Manual de Programación Basic Stamp 2

Versión 1.1

ATENCION

El contenido de este manual no pretende ser la traducción literal al castellano del manual original en ingles “Basic Stamp 2 manual versión 1.9”, sin embargo se ha tratado de mantener lo más fiel posible el contenido de las instrucciones. Sé a agregado una breve introducción a los microcontroladores, nuevos ejemplos, graficas y algunas notas de experiencia del autor con este producto.

Este manual pretende ser una guía para personas que estén interesadas por el mundo de los microcontroladores y temas relacionados con automatizaciones industriales, estudiantes de electrónica e informática y cualquier persona entusiasta. Este manual no pretende ser una guía absoluta para el estudio de los microcontroladores Basic Stamp 2. El autor recomienda siempre el manual original en ingles como guía absoluta. El cual es suministrado por el fabricante y se puede descargar gratuitamente por el Internet.

El autor no asume responsabilidad alguna si usted utiliza estos conocimientos para actos maliciosos o dañinos para la humanidad o el medio ambiente. Este manual se ha traducido al castellano para fines educativos y para beneficio de la comunidad de habla hispana.

Este manual es el esfuerzo de varias horas de trabajo, surge por que nadie sé habia interesado en traducirlo al castellano. El mismo puede contener varios errores ortograficos y de sintaxis. Sugerencias, ejemplos y aportes serán bien acogidos por el autor para beneficio de futuros lectores en una próxima edición.

Basic Stamp 2 es un producto fabricado y patentado por PARALLAX.

Si usted es una persona que piensa que todo esta creado y que no puede aportar nada nuevo a la sociedad, entonces no lea este manual.

El Autor:

Diego M. Pulgar G.
dpulgar@comser.com.do

Contenido

1: Introducción a los microcontroladores

¿Que es un microcontrolador?

Es un circuito integrado programable que acepta un listado de instrucciones y contiene todos los componentes de un computador. Se utilizan para realizar determinadas tareas o para gobernar dispositivos, debido a su reducido tamaño, suele ir incorporado en el propio dispositivo que gobierna.

El microcontrolador es un dispositivo dedicado. En su memoria solo reside un programa destinado a gobernar una aplicación determinada, sus líneas de entradas y salidas (I/O) permiten la conexión de sensores y relay. Una vez programado y configurado el microcontrolador solamente sirve para gobernar la tarea asignada.

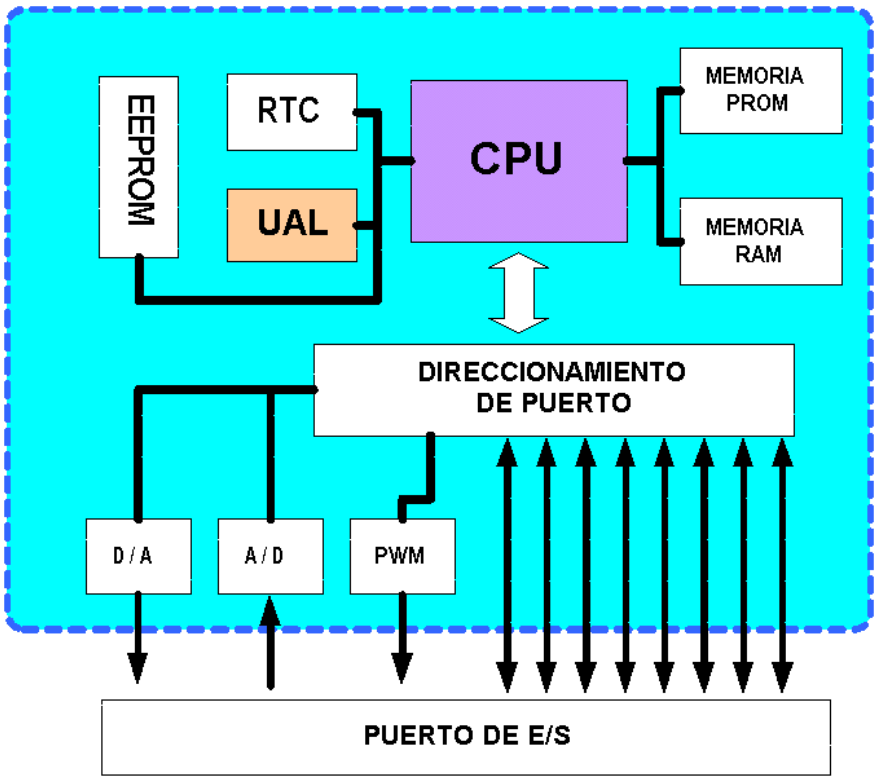


Figura 1.1 Diagrama en bloque de un Microcontrolador

Un microcontrolador dispone normalmente de los siguientes componentes:

- Procesador o UCP (Unidad Central de Proceso).
- Memoria RAM para Contener los datos.
- Memoria para el programa tipo ROM/PROM/EPROM/EEPROM & FLASH.
- Líneas de (entrada / salida) para comunicarse con el exterior.
- Diversos módulos para el control de periféricos (temporizadores, Puertos Serie y Paralelo, A/D y D/A, etc.).
- Generador de impulsos de reloj que sincronizan el funcionamiento de todo el sistema.

Evidentemente, el corazón del microcontrolador es un microprocesador, pero cabe recordar que el microcontrolador es para una aplicación concreta y no es universal como el microprocesador.

El microcontrolador es en definitiva un circuito integrado que incluye todos los componentes de un computador. Debido a su reducido tamaño es posible montar el controlador en el propio dispositivo al que gobierna. En este caso el controlador recibe el nombre de controlador empotrado (embedded controller).

¿Diferencia entre microprocesadores y microcontroladores?

El microprocesador es un circuito integrado que contiene la Unidad Central de Proceso (CPU), también llamado procesador, de un computador. El CPU está formado por la Unidad de Control, que interpreta las instrucciones, y el BUS de Datos, que los ejecuta.

Los pines de un microprocesador sacan al exterior las líneas de sus buses de direcciones, datos y control, para permitir conectarle con la Memoria y los Módulos de (ENTRADA / SALIDA) E/S y configurar un computador implementado por varios circuitos integrados.

Se dice que un microprocesador es un sistema abierto porque su configuración es variable de acuerdo con la aplicación a la que se destine.

El microcontrolador es un sistema cerrado. Todas las partes del computador están contenidas en su interior y sólo salen al exterior las líneas que gobiernan los periféricos. Usted podría pensar que las características de un sistema cerrado representan una desventaja con relación a los Microprocesadores, pero en la práctica cada fabricante de microcontroladores oferta un elevado número de modelos diferentes, desde los más sencillos hasta los más poderosos. Es difícil no encontrar uno que se adapte a nuestros requerimientos del momento.

Es posible seleccionar la capacidad de las memorias, el número de líneas de (ENTRADA / SALIDA) E/S, la cantidad y potencia de los elementos auxiliares, la velocidad de funcionamiento, etc. Por todo ello, un aspecto muy destacado del diseño es la selección del microcontrolador a utilizar.

Podemos concluir con que la diferencia fundamental entre un Microprocesador y un Microcontrolador: es que el Microprocesador es un sistema abierto con el que se puede contruirse un computador con las caracteristicas que se desee, acoplandole los modulos necesarios.
Un Microcontrolador es un sistema cerrado que contiene un computador completo y de presentaciones limitadas que no se pueden modificar.

1: Introducción a los microcontroladores

Sistemas numéricos

Realmente somos educados bajo el sistema numérico decimal. Un numero decimal como 5249 representa una cantidad igual a 5 millares, más 2 centenas, más 4 decenas, más 9 unidades. Los millares, centenas, decenas y unidades, son potencia de 10 implicadas por la posición de los coeficientes. Para ser más exactos, 5249 debe escribirse como:

$5x10^3 + 2x10^2 + 4x10^1 + 9x10^0$
5x1000 + 2x100 + 4x10 + 9x1
5000 + 200 + 40 + 9
5249

El sistema de número decimales se dice que es de base, o raíz 10 debido a que usa 10 símbolos y los coeficientes se multiplican por potencia de 10.

A parte del sistema decimal existen otros sistemas numéricos como son el sistema binario, el sistema octal y el sistema hexadecimal. Realmente los microcontroladores manejan el sistema binario; pero en la programación el más conveniente es el hexadecimal y para calculos matematicos el decimal. En la siguiente tabla vea la conversión equivalente entre el sistema decimal, hexadecimal y binario.

Dec	Hex	Bin
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

El sistema decimal esta compuesto por 10 símbolos (0-9), el sistema hexadecemial esta compuesto por 16 símbolos (0-9, A,B,C,D,E,F) y el sistema binario esta compuesto por dos símbolos (0-1).

Cualquier sistema numérico es infinito, mientras el sistema contenga mayor cantidad de símbolos su representación será más abreviada; como es el caso del sistema hexadecimal.

Para convertir de un sistema numérico a otro usted puede auxiliarse de una calculadora científica o de la calculadora incluida en el sistema operativo Windows.

Los Basic Stamp y la mayoría de los microcontroladores trabajan con el sistema binario, decimal y hexadecimal. No son necesario las conversiones de un sistema a otro. Pero sin embargo necesitan un formato de representación para que el PBASIC pueda reconocerlos.

Por ejemplo suponga que se tiene el siguiente valor decimal: Mil Ciento Diez (1110):

Este valor se puede interpretar:

- En binario (1110) : Decimal 14
- En hexadecimal (1110) : Decimal 4368

La misma representación puede interpretarse como 3 valores diferentes. El BS2 distingue las cantidades decimales de forma natural y las hexadecimales y binarias por símbolos a la izquierda de la cifra a representar.

Lógica Binaria o de 2 estados

La lógica binaria trata con variables que toman dos valores distantes y con operaciones que tienen significado lógico. Los dos valores que toman las variables pueden designarse con nombres diferentes (verdadero y falso, si y no, true y false, 0 y 1, etc.), pero para este propósito no es conveniente pensar en términos de BITS y asignarles los valores de 1 y 0. La lógica binaria se usa para describir, en forma matemática, la manipulación y el proceso de la información binaria. Existe una analogía directa entre las señales binarias, los elementos de circuito binario y digito binario.

Un numero binario de *n* dígitos, por ejemplo puede representarse por *n* elementos de números binarios, cada uno con una señal de salida equivalente a 0 o al 1. Los sistemas digitales representan y manipulan no sólo números binarios, sino también otro muchos elementos discretos de información.

Un BIT, por definición, es un digito binario. Cuando se usa junto con un código binario, es mejor considerarlo como si denotara una cantidad binaria igual a 0 o 1. Para representar un grupo de 2^n elementos distintos en un código binario, se requiere un mínimo de *n* BITS. Esto se debe a que es posible ordenar *n* BITS en 2^n formas distintas. Por ejemplo un grupo de 16 elementos puede representarse mediante un código de 4 BITS. $2^4 = 16$ elementos.

Durante todo el contenido de este material usted deberá estar familiarizado con los términos (1) lógico, (0) lógico, señal alta, señal baja, HIGH, LOW, 0 y 1. Cuando se dice que una señal es alta quiere decir que mide +5 Voltios con relación a tierra, cuando una señal es baja mide +0 Voltios con relación a tierra. Los BS2 trabajan con la lógica TTL, esta opera con +5 Voltios como fuente de alimentación. Las señales se fundamentan entre +0 Voltios y +5 Voltios.

Esto es aplicable tanto para las entradas como para las salidas, en la siguiente tabla se puede apreciar los diferentes términos para referirse a la lógica binaria.

Digito Binario	Nombres designados				
0	+0 V	S. Baja	F	LOW	0 Lógico
1	+5 V	S. Alta	V	HIGH	1 Lógico

Operaciones lógica básicas

Existen 3 operaciones lógicas llamadas: AND, OR y NOT.

1. **AND** esta función es verdadera cuando todas sus entradas son verdaderas. Y es falso cuando cualquiera de sus entrada son falsas. Se interpreta como la multiplicación binaria.
2. **OR** esta función es falsa cuando todas sus entradas son falsas. Y es verdadera cuando cualquiera de sus entrada sea verdadera. Se interpreta como la suma binaria.
3. **NOT** es la negación del resultado si es verdadero lo convierte en falso. Si es falso lo convierte en verdadero.

AND			OR			NOT	
x	y	**x.y**	x	y	**x+y**	x	**x'**
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

Estas son las 3 operaciones fundamentales en la lógica binaria, a partir de estas funciones se derivan otras más que son las combinaciones de las 3 funciones básicas.

El Microcontrolador Basic Stamp 2 (BS2)

El BASIC Stamp II es un pequeño computador que ejecuta programas en lenguaje PBASIC. El BS2-IC tiene 16 pines de (entrada / salida) I/O que pueden ser conectados directamente a dispositivos digitales o de niveles lógicos, tales como botones, diodos LEDs, altavoces, potenciómetros, y registros de desplazamiento. Además, con unos pocos componentes extras, estos pines de I/O pueden ser conectados a dispositivos tales como solenoides, relay, servomotores, motores de paso a paso, y otros dispositivos de alta corriente o tensión.

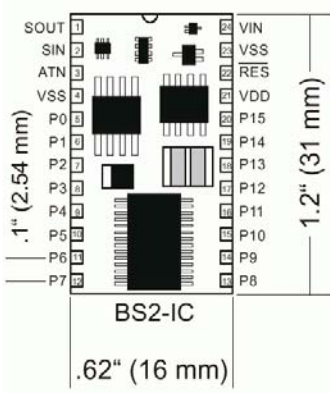


Figura 2.1: Diagrama esquematico BS2

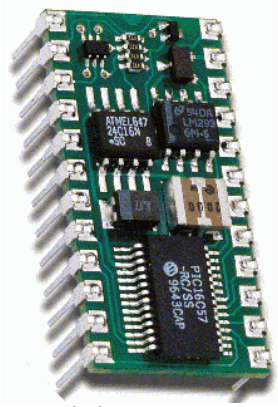


Figura 2.2: Basic Stamp 2

Equipos necesarios para trabajar con el BS2

- Un microcontrolador Basic Stamp 2, Ref. #BS2-IC
- Un cable serial RS-232 (no null modem)
- Un Pulsador Momentáneo (N.O.)
- Fuente de alimentación (+5 V - +15 V)
- Una computadora personal PC; S.O. Windows 95/98/NT4/2000
- Programa Editor PBASIC
- Una tablilla de experimentación BreadBoard

Opcionales:

- INEX-1000, Ref. #28135

El programa editor es gratuito y se puede descargar directamente de la pagina: http://www.parallax.com/

El INEX-1000 aunque no imprescindible para el funcionamiento del BS2, es muy importante para quienes van a desarrollar aplicaciones con microcontroladores BS2. El INEX-1000 contiene: un BreadBoard, fuente de alimentación, Pulsador de reset, 2 conectores DB9, pantalla de cristal liquido LCD, frecuencimetro, 4 display de 7 segmentos, 8 pulsadores momentaneos, 8 DIP switch, 16 diodos leds para el monitoreo de las salidas, 7 salidas amplificadas para controlar relays o motores de paso a paso. Entre otros componentes.

Las ventajas que ofrece este entrenador es que una vez insertado el BS2, no hay que trabajar con la polarización ni la interconexión del puerto serial. Reduciendo así posibles accidentes en malas interconexiones.

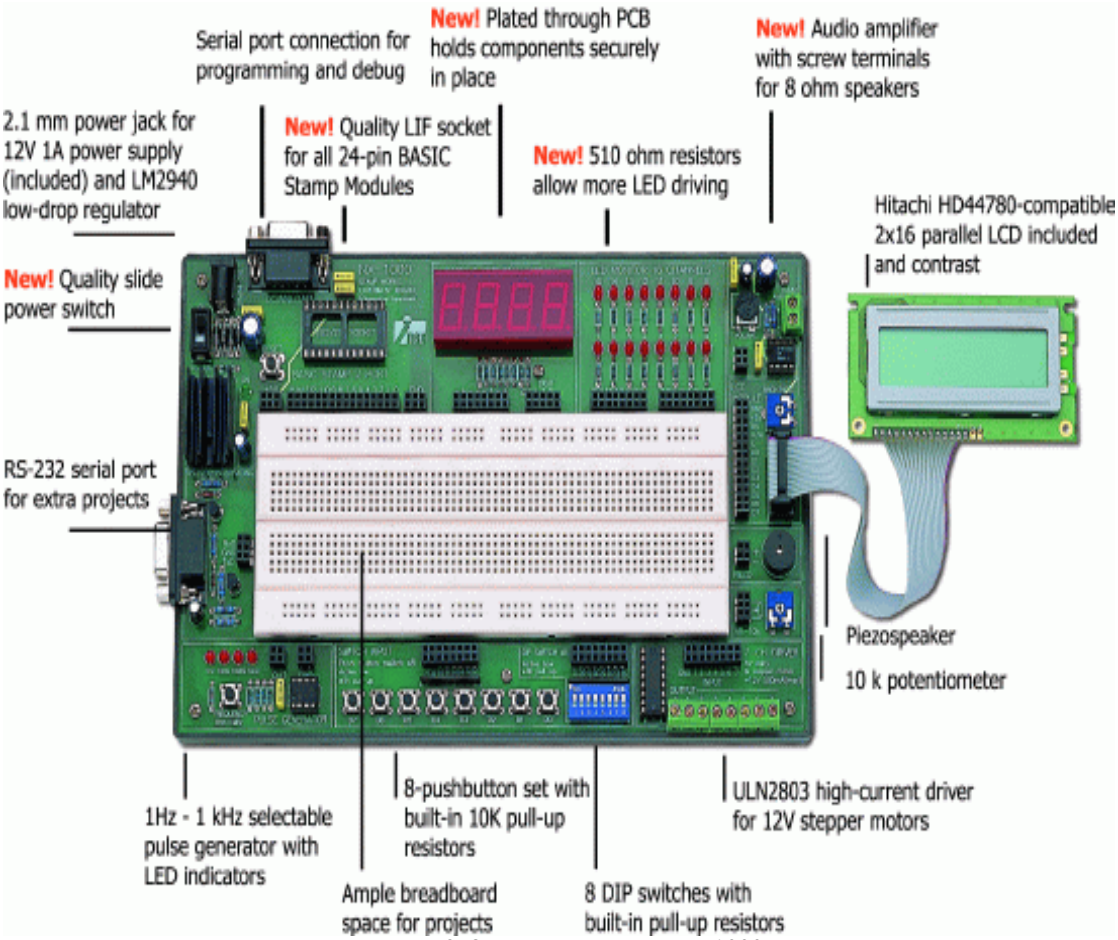


Figura 2.3: Entrenador INEX-1000

Formato de conversión numérica del BS2

El editor PBASIC utiliza símbolos para identificar los distintos sistemas numéricos. Los números hexadecimales se representan con el signo de moneda (**$**), los números binarios con el símbolo de porcentaje (**%**), los caracteres ASCII encerrados entre comillas (") y los números decimales de forma directa. Vea el siguiente ejemplo:

```
75          'Decimal
%01001      'Binario
$65         'Hexadecimal
"A"         'ASCII "
```

Las 3 instrucciones siguientes contienen el mismo significado:

```
DIRS = 14
DIRS = $E
DIRS = %1110
```

Funcionamiento Interno del BS2

El diseño físico consiste en un regulador de 5+ voltios, un oscilador de 20 MHz, una memoria EEPROM de 2K, un detector de bajo voltaje e chip intérprete PBASIC. Un programa compilado en PBASIC es almacenado en la EEPROM, desde donde el chip intérprete grabado en el microcontrolador lee y escribe las instrucciones.

Este chip intérprete ejecuta una instrucción cada vez, realizando la operación apropiada en los pines de I/O o en la estructura interna del chip intérprete. Debido a que el programa PBASIC es almacenado en una EEPROM, puede ser reprogramado una cantidad cercana a 10 millones de veces.

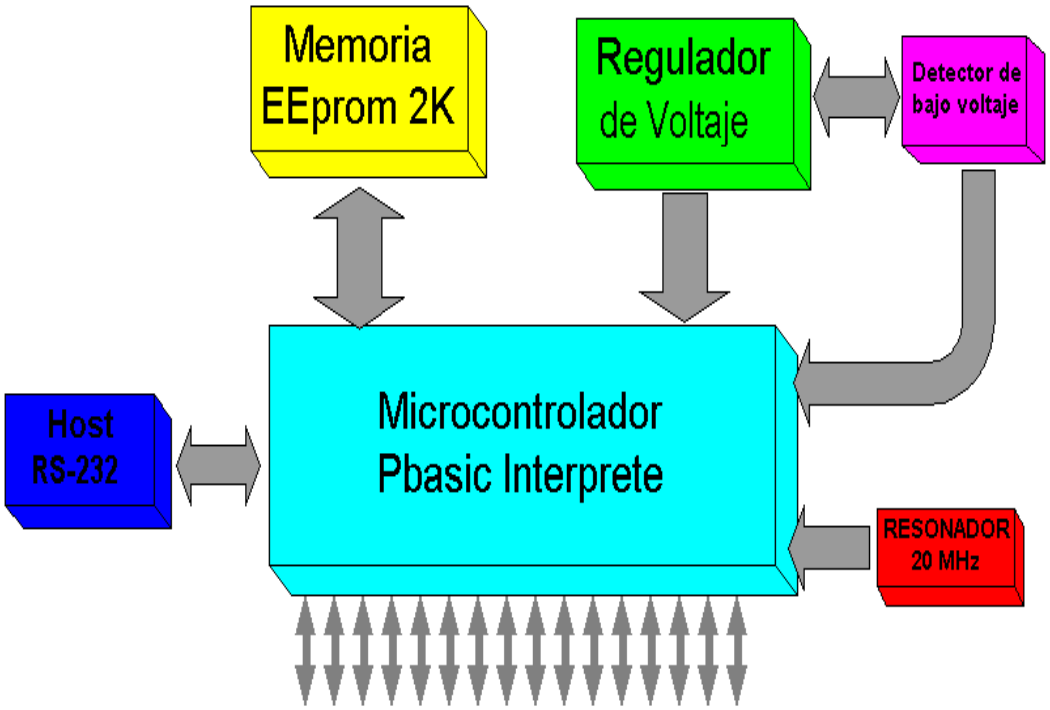


Figura 2.4: Diagrama en Bloque del BS2

La programación del BS2 se realiza directamente desde un computador personal PC, descargando los programas desde el software editor proporcionado gratuitamente por Parallax, Inc.

2: Introducción al BASIC Stamp 2

El Basic Stamp II es capaz de almacenar entre 500 y 600 instrucciones de alto nivel (PBASIC) y ejecuta un promedio de 4000 instrucciones / segundo.

Para programar el BS2-IC, simplemente conéctele un cable serial preparado entre el BS2 y un PC, y ejecute el software editor para crear y descargar su programa, a través del cable serial.

Variedad según sus requerimientos

Los Basic Stamps están disponibles en diversos tamaños y velocidades, todas las versiones poseen el mismo diseño lógico, que consiste en un regulador de voltaje, osciladores, EEPROM, y chip PBASIC interprete. El programa en PBASIC es almacenado en la memoria EEPROM, el cual es leído por el chip interprete. Este chip interprete "extrae" las instrucciones una a la vez y realiza la operación adecuada en los pines I/O o en las estructuras internas dentro del chip interprete. Como el programa PBASIC es almacenado en la memoria EEPROM, Las Basic Stamps pueden ser programadas y reprogramadas millones de veces, sin necesidad de borrar la memoria. Para programar una BASIC Stamp, Ud. sólo debe conectarla a un PC o compatible y hacer correr el software editor para editar y descargar sus programas.

Ventajas del BS2 con otros Microcontroladores

La gran ventaja de los BS2 respecto a otros microcontroladores es sin duda que incorporan un chip interprete de PBASIC, permitiendo ahorrar muchísimo tiempo en el desarrollo de aplicaciones dada su sencillez. El PBASIC es un lenguaje de programación basado en un BASIC estructurado orientado a entrada y salida de señales. La utilización de sencillas instrucciones de alto nivel, permite programar los Basic Stamps para controlar cualquier aplicación llevada a cabo por un microcontrolador.

Las instrucciones de PBASIC permiten controlar las líneas de (entrada / salida), realizar temporizaciones, realizar trasmisiones serie asincrónica, utilizar el protocolo SPI, programar pantallas LCD, capturar señales analógicas, emitir sonidos, etc. y todo ello en un sencillo entorno de programación que facilita la creación de estructuras condicionales y repetitivas con instrucciones como IF...THEN o FOR...NEXT y la creación de etiquetas de referencia.

Algunas aplicaciones de los BS2

La única limitante de los Microcontroladores es su imaginación. La facilidad de un puerto abierto de (entrada / salida), la capacidad de evaluación de señales para luego decidir una acción y poder controlar dispositivos externos. Hacen que el microcontrolador sea el cerebro de los equipos.

Estos son algunos ejemplos de áreas de aplicaciones:

- Electrónica Industrial (Automatizaciones)
- Comunicaciones e interfase con otros equipos (RS-232)
- Interfase con otros Microcontroladores
- Equipos de Mediciones
- Equipos de Diagnósticos
- Equipos de Adquisición de Datos
- Robótica (Servo mecanismos)
- Proyectos musicales
- Proyectos de Física
- Proyectos donde se requiera automatizar procesos artísticos
- Programación de otros microcontroladores
- Interfase con otros dispositivos de lógica TTL:
    - Teclado
    - Pantallas LCD
    - Protocolo de comunicación X-10
    - Sensores
    - Memorias
    - Real Time Clock (RTC)
    - A/D, D/A, Potenciómetros Digitales

Hardware del BS2

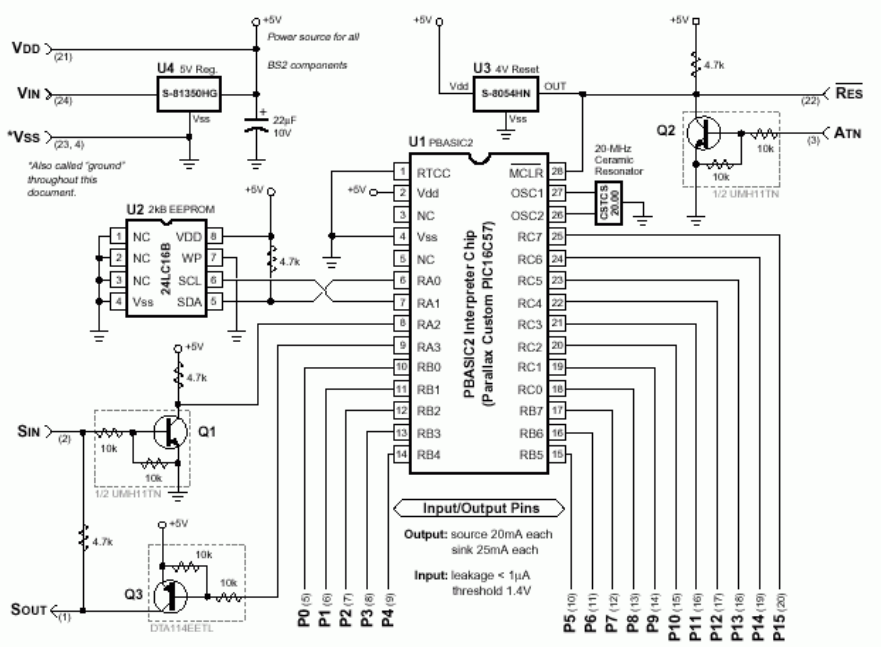


Figura 3.1: Diagrama electrico del Basic Stamp 2

El chip intérprete del Basic Stamp II (U1)

El cerebro del BS2 lo constituye un microcontrolador PIC16C57, de la familia de Microchip. U1 esta programado permanentemente de fabrica con un conjunto de instrucciones predefinidas del lenguaje PBASIC. Cuando usted programa el BS2, usted le está diciendo a U1 que salve las instrucciones compiladas, llamadas fichas de instrucciones hexadecimales, en la memoria EEPROM (U2). Cuando su programa se ejecuta, U1 extrae las fichas de instrucciones hexadecimales de la memoria (U2), los interpreta como instrucciones PBASIC, y ejecuta las instrucciones equivalentes.

U1 ejecuta su programa interno a una velocidad de 5 millones de instrucciones por segundo. Algunas instrucciones internas entran en una sola instrucción PBASIC2, así que PBASIC2 ejecuta más lentamente aproximadamente 3000 a 4000 instrucciones por segundo.

El PIC16C57 tiene 20 pines en total, 16 están destinados a entrada / salida (I/O); 4 estan destinados a la comunicación serial RS-232. En el circuito BS2 16 contactos están disponibles para uso general por sus programas. Dos de las otros se pueden también utilizar para la comunicación serial asincrónica. Los dos restantes se utilizan solamente para interconectar con el EEPROM y no se pueden utilizar.

Los contactos de uso general de I/O, (P0-P15), se pueden interconectar con toda la lógica de +5 voltios moderna, de TTL (lógica del transistor-transistor) con CMOS (semiconductor de óxido metálico

complementario). Las características son muy similares a las de los dispositivos de la serie lógica 74HCTxxx.

La dirección de entrada y salida de un contacto dado está enteramente bajo el control de su programa. Cuando un contacto es declarado como una entrada de información, tiene muy poco efecto en los circuitos conectados con él, con menos de 1 microamperio (uA) de consumo interno.

Hay dos propósitos para poner un pin en modo de entrada de información: (1) leer en modo pasivo el estado (1 o 0) de un circuito externo, o (2) para desconectar las salidas que manejan el pin. Para que el consumo de corriente sea él más bajo posible, las entradas de información deben siempre estar cerca de +5 voltios o cercano a la tierra. Los pins no utilizados en sus proyectos no se deben dejar libres en modo de entrada.

Los pins no usados deben ser declarados como salida aunque no estén conectados; esto es para evitar que las entradas esten interpretando el ruido externo como señales logicas.

Cuando un pin esta en modo de salida, internamente está conectado a la tierra o +5 voltios a través de un interruptor muy eficiente del circuito CMOS. Si se carga ligeramente (< 1mA), el voltaje de la salida estará dentro de algunos milivoltios cercanos de la fuente de alimentación (tierra para 0; +5V para 1). Cada pin puede manejar unos 25 mA. Pero Cada puerto de 8 pins no debe exceder de los 50 mA, con el regulador externo y 40 mA con el regulador interno; los pins de P0 a P7 conforman un puerto de 8 BITS y los pins de P8 a P15 el otro.

2048-byte de memoria borrable eléctricamente (U2)

U1 se programa permanentemente en la fábrica y no puede ser reprogramada, así que sus programas PBASIC2 se deben grabar en otra parte. Ése es el propósito de U2, una memoria EEPROM modelo 24LC16B eléctricamente borrable; la EEPROM es un buen medio para el almacenaje del programa porque conserva datos permanentemente aun sin energía y se puede reprogramar fácilmente.

Las EEPROMs tiene dos limitaciones: (1) toman un tiempo relativamente largo para programarlas (tanto como varios milisegundos), y (2) el límite de reprogramciones es de (aproximadamente 10 millones). El propósito primario del BS2 es almacenar un programa, ni unos ni otros de éstos son normalmente un problema. ¡Tomaría muchos cursos en la vida para escribir y para descargar 10 millones de programas PBASIC2! .

Circuito de Reset (U3)

Cuando usted enciende al BS2, toma una fracción de segundo a la fuente el voltaje estabilizarse y alcanzar el voltaje de operación unos 5+ voltios. Durante esta operación el circuito de Reset entra en acción.

La finalidad es detectar el voltaje de operación si es menor de 4.5+ el circuito de Reset mantendrá el Microcontrolador desconectado, cuando alcance un voltaje de unos 5+ voltios el circuito de Reset espera unos 30 mili-segundos para conectar al BS2.

Esta previsión evita posibles fallas del procesador y de la memoria (U1 y U2) que pueden incurrir en equivocaciones o bloqueos involuntario. El circuito de Reset también es conectado externamente para reiniciar al microcontrolador.

Fuente de alimentación (U4)

El BS2 tiene dos formas de polarizarlo la primera consiste a través de un voltaje de alimentación no regulado el cual puede variar de (5.5+ a 15+ Voltios). Este es un regulador de superficie S-81350HG, este puede proveer unos 50 mA. La segunda consiste polarizándolo directamente a través de VDD. Particularmente yo prefiero la segunda, pero se debe tener en cuenta que este voltaje no debe exceder los 5 Voltios. Y se puede realizar a través de un regulador externo como el LM7805.

Host RS-232 (Q1, Q2, y Q3)

Unas de las características más notables del BS2 es su capacidad para comunicarse con otras computadoras a través del puerto serial RS-232, esto se realiza de una manera natural. El puerto de interfase host RS232 tiene dos funciones básicas la primera es para reprogramar al BS2, y la segunda para comunicarse externamente con otros dispositivos compatibles de comunicación asincrónica de formato RS-232 estándar.

Pero el puerto RS-232 opera con un voltaje de (+12 V, para indicar un 1 lógico y –12 V, para indicar un 0 lógico). Mientras que el BS2 opera con (+5 V, para indicar 1 lógico y 0 V, para indicar un 0 lógico). El circuito de interfase se encarga entonces de las conversiones de voltajes necesarias para su correcta operación.

Conexión entre la PC y el BS2

La siguiente figura muestra un conector DB9, el cual utiliza 6 pines de los cuales 4 van destinado al BS2 y dos conectados internamente.

Preparando este cable usted puede empezar a programar los BS2. En caso de que usted utilice un cable serial, debe realizar la conexión de los pines 6 y 7 del cable. Si utiliza cualquiera de los entrenadores de Parallax. Esto no será necesario.

