GUÍA de programación para microcontroladores

Ejercicios prácticos

DIANA GUERRÓN

INSTITUTOS
Superiores Técnicos y Tecnológicos

Instituto Superior Tecnológico “Vicente Fierro”

GUÍA de programación para microcontroladores

Diana Guerrón
dianagbolanos@gmail.com

TECNOLOGÍA SUPERIOR EN ELECTRICIDAD

Asignaturas relacionadas:

- **Electrónica**
- **Automatización y control de procesos**

Coordinación de Investigación, Desarrollo Tecnológico e Innovación

Datos de catalogación bibliográfica
Guerrón, Diana. (2019). Guía de programación para microcontroladores. Ejercicios prácticos. Tulcán: Instituto Tecnológico Superior "Vicente Fierro".

Esta guía es el resultado final de un proyecto presentado, aprobado y desarrollado en el Instituto Superior Tecnológico "Vicente Fierro", articulando las funciones de Investigación, Vinculación, Docencia y Bienestar, durante los años 2019-2020.

Autora:
Diana Aracely Guerrón Bolaños
dianagbolanos@gmail.com

Diseño:
Lic. Edgar Fernando Pazmiño Palma

Portada:
Elaborada por Sai Kiran Anagani
Tomada de https://unsplash.com/s/photos/technology-ilustrations

ISBN:
978-9942-8747-6-4
Impreso en Tulcán, Ecuador.
Primera edición, diciembre 2019.

Aval académico:
Ing. Roberth Narváez MBA.
Rector del Instituto Superior Tecnológico "Vicente Fierro"

Ing. Karina Játiva
Coordinadora de Investigación, Desarrollo Tecnológico e Innovación

Ing. Hugo Revelo
Responsable de publicaciones

Pares revisores:
Lic. Nathalia Narváez
Ing. Diego Romo

Materia:
Programación

Tópico:
Programación Orientada a Objetos (OOB)

SENESCYT
Instituto Superior Tecnológico "Vicente Fierro".
Avenida Andrés Bello y Panamericana Norte.
Tucán, Carchi, Ecuador.

Índice

Índice de Tablas

Índice de Figuras

METODOLOGÍA – ORIENTACIÓN

La presente guía está orientada a la programación de microcontroladores, principalmente los basados en ATMEGA 328.

En primera instancia se introduce al estudiante a los conceptos básicos de la programación en lenguaje de alto nivel C++, puesto que el estudio de microcontroladores no se basa en conocer solo su arquitectura sino también en conocer la aplicación de las instrucciones que le permitirán comprender el funcionamiento básico de todos los sistemas automáticos.

Los problemas presentes en esta guía que estimulan la creatividad del lector mediante la realización de estos proyectos basados en aplicaciones reales que le permitirán comprender que para dar solución a un problema de control hay más de un camino.

COMPETENCIAS

Al finalizar esta guía, el lector estará en la capacidad de dar Solución a problemas a través del manejo de microprocesadores o microcontroladores y periféricos, haciendo uso de herramientas de programación en lenguaje C, aplicándolos al desarrollo de sistemas automáticos y el desarrollo de aplicaciones destinadas a solucionar procesos industriales, de forma clara y concreta.

CAPÍTULO I: LENGUAJE DE PROGRAMACIÓN C

1.1. Lenguaje de programación C

C es un lenguaje de alto nivel, frecuentemente es utilizado para programar microprocesadores y aplicaciones avanzadas, cuenta con un compilador cuya función es simplificar el trabajo del programador haciendo que la interfaz con el usuario sea amigable. Permite crear una programación estructurada, es decir, el programa se divide en módulos o funciones independientes entre sí (Kernighan & Ritchie, 1991).

1.2. Diseño de un programa

La programación en lenguaje C, consta de las siguientes etapas:

1. Escribir un código basado en comandos C, que representa la lógica diseñada en el algoritmo del programa y,

2. La traducción del código a lenguaje de máquina mediante el compilador, lo que permitirá su posterior ejecución.

El proceso de diseño de un programa conlleva de las fases descritas a continuación:

1. Análisis de los objetivos del programa, se establecen los datos de entrada, salida, la forma en la que se van a procesar y la funcionalidad total del programa. Para esto el programador no se tiene que asociar a ningún lenguaje de programación ni a comandos especiales.

2. Diseño del algoritmo, una vez que se tiene claro el problema a resolver, se define la organización del programa, representación de los datos y la comunicación con el usuario.

3. Codificación del algoritmo es la traducción del algoritmo en la secuencia necesaria de instrucciones usando exclusivamente comandos C y que generalmente se escriben en un archivo simple de texto al que se le llama archivo de código fuente.

1.3. Estructura de un programa en C

Antes de comenzar a desarrollar cualquier programa se debe tener en cuenta una serie de elementos básicos de su estructura como se indica en la figura 1.

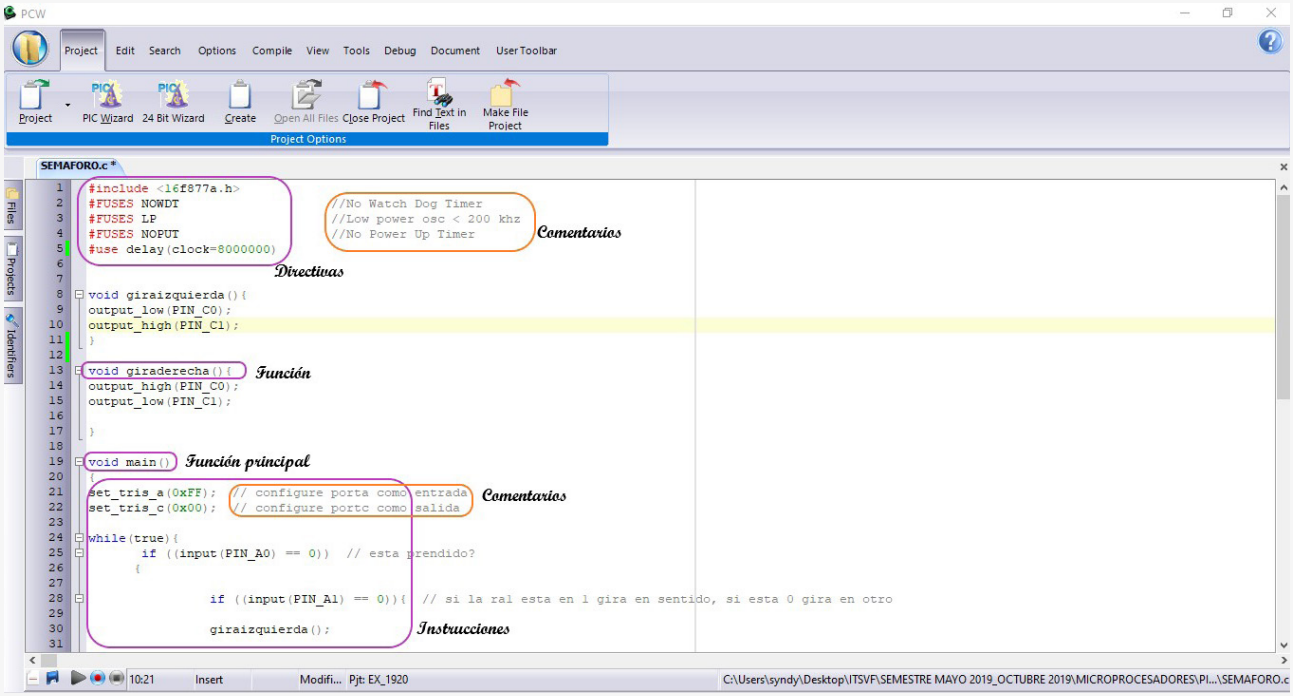
- **Directivas de procesamiento o librerías:** Intervienen en la conversión de las instrucciones del programa en lenguaje máquina.

