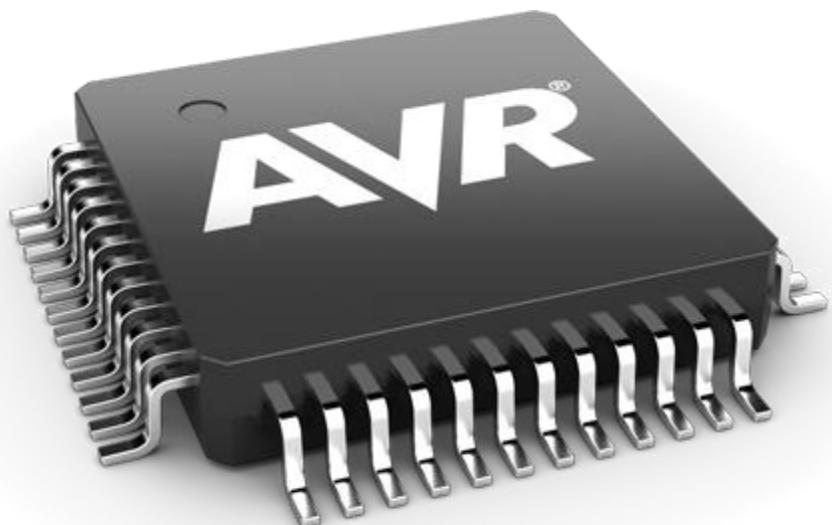


Aguascalientes, 2018

Manual de microcontroladores

MC María Teresa Orvañanos Guerrero



*“Nunca consideres el estudio
como una obligación,
sino como una oportunidad
para penetrar en el bello y
maravilloso mundo del saber.”*

Albert Einstein.

CONTENIDOS

CONTENIDOS	2
PREFACIO.....	6
AGRADECIMIENTOS	7
CONCEPTOS DE MICROCONTROLADORES.....	8
Números y operaciones.....	8
Números	8
Operaciones Aritméticas	12
Operaciones de desplazamiento	15
Estructura básica de los microprocesadores.....	17
Unidades funcionales	17
Conceptos básicos de funcionamiento.....	19
Unidad de procesamiento básica	24
Conceptos fundamentales.....	24
Ejecución completa de una instrucción	28
Ejecución completa de una instrucción de brinco.....	29
Estructura del Bus	31
Memoria	31
Posiciones y direcciones de memoria.....	31
Operaciones de memoria	33
Pilas y Colas.....	34
Códigos de condición	35
Modos de direccionamiento	35
Subrutinas	40
Organización de Entrada / Salida	40
Acceso a dispositivos de Entrada / Salida.....	40
Interrupciones	41
Interrupciones Hardware	41
Interrupciones vectorizadas	43
Anidamiento de interrupciones.....	43
Peticiones simultáneas	44
MICROCONTROLADOR ATMEL ATmega16A	45

Hardware del programador de AVR	45
Microcontrolador Atmel ATmega16A	47
Registros.....	47
Registros de acceso inmediato	48
Entrada/Salida: Puertos	49
Definiendo puertos de entrada o salida	50
Configurando resistencias de Pull-Up para puertos de entrada	50
Enviando datos a través de un puerto de salida	51
Recibiendo datos a través de un puerto de entrada.....	51
Códigos de condición – Registro de Estado	52
Instrucciones que modifican el registro de estado	53
Utilizando botones.....	54
Teclado Matricial	69
Interrupciones Externas	75
Registro MCUCR.....	76
Registro MCUCSR.....	76
Registro GICR.....	77
Registro GIFR	77
Programando las interrupciones externas	78
Fusibles del microcontrolador	89
a) Método de programación.....	91
b) Método de fusibles.....	91
Opciones para el reloj interno del microcontrolador	92
Timer0 (timer de 8 bits).....	98
Registro TCCR0 – Timer/Counter Control Register.....	98
Timer0 en modo normal y CTC.....	100
Registro TCNT0 – Timer/Counter Register	100
Registro TIMSK – Time/Counter Interrupt Mask Register.....	100
Registro OCRO – Output Compare Register.....	101
Registro TIFR – Timer/Counter Interruption Flag Register	102
DIFERENCIA ENTRE MODO CTC Y MODO NORMAL.....	102
Salida por el pin OC0 del Timer0 modo Normal o CTC	104
PWM (Pulse Width Modulation).....	115
Ciclo de trabajo.....	115
Timer0 en modo FastPWM	116
Punteros	129

Datos en la memoria del microprocesador	130
LCD (Liquid Crystal Display)	141
INSTRUCCIONES.....	142
Bandera de ocupado.....	145
Mandar instrucciones.....	145
Datos.....	146
Mandar datos	146
Inicializar el LCD	147
POSICIONES DEL LCD SEGÚN EL TIPO DE INICIALIZACIÓN	148
ADC (Analog to digital converter)	157
ADMUX – ADC Multiplexer Selection Register	157
SFIOR – Special Function IO Register	161
ADCSRA – ADC Control and Status Register A	162
ADCH:ADCL – ADC Data Register	165
EEPROM.....	175
EEARH:eearl – eeprom aDDRESS REGISTER.....	175
EEDR – EEPROM DATA REGISTER	176
EECR – EEPROM CONTROL REGISTER.....	176
PROCESO DE ESCRITURA EN LA MEMORIA EEPROM	178
PROCESO DE LECTURA EN LA MEMORIA EEPROM.....	179
Fundamentos de comunicación serial	185
Comunicación serial asíncrona	185
RS232	186
El conector DB9	186
Conexión del microcontrolador AVR al puerto serial de la computadora	187
Cable de conexión.....	188
USART (Universal Synchronous and Asynchronous serial receiver and transmitter).....	189
UDR - USART I/O Data Register	189
UCSRA – USAR Control and Status Register A	190
USCRB – USART Control and Status Register B.....	191
UCSRC – Control and Status Register C	193
USART Baud Rate Registers – UBRRH and UBRRL	195
Reloj interno del USART	199
Inicialización del USART	199
PROGRAMACIÓN EN LENGUAJE C	211
Puertos	212
Delays.....	213
Variables y Tipos de Datos.....	214

Declaración de variables.....	215
Especificadores de tipo de datos.....	216
Sentencias de bifurcación.....	217
If.....	217
if – else.....	218
if – else – if escalonada.....	219
switch.....	219
Sentencias iterativas	221
while	221
do - while	222
for	223
Operadores.....	224
Operadores aritméticos.....	224
Operadores de bits	225
Operadores relacionales.....	226
Operadores lógicos.....	226
Composición de operadores.....	227
Precedencia de operadores.....	227
Funciones.....	228
Funciones sin parámetros.....	228
Funciones con parámetros por valor.....	229
Funciones con parámetros por referencia	230
Prototipos de funciones.....	231
Variables locales y variables globales	232
Variables globales y variables locales.....	232
Variables static.....	233
Variables volatile	234
Manejo de pines en forma individual	235
Funciones de Interrupción.....	237
Control de las Interrupciones	238

PREFACIO

Este manual ha sido resultado del trabajo de preparación durante varios años de la clase de la materia de microcontroladores, cada año he ido tratando de realizar las mejoras pertinentes de forma puedas encontrar en él una guía rápida para programar los temas principales que se ven durante el curso.

En forma general se encuentran tres grandes secciones, la primera “CONCEPTOS DE MICROCONTROLADORES” corresponde a una muy breve introducción respecto a los conceptos más necesarios para entender los principios básicos de la arquitectura de un microcontrolador y de su forma de trabajar.

En la segunda sección “MICROCONTROLADOR ATmega16A” se encuentran los conceptos de funcionamiento, puertos y registros de los principales temas que se trabajan a lo largo del programa de la materia. Dentro de cada tema se incluyen algunas prácticas específicas que ayudarán a reafirmar los conocimientos que hayas adquirido. Para aprender a utilizar un microcontrolador lo más importante es la práctica pues sólo de ella se puede desprender una adecuada lógica de programación y el expertis necesario para optimizar los códigos.

En la tercera sección “PROGRAMACIÓN EN LENGUAJE C” se hace un breve resumen de la programación de microcontroladores utilizando el lenguaje C. Es importante reconocer la importancia de saber programar un microcontrolador tanto en lenguaje ensamblador como en lenguaje C, puesto que cada uno presenta sus ventajas y desventajas que, al saberlos manejar, hacen posible que elijas el más adecuado de acuerdo a la aplicación concreta que estés realizando.

Verdaderamente espero que este manual te sea de utilidad...

M.C. María Teresa Orvañanos Guerrero

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 mi el gusto por la electrónica digital y por la investigación.
- Al Dr. Ramiro Velázquez Guerrero y al Dr. Mario Acevedo Alvarado mis asesores en el Doctorado, quienes me han impulsado y animado para continuar investigando, gracias por su apoyo incondicional, por las horas invertidas en mi formación y por su paciencia en todas las situaciones.
- A la Universidad Panamericana, campus Bonaterra, por haber confiado en mi desde el inicio y por brindarme la posibilidad de trabajar con ellos en esta labor que para mi resulta apasionante.
- A mis colegas en la universidad por sus consejos y amistad, gracias por su invaluable apoyo.
- A Celina Villicaña, una excelente exalumna, quien me ha apoyado con la revisión de esta última edición, gracias por tu disponibilidad.

CONCEPTOS DE MICROCONTROLADORES

Números y operaciones

NÚMEROS

Los procesadores están formados por circuitos lógicos, los cuales pueden tomar dos valores posibles: 0 y 1, conocidos como bits.

La forma más común de representar un número es mediante una cadena de bits que se llama número binario.

REPRESENTACIÓN DE NÚMEROS BINARIOS

En los números binarios se requiere representar tanto números positivos como negativos. Para eso existen tres sistemas de representación diferentes:

- Representación en signo y magnitud
- Representación en complemento a uno
- Representación en complemento a dos

A) REPRESENTACIÓN EN SIGNO Y MAGNITUD

Cuando un número binario cualquiera es negativo únicamente es necesario cambiar el valor del bit más significativo MSB (el que se encuentra más hacia la izquierda) de 0 a 1 para representar los números negativos.

Número decimal	102	- 102	-48	-10
Binario sin signo				
Signo y magnitud				

Cabe hacer notar que cuando se emplea la representación en signo y magnitud, el bit más significativo corresponde sólo al signo y no forma parte del número en sí, es por ello que cuando se utiliza esta representación se pueden representar valores desde $-2^{n-1}+1$ hasta $2^{n-1}-1$ (donde n es el número de bits usados para la representación).

B) REPRESENTACIÓN EN COMPLEMENTO A UNO

Cuando un número es negativo y se desea representar utilizando complemento a uno, únicamente es necesario obtener el complemento de cada uno de los bits del valor positivo correspondiente.

Número decimal	102	- 102	-48	-10
Binario sin signo				
Complemento uno				

Cabe hacer notar que cuando se emplea la representación en complemento a uno se pueden representar valores desde $-2^{n-1}+1$ hasta $2^{n-1}-1$ (donde n es el número de bits usados para la representación).

C) REPRESENTACIÓN EN COMPLEMENTO A DOS

Cuando un número es negativo y se desea representar utilizando complemento a dos, es necesario calcular el complemento a 1 del número binario sin signo y posteriormente sumarle 1.

Número decimal	102	- 102	-48	-10
Binario sin signo				
Complemento dos				

Si se conoce un número negativo en complemento a dos y se desea saber cuál es su valor, éste puede determinarse sacándole su complemento y sumándole un 1.

Cabe hacer notar que cuando se emplea la representación en complemento a dos se pueden representar valores desde -2^{n-1} hasta $2^{n-1}-1$ (donde n es el número de bits usados para la representación).

El complemento a dos es el modo más eficiente para efectuar operaciones de suma y resta, entre otras cosas debido a que únicamente tiene un cero.

EJERCICIOS

Complete la siguiente tabla

b3 b2 b1 b0	Signo y magnitud	Complemento a uno	Complemento a dos
0111			
0110			
0101			
0100			
0011			
0010			
0001			
0000			
1000			
1001			
1010			
1011			
1100			
1101			
1110			
1111			

Complete las siguientes tablas con los valores correspondientes, **utilizando siempre 8 bits**

Número decimal	-115	-93	67	-200
Binario sin signo				
Signo y magnitud				
Complemento uno				
Complemento dos				

Número decimal				
Binario sin signo				
Signo y magnitud	10010011	01111111	11111011	11001101
Complemento uno				
Complemento dos				

Número decimal				
Binario sin signo				
Signo y magnitud				
Complemento uno	10010011	01111111	11111011	11001101
Complemento dos				

Número decimal				
Binario sin signo				
Signo y magnitud				
Complemento uno				
Complemento dos	10010011	01111111	11111011	11001101

OPERACIONES ARITMÉTICAS

SUMA DE NÚMEROS POSITIVOS

Cuando se suman pares de bits, se empieza siempre por los bits menos significativos LSB (los que se encuentran más a la derecha) lo acarreos se propagan a los de mayor orden.

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ y hay un acarreo de 1}$$

EJERCICIOS

Sume los siguientes números positivos:

$$\begin{array}{r}
 1010 \\
 +0001 \\
 \hline
 \end{array}
 \quad
 \begin{array}{r}
 1000 \\
 +0110 \\
 \hline
 \end{array}
 \quad
 \begin{array}{r}
 0001 \\
 +1111 \\
 \hline
 \end{array}
 \quad
 \begin{array}{r}
 1011 \\
 +0011 \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 1010111 \\
 +1001101 \\
 \hline
 \end{array}
 \quad
 \begin{array}{r}
 1010001 \\
 +0100100 \\
 \hline
 \end{array}
 \quad
 \begin{array}{r}
 0101001 \\
 +1011101 \\
 \hline
 \end{array}
 \quad
 \begin{array}{r}
 1001100 \\
 +0011011 \\
 \hline
 \end{array}$$

SUMA DE NÚMEROS CON SIGNO

Se escriben ambos números en su representación a complemento a dos

El acarreo final de la operación se debe ignorar.

$$\begin{array}{r} 0010 (+2) \\ + 0011 (+3) \\ \hline \end{array} \quad \begin{array}{r} 0100 (+4) \\ + 1010 (-6) \\ \hline \end{array} \quad \begin{array}{r} 1011 (-5) \\ + 1110 (-2) \\ \hline \end{array} \quad \begin{array}{r} 1101 (-3) \\ + 0111 (+7) \\ \hline \end{array}$$

EJERCICIOS

Sume los siguientes números en complemento a dos y verifique el resultado

$$\begin{array}{r} 0110 (6) \\ + 1111 (-1) \\ \hline \end{array} \quad \begin{array}{r} 1000 (-8) \\ + 0111 (7) \\ \hline \end{array} \quad \begin{array}{r} 0001 (1) \\ + 1001 (-7) \\ \hline \end{array} \quad \begin{array}{r} 1011 (-5) \\ + 0011 (3) \\ \hline \end{array}$$

$$\begin{array}{r} 1110 (-2) \\ + 0111 (7) \\ \hline \end{array} \quad \begin{array}{r} 0001 (1) \\ + 1111 (-1) \\ \hline \end{array} \quad \begin{array}{r} 0011 (3) \\ + 1101 (-3) \\ \hline \end{array} \quad \begin{array}{r} 1011 (-5) \\ + 1110 (-2) \\ \hline \end{array}$$

DESBORDAMIENTO

Como ya se sabe, en complemento a dos con n bits se pueden representar valores en el rango -2^{n-1} hasta $2^{n-1} - 1$. Cuando el resultado de una operación está fuera del rango representable, se dice que se ha producido un desbordamiento aritmético.

El desbordamiento solamente puede ocurrir cuando se suman dos números con el mismo signo (es decir cuando ambos números son positivos y su MSB es 0 o bien cuando ambos números son negativos y su MSB es 1)

Una manera sencilla para detectar el desbordamiento es examinar los bits más significativos. Si el bit más significativo (MSB) del resultado no es igual a los bits más significativos de los sumandos, entonces quiere decir que se produjo un desbordamiento y por lo tanto el resultado NO es correcto.

EJERCICIOS

Realice las siguientes operaciones, verifique sus resultados e indique en el recuadro correspondiente si hubo acarreo al final de la operación y si hubo desbordamiento.

$$\begin{array}{r}
 1110 \text{ (-2)} \quad \text{Acarreo} \quad \boxed{} \\
 + 1000 \text{ (-8)} \quad \text{Desbordamiento} \quad \boxed{} \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 0111 \text{ (7)} \quad \text{Acarreo} \quad \boxed{} \\
 + 1101 \text{ (-3)} \quad \text{Desbordamiento} \quad \boxed{} \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 0010 \text{ (2)} \quad \text{Acarreo} \quad \boxed{} \\
 + 0100 \text{ (4)} \quad \text{Desbordamiento} \quad \boxed{} \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 0111 \text{ (7)} \quad \text{Acarreo} \quad \boxed{} \\
 + 0111 \text{ (7)} \quad \text{Desbordamiento} \quad \boxed{} \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 1100 \text{ (-4)} \quad \text{Acarreo} \quad \boxed{} \\
 + 1001 \text{ (-7)} \quad \text{Desbordamiento} \quad \boxed{} \\
 \hline
 \end{array}$$

$$\begin{array}{r}
 0011 \text{ (3)} \quad \text{Acarreo} \quad \boxed{} \\
 + 0001 \text{ (1)} \quad \text{Desbordamiento} \quad \boxed{} \\
 \hline
 \end{array}$$

REPRESENTACIÓN DE NÚMEROS HEXADECIMALES

Los números hexadecimales normalmente se manejan como números sin signo.

Número decimal	102	250	48	10
Binario sin signo	0110 0110	1111 1010	0011 0000	0000 1010
Hexadecimal	6 6	F A	3 0	0 A

EJERCICIOS

Complete las siguientes tablas

Número decimal	24	78	5	13
Binario sin signo				
Hexadecimal				

Número decimal	180	201	255	137
Binario sin signo				
Hexadecimal				

Número decimal				
Binario sin signo				
Hexadecimal	5 F	1 8	A B	F 1

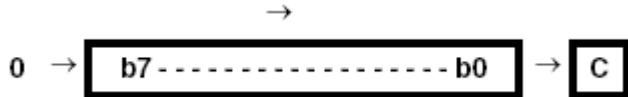
Número decimal				
Binario sin signo				
Hexadecimal	0 7	5 A	C A	B 2

OPERACIONES DE DESPLAZAMIENTO

LSR – LOGICAL SHIFT RIGHT

Corresponde a un desplazamiento lógico hacia la derecha, de forma que el bit menos significativo (LSB) pasa al bit de carry del microprocesador, mientras que todos los bits del registro se desplazan un lugar hacia la derecha y el bit más significativo (MSB) se llena con un 0.

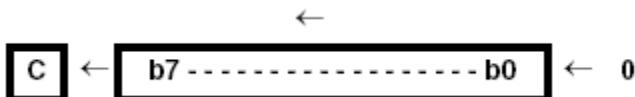
Esta operación equivale a dividir un número sin signo entre 2.



LSL – LOGICAL SHIFT LEFT

Corresponde a un desplazamiento lógico hacia la izquierda, de forma que el bit más significativo (MSB) pasa al bit de Carry del microprocesador, mientras que todos los bits del registro se desplazan un lugar hacia la izquierda y el bit menos significativo (LSB) se llena con un 0.

Esta operación equivale a multiplicar un número sin signo por 2.



ROR – ROTATE RIGHT TROUGH CARRY

Corresponde a un desplazamiento circular con acarreo a la derecha, de forma tal que todos los bits del registro se desplazan un hacia la derecha, y el bit menos significativo (LSB) pasa a ocupar el lugar del bit más significativo (MSB). Dicho bit también queda almacenado en el carry.



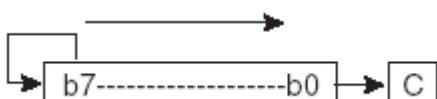
ROL – ROTATE LEFT TROUGH CARRY

Corresponde a un desplazamiento circular con acarreo a la izquierda, de forma tal que todos los bits del registro se desplazan hacia la izquierda, y el bit más significativo (MSB) pasa a ocupar el lugar del bit menos significativo (LSB). Dicho bit también queda almacenado en el carry.



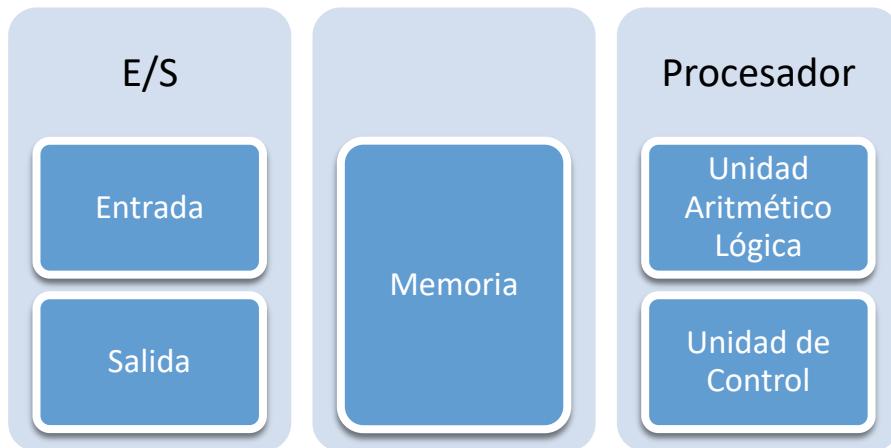
ASR – ARITHMETIC SHIFT RIGHT

Corresponde a recorrer todos los bits de un registro un lugar hacia la derecha, manteniendo siempre constante el valor del bit más significativo (MSB). El bit menos significativo LSB es almacenado en el Carry del microprocesador. Esta operación realiza una división entre dos para números con signo, sin perder el signo en el resultado. La bandera de carry puede emplearse para redondear el resultado.



Estructura básica de los microprocesadores

UNIDADES FUNCIONALES



UNIDAD DE ENTRADA / SALIDA (E/S)

La unidad de entrada acepta la información codificada que viene de operadores humanos, de dispositivos electromecánicos tales como teclados u otros dispositivos electrónicos.

La información recibida puede ser almacenada en la memoria para referencias posteriores o bien puede ser utilizada de inmediato por la circuitería de la unidad aritmética lógica (ALU) para realizar las operaciones deseadas. Los pasos del procesamiento son determinados por un programa que se encuentra almacenado en la memoria.

Para terminar, los resultados se envían al mundo exterior mediante la unidad de salida.

Todos los pasos descritos anteriormente son coordinados mediante la unidad de control.

UNIDAD DE MEMORIA

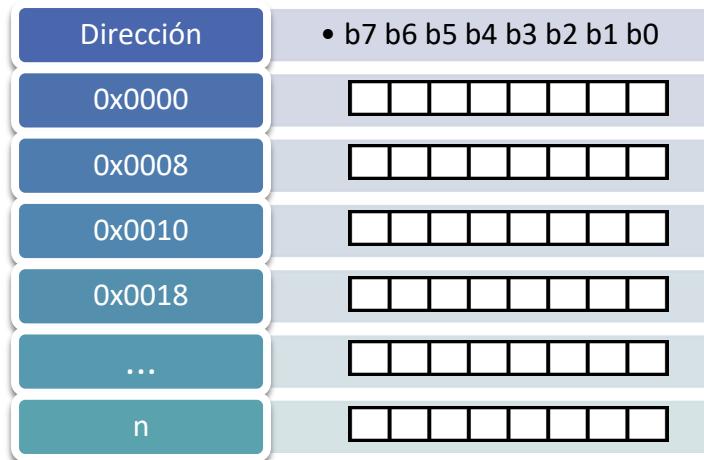
Su función es almacenar programas y datos. Existen dos clases de almacenamiento:

a) *Almacenamiento primario (Por ej. Registros, caché, memoria principal)*

Es la memoria rápida que funciona a velocidades electrónicas. Aquí se almacenan los programas mientras se ejecutan.

Una memoria tiene un gran número de celdas de almacenamiento, cada una capaz de almacenar un bit de información. Es muy raro que sea necesario leer una sola celda, casi siempre se procesan grupos fijos llamados palabras (Una palabra tiene "n" bits).

Para tener acceso rápido a una palabra, se asocia una dirección a cada una de ellas.



Al número de bits se le conoce como longitud de palabra (en el ejemplo la longitud es de 8 bits). Normalmente la longitud de palabra es entre 8 bits y 64 bits.

Algunos tipos de Almacenamiento Primario son:

- Los registros del procesador. Es el sistema más rápido de los distintos tipos de almacenamientos del microprocesador. Funcionan como "flip-flop" electrónicos.
- La memoria caché. Es un tipo especial de memoria interna usada para mejorar su eficiencia o rendimiento. Parte de la información de la memoria principal se duplica en la memoria caché. Comparada con los registros, la caché es ligeramente más lenta pero de mayor capacidad.
- La memoria principal. Contiene los programas en ejecución y los datos con que operan. Se puede transferir información muy rápidamente entre un registro del microprocesador y localizaciones del almacenamiento principal. En las computadoras modernas se usan memorias de acceso aleatorio (RAM) basadas en electrónica del estado sólido, que está directamente conectada a la CPU a través de buses de direcciones, datos y control.

b) Almacenamiento secundario (Por ej. Memorias USB, CD, DVD, Discos duros)

Es más barato, pero es mucho más lento que el almacenamiento primario, el tiempo necesario para acceder a datos en el almacenamiento primario se encuentra en el orden de los nanosegundos, mientras que en el almacenamiento secundario está en el orden de microsegundos. Sirve para almacenar gran cantidad de datos o programas a los que no se accede con tanta frecuencia. También se llama almacenamiento masivo.

UNIDAD ARITMÉTICO – LÓGICA

Ahí se ejecutan la mayoría de las opciones del microprocesador. Cuando queremos sumar dos palabras localizadas en la memoria, la unidad de control las capta y el ALU (la unidad aritmético lógica) se encarga de realizar la suma. Después de eso el resultado puede almacenarse en la memoria o enviarse a la unidad de salida.

Las unidades de control son mucho más rápidas que otros dispositivos del microprocesador, es por eso que un solo microprocesador puede controlar al mismo tiempo muchos dispositivos externos como teclados, sensores, etc.

UNIDAD DE CONTROL

Se encarga de coordinar todas las acciones del procesamiento a través de señales de temporización. Estas señales determinan cuándo debe producirse determinada acción.

CONCEPTOS BÁSICOS DE FUNCIONAMIENTO

Instrucciones:

- Regulan la transferencia de información dentro del procesador, y entre el procesador y los dispositivos de entrada y salida.
- Especifican las operaciones aritméticas y lógicas a realizar.

En la figura se muestra un diagrama más detallado de las conexiones entre la memoria y el procesador.

a) MAR (Memory Address Register)

Es el registro de dirección de memoria que contiene la dirección de la memoria a la que se desea acceder ya sea para lectura o escritura.

b) MDR (Memory Data Register)

Es el registro de datos de memoria que contiene los datos que se van a escribir en la memoria, o bien los datos que se leyeron de ella.

c) PC (Program Counter)

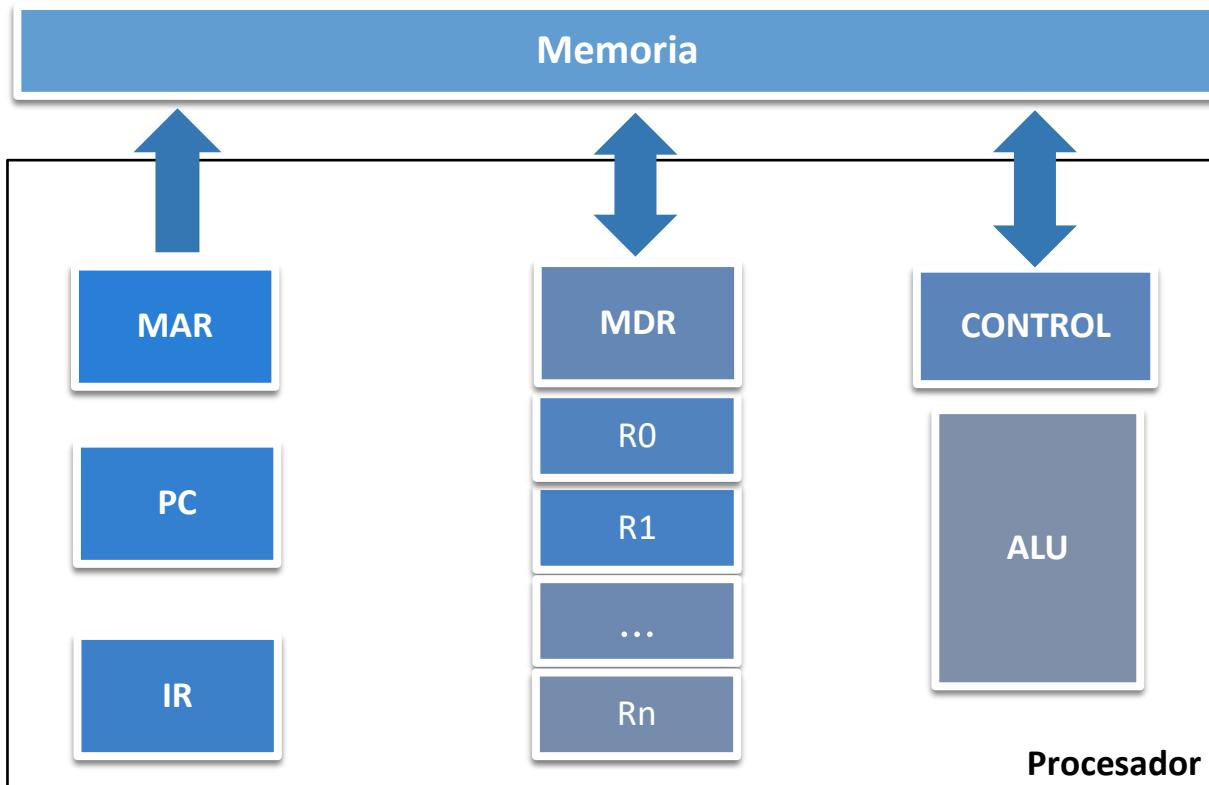
Es el contador del programa que contiene la dirección de la memoria de la siguiente instrucción a ser ejecutada. Mientras una instrucción se ejecuta, los contenidos del PC se actualizan para apuntar a la siguiente instrucción.

d) IR (Instruction Register)

Es el registro de instrucciones que contiene la instrucción que se está ejecutando en ese momento.

e) RO...Rn (Register)

Son los registros que el microprocesador puede utilizar para almacenar información.



Cuando una instrucción implica traer un dato de una dirección de memoria, primero la dirección que se quiere leer tiene que almacenarse en el registro MAR, entonces el CONTROL se encarga de enviar los pulsos necesarios para iniciar un ciclo de lectura, la memoria contesta enviando el dato que se encuentra en la dirección almacenada en MAR al registro MDR y de ahí ese dato puede transferirse a los registros de procesador o al ALU.

Supongamos que la instrucción LDS tiene el siguiente formato:

LDS DESTINO (Registro), FUENTE (Memoria)

Supongamos también que en la memoria del procesador se tiene almacenado el programa que contiene la instrucción LDS, y que a la dirección de memoria 0x1001 se le asigna la etiqueta DirA.

Si queremos guardar el dato almacenado en DirA en el registro R10 del procesador, la instrucción completa se escribiría

LDS R10, DirA

Por tanto la memoria contendrá la siguiente información:



Para ejecutar esa línea del programa los pasos que se llevarán a cabo dentro del procesador serán:

Primero es necesario que el procesador lea la línea del programa donde se encuentra la instrucción, para eso el PC carga un 0000, pues esa es la primera dirección del programa, de esa forma PC “apunta” a 0x0000

PC ← 0000

Después, como lo que se quiere es leer la memoria, es necesario cargar a MAR la dirección de memoria que se quiere leer (esa dirección es el 0x0000 que se encuentra almacenado en el PC)

MAR ← PC

Ya que se tiene la dirección en el MAR, el CONTROL se encarga de mandar los pulsos necesarios a la memoria para que se realice una lectura

READ

Entonces hay que esperar a que la memoria complete la lectura y envíe el contenido de la dirección que se leyó al MDR

RMFC (Read Memory Function Completed)

En este momento ya se tiene en MDR el contenido de la localidad de memoria 0x0000 que fue la que se leyó, es decir, en MDR se encuentra LDS R10, DirA. Ahora, para que el procesador pueda realizar esta instrucción, es necesario que la instrucción se almacene en el registro IR.

IR ← MDR

Con esto se completa la fase de lectura de la instrucción, por lo tanto, automáticamente el CONTROL se encarga de incrementar el contenido del PC, para que se quede apuntando a la localidad de memoria en donde se encuentra la siguiente instrucción, es decir a la localidad de memoria 0x0001

INC PC

Una vez que la instrucción está en el IR, la unidad de CONTROL la reconoce y por lo tanto “entiende” lo que se desea hacer: traer el dato de la memoria de la dirección DirA y guardarlo en el registro R10. Entonces, para leer la memoria, carga en MAR la dirección DirA, es decir un 0x1001

MAR ← DirA (0x1001)

Después de eso el CONTROL se encarga de mandar los pulsos necesarios para realizar la lectura de la memoria.

READ

Y hay que esperar a que se complete la lectura

RMFC

En este momento ya se tiene en el MDR el contenido de la dirección de memoria DirA, es decir, el contenido de la dirección de memoria 0x1001. Por lo tanto en MDR está el dato 0b10101010. Ahora es necesario pasar ese dato al Registro R0

R10 ← MDR

Y con esto se termina de ejecutar la instrucción.

EJERCICIOS

Suponga que la instrucción ADD realiza la suma entre el Operando1 y el Operando2 y guarda el resultado de la operación en el Operando1.

ADD Operando1, Operando2

Describa todos los pasos necesarios para ejecutar las siguientes instrucciones (en forma independiente), suponiendo que para cada caso la instrucción se encuentra almacenada en la línea 0000 y la memoria se encuentra como se muestra a continuación:

ADD R0, R1

ADD R0, DirA

ADD DirA, R0

ADD DirA, DirB



Unidad de procesamiento básica

CONCEPTOS FUNDAMENTALES

Para llevar a cabo la ejecución de un programa, el procesador toma una instrucción a la vez y realiza la operación especificada. Las instrucciones son tomadas de locaciones de memoria sucesivas hasta que se encuentra un brinco. El procesador lleva un registro de la última dirección de memoria en donde se encuentra la instrucción ejecutada usando el contador de programa. Después de ejecutar una instrucción el PC se actualiza para que apunte a la siguiente.

Otro registro importante en el procesador es el IR. Supongamos que cada instrucción emplea 4 bytes que son almacenados en una palabra de memoria. Para llevar a cabo una instrucción, el procesador lleva a cabo los siguientes pasos:

- 1) El PC apunta a la dirección de memoria. Entonces se determina que el contenido de esta dirección es una instrucción y ésta se almacena en el IR.
- 2) El contenido del PC se incrementa 4 bytes
- 3) Se lleva a cabo la instrucción especificada en el IR

En caso de que las instrucciones ocupen más de una palabra, los pasos 1 y 2 deben repetirse las veces necesarias para terminar de cargar la instrucción. Estos dos primeros pasos son conocidos como “fetch phase” y el paso 3 se conoce como “execution phase”.

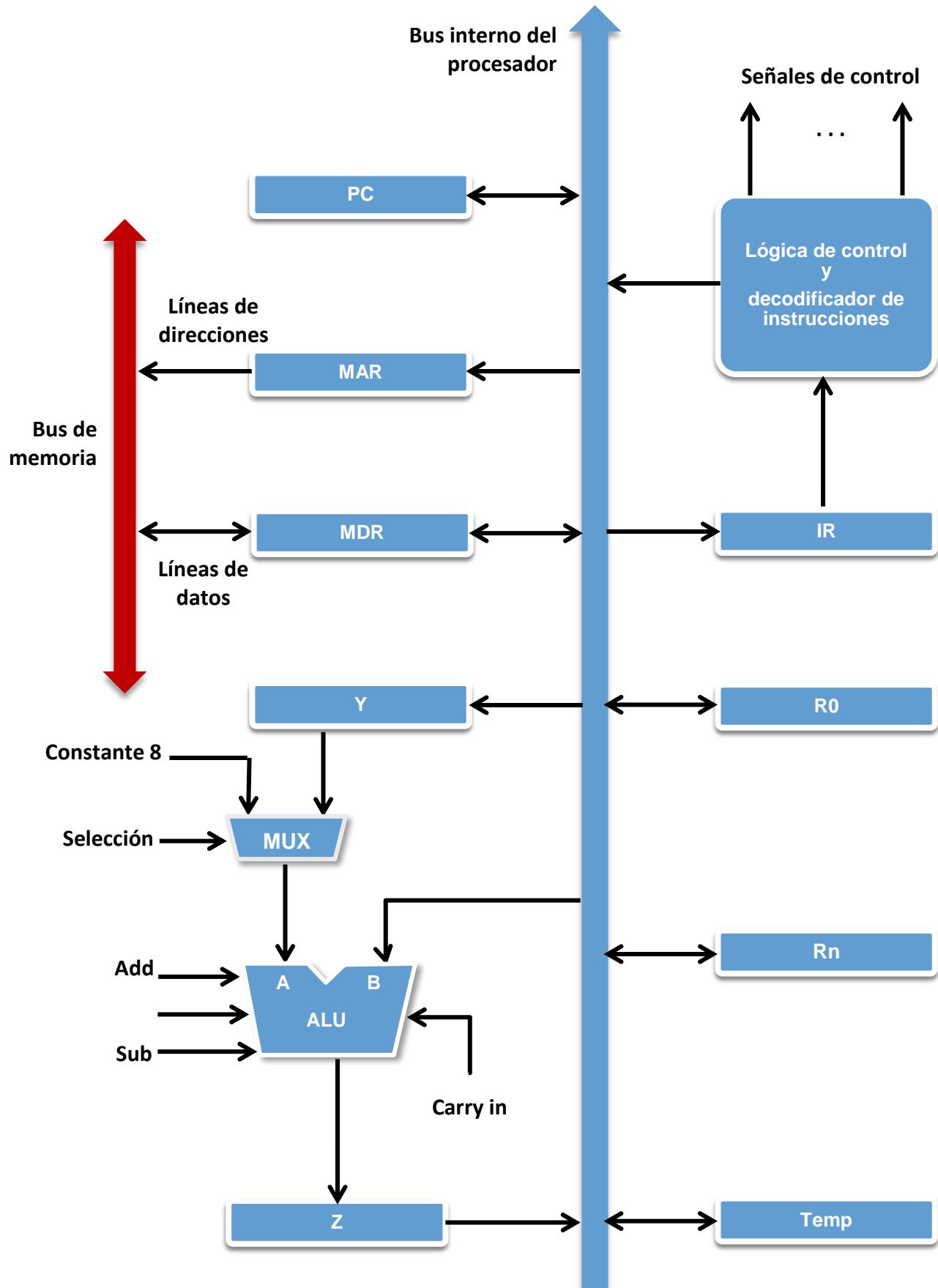
En la figura se muestra la construcción interna de un procesador en el cual, tanto el ALU como los registros, se encuentran interconectados a un bus común. Este bus se encuentra en forma interna dentro del procesador y no debe confundirse con el bus externo que conecta al procesador con los dispositivos de entrada y salida.

Los registros, el ALU y el bus interno usualmente son conocidos en conjunto como “datapath”.

Con muy pocas excepciones, una instrucción puede ser ejecutada llevando a cabo una o más de las siguientes operaciones, en algún momento determinado:

- Transferir una palabra de datos de un registro a otro, o bien al ALU.
- Llevar a cabo una operación lógica o aritmética y guardar el resultado en el procesador.
- Tomar el contenido de una locación de memoria específica y almacenarlo en un registro de procesador.
- Almacenar una palabra de datos de un registro en una localidad de memoria específica.

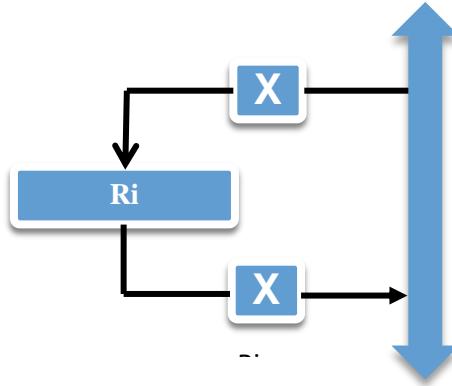
En los temas siguientes se verá en forma específica como es llevada a cabo cada una de estas operaciones utilizando un procesador simple como el mostrado en la figura.



A) TRANSFERENCIA DE REGISTROS

La ejecución de una instrucción implica una serie de pasos en los cuales los datos deben ser transferidos de un registro a otro. Para cada registro se emplean dos señales: una para transferir el dato del registro hacia bus y otra para permitir que los datos del bus entren en el registro.

Tanto la entrada como la salida de un registro R_i , se encuentran conectadas al bus por medio de switches controlados por las señales $R_{i\ in}$ y $R_{i\ out}$



Todas las operaciones y transferencia de datos son llevadas a cabo en periodos de tiempo definidos por el reloj del procesador. Puede suponerse que los registros consisten en flip-flops que se activan por flancos de subida (esta será la forma en que se maneje la explicación, sin embargo también podrían darse otros casos, como que los flip-flops se activen por flancos de bajada).

B) OPERACIONES ARITMÉTICO LÓGICAS

El ALU no cuenta con almacenamiento interno, únicamente realiza operaciones aritméticas y lógicas a dos operandos aplicados en las entradas A y B.

Para ver un ejemplo de cómo funcionaría la transferencia de registros, en conjunto con la realización de operaciones lógicas vea la figura de la siguiente página.

C) OBTENCIÓN DE UNA PALABRA DE LA MEMORIA

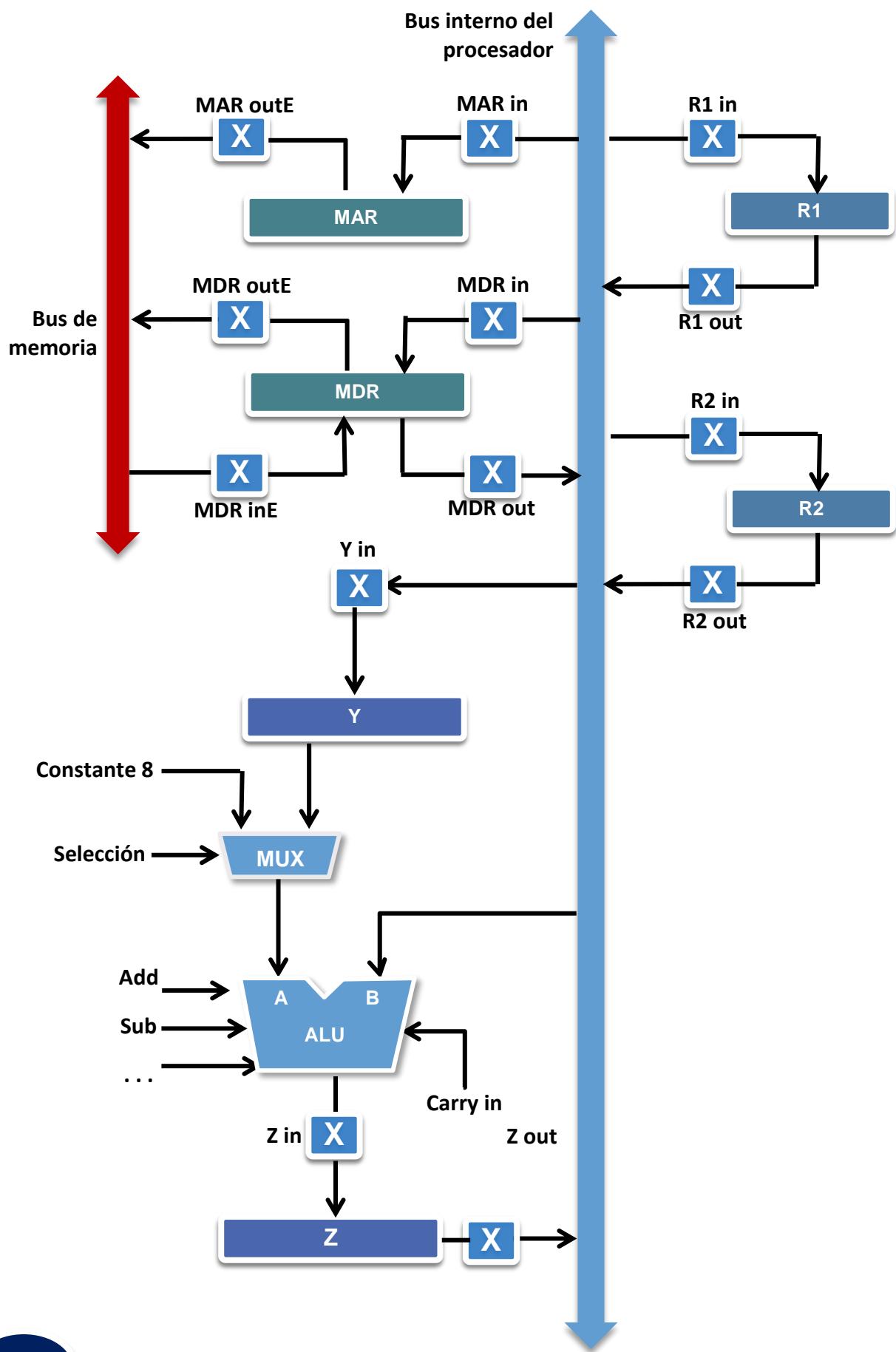
Para obtener una palabra de información de la memoria, el procesador tiene que especificar la dirección de la localidad de memoria donde la palabra se encuentra almacenada, y también tiene que indicar que desea realizar una operación de lectura.

El procesador transfiere la dirección requerida al MAR, cuya salida se encuentra conectada con líneas de direcciones del bus de memoria; al mismo tiempo el procesador utiliza las líneas de control del bus de memoria para indicar que se requiere una operación de lectura. Cuando el dato solicitado se recibe en la memoria es almacenado en el MDR, de donde puede ser transferido a otro registro en el procesador.

Las conexiones requeridas se muestran en la figura de la siguiente página.

Tanto el proceso de lectura como el de escritura de la memoria, deben encontrarse regulados por ciclos de reloj. El procesador completa una transferencia interna en un ciclo de reloj, sin embargo cuando se trata de una transferencia a dispositivos externos, la velocidad de respuesta varía dependiendo del dispositivo.

Para compensar la variabilidad en el tiempo de respuesta, el procesador espera hasta que reciba una señal en la que se le indique que la operación de lectura ha sido completada. A esta señal se le conoce como MFC (Memory Function Completed).



A manera de ejemplo digamos que se tiene la siguiente instrucción:

LD R4, X

La cual debe cargar en R4 el contenido de la dirección a la que apunta el registro X.

Después de haber ejecutado el fetch phase, los pasos necesarios para llevar a cabo esta operación son:

- 1) X_{OUT} , MAR_{IN} , READ
- 2) RMFC (Read Memory Function Completed)
- 3) MDR_{OUT} , $R4_{IN}$

Estos pueden llevarse a cabo en forma separada o bien varios a la vez. Cada uno puede completarse en un ciclo de reloj, excepto el segundo, que depende de la velocidad de respuesta de la memoria.

D) ALMACENAMIENTO DE UNA PALABRA EN MEMORIA.

Para escribir una palabra en memoria, se lleva a cabo un procedimiento muy similar al que se emplea para obtenerla de ella. El dato a ser escrito debe de ser cargado en MDR, la dirección en donde debe escribirse debe encontrarse en MAR y se envía entonces una señal de escritura.

A manera de ejemplo digamos que se tiene la siguiente instrucción:

ST X, R4

La cual debe cargar en la posición de memoria a la que apunta el registro X, el contenido del registro R4.

Después de haber ejecutado el fetch phase, los pasos necesarios para llevar a cabo esta operación son:

- 1) $R4_{OUT}$, MDR_{IN} ,
- 2) X_{OUT} , MAR_{IN} , WRITE
- 3) WMFC (Write Memory Function Completed)

Al igual que en el caso anterior, después de solicitar la operación de escritura, el procesador espera hasta recibir la señal de respuesta indicando que se ha finalizado el proceso de escritura.

EJECUCIÓN COMPLETA DE UNA INSTRUCCIÓN

Ahora se unirán todos los elementos necesarios para ejecutar una instrucción.

Considere por ejemplo la instrucción: LD R15, X (En donde X es un registro del microprocesador que contiene una dirección de memoria). Esta instrucción se encarga de copiar el contenido de la localidad de memoria a la que apunta X al registro R15. Para llevar a cabo esta instrucción se requieren las siguientes operaciones:

1. Obtener la instrucción
2. Obtener el contenido de la localidad de memoria a la que apunta X
3. Almacenar ese contenido en el registro R15.

Los pasos en forma detallada son:

Paso	Operación
1	PC_{out} , MAR_{in} , SELECT8, ADD, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , READ, RMFC
3	MDR_{out} , IR_{in}
4	X_{out} , MAR_{in} , READ, RMFC
5	MDR_{out} , $R15_{in}$, END

En el paso 1 se inicia la obtención de la instrucción, cargando el contenido del PC en el MAR y enviando una instrucción de lectura a la memoria. Mientras tanto, la señal de SELECT es puesta en SELECT8, lo cual causa que el multiplexor deje pasar la constante 8. Ese valor es sumado al operando que se encuentra en B (que es el contenido del PC). El resultado queda momentáneamente en el registro Z, entonces ese valor es direccionado nuevamente para almacenarse en PC. Todo esto se hace mientras que se aprovecha para esperar que la memoria mande el pulso de respuesta de que ha terminado el proceso de lectura.

En el tercer paso, la palabra obtenida de la dirección de memoria a la que apuntaba PC, es almacenada en IR, de forma que en este punto se conoce la instrucción que se desea ejecutar. Del paso 1 al 3, se lleva a cabo lo que se conoce como “fetch phase”, y siempre se lleva a cabo de la misma forma, sin importar la instrucción (pues precisamente estos tres pasos sirven para que el microprocesador pueda cargar dicha instrucción en el IR y así reconocerla).

A partir del paso 4, los circuitos de control se encargan de interpretar el contenido del IR. Para la instrucción que se propuso en el ejemplo, el microprocesador necesita leer el contenido de la dirección a la que apunta el registro X, es por ello que lo primero que hace es transferir el contenido de X a MAR y mandar un pulso de lectura. Después deberá esperar hasta que la memoria termine con la lectura e indique que el dato ya se encuentra en MDR. Entonces se deja pasar el dato de MDR a R15. Después de esto se indica que se ha terminado el ciclo y por lo tanto se puede proceder a realizar otra instrucción comenzando nuevamente con el paso uno.

En esta instrucción se utilizaron todas las señales que se muestran en la tabla de instrucciones, con excepción de Y_{in} en el paso 2, pues no fue necesario almacenar el contenido del PC en el registro Y al ejecutar la instrucción, sin embargo, esto será útil cuando se desean realizar instrucciones de brinco.

EJECUCIÓN COMPLETA DE UNA INSTRUCCIÓN DE BRINCO

Las operaciones de salto se encargan de cargar un valor nuevo al contenido del Program Counter (PC).

Ejemplo RJMP Etiqueta (RELATIVE JUMP)

Hay instrucciones de salto condicional que únicamente provocan un salto cuando se satisface determinada condición.

Ejemplo	BREQ	Etiqueta	(BRANCH IF EQUAL)
	BRNE	Etiqueta	(BRANCH IF NOT EQUAL)

La instrucción RJMP reemplaza el contenido del PC por la dirección a dónde se desea “brincar”. Esta dirección usualmente se obtiene sumándole al PC una cantidad X que es indicada por la instrucción. A continuación, se puede ver la secuencia que se encarga de implementar la instrucción RJMP (que es un brinco incondicional).

Paso	Operación
1	$PC_{out}, MAR_{in}, SELECT8, ADD, Z_{in}$
2	$Z_{out}, PC_{in}, Y_{in}, READ, RMFC$
3	MDR_{out}, IR_{in}
4	$Offset-de-IR_{out}, ADD, Z_{in}$
5	Z_{out}, PC_{in}, END

El proceso inicia con el fetch phase que es el mismo para cualquier instrucción (pasos del 1 al 3).

En el paso 4, el valor de X que se le agrega al PC (valor de Offset) es extraído del IR. Recordemos que el valor del PC se encuentra entonces en el registro Y, y el valor de Offset se encuentra en el bus y entonces se suma con PC. El resultado queda temporalmente almacenado en la variable Z.

En el quinto paso el valor de Z es enviado al PC y entonces se envía la señal de END para indicar que se puede continuar con la siguiente instrucción.

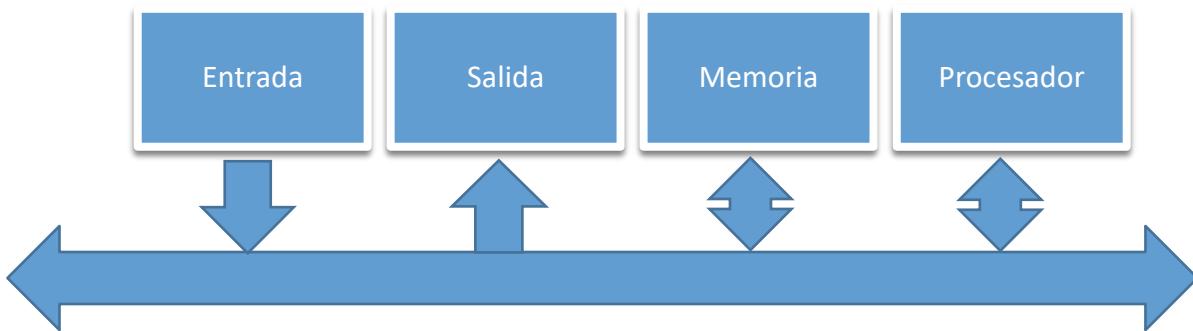
A manera de ejemplo digamos que, si la instrucción de RJMP se encuentra en la localidad de memoria 0x1000 y el brinco debe ir a la dirección 0x1200, el valor del offset será de 0x200.

Estructura del Bus

Ya al momento de explicar los conceptos básicos del funcionamiento, se observó que las unidades funcionales del procesador, deben estar conectadas de algún modo organizado.

Para alcanzar velocidades de procesamiento razonables, el procesador debe estar organizado de forma tal que sus unidades puedan gestionar una palabra completa de datos (n bits) en un momento dado.

Cuando una palabra de datos se transfiere, todos los bits se transfieren en paralelo (es decir, todos los bits se transfieren al mismo tiempo) por lo tanto se emplea una línea para cada uno de los bits. Al grupo de líneas que sirve como conexión entre los dispositivos se le llama bus. Un bus debe tener líneas de datos, líneas de direcciones y líneas de control.



Algunos dispositivos conectados al bus son más rápidos (Discos, memorias) y hay algunos otros más lentos (teclados, impresoras, etc.)

Para que un dispositivo lento no ocupe el bus por largos períodos de tiempo, es típico incluir registros de buffer que se encargan de almacenar la información durante la transferencia.

Por ejemplo, si un procesador tiene que enviar información a la impresora (que es lenta), entonces la envía por el bus al buffer de la impresora (que es un registro electrónico y por tanto puede funcionar a velocidad rápida) y ahí se almacena. Una vez que el buffer de la impresora se ha cargado, la impresora puede comenzar a trabajar, sin que el procesador intervenga más y sin que el Bus se encuentre ocupado.

Memoria

POSICIONES Y DIRECCIONES DE MEMORIA

Cada celda de memoria almacena un bit de información

Una palabra es un grupo de n bits (normalmente entre 8 y 64)

Un byte corresponde exactamente a 8 bits o bien a dos valores hexadecimales

Una instrucción puede requerir más de una palabra para almacenarse

Para tener acceso a una palabra o byte se requiere que la memoria tenga direcciones para cada posición posible.

DIRECCIONAMIENTO DE BYTES EN PALABRAS

Supongamos que se desea almacenar el siguiente valor (una palabra de cuatro bytes)

110010101111110101110101011110

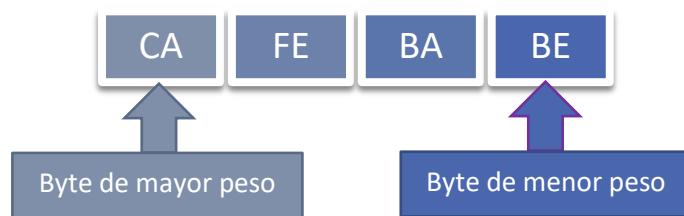
O lo que es lo mismo en forma hexadecimal:

CAFEBABE

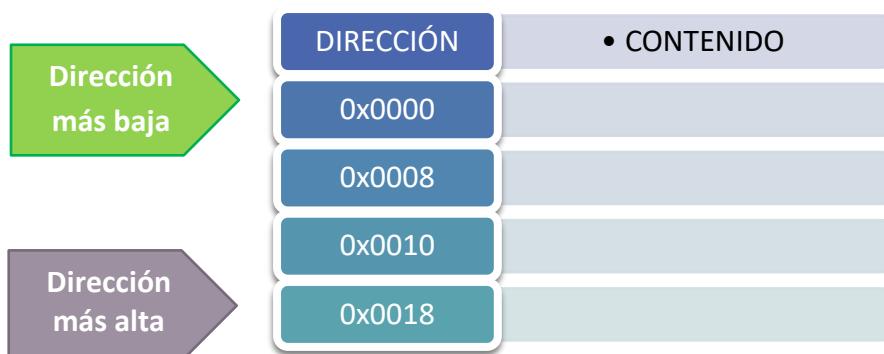
Separándolo por bytes se tiene

11001010 11111110 10111010 10111110

O bien



Y que se tiene el siguiente espacio de memoria para almacenar dichos bytes



Existen dos formas de almacenar los bytes en las direcciones de memoria:

- a) Big Endian
- b) Little Endian

A) BIG ENDIAN

Significa que el byte de mayor peso se almacena en la dirección más baja de memoria y el byte de menor peso en la dirección más alta.

DIRECCIÓN	• CONTENIDO
0x0000	• CA
0x0008	• FE
0x0010	• BA
0x0018	• BE

B) LITTLE ENDIAN

Significa que el byte de menor peso se almacena en la dirección más baja de memoria y el byte de mayor peso en la dirección más alta.

DIRECCIÓN	• CONTENIDO
0x0000	• BE
0x0008	• BA
0x0010	• FE
0x0018	• CA

OPERACIONES DE MEMORIA

Hay dos operaciones básicas que involucran la memoria y el procesador

- Carga o lectura
- Almacenamiento o escritura
-

A) CARGA O LECTURA

Esta operación copia el contenido de una dirección de memoria al procesador. Los datos de la memoria permanecen inalterados.

B) ALMACENAMIENTO O ESCRITURA

Esta operación transfiere la información del procesador a la memoria.

PILAS Y COLAS

Los datos con los que opera un programa se pueden organizar de distintas maneras:

- a) Pilas
- b) Colas

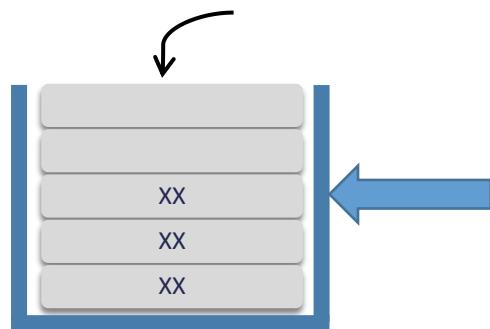
a) Pilas

Es una lista de elementos en donde éstos sólo pueden añadirse o eliminarse por uno de los extremos de la lista.

Se describen con la frase “El último que entra es el primero que sale” – LIFO (last in, first out).

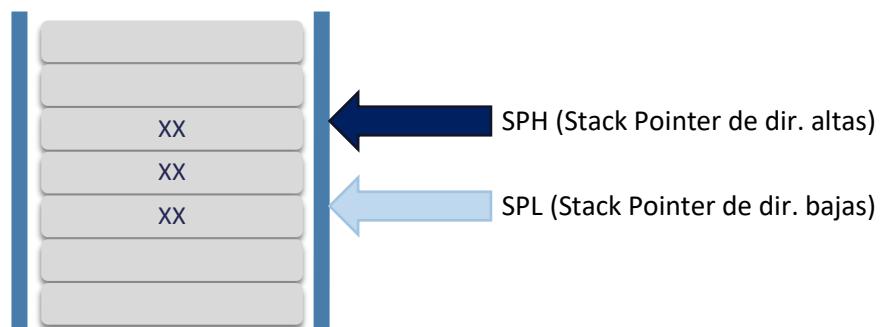
Se emplean los términos Push (insertar) y Pop (extraer) para describir la introducción de los nuevos elementos o la extracción (respectivamente)

Se emplea un puntero de pila, que es un registro del procesador que contiene la dirección del elemento que se encuentra en el tope de la pila en cada momento.



b) Colas

Los datos se almacenan y se recuperan según el principio “El primero que entra es el primero que sale” – FIFO (First in, first out).



Diferencias en la implementación de una pila y una cola

PILA	COLA
Uno de los extremos está fijo mientras el otro crece o decrece.	Ambos extremos se mueven hacia los dos extremos.
Sólo requiere un puntero SP para apuntar al tope de la pila.	Se requiere tener dos punteros para hacer referencia a los dos extremos de la cola.

Otra diferencia es que, sin algún tipo de control la cola se movería a través de la memoria. Una forma de limitar la cola a una región fija de memoria es utilizando un buffer circular.

Códigos de condición

La información sobre el resultado de diversas operaciones es guardada por el procesador para su uso en posteriores instrucciones de salto condicional. Esta información se guarda en bits individuales que se conocen como indicadores de código de condición o banderas y que se agrupan en un registro especial llamado Registro de Estado.

Normalmente se manejan por lo menos cuatro banderas:

N (NEGATIVO) – Su valor es 1 cuando el resultado de la operación es negativo y 0 cuando es positivo

Z (ZERO) – Su valor es 1 cuando el resultado de la operación es cero y 0 cuando es cualquier otro valor.

C (CARRY) – Su valor es 1 cuando se produce un acarreo durante una operación aritmética.

V (DESBORDAMIENTO) – Su valor es 1 cuando ocurre un desbordamiento en una operación aritmética.

Modos de direccionamiento

Un programa opera con datos que residen en la memoria del procesador.

Se llama “modo de direccionamiento” a las diferentes formas en que la dirección de memoria de un operando puede ser especificado en una instrucción

OPERACIÓN Destino, Fuente

A) DIRECCIONAMIENTO MODO INMEDIATO

Sintaxis en ensamblador: VALOR

Dirección efectiva: El Operando = VALOR

El operando se proporciona en forma explícita en la instrucción, puede ser en forma decimal (o en binario si se escribe Ob antes del número, o en hexadecimal si se escribe 0x antes del número)

Ejemplos: LDI R10, 200 (LOAD IMMEDIATE)
CPI R10, 0Xff (COMPARE WITH IMMEDIATE)
ORI R10, 0b11110000 (LOGICAL OR WITH IMMEDIATE)

B) DIRECCIONAMIENTO MODO REGISTRO

Sintaxis en ensamblador: Ri

Dirección efectiva: Ri

El operando corresponde al contenido del registro del procesador, el nombre o dirección del registro se proporciona en la instrucción.

Ejemplos: CP R10, R11 (COMPARE)
MOV R10, R11 (COPY)

C) DIRECCIONAMIENTO MODO ABSOLUTO (MODO DIRECTO)

Sintaxis en ensamblador: DIRECCIÓN

Dirección efectiva: DIRECCIÓN

También se conoce como modo directo, el operando se encuentra en la dirección de memoria, la dirección se proporciona explícitamente en la instrucción.