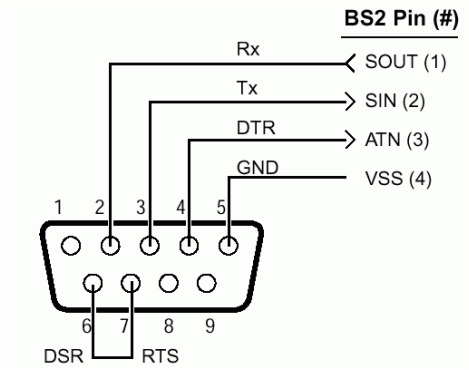


Figura 3.2: Conexión del conector DB9

Preparando un conector DB9, bajo esta configuración se puede empezar a programar al BS2. Con el programa Basic Stamp editor.

Descripción de los pines del BS2

Pin	Nombre	Descripción
1	SOUT	Serial Out: Conectar al puerto serial RX (DB9 pin 2)
2	SIN	Serial In: Conectar al puerto serial TX (DB9 pin 3)
3	ATN	Atención: Conectar al puerto serial DTR (DB9 pin 4)
4	GND	Tierra entre el puerto serial y el BS2
5-20	P0-P15	Puerto de propósitos generales, cada uno puede entregar 25 mA, sin embargo, el total de la corriente no puede exceder los 75 mA utilizando el regulador interno y 100 mA utilizando +5V externo
21	VDD	Voltaje regulado a +5 VDC
22	RES	Reset, Basta con aterrizar y el BS2 reinicialaza
23	GND	Tierra del BS2
24	PWR	Voltaje no regulado entre +5.5 a +15 VDC, si VDD es utilizado VIN no puede ser utilizado

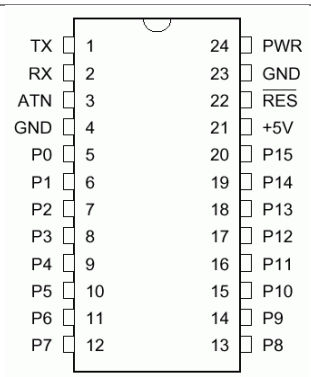


Figura 3.3: Ubicación de cada PIN

Conexión típica para su funcionamiento

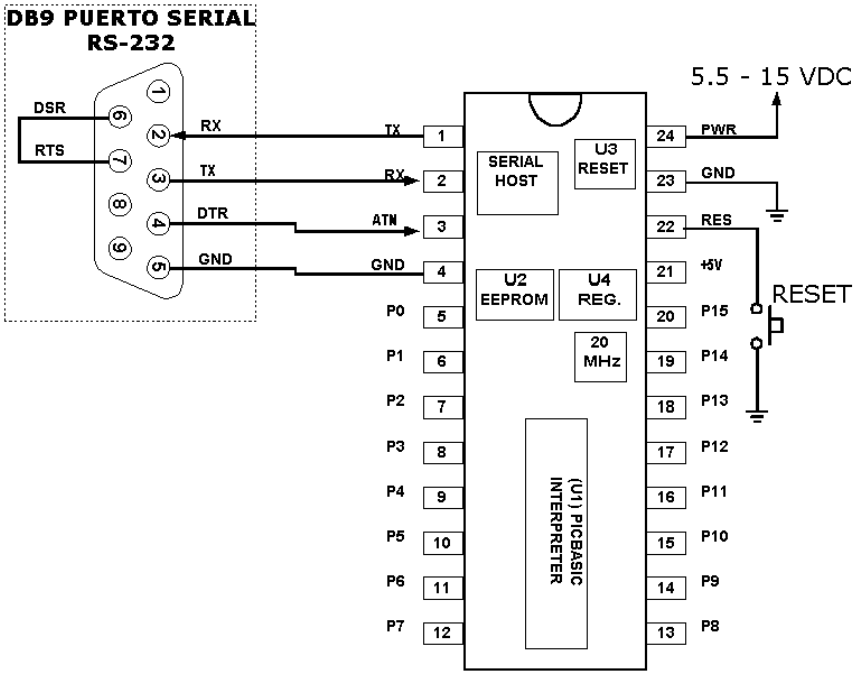


Figura 3.4: Conexión tipica para su funcionamiento

Lenguaje de programación PBASIC

El lenguaje de programación PBASIC fue creado específicamente para programar a los BS2, es pariente cercano del lenguaje de programación BASIC, la ventaja que ofrece el PBASIC con otros lenguajes es su facilidad de aprendizaje. Este manual explica en detalle cada comando con uno a varios ejemplos.

PBASIC Editor

El PBASIC Editor es el programa donde escribimos el conjunto de instrucciones para el Basic Stamp. Es similar en apariencia a cualquier editor de texto del sistema operativo WINDOWS. El editor contiene una serie de herramientas como son identificador del Basic Stamp, Corrector ortográfico de sintaxis, Mapa de memoria y Ventana del depurador.

El editor tiene la capacidad para abrir 16 ventanas simultáneamente. La capacidad de cortar, copiar y pegar se mantiene innata. Su entorno es muy sencillo y usted se familiarizara muy pronto.

Los comandos más importantes son:

F1	Muestra la ayuda en pantalla
Ctrl-O	Abre un archivo
Ctrl-S	Salva un archivo
Ctrl-P	Imprime el archivo actual
F9 o Ctrl-R	Descarga el programa en el BS2
F7 o Ctrl-T	Corrector de Sintaxis
F8 o Ctrl-M	Muestra el mapa de memoria
F6 o Ctrl-I	Muestra él numero de versión de PBASIC
ESC	Cierra la ventana actual

Estos son algunos de los comandos más importantes, aunque usted no esta obligado a memorizarlos, es conveniente recordar a [Ctrl-R]. El cual descarga el programa al Basic Stamp.

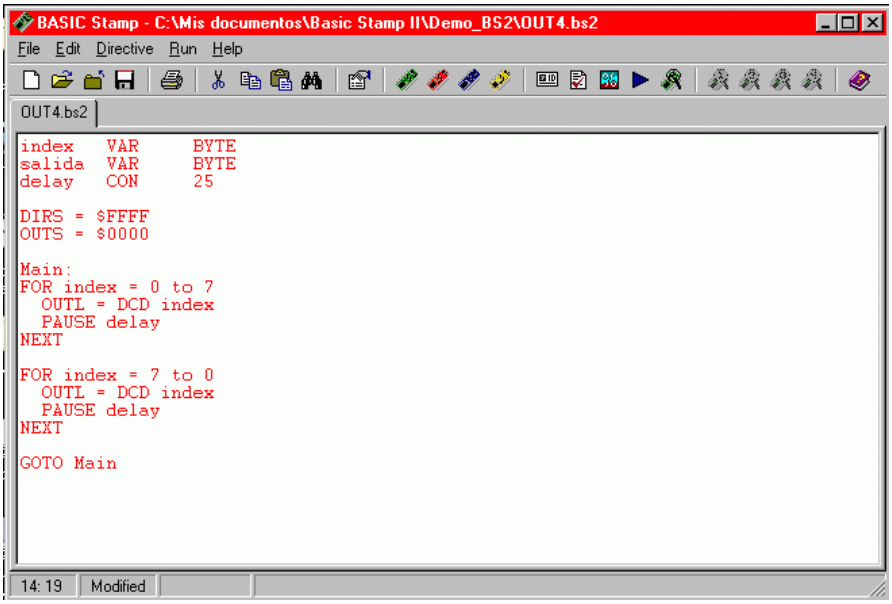


Figura 4.1 Pantalla de Editor de Pbasic

El editor contiene un grupo de iconos los cuales usted memorizara más efectivamente. Con solo apuntar con el Mouse el comando será ejecutado. Además de un menú interactivo.

Procedimiento para descargar el programa al BS2:

1. Con el BS2 previamente energizado y conectado como lo indica la figura 3.4, cargue el editor PBASIC.

2. Cuando el editor este listo presione [Ctrl-I], si todo esta bien conectado el editor le dará un mensaje de "**Found BS2-IC (firmware v1.0)**". Esto indica que va por buen camino.

3. Usted puede digitar su programa o cargar uno previamente del disco.

4. Para asegurar que el código digitado este bien, presione el corrector de sintaxis [Ctrl-T], si existe algún problema lo indicara con un mensaje de error. Si todo marcha bien le indicara un mensaje de "**Tokenize Successful**"

5. Ahora estará listo para descargar el programa en el BS2, presione [Ctrl-R], y el programa se descargara permanentemente en la EEprom del BS2. En caso de que usted no revise con el corrector de sintaxis, antes del programa descargarse en el BS2, este lo realiza por su cuenta.

6. Apague el Basic Stamp 2, Retire el cable serial del BS2.

7. Encienda el Basic Stamp 2 y la aplicación permanecerán hasta que usted le modifique nuevamente reprogramandolo por el puerto serial.

¿Que es un programa?
Un programa es un conjunto de instrucciones definidas, las cuales contienen ciertos parámetros y ejecutan una función determinada. Podemos decir que un programa de computación es comparado a una receta de cocina. La cual contiene un procedimiento, los ingredientes son cada una de las funciones y el modo de elaboración es como se combinan estas funciones para ejecutar uno o varios procedimientos determinados.

Conjunto de instrucciones del Pbasic:
El programa Pbasic para el Basic Stamp 2 consiste en un conjunto de 36 comandos orientados a entrada y salida de señales y evaluación de variables para luego tomar una desición, ademas de un conjunto de funciones matematicas basicas.

Literalmente le damos ordenes al BS2:

- Ponte en alerta
- Envia un pulso por el Pin?
- Mide el tiempo de esa señal por el Pin?
- Envia una frecuencia por el Pin?
- Cuenta el tren de pulso por el Pin?
- Envia un estado logico de +5 V, por el pin?
- Revisa el puerto?
- Descansa 2 minutos
- Espera 100 milisegundos
- Envia un dato por el puerto serial

El método de programación es lineal, es decir se ejecuta un comando a la vez, por lo general se programa de una forma en que se repita las instrucciones en un ciclo cerrado.

Los programas de Pbasic contienen: variables de memoria, constantes, direccionamiento de puertos, instrucciones y sub-rutinas.

4: Modo de programación del BASIC Stamp 2

En este manual usted aprenderá en detalles la utilización de cada comando así como varios ejemplos que complementan la teoría.

Estilo de programación

En la programación cada cual adopta un estilo único, dados los conocimientos básicos de cada instrucción, se plantea el problema el cual debemos solucionar con las funciones disponibles. La combinación de funciones es realmente ilimitada. El mismo problema se puede solucionar de diversas formas, cada cual lo hará con su criterio lógico aprendido.

¿Cuál es el código más eficiente?
Aunque un mismo problema tiene varias soluciones el código más eficiente será el que se realice con las menos instrucciones posibles.

Usted al inicio de crear un código, no debe de preocuparse de la eficiencia. Debe enfocarse en como resolvera el problema a partir de las funciones disponibles. Luego resuelto el problema satisfactoriamente; entonces procede a optimizar el código. De seguro siempre existe un mejor metodo que el anterior.

Código Modelo

Aunque cada quien adopta un estilo propio, es conveniente seguir un patron de ordenamiento, el sigiente ejemplo nos muestra un código modelo:

```
' -----[ Titulo ]-------------------------------------------------------
' Archivo.......... Write 01.bs2
' Proposito........ Aprender el uso del comando WRITE
' Autor............ Diego Pulgar
' E-mail........... dpulgar@comser.com.do
' WEB.............. www.dpgelectric.com
' Versión.......... 1.2
' Fecha de Inicio.. 12/ENE/2002
' Fecha de Final... 02/JUN/2002
' Cliente.......... Caribean Tecnologies

' -----[ Descripción del Programa ]---------------------------------------


' -----[ I/O Definiciones ]-----------------------------------------------


' -----[ Constantes ]----------------------------------------------------
'

' -----[ Variables ]-----------------------------------------------------


' -----[ EEPROM Data ]---------------------------------------------------


' -----[ Inicializaciones ]----------------------------------------------


' -----[ Main Code ]-----------------------------------------------------
```

```
' -----[ Sub-rutinas ]-------------------------------------------------------
```

La idea de este estilo es documentar todo el contenido y distinguir todos los procedimientos. Por lo general en un código PBASIC se debe llevar el siguiente orden:

1. Definir las constantes
2. Definir las variables de Entrada y Salida
3. Definir las variables de Programa
4. inicializar el puerto
5. Direccionar las entradas y salidas
6. Iniciar los circuitos perifericos, si existen
7. Rutina principal (Main)
8. Rutinas Secundarias
9. Sub-Rutinas (Rutinas que se repiten)

Cuanto más explicito sea escribiendo su código menos tiempo perdera cuando tenga que modificarlo. Los comentarios en el Pbasic no ocupan espacio en la memoria del BS2, simplemente el editor ignora los comentarios a la hora de descargar el codigo objeto al BS2.

5: Organización de memoria del BASIC Stamp 2

Organización de memoria del BS2

El BS2 tiene dos tipos de memoria; RAM para las variables de su programa, y EEPROM para almacenar los programas en sí. La memoria EEPROM puede ser utilizada para almacenar datos de la misma forma que lo hace una computadora personal PC.

Una importante diferencia entre la memoria RAM y EEPROM:

- **RAM** pierde el contenido cuando el BS2 no tiene energía, cuando retorna la energía o cuando se reinicializa el BS2 el contenido completo de RAM se inicializa con 0.

- **EEPROM** retiene el contenido sin energía o con energía, mientras no se sobre escriba con otro programa o con la sentencia WRITE.

Memoria RAM del BS2

El BS2 tiene 32 BYTES de memoria RAM, 6 BYTES están reservados para los registros de entradas, salidas y direccionamiento del puerto para el control de (entradas / salidas) I/O. Los 26 BYTES restantes están destinados a variables de uso general.

La siguiente tabla muestra los nombres de los registros de entradas, salidas y direccionamiento del puerto del BS2.

Jerarquías del Puerto P0-P15 (Registros: Dirs, Ins & Out)

DIRS															
DIRH								**DIRL**							
DIRD				DIRC				DIRB				DIRA			
DIR15	DIR14	DIR13	DIR12	DIR11	DIR10	DIR9	DIR8	DIR7	DIR6	DIR5	DIR4	DIR3	DIR2	DIR1	DIR0
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

OUTS															
OUTH								**OUTL**							
OUTD				OUTC				OUTB				OUTA			
OUT15	OUT14	OUT13	OUT12	OUT11	OUT10	OUT9	OUT8	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1	OUT0
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

INS															
INH								**INL**							
IND				INC				INB				INA			
IN15	IN14	IN13	IN12	IN11	IN10	IN9	IN8	IN7	IN6	IN5	IN4	IN3	IN2	IN1	IN0

P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

Como se puede apreciar los registros de direccionamiento **DIRS**, de entrada **INS** y de salida **OUTS**, contienen una jerarquía, el registro OUTS contiene el puerto completo del BS2, abarcado desde P0-P15. Pero la flexibilidad jerárquica consiste en que se pueden dividir y sub-dividir los registros según la necesidad. Si por ejemplo queremos manejar un puerto de 4 BITS, elegimos cualquiera de los puertos tipo NIB. También podemos referirnos de forma individual a cada pin. Si queremos controlar el Pin 7, elegimos OUT7.

Direccionamiento del puerto P0-P15

Desde el principio usted a leído en este manual los términos (entrada / salida), E/S, Input/Output y I/O. Los microcontroladores por lo general contienen un puerto direccionable. Esto quiere decir que usted elige que pines serán salidas y cuales serán entradas.

Las personas que han trabajado con PLC (controles lógicos programables), saben que los PLC tienen definidas sus entradas y sus salidas. Por lo general más entradas que salidas.

En el caso de los microcontroladores esto es muy flexible hay aplicaciones donde todo podría ser salidas, otras donde solo serian entradas, y otras donde exista la combinación de ambas. La ultima es la más común. Pero usted se preguntara como un pin puede ser una entrada y luego se puede ser una salida.

Esto es posible gracias a un circuito que aísla la entrada, el direccionamiento es un interruptor lógico que acciona el pin para fijarlo en modo de entrada o modo de salida. Cuando usted enciende el BS2 todo el puerto se convierte en entrada automáticamente, hasta que usted no le indica que quiere cambiar el estado a modo de salida el puerto permanece como entrada.

Las entradas de los BS2 tienen una impedancia bastante elevada y el consumo de corriente que les toman a los dispositivos externos conectados a ellos es de menos 1 uA, mientras que el voltaje máximo en una entrada no puede exceder los +5 Voltios. Por lo general usted puede polarizar las entradas directamente desde +5 Voltios o directamente a tierra.

En modo de salida si se deben tomar todas las precauciones, cada salida es capaz se suministrar hasta 25 mA. Pero que sucede si un pin determinado que funciona en momento como entrada y esta puesto a tierra y luego este pin se direcciona como salida. El resultado es un corto circuito en este pin y un daño irreparable en el microcontrolador.

Por lo general nunca se debe manejar directamente hacia las entradas voltajes directos, se debe hacer a través de resistencias en serie por

lo orden de 10 kOHM, esto evitara posibles daños. En el caso de que una salida se convierta en entrada no tiene efecto alguno en los dispositivos que maneja. Simplemente dejara de fluir la corriente eléctrica.

En el siguiente figura se puede visualizar un diagrama de cómo funciona internamente el direccionamiento de un pin. En la figura 5.1 A, el pin esta en modo de entrada y en la figura 5.1 B, el mismo pin esta en modo de salida.

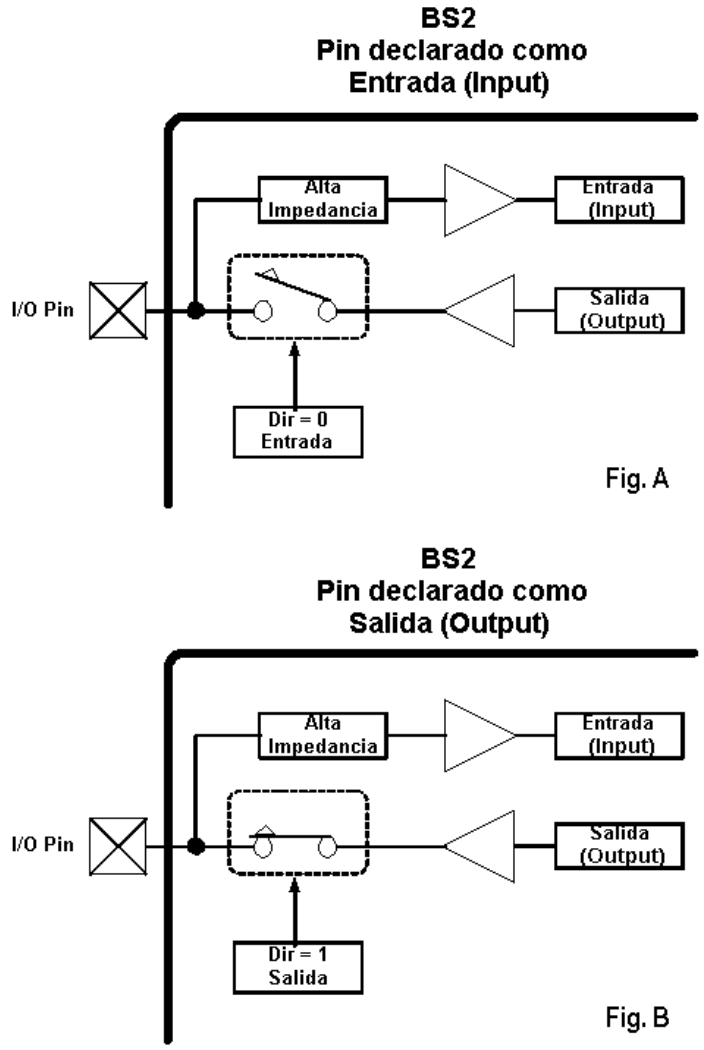


Figura 5.1 Diagrama en bloque del funcionamiento del direccionamiento

En la figura 5.1 B, se puede apreciar que cuando el pin esta en modo de salida, el circuito de entrada (input), se mantiene leyendo el estado de la salida, esto no causa mayor efecto recuerden que la entrada tiene una alta impedancia. El registro de entrada INS, es capaz en todo momento de leer el estado de cualquiera de sus pines desde P0-P15, sin

importar que estén declarados como salida. Este registro puede leer la situación de cada pin

Una vez se direcciona un pin o un puerto este permanecerá indefinidamente en ese estado o hasta que se le indique otra dirección.

Para direccionar un pin como salida bastara con:

- **DIR0 = 1** Direcciona el Pin 0 como salida

Para direccionar un pin como entrada bastara con:

- **DIR0 = 0** Direcciona el Pin 0 como entrada

Un 1 direcciona un pin como salida, mientras que un 0 direcciona un pin como entrada, al principio esto le puede resultar extraño, pero luego le resultara natural. El direccionamiento se coloca por lo general al principio del programa. Si se quiere direccionar el puerto completo como salida el formato será (DIRS = %1111111111111111), el registro DIRS contiene el puerto completo.

En la siguiente tabla se tiene que (DIRD = %0000), (DIRC = %1111), (DIRB = %1101) Y (DIRA = %0001).

DIRD				DIRC				DIRB				DIRA			
0	0	0	0	1	1	1	1	1	1	0	1	0	0	0	1
P15	P14	P13	P12	P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0

Esto quiere decir que el puerto D, esta definido como entrada, el puerto C, esta definido como salida, el puerto B, contiene 3 salidas y una entrada y el puerto A, contiene 3 entrada y una salida. Nótese en la tabla anterior el orden de los pines de (P15 – P0). Este es el orden que siempre debe llevar para mantener la secuencia. Del BIT más significativo y el menos significativo.

Esta definición también seria equivalente a (DIRS = %0000111111010001) o equivalente a (DIRH = %00001111) y (DIRL = %11010001).

Este formato es equivalente para los registros INS y OUTS.

Mapa de memoria completo del BS2

Nombre			
WORD	**BYTE**	**NIB**	**BIT**
INS	INL INH	INA,INB INC,IND	IN0,IN7 IN8,IN15
OUTS	OUTL OUTH	OUTA,OUTB OUTC,OUTD	OUT0,OUT7 OUT8,OUT15
DIRS	DIRL DIRH	DIRA,DIRB DIRC,DIRD	DIR0-DIR7 DIR8-DIR15
W0	B0 B1		
W1	B2 B3		
W2	B4 B5		
W3	B6 B7		
W4	B8 B9		
W5	B10 B11		
W6	B12 B13		
W7	B14 B15		
W8	B16 B17		
W9	B18 B19		
W10	B20 B21		
W11	B22 B23		
W12	B24 B25		

Variables de nombres fijos del BS2

Las variables son donde se guardan los datos en forma temporal. Una variable es un símbolo que contiene un cierto valor. Ese valor puede ser cambiado bajo el control del programa y por lo tanto, el valor de las variables puede cambiar, pero su nombre no.

PIBASIC puede utilizar variables con nombres de fabrica, como lo que se exponen en la tabla anterior o variables con nombres definidos por el usuario. En cualquier caso el numero de variable de 26 BYTE no varia. Las variables fijas tienen su orden de jerarquía (W0 es una variable tipo WORD de 16 BITS, que contiene a su vez a dos variables tipo BYTE de 8 BITS: B0 y B1). Por ejemplo suponga que la variable W0 contiene el valor binario (%0011101011101001), entonces B0 contiene la parte baja de 8 BITS y B1 la parte alta de los 8 BITS.

```
W0 = %0011101011101001
B0 = %11101001
B1 = %00111010
```

Las variables predefinidas de fabrica no necesitan ser declaradas PBASIC las reconoce. Pero puede ser algo confuso sobre todo cuando se tiene un programa muy extenso.

Afortunadamente PBASIC da la libertad de que usted defina sus propias variables con el nombre más apropiado siempre relacionándola a la acción a ejecutar. En otras palabras usted puede personalizar los nombres ejemplo: conteo_general = 56, en vez de B1 = 56, es mucho más fácil relacionar un nombre que un nombre fijo como B1.

Limite de longitud en los nombres de variables

En PBASIC, los nombres de las variables pueden tener una longitud de hasta 32 caracteres. La longitud del nombre no tiene ninguna influencia en la velocidad de ejecución del programa. Por ejemplo, la instrucción: x = 38, tendrá la misma velocidad de ejecución que: éste_es_un_nombre_muy_largo = 38.

De cualquier manera, en lugar de usar las variables predefinidas les puedo recomendar utilizar un nombre especifico para cada variable de acuerdo a algo relacionado con la aplicación utilizada o usando nombres con significado para usted.

5: Organización de memoria del BASIC Stamp 2

Declaración de Variables del BS2

La declaración de variables consiste en fijarle un nombre de menos de 32 caracteres y un tamaño en BITS. Las declaraciones de variables hay que realizarlas al principio del programa o antes de utilizarlas. Para declarar variables se utiliza el comando **VAR**. La sintaxis es la siguiente:

nombre_variable **VAR** tamaño

Donde:

- *Nombre_variable* es el nombre que usted le asignara a la variable no debe ser mayor de 32 caracteres, puede contener una secuencia de letras combinadas con números también acepta el guión largo "_". En PBASIC, los nombre_variable no son sensible a mayúsculas y minúsculas.

- *Tamaño* establece el numero de BITS a reservado. PBASIC da 4 tipos de tamaño:

Tipo			Elementos	Valores
BIT	1 Bit	2^1	2	(0-1)
NIB	4 Bits	2^4	16	(0-15)
BYTE	8 Bits	2^8	256	(0-255)
WORD	16 Bits	2^{16}	65536	(0-65535)

El espacio para cada variable es automáticamente destinado en la memoria del BasicStamp.

El tamaño de las variables a utilizar depende de la cantidad de variaciones que necesitemos, ejemplo de algunos casos utilizando nuestras propias definiciones con la sentencia VAR:

```
hormiga         VAR     bit             ' Puede tomar 2 elementos 0 y 1
gato            VAR     nib             ' Puede tomar 16 elementos desde 0 a 15
perro           VAR     byte            ' Puede tomar 256 elementos desde 0 a 255
elefante        VAR     word            ' Puede tomar 65,536 elementos desde 0 a 65,535
```

Es buena practica en principio que para cada sub-rutina, evento o formula matemática utilizar o definir una variable para cada caso. Esto evitaría algún conflicto o error inesperado. Luego en la optimización de su aplicación usted podrá notar como se pueden compartir muchas variables las cuales se utilizan en un evento y luego quedan libres sin efecto.

Si por ejemplo necesitamos leer una entrada del microcontrolador necesitamos una variable tipo **bit**, pues la entrada solo tiene dos valores posible 0 o 1 lógico.

Si necesitamos realizar un conteo del 1 al 10 es suficiente con una de tipo **nib**, pues esta puede contener 16 elementos. En este caso se podría utilizar una de tipo **byte**, pero la estaríamos sub-utilizando.

Si queremos almacenar un conteo de 10,000 necesitamos una tipo **word**, que puede almacenar hasta 65,536 elementos, la tipo **byte** en este caso seria menos que insuficiente, pues solo puede contener 256 elementos.

¿Que sucede cuando una variable excede el limite de su tamaño?
Cuando esto sucede la variable retorna a su origen es decir a cero. Por ejemplo si una variable tipo byte realiza un conteo de 258 elementos el resultado seria 2, pues la variable cuando llega a 255 en el próximo conteo de 256 se desborda a cero, luego a uno y después a dos.

Variables de grupo ARRAYS (ARREGLOS)

Los arreglos de variables pueden ser creados en una manera similar a las variables:

nombre_variable **VAR** tamaño(n)

Donde:

- nombre_variable y tamaño es el mismo de las declaraciones de variables. El nuevo elemento es (n), y le dice PBASIC cuanto espacio reservar para la misma variable del tamaño especificado.

Algunos ejemplos de creación de arreglo son los siguientes:

```
automovil   var   byte (10)   ' Crea 10 variables tipo byte
```

La primera ubicación dentro del arreglo es el elemento cero. En el arreglo automovil anterior los elementos están numerados automovil(0) a automovil(9) conteniendo 10 elementos en total. Dada la forma en que los arreglos están localizados en memoria hay límites de tamaño para cada tipo.

Tamaño	Número máximo de elementos
BIT	208
NIB	52
BYTE	26
WORD	13

Los arreglos son muy convenientes para recolección de datos, en vista de que el numero de elementos (*n*) puede ser sustituido por otra variable. Vea el siguiente ejemplo:

```
frutas          var     byte(5)
indice          var     nib
```

```
frutas(0) = 161
frutas(1) = 42
frutas(2) = 121
frutas(3) = 214
frutas(4) = 254

FOR indice = 0 TO 4
  DEBUG ? frutas(indice),CR
NEXT
STOP
```

Se visualizan de forma dinámica las siguientes 5 variables almacenadas previamente.

Nótese que el (numero de elementos) fue sustituido por una variable llamada índice, la cual puede tomar cualquier valor entre 0 a 15. Otra forma de realizar el mismo ejemplo pero menos eficiente es:

```
frutas        var    byte(5)

frutas(0) = 161
frutas(1) = 42
frutas(2) = 121
frutas(3) = 214
frutas(4) = 254

DEBUG ? frutas(0)
DEBUG ? frutas(1)
DEBUG ? frutas(2)
DEBUG ? frutas(3)
DEBUG ? frutas(4)
STOP
```

Se puede ver como en este ejemplo hubo que repetir la sentencia DEBUG 5 veces en vez de una como en el ejemplo anterior.

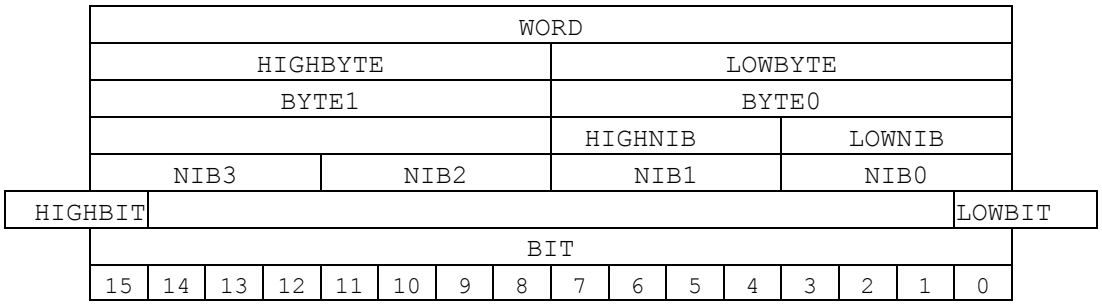
ALIAS (Modificadores)de variables

VAR también puede ser usado para crear un alias para otra variable. Esto es muy útil para acceder al interior de una variable.

```
dog   var   byte        ´ dog es una variable tipo byte
fido  var   dog         ´ fido es otro nombre de dog
```

En este ejemplo, fido es el alias de la variable dog. Cualquier valor almacenado en dog puede ser mostrado por fido y viceversa. Ambos nombres se refieren a lo mismo.

Con los ALIAS podemos acceder en cualquier momento al interior de una variable sin causar ninguna alteración a la variable original. En la siguiente tabla se puede ver la jerarquía de los modificadores o alias de las variables.




```
cuerpo_humano     VAR   WORD
cabeza            VAR   cuerpo_humano.HIGHBYTE
extremidades      VAR   cuerpo_humano.BYTE0
ojos              VAR   cabeza.BIT0
nariz             VAR   cabeza.BIT1
oido              VAR   cabeza.BIT2
boca              VAR   cabeza.BIT3
pie_izquierdo     VAR   extremidades.LOWNIB
pie_derecho       VAR   extremidades.HIGHNIB
cabello           VAR   cuerpo_humano.LOWBIT
dedo              VAR   cuerpo_humano.HIGHBIT

cuerpo_humano = %1101101010011000

DEBUG BIN16 cuerpo_humano, CR       'Muestra    1101101010011000
DEBUG BIN16 cabeza, CR              'Muestra    1101101010011000
DEBUG BIN16 extremidades, CR        'Muestra    1101101010011000
DEBUG BIN16 ojos, CR                'Muestra    1101101010011000
DEBUG BIN16 nariz, CR               'Muestra    1101101010011000
DEBUG BIN16 oido, CR                'Muestra    1101101010011000
DEBUG BIN16 boca, CR                'Muestra    1101101010011000
DEBUG BIN16 pie_izquierdo, CR       'Muestra    1101101010011000
DEBUG BIN16 pie_derecho, CR         'Muestra    1101101010011000
DEBUG BIN16 cabello, CR             'Muestra    1101101010011000
DEBUG BIN16 dedo, CR                'Muestra    1101101010011000
```

¿Que significa MSB y LSB?
MSB Most Significative Bit, es el bit mas significativo y LSB Lest Significative Bit, es el bit menos significativo.

Declaración de Constantes del BS2

Las llamadas constantes pueden ser creadas de manera similar a las variables. Puede ser más conveniente utilizar un nombre de constante en lugar de un número.

Son creadas usando la palabra clave **CON**, Si el número necesita ser cambiado, únicamente habría que cambiarlo en un parte del programa donde se define la constante. No pueden guardarse datos variables dentro de una constante. Esto es evidente. Mas adelante se podrá apreciar la importancia de definir constantes. En vez de fijar un numero. La sintaxis es la siguiente:

nombre_constante **CON** valor_numerico

Donde:

- *nombre_constante* es el nombre que usted le asignara a la variable no debe ser mayor de 32 caracteres, puede contener una secuencia de letras combinadas con números también acepta el guión largo "_". En PBASIC, los nombre_variable no son sensible a mayúsculas y minúsculas.

- *valor_numerico* es un valor de (0-65535).