- **Funciones:** Son un conjunto de instrucciones presentes en el desarrollo del programa. Puede haber más de una función, pero siempre debe estar presente la función principal *main*.

- **Instrucciones:** Líneas de código que determinan el comportamiento del microcontrolador.

- **Comentarios:** Describen la funcionalidad de cada instrucción.

Figura 1. Estructura de programa en lenguaje C



Elaborado por: Guerrón D.

1.4. Tipos de datos

Los más utilizados en el desarrollo de programas en C son los descritos a continuación en la tabla 1.

Tabla 1.1. Tipos y características de datos utilizados en programación

TIPO	TAMAÑO	RANGO	DESCRIPCIÓN
int1 short	1 bit	0 a 1	Entero de 1 bit
Int int8	8 bits	0 a 255	Entero
int16 long	16 bits	0 a 65535	Entero de 16 bits
int32	32 bits	0 a 4294967295	Entero de 32 bits
Float	32 bits	$\pm 1175 \times 10^{-38}$ a $\pm 3402 \times 10^{+38}$	Coma flotante
Char	8 bits	0 a 255	Carácter
Void	-	-	Sin valor
Signed int8	8 bits	-128 a +127	Entero con signo
Signed int16	16 bits	-32768 a +32767	Entero largo con signo
Signed int32	32 bits	-2^{31} a $+(2^{31}-1)$	Entero 32 bits con signo

Tomado de: (Kernighan & Ritchie, 1991)

1.5. Constantes

Las constantes pueden ser definidas de varias maneras, como se muestra en las tablas 1.2 y 1.3; con el fin de configurar los puertos del microcontrolador.

Tabla 1.2. Tipos de numeración empleados en programación

123	Decimal
0123	Octal (0)
0x123	Hexadecimal (0x)
0b010011	Binario (0b)
'x'	Carácter
'\010'	Carácter octal
'\xA5'	Carácter hexadecimal

Tomado de: (Kernighan & Ritchie, 1991)

Tabla 1.3. Constantes definidas con sufijo

int8	127U
Long	80UL
Signed int16	80L
Float	3.14L
Char	Con comillas simples'c'

Tomado de: (Kernighan & Ritchie, 1991)

En la programación se requiere que el programa se encuentre bien estructurado con el fin de que el usuario pueda visualizar de mejor manera el resultado del programa compilado, esto se consigue haciendo uso de los denominados caracteres especiales que se muestran en la tabla 1.4.

Tabla 1.4. Caracteres especiales para ordenar datos en la compilación del programa

\n	Línea nueva
\r	Retorno de carro
\t	Tabulación
\b	Backspace (Espacio entre caracteres o números)

Elaborado por: Guerrón D.

1.6. Variables

Reservan el espacio de memoria RAM e indican el tipo de valor que se va a almacenar en la variable. Dependiendo del lugar donde son declaradas pueden ser globales o locales, las variables globales pueden ser utilizadas en todas las funciones del programa, mientras que las variables locales son utilizadas únicamente en la función donde fue declarada.

Todas las variables deben ser declaradas con un tipo de dato antes de ser llamadas en las instrucciones, las variables locales pueden ser inicializadas o no, es decir pueden tener o no un valor inicial, para las variables globales es recomendable inicializarlas con un valor que puede ser 0 (García Briejo, 2008). Se sigue la siguiente estructura:

(TIPO DE DATO) (NOMBRE DE LA VARIABLE) (=VALOR INICIAL);
int16 contador = 0;
float velocidad = 50,36;

1.7. Operadores

1.7.1. Asignación:

Este tipo de operadores son una combinación de operadores aritméticos que asignan a una variable un valor de una operación de forma directa. Estos operadores se muestran en la tabla 1.5.

Tabla 1.5. Lista de operadores de asignación para cada operación y función lógica

OPERADOR	DESCRIPCIÓN
+=	Asignación de suma (x+=y es lo mismo que x=x+y)
-=	Asignación de resta (x-=y es lo mismo que x=x-y)
*=	Asignación de multiplicación (x*=y es lo mismo que x=x*y)
/=	Asignación de división (x/=y es lo mismo que x=x/y)
%=	Asignación del resto de la división (x%=y es lo mismo que x=x%y)
«=	Asignación de desplazamiento a la izquierda (x«=y es lo mismo que x=x«y)
»=	Asignación de desplazamiento a la derecha (x»=y es lo mismo que x=x»y)
&=	Asignación AND de bits (x&=y es lo mismo que x=x&y)
|=	Asignación OR de bits (x|=y es lo mismo que x=x|y)
^=	Asignación OR EXCLUSIVA de bits (x^=y es lo mismo que x=x^y)

Tomado de: (Kernighan & Ritchie, 1991)

1.7.2. Aritméticos:

En la tabla 1.6 se muestran los operadores aritméticos, en su mayoría son los empleados en operaciones matemáticas.

Tabla 1.6. Operadores aritméticos en programación

OPERADOR	DESCRIPCIÓN
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo, resta de la división
++	Incremento
--	Decremento
Sizeof	Determina el tamaño en bytes de un operando

Tomado de: (Kernighan & Ritchie, 1991)

1.7.3. Lógicos:

Representa las funciones básicas de la lógica digital que comparan o relacionan los estados lógicos de dos o más variables.

Tabla 1.7. Operadores lógicos

OPERADOR	DESCRIPCIÓN
!	Not
&&	AND
||	OR

Elaborado por: Guerrón D.

1.7.4. Relacionales:

Realizan la comparación de estados o cantidades asignadas a 2 o más variables.

Tabla1.8. Operadores relacionales

OPERADOR	DESCRIPCIÓN
<	Menor que
>	Mayor que
>=	Mayor o igual que
<=	Menor o igual que
==	Igual
!=	Distinto
?:	Expresión condicional

Elaborado por: Guerrón D.

1.8. Funciones

Las funciones son bloqueas de sentencias, se deben enmarcar dentro de las funciones y deben ser definidas antes de utilizarse.

Una función puede ser invocada desde una sentencia de otra función; puede devolver un valor a la sentencia que la ha llamado. El tipo de dato se indica en la definición de la función; en el caso de no indicarse se asume que es un *int8* y en el caso de no devolver un valor se debe especificar el valor *VOID*. La función puede recibir parámetros o argumentos (García Briejo, 2008).

La estructura de una función es:

```
Tipo_de_dato Nombre_de_la_función (parámetro/s) {
Sentencias
}
```

1.9. Sentencias de Control

Los lenguajes de programación disponen de dos formas para controlar el *flujo del programa*, estas son:

- **Decisión:** Decide ejecutar sentencias entre acciones alternativas.
- **Repetición:** Repite una secuencia de sentencias hasta que se cumpla una determinada condición.

1.9.1. Sentencias de decisión binaria

Decide la ejecución de las sentencias entre dos alternativas.

- **Sentencia if:** Elije si una sentencia o sentencias se ejecutan o no. Su estructura es la siguiente:

```
if (expresión) {
        sentencia;
        }
```

Si la expresión es verdadera la sentencia se ejecuta, por el contrario, si la expresión es falsa, la sentencia se ignora.

Ejemplo:

```
if (num==0){
printf (“El número es igual a cero”);
num++;
}
```

- Sentencia if-else: Elije entre dos acciones diferentes, es decir, permite la elección entre dos sentencias. Su estructura es la siguiente:

```
        if (expresión) {
                sentencia1;
        }else{
                Sentencia2;
}
```

Si la expresión es verdadera se ejecuta la sentencia1 y excluye la sentencia2; si la expresión es falsa se ejecuta la sentencia2 siendo excluyente con la sentencia1.