Ejemplos: LDS R2, 0XFF00 (LOAD DIRECT FROM DATA SPACE)
LDS R10, 0x0FOF (LOAD DIRECT FROM DATA SPACE)

D) DIRECCIONAMIENTO MODO INDIRECTO

Sintaxis en ensamblador: X, Y, Z

Dirección efectiva: lo que hay en la localidad de memoria a la que apunta X, Y o Z

La dirección efectiva se encuentra en la posición de memoria cuya dirección se encuentra almacenada en la palabra X, Y o Z

Ejemplos: LD R31, X (LOAD INDIRECT FROM DATA SPACE TO REGISTER USING INDEX X)
ST Y, R1 (STORE INDIRECT FROM REGISTER TO DATA SPACE USING INDEX Y)

Analizando el primer ejemplo LD R31, X. La instrucción LOAD se encarga de copiar el contenido de la localidad de memoria a la que apunta X al registro R31, así pues, al final de la operación el Registro R31 tendrá guardado el dato al que apunta la dirección de memoria almacenada en X (010A), es decir tendrá un FF

DIRECCIÓN	• CONTENIDO
0000	• LD R1, X
...	
010A	• FF
...	

REGISTRO	• CONTENIDO
Byte bajo de X	
R26	• 0A
Byte alto de X	
R27	• 01
R28	
R29	
R30	
R31	• aquí quedará el dato FF

E) DIRECCIONAMIENTO MODO INDEXADO

Sintaxis en ensamblador: X+q, Y+q, Z+q

Dirección efectiva: La dirección a la que apunta el registro Y + q

La dirección efectiva del operando corresponde a la dirección a la que apunta el registro Y más el valor constante q que se conoce como Offset.

STD Y+2, R26 (STORE INDIRECT WITH DISPLACEMENT FROM REGISTER TO DATA SPACE USING INDEX Y)

La instrucción STD que se muestra en el ejemplo sumará el contenido del registro Y, que es 0102 más la constante q, que en este caso es 2. Entonces $0102+2=0104$. Y almacenará el contenido de R26 (es decir un 00) en la dirección de memoria a la que apunta Y+2, es decir en la dirección de memoria 0104.

DIRECCIÓN	• CONTENIDO
0000	• LD R1, X
...	
0104	• aquí quedará dato el 00
...	

REGISTRO	• CONTENIDO
R26	• 00
R27	
R28	• 02
R29	• 01
R30	
R31	

Byte bajo de Y

Byte alto de Y

F) DIRECCIONAMIENTO MODO AUTOINCREMENTO

Sintaxis en ensamblador: X+, Y+, Z+

Dirección efectiva: La dirección a la que apunta el contenido del registro X, Y o Z.

La dirección efectiva del operando corresponde a la dirección a la que apunta el registro X, Y o Z, después de realizar la operación incrementa el contenido del registro X, Y o Z de forma que $X = X + 1$ o $Y = Y + 1$ o $Z = Z + 1$.

Ejemplo: LD R4, X+ (LOAD INDIRECT FROM DATASPACE TO REGISTER USING INDEX X)
 ST Y+, R5 (STORE INDIRECT FROM REGISTER TO DATASPACE USING INDEX Y)

G) DIRECCIONAMIENTO MODO AUTODECREMENTO

Sintaxis en ensamblador: -X, -Y, -Z

Dirección efectiva: La dirección a la que apunta el contenido del registro $X - 1$, $Y - 1$ o $Z - 1$.

Este tipo de direccionamiento decrementa el contenido del registro X, Y o Z, de forma que $X = X - 1$ o $Y = Y - 1$ o $Z = Z - 1$, y la dirección efectiva del operando corresponde a la dirección a la que apunta el registro X, Y o Z ya decrementado.

Ejemplo: LD R4, -X (LOAD INDIRECT FROM DATASPACE TO REGISTER USING INDEX X)
 ST -Y, R5 (STORE INDIRECT FROM REGISTER TO DATASPACE USING INDEX Y)

EJERCICIOS

En las siguientes instrucciones, en cada operando indica el modo de direccionamiento empleado y explica cuál será el resultado de llevar a cabo dicha instrucción (usando los datos proporcionados, y suponiendo que cada operación es independiente de las demás).

DIRECCIÓN	• CONTENIDO
0x0100	• 0x15
0x0108	• 0x31
0x0110	• 0x89
0x0118	• 0x10
0x0120	• 0x43
0x0128	• 0x32

REGISTRO	• CONTENIDO
R26	• 0X18
R27	• 0x01
R28	• 0x00
R29	• 0x01
R30	• 0x20
R31	• 0x01

INSTRUCCIÓN	OPERANDO 1	OPERANDO 2	¿QUÉ CAMBIA AL FINAL DE LA INSTRUCCIÓN?
MOV R31, R30			
LD R31, X			
LDS R30, 0x0128			
ST -X, R26			
STD X+8, R31			
CPI R30, 0x43			
LD R29, X+			

Subrutinas

En los programas a menudo es necesario realizar una sub tarea que se tenga que repetir en varias ocasiones, sin embargo, si esa sub tarea se programa cada vez que se requiere usar, se ocuparían muchos lugares de memoria. Es por ello que, para ahorrar espacio, se guarda solamente una copia de ese conjunto de instrucciones que se repiten constantemente, y de esta forma se les manda a llamar cada vez que sea necesario. A este conjunto de instrucciones se le llama subrutina. Tras la ejecución de una subrutina el programa que lo llamó debe regresar al punto en el que se había quedado.

Para llamar a una subrutina se emplea la instrucción CALL o RCALL, la cual realiza las siguientes operaciones

- Almacena el contenido del Program Counter (PC) en un registro temporal
- Salta a la dirección de la subrutina

Para que la subrutina termine y el programa continúe ejecutándose se emplea la instrucción RETURN o RET, la cual realiza la siguiente operación.

- Le devuelve al PC el valor que se había almacenado en un registro temporal cuando se había ejecutado la instrucción CALL.

Organización de Entrada / Salida

ACCESO A DISPOSITIVOS DE ENTRADA / SALIDA

Usualmente a cada dispositivo de Entrada / Salida se le asigna un conjunto único de direcciones. Cuando el procesador introduce una dirección en la línea de direcciones del bus, el dispositivo que reconoce como propia dicha dirección responde en las líneas de control. Entonces el procesador solicita una operación de lectura o escritura y los datos son transferidos a través de la línea de datos.

Con una Entrada / Salida asignada a memoria, cualquier instrucción que puede acceder a la memoria, puede también transferir datos desde o hacia el dispositivo de Entrada / Salida.

Por ejemplo:

Uno de los puertos de salida del microprocesador tiene asignada la dirección de memoria 0x18 (es decir la dirección 18 en hexadecimal). Por lo tanto, si requiere enviar al puerto una señal que se encargue de prender los LEDs que se encuentran conectados en él, esto se puede hacer directamente mediante la instrucción

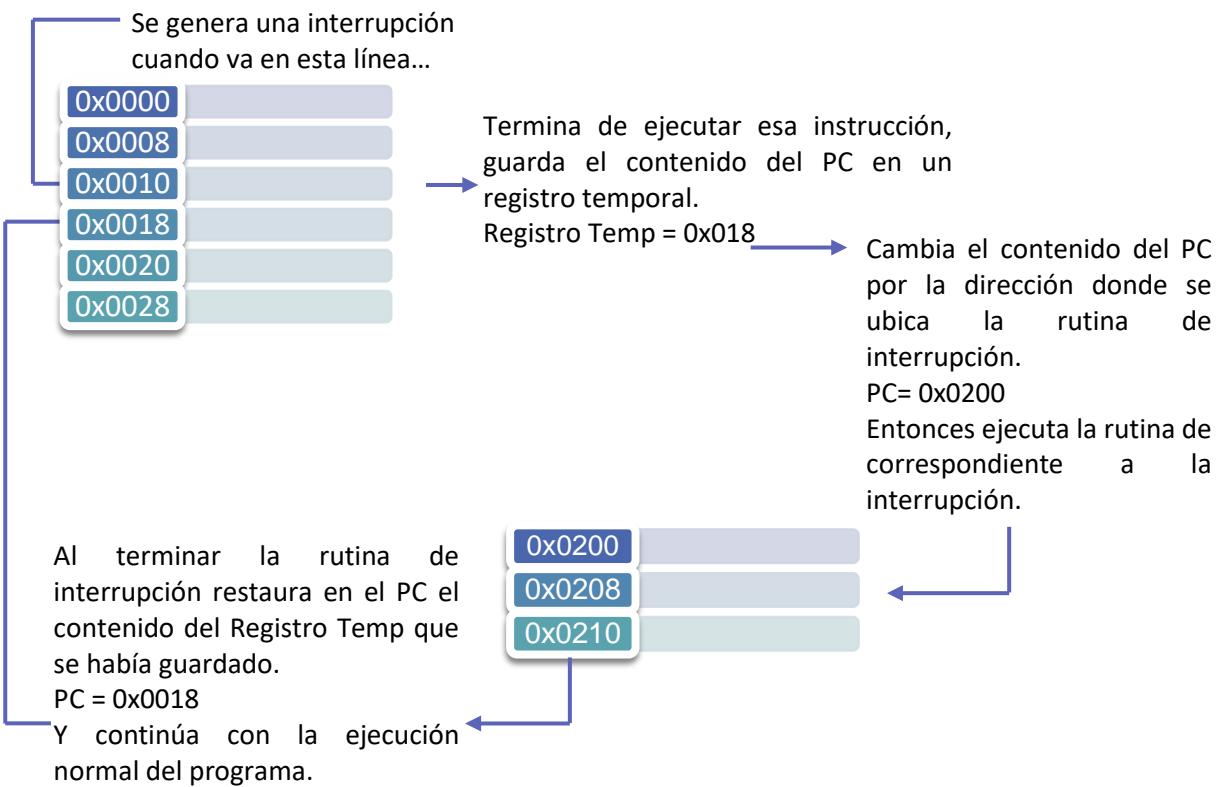
OUT 0x18, R16 *;Que sacaría por el puerto el contenido del registro R16*

Interrupciones

Existen situaciones en las cuales es necesario que el programa deje de hacer la tarea que está realizando, para darle lugar a alguna otra un poco más urgente. Esto lo puede realizar mediante el envío de una señal de hardware al procesador denominada interrupción. Normalmente se dedica una de las líneas del bus de control, llamada “línea de petición de interrupción”, para este propósito.

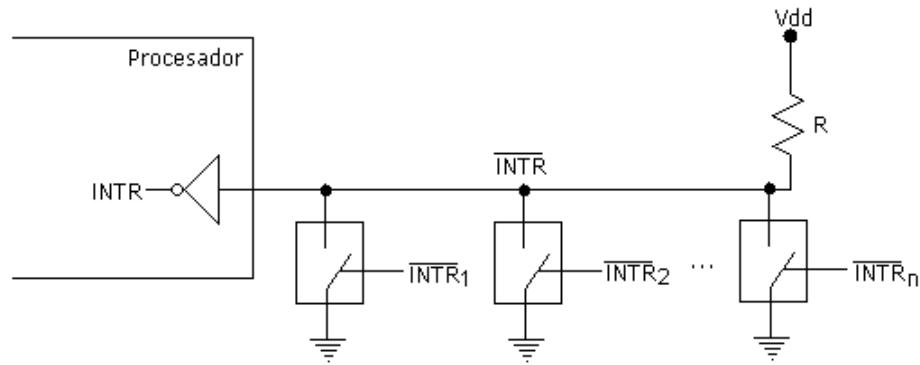
Cuando se ejecuta una interrupción, se realiza un proceso muy parecido a cuando se realiza una subrutina, pero la diferencia está en que una subrutina se realiza cuando el programa la llama, sin embargo, una rutina de interrupción se puede ejecutar en cualquier momento en que se recibe la interrupción, sin importar en qué parte del programa se encuentre en ese momento.

Es importante hacer notar que en procesador ATMEL con el que se trabajará durante el curso, al mandar a llamar una interrupción, los contenidos de los registros no se respaldan, por lo cual, si dentro de la interrupción se modifica el contenido de éstos, el cambio prevalecerá al regresar a ejecutar comúnmente el programa.



INTERRUPCIONES HARDWARE

Habrá ocasiones en las que se desee que múltiples dispositivos tengan la posibilidad de activar una misma interrupción. En la figura se ilustra la manera en la que dichos dispositivos pueden ser conectados.



Para pedir una interrupción el dispositivo cierra la línea correspondiente, así si todos los dispositivos tienen sus interrupciones desactivadas, la línea INTR tendrá Vdd (1 lógico) y el estado de INTR en el procesador será 0 lógico, pero si una de las interrupciones de los dispositivos se activa, entonces el circuito se cerrará, la línea INTR tendrá tierra (0 lógico) y el procesador tendrá un 1 lógico que le indicará que una interrupción ha sido activada.

HABILITANDO Y DESHABILITANDO INTERRUPCIONES

Un programador debe tener la capacidad de decidir en todo momento si quiere que las interrupciones detengan la ejecución del programa para llevar a cabo la rutina de interrupción, o si no desea reconocerlas en algún momento. Una propiedad importante que tienen los procesadores es la capacidad de habilitar y deshabilitar las interrupciones en la forma que se deseé.

Consideremos un caso cuando se envía una interrupción desde un dispositivo, entonces se activará la señal de interrupción y se entrará a la rutina correspondiente, mientras tanto el dispositivo mantendrá activa la señal de interrupción hasta que observe que el procesador la atiende, es decir que la señal estará activa mientras se ejecuta la interrupción. Es esencial asegurar que esta señal de petición activa no conlleva sucesivas interrupciones, causando que el sistema entre en un bucle infinito del cual no podría salir, para ello se emplean diversos mecanismos, a continuación describiremos tres de ellos:

- Que el hardware del procesador ignore la línea de petición de interrupción hasta que se haya terminado de ejecutar la primera línea de la rutina de interrupción, después, mediante la instrucción de deshabilitar las interrupciones (en la primera línea) se asegura que no se cicle el programa. Generalmente, si se usa este método, la última instrucción de la rutina de interrupción será volver a habilitar las interrupciones.
- Que el procesador automáticamente deshabilite las interrupciones antes de comenzar una rutina de interrupción, y vuelva a habilitarlas automáticamente cuando termine de ejecutar dicha rutina.
- Que los circuitos que detectan la interrupción respondan solo por flanco, es decir que sean “disparados por flanco”, en este caso el procesador recibirá una única petición independientemente de cuánto tiempo esté activa la línea.

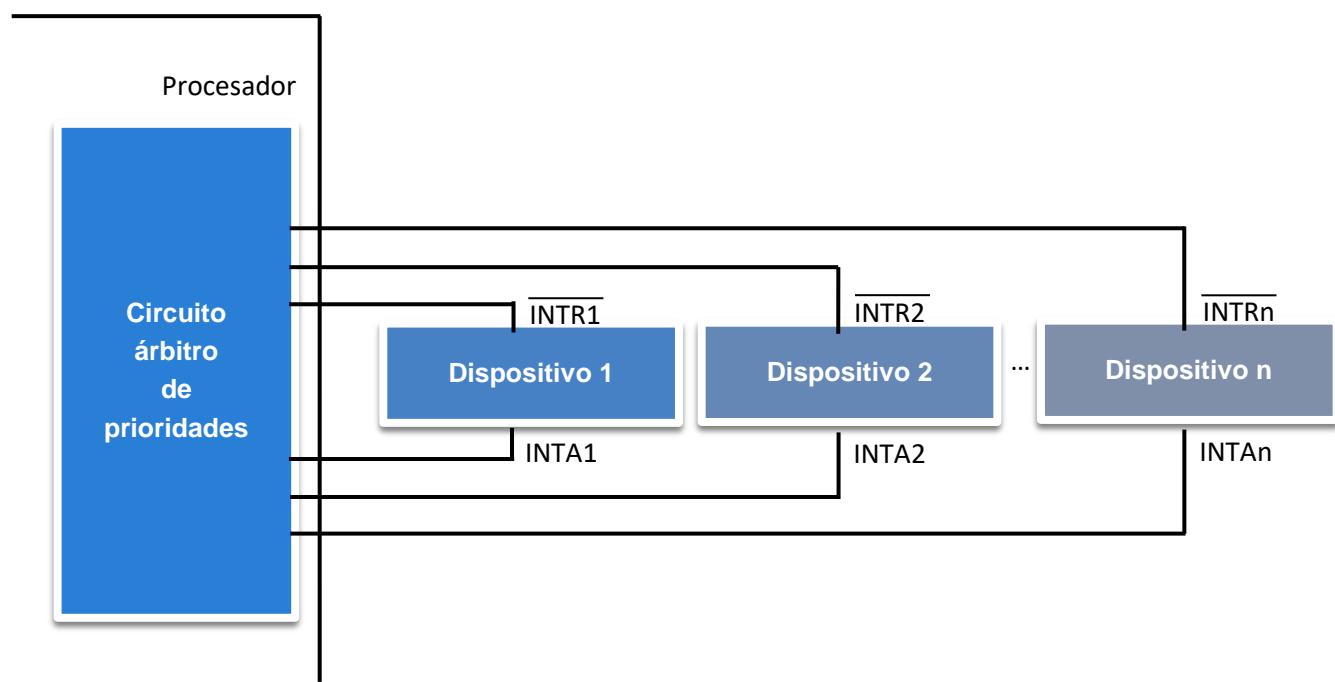
INTERRUPCIONES VECTORIZADAS

Cuando varios dispositivos tienen la capacidad de enviar una interrupción a través de la misma línea, es importante que el procesador tenga la capacidad de reconocer cuál dispositivo es el que ha generado dicha interrupción, una posibilidad para esto es que dicho dispositivo se auto identifique enviando un código especial por el bus, este código suele representar la dirección de comienzo de la rutina de interrupción para dicho dispositivo. Esta configuración implica que las rutinas de interrupción para un dispositivo dado deban comenzar siempre en la misma posición.

ANIDAMIENTO DE INTERRUPCIONES

Los dispositivos conectados a un microprocesador pueden ser organizados en una estructura de prioridades, una petición de interrupción por un dispositivo de alta prioridad debe ser aceptada mientras el procesador está ejecutando otra petición de un dispositivo de baja prioridad, esto significa que mientras se está ejecutando una rutina de interrupción el procesador deberá tener la capacidad de atender a algunas peticiones de interrupción de mayor prioridad, y de descartar aquellas generadas por dispositivos de menor prioridad.

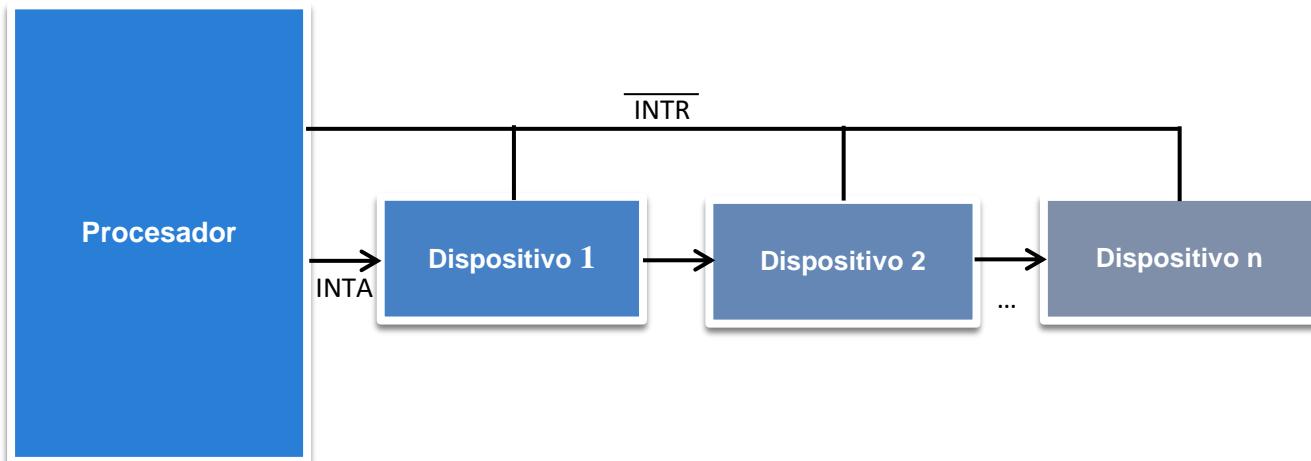
Un método sencillo para implementar un esquema con múltiples prioridades consiste en utilizar líneas separadas para peticiones de interrupción y de reconocimiento de interrupciones. Cada línea tendría asignado un nivel de prioridad. Las peticiones de interrupción recibidas en estas líneas se envían a un circuito árbitro de prioridades del procesador, y solo se aceptan aquellas con prioridad mayor a la interrupción que esté ejecutando en un momento dado el procesador.



PETICIONES SIMULTÁNEAS

Considere que llegan simultáneamente peticiones de interrupción de dos o más dispositivos. El procesador debe decidir qué petición servir primero. Si se utiliza un esquema de prioridades como el que se explicó antes, la solución es directa y el procesador realiza primero la rutina de interrupción de aquel dispositivo con mayor prioridad.

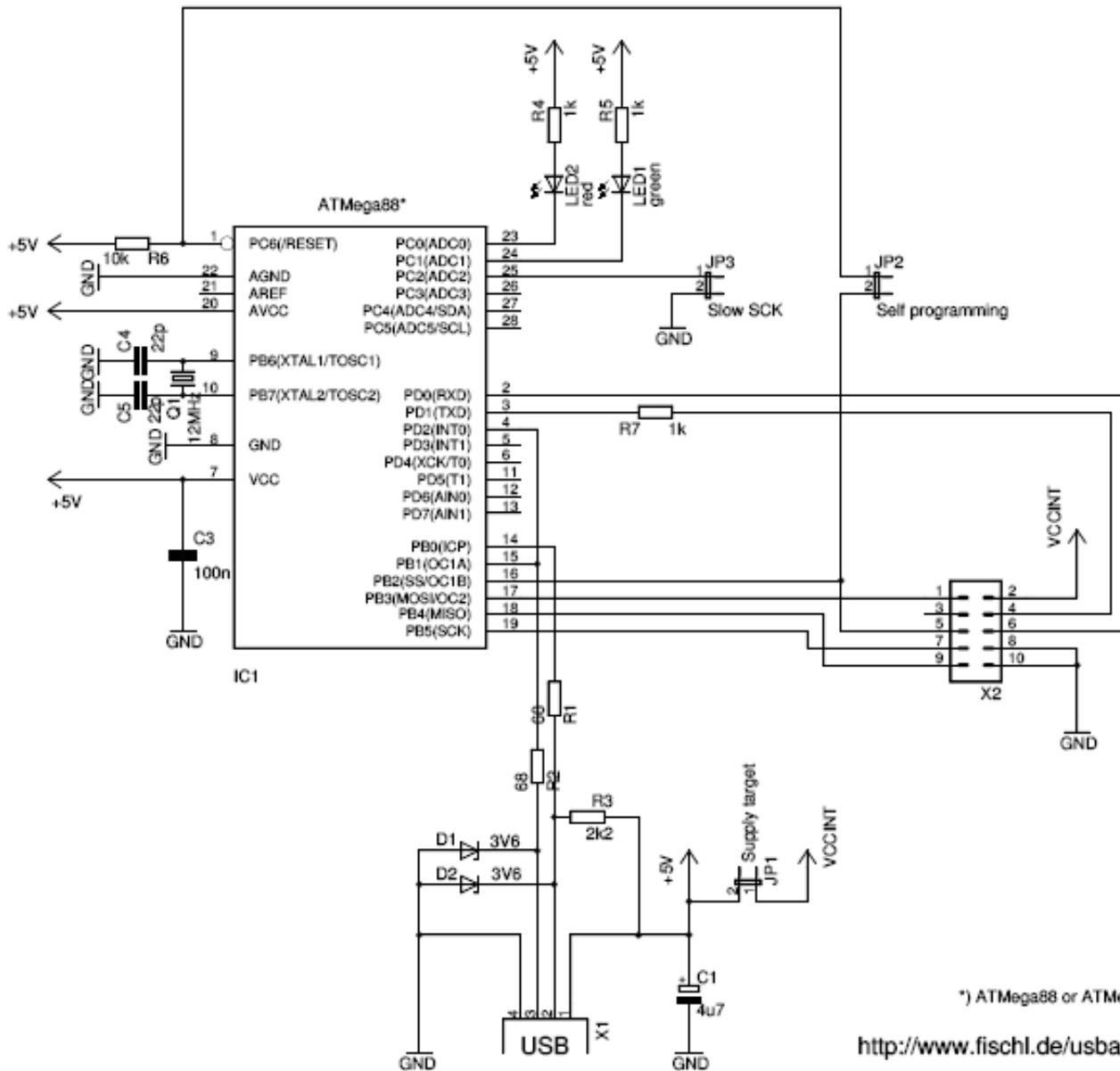
Cuando se emplean interrupciones vectorizadas se debe asegurar que solamente un dispositivo envía a través del bus su código de vector de interrupción, entonces un esquema utilizado frecuentemente es conectar los dispositivos en una lista encadenada como se muestra en la figura, ahí la línea de Interrupción es común a todos los dispositivos. La línea de reconocimiento de interrupción se propaga a través de los dispositivos de manera que cuando alguno de ellos envía una petición de interrupción activa la línea INTR, el procesador responde entonces fijando la línea INTA a 1, la señal es recibida por el dispositivo 1 y si él no fue quien generó la interrupción entonces pasa la señal al dispositivo 2. En caso de que el dispositivo que recibe la señal por INTA haya sido el que generó la interrupción, entonces ya no envía la señal por INTA y procede a enviar su código de identificación por la línea de datos. Entonces, cuando se tiene una lista encadenada, el dispositivo con mayor prioridad sería el que recibe primero la señal de INTA.



MICROCONTROLADOR ATMEL ATMEGA16A

Hardware del programador de AVR

La siguiente información ha sido obtenida de la página web <http://www.fischl.de/usbasp/>, en donde Thomas Fischl ha publicado el diseño de su Programador para AVR; se empleará un ATmega8 como “cerebro” para el microprocesador.

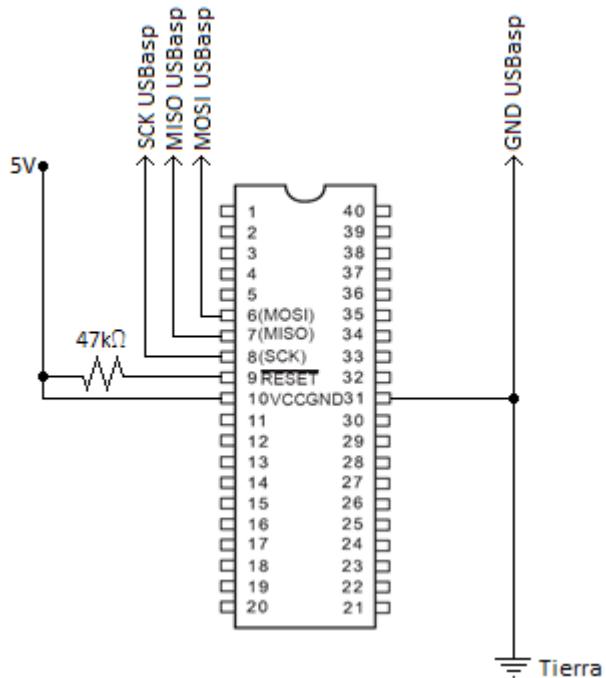


Para soldar el circuito impreso ponga una base en el lugar del microprocesador, de forma que pueda ser removido cuando sea necesario.

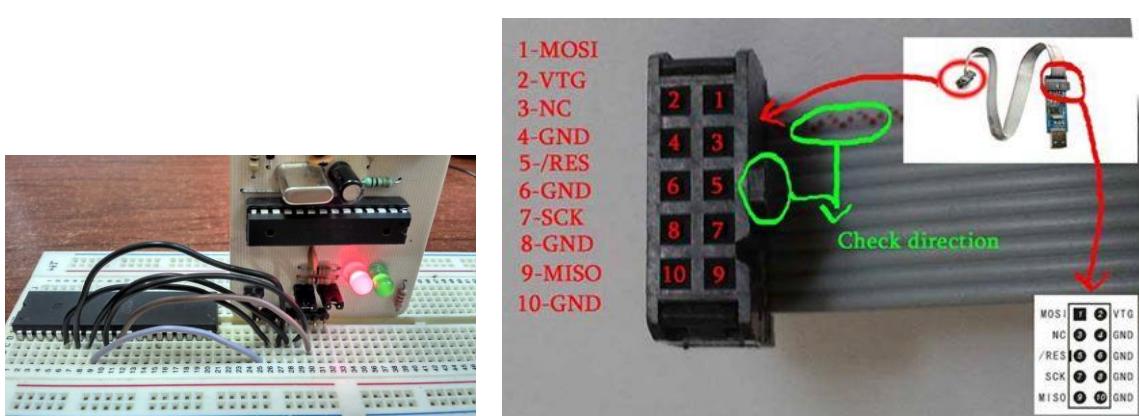
Una vez que tenga el circuito deberá pre programar su microprocesador ATMega8 con el software que se indicará.

Para probar su diseño deberá instalar en su computadora los drivers para el Programador USBasp y una vez que este haya sido detectado correctamente, deberá instalar en eXtremeBurnerAVRSetupV1.2.exe que será empleado para programar los microprocesadores.

Cuando se desee programar un micro este deberá conectarse con el USBasp como se muestra a continuación:



Debido a que el microprocesador funciona por default a una frecuencia de 1Mhz será necesario poner el jumper SLOW SCK en la tarjeta de programación (pues este Jumper debe estar activo cuando la frecuencia es <1.5kHz). No se sugiere utilizar el voltaje de su puerto USB para alimentar al circuito del microprocesador que desea programar, pues en caso de que se produzca un cortocircuito éste podría llegar a quemar su puerto USB, es por ello que se recomienda utilizar una fuente externa (protegiendo así su microprocesador)



Microcontrolador Atmel ATmega16A

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 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 Immediate**) 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 resultado 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 2^a. 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

Entrada/Salida: Puertos

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>
```

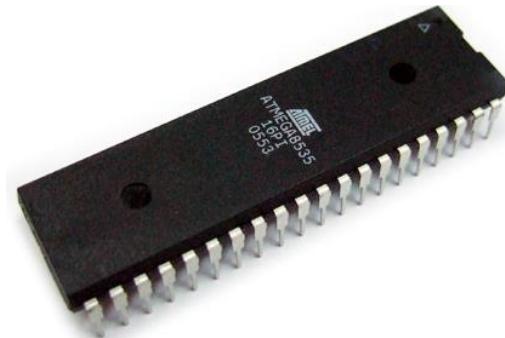
(se sugiere abrirlo mediante un block de notas, para ver cómo es, pero cuidando de no modificar nada).

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.

(XCK/T0) PB0	1	40	PA0 (ADC0)
(T1) PB1	2	39	PA1 (ADC1)
(INT2/AIN0) PB2	3	38	PA2 (ADC2)
(OC0/AIN1) PB3	4	37	PA3 (ADC3)
(SS) PB4	5	36	PA4 (ADC4)
(MOSI) PB5	6	35	PA5 (ADC5)
(MISO) PB6	7	34	PA6 (ADC6)
(SCK) PB7	8	33	PA7 (ADC7)
RESET	9	32	AREF
VCC	10	31	GND
GND	11	30	AVCC
XTAL2	12	29	PC7 (TOSC2)
XTAL1	13	28	PC6 (TOSC1)
(RXD) PD0	14	27	PC5
(TXD) PD1	15	26	PC4
(INT0) PD2	16	25	PC3
(INT1) PD3	17	24	PC2
(OC1B) PD4	18	23	PC1 (SDA)
(OC1A) PD5	19	22	PC0 (SCL)
(ICP1) PD6	20	21	PD7 (OC2)



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	SFIOR
Read/Write	ADTS2 R/W	ADTS1 R/W	ADTS0 R/W	- R	ACME R/W	PUD R/W	PSR2 R/W	PSR10 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

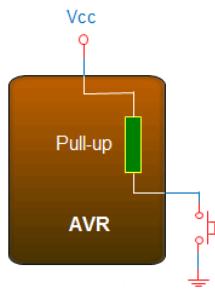
ldi r16, 0b00000100
out SFIOR, r16

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	
Read/Write	R/W	SREG							
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

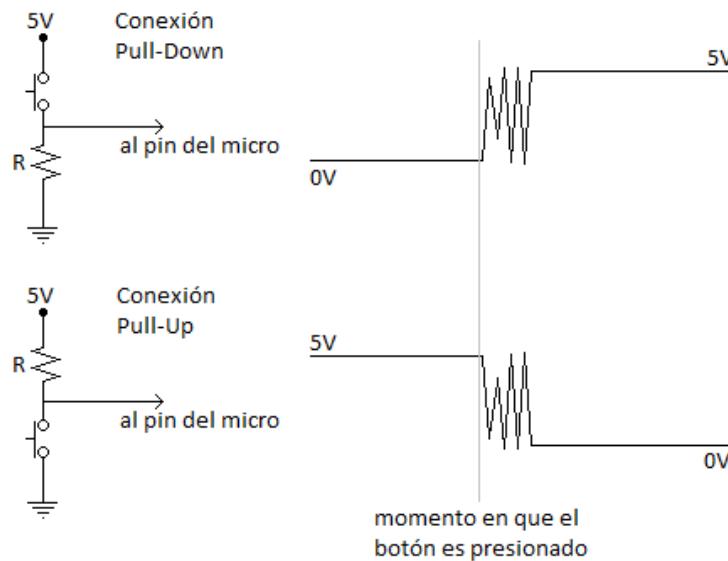
La mayoría de los bits del registro SREG se modifican en forma automática después de realizar operaciones o llevar a cabo ciertas instrucciones. A continuación, se tiene una tabla de todas las instrucciones de ensamblador que pueden modificar cada uno de estos bits.

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 sola 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

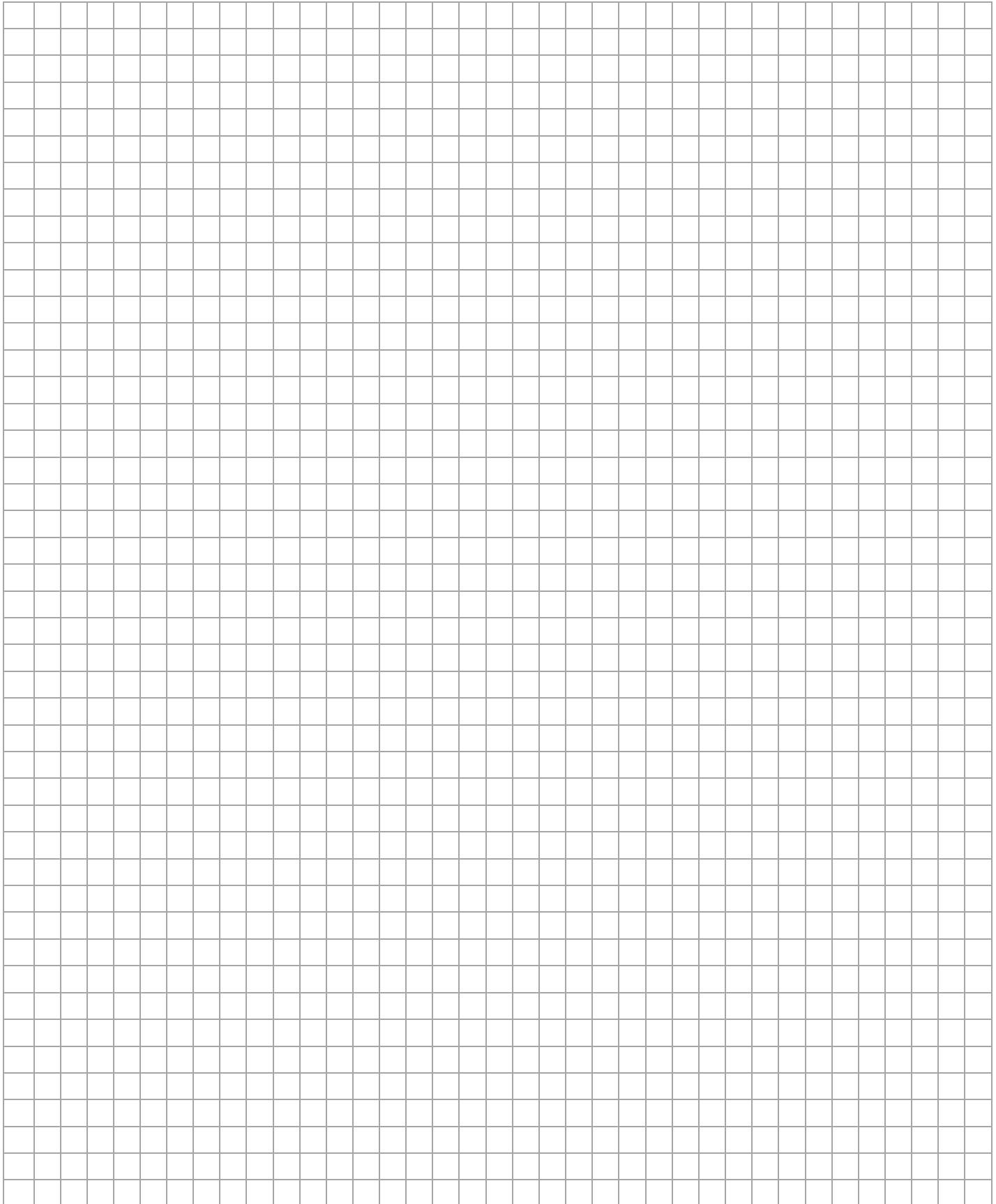
Retardo:

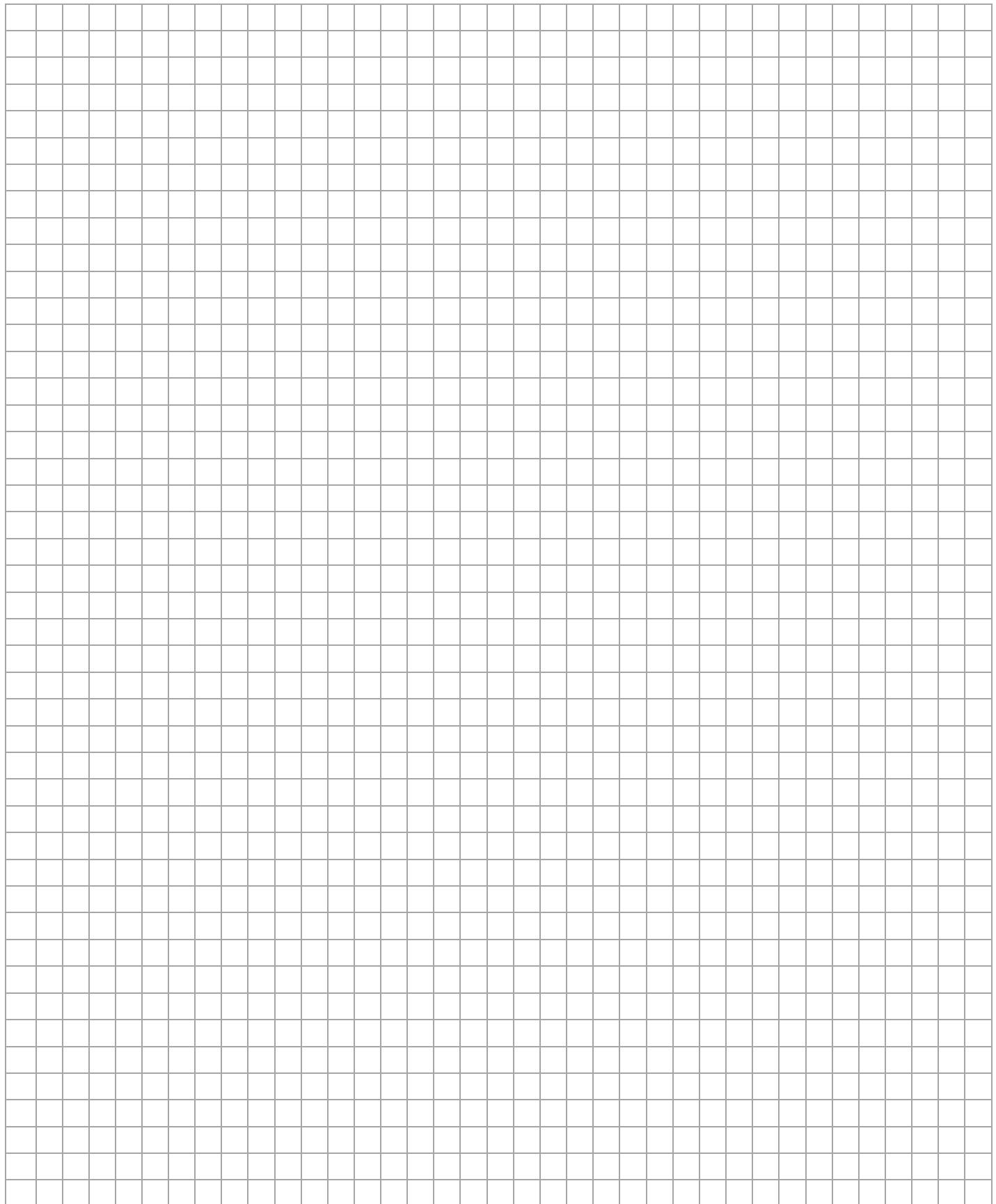
```
ldi Temporal1,0xff
ciclo1:
dec Temporal
breq Salir ;Brinca cuando la bandera Z (cero) está activa
ldi Temporal2,0xff
ciclo2:
    dec Temporal2
    breq ciclo1 ;Brinca cuando la bandera Z (cero) está activa
    rjmp ciclo2
```

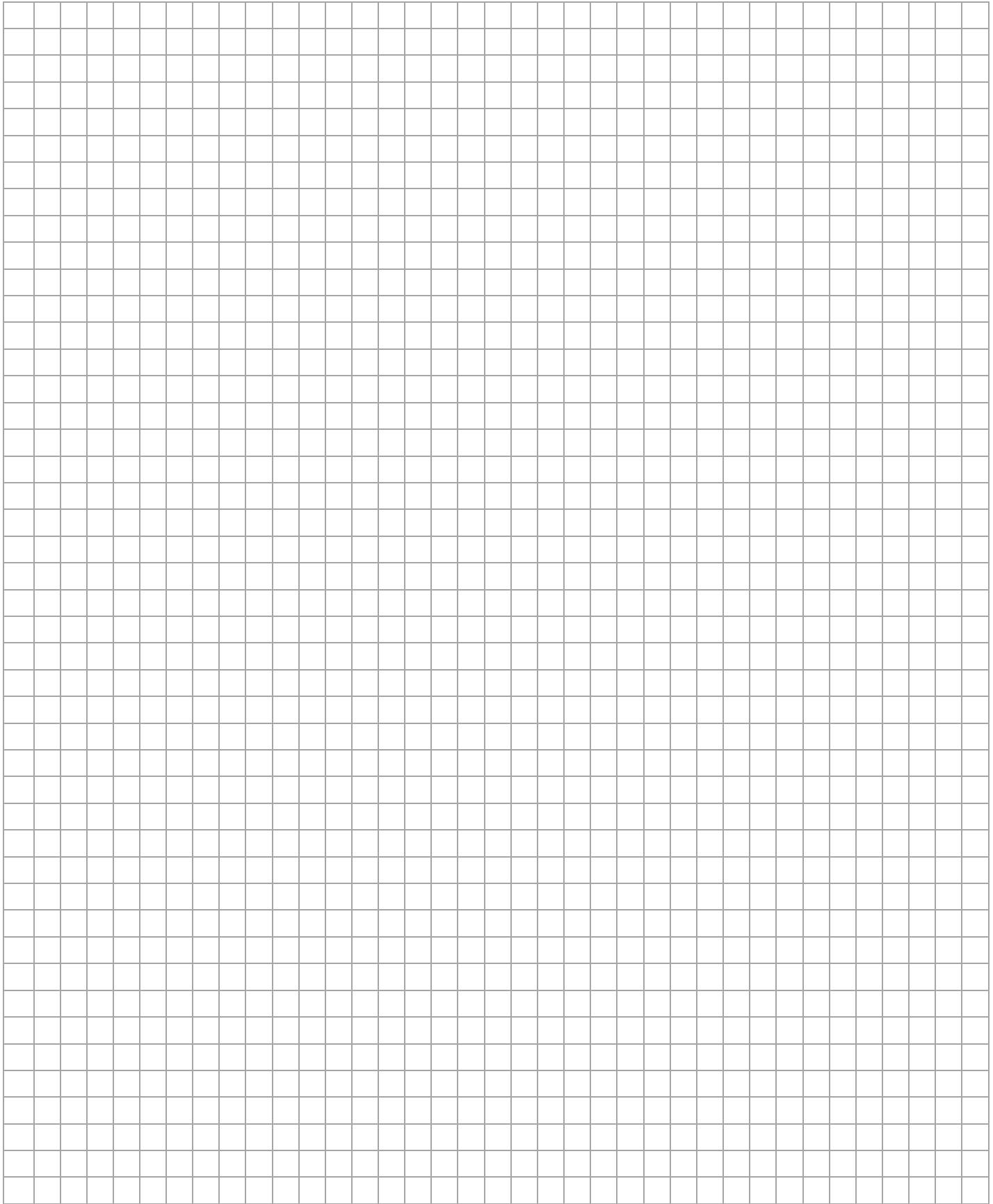
Salir:

```
ret
```

NOTAS PERSONALES – ENTRADA / SALIDA: PUERTOS







MICROCONTROLADOR COMO CABLE [ENTRADA/SALIDA:PUERTOS]

Con la finalidad de que sus programas le puedan ser útiles en un futuro se sugiere incluir al principio los siguientes comentarios:

```
/*
 * _01_Practica.asm
 *
 * Fecha de creación: Poner aquí la fecha
 * Autor: Poner aquí el nombre del programador
 * Se configura el puerto A como puerto de Entrada con resistencias de Pull-Up y el puerto B como puerto de salida, este programa transmite lo que entra por el puerto A al puerto B (sin hacerle ninguna modificación). En el puerto A se conectará un dip switch y en el puerto B se conectarán ocho LEDs con sus respectivas resistencias.
 */
```

En seguida, dependiendo del microcontrolador que se esté utilizando es necesario incluir el archivo de definiciones. Resulta interesante abrir este archivo en un bloc de notas y analizar su contenido para entender a detalle su funcionamiento (cuidando de ir a realizar cambios en él).

.include "m16adef.inc" ; Incluye el archivo de definiciones del ATmega16A

A continuación es recomendable nombrar los registros y definir las constantes que se emplearán a lo largo del programa. Esto podría hacerse en cualquier lugar del código sin embargo al estar al principio se facilita su consulta en momentos posteriores.

```
.def Temporal = R16 ; Le pone el nombre "Temporal" al registro R16
.equ Cinco = 5 ;En cualquier lugar donde se escriba Cinco corresponderá al número
```

Posterior a esto es necesario garantizar que el código se almacene en la posición de memoria 0x0000 para que ahí se comience a almacenar el vector de interrupciones.

```
.org 0x0000
jmp RESET ; Reset Handler
jmp EXT_INT0 ; IRQ0 Handler
jmp EXT_INT1 ; IRQ1 Handler
jmp TIM2_COMP ; Timer2 Compare Handler
jmp TIM2_OVF ; Timer2 Overflow Handler
jmp TIM1_CAPT ; Timer1 Capture Handler
jmp TIM1_COMPA ; Timer1 CompareA Handler
jmp TIM1_COMPB ; Timer1 CompareB Handler
jmp TIM1_OVF ; Timer1 Overflow Handler
jmp TIM0_OVF ; Timer0 Overflow Handler
jmp SPI_STC ; SPI Transfer Complete Handler
jmp USART_RXC ; USART RX Complete Handler
jmp USART_UDRE ; UDR Empty Handler
jmp USART_TXC ; USART TX Complete Handler
jmp ADC_COMP ; ADC Conversion Complete Handler
```

```

jmp EE_RDY ; EEPROM Ready Handler
jmp ANA_COMP ; Analog Comparator Handler
jmp TWI ; Two-wire Serial Interface Handler
jmp EXT_INT2 ; IRQ2 Handler
jmp TIM0_COMP ; Timer0 Compare Handler
jmp SPM_RDY ; Store Program Memory Ready Handler

```

Incluya la etiqueta Reset y dentro de ella configure el Stack Pointer (Puntero de Pila) para que el programa funcione correctamente

Reset:

```

ldi r16,high(RAMEND)
out SPH,r16
ldi r16,low(RAMEND)
out SPL,r16

```

Ahora incluya el código para configurar los puertos de entrada y salida.

```

ldi Temporal, 0x00 ; Guarda un 0b00000000 en R16 (que se llama Temporal)
out DDRA, Temporal ; Para que el puerto A sea un puerto de entrada
ldi Temporal, 0xFF ; Guarda un 0b11111111 en R16 (que se llama Temporal)
out DDRB, Temporal ; Para que el puerto B sea un puerto de salida

```

Y el código necesario para configurar las resistencias de Pull-Up en el puerto A

```

ldi Temporal, 0xFF ; Guarda un 0b11111111 en R16 (que se llama Temporal)
out PORTA, Temporal ; Para que el puerto A tenga las resistencias de Pull-Up activas

```

Ahora escriba el código para leer el puerto A y lo que haya leído sáquelo por el puerto B

```

in Temporal, PINA ; Guarda en R16 lo que hay en el Puerto de entrada A
out PORTB, Temporal ; Saca por el Puerto B lo que quedó guardado en R16

```

Pero con estas líneas el programa únicamente revisará una vez lo que hay en la entrada y lo sacará de inmediato, pero lo que se desea es que esté revisando constantemente y sacando lo que haya encontrado, es por ello que se requiere hacer un ciclo, por lo que las dos líneas anteriores deben quedar:

Ciclo:

```

in Temporal, PINA ; Guarda en R16 lo que hay en el Puerto de entrada A
out PORTB, Temporal ; Saca por el Puerto B lo que quedó guardado en R16
rjmp Ciclo ; Para volver a leer el puerto...