PBASIC permite definir constantes numéricas en tres bases:
decimal, binaria y hexadecimal. Valores binarios son definidos usando el prefijo "**%**" y valores hexadecimales usando el prefijo "**$**". Los valores decimales se toman por defecto y no requieren prefijo. Ejemplo:

```
100   ´ valor decimal 100
%100  ´ valor binario para el decimal 4
$100  ´ valor hexadecimal para el decimal 256.
"A"   ´ ASCII equivalente a decimal (65).
```

Algunos ejemplos son:

```
bateria     CON   12
continentes CON   5
libro       CON   $E7
encendido   CON   %1101
detener     CON   "s"
```

Es posible calcular expresiones a traves de constantes previamente definidas:

```
temperatura       con   37
grados_c          con   (temperatura*5)/9
grados_f          con   (grados_c -32)
```

En el siguiente ejemplo, se demuestra la importancia en utilizar constantes numericas en vez de valores numericos. Conecte en el puerto A, cuatro diodos LEDS con su respectiva resistencia en serie.

```
DIRA = %1111          'Define el puerto A como salida

Ciclo:                'Etiqueta de referencia
  OUTA = %0000        'Salida A = 0000
  PAUSE 500           'Pausa de 500 mS
  OUTA = %0001
  PAUSE 500
  OUTA = %0011
  PAUSE 500
  OUTA = %0111
  PAUSE 500
  OUTA = %1111
  PAUSE 500
GOTO Ciclo            'Repetir el ciclo
```

En este ejemplo, suponga que requiere cambiar el ciclo de pausa de 500 ms por 250 ms. Para esto sería necesario cambiar o editar en cada sentencia de PAUSE.

```
DIRA = %1111          'Define el puerto A como salida

tiempo  CON    250
Ciclo:                'Etiqueta de referencia
  OUTA = %0000        'Salida A = 0000
  PAUSE tiempo        'Pausa = a la constante "tiempo"
  OUTA = %0001
  PAUSE tiempo        'Pausa = a la constante "tiempo"
  OUTA = %0011
  PAUSE tiempo        'Pausa = a la constante "tiempo"
  OUTA = %0111
  PAUSE tiempo        'Pausa = a la constante "tiempo"
  OUTA = %1111
  PAUSE tiempo        'Pausa = a la constante "tiempo"
GOTO Ciclo            'Repetir el ciclo
```

En este ejemplo se adiciono una constante llamada "tiempo" la cual contiene un valor numerico decimal de 250, en cada evento de PAUSE ejecuta el valor contenido en tiempo.

Lo que se puede lograr aquí es que si se desea cambiar el valor de la pausa, solamente es necesario cambiar el valor en la constante de "tiempo", en rutinas de programas extensas es muy importante trabajar con constantes.

Etiquetas de direccionamiento (labels)

Para marcar una referencia o dirección dentro del programa pueda referenciar con los comandos GOTO ó GOSUB, PBASIC usa etiquetas de línea. PBASIC no permite número de línea y no requiere que cada línea sea etiquetada. Cualquier línea PBASIC puede comenzar con una etiqueta de línea que es simplemente un identificador finalizando por dos puntos (:). Las etiquetas no deben ser mayores de 32 caracteres, pueden contener una secuencia de letras combinadas con números también acepta el guión largo "_". Las etiquetas no son sensibles a mayúsculas y minúsculas. En otras palabras las etiquetas:

```
Inicio_de_Programa_01:
inicio_de_programa_01:
INICIO_DE_PROGRAMA_01:
iNICIO_DE_pROGRAMA_01:
```

Tienen el mismo significado o valor para el editor Pbasic.

```
Inicio_de_Programa_01:        'Inicio de la Etiqueta
  DEBUG "Hola Mundo",CR
GOTO Inicio_de_Programa_01    'Volver al Inicio de la Etiqueta
```

Comentarios

Un comentario de PBASIC comienza con el apóstrofe (´). Todos los demás caracteres de esa línea se ignoran.

Los comentarios aunque no son obligatorios y a veces pueden aparentar innecesarios son de vital importancia, pues por mi experiencia puedo decir que he escrito programas y luego de unos pocos meses no puedo recordar el por que se escribieron algunas rutinas.

Al momento de realizar un programa todo puede aparentar muy claro. Pero les puedo asegurar que después de un tiempo le podrá aparentar sin sentido. Como regla especifica comenten cada línea de programación gástense unos segundos en el momento y no unas horas después.

Realmente los comentarios no ocupan espacio en memoria pues el compilador lo ignora como función, asi que no debe de preocuparse por documentar explicitamente sus programas.

Declaraciones múltiples

Para permitir programas más compactos y agrupamientos lógicos de comandos relacionados, PBASIC soporta el uso de (:) Para separar comandos ubicados en la misma línea. Los siguientes dos ejemplos son equivalentes.

```
W2 = W0
W0 = W1
```

```
W1 = W2
Es lo mismo que:

W2 = w0 : W0 = W1 : W1 = W2
```

En los dos casos, el tamaño del código generado es el mismo.

Otro ejemplo:

```
Loop:
  HIGH 0
  PAUSE 500
  LOW
  PAUSE 500
GOTO Loop
```

Es lo mismo que:

```
Loop:  HIGH 0 : PAUSE 500 : LOW 0 : PAUSE 500 : GOTO Loop
```

Aunque ambos codigos realizan la misma función, este ultimo puede resultar confuso.

Operadores matemáticos

Pbasic efectúa las operaciones matemáticas en el orden que se escriben de izquierda a derecha. En la suma y la resta esto no es problema pero cuando hablamos de multiplicación y división la historia es diferente. Por ejemplo la expresión **W1 = 45 + 56 * 4 / 3,** Pbsic resolveria este problema de la siguiente manera.

```
45 + 56 = 101
101 * 4 = 404
404 / 3 = 134
```

Otros lenguajes de programación incluyendo el Basic en las operaciones matemáticas tienen una jerarquía de prioridad por ejemplo: primero la multiplicación, segundo la división, tercero la suma y por ultimo la resta. Para resolver problemas de orden de prioridad podemos utilizar las reglas de paréntesis. Por ejemplo la expresión **W1 = 45 + (56 * 4) / 3** seria:

```
(56 * 4) = 224
45 + 224 = 269
269 / 3 = 89
```

Se pueden incluir paréntesis dentro paréntesis para especificar el orden deseado. Por ejemplo la expresión **W1 = 45 + ((56 * 4) / 3)** seria:

```
(56 * 4) = 224
(224 / 3) = 74
45 + 74 = 119
```

Se pueden utilizar un máximo de 8 paréntesis por operación en la ultima expresión se utilizaron 4 paréntesis. Esta regla de 8 paréntesis por operación no es problema pues si la formula es muy compleja se puede realizar en varios procedimientos. Por ejemplo la expresión **W1 = 45 + 56 * 4 / 3,** se puede expresar:

```
W1 = 45 + 56
W1 = W1 * 4       ' W1 Contiene la suma 45 + 56
W1 = W1  / 3      ' W1 Contiene la suma (45 + 56) por la multiplicación
de 4
```

O en forma de declaración múltiple de una línea:

```
W1 = 45 + 56 : W1 = W1 * 4 : W1 = W1  / 3
```

Se puede apreciar como la misma expresión matemática puede arrojar tres resultados diferentes, por esto usted debe tener mucho cuidado en las formulas matemáticas, si usted sigue la regla de izquierda a derecha difícilmente tenga problema.

Matemática Entera

Todas las operaciones matemáticas se realizan con cantidades positivas y con números enteros de (0 a 65535). Este es el valor numérico máximo que BS2 puede generar pues esta fundamentado en 16 Bits. Pbasic puede manejar también cantidades con signos negativos en este caso los valores máximos y mínimos serian: +32,767 y −32,767, en el caso de cantidades con signos negativos la presición de reduce a 15 Bits en vez de 16 Bits, esto tiene un significado y es que el Bit numero 16 expresa el signo matematico.

Operadores Binarios de (2 argumentos):

Operador	Descripción
+	Suma
-	Resta
/	División entera
//	Residuo de división (módulo)
*****	Multiplicación
******	Multiplicación de doble precisión
***/**	Multiplicación fraccionaria de 8 Bits
<<	Desplazamiento izquierdo de Cifras
>>	Desplazamiento derecho de Cifras
DIG	Selector de Digito de Cifras
MAX	Máximo
MIN	Mínimo
REV	Revertir bits
&	Operador lógico de cifras AND
|	Operador lógico de cifras OR
^	Operador lógico de cifras XOR

Suma (+)

Suma variables y constantes, devuelve el resultado de 16 bit, trabaja con cantidades enteras en un rango de 0 a 65535, si el resultado de la suma es mayor de 65535, se produce un desbordamiento y el resultado será la diferencia del desborde.

```
W1 = 452
W2 = 1023
W1 = W1 + W2
DEBUG DEC ? W1          ' Muestra el resultado (1475)
```

Resta (-)

Resta variables y constantes, devuelve el resultado de 16 bit, trabaja con cantidades enteras en un rango de 0 a 65535, si el resultado es negativo, podría expresarse con su signo. Con la expresión **SDEC**.

```
W1 = 199
W2 = 100
W1 = W1 - W2
DEBUG DEC ? W1          ' Muestra el resultado (99)
DEBUG SDEC ? W1         ' Muestra el resultado (99)
```

División (/)

Divide variables y constantes, devuelve la parte entera de la división con el resultado de 16 bit, trabaja con cantidades enteras en un rango de 0 a 65535, las cantidades deben ser positivas.

```
W1 = 500
W2 = 3
W1 = W1 / W2
DEBUG DEC ? W1          ' Muestra el resultado (166)
```

División (//) Residuo

Divide variables y constantes, devuelve la parte del residuo de una división con el resultado de 16 bit, trabaja con cantidades enteras en un rango de 0 a 65535, las cantidades deben ser positivas.

```
W1 = 500
W2 = 3
W1 = W1 // W2
DEBUG DEC ? W1          ' Muestra el resultado (2)
```

Multiplicación (*)

Multiplica variables y constantes, devuelve un resultado de 16 bits. Trabaja con cantidades enteras en un rango de 0 a 65535, las cantidades pueden ser positivas o negativas. Si el resultado excede el valor de 65535, el exceso se pierde, la multiplicación mantiene las reglas de los signos. Signos negativos por Signos Positivos Resultado Negativo, Signos iguales por Signos iguales Resultado Positivo. Si se trabaja con cantidades negativas los resultados máximos y mínimos serian +32,767 y –32,767.

```
W1 = 750
W2 = 23
W1 = W1 * W2
DEBUG DEC5  W1         ' Muestra el resultado (17250)

W1 = 750
W2 = -23
W1 = W1 * W2
DEBUG SDEC5  W1        ' Muestra el resultado (-17250)

W1 = -750
W2 = -23
W1 = W1 * W2
DEBUG SDEC5  W1        ' Muestra el resultado (17250)

W1 = 1248
W2 = 50
W1 = W1 * W2
DEBUG DEC5  W1         ' Muestra el resultado (62400)

W1 = 1248
W2 = 55
W1 = W1 * W2
DEBUG DEC5  W1         ' Muestra el resultado (03104)
```

En este ultimo ejemplo la cantidad 1248 x 55 = 68,640, como este valor excede los 65,535, el resultado es el sobrante después del exceso 3,104 si este valor le sumamos a 65,536 el resultado final es 68,640. Afortunadamente existe un operador matemático que nos permite saber la cantidad de desbordamiento. Este operador es la multiplicación doble **.

Multiplicación doble (**)

Como se puede apreciar la multiplicación no puede exceder el valor de 16 BITs o de 65,535. Como valor numérico decimal máximo. La multiplicación doble nos permite solucionar esto y es posible obtener resultados de hasta 32 BITs o 4,294,967,295. Realmente la multiplicación doble obtiene los 16 BITs superiores de los 32 BITs, los 16 BITs inferiores se obtienen con la multiplicación. En otras palabras la multiplicación doble nos dice cuantas veces se desbordaron los 16 BITs. Superiores Ejemplo:

```
W1 = 2560
W2 = 27
W3 = W1 * W2
DEBUG DEC5  W3, CR           ' Muestra el resultado (03584)
W3 = W1 ** W2
DEBUG DEC5  W3, CR           ' Muestra el resultado (00001)
```

El valor de 2,560 * 27 = 69,120, la multiplicación simple de Pbasic es 3,584 y la multiplicación doble nos indica 1, que en otras palabras nos dice que ocurrió un desborde de 65,536. Si sumamos esta cantidad con el valor 3,584 obtendremos el valor de 69,120 que es el valor correcto.

```
W1 = 2560
W2 = 120
W3 = W1 * W2
DEBUG DEC5  W3, CR            ' Muestra el resultado (45056)
W3 = W1 ** W2
DEBUG DEC5  W3, CR            ' Muestra el resultado (00004)
```

El resultado de la multiplicación doble nos indica que ocurrieron 4 desbordamiento en este caso seria 4 * 65,536 = 262,144 + 45056 = 307,200 que es el resultado de la operación 2,560 * 120.

Multiplicación de fracciones de 8 BITs (*/)

Que sucede si queremos multiplicar alguna fracción de numero por un numero entero. Pbasic solo maneja cantidades enteras. Pero es posible manejar fracciones para obtener resultados enteros. Por ejemplo si queremos multiplicar 100 * 3.5 el resultado es 350 el cual es un numero entero. La fracción 3.5 es el resultado de dos números enteros (7/2) = 3.5.

```
W1= 100 * (7/2)
DEBUG DEC5  W1, CR            ' Muestra el resultado (00300)
```

Como se puede ver el resultado es 300 en vez de 350, esto es por que el resultado de (7/2) es 3 y luego 100 * 3 = 300. Para resolver este problema seria de la siguiente manera:

```
W1= 100 * 7/2
DEBUG DEC5  W1, CR            ' Muestra el resultado (00350)
```

Pbasic ejecuta primero 100*7 = 700 y luego 700/2 = 350, Pbasic aplica la regla de operación aritmética de izquierda a derecha. Es claro de cómo podemos realizar operaciones con fracciones valiéndonos de artificios matemáticos. El valor 3.5 es el resultado de la división de dos números enteros (7/2). Pero que sucede si tenemos fracciones mas complejas como la constante del pi **Π**, 3.141592654, aunque la fracción (22/7) = 3.142857143 es muy semejante, se aprecia que después de la tercera cifra después del punto decimal las fracciones son diferentes.

La multiplicación de Fracción de 8 BITs resuelve este problema. Consiste en una constante Hexadecimal de 4 lugares donde los primeros dos lugares corresponden a la parte entera de (0 a 255) y los restante a una parte de fracción de (1/256 a 255/256). La resolución fraccionaria es de 1/256 = (0.003906). Realmente podemos obtener cualquier fracción a razón de los enteros comprendidos entre el rango de: (1 a 255) / 256. Por ejemplo:

```
1/2 = 0.50        128 / 256 = 0.50
1/3 = 0.33        085 / 256 = 0.33
5/8 = 0.625       160 / 256 = 0.625
```

***Muy Importante** : Para aplicar la Multiplicación de Fracciones (*/) se deben convertir los valores decimales a Hexadecimales.*

Pasos para encontrar la fracción del pi Π, 3.141592654.

1. Tomamos la parte de la fracción de 3.141592654
2. 0.141592654 multiplicamos la fracción por (256) = 36.24771932
3. Tomamos solo la parte entera del resultado anterior 36
4. Convertimos 36 Decimal a Hexadecimal $24
5. Tomamos la parte entera de 3.141592654
6. Convertimos el entero 3 Decimal a Hexadecimal $03
7. Unimos la parte entera con la equivalente decimal
8. El valor equivalente de la constante 3.141592 es equivalente a $0324

Donde $03 representa la parte entera y $24 la parte decimal.

Si queremos multiplicar la constante pi Π 3.141592, por el entero 150 la operación es:

```
W1 = 150 */ $0324
DEBUG DEC5  W1, CR            ' Muestra el resultado (00471)
```

Se puede obtener el mismo resultado a partir de la fracción (22/7). Recuerde que 22/7 = 3.142857143

```
W1 = 150 * 22 / 7
DEBUG DEC5  W1, CR            ' Muestra el resultado (00471)
```

Aunque ambos resultados son el mismo resulta mas seguro utilizar el numero Hexadecimal $0324 como constante equivalente de pi Π.

El valor Hexadecimal "$0324 en multiplicación de fracciones */" resulta realmente en valor decimal 3.14063 con relación al pi Π, original que es 3.14159, existe una diferencia de menos del 0.1%. Esto para la mayoría de casos es menos que despreciable.

***Desafió:** Hallar la fracción equivalente para el numero 17.51953.*

Desplazamiento de cifras (<< y >>)

Los operadores **<<** y **>>** desplazan un valor hacia la izquierda ó derecha respectivamente. (n) numero de lugares en la cifra o valor, Los bits desplazados se colocan en 0.

El comando (<<) desplaza la cifra hacia la izquierda (n) numero de lugares el resto es llenado por ceros. Este comando es equivalente a aplicar una multiplicación de 2^n donde n es el numero de lugar a desplazar. Por ejemplo:

```
valor       CON   %100101101111

W1 = valor << 2
DEBUG BIN16  W1, CR     ' Muestra el resultado (1001011011100)
```

← Corre la Cifra 2 Lugares hacia la izquierda rellena hacia la derecha dos lugares con ceros.

```
W1 = valor << 4
DEBUG BIN16  W1, CR     ' Muestra el resultado (1001011011110000)
```

← Corre la Cifra 4 Lugares hacia la izquierda rellena hacia la derecha cuatro lugares con ceros.

Nótese que el resultado de izquierda a derecha se pierde. En otras palabras cuando decimos: 234 << 3. Es equivalente a 234 * 2^3 = 234 * 8.

El comando (>>) desplaza la cifra hacia la derecha (n) numero de lugares el resto es llenado por ceros. Este comando es equivalente a aplicar una división de 2^n donde n es el numero de lugar a desplazar. Por ejemplo:

```
valor       CON   %100101101111

W1 = valor >> 2
DEBUG BIN16  W1, CR          ' Muestra el resultado (001001011011111)
```

→ Corre la Cifra 2 Lugares hacia la derecha rellena hacia la izquierda dos lugares con ceros.

```
W1 = valor >> 4
DEBUG BIN16  W1, CR          ' Muestra el resultado (0000100101101111)
```

→ Corre la Cifra 4 Lugares hacia la derecha rellena hacia la izquierda cuatro lugares con ceros.

Nótese que el resultado de derecha a izquierda se pierde. En otras palabras cuando decimos: 234 >> 3. Es equivalente a 234 / 2^3 = 234 / 8.

DIG

DIG retorna el valor de un dígito decimal de una cifra de 16 BITs. Simplemente se le indica el número de dígito a conocer entre (0 – 4), siendo 0 el primero de la derecha o el menos significativo y el 4 el

primero de la izquierda o el más significativo. En otras palabras el numero 63892.

Numero 6-3-8-9-2
Digito 4-3-2-1-0

```
VALOR  CON    47812
DEBUG "VALOR: ", DEC5 VALOR, CR, CR

FOR B0 = 0 TO 4
  DEBUG DEC1 B0, "-", DEC1 VALOR DIG B0, CR
NEXT
STOP
```

MAX y MIN

MAX y MIN devuelven el máximo y mínimo, respectivamente, de dos números. Se usan normalmente para limitar números a un valor limite.

MAX

Limita un valor positivo de 16 BITs a un valor máximo de limite. La sintaxis para MAX es la siguiente:

valor **MAX** Limite

Donde:

- Valor es el numero a evaluar para MAX funcione
- Limite es el máximo valor que podrá alcanzar el valor

Su lógica es, si el valor es mayor que el límite, entonces haz el valor = el límite; si el valor es menos o igual al limite, deja el valor igual. MAX sólo trabaja con matemática positiva. Sólo utilice MAX con enteros sin signos.

MIN

Limita un valor positivo de 16 BITs a un valor máximo de limite. La sintaxis para MIN es la siguiente:

Valor **MIN** Limite

Donde:

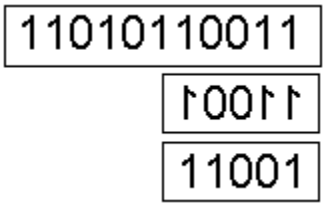
- Valor es el numero a evaluar para MIN funcione
- Limite es el máximo valor que podrá alcanzar el valor

Su lógica es, si el valor es menor que el límite, entonces haz el valor = el límite; si el valor es mayor o igual al limite, deja el valor igual. MIN sólo trabaja con matemática positiva. Sólo utilice MIN con enteros sin signos.

REV

Retorna el reflejo invertido en orden de una cifra, empezando de derecha a izquierda o por el bits menos significativo. El numero de Bit a reflejar es de 1 a 16. Ejemplo:

```
DEBUG BIN0 %110101[10011] REV 5       ' Invierte los primeros 5 BITs (%11001)
```



Operadores Logicos (AND, OR, XOR)

AND (&)

Retorna la comparación lógica **AND** de dos BITs a evaluar. La comparación AND es la siguiente:

```
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1
```

Ejemplo de una comparación AND &:

```
b0 = %11110000
b1 = %10011001
b1 = b0 & b1
DEBUG BIN8 b1, CR       ' Muestra el resulta AND de b0 & b1 (%1001000)
```

La comparación binaria trabaja de forma individual y evalúa cada bits de acuerdo a su posición. En otras palabras:

	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1
B0	1	1	1	1	0	0	0	0
B1	1	0	0	1	1	0	0	1
B0 & B1	1	0	0	1	0	0	0	0

OR (|)

Retorna la comparación lógica **OR** de dos BITs a evaluar. La comparación OR es la siguiente:

```
0 OR 0 = 0
0 OR 1 = 1
```

```
      1 OR 0 = 1
      1 OR 1 = 1
```

Ejemplo de una comparación OR |:

```
b0 = %11110000
b1 = %10011001
b1 = b0 | b1
DEBUG BIN8 b1, CR       ' Muestra el resulta OR de b0 | b1 (%11111001)
```

La comparación binaria trabaja de forma individual y evalúa cada bit de acuerdo a su posición. En otras palabras:

	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1
B0	1	1	1	1	0	0	0	0
B1	1	0	0	1	1	0	0	1
B0 & B1	1	1	1	1	1	0	0	1

XOR (^)
Retorna la comparación lógica **XOR** de dos BITs a evaluar. La comparación **XOR** es la siguiente:

```
      0 XOR 0 = 0
      0 XOR 1 = 1
      1 XOR 0 = 1
      1 XOR 1 = 0
```

Ejemplo de una comparación XOR ^:

```
b0 = %11110000
b1 = %10011001
b1 = b0 ^ b1
DEBUG BIN8 b1, CR       ' Muestra el resulta OR de b0 ^ b1 (%01101001)
```

La comparación binaria trabaja de forma individual y evalúa cada bit de acuerdo a su posición. En otras palabras:

	Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1
B0	1	1	1	1	0	0	0	0
B1	1	0	0	1	1	0	0	1
B0 & B1	0	1	1	0	1	0	0	1

Los operadores lógicos son empleados para aislar bits o para agregar bits dentro de un valor. Estos operadores actúan sobre cada bit de un valor en forma booleana.

```
B0 = B0 & %00001111 ´ aísla desde el bit 0 hasta el bit 3 de B0
B0 = B0 ÷ %00000001 ´ Hace que el bit 0 de B0 sea uno
B0 = B0 ^ %00000001 ´ Invierte el estado del bit 0 de B0
```

Operadores Binarios de (un argumento):

Operador	Descripción
ABS	Retorna el valor absoluto de un valor.
SQR	Retorna la Raíz Cuadrada de un valor.
DCD	$2^{n\ (0-15)}$ Elevar a la potencia
NCD	Prioridad de Bit codificado
SIN	Retorna el segundo complemento del Seno
COS	Retorna el segundo complemento del Coseno
-	Signo Negativo de una Cantidad
~	Complemento de un Numero

ABS

Devuelve el valor absoluto de un número. El valor absoluto de un número es el número sin su signo.

```
W1 = -47
DEBUG SDEC ? W1
W1 = ABS W1
DEBUG SDEC ? W1
```

SQR

SQR devuelve la raíz cuadrada de un valor. Como PBASIC solo trabaja con enteros, el resultado será siempre un entero de 8 bits no mayor que el valor original.

```
DEBUG CLS
FOR W0 = 1 TO 100 STEP 5
  W1 = W0
  W1 = SQR W1
  DEBUG DEC3 W0,"-",DEC3 W1, CR
NEXT
STOP
```

DCD

DCD devuelve la potenciación en base 2 entre un rango de exponentes de (0 a 15). En otras palabras:

B0 = DCD 4 ´ Muestra B0 como % 00001000

Valor	Decimal	Binario
0	1	0000 0000 0000 0001
1	2	0000 0000 0000 0010
2	4	0000 0000 0000 0100
3	8	0000 0000 0000 1000
4	16	0000 0000 0001 0000
5	32	0000 0000 0010 0000
6	64	0000 0000 0100 0000
7	128	0000 0000 1000 0000
8	256	0000 0001 0000 0000
9	512	0000 0010 0000 0000
10	1024	0000 0100 0000 0000
11	2048	0000 1000 0000 0000
12	4096	0001 0000 0000 0000
13	8192	0010 0000 0000 0000
14	16384	0100 0000 0000 0000
15	32768	1000 0000 0000 0000

Es equivalente a $2^4 = 16$

B0 = DCD 5 ´ Muestra B0 como % 00010000
Es equivalente a $2^5 = 32$

```
DEBUG CLS
W1 = 0
FOR B0 = 0 TO 15
  W1 = DCD B0
  DEBUG DEC2 B0,"-",BIN16 W1,"-",DEC5 W1, CR
  PAUSE 10
NEXT
STOP
```

NCD

NCD devuelve el número de prioridad de bit codificado en un rango de (1-16) de un valor. Se usa para encontrar la posición del bit más significativo de una cantidad. Retorna la posición donde se encuentre si existe, devuelve 0 si no existen bits con valor 1.

```
DEBUG CLS
W1 = %00000100
DEBUG DEC2 NCD W1, CR  ' MUESTRA LA POS. 3

W1 = %00100100
DEBUG DEC2 NCD W1, CR  ' MUESTRA LA POS. 6

W1 = %01100100
DEBUG DEC2 NCD W1, CR  ' MUESTRA LA POS. 7

W1 = %00000000
DEBUG DEC2 NCD W1, CR  ' MUESTRA LA POS. 0

STOP
```

SIN

SIN es seno en 8 bits de un valor. El está dado en dos complementos (-127 a 127). Usa una tabla de cuarto de onda para encontrar el resultado. Comienza con un valor en radianes binarios , 0 a 255 , en lugar de los usuales 0 a 359 grados.

Retorna el segundo complemento, de la función trigonometrica Seno con una resolución de 8-bit para un ángulo especificado entre (0 a 255) unidades. Para entender la FUNCIÓN SIN más, miremos una función típica del seno. Por definición: dado un círculo con un radio de 1 unidad (conocida como círculo de la unidad), el seno es la distancia de la coordenada (y) del centro del círculo a su borde del ángulo dado.

En el punto de origen el seno es (0 grados), porque en este punto tiene la misma coordenada (vertical) de (y) hacia el centro del círculo; en 45 grados, el seno es 0.707; en 90 grados, 1; 180 grados, 0 otra vez; 270 grados, -1.

La FUNCIÓN SIN divide el círculo en 255 unidades en vez de 0 a 359 grados. Algunos libros de textos llaman a esta unidad (radián binario). Cada radián es equivalente a 1.406 grados. Y en vez de una unidad de círculo, que da lugar a valores fraccionarios del seno 0 y 1, la FUNCIÓN SIN se basa en un círculo de 127 unidades. Resultando el segundo complemento para acomodar los valores negativos. Así pues, en el origen, el seno es 0; en 45 grados (32 radianes binarios), 90; 90 grados (64 radianes binarios), 127; 180 grados (128 radianes binarios), 0; 70 grados (192 radianes binarios), -127.

Para convertir los radianes binarios a grados, multiplíquese por 180 y luego se dividen por 128; Para convertir grados a los radianes binarios, multiplique por 128, entonces se dividen por 180.

COS

COS es coseno en 8 bit de un valor dado. El resultado está dado en forma de dos complementos. (-127 a 127). Utiliza una tabla de un cuarto de onda para encontrar el resultado. El coseno comienza con un valor en radianes binarios, 0 a 255, en lugar de los comunes 0 a 358 grados.

Retorna el segundo complemento, de la función trigonometrica Coseno con una resolución de 8-bit para un ángulo especificado entre (0 a 255) unidades. La función Coseno es similar a la Seno excepto que la función de coseno devuelve la distancia de (x) en vez de la distancia de (y).

(**-**)

Niega un numero de 16 BITs (Convierte el valor al segundo complemento).

```
W0 = 78
DEBUG SDEC5 W0, CR
W0 = -W0
DEBUG SDEC5 W0, CR
END
```

(**~**)

Invierte los bits de un numero. Si el bit vale 0 lo invierte a 1, si es 1 lo invierte a 0.

```
W0 = %10011110
DEBUG BIN8 W0, CR            ' MUESTRA (10011110)
W0 = ~W0
DEBUG BIN8 W0, CR            ' MUESTRA (01100001)
END
```

Clasificación de comandos:

El lenguaje Pbasic esta conformado por 37 comandos, 24 funciones matemáticas. Instrucciones para definiciones de variables, constantes y etiquetas de referencia. La combinación de estos comandos con las referencias de direcciones (etiquetas), conformaran un programa en Pbasic. La complejidad del mismo dependerá de lo que usted quiera realizar.

La gran mayoría de instrucciones de Pbasic esta orientada al procesamiento de señales de entrada y salida de uso industrial, las otras están destinadas a la evaluación de datos y cálculos matemáticos.

Cada una de estas funciones se explicara en detalle con sus reglas y parametros asi tambien con algunos ejemplos.

Bifurcaciones

IF...THEN	Evaluación para tomar una decisión según la condición sea Falso o Verdadera.
BRANCH	GOTO computado (equiv. a ON..GOTO).
GOTO	Salta a una posición especificada dentro del programa, a través de una dirección de etiqueta.
GOSUB	Llama a una subrutina PBASIC en la dirección de etiqueta especificada.

Ciclos repetitivos controlados

FOR...NEXT	Bucle controlado, ejecuta declaraciones en forma repetitiva.

Acceso de Datos a la EEprom

DATA	Almacena datos en la EEPROM del BS2.
READ	Lee un BYTE de la EEPROM del BS2.
WRITE	Graba un BYTE en EEPROM del BS2.

Búsqueda de Datos y Tabla de Datos

LOOKUP	Obtiene un valor constante de una tabla.
LOOKDOWN	Busca un valor en una tabla de constantes.
RANDOM	Genera numero aleatorio (0-65535).

Señales Digitales

INPUT	Convierte un pin en entrada.
OUTPUT	Convierte un pin en salida.
REVERSE	Convierte un pin de salida en entrada o uno de entrada en salida.
LOW	Hace bajo la salida de un pin.
HIGH	Hace alto la salida del pin.
TOGGLE	Cambia el estado de un pin si es alto lo convierte en bajo, si es bajo lo convierte alto.
PULSIN	Mide el ancho de pulso en un pin.
PULSOUT	Genera pulso en un pin.
BUTTON	Entrada de pulsadores momentáneos, Anti-rebote y auto-repetición de entrada en el pin especificado.
COUNT	Cuenta él numero de pulsos en un pin en un tiempo determinado.
XOUT	Salida X-10 Formato de control de electrodomésticos y alarmas a través de la red eléctrica 110 AC / 60Hz.

Comunicación Asincrónica

SERIN	Entrada serial asincrónica (RS-232).
SEROUT	Salida serial asincrónica (RS-232).

Comunicación Sincrónica

SHIFTIN	Entrada serial sincrónica (SPI).
SHIFTOUT	Salida serial sincrónica (SPI).

Señales Análogas

PWM	Salida modulada en ancho de pulso a un pin.
RCTIME	Mide capacitores o resistores en función del tiempo de carga conformado por un circuito (RC).

Funciones de Tiempo

PAUSE	Hace una pause de (0-65535) milisegundos.

Funciones de Sonido

FREQOUT	Produce una o dos 2 frecuencias en un pin especificado.
DTMFOUT	Produce tonos DTMF de formato telefónico en un pin especifico.

Control de Energía

NAP	Apaga el procesador por un corto periodo de tiempo.
SLEEP	Descansa el procesador por un periodo de tiempo.
END	Detiene la ejecución e ingresa en modo de baja potencia

Depuración de Programa

DEBUG	Salida de Datos por el puerto de programación

BRANCH

BRANCH Indice, [Etiqueta0,Etiqueta1,... EtiquetaN]

Función
Salta o se dirige a la etiqueta señalada por índice sí esta en el rango.

- **Indice** es una variable/constante/expresión por lo general tipo BYTE que especifica la cantidad de etiquetas desde (0 – 255).

- **Etiqueta** son las direcciones de referencia o bloque de inicio de programas hacia donde apuntara.

Explicación
La variable (indice) selecciona una etiqueta de una lista de acuerdo con su orden. La ejecución comienza en la etiqueta especificada. Por ejemplo, si (indice) vale 0, el programa salta a la primera etiqueta especificada en la lista, si (indice) es 1, salta a la segunda y así sucesivamente. Si (indice) es mayor ó igual al número de etiquetas, no se toma ninguna acción y la ejecución continúa con la declaración siguiente al BRANCH. Se pueden usar hasta 256 etiquetas en una intrucción BRANCH.

Ejemplo 1

```
{$STAMP BS2}

caracter       VAR    BYTE

Principal:

  DEBUG 2,1,1,"Seleccione una Fruta del [0 - 7]", CR
  SERIN 16, 16468, [DEC1 caracter]
  BRANCH caracter,[Uva, Pera, Manzana, Guineo, Melón, Chinola, Freza, Cereza]
  DEBUG CLS
  DEBUG 2,1,2,"No ha seleccionado correctamente"
  PAUSE 1500
  DEBUG CLS

GOTO Principal

Uva:
  DEBUG 2,2,4,"Ha seleccionado Uva......"
GOTO Principal

Pera:
  DEBUG 2,2,4,"Ha seleccionado Pera....."
GOTO Principal

Manzana:
  DEBUG 2,2,4,"Ha seleccionado Manzana.."
GOTO Principal

Guineo:
  DEBUG 2,2,4,"Ha seleccionado Guineo..."
GOTO Principal
```

```
Melón:
  DEBUG 2,2,4,"Ha seleccionado Melón...."
GOTO Principal

Chinola:
  DEBUG 2,2,4,"Ha seleccionado Chinola.."
GOTO Principal

Freza:
  DEBUG 2,2,4,"Ha seleccionado Freza...."
GOTO Principal

Cereza:
  DEBUG 2,2,4,"Ha seleccionado Cereza..."
GOTO Principal
```

BUTTON

BUTTON Pin, Downstate, Delay, Rate, Workspace, Targetstate, Etiqueta

Función

La instrucción BUTTON es utilizada para leer el estado de pulsadores momentáneos (Push-Botón), su funcionamiento es similar al de un teclado de una computadora. Incluye retrasos antes de tomar una acción, auto repeticiones y no-auto repeticiones. Cuando un Push-Botón es accionado bajo el dominio la instrucción BUTTON este se dirige a la dirección o etiqueta señalada.