Ejemplo:

```
        if(num<=0){
y=num;
}else{
y=num++;
}
```

- Sentencia else-if: Se emplea para elegir una única acción entre más de dos alternativas. Su estructura es la siguiente:

```
                if (expresión1) {
                        sentencia1;
                }else if (expresión2){
                        Sentencia2;
}else{
Sentencia3;
                }
```

Si la expresión1 es verdadera se ejecuta la sentencia1 y excluye la sentencia2 y sentencia3; pero si la expresión2 es verdadera se ejecuta la sentencia2 siendo excluyente con la sentencia1 y sentencia3; pero se la expresión1 y expresion2 son falsas se ejecuta la sentencia3.

Ejemplo:

```
if (cuenta<1000) {
                bono=0;
        }else if (bono<3000){
                bono=1;
}else{
bono=3;
}
```

1.9.2. Sentencia de decisión múltiple: Switch

Se emplea cuando se necesita elegir una o varias acciones entre algunas alternativas, por lo tanto, esta sentencia no es excluyente. Su estructura es la siguiente:

```
switch (expresión){
                case etiqueta1: sentencias:
break;
case etiqueta2: sentencias:
break;
        }
```

Ejemplo:

```
#include <conio.h>
#include <stdio.h>
int main() {
        char opcion;
        int n1, n2;
printf( "\n   1. suma ");
printf( "\n   2. resta " );
```

```
printf( "\n   3. multiplicacion ");
printf( "\n   4. division " );
printf( "\n   5. salir.\n" );
printf( "\n   Introduzca opción (1-5): " );
scanf( "%c", &opcion);
        if(opcion>0 && opción <=5){
                printf( "primer valor a sumar: " );
                scanf( "%d", &n1);
                printf( "segundo valor a sumar: " );
                scanf( "%d", &n2);
        }
        switch ( opcion){
                case '1':
                printf( "\n   %d + %d = %d\n", n1, n2, n1 + n2 );
                break;
                case '2':
                printf( "\n   %d - %d = %d\n", n1, n2, n1 - n2 );
                break;
                case '3':
                printf( "\n   %d * %d = %d\n", n1, n2, n1 * n2 );
                break;
                case '4':
                if (n2!= 0 )
                printf( "\n   %d div %d = %d ( Resto = %d )\n", n1, n2, n1/n2, n1%n2 );
                else
                        printf( "\n No se puede dividir entre cero.\n" );
                break:
}
        }
        return 0;
}
```

La sentencia *break* es una sentencia de salto que hace que el control del programa se salga de la sentencia switch y se dirija a la sentencia situada inmediatamente después de la misma.

La *expresión* debe estar encerrada entre paréntesis y debe tener un valor entero o de tipo char.

La *etiqueta* debe ser de valor constante o ser una expresión formada únicamente por contantes de tipo entero o de tipo char.

El tipo de dato de la expresión como el de la etiqueta deben tener el mismo tipo de dato.

1.10. Sentencias de repetición

1.10.1. Sentencia while

Crea un bucle o ciclo que se repite hasta que la expresión se vuelva falsa. La estructura general es:

```
while(expresión){
        sentencia;
}
```

Si la expresión es verdadera la sentencia se ejecuta, y la expresión (test) se vuelve a evaluar para comprobar si su valor de verdad se mantiene. Este ciclo de test y ejecución se repite hasta que la expresión se vuelva falsa. Cuando se construye un bucle de este tipo se debe incluir una variación del valor de la expresión del valor del test de tal manera que el bucle acabe cuando la expresión sea falsa.

Ejemplo:

```
#include <math.h>
#include <stdio.h>
int main(){
        float radio;
        printf( "Introduzca radio: " );
        scanf( "%f", &radio );
        while ( radio <= 0 ){
                printf( "ERROR: El radio debe ser mayor que cero." );
                printf( "\nIntroduzca radio: " );
                scanf( "%f", &radio );
        }
        printf("El area de la esfera de radio %f es: %f", radio, 1* 3.141592 * pow( radio, 2 ) );
        return 0;
}
```

1.10.2. Sentencia for

Agrupa en un bucle tres acciones: inicializar un contador, compararlo con un límite e incrementarlo cada vez que se ejecute el bucle. La estructura general es:

```
for(expresión1; expresión2: expresión3){
sentencia;
}
```

La *expresión1* es normalmente la inicialización del contador y se ejecuta una sola vez al comenzar el bucle.

La *expresión2* es el test de comparación, se evalúa antes de cada ejecución potencial del bucle y cuando esta expresión es falsa termina el bucle.

La *expresión3* es normalmente el incremento del contador (actualización), se evalúa al final de casa ejecución del lazo, e inmediatamente después se efectúa el nuevo test de comparación.

Ejemplo:

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[]) {
        int x;
        for (int i=1;i<=10;i++){
                x=(i*i);
                printf("\n %i", x);
        }
                return 0;
}
```

1.10.3. Sentencia do-while

Crea un bucle que se repite hasta que la expresión de control test sea falsa. La diferencia con la sentencia while radica en que la condición del test debe estar la final del cuerpo del bucle. La estructura general de esta sentencia es:

```
do{
        sentencia;
}while (expresión);
```

El bucle do-while se ejecuta siempre una vez como mínimo, dado que la condición de evaluación se encuentra después de las sentencias.

Ejemplo:

```
do{
        printf("Ingrese un número entero positivo menor a 100");
        scanf("%d",&num);
}while (num>=0 || num<=100);
```

1.11. Ejercicios propuestos

1. Diseñe un programa que, ingresando la distancia recorrida, determine el monto a pagar por el alquiler de un vehículo y el monto incluido del impuesto. Basándose en el año de fabricación y el peso del automóvil en una fábrica se determina la tarifa de registro según la siguiente tabla:

Año (modelo)	Peso (lb.)	Categoría de Peso	Tarifa de Registro
1970 o anterior	Menos de 2.700	1	$ 11.600
	2.700 a 3.800	2	$ 23.200
	Más de 3.800	3	$ 34.800
1971 a 1979	Menos de 2. 700	4	$ 13.000
	2.700 a 3.800	5	$ 26.000
	Más de 3.800	6	$ 39.000
1980 o posterior	Menos de 3.500	7	$ 12.000
	3.500 o más	8	$ 46.000

2. Realizar un programa que tecleando 2 valores permita identificar si es mayor, sumar o multiplicar

3. Una institución benéfica europea ha recibido tres donaciones en soles, dólares y marcos. La donación será repartida en tres rubros: 60% para la implementación de un centro de salud, 30% para un comedor de niños y el resto para gastos administrativos. Diseñe un algoritmo que determine el monto en euros que le corresponde a cada rubro. Considere que: 1 dólar = 3.52 soles, 1 dólar = 2.08 marcos, 1 dólar = 1.07 euros.

4. Crear un programa que pida al usuario su contraseña numérica deberá terminar cuando el usuario introduzca como contraseña 4,5,6,7.

5. Lea los tres lados de un triángulo y determine si corresponde a un triángulo rectángulo en caso afirmativo calcule el área

6. Desarrollar un algoritmo que determine en un conjunto de cien números la cantidad de negativos, cuántos son mayores de 50 y cuántos están comprendidos entre 25 y 45.

7. Leer una serie de números para determinar si los números son primos o no hasta digitar el número 0. Un número primo es aquel que puede dividirse únicamente por si mismo y por la unidad.

8. Realizar un programa que ingrese desde el teclado un número entero binario e imprimir su equivalente decimal. Validar que los dígitos del número sean binarios solo 0 y 1.por ejemplo, el equivalente decimal del número 1011 binario es: 1*8+0*4+1*2+1*1= 8+0+2+1, es decir 11.

CAPÍTULO II: INTRODUCCIÓN A LOS MICROCONTROLADORES

2.1. Microcontrolador

"Es un circuito integrado programable, capaz de ejecutar órdenes o secuencias grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica dentro del ordenamiento del mismo y a su vez permiten obtener configuraciones diferentes". (Esparza Silva, 2003)

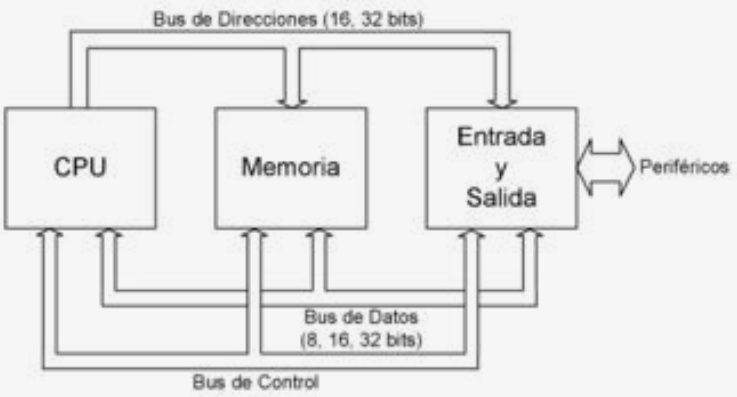
2.2. Microprocesador VS Microcontrolador

"Para que un microprocesador pueda funcionar como un sistema aplicado al control, son necesarios chips adicionales, por ejemplo, dispositivos de memoria para almacenar programas y datos, así como puertos de entrada/salida para permitir que se comunique con el mundo exterior y reciba señales desde él. El microcontrolador integra en un chip de microprocesador con memoria, interfaces de entrada/salida y otros dispositivos periféricos como temporizadores.

Un microcontrolador común tiene terminales para la conexión externa de entradas y salidas, alimentación eléctrica y señales de reloj y de control. Las conexiones de entrada y salida se agrupan en unidades denominadas puertos de entrada/salida. Por lo general, estos puertos tienen ocho líneas para poder transportar una palabra de datos de 8 bits. Para una palabra de 16 bits utilizan dos puertos, uno para transmitir los 8 bits inferiores, y otro para los 8 bits superiores. Los puertos pueden ser sólo entrada o sólo salida, o programables para funcionar como entrada o salida". (Bolton , 2013)

En la figura 2, se puede observar la estructura típica de un microprocesador, con sus componentes fundamentales: CPU (Unidad Central de Procesos), memoria, entrada y salida. Los bloques están conectados mediante líneas eléctricas llamadas buses, los cuales pueden ser de direcciones (transportan direcciones de memoria de entrada o salida), de datos (transportan datos o instrucciones), o de control (transportan señales de control diversas). (Barra Zapata & Barra Zapata, 2015)

Figura 2. Esquema general de un microprocesador



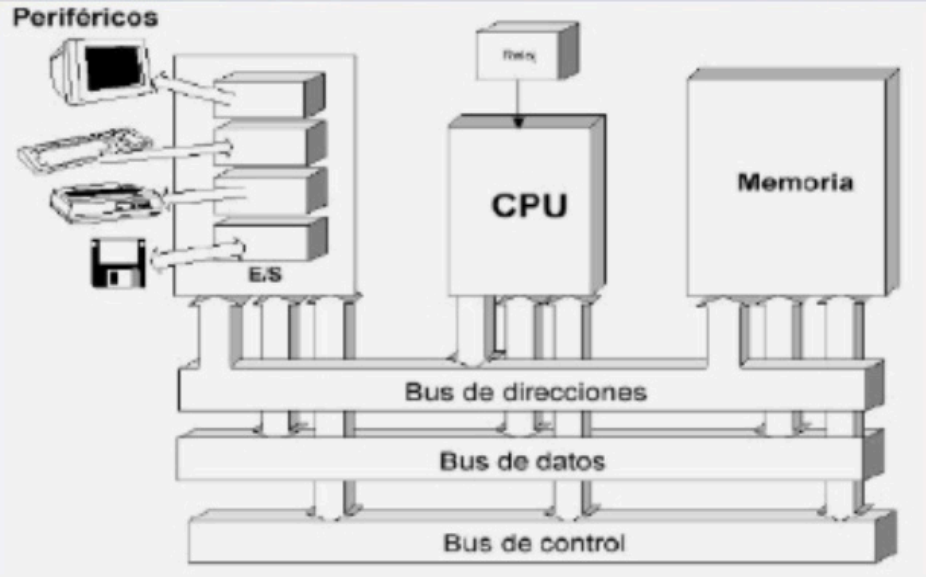
Tomado de: (García Briejo, 2008)

2.3. Arquitectura básica del microcontrolador

2.3.1. Arquitectura de Von Neumann

Dispone de una sola memoria principal donde se almacenan datos e instrucciones de forma indistinta, a esta memoria se accede a través de un sistema de buses único (direcciones, datos y control). (Valdés Pérez & Pallás Areny, 2007)

Figura 3. Estructura de la Arquitectura de Von Neumann



Tomado de: (Valdés Pérez & Pallás Areny, 2007)

2.3.2. Arquitectura de Harvard

Cuenta con dos memorias independientes, una que contiene solo instrucciones y la otra contiene solo datos, sin embargo, las dos cuentan con sus respectivos buses de acceso que permiten realizar operaciones de acceso (lectura o escritura) simultáneamente. (Valdés Pérez & Pallás Areny, 2007)

Figura 4. Estructura de la Arquitectura de Harvard

Tomado de: (Valdés Pérez & Pallás Areny, 2007)

2.4. Procesador

Determina las características principales a nivel de hardware y software. Su función es direccionar la memoria de instrucciones, recibir el código de la instrucción en curso, su decodificación

y la ejecución de la operación que implica la instrucción, así como la búsqueda de los operandos y el almacenamiento de los resultados.

Por su arquitectura y funcionalidad, se clasifican en:

- **CISC (Computadores de juego de instrucciones complejo):** Abarca más de 80 instrucciones máquina, algunas de las cuales requieren muchos ciclos para su ejecución. Una de las ventajas de estos procesadores, es que ofrecen al programador instrucciones complejas que actúan como macros.

- **RISC (Computadores de juego de instrucciones reducido):** Cuenta con un repertorio de instrucciones máquina muy reducido y simples, por lo general estas instrucciones se ejecutan en un ciclo, permitiendo optimizar el hardware y software.

- **SISC (Computador de juego de instrucciones específico):** Los microcontroladores que poseen este tipo de procesador, son empleados para aplicaciones concretas, el juego de instrucciones es reducido y específico. (Mandado , Menéndez, Fernández, & López, 2007)

2.5. Unidad de control

Esta unidad es de las más importantes en el procesador, en ella recae la lógica necesaria para la decodificación y ejecución de las instrucciones, el control de los registros, la ALU (realiza operaciones como sumas, restas, y operaciones lógicas típicas del álgebra de Boole) y los buses.