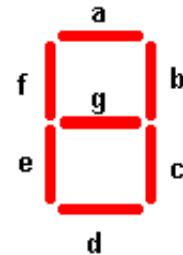
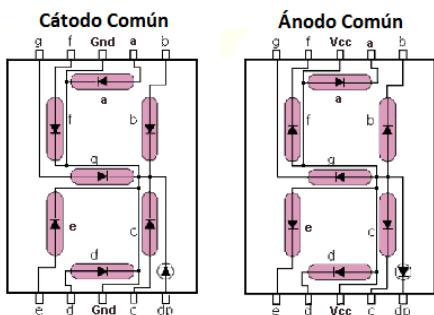
```

CONVERTIDOR BCD A SIETE SEGMENTOS [ENTRADA/SALIDA:PUERTOS]

Se desea realizar un programa en el que se utilicen 4 bits del puerto A como entrada (con sus resistencias de pull-up) a las que estarán conectadas 4 switches de un dip-switch. En el puerto B se conectará un display de siete segmentos (con sus resistencias necesarias).

El programa deberá convertir el código BCD que reciba en el puerto A (en los cuatro bits menos significativos) de forma de que por el display de siete segmentos conectado en el puerto B (en los siete bits menos significativos), muestre el número correspondiente (no se permite el uso del 7447 o 7448). Será importante que antes de diseñar su programa elija que tipo de display de siete segmentos empleará (ánodo común o cátodo común), pues de ello depende el valor que tendrá que sacar para mostrar el número deseado.

En caso de que se introduzca un valor BCD equivocado (mayor a 1001) el display deberá quedar apagado.



Decimal	BCD (Entradas del puerto A)				Siete segmentos (Salidas del puerto B)						
	A3	A2	A1	A0	B6 - a	B5 - b	B4 - c	B3 - d	B2 - e	B1 - f	B0 - g
0	0	0	0	0							
1	0	0	0	1							
2	0	0	1	0							
3	0	0	1	1							
4	0	1	0	0							
5	0	1	0	1							
6	0	1	1	0							
7	0	1	1	1							
8	1	0	0	0							
9	1	0	0	1							

Se recomienda que antes de empezar a programar realice un diagrama de flujo para tener en claro el orden del programa que realizará (si requiere apoyo en su código, se le pedirá que muestre su diagrama de flujo, lo más claro posible, para poder entender lo que intentó hacer y poder explicar dónde está el problema).

Probablemente necesitará utilizar las instrucciones CPI, BREQ, BRNE (investigue al respecto de ellas y de todas las que deseé) en el archivo AVR Instruction Set.pdf.

CONTADOR SIMPLE [ENTRADA/SALIDA:PUERTOS]

Para esta práctica se conectará un botón (sin algún elemento para quitar físicamente el rebote) al Pin 0 del puerto A. En los cuatro bits menos significativos del puerto B se conectará 74LS47 (ánodo común) o bien un 74LS48 (cátodo común) y el display de siete segmentos correspondiente.

Inicialmente el display deberá mostrar un cero. Se desea realizar un programa que se encargue de contar de uno en uno cada vez que se presiona el botón. Si el contador se encuentra en 9 y se presiona nuevamente el botón la cuenta deberá regresar a cero.

NOTA IMPORTANTE RESPECTO AL REBOTE:

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. Esto normalmente puede ser corregido implementando algún tipo de filtro sin embargo, cuando se utiliza un microprocesador resulta aún más sencillo y económico añadir una rutina antirrebote al programa.

La manera más sencilla de realizar esta rutina antirrebote 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 stabilice (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 sola vez no incremente la cuenta muchas veces. (Es decir... deberá detectar cuándo el usuario presiona el botón y hacer que su código se quede trabado sin poder contar más hasta el momento en que suelte el botón)

Analice el siguiente código (generador de un retardo), el cual se basa en ir decrementando el contenido de un registro de 255 a cero por 255 veces (el contenido de otro registro), es decir en total 65025. Para utilizarla, al principio de su programa deberá declarar los registros Temporal1 y Temporal2.

Retardo:

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

Note que esta es una función pues dentro de ella se tiene la instrucción RET. Debido a esto, cuando usted desee usarla, únicamente la necesita mandar a llamar como RCALL Retardo, y una vez que la rutina termine, automáticamente volverá al punto siguiente de dónde la mandó a llamar.

Se sugiere investigar el uso de la instrucción **DEC, INC, SBRS, SBRC, SBIS, SBIC**

CONTADORES INDEPENDIENTES [ENTRADA/SALIDA:PUERTOS]

Ya se ha explicado con anterioridad que la manera más sencilla de realizar una 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).

Ahora bien, lo que el programa realice después de este tiempo de retardo es lo que determinará si la respuesta se da al momento de presionar o de liberar el botón. Cuando se desea que el programa responda al momento de presionar el botón, al terminar el retardo se hace inmediatamente la rutina deseada para responder al pulso de botón y luego se cicla el programa hasta que el voltaje vuelva a su estado normal; por el contrario, si lo que se desea es que el botón responda al momento de ser liberado entonces al terminar el retardo se vuelve a revisar el puerto y se cicla el programa hasta que el voltaje vuelva a su estado normal (cuando el botón no está presionado).

Las conexiones necesarias para el programa de esta práctica son:

En los cuatro bits menos significativos del puerto A, el microcontrolador tendrá conectados cuatro botones (Botón 1, Botón 2, Botón 3 y Botón 4), en los cuatro bits menos significativos del puerto B se conectarán cuatro LEDs (LED 1, LED 2, LED 3, LED4) y por último en los cuatro bits más significativos del puerto B se tendrán conectados otros cuatro LEDs (LED 5, LED 6, LED 7, LED8).

El programa deberá estar leyendo constantemente el puerto en donde están conectados los botones. Los botones 1 y 2 servirán para controlar a los cuatro primeros LEDs, mientras que los botones 3 y 4 se utilizarán para controlar a los otros cuatro LEDs.

En un principio todos los LEDs deberán estar apagados.

Cuando el usuario presione el botón 2, deberá incrementarse la cuenta (en binario) de los primeros cuatro LEDs, y así se seguirá incrementando cada vez que el usuario presione el botón 2, hasta llegar a 1111, de tal forma que si el usuario volviese a presionar el botón 2 la cuenta de esos LEDs regresará a 0000.

Cuando el usuario presione el botón 1, la cuenta que se llevaba en los tres primeros LEDs deberá volver a 0000.

Cuando el usuario suelte el botón 4, deberá incrementarse la cuenta (en binario) de los segundos cuatro LEDs, y así se seguirá incrementando cada vez que el usuario suelte el botón 4, hasta llegar a 1111, de tal forma que si el usuario volviese a soltar el botón 4 la cuenta de esos LEDs regresará a 0000.

Cuando el usuario suelte el botón 3, la cuenta que se llevaba en los cuatro segundos LEDs deberá volver a 0000.

Los botones podrán ser presionados en el orden que el usuario desee, y tantas veces como se quiera.

Se sugiere investigar el uso de la instrucción **MOV, OR, AND, ORI, ANDI, LSL, LSR, CLR, SER, DEC, SWAP**

SUMADOR COMPARADOR [ENTRADA/SALIDA:PUERTOS]

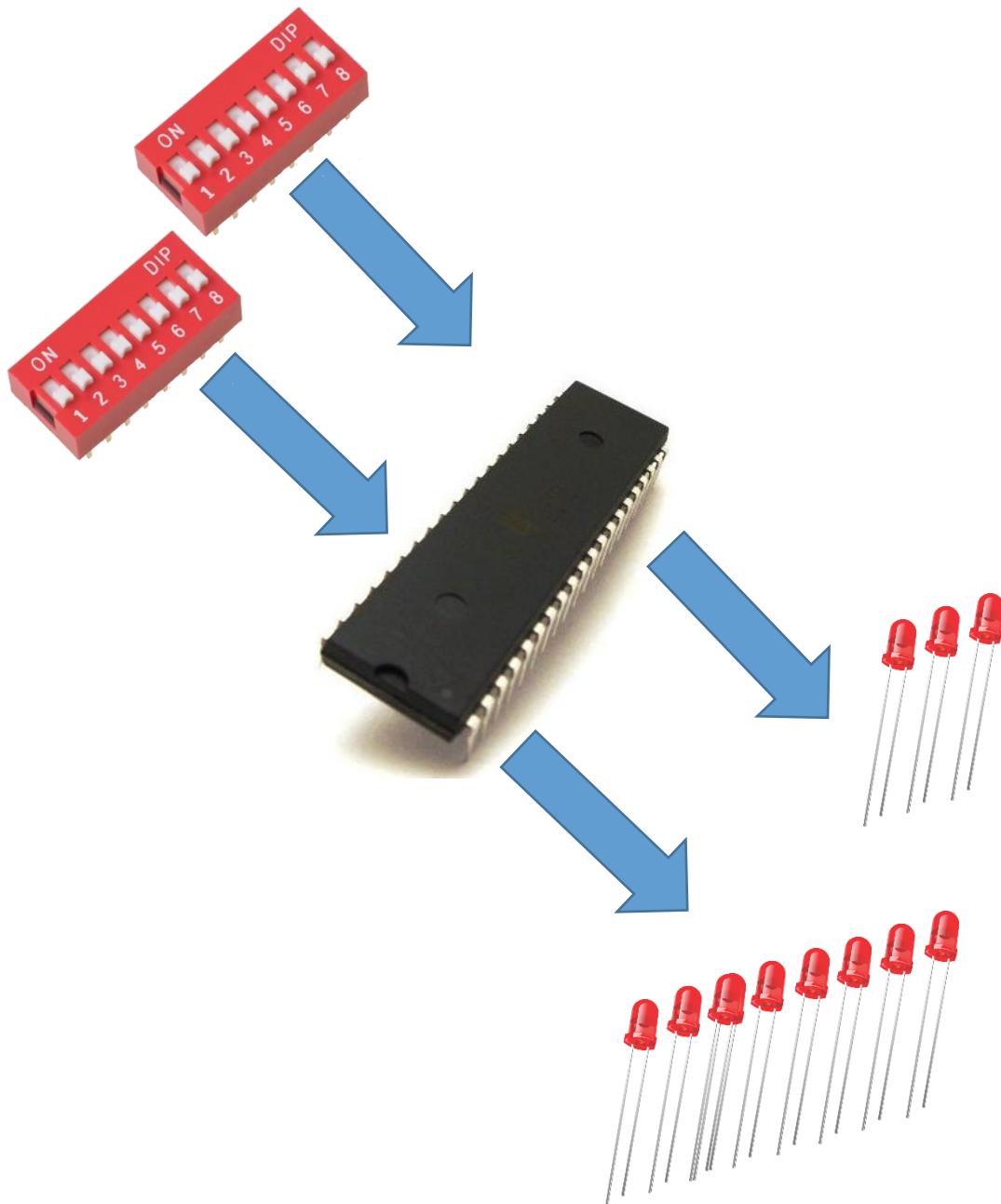
Se desea diseñar un circuito al cual se le introducirán dos números binarios A y B por medio de dos dip switches (cada uno de 8) y que entregará el resultado de la suma de esos números (sin considerar el acarreo) a través de ocho LEDs.

Además el circuito tendrá conectados otros tres LEDs, cada uno servirá para indicar las siguientes condiciones:

$A > B$

$A = B$

$A < B$



CONTROL BÁSICO DE MOTOR DE PASOS [ENTRADA/SALIDA:PUERTOS]

Será necesario que usted conozca la secuencia correcta para hacer funcionar su motor de pasos, en caso de que no la conozca deberá investigarla haciendo pruebas, para saber cómo hacerlas existe gran cantidad de información en libros y en internet. (Busque un motor de pasos unipolar en forma de paso completo y de medio paso).

En segundo lugar será necesario que determine el circuito que empleará para proporcionar a sus motores de pasos la corriente necesaria para su funcionamiento, pues hay que recordar que el microcontrolador ATmega16A es capaz de proporcionar una corriente mínima en sus salidas (aproximadamente 20mA), se sugiere usar un ULN2003, sin embargo existen múltiples soluciones y está en libertad de implementar la que mejor le convenga.

Configure el AVR de la siguiente forma

Puerto A - Entrada

Puerto B – Salida

Elija los bits menos significativos del puerto B para realizar las conexiones necesarias para su motor de pasos. Se sugiere que en un principio no conecte directamente el motor sino que pruebe su programa con LEDs, de forma que pueda verificar que realmente se está produciendo la secuencia deseada.

Conecte dos botones en los bits menos significativos del puerto A, dichos botones tendrán la siguiente funcionalidad:

Botón 0 – Mientras que el botón 0 se encuentre presionado, el motor de pasos estará girando en forma continua en sentido de las manecillas del reloj. (Para esta práctica no nos interesa la velocidad de giro).

Botón 1 – Mientras que el botón 1 se encuentra presionado, el motor de pasos estará girando en forma continua en sentido contrario a las manecillas del reloj. (A la misma velocidad con la que giraba en el sentido contrario).

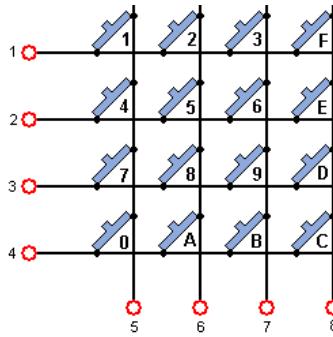
Una vez que su programa esté funcionando correctamente y que usted lo pueda verificar revisando las secuencias en LEDs, realice las conexiones necesarias para probar el motor y verifique su funcionamiento.

BRAZO SELECCIONADOR DE OBJETOS [ENTRADA/SALIDA:PUERTOS]

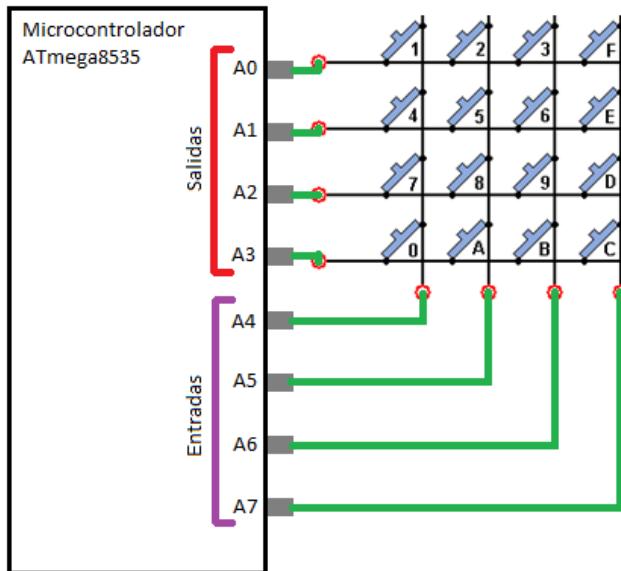
Teclado Matricial

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.

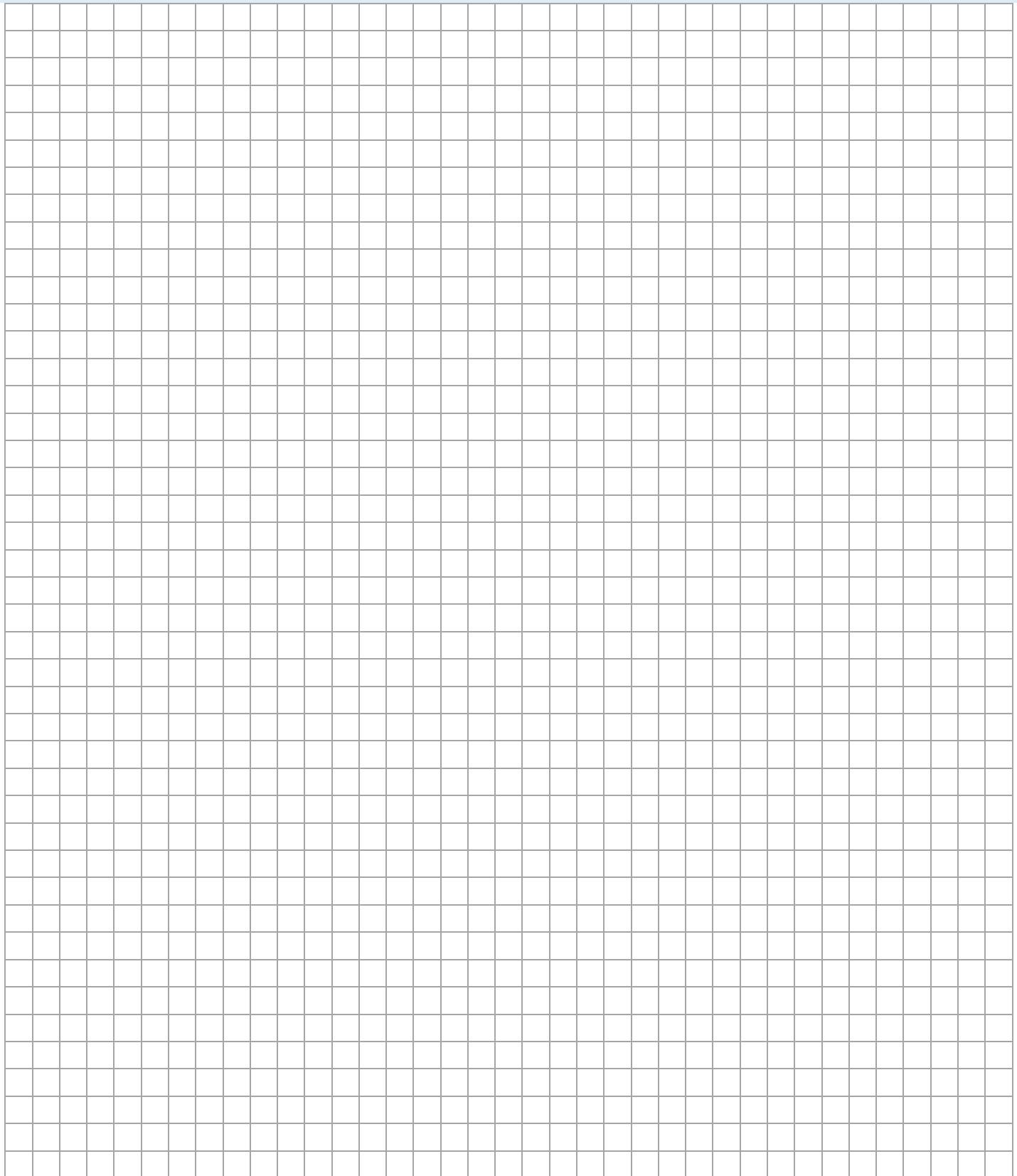


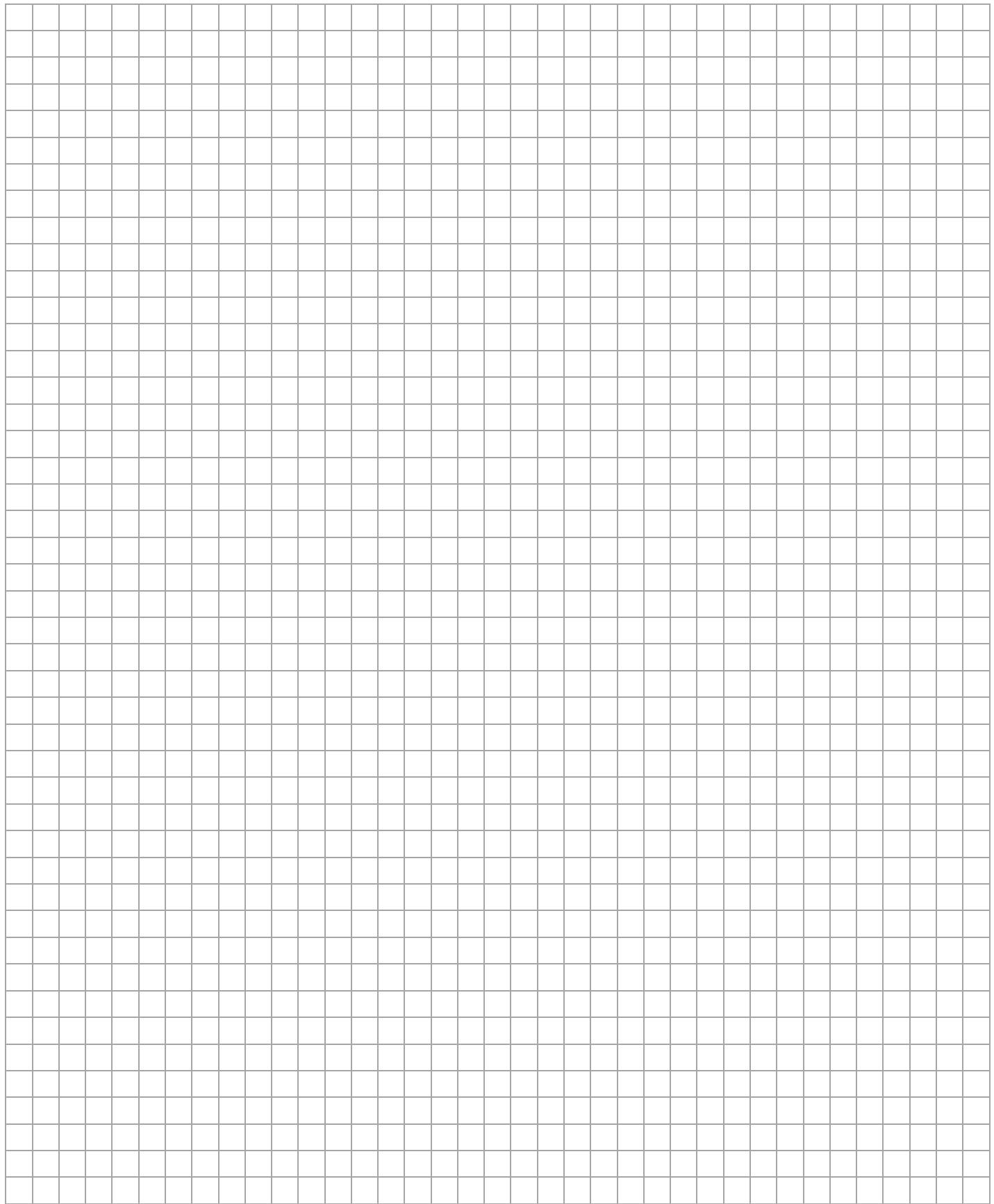
Para utilizar un teclado matricial con el microprocesador se conectan las filas a pines de salida y las columnas a pines de entrada o viceversa. Es recomendable utilizar las resistencias de pull-up internas del microcontrolador para no tener que hacer conexiones externas. En la siguiente figura se muestra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

NOTAS PERSONALES – TECLADO MATRICIAL

A large grid of squares, approximately 20 columns by 25 rows, designed for taking notes or drawing diagrams.



TECLADO BÁSICO [TECLADO MATRICIAL]

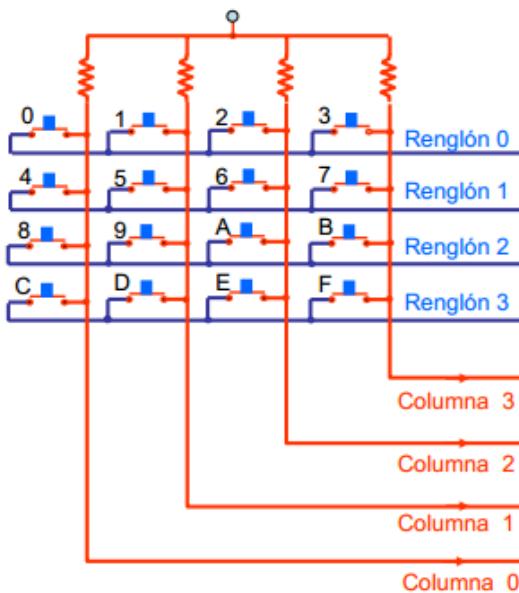


Se conectarán el teclado matricial en el puerto A del microcontrolador, los bits menos significativos serán definidos como salidas y los más significativos se definirán como entradas.

En los cuatro bits menos significativos del puerto B del microcontrolador se conectarán un decodificador de BCD a siete segmentos y su correspondiente display (al que llamaremos display 1).

En los cuatro bits más significativos del puerto B del microcontrolador se conectarán otro decodificador de BCD a siete segmentos y su correspondiente display (al que llamaremos display 2).

Al prender el circuito ambos displays deberán estar totalmente apagados, cuando el usuario presione una tecla, el número que presionó deberá mostrarse en el display 1 (y el otro continuará apagado), al presionar la siguiente tecla el número del display 1 pasará al display 2, mientras que el número presionado aparecerá en el display 1, y de esta forma continuará funcionando.



CANDADO DIGITAL [TECLADO MATRICIAL]

Configure el AVR de la siguiente forma

Puerto A – Salida

Puerto B – Entrada

En el puerto B conecte un teclado matricial. En el bit menos significativo del puerto A (bit0) conecte un LED que nos servirá para visualizar el dato que se enviará al circuito de la cerradura digital, el cual con un 0 se encuentra desactivado y permite abrir la puerta, mientras que con 1 se encuentra activo y la puerta se encuentra cerrado. En el bit 1 del puerto A arme el circuito necesario para el funcionamiento de un buzzer.

Se requiere diseñar un candado digital que ocupará los botones A, B y C del teclado matricial (en caso de que no los tenga, puede elegir cuales botones los representarán) y que tendrá dos salidas: cerradura y alarma (El estado inicial de cerradura es un 1 que equivale a se encuentra activa y la alarma deberá estar en silencio). El candado deberá cumplir con los siguientes requisitos:

1. Cuando se presione la combinación B-C-A-C la cerradura deberá abrirse, es decir se le enviará un cero.
2. Cuando la cerradura ya está abierta, se cerrará presionando cualquier tecla.
3. Para resetear el candado a su estado inicial se deberá presionar el botón Reset, la cual deberá funcionar desde cualquier estado pero no funcionará cuando la alarma esté activada.
4. Para prevenir que un usuario “busque” la combinación de entrada, se desea que la alarma se active solamente cuando, después del estado inicial, se han presionado cuatro botones en una combinación errónea, por ejemplo: B-C-B-A, A-B-C-A, C-C-B-A, etc.)
5. Una vez que la alarma se activa, el circuito se quedará en ese estado y la única manera de que el candado salga de la alarma es presionando la secuencia C-A.

Se sugiere considerar que las instrucciones **SBIS** y **SBIC** pueden usarse incluso para leer pines de puertos de salida (es decir que pueden usarse no sólo para leer pines de puertos de entrada, sino también para verificar lo que estamos sacando por un pin de un puerto en cualquier momento).

Se sugiere investigar el uso de la instrucción **ANDI, ORI, SBI, CBI**

NÚMEROS DE CUATRO DÍGITOS [TECLADO MATRICIAL]

Para este programa se requiere conectar el teclado matricial en el Puerto A del microcontrolador, por otra parte se requieren cuatro decodificadores BCD a Siete segmentos con sus correspondientes displays, las unidades deberán quedar en los cuatro bits menos significativos del puerto B, las decenas en los cuatro bits más significativos del puerto B, las centenas en los cuatro bits menos significativos del puerto C y los millares en los cuatro bits más significativos del puerto C.

Al iniciar el programa todos los displays aparecerán con ceros, cada vez que se presione un número en el teclado matricial deberá aparecer en el display de las unidades y recorrer los otros números hacia la izquierda

A manera de ejemplo digamos que recién inicie el programa tendremos 0000 en los displays



Si presionamos el 9 en el teclado matricial, sin importar el tiempo que permanezca presionada la tecla, los displays deberán cambiar a:



Si presionamos entonces el 3 deberá aparecer:



Supongamos que presionamos entonces la tecla 7 y posteriormente la tecla 2, en los displays aparecería



Si después de esto presionamos la tecla 4, entonces el 9 desaparecerá, los otros números se recorrerán y quedará:



y así continuará funcionando.

Alguna de las teclas que contiene el teclado matricial (Una tecla no numérica) deberá ser asignada para que, al momento de presionarla se resetee el programa y los displays vuelvan a 0000.

Interrupciones Externas

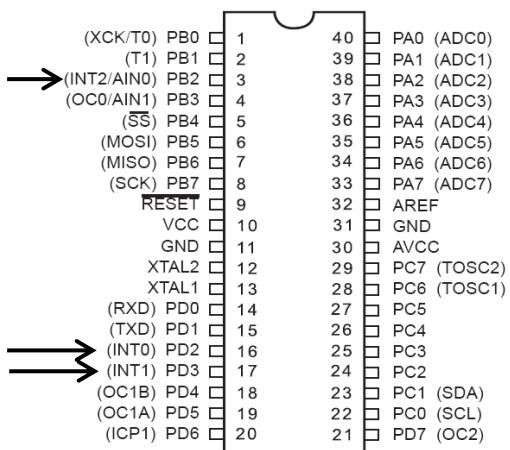
Hay muchas ocasiones en las que es necesario que nuestro programa reaccione ante determinados eventos de hardware, es decir, cuando el microcontrolador detecte un cambio en un pin de entrada. Esto puede hacerse programando un ciclo y monitoreando constantemente el pin deseado para que en el momento que 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 la misma. 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 primer registro a configurar es el **MCU Control Register (MCUCR)**, que tiene los siguientes bits:

Bit	7	6	5	4	3	2	1	0	MCUCR
Read/Write	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 bit 0 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	MCUCSR
	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF	
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	GICR
	INT1	INT0	INT2	-	-	-	IVSEL	IVCE	
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	

Initial Value

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

El último de los registros que se relaciona con las interrupciones es el **General Interrupt Flag Register (GIFR)**

Bit	7	6	5	4	3	2	1	0	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 INTFO, 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.

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 configuraremos 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 generarlos puede ayudarse de las siguientes tablas:

GIFR

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

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

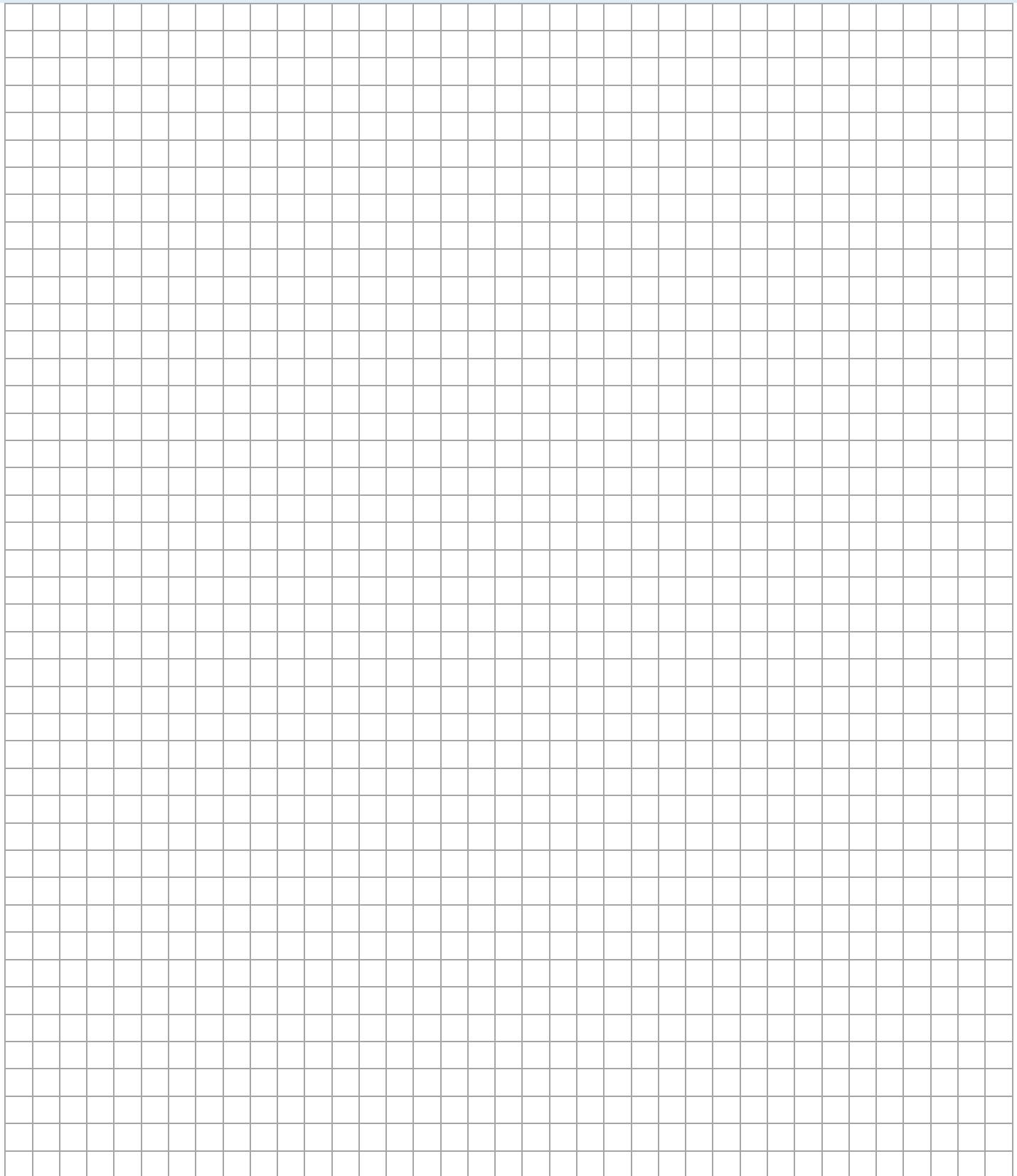
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
			0	0	0	0	0
INT1	INT0	INT2	-	-	-	IVSEL	IVCE
0 = int. no habilitada 1 = int. habilitada	0 = int. no habilitada 1 = int. habilitada	0 = int. no habilitada 1 = int. habilitada				Interrupt vector select	Interrupt vector change enable

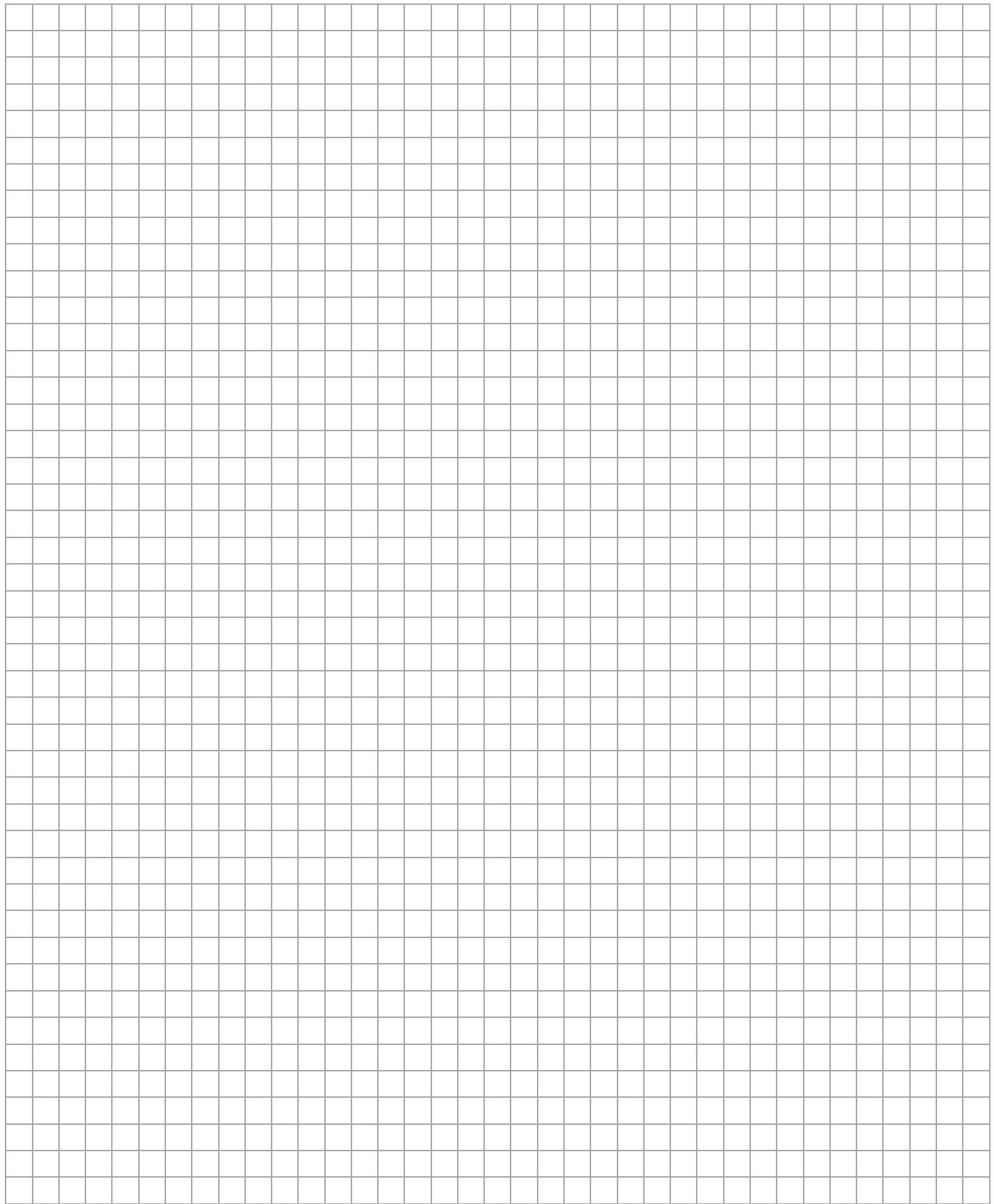
Por lo tanto, el código necesario para habilitar las interrupciones externas sería:

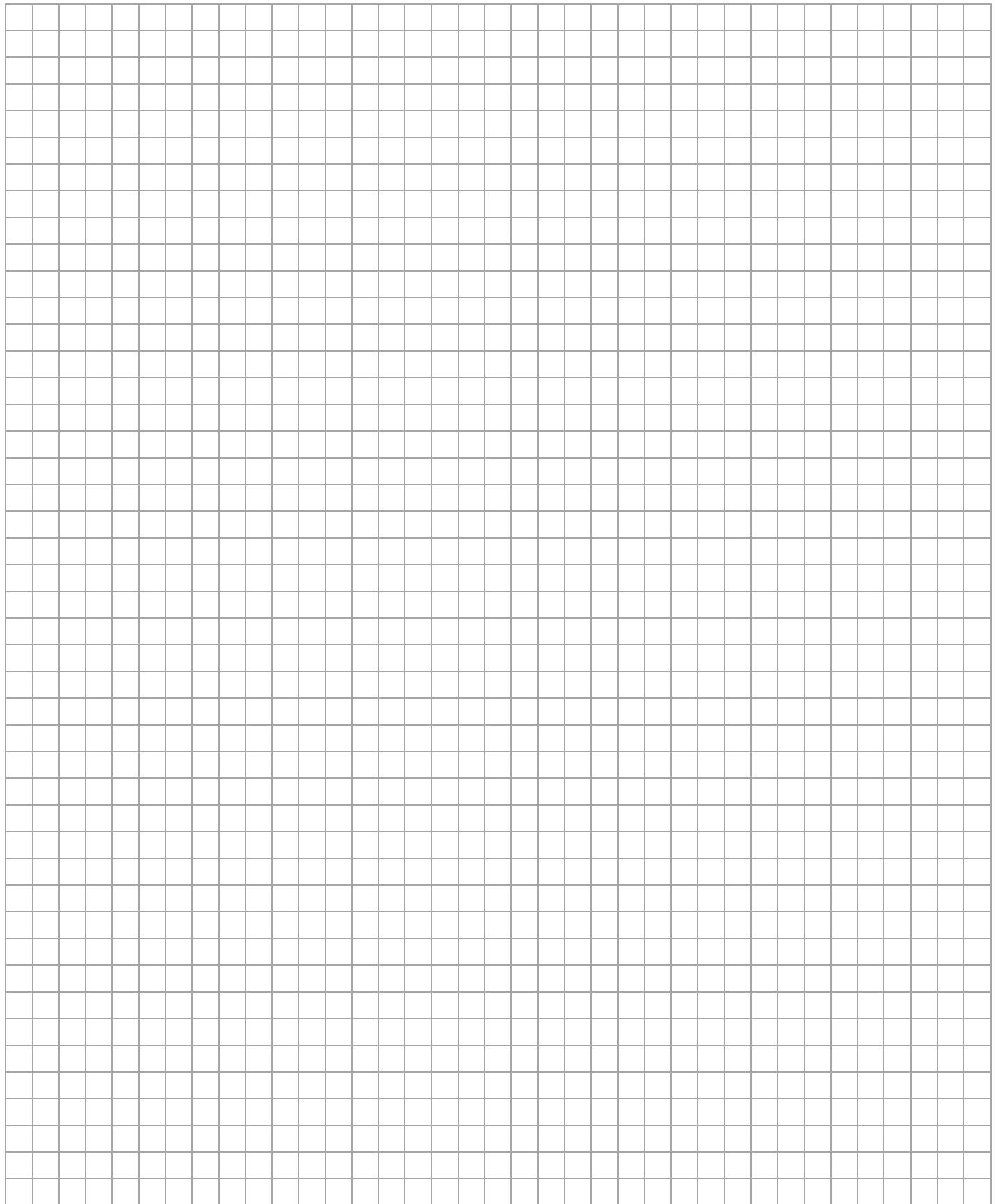
```
ldi R16, 0b_ _ _ 00000 ; se eligen las interrupciones externas que estarán habilitadas
out GICR, R16 ; lo mandamos al GICR
```

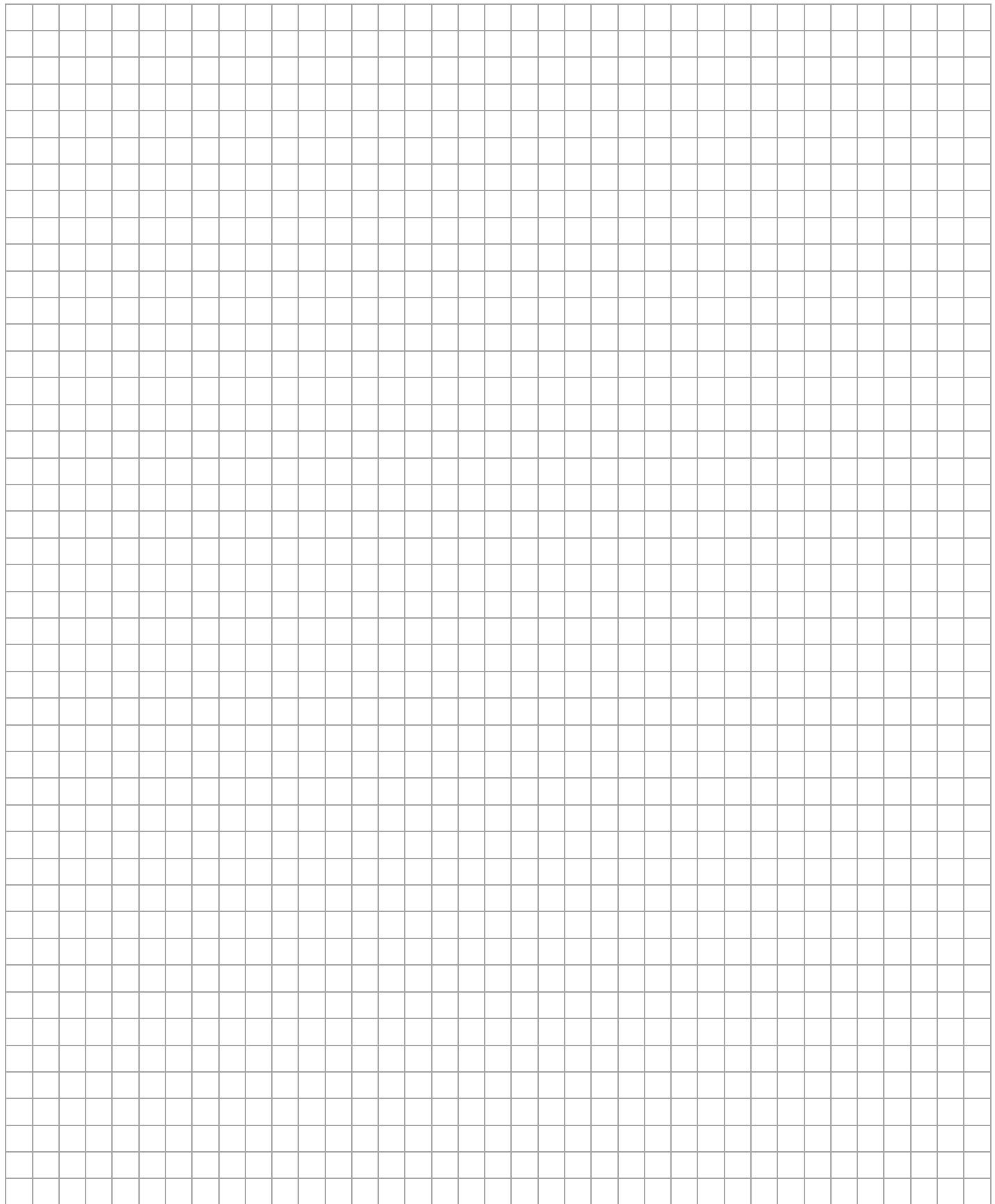
Y de esta forma quedaría listo el programa para que, al momento de generarse una interrupción, se ejecute la rutina que hayamos especificado en los vectores de interrupción al comienzo del programa.

NOTAS PERSONALES – INTERRUPCIONES EXTERNAS

A large grid of squares, approximately 20 columns by 25 rows, designed for taking notes on external interruptions. The grid is composed of thin, light gray lines on a white background.







CONTADOR AUTOMÁTICO CON INTERRUPCIONES [INTERRUPCIONES EXTERNAS]

Se conectarán al microcontrolador dos displays de 7 segmentos (UNIDADES y DECENAS) y dos botones (INICIAR y RESETEAR) (tal cual se muestran en el archivo de proteus, en caso de que usted lo desee podrá hacer los cambios que requiera en las conexiones).

Al momento de encender el dispositivo este deberá mostrar 00 en los displays.

Cuando el usuario presione el botón INICIAR los displays comenzarán a incrementarse cada 5 segundos (generados a través de un retardo), es decir, se irán incrementando de uno en uno cada 5 segundos (aproximadamente), desde 00 hasta 11, al llegar al 11 una vez que transcurran 5 segundos volverá a 00 y seguirá contando normalmente. Una vez que la cuenta ha iniciado, si el usuario vuelve a presionar este botón NO deberá suceder absolutamente nada.

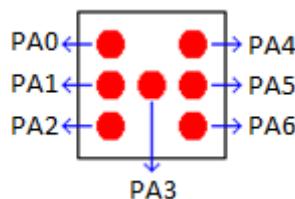
Si la cuenta no ha comenzado y el usuario presiona el botón de RESETEAR, NO deberá suceder absolutamente nada, sin embargo, una vez que la cuenta inicia, si el usuario presiona el botón de RESETEAR los displays deberán aparecer nuevamente en 00 y permanecer detenidos.

Para que el botón de RESETEAR funcione inmediatamente cuando el botón sea presionado, dicho botón deberá ser programado dentro de una rutina de interrupción externa. (No olvide que el botón requiere una resistencia, que puede ser de pull up o de pull down, y presentará rebotes).

DADO DIGITAL [INTERRUPCIONES EXTERNAS]

Configure el puerto A de su microprocesador como puerto de salida, de forma que le conecte siete leds que simularán un dado digital.

Acomode los LED de forma que queden conectados en el orden que se muestra en la siguiente figura.
(También deberá acomodar los leds en el protoboard para que simulen un dado.)



Conecte un push button en el pin necesario para que funcione con la interrupción 0 del microcontrolador. Inicialice su programa sin olvidar configurar el vector de interrupciones, el stack pointer, el puerto de salida, y habilitar las interrupciones.

Utilizaremos el registro R16 para el número que saldrá aleatoriamente en el dado. El programa únicamente deberá estar incrementando constantemente el valor de R16 desde 0x00 hasta 0xFF.

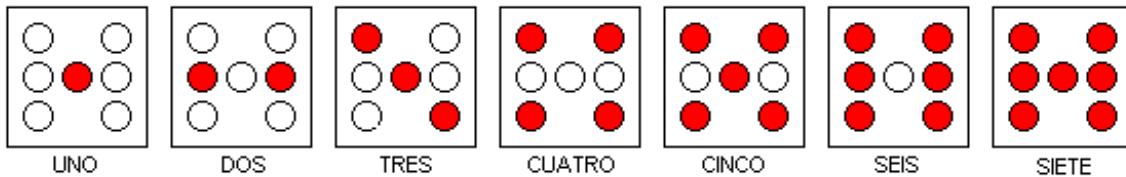
Cabe hacer notar que como nosotros únicamente desplegaremos números del 1 al 7, esto corresponde únicamente a los 3 bits menos significativos de R16, y no nos interesa el contenido de los otros 5 bits más significativos. Es por ello que, si queremos “poner en cero” el contenido de los bits que no nos sirven podemos ocupar la instrucción:

andi R16, 0x0000 0111

que realizará un AND en forma inmediata entre lo que haya en el registro y el número 00000111, así si por ejemplo en el registro hay un 00100011 que no puede ser utilizado en el dado, al momento de realizar la operación andi, el resultado que quedará en R16 será 00000011, que es un número que nos sirve para comparar y mostrar el valor correspondiente en el dado.

Programe la interrupción 0 del microcontrolador, para que al detectar un flanco de bajada del botón, muestre en los LEDs del dado el número que corresponda al contenido los últimos tres bits de R16 en ese momento. (Note que NUNCA SE mostrará el cero, por lo cual si R16 tiene un 0 deberá mostrar un 1)

Los números deberán desplegarse como se muestra en la siguiente figura:

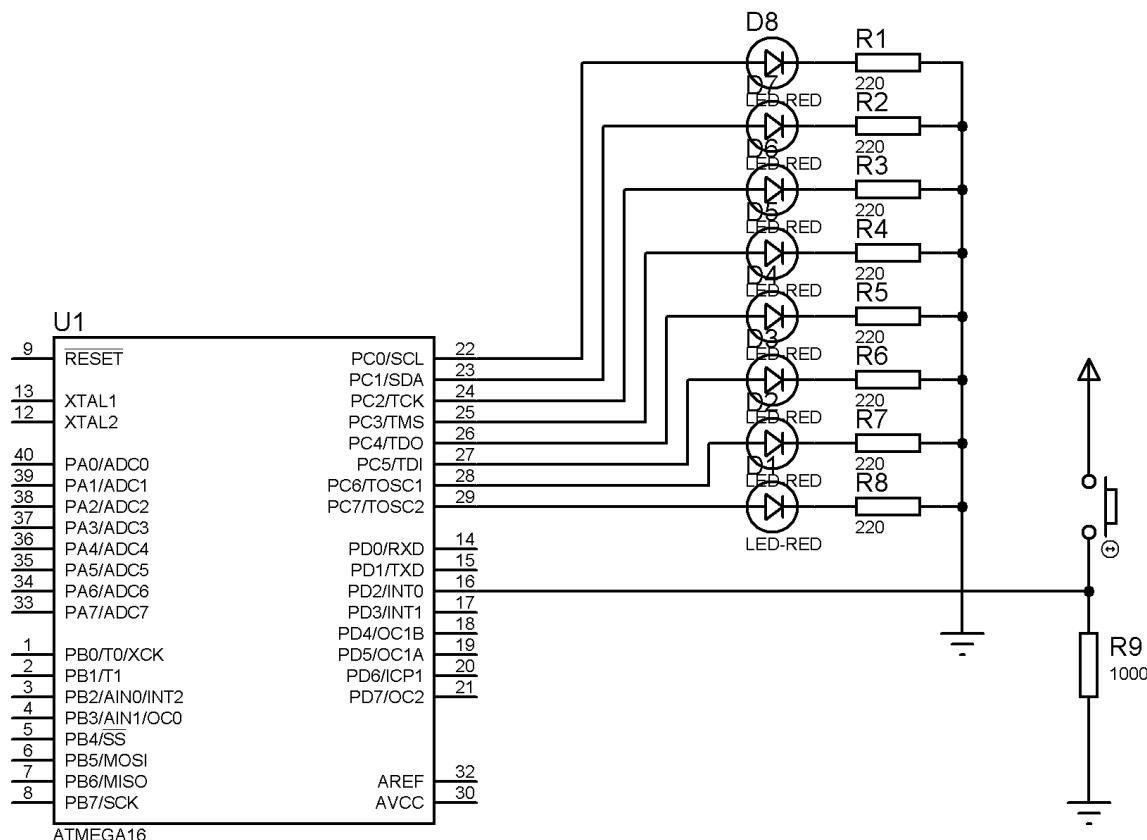


No olvide que el botón que empleará tiene rebote, y es necesario quitarlo para que la interrupción no se genere muchas veces al presionar una sola vez el botón.

SECUENCIA DE LEDS DETENIDA POR INTERRUPCIONES [INTERRUPCIONES EXTERNAS]

Se desea realizar un programa en ensamblador para el microcontrolador ATmega16A, el cual tendrá 8 LEDs conectados en el puerto C. Dichos LEDs se encontrarán encendiendo uno a uno en forma secuencial por un periodo de tiempo de dos segundos. Cuando el botón que se encuentra conectado a la interrupción sea presionado, la secuencia deberá detenerse y los LEDs encenderán y apagarán todos juntos 3 veces por intervalos de dos segundos (3 encendido – 3 apagado) y posteriormente continuarán en la secuencia de encender secuencialmente, retomando ésta en donde se había quedado.

Es requisito indispensable que el botón se encuentre conectado y funcionando a través de uno de los pines de interrupción del microcontrolador.

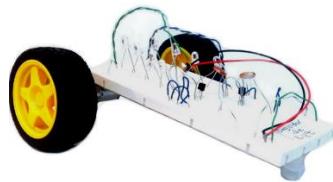


SEGUIDOR DE LÍNEA [INTERRUPCIONES EXTERNAS]

En esta práctica se le pide que realice un vehículo seguidor de línea. Las dimensiones y peso de dicho vehículo podrán ser definidas por cada uno, siempre y cuando funcione adecuadamente. La pista estará marcada por una línea negra (del grosor de una cinta de aislar estándar) sobre una superficie de color blanco. Las curvas más cerradas tendrán un radio de 30 cm.

Realice el diseño necesario para ajustar a su protoboard dos llantas controladas por motores en forma independiente, así como una “rueda loca”.

Realice también el diseño para fijar en este vehículo al menos dos sensores infrarrojos reflexivos, muy cercanos al piso (y los más aislados posible de la luz exterior), al realizar las conexiones para dichos sensores, recuerde verificar que el color blanco lo detecte lo más cercano posible a 0V y el negro lo más cercano posible a 5V (o viceversa).