- **Pin** puede ser variable/constante/expresión de (0-15), especifica el pin a utilizar. Este pin se declara como entrada automáticamente.

- **Downstate** puede ser variable o constante (0-1), especifica el estado lógico cuando se presiona el botón. Ver grafico.

- **Delay** puede ser variable o constante (0-255)y especifica cuánto tiempo el botón debe ser presionado antes comenzar la auto-repeticiones. Delay se mide en los ciclos de la rutina de BUTTON. Delay tiene dos configuraciones especiales: 0 y 255. Si Delay es 0, no realiza ningún antirebotes o auto-repeticiones. Si Delay es 255, el botón realiza antirebotes, pero no auto-repeticiones.

- **Rate** puede ser variable o constante (0-255)y especifica el número de ciclos entre las auto-repeticiones. El Rate se expresa en los ciclos de la rutina de BUTTON.

- **Workspace** es una variable de trabajo que la función BUTTON, necesita para operar, Debe ser inicializada antes de utilizarla.

- **Targetstate** puede ser variable o constante (0-1) y especifica cual estado debe tomar el botón para que ocurra una acción a la dirección señalada. (0 = no presionado, 1 = presionado).

- **Etiqueta** es la dirección de referencia que especifica a donde apuntara el programa si el botón es presionado.

Explicación

Cuando usted presiona un botón o mueve un interruptor, los contactos producen una pequeña explosión o chispa que puede durar entre (1 a 20-ms), el cambio brusco de un estado a otro provoca un ruido o señal no deseada. Esta señal de ruido el Microcontrolador la puede interpretar como un tren de pulsos continuos. En 20-ms El Microcontrolador es capaz de leer unas 2,500 veces. Este efecto puede provocar una lectura errónea o no deseada. Este fenómeno también se le conoce como rebote. Existen circuitos electrónicos y pulsadores que ayudan a minimizan este

efecto no deseado, se les llama circuitos antirebotes o (Debounce). La instrucción BUTTON evita los rebotes de manera inteligente, puede distinguir entre un ruido y una pulsación verdadera.

La instrucción BUTTON es muy similar al comportamiento de un teclado de computadora. Cuando usted presiona cualquier tecla en una PC el carácter aparece inmediatamente en la pantalla. Si usted deja la tecla presionada luego de un tiempo prudente comienza las auto-repeticiones y los caracteres aparecen repetidamente.

La instrucción BUTTON esta diseñada para ser utilizada dentro de un bucle del programa. Cada vez a través del bucle, el botón controla el estado del contacto especificado.

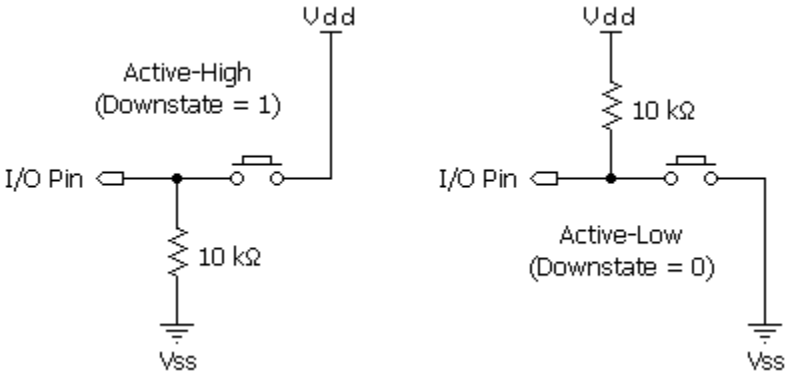


Figura 8.1 Conexión para tipica para la entrada de interruptores y pulsadores momentaneos (Push-Button)

Un ejemplo vale más que 1000 palabras.

Ejemplo 1

```
{$STAMP BS2}

btn t          VAR     BYTE

pin0           CON     0       ' Conectar como esta en el diagrama
downstate      CON     0
delay          CON     255
rate           CON     250
targetstate    CON     1

Loop_Main:
  BUTTON pin0, downstate, delay, rate, btn_t, targetstate, No_Press
  DEBUG "*"

No Press:
  GOTO Loop_Main              'Retorna a Loop_Main
```

Luego cambie los parametros: delay, rate y targetstate.

COUNT

COUNT Pin, Periodo, Variable

Función

Cuenta él numero de ciclos de una señal (0-1-0 o 1-0-1) en el pin especificado durante el período especificado en milisegundos y almacena él numero de ciclos en una variable.

- **Pin** puede ser variable/constante/expresión de (0-15), especifica el pin a utilizar para la entrada de señal. Este pin se declara como entrada automáticamente.

- **Período** puede ser variable/constante/expresión de (1-65535) especifica el tiempo en milisegundos durante los cuales contar.

- **Variable** donde se almacenaran los números de ciclos (generalmente es de tipo Word) o 16 Bits.

Limites

Ancho del pulso Minimo	8 µS
Máxima Frecuencia	120,000 Hz

Explicación

La instrucción COUNT declara el pin especificado como entrada, entonces cuenta la cantidad de ciclos de 1 a 0 a 1, o de 0 a 1 a 0, en un tiempo en milisegundos preestablecido por Periodo. La cantidad de ciclos contados durante ese tiempo es almacenada en la Variable.

La instrucción COUNT puede responder a las transiciones tan rápidamente como 4 microsegundos (µs). Un ciclo consiste en dos transiciones (0 a 1, entonces 1 a 0), así que la cuenta puede responder a las ondas cuadradas con los períodos tan cortos como 8 µs; o frecuencia de no más de 120,000Hz.

Si usted quiere medir señales débiles o con ruidos, usted puede pasar la señal a través de un disparador Schmitt, como el inversor 74HCT14. Esto evitara conteos erróneos.

La instrucción COUNT es la manera más apropiada de medir frecuencia, la frecuencia es el numero de ciclos en un segundo. Cuando una señal es de 60Hz quiere decir que contiene 60 variaciones o ciclos en un segundo. Frecuencia = 1/T.

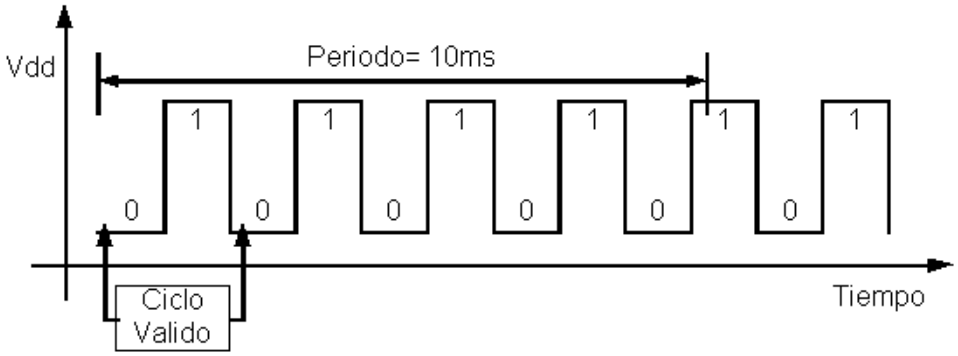


Figura 8.2 Tren de pulsos, se aprecia una onda cuadrada típica, el numero de ciclos en 10 mili-segundos es de 4 ciclos validos.

Ejemplo 1

```
{$STAMP BS2}

F_in            CON     0       ' Entrada de la señal por P0 del BS2
freq            VAR     WORD    ' Variable

Main:
  COUNT F_in, 10, freq          ' Cuenta él numero de ciclos en 10 mseg.
  DEBUG CLS, "No. de Ciclos: ", DEC5 freq    ' Muestra 4 ciclos
GOTO Main                       ' Ir a Main
END                             ' Fin del programa
```

Si se establece el Periodo a 1000 milisegundos, es decir, Un 1 segundo, el numero de ciclos contados en Un 1 segundo es igual a la frecuencia. Pero muchas veces no es necesario esperar 1000 milisegundos, para obtener él numero de ciclos por segundo, si en vez de esperar Un 1 segundo, esperamos ½ medio segundo 500 milisegundo, la muestra obtenida en la variable se multiplica por dos. Y se obtiene la frecuencia.

Ejemplos con diferentes Periodos:

```
COUNT F_in, 1000, freq          ' Cuenta el numero de ciclos en 1 seg.
DEBUG CLS, "Frecuencia : ", DEC5 freq
END
```

```
COUNT F_in, 500, freq           ' Cuenta el numero de ciclos en ½ seg.
freq = freq * 2
DEBUG CLS, "Frecuencia : ", DEC5 freq
END
```

```
COUNT F_in, 100, freq           ' Cuenta el numero de ciclos en 100 mseg.
freq = freq * 10
DEBUG CLS, "Frecuencia : ", DEC5 freq
END
```

En la figura que se muestra arriba la frecuencia es 40 Hz:
4 ciclos * 10mseg = 40Hz.

DATA

{Etiqueta} DATA DatoTipo {, DatoTipo...}

Función
Almacena datos en la memoria EEprom del BS2 en tiempo de programación.

- **Etiqueta** es opcional pero muy importante y nos indica el comienzo de donde empieza un dato, más adelante se podrá apreciar su valioso uso. Es una referencia de direccionamiento.

- **DatoTipo** es una constante/expresión (0-255) indica el valor a almacenar.

Explicación
Cuando usted descarga un programa en el Basic Stamp, este almacena las instrucciones en una memoria EEprom de 2K BYTE. Por lo general esta memoria EEprom no es ocupada totalmente por las instrucciones de programación. Es posible utilizar la parte no ocupada para almacenar: cadena de texto, tabla de datos, clave de acceso, números telefónicos en fin cualquier dato tipo BYTE de (0-255), puede ser almacenado en la memoria EEprom del BS2.

Si la memoria es utilizada para almacenar las instrucciones de un programa, como no interfiere con los datos. Esto es posible gracias a que las instrucciones se almacenan en la memoria EEprom de abajo hacia arriba, es decir, de la dirección 2047 a la dirección 0 y la data se almacena de la dirección 0 hacia la dirección 2047. De esta manera se logra que las instrucciones de programación no interfieran con los datos almacenados. Obviamente esto es posible siempre que el programa no ocupe toda la porción de la memoria EEprom. Para visualizar el contenido de la memoria en tiempo de programación es posible con el editor de pbasic presionando la combinación de teclas [Ctrl+M].

Con la función DATA es posible almacenar cadenas completa de texto y datos de forma secuencial, el siguiente ejemplo nos muestra como se utiliza este comando:

```
DATA  100,200,65,90,87
```

En este ejemplo cuando se descarga el programa hacia el BS2, los valores 100,200,65,90,87 son escritos en la EEprom en las direcciones 0,1,2,3 y 4 respectivamente. Usted puede utilizar el comando READ para leer el contenido de estas direcciones.

DATA utiliza un contador automático, llamado apuntador, el cual mantiene el rastro de la dirección de localización de la EEprom. El valor inicial de este apuntador es (0) cero. Cuando un programa es descargado, y se utiliza el comando DATA el apuntador se incrementa automáticamente de uno en uno por cada valor de DATA. Usted no tiene

que preocuparse en direccionar las posiciones de memoria esto lo realiza el apuntador de forma automática. En el siguiente ejemplo vea como trabaja el apuntador:

```
DATA 67,70,56,78
DATA 99,100,45,88,74
```

	Localización de la EEprom								
Dirección	0	1	2	3	4	5	6	7	8
DatoTipo	67	70	56	78	99	100	45	88	74

Se puede apreciar que cuando finaliza el primer comando DATA que termina con el DatoTipo (78), en la dirección 3, el siguiente DATA asigna automáticamente la siguiente dirección la numero (4), con el DatoTipo (99) y finalizando en la dirección numero (8).

Pero que sucede si no queremos que el apuntador comience en (0) cero? Afortunadamente la función DATA tiene una opción que le especifica al apuntador donde debe iniciar. Observe el siguiente ejemplo:

```
DATA @50, 67,70,56,78,99,100,88,74
```

En este ejemplo se le indica al apuntador que comience en la dirección numero (50). **@**x, le indica que comience donde indica x, y que a partir de hay incremente automáticamente en una unidad.

	Localización de la EEprom								
Dirección	50	51	52	53	54	55	56	57	58
DatoTipo	67	70	56	78	99	100	45	88	74

En el caso que usted quiera almacenar el DatoTipo (56) en la posición 100 y el DatoTipo (98) en la posición 120, el formato seria el siguiente:

```
DATA @100, 56, @120, 98
```

El uso de la referencia o etiqueta en DATA

Suponga que usted tiene una serie de mensajes de texto para utilizar en una pantalla de cristal liquido (LCD).

```
DATA  "Sistema Ver 1.5"
DATA  "Temperatura C:"
DATA  "Temperatura F:"
DATA  "Error en la medida"
DATA  "Remplace las Baterías"
```

La siguiente información se grabara de manera secuencial en la memoria EEprom comenzando por la posición (0). Hasta ahora no hay problema si queremos desplegar cualquier mensaje debemos conocer en que dirección comienza el mensaje y cual termina. En el primer mensaje sabemos que comienza en la dirección (0). Para el segundo mensaje tenemos que contar cuantos caracteres contiene el primero. Para el tercer mensaje tenemos que contar cuantos caracteres tiene el segundo más el tercero.

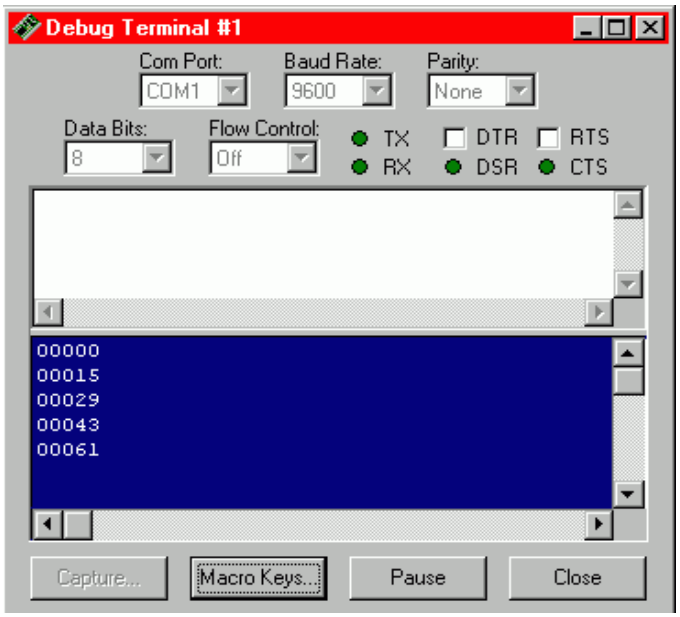
En fin esto resulta extremadamente tedioso e incomodo de realizar. La solución es las etiquetas de referencia.

Etiquetas de referencia en DATA

En realidad cuando agregamos una etiqueta de referencia al comando DATA, no afecta en si ni ocupa un espacio adicional en la memoria. Las etiquetas de referencia nos indican el sitio exacto donde comienza cada mensaje o tabla que se genere con el comando DATA. Al ejemplo anterior le agregaremos etiquetas de referencia. Las etiquetas de referencia contienen el mismo formato de las referencia para declaraciones de variables, constates y etiqueta de programación. No más de 32 caracteres, combinación de números y letras unidas, no importa que este en mayúscula o minúscula.

Ejemplo 1

```
Mensaje_No_01   DATA    "Sistema Ver 1.5"
Mensaje_No_02   DATA    "Temperatura C:"
Mensaje_No_03   DATA    "Temperatura F:"
Mensaje_No_04   DATA    "Error en la medida"
Mensaje_No_05   DATA    "Remplace las Baterías"

DEBUG DEC5 Mensaje No 01,CR
DEBUG DEC5 Mensaje No 02,CR
DEBUG DEC5 Mensaje No 03,CR
DEBUG DEC5 Mensaje_No_04,CR
DEBUG DEC5 Mensaje_No_05,CR
```



El resultado obtenido en la pantalla por el comando DEBUG es 0,15,29,43 y 61, que indica la posición de inicio de cada mensaje. Usted no tiene que saber él numero de la posición usted solamente refiere el nombre de la etiqueta.

Hasta el momento hemos resuelto parte del problema pero aun no sabemos dónde termina cada mensaje. Podemos decir entonces que el primer mensaje comienza en (Mensaje_No_01) y termina en (Mensaje_No_02-1) y el segundo comienza e (Mensaje_No_02) y termina en (Mensaje_No_03-1) y así sucesivamente. Otra forma muy conveniente es agregar un valor final de (0), a cada mensaje con la finalidad de poder depurar el fin del mensaje. Como la tabla de caracteres ASCII comienza en decimal (65) no habrá mayores inconvenientes. Vea el siguiente ejemplo:

Ejemplo 2

```
Indice          VAR     WORD
Carácter        VAR     BYTE

Mensaje_No_01   DATA    "Sistema Ver 1.5",0
Mensaje_No_02   DATA    "Temperatura C:",0
Mensaje_No_03   DATA    "Temperatura F:",0
Mensaje_No_04   DATA    "Error en la medida",0
Mensaje_No_05   DATA    "Remplace las Baterías",0

Indice = Mensaje_No_01
GOSUB ReadChar

Indice = Mensaje No 02
GOSUB ReadChar

Indice = Mensaje_No_03
GOSUB ReadChar

Indice = Mensaje No 04
GOSUB ReadChar

Indice = Mensaje_No_05
GOSUB ReadChar

END

ReadChar:
  READ Indice, Carácter
  IF Carácter = 0 THEN Exit Read
  DEBUG Carácter
  Indice = Indice + 1
  GOTO ReadChar
Exit_Read:
DEBUG, CR
RETURN
```

Esta rutina de programación es muy funcional para desplegar mensajes por pantalla LCD. Se apunta al inicio con la etiqueta del mensaje luego se evalúa hasta encontrar al numero (0) el cual indica fin del mensaje.

La función DATA se complementa con la función READ, DATA almacena la información durante la programación, es decir, los datos quedan almacenados cuando se descarga el programa al BS2. Una vez recargado el programa los datos permanecerán definitivamente o hasta que se sobrescriba nuevamente con un programa nuevo. Es conveniente recordar las características de las memorias EEprom de conservar la información por más de 10 años una vez grabadas aun sin energía y la capacidad de borrarlas y escribirlas nuevamente hasta 10,000,000 de veces sin alterar sus propiedades físicas.

Diferentes formatos de almacenamiento

Por lo general el uso más frecuente de DATA es almacenar cadenas de caracteres. Cada carácter ocupa un BYTE, DATA tiene tres forma de aceptar los DatoTipo los cuales son:

```
DATA  72,69,76,76,79
DATA  "H","E","L","L","O"
DATA  "HELLO"
```

Información general de las memorias EEprom

Se trata de memorias de sólo lectura, programables y borrables eléctricamente EEPROM (Electrical Erasable Programable Read Only Memory). Tanto la programación como el borrado, se realizan eléctricamente. Son muy cómoda y rápida la operación de grabado y la de borrado. No disponen de ventana de cristal en la superficie como las Eprom tradicionales. Las cuales se borraban con una luz ultravioleta.

Las memorias EEPROM, pueden grabarse y borrarse hasta un 1,000,000 de veces sin alterar su estructura física, esto lo hace sumamente útil durante el desarrollo de una aplicación.

Su gran flexibilidad y rapidez a la hora de realizar modificaciones en el programa de trabajo. El número de veces que puede grabarse y borrarse una memoria EEPROM es finito, por lo que no es recomendable una reprogramación continua. Son muy idóneas para la enseñanza y la Ingeniería de diseño.

Se va extendiendo en los fabricantes la tendencia de incluir una pequeña zona de memoria EEPROM en los circuitos programables para guardar y modificar cómodamente una serie de parámetros que adecuan el dispositivo a las condiciones del entorno.

La acción de retener la información aun sin energía constituye la forma más practica para almacenar información, es lo mas parecido a almacenar información en medios magnéticos pero aun mejor.

DEBUG

DEBUG Outputdata{, Outputdata ...}

Función

Visualiza variables y mensajes por la pantalla de la PC en combinación con el editor del BS2. Este comando es utilizado para visualizar textos y números en varios formatos.

- **OutputData** salida de datos pueden ser variable/constante/expresión, del rango comprendido entre (0-65535) especifica la salida de datos. La salida de datos puede estar en caracteres ASCII (Texto entre comillas " ", y caracteres de control), los números decimales (0-65535), los números hexadecimales ($0000-$FFFF), y los números binarios (%0000000000000000-%1111111111111111). La data numérica puede ser modificada con formatos como se explicara mas adelante.

Explicación

DEBUG es la manera más conveniente de visualizar por pantalla mensajes de datos desde el BS2 hacia una PC. El comando DEBUG es también una gran manera de ensayar técnicas de programación. DEBUG se le conoce también como depuración, en muchos lenguajes de programación el DEBUG es conocido como Debuging y es utilizado para visualizar variables y eventos durante la programación. En el caso del BS2, DEBUG proporciona la herramienta de visualizar cualquier variable o desplegar cualquier mensaje. La ventaja que nos ofrece es que podemos tener acceso al interior de cualquier variable o registro de entrada y salida o direccionamiento del puerto del BS2. Que sucedería si sumamos dos variables y quisiéramos saber el resultado, o si creamos una formula matemática pero no estamos seguros si el resultado será el esperado. Una manera eficaz de realizarlo es a través de DEBUG. El siguiente ejemplo nos muestra como desplegar un mensaje de texto, fíjese como el texto esta entre comillas.

Ejemplo 1

```
DEBUG "Aprender BS2 es muy facil!" ' Mensaje de prueba.
END
```

Después de que usted corra este programa con (Ctrl + R), el editor del BS2 abrirá una ventana que se llama "Debug Terminal" y usted podrá visualizar:

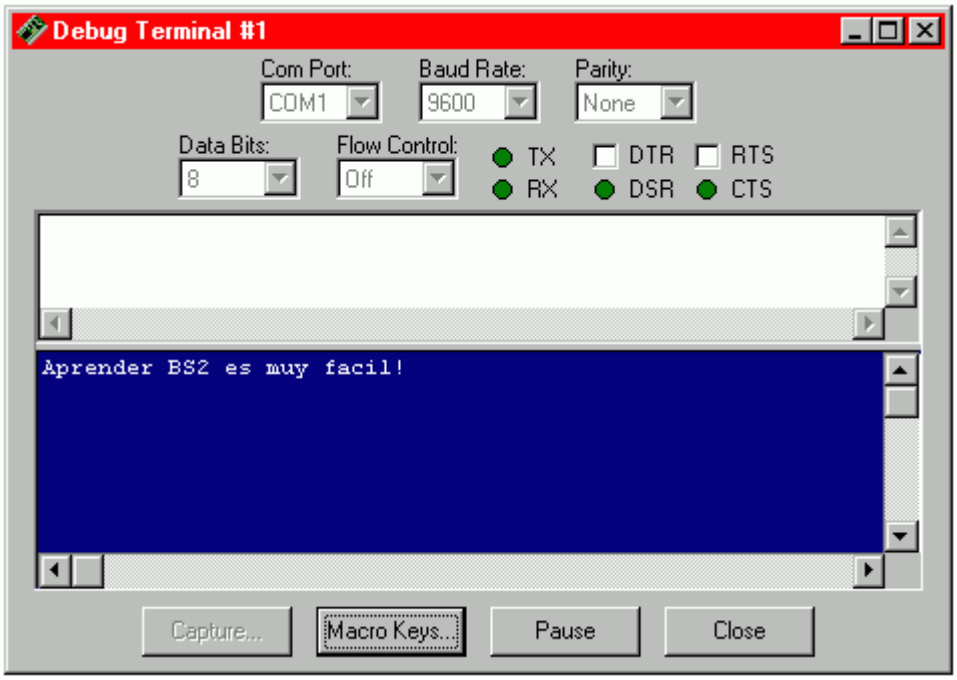


Figura 8.4 Pantalla tipica de la función DEBUG.

Presionando cualquier tecla con excepción de la barra espaciador se puede eliminar la ventana del Debug Terminal. Realmente el BS2 guarda el mensaje en su memoria EEPROM, para volver a ejecutar el DEBUG presione (CTRL + D) aparecerá nuevamente la ventana del Debug Terminal luego inicialice el BS2 presionando el botón de Reset y el mensaje aparecerá nuevamente.

Se pueden enviar múltiples datos y comandos con una solo una instrucción DEBUG separándolos por comas (,). El siguiente ejemplo produce el mismo efecto que el ejemplo anterior.

```
DEBUG "Aprender BS2 ", "es muy facil!" ' Mensaje de prueba.
END
```

Hasta el momento solo se han desplegado caracteres pero que sucede si queremos visualizar números.

```
numero      VAR   BYTE        ' Declaración de una variable tipo Byte
numero = 65                   ' Fijando el valor 65 en numero
DEBUG numero                  ' Visualizar el valor por pantalla
```

Usted podría esperar que apareciera el decimal "65", en la pantalla, sin embargo, lo que aparece es la letra "A". El problema que sucede es que nosotros le debemos especificar al BS2 si es un valor numérico o un carácter. El BS2 entiende por defecto los valores de tabla ASCII. De (0 – 31) como caracteres de control y de (32 – 127) como caracteres del alfabeto. Cuando al BS2 no se le especifica el formato de salida

entonces el valor será el que corresponda en la tabla del código ASCII. Para desplegar valores numéricos necesitamos auxiliarnos de los Modificadores de Formatos.

Los modificadores de formatos son parámetros adicionales que especifican al DEBUG el valor de conversión que puede ser ASCII, Decimal, Hexadecimal y Binario, la cantidad de espacio reservado para cada valor y despliegue de conjunto de arreglos completos.

MODIFICADORES DE FORMATOS

Formato	Descripción
?	Muestra el "Nombre de la variable = x" + un retorno de carro (CR), donde (x), representa al valor numérico decimal.
ASC ?	Muestra el "Nombre de la variable = x" + un retorno de carro (CR), donde (x), representa al valor ASCII.
DEC{1..5}	Numero decimales, opcional de 1 a 5 dígitos.
SDEC{1..5}	Numero decimales con signo, opcional de 1 a 5 dígitos.
HEX{1..4}	Numero hexadecimales, opcional de 1 a 4 dígitos.
SHEX{1..4}	Numero hexadecimales con signo, opcional de 1 a 4 dígitos.
IHEX{1..4}	Numero hexadecimales con el prefijo ($, ejemplo $FF08) opcional de 1 a 4 dígitos.
ISHEX{1..4}	Numero hexadecimales con signo y el prefijo ($, ejemplo -$FF08) opcional de 1 a 4 dígitos.
BIN{1..16}	Numero binarios, opcional de 1 a 16 dígitos.
SBIN{1..16}	Numero binarios con signo, opcional de 1 a 16 dígitos.
IBIN{1..16}	Numero binarios con el prefijo (%, ejemplo %1001001) opcional de 1 a 16 dígitos.
ISBIN{1..16}	Numero binarios con signo y el prefijo (%, ejemplo -%1001001) opcional de 1 a 16 dígitos.
STR Arreglos	Cadena ASCII desde un arreglo completo hasta que encuentre un byte = 0.
STR Arreglos\n	Cadena ASCII desde un arreglo hasta un numero (n) especificado.
REP Byte\n	Muestra un carácter ASCII (n) veces.

Los siguientes ejemplos nos despejaran cualquier duda:

Para corregir el problema anterior basta con especificar el modificador de formato (DEC).

```
numero      VAR   BYTE        ' Declaración de una variable tipo Byte
numero = 65                   ' Fijando el valor 65 en numero
DEBUG DEC numero              ' Visualizar el valor "65" por pantalla
```

```
numero      VAR   BYTE        ' Declaración de una variable tipo Byte
numero = 65                   ' Fijando el valor 65 en numero
DEBUG DEC4 numero             ' Visualizar el valor "0065" por pantalla


numero      VAR   BYTE        ' Declaración de una variable tipo Byte
numero = 65                   ' Fijando el valor 65 en numero
DEBUG ? numero                ' Visualizar "numero = 65"


numero      VAR   BYTE        ' Declaración de una variable tipo Byte
numero = 65                   ' Fijando el valor 65 en numero
DEBUG ASC ? numero            ' Visualizar "numero = A"


numero      VAR   BYTE        ' Declaración de una variable tipo Byte
numero = 65                   ' Fijando el valor 65 en numero
DEBUG DEC numero, " ", HEX numero, " ", BIN numero
' Visualiza el 65 decimal, 41 hexadecimal y 1000001 binario


numero      VAR   BYTE        ' Declaración de una variable tipo Byte
numero = 65                   ' Fijando el valor 65 en numero
DEBUG DEC numero, " ", IHEX numero, " ", IBIN numero
' Visualiza el 65 decimal, $41 hexadecimal y %1000001 binario


numero      VAR   BYTE        ' Declaración de una variable tipo Byte
numero = 65                   ' Fijando el valor 65 en numero
DEBUG DEC5 numero, " ", HEX4 numero, " ", BIN8 numero
' Visualiza el 00065 decimal, 0041 hexadecimal y 01000001 binario
```

Para los valores con signos se sigue el mismo formato:

```
numero      VAR   WORD        ' Declaración de una variable tipo Byte
numero = -65                  ' Fijando el valor 65 en numero
DEBUG "Signos.....: ",SDEC numero," ",SHEX numero," ",SBIN numero,13
DEBUG "Sin Signos.: ",DEC numero," ",HEX numero," ",BIN numero,13
```

Este código genera los siguientes resultados:

```
Signos.....: -65 -41 -1000001
Sin Signos.: 65471 FFBF 1111111110111111
```

Se puede ver que los números con formatos con signos en la línea de arriba son realmente el resultado esperado su conversión equivalente pero con signo negativo. La segunda línea los números sin signos lucen algo extraño. Esto es porque los números negativos en los BS2 son almacenados como el segundo complemento.

Mostrando cadenas de texto (Arreglos matriciales).
Si tiene un conjunto de caracteres en un arreglo matricial el formato STR, nos ayuda a desplegarlo de forma inmediata.

Ejemplo 2

```
Car     VAR     BYTE(8) ' Variable Dimensionada
Car(0) = "B"
Car(1) = "A"
Car(2) = "S"
Car(3) = "I"
Car(4) = "C"
Car(5) = " "
Car(6) = "2"
Car(7) = 0              ' Valor de control

DEBUG STR Car           ' Muestra "BASIC 2"
```

Otra forma de hacer lo mismo es:

```
DEBUG Car(0),Car(1),Car(2),Car(3),Car(4),Car(5),Car(6)
' Muestra "BASIC 2"
```

STR Arreglos
Obviamente la diferencia es considerable. El formato STR necesita encontrar un carácter de control de valor 0 decimal no "0" carácter. Este valor de control le indica el final del arreglo. Si el valor de cero (0) no es encontrado al final del arreglo, BS2 seguirá leyendo el contenido completo de toda la memoria RAM.

STR Arreglos\n
Una forma de mejorar esto es limitando él numero de caracteres a visualizar por pantalla.

```
DEBUG STR Car\5          ' "Muestra "BASIC", solo muestra 5 caracteres

DEBUG STR Car\7          ' "Muestra "BASIC 2", muestra 7 caracteres
```

REP x \n
Si necesitamos escribir una serie de cadenas de un mismo carácter, REP es la mejor vía. Donde (x) representa el carácter a visualizar y (n) es él numero de repeticiones. Supongamos que necesitemos visualizar una serie de guiones que pueden representar una línea ejemplo:

```
DEBUG "----------------------------------------"
```

Este programa imprime esta línea tal como se visualiza, lo que sucede es que tenemos 40 carácter "-" repetidos, cada carácter toma un espacio de memoria de un Byte en total perderíamos 40 Byte de programa, lo cual es muy valioso. Una forma eficiente de realizar esto es:

```
DEBUG REP "-"\40
```

Tendríamos el mismo conjunto de guiones del ejemplo anterior. Lo que sucede internamente es que REP ejecuta un bucle repitiendo (n) cantidad de veces el mismo carácter. Economizando considerablemente espacio de memoria de programación.

Códigos de Control

Hasta el momento nos hemos limitado a visualizar por pantalla textos de mensajes simples de una línea. Sin tener control absoluto de la posición del texto, limpieza de la pantalla, retornos de carro, movimientos del cursor. En fin existen una serie de códigos de control para estos fines. Los códigos de control están comprendidos entre el rango de (0 - 13) decimal. Se pueden combinar códigos de control en conjunto con caracteres y modificadores de formatos separándolos por coma (,) Ejemplo:

```
DEBUG 0, "Hola a Todos soy BS2", 13, 13
```

1. Limpia la pantalla, (0)
2. Imprime "BS2 es Genial..."
3. Retornos de Carro, (13)

Ahora pruebe con:

```
DEBUG "Hola a Todos soy BS2", 13
```

1. Imprime "Hola a Todos soy BS2"
2. Retornos de Carro, (13)

Aquí no limpiara la pantalla y solo avanzara un retorno de carro, Inicialice al BS2 con Reset.

En la siguiente tabla encontrara el significado de cada uno de los códigos de control. Se pueden realizar distintas combinaciones según sean requeridas siguiendo la regla de separación por comas (,).