La unidad de control es uno de los elementos fundamentales que determinan las prestaciones del procesador, ya que su tipo y estructura, determina parámetros tales como el tipo de conjunto de instrucciones, velocidad de ejecución, tiempo del ciclo de máquina, tipo de buses que puede tener el sistema, manejo de interrupciones, entre otras.

Las unidades de control, son el elemento más complejo de un procesador y normalmente están divididas en unidades más pequeñas trabajando de conjunto. La unidad de control agrupa componentes tales como la unidad de decodificación, unidad de ejecución, controladores de memoria cache, controladores de buses, controlador de interrupciones, pipelines, entre otros elementos, dependiendo siempre del tipo de procesador. (Esparza Silva, 2003)

2.6. Memoria

La memoria de instrucciones y datos está integrada en el propio chip. Una parte debe ser no volátil, destinada a contener el programa de instrucciones que gobierna la aplicación (ROM); otra parte de la memoria debe ser volátil, destinada a guardar variables y datos (tipo RAM).

- **Memoria RAM:** Este tipo de memorias pierden la información cuando son desconectadas del sistema de alimentación. En el microcontrolador está destinada a guardar variables y datos y se las conoce como RAM (Random Access Memory).

- **Memoria ROM:** Contiene el conjunto de instrucciones que conforman el programa y para almacenar un número pequeño de datos que deben conservarse pese a que se deje de aplicar la alimentación del microcontrolador.

Según el tipo de memoria ROM que dispongan los microcontroladores, la aplicación y utilización de los mismos es diferente. Las memorias ROM que se pueden encontrar son:

- **ROM con máscara:** Es una memoria no volátil de solo lectura cuyo contenido se graba durante la fabricación del chip. Este tipo de microcontroladores en utilizado cuando se requiere trabajar con cantidades superiores a varios miles de unidades.

- **OTP (One Time Programmable:** Es una memoria no volátil de solo lectura programable una sola vez por el usuario. El usuario puede escribir el programa en el chip haciendo uso de un grabador controlado por un programa de computador.

- **EPROM (Erasable Programmable Read Only Memory):** Puede borrar y grabar el programa varias veces. La grabación se realiza empleando el grabador controlado por la computadora. Para borrar el programa la EPROM debe someterse a rayos ultravioleta durante varios minutos.

- **EEPROM (Electrical Erasable Programmable Read Only Memory):** Es una memoria de solo lectura, programable y borrable eléctricamente, es decir, la programación y el borrado se realizan eléctricamente con el grabador controlado por el programa de computador. Este tipo de memoria es relativamente lenta.

- **Flash:** Es una memoria de solo lectura, de bajo consumo que se puede escribir y borrar. Funciona como una ROM y una RAM pero consume menos y es más pequeña. (Esparza Silva, 2003)

2.7. Puertas de entrada y salida

Los pines del encapsulado que contiene al microcontrolador tiene la función de soportar las líneas de entrada y salida que sirven de comunicación entre el computador y los periféricos exteriores. (Bolton , 2013)

2.8. Reloj Principal

Es un circuito oscilador que genera una onda cuadrada de alta frecuencia, cuya función es configurar los impulsos de reloj usados en la sincronización de todas las operaciones del sistema. Para seleccionar y estabilizar la frecuencia de trabajo se requiere de componentes externos, generalmente se emplea el cristal de cuarzo y elementos pasivos. (Bolton , 2013)

2.9. Recursos especiales

2.9.1. Timers o temporizadores

De acuerdo a los requerimientos del programa se configuran como temporizadores para controlar periodos de tiempo y como contadores para llevar la cuenta de los acontecimientos que suceden en el exterior.

2.9.2. Watchdog o perro guardián

Es un contador interno que origina un reset cuando el temporizador se desborda y pasa por 0, provoca un reset automáticamente en el sistema.

2.9.3. Brownout o protección ante fallo de alimentación

Es un circuito que resetea el microcontrolador cuando el voltaje de alimentación (VDD) es inferior a un voltaje mínimo (Brownout). Mientras el voltaje de alimentación sea inferior al de

brownout, el dispositivo se mantiene reseteado y empieza a funcionar normalmente cuando sobrepasa dicho valor.

2.9.4. Estado de reposo o bajo consumo

En algunos casos se presentan tiempos de reposo en los cuales el microcontrolador no trabaja y debe esperar a que un impulso externo genere la señal de activación, por tal razón y para ahorrar energía los microcontroladores disponen de una interrupción por la que pasan al estado de reposo o bajo consumo, en los cuales los requerimientos de potencia son mínimos.

2.9.5. Conversor analógico digital (A/D)

Posee un multiplexor que permite medir señales analógicas en forma digital.

2.9.6. Conversor digital análogo (D/A)

Transforma los datos digitales en su correspondiente señal analógica.

2.9.7. Modulador de ancho de impulsos

Permite obtener una señal periódica en la que se puede modificar su ciclo de trabajo. Es decir, puede variarse el tiempo en el cual la señal está en nivel alto frente al tiempo que está a nivel bajo.

2.10. Registro

Son un espacio de memoria muy reducido pero necesario para cualquier microprocesador, de aquí se toman los datos para varias operaciones que debe realizar el resto de los circuitos del procesador. Los registros sirven para almacenar los resultados de la ejecución de instrucciones, cargar datos desde la memoria externa o almacenarlos en ella.

Aunque la importancia de los registros parezca trivial, no lo es en absoluto. De hecho, una parte de los registros, la destinada a los datos, es la que determina uno de los parámetros más importantes de cualquier microprocesador. Cuando escuchamos que un procesador es de 4, 8, 16, 32 o 64 bits, nos estamos refiriendo a procesadores que realizan sus operaciones con registros de datos de ese tamaño, y por supuesto, esto determina muchas de las potencialidades de estas máquinas.

Mientras mayor sea el número de bits de los registros de datos del procesador, mayores serán sus prestaciones, en cuanto a poder de cómputo y velocidad de ejecución, ya que este parámetro determina la potencia que se puede incorporar al resto de los componentes del sistema, por ejemplo, no tiene sentido tener una ALU de 16 bits en un procesador de 8 bits.