Esta práctica puede realizarse utilizando entradas normales (como las que se han venido utilizando hasta ahora), sin embargo, con el fin de practicar el tema de interrupciones, es requisito que en esta ocasión las salidas de los sensores deberán encontrarse conectados a las interrupciones del microcontrolador.

Recuerde que la corriente de salida que puede proporcionar cada uno de los pines del microcontrolador es muy pequeña, por lo cual no servirá para mover los motores, de ahí que es necesario utilizar algún circuito intermedio para tomar la corriente directamente de la fuente de alimentación (se sugiere utilizar el mismo circuito empleado en la práctica de motores de pasos).

DISPLAY DE MATRIZ DE PUNTOS [INTERRUPCIONES EXTERNAS]



Realice un programa que en un display de matriz de puntos de al menos 5x7, que muestre por un intervalo de tiempo visible las tres primeras líneas iluminadas, después deberá apagar la primera línea y encender la cuarta, en seguida apagará la segunda y mostrará la quinta, después apagará la tercera y mostrará la sexta, en seguida apagará la cuarta y encenderá la séptima, posteriormente apagará la quinta y encenderá la primera, después apagará la sexta y encenderá la segunda, y entonces comenzará de nuevo la secuencia indicada.

El programa deberá tener conectado un botón en una de las interrupciones, en cualquier momento que el botón sea presionado deberá aparecer una carita feliz en el display (usted elija los LEDs que deberán encender).

El reto de este programa consiste en entender y saber manejar el display de matriz de puntos.

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
SUTO	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 utilizarse 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	MCUCSR
Read/Write	JTD	ISC2	-	JTRF	WDRF	BORF	EXTRF	PORF	
Initial Value	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	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 utilizarse 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 JTGEN 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	SUTO	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 deseé 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 ($\pm 3\%$) 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.

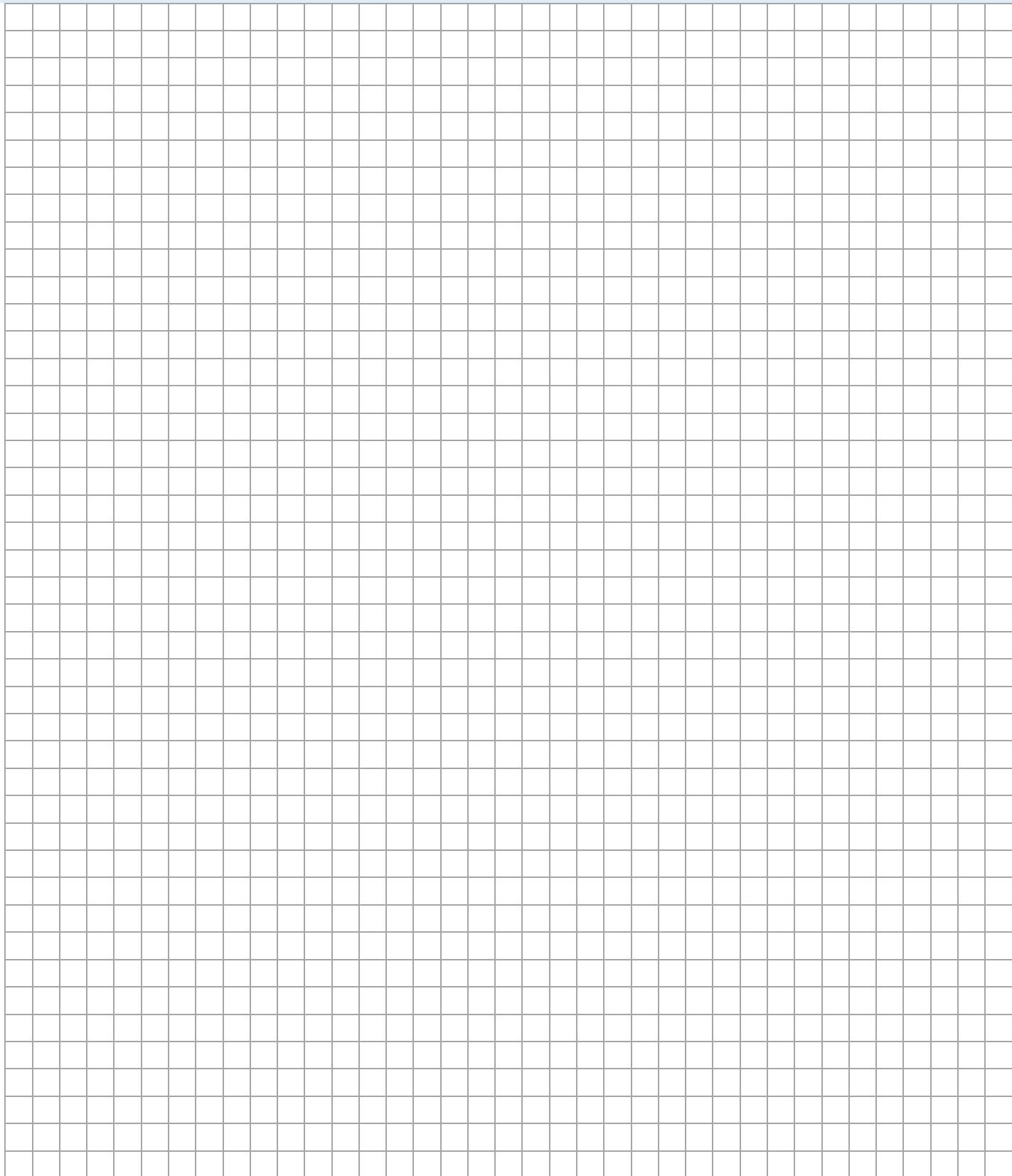
Cuando se selecciona este tipo de oscilador (reloj interno) se puede determinar también el tiempo de inicialización del micro, mediante los fusibles SUT localizados en el byte de fusibles bajo. A continuación se muestra la tabla de los valores que se pueden asignar a los bits SUT.

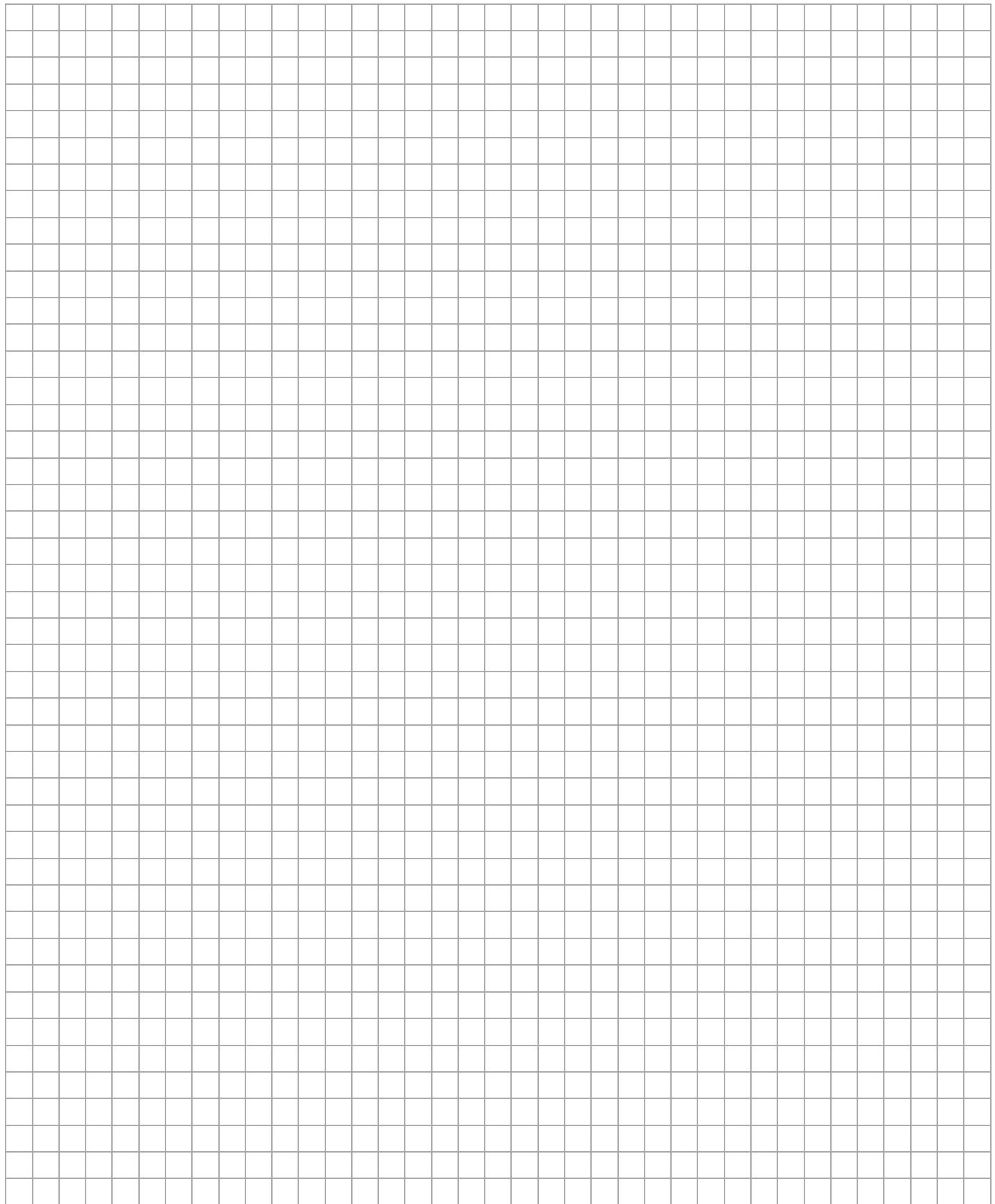
Table 8-9. Start-up Times for the Internal Calibrated RC Oscillator Clock Selection

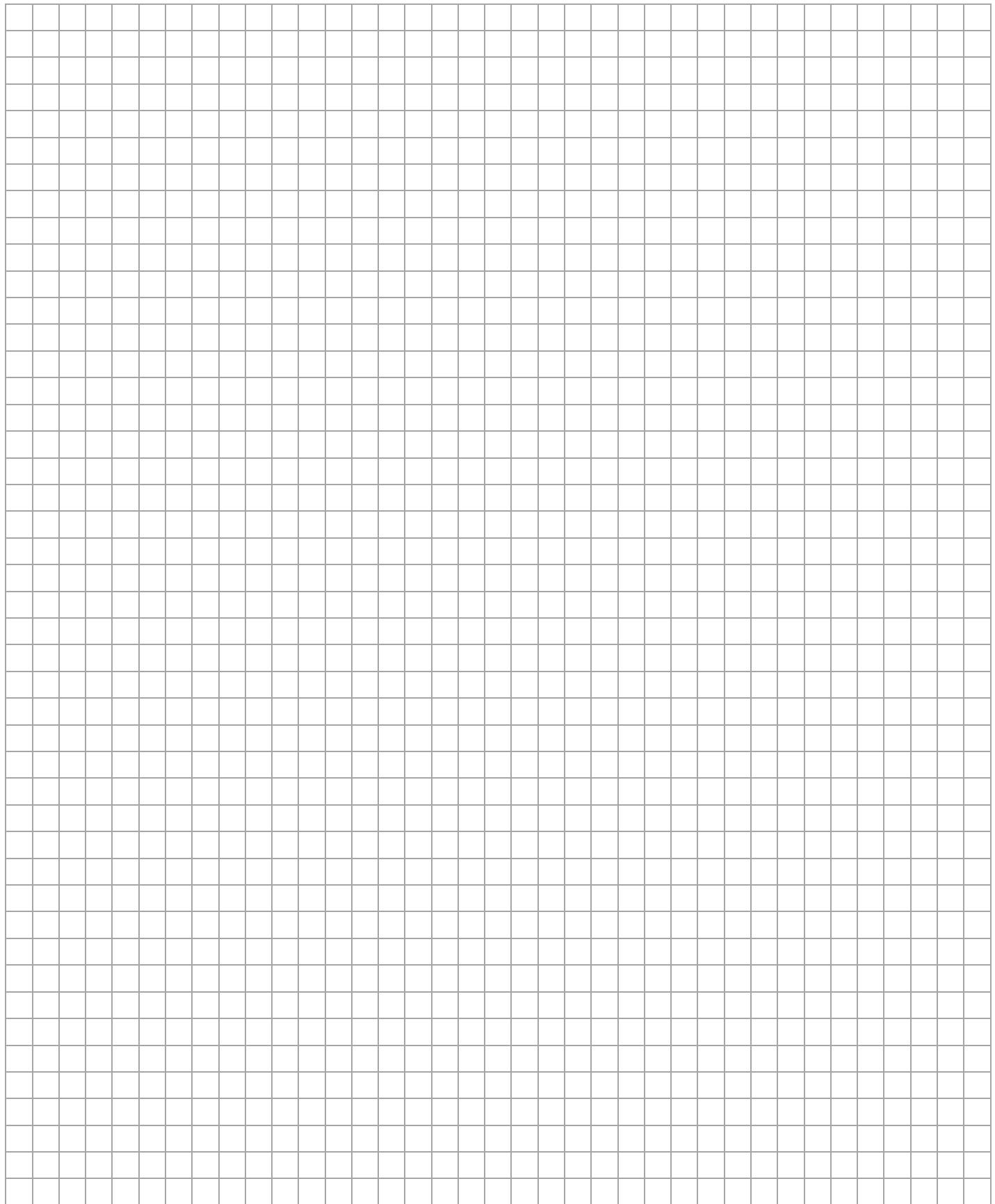
SUT1:0	Start-up Time from Power-down and Power-save	Additional Delay from Reset ($V_{CC} = 5.0V$)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1ms	Fast rising power
10 ⁽¹⁾	6 CK	65ms	Slowly rising power
11	Reserved		

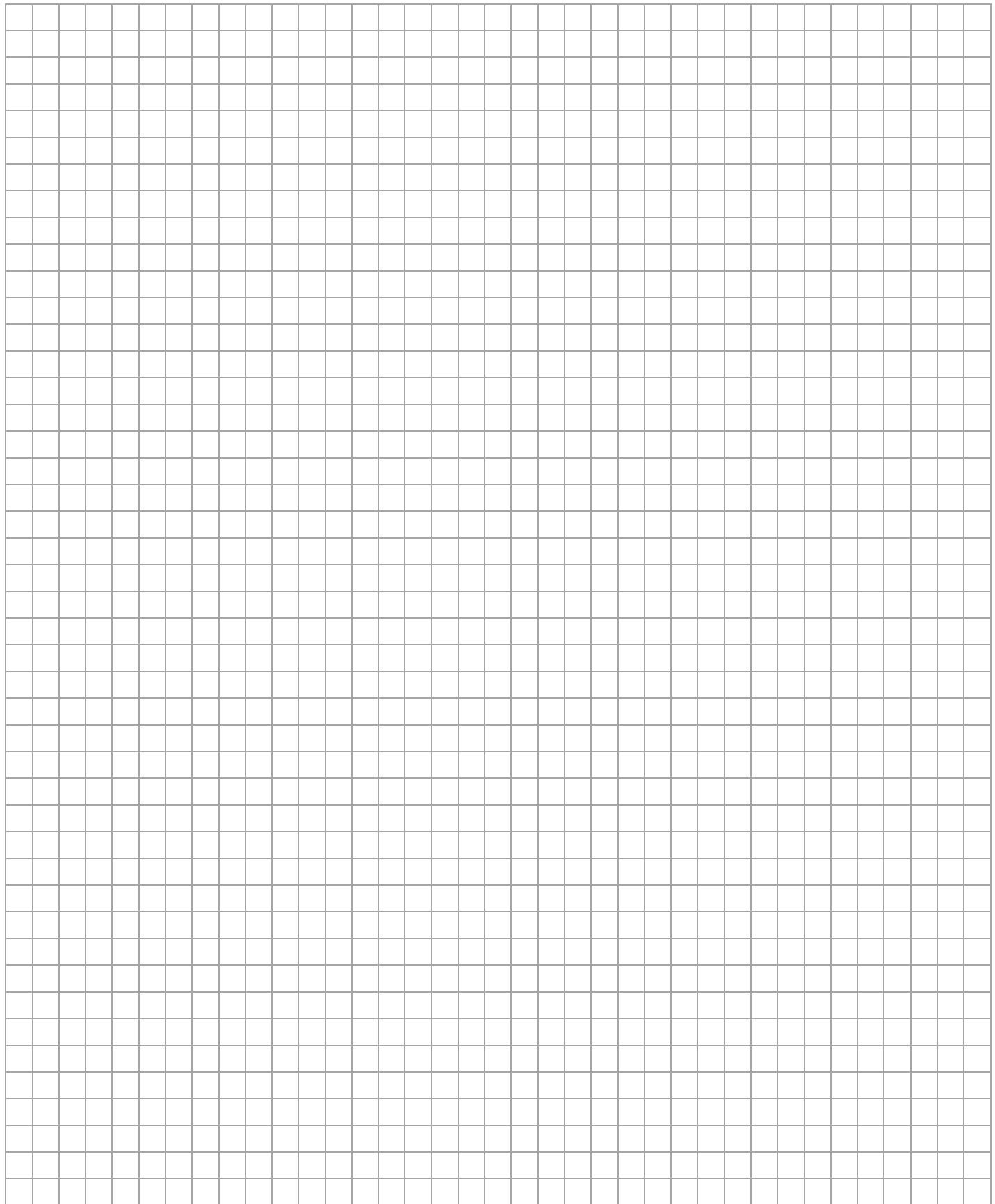
Note: 1. The device is shipped with this option selected.

NOTAS PERSONALES – FUSIBLES DEL MICROCONTROLADOR

A large grid of squares, approximately 20 columns by 30 rows, designed for writing personal notes or sketches related to the topic of fuses in microcontrollers.







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 TCCRO.

REGISTRO TCCRO – TIMER/COUNTER CONTROL REGISTER

Bit	7	6	5	4	3	2	1	0	TCCRO
Read/Write	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	
	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	$\text{clk}_{\text{IO}}/(\text{No prescaling})$
0	1	0	$\text{clk}_{\text{IO}}/8$ (From prescaler)
0	1	1	$\text{clk}_{\text{IO}}/64$ (From prescaler)
1	0	0	$\text{clk}_{\text{IO}}/256$ (From prescaler)
1	0	1	$\text{clk}_{\text{IO}}/1024$ (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 $(1/4\text{MHz} = 0.25\mu\text{s})$ cada ciclo de reloj tarda $0.25\mu\text{s}$, pero como estamos configurando para $\text{clk}/64$, nuestro timer funcionará a $4\text{MHz}/64 = 62.5\text{kHz}$ es decir que ahora el timer se incrementará cada $(1/62.5\text{kHz} = 16 \mu\text{s})$ 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 TCCRO se carga hasta después de haber configurado los otros registros necesarios para el adecuado funcionamiento del timer.

TCCRO

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0		0	0				
FOCO	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 TCCRO se emplea el siguiente código:

*Ldi r16, 0b0 _ 0 0 _ _ _ _
Out TCCRO, 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	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	TIMSK
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, 0b0000000_ _
Out TIMSK, r16*

REGISTRO OCR0 – OUTPUT COMPARISON 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	TIFR
Read/Write	OCF2 R/W	TOV2 R/W	ICF1 R/W	OCF1A R/W	OCF1B R/W	TOV1 R/W	OCF0 R/W	TOV0 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 OCRO 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 overflow 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 OCRO), 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:

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

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 TCCRO.

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:

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

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 TCCRO.

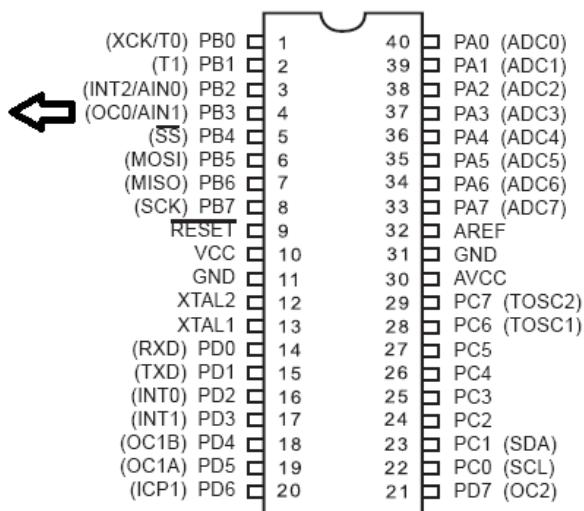
OCRn = Es el valor con que se haya configurado el registro OCR0

Si se desea que el Timer 0 quede configurado en modo normal, pero que los períodos de tiempo por interrupciones de comparación se generen en estos mismos períodos 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

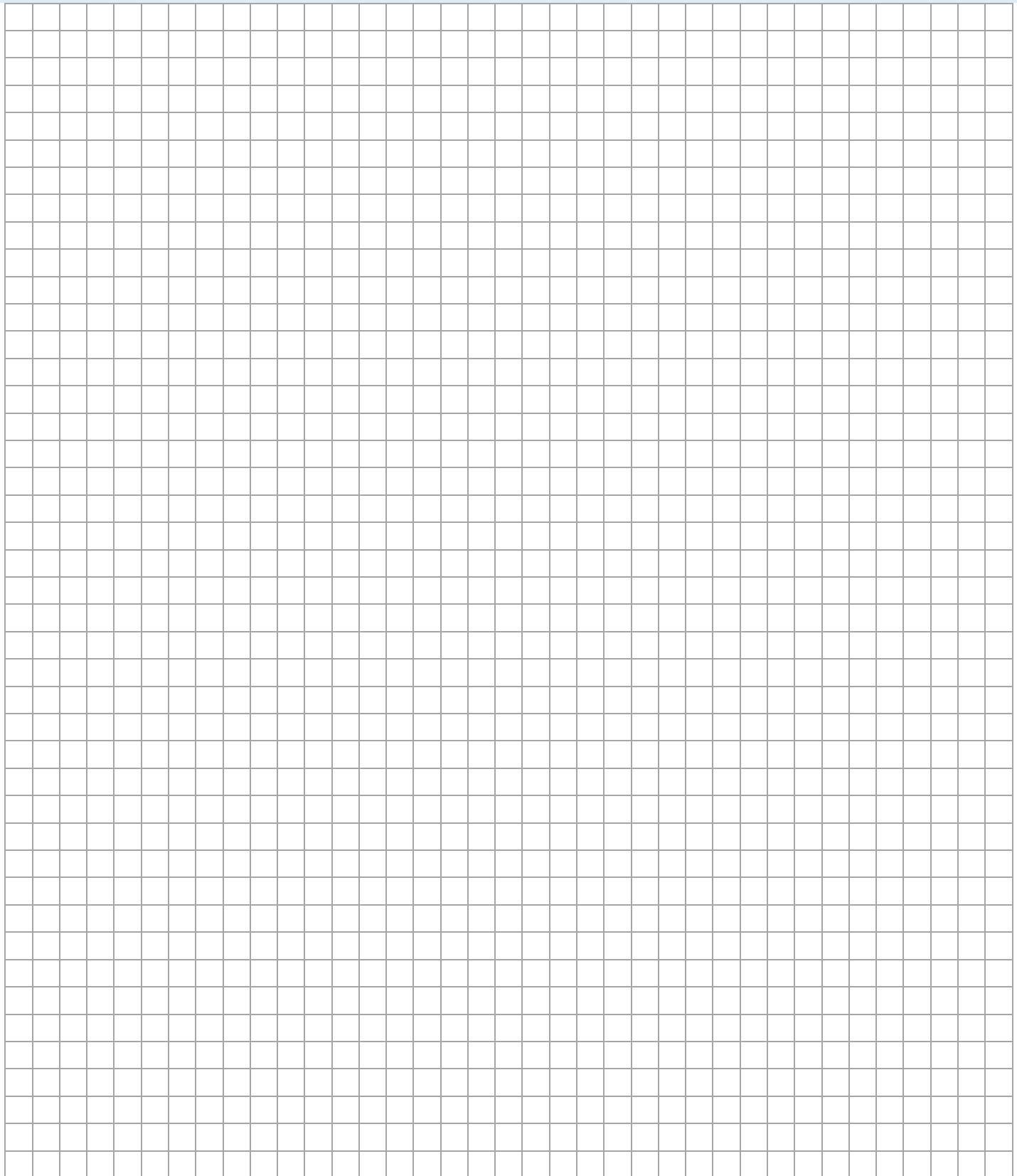
Si COM01..00 = 11 entonces cada vez que TCNT0=OCR0 el contenido del pin OC0 (PB3) automáticamente se pone en 1

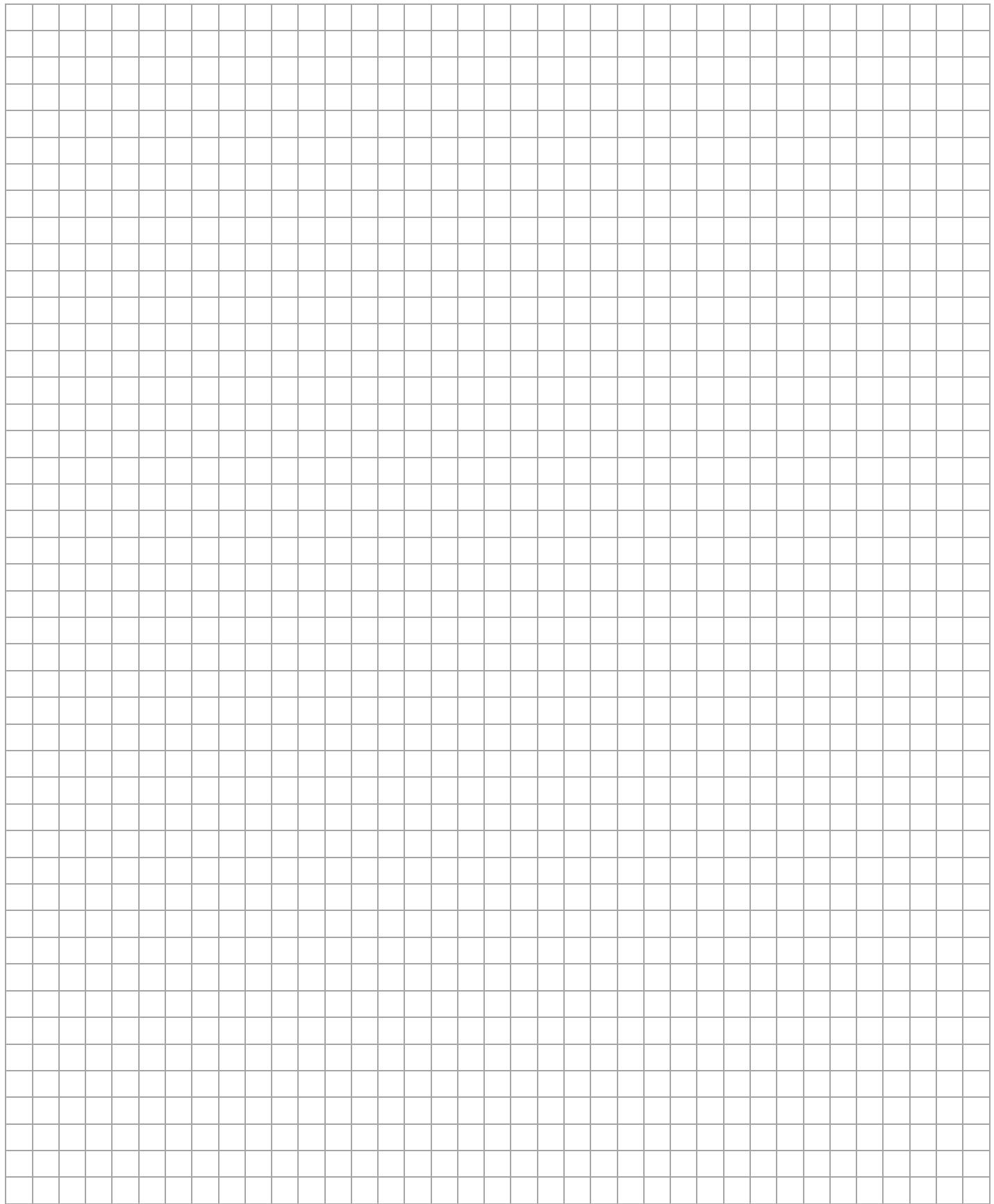
Entonces, si se desea que haya una salida que tenga algún comportamiento especial en forma automática al momento que TCNT0 sea igual que OCR0, es necesario configurar los bits COM01 y COM00 del registro TCCR0.

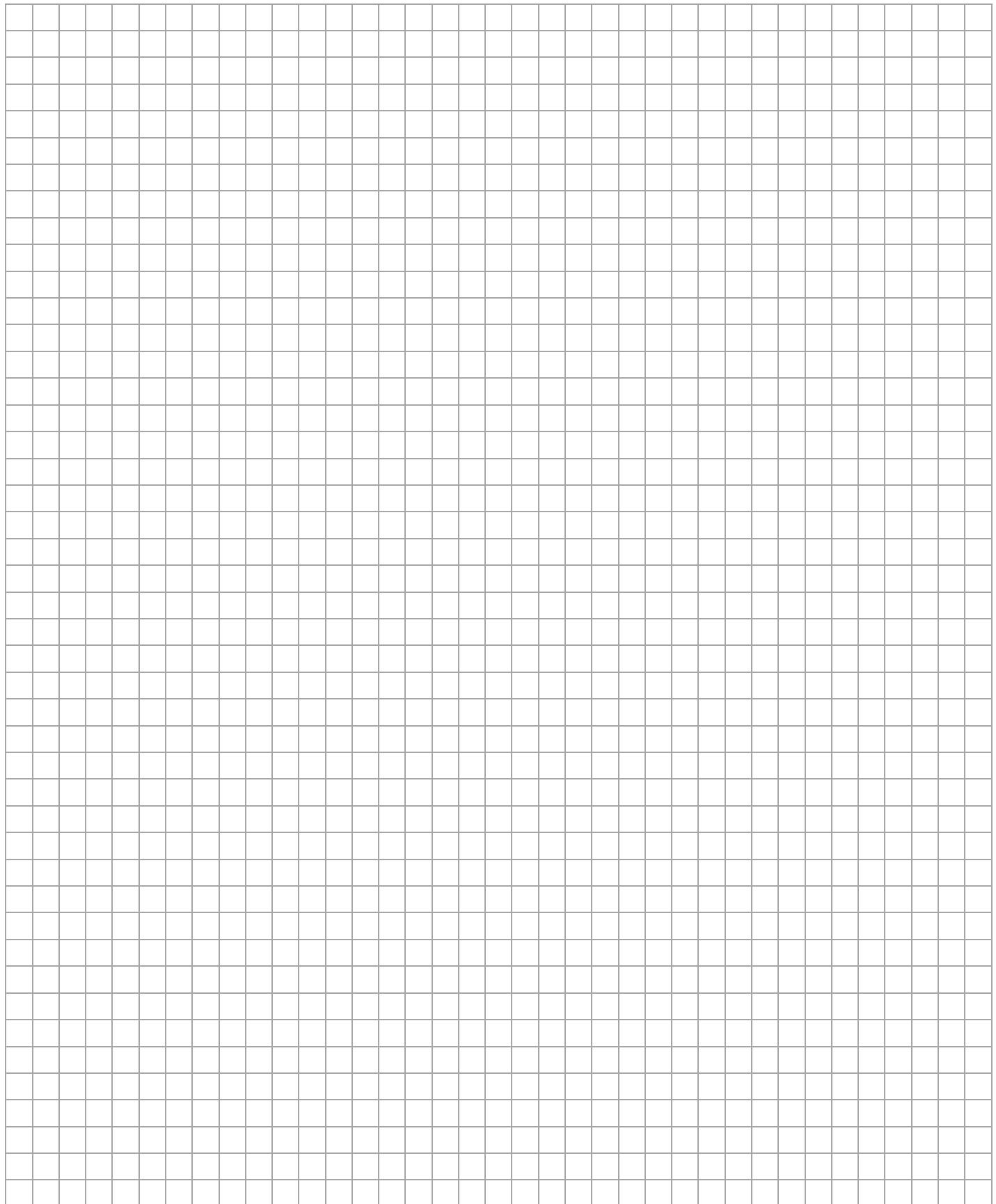
TCCR0

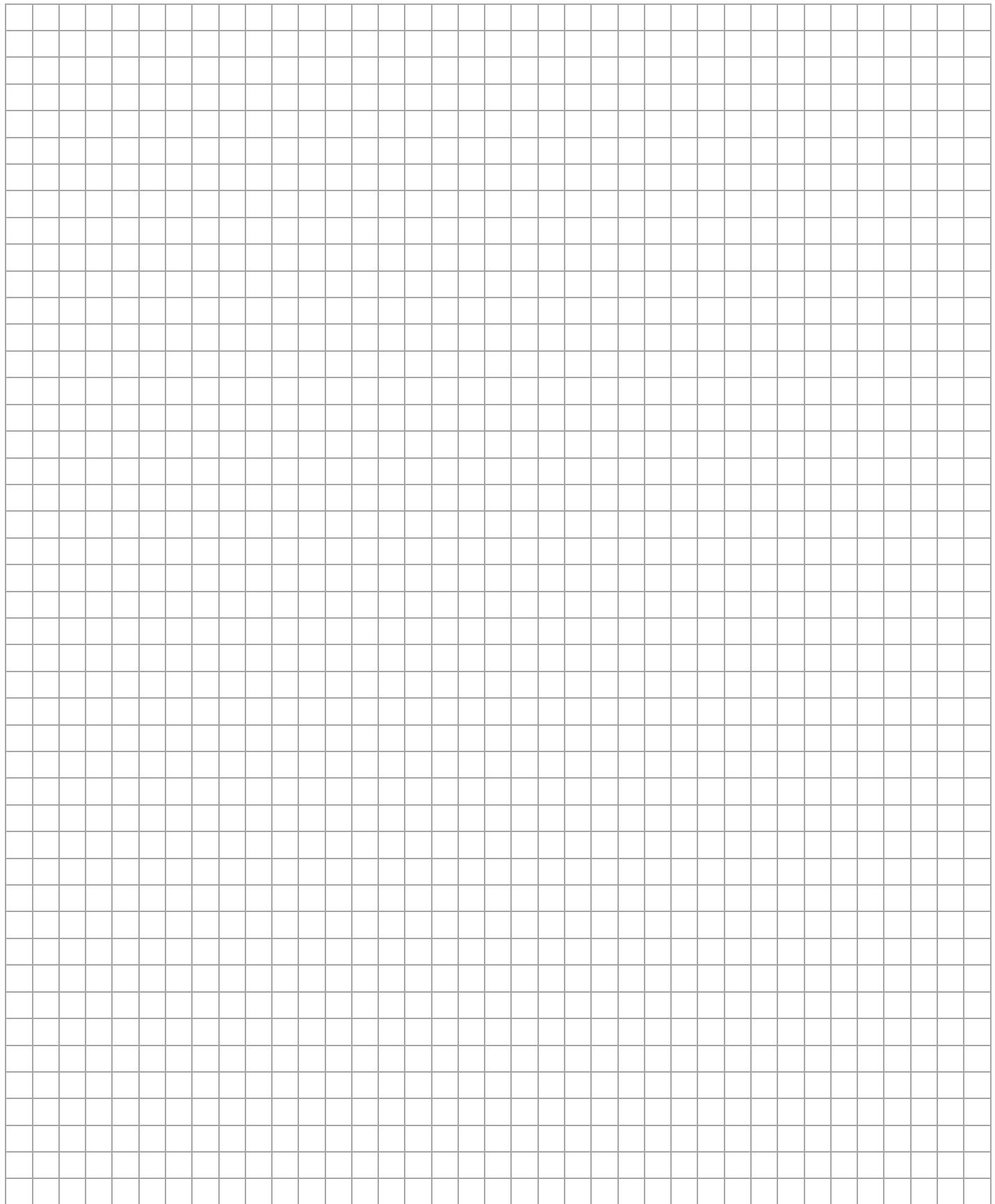
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0							
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
	Ver tabla 39 .WGM01..00 = 10 CTC = 00 Normal	Ver tabla 40 (cuando se desea salida OC0)	Ver tabla 40 (cuando se desea salida OC0)	Ver tabla 39 .WGM01..00 = 10 CTC = 00 Normal	Ver tabla 43 (prescaler)	Ver tabla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

NOTAS PERSONALES – TIMERO MODO NORMAL Y MODO CTC

A large grid of squares, approximately 20 columns by 25 rows, designed for taking notes or drawing diagrams.



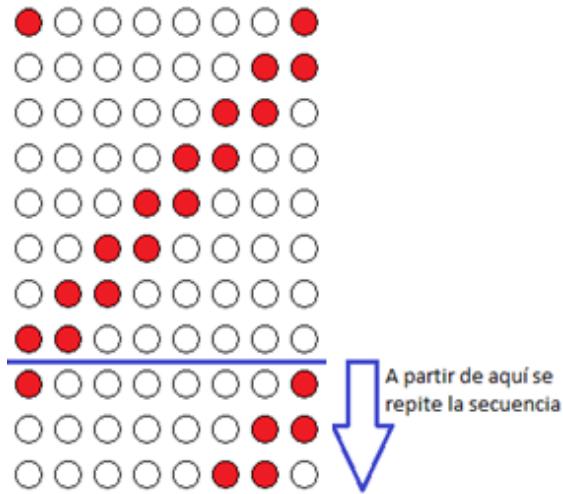




SECUENCIA DE LEDS [TIMERO MODO NORMAL Y MODO CTC]

Utilizando un microcontrolador ATmega16A diseñe un programa en el cual conectará 8 LEDs en uno de los puertos del microcontrolador y dos botones en otro de los puertos.

Al comenzar el programa los LEDs deberán presentar la siguiente secuencia, realizando los cambios cada segundo (el tiempo deberá ser controlado utilizando el timer0 en modo CTC)



Cada vez que el usuario presione el botón 0 el tiempo de cambio deberá ir disminuyendo de la siguiente forma (a partir del tiempo en el que se encuentre)

4 s → 3 s → 2 s → 1 s → 0.5 s → 0.25s

Cada vez que el usuario presione el botón 1 el tiempo de cambio deberá ir aumentando de la siguiente forma

(a partir del tiempo en el que se encuentre)

0.25 s → 0.5 s → 1 s → 2 s → 3 s → 4 s

Al entregar esta práctica se deberá mostrar el código en la computadora (garantizando el uso del timer 0 para la generación de los tiempos).

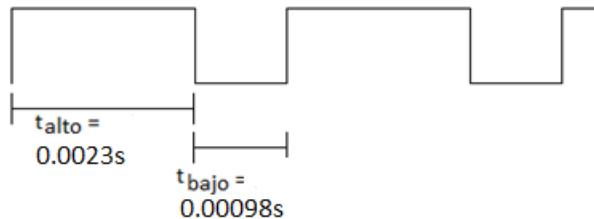
GENERADOR DE UNA SEÑAL [TIMERO MODO NORMAL Y MODO CTC]

Para este programa se requiere conectar un botón en el bit menos significativo del puerto A y una punta de osciloscopio en el bit más significativo del puerto A.

Al principio del programa la señal por el bit más significativo del puerto A deberá permanecer en 0V. Cuando el usuario presione el botón A el bit más significativo del puerto A deberá presentar una señal con la frecuencia que se muestra en el siguiente diagrama de tiempos (sólo se generará la frecuencia mientras el botón se encuentre presionado, al momento en que el botón sea liberado entonces la señal deberá apagarse y permanecer así hasta que se vuelva a presionar el botón).

Se pide que configure el timer0 de su microcontrolador, utilizando el modo de funcionamiento normal. Elija la frecuencia de operación del microcontrolador y el prescaler que más se adapte a sus necesidades.

Debe considerar que cada instrucción que ejecuta el microcontrolador le toma determinado número de ciclos de reloj, es por ello que a pesar de que usted configure sus timers en forma exacta, al final habrá que hacer ajustes para lograr los tiempos precisos.



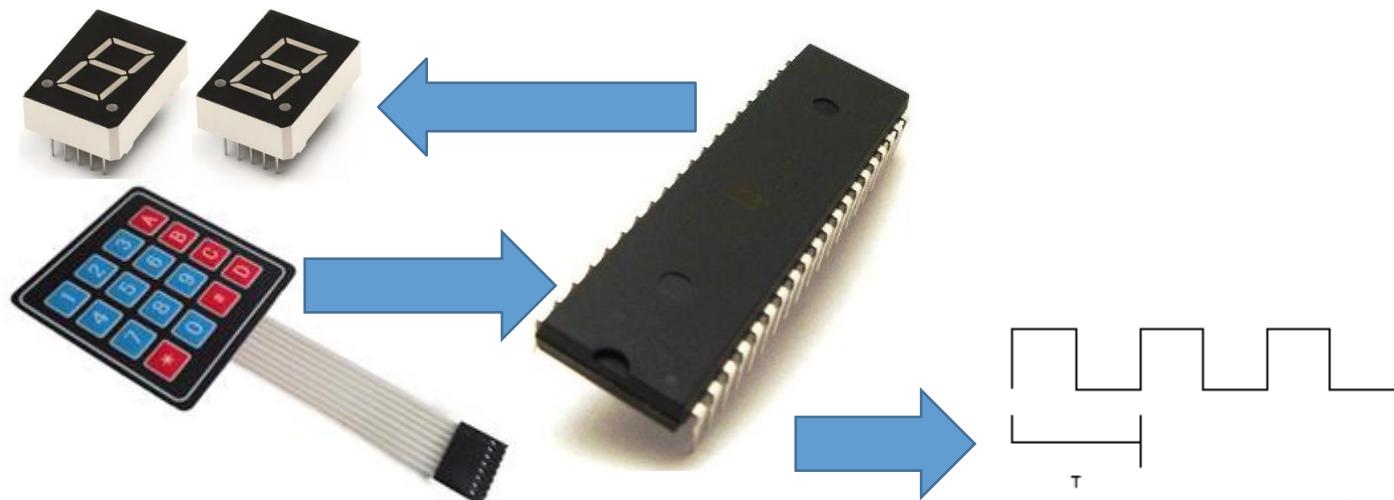
GENERADOR DE SEÑALES [TIMERO MODO NORMAL Y MODO CTC]

El programa a realizar requiere el uso del teclado matricial conectado en alguno de los puertos del microcontrolador. También deberá conectar dos displays de siete segmentos (usted tiene la libertad de decidir si esta conexión la realiza utilizando el 74LS47 o bien si los conecta directamente al microcontrolador). Un osciloscopio se encontrará conectado en uno de los pines del microcontrolador por el cual saldrá la señal generada.

Al encender el dispositivo ambos displays se mostrarán en 00 y por la salida la señal deberá mantenerse en 0V. Entonces el usuario comenzará a introducir a través del teclado valores seguidos de la tecla “D” la cual equivaldrá a un enter. El primer valor que se introduzca deberá aparecer en el display de la derecha (por ejemplo si el usuario introduce un 5 deberá mostrarse un 05 en los displays). Si el usuario decide introducir otro valor entonces el valor del display de la derecha pasará al display de la izquierda, y el valor introducido aparecerá en el display de la derecha (por ejemplo si el usuario introduce un 1 deberá mostrarse un 51 en los displays). Después de haber introducido dos valores, si el usuario presiona cualquier otra tecla numérica no deberá suceder nada; sin embargo en cualquier momento puede presionar la tecla “C” la cual deberá hacer que ambos displays vuelvan a mostrar 00.

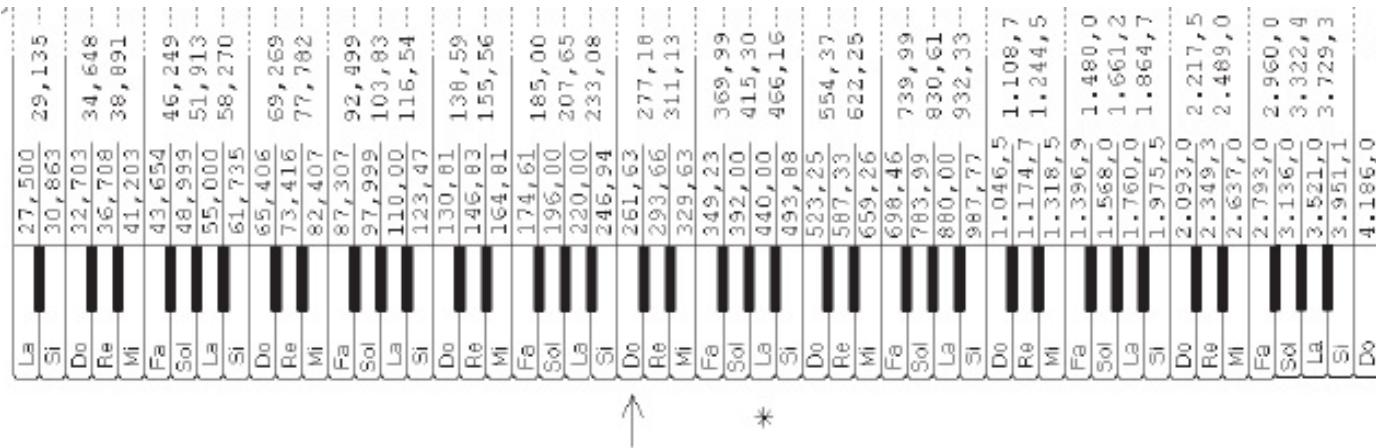
Durante todo el proceso de introducir valores, la salida por la cual se muestra la señal generada deberá mantenerse en lo que se tenía antes de empezar a cambiar los valores. Cuando el usuario presione la tecla “D” entonces, en el osciloscopio conectado al pin de salida de señal, deberá mostrarse una señal cuadrada, cuyo periodo T corresponderá al tiempo en microsegundos que el usuario haya introducido. (Cabe recordar que si el tiempo introducido es 00, la señal que salga deberá ser constante de 0V).

Para esta práctica se pide configurar el reloj del microcontrolador para trabajar en una frecuencia DIFERENTE a 1Mhz.



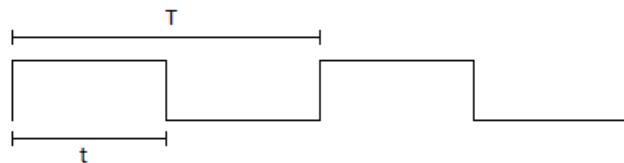
NOTAS MUSICALES [TIMERO MODO NORMAL Y MODO CTC]

Deberá conectar 7 botones a los pines de uno de los puertos del microcontrolador ATmega16A.



Como usted sabe, cada nota musical es producida por una onda sonora que se caracteriza por tener determinada frecuencia. Esta frecuencia puede ser generada a través de un pulso cuadrado de forma que al conectar una bocina como salida a uno de los pines del microcontrolador (se sugiere utilizar el pin B3) podrá escucharse su sonido correspondiente.

Por ejemplo, la nota DO (que se muestra con una flecha en el diagrama) se producirá con una frecuencia de 261.63Hz, es decir con una salida que tenga un periodo $T=0.00382219s$

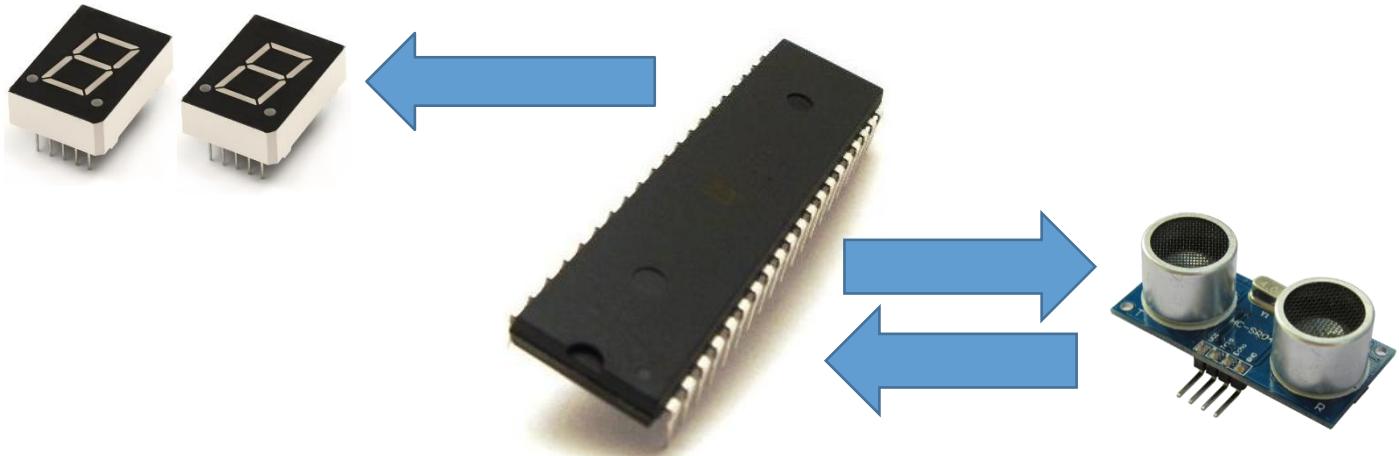


En esta práctica se le pide que cada uno de los botones corresponda a DO, RE, MI, FA, SOL, LA, SÍ (respectivamente), de forma que mientras el botón permanezca presionado se genere en forma constante el sonido que le corresponde. Cuando los botones se sueltan deberá haber silencio.

MEDIDOR DE DISTANCIA [TIMERO MODO NORMAL Y MODO CTC]

Se desea diseñar un dispositivo (a través del uso de un sensor ultrasónico), dicho dispositivo deberá mostrar en dos displays de siete segmentos la distancia que existe (medida en centímetros) entre él y los objetos que se encuentran a su alrededor. Las distancias que deberá mostrar serán hasta un máximo de 99cms. En caso de que se encuentre un objeto a una distancia mayor o igual a 1 metro los displays deberán permanecer apagados.

La distancia medida deberá actualizarse automáticamente en forma constante.



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.

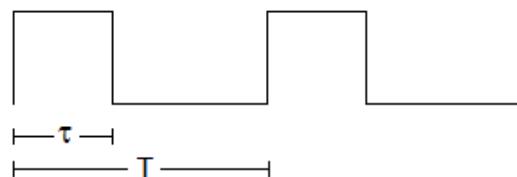
Es decir:

$$D = \frac{\tau}{T}$$

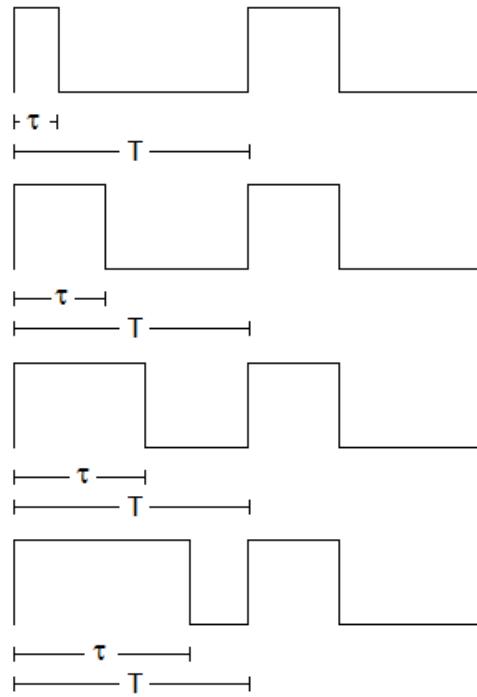
Donde D = Ciclo de trabajo

τ = 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).

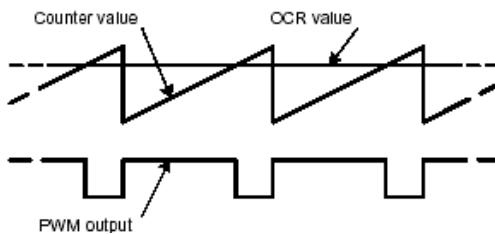
Bit	7	6	5	4	3	2	1	0	TCCR0
Read/Write	W	R/W							
Initial Value	0	0	0	0	0	0	0	0	

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:

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

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
2Mhz	0.0000005	1024	0.262144	3.814697
		1	0.000128	7812.5
		8	0.001024	976.5625
		64	0.008192	122.0703
4Mhz	0.00000025	256	0.032768	30.51758
		1024	0.131072	7.629395
		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 OCRO, 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.

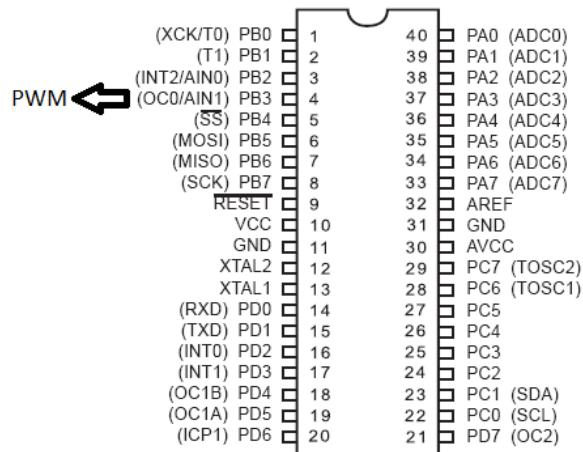
Otros bits que son importantes para la configuración del “Fast PWM” el COM01 y el COM00, los cuales se encargan de indicar si la señal que se generará será normal, o invertida.

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

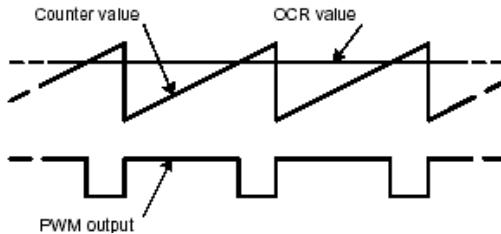
COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on Compare Match, set OC0 at TOP (Non-Inverting).
1	1	Set OC0 on Compare Match, clear OC0 at TOP (Inverting)

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.

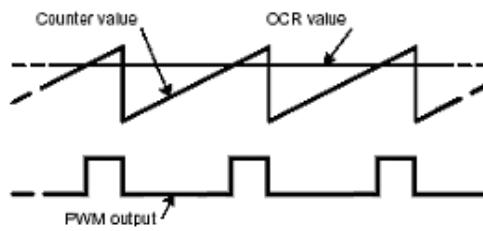
En caso de que los pines COM01 y COM00 hayan sido configurados, también es necesario habilitar el pin correspondiente a OC0 como pin de salida, es decir hay que configurar el pin3 del puerto B como salida.



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



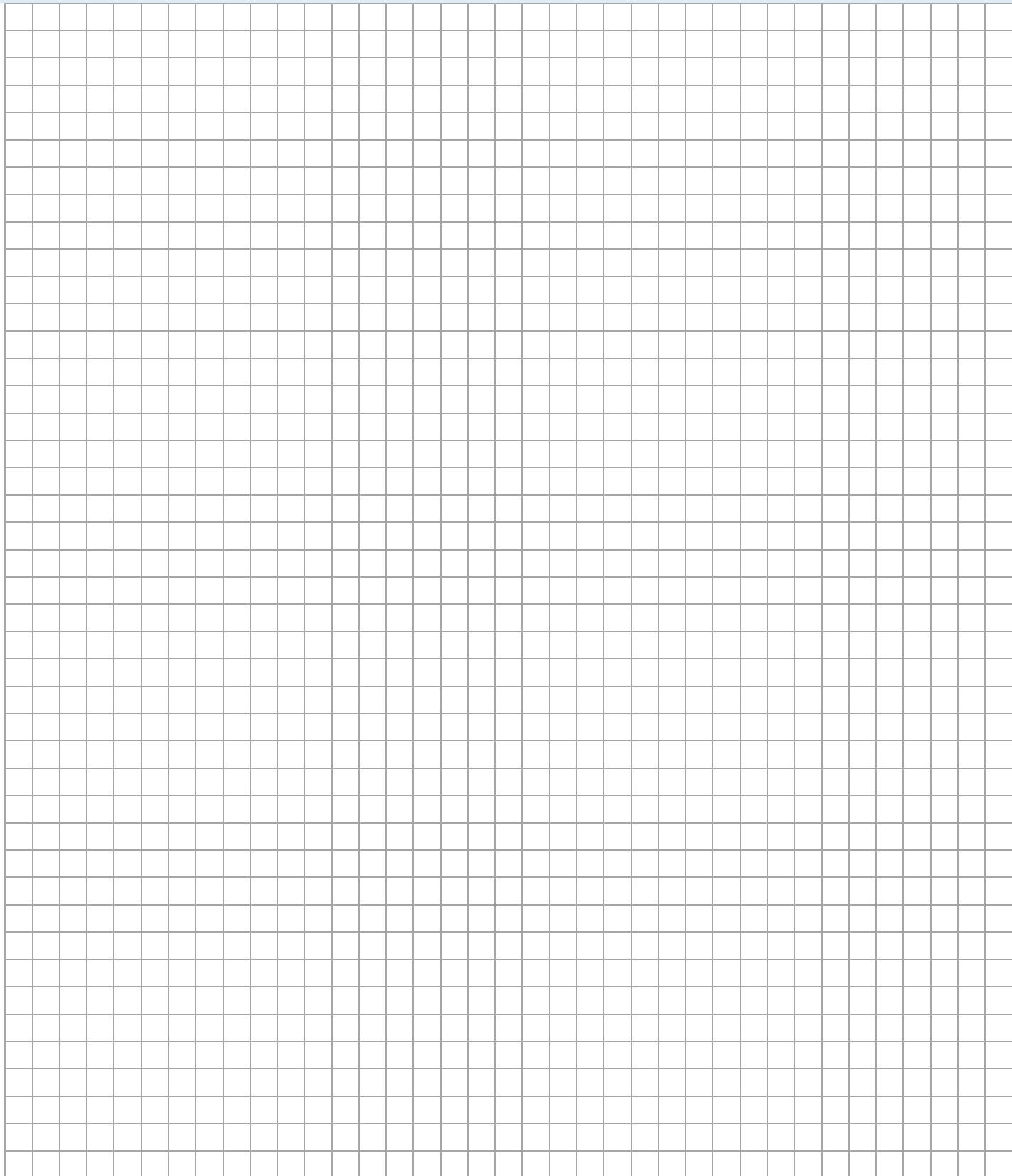
En cambio, si configuramos el COM01:0 con el valor 11, cuando TCNT0 coincida con el valor de OCRO, la señal de PWM cambiará a valor alto y permanecerá así hasta que el contador vuelva a comenzar en 0. La siguiente figura ilustra este funcionamiento:

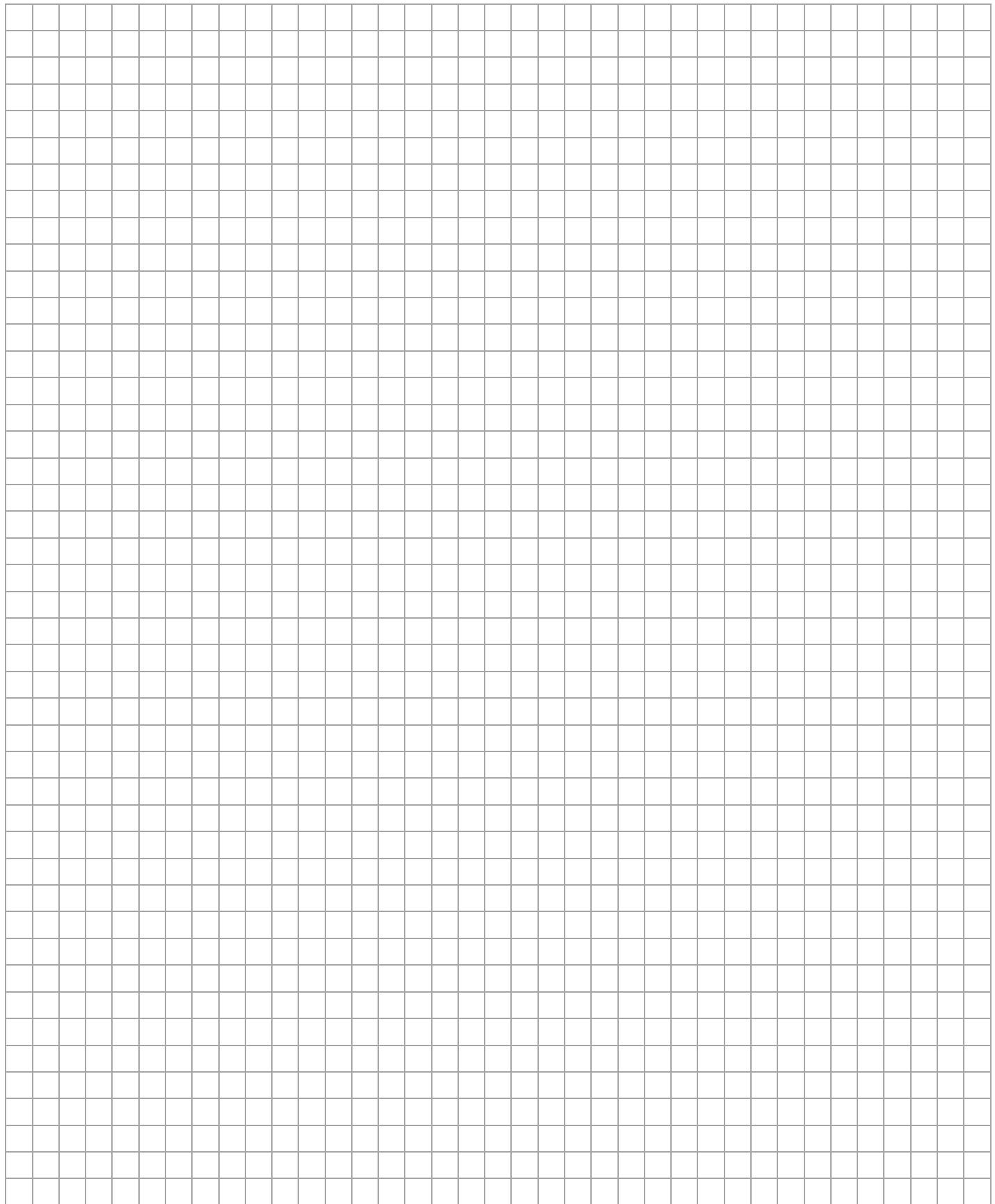


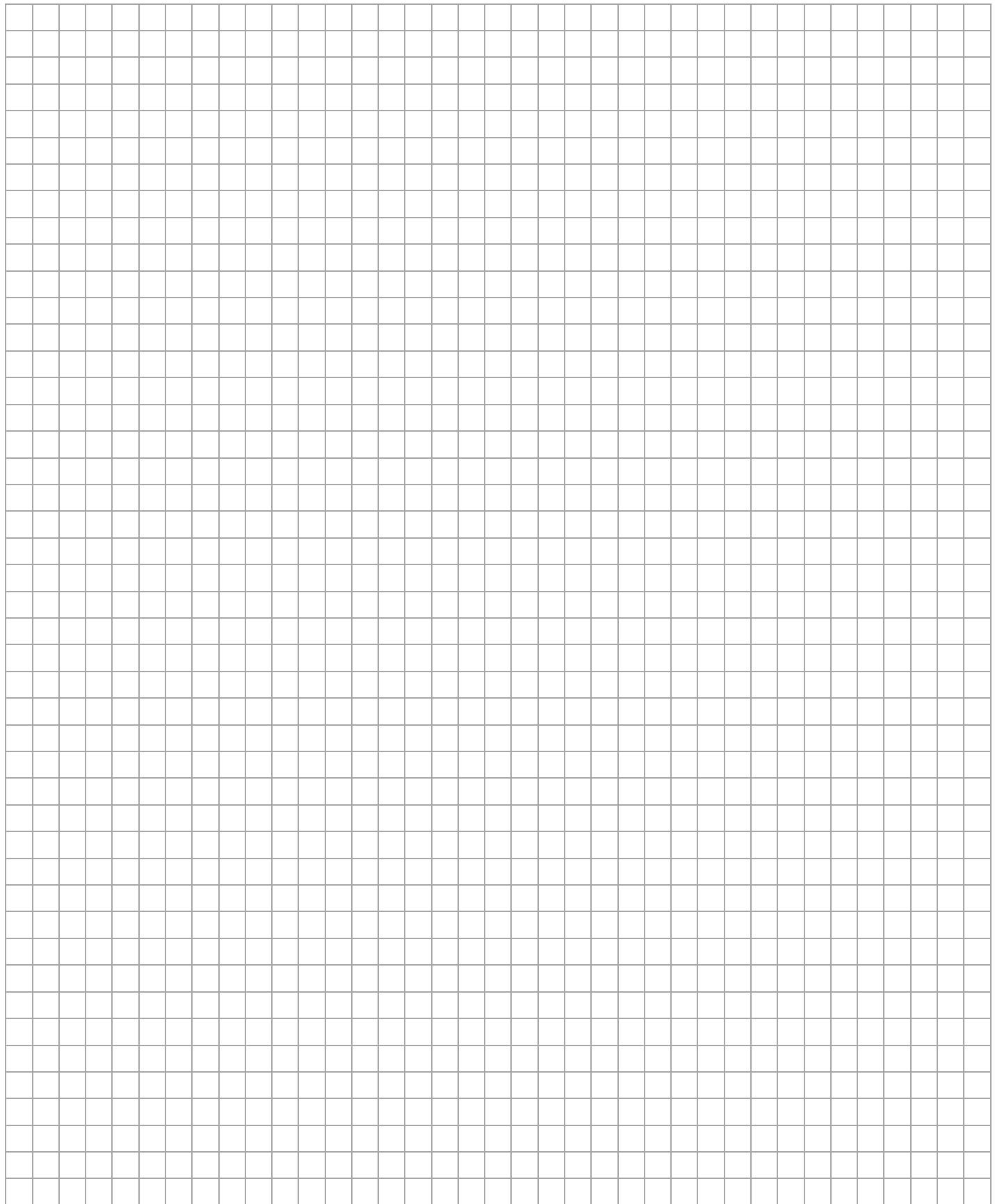
TCCRO

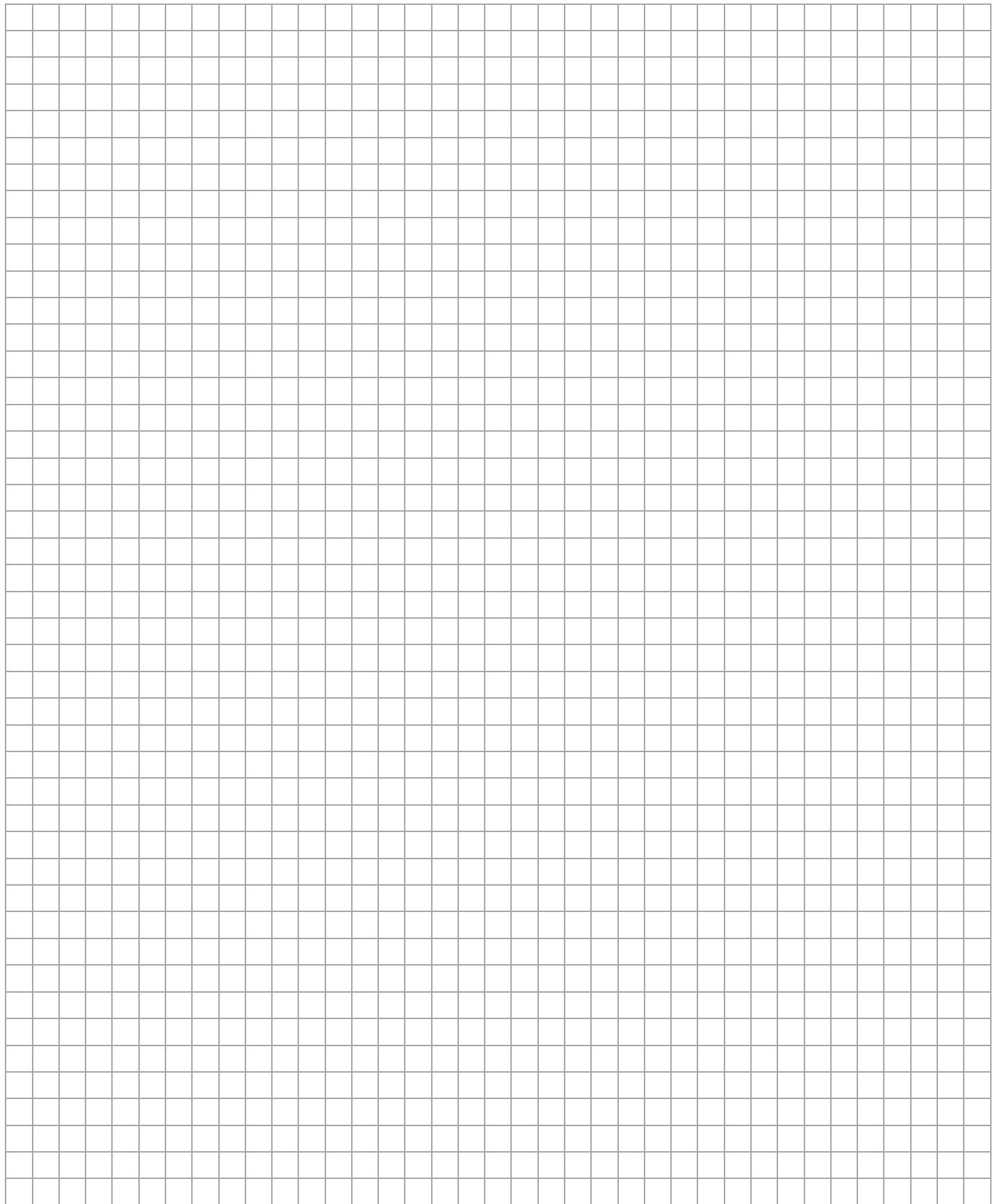
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	1			1			
FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
	Ver tabla 39 (11 en WGM01..00 es FastPWM)	Ver tabla 41 (10 o 11 para que salga señal PWM por OC0)	Ver tabla 41 (10 o 11 para que salga señal PWM por OC0)	Ver tabla 39 (11 en WGM01..00 es FastPWM)	Ver tabla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

NOTAS PERSONALES – TIMERO MODO FAST PWM

A large grid of squares, approximately 20 columns by 25 rows, designed for writing personal notes or sketches.







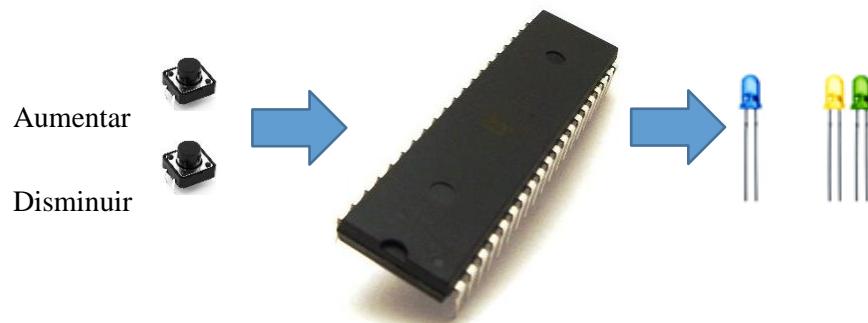
CONTROL DE INTENSIDAD DE UN LED [TIMERO MODO PWM]

En esta práctica se controlará la intensidad de luz de un LED mediante una señal de PWM, para ello se requiere conectar un LED en el bit correspondiente a la salida de PWM del microcontrolador (Puerto B, pin3) (el cual deberá estar apagado al momento de encender el dispositivo). Se conectarán dos “push button” en los bits menos significativos del puerto A, los cuales deberán ser utilizados de la siguiente manera:

- Botón 1: al ser presionado irá aumentando la intensidad de luz del LED (el pwm deberá permanecer en nivel alto por más tiempo)
- Botón 2: al ser presionado irá disminuyendo la intensidad de luz del LED (el pwm deberá permanecer en nivel bajo por menos tiempo)

Se conectarán también 2 LEDs adicionales en los dos bits más significativos del puerto A, cuya finalidad será indicar los momentos en que se hayan alcanzado la intensidad máxima y la intensidad mínima.

- LED 1: permanecerá apagado a menos que el LED se encuentre encendido a su máxima capacidad. Después de que enciende, si presionamos el botón 2 deberá apagarse nuevamente.
- LED 2: permanecerá apagado a menos que el LED se encuentre encendido en su mínima capacidad. Después de que enciende, si presionamos el botón 1 deberá apagarse nuevamente.

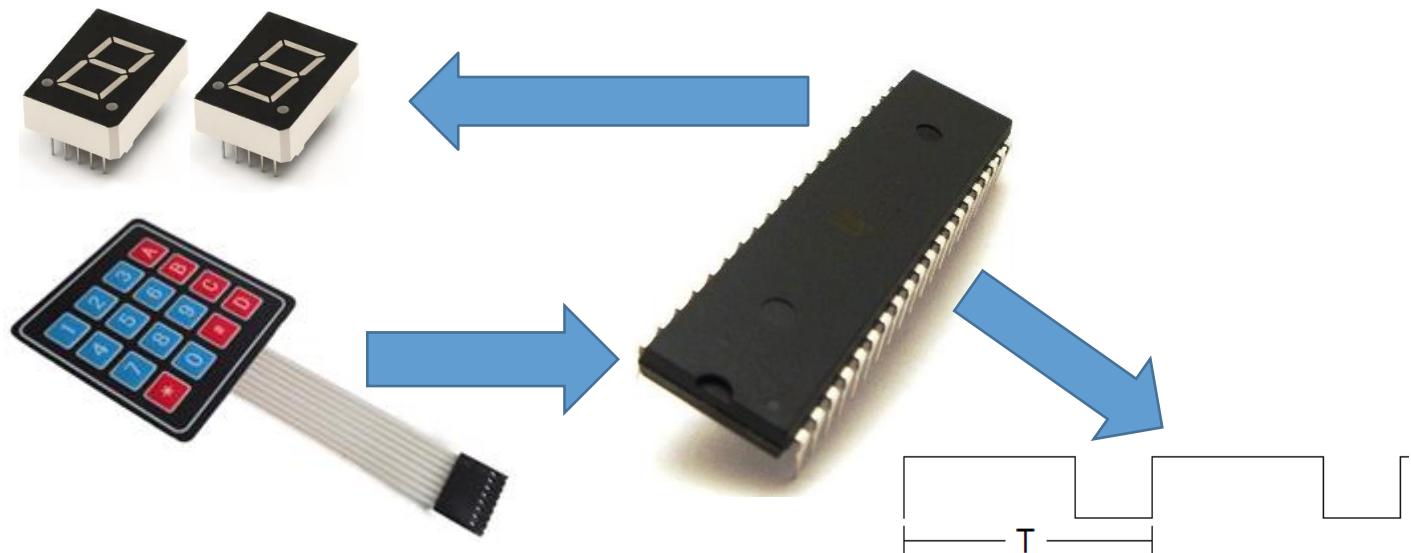


CONTROL DE GENERADOR DE PWM [TIMERO MODO PWM]

Se deberán conectar al microcontrolador el teclado matricial y dos displays de 7 segmentos (decenas y unidades). Al encender el dispositivo los displays deberán mostrar 00 y en la salida de PWM deberá haber una señal constante de 0 Volts.

Mediante el teclado, el usuario podrá introducir valores, el primer número presionado deberá aparecer en el valor de las decenas, y entonces el valor de las unidades deberá quedarse apagado en espera de que se presione su valor. Una vez que se han introducido las decenas y las unidades, por la salida de PWM deberá salir una señal cuyo ciclo de trabajo corresponda al porcentaje que se introdujo en los displays. (sugerencia... si el PWM tiene un ciclo de trabajo de 0% cuando el OCR0 está configurado en 0 y tiene un ciclo de trabajo de 100% cuando el OCR está configurado en 255, puede considerar para fines de esta práctica que el 100% lo alcanza cuando está configurado en 250, de forma tal que si consideramos el valor que se presenta en los displays y lo llamamos DISPLAY, entonces ($OCRO = DISPLAY + DISPLAY + DISPLAY/2$), recuerde que en ensamblador no hay una operación para realizar divisiones como tales, sin embargo a través de un corrimiento específico es posible dividir un número a la mitad.)

Una vez introducido el valor, y en cualquier momento que el usuario lo desee, podrá volver a presionar los números del teclado, para introducir un nuevo ciclo de trabajo, y nuevamente después de que se introduzcan las unidades el valor de PWM en la salida deberá cambiar acorde a lo solicitado por el usuario.



DISPOSITIVO TEMPORIZADOR [TIMERO MODO PWM]

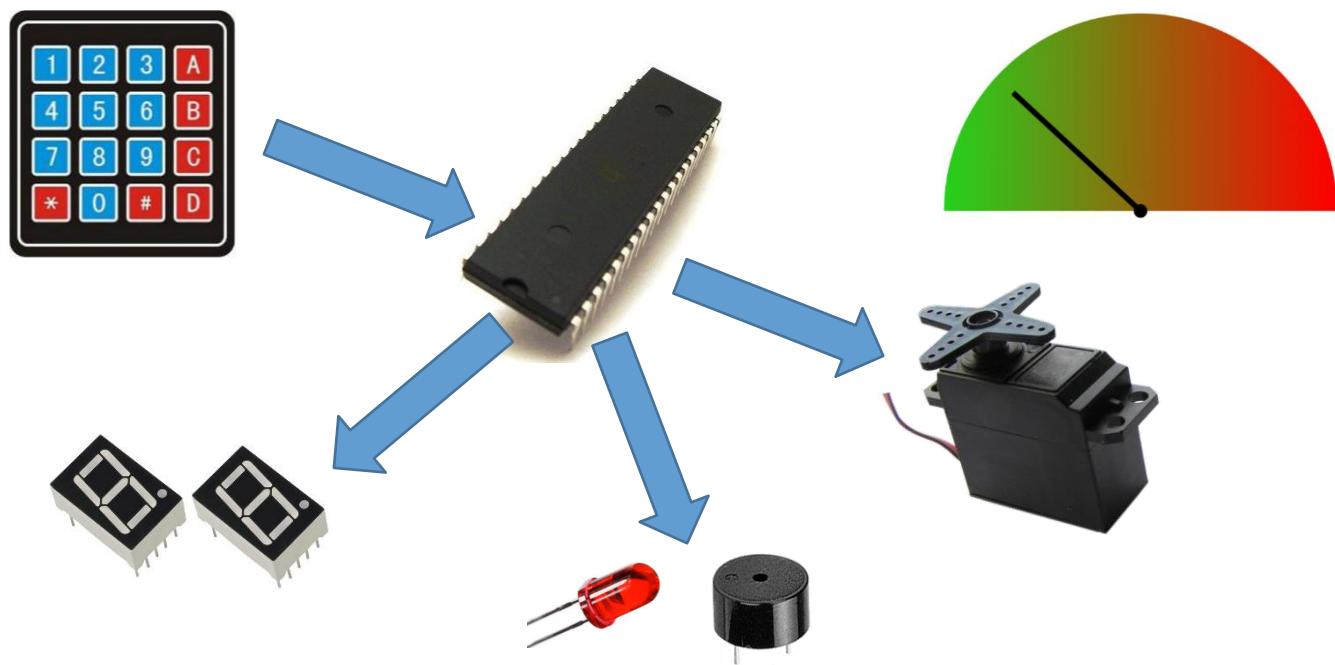
Para comenzar a plantear esta práctica primero será necesario que respondas las siguientes preguntas

- 1) ¿Con qué frecuencia de trabajo del microcontrolador puedo obtener mayor precisión para el giro del mismo?
- 2) ¿Aproximadamente qué cantidad de posiciones diferentes entre 0° y 180° puedo lograr trabajando a esa frecuencia?
- 3) Utilizando la frecuencia definida en la primera pregunta genera una rutina **DELAY** (Retardo) que consuma 0.25 segundos (una cuarta parte de un segundo), dicha rutina deberá crearse de forma que pueda ser llamada con la instrucción **RCALL DELAY** (por tanto, deberá terminar con un **RET**).

DISPOSITIVO

Se desea crear un dispositivo especial para controlar límites de tiempo en niños de edad prescolar, de forma que ellos puedan ir viendo con claridad una aguja que avanza desde un punto inicial hasta un punto final, momento en el cual el tiempo que se les haya asignado para una tarea habrá concluido. Para ello se desea conectar al microcontrolador un teclado matricial, dos displays de siete segmentos y un servomotor que tendrá conectado en émbolo un “indicador”.

Al encender el dispositivo la aguja deberá moverse hasta el lado derecho, como si el tiempo hubiese concluido, los dos displays deberán aparecer mostrando 00. Entonces el usuario podrá introducir a través del teclado matricial dos números (entre 0 y 9), los cuales se mostrarán en los displays, será importante aclarar que siempre el dispositivo esperará a tener ambos, por lo que si sólo se quisiera introducir un 2 el usuario deberá poner 02. Una vez que ambos números se hayan introducido podrá presionarse la tecla D (si es presionada antes deberá ser ignorada), en ese momento la aguja se recorrerá totalmente hasta la izquierda (a su posición de 0° y deberá de comenzar a avanzar tal como se explica a continuación:



El número que el usuario haya introducido en los displays de siete segmentos corresponderá al número de veces que deberá llamarse la rutina de **DELAY** (que fue creada al principio de esta práctica) antes de que el servomotor pueda avanzar a su posición inmediata siguiente con giro en sentido de las manecillas de reloj.

Por ejemplo, si el usuario introduce un 32, esto querrá decir que el microcontrolador repetirá la rutina **DELAY** (que dura 0.25s) por 32 ocasiones, es decir que tardará aproximadamente 8 segundos en avanzar a su siguiente posición, después volverá a repetir 32 veces la rutina de **DELAY** y volverá a avanzar y así sucesivamente hasta llegar a su posición a 180° de donde comenzó el giro. En el momento que llegue al tope se encenderá el LED ROJO y sonará el BUZZER durante 2 segundos. Al terminar esto el dispositivo quedará listo para que se introduzca nuevamente un valor.

En esta primera versión del dispositivo, una vez que la aguja empieza a avanzar no habrá forma de detenerla hasta que llega al final del giro, la única opción que se tendrá será un botón conectado al pin de **RESET** del microcontrolador el cual obligará al dispositivo a reiniciarse.

Una vez que haya terminado su dispositivo estará en posibilidad de contestar estas preguntas

4) ¿Cuál es el tiempo mínimo que puede tardar el dispositivo desde que inicia hasta que termina? Fundamente su respuesta con cálculos

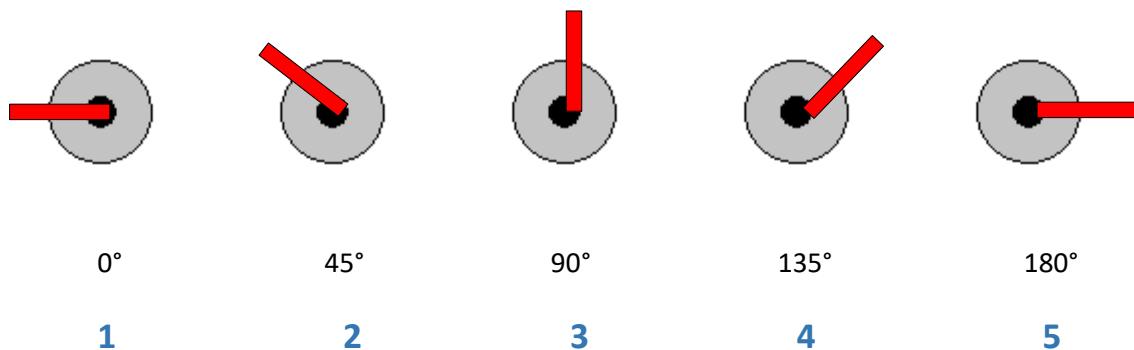
5) ¿Cuál es el tiempo máximo que puede tardar el dispositivo desde que inicia hasta que termina? Fundamente su respuesta con cálculos

6) Realice un archivo de Excel en el que guarde una tabla que indique en su primera columna el valor entre 01 y 99 introducido por el usuario, y en su segunda columna muestre el tiempo que tardará en total la aguja en ir desde su posición izquierda hasta su límite a 180°.

CONTROL DE UN SERVOMOTOR [TIMER0 MODO PWM]

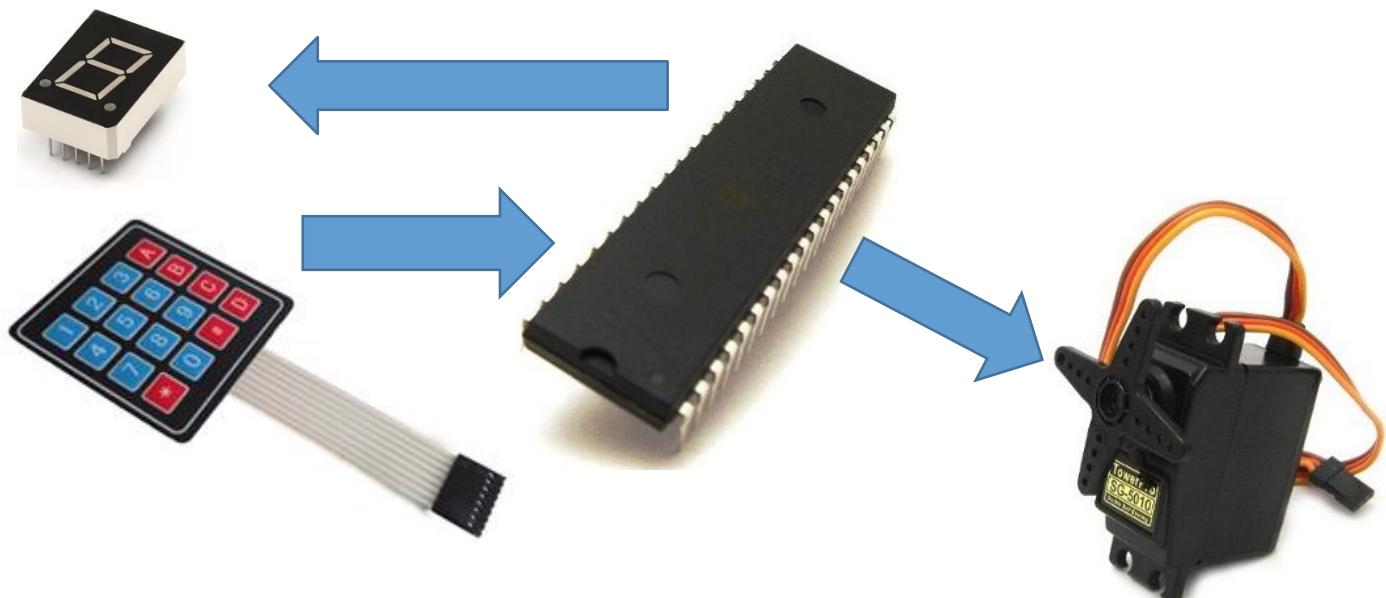
El objetivo de esta práctica es lograr el control de un servomotor, de forma tal que logremos posicionarlo en un ángulo deseado en cualquier momento. Dependiendo concretamente del servomotor que usted haya conseguido, este tendrá un giro entre 0° y 180° , o hay otros que incluso pueden lograr más grados de desplazamiento.

Sin importar el servomotor que se haya conseguido, se desea realizar un control que permita posicionar el motor en los ángulos que se muestran a continuación:



Para ello a cada una de las posiciones se le asigna un número (el que se encuentra indicado en color azul debajo del ángulo). Al encender el circuito el motor deberá tomar la posición “3”.

Se le pide que, además del servomotor, el diseño tenga un teclado matricial, a través del cual el usuario podrá ingresar un número entre 1 y 5 para indicar la posición deseada en cualquier momento (los otros botones no harán nada). También deberá tener un display de 7 segmentos en el cual usted indicará al usuario en qué posición se encuentra.



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 Dirección = RAMEND

Idi YH, HIGH(Dirección)

Idi YL, LOW(Dirección)

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 tabla 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*

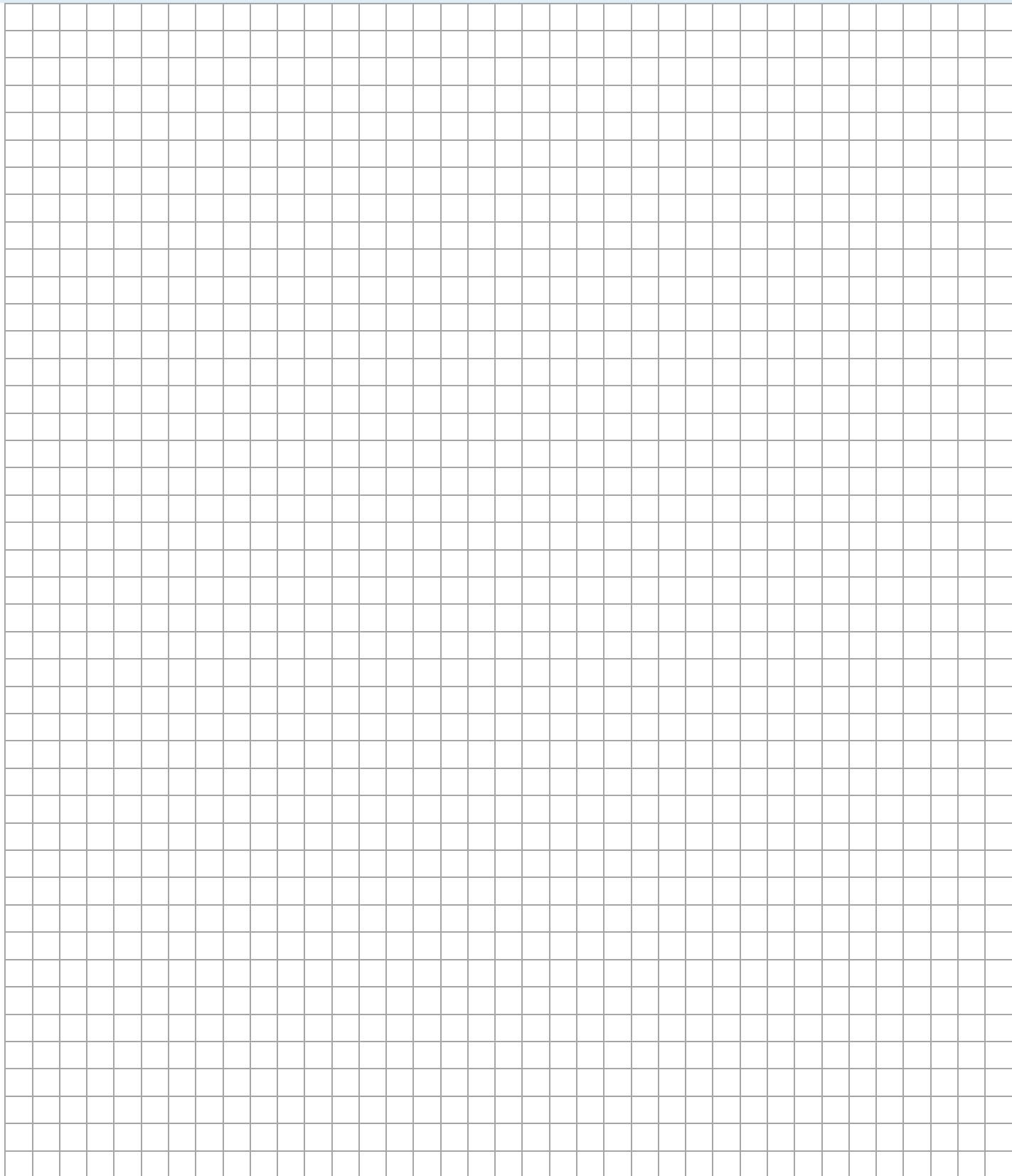
Incr:

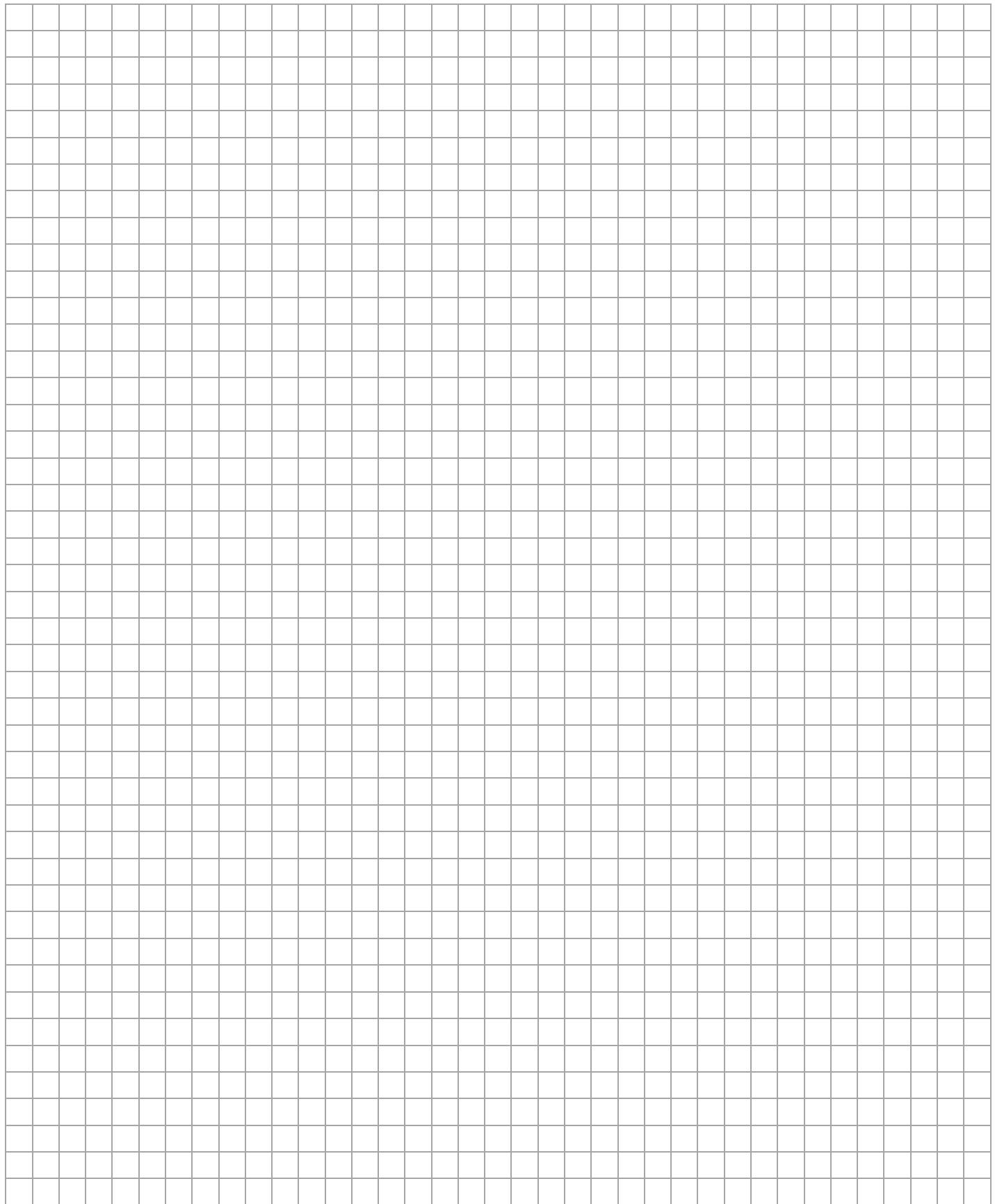
*INC R16
;Ahora verificamos si ya fueron el número de datos que se requieren
CPI R17, 0x01
BRNE Otro
CPI R16, 0x2C
BRNE Otro
RJMP _____ ; a donde se requiera*

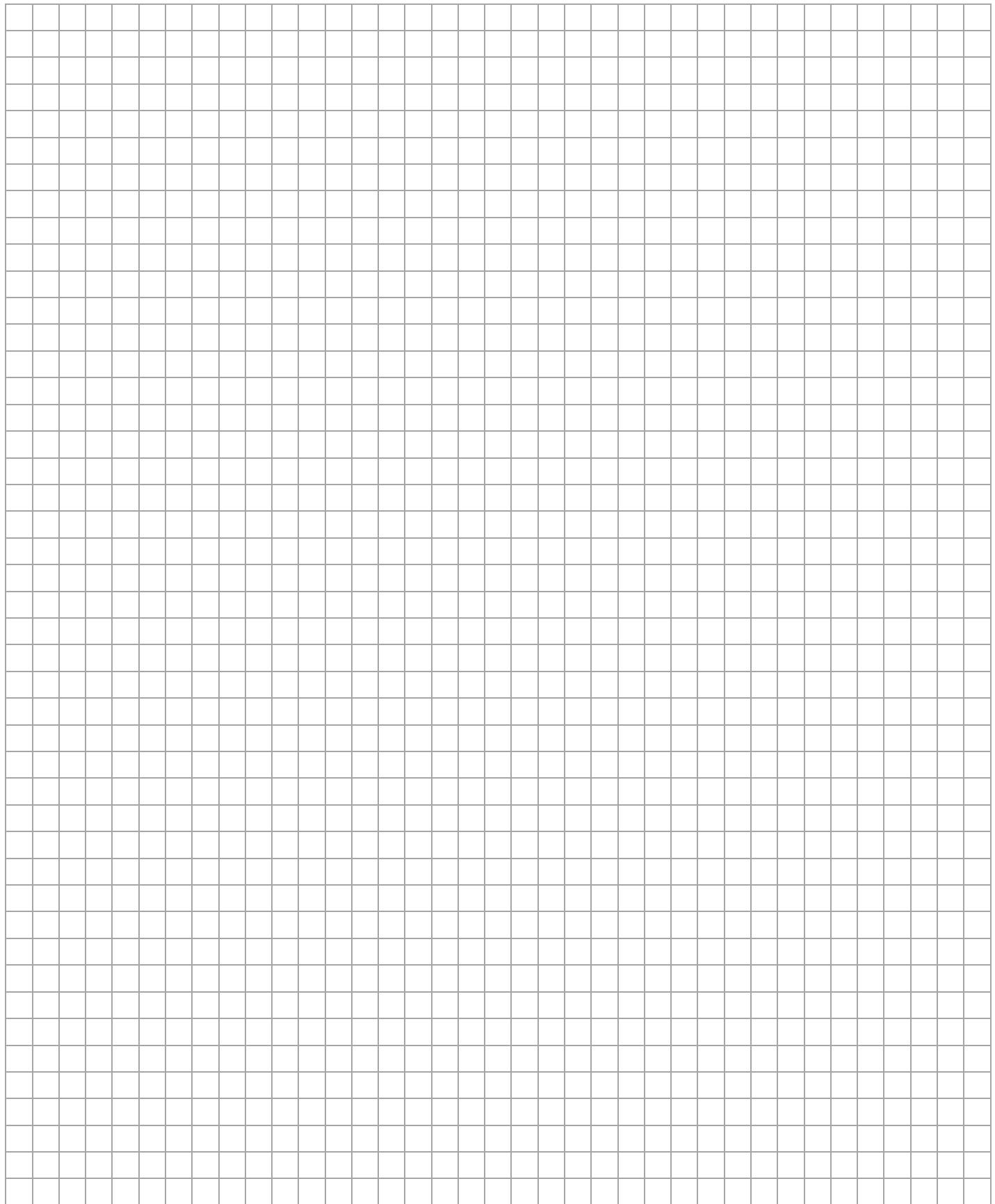
EJERCICIO

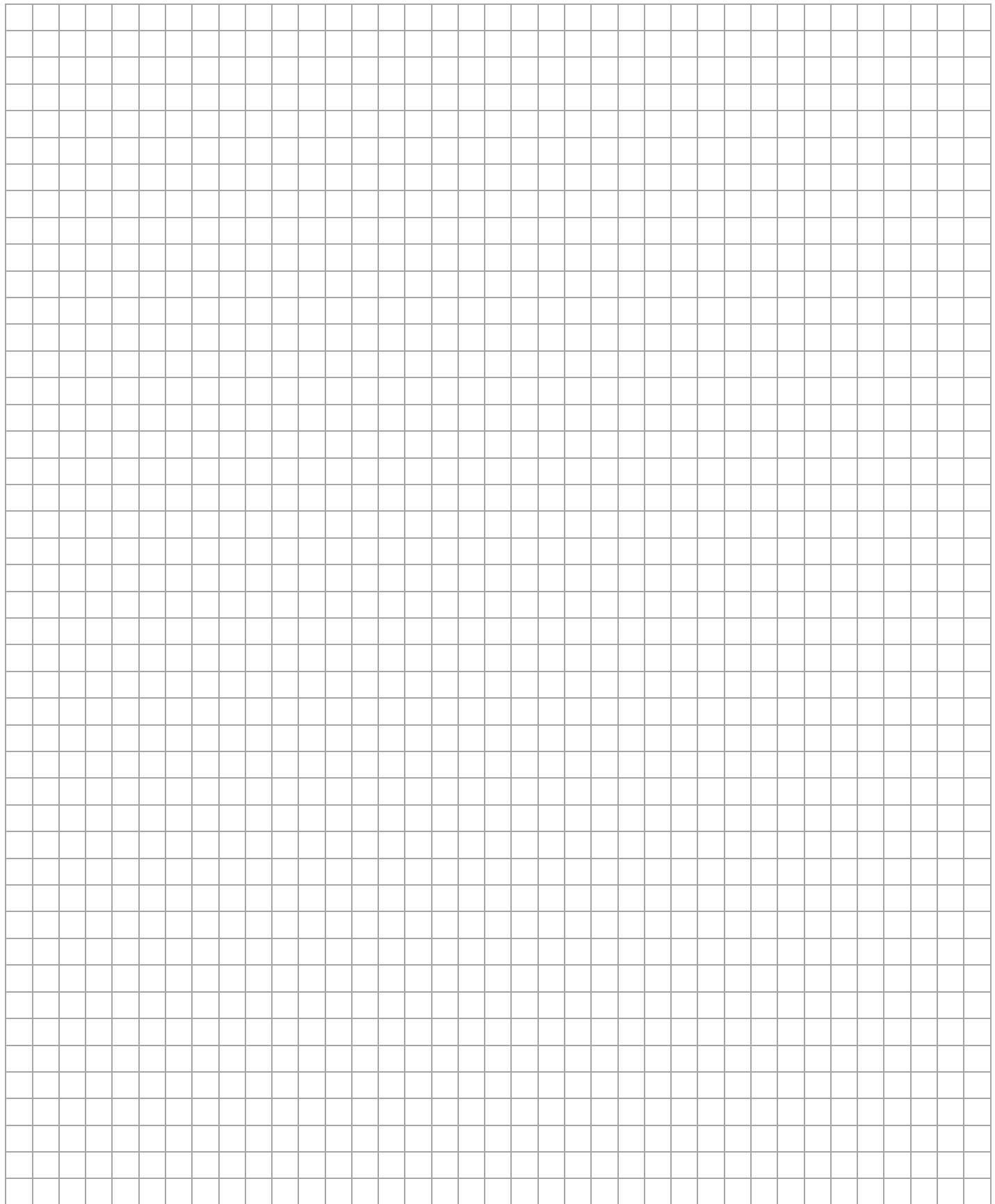
Analice el último código de ejemplo y determine cuántos datos de una tabla leería en forma cíclica ese código

NOTAS PERSONALES – PUNTEROS

A large grid of squares, approximately 20 columns by 25 rows, designed for writing notes or pointers.







SALIDAS CONTROLADAS POR TECLADO MATRICIAL UTILIZANDO PUNTEROS

Se le pide diseñar un dispositivo que tendrá conectados un teclado matricial de 16 teclas en el puerto B y ocho LEDs en el puerto A. Al momento de encender el dispositivo todos los LEDs deberán estar apagados.

Al momento en que el usuario presione una tecla del teclado matricial, la información que se muestra en los displays deberá de cambiar de acuerdo a la información que se muestra en la siguiente tabla:

TECLA	Salida en LEDs
0	00000001
1	00000011
2	00000111
3	00001111
4	00011111
5	00111111
6	01111111
7	11111111
8	10000000
9	11000000
A	11100000
B	11110000
C	11111000
D	11111100
*	11111110
#	10101010

La información mostrada deberá permanecer en los LEDs mientras que la tecla continúe presionada, y al momento en que esta se libera deberá desaparecer y los LEDs permanecerán apagados hasta que se presione nuevamente otra tecla.

Para esta práctica es requisito indispensable que las salidas se encuentren guardadas en la memoria del microcontrolador se la siguiente forma:

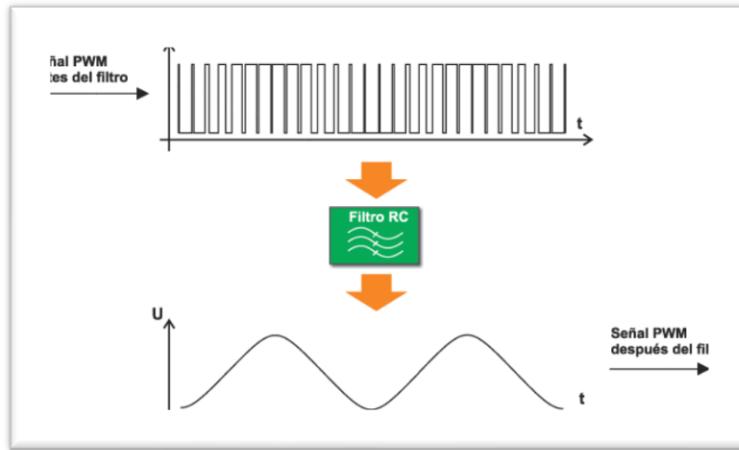
SALIDAS:

*.db 0b00000001, 0b00000011, 0b00000111, 0b00001111, 0b00011111, 0b00111111, 0b01111111, 0b11111111,
.db 0b10000000, 0b11000000, 0b11100000, 0b11110000, 0b11111000, 0b11111100, 0b11111110, 0b10101010*

Y que sean consultadas cada vez que se presione una tecla.

GENERADO DE SEÑALES [PUNTEROS]

Se desea generar una onda senoidal entre 0 V y 5 V utilizando tan solo el microprocesador AVR y un filtro pasa bajas. (Cabe hacer notar que si se desea conectar una carga a la salida de la señal senoidal, será necesario adaptar la señal, pues el micro no es capaz de entregar corriente suficiente, sin embargo, para esta práctica no será necesario puesto que únicamente se requiere ver la señal senoidal en el osciloscopio)



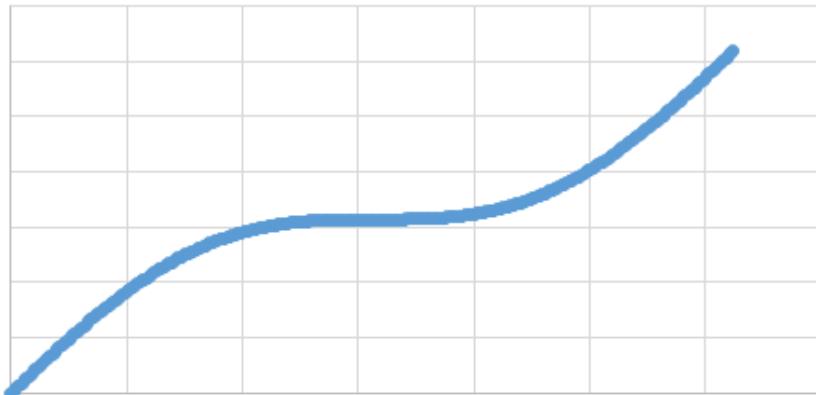
Es necesario que a la salida del microcontrolador conecte un filtro pasivo, los valores de la resistencia y el capacitor deberán ser calculados adecuadamente, en esta práctica su trabajo de investigación consiste en encontrar el tipo de filtro y en calcular los valores adecuados de R y C para obtener el resultado deseado.

Realice el programa necesario utilizando PWM, de forma tal que cuando se genere una interrupción por overflow del timer configurado para PWM se cambie el valor del OCR en el orden que se indica en la siguiente tabla:

SEÑAL_SENOIDAL:

.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 254,254,254,254,254,254,253,253,252,251,250,250,249,248,246
.db 245,244,243,241,240,239,237,235,234,232,230,228,226,224,222,220
.db 218,216,213,211,209,206,204,201,199,196,193,191,188,185,182,179
.db 177,174,171,168,165,162,159,156,153,150,147,144,140,137,134,131
.db 128,125,122,119,116,112,109,106,103,100,97,94,91,88,85,82
.db 79,77,74,71,68,65,63,60,57,55,52,50,47,45,43,40
.db 38,36,34,32,30,28,26,24,22,21,19,17,16,15,13,12
.db 11,10,8,7,6,6,5,4,3,3,2,2,1,1,1
.db 1,1,1,1,2,2,2,3,3,4,5,6,6,7,8,10
.db 11,12,13,15,16,17,19,21,22,24,26,28,30,32,34,36
.db 38,40,43,45,47,50,52,55,57,60,63,65,68,71,74,77
.db 79,82,85,88,91,94,97,100,103,106,109,112,116,119,122,125

GENERADO DE SEÑALES [PUNTEROS]



La grafica que se muestra corresponde a la señal

$$y = \operatorname{sen}(x) + x$$

en el periodo de 0 a 2π

Se le pide que module esta señal con 300 muestras a través de una señal de PWM que salga por el puerto 0C0 del microcontrolador en forma periódica y posteriormente coloque el filtro pasivo necesario obtener tal cual esta señal ocupando la máxima amplitud posible (de 0V a 5V)

Deberá entregar la práctica funcionando correctamente, así como el archivo de excell en donde se justifique la obtención de los valores que haya introducido en la tabla del programa.

LCD (Liquid Crystal Display)

Lo más probable es que estés usando un display JHD 162A que es un LCD de 2x16 caracteres con la misma configuración del HD44780.

La configuración de los pines se muestra a continuación:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Gnd	+5V	Vee	RS	R/W	E	D0	D1	D2	D3	D4	D5	D6	D7	+	-

Este LCD puede ser utilizado enviándole los datos mediante 8 bits o mediante 4 bits, dependiendo de los pines de entrada y salida con los que se cuente en el microprocesador. De momento se configurará el LCD utilizando 8 bits puesto que es más sencillo. Los pines 15 y 16 se utilizan para controlar el backlight del LCD (En caso de que lo tenga), pero no es recomendado usarlo cuando los proyectos son alimentados mediante baterías puesto que se consume más corriente. En cuanto al tercer pin Vee puede ser conectado a un potenciómetro para controlar el contraste del LCD (un lado a Vcc otro a tierra y conectar Vee con el pin central)

En seguida se explican un poco más detalladamente las funciones de los pines más importantes.

RS – Registro Select

Si se escribe un 0 en este registro quiere decir que se va a dar una instrucción.

Si se escribe un 1 en este registro quiere decir que se va a dar un dato para desplegar.

R/W – Read/Write

Un 0 en este pin es empleado para escribir un dato

Un 1 en este pin es empleado para leer datos.

E – Enable

Cuando este bit se pone en alto se le indica al LCD que debe comenzar la operación de lectura o escritura que se le haya indicado con anterioridad.

D4..D7 – Bus de Datos (más significativo)

Pines más significativos del bus de datos del LCD, se emplean para enviar comandos o datos.

D0..D3 – Bus de Datos (menos significativo)

Pines menos significativos del bus de datos del LCD, estos pines no se emplean si se decide configurar al LCD para trabajar en el modo de 4-bits.

INSTRUCCIONES

Analizaremos a continuación las posibilidades que se pueden configurar en los registros RS y R/W con las diferentes configuraciones.

Register Selection		
RS	R/W	Operation
0	0	IR write as an internal operation (display clear, etc.)
0	1	Read busy flag (DB7) and address counter (DB0 to DB6)
1	0	DR write as an internal operation (DR to DDRAM or CGRAM)
1	1	DR read as an internal operation (DDRAM or CGRAM to DR)

RS → 0 ; R/W → 0

Ambos bits deben ponerse en 0's antes de indicarle al LCD alguna instrucción (Limpiar el display, encender el display, etc.)

Ahora bien, si deseamos enviar una instrucción al LCD, antes de configurar los pines RS y R/W (RS=0 y R/W=0) debemos indicar en los pines DB7..DB0 la instrucción que deseamos que sea ejecutada. A continuación se muestra una tabla con las instrucciones posibles.

Instructions

Instruction	Code										Description	Execution Time (max) (when f_{sp} or f_{osc} is 270 kHz)	
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0			
Clear display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DDRAM address 0 in address counter.		
Return home	0	0	0	0	0	0	0	0	1	—	Sets DDRAM address 0 in address counter. Also returns display from being shifted to original position. DDRAM contents remain unchanged.	1.52 ms	
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction and specifies display shift. These operations are performed during data write and read.	37 μ s	
Display on/off control	0	0	0	0	0	0	0	1	D	C	B	Sets entire display (D) on/off, cursor on/off (C), and blinking of cursor position character (B).	37 μ s
Cursor or display shift	0	0	0	0	0	1	S/C	R/L	—	—	—	Moves cursor and shifts display without changing DDRAM contents.	37 μ s
Function set	0	0	0	0	1	DL	N	F	—	—	—	Sets interface data length (DL), number of display lines (N), and character font (F).	37 μ s
Set CGRAM address	0	0	0	1	ACG	Sets CGRAM address. CGRAM data is sent and received after this setting.	37 μ s						
Set DDRAM address	0	0	1	ADD	Sets DDRAM address. DDRAM data is sent and received after this setting.	37 μ s							
Read busy flag & address	0	1	BF	AC	Reads busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 μ s							

Las instrucciones que más se emplearán de esta tabla son las siguientes:

Clear display

Limpia el LCD (en realidad lo que sucede es que escribe “espacios” en todos los caracteres del LCD) y el cursor queda listo para escribir en el primer carácter del LCD

Entry mode set

Bit I/D – Dentro de esta instrucción tenemos la posibilidad de indicar en que dirección queremos que el cursor se desplace cada vez que escribimos un carácter, esto se realiza mediante el bit indicado como I/D, si escribimos un 1, el cursor se irá recorriendo incrementándose, si escribimos un 0 entonces el cursor se irá recorriendo decrementándose, en otras palabras... si escribimos un 1 en el bit I/D el cursor se irá moviendo hacia la derecha (lo cual comúnmente es lo normal), si escribimos un 0 en el bit I/D el cursor se irá moviendo hacia la izquierda.

Bit S – Mediante este bit se puede indicar al LDC que cada vez que se le mande escribir un nuevo carácter, el resto de los caracteres serán desplazados hacia la derecha o hacia la izquierda (como si se fuesen recorriendo). Para que funcione de esta manera el bit S deberá ser escrito con un 1.

Nota.- Si se configuro S=1 entonces el contenido de I/D determinará el sentido en que se desplazarán los caracteres del LCD con I/D=0 será hacia la derecha mientras que con I/D=1 será hacia la izquierda.

Display on/off control

Bit D – Si D=1 entonces el display está encendido, si D=0 entonces el display se apaga.

Bit C – Sirve para controlar si queremos que el cursor sea desplegado (C=1) o no sea desplegado (C=0). Cabe hacer notar que, aunque el cursor no sea desplegado, las funciones especificadas en el “Entry mode set” seguirán llevándose a cabo correctamente.

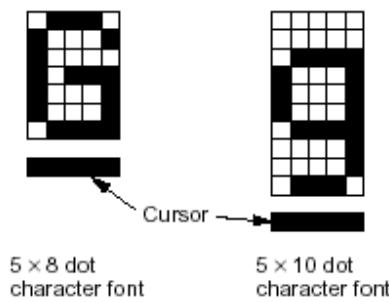
Bit B – Cuando B=1 el carácter indicado por el cursor estará “parpadeando”, mientras que si B=0 no lo hará.

Function set

Bit DL – Sirve para indicar al display si se trabajará en el modo de 4 bits (DL= 0) o de 8 bits (DL= 1).

Bit N – Este bit se emplea para indicar el número de líneas del display, si se configura con un 0 indica que se empleará una línea, si se configura con un 1 indica que se emplearán 2 líneas.

Bit F – Se utiliza para especificar la letra que se empleará, si se configura con un 0 se utilizará la letra de 5x8 puntos, mientras que si se configura con un 1 se le indica que se desea emplear la letra de 5x10 puntos. En seguida se muestra la diferencia entre estas letras.



Cabe hacer notar que si se configura la letra de 5x10 puntos no es posible utilizar dos líneas (puesto que no caben en el LCD)

N	F	No. of Display Lines	Character Font	Duty Factor	Remarks
0	0	1	5 x 8 dots	1/8	
0	1	1	5 x 10 dots	1/11	
1	*	2	5 x 8 dots	1/16	Cannot display two lines for 5 x 10 dot character font

Note: * Indicates don't care.

Set DDRAM Address – Especifica la dirección en donde se escribirá el siguiente dato, esta instrucción nos sirve mucho cuando queremos cambiar entre lo que se escribe en la primera y la segunda línea.

Cuando configuramos para trabajar con una sola línea los valores de ADD pueden ir de 0x00 a 0x4F.

Cuando configuramos para trabajar con dos líneas, los valores de ADD pueden ir de 0x00 a 0x27 en la primera línea y de 0x40 a 0x67 en la segunda línea

BANDERA DE OCUPADO

RS → 0 ; R/W → 1

Esta configuración es utilizada cuando deseamos leer la bandera de ocupado (correspondiente al pin DB7 del LCD). La bandera de ocupado del LCD (es decir el pin DB7) se pone en alto cuando una instrucción está siendo ejecutada y regresa a 0 cuando la instrucción se termina de ejecutar. Ninguna otra instrucción debe enviarse al LCD hasta que esta bandera cambia de estado.

MANDAR INSTRUCCIONES

Para mandar cualquiera de estas instrucciones al LCD lo que debemos hacer es:

- 1) “Escribimos” los bits de la instrucción en los pines DB7..DB0
- 2) Configuramos RS=0
- 3) Configuramos R/W=0
- 4) Configuramos E=1
- 5) Esperamos un tiempo
- 6) Desabilitamos E (E=0)
- 7) Verificamos la bandera “Busy” en DB7 (o bien esperamos un tiempo para garantizar que ya no esté ocupado)

DATOS

RS → 1 ; R/W → 0

Los bits correspondientes a RS y R/W se configuran de esta forma para indicarle al LCD que lo que se mandará es un dato (Para escribir algo en el LCD)

Cuando conectamos el LCD a un microprocesador, podemos indicarle el dato que se va a enviar como un string, es decir enviándole directamente la letra que deseamos, usualmente los textos que se envían al LCD se almacenan en tablas (utilizando el comando .db) sin embargo también existe la posibilidad de enviar los caracteres que se desean desplegar mediante su código binario. A continuación se muestra la tabla correspondiente.

Correspondence between Character Codes and Character Patterns (ROM Code: A00)

Lower 4 Bits	Upper 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)	Ø	ø	P	ø									—	タ	ミ	ø
xxxx0001	(2)	!	1	A	Q	ø	ø							ア	フ	4	ä
xxxx0010	(3)	"	2	B	R	b	r							イ	ツ	×	ø
xxxx0011	(4)	#	3	C	S	c	s							ウ	テ	モ	ø
xxxx0100	(5)	\$	4	D	T	d	t							イ	ト	ト	ø
xxxx0101	(6)	%	5	E	U	e	u							オ	ナ	ユ	ø
xxxx0110	(7)	&	6	F	V	f	v							カ	ニ	ヨ	ø
xxxx0111	(8)	'	7	G	W	ø	w							キ	ア	ラ	ø
xxxx1000	(1)	(8	H	X	ø	x							ウ	ネ	リ	ø
xxxx1001	(2))	9	I	Y	ø	y							レ	ル	ー	ø
xxxx1010	(3)	*	:	J	Z	j	z							コ	ヒ	レ	ø
xxxx1011	(4)	+	;	K	C	k	c							サ	ヒ	□	ø
xxxx1100	(5)	,	<	L	ø	1	1							シ	フ	ワ	ø
xxxx1101	(6)	-	=	M	J	m	ø							ス	ヘ	ン	ø
xxxx1110	(7)	.	>	N	ø	ø	ø							セ	ホ	ン	ø
xxxx1111	(8)	/	?	O	_	ø	ø							ユ	ニ	ø	ø

MANDAR DATOS

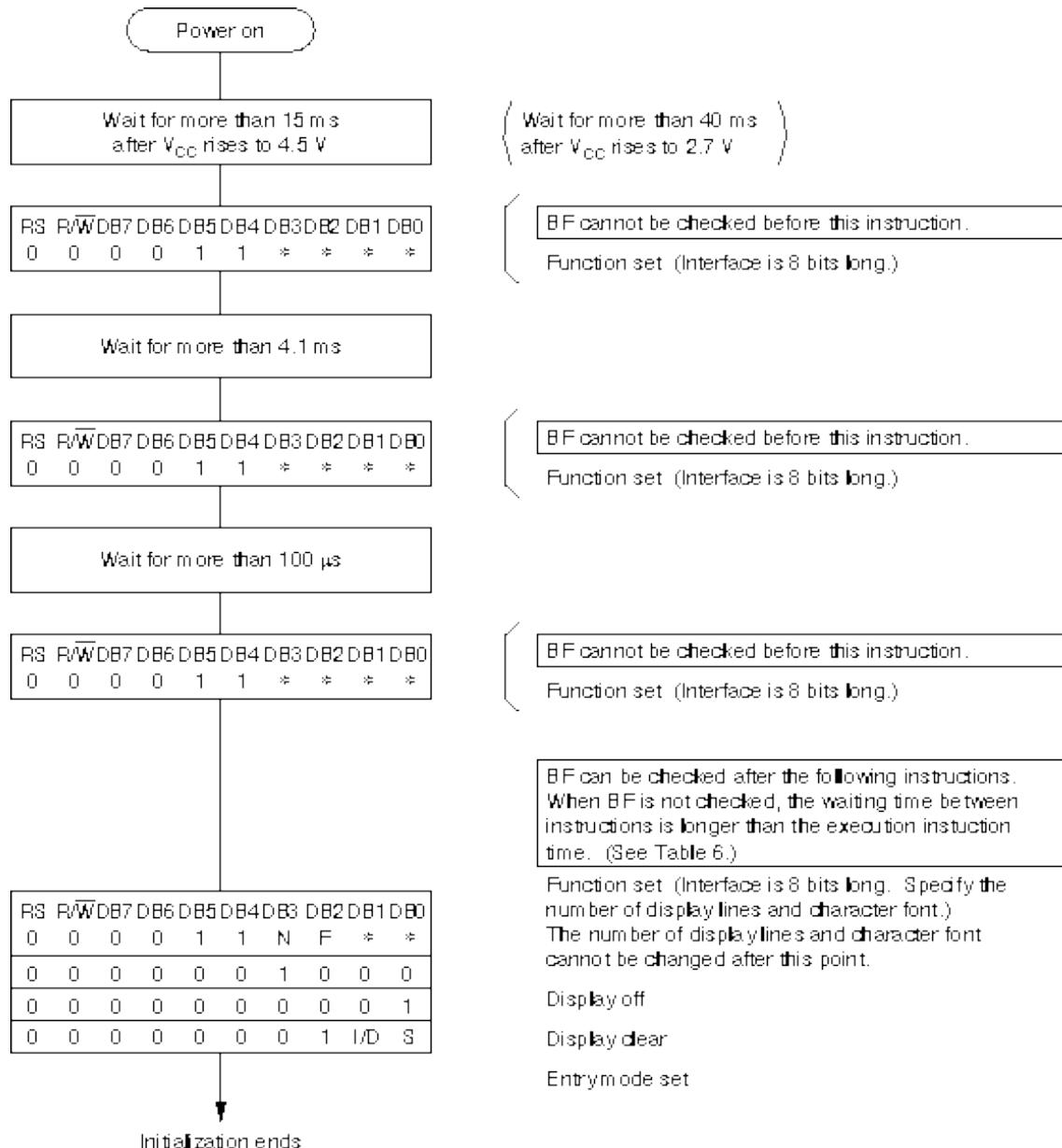
Para mandar cualquiera dato al LCD lo que debemos hacer es:

- 1) “Escribimos” los bits de la instrucción en los pines DB7..DB0 o bien enviamos al puerto correspondiente la letra deseada
- 2) Configuramos RS=1
- 3) Configuramos R/W=0

- 4) Configuramos E=1
- 5) Esperamos un tiempo
- 6) Deshabilitamos E (E=0)
- 7) Verificamos la bandera “Busy” en DB7 (o bien esperamos un tiempo para garantizar que ya no esté ocupado)

INICIALIZAR EL LCD

Antes de poder enviar cualquier instrucción o cualquier dato al LCD, lo primero que debe hacerse (solamente una vez al iniciar nuestro programa en el micro) es inicializarlo. A continuación se muestra un diagrama de flujo, que se encuentra en el archivo pdf del HD44780 de Hitachi, el cual indica la manera en que debe ser inicializado el LCD para trabajar con 8 bits.



POSICIONES DEL LCD SEGÚN EL TIPO DE INICIALIZACIÓN

- Inicializando I/D=1, S=0 (sin shift cursor a la derecha)

\$00	\$01	\$02	\$03	\$04	\$05	\$06	\$07	\$08	\$09	\$0A	\$0B	\$0C	\$0D	\$0E	\$0F
\$40	\$41	\$42	\$43	\$44	\$45	\$46	\$47	\$48	\$49	\$4A	\$4B	\$4C	\$4D	\$4E	\$4F

- Inicializando I/D =0 S=0 (sin shift, cursor a la izquierda) también aplica la misma distribución

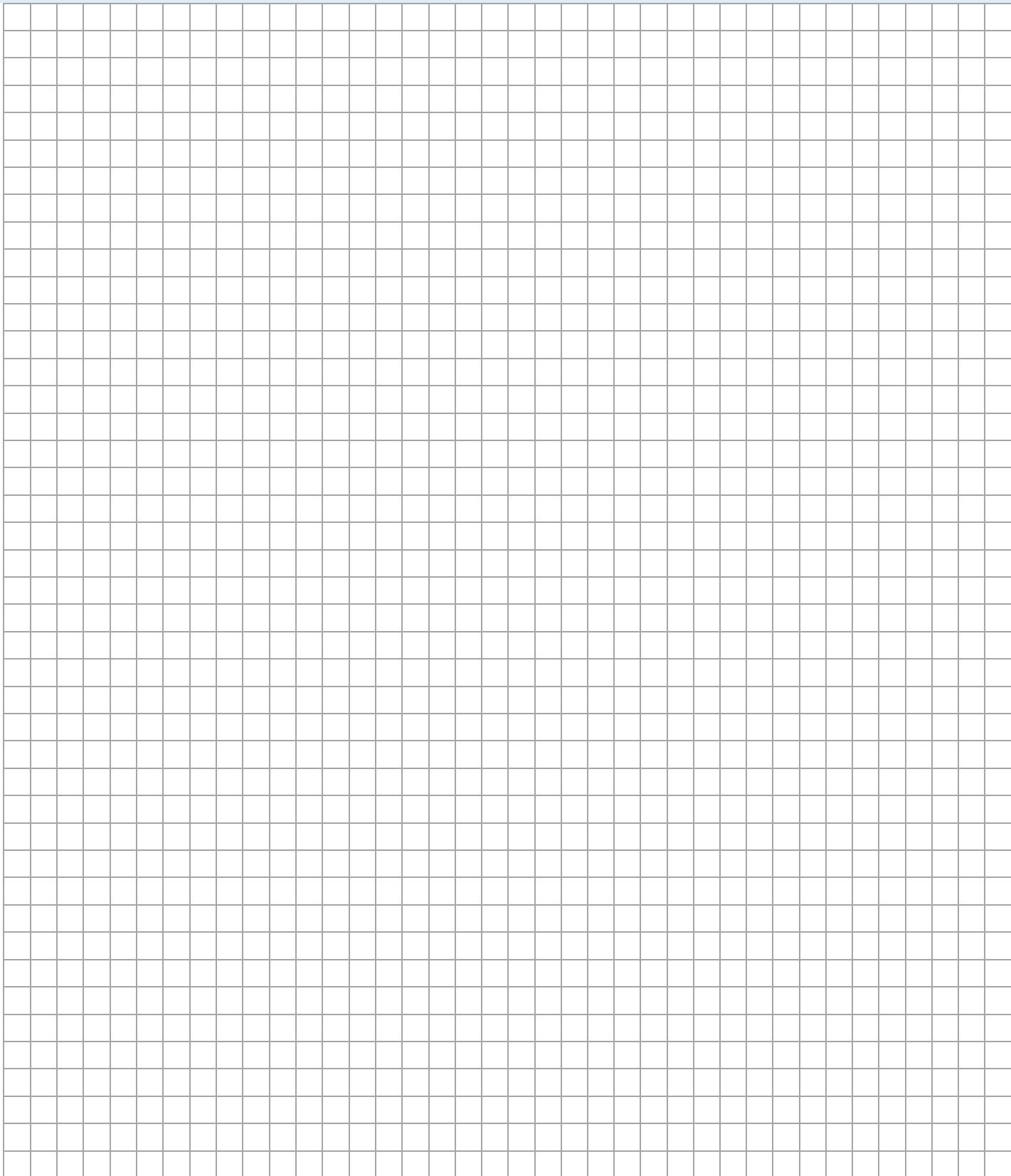
- Inicializando I/D=1, S=1 (Con shift, desplazamiento a la izquierda)

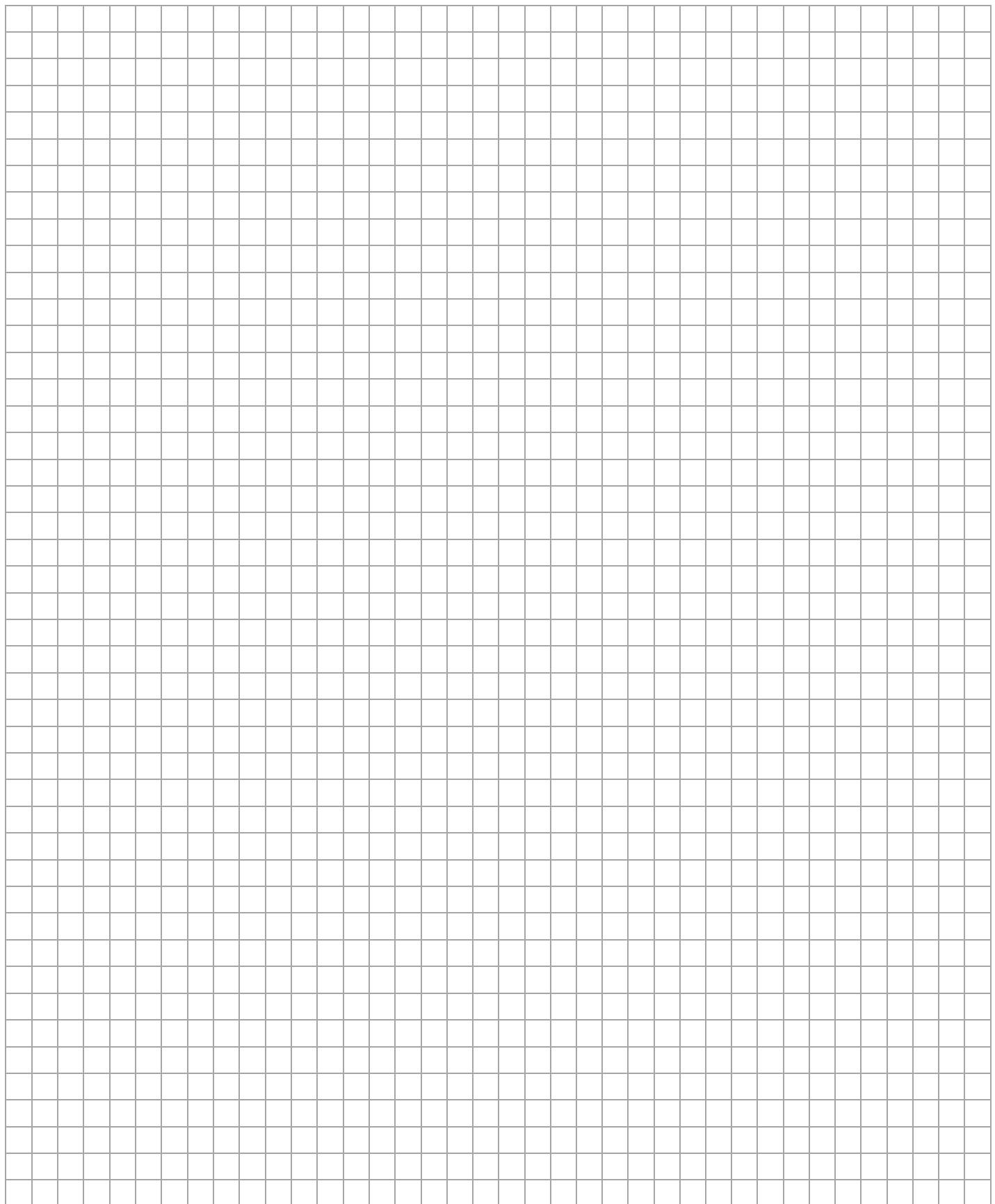
\$01	\$02	\$03	\$04	\$05	\$06	\$07	\$08	\$09	\$0A	\$0B	\$0C	\$0D	\$0E	\$0F	\$10
\$41	\$42	\$43	\$44	\$45	\$46	\$47	\$48	\$49	\$4A	\$4B	\$4C	\$4D	\$4E	\$4F	\$50

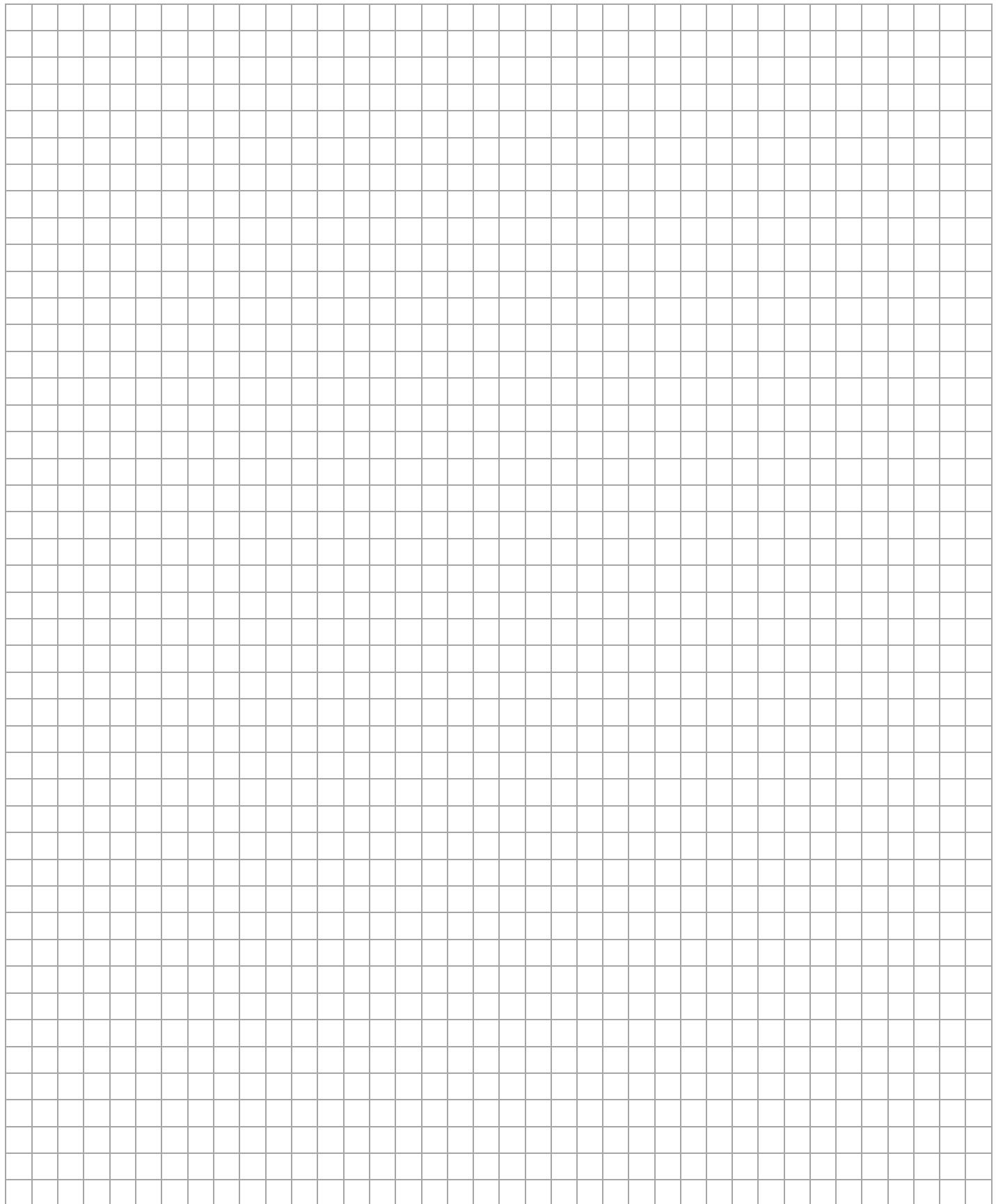
- Inicializando I/D=0 , S=1 (Con shift desplazamiento a la derecha)

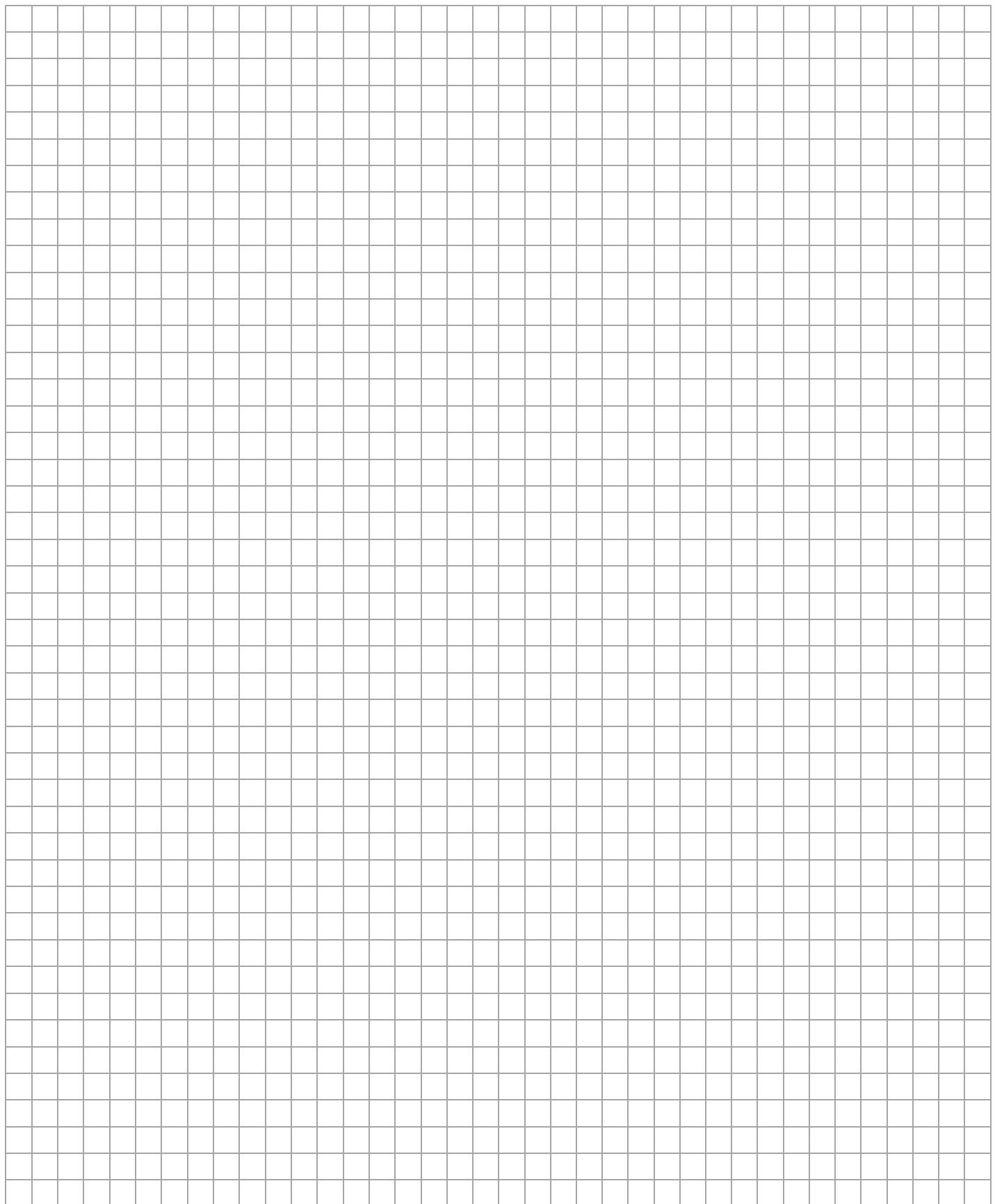
\$27	\$28	\$29	\$2A	\$2B	\$2C	\$2D	\$2E	\$2F	\$30	\$31	\$32	\$33	\$34	\$35	\$36
\$67	\$68	\$69	\$6A	\$6B	\$6C	\$6D	\$6E	\$6F	\$70	\$71	\$72	\$73	\$74	\$75	\$76

NOTAS PERSONALES – LCD

A large grid of squares, approximately 20 columns by 30 rows, designed for writing personal notes or sketches.







NÚMEROS PRIMOS [LCD]

Se desea realizar un dispositivo en el que a través un LCD se mostrarán al usuario los números primos existentes hasta antes del 100 (que son: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89 y 97).

Al encender el dispositivo el LCD deberá permanecer totalmente apagado. Cuando el botón 1 sea presionado entonces el LCD mostrará al usuario “2”, y cada segundo deberán ir cambiando de forma que muestren el siguiente número primo.

Si en algún momento durante la cuenta el usuario vuelve a presionar el botón 1 entonces el avance deberá detenerse y permanecer ahí de forma que sólo volverá a avanzar cuando el usuario presione nuevamente el botón 1.

Una vez que se haya mostrado el último número primo de esta serie, el dispositivo deberá volver a apagar el LCD y permanecerá así hasta que se presione nuevamente el botón 1 (para iniciar desde el principio).

Además el dispositivo deberá tener otros dos botones botón 2 = “disminuir velocidad” y botón 3 = “aumentar velocidad” (que pueden ser utilizados en cualquier momento, incluso cuando la cuenta no se haya inicializado).

Las velocidades disponibles para este programa deberán ser:

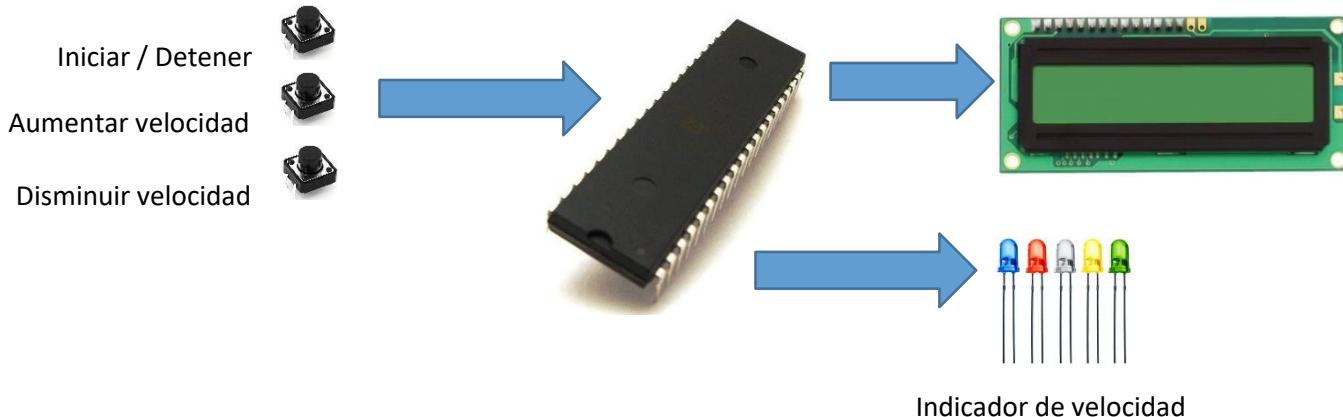
1s → 2s → 3s → 4s → 5s

La velocidad en la que se encuentre funcionando el dispositivo deberá ser mostrada a través de 5 Leds conectados al dispositivo. (1s = 1 led encendido, 2s = 2 leds encendidos, 3s = 3 leds encendidos y así respectivamente, considere que al encender el dispositivo la velocidad por default será 1s, por tanto habrá un led encendido).

Un requisito para que esta práctica sea considerada válida es que los valores que se van a desplegar deberán encontrarse almacenados en una tabla dentro del programa y ahí deberán irse consultando.

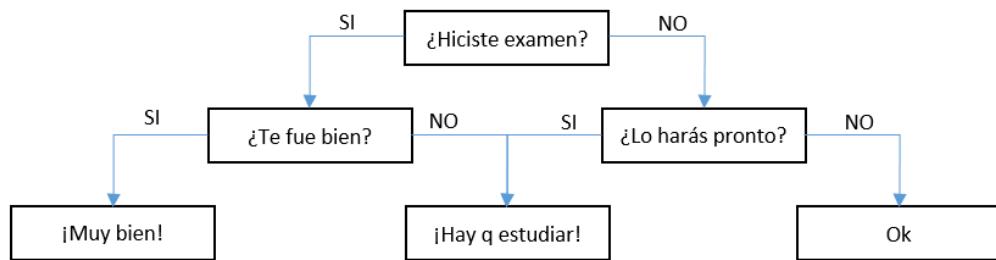
PRIMOS:

.db “02030507111317192329313741434753596167717379838997”



MENÚ [LCD]

Se conectarán al microprocesador dos botones, uno será la respuesta SI y el otro será la respuesta NO. También se conectará un LCD que se irá encargando de presentar al usuario las frases que se presentan a continuación, de acuerdo a las respuestas que vaya dando.



La segunda línea del LCD deberá aparecer siempre vacía.

Realice un programa que contenga la siguiente tabla:

Mensaje1:

.db "Hiciste examen?"

Mensaje2:

.db "Te fue bien?"

Mensaje3:

.db "¡Muy bien!"

Mensaje4:

.db "Lo harás pronto?"

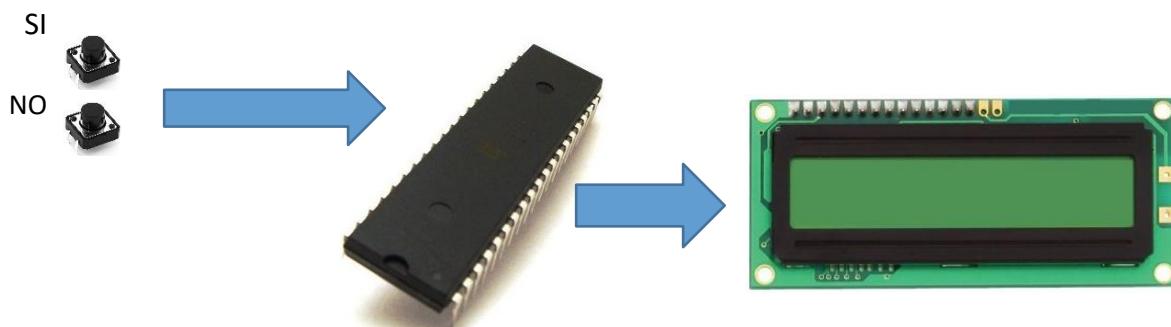
Mensaje5:

.db "Hay q estudiar!"

Mensaje6:

.db "Ok"

Cuando se haya llegado a cualquiera de las últimas opciones, si el usuario vuelve a presionar cualquier botón, volverá al menú principal (a la primera pregunta).



TECLADO MATRICIAL Y LCD [LCD]

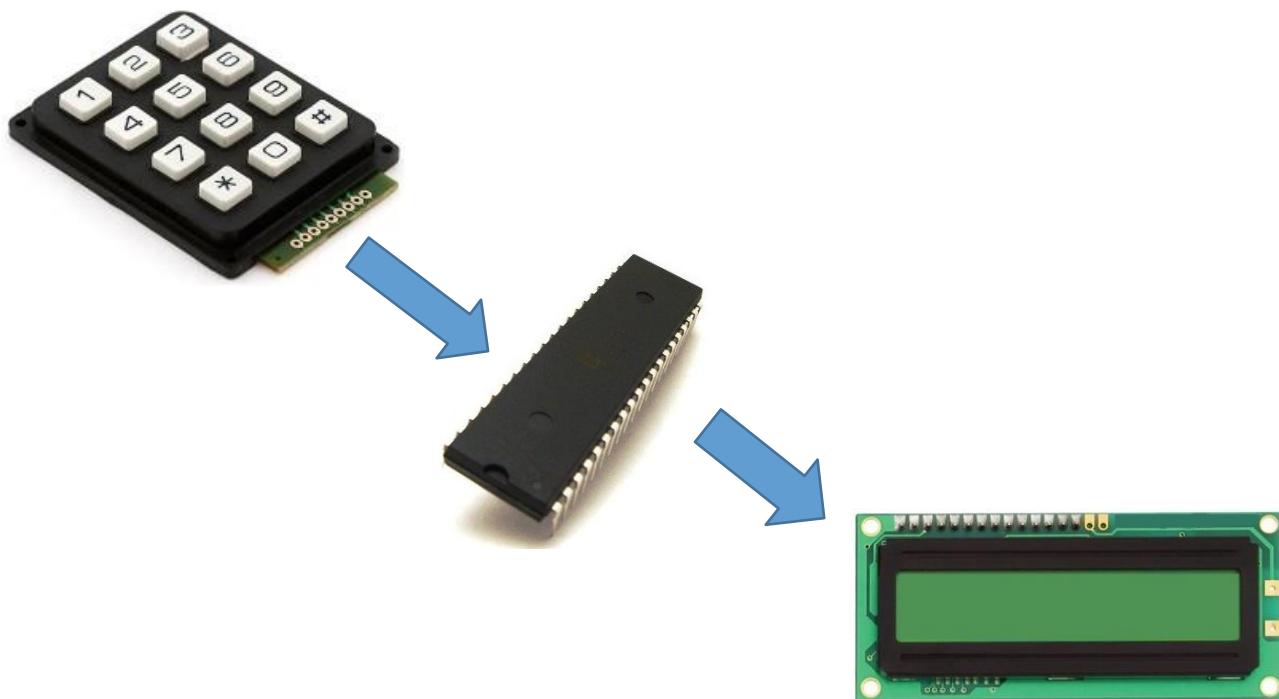
Deberá conectar al microprocesador tanto el teclado matricial como el LCD. Al encender el microprocesador el LCD aparecerá en blanco. Cada vez que se presione una tecla del teclado matricial, el valor presionado deberá aparecer en el LCD en la posición superior derecha y los datos que existan con anterioridad deberán pasar hacia la izquierda.

Por ejemplo:

- Se enciende el circuito y el LCD aparece en blanco.
- Se presiona la tecla 2, entonces en el LCD aparece un “2”.
- Se presiona la tecla 5, entonces el 2 se recorre de forma que aparezca un “25”.
- Se presiona la tecla 3, entonces el 2 y el 5 se recorren de forma que aparezca un “253”.

Si se llena la primera línea de caracteres, en la línea inferior deberá aparecer el mensaje “EXCESO DE DATOS” (sin borrar los datos de la línea superior) y no admitirá más información (por más que se presionen las teclas numéricas no deberá hacer nada).

Una de las teclas que no corresponden a número deberá estar configurada para que, al momento de presionarse (no importará si se tiene un dato, o si bien ya está llena la pantalla) borre la información del LCD y pueda comenzar nuevamente en blanco.



TECLADO ALFANUMÉRICO [LCD]

Se desea desarrollar un sistema que tendrá conectado un LCD y un teclado matricial. (también el puerto serial)

Al momento de encender el sistema la primera línea del LCD dirá:

“Ingrese nombre:”

Y en la segunda línea del LCD aparecerá el cursor parpadeando en el **extremo derecho**.

Entonces el usuario podrá presionar los botones numéricos de su teclado matricial (a excepción del número uno). Siguiendo esta lógica:

Si el usuario presiona una sola vez el botón 2 entonces aparecerá en la pantalla una A, si antes de que pasen 2 segundos el usuario vuelve a presionar el botón 2, entonces la A deberá “borrarse” y aparecer en su lugar una “B” (en la programación en realidad no borramos la A, sino que sencillamente indicamos la posición y sobre escribimos una B). Si antes de 2 segundos el usuario vuelve a presionar la tecla 2 entonces cambiará a una “C”. Si antes de 2 segundos el usuario presiona nuevamente la tecla 2 entonces cambiará por una “A” y así sucesivamente. (El sistema solamente funciona con las letras en mayúsculas).

Si después de presionar la tecla 2 pasan más de dos segundos, y el usuario presiona nuevamente el 2, entonces deberá aparecer otra “A” en la siguiente posición.

En forma breve el teclado funcionará tal como funciona el teclado de un celular antiguo. Pero las letras se irán recorriendo de derecha a izquierda. Es decir si se pone una “A” esta aparecerá en el extremo inferior derecho del LCD, si luego se presiona la “M” entonces en el LCD se verá “AM” alineado al lado derecho del LCD. (Las letras se irán recorriendo). El mensaje que aparece en la primera línea nunca deberá cambiar ni moverse.

Los tiempos deberán ser controlados a través de timers. (No es válido usar ciclos para perder el tiempo).

Al momento que el usuario presione la tecla “D” todo lo que se encuentra escrito en la segunda línea deberá borrarse.

Por lo tanto las funciones de las teclas serán:

2 → A B C

3 → D E F

4 → G H I

5 → J K L

6 → M N O