CODIGOS DE CONTROL

Nombre	Valor	Efecto
CLS	0	Limpia la pantalla
HOME	1	Envía el cursor al comienzo de la pantalla
Mueve (x,y)	2	Mueve el cursor en las coordenadas: x, y
Cursor Izq.	3	Mueve el cursor a la Izquierda
Cursor Der.	4	Mueve el cursor a la Derecha
Cursor Arriba	5	Mueve el cursor Arriba
Cursor Abajo	6	Mueve el cursor Abajo
BELL	7	Emite un sonido acústico
BKSP	8	Retrocede un espacio
TAB	9	Avanza 8 espacio, similar a la tabulación
Line Feed	10	Mueve el Cursor una línea abajo
Clear Right	11	Limpia el contenido a la derecha del cursor

Clear Down	12	Limpia el contenido abajo del cursor
CR	13	Retorno de Carro, avanza a la siguiente línea

```
DEBUG 0, 2, 40,12, "BS2 es Genial...", 7,7,7

Limpia la pantalla, (0)
Mueve el cursor a las coordenadas x=40, y=12, (2)
Imprime "BS2 es Genial..."
Emite 3 sonidos acústicos, (7)
```

Funcionamiento Técnico del DEBUG

DEBUG es realmente un caso especial de la instrucción SEROUT. DEBUG establece unos parámetros fijos de comunicación por el mismo puerto serial donde se programa el BS2. Los parámetros establecidos para la comunicación son los siguientes:

1. Salida Invertida
2. Velocidad 9600 Baudios
3. Paridad Ninguna
4. Bits de Datos 8
5. Bits de Parada 1

La función,

```
DEBUG "Hola"
```

Es exactamente igual a:

```
SEROUT 16, $4054,[ "Hola"]
```

En términos funcionales ambas expresiones son iguales. Pero DEBUG toma menos espacio de programa y es mucho más fácil de escribir.

Se puede visualizar a DEBUG utilizando cualquier programa de comunicaciones como HyperTerminal de Windows, y fijando los parámetros expuestos. Pero hay que desconectar el pin 3 del BS2, que corresponde al pin 4 del conector DB9. El pin 3 del BS2 es el que prepara al chip interprete de que acepte los datos para la programación de la memoria EEPROM.

DTMFOUT

DTMFOUT Pin, { OnTime ,OffTime,} [Tone {,Tone...}]

Función

Genera un tono doble de multi-frecuencia mejor conocido como: dual-tone, multifrecuency (DTMF), es el codigo de marcado de los teléfonos de teclas.

- **Pin** puede ser variable/constante/expresión (0-15), especifica el pin para la salida de la señal. Este pin se declara temporalmente como salida durante la generación del (DTMF). Después de la generación del tono, el pin se deja en modo de entrada, incluso si era previamente una salida.

- **OnTime** es opcional puede ser variable/constante/expresión (0-65535), especifica la duración del tono en milisegundos. Por defecto este valor es de 200 ms.

- **OffTime** es opcional puede ser variable/constante/expresión (0-65535), especifica la duración de pausa de silencio en milisegundos entre cada tono, si se especifica más de un tono. Por defecto este valor es de 50 ms.

- **Tone** puede ser variable/constante/expresión (0-15), especifica los tonos DTMF a generar. Los tonos del 0 al 11 corresponden al teclado estándar del teléfono, mientras del 12 al 15 es la cuarta columna utilizado por equipos de teléfonos de prueba y radio aficionados. Corresponden a las teclas extendidas (A –D).

Explicación

DTMF se utiliza principalmente para marcar teléfonos o para controlar remotamente cierto equipos de radio que utilicen el formato DTMF. El BS2 puede generar estos tonos digitales utilizando la instrucción de DTMFOUT. La figura 8.5A demuestra cómo preamplificar esta señal y la figura 8.5B como conectar una bocina de más de 40 ohmios de impedancia para oír estos tonos; la figura 8.6 demuestra cómo conectar el BS2 directamente con la línea telefónica. Una instrucción típica de DTMFOUT de marcar un teléfono a través del pin 0 con el circuito de interfaz de la figura 8.6 parecería esto:

```
DTMFOUT 0, [5,6,8,2,2,2,2]          ' Marca el telefono [568-2222] por el pin 0
```

Esta instrucción sería equivalente a marcar 568-2222 del teclado numérico de un teléfono. Si usted desea retardar el paso de marcar para acomodar una línea telefónica ruidosa o un acoplamiento de radio, usted podría utilizar los valores opcionales del Ontime y del Offtime:

```
DTMFOUT 0,500,100,[ 5,6,8,2,2,2,2] ' Marca la Hora CODETEL lentamente.
```

En esta instrucción, el Ontime se fija a 500 ms (segundos del 1/2) y el Offtime a 100 ms (1/10 segundos).

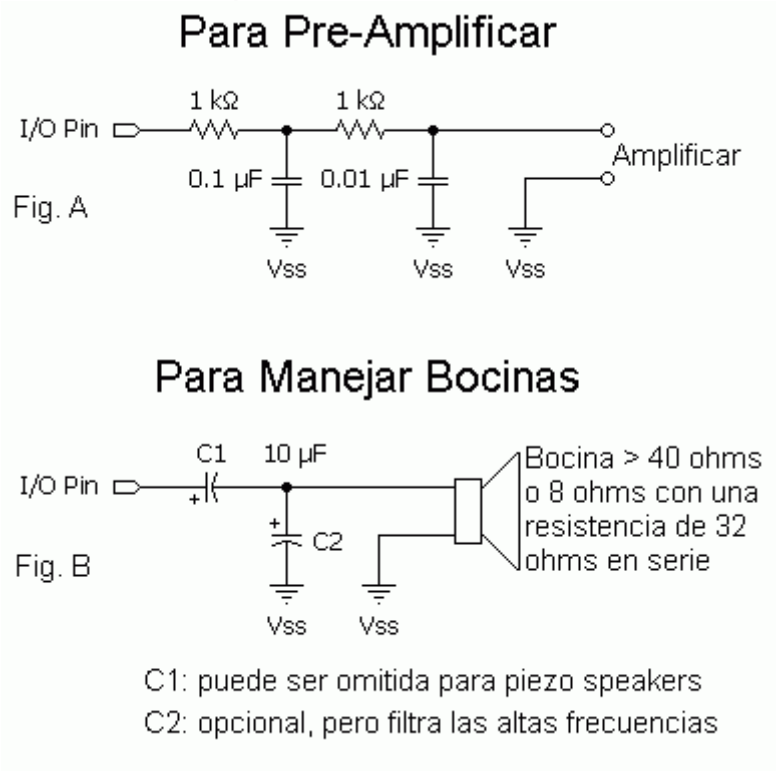


Figura 8.5

CODIGOS DTMF

Valor del Tono	Tecla correspondiente del teléfono
0 - 9	Dígitos de 0 a 9
10	Asterisco (*)
11	Numero (#)
12-15	Cuarta columna del (A – D)

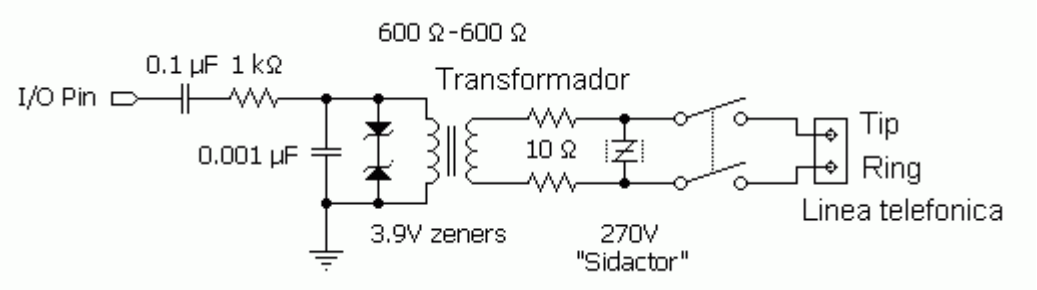


Figura 8.6

Funcionamiento Técnico del DTMFOUT

El Microcontrolador BS2 es un dispositivo puramente digital. Los tonos de DTMF son formas de onda análogas, consistiendo en una mezcla de dos ondas del seno en diversas frecuencias audio. ¿Entonces cómo un dispositivo digital genera una salida análoga? El BS2 crea y mezcla las ondas del seno matemáticamente, entonces utiliza la corriente que resulta de números para controlar el ciclo una rutina muy rápida de la modulación de pulsos o (PWM). Podemos decir que realmente lo que genera el BS2 es una corriente rápida de pulsos. El propósito de los arreglos de filtración demostrados en los diagramas esquemáticos de las figuras 8.5A e 8.5B es alisar la alta frecuencia del PWM, dejando salir solamente el audio de una frecuencia más baja detrás. Eliminándoles algunos armónicos generados.

Tenga esto presente si usted desea producir los DTMF a través del BS2 Cerciórese de filtrar el DTMF. Los circuitos demostrados aquí son solamente un punto de partida; usted puede diseñar o utilizar un filtro paso bajo activo alrededor de los 2 Khz.

	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	**1**	**2**	**3**	**A**
770 Hz	**4**	**5**	**6**	**B**
852 Hz	**7**	**8**	**9**	**C**
941 Hz	*****	**0**	**#**	**D**

Tabla de frecuencia para generar lo DTMF

Cuando usted pulsa un 1 en el teclado telefónico se produce una mezcla de frecuencia de 697 Hz y 1209 Hz. En cada tecla se combina la frecuencia de la fila con la frecuencia de la columna. Entonces cada tono audible es único.

En caso de que usted quiera interpretar estas señales, existe en el mercado una gama de circuitos integrados que se llaman DTMF Decoder.

END

END

Función

Finaliza el programa, poniendo al BS2 en modo de bajo consumo indefinidamente. El comando END se utiliza mas bien como parámetro de referencia final de un programa. Es opcional y es raramente utilizado.

Limites

Consumo normal en operación	8 mA
Consumo despues de END	40 µA

Explicación

END Pone el BS2 en modo inactivo o de bajo consumo de energía. En este modo, el consumo de corriente (excluyendo las cargas conducidas por las Entradas y Salidas) es aproximadamente de 50µA. El comando END mantiene el BS2 inactivo hasta que se reinicialaza con el botón de reset, cuando completamos un ciclo (OFF-ON) o cuando se recarga un nuevo programa.

FOR...NEXT

```
FOR Counter = ValorInicial TO ValorFinal {STEP ValorIncremento}
      {Cuerpo}
NEXT
```

Función

Crea un bucle programado entre un rango de valores iniciales y finales, el cuerpo del bucle queda comprendido en el medio de FOR y NEXT, El bucle puede incrementar o decrementar la variable Counter acorde con el ValorIncremento establecido. Si no se establece un ValorIncremento asume que el incremento será de uno (1). El bucle finaliza cuando la variable Counter llegue al ValorFinal establecido.

- **Counter** es una variable generalmente de tipo Byte o Word utilizada para el conteo del ciclo.

- **ValorInicial** puede ser variable/constante/expresión de (0-65535) que especifica el valor inicial de la variable (Counter).

- **ValorFinal** puede ser variable/constante/expresión de (0-65535) que especifica el valor final de la variable (Counter). Cuando el valor de la variable (Counter) esta fuera de rango del ValorInicial y ValorFinal, el comando FOR...NEXT detiene el bucle y ejecuta la línea siguiente después del NEXT.

- **ValorIncremento** es opcional puede ser variable o constante de (0-65535) y es el valor a incrementar o decrementar en cada iteración del bucle FOR...NEXT. Si el ValorInicial es mayor que el ValorFinal, Pbasic entiende que el valorIncremento es negativo, aunque no se utilice ningún signo de menos.

Explicación

Los FOR...NEXT se utilizan para realizar un proceso repetidas veces. Se denomina también lazo o loop controlados. El código incluido entre FOR...NEXT se ejecutara mientras se cumplan unas determinadas condiciones. Por lo general el incremento de repeticiones es de uno. El ValorIncremento es opcional y se utiliza cuando el incremento es diferente a la unidad. El siguiente ejemplo muestra como trabaja:

Ejemplo 1

```
Contador VAR BYTE       ' Definición de la variable para Counter

FOR Contador = 1 TO 5   ' Repetir el Proceso 5 Veces
  DEBUG "+"             ' (Cuerpo del ciclo), En cada repetición Imprime un "+"
NEXT                    ' Repite si contador no ha llegado a su limite

END                     ' Termina cuando Contador = 5
```

El ejemplo expuesto arriba es una aplicación típica del FOR...NEXT, abajo la equivalencia del mismo código.

```
DEBUG "+"          ' Imprime un "+" 1
DEBUG "+"          ' Imprime un "+" 2
DEBUG "+"          ' Imprime un "+" 3
DEBUG "+"          ' Imprime un "+" 4
DEBUG "+"          ' Imprime un "+" 5
```

El siguiente ejemplo Imprime por pantalla los números del 1 al 10.

Ejemplo 2

```
Valor VAR BYTE                ' Definición de la variable tipo Byte

FOR Valor = 1 TO 10           ' Repetir el Proceso 10 Veces
  DEBUG DEC2 Valor, 13        ' Imprime la Variable Contador
NEXT                          ' Repite si Valor no ha llegado a su limite

END                           ' Termina cuando Contador = 10
```

En caso de que se quiera imprimir en orden descendente del 10 al 1.

Ejemplo 3

```
Valor VAR BYTE                ' Definición de la variable para Valor

FOR Valor = 10 TO 1 STEP 1    ' Repetir el Proceso 10 Veces con un decremento de 1.
  DEBUG DEC2 Valor, 13        ' Imprime la Variable Valor
NEXT                          ' Repite si Valor no ha llegado a su limite.

END                           ' Termina cuando Contador = 1
```

En el ejemplo expuesto arriba el valorIncremento (STEP –1), realmente no es necesario, Pbasic, entiende que cuando el ValorInicial es > que ValorFinal la variable Counter se tiene que decrementar. El siguiente código tiene el mismo efecto que el anterior.

Ejemplo 4

```
Contador VAR BYTE             ' Definición de la variable para Counter

FOR Contador = 10 TO 1        ' Repetir el Proceso 10 Veces con un Decremento de 1.
  DEBUG DEC2 Contador, 13     ' Imprime la Variable Contador
NEXT                          ' Repite si contador no ha llegado a su  limite.

END                           ' Termina cuando Contador = 1
```

Es posible que tengamos un bucle de FOR...NEXT, pero que no podemos esperar a que finalice para tomar una acción. Es decir, nos salimos del bucle antes de que este finalice. El siguiente ejemplo nos muestra como realizarlo:

Ejemplo 5

```
Tiempo  VAR     BYTE          ' Definición de la variable para Counter

Main:
DEBUG CLS
FOR Tiempo = 1 TO 100         ' Repetir el Proceso 100 Veces
  DEBUG 2,5,5, DEC2 Tiempo    ' Imprime la variable Tiempo
```

```
  PAUSE 250                          ' Espera 100 mSeg.
  IF IN15=0 THEN Otro Evento         ' Evalúa la Entrada 15
NEXT                                 ' Repite hasta Tiempo=ValorFinal

Otro Evento:
  DEBUG 13, DEC Tiempo, 13
  DEBUG "Fin del Evento"             ' Imprime Fin del Evento
  PAUSE 5000                         ' Espera 5 Segundos
GOTO Main                            ' Vuelve al Inicio

END
```

En el ejemplo expuesto arriba el bucle comprendido entre FOR y NEXT, evalúa cada 250 milisegundos el Pin 15, si P15 es igual a uno (1), entonces se sale del bucle. Realmente la variable (Tiempo) se lleva el ultimo valor antes de salir del FOR...NEXT. No es necesario inicializar la variable (Tiempo), pues al inicio de cada bucle FOR...NEXT, los parámetros ValorInicial y ValorFinal son inicializado nuevamente.

Los FOR...NEXT les podrían generar un posible BUG, si no se tiene cuidado con las definiciones de las variables el siguiente ejemplo nos muestra lo que sucede cuando el ValorInicial o el ValorFinal excede el limite de la variable. Suponiendo que declaremos una variable tipo NIB, la cual contiene 16 elementos para Counter y queremos imprimir los números del 1 al 100.

Ejemplo 6

```
Repetir VAR NIB                  ' Definición de la variable para Counter
FOR Repetir = 1 TO 100           ' Repetir el Proceso 100 Veces con un Incremento de 1.
  DEBUG DEC3 Repetir,13          ' Imprime la Variable Repetir
NEXT                             ' Repite si contador no ha llegado a su limite.
DEBUG 13,"Fin del Proceso"       ' Imprime "Fin del Proceso"
END                              ' Termina cuando Contador = 100
```

En este caso usted esperaría que se imprimieran los números del 1 al 100. Pero el resultado de este ejemplo es que solo imprime del 1 al 15. y vuelve y repite infinitamente del (0 al 15) y el bucle de FOR...NEXT nunca termina, el programa entra en un "loop endlessly" o bucle infinito. Lo que sucede es que la variable (Repetir) es tipo NIB y solo contiene 16 elementos (0-15), como el ValorFinal es de 100 cuando la variable (Repetir) llega a 15 y incrementa uno su próximo valor es cero. Como la comparación interna para la terminación del bucle es que Counter > ValorFinal, esto no es posible. Realmente esto no debe suceder pues usted tiene que tener claro la cantidad de ciclos a utilizar.

Consideraciones Finales

1. Es posible anidar hasta 16 FOR...NEXT.
2. Para Incrementar ValorInicial < ValorFinal.
3. Para Decrementar ValorInicial > ValorFinal.
4. El bucle termina cuando la variable Counter es mayor que ValorFinal.

FREQOUT

FREQOUT, Pin, Periodo, freq1{, freq2 }

Función
Generan uno o dos tonos de señales senosoidal durante un periodo especificado.

- **Pin** puede ser variable/constante/expresión de (0-15), especifica el pin para la salida de la señal. Este pin se declara como salida.

- **Periodo** puede ser variable/constante/expresión de (0-65535) especifica la permanencia del tono a generar. La unidad del periodo es de un 1 milisegundo.

- **Freq1** puede ser variable/constante/expresión de (0-32767) especifica la frecuencia en hertz del primer tono.

- **Freq2** puede ser variable/constante/expresión exactamente igual que Freq1. Cuando se especifican dos frecuencias, lo que se obtiene es la mezcla de los dos tonos especificados. Freq1 y Freq2 se rigen por el mismo Periodo.

Limites

Unidad en Periodo	1 ms
Unidad en Frecuencia	1 Hz
Rango de frecuencia	0 a 32767 Hz

Explicación
FREQOUT genera dos ondas senosoidales utilizando un algoritmo de PWM rápido. FREQOUT es muy similar a DTMF incluso se puede obtener el mismo resultado. Los circuitos demostrados en la figura 8.7 filtran el PWM para suavizar los tonos altos a través de un altavoz o de un amplificador de audio. Aquí está una instrucción de FREQOUT:

```
FREQOUT 7, 1000, 2500
```

Esta instrucción genera un tono 2500-Hz por un 1 segundo (1000 ms) a través del Pin 7. Para hacer sonar dos frecuencias:

```
FREQOUT 2, 5000, 2500, 4500
```

Esta instrucción genera una mezcla de dos frecuencias: un tono de 2500-Hz y otro de 4500-Hz, por un periodo de 5 segundos (5000 ms) a través del Pin 2. Las frecuencias se mezclan juntas para generar un sonido similar a una campana. Para generar una pausa silenciosa, especifique el valor de la frecuencia a 0.

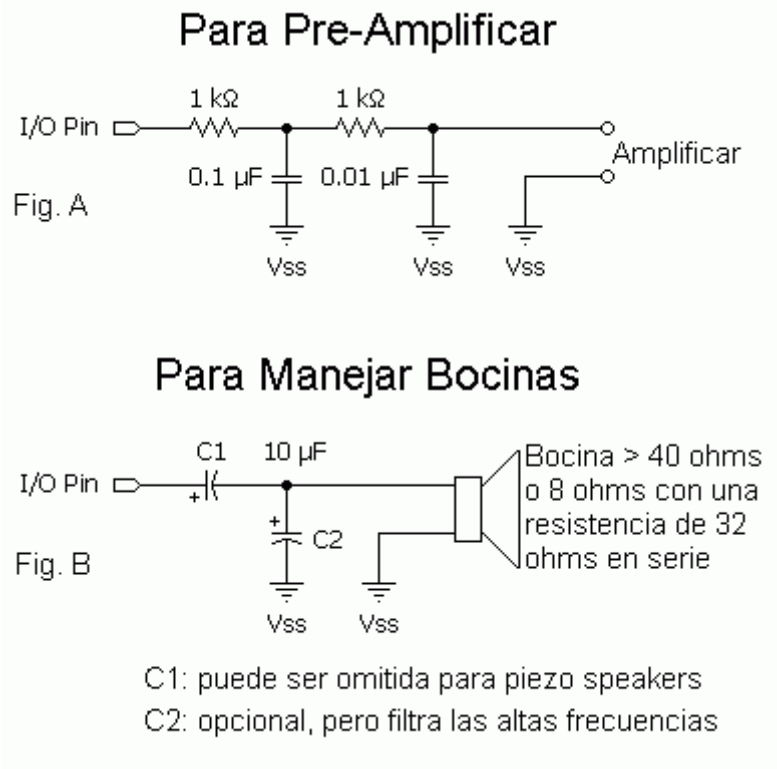


Figura 8.7

El programa siguiente toca una melodía "Mary Had a Little Lamb" leyendo las notas musicales de una tabla de valores. Para demostrar el efecto de mezclar dos ondas senosoidales, la primera frecuencia es la nota musical, mientras que la segunda es 8-Hz más baja. Cuando las ondas senosoidales se mezclan, generan una frecuencia resultante de la suma y de la diferencia. La frecuencia de la diferencia impone un envoltorio de 8-Hz antes de cada nota musical.

Ejemplo 1

```
'{$STAMP BS2}
PAUSE 1000              ' Espera un segundo

i       VAR     BYTE    ' Declara una variable tipo Byte
f       VAR     WORD    ' Declara una variable tipo Word

C       CON     523     ' Nota Musical C
D       CON     587     ' Nota Musical D
E       CON     659     ' Nota Musical E
G       CON     784     ' Nota Musical G
R       CON     8       ' Silencio

FOR i=0 TO 28  ' Ciclo controlado para reproducir las 29 notas musicales
  LOOKUP i,[E,D,C,D,E,E,E,R,D,D,D,R,E,G,G,R,E,D,C,D,E,E,E,E,D,D,E,D,C],f
  FREQOUT 15, 225,f,(f-8) MAX 32768
NEXT
END                     ' Fin del programa
```

En el siguiente ejemplo se simulan los tonos DTMF, para marcar un numero telefonico utilizando FREQOUT.

Ejemplo 2

```
'{$STAMP BS2}
PAUSE 500              ' Espera 1/2 segundo

C1              CON     1209
C2              CON     1336
C3              CON     1477
C4              CON     1633
F1              CON     697
F2              CON     770
F3              CON     852
F4              CON     941
On Time         CON     350
Off_Time        CON     100

' Marcar el numero 568-2222

FREQOUT 15, On_Time, F2,C2    ' Genera el tono 5
PAUSE Off Time
FREQOUT 15, On_Time, F2,C3    ' Genera el tono 6
PAUSE Off_Time
FREQOUT 15, On_Time, F3,C2    ' Genera el tono 8
PAUSE Off Time
FREQOUT 15, On_Time, F1,C2    ' Genera el tono 2
PAUSE Off Time
FREQOUT 15, On_Time, F1,C2    ' Genera el tono 2
PAUSE Off_Time
FREQOUT 15, On_Time, F1,C2    ' Genera el tono 2
PAUSE Off Time
FREQOUT 15, On_Time, F1,C2    ' Genera el tono 2
PAUSE Off_Time
END                    ' Fin del programa
```

En los siguientes ejemplos se muestran varios sonidos telefonicos, los cuales se pueden simular con el comando FREQOUT.

Tono de marcado:

```
FREQOUT 0,3000,350,440        ' combinar 350 Hz & 440 Hz
```

Tono de Ocupado:

```
x VAR NIB

FOR x = 1 TO 5                      ' Repetir 5 Veces
  FREQOUT 0,400,480,620             ' combinar 480 Hz and 620 Hz
  PAUSE 620                         ' Pausa de 620 milisegundos
NEXT
```

Sonido de Timbrando:

```
x VAR NIB
FOR x = 1 TO 3                      ' Repetir 3 Veces
  FREQOUT 0,2000,440,480            ' combinar 440 Hz and 480 Hz
  PAUSE 4000                        ' Pausa de 4 segundos
NEXT
```

GOSUB

GOSUB Dirección_Etiqueta

Función
Almacena la dirección de la siguiente instrucción después del GOSUB, entonces va a la Dirección_Etiqueta especificada por el GOSUB, ejecuta todos los comandos a partir de Dirección_Etiqueta y retorna cuando encuentra un comando de **RETURN.**

- **Dirección_Etiqueta** es una dirección o referencia que especifica a donde ir dentro del código escrito.

Limites

Máximos GOSUB por programas	255
Máximos GOSUB anidados	4

Explicación
GOSUB es un pariente cercano a GOTO. Después de GOSUB, el programa ejecuta el código que comienza en la etiqueta de dirección especificada. GOSUB también almacena la dirección de la instrucción inmediatamente después de sí mismo. Cuando el programa encuentra una instrucción de RETURN, la interpreta que retorne a la instrucción que sigue el GOSUB más reciente.

BS2 admite hasta 255 GOSUBs por programa, y pueden ser anidados solamente cuatros 4. Es decir, un subprograma que es la destinación de un GOSUB puede contener un GOSUB a otro subprograma, etcétera, hasta una profundidad máxima de cuatros niveles. Más de 4 niveles de anidamiento, y el programa nunca encontrará como retornar a su punto de origen.

Por cada GOSUB tiene que haber un comando RETURN, cuando se anidan GOSUBs, cada RETURN regresa a la instrucción más reciente después del GOSUB.

GOSUB es realmente conocido en muchos programas como sub-rutinas, una sub-rutina es un pequeño código, el cual por lo general se repite más de una vez en el programa. Si una serie de instrucciones se utiliza en más de un punto en su programa, usted puede conservar memoria del programa colocando esas instrucciones en un subprograma. Entonces, donde quiera que usted hubiera tenido que insertar ese código, usted puede simplemente escribir (GOSUB Etiqueta) la etiqueta es el nombre de su subprograma. Escribir subprogramas es como adicionar nuevos comandos a PBASIC.

Usted puede evitar potenciales bugs al usar subprogramas cerciorándose de que su programa no quede vagando sin ejecutar un GOSUB. En el programa ejemplo, qué sucedería si la instrucción STOP fuera eliminada? Después de que el FOR...NEXT termine, la ejecución continuara en

pickAnumber. Cuando alcance el Return, el programa saltaría nuevamente dentro al medio del bucle FOR...NEXT, porque esta fue la ultima dirección asignada por GOSUB antes de finalizar, lo que provocaría un bucle FOR...NEXT indefinido.

Ejemplo 1

```
rounds  VAR     NIB     ' Numero de repeticiones
numGen  VAR     WORD    ' Almacenamiento de numero aleatorio de 16 BITS
myNum   VAR     NIB     ' Numero aleatorio de, 1-10.

FOR ROUNDS = 1 TO 3     ' 3 Repeticiones.
  DEBUG CLS,"PICK A NUMBER FROM 1 TO 10",CR
  GOSUB PICKANUMBER     ' Sub-Rutina para obtener un numero aleatorio.
  PAUSE 2000            ' Pausa de 2 Segundos.
' Muestra el Numero.
  DEBUG "MY NUMBER WAS: ", DEC MYNUM
  PAUSE 2000            ' Otra Pausa de 2 Segundos.
NEXT                    ' Retorna al FOR
STOP                    ' Sé detiene cuando el FOR...NEXT finaliza.

'-------------------[Sub-Rutinas] -------------------
PICKANUMBER:
  RANDOM NUMGEN                 ' Genera un numero aleatorio.
  MYNUM = NUMGEN/6550 MIN 1     ' En una escala del 1 al 10.
RETURN                          ' Finaliza la Sub-Rutina y Retorna
```

Se deben evitar en las sub-rutinas los GOTO fuera del propio GOSUB, pues si se sale de una sub-rutina sin finalizar con un RETURN, este quedara pendiente, realmente cuando en el código aparece un comando RETURN este regresa a la instrucción GOSUB ejecutada más reciente.

En caso de que en un código se encuentre un comando RETURN y previamente no se haya ejecutado un GOSUB. Esto provocara un "Over Run" o que el programa vuelva al inicio. Estos errores son imprevisibles por eso se les llama BUGS, en un código sencillo las posibilidades de eventos podrían resultar impredecibles de cuantificar. Mi recomendación es que las sub-rutinas deben ser códigos cerrados que se ejecuten completo y que luego retornen. En caso de que se tenga que salir de una sub-rutina entonces tenga pendiente de finalizar con un RETURN.

Como trabaja GOSUB

Aunque cuando se escribe un código en PICBASIC no es necesario marcar la dirección de cada instrucción, solamente cuando queremos realizar una referencia especifica creamos una etiqueta de referencia. Internamente el compilador PICBASIC si marcara cada referencia o dirección. Observe el siguiente código:

```
Cont    VAR     BYTE
DEBUG CLS
DEBUG "SOY BS2", CR
Principal:
  FOR Cont=1 TO 10
    DEBUG DEC 2 CONT, CR
    NEXT
  PAUSE 500
GOTO PRINCIPAL
```

Internamente seria:

```
$000    DEBUG CLS
$001    DEBUG "SOY BS2", CR
$002    Principal:
$003      FOR Cont=1 TO 10
$004        DEBUG DEC 2 CONT, CR
$005      NEXT
$006      PAUSE 500
$007    GOTO Principal
```

Cada instrucción contiene internamente una posición física, la etiqueta "Principal", su valor es $002, cuando se ejecuta la instrucción: "GOTO Principal", realmente lo que sucede es que el puntero se va a la dirección $002, claro nosotros no tenemos acceso a visualizar los valores de las posiciones de direcciones. Esto es solo una idea de cómo trabaja el PICBASIC. En el caso del GOSUB sucede lo siguiente:

```
Coun    VAR     BYTE
PIN1    CON     1
PIN7    CON     7

$000 Principal:
$001    GOSUB Sonido
$002    DEBUG "Cargando Sistema", 13
$003    PAUSE 1000
$004    IF IN15=0 THEN S Listo
$005    GOSUB Flash_Led
$006 GOTO Principal

$007 S Listo:
$008    GOSUB Sonido
$009    DEBUG "Sistema Listo...", 13
$00A END

'---------------------[Sub-Rutinas] ----------------------
$00B Flash Led:
$00C    FOR Coun=1 TO 10
$00D      HIGH PIN1
$00E      PAUSE 250
$00F      LOW PIN1
$010      PAUSE 250
$011    NEXT
$012 RETURN

$013 Sonido:
$014    FREQOUT PIN7, 1000, 2000, 3500
$015 RETURN
```

Secuencia de funcionamiento por direcciones:

$000 Etiqueta de referencia.
$001 GOSUB Sonido, Almacena la dirección ($001+1), Luego se va a la dirección (Sonido $013).
$014 Ejecuta comando FREQOUT.
$015 RETURN, Retorna a la dirección $002.
$002 Ejecuta DEBUG "Cargando Sistema", 13.
$003 Espera 1 Segundo.

$004 Evalúa la entrada 15 si es 0 entonces se dirige a la dirección (S_Listo $007).
$008 GOSUB Sonido, Almacena la dirección ($008+1), Luego se va a la dirección (Sonido $013).
$014 Ejecuta comando FREQOUT.
$015 RETURN, Retorna a la dirección $009.
$009 Ejecuta DEBUG "Sistema Listo...", 13.
$00A END, Fin del programa.

Recuerden estas direcciones o posiciones físicas son ilustrativas y pretenden demostrar el funcionamiento del comando GOSUB. PICBASIC no admite numeración de líneas de etiquetas. Aunque internamente el las crea como referencia.

Por ultimo Normas de aplicaciones de las Sub-Rutinas:

- Un programa en PICBASIC solo admite un máximo de 255 Sub-Rutinas.
- Una Sub-Rutina puede constar de cualesquiera números de líneas y estas pueden contener cualquier comando o instrucción de PICBASIC.
- Las Sub-Rutinas deben colocarse al final del programa o después de una instrucción END.
- Al final de cada Sub-Rutina debe existir una sentencia RETURN.
- Una Sub-Rutina puede ser llamada por otra Sub-Rutina, siempre que no exceda de 4 niveles de anidamiento.

GOTO

GOTO Dirección_Etiqueta

Función
Dirige el puntero a la Dirección_Etiqueta especificada.

- **Dirección_Etiqueta** es una dirección o referencia que especifica a donde ir dentro del código escrito.

Explicación
El comando GOTO hace que el Basic Stamp ejecute el código desde el inicio a una dirección especifica. El Basic Stamp lee los códigos de PICBASIC de Izquierda a Derecha y de Arriba hacia Abajo. El comando GOTO puede saltar a cualquier dirección del código.

Un uso común para GOTO es crear bucles infinitos o cerrados; programas que repiten un grupo de instrucciones repetidamente. Por ejemplo:

```
LOOP:                          ' Etiqueta LOOP
  DEBUG "HOLA A TODOS!", CR    ' Imprime "HOLA A TODOS!" + CR
  PAUSE 500                    ' Espera ½ Segundo (500 mSeg.)
GOTO LOOP                      ' Se dirige a la Etiqueta LOOP
```

El código arriba descrito es un bucle infinito imprime "HOLA A TODOS!" Cada ½ medio segundo.

GOTO requiere una etiqueta de dirección para las destinación del puntero.

El siguiente ejemplo no es muy practico ya que no lleva una secuencia cronológica, sin embargo, se puede demostrar los diferentes saltos entre una rutina y otra.