Por otro lado, un procesador de 16 bits, puede que haga una suma de 16 bits en un solo ciclo de máquina, mientras que uno de 8 bits deberá ejecutar varias instrucciones antes de tener el resultado, aun cuando ambos procesadores tengan la misma velocidad de ejecución para sus instrucciones. El procesador de 16 bits será más rápido porque puede hacer el mismo tipo de tareas que uno de 8 bits, en menos tiempo. (García Briejo, 2008)

2.11. Buses

Son el medio de comunicación que utilizan los diferentes componentes del procesador para intercambiar información entre sí, eventualmente los buses o una parte de ellos estarán reflejados en los pines del encapsulado del procesador.

En el caso de los microcontroladores, no es común que los buses estén reflejados en el encapsulado del circuito, ya que estos se destinan básicamente a las E/S de propósito general y periféricos del sistema.

Existen tres tipos de buses:

- Dirección: Se utiliza para seleccionar al dispositivo con el cual se quiere trabajar o en el caso de las memorias, seleccionar el dato que se desea leer o escribir.
- Datos.
- Control: Se utiliza para gestionar los distintos procesos de escritura lectura y controlar la operación de los dispositivos del sistema. (Kernighan & Ritchie, 1991)

CAPÍTULO III: ATMEGA 328

3.1. ARDUINO

"Es un circuito integrado programable, capaz de ejecutar órdenes o secuencias grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica dentro del ordenamiento del mismo y a su vez permiten obtener configuraciones diferentes". (Esparza Silva, 2003)

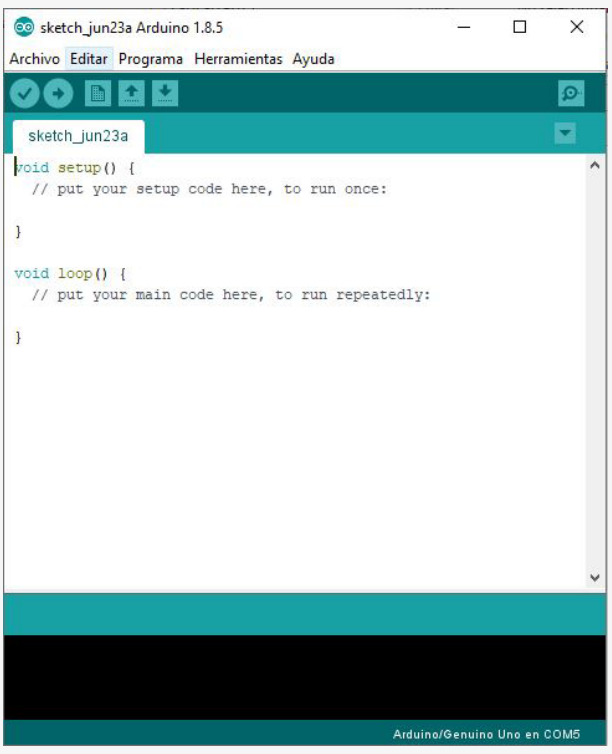
Arduino Uno es una placa electrónica basada en el microcontrolador ATmega328. Cuenta con 14 entradas/salidas digitales, de las cuales 6 se pueden utilizar como salidas PWM (Modulación por ancho de pulsos) y otras 6 son entradas analógicas. Además, incluye un resonador cerámico de 16 MHz, un conector USB, un conector de alimentación, una cabecera ICSP y un botón de reseteo. (Arduino, 2015)

3.1.1. IDE de ARDUINO

El IDE de Arduino es un entorno de desarrollo que está basado en el lenguaje de programación processing, con lo que el usuario aprenderá a programar con el dominio de desarrollo Arduino. Funciona en sistemas operativos Windows y Linux. El software está publicado bajo una licencia libre, lo que permite a los programadores y desarrolladores de software ampliarlo haciendo uso de librerías de C++. (Tapia Ayala & Manzano Yupa, 2013)

Permite enviar y recibir datos de la placa mediante la herramienta Monitor Serie. La placa se conecta al PC a través del cable USB, esta se instala automáticamente y posteriormente se debe definir el tipo de tarjeta con la que se va a trabajar a través del menú herramientas.

Figura 5. IDE o entorno de programación de Arduino

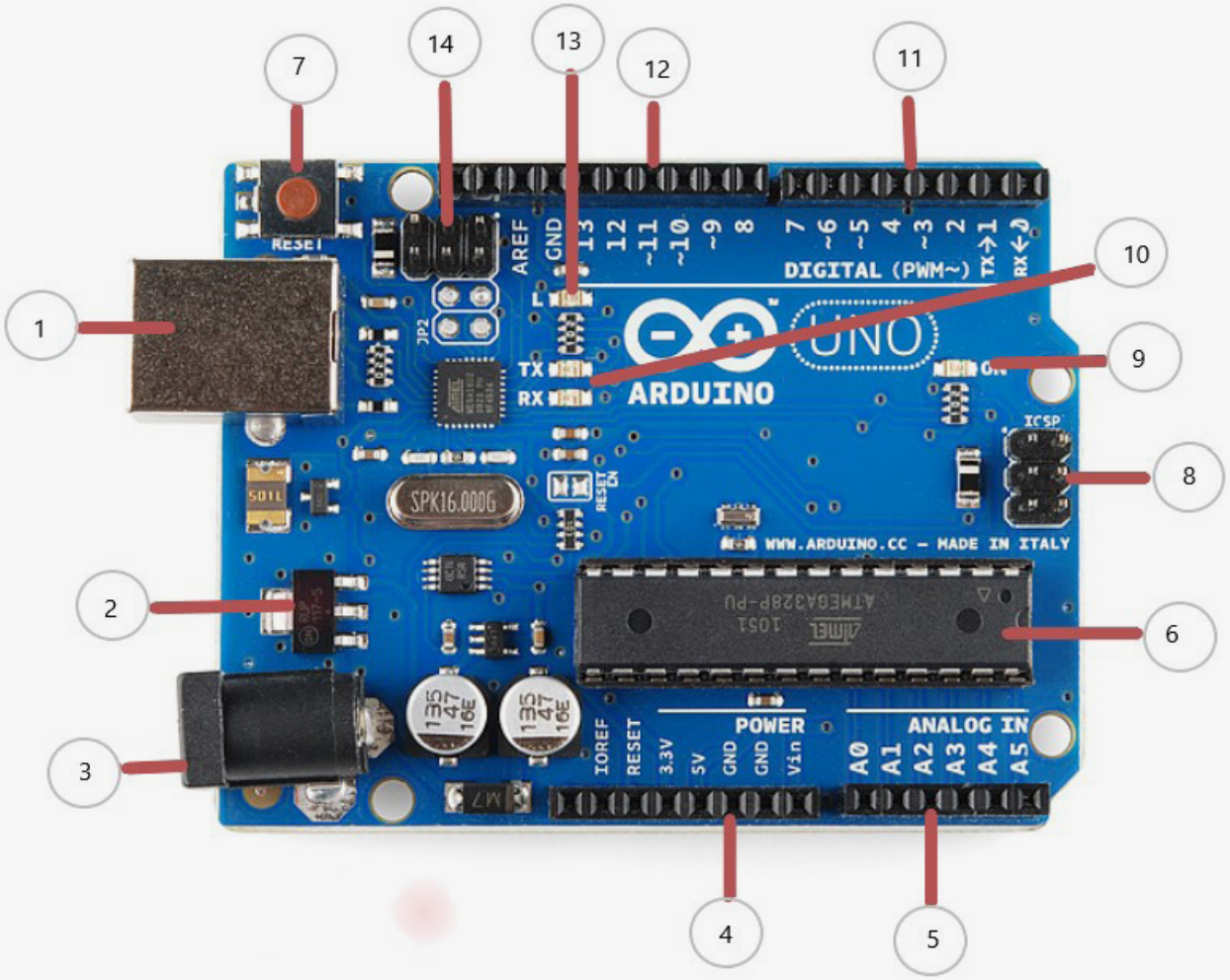


Tomado de: IDE Arduino

3.1.2. Placa de Arduino

Es una placa electrónica que se puede adquirir ensamblada o construirla.

Figura 6. Partes de la placa de Arduino Uno



Tomado de: (Esparza Silva, 2003)

- **Conector USB:** Proporciona la comunicación para la programación y la toma de datos, también provee una fuente de alimentación de 5VDC a la placa, la corriente que proporciona es baja por lo cual no sirve para alimentar motores de gran potencia.

- **Regulador de voltaje de 5V:** Se encarga de convertir el voltaje ingresado por el plug 3, en un voltaje de 5V regulado necesario para el funcionamiento de la placa y para alimentar circuitos externos.

- **Plug de conexión para fuente de alimentación externa:** Es el voltaje que se suministra que debe ser directo y estar entre 6V y 20V, generalmente se debe tener cuidado de que el terminal del centro del plug quede conectado a positivo, esto debido a que algunos adaptadores incorporan la opción de intercambiar la polaridad de los cables.

- **Puertos de conexiones:** Es constituido por 6 pines de conexión con las funciones de RESET que permite reseteas el microcontrolador al enviarle un cero lógico.

  Pin 3.3V y 5V proveen de una fuente de 3.3VDC y 5VDC respectivamente para conectar dispositivos externos.

  Dos pines GND que permiten la salida de cero voltios para dispositivos externos.

  Pin Vin está conectado con el positivo del plug 3 por lo que se usa para conectar la alimentación de la placa con una fuente externa entre 6 y 12VDC en lugar del plug 3 o la alimentación por el puerto USB.

- **Puertos de entradas analógicas:** Se conectan las salidas de los sensores análogos. Estos pines solo funcionan como entradas recibiendo voltajes entre 0 y 5 VDC.

- **Microcontrolador Atmega 328:** Implementado en los Arduino Uno en montaje superficial.

- **Botón reset:** Permite resetear el microcontrolador haciendo que reinicie el programa.

- **Pines de programación ICSP:** Usados para programar microcontroladores en protoboard o sobre circuitos impresos sin tener que retirarlos de su sitio.

- **LED ON:** Enciende cuando el Arduino esta encendido.

- **Leds de recepción y transmisión:** Se encienden cuando la tarjeta se comunica con la PC. El Tx indica la transmisión de datos y el Rx la recepción.

- **Puertos de conexiones de pines de entradas o salidas digitales:** La configuración como entrada o salida debe ser incluida en el programa. Cuando se usa el terminal serial es conveniente no utilizar los pines cero (Rx) y uno (Tx). Los pines 3, 5 y 6 están precedidos por el símbolo □, lo que indica que permiten su uso como salidas controladas por ancho de pulso PWM.

- **Puerto de conexiones 5 entradas o salidas adicionales:** Las salidas 9, 10 y 11 permiten control por ancho de pulso; la salida 13 es un poco diferente pues tiene conectada una resistencia en serie, lo que permite conectar un led directamente entre ella y tierra. Finalmente hay una salida a tierra GND y un pin AREF que permite ser empleado como referencia para las entradas análogas.

- **Led pin 13:** Indica el estado en que se encuentra la tarjeta.

- **Pines de programación ICSP:** Usados para programar microcontroladores en protoboard o sobre circuitos impresos sin tener que retirarlos de su sitio.

- **Chip de comunicación:** Permite la conversión de serial a USB.

(Tapia Ayala & Manzano Yupa, 2013)

3.1.3. Instrucciones

Setup()

La función setup () se invoca por única vez al inicio del programa. Se emplea para configurar todos los pines que se van a utilizar en el desarrollo del programa o puerto serie.

Loop()

Es un bucle o ciclo que ejecuta las instrucciones escritas dentro de esta función lo que permite que el programa responda a los eventos que se producen en la tarjeta.

pinMode (pin, mode)

Permite configurar el modo de trabajo cada Pin de la tarjeta, pudiendo ser de modo INPUT (entrada) u OUTPUT (salida), esta configuración se realiza dentro de la función de configuración setup ().