7 → P Q R S

8 → T U V

9 → W X Y Z

0 → cada vez que se presione provocará un espacio (aquí no aplican los 2 segundos, pues es su única función)

D → cada vez que se presione borrará la segunda línea del LCD (aquí no aplican los 2 segundos, pues es su única función)

Otras teclas → No provocan ningún cambio en el LCD.



ADC (Analog to digital converter)

El AVR ATmega16A cuenta internamente con un convertidor análogo digital de 10 bits que realiza las conversiones mediante aproximaciones sucesivas, los 10 bits resultantes de la conversión se guardan en los registros ADCH:ADCL que tiene la posibilidad de configurarse para que el resultado se ajuste hacia la derecha (por default) o hacia la izquierda (en caso de que solo queramos ocupar los 8 bits más significativos)

El ADC interno del AVR tiene sus propios pines de alimentación, independientes del VCC y la tierra que se conectan siempre al AVR. Dichos pines se conocen como AVCC y AGND. El AGND deberá conectarse a tierra, mientras que el AVCC no deberá diferir más de $\pm 0.3V$ con respecto al VCC.

Con el objetivo de indicarle al ADC el rango de voltajes en el que deberá realizar las conversiones, hay un voltaje externo de referencia debe ser también aplicado al pin VREF y debe de encontrarse entre 2.56V y AVCC. Existe la posibilidad de no conectar el pin de VREF directamente al voltaje, sino realizar una configuración interna en el AVR (dicha configuración se explicará más adelante)

Otra de las ventajas que tiene el ADC del AVR ATmega16A con respecto al LS16A son las entradas diferenciales que pueden amplificar la diferencia entre un par de entradas y posteriormente realizar su conversión.

ADMUX – ADC MULTIPLEXER SELECTION REGISTER

Mediante este registro es posible configurar el tipo de voltaje que se aplicará al pin AREF para referencia del ADC, también se tiene que configurar el ADC para indicarle cuál pin del puerto A será el que se ocupe para llevar a cabo la conversión (o cuáles pines en caso de que se desee emplear el diferencial), y definir si queremos que el resultado de la conversión se almacene recorrido hacia la izquierda o hacia la derecha.

El registro tiene la configuración de bits que se muestra a continuación, y que se explicarán a detalle más adelante:

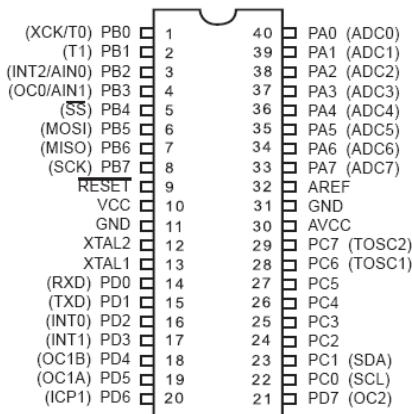
Bit	7	6	5	4	3	2	1	0	ADMUX
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

Bits 7..6 – REFS1..0 – Referente Selection Bits

Estos bits se emplean para determinar el voltaje de referencia que se empleará en el pin VREF. Es importante aclarar que los voltajes internos de referencia nunca deben de ser ocupados si se está aplicando un voltaje externo (en VREF).

Table 84. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin



Cuando se desea utilizar un voltaje de referencia de Vcc (normalmente 5V) se configuran los pines REFS1 y REFS2 con los valores 01 respectivamente y el pin AVCC (pin 30) debe conectarse al voltaje Vcc. (Conviene hacer notar que tanto el pin 11 como el pin 31 (GND) deberán estar conectados a tierra).

Cuando se desea utilizar un voltaje de referencia diferente de Vcc se configuran los pines REFS1 y REFS2 con los valores 00 respectivamente y el valor de referencia deseado debe introducirse por el pin AREF (pin 32). (Conviene hacer notar que tanto el pin 11 como el pin 31 (GND) deberán estar conectados a tierra).

Bit 5 – ADLAR – ADC Left Adjust Result

Mediante el bit ADLAR se le indica al ADC la manera en la que se desea que quede almacenado el resultado de la conversión en los registros ADCH:ADCL.

Si el bit ADLAR es configurado con un 0, el resultado se almacenará alineado a la derecha, es decir:

7	6	5	4	3	2	1	0
-	-	-	-	-	-	Bit 9	Bit 8

ADCH

7	6	5	4	3	2	1	0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

ADCL

Si el bit ADLAR es configurado con un 1, el resultado se almacenará alineado a la izquierda, es decir:

7	6	5	4	3	2	1	0
Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2

ADCH

7	6	5	4	3	2	1	0
Bit 1	Bit 0	-	-	-	-	-	-

ADCL

Este tipo de alineación nos da la posibilidad de que si, únicamente deseamos emplear 8 bits de resolución, puede leerse únicamente el registro ADCH, e ignorar el contenido de ADCL.

Bits4..0 – Mux4..0 – Analog Channel and Gain Selection Bits

Mediante estos bits es posible configurar el pin del Puerto A que tomaremos como entrada para realizar la conversión con el ADC, poniendo en estos bits los valores que van de 00000 a 00111 es posible seleccionar uno de los pines específicos del puerto A, en cambio, con los valores siguientes se seleccionarían dos entradas de las cuales se sacaría el diferencial y se le aplicaría una ganancia determinada antes de convertirla en dato digital.

La tabla que muestra los valores de configuración para estos bits, se muestra en la siguiente página.

ADMUX

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
Ver tabla 84 (Voltaje de referencia)	Ver tabla 84 (Voltaje de referencia)	0 se alinea a la der. 1 se alinea a la izq.	Ver tabla 85 (pines)				

Table 85. Input Channel and Gain Selections

MUX4..0	Single Ended Input	Pos Differential Input	Neg Differential Input	Gain
00000	ADC0	N/A	N/A	N/A
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000	1.22V (V _{BE})	ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	0V (GND)	N/A		
11111				

SFIOR – SPECIAL FUNCTION IO REGISTER

El ADC puede funcionar de 2 modos diferentes:

- “Free running” – Utilizando esta configuración, el ADC se encuentra disponible para realizar una nueva conversión en el momento que se le indique a través del bit de inicio de conversión.
- “Single Conversion” – Al utilizar este modo de operación, cada una de las conversiones debe ser inicializada por algún evento (trigger).

Para configurar el modo de funcionamiento del ADC nos interesan los bits del 7 al 5 del registro SFIOR.

Bit	7	6	5	4	3	2	1	0	SFIOR
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	
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 0

ADTS2..0 – Auto Trigger Source Selections

A continuación se presentan las posibilidades para configurar los bits ADTS2..0:

Table 87. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

Cabe hacer notar que si se desea emplear algún trigger (diferente al free running mode) el bit ADATE del registro ADCSRA deberá ser configurado igual a 1.

SFIOR

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
			0	0	0	0	0
ADTS2	ADTS2	ADTS0	-	ACME	PUD	PSR2	PSR10
Ver tabla 87 (Para configurar trigger)	Ver tabla 87 (Para configurar trigger)	Ver tabla 87 (Para configurar trigger)	No se utiliza				

Es oportuno reconocer que si se desea que el ADC funcione en modo Free running, no es obligatorio cargarle ningún valor al registro SFIOR (a menos que su valor haya sido cambiado antes en el programa) puesto que por default este registro contiene 0b00000000 (es decir está en modo free running).

ADCSRA – ADC CONTROL AND STATUS REGISTER A

Este registro nos sirve tanto para configurar el ADC como para habilitarlo, a continuación se muestran los bits que lo componen y se da una breve explicación de cada uno:

Bit	7	6	5	4	3	2	1	0	ADCSRA
Read/Write	R/W								
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – ADEN – ADC Enable

Para activar el ADC es necesario configurar el pin ADEN = 1 en el registro ADCSRA, configurando ADEN = 0 el ADC se encontrará desactivado.

Para que el ADC no consuma energía, el ADEN deberá estar configurado como un cero. De igual manera, si el ADEN se cambia a 0 mientras se está realizando una conversión, esto ocasionará que la conversión se cancele.

Bit 6 – ADSC – ADC Start Conversion

Cuando se está empleando el modo “Free running” se escribirá un 1 en este bit para indicarle el inicio de la primera conversión.

Cabe hacer notar que la primera vez que se escribe un 1 en el bit ADCS después de haber habilitado el ADC (con el bit ADEN) se llevará a cabo una “conversión extendida”, antes de realizar la conversión real, la cual toma más tiempo y realiza la inicialización del ADC.

Mientras la conversión está siendo llevada a cabo, este bit permanecerá en un 1, y su valor cambiará automáticamente a 0 en el momento en que se termine la conversión.

Normalmente, al inicializar el LCD este bit se carga con un cero (0) y en el código, al momento que se desea que se inicien las conversiones puede ejecutarse la instrucción:

SBI ADCSRA, ADSC ; *Inicia las conversiones*

De forma similar, si en cualquier momento se desean detener las conversiones, es posible hacerlo ejecutando la instrucción:

CBI ADCSRA, ADSC ; *Detiene las conversiones*

En caso de que no se estén utilizando interrupciones para saber cuándo una conversión ha terminado, podría optarse por verificar el estado de este bit (ADSC) (recordemos que tomará por un breve lapso de tiempo el valor de 0 cuando la conversión se haya terminado).

Bit 5 – ADATE – ADC Auto Trigger Enable

Si se desea que una señal a la que llamaremos “Trigger” sea la encargada de indicar el momento en que se inicia una nueva conversión (el trigger concreto se especifica en el registro SFIOR), entonces es necesario cargar un 1 a este bit.

Cuando se tiene 1 uno en el bit ADATE, el inicializador automático del ADC está activo, de forma que cada vez que se detecte un pulso de subida en la señal de “Trigger” se iniciará una nueva conversión.

Bit 6 – ADIF – ADC Interrupt Flag

El ADC cuenta además con una interrupción que puede activarse cada vez que se termine una conversión. Este bit es la bandera que se levanta cuando la interrupción ha sido configurada y puede limpiarse escribiendo un uno lógico en este bit.

Cuando se genera una interrupción el bit ADIF cambia de valor y automáticamente el programa ejecuta la interrupción correspondiente. El valor de ADIF regresa a su valor original al momento que encuentra la instrucción reti de la rutina de interrupción. Cabe hacer notar que para que esta interrupción funcione, es necesario que se encuentre activo el bit ADIE.

Bit 3 – ADIE – ADC Interrupt Enable

Cuando este bit se encuentra configurado como un 1, y se hayan habilitado las interrupciones del microcontrolador (instrucción sei) se generará automáticamente una interrupción cada vez que se termine una conversión.

Bit 2..0 – ADPS2..0 - ADC Prescaler Select Bit

Debido a que el ADC funciona mediante aproximaciones sucesivas, se requiere de una frecuencia de reloj entre 50kHz y 200kHz para determinar su resolución.

Si deseamos utilizar una resolución de menos de 10 bits, la frecuencia del reloj del ADC podría ser incluso mayor a 200kHz, teniendo una tasa de muestreo más alta.

Es posible configurar los bits ADPS2..0 en el registro ADCSR, para determinar el prescaler que se empleará para dividir la frecuencia de reloj del microprocesador, y tener así la frecuencia adecuada para el reloj del ADC. El prescaler comenzará a contar en el momento en que el registro ADEN sea activado y permanecerá funcionando siempre que no se cambie el contenido de ADEN a 0.

Normalmente una conversión análogo digital toma 13 ciclos de reloj del ADC aproximadamente (recuerde que son diferentes a los ciclos del reloj del microprocesador), aunque la primera de las conversiones que se realice al momento de activar el ADC (“Conversión extendida”) podría tomar hasta 25 ciclos de reloj del ADC.

A continuación se muestran los factores de división del prescaler, dependiendo de la configuración de los bits ADPS2..0:

Table 86. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ADCSRA

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
			0				
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Para activar 1=Activo 0=Inactivo	Start conversión 1= Inicia conversión 0= No hace conversiones	Iniciar conversión mediante señal de trigger 1= trigger 0= sin trigger	Bandera de Interrupción del ADC	1= Habilita interrupción al terminar la conversión 0= No habilita interrupción	Ver tabla 86 (prescaler para frecuencia del reloj del ADC)	Ver tabla 86 (prescaler para frecuencia del reloj del ADC)	Ver tabla 86 (prescaler para frecuencia del reloj del ADC)

ADCH:ADCL – ADC DATA REGISTER

En estos registros se muestra el resultado de la conversión realizada por el ADC, en caso de que se haya usado el modo diferencial (el resultado puede ser negativo) se almacena en complemento a dos, cabe hacer notar que se emplean dos registros puesto que el resultado de la conversión será un dato de 10 bits.

Cuando se leen estos registros es importante que primero sea leído el ADCL y solamente después de eso sea leído el ADCH, para garantizar así que las 2 lecturas corresponden al mismo dato, esto sucede debido a que, una vez que se leyó el registro ADCL, el registro de datos del ADC (ADCH:ADCL) se bloquea, de forma automática, de modo que su contenido no puede ser modificado hasta el momento en que ADCH sea leído.

En caso de que se esté empleando una resolución de 8 bits y se haya configurado ADCL = 1 no es necesario leer el ADCL puesto que su valor no lo empleará.

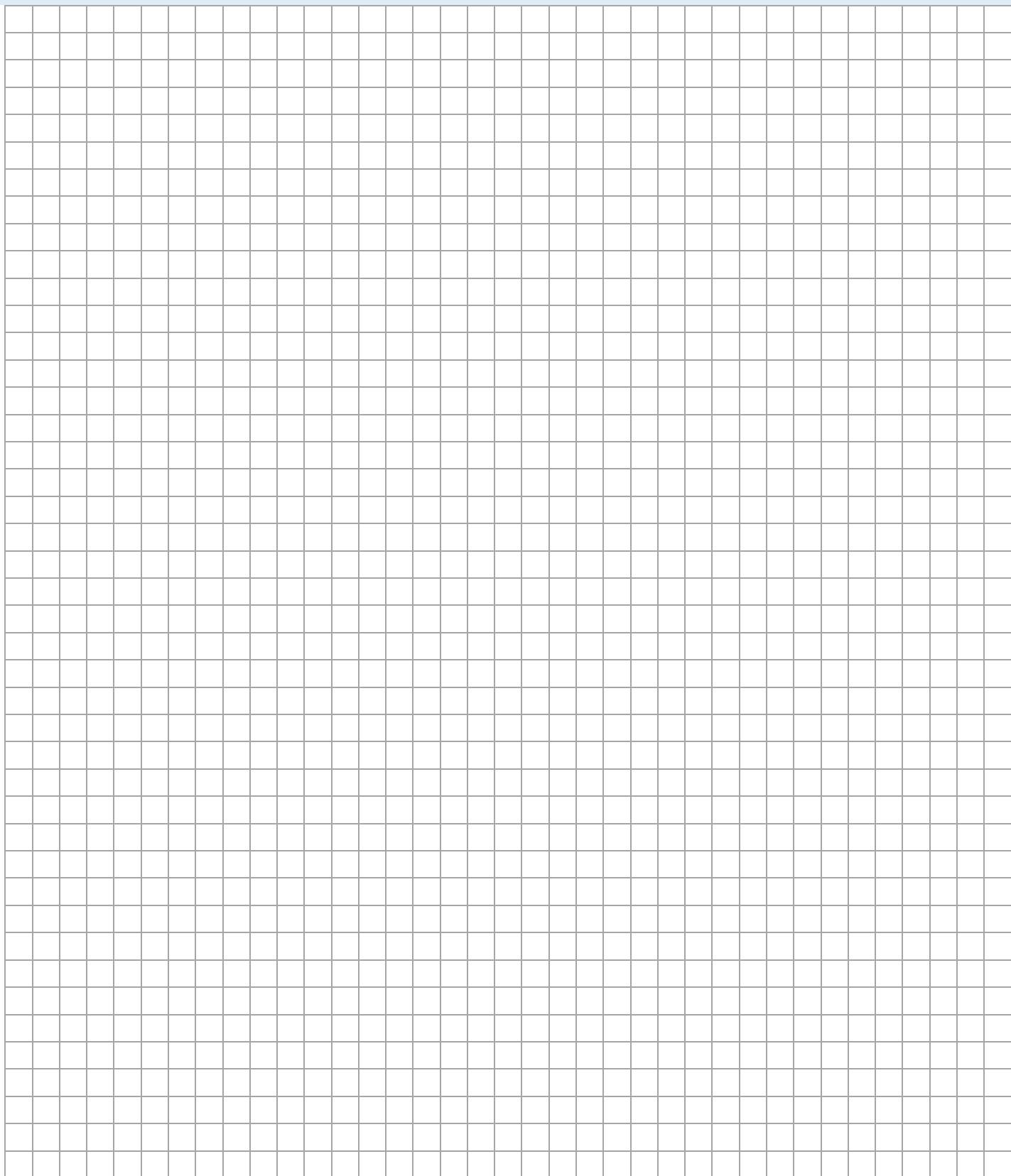
ADLAR = 0

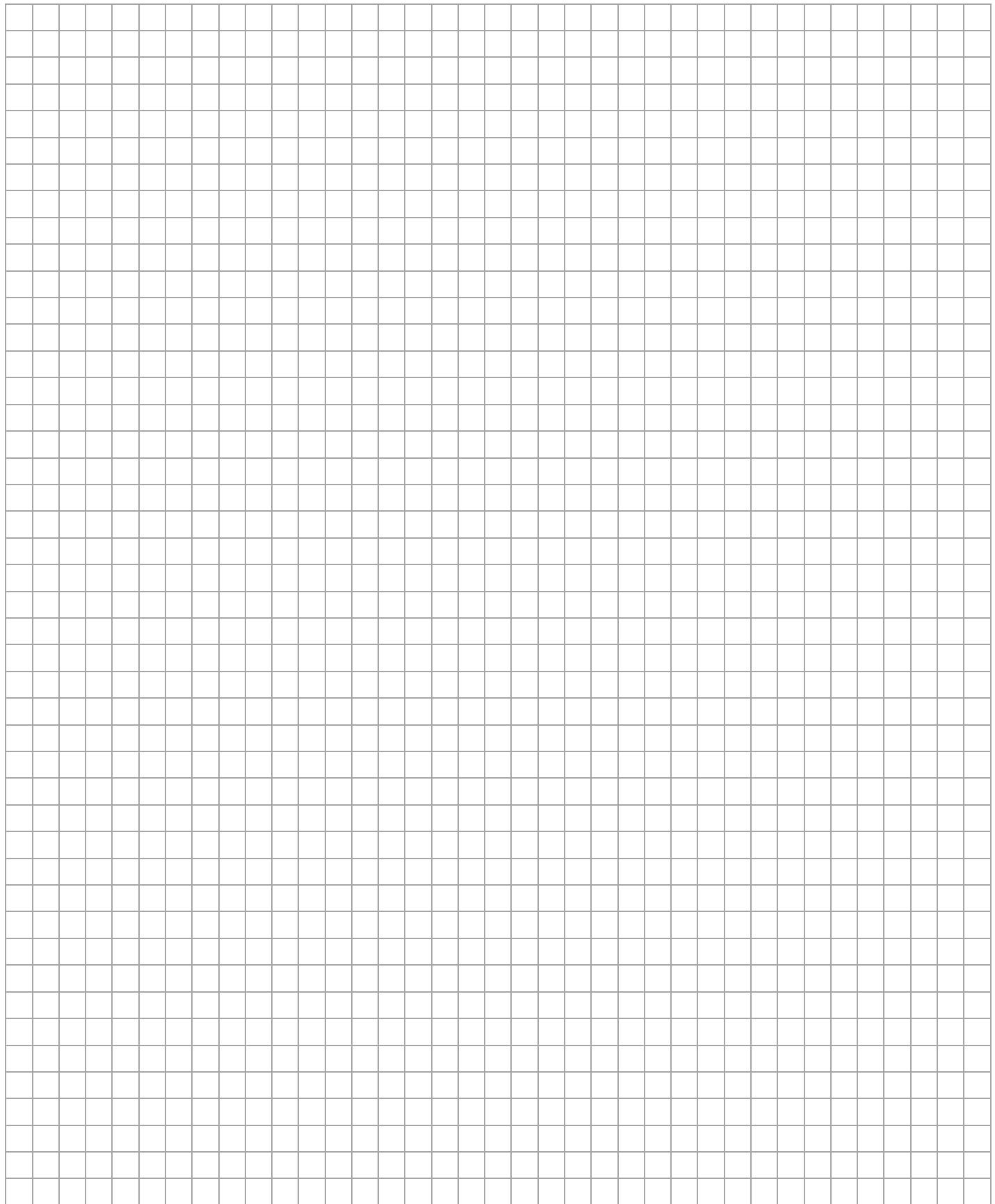
Bit	15	14	13	12	11	10	9	8	ADCH
	–	–	–	–	–	–	ADC9	ADC8	ADCL
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

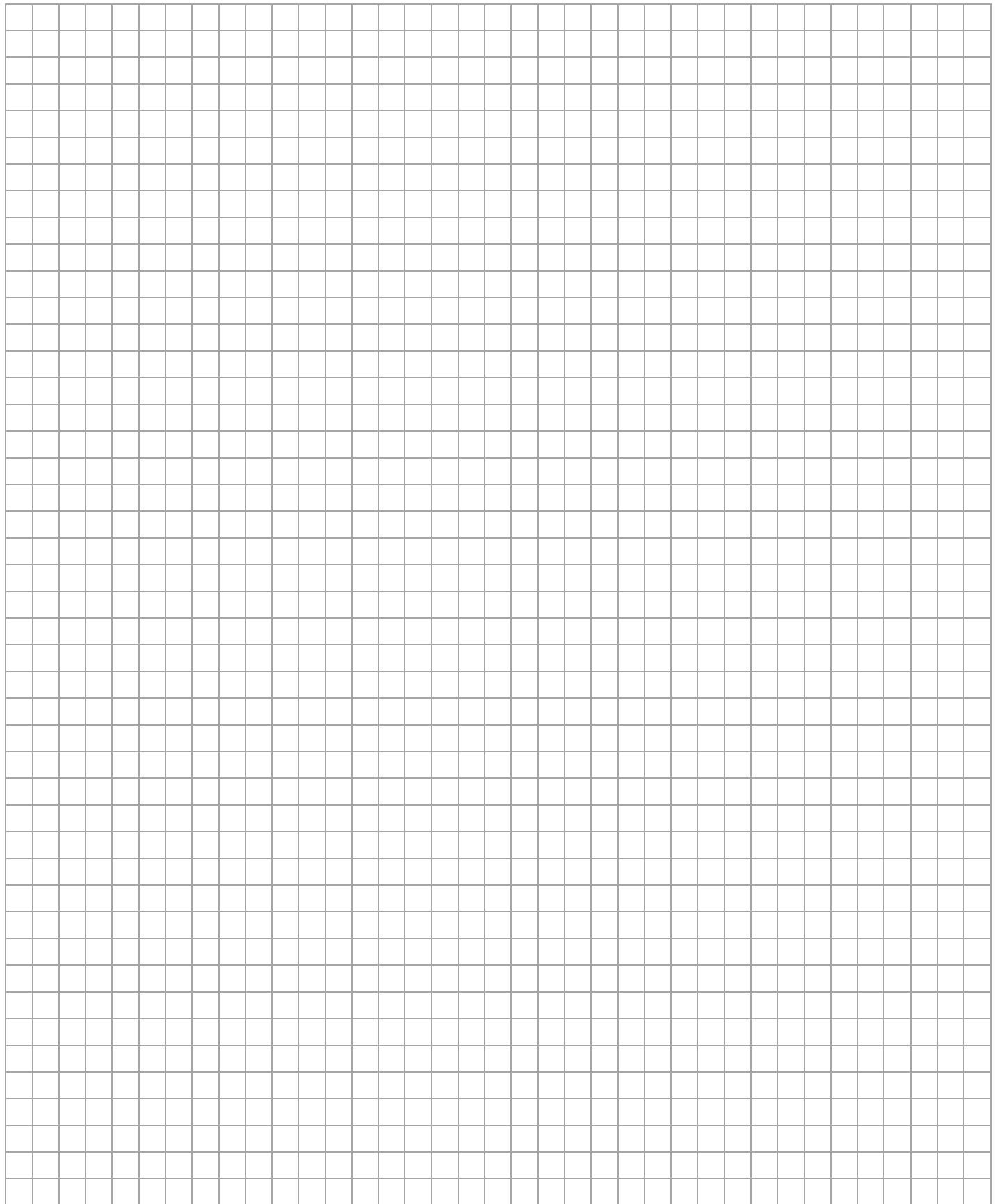
ADLAR = 1

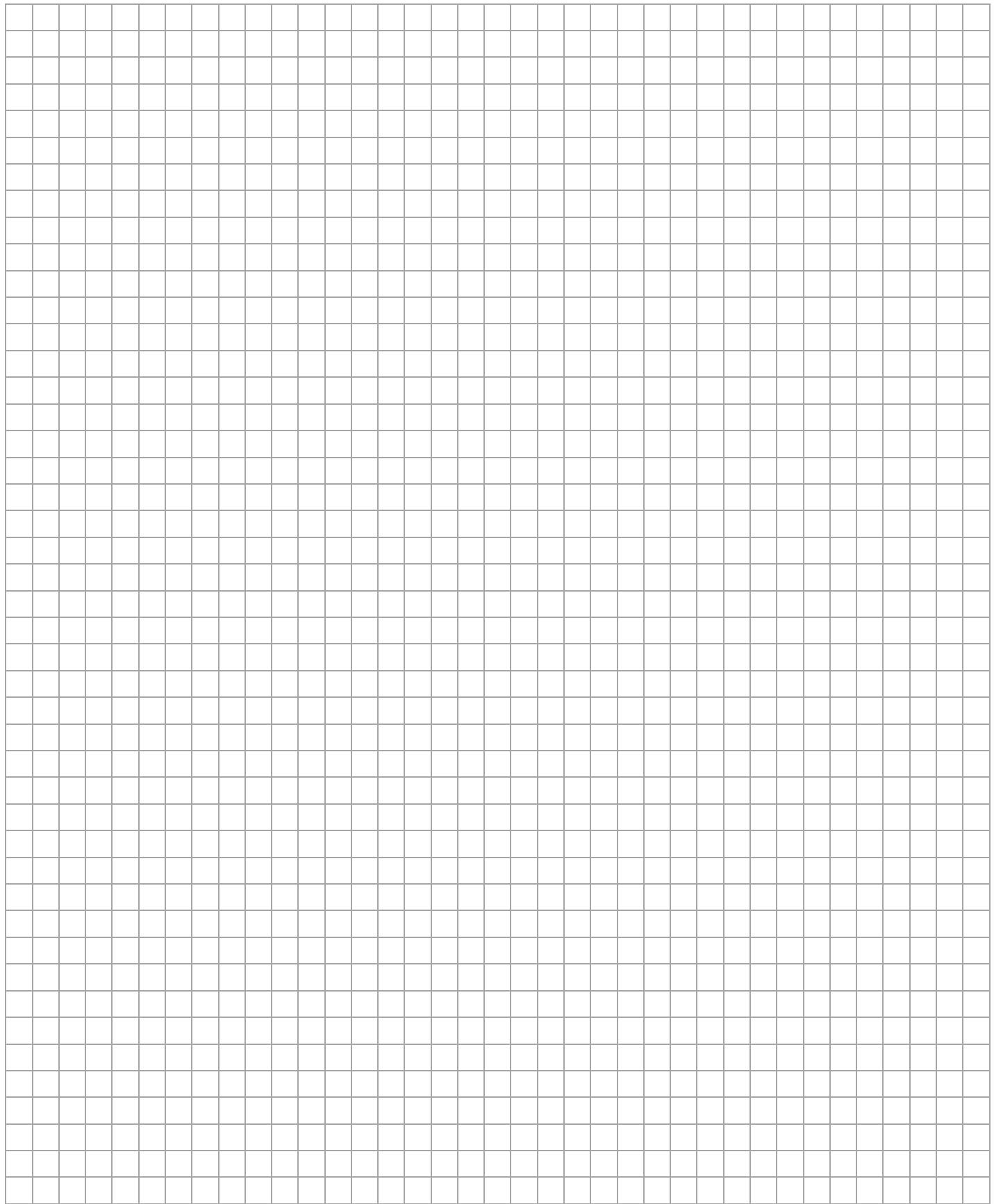
Bit	15	14	13	12	11	10	9	8	ADCH
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCL
	ADC1	ADC0	–	–	–	–	–	–	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

NOTAS PERSONALES – ADC

A large grid of squares, approximately 20 columns by 25 rows, designed for writing personal notes or sketches.







VOLTÍMETRO MEDIANTE ESCALA DE LEDS [ADC]

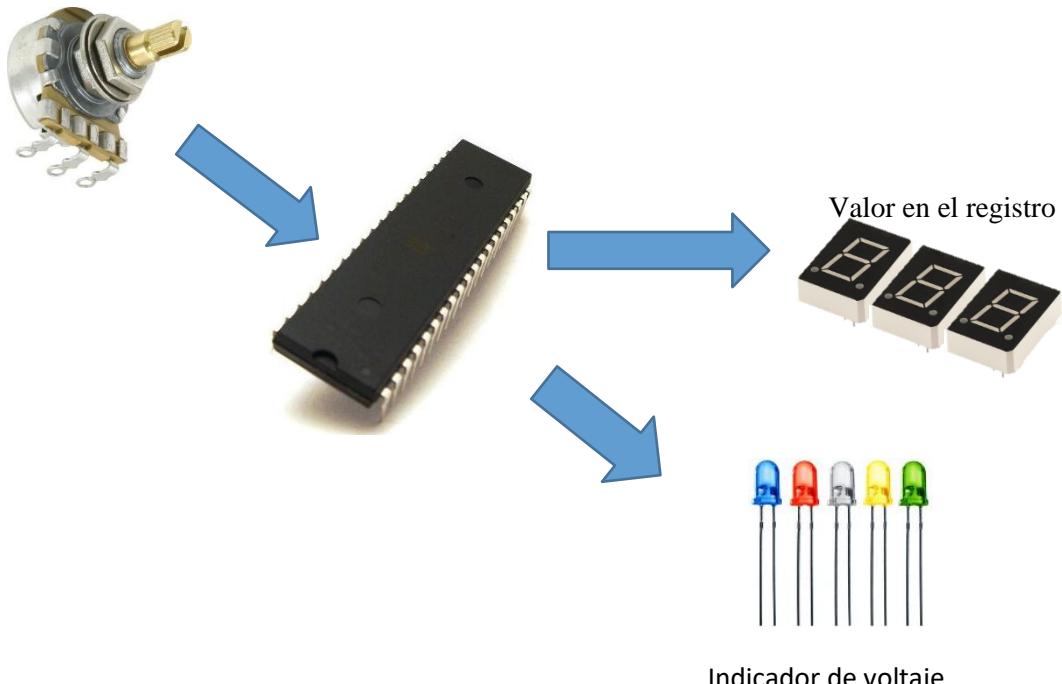
Se requiere conectar al microcontrolador ATMEGA8535 cinco LEDs (ordenados en hilera), un potenciómetro (que se describe más adelante), tres displays de siete segmentos, y se utilizará una entrada del ADC.

A la entrada del ADC se le proporcionará un voltaje variable (de 5V como máximo) que se obtendrá mediante un potenciómetro (un extremo conectado a 5V, el otro extremo conectado a tierra y la derivación conectada en el pin del microcontrolador configurado como entrada para el ADC).

El programa que se desarrolle deberá revisar constantemente el voltaje que esté entrando por el pin del ADC y responderá de la siguiente forma:

- Cuando el voltaje en la entrada del ADC se encuentre entre 0V y 0.5V no deberá prender ningún LED.
- Cuando el voltaje en la entrada del ADC se encuentre entre 0.5V y 1.5V deberá prender únicamente el LED menos significativo.
- Cuando el voltaje en la entrada del ADC se encuentre entre 1.5V y 2.5V deberán prender los dos LEDs menos significativos.
- Cuando el voltaje en la entrada del ADC se encuentre entre 2.5V y 3.5V deberán prender los tres LEDs menos significativos.
- Cuando el voltaje en la entrada del ADC se encuentre entre 3.5V y 4.5V deberán prender los cuatro LEDs menos significativos.
- Cuando el voltaje en la entrada del ADC se encuentre entre 4.5V y 5V deberán prender todos los LEDs.

Además de encender los LEDs en la forma en que se indica, en los displays de 7 segmentos deberá mostrarse en valor (en decimal... entre 000 y 255) que se está obteniendo de realizar la conversión (para ello se le sugiere que utilice la rutina que programó en el examen en la que un registro se convierte en centenas, decenas y unidades).

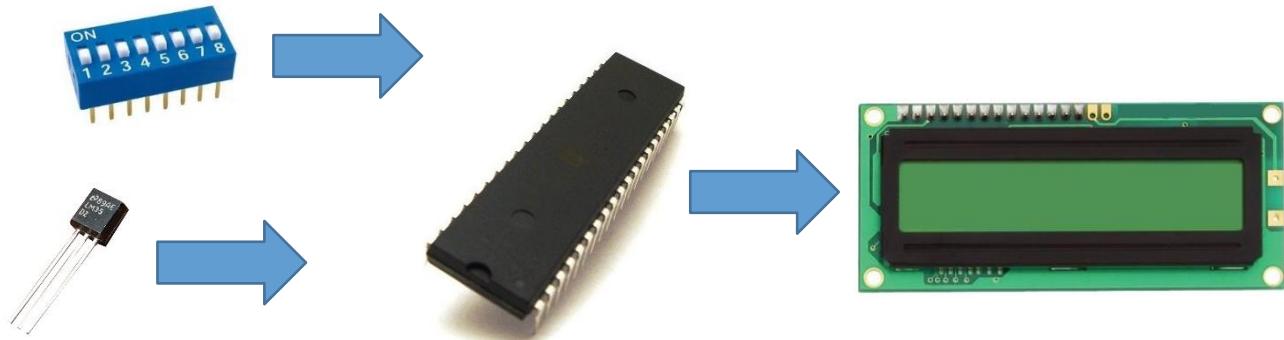


TERMÓMETRO DIGITAL [ADC]

Se desea diseñar un termómetro digital que medirá la temperatura detectada a través de un sensor (se puede usar el LM35 o cualquier otro de su preferencia). El circuito tendrá un dip switch a través del cual será posible seleccionar si se desea que la temperatura esté medida en °C o bien en °F (dependiendo de la posición en que ese dip switch se encuentre).

La temperatura será mostrada a través de un LCD en el cual se mostrará en la primera línea el mensaje “Temperatura” alineado al lado derecho y en la segunda línea aparecerá la temperatura seguida por °C o °F según sea lo que se esté midiendo.

Para este programa usted puede programar su propia librería para el LCD o puede bajar alguna que funcione (ese código no se revisará), sin embargo el resto de código deberá ser totalmente de su autoría.



MEDIDOR DE ÁNGULO [ADC]

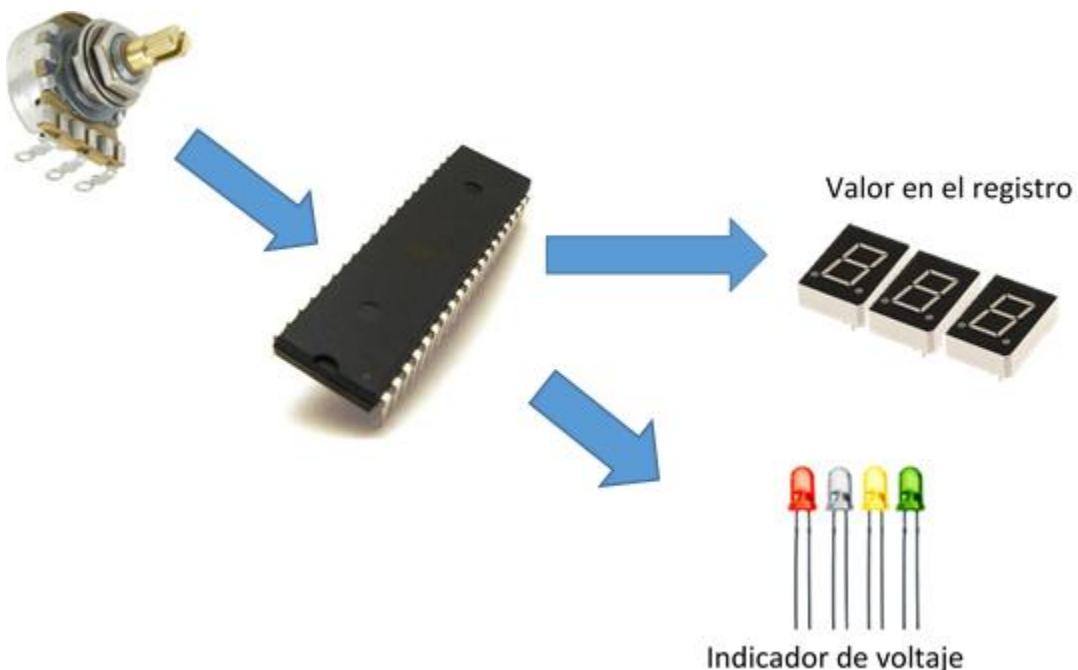
Se requiere conectar al microcontrolador cuatro LEDs (ordenados en hilera), un potenciómetro (que se describe más adelante), tres displays de siete segmentos, y se utilizará una entrada del ADC.

A la entrada del ADC se le proporcionará un voltaje variable que se obtendrá mediante un potenciómetro (un extremo conectado a 5V, el otro extremo conectado a tierra y la derivación conectada en el pin del microcontrolador configurado como entrada para el ADC), el potenciómetro deberá fijarse sobre una “estructura” con unos topes mecánicos que solo le permitan moverse 180°.

El programa que se desarrolle deberá revisar constantemente el voltaje que esté entrando por el pin del ADC y responderá de la siguiente forma:

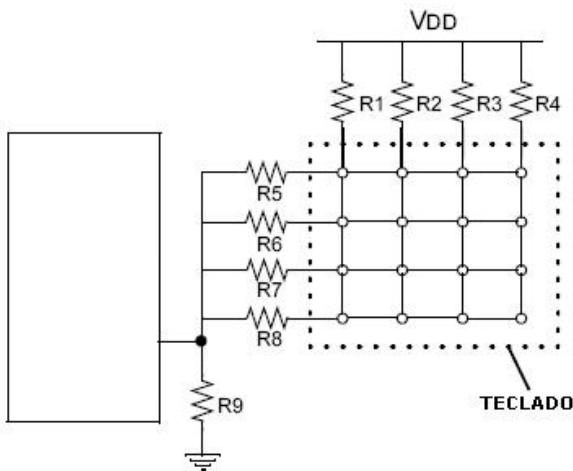
- Cuando el potenciómetro se encuentre entre 0° y 45° deberá encender únicamente un LED
- Cuando el potenciómetro se encuentre entre 45° y 90° deberán encender dos LEDs
- Cuando el potenciómetro se encuentre entre 90° y 135° deberán encender tres LEDs
- Cuando el potenciómetro se encuentre entre 135° y 180° deberán encender cuatro LEDs

Además de encender los LEDs en la forma en que se indica, en los displays de 7 segmentos deberá mostrarse en valor (en decimal... entre 000 y 255) que se está obteniendo de realizar la conversión (para ello se le sugiere que utilice la rutina que programó en el examen en la que un registro se convierte en centenas, decenas y unidades).



TECLADO MATRICIAL USANDO 1 PIN [ADC]

Cuando usted de encuentre desarrollando proyectos, habrá ocasiones en que necesitará utilizar muchos pines de entradas y de salidas, si a ello se le agrega el tener que emplear un teclado matricial utilizando los ocho bits que hasta ahora hemos empleado, entonces el número de pines para otras aplicaciones se ve drásticamente disminuido.



En esta práctica se desea lograr el control del teclado matricial que hemos venido empleando, pero utilizando únicamente un pin del microcontrolador. La clave está en colocar una red de resistencias de valores distintos y conocidos en las filas y columnas de manera que cada tecla configure un divisor de voltaje distinto (si lo desea puede usar potenciómetros). La señal obtenida será leída con un pin del ADC del microcontrolador y de esta forma podrá ser posible identificar la tecla presionada.

Cada círculo representa una de las teclas, que al presionar se une una de las resistencias de R1 a R4 conectadas a VDD con otra de R5 a R8 conectadas al microcontrolador. Así si presionamos en la tecla situada en la esquina superior izquierda se tendrá que VDD le llega al micro después de atravesar R1+R5. Si por el contrario presionamos la tecla inferior derecha la corriente llegará a través de la unión entre R4+R8. Siempre que presionemos una tecla cualquiera obtendremos un voltaje de caída entre la suma de dos resistencias R columna + R fila. Si la R9 no se pusiera, cuando se presionara ninguna tecla el microcontrolador no estaría conectado a nada, para evitar ese efecto se coloca R9 que mantendrá el pin conectado a GND mientras no se presione una tecla.

Esta configuración es conocida como Divisor de Tensión en la que tenemos una resistencia conectada a VDD y otra a GND y nosotros tomamos el valor del voltaje en la unión que hay entre ellas.

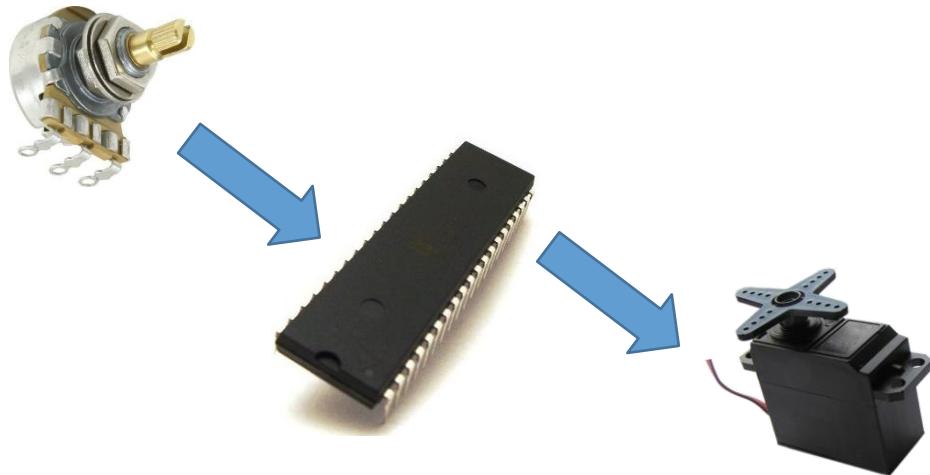
Se desea conectar el teclado al microcontrolador ATmega16A a través de un pin del ADC, se conectará también el LCD y cada vez que se presione una tecla, el símbolo correspondiente deberá aparecer en la parte superior derecha del teclado y desplazar hacia la izquierda a los caracteres previos. (Deberán funcionar todas las teclas).

SERVOMOTOR CON POSICIONAMIENTO MEDIANTE VOLTAJE ANALÓGICO [ADC]

Se requiere conectar al microcontrolador con un potenciómetro en forma de divisor de voltaje a través de algún pin que permita la conversión analógica digital, y con un servomotor en la salida del microcontrolador capaz de generar en forma automática un pulso de PWM.

Este circuito deberá funcionar de forma que al girar el potenciómetro entre su mínimo y su máximo voltaje, el servomotor se posicione en ángulos proporcionales entre su máximo y su mínimo, dependiendo de la cantidad de voltaje que se esté introduciendo en cada momento por medio del potenciómetro.

Por ejemplo, de acuerdo a como usted conecte su potenciómetro, el mínimo voltaje que llegará por el pin del ADC será de 1V y el máximo voltaje que llegará es de 4V, entonces cuando se esté introduciendo un voltaje de 1V el servomotor estará en su posición de 0°, cuando se introduzca un voltaje de 4V el servomotor estará en su posición de 180°, si se introduce un voltaje exactamente a la mitad entre 1V y 4V, es decir 2.5V entonces el servomotor se posicionaría a 90° y así irá variando en forma proporcional, pudiendo tomar cualquier posición dependiendo del voltaje introducido.



EEPROM

El microcontrolador ATmega16A contiene 512 bytes de memoria EEPROM, los cuales pueden ser leídos y/o escritos al menos 100,000 veces. La EEPROM es un tipo especial de memoria que se caracteriza por permanecer almacenada aún cuando el microcontrolador pierda su fuente de alimentación permitiendo así guardar datos en forma permanente, esto resulta sumamente útil en gran cantidad de aplicaciones.

A continuación se describen los registros que son empleados para el manejo de la memoria EEPROM

EEARH:EEARL – EEPROM ADDRESS REGISTER

Bit	15	14	13	12	11	10	9	8	EEARH	EEARL
	-	-	-	-	-	-	-	EEAR8		
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0		
	7	6	5	4	3	2	1	0		
Read/Write	R	R	R	R	R	R	R	R/W		
	R/W									
Initial Value	0	0	0	0	0	0	0	X		
	X	X	X	X	X	X	X	X		

Estos dos registros en conjunto corresponden a la dirección (que puede ir de 0 a 512) en donde un dato desea ser almacenado o de donde quiere ser leído. En el registro EEARH los bits 7:1 se encuentran reservados por lo cual deberán tener siempre el valor de 0. En el resto de los bits deberá especificarse en forma binaria la dirección a la cual se desea tener acceso, por ejemplo si se desea leer o escribir un dato en la dirección 500, esa dirección en forma binaria se representa como 0b1_1111_0100, por lo tanto el código correspondiente para cargar esa dirección correspondería a:

```
LDI R16, 0b0000_0001
OUT EEARH, R16
LDI R16, 0b1111_0100
OUT EEARL, R16
```

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
0	0	0	0	0	0	0		
-	-	-	-	-	-	-		EEAR8
								Bit más significativo de la dirección

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0
Ocho bits menos significativos correspondientes a la dirección EEPROM en donde quiere escribirse o leerse un dato							

EEDR – EEPROM DATA REGISTER

La memoria EEPROM puede utilizarse llevando a cabo dos tipos de operaciones: lectura y escritura.

Cuando un dato va a ser escrito en la memoria EEPROM, deberá almacenarse previamente en el registro EEDR para posteriormente ser transferido a la memoria.

Cuando un dato es leído de la memoria éste quedará almacenado en el registro EEDR.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
MSB							LSB
Dato que será escrito o que fue leído en o de la memoria EEPROM							

EECR – EEPROM CONTROL REGISTER

Bit 7..4 – Bits reservados

Siempre tendrán el valor de 0

Bit 3 – EERIE: EEPROM Ready Interrupt Enable

Este bit es utilizado para activar o desactivar la interrupción encargada de avisar que la EEPROM se encuentra lista para ejecutar una nueva operación de lectura o escritura. Dicha interrupción se genera en forma constante cuando EEWE se encuentra inactivo. Cuando EERIE es configurado como 1 la interrupción se encontrará activa, en cambio si EERIE es configurado como 0 entonces la interrupción estará inactiva y nunca se generará.

Bit 2 – EEMWE: EEPROM Master Write Enable

Mediante este bit es posible controlar en forma general el acceso a la escritura de la EEPROM, cuando EEMWE se configura como 1 es posible utilizar el bit 1 (EEWE) para indicar a la memoria que realice una operación de escritura, sin embargo cuando EEMWE se configura como 0 aún cuando se indique una operación de escritura a través del bit EEWE ésta no se llevará a cabo.

Es importante hacer notar que cuando EEMWE es configurada como 1 a través del software éste volverá a ser 0 después de 4 ciclos de reloj.

Bit 1 – EEWE: EEPROM Write Enable

Cuando se desea guardar un dato en la memoria EEPROM, una vez que la dirección y los datos han sido especificados en los registros correspondientes, es necesario escribir un 1 en el bit EEMWE y posteriormente escribir también un 1 en el bit EEWE, de esta forma el dato será guardado en la memoria. Un poco más adelante se presentará un diagrama de flujo que corresponde al proceso de escritura en la memoria EEPROM.

Es importante hacer notar que si después de poner el bit EEMWE=1 y exactamente antes de poner EEWE=1 se llevase a cabo una interrupción (de timer, externa, ADC, o cualquier otra) esto podría ocasionar que el proceso de escritura no se realice correctamente es por ello que resulta recomendable que al realizar el proceso de escritura se deshabiliten temporalmente las interrupciones con la finalidad de evitar fallas.

Cuando el proceso de escritura termina el bit EEWE vuelve a ser 0 en forma automática.

Bit 0 – EERE: EEPROM Read Enable

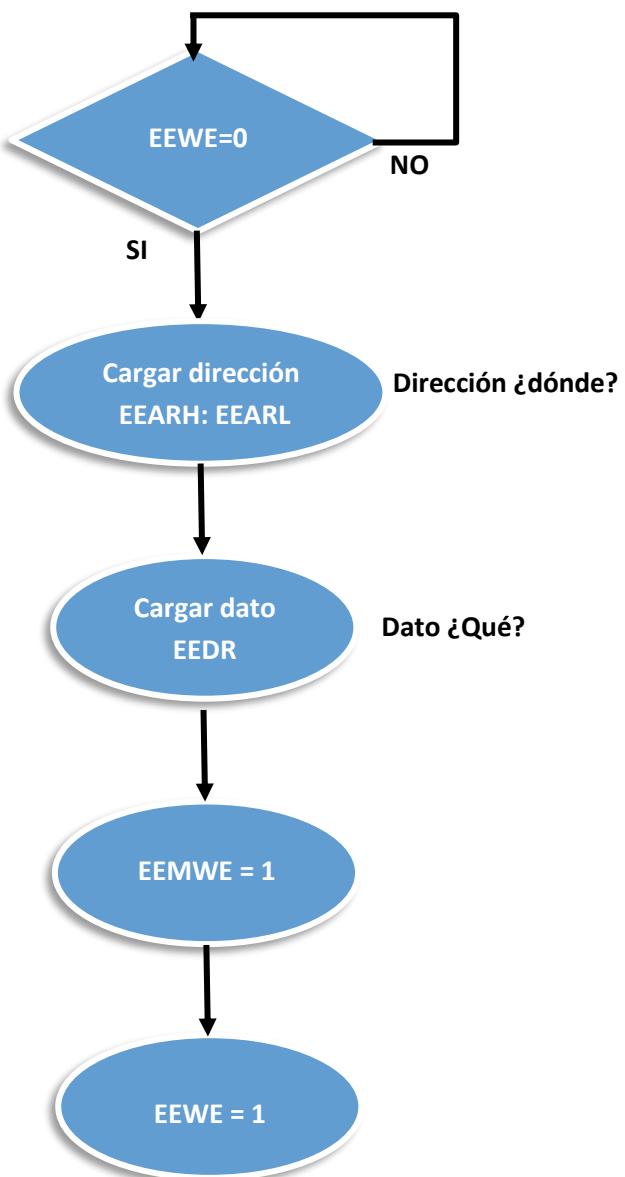
Cuando se desea leer un dato de la memoria EEPROM el bit EERE deberá configurarse como un 1 y esto activará el proceso de lectura de la dirección que se haya especificado con anterioridad. Es importante que al realizar la operación de lectura se garantice que no haya una operación de escritura en proceso (verificando en bit EEWE) pues ello impediría llevar a cabo la lectura en forma correcta.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	0	0				
				EERIE	EEMWE	EEWE	EERE
				1 = Activar interrupción de aviso de que EEPROM se	1 = Habilitar la escritura (general)	1-Escribir el dato en la EEPROM (Previamente se deben	1-Lee el dato de la dirección

				encuentra listo 0= Desactivar interrupción	de datos en EEPROM 0= Deshabilitar escritura de datos	haber almacenado el dato, la dirección y EEMWE=1)	0- Deshabilita la escritura de datos
--	--	--	--	---	---	---	---

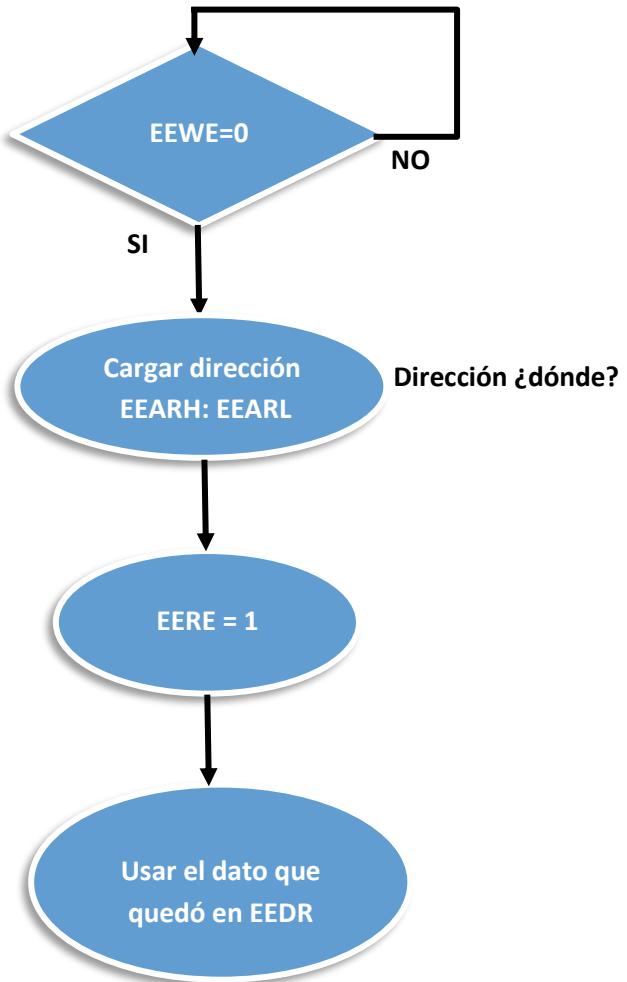
PROCESO DE ESCRITURA EN LA MEMORIA EEPROM

El siguiente diagrama de flujo muestra los pasos necesarios para llevar a cabo una operación de escritura en la EEPROM del microcontrolador



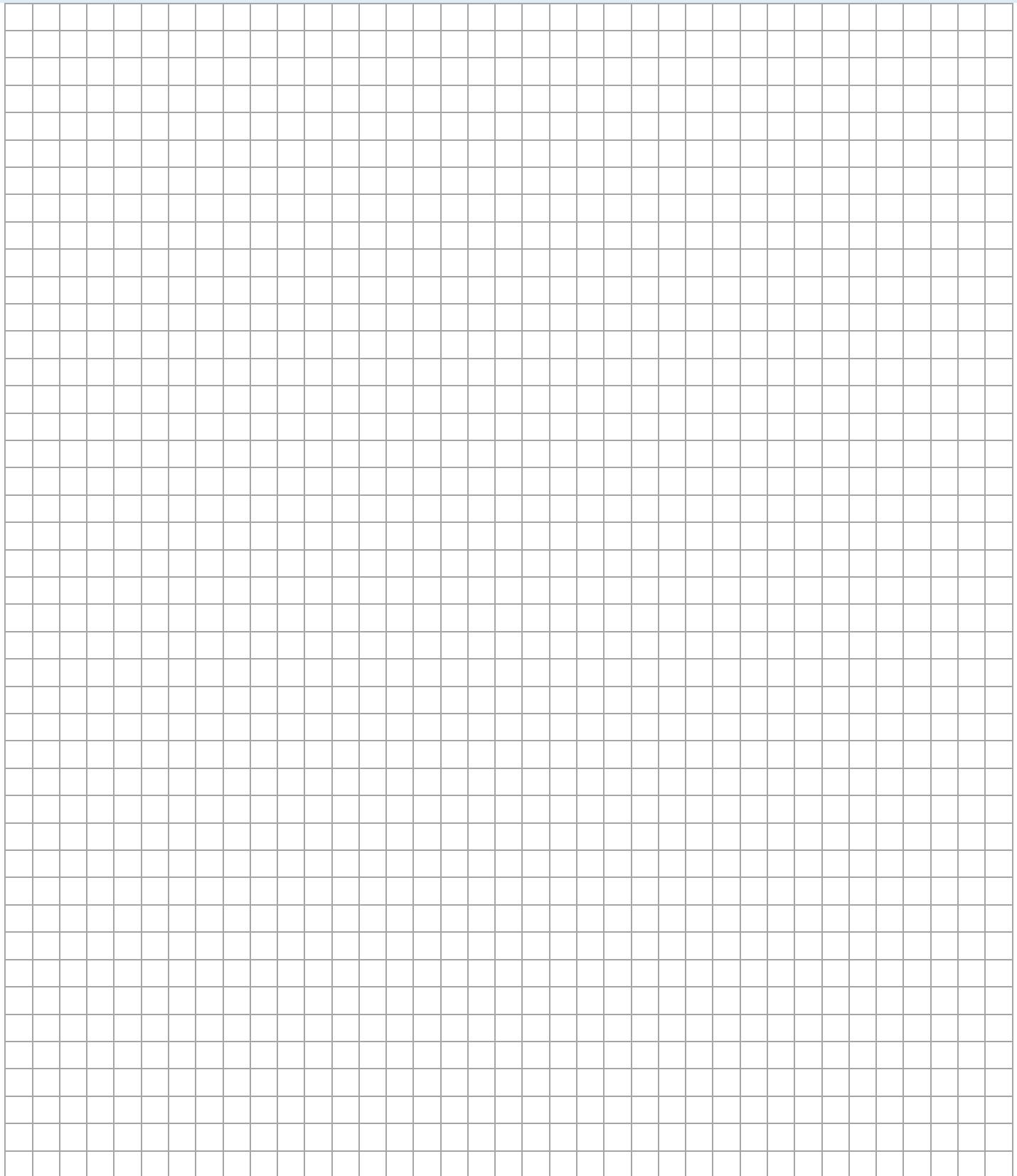
PROCESO DE LECTURA EN LA MEMORIA EEPROM

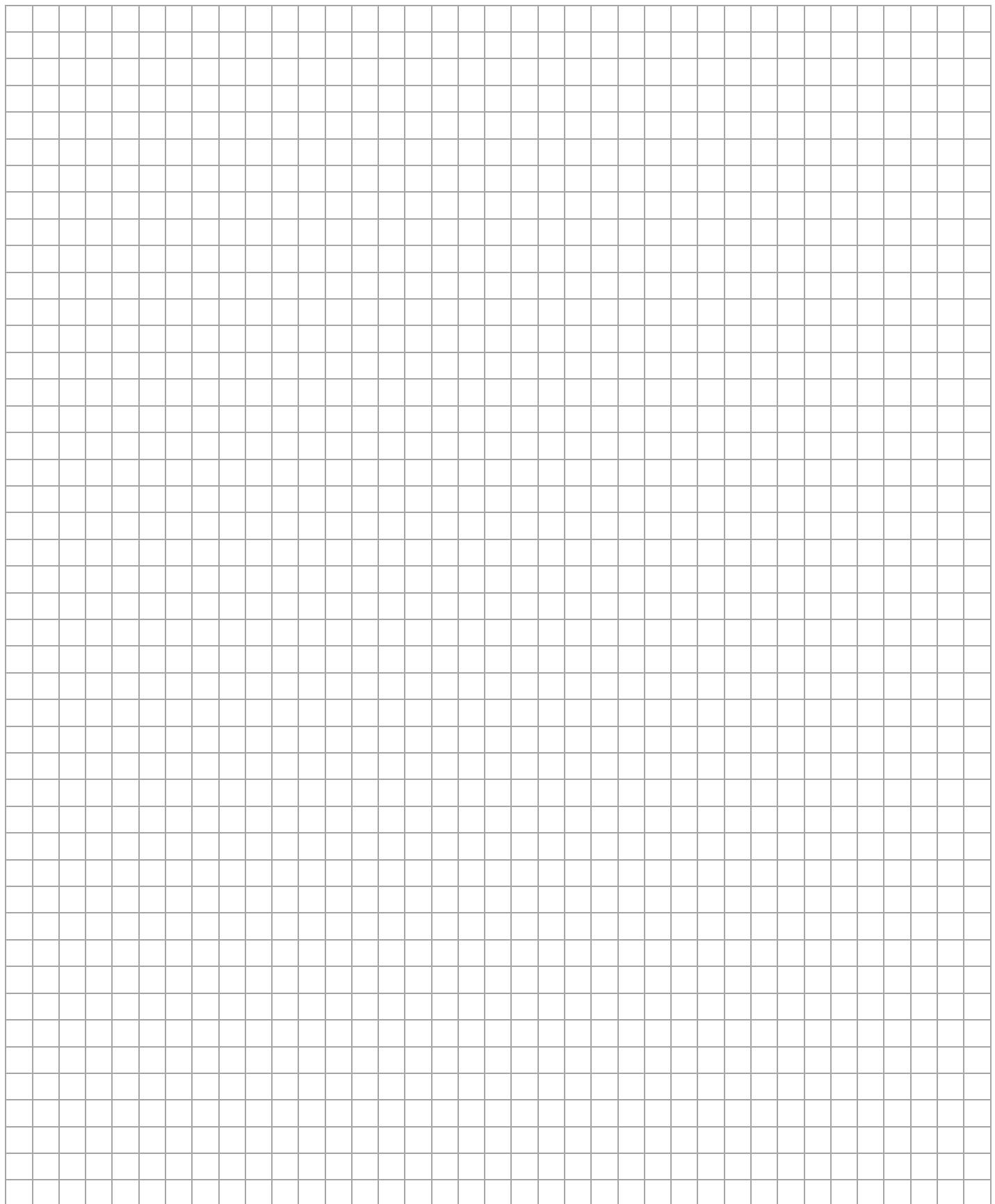
El siguiente diagrama de flujo muestra los pasos necesarios para llevar a cabo una operación de lectura en la memoria EEPROM del microcontrolador.

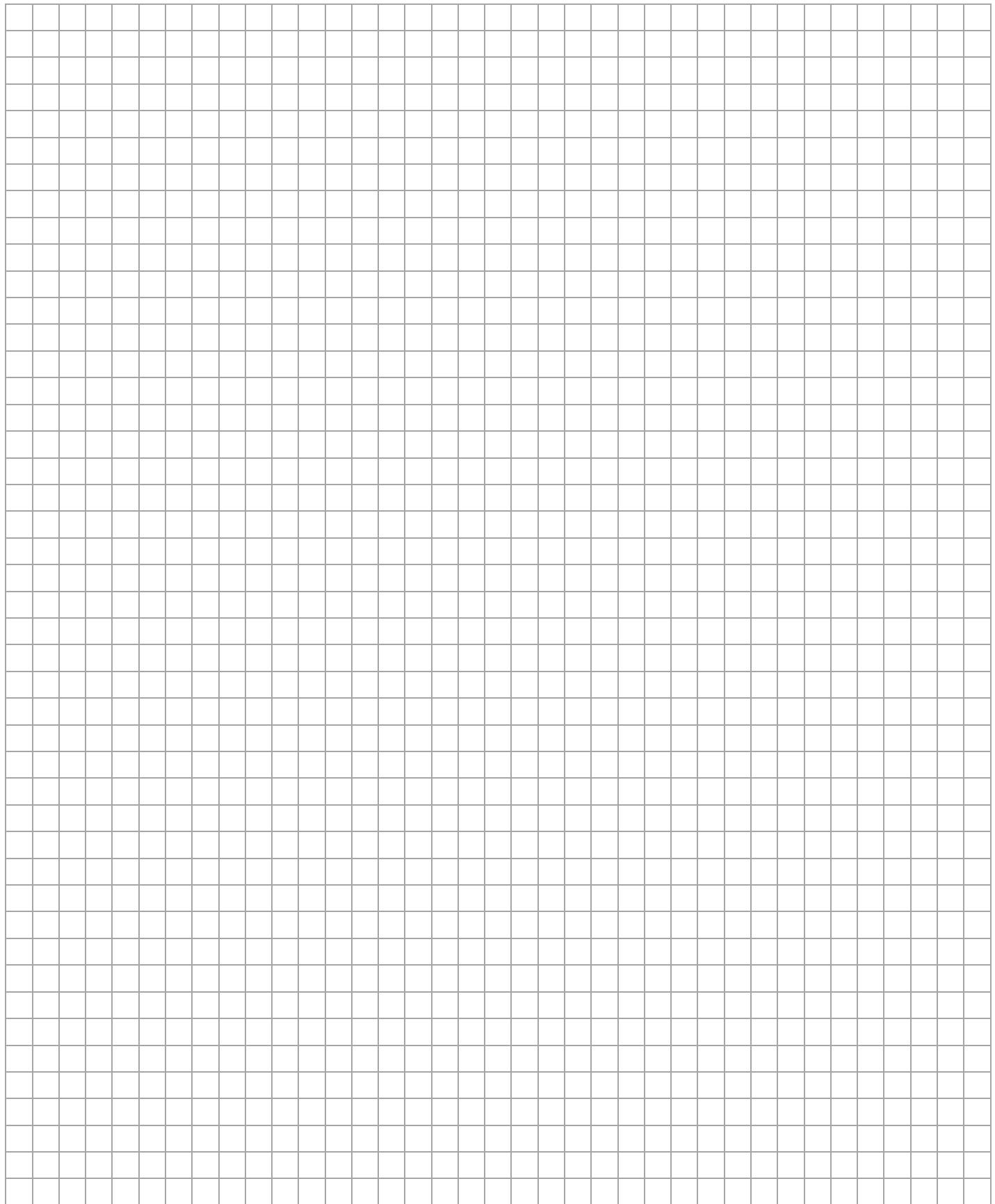


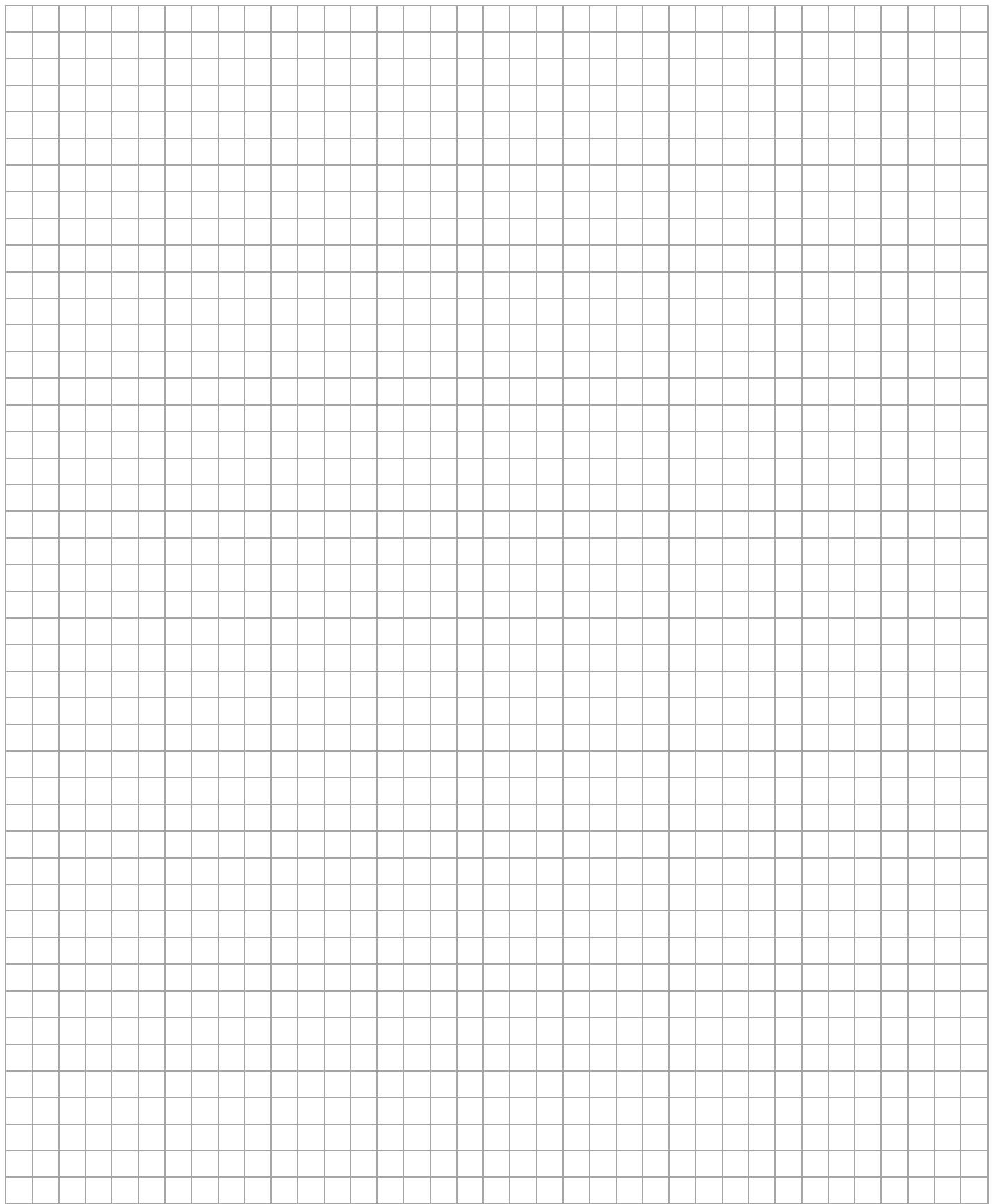
Vale la pena hacer notar que una vez que el dato se leído se encuentra en el registro EEDR, este deberá almacenarse en un registro de uso temporal a través de la instrucción **IN R__, EEDR** para posteriormente darle el uso que se requiera.

NOTAS PERSONALES – EEPROM

A large grid of squares, approximately 20 columns by 30 rows, designed for writing personal notes or sketches.

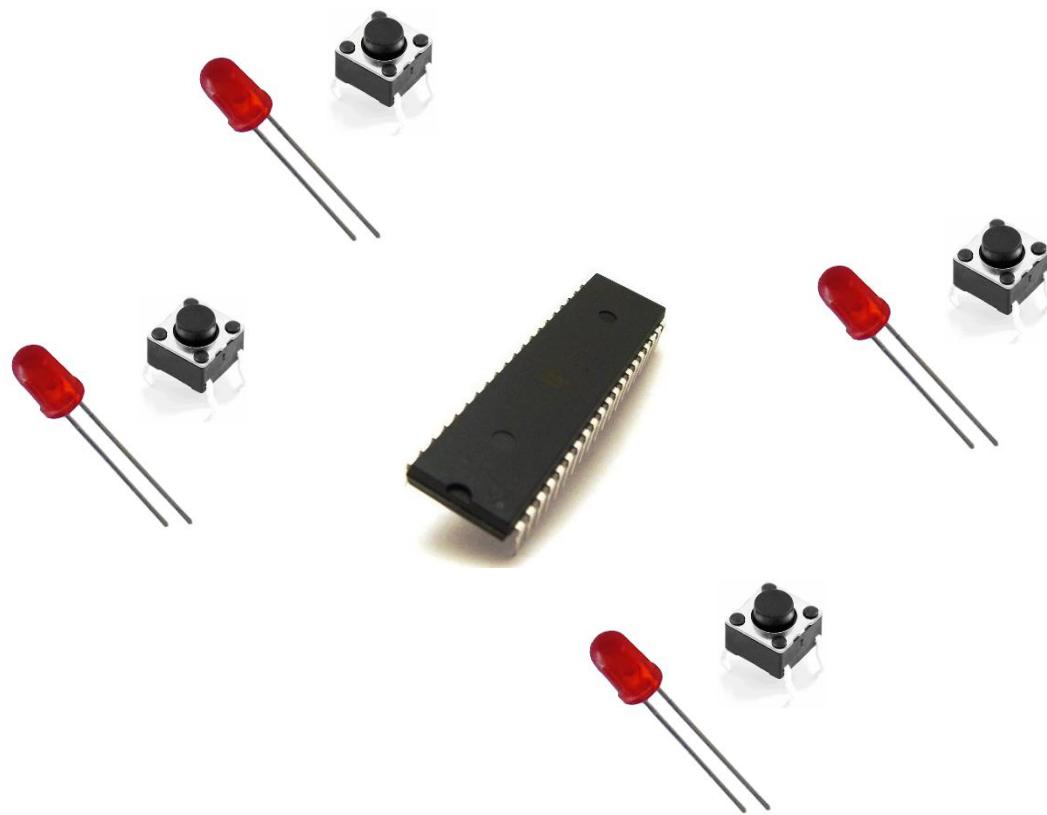






GUARDAR ESTADO DE LEDS [EEPROM]

Se le pide que conecte al microcontrolador un botón y un LED en cada uno de los cuatro puertos (A,B,C,D). Al encender por primera vez el dispositivo todos los LEDs deberán encontrarse apagados. Cada vez que el botón correspondiente a un LED sea presionado su estado deberá cambiar (apagado→encendido, encendido→apagado) y deberá irse guardando en la EEPROM de forma tal que aún cuando en dispositivo se desconecte de la energía eléctrica, si este se vuelve a energizar los LEDs deberán aparecer en el estado en el que se quedaron antes de perder la energía.



Fundamentos de comunicación serial

COMUNICACIÓN SERIAL ASÍNCRONA

Una manera de conectar dos dispositivos es realizando una comunicación serial asíncrona. En este tipo de comunicación los bits de datos se transmiten "en serie" (uno de tras de otro) y cada dispositivo tiene su propio reloj (por ello se denomina asíncrona). Con la finalidad de poder llevar a cabo este tipo de comunicaciones es necesario que con anterioridad se hayan configurado ambos dispositivos para la transmisión de datos a la misma velocidad (o de lo contrario no será posible que se comuniquen en forma adecuada).

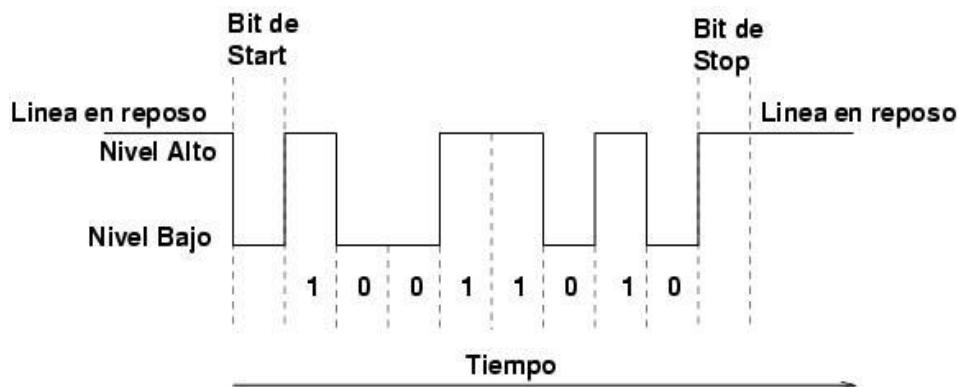
Cada vez que se envía un dato utilizando este tipo de comunicación, los dispositivos se encargan automáticamente de encapsular los datos en serie en tramas de la forma:



Primero se envía un bit de start, a continuación los bits de datos (primero el bit de mayor peso) y finalmente los bits de STOP.

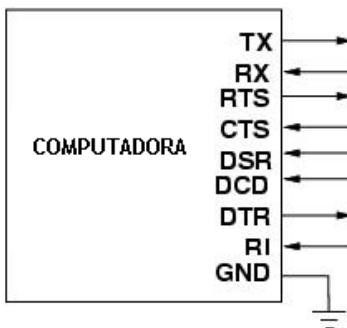
El número de bits de datos y de bits de Stop es uno de los parámetros configurables, así como el criterio de paridad par o impar para la detección de errores. Normalmente, las comunicaciones serie tienen los siguientes parámetros: 1 bit de Start, 8 bits de Datos, 1 bit de Stop y sin paridad.

En esta figura se puede ver un ejemplo de la transmisión del dato binario 10011010. La línea en reposo está a nivel alto:



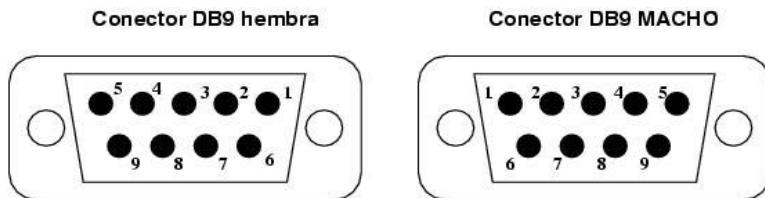
RS232

La Norma RS-232 fue definida para conectar una computadora a un modem. Además de transmitirse los datos de una forma serie asíncrona son necesarias una serie de señales adicionales, que se definen en la norma. Los voltajes empleados están comprendidos entre +15/-15 volts.



EL CONECTOR DB9

En la computadora normalmente se encuentra un conector DB9 macho, de 9 pines, por el que se conectan los dispositivos al puerto serie.

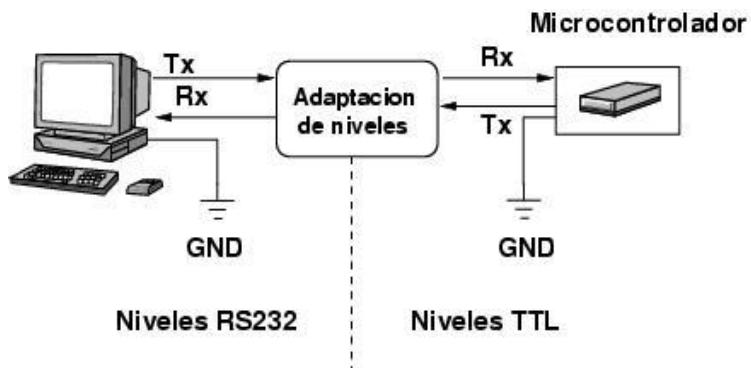


La información asociada a cada uno de los pines es la siguiente:

Número de pin	Señal
1	DCD (Data Carrier Detect)
2	RX
3	TX
4	DTR (Data Terminal Ready)
5	GND
6	DSR (Data Sheet Ready)
7	RTS (Request To Send)
8	CTS (Clear To Send)
9	RI (Ring Indicator)

CONEXIÓN DEL MICROCONTROLADOR AVR AL PUERTO SERIAL DE LA COMPUTADORA

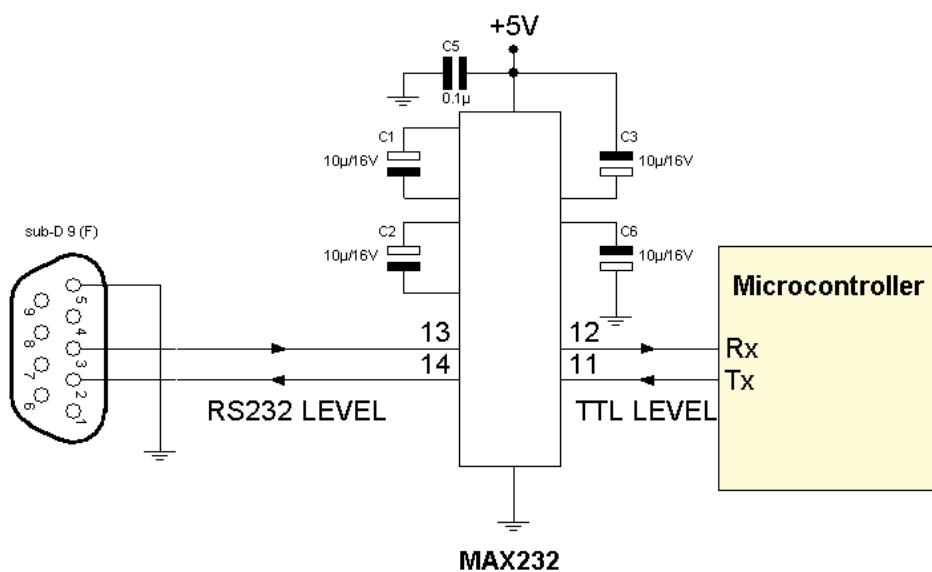
Para conectar un microcontrolador con la computadora, a través del puerto serial, se utilizan las señales Tx, Rx y GND. La computadora utiliza la norma RS232, por lo que los niveles de voltaje de los pines están comprendidos entre +15V y -15V. Los microcontroladores normalmente trabajan con niveles TTL (de 0V a 5V). Es necesario por tanto contar con un circuito que se encargue de adaptar los niveles de voltaje.



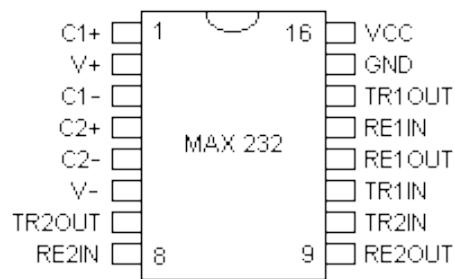
El circuito que normalmente se utiliza con esta finalidad es el MAX232

El circuito integrado MAX 232

Este circuito integrado permite adaptar los niveles RS232 y TTL, haciendo posible conectar un puerto serial de la computadora con un microcontrolador. Sólo es necesario este chip y 4 capacitores electrolíticos. El esquema es el siguiente:

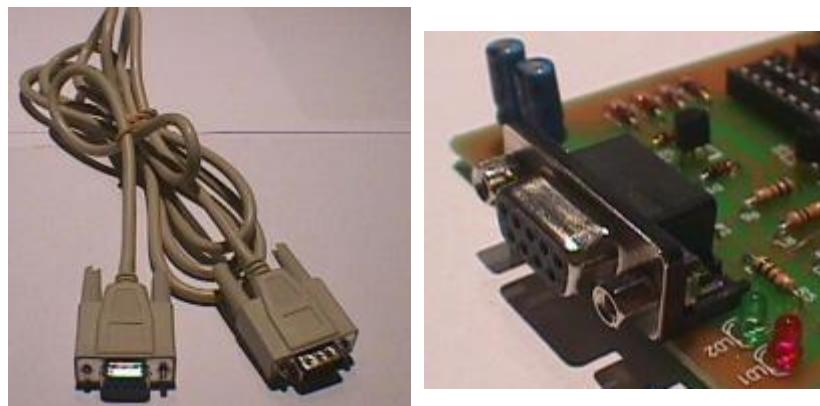


Para facilitar la conexión del circuito, a continuación se muestra la distribución de pines del MAX232:

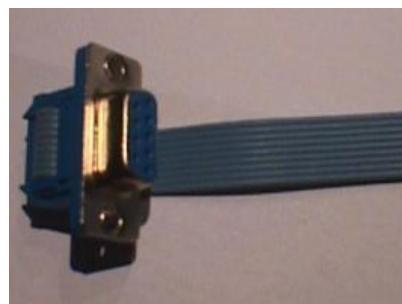


CABLE DE CONEXIÓN

Para realizar la conexión entre la computadora y nuestro circuito podemos usar diferentes alternativas. Una de ellas es utilizar un cable serie macho-hembra no cruzado, y en el circuito un conector hembra db9 para circuito impreso:



Cuando conectamos un microcontrolador a la computadora, normalmente sólo usamos los pines TX, RX y GND, sin embargo, en este tipo de cables cuando ya vienen fabricados tienen en su interior los 9 pines interconectados. Por ello puede resultar útil el utilizar otro tipo de cable o se puede fabricar un cable serie utilizando cable plano de bus y un conector db9 hembra:



USART (Universal Synchronous and Asynchronous serial receiver and transmitter)

El USART del AVR ATmega16A puede funcionar de cuatro modos diferentes según la configuración de su reloj, estos modos son:

- Normal Asíncrono
- Asíncrono de doble velocidad
- Síncrono Maestro
- Síncrono Esclavo

Para hacer la selección entre comunicación síncrona y asíncrona se debe configurar el bit UMSEL en el registro UCSRC, configurándolo como 0 funcionará de forma asíncrona, mientras que si se configura con un 1 será en forma síncrona. Para configurar doble velocidad se deberá activar el bit U2X del registro UCSRC (en caso de que se esté trabajando en forma asíncrona). Si lo que se desea es hacer la configuración de comunicación síncrona, para especificar la función como Maestro o esclavo se configura el pin XCK para controlar si la fuente del reloj será interna (Maestro) o externa (esclavo). Cabe resaltar que el pin XCK solo está activo cuando se configura en modo síncrono.

UDR - USART I/O DATA REGISTER

Bit	7	6	5	4	3	2	1	0	UDR (Read)	UDR (Write)
	RXB[7:0]									
	TXB[7:0]									
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	UDR (Read)	UDR (Write)
Initial Value	0	0	0	0	0	0	0	0		

Este registro es utilizado como buffer para transmisión o recepción, es decir que en este registro deben ponerse los datos que se desea enviar por el serial, o bien en este registro pueden leerse los datos recibidos. Este buffer únicamente puede ser leído o modificado cuando la bandera UDRE del registro UCSRA está en 1.

Si se trata de escribir un dato y la bandera no está habilitada el dato será ignorado. Si se escribe un dato este es almacenado en el registro de transmisión serial y será enviado a través del pin TxD.

El buffer de recepción consiste en un FIFO de dos niveles, el cual cambiará de estado cada vez que se accese al buffer, es por ello que resulta importante no usar instrucciones como SBI o CBI en este registro. Y debe de tenerse también cuidado en el uso de instrucciones como SBIC o SBIS puesto que también cambiarán el contenido del FIFO.

UCSRA – USAR CONTROL AND STATUS REGISTER A

Bit	7	6	5	4	3	2	1	0	UCSRA
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	
Read/Write	R	R/W	R	R	R	R	R/W	R/W	

Initial Value

0

0

1

0

0

0

0

0

0

UCSRA

Bit 7 – RXC: USART Receive Complete

Esta bandera permanecerá activada cuando haya datos sin leer en el buffer de recepción, y solamente quedará limpia cuando los datos hayan sido leídos en su totalidad. Esta bandera puede ser empleada para activar la interrupción de Recepción (vea la descripción del pin RXCIE)

Bit 6 – TXC: USART Transmit Complete

Esta bandera se activa cuando la totalidad de los datos han sido enviados y no hay datos nuevos en el buffer de transmisión. Automáticamente se limpia cuando la interrupción de Transmisión se ejecuta o puede limpiarse escribiéndole un 1. Esta bandera puede generar una interrupción (vea la descripción del pin TXCIE)

Bit 5 – UDRE: USART Data Register Empty

Esta bandera nos indica que el registro de transmisión UDR está listo para recibir nuevos datos, si UDRE es 1, el buffer está vacío y por lo tanto pueden enviarse nuevos datos. Esta bandera puede generar una interrupción (vea la descripción del pin UDRIE)

Bit 4 – FE: Frame Error

Si existe un error en los datos que se reciben se habilita este bit.

Bit 3 – DOR: Data OverRun

Cuando se detecta un “Overrun” este bit cambiará a uno. Esto ocurre cuando el buffer de recepción está lleno y hay un byte en espera de ser recibido.

Bit 2 – PE: Parity Error

Si se detecta un error de paridad en la recepción se habilitará este pin. (Solamente funciona si el chequeo de paridad se encuentra habilitado (UPM1=1)

Bit 1 – U2X: Double the USART Transmission Speed

Este bit solamente funciona cuando se está empleando la transmisión asíncrona, y debe quedar como 0 cuando se está empleando operación síncrona.

Si el contenido de este bit es un 1, se reducirá el divisor para la tasa de transferencia de datos de 16 a 8, lo cual producirá una comunicación asíncrona al doble de velocidad.

Bit 0 – MPCM: Multi-processor Communication Mode

Con este bit se puede habilitar el modo de Multi-procesadores. De manera que todas las señales que se reciban y que no contengan la información relativa a la dirección de la que provienen, serán ignoradas. (El funcionamiento de la transmisión de señales no cambia). Para utilizar este modo es necesario entender completamente su funcionamiento, para lo cual puede consultar la página 162 del datasheet del MEGA16A.

UCSRA

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	1	0	0	0	0	0
RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM
Bandera Se activa cuando hay datos recibidos sin leer.	Bandera Se activa cuando se han terminado de enviar todos los datos.	Bandera Se activa si el registro de transmisión está listo para más datos.	Se activa cuando ha habido un error en la recepción de datos.	Bandera Se activa cuando el buffer de recepción está lleno y llega otro dato.	Bandera Se activa cuando se detecta un error en la paridad	Si Síncrona=0 Si Asíncrona: 0=velocidad normal 1=doble velocidad	0 – Modo normal. 1 – Modo multi procesador.

USCRB – USART CONTROL AND STATUS REGISTER B

Bit	7	6	5	4	3	2	1	0	USCRB
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	

Bit 7 – RXCIE: RX Complete Interrupt Enable

Escribiéndole un 1 a este bit se habilitará la interrupción generada por la bandera RXC, es decir que se generará una interrupción cada vez que se reciba un dato. (Cabe hacer mención de que esta interrupción funcionará únicamente si se tienen habilitadas las interrupciones, lo cual se realiza con la instrucción sei)

Bit 6 – TXCIE: TX Complete Interrupt Enable

Si se escribe un 1 en este bit se habilitará la interrupción generada por la bandera TXC, lo que quiere decir que se generará una interrupción cada vez que la totalidad de los datos hayan sido enviados y no haya en el buffer ninguna información pendiente para enviar. (Cabe hacer mención de que esta interrupción funcionará únicamente si se tienen habilitadas las interrupciones, lo cual se realiza con la instrucción sei)

Bit 5 – UDRIE: USART Data Register Empty Interrupt Enable

Escribiendo un 1 a este bit se habilita la interrupción generada por la bandera UDRE. Cabe hacer mención de que esta interrupción funcionará únicamente si se tienen habilitadas las interrupciones, lo cual se realiza con la instrucción sei)

Bit 4 – RXEN: Receiver Enable

Sirve para habilitar la recepción de datos en el pin RxD, si este pin tiene un 0, entonces no podrán recibirse datos ni funcionarán las banderas FE, DOR ni PE.

Bit 3 – TXEN: Transmitter Enable

Este bit es empleado para habilitar la transmisión de datos. Cuando se escribe un 1 en este pin se habilita el pin TxD del microprocesador.

Bit 2 – UCSZ2: Carácter Size

Los registros UCSZ2:0 se emplean para configurar el número de bits de datos que se emplearán en la transmisión y en la recepción. En este registro solamente se encuentra el UCSZ2, mientras que el UCSZ1 y UCSZ0 se encuentran en el registro UCSRC

Table 67. UCSZ Bits Settings

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

Bit 1 – RXB8: Receive Data Bit 8

Si se está trabajando con datos de 9 bits, este será el noveno bit de la recepción, debe ser leído antes de consultar el registro UDR.

Bit 0 – TXB8: Transmit Data Bit 8

Si se está trabajando con datos de 9 bits, este será el noveno bit de la transmisión, debe ser escrito antes de consultar el registro UDR.

UCSRB

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	0	0	0	0	0	0
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB0
1 – interrupción cada vez que recibe un dato. 0 – sin interrupción.	1 – interrupción cada vez que termina envío de datos.. 0 – sin interrupción.	1- interrupción cuando esté listo para enviar más datos. 0 – sin interrupción.	0 – no recibe datos por serial. 1 – habilita recepción de datos.	0 – no envía datos por serial. 1 – habilita envío de datos.	Configuración del número de bits que se envían o reciben. Ver tabla 67.	Si se usan 9 bits este el noveno bit de recepción	Si se usan 9 bits este el noveno bit de envío.

UCSRC – CONTROL AND STATUS REGISTER C

Bit	7	6	5	4	3	2	1	0	UCSRC
Read/Write	R/W								
Initial Value	1	0	0	0	0	1	1	0	

Bit 7 – URSEL: Register Select

Este bit selecciona sirve para seleccionar entre el acceso al registro UCSRC y el registro UBRRH. Cuando se va a escribir algún bit del registro UCSRC este bit deberá contener un 1.

Bit 6 – UMSEL: USART Mode Select

Este bit se utiliza para seleccionar entre la operación síncrona o asíncrona del USART del microprocesador.

Table 64. UMSEL Bit Settings

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

Bit 5:4 – UPM 1:0 Parity Mode

Estos bits sirven para configurar el USART de modo que se genere paridad y se realice el chequeo de la misma. Si se encuentra habilitada, el microprocesador automáticamente generará y enviará los bits de paridad correspondientes a cada dato, y realizará las comparaciones pertinentes para saber si los datos recibidos se encuentran íntegros.

Table 65. UPM Bits Settings

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

Bit 3 – USBS: Stop Bit Select

Este bit selecciona los números de bits de “Stop” que serán enviados por el transmisor.

Table 66. USBS Bit Settings

USBS	Stop Bit(s)
0	1-bit
1	2-bit

Bit 2:1 – UCSZ1:0 Character Size

Estos bit, combinados con el bit UCSZ2 que ya se explicó con anterioridad, indican cuantos bits serán empleados para la transmisión y la recepción de información:

Bit 0 – UCPOL: Clock Polarity

Este bit se emplea únicamente para la transmisión síncrona, debe quedar como 0 cuando se está empleando comunicación asíncrona.

Table 68. UCPOL Bit Settings

UCPOL	Transmitted Data Changed (Output of TxD Pin)	Received Data Sampled (Input on RxD Pin)
0	Rising XCK Edge	Falling XCK Edge
1	Falling XCK Edge	Rising XCK Edge

UCSRC

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
1	0	0	0	0	1	1	0
URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL
Si quiero usar UCSRC debe ser igual a 1	0 = Modo asíncrono 1= Modo síncrono	Configurar bits de paridad Ver tabla 65	Configurar bits de paridad Ver tabla 65	Número de bits de "stop" 0 = 1 bit 1 = 2 bits	Configuración del número de bits que se envían o reciben. Ver tabla 67.	Configuración del número de bits que se envían o reciben. Ver tabla 67.	Sólo se usa si es transmisión síncrona. Ver tabla 68.

USART BAUD RATE REGISTERS – UBRRH AND UBRRH

Bit	15	14	13	12	11	10	9	8	URSEL	–	–	–	UBRR[11:8]	UBRRH
	UBRR[7:0]												UBRRL	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	7	6	5	4	3	0
	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	0	0	0	0	0	0
	0	0	0	0	0	0	0	0						

Note: 1. The UBRRH Register shares the same I/O location as the UCSRC Register. See the "Accessing UBRRH/UCSRC Registers" on page 163 section which describes how to access this register.

Bit 15 – URSEL: Register Select

Este bit selecciona entre el acceso al registro UBRRH y el UCSRC. URSEL deberá ser un 0 cuando se vaya a escribir información en el registro UBRRH.

Bits 14:12 – Reservados

No son importantes para la comunicación USART

Bits 11:0 – UBRR11:0 USART Baud Rate Register

Estos bits contienen la información referente a la tasa de transferencia con la que trabajará el USART.

UBRRH

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	0	0	0	0	0	0
URSEL	-	-	-	UBRR11	UBRR10	UBRR9	UBRR8
Si quiero usar UBRRH debe ser igual a 0	No se emplea.	No se emplea.	No se emplea.	Bit 11 de la tasa de transferencia. Ver tablas 69ss.	Bit 10 de la tasa de transferencia. Ver tablas 69ss.	Bit 9 de la tasa de transferencia. Ver tablas 69ss.	Bit 8 de la tasa de transferencia. Ver tablas 69ss.

UBRRL

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0							
UBRR7	UBRR6	UBRR5	UBRR4	UBRR3	UBRR2	UBRR1	UBRR0
Bit 7 de la tasa de transf. Ver tablas 69ss.	Bit 6 de la tasa de transf. Ver tablas 69ss.	Bit 5 de la tasa de transf. Ver tablas 69ss.	Bit 4 de la tasa de transf. Ver tablas 69ss.	Bit 3 de la tasa de transf. Ver tablas 69ss.	Bit 2 de la tasa de transf. Ver tablas 69ss.	Bit 1 de la tasa de transf. Ver tablas 69ss.	Bit 0 de la tasa de transf. Ver tablas 69ss.

En las siguientes páginas se muestran algunas tablas con los valores que deberán cargarse al registro UBRR en función de la tasa de transferencia deseada y del valor configurado en el bit U2X.

Table 69. Examples of UBRR Settings for Commonly Used Oscillator Frequencies

Baud Rate (bps)	$f_{osc} = 1.0000$ MHz				$f_{osc} = 1.8432$ MHz				$f_{osc} = 2.0000$ MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	25	0.2%	51	0.2%	47	0.0%	95	0.0%	51	0.2%	103	0.2%
4800	12	0.2%	25	0.2%	23	0.0%	47	0.0%	25	0.2%	51	0.2%
9600	6	-7.0%	12	0.2%	11	0.0%	23	0.0%	12	0.2%	25	0.2%
14.4k	3	8.5%	8	-3.5%	7	0.0%	15	0.0%	8	-3.5%	16	2.1%
19.2k	2	8.5%	6	-7.0%	5	0.0%	11	0.0%	6	-7.0%	12	0.2%
28.8k	1	8.5%	3	8.5%	3	0.0%	7	0.0%	3	8.5%	8	-3.5%
38.4k	1	-18.6%	2	8.5%	2	0.0%	5	0.0%	2	8.5%	6	-7.0%
57.6k	0	8.5%	1	8.5%	1	0.0%	3	0.0%	1	8.5%	3	8.5%
76.8k	—	—	1	-18.6%	1	-25.0%	2	0.0%	1	-18.6%	2	8.5%
115.2k	—	—	0	8.5%	0	0.0%	1	0.0%	0	8.5%	1	8.5%
230.4k	—	—	—	—	—	—	0	0.0%	—	—	—	—
250k	—	—	—	—	—	—	—	—	—	—	0	0.0%
Max ⁽¹⁾	62.5 kbps		125 kbps		115.2 kbps		230.4 kbps		125 kbps		250 kbps	

Table 70. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	$f_{osc} = 3.6864$ MHz				$f_{osc} = 4.0000$ MHz				$f_{osc} = 7.3728$ MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	95	0.0%	191	0.0%	103	0.2%	207	0.2%	191	0.0%	383	0.0%
4800	47	0.0%	95	0.0%	51	0.2%	103	0.2%	95	0.0%	191	0.0%
9600	23	0.0%	47	0.0%	25	0.2%	51	0.2%	47	0.0%	95	0.0%
14.4k	15	0.0%	31	0.0%	16	2.1%	34	-0.8%	31	0.0%	63	0.0%
19.2k	11	0.0%	23	0.0%	12	0.2%	25	0.2%	23	0.0%	47	0.0%
28.8k	7	0.0%	15	0.0%	8	-3.5%	16	2.1%	15	0.0%	31	0.0%
38.4k	5	0.0%	11	0.0%	6	-7.0%	12	0.2%	11	0.0%	23	0.0%
57.6k	3	0.0%	7	0.0%	3	8.5%	8	-3.5%	7	0.0%	15	0.0%
76.8k	2	0.0%	5	0.0%	2	8.5%	6	-7.0%	5	0.0%	11	0.0%
115.2k	1	0.0%	3	0.0%	1	8.5%	3	8.5%	3	0.0%	7	0.0%
230.4k	0	0.0%	1	0.0%	0	8.5%	1	8.5%	1	0.0%	3	0.0%
250k	0	-7.8%	1	-7.8%	0	0.0%	1	0.0%	1	-7.8%	3	-7.8%
0.5M	—	—	0	-7.8%	—	—	0	0.0%	0	-7.8%	1	-7.8%
1M	—	—	—	—	—	—	—	—	—	—	0	-7.8%
Max ⁽¹⁾	230.4 kbps		460.8 kbps		250k bps		0.5 Mbps		460.8 kbps		921.6 kbps	

Table 71. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	$f_{osc} = 8.0000$ MHz				$f_{osc} = 11.0592$ MHz				$f_{osc} = 14.7456$ MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	—	—	2	-7.8%	1	-7.8%	3	-7.8%
1M	—	—	0	0.0%	—	—	—	—	0	-7.8%	1	-7.8%
Max ⁽¹⁾	0.5 Mbps		1 Mbps		691.2 kbps		1.3824 Mbps		921.6 kbps		1.8432 Mbps	

Table 72. Examples of UBRR Settings for Commonly Used Oscillator Frequencies (Continued)

Baud Rate (bps)	$f_{osc} = 16.0000$ MHz				$f_{osc} = 18.4320$ MHz				$f_{osc} = 20.0000$ MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	416	-0.1%	832	0.0%	479	0.0%	959	0.0%	520	0.0%	1041	0.0%
4800	207	0.2%	416	-0.1%	239	0.0%	479	0.0%	259	0.2%	520	0.0%
9600	103	0.2%	207	0.2%	119	0.0%	239	0.0%	129	0.2%	259	0.2%
14.4k	68	0.6%	138	-0.1%	79	0.0%	159	0.0%	86	-0.2%	173	-0.2%
19.2k	51	0.2%	103	0.2%	59	0.0%	119	0.0%	64	0.2%	129	0.2%
28.8k	34	-0.8%	68	0.6%	39	0.0%	79	0.0%	42	0.9%	86	-0.2%
38.4k	25	0.2%	51	0.2%	29	0.0%	59	0.0%	32	-1.4%	64	0.2%
57.6k	16	2.1%	34	-0.8%	19	0.0%	39	0.0%	21	-1.4%	42	0.9%
76.8k	12	0.2%	25	0.2%	14	0.0%	29	0.0%	15	1.7%	32	-1.4%
115.2k	8	-3.5%	16	2.1%	9	0.0%	19	0.0%	10	-1.4%	21	-1.4%
230.4k	3	8.5%	8	-3.5%	4	0.0%	9	0.0%	4	8.5%	10	-1.4%
250k	3	0.0%	7	0.0%	4	-7.8%	8	2.4%	4	0.0%	9	0.0%
0.5M	1	0.0%	3	0.0%	—	—	4	-7.8%	—	—	4	0.0%
1M	0	0.0%	1	0.0%	—	—	—	—	—	—	—	—
Max ⁽¹⁾	1 Mbps		2 Mbps		1.152 Mbps		2.304 Mbps		1.25 Mbps		2.5 Mbps	

RELOJ INTERNO DEL USART

La comunicación serial del AVR emplea un reloj interno, que es posible configurar de acuerdo a las necesidades de la transmisión, la tasa de transferencia que se desea emplear y la frecuencia del oscilador del microprocesador.

Un contador decreciente es cargado con el contenido de UBRR cada vez que llega al valor de 0, esto produce un cambio en el estado del reloj interno del USART.

En las tablas anteriores se dieron algunos ejemplos de las tasas de transferencia más comunes, sin embargo, en caso de que la tasa de transferencia no se encuentre en esas tablas, ésta puede ser calculada mediante las siguientes fórmulas:

Table 61. Equations for Calculating Baud Rate Register Setting

Operating Mode	Equation for Calculating Baud Rate ⁽¹⁾	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR + 1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Note: 1. The baud rate is defined to be the transfer rate in bit per second (bps).

BAUD Baud rate (in bits per second, bps)

f_{osc} System Oscillator clock frequency

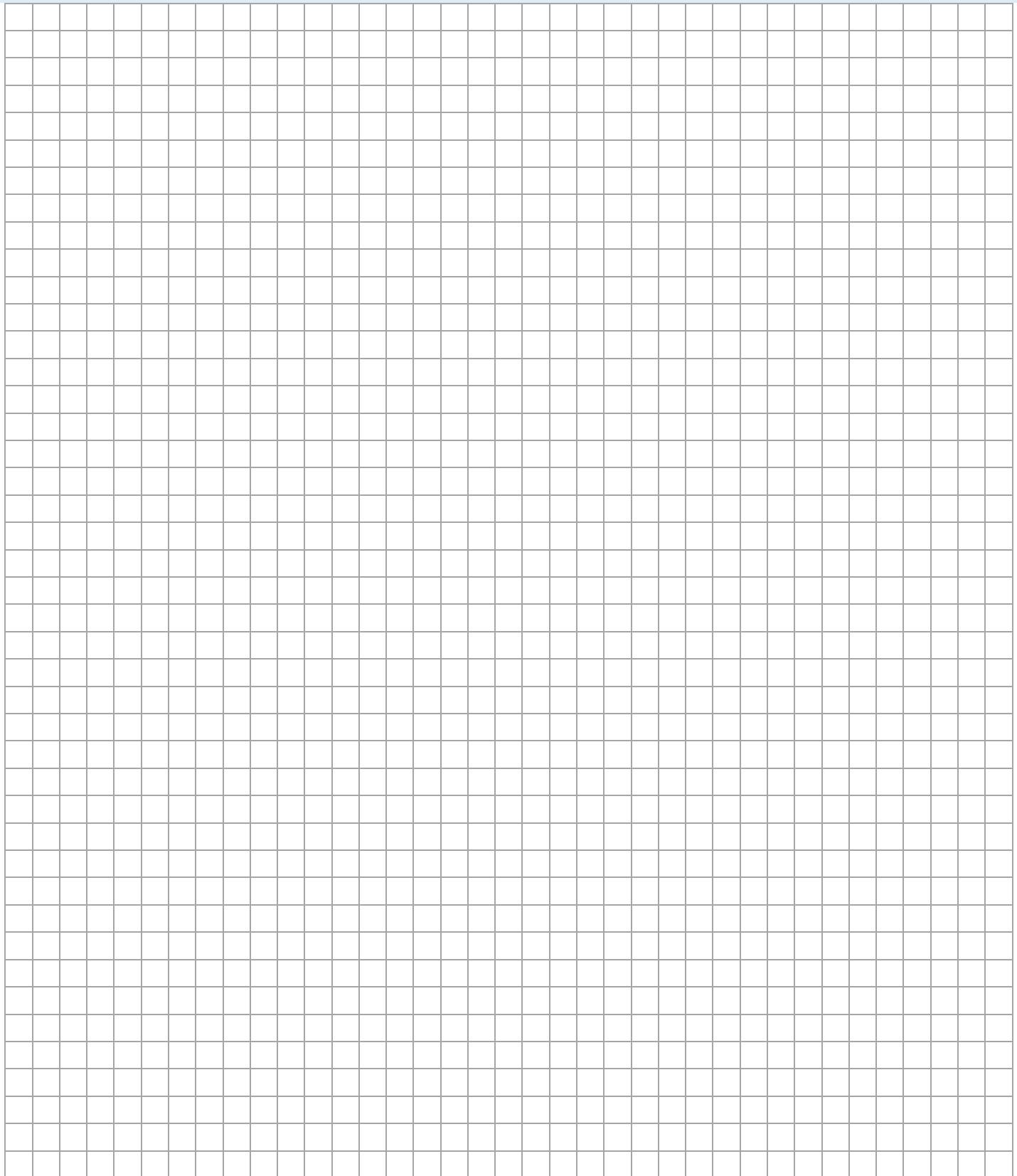
UBRR Contents of the UBRRH and UBRRRL Registers, (0 - 4095)

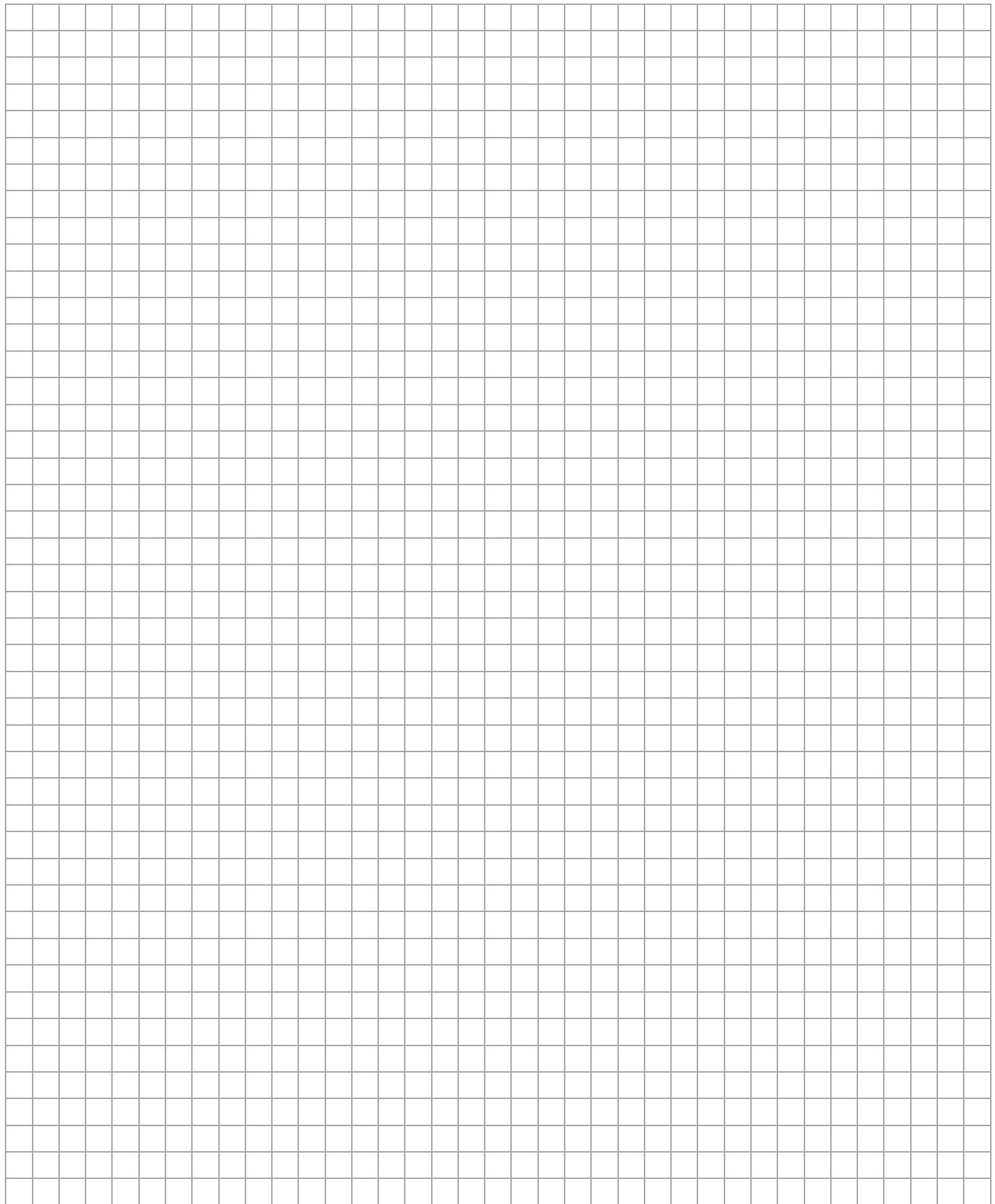
INICIALIZACIÓN DEL USART

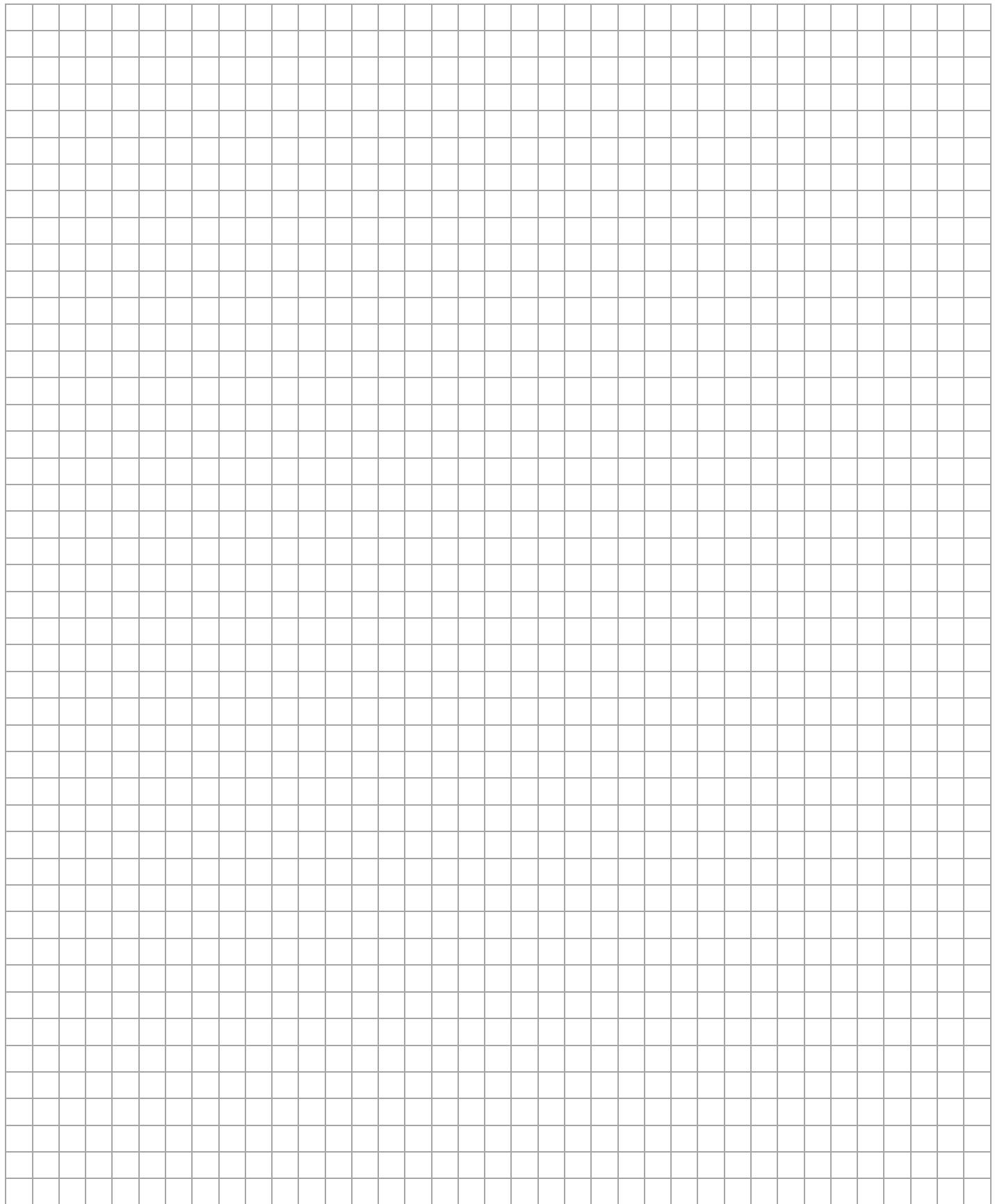
El USART debe ser inicializado antes de realizar cualquier tipo de comunicación, este proceso de inicialización consiste en especificar los Bauds a los que se trabajará, establecer el modo de operación y habilitar la transmisión o la recepción, dependiendo del uso que se les vaya a dar. Si se empleará el USART controlado por interrupciones, es importante que la bandera de interrupciones globales (la que se habilita con la instrucción sei) se encuentre deshabilitada al momento de hacer la inicialización. (Es decir que la instrucción sei deberá ser escrita hasta después de haber inicializado el USART).

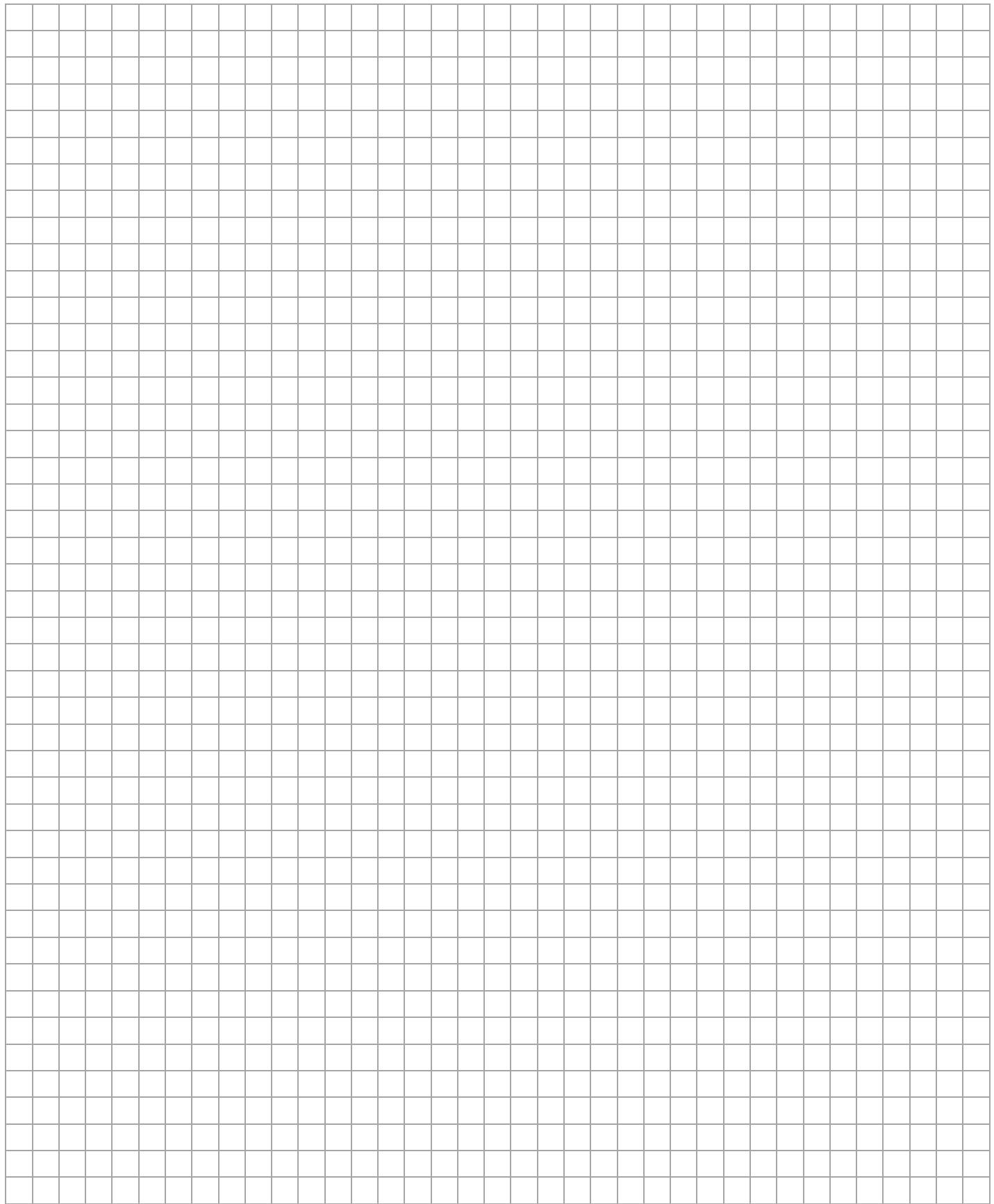
Si durante alguna parte del programa se desea reinicializar el USART para cambiar los bauds a los que se está trabajando o el modo de funcionamiento, se debe de tener cuidado de no estar realizando una transmisión o recepción de información en ese momento, para ello se pueden revisar las banderas TXC y RXC.

NOTAS PERSONALES – COMUNICACIÓN SERIAL

A large grid of squares, approximately 20 columns by 25 rows, designed for taking notes or drawing.







TRANSMISIÓN DE DATOS POR EL PUERTO SERIAL – VOLTÍMETRO [PUERTO SERIAL]

Se desea diseñar un voltímetro con la capacidad de medir voltajes de 0 a 5V a través de un pin del microcontrolador. Con el objetivo de realizar esta prueba, el voltaje será obtenido de un potenciómetro conectado entre tierra y 5V. El resultado del voltaje medido deberá ser desplegado en la pantalla de la computadora (se sugiere enviar a la computadora el valor resultante del ADC y utilizar la computadora para hacer los cálculos necesarios).

EL PROGRAMA EN EL MICRO...

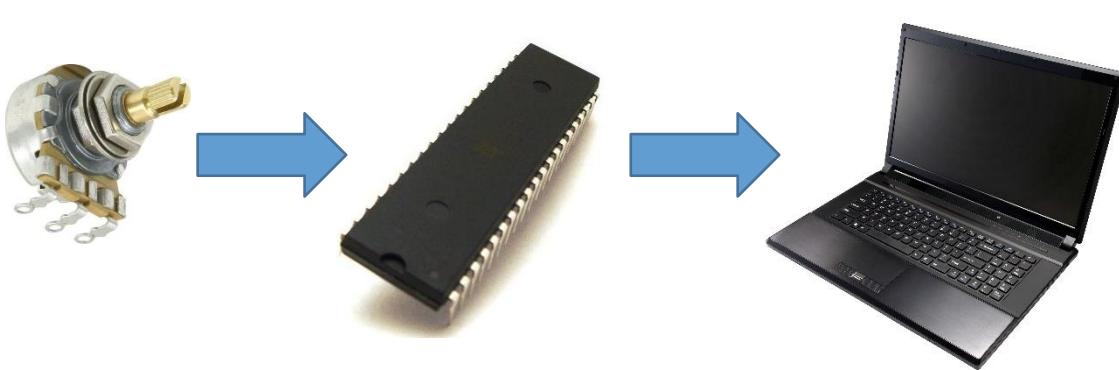
El microcontrolador deberá estar configurado para realizar conversiones con el ADC interno en forma constante. El dato leído (8 bits) deberá ser enviado constantemente a través del puerto serial.

EL PROGRAMA DE LA COMPUTADORA...

El programa desarrollado en la computadora deberá recibir cada dato y hacer las operaciones necesarias para mostrar en la pantalla el voltaje que se está introduciendo en cada momento.

SUGERENCIAS DE PRUEBA...

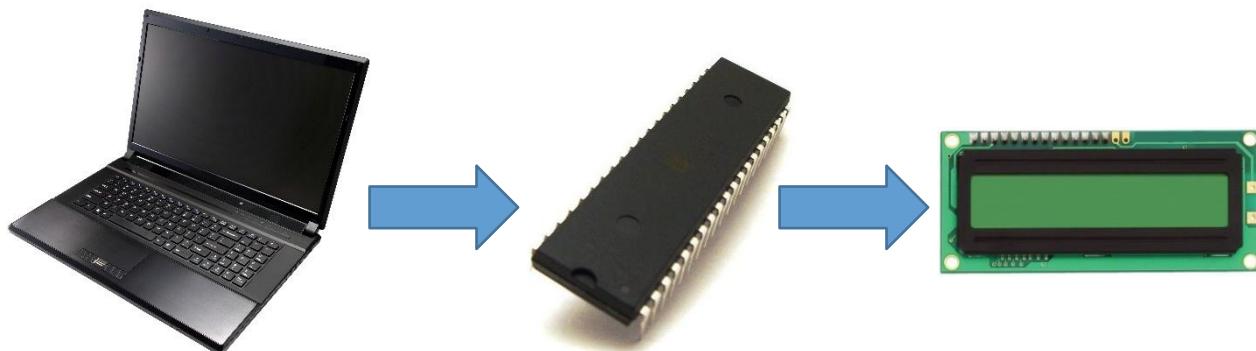
Se sugiera probar primero el programa del microcontrolador conectándolo con la computadora y utilizando un software de comunicación serial para verificar que los datos estén siendo enviados en forma correcta, una vez que esté seguro de que el programa del microcontrolador funciona correctamente entonces comience con el programa de Visual Basic (o algún lenguaje que usted desee emplear) para recibir el dato y hacer los cálculos correspondientes.



RECEPCIÓN DE DATOS POR EL PUERTO SERIAL – VOLTÍMETRO [PUERTO SERIAL]

Se desea diseñar un sistema que permita capturar datos a través del teclado de la computadora, los datos mínimos que se pueden capturar deberán ser todas las letras (mayúsculas y minúsculas) así como todos los números (Pero si usted lo desea también podrá capturar cualquier símbolo). Los datos que se van capturando deberán ser mostrados en la pantalla de la computadora (a través del programa terminal, o mejor aún en un programa que usted desarrolle usando Visual C#) y a su vez deberán ser enviados a través del puerto serial hacia un microprocesador, el cual recibirá estos datos y los mostrará también en un LCD.

En este programa se calificará también el código diseñado, considerando que usted no deberá elaborar comparaciones en el microcontrolador para imprimir cada letra deseada.



JUGUETE PARA NIÑOS [PUERTO SERIAL]

Se conectarán al microprocesador ATmega16A ocho botones, una bocina y el puerto serial.

Se desea diseñar un juguete didáctico para niños pequeños, los cuales tendrán conectado a la computadora un panel especial con 8 botones de colores, y una bocina. Únicamente será necesario tener ejecutándose el programa de Visual C# y todo el juego será controlado desde el panel del niño.

Cuando el niño desee iniciar su juego, deberá presionar el botón BLANCO. Posteriormente la pantalla de la computadora le irá indicando cuál tecla es la que debe presionar (llevando la cuenta si comete algún error). Cada vez que el niño presione la tecla CORRECTA, ésta emitirá un sonido, correspondiendo ROJO → DO, NARANJA → RE, AMARILLO → MI, VERDE → FA, AZUL → SOL, MORADO → LA, ROSA → SI. (Si la tecla que el niño presionó NO ES LA CORRECTA entonces ésta no deberá emitir ningún sonido). Usted puede elegir la octava que le resulte más conveniente (para su facilidad aquí se muestra una tabla con la información de algunas octavas).

Una vez que la computadora termine de presentar al niño la melodía que se está desarrollando, le indicará cuántos errores cometió y el tiempo que le tomó repetir correctamente la secuencia.

Al terminar la melodía, la computadora estará lista para esperar a que se presione nuevamente el botón BLANCO y empezar así nuevamente.

OCTAVA -2			OCTAVA -1			OCTAVA 0		
NOTA	Nº	Hz	NOTA	Nº	Hz	NOTA	Nº	Hz
DO	0	8.18	DO	12	16.35	DO	24	32.70
DO#	1	8.66	DO#	13	17.32	DO#	25	34.65
RE	2	9.18	RE	14	18.35	RE	26	36.71
RE#	3	9.72	RE#	15	19.45	RE#	27	38.89
MI	4	10.30	MI	16	20.60	MI	28	41.20
FA	5	10.91	FA	17	21.83	FA	29	43.65
FA#	6	11.56	FA#	18	23.12	FA#	30	46.25
SOL	7	12.25	SOL	19	24.50	SOL	31	49.00
SOL#	8	12.98	SOL#	20	25.96	SOL#	32	51.91
LA	9	13.75	LA	21	27.50	LA	33	55.00
LA#	10	14.57	LA#	22	29.14	LA#	34	58.27
SI	11	15.43	SI	23	30.87	SI	35	61.74

OCTAVA 1			OCTAVA 2			OCTAVA 3		
NOTA	Nº	Hz	NOTA	Nº	Hz	NOTA	Nº	Hz
DO	36	65.41	DO	48	130.81	DO	60	261.63
DO#	37	69.30	DO#	49	138.59	DO#	61	277.18
RE	38	73.42	RE	50	146.83	RE	62	293.66
RE#	39	77.78	RE#	51	155.56	RE#	63	311.13
MI	40	82.41	MI	52	164.81	MI	64	329.63
FA	41	87.31	FA	53	174.61	FA	65	349.23
FA#	42	92.50	FA#	54	185.00	FA#	66	369.99
SOL	43	98.00	SOL	55	196.00	SOL	67	392.00
SOL#	44	103.83	SOL#	56	207.65	SOL#	68	415.30
LA	45	110.00	LA	57	220.00	LA	69	440.00
LA#	46	116.54	LA#	58	233.08	LA#	70	466.16
SI	47	123.47	SI	59	246.94	SI	71	493.88

Protocolo de comunicación

Al iniciar el programa en la computadora, ésta se encontrará esperando a recibir el ascii de la letra 'I' (i mayúscula), el cual provocará que automáticamente la computadora envíe por el puerto serial el código ascii correspondiente al número de nota que el niño deberá presionar (por ejemplo, si encendió el botón ROJO, enviará por el puerto un '1' correspondiente a la tecla DO). En ese momento la computadora se encontrará en espera de recibir el código ascii de la tecla que el niño presione (SIN IMPORTAR SI ES LA CORRECTA O NO, ES DECIR CUALQUIER TECLA PRESIONADA DEBERÁ SER ENVIADA POR EL PUERTO SERIAL HACIA LA COMPUTADORA). Cuando el niño presione la tecla CORRECTA entonces la computadora mostrará la nueva tecla que el niño debe presionar y volverá a enviar por el puerto serial el código ascii correspondiente al número de nota que ahora corresponde presionar. Cuando el niño presione una tecla INCORRECTA, la computadora incrementará el número de errores que ha cometido el niño, pero no enviará ninguna información... sólo se quedará esperando a recibir nuevamente otra tecla presionada por el niño.

Para generar la frecuencia de cada nota se empleará el TIMER 0 del microcontrolador. Por favor llene la información que se le pide a continuación:

FRECUENCIA DE OPERACIÓN ELEGIDA PARA EL ATMEGA16A: _____

OCTAVA ELEGIDA DE ACUERDO AL CUADRO QUE SE MUESTRA: _____

NOTA	FRECUENCIA (Hz)	PERIODO (S)	PRESCALER DEL TIMER	VALOR PARA OCRO
DO				
RE				
MI				
FA				
SOL				
LA				
SI				

VELOCIDAD ELEGIDA PARA EL PUERTO SERIAL: _____

PARIDAD DEL PUERTO SERIAL: _____

NÚMERO DE BITS DE PARADA DEL PUERTO SERIAL: _____

SECUENCIA DE LEDS CONTROLADA POR PUERTO SERIAL Y ADC [PUERTO SERIAL]

Se desea generar un programa a 8Mhz que recibirá datos de 8 bits en forma serial asíncrona a una velocidad de 4800bps, con paridad par, con 1 bit de parada. Se conectarán 8 LEDs en el puerto C del AVR ATmega16A, al momento de encender el dispositivo los LEDs deberán aparecer todos apagados. Cuando el dato recibido sea un 49 (que corresponde al código ASCII del '1') entonces el o los LEDs deberán comenzar a encender, de derecha a izquierda, permaneciendo cada uno encendido durante un tiempo visible antes de realizar el corrimiento.

Por el pin ADC7 estará entrando un voltaje analógico que el microcontrolador deberá revisar al momento de recibir un dato por el puerto serial, dicho voltaje analógico se encontrará entre 0V y 5V. Si el voltaje en ese momento es menor a 2.5V entonces los LEDs deberán encender de uno en uno, por otra parte, si el voltaje es mayor o igual a 2.5V entonces los LEDs encenderán de dos en dos.

TEMPERATURA/HUMEDAD/PH [PUERTO SERIAL]

Deberá diseñar un circuito que tendrá la capacidad de monitorear la humedad, la temperatura y el ph de la tierra a través de tres sensores diferentes conectados al microcontrolador ATmega16A.

El sensor de humedad estará conectado en el ADC0, el sensor de temperatura en el ADC1 y el sensor de ph en el ADC2 del microcontrolador. También se tendrán conectados 4 Leds para mostrar una escala relacionada con la temperatura, otros 4 Leds para mostrar una escala relacionada con la humedad y por último otros 4 Leds para una escala relacionada con el PH, estas escalas deberán estarse actualizando en forma constante. **(Deberá respetar las conexiones que se le muestran en el diagrama de proteus, no está permitido realizar cambios en las mismas).** En el proteus que se le proporciona para la prueba del circuito los sensores se simularán a través de 3 potenciómetros y las escalas en LEDs deberán mostrarse según el valor recibido en el ADC, para ello se le pide que llene la siguiente tabla.

Voltaje	LEDs encendidos	Valor leído en el ADC
0 V ≤ Voltaje < 1V	0	
1V ≤ Voltaje < 2V	1	
2V ≤ Voltaje < 3V	2	
3V ≤ Voltaje < 4V	3	
4V ≤ Voltaje < 5V	4	

Por otra parte, el circuito se encontrará conectado a través del puerto serial a una computadora en la cual correrá el programa que se encarga en ir monitoreando cada determinado tiempo los valores para guardarlos en una base de datos (de forma que posteriormente puedan ser utilizados para análisis estadísticos). El tiempo estará determinado de acuerdo en el programa de la computadora, la cual enviará al microcontrolador el código ASCII correspondiente a uno de tres caracteres diferentes para solicitar al circuito que le devuelva **el último valor leído** en ese sensor por el ADC.

Elija los valores que el programa de la computadora enviará para solicitar cada uno de los datos

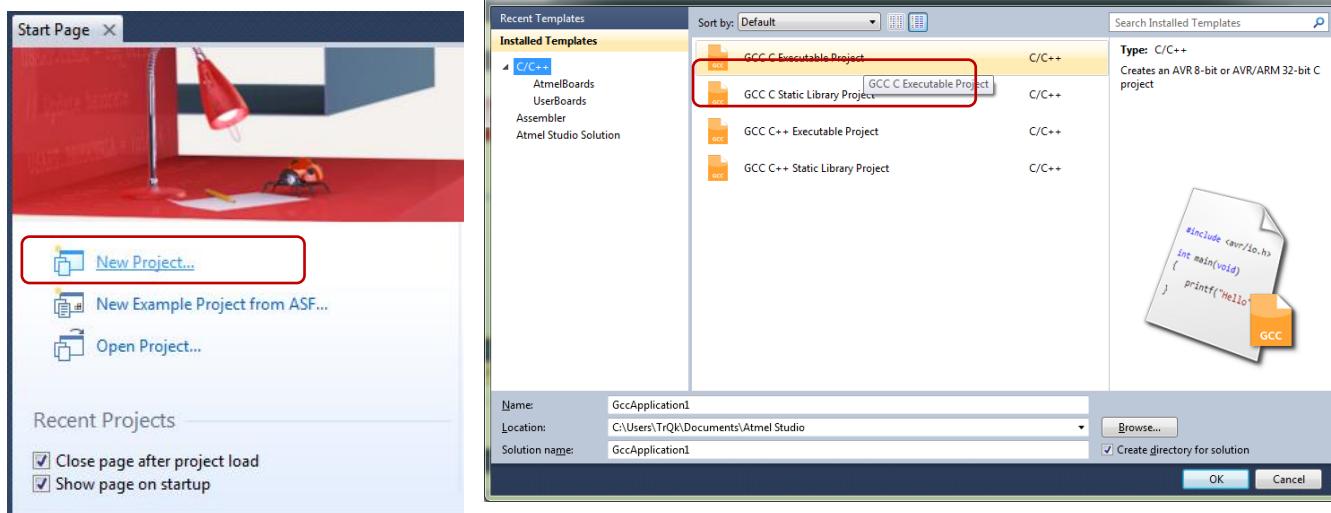
Sensor	Valor enviado (carácter)
Temperatura	
Humedad	
PH	

Para probar su circuito sin necesidad de realizar un programa en Visual C# usted puede utilizar el programa “Terminal”, a través del cual enviará al microcontrolador uno de los caracteres elegidos para preguntar por un sensor, entonces el micro deberá responder regresando el valor correspondiente a la última lectura del sensor (entre 0 y 255), posteriormente podrá enviar desde la computadora otro carácter para preguntar otro dato y así sucesivamente.

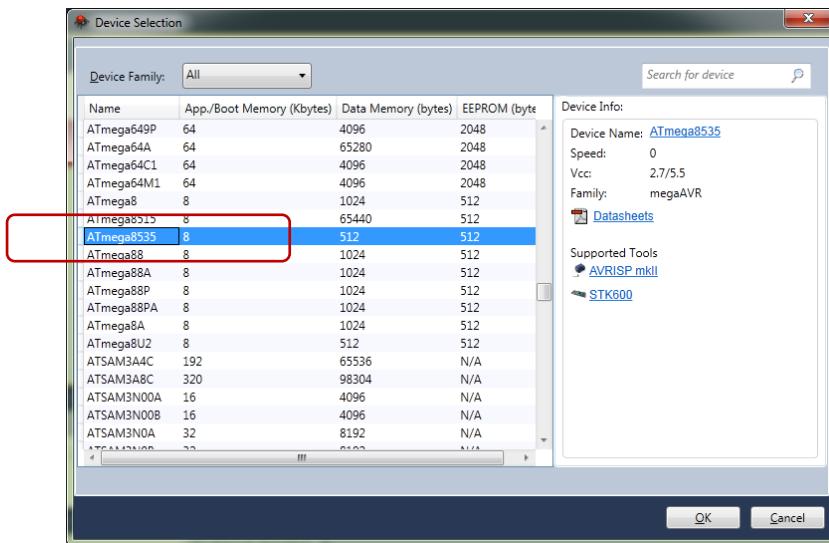
Este programa deberá estar realizado para funcionar en un microcontrolador a una frecuencia de 4Mhz, por otra parte, se le pide que la velocidad de comunicación con el puerto serial se encuentre fija a 9600bps.

PROGRAMACIÓN EN LENGUAJE C

Para crear un nuevo proyecto de programación del AVR ATmega8535 utilizando ATMEL STUDIO 6, abra el software y cree un nuevo proyecto y en seguida elija GCC C Executable Project.



Especifique el nombre del proyecto y la carpeta en donde desea guardarla; se sugiere elegir la opción de crear un folder para el proyecto, pues éste estará formado por varios archivos y así le será más fácil mantener organizada su información. Entonces presione el botón OK



En la siguiente ventana elija el procesador ATmega16A y presione el botón OK.

Entonces se abrirá la ventana de programación para el nuevo proyecto, donde se mostrará el siguiente código para iniciar el programa:

```
/*
 * GccApplication1.c
 *
 * Created: 03/04/2000 00:00:00 p.m.
 * Author: María Teresa Orvañanos Guerrero
 */

#include <avr/io.h>

int main(void)
{
    while(1)
    {
        //TODO:: Please write your application code
    }
}
```

Por default el programa agrega la primera línea `#include <avr/io.h>` esto es importante pues en este include se especifican todas las características concretas del micro que se eligió programar, las direcciones de los puertos, etc.

También tiene un punto en donde comienza a correr el programa, éste es el main, dentro de este bloque de código antes de llegar al bloque `while(1)`, se encontrarán definiciones de puerto y configuraciones que sólo sea necesario ejecutar una vez; casi siempre los programas del microprocesador llegan a un punto en el cual se da un ciclo infinito, este ciclo es todo aquello que se programe dentro del `while(1) { }`, es importante mencionar que este código se estará repitiendo infinitamente puesto que la sentencia `while(1)` corresponde a algo que siempre será verdadero.

Puertos

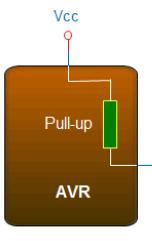
Al igual que en ensamblador, para definirnos a los puertos del micro controlador podemos hacerlo a través de

- Port x Data Register – PORTx
- Port x Data Direction Register – DDRx
- Port x Input Pins Arddess – PINx

El registro DDRx se utiliza para definir un puerto como entrada o salida, así a los pines a los que se envíe 0 quedarán definidos como entradas, mientras que aquellos a los que se envíe 1 quedarán definidos como salidas.

`DDRx = 0b11110000; // De esta forma queda definido como entrada en los bits bajos y salida en los bits altos`

`DDRx = 0b11111111; // De esta forma queda definido totalmente como salida`



Cuando un pin se configura como entrada, el registro PORTx se emplea para configurar la resistencia de pull up (poniéndole un 1 al bit correspondiente). Cuando un pin se configura como salida, el registro PORTx se emplea para sacar los datos que se le pongan a esos bits.

Si el puerto está configurado como entrada entonces

```
PORTx = 0x11111111; // activará las resistencias de pull up internas.
```

Si el puerto está configurado como salida entonces

```
PORTx = 0x11111111; // hará que salgan 5V por todos los pines del puerto.
```

Por último, PINx en la forma de referirse al puerto como entrada (es decir que es el registro que se emplea para leer un puerto), por lo tanto si se desea leer lo que hay en un momento determinado en un puerto y luego guardarlo en una variable, eso se puede realizar mediante las instrucciones:

```
unsigned int entrada; // de esta forma se define la variable entrada como integer sin signo
```

```
entrada = PINx; // guarda en la variable entra lo que hay en ese momento en el puerto x
```

Delays

En C, existe una librería que permite incluir fácilmente retardos, para ello hay que incluir la línea

```
#include <util/delay.h> //Libreria de los delays
```

Después de la que por default agregó el programa.

Una vez incluida ésta librería, puede usarse en forma sencilla a través de:

```
_delay_ms(1000); // Espera 1000ms
```

Conviene hacer notar que la función _delay_ms de la librería delay.h, tiene un límite de retardo que está relacionado con la velocidad a la que está trabajando el micro controlador. De esta forma,

$$t_{\text{max en ms}} = \frac{262.14}{F_{\text{CPU (En MHz)}}}$$

Variables y Tipos de Datos

Tabla de variables y tipos de datos del lenguaje C

Tipo de dato	Tamaño en bits	Rango de valores
char	8	0 a 255 ó -128 a 127
signed char	8	-128 a 127
unsigned char	8	0 a 255
(signed) int	16	-32,768 a 32,767
unsigned int	16	0 a 65,536
(signed) short	16	-32,768 a 32,767
unsigned short	16	0 a 65,536
(signed) long	32	-2,147,483,648 a 2,147,483,647
unsigned long	32	0 a 4,294,967,295
(signed) long long (int)	64	- 2^{63} a $2^{63} - 1$
unsigned long long (int)	64	0 a $2^{64} - 1$
float	32	$\pm 1.18E-38$ a $\pm 3.39E+38$
double	32	$\pm 1.18E-38$ a $\pm 3.39E+38$
double	64	$\pm 2.23E-308$ a $\pm 1.79E+308$

Por default el tipo double es de 32 bits en los microcontroladores sin embargo algunos compiladores como el que se emplea en este curso ofrecen la posibilidad de configurarlo para que sea de 64 bits y poder trabajar con datos más grandes y de mayor precisión.

Los especificadores signed (con signo) mostrados entre paréntesis son opcionales. Es decir, da lo mismo poner int que signed int, por ejemplo. Es una redundancia que se suele usar para “reforzar” su condición o para que se vea más ilustrativo.

El tipo char está pensado para almacenar un solo carácter ASCII como las letras. Puesto que estos datos son a fin de cuentas números también, es común usar este tipo para almacenar números de 8 bits.

En el entorno de los microcontroladores AVR, los tipos de datos int y short tienen el mismo tamaño y aceptan el mismo rango de valores, es decir, al compilador le da lo mismo si ponemos int o short. (En el lenguaje C para PC, el tipo short fue y siempre debería ser de 16 bits, en tanto que int fue concebido para adaptarse al bus de datos del procesador. Esto

todavía se cumple en la programación de las computadoras, por ejemplo, un dato int es de 32 bits en un Pentium IV y es de 64 bits en un procesador Core i7).

Debido a estos tipos de variables, pueden aparecer ciertas imprecisiones en los tipos de datos que pueden perturbar la portabilidad de los programas entre los diferentes compiladores. Es por esto que el lenguaje C/C++ provee la librería stdint.h para definir tipos enteros que serán de un tamaño específico independientemente de los procesadores y de la plataforma software en que se trabaje. A continuación se presentan esos tipos de variables.

Tabla de variables y tipos de datos del lenguaje C

Tipo de dato	Tamaño en bits	Rango de valores
int8_t	8	-128 a 127
uint8_t	8	0 a 255
int16_t	16	-32,768 a 32,767
uint16_t	16	0 a 65,536
int32_t	32	-2,147,483,648 a 2,147,483,647
uint32_t	32	0 a 4,294,967,295
int64_t	64	-2^{63} a $2^{63} - 1$
uint64_t	64	0 a $2^{64} - 1$

Para poder utilizar este tipo de variables, deberá incluirse el archivo stdint.h

```
#include <stdint.h>
```

DECLARACIÓN DE VARIABLES

Lo que se explicará a continuación, es similar a lo que se hace cuando se identifican las variables del ensamblador con la directiva .def. No se puede usar una variable si antes no se ha declarado. La forma general más simple de hacerlo es la siguiente:

Tipo_de_dato Variable:

donde Tipo_de_dato es un tipo de dato y Variable es un identificador cualquiera que se le quiere dar para referirnos a él, siempre que no sea palabra reservada.

```
unsigned char d; // Variable para enteros de 8 bits sin signo
```

```
char b;          // Variable de 8 bits (para almacenar caracteres ascii)
signed char c;  // Variable para enteros de 8 bits con signo
int i;           // i es una variable int, con signo
signed int j;   // j también es una variable int con signo
unsigned int k; // k es una variable int sin signo
```

También es posible declarar varias variables del mismo tipo, separándolas con comas.

```
float area, side; // Declarar variables area y side de tipo float
unsigned char a, b, c; // Declarar variables a, b y c como unsigned char
```

ESPECIFICADORES DE TIPO DE DATOS

A la declaración de una variable se le puede añadir un especificador de tipo como const, static, volatile, extern, register, etc. En esta sección únicamente nos concentraremos en el tipo const (constante).

Una variable const debe ser inicializada en su declaración. Después de eso el compilador solo permitirá su lectura, pero no será posible cambiarle el valor.

```
const int a = 100; // Declarar constante a
```

Sin embargo, al declarar una constante de esta forma, se ocuparán posiciones en la RAM del microcontrolador, por lo cual es mucho mejor el definir las constantes del programa con las clásicas directivas #define (como se hace en el ensamblador) ya que de esa forma no se ocupa RAM.

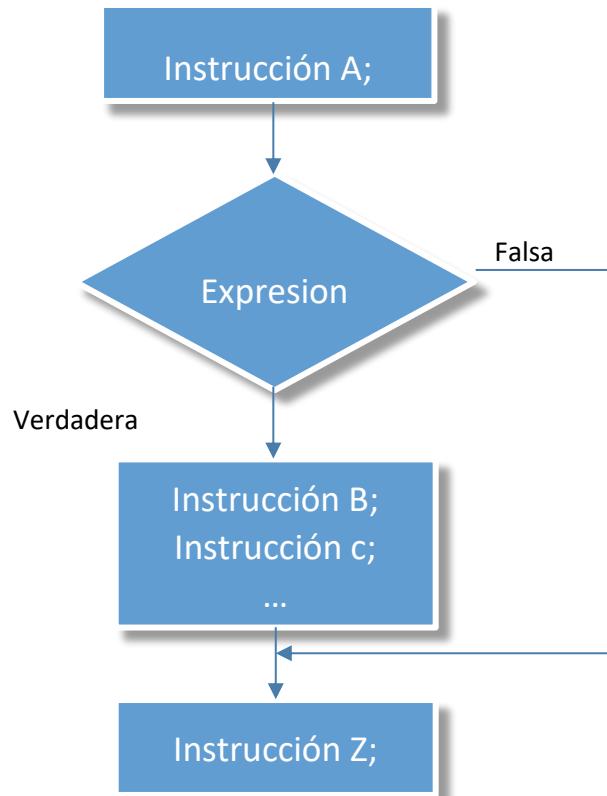
```
#define a 100 // Definir constante a
```

Sentencias de bifurcación

También son llamadas sentencias selectivas y sirven para redirigir el flujo de un programa según la evaluación de alguna condición lógica es decir permiten ejecutar una de entre varias acciones en función del valor dicha expresión.

IF

La sentencia if (si condicional) hace que un programa ejecute una sentencia o un grupo de ellas si una expresión es cierta.

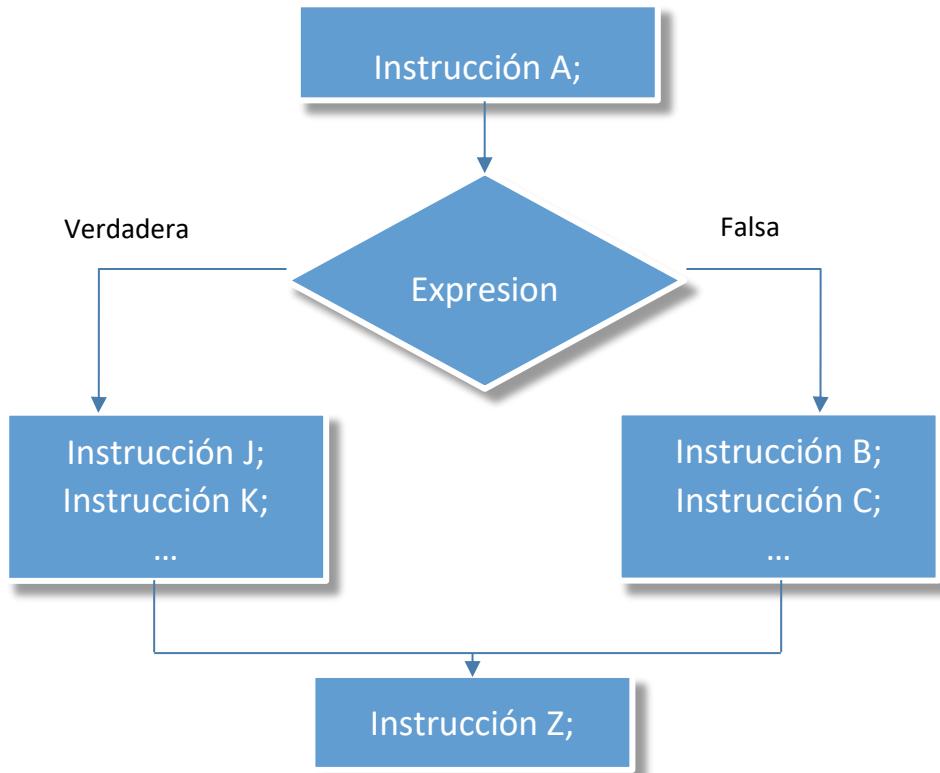


Lo cual en lenguaje C se escribe como:

```
InstruccionA;  
if ( expresion ) // Si expresión es verdadera, ejecutar el siguiente bloque  
{  
    // apertura de bloque  
    InstruccionB;  
    InstruccionC;  
    // algunas otras sentencias  
}  
    // cierre de bloque  
InstruccionZ;
```

IF – ELSE

La sentencia if brinda una rama que se ejecuta cuando una condición lógica es verdadera. Cuando el programa requiera dos ramas, una que se ejecute si cierta expresión es cierta y otra si es falsa, entonces se debe utilizar la sentencia if – else.



Lo cual en lenguaje C se escribe como:

```
InstrucciónA
if ( expresion )          // Si expression es verdadera, ejecutar
{                          // este bloque
    InstrucciónB;
    InstrucciónC;
    // ...
}
else                      // En caso contrario, ejecutar este bloque
{
    InstrucciónJ;
    InstrucciónK;
    // ...
}
InstrucciónZ;
// ...
```

IF – ELSE – IF ESCALONADA

Es la versión ampliada de la sentencia if – else.

En el siguiente código se comprueban tres condiciones lógicas, aunque podría haber más. Del mismo modo, se han puesto dos instrucciones por bloque solo para simplificar el esquema.

```
if ( expresion1 )      // Si expresion1 es verdadera ejecutar este bloque
{
    Instruccion1;
    Instruccion2;
}
else if ( expresion2 ) // En caso contrario y si expresion2 es verdadera, ejecutar este bloque
{
    Instruccion3;
    Instruccion4;
}
else if ( expresion3 ) // En caso contrario y si expresion3 es verdadera, ejecutar este bloque
{
    Instruccion5;
    Instruccion6;
}
else                  // En caso contrario, ejecutar este bloque
{
    Instruccion7;
    Instruccion8;
};
// ...
```

Las “expresiones” se evalúan de arriba a abajo. Cuando alguna de ellas sea verdadera, se ejecutará su bloque correspondiente y los demás bloques serán saltados. El bloque final (de else) se ejecuta si ninguna de las expresiones es verdadera. Además, si dicho bloque está vacío, puede ser omitido junto con su else.

SWITCH

La sentencia switch puede ser considerada como una forma más estructurada de la sentencia if – else – if escalonada. Para elaborar el código en C se usan las palabras reservadas switch, case, break y default.

El siguiente esquema presenta tres case's pero podría haber más, así como cada bloque también podría tener más instrucciones.

```
switch ( expresion )
{
    case constante1: // Si expresion = constante1, ejecutar este bloque
        instruccion1;
        instruccion2;
        break;
    case constante2: // Si expresion = constante2, ejecutar este bloque
        instruccion3;
        instruccion4;
        break;
}
```

```

        case constante3: // Si expresion = constante3, ejecutar este bloque
            instruccion5;
            instruccion6;
            break;
        default:          // Si expression no fue igual a ninguna de las constantes anteriores,
ejecutar este bloque
            instruccion7;
            instruccion8;
            break;
    }
instruccionX;
// ...

```

Donde constante1, constante2 y constante3 deben ser constantes enteras, por ejemplo, 2, 0x45, 'a', etc. ('a' tiene su correspondiente código ASCII 165, que es, a fin de cuentas, un entero.)

El programa solo ejecutará uno de los bloques dependiendo de qué constante coincida con la expresión. Usualmente los bloques van limitados por llaves, pero en este caso son opcionales, dado que se pueden distinguir fácilmente. Los bloques incluyen la sentencia break que hace que el programa salga del bloque de switch y ejecute la sentencia que sigue (en el código: sentenciaX). ¡Importante!: de no poner break, también se ejecutará el bloque del siguiente case, sin importar si su constante coincide con la expresión o no.

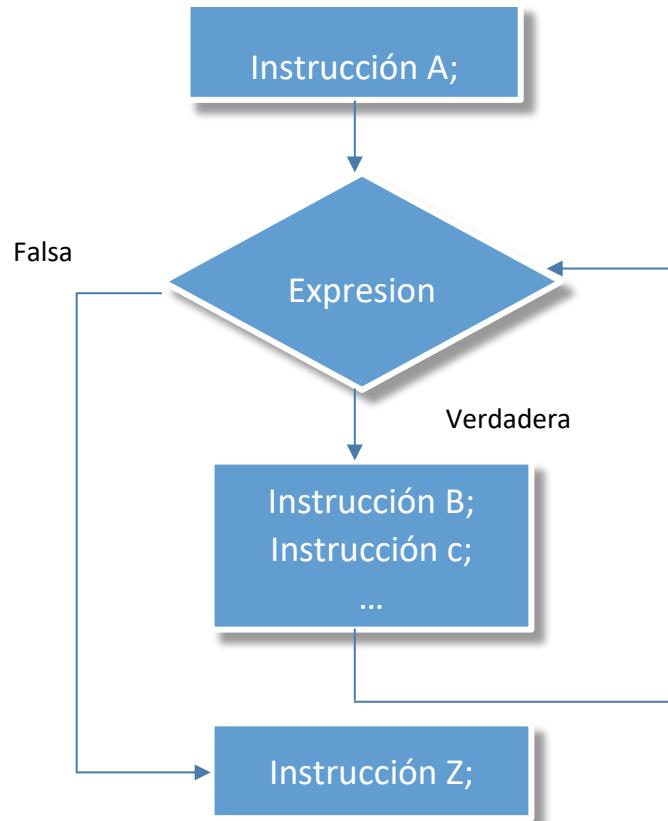
No es necesario poner el default si este bloque está vacío.

Sentencias iterativas

Las sentencias iterativas sirven para que el programa execute una sentencia o un grupo de ellas un número determinado o indeterminado de veces.

WHILE

El cuerpo o bloque de este bucle se ejecutará una y otra vez mientras (while) la expresión sea verdadera.



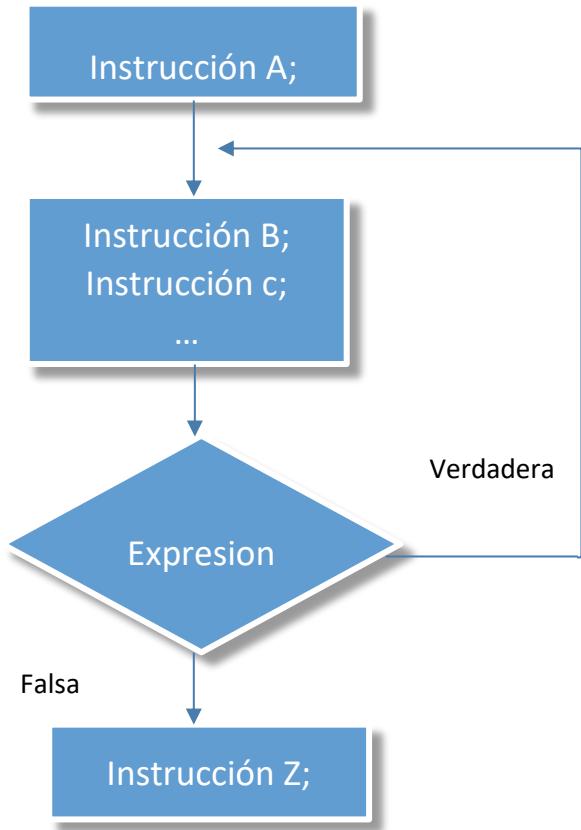
El bucle while en C se lee así: mientras (while) expresión sea verdadera, ejecutar el siguiente bloque.

```
instruccionA;  
while ( expresion ) // Mientras expression sea verdadera, ejecutar el siguiente bloque  
{  
    instruccionB;  
    instruccionC;  
    // ...  
}  
instruccionZ;  
// ...
```

Se debe notar que en este caso primero se evalúa la expresión. Por lo tanto, si desde el principio la expresión es falsa, el bloque de while no se ejecutará nunca. Por otro lado, si la expresión no deja de ser verdadera, el programa se quedará dando vueltas “para siempre”.

DO - WHILE

Es una variación de la sentencia while simple. La principal diferencia es que la condición lógica (expresión) de este bucle se presenta al final. Como se ve en la siguiente figura, esto implica que el cuerpo o bloque de este bucle se ejecutará al menos una vez.



La sentencia do – while se lee: Ejecutar (do) el siguiente bloque, mientras (while) expresión sea verdadera.

```
instrucionA;  
do  
{  
    instrucionB;  
    instrucionC;  
    // ...  
} while ( expresion );  
instrucionZ;  
// ...
```

FOR

Las dos sentencias while y do – while, se suelen emplear cuando no se sabe de antemano la cantidad de veces que se va a ejecutar el bucle. En los casos donde el bucle involucra alguna forma de conteo finito es preferible emplear la sentencia for.

Ésta es la sintaxis general de la sentencia for en C:

```
for ( expresion_1 ; expresion_2 ; expresion_3 )
{
    instruccion1;
    instruccion2;
    // ...
}
```

Funciona de la siguiente forma:

expresion_1 suele ser una sentencia de inicialización, desde donde se va a empezar a contar.

expresion_2 se evalúa como condición lógica para que se ejecute el bloque, en que momento va a parar el ciclo.

expresion_3 es una sentencia que controla a expresion_2, es decir cómo se va a ir incrementando la cuenta después de cada iteración.

Por ejemplo, si i es una variable

```
for ( i = 0 ; i < 10 ; i++ )
{
    instrucciones;
}
```

Se lee: para (for) i igual a 0 hasta que sea menor que 10 ejecutar sentencias. La sentencia i++ indica que i se incrementa en uno tras cada ciclo. Así, el bloque de for se ejecutará 10 veces, cuando i se incremente desde 0 hasta 9.

En este otro ejemplo las sentencias se ejecutan desde que i valga 10 hasta que valga 20. Es decir, el bucle dará 11 vueltas en total.

```
for ( i = 10 ; i <= 20 ; i++ )
{
    instrucciones;
}
```

El siguiente bucle for empieza con i inicializado a 100 y su bloque se ejecutará mientras i sea mayor o igual a 0. Por supuesto, en este caso i se decrementa tras cada ciclo.

```
for ( i = 100 ; i >= 0 ; i-- )
{
    instrucciones;
}
```

Operadores

Sirven para realizar operaciones aritméticas, lógicas, comparativas, etc. Según esa función se clasifican en los siguientes grupos.

OPERADORES ARITMÉTICOS

Además de los típicos operadores de suma, resta, multiplicación y división, están los operadores de módulo, incremento y decremento.

Operador	Acción
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo. Retorna el residuo de una división entera. Solo se debe usar con números enteros.
++	Incrementar en uno
--	Decrementar en uno

Ejemplos:

```
int a, b, c;           // Declarar variables a, b y c
c = a + b;             // Sumar a y b. Almacenar resultado en c
b = b * c;             // Multiplicar b por c. Resultado en b
b = a / c;             // Dividir a entre c. Colocar resultado en b
a = a + c - b;         // Sumar a y c y restarle b. Resultado en a
c = (a + b) / c;       // Dividir a+b entre c. Resultado en c
b = a + b / c + b * b; // Sumar a más b/c más b×b. Resultado en b
c = a % b;             // Residuo de dividir a÷b a c
a++;
b--;
++c;
--b;
```

Si ++ o -- están antes del operando, primero se suma o resta 1 al operando y luego se evalúa la expresión.

Si ++ o -- están después del operando, primero se evalúa la expresión y luego se suma o resta 1 al operando.

```
int a, b;           // Declarar variables enteras a y b
a = b++;           // Lo mismo que a = b; y luego b = b + 1;
a = ++b;           // Lo mismo que b = b + 1; y luego a = b;
if (a++ < 10)      // Primero comprueba si a < 10 y luego incrementa a en 1
{
    // algún código
}
if (++a < 10)      // Primero incrementa a en 1 y luego comprueba si a < 10
{
    // algún código
}
```

OPERADORES DE BITS

Se aplican a operaciones lógicas con variables a nivel binario. Si bien son operaciones que producen resultados análogos a los de las instrucciones de ensamblador los operadores lógicos del C pueden operar sobre variables de distintos tamaños, ya sean de 1, 8, 16 ó 32 bits.

Operador	Acción
&	AND a nivel de bits
	OR nivel de bits
^	XOR (OR exclusiva) a nivel de bits
~	Complemento a uno a nivel de bits
<<	Desplazamiento a la izquierda
>>	Desplazamiento a la derecha

```
char m;           // variable de 8 bits
int n;            // variable de 16 bits
m = 0x48;          // m será 0x48
m = m & 0x0F;      // Despues de esto m será 0x08
m = m | 0x24;      // Despues de esto m será 0x2F
m = m & 0b11110000; // Despues de esto m será 0x20
n = 0xFF00;        // n será 0xFF00
```

```

n = ~n;           // n será 0x00FF
m = m | 0b10000001; // Setear bits 0 y 7 de variable m
m = m & 0xF0;      // Limpiar nibble bajo de variable m
m = m ^ 0b00110000; // Invertir bits 4 y 5 de variable m
m = 0b00011000;    // Cargar m con 0b00011000
m = m >> 2;        // Desplazar m 2 posiciones a la derecha, ahora m será 0b00000110
n = 0xFF1F;        // Desplazar n 12 posiciones a la izquierda, ahora n será 0xF000;
n = n << 12;       // Desplazar m 8 posiciones a la izquierda, ahora m será 0x00
m = m << 8;        // Despues de esto m será 0x00

```

Cuando una variable se desplaza hacia un lado, los bits que salen por allí se pierden y los bits que entran por el otro lado son siempre ceros. Es por esto que en la última sentencia, $m = m << 8$, el resultado es 0x00. En el lenguaje C no existen operadores de rotación.

OPERADORES RELACIONALES

Se emplean para construir las condiciones lógicas de las sentencias de control. La siguiente tabla muestra los operadores relacionales disponibles.

Operador	Acción
==	Igual
!=	Diferente (no igual)
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

OPERADORES LÓGICOS

Generalmente se utilizan para enlazar dos o más condiciones lógicas simples.

Operador	Acción
&&	AND lógica
	OR lógica
!	Negación lógica

```

if( !(a==0) )           // Si a igual 0 sea falso
{
    // instrucciones
}

if( (a<b) && (a>c) ) // Si a<b y a>c son verdaderas
{
    // instrucciones
}

while( (a==0) || (b==0) ) // Mientras a sea 0 ó b sea 0
{
    // instrucciones
}

```

COMPOSICIÓN DE OPERADORES

Se utiliza en las operaciones de asignación y permite escribir código más abreviado.

Es importante resaltar que no debe haber ningún espacio entre el operador y el signo igual.

```

int a;                  // Declarar a
a += 50;                // Es lo mismo que a = a + 50;
a += 20;                // También significa sumarle 20 a a
a *= 2;                 // Es lo mismo que a = a * 2;
a &= 0xF0;               // Es lo mismo que a = a & 0xF0;
a <<= 1;                // Es lo mismo que a = a << 1;

```

PRECEDENCIA DE OPERADORES

Una expresión puede contener varios operadores, de esta forma:

```
b = a * b + c / b;      // a, b y c son variables
```

En esta sentencia no queda claro en qué orden se ejecutarán las operaciones indicadas. Hay ciertas reglas que establecen dichas prioridades; por ejemplo, las multiplicaciones y divisiones siempre se ejecutan antes que las sumas y restas. Pero es más práctico emplear los paréntesis, los cuales ordenan que primero se ejecuten las operaciones de los paréntesis más internos.

Por ejemplo, las tres siguientes sentencias son diferentes.

```

b = (a * b) + (c / b);
b = a * (b + (c / b));
b = ((a * b) + c) / b;

```

También se pueden construir expresiones condicionales, así:

```
if ( (a > b) && ( b < c) ) // Si a>b y b<c, ...
{
    // ...
}
```

Funciones

Una función es un bloque de instrucciones identificado por un nombre; puede recibir y devolver datos. En bajo nivel, en general, las funciones operan como las subrutinas de ensamblador, es decir, al ser llamadas, se guarda en la Pila el valor actual del PC (Program Counter), después se ejecuta todo el código de la función y finalmente se recobra el PC para regresar de la función.

A continuación se representa la forma general de una función:

```
tipo_de_dato1 Nombre_de_funcion(tipo_de_dato2 argumento1, tipo_de_dato3 argumento2, ... )
{
    // Cuerpo de la función
    // ...
    return Dato_de_Regreso; // Necesario solo si la función retorna algún valor
}
```

Donde:

El Nombre_de_funcion puede ser un identificador cualquiera.

tipo_de_dato1 es un tipo de dato que identifica el parámetro de salida; se utiliza únicamente cuando la función regresará un valor, en caso de que se desee programar una función que no regrese ningún valor se debe poner la palabra reservada void (vacío, en inglés).

argumento1 y argumento2 (puede haber más) son las variables de tipos tipo_de_dato2, tipo_de_dato3..., que recibirán los datos que se le pasen a la función. Si no hay ningún parámetro de entrada, se pueden dejar los paréntesis vacíos o escribir un void entre ellos.

FUNCIONES SIN PARÁMETROS

Para una función que no recibe ni devuelve ningún valor, el código anterior se reduce al siguiente esquema:

```
void Nombre_de_funcion( void )
{
    // Cuerpo de la función
}
```

Y se llama escribiendo su nombre seguido de paréntesis vacíos, así:

```
Nombre_de_funcion();
```

La función principal main es otro ejemplo de función sin parámetros. Dondequiera que se ubique, siempre debería ser la primera en ejecutarse; de hecho, no debería terminar.

```
void main (void)
{
    // Cuerpo de la función
}
```

FUNCIONES CON PARÁMETROS POR VALOR

El valor devuelto por una función se indica con la palabra reservada return.

Para llamar a una función con parámetros es importante respetar el orden y el tipo de los parámetros que ella recibe. El primer valor pasado corresponde al primer parámetro de entrada; el segundo valor, al segundo parámetro; y así sucesivamente si hubiera más.

Cuando una variable es entregada a una función, en realidad se le entrega una copia suya. De este modo, el valor de la variable original no será alterado.

```
int minor ( int arg1, int arg2, int arg3 )
{
    int min;           // Declarar variable min
    min = arg1;        // Asumir que el menor es arg1

    if ( arg2 < min ) // Si arg2 es menor que min
        min = arg2;    // Cambiar a arg2

    if ( arg3 < min ) // Si arg3 es menor que min
        min = arg3;    // Cambiar a arg3

    return min;        // Regresar valor de min
}
void main (void)
{
    int a, b, c, d;    // Declarar variables a, b, c y d

    /* Aquí asignamos algunos valores iniciales a 'a', 'b' y 'c' */
    /* ... */

    d = minor(a,b,c); // Llamar a minor
    // En este punto 'd' debería ser el menor entre 'a', 'b' y 'c'
    while (1);         // Bucle infinito
}
```

En el programa mostrado la función minor recibe tres parámetros de tipo int y devuelve uno, también de tipo int, que será el menor de los números recibidos.

Al llamar a minor el valor de a se copiará a la variable arg1; el valor de b, a arg2 y el valor de c, a arg3. Después de ejecutarse el código de la función el valor de retorno (min en este caso) será copiado a una variable temporal y de allí pasará a d.

FUNCIONES CON PARÁMETROS POR REFERENCIA

La función que recibe un parámetro por referencia puede cambiar el valor de la variable pasada. La forma clásica de estos parámetros se puede identificar por el uso del símbolo &, tal como se ve en el siguiente código:

```
int minor ( int & arg1, int & arg2, int & arg3 )
{
    // Cuerpo de la función.
    // arg1, arg2 y arg3 son parámetros por referencia.
    // Cualquier cambio hecho a ellos desde aquí afectará a las variables
    // que fueron entregadas a esta función al ser llamada.
}
```

En el código anterior, no es necesario indicar que se pasara la referencia de la variable, y esta función se manda a llamar de manera normal. Sin embargo, hay otra forma de pasar parámetros por referencia

```
int minor ( int *arg1, int *arg2, int *arg3 )
{
    int min;           // Declarar variable min
    min = *arg1;       // Asumir que el menor es arg1, es necesario agregar el asterisco

    if ( arg2 < min )
        min = *arg2;

    if ( arg3 < min )
        min = *arg3;

    return min;        // Regresar valor de min, los valores a , b, c han sido afectados
}

void main (void)
{
    int a, b, c, d;      // Declarar variables a, b, c y d

    /* Aquí asignamos algunos valores iniciales a 'a', 'b' y 'c' */
    /* ... */

    d = minor(&a,&b,&c); // Llamar a minor, los valores de a b y seran afectados, y es
    // necesario enviarlos con un &
    while (1);           // Bucle infinito
}
```

PROTOTIPOS DE FUNCIONES

El prototipo de una función le informa al compilador las características que tiene, como su tipo de retorno, el número de parámetros que espera recibir, el tipo y orden de dichos parámetros. Por eso se deben declarar al inicio del programa.

El prototipo de una función es muy parecido a su encabezado, se pueden diferenciar tan solo por terminar en un punto y coma (;). Los nombres de las variables de entrada son opcionales.

Por ejemplo, en el siguiente boceto de programa los prototipos de las funciones main, func1 y func2 declaradas al inicio del archivo permitirán que dichas funciones sean accedidas desde cualquier parte del programa. Además, sin importar dónde se ubique la función main, ella siempre será la primera en ejecutarse. Por eso su prototipo de función es opcional (si se desea puede no incluirse en las declaraciones al comienzo)

```
#include <avr.h>

void func1(char m, long p); // Prototipo de función "func1"
char func2(int a); // Prototipo de función "func2"
void main(void); // Prototipo de función "main". Es opcional

void main(void)
{
    // Cuerpo de la función
    // Desde aquí se puede acceder a func1 y func2
}
void func1(char m, long p)
{
    // Cuerpo de la función
    // Desde aquí se puede acceder a func2 y main
}
char func2(int a)
{
    // Cuerpo de la función
    // Desde aquí se puede acceder a func1 y main
}
```

Si las funciones no tienen prototipos, el acceso a ellas será restringido. El compilador solo verá las funciones que están implementadas encima de la función llamadora o, de lo contrario, mostrará errores de “función no definida”.

Variables locales y variables globales

Cada variable tiene un ámbito, un área desde donde es accesible, es decir su alcance.

VARIABLES GLOBALES Y VARIABLES LOCALES.

Las variables declaradas fuera de todas las funciones y antes de sus implementaciones tienen carácter global y podrán ser accedidas desde todas las funciones.

Las variables declaradas dentro de una función, incluyendo las variables del encabezado, tienen ámbito local. Ellas solo podrán ser accedidas desde el cuerpo de dicha función.

De este modo, puede haber dos o más variables con el mismo nombre, siempre y cuando estén en diferentes funciones. Cada variable pertenece a su función y no tiene nada que ver con las variables de otra función, por más que tengan el mismo nombre.

En la mayoría de los compiladores C para microcontroladores las variables locales deben declararse al principio de la función.

Por ejemplo, en el siguiente código hay dos variables globales (speed y limit) y cuatro variables locales, tres de las cuales se llaman count.

```
char foo(long);           // Prototipo de función

int speed;                // Variable global
const long limit = 100;   // Variable global constante

void inter(void)
{
    int count;           // Variable local
    /* Este count no tiene nada que ver con el count de las funciones main o foo */
    speed++;             // Acceso a variable global speed
    vari = 0;             // Esto dará ERROR porque vari solo pertenece a la función foo. No
    compilará.
}
void main(void)
{
    int count;           // Variable local count
    /* Este count no tiene nada que ver con el count de las funciones inter o foo */
    count = 0;            // Acceso a count local
    speed = 0;            // Acceso a variable global speed
}
char foo(long count)      // Variable local count
{
    int vari;            // Variable local vari
}
```

Algo muy importante: a diferencia de las variables globales, las variables locales tienen almacenamiento temporal, es decir, se crean al ejecutarse la función y se destruyen al salir de ella.

Si dentro de una función hay una variable local con el mismo nombre que una variable global, la precedencia en dicha función la tiene la variable local. Lo más recomendable es no usar variables globales y locales con el mismo nombre.

VARIABLES STATIC

Es importante aclarar que una variable static local tiene diferente significado que una variable static global. Ahora vamos a enfocarnos al primer caso por ser el más común.

Cuando se llama a una función sus variables locales se crearán en ese momento y cuando se salga de la función se destruirán. Se entiende por destruir al hecho de que la locación de memoria que tenía una variable será luego utilizada por el compilador para otra variable local (así se economiza la memoria). Como consecuencia, el valor de las variables locales se pierde entre llamadas de función.

Para que una variable tenga una locación de memoria independiente y su valor no cambie entre llamadas de función se puede elegir entre declararla como global, o declararla como local estática (lo cual es lo más recomendable).

Una variable se hace estática anteponiendo a su declaración el especificador static. Por defecto las variables estáticas se auto inicializan a 0, pero se le puede dar otro valor en la misma declaración (dicha inicialización solo se ejecuta la primera vez que se llama a la función), en cada llamada de la función el valor de la variable será la misma, aunque este valor si podrá ser modificado de la siguiente manera:

```
static int var1;           // Variable static (inicializada a 0 por defecto)
static int var2 = 50;      // Variable static inicializada a 50
```

Ejemplo:

```
void increm()
{
    static int a = 5;    // Variable local estática inicializada a 5
    a++;                // Incrementar a
}
void main()
{
    int i;              // Declarar variable i
    // El siguiente código llama 10 veces a increm
    for(i=0; i<10; i++)
        increm();
    // Ahora la variable a sí debería valer 15
    while(1);          // Bucle infinito
}
```

VARIABLES VOLATILE

A diferencia de los ensambladores, los compiladores tienen cierta “inteligencia”. Por ejemplo, veamos el siguiente pedazo de código para saber lo que suele pasar con una variable ordinaria:

```
int var;           // Declarar variable var
//...
var = var;         // Asignar var a var
```

El compilador creerá (probablemente como nosotros) que la sentencia `var = var` no tiene sentido (y quizás tenga razón) y no la tendrá en cuenta, la ignorará. Ésta es solo una muestra de lo que significa optimización del código.

El ejemplo anterior fue algo burdo, pero habrá códigos con redundancias aparentes y más difíciles de localizar, cuya optimización puede ser contraproducente. El caso más notable que destacan los manuales de los compiladores C para microcontroladores es el de las variables globales que son accedidas por la función de interrupción y por cualquier otra función.

Para que un compilador no intente “pasarse de listo” con una variable debemos declararla como `volatile`, anteponiéndole dicho calificador a su declaración habitual.

Al utilizar interrupciones propias del microcontrolador, las variables que son utilizadas dentro del código de la interrupción deberán ser declaradas como tipo `volatile`, pues de otra forma sólo se modificarían dentro del código de la interrupción pero al salir seguirían conservando su valor anterior sin mostrar ningún cambio.

Por ejemplo, en el siguiente código de programa la variable `count` debe ser accedida desde la función `interrupt` como desde la función `main`; por eso se le declara como `volatile`.

```
volatile int count;      // count es variable global volátil

void interrupt(void)    // Función de interrupción
{
    // Código que accede a count
}

void main(void)         // Función principal
{
    // Código que accede a count
}
```

Manejo de pines en forma individual

En ocasiones es necesario consultar o escribir en determinados pines o bits en forma individual sin afectar a los demás bits.

Por ejemplo, si se desea activar el bit 0 del puerto D como pin de salida, se puede mandar la instrucción:

`DDRD = 0b00000001;`

Sin embargo con esta instrucción se afectan todos los bits; si lo que se desea es modificar únicamente un bit entonces podría escribirse:

`DDRD = DDRD | 0b00000001;`

Otra forma de enviar esta instrucción sería:

`DDRD |= (1<<DD0); //Provoca que DD0=1 (salida) ; es lo mismo que DDRD = DDRD | 0b00000001`

En ambas instrucciones lo que se indica es que el nuevo valor de DDRD será igual a la operación lógica OR entre el valor anterior de DDRD y 0b00000001.

De forma similar estas instrucciones pueden emplearse con PORTD, por ejemplo

`PORTD |= (1<<PD0); // PD0=0, el resto no cambia; es lo mismo que PORTD = PORTD | 0b00000001`

Ahora, si lo que se desea es que un pin específico cambie su valor a 0, puede emplearse la siguiente expresión

`DDRD &= ~(1<<DD0); //Provoca que DD0=0 (entrada) ; es lo mismo que DDRD = DDRD & 0b11111110`

O bien también podría usarse con:

`PORTD &= ~(1<<PD0); //PD0=0, el resto no cambia; es lo mismo que PORTD = PORTD & 0b11111110`

Esto sucede debido a que $\sim(1<<PD0) = 0b11111110$ y se realiza una operación AND entre PORTD y 0b11111110 en la cual todos los bits más significativos conservan su valor original, y solamente el bit menos significativo cambia a 0.

Las instrucciones lógicas que pueden emplearse de esta forma son:

& AND	OR	\wedge XOR	\sim NOT
0&0=0	0 0=0	0 \wedge 0=0	\sim 0=1
0&1=0	0 1=1	0 \wedge 1=1	\sim 1=0
1&0=0	1 0=1	1 \wedge 0=1	
1&1=1	1 1=1	1 \wedge 1=0	

EJERCICIO

Analice el siguiente código

```
#include <avr/io.h>
#include <util/delay.h>
int main(void)
{
    DDRD |= (1<<DD0); //
    while(1)
    {
        PORTD ^= (1<<PD0); //
        _delay_ms(100);
    }
}
```

EJERCICIO

Analice el siguiente código

```
#define F_CPU 1000000L           //Fijamos la velocidad del CPU a 1MHz

#include <avr/io.h>             //Libreria de entrada y salida
#include <util/delay.h>          //Libreria de los delays

int main(void)
{
    DDRB |= (1<<DDB1);          // DDB1=1=salida
    DDRB &= ~(1<<DDB0);          // DDB0=0=entrada

    PORTB |= (1<<PB0);          // pull-up activa para PB0
    PORTB &= ~(1<<PB1);          // inicia con el led apagado
    while(1)
    {
        if (!(PINB & (1<<PB0))) //Si el botón está presionado...
            PORTB |= (1<<PB1);   //enciende el led
        else
            PORTB &= ~(1<<PB1);   //apaga el led
    }
}
```

Funciones de Interrupción

La Función de Interrupción o ISR va siempre identificada por su Vector de Interrupción, y su esquema varía ligeramente entre un compilador y otro, puesto que no existe en el lenguaje C un formato estándar. Lo único seguro es que es una función que no puede recibir ni devolver ningún parámetro.

En el compilador AVR GCC (WinAVR) la función de interrupción se escribe con la palabra reservada ISR acompañada del Vector_de_Interrupcion.

El Vector_de_Interrupcion tiene exactamente el mismo nombre que el indicado en el datasheet del ATmega8535 pero con la terminación _vect . Las definiciones según el archivo iom8535.h son las siguientes:

```
/* Interrupt vectors */
/* External Interrupt 0 */
#define INT0_vect           _VECTOR(1)
/* External Interrupt 1 */
#define INT1_vect           _VECTOR(2)
/* Timer/Counter2 Compare Match */
#define TIMER2_COMP_vect    _VECTOR(3)
/* Timer/Counter2 Overflow */
#define TIMER2_OVF_vect     _VECTOR(4)
/* Timer/Counter1 Capture Event */
#define TIMER1_CAPT_vect    _VECTOR(5)
/* Timer/Counter1 Compare Match A */
#define TIMER1_COMPA_vect   _VECTOR(6)
/* Timer/Counter1 Compare Match B */
#define TIMER1_COMPB_vect   _VECTOR(7)
/* Timer/Counter1 Overflow */
#define TIMER1_OVF_vect     _VECTOR(8)
/* Timer/Counter0 Overflow */
#define TIMER0_OVF_vect     _VECTOR(9)
/* SPI Serial Transfer Complete */
#define SPI_STC_vect         _VECTOR(10)
/* USART, RX Complete */
#define USART_RX_vect        _VECTOR(11)
/* USART, Data Register Empty */
#define USART_UDRE_vect     _VECTOR(12)
/* USART, TX Complete */
#define USART_TX_vect        _VECTOR(13)
/* ADC Conversion Complete */
#define ADC_vect              _VECTOR(14)
/* EEPROM Ready */
#define EE_RDY_vect          _VECTOR(15)
/* Analog Comparator */
#define ANA_COMP_vect         _VECTOR(16)
/* Two-wire Serial Interface */
#define TWI_vect              _VECTOR(17)
/* External Interrupt Request 2 */
#define INT2_vect             _VECTOR(18)
/* TimerCounter0 Compare Match */
#define TIMER0_COMP_vect     _VECTOR(19)
/* Store Program Memory Read */
#define SPM_RDY_vect          _VECTOR(20)
```

De ahí que una vez configurada una interrupción, el código correspondiente se debe introducir en:

```
ISR (Vector_de_Interrupcion)
{
    // Código de la función de interrupción.
    // No requiere limpiar el flag respectivo. El flag se limpia por hardware
}
```

CONTROL DE LAS INTERRUPCIONES

Cada interrupción habilitada y disparada, saltará a su correspondiente Función de Interrupción o ISR, de modo que a diferencia de algunos otros microcontroladores no será necesario sondear los flags de interrupción para conocer la fuente de interrupción.

Los AVR tienen un hardware que limpia automáticamente el bit de Flag apenas se empieza a ejecutar la función de interrupción. Pero puesto que los flags se habilitan independientemente de si las interrupciones están habilitadas o no, en ocasiones será necesario limpiarlos por software y en ese caso debemos tener la especial consideración de hacerlo escribiendo un uno y no un cero en su bit respectivo..

Al ejecutarse la función de interrupción también se limpia por hardware el bit enable general I (el que se habilita con la instrucción sei();) para evitar que se disparen otras interrupciones cuando se esté ejecutando la interrupción actual. Sin embargo, la arquitectura de los AVR le permite soportar ese tipo de interrupciones, llamadas recurrentes o anidadas, y si así lo deseamos podemos setear en el bit I dentro de la ISR actual.

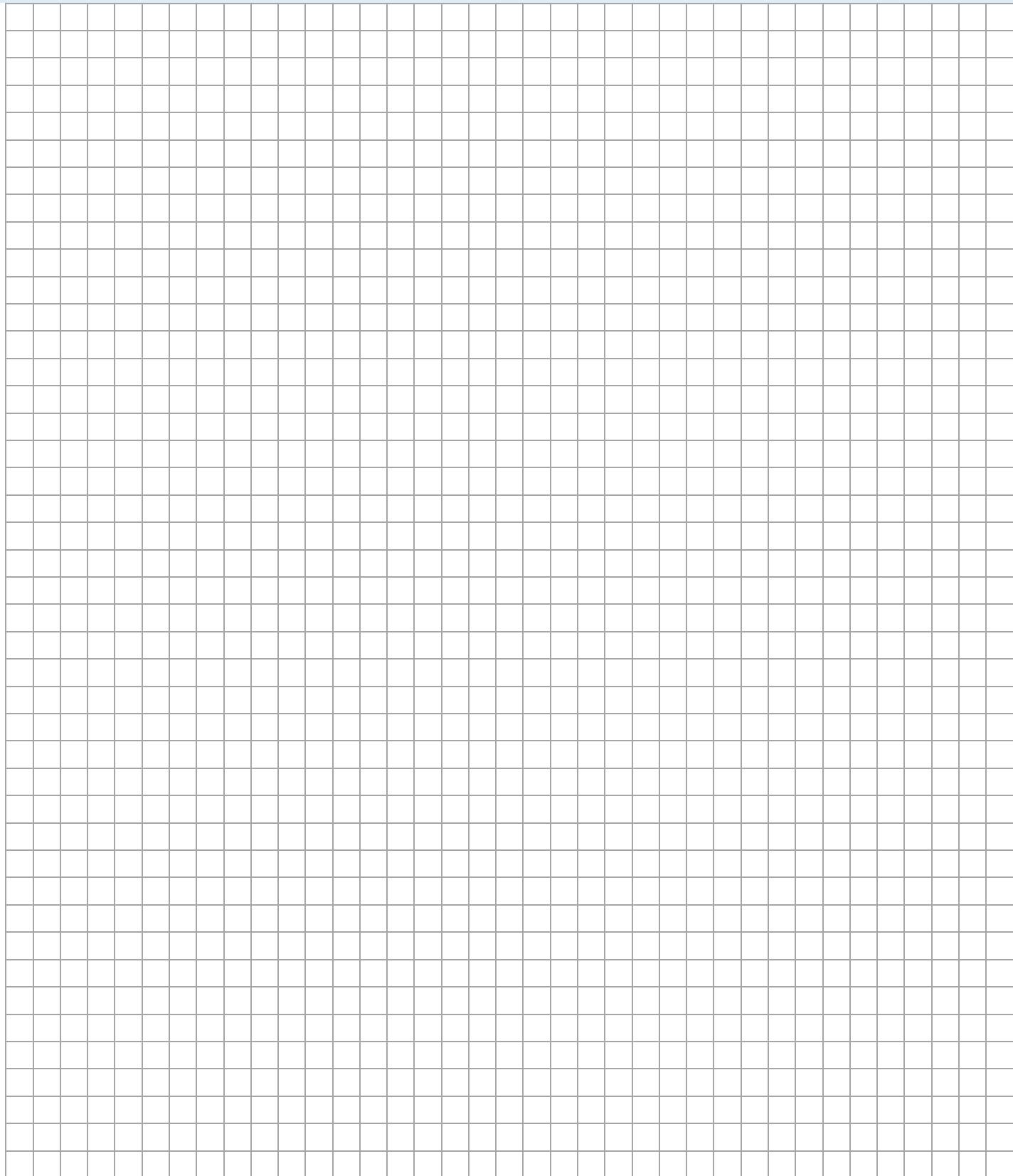
Este bit de enable general para las interrupciones, es de especial importancia, es por ello existen dos exclusivas instrucciones de ensamblador llamadas sei (para setear I) y cli (para limpiar I). En C las instrucciones correspondientes son:

```
sei();
cli();
```

Para poder emplearlas es necesario agregar la siguiente librería:

```
#include <avr/interrupt.h> //Libreria de interrupciones
```

NOTAS PERSONALES – PROGRAMACIÓN EN LENGUAJE C

A large grid of squares, approximately 20 columns by 25 rows, designed for taking notes or drawing diagrams.