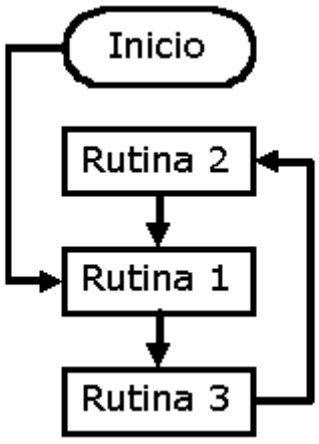


Figura 8.8 Diagrama en bloque

Ejemplo 1

```
GOTO RUTINA_1
RUTINA 2:
  DEBUG "ESTA ES LA RUTINA 2",CR
  PAUSE 1000
  GOTO RUTINA_1

RUTINA 3:
  DEBUG "ESTA ES LA RUTINA 3",CR
  PAUSE 1500
  GOTO RUTINA_2

RUTINA_1:
  DEBUG "ESTA ES LA RUTINA 1",CR
  PAUSE 500
  GOTO RUTINA_3
```

HIGH

HIGH Pin

Función
Asigna un 1 lógico al Pin especificado, esta salida es compatible con la familia lógica TTL.

- **Pin** puede ser una variable/constante/expresión de (0-15) del puerto de Entrada / salida del BS2. Este Pin se direcciona como salida automáticamente.

Explicación
El comando HIGH se utiliza para establecer una salida lógica alta de (+5V) por el pin especificado. Una vez establecido el comando HIGH este mantiene su estado indistintamente el BasicStamp realice otras tareas. Para establecer una salida lógica baja de (+0V) se utiliza el comando **LOW**.

Ejemplo 1

```
HIGH 7                          ' Ajusta el Pin7 a 5+ Voltios (Enciende el Led)
DEBUG "Comando HIGH",CR         ' Imprime "Comando HIGH"
PAUSE 1500                      ' Espera 1.5 Segundos
LOW 7                           ' Ajusta el pin7 a 0+ Voltios (Apaga el Led)
DEBUG "Comando LOW"             ' Imprime "Comando LOW"
PAUSE 1500                      ' Espera 1.5 Segundos
END                             ' Fin del Programa
```

El comando HIGH 7, es equivalente a:

```
OUT7=1              ' Ajusta la Salida 7 a 5+ Voltio.
DIR7=1              ' Direcciona el Pin 7 como salida
```

La ventaja del comando HIGH es que direcciona el pin automáticamente como salida.

IF...THEN

IF Condición THEN Dirección_Etiqueta

Función
Evalúa una condición lógica, y si es verdadera, entonces el programa se dirigirá a la dirección indicada.

- **Condición** es una evaluación o comparación lógica sobre una variable y una constante o sobre dos variables. El resultado de la evaluación puede ser falso o verdadero.

- **Dirección_Etiqueta** es la etiqueta que especifica a donde apuntara el programa en caso de que el resultado sea verdadero.

Explicación
IF...THEN es la manera que tiene PICBASIC de tomar decisiones. Si la comparación lógica evaluada resulta verdadera el programa apuntará a la dirección señalada y si es falso continua en la siguiente línea después del IF...THEN. Las comparaciones lógicas se efectúan en base a los operadores de comparación los cuales son:

Operador	Descripción
=	Igual
<>	No igual
<	Menor
>	Mayor
<=	Menor o igual
>=	Mayor o igual

Las comparaciones se escriben de la siguiente manera: Valor1 operador Valor2, los valores a comparar pueden ser entre una variable y una constante, o entre dos variables. El siguiente ejemplo nos muestra una comparación con la sentencia IF...THEN:

```
MAIN:
IF 10 > 100 THEN MAIN
END
```

Este fragmento de código compara el numero 10 con el numero 100, y nos dice que si 10 es mayor que 100 entonces que se dirija a la dirección MAIN. En este caso sabemos que el resultado lógico de esta comparación es falso pues el decimal 10 no es mayor que el decimal 100, por lo tanto saltara a la siguiente línea la cual en este caso es END. Por supuesto no tiene ningún sentido comparar dos constantes pues el resultado es siempre predecible. En el siguiente ejemplo comparamos una variable y una constante.

```
FLAG  CON   0

MAIN:
PAUSE 1000
IF IN15=FLAG THEN BOTON_ON
  DEBUG "RESULTADO FALSO",CR
  DEBUG "INTENTE DE NUEVO",CR
  PAUSE 500
  GOTO MAIN

BOTON_ON:
DEBUG "EL BOTON FUE PRESIONADO",CR
  GOTO MAIN
```

Aquí el BS2, compara el valor de la entrada 15 la cual es una variable de un BIT con la constante FLAG, si el resultado es verdadero salta a la dirección BOTON_ON, si es FALSO continua en la línea siguiente.

En el BS2, BS2e, BS2sx y BS2p, los valores a evaluar pueden ser expresiones. Esto conduce a comparaciones muy flexibles y sofisticadas. Ejemplo:

```
IF Valor < 45*100-(25*20) THEN LOOP
```

Aquí el Basic Stamp evalúa la expresión, siguiendo las reglas matemáticas de PBASIC: 45*100=4500, (25*20)=500, y 4500-500=4000. entonces la expresión seria equivalente a:

```
IF Valor < 4000 THEN LOOP
```

Es de vital importancia que toda evaluación se realice utilizando cantidades enteras positivas de 16-BIT. Utilizar cantidades con signos negativos conduce a resultados extraños. Observe lo que sucede cuando se evalúa cantidades con signo negativo.

```
IF -99 < 100 THEN Es_Menor
DEBUG "ES MAYOR QUE 100"
END

Es_Menor:
DEBUG "ES MENOR QUE 100"
END
```

Aunque (-99) es obviamente menor que 100, el programa evalúa que es mayor. El problema es que -99 internamente esta representado como el segundo complemento 65437, el cual utilizando matemáticas sin signos es mayor que 100.

IF...THEN soporta los operadores condicionales de la lógica NOT, AND, Y OR. Vea la siguiente tabla de los operadores y de sus efectos.

Condición A	NOT A
Falso	Verdadero
Verdadero	Falso

Condición A	Condición B	A AND B
Falso	Falso	Falso
Falso	Verdadero	Falso
Verdadero	Falso	Falso
Verdadero	Verdadero	Verdadero

Condición A	Condición B	A OR B
Falso	Falso	Falso
Falso	Verdadero	Verdadero
Verdadero	Falso	Verdadero
Verdadero	Verdadero	Verdadero

El operador **NOT** invierte el resultado de una condición, al cambiar falso por verdadero, y verdadero por falso. Los siguientes IF... THENs son equivalentes:

```
IF X <> 100 THEN Es_Diferente
IF NOT X = 100 THEN Es_Diferente
```

Los operadores **AND** y **OR** pueden ser usados para evaluar dos resultados a la vez. Y producir un resultado que puede ser verdadero o falso. **AND** y **OR** trabajan de la misma manera en la que cada día tomamos decisiones. El siguiente ejemplo nos muestra como trabaja utilizando **AND**, usted puede cambiar el operador **AND** por el operador **OR**.

```
Value1      VAR   BYTE
Value2      VAR   BYTE

Value1 = 5
Value2 = 9

IF Value1 = 5 AND Value2 = 10 THEN Cierto
  DEBUG "Sentencia fue Falsa"
END

Cierto:
  DEBUG "Sentencia fue Verdadera"
END
```

La condición "value1 = 5 **AND** value2 = 10" no es cierta. Aunque el value1 es 5, el value2 no es 10. El operador **AND** es verdadero cuando ambas condiciones son verdaderas.

La tabla anterior de los operadores lógicos resume los efectos de los operadores lógicos condicionales. Como con las matemáticas, usted puede alterar el orden en la cual las comparaciones y las operaciones lógicas son realizadas utilizando paréntesis. Las operaciones son normalmente evaluadas de izquierda a derecha. Poner paréntesis alrededor de una operación impone a PBASIC2 a evaluarlo antes de cualquier operación que no este en paréntesis.

Internamente, El BS2 define como "falso" si el contenido es cero (0) y verdadero si es un valor diferente de cero.

Falso	= 0	Igual a cero
Cierto	<> 0	Diferente a cero

Consideremos el siguiente ejemplo:

```
FLAG  VAR   BIT
FLAG = 1

IF FLAG THEN Es_Cierto
 DEBUG "ES FALSO"
 END

Es_Cierto:
 DEBUG "ES VERDADERO"
 END
```

Aquí como FLAG = 1 la sentencia IF...THEN evalúa que es cierto y se dirige a la dirección "Es_Cierto", el cual despliega un mensaje por pantalla de "ES VERDADERO". Pero supongamos el mismo ejemplo pero negando con la condición NOT.

```
FLAG  VAR   BIT
FLAG = 1

IF NOT FLAG THEN Es_Cierto
 DEBUG "ES FALSO"
 END

Es_Cierto:
 DEBUG "ES VERDADERO"
 END
```

Lo mas lógico aquí es que NOT FLAG, sea cero (0), y la evaluación sea falsa. Pero sorpresa el resultado sigue siendo verdadero! Que esta sucediendo aquí. Algo no funciona bien.

Internamente la sentencia IF...THEN evalúa las variables como un numero completo de 16-BITS. Aunque una variable de un BIT que contenga un uno (1), IF...THEN la visualiza como un numero de 16-BITS %0000000000000001, la negación de este valor con la función NOT es

%1111111111111110 o decimal 65534. como cualquier número diferente a cero (0) se mira como verdad, NOT FLAG es verdad. Extraño pero cierto.

La manera más fácil de evitar estas clases de problemas es utilizar siempre a un operador condicional con IF...THEN. Cambie el ejemplo anterior por:

```
FLAG  VAR   BIT
FLAG = 1

IF NOT FLAG = 1 THEN Es_Cierto
 DEBUG "ES FALSO"
 END

Es_Cierto:
 DEBUG "ES VERDADERO"
 END
```

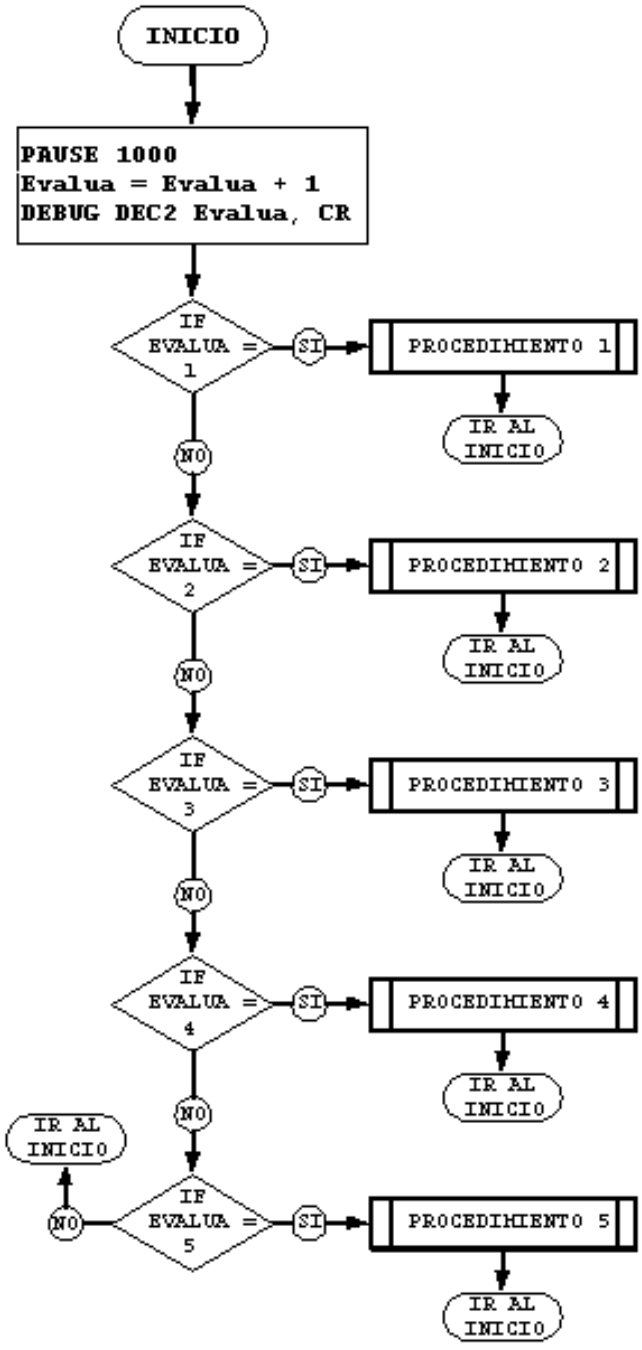
Aquí se evaluará correctamente. El operador NOT actúa de forma contraria al resultado lógico, en cuanto el operador determina si es cierto o falso entonces el operador NOT invierte el resultado.

Usted solo debe utilizar los operadores condicionales "nombrados" de la lógica NOT, AND y OR con IF...THEN. Los operadores lógicos, bitwise representados por los símbolos: ~ & | y ^ no son operadores lógicos binarios. Los ultimos se utilizan para resultados numericos.

Ejemplo 1

```
Evalua  VAR    NIB            ' Define una variable de 16 elementos

Inicio:
  PAUSE 1000                  ' Espera 1 un segundo
  Evalua = Evalua + 1         ' Incrementa uno
  DEBUG DEC2 Evalua, CR       ' Imprime el valor de la variable Evalua
  IF Evalua = 1 THEN Procedimiento_01
  IF Evalua = 2 THEN Procedimiento_02
  IF Evalua = 3 THEN Procedimiento_03
  IF Evalua = 4 THEN Procedimiento_04
  IF Evalua = 5 THEN Procedimiento_05
GOTO Inicio

Procedimiento_01:
  DEBUG "Ejecutando el Procedimiento No. 1", CR
GOTO Inicio
Procedimiento 02:
  DEBUG "Ejecutando el Procedimiento No. 2", CR
GOTO Inicio
Procedimiento_03:
  DEBUG "Ejecutando el Procedimiento No. 3", CR
GOTO Inicio
Procedimiento 04:
  DEBUG "Ejecutando el Procedimiento No. 4", CR
GOTO Inicio
Procedimiento_05:
  DEBUG "Ejecutando el Procedimiento No. 5", CR
GOTO Inicio
```

Diagrama de flujo del ejemplo anterior



INPUT

INPUT Pin

Función
Declara el Pin especificado, como modo de entrada.

- **Pin** puede ser una expresión variable o constante de (0-15) del puerto de Entrada / salida del BS2. Este Pin se ajusta como modo de entrada.

Explicación
Existen varias maneras para declarar un pin especifico como entrada. Cuando el BS2 sé inicializa el puerto completo es declarado por defecto como entrada. También la mayoría de los comandos de entradas de BS2, direcciona el puerto automáticamente como entrada como por ejemplo los comandos PULSIN y SERIN. Direccionando la variable DIRS=0 establecemos el puerto completo como entrada. O utilizando un pin especifico como por ejemplo DIR8=0, establece al pin 8 como entrada.

Para leer el puerto de entrada se utiliza la variable de entrada INS.

Ejemplo 1

```
INPUT 8                              ' Declara P8 como entrada.
Repetir Lectura:
  PAUSE 250                          ' Pausa de 250 mili-segundos.
  IF IN8=0 THEN Repetir_Lectura      ' Lee el estado de p8
    DEBUG "Fin de Lectura"           ' Imprime Fin de Lectura"
END                                  ' Fin del Programa.
```

Que sucede si en un pin, el cual ha sido declarado como entrada y se escribe intencionalmente con la variable de salida OUTS? Realmente no tiene ningún efecto sobre la entrada aunque la variable queda almacenada en OUTS. Para entender un poco mas observe el grafico del funcionamiento del Direccionamiento del BS2, en la pagina xx. Cuando el puerto es declarado como entrada (DIRS=0) aísla la salida. Sin embargo, cuando el puerto es declarado como salida (DIRS=1), las variables INS pueden leer la situación del pin de salida, si esta en estado alto o en estado bajo. El programa siguiente muestra como esto trabaja.

Ejemplo 2

```
INPUT 7
DEBUG "Estado del Pin 7: ", BIN1 IN7,CR

OUT7=0
DEBUG "Después de escribir un 0 al Pin 7: ", BIN1 IN7,CR

OUTPUT 7
DEBUG "Después de cambiar el Pin 7 como salida: ", BIN1 IN7,CR
END
```

LOOKDOWN

LOOKDOWN Target, {Comparación} [Valor0, Valor1... ValorN],Variable

Función

Busca en una lista de valores el valor Target y si coinciden, almacena su posición física en Variable. Si el valor es el primero de la lista, Variable = 0. Si es el segundo, Variable = 1 y así, sucesivamente. Si no se encuentra, no se toma ninguna acción y Variable permanece sin cambios.

- **Target** puede ser una variable/constante/expresión de (0-65535) para ser comparada con los valores de la lista.
- **Comparación** es opcional, utiliza los operadores lógicos de IF...THEN, cuando se omite, se asume que la comparación es de (=) igualdad.
- **Valores** pueden ser una variable/constante/expresión de (0-65535) y son los que se compararan con Target.
- **Variable** es una variable por general tipo (byte), y es donde se almacena el numero de posición de la lista en caso de que el target concuerde con uno de los valores de la lista. En caso de que no concuerden variable permanece intacta.

Limites

Numero máximo de Valores	256
Valor inicial de índice	0
Si el valor no esta en la lista	La Variable permanece sin cambios

Explicación

LOOKDOWN trabaja igual al índice de un libro. En un índice usted busca el tema que le interesa y cuando lo encuentra mira hacia la derecha y esta el numero de la pagina donde se encuentra el tema buscado. LOOKDOWN realiza una búsqueda sobre la base del valor de Target, comparando los valores de la lista, en caso de que encuentre una coincidencia entre target y algún valor de la lista entonces, el índice es almacenado en Variable. Por ejemplo:

```
Target      VAR   BYTE
Resultado   VAR   NIB

Target = 17
Resultado = 15
```

```
LOOKDOWN Target,[26,177,13,1,0,17,99], Resultado
DEBUG "El valor que coincide esta ubicado en la Pos.: ", DEC2 Resultado
END
```

En este caso DEBUG imprime 05, por que el valor buscado por Target es 17 en la lista de valores [26,177,13,1,0,17,99] el numero 17 se encuentra localizado en la posición 6, aunque físicamente este en la posición 6, el valor mostrado es 5. Esto sucede por que el conteo se inicia con el valor (0) cero.

Indice	Valores
0	26
1	177
2	13
3	1
4	0
5	17
6	99

Que sucede si Target no encuentra un valor en la lista? Trate de cambiar el (Target = 17 por Target = 33), como el numero 33 no se encuentra en la lista el Resultado no se ve afectado y mantiene su valor original el cual es, (Resultado = 15). En este caso DEBUG imprime 15.
Es importante inicializar a Variable con un numero diferente a (0) cero y mayor que el numero de elementos de la lista de valores. Esto es para diferenciar si hubo coincidencia en la búsqueda. Por ejemplo si la lista contiene (9) elementos Variable se puede inicializar con (15). Si tiene (50) elementos se puede inicializar con (255).

Es posible incluir caracteres ASCII en LOOKDOWN, para buscar un carácter en especifico.

Ejemplo 1

```
Target          VAR     BYTE
Resultado       VAR     BYTE

Target = "n"
Resultado = 255

LOOKDOWN Target,["Republica Dominicana"], Resultado
DEBUG "El valor igual a Target esta ubicado en la Pos.: ", DEC Resultado
END
```

DEBUG imprime, el numero 18, porque el carácter "n" esta ubicado en la posición 18 de la lista de valores. ["Republica Dominicana"].

El comando LOOKDOWN puede trabajar con otro criterio de búsqueda que no sea (Target = Valor). Hasta ahora los dos ejemplos anteriores el criterio de búsqueda eran con el operador de igualdad (=), por defecto es omitido. Pero si se quiere otro criterio de búsqueda con otro operador como por ejemplo (Target > Valor).

Ejemplo 2

```
Objetivo        VAR     BYTE
Indice          VAR     NIB

Objetivo = 17
Indice = 15

LOOKDOWN Objetivo,[26,177,13,1,0,17], Indice
DEBUG "El valor buscado esta ubicado en la Pos.: ", DEC2 Indice
END
```

Objetivo	Operador	Valores	Condición	Indice
17	>	26	Falso	0
17	>	177	Falso	1
17	>	13	Verdadero	2
17	>	1	Verdadero	3
17	>	0	Verdadero	4
17	>	17	Falso	5

Aquí DEBUG imprime 02, como 17 > 13, la condición es verdadera, LOOKDOWN finaliza cuando encuentra una condición verdadera o cuando termina de comparar el ultimo valor de la lista.

LOOKDOWN utiliza el mismo criterio de comparación que la instrucción IF...THEN, las comparaciones deben de hacerse con números positivos y la evaluación será de 16 BITS.

Operadores de Comparación

Operador	Descripción
=	Igual
<>	No igual
<	Menor
>	Mayor
<=	Menor o igual
>=	Mayor o igual

Un uso común de LOOKDOWN es usarlo en conjunción con la instrucción BRANCH para crear saltos selectivos con una simple variable de entrada:

Ejemplo 3

```
Cmd     VAR     BYTE
Cmd = "M"

LOOKDOWN Cmd,["SLMH"], Cmd
BRANCH Cmd,[ Stop, Low, Medium, High]
DEBUG "El comando seleccionado no esta en la Lista"
END

_Stop:          DEBUG "Stop"
END

 Low:           DEBUG "Low"
END
```

```
 Medium:        DEBUG "Medium"
END

_High:          DEBUG "High"
END
```

En este ejemplo, Cmd contiene a "M" (ASCII 77). LOOKDOWN encuentra que este valor corresponde el numero dos de la lista de valores, entonces lo almacena en Cmd. BRANCH ira al ítem numero dos de su lista de opciones en este caso es "_Medium".

LOOKUP

LOOKUP Índice,[Valor0, Valor1,... ValorN],Variable

Función

Recupera valores de una tabla de datos no mayor de 256 elementos, índice contiene la posición del valor a buscar en la lista, cuando índice recupera el valor lo transfiere a variable. Si el índice excede el numero de elementos la variable permanece sin cambios.

- **Índice** puede ser una variable/constante/expresión de (0-255) indica la posición del valor en la lista a recuperar.
- **Valores** pueden ser una variable/constante/expresión de (0-65535) y son los valores de la lista.
- **Variable** es una donde se transfiere el valor indicado por índice en la lista de valores. Si el índice excede el numero de elementos la variable permanece sin cambios.

Limites

Numero máximo de Valores	256
Valor inicial de índice	0
Si Indice es mayor que el numero de valores	La Variable permanece sin cambios

Explicación

LOOKUP trabaja como una tabla de valores donde el índice es lineal de (0-255) elementos y los datos pueden contener cualquier valor de (0-65535). Suponga que se tiene la siguiente tabla de valores:

índice	Valores
0	63
1	6
2	91
3	79
4	102
5	109
6	125
7	7
8	127
9	111

Para listar algún valor de la tabla el código es:

Ejemplo 1

```
índice          VAR     BYTE
Resultado       VAR     BYTE

índice = 5
Resultado = 255

LOOKUP índice,[63,6,91,79,102,109,125,7,127,111],Resultado
DEBUG "El valor correspondiente es :", DEC3 Resultado
```

En este ejemplo, DEBUG imprime 109, el cual es el valor que se encuentra en la posición No. 5, de la lista de valores.

Es posible listar caracteres ASCII, unas de las maneras de enviar información por pantallas de cristal liquido (LCD). Es con LOOKUP, observe el siguiente ejemplo:

Ejemplo 2

```
índice          VAR     BYTE
Carácter        VAR     BYTE
I_For           VAR     NIB

DEBUG CLS

FOR I For = 0 TO 25
  LOOKUP índice, ["Programar BS2 es muy Fácil"], Carácter
  DEBUG ASC? Carácter
  PAUSE 500
NEXT
END
```

En este ejemplo se imprime cada carácter de forma individual, usted puede pensar que escribiendo el código:

DEBUG "Programar BS2 es muy Fácil"

Pueden obtener lo mismo y de echo es cierto, pero cuando se trata de pantallas de cristal liquido u otra forma de enviar un conjunto de caracteres esta es la forma mas practica de realizarlo. El siguiente ejemplo nos muestra un efecto visual:

Ejemplo 3

```
índice          VAR     BYTE
Carácter        VAR     BYTE

DEBUG CLS
LOOP:
  LOOKUP índice, ["-/-\!·"], Caracter
  DEBUG 2,5,5, Caracter
  PAUSE 250
  índice = Indice+1 //5
GOTO LOOP
END
```

Un magnifico uso de LOOKUP es combinarlo con LOOKDOWN, para tablas de datos no lineales. La tabla mostrada abajo corresponde a un grupo de valores no continuos:

Valores Recibidos	Valores Equivalentes
5	16
14	17
1	18
43	24
26	10
22	12
30	11

En esta tabla de valores no lineales, es difícil de crear un algoritmo que pueda resolver este problema, el siguiente código nos ayuda a solucionarlo:

Ejemplo 4

```
Value          VAR     BYTE

LOOKDOWN, Value,[5,14,1,43,26,22,30], Value
LOOKUP, Value,[16,17,18,24,10,12,11], Value
END
```

Esta combinación de estas dos instrucciones compartiendo la variable "Value", resuelve el problema de valores no continuos. Por ejemplo en caso de que se reciba el numero 26, LOOKDOWN se encarga de buscar este valor en su lista de valores, lo encuentra en la posición numero 4, el numero 4 es almacenado en Value. En LOOKUP Value es igual a 4, LOOKUP transfiere el valor de la posición numero 4 a Value. Aquí para cada valor de la izquierda le corresponde un valor de la derecha de la tabla.

LOW

LOW Pin

Función

Asigna un 0 lógico al Pin especificado, esta salida es compatible con la familia lógica TTL.

- **Pin** puede ser una variable/constante/expresión de (0-15) del puerto de Entrada / salida del BS2. Este Pin se direcciona como salida automáticamente.

Explicación

El comando LOW se utiliza para establecer una salida lógica baja de (+0V) por el pin especificado. Una vez establecido el comando LOW este mantiene su estado indistintamente el BasicStamp realice otras tareas. Para establecer una salida lógica alta de (+5V) se utiliza el comando HIGH.

Ejemplo 1

```
LOOP:
  LOW 0               ' Ajusta el pin7 a 0+ Voltios (Apaga el Led)
  PAUSE 500           ' Pausa de ½ Segundo
  HIGH 0              ' Ajusta el Pin7 a 5+ Voltios (Enciende el Led)
  PAUSE 500           ' Pausa de ½ Segundo
GOTO LOOP             ' Retorna al Inicio
```

El comando LOW 7, es equivalente a:

```
OUT0=0      ' Ajusta la Salida 7 a 0 Voltios.
DIR0=1      ' Direcciona el Pin 7 como salida
```

La ventaja del comando LOW es que direcciona el pin automáticamente como salida. Y es más rapido.

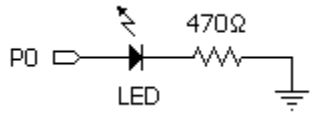


Figura 8.9 conexión de circuito

NAP

NAP Periodo

Función
El BS2 entra en modo de descanso por un periodo especificado. El consumo de energía se reduce como indica la tabla. Asumiendo que no hay cargas conectadas.

- **Periodo** puede ser una variable/constante/expresión de (0-7) especifica la duración en el que el BS2 estará en modo descanso, según la siguiente expresión: la Duración es (2^Periodo)*18ms.

Limites

Consumo de corriente en modo Normal	8 mA
Consumo de corriente en modo NAP	40 uA

Explicación
NAP utiliza el mismo mecanismo de shutdown/startup de la instrucción SLEEP, con una gran diferencia. Durante SLEEP, el BS2 compensa automáticamente las variaciones en la velocidad del "perro guardián" que sirve como su despertador. Consecuentemente, intervalos más largos de SLEEP tiene una exactitud de aproximadamente ±1 por ciento.

Periodo	Longitud de Nap
0	18 ms
1	36 ms
2	72 ms
3	144 ms
4	288 ms
5	576 ms
6	1152 ms
7	2304 ms

Los intervalos de NAP son controlados directamente por el contador de tiempo del "perro guardián" sin compensación. En la variación de temperatura, fuente voltaje y de la tolerancia de fabricación del chip interprete que puede causar una variación de sincronización -50, +100 por ciento. Como NAP usa el Watchdog Timer es independiente de la frecuencia del oscilador, es decir, un período de, (NAP 0) puede tener un rango de 9-36 ms. Con temperatura ambiente normal con batería nueva o una fuente de alimentación estable, las variaciones en la longitud de NAP serán menos que el ±10 por ciento.

Un gran uso de la instrucción NAP es cuando se utilizan al BS2 con baterías donde por lo menos una cierta cantidad de tiempo el BS2 no hace nada. Por ejemplo, usted puede tener un programa con un bucle infinito, realizando ciertas tareas, y deteniéndose brevemente por 100ms aproximadamente cada vez a través del bucle. Usted podría

sustituir la instrucción PAUSA 100 por NAP 3, mientras la sincronización de la pausa de 100 ms no era crítica. NAP 3 detendría brevemente su programa por unos 144ms y, al mismo tiempo, colocaría al BS2 en modo de baja potencia, lo que prolongaría la vida de su batería.

Recuerde la instrucción NAP no es exacta es muy similar a END cuando se ejecuta NAP todas las salidas se convierten momentáneamente en entradas por unos 18ms.

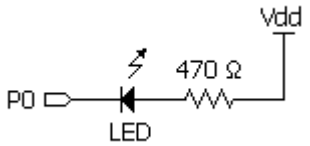


Figura 8.10 conexión de circuito

Ejemplo 1

```
'Construya el circuito de la figura anterior
LOW 0
Descansa:
  NAP 4
GOTO Descansa
```

OUTPUT

OUTPUT Pin

Función
Declara el Pin especificado, como modo de salida.

- **Pin** puede ser una variable/expresión/constante de (0-15) del puerto de Entrada / salida del BS2. Este Pin se ajusta como modo de salida automaticamente.

Explicación
Existen varias maneras para declarar un pin especifico como salida. La mayoría de los comandos de salidas de BS2, direcciona el puerto automáticamente como salida como por ejemplo los comandos PULSOUT y SEROUT. Direccionando la variable DIRS=$ffff establecemos el puerto completo como salida. O utilizando un pin especifico como por ejemplo DIR8=1, establece al pin 8 como salida.

El comando OUTPUT 8, es equivalente a: DIR8=1. Cuando se utiliza el comando OUTPUT se establece automáticamente el pin como salida.

Ejemplo 1

```
OUTPUT 4        ' Declara al Pin 4 como Salida

LOOP:
  OUT4=1        ' Estable un 1 lógico en el Pin 4
  PAUSE 500     ' Espera ½ segundo
  OUT4=0        ' Estable un 1 lógico en el Pin 4
  PAUSE 500     ' Espera ½ segundo
GOTO LOOP       ' Se dirige a la etiqueta LOOP
```

PAUSE

PAUSE Periodo

Función
Detiene el programa momentáneamente por el periodo especificado.

- **Periodo** puede ser una expresión/variable/constante de (0-65535) especifica la duración de la pausa en milisegundos.

Limites

Pausa Mínima	1 ms
Pausa Máxima	65535 ms (65.535 Segundos)

Explicación
PAUSE retrasa la ejecución del programa y cuando se cumple el periodo especificado continua con la siguiente instrucción, por ejemplo:

Ejemplo 1

```
Flash:          ' Etiqueta de referencia
  LOW 0         ' Estado lógico 0
  PAUSE 500     ' Espera ½ segundo
  HIGH 0        ' Estado lógico 1
  PAUSE 500     ' Espera ½ segundo
GOTO Flash      ' Volver a Flash
```

Los retrasos de tiempo producidos por PAUSE están basados en la precisión del cristal de 20MHz del BS2, con una precisión de ±1 un por ciento.

PULSIN

PULSIN Pin, Estado, Variable

Función

Mide el ancho de un pulso por el pin y el estado especificado y almacena el resultado en variable.

- **Pin** puede ser variable/constante/expresion de (0-15) que especifica el pin a utilizar. Este pin debe ser ajustado como entrada antes de utilizarse.

- **Estado** puede ser una variable/constante/expresion de (0-1) que especifica donde comenzara la medición del pulso, si con la transición de (0-1) el estado es (1) o con la transición de (1-0) el estado es (0). Ver grafica 8.11.

- **Variable** es donde se almacena la duración del pulso medido en unidades de 2 µS. Por lo general es de tipo WORD. Ver grafica 8.12.

Limites

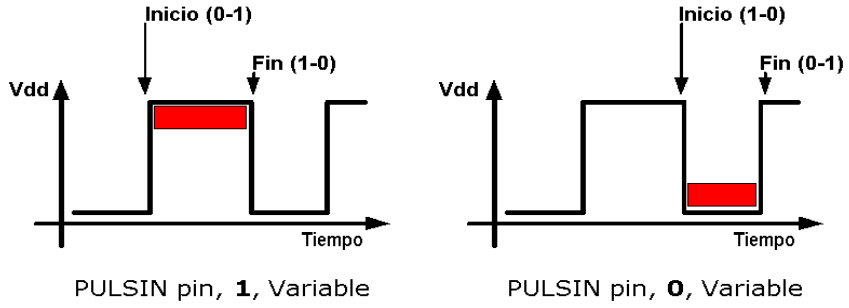
Pulso Mínimo	2 µS
Pulso Máximo	131.07 ms

Explicación

PULSIN funciona como un cronómetro rápido que es accionado por un cambio en estado de una señal (0 o 1) en el pin especificado. El ancho del pulso (alto o bajo) se mide con una resolución de 2µS y es almacenado en variable.

Muchas características análogas como (voltaje, resistencia, capacitancia, y frecuencia) se pueden medir desde el punto de vista de las duraciones de su pulso. Esto hace a PULSIN una forma valiosa de la conversión de analógico a digital.

PULSIN trabaja de la siguiente manera si el estado es (0) mide el pulso bajo. Si el estado es (1) mide el pulso alto. PULSIN esperará el pulso deseado, y comenzara a medir a partir de la transición de 0-1 o de 1-0. y finalizara de medir con otra transición de 0-1 o de 1-0. Esto se puede apreciar mejor en la grafica 8.11.



PULSIN pin, **1**, Variable PULSIN pin, **0**, Variable

Figura 8.11 diferente mediciones del pulso según su estado

Pulsin esperara la transición correcta por un máximo de 0.131 segundos, si en este tiempo no sucede nada entonces continua con la siguiente instrucción y variable retorna con un cero. De la misma manera que si el pulso medido es mayor que 0.131 segundos o 131 mili-segundo la instrucción termina y variable retorna en blanco.

Si la variable es tipo WORD, el valor devuelto por Pulsin puede extenderse a partir de 1 a 65,535 unidades de 2 µs. Si la variable es tipo BYTE, el valor devuelto puede extenderse a partir de 1 a 255 unidades de 2 µs. Sin importar el tamaño de la variable, Pulsin internamente utiliza un contador de tiempo 16-bit. Cuando su programa especifica una variable tipo BYTE, Pulsin almacena los 8 BITS más bajos del contador interno en él. Si se elige una variable tipo BYTE el pulso máximo que se podrá medir es de 510 µs equivalente a 255 unidades. En el caso de que usted elija una variable tipo BYTE y Pulsin mida un pulso de 523 µs equivalente a 261 unidades, el valor 261 decimal en binario es [00000001-00000101], pero como usted eligió una variable tipo BYTE el solo tomara los 8 BITS más bajo 00000001-00000101 , este valor sería equivalente a (5) cinco decimal. Lo cual seria un error. Es por esta razon que hay que escoger bien la variable, yo recomiendo utilizar una tipo WORD siempre.

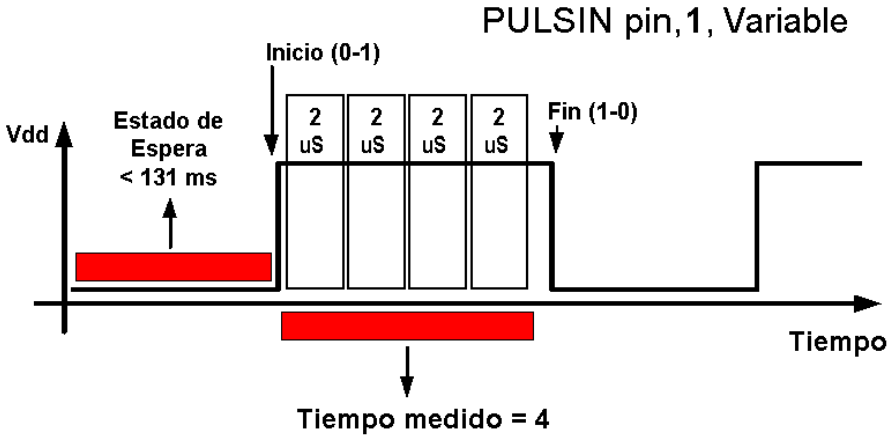


Figura 8.12 funcionamiento del contador cada 2uS

Ejemplo 1

```
pulse pin      CON     0
pulse estado   CON     1

pulse variable VAR     WORD

DEBUG CLS

Repetir:
  PULSIN pulse_pin, pulse_estado, pulse_variable
  DEBUG HOME, DEC5 pulse variable
  PAUSE 500
GOTO Repetir
```

Para calcular el tiempo real del pulso:

Tr = Variable * 2 µs ' Tiempo Real del pulso en micro-segundos.

Midiendo frecuencia a partir de un pulso

Es posible medir la frecuencia de una señal a partir de una sola muestra de un pulso, teniendo en cuenta que la señal es preferiblemente cuadrada y que su duración de ciclo es de un 50% en los pulsos altos y bajos.

Nota de cálculos matemáticos:

Tr = pulse_variable x 2 µs ' Tr : Tiempo real

Como solo se obtiene la mitad del periodo, se debe multiplicar por 2 para obtener el periodo completo en micro-segundos

P = (Tr x 2) µs

El periodo es el intervalo de tiempo entre picos sucesivos de una forma de onda periódica. Generalmente se representa por la letra mayúscula T.

La frecuencia es la cantidad de periodos que ocurren en un segundo.

F = 1/P (Hz)

Sin embargo, esto es más fácil decirlo que hacerlo, por ejemplo suponga que obtiene con la función Pulsin un resultado de (325).

Paso No. 1:
325 x 2 µs = 650 µs, este es el ancho del pulso.

Paso No. 2:
650 x 2 = 1300 µs, es el periodo, recuerde que este valor es equivalente a 0.0013 Segundos, el próximo paso seria dividir uno (1) entre 0.0013 segundos y se obtiene la frecuencia. Pero deben recordar que solo manejamos cantidades enteras. Entonces en vez de dividir 1/0.0013 segundos, podemos dividir 1,000,000/1,300 µs. Pero tenemos

otro problema la cantidad más alta que podemos manejar es de 65,535. Si nos fijamos bien podemos resumir el calculo anterior a:

F = 1,000,000/(325)x 2 x 2
F = 1,000,000/(325)x 4
F = 1,000,000/4 x (325)
F = 250,000/(325)

Esta equivalencia resulta menos compleja, pero aun seguimos teniendo problemas pues no existe la cantidad (250,000. Podemos entonces:

F = (25,000/(325))x 10

Si eliminamos un cero a la cantidad de 250,000 el resultado seria 25,000 la cual es una cantidad manejable menor de 16 Bits, luego se divide con la variable (325) y el resultado de la división le agregamos el cero multiplicando por 10. Ejemplo:

Ejemplo 2

```
pulse_pin      CON     0
pulse estado   CON     1

pulse variable VAR     WORD

DEBUG CLS

Repetir:
  PULSIN pulse_pin, pulse_estado, pulse_variable
  pulse variable = 25000/pulse variable * 10
  DEBUG HOME, DEC5 pulse variable
  PAUSE 500
GOTO Repetir
```

PULSOUT

PULSOUT Pin, Periodo

Función
Genera un pulso por el Pin y Periodo especificado.

- **Pin** puede ser variable/constante/expresion de (0-15) que especifica el pin a utilizar. Este pin debe ser ajustado como salida antes de utilizarse.

- **Periodo** puede ser variable/constante/expresion de (0-65535) que especifica la duración del pulso.

Limites

Pulso Mínimo	2 µS
Pulso Máximo	131.07 ms

Explicación
Antes de utilizar PULSOUT debe ajustar el Pin como salida, tanto la función Pulsin y Pulsout no ajustan automáticamente los estado del Pin. Por ejemplo si queremos generar un pulso de 500 µS por el Pin 15:

```
PULSOUT 15, 250    'Genera un pulso de 500 µS por el Pin 15
```

Sin embargo, esta instrucción no funcionara, ya que previamente no se ajusto el Pin 15 como salida la manera correcta de realizarlo es:

```
DIR15 = 1          'Direcciona a P15 como salida
PULSOUT 15, 250    'Genera un pulso de 500 µS por el Pin 15
```

La polaridad del pulso de salida dependerá de su estado lógico anterior, por ejemplo si p15 esta en un estado lógico bajo el pulso será positivo, si p15 esta en un estado lógico alto el pulso será negativo.

Ejemplo 1

```
LOW 15                'Declara el Pin 15 como salida y un estado lógico bajo
PULSOUT 15, 50000     'Genera un pulso de 100 mili-segundos por el Pin 15
STOP                  'Detiene el programa
```

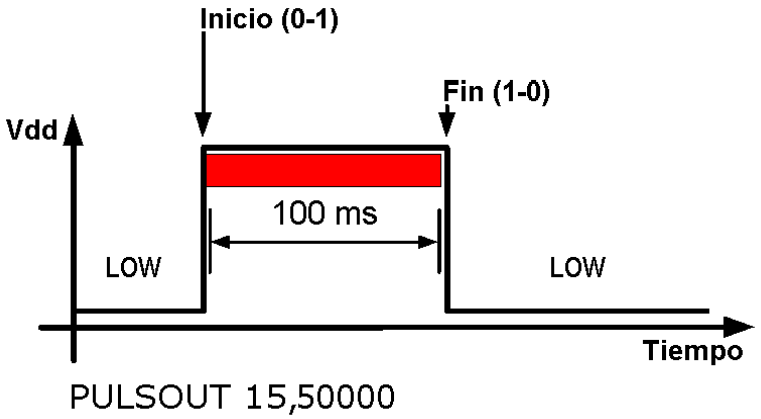


Figura 8.13 pulso generado por el ejemplo 1

Ejemplo 2

```
HIGH 15                 'Declara el Pin 15 como salida y un estado lógico alto
PULSOUT 15, 50000       'Genera un pulso de 100 mili-segundos por el Pin 15
STOP                    'Detiene el programa
```

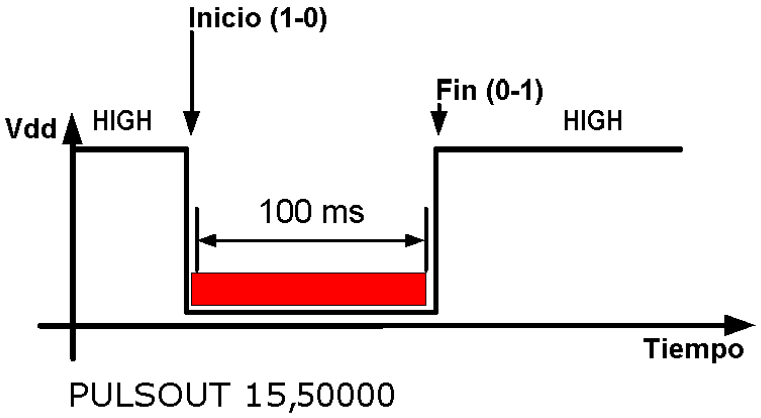


Figura 8.14 pulso generado por el ejemplo 2

La instrucción PULSOUT es equivalente a las instrucciones:

```
LOW 15              'Declara el Pin 15 como salida y un estado lógico
alto
HIGH 15             'Declara el Pin 15 como salida y un estado lógico
alto
PAUSE 100           'Espera 100 mili-segundos
LOW 15              'Declara el Pin 15 como salida y un estado lógico
alto
STOP                'Detiene el programa
```

Otra función equivalente seria:

```
DIR15 = 1          'Declara el Pin15 como salida
OUT15 = 0          'Declara la salida 15 en un estado lógico bajo
OUT15 = 1          'Declara la salida 15 en un estado lógico alto
PAUSE 100          'Espera 100 mili-segundos
OUT15 = 0          'Declara la salida 15 en un estado lógico bajo
STOP               'Detiene el programa
```

Ejemplo 3

```
DIR15 = 1              'Declara el Pin15 como salida
OUT15 = 0              'Declara la salida 15 en un estado lógico bajo

Repite:
  PULSOUT 15, 50000    'Genera un pulso de 100 mili-segundos por el Pin 15
  PAUSE 1000           'Espera un segundo
GOTO Repite            'Retorna a la línea Repite
```

PWM

PWM Pin, Duty, Ciclo

Función

Convierte la señal digital en análoga por modulación de pulsos.

- **Pin** puede ser variable/constante/expresion de (0-15) que especifica el pin a utilizar. Este pin debe ser ajustado como salida antes de utilizarse.

- **Duty** puede ser variable/constante/expresion de (0-255) que especifica la variación de voltaje análogo de (0-5V).

- **Ciclo** puede ser variable/constante/expresion de (0-255) que especifica la duración de la señal PWM.

Limites

Unidad en Ciclo	1 ms
Formula para calcular el voltaje	V = (Duty/255) * 5 Voltios
Tiempo de carga Ciclos	C = 4*R*C

Explicación

PWM Pulse-width modulation, la modulación por ancho de pulsos es capaz de generar un voltaje análogo, recuerde que con los microcontroladores solo podemos obtener dos valores discretos de (0 a 5 Voltios), cero (0v) cuando la señal es baja y cinco (5v) cuando la señal es alta.

La modulación por ancho de pulsos permite que un dispositivo puramente digital genere un voltaje análogo. La idea básica es ésta: Si usted define un Pin como salida, solo puede obtener un voltaje de 0 voltios o de 5 voltios, pero si usted logra cambiar rápidamente la salida de una señal baja a una señal alta el resultado es un tren de pulsos de modo que estarían la mitad del tiempo fuera y la otra mitad dentro, cuando se define el Duty a la mitad, es decir (128) el voltaje medio sería entre 0 y 5V (2.5V). Ésta es la idea de PWM; que usted puede producir un voltaje análogo haciendo salir una corriente de 1s digital y de 0s.

La proporción de 1s a 0s en PWM se llama el ciclo. El ciclo controla el voltaje análogo de una manera muy directa; y es directamente proporcional, mientras más alto es el ciclo más alto debe ser el voltaje de salida. En el caso del BS2, el ciclo de Duty puede extenderse a partir de la 0 a 255. El Duty es literalmente la proporción de 1s a 0s de la instrucción de PWM. Para determinar el voltaje proporcional de la salida de PWM, utilice esta fórmula: (duty/255) * 5V. Por ejemplo, si el Duty es 100, (100/255) * 5V = 1.96V; PWM hace salir un tren de los pulsos que voltaje medio es 1.96V.

Para convertir PWM en un voltaje análogo tenemos que filtrar los pulsos y almacenar el voltaje medio. La combinación de resistor/capacitor como se muestra en la figura hará el trabajo. El capacitor mantendrá por unos instantes el voltaje de PWM incluso después la instrucción haya acabado. Cuánto tiempo sostendrá el voltaje depende de cuánto corriente sea consumida por el circuito externo, y la salida interna del capacitor. Para llevar a cabo el voltaje relativamente constante, el programa debe repetir periódicamente la instrucción de PWM en un ciclo cerrado para mantener al capacitor con una carga fresca.

Así como toma tiempo el descargar al capacitor, también toma tiempo para cargarlo en el primer lugar. La instrucción de PWM le deja especificar el tiempo de carga en términos de los ciclos de PWM. Cada ciclo es un período de aproximadamente 1ms. así que cargar un capacitor de 5ms, usted especificaría 5 ciclos en la instrucción de PWM.

¿Cómo usted determina cuanto tiempo necesita para cargar un capacitor? Utilice esta fórmula: **Tiempo de Carga** = 4 * R * C. Por ejemplo, en la figura anterior una resistencia de 10k y un capacitor de 1µF.

Tiempo de Carga = 4 * R * C.
Tiempo de Carga = 4 * 10,000 * 1E-6
Tiempo de Carga = 40 ms

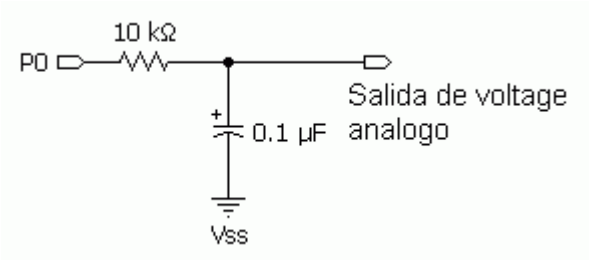


Figura 8.15 Filtro RC para la salida analoga

Puesto que cada ciclo es aproximadamente de un milisegundo, tomaría por lo menos 40 ciclos para cargar el capacitor. Asumiendo que el circuito este conectado al Pin0 la instrucción completa de PWM:

Ejemplo 1

```
Repetir:
  PWM 0, 100,40
  PAUSE 1000
GOTO Repetir
```

Particularmente yo no comparto la idea de dedicar un Microcontrolador a producir señales análogas, un microcontrolador esta diseñado para realizar ejecuciones más inteligentes, existen el mercado una gama completa de circuitos integrados que producen señales PWM y que usted la puede controlar de forma lógica con un microcontrolador.

RANDOM

RANDOM Variable

Función
Genera un numero aleatorio o al azar en un rango de (0 a 65535).

- **Variable** usualmente es de tipo WORD donde los bits internos son revueltos para generar un numero al azar.

Explicación
La función RANDOM genera un numero al azar independientemente del tipo de variable que usted elija RANDOM trabaja en función de 16 Bits o de (0 a 65535), la función RANDOM actúa sobre la variable de entrada donde toma el ultimo valor lo revuelve y el resultado lo almacena en la misma variable sobrescribiendo el valor anterior.

Esta función es muy utilizada en juegos electrónicos donde el resultado puede ser aleatorio.

Ejemplo 1

```
ALEAT  VAR    WORD    ' Define una variable tipo WORD
LOOP:                 ' Etiqueta de referencia LOOP
  RANDOM ALEAT        ' Genera un numero aleatorio y lo almacena en ALEAT
  DEBUG DEC5 ALEAT,CR ' Imprime el numero ALEAT
  PAUSE 250           ' Espera una Pausa de 250 milisegundos
GOTO LOOP             ' Vuelve a LOOP
```

RCTIME

RCTIME Pin, Estado, Variable

Función

Mide el tiempo en el cual se carga o descarga un capacitor conformado por un circuito (RC) Resistor y capacitor, el que se mida la carga o descarga del capacitor dependerá del parámetro Estado el cual puede ser (0-1).

- **Pin** puede ser variable/constante/expresión de (0-15) que especifica el pin a utilizar. Cuando finaliza la función este Pin se convierte en entrada.

- **Estado** puede ser una expresión/variable/constante de (0-1) que especifica si la medición será realizada durante la carga o descarga del capacitor a medir.

- **Variable** es donde se almacena la duración del tiempo medido en unidades de 2 µS. Por lo general es de tipo WORD.

Limites

Medición Mínima	2 µS
Medición Máxima	131.07 ms

Explicación

RCTIME es una función similar a PULSIN y se fundamenta en el tiempo que toma un capacitor en descargarse o cargarse. En otras palabras esta función es capaz de medir capacitancia o resistencia de elementos desconocidos, que podrían ser: Sensores de Humedad, Sensores de Presión, Sensores de Temperaturas, en fin la mayoría de los sensores se fundamenta sobre la base de un cambio en su resistencia o en su capacitancia. Si queremos medir elementos resistivos fijaremos una capacitancia de un valor conocido, si se requiere medir elementos capacitivos se fija un elemento resistivo de un valor conocido. Realmente la función RCTIME utiliza el principio de medición de voltaje de los A/D o conversores analogos/digitales.

Cuando se ejecuta la función RCTIME, esta comienza un conteo cada 2 µS, desde su valor inicial hasta su valor final, el caso de los BS2 el valor inicial comienza sobre los 1.5 Voltios y el valor final concluye cercano a los 5.0 Voltios, esto se puede apreciar mejor en la siguiente grafica:
Cuando se decide utilizar RCTIME hay que tener en cuenta el tiempo de carga del circuito RC, este nunca debe exceder los 131 ms, en caso de que el tiempo exceda el resultado en la Variable será cero (0). Esto realmente no es problema pues solo hay que combinar los elementos correctos para que el tiempo no exceda de los 131 ms.

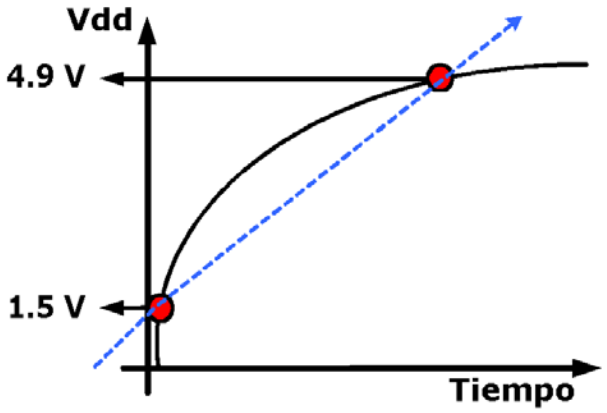


Figura 8.16, grafica de carga de un circuito RC

Existen dos tipos de Estado, como se muestra en la figura 8.17, el Estado a elegir dependerá de la situación del circuito, pero el fabricante aconseja el circuito de la Fig(A).

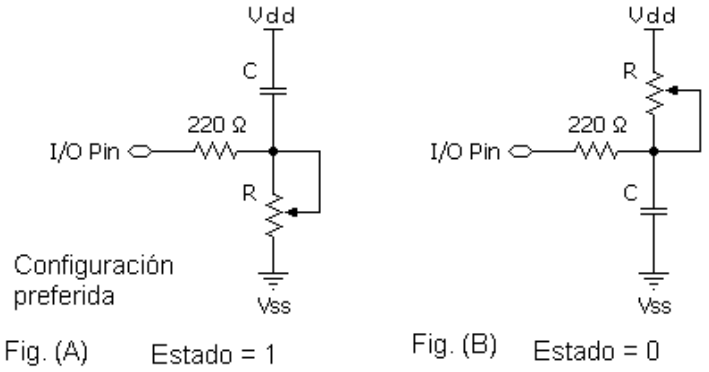


Figura 8.17, circuito de conexión tipica

Antes de medir la carga del capacitor hay que descargarlo totalmente. Esto tampoco es un problema pues con dos simple instrucciones esto se logra como se puede apreciar en el siguiente ejemplo:

Ejemplo 1

```
Resultado VAR  WORD     'Se define una variable tipo WORD
HIGH 15                 'Se descarga el capacitor polarizándolo (+5)
PAUSE 1                 'Espera un milisegundo para asegurar la descarga
RCTIME 15,1, resultado 'Inicia la función RCTIME por el pin 15
DEBUG DEC5 resultado    'Imprime el resultado del tiempo
```

En general la formula para calcular la carga o descarga de un circuito (RC) es t = R.C, donde t: es el tiempo en segundo, R: el valor de la resistencia y C: el valor de la capacitancia.

Si por ejemplo tenemos una resistencia de un valor de 10K y un capacitor de 2.2µF, el tiempo en cargarse seria:

t = R x C
t = 10K x 2.2µF
t = $10E^{+3}$ x $2.2E^{-6}$
t = 0.022 Segundos
t = 22 mili-segundos

Esta es la formula general para la carga total desde un valor inicial de (0 Voltios) hasta un valor final de la fuente de alimentación. Pero que sucede si queremos tomar un valor inicial y un valor final y, además, tenemos en cuenta que esta carga no es una función lineal sino mas bien una función logarítmica, entonces la formula resulta ser un poco más científica:

$$tiempo = -\tau\left[\ln\left[\frac{Vfinal}{Vinicial}\right]\right]$$

Donde:

tiempo:	En segundos
τ :	R x C
Vfinal:	5.0 Voltios
Vinicial:	1.5 Voltios

$$\ln\left[\frac{Vfinal}{Vinicial}\right] \;:\; \ln\left[\frac{5.0}{1.5}\right] \;:\; \ln[3.33] \;:\; [1.2039]$$

Simplificando tenemos la formula reducida a:

Tiempo = $-\tau$ x 1.2039
Tiempo = (R x C) x 1.2039

Calculando el ejemplo anterior con una resistencia de 10K y un capacitor de 2.2µF, el tiempo resultante seria:

Tiempo = (10k x 2.2µF) x 1.2039
Tiempo = (0.022) x 1.2039
Tiempo = (0.022) x 1.2039
Tiempo = 26 mili-segundos

En el BS2 26 milisegundos esta por debajo del limite máximo de 131 milisegundos, realmente el resultado en variable es el siguiente:

Tiempo = 26 mili-segundos
Tiempo = 26487 µS

Como el contador mide cada 2 µS el resultado en Variable es T/2µS

Variable = 26487 µS / 2 µS
Variable = 13244

Este resultado es el que obtendríamos con una Resistencia de 10K y un Capacitor de 2.2 µF, nótese que el valor (13244) esta en función de un tiempo, no de un valor de capacitancia o resistencia para convertir este valor a su equivalente en resistencia o capacitancia es otra historia.

Res Kohm	Cap µF	Resultado
10	2.20	13244
10	2.10	12642
10	2.00	12040
10	1.90	11438
10	1.80	10836
10	1.70	10234
10	1.60	9632
10	1.50	9030
10	1.40	8428
10	1.30	7826
10	1.20	7224
10	1.10	6622
10	1.00	6020
10	0.90	5418

Fijando una Resistencia de un valor conocido de 10K y un capacitor variable en un rango de (2.2 µF a 0.90 µF), se puede apreciar en la tabla de la izquierda el rango de variación, el cual es directamente proporcional al valor de la capacitancia, a mayor capacitancia mayor es el tiempo obtenido. En caso de que se requiera imprimir el valor real del capacitor bastara con dividir el resultado de variable con el valor real del capacitor en otras palabras 13244/2.2 = 6019.

En la practica esto se conoce como calibración, recuerde este valor de la constante 6019 es solamente para este caso en especifico. En la practica se recomienda obtener por lo menos 4 valores y medir con un equipo de precisión el componente variable.

En la tabla siguiente se puede apreciar que la división de la variable resultado con el valor real del capacitor en cada caso de se obtiene una constante, en la practica esta constante puede variar el promedio de variaciones es entonces la constante, para luego con cálculos matemáticos obtener el valor real del dispositivo desconocido. En el caso de los Sensores, los fabricantes suministran una tabla de valores de variación de capacitancía en función del elemento a medir.

Res Kohm	Cap µF	Resultado	Constante
10	2.20	13244	6019.8
10	2.10	12642	6019.8
10	2.00	12040	6019.8
10	1.90	11438	6019.7
10	1.80	10836	6019.7
10	1.70	10234	6019.7
10	1.60	9632	6019.7
10	1.50	9030	6019.7
10	1.40	8428	6019.6

10	1.30	7826	6019.6
10	1.20	7224	6019.6
10	1.10	6622	6019.5
10	1.00	6020	6019.5
10	0.90	5418	6019.4

En el caso en que fijemos una resistencia de 10K, cual seria el valor máximo de capacitancia que se puede medir con el BS2 sin que sobre pase el limite de 131 ms?

Tiempo = (R x C) x 1.2039
C = Tiempo / 1.2039 x R
Tiempo máximo = 131 ms
Tiempo máximo = 0.131 s
C = 0.131 / 1.2039 x R
C = 0.131 / 1.2039 x 10K
C = 0.131 / 1.2039 x 10000
C = 0.131 / 12039
C = 10 µF

Con un capacitor de 10 µF y una resistencia de 10 K el resultado en variable es de 60199, en otras palabras muy cercano a 65535, pero por debajo.

Entonces si queremos medir capacitancia de 100 µF que podemos hacer?

Muy sencillo reducir el valor de la resistencia por un valor en que el tiempo de carga sea menor a 131 ms. Por ejemplo si fijamos una resistencia de 1 K y un capacitor de 100 µF el resultado en variable es de 60199. Un valor bastante aceptable.

La resistencia de 220 ohms

Otro punto muy importante lo representa la resistencia de 220 ohms, la cual desempeña un papel de protección contra sobre corriente, recuerde que antes de medir la carga del capacitor hay que descargarlo totalmente, esto se realiza con una función de salida HIGH, cuando se descarga un capacitor la corriente de descarga tiende a infinito, en otras palabras se produce un cortocircuito por un periodo muy breve de tiempo pero que afectaría al microcontrolador, recuerden el BS2 solo puede manejar unos 25 mA por salida. Entonces la resistencia de 220 ohms impide esto limitando la corriente de salida:

I = V / R
I = 5 / 220
I = 22.7 mA

Esta resistencia no afecta en lo mínimo al circuito RC y tampoco durante el proceso de medición de voltaje de todo modo recuerde que el BS2 cuando esta en modo de lectura la impedancia es extremadamente

alta. Sin embargo, a la hora de descargar el capacitor si hay que tomar en cuenta el tiempo que tardaría en descargarse completamente, para iniciar con una lectura desde cero.

Para calcular el tiempo de descarga segura:

TDS = 4 x R x C
TDS = 4 x 220 x C

En el caso de que sé este trabajando con un capacitor de 100 µF

TDS = 4 x 220 x 100 µF
TDS = 88 ms

Entonces el código seria así:

Ejemplo 2

```
Resultado VAR  WORD     'Se define una variable tipo WORD
HIGH 15                 'Se descarga el capacitor polarizándolo (+5)
PAUSE 100               'Espera 100ms para asegurar la descarga
RCTIME 15,1, resultado 'Inicia la función RCTIME por el pin 15
DEBUG DEC5 resultado    'Imprime el resultado del tiempo
```

Es posible también con la función RCTIME medir el retardo de los contactos de un relay después de energizarlo, como se muestra en el siguiente ejemplo:

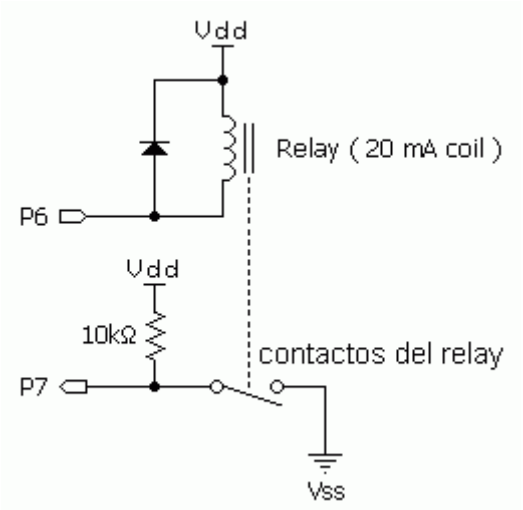


Figura 8.18 conexión tipica para medir el tiempo de respuesta de un relay

Con la siguiente configuración es posible determinar rapidez con que un relay responde cuando es energizado, aunque este dato es suministrado por cada fabricante de relay, es muy importante poder comprobarlo, en este ejemplo es un relay típico de polarización de 5Vdc y un consumo de corriente de 20mA, el cual se puede energizar directamente con el BS2 a través de P6, como lo muestra la figura de la izquierda, una vez

energizado los relay mecánicos tardan un tiempo en cerrar los contactos este tiempo puede estar en el rango de (70 ms a 900 ms). Con el siguiente código se puede determinar en mili-segundos el tiempo de cierre de un relay. Típico de 5Vdc/20mA, marca Radio Shack 275-232. Nótese la polarización en los contactos con la resistencia de 10K.

Ejemplo 3

```
tiempo c VAR   WORD           'Se define una variable tipo WORD

Principal:
  LOW 6                       'Energiza el relay
  RCTIME 7,1, tiempo_c        'Inicia la función RCTIME por el pin 7
  tiempo c = (tiempo c/1000)*2
  DEBUG DEC5 tiempo c,"ms"    'Imprime el resultado del tiempo
  HIGH 6                      'Se desenergiza el relay
  PAUSE 1000                  'Espera 1s
GOTO Principal                'Vuelve a Principal
```

READ

READ Localización, Variable

Función
Lee de la EEprom interna del BS2 un dato tipo BYTE previamente almacenado con la función DATA o con la función WRITE.

- **Localización** puede ser una variable/constante/expresión de (0-2047) que indica la dirección en la memoria EEprom del BS2, la memoria EEprom del BS2 contiene 2048 direcciones en la que puede almacenar un BYTE por dirección.

- **Variable** es donde se almacena el dato leído. Por lo general es de tipo BYTE.

Limites

Dirección inicial	0
Dirección final	2047

Explicación
El BS2 utiliza una memoria EEprom de 2K BYTE para almacenar las instrucciones del programa. Por lo general esta memoria no es ocupada totalmente por las instrucciones de programación. Es posible utilizar la parte no ocupada para almacenar: cadena de texto, tabla de datos, clave de acceso, números telefónicos en fin cualquier dato tipo BYTE de (0-255), puede ser almacenado en la memoria EEprom del BS2.

Si la memoria es utilizada para almacenar las instrucciones de un programa, como no interfiere con los datos. Esto es posible gracias a que las instrucciones se almacenan en la memoria EEprom de abajo hacia arriba, es decir, de la dirección 2047 a la dirección 0 y la data se almacena de la dirección 0 hacia la dirección 2047. De esta manera se logra que las instrucciones de programación no interfieran con los datos almacenados. Obviamente esto es posible siempre que el programa no ocupe toda la porción de la memoria EEprom. Para visualizar el contenido de la memoria en tiempo de programación es posible con el editor de pbasic presionando la combinación de teclas [Ctrl+M].

Con la función READ es posible leer todo el contenido de la memoria EEprom desde la dirección 0 a la dirección 2047, incluyendo el espacio de la programación.

La función READ se complementa con la función DATA, DATA almacena la información durante la programación, es decir, los datos quedan almacenados cuando se recarga el programa. Una vez recargado el programa los datos permanecerán definitivamente o hasta que se sobrescriba nuevamente con un programa nuevo. Es conveniente recordar las características de las memorias EEprom de conservar la información por más de 10 años una vez grabadas aun sin energía y la capacidad de

borrarlas y escribirlas nuevamente hasta 10,000,000 de veces sin alterar sus propiedades físicas.

Si por ejemplo usted tiene la siguiente cadena de texto: "BS2 es lo máximo"
Para almacenar esta cadena necesitaríamos de una variable de 16 BYTES para almacenar un dato por cada carácter. Como se muestra en el siguiente ejemplo:

```
caracter       VAR    BYTE(16)       'Declaración de una variable múltiple

carácter(0) = "B"
carácter(1) = "S"
carácter(2) = "2"
carácter(3) = " "
carácter(4) = "e"
carácter(5) = "s"
carácter(6) = " "
carácter(7) = "l"
carácter(8) = "o"
carácter(9) = " "
carácter(10) = "m"
carácter(11) = "a"
carácter(12) = "x"
carácter(13) = "i"
carácter(14) = "m"
carácter(15) = "o"

DEBUG CLS, STR carácter,CR           'Imprime el mensaje "BS2 es lo máximo"
END
```

Definitivamente esta no es la mejor forma de realizarlo, además, que no es nada practico el desperdiciar memoria RAM para datos. Con la instrucción DATA y READ solucionamos este problema:

Ejemplo 1

```
indice         VAR    BYTE    ' Variable para la localización
caracter       VAR    BYTE    ' Variable para almacenar el contenido

DATA   "BS2 es lo máximo"

FOR indice = 0 TO 15           ' Direcciona de posición 0 a 15 de la EEprom
  READ indice, carácter        ' Lee el contenido de la dirección
  DEBUG carácter               ' Imprime el contenido leido
NEXT
END
```

El siguiente ejemplo imprime por el DEBUG un mapa de memoria similar al del editor del BasicStamp 2.

Ejemplo 2

```
'Generar un mapa de memoria

DATA "EL BS2 ES FACIL"
DATA "ES MUY RAPIDO"
DATA "MUCHOS PIENSAN QUE ES UN JUGUETE"
DATA "USTED APRENDE MEJOR CUANDO SE DIVIERTE"
DATA "ES LA MEJOR MANERA DE COMENZAR"
```

```
DATA "A ESTUDIAR EL MUNDO DE LOS"
DATA "MICROCONTROLADORES"
DATA "ANTES EL TEMA DE LOS MICROCONTROLADORES"
DATA "ESTABA RESERVADO PARA UN GRUPO DE CIENTIFICOS"
DATA "HOY PUEDE LLEGAR A PERSONAS COMO USTED"
DATA "LOS BS2 NOS HACEN QUE LA ELECTRONICA SEA MAS FACIL"
DATA "PUEDE DESARROLLAR GRANDES PROYECTOS"
DATA "CON MUY POCOS COMPONENTES EXTERNOS"
DATA "REALICE SU PROYECTO Y PRESENTELO DE FORMA INMEDIATA"
DATA "IMPRESIONE A SU CLIENTE"
DATA "CLARO SU CLIENTE NO DEBE SABER QUE ES FACIL PARA USTED"
DATA "HAGA QUE SUS EQUIPOS ADQUIERAN PERSONALIDAD"

index 1 VAR    WORD
index 2 VAR    BYTE
index 3  VAR   WORD
carac   VAR    BYTE

FOR index_1 = 0 TO 2047 STEP 16
  DEBUG HEX3 index 1," "
  FOR index 2 = 0 TO 15
    index 3 = index 1 + index 2
    READ index_3, carac
    DEBUG HEX2 carac," "
  NEXT
  DEBUG CR
NEXT

END
```

Para más información consulte el comando función DATA.

Información general de las memorias EEprom

Se trata de memorias de sólo lectura, programables y borrables eléctricamente EEPROM (Electrical Erasable Programable Read Only Memory). Tanto la programación como el borrado, se realizan eléctricamente. Son muy cómoda y rápida la operación de grabado y la de borrado. No disponen de ventana de cristal en la superficie como las Eprom tradicionales. Las cuales se borraban con una luz ultravioleta.

Las memorias EEPROM, pueden grabarse y borrarse hasta un 1,000,000 de veces sin alterar su estructura física, esto lo hace sumamente útil durante el desarrollo de una aplicación.

Su gran flexibilidad y rapidez a la hora de realizar modificaciones en el programa de trabajo. El número de veces que puede grabarse y borrarse una memoria EEPROM es finito, por lo que no es recomendable una reprogramación continua. Son muy idóneas para la enseñanza y la Ingeniería de diseño.

Se va extendiendo en los fabricantes la tendencia de incluir una pequeña zona de memoria EEPROM en los circuitos programables para guardar y modificar cómodamente una serie de parámetros que adecuan el dispositivo a las condiciones del entorno.

La acción de retener la información aun sin energía constituye la forma más practica para almacenar información, es lo mas parecido a almacenar información en medios magnéticos pero aun mejor.

RETURN

RETURN

Función
RETURN es el complemento de la función GOSUB, RETURN indica la finalización de una sub-rutina.

Explicación
La función RETURN es un indicativo de que la sub-rutina más reciente invocada por GOSUB termina y retornar a la línea próxima después del ultimo GOSUB invocado. Para obtener una idea mas clara vea la función GOSUB.

El siguiente ejemplo nos muestra una sub-rutina simple en la que se imprime un mensaje a través del DEBUG.

Ejemplo 1

```
DEBUG CLS
GOSUB Imprime mensaje
Principal:
  PAUSE 1000
  GOSUB Demo1
  PAUSE 1000
  GOSUB Demo2
  PAUSE 1000
  DEBUG "Fin del Programa"
END

'-------[SUB-RUTINAS]---------------------------
Demo1:
  DEBUG "Ejecutando demostración No. 1",CR
  RETURN

Demo2:
  DEBUG "Ejecutando demostración No. 2",CR
  RETURN

Imprime mensaje:
DEBUG "Iniciando Programa",CR
  RETURN
```

REVERSE

REVERSE Pin

Función
Invierte la dirección de Entrada/Salida de un pin especificado.

- **Pin** puede ser una variable/constante/expresión de (0-15) del puerto de Entrada/salida del BS2. Este Pin se direcciona opuestamente a su estado lógico anterior.

Explicación
La función REVERSE es la mejor manera de invertir una dirección de un Pin especifico. Si el Pin esta declarado como entrada, una vez se aplica REVERSE este se convierte en salida. Si esta declarado como salida una vez que se aplique REVERSE se convierte en entrada.

Recuerde las "Entrada" realmente tienen dos significados:

1- Poner un Pin como entrada hace posible el controlar el estado (1-0) de un circuito externo conectado a este Pin. El estado de este Pin se almacena en el registro INS.
2- Poner un Pin como entrada desconecta la salida la cual es manejada por el registro OUTS.

El programa de abajo muestra el segundo punto descrito con dos tonalidades de un LED parpadeando.

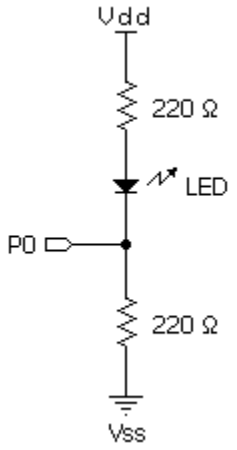


Figura 8.19 conexión para el ejemplo descrito abajo

```
' Program: REVERSE.BS2

' {$STAMP BS2}         ' STAMP directive

OUT0 = 0               ' Pone en low el Pin 0.

Again:
  PAUSE 500            ' Espera (1/2 segundo).
  REVERSE 0            ' Invierte la dirección del Pin 0.
GOTO Again             ' Repite para siempre.
```

El siguiente ejemplo muestra una función equivalente a REVERSE, en mi caso particular nunca he utilizado dicha función, entiendo que si quiero cambiar de dirección lo realizo con DIR.

Ejemplo 2

```
' {$STAMP BS2}         ' STAMP directive

DIRS = $FFFF           ' Puerto completo declarado como salida
OUTS = $0000           ' Puerto completo en estado de LOW

DEBUG BIN16 DIRS,CR    ' Imprime el estado del puerto
PAUSE 1000             ' Espera 1 segundo

Main:                  ' Etiqueta de referencia
  DIR0 = ~ DIR0        ' Invierte la dirección
  DEBUG BIN16 DIRS,CR  ' Imprime el estado del puerto
  PAUSE 1000           ' espera 1 segundo
GOTO Main              ' Vuelve a Main
```

SERIN

SERIN Rpin{\Fpin},Baudmode,{Plabel,}{Timeout,Tlabel,}[InputData]

Función

Recibe uno ó más datos en el Pin especificado en formato estándar asincrónico RS-232.

- **Rpin** puede ser variable/constante/expresión de (0-16) que especifica el pin a utilizar. rpin es colocado como entrada en forma automática. Si se especifica rpin a 16, quiere decir que se utilizara el puerto de programación. Es decir, el que utiliza el DEBUG.

- **Fpin** puede ser variable/constante/expresión de (0-15) es opcional. Se utiliza si se quiere establecer una comunicación con control de datos, muy importante para comunicación de microcontroladores entre sí.

- **Baudmode** puede ser variable/constante/expresión de (0-65535) especifica la velocidad de transmisión y configuración. Ver tabla de configuración.

- **Plabel** es un parámetro opcional indica en caso de que ocurra un error, en realidad Plabel es una etiqueta de referencia. Si ocurre un error saltara al nombre de la etiqueta. Este argumento solo puede utilizarse si el Baudmode es 7 bits, y paridad par.

- **Timeout** es un parámetro opcional, puede ser variable/constante/expresión de (0-65535) le indica a SERIN que si en el tiempo establecido por Timeout en milisegundos no arriban los datos, entonces salta a Tlabel.

- **Tlabel** es un parámetro opcional, que funciona con Timeout, Tlabel es una etiqueta de referencia, que indica que los datos no arribaron, en el tiempo establecido por Timeout.

- **InputData** es una lista de variables que serán recibidas a través del puerto serial RS-232. puede ser una variable o un arreglo de variables en forma matricial. De formato de texto, decimal, binario o hexadecimal

Explicación

El BS2 puede enviar o recibir datos serial asincrónico a una velocidad de 50,000 bits por segundo. Realmente SERIN es uno de los comandos más extensos y completos del BS2, el BS2 maneja con naturaleza la comunicación serial de hecho cuando usted trabaja con el comando DEBUG sé esta comunicando de forma Serial asincrónica.

El BS2 puede recibir comunicación RS-232 en cualquiera de sus 16 pines de (0-15), pero es posible recibir información a través del puerto de programación. El cual es Rpin = 16, Esto es muy conveniente. Pues no tenemos que sacrificar un pin extra para comunicarnos, recuerden que una vez que el BS2 esta programado el puerto de comunicación queda disponible.
La posible desventaja de utilizar el puerto de programación Rpin = 16 es que la velocidad y el formato no permiten variaciones.

Modo de operación del RS-232

Eléctricamente el formato de trasmisión serial RS-232 funciona de manera digital con dos valores lógicos donde 0 representa -12 voltios y 1 representa los +12 voltios. Esto es a diferencia de lo acostumbrado de 0 a 5 voltios. Esto fue creado con la finalidad de poder mantener dos señales de valores distantes y poder compensar la caída de voltaje en el cableado. El problema consiste en que los microcontroladores manejan señales de 0 a 5 voltios y no de -12 a +12 voltios, por lo que se recurre a un circuito de acoplamiento de señal. El puerto de programación contiene un circuito de acoplamiento de señal a base de transistores. Es posible utilizar también un circuito integrado de acoplamiento de señal como el MAX232.
Otra forma razonable de resolver esto es colocar una resistencia en serie de 22K al pin a utilizar del BS2, esta resistencia provoca una caída de tensión y protege al BS2 de los voltajes de -12 y +12.

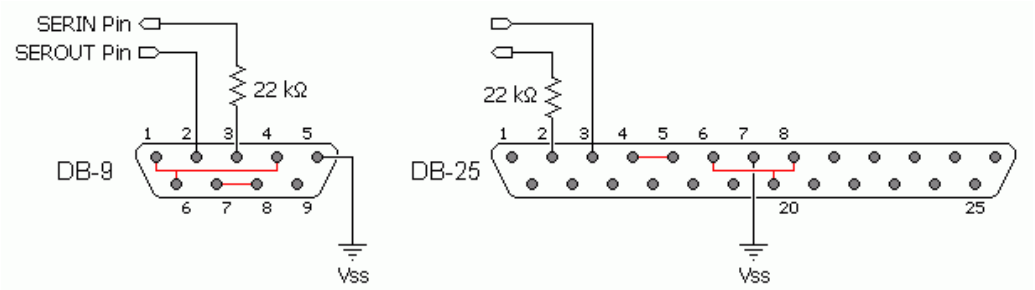


Figura 8.20 configuración del puerto seria RS-232 para recibir o trasmitir datos a través del BS2

Descripción de Pin		
Función	**DB-9**	**DB-25**
Data Carrier Detect (DCD)	1	8
Receive Data (RD)	2	3
Transmit Data (TD)	3	2
Data Terminal Ready (DTR)	4	20
Signal Ground (SG)	5	7
Data Set Ready (DSR)	6	6
Request To Send (RTS)	7	4
Clear To Send (CTS)	8	5

Tabla de configuración de Formatos

	INVERTIDO		NO INVERTIDO	
	8 BIT	7 BIT	8 BIT	7 BIT
VELOCIDAD	NO PARIDAD	PARIDAD PAR	NO PARIDAD	PARIDAD PAR
300	19697	27889	3313	11505
600	18030	26222	1646	9838
1200	17197	25389	813	9005
2400	16780	24972	396	8588
4800	16572	24764	188	8380
9600	16468	24660	84	8276
19200	16416	24608	32	8224
38400	16390	24582	6	8198

Ejemplo 1

```
'Para recibir por el pin 1, del BS2 a 9600 bps, 8N, invertido:

Dato Recivido  VAR    BYTE

SERIN 1, 16468,[Dato_Recivido]
```

El comando SERIN esperara hasta que arribe el dato, si el dato no llega el programa no avanza hacia la otra línea. En el ejemplo 1, no se le especifico el formato a recibir. Por lo que acepta cualquier valor ASCII del conjunto de caracteres de (0-255). Si por ejemplo se le envía el carácter "A", Dato_Recivido = 65 que es el valor equivalente de la letra "A". SERIN acepta el conjunto ASCII de forma natural. Pero que sucede si queremos enviar el decimal 214. Si no se especifica el formato a recibir SERIN solo acepta el primer carácter. Realmente el decimal 214 esta conformado por 3 caracteres ASCII 2 1 4.

La instrucción SERIN al igual que DEBUG contiene una tabla de modificadores de formatos.

MODIFICADORES DE FORMATOS

Formato	Descripción
DEC{1..5}	Numero decimales, opcional de 1 a 5 dígitos.
SDEC{1..5}	Numero decimales con signo, opcional de 1 a 5 dígitos.
HEX{1..4}	Numero hexadecimales, opcional de 1 a 4 dígitos.
SHEX{1..4}	Numero hexadecimales con signo, opcional de 1 a 4 dígitos.
IHEX{1..4}	Numero hexadecimales con el prefijo ($, ejemplo $FF08) opcional de 1 a 4 dígitos.
ISHEX{1..4}	Numero hexadecimales con signo y el prefijo ($, ejemplo -$FF08) opcional de 1 a 4 dígitos.

BIN{1..16}	Numero binarios, opcional de 1 a 16 dígitos.
SBIN{1..16}	Numero binarios con signo, opcional de 1 a 16 dígitos.
IBIN{1..16}	Numero binarios con el prefijo (%, ejemplo %1001001) opcional de 1 a 16 dígitos.
ISBIN{1..16}	Numero binarios con signo y el prefijo (%, ejemplo -%1001001) opcional de 1 a 16 dígitos.

Ejemplo 2

```
Para recibir por el pin 1, del BS2 a 9600 bps, 8N, invertido:

Dato_Recivido  VAR    BYTE

SERIN 1, 16468,[DEC Dato_Recivido]
```

El modificador de formato DEC acepta un valor decimal.

SEROUT

SEROUT Tpin{\Fpin},Baudmode,{Timeout,Tlabel,}[OutData]

Función

Transmite uno ó más datos en el Pin especificado en formato estándar asincrónico RS-232.

- **Rpin** puede ser variable/constante/expresión de (0-16) que especifica el pin a utilizar. tpin es colocado como salida en forma automática. Si se especifica rpin a 16, quiere decir que se utilizara el puerto de programación. Es decir, el que utiliza el DEBUG.

- **Fpin** puede ser variable/constante/expresión de (0-15) es opcional. Se utiliza si se quiere establecer una comunicación con control de datos, muy importante para comunicación de microcontroladores entre sí.

- **Baudmode** puede ser variable/constante/expresión de (0-65535) especifica la velocidad de transmisión y configuración. Ver tabla de configuración.

- **Timeout** es un parámetro opcional, puede ser variable/constante/expresión de (0-65535) le indica a SERIN que si en el tiempo establecido por Timeout en milisegundos no arriban los datos, entonces salta a Tlabel.

- **Tlabel** es un parámetro opcional, que funciona con Timeout, Tlabel es una etiqueta de referencia, que indica que los datos no arribaron, en el tiempo establecido por Timeout.

- **OutData** es una lista de variables que serán enviados a través del puerto serial RS-232. puede ser una variable o un arreglo de variables en forma matricial.

Explicación

Realmente SEROUT es uno de los comandos más extensos y completos del BS2, el BS2 maneja con naturaleza la comunicación serial de hecho cuando usted trabaja con el comando DEBUG sé esta comunicando de forma Serial asincrónica. Para más detalle vea la teoria de operación del comando **SERIN**.

Tabla de configuración de Formatos

	INVERTIDO		NO INVERTIDO	
	8 BIT	7 BIT	8 BIT	7 BIT
VELOCIDAD	NO PARIDAD	PARIDAD PAR	NO PARIDAD	PARIDAD PAR
300	19697	27889	3313	11505
600	18030	26222	1646	9838
1200	17197	25389	813	9005
2400	16780	24972	396	8588
4800	16572	24764	188	8380
9600	16468	24660	84	8276
19200	16416	24608	32	8224
38400	16390	24582	6	8198

MODIFICADORES DE FORMATOS

Formato	Descripción
DEC{1..5}	Numero decimales, opcional de 1 a 5 dígitos.
SDEC{1..5}	Numero decimales con signo, opcional de 1 a 5 dígitos.
HEX{1..4}	Numero hexadecimales, opcional de 1 a 4 dígitos.
SHEX{1..4}	Numero hexadecimales con signo, opcional de 1 a 4 dígitos.
IHEX{1..4}	Numero hexadecimales con el prefijo ($, ejemplo $FF08) opcional de 1 a 4 dígitos.
ISHEX{1..4}	Numero hexadecimales con signo y el prefijo ($, ejemplo -$FF08) opcional de 1 a 4 dígitos.
BIN{1..16}	Numero binarios, opcional de 1 a 16 dígitos.
SBIN{1..16}	Numero binarios con signo, opcional de 1 a 16 dígitos.
IBIN{1..16}	Numero binarios con el prefijo (%, ejemplo %1001001) opcional de 1 a 16 dígitos.
ISBIN{1..16}	Numero binarios con signo y el prefijo (%, ejemplo -%1001001) opcional de 1 a 16 dígitos.

Ejemplo 1

```
'Envía por el puerto de programación el mensaje "Hola Mundo"
SEROUT 16, 16468,["Hola Mundo"]
```

Ejemplo 2

```
'Envía por el pin 1, 9600 bps, 8, N, invertido.
grados  VAR     BYTE
grados = 90
SEROUT 1, 16468,["La temperatura es:", dec3 grados, "C"]
```

SHIFTIN

SHIFTIN Dpin, Cpin, Modo,[Variable{\bits}{, Variable {\bits}...}]

Función

Recibe uno ó más datos en el Pin especificado en formato estándar sincrónico SPI.

- **Dpin** puede ser variable/constante/expresión de (0-15) que especifica el pin a utilizar. Dpin es colocado como entrada en forma automática. Dpin significa Data pin.

- **Cpin** puede ser variable/constante/expresión de (0-15) que especifica el pin a utilizar. Cpin es colocado en modo de salida automáticamente. Cpin significa Clock pin.

- **Modo** puede ser variable/constante/expresión de (0-3), o uno de 4 símbolos predefinidos. Esto le indican el orden de arribar los datos.

- **Variable** es una variable en la cual se almacenan los datos recibidos.

- **Bits** es opcional especifica cuantos bits deben arribar de (1-16). Si no se especifica arribaran 8 Bits.

Explicación

SHIFTIN es la recepción de DATOS de forma sincrónica o formato SPI. Actualmente la mayoría de los dispositivos externos como: A/D, D/A, Potenciómetros Digitales, Memoria EEprom, sensores vienen en este formato de transmisión conocido también con las siglas SPI. El formato de transmisión serial sincrónica es ampliamente utilizado en microcontroladores y periféricos externos.

El formato utiliza dos pines del microcontrolador, uno se llama Data y el otro se llama Clock. Por cada pulso de Clock enviado el recibe un BIT de dato el cual va acumulando en variable. Si el pulso del Clock no le llega esta no envía el siguiente dato. El tiempo en este caso no es importante la coordinación la mantiene el pulso Clock.

Para trabajar con esta instrucción debemos conocer el modo de transmisión o de recepción de datos de cada dispositivo. Esto se consigue con las especificaciones técnicas u hoja de datos de cada fabricante. Esta instrucción en conjunto con SHIFTOUT conforman unas de las instrucciones más útiles para la comunicación serial sincrónica.

Una de las ventajas del formato SPI es que con solo dos pines del microcontrolador se pueden colocar al mismo bus: Data & Clock, diversos dispositivos y un solo pin por cada dispositivo el cual se llama (CS)

Chip Select. Si por ejemplo tenemos 4 dispositivos con formato SPI, gastaríamos dos pines para Data & Clock y 4 Pines para manejar individualmente a cada componente. En total se gastarían 6 Pines. Lo cual es bastante considerable si tenemos en cuenta que estamos manejando 4 dispositivos externos.

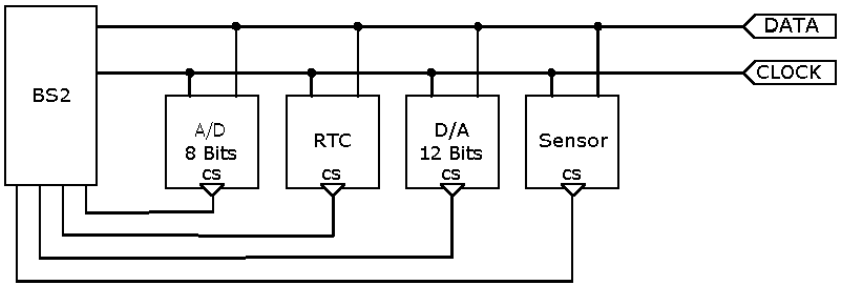


Figura 8.21 comunicación SPI

Este formato es también utilizado en los registros de corrimiento de Datos, los cuales tienen una cadena de 8 o FLIP-FLOP, y los datos son transferidos de un FLIP-FLOP a otro por cada pulso.

Tabla de MODO		
Símbolo	**Valor**	**Operación**
MSBPRE	0	Primero desplaza datos en el BIT superior, lee datos antes de mandar Clock
LSBPRE	1	Primero desplaza datos en el BIT inferior, lee datos antes de mandar Clock
MSBPOST	2	Primero desplaza datos en el BIT superior, lee datos después de mandar Clock
LSBPOST	3	Primero desplaza datos en el BIT inferior, lee datos después de mandar Clock

Para representar esta función necesitamos un dispositivo periférico externo, en este caso no podemos realizar un ejemplo concreto. Por lo exponemos 4 ejemplos de lo que representaría cada función. Y un quinto ejemplo con un dispositivo externo para demostrar físicamente el funcionamiento del comando SHIFTIN.

Ejemplo 1

```
Resultado      VAR     BYTE

SHIFTIN 0,1, MSBPRE,[Resultado]
' Data es pin 0
' Clock es pin 1
' El modo de Formato es MSBPRE
' Recibe 8 BITS por la variable [Resultado]
```

Ejemplo 2

```
D_pin          CON     0
C_pin          CON     1
Resultado      VAR     BYTE

SHIFTIN D_pin, C_pin, MSBPRE,[Resultado]
' Data es pin 0
' Clock es pin 1
' Recibe 8 BITS por la variable [Resultado]
```

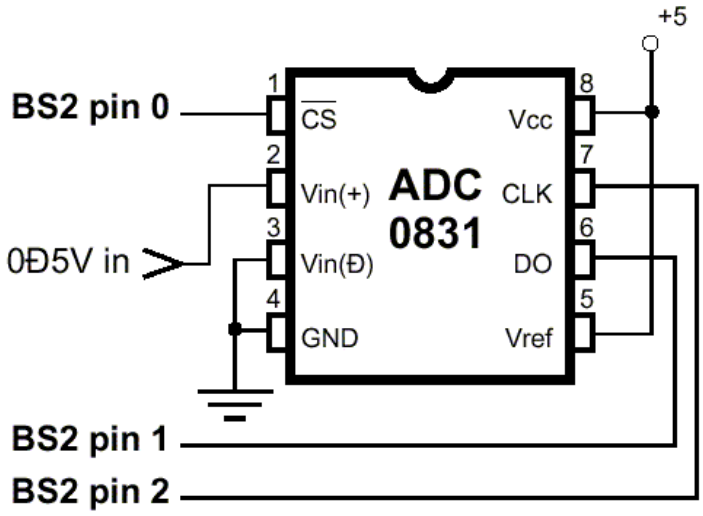
Ejemplo 3

```
D pin          CON     0
C_pin          CON     1
Resultado      VAR     BYTE

SHIFTIN D_pin, C_pin, MSBPOST,[Resultado\4]
' Data es pin 0
' Clock es pin 1
' El modo de Formato es MSBPOST
' Recibe 4 BITS por la variable [Resultado]
```

Ejemplo 4

```
D pin          CON     0
C_pin          CON     1
Result_Low     VAR     WORD
Result High    VAR     NIB

SHIFTIN D_pin, C_pin, MSBPOST,[Result_Low \16, Result_High\4]
' Data es pin 0
' Clock es pin 1
' El modo de Formato es MSBPOST
' Recibe 16 BITS por la variable [Result_Low] y 4 BITS por [Result_High]
```

El siguiente ejemplo se utiliza un convertidor análogo digital ADC0831 de 8 BITS de resolución de la National Semiconductor el cual trabaja con formato SPI. Este convertidor es muy versátil dada su condición de 8 pines. Es ampliamente utilizado. El voltaje de entrada es de (0-5) Voltios.



Ejemplo 5

```
' {$STAMP  BS2}              ' STAMP directiva
adRes     VAR       Byte     ' Resultado de la conversión A/D.

CS        CON       0        ' Chip Select en pin 0.
AData     CON       1        ' Data Pin es pin 1.
Clock     CON       2        ' Clock Pin es pin 2.

Init:
  HIGH  CS                   ' Reinicia el A/D.


Again:
  LOW CS                                        ' Activa a ADC0831.
  SHIFTIN AData, Clock, MSBPOST, [adRes\9]      ' Proceso de recuperación de datos.
  HIGH CS                                       ' Desactiva a ADC0831.
  DEBUG HOME, DEC5 adRes                        ' Muestra el resultado de la conversión.
  PAUSE 500                                     ' Espera ½ Segundo.
GOTO Again                                      ' Vuelve a Again.
```

SHIFTOUT

SHIFTOUT Dpin, Cpin, Modo,[Data{\bits}{,Data{\bits}...}]

Función

Envía uno ó más datos en el Pin especificado en formato estándar sincrónico SPI.

- **Dpin** puede ser variable/constante/expresión de (0-15) que especifica el pin a utilizar. Dpin es colocado como salida en forma automática. Dpin significa Data pin.
- **Cpin** puede ser variable/constante/expresión de (0-15) que especifica el pin a utilizar. Cpin es colocado en modo de entrada automáticamente. Cpin significa Clock pin.
- **Modo** es un valor de (0-1) predefinidos e indican el orden de enviar los datos.
- **Data** es una variable/constante/expresión que contiene el Dato a ser enviado.
- **Bits** es opcional especifica cuantos bits debe enviar de (1-16). Si no se especifica envia 8 Bits.

Explicación

SHIFTOUT es la transmisión de DATOS en forma sincrónica o formato SPI. Ver comando SHIFTIN para comprender la teoría de funcionamiento del formato SPI.

Tabla de MODO		
Símbolo	**Valor**	**Operación**
LSBFIRST	0	Primero desplaza datos del BIT inferior
MSBFIRST	1	Primero desplaza datos del BIT superior

Para representar esta función necesitamos un dispositivo periférico externo, en este caso no podemos realizar un ejemplo concreto. Por lo exponemos 3 ejemplos de lo que representaría cada función. Y un cuarto ejemplo con un dispositivo externo para demostrar físicamente el funcionamiento del comando SHIFTOUT.

Ejemplo 1

```
D pin   CON     0       ' Data es pin 0
C pin   CON     1       ' Clock es pin 1

SHIFTOUT D pin, C pin, MSBFIRST,[250]

' El modo de Formato es MSBFIRST
' Envía el numero 250 de forma sincrónica
```

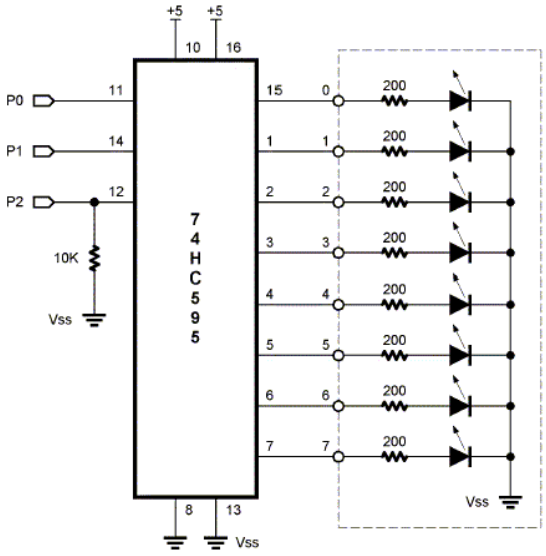
Ejemplo 2

```
D_pin   CON     0       ' Data es pin 0
C_pin   CON     1       ' Clock es pin 1

SHIFTOUT 0,1, MSBFIRST,[250\4]
' El modo de Formato es MSBFIRST
' Envía los 4 BITS inferiores de 250 en este caso %0000
```

Ejemplo 3

```
D pin   CON     0       ' Data es pin 0
C pin   CON     1       ' Clock es pin 1

SHIFTOUT 0,1, MSBFIRST,[250\4,1024\16]
' El modo de Formato es MSBFIRST
' Envía los 4 BITS inferiores de 250 en este caso %0000
' Envía los 16 BITS del numero 1024
```

El siguiente ejemplo se muestra un registro de corrimiento de 8 BITS conocido como 74HC595, el cual convierte los datos de serial a paralelo, el circuito una vez conexionado y el programa cargado funciona como un contador binario ascendente. Utilizando solamente 3 pines del BS2. El 74HC595 tiene la capacidad de colocarse en serie con otro 74HC595 y conformar múltiples circuitos por cada 74HC595 que se adicione las salidas aumentan de 8 en 8, manteniendo la misma cantidad de pines del BS2.



Ejemplo 4

```
' {$STAMP  BS2}
DPin      CON       0         ' Data pin para 74HC595 pin 11.
Clock     CON       1         ' Clock pin para 74HC595 pin 14.
Latch     CON       2         ' Transfiere los datos pin 12.

counter   VAR       Byte      ' Variable para el contador

Again:
  SHIFTOUT DPin,Clock,MSBFIRST,[counter]        ' Envía los bits
  PULSOUT Latch,1                               ' Transfiere las salidas
  PAUSE 50                                      ' Espera 50 ms
  counter = counter + 1                         ' Incrementa a counter
  GOTO Again                                    ' Vuelve a Again
```

SLEEP

SLEEP Segundos

Función
Pone el BS2 en modo de bajo consumo de energía por él numero de segundos especificados.

- **Segundos** puede ser una variable/constante/expresión de (1-65535) especifica la duración en el que el BS2 estará en modo descanso, la unidad de un periodo en realidad para el BS2 hay que multiplicarla por 2.3 segundos.

Limites

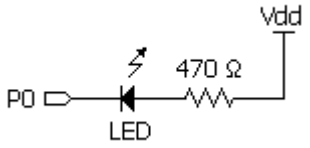
Consumo de corriente en modo Normal	8 mA
Consumo de corriente en modo NAP	40 uA

Explicación
SLEEP utiliza el mismo mecanismo de modo de ahorro de energía de la computadoras actuales. El rango puede ser desde 2.3 segundos a 18 horas.

Durante la función SLEEP el puerto mantiene su estado de direccionamiento. Sin embargo, las salidas son interrumpidas cada de 2.3 segundos.

Es posible utilizar SLEEP para equipos que utilicen baterías y que en ciertos periodos no esta realizando ninguna función importante. Sin embargo, hay que tener en cuenta que las salidas se desconectan brevemente cada 2.3 segundos.

En la siguiente demostración conecte un diodo led a P0:



Ejemplo 1

```
DEBUG CLS                    'Limpia la pantalla
HIGH 0                       'Enciende el LED
Duerme:                      'Etiqueta de Referencia
  DEBUG "Voy a Dormir", CR
  SLEEP 10                   'Duerme por 23 Segundos
  DEBUG "Estoy Despierto", CR
  PAUSE 500                  'Espera ½ segundo
GOTO Duerme                  'Retorna a Duerme
```

STOP

STOP

Función
Detiene la ejecución de un programa.

Explicación
STOP detiene la ejecución de un programa. Este comando se utiliza para depurar programas supóngase que usted esta programando y las cosas no le están saliendo bien. Aunque el corrector de sintaxis no le indica error alguno, pero el programa no realiza lo que usted esperaba.

Entonces usted comienza a depurar el programa por etapas. Donde usted inserta un comando STOP. Hasta ese punto le funcionara el programa. El programa permanecerá indefinidamente hasta que:

1- Se presione Reset.
2- Se recargue un nuevo programa.
3- Desconectándolo y reconectándolo nuevamente.

El programa se inicia desde el principio y se detiene cuando encuentre un comando STOP.

STOP es diferente a END, STOP detiene la ejecución y el consumo de energía se mantiene constante. El comando END detiene la ejecución y el consumo de energía disminuye totalmente.

Ejemplo 1

```
DEBUG CLS                    'Limpia la pantalla
DEBUG "Proceso No. 1",CR
DEBUG "Proceso No. 2",CR
DEBUG "Proceso No. 3",CR
STOP                         'Detiene la ejecución
DEBUG "Proceso No. 4",CR
DEBUG "Proceso No. 5",CR
```

Inserte STOP en diferentes puntos del programa y vea la diferencia.

TOGGLE

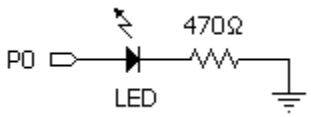
TOGGLE Pin

Función
Invierte el estado de salida de Pin especificado.

- **Pin** puede ser una variable/constante/expresión de (0-15) del puerto de