```
void setup() { //configuración de pin como salida
  pinMode(5,OUTPUT); //pinmode=configura el pin de arduino
                     //el número 5 indica el pin de la tarjeta
                     //OUTPUT(SALIDA)=estado del actuador conectado a este pin
  pinMode(6,OUTPUT);
  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);
} // fin de configuración de pin
```

digitalread (pin)

Realiza la lectura de un pin dando un resultado HIGH o LOW. El pin se puede especificar ya sea como una variable o una constante.

```
void loop() {
  Entrada=digitalRead(2);
```

En este caso hace que el Entrada tome el estado leído en el pin 2.

digitalwrite (pin, value)

Envía al pin configurado como OUTPUT (en setup) el valor HIGH o LOW. El pin se puede especificar ya sea como una variable o una constante.

```
int Entrada =0;

void setup() {
pinMode(2,INPUT);
pinMode(12,OUTPUT);

}

void loop() {
  Entrada=digitalRead(2);
  if(Entrada==1){
    digitalWrite(12,HIGH);
    delay(2000);
digitalWrite(12,LOW);
  //  delay(1000);


  }
```

Analogread (pin)

Lee el valor de un determinado pin definido como entrada analógica con una resolución de 10 bits. Esta instrucción sólo funciona en los pines (0-5). El rango de valor que puede leer oscila de 0 a 1023.

```
cont=analogRead(2)
```

Los pines analógicos no necesitan ser declarados como INPUT u OUTPUT ya que siempre están configurados como INPUT´s,

Analogwrite (pin, value)

Esta instrucción sirve para escribir un pseudo-valor analógico utilizando el procedimiento de modulación por ancho de pulso PWM a uno de los pines de la tarjeta marcados como pin PWM. El valor que se puede enviar a estos pines de salida analógica puede darse en forma variable o constante, siempre debe manejarse un margen de 0–255.

```
analogWrite(agua,HIGH);
```

Delay (ms)

Detiene la ejecución del programa la cantidad de tiempo en ms que se indica en la propia instrucción.

```
delay (500);// tiempo de retardo
```

Serial.begin (rate)

Abre el puerto serie y fija la velocidad en baudios para la transmisión de datos en serie. El valor típico de velocidad para comunicarse con el ordenador es 9600.

```
void setup() {
 serial.begin(9600); // abre el puerto serie

}
```

Se debe tener en cuenta que cuando se utiliza la comunicación serie los pines digitales 0 (RX) y 1 (TX) no pueden utilizarse al mismo tiempo.

Serial.println (data)

Imprime los datos en el puerto serie, seguido por un retorno de carro automático y salto de línea. Este comando toma la misma forma que Serial.println(), pero es más fácil para la lectura de los datos en el Monitor Serie del software. (Tapia Ayala & Manzano Yupa, 2013)

```
void setup() { //ingresar la configuración del código
  Serial.begin(9600); //configura el puerto serie a 9600bps
}

void loop() {
  Serial.println(analogRead (0)); //envía la lectura del valor analógico
  delay (2000); //tiempo de espera 2 segundos
}
```

Serial.avaible()

Obtiene un número entero con el número de bytes disponibles para leer o capturar desde el puerto serie. Devuelve un entero con el mismo número de bytes disponibles para leer desde el buffer serie o 0 si no hay ninguno. Si hay dato disponible Serial.avaible () será mayor que 0. El buffer serie puede almacenar como máximo 64 bytes. (Tapia Ayala & Manzano Yupa, 2013)

Serial.read ()

Lee o captura un byte desde el puerto serie.

3.2. EJEMPLOS

1. Diseñar un programa, que representa una pista de aterrizaje de aviones; la pista está dispuesta de una hilera de 10 Leds, los cuales se encuentran distribuidos 5 en el lado izquierdo y 5 al lado derecho, los leds se encenderán a medida que el avión avance de dos en dos (uno en cada lado), a medida que el avión avanza los leds deberán apagarse.

```
void loop() {  //comienza bucle principal del programa
  digitalWrite(5,HIGH); /*digitalWrite = envía el estado high(alto)
                          al pin 5 configurado como salida en el setup*/
  delay(tiempo);        //delay = tiempo de retardo
  digitalWrite(5,LOW);  /*digitalWrite = envía el estado low(bajo)
                          al pin 5 configurado como salida en el setup*/
  delay(tiempo);
  digitalWrite(6,HIGH);
  delay(tiempo);
  digitalWrite(6,LOW);
  delay(tiempo);
  digitalWrite(7,HIGH);
  delay(tiempo);
  digitalWrite(7,LOW);
  delay(tiempo);
  digitalWrite(8,HIGH);
  delay(tiempo);
  digitalWrite(8,LOW);
  delay(tiempo);
  digitalWrite(9,HIGH);
  delay(tiempo);
  digitalWrite(9,LOW);
  delay(tiempo);
} //fin de bucle
```

2. Diseñar un programa que permita variar la intensidad lumínica de un led por medio de un potenciómetro o una resistencia variable, la intensidad lumínica del led variara de acuerdo a la cantidad de resistencia que se varíe en el potenciómetro.

```
int pot=0;      //pin 0 corresponde al potenciometro
int led=9;      //pin 9 corresponde al led

void setup() { //configuración de pin
 pinMode(led,OUTPUT);  /*pinmode=configura el pin de arduino
                         led indica el pin de la tarjeta
                         OUTPUT(SALIDA)=estado del actuador conectado a este pin*/
}//fin de configuración de pin

void loop() {   //inicio bucle principal del programa
 for(pot=0;pot<=255;pot++) {    /*bucle FOR establece las condiciones de inicio,
                                  variación y limite de variación*/
 }
   analogWrite(led,pot);        //recibe señal analógica
   delay(20);                   //delay = tiempo de retardo
 }   //fin de bucle for
 for(pot=255;pot>=0;pot--) {    /*una vez alcanzada la máxima luminocidad se
                                  varia hasta apagar la luz.*/
   analogWrite(led,pot);
   delay(1000);
 }
}  //fin de bucle
```

3. Diseñar un programa que permita variar la velocidad de un motor por medio de un potenciómetro o una resistencia variable, la tarjeta recibirá la señal del potenciómetro y enviará la señal de variar la velocidad del motor, imprimir la velocidad del motor.

