconectado y no apreciaríamos ninguna salida.

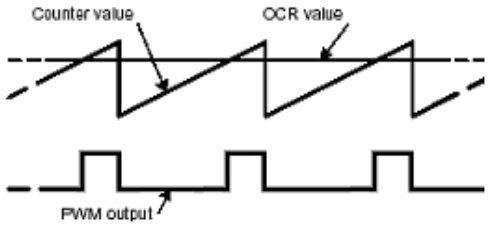
En caso de que los pines COM01 y COM00 hayan sido configurados, también es necesario habilitar el pin correspondiente a OC0 como pin de salida, es decir hay que configurar el pin3 del puerto B como salida.



Si configuramos COM01:0 con el valor 10, entonces cuando la comparación de TCNT0 con OCR0 sea igual, la señal de PWM bajará y permanecerá baja hasta que el contador TCNT0 vuelva a comenzar en 0. Esto se aprecia en el siguiente diagrama:



En cambio, si configuramos el COM01:0 con el valor 11, cuando TCNT0 coincida con el valor de OCR0, la señal de PWM cambiará a valor alto y permanecerá así hasta que el contador vuelva a comenzar en 0. La siguiente figura ilustra este funcionamiento:



TCCR0

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	**1**			**1**			
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
	Ver tabla 39 (11 en WGM01..00 es FastPWM)	Ver tabla 41 (10 o 11 para que salga señal PWM por OC0)	Ver tabla 41 (10 o 11 para que salga señal PWM por OC0)	Ver tabla 39 (11 en WGM01..00 es FastPWM)	Ver tabla 43 (prescaler)	Ver tabla 43 (prescaler)	Ver tabla 43 (prescaler)

Punteros

En el microprocesador existen algunos registros que tienen la capacidad de funcionar como palabras (una palabra se compone de 16 bits). Dichos registros tienen asignados ya los nombres X, Y y Z, y son los siguientes:

Nombre de la palabra	Corresponde a los registros
X	R26:R27
Y	R28:R29
Z	R30:R31

Si deseamos referirnos al byte alto de la palabra X podemos hacerlo mediante el nombre XH, lo cual correspondería al registro R27. En cambio, si queremos referirnos al byte bajo de la palabra X, podemos hacerlo mediante el nombre XL que sería el registro R26. A continuación, se muestra una tabla con los nombres de los registros que ya se encuentran definidos por default en el archivo m16Adef.inc

Nombre del Registro	Corresponde al registro
XH	R27
XL	R26
YH	R29
YL	R28
ZH	R31
ZL	R30

Cuando deseamos almacenar la parte alta o la parte baja de una palabra en un registro, podemos hacer uso de los comandos HIGH(palabra) y LOW(palabra), por ejemplo:

```
.EQU Direccion = RAMEND
ldi YH, HIGH(Direccion)
ldi YL, LOW(Direccion)
```

DATOS EN LA MEMORIA DEL MICROPROCESADOR

Supongamos que desea insertar una tabla con valores en la memoria del microprocesador, para posteriormente utilizarla durante su programa. Para hacer esto se cuenta con la instrucción .DB y .DW para insertar bytes o palabras en una tabla (.DB es para insertar bytes y .DW se utiliza para insertar palabras de 16 bits)
Antes de insertar una tabla de datos, es recomendable ponerle un nombre, así por ejemplo para definir una serie de valores se escribiría:

```
TABLA:
    .DB 128,131,134,137,140,144,147,150,153,156,159,162,165,168,171,174
    .DB 177,179,182,185,188,191,193,196,199,201,204,206,209,211,213,216
    .DB 218,220,222,224,226,228,230,232,234,235,237,239,240,241,243,244
    .DB 245,246,248,249,250,250,251,252,253,253,254,254,254,254,254,254
    .DB “También pueden ponerse una serie de caracteres”
```

Es muy importante hacer notar que en cada una de las líneas en las que se definen bytes deben escribirse un número par de datos, puesto que, si se escribiese un número impar, el microprocesador, al momento de compilar el código, automáticamente agregará un dato "0" al final de cada renglón.

Ahora digamos que queremos ir recuperando los datos que se guardaron en la tabla… cuando en ensamblador nosotros utilizamos la etiqueta que se le puso a nuestra tabla (TABLA) en realidad nos estamos refiriendo a la ubicación en la cual la misma fue almacenada, dicha dirección se compone de 16 bits por lo cual para almacenarla en nuestros registros, tenemos que hacer referencia a una palabra de memoria como X, Y o Z… A continuación, se muestra un ejemplo de la forma en la que se puede almacenar la dirección en ZH y ZL.

```
LEER:
        LDI ZH, HIGH(TABLA*2)
        LDI ZL, LOW(TABLA*2)

TABLA:
        .DB 128,131,134,137,140,144,147,150,153,156,159,162,165,168,171,174
        .DB 177,179,182,185,188,191,193,196,199,201,204,206,209,211,213,216
```

Con el código anterior se tiene ya almacenada la dirección de la tabla en la palabra Z, ahora solamente es necesario conocer el comando que se puede utilizar para guardar lo que hay almacenado en esa dirección de memoria en uno de los registros de nuestro microprocesador, para poder ocuparlo posteriormente. Dicho comando es LPM y lo que hace es que toma el contenido de lo que hay en la dirección de memoria que se encuentra guardada en la palabra Z, y lo guarda en el registro R0. Es importante hacer notar que al escribir la instrucción LPM no es necesario indicarle ningún operando pues siempre trabajará con el contenido de Z y R0.

```
        LPM
```

Reuniendo todo lo que se ha explicado, digamos que queremos ir leyendo uno a uno los datos de una tabla mediante una rutina que se llama leer, de forma tal que cada vez que se entre a la rutina, el siguiente dato quede almacenado en el registro R0, la rutina sería:

```
LEER:
        LDI ZH,HIGH(TABLA*2)
        LDI ZL,LOW(TABLA*2)
        ADD ZL, r17
        ADC ZH, r18
        LPM
        INC R17
RET
```

Cada vez que la rutina anterior sea ejecutada utilizando el comando RCALL LEER al regresar de la rutina en R0 se tendrán uno a uno los valores que se almacenaron en la table correspondiente. Para utilizar este código habría que asegurarnos de que al comienzo tanto R17 como R18 tendrán como valor cero. R18 solo lo empleamos para sumar el contenido de ZH con el carry que pudo haberse generado al sumar ZL con R17. Es importante destacar que si el código del ejemplo anterior se ejecutara constantemente una y otra vez serviría para leer en forma repetitiva tablas de 256 bytes, puesto que los valores de r17 irán siempre de 0 a 255.

Suponga ahora que se tiene una tabla que contiene menos de 256 datos y que quiere ser leída en forma repetitiva, un ejemplo del código que podría utilizar sería:

```
Empieza:
    LDI R16,0
    LDI R17,0
Otro:
    LDI ZH, HIGH(TABLA *2)
    LDI ZL, LOW(TABLA *2)
    ADD ZL, R16
    ADD ZH, R17
    LPM
    ;Los datos se encuentran en R0
    INC R16
    CPI R16, 5 ; # de datos que se desea leer
    BRNE Otro
    RJMP _______ ; a donde se requiera
```

Ahora bien, si la tabla tuviese más de 256 datos, sería necesario modificar el código, un ejemplo de cómo podría quedar es:

```
Empieza:
    LDI R16,0
    LDI R17,0
Otro:
    LDI ZH, HIGH(TABLA *2)
    LDI ZL, LOW(TABLA *2)
    ADD ZL, R16
    ADD ZH, R17
    LPM
    ;Los datos se encuentran en R0
    CPI R16, 0xFF
    BRNE Incr
    INC R17
Incr:
    INC R16
    ;Ahora verificamos si ya fueron el número de datos que se requieren
    CPI R17, 0x01
    BRNE Otro
    CPI R16, 0x2C
    BRNE Otro
    RJMP _______ ; a donde se requiera
```

UNIVERSIDAD PANAMERICANA

UNIVERSIDAD PANAMERICANA®
Aguascalientes

Agradecimientos

- A a todos los alumnos que han llevado mi curso de microcontroladores a lo largo ya de varios años, ustedes han sido la motivación para realizar este manual, buscando con el facilitar la adquisición de conocimientos e incentivar su capacidad de investigación para adquirir aún más. De no ser por ustedes no hubiese sido posible la realización de este manual.

- A mi mamá y a mi hija, quien siempre ha sido un gran apoyo para mi al motivarme e impulsarme para buscar siempre crecer tanto a nivel personal como profesional.

- Al Dr. Enrique Arámbula, quien fue mi profesor de microcontroladores al estudiar la carrera de Ingeniería en Electrónica y Sistemas Digitales, gracias por formar en mí el gusto por la electrónica digital y por la investigación.

- A la Universidad Panamericana, campus Aguascalientes, por haber confiado en mí desde el inicio y por brindarme la posibilidad de trabajar con ellos en esta labor que para mí resulta apasionante.

- A mis colegas en la universidad por sus consejos y amistad, gracias por su invaluable apoyo.