```
int motor = 3; /*Asigna la variable tipo entera motor al puerto 3 de la tarjeta arduino,
                 con lo que en adelante se llamará al puerto con el nombre de la variable*/
float pot=0;  //el potenciómetro (pot) toma el valor de la lectura del pin A0
void setup() { //configuración de los pines de la tarjeta
  pinMode (motor, OUTPUT); //Se configura el puerto 3 como salida
  Serial.begin (9600); //configura la valocidad de la lectura de datos en 9600 bps
}
void loop() {
  float cont=analogRead(pot)/4; //la variable cont toma el valor del potenciometro dividido entre 4
  for(cont=0; cont<=255; cont++){ /*la sentencia for registra el valor del contador y lo incrementa,
                                    conforme la variación del potenciometro*/
    analogWrite(motor,cont); //la velocidad del motor varía según el registro del contador
    Serial.println(cont); //permite visulaisar la variación de la velocidad del motor
    delay(200); // tiempo de espera de 200 ms
  }
  for(cont=255; cont>=0; cont--){ /*una ves registrado el valor máximo de la velocidad, se decrementa
                                    la velocidad.*/
    analogWrite(motor,cont);
    Serial.println(cont);
    delay(200);
  }
}
```

3.3 EJERCICIOS PROPUESTOS

- Diseñar un programa que permita activar 6 leds en forma ascendente desde el pin menos significativo hacia el más significativo y se mantengan encendido mientras el pulsador o interruptor se encuentran estado activo.

- Repetir el programa anterior utilizando la estructura o sentencia for.

- Diseñar un programa que por medio de un pulsador en pull up se imprima en la lcd SATISFACTORIO, si se presiona nuevamente aparezca EXCELENTE, si presiona por tercera vez no imprimirá nada y se reinicia el programa.

- Diseñar un programa para indicar los números de 0 hasta 9 en un display de 7 segmentos, esto debe ser programado por medio de la función for, con aumento de uno en un tiempo de 3000 milisegundos.

- Diseñar un programa de un sistema de luces las cuales se encenderán automáticamente hasta un cierto tiempo, después de ese tiempo las luces solo se encenderán con un pulsador, y se apagarán de la misma manera solo por medio del pulsador.

- Diseñar un programa que permita el control de un sistema de lavado de tanques, mediante un pulsador o interruptor se activa agua caliente y mediante otro pulsador o interruptor el agua fría además de contar con un botón de paro o reinicio de sistema, el tiempo y operación por cada uno es de 30 segundos, para efectos didácticos se dará inicio por medio dos leds como bombas.

- Diseñar un programa de un portón eléctrico que se activa por medio de dos pulsadores. Se activa una luz durante 5 segundos antes de que empiece el proceso, después de esto se activa el motor que abre; si detecta algún obstáculo se detiene hasta que éste sea retirado y continua, para cerrar se realiza por la inversión de giro del motor.

- Diseñar un programa para el control de un sistema de calefacción, por medio de un pulsador el sistema se enciende, haciendo uso de un sensor de temperatura se activa/desactiva el calefactor así, si la temperatura es de nivel bajo se enciende y al llegar a un determinado grado de temperatura se apagará automáticamente. Este sistema cuenta con un botón de paro.

- Diseñar un programa para el control de una banda clasificadora, por medio de un final de carrera se detecta el tamaño de la caja para activar las electroválvulas que las empujan por medio de un pistón para su clasificación. Registrar el número de cajas grandes y pequeñas en una LCD.

- Diseñar un programa que controla una mezcladora de líquidos. El sistema consta de 3 tolvas distribuidas así: tolva 1 almacena producto A, tolva 2 almacena producto B y tolva C o tolva de mezcla; las tolvas A y B tienen su propio sistema de alimentación 2 botones o pulsadores uno de encendido y otro que funciona como paro de emergencia, es decir al ser activado en cualquier momento, el proceso se detiene.

- Al encender el botón 1 el sistema arranca. Si se detecta que cualquiera de las tolvas A o B se encuentran con un nivel de llenado inferior al máximo (1.5 ltrs), se activa el sistema de alimentación de éstas hasta llegar al nivel requerido. Se abre la válvula de cada tolva permitiendo que el líquido de cada una llegue hasta la tolva de mezcla donde se activa el motor 30 segundos después de que se hayan vaciado las tolvas A y B, pasado este tiempo se apertura la válvula de la tolva para descargar en un sistema de riego.

BIBLIOGRAFÍA

Arduino. (2015). *Arduino.* Obtenido de https://www.arduino.cc/

Barra Zapata , O., & Barra Zapata, F. (2015). *Microcontroladores PIC con programación PBP.* España: RA-MA.

Bolton , W. (2013). MECATRÓNICA: Sistemas de control electrónico en la ingeniería mecánica y eléctrica (Quinta ed.). México: Alfaomega.

Esparza Silva, F. E. (2003). *Sistema de seguridad para el acceso de entradas y salidas utilizando microcontroladores.* PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR SEDE AMBATO, AMBATO. Obtenido de http://repositorio.pucesa.edu.ec

García Briejo, E. (2008). Compilador C CCS y simulador PROTEUS para microcontroladores PIC. México: Alfaomega.

Kernighan, B. W., & Ritchie, D. M. (1991). *El lenguaje de programación C.* México: Prentice Hall.

Mandado , E., Menéndez, L. M., Fernández, L., & López, E. (2007). *Microcontroladores PIC. Sistema Integrado para el autoaprendizaje.* España: MARCOBO.

Tapia Ayala, C. H., & Manzano Yupa, H. M. (2013). Evaluación de la plataforma Arduino e implementación de un sistema de control de posición horizontal. Universidad Politécnica Salesiana, Guayaquil.

Valdés Pérez, F., & Pallás Areny, R. (2007). *Microcontroladores Fundamentos y Aplicaciones con PIC.* España: MARCOBO.

GUÍA
de programación para microcontroladores

Al hablar de microcontroladores se hace referencia a un circuito integrado programable que contiene la estructura de una microcomputadora, es decir, que está conformado de CPU (Unidad Central de Proceso), memoria RAM, memoria ROM, memoria EEPROM (Memoria de lectura y escritura no volátil los datos no se pierden cuando el circuito es desconectado), puertos de Entrada/Salida, módulos periféricos: conversores analógico/digital (A/D), módulos PWM (control por ancho de pulso), módulos de comunicaciones seriales o en paralelo, entre otros. Estos dispositivos nacieron a finales de la década del 70, creados para dar solución a los caros y complejos sistemas basados en lógica discreta. Existe una gran cantidad de modelos de microcontroladores cuyas características y prestaciones varían de un modelo a otro. Los distintos modelos de microcontroladores se agrupan por familias; una familia puede estar formada por un conjunto de modelos cuyas características y prestaciones son bastante similares. Los microcontroladores están presentes en nuestro trabajo, casa y en nuestra vida en general. Se pueden encontrar controlando el funcionamiento de los computadores, teléfonos, hornos microondas, televisores y demás aparatos electrónicos controlados o automáticos.

Diana Aracely Guerrón Bolaños, Ingeniera en Mecatrónica por la Universidad Tecnológica Equinoccial. En el ámbito laboral, ha colaborado con empresas de desarrollo de software, principalmente de control de asistencia. Actualmente se desempeña como docente de la carrera de Electricidad en el Instituto Superior Tecnológico "Vicente Fierro", ha realizado dos publicaciones: Manual de seguridad para el laboratorio de electricidad y un artículo sobre la Construcción de un prototipo de dispositivo electrónico para personas invidentes.

INSTITUTOS
Superiores Técnicos y Tecnológicos

ISBN: 978-9942-8747-6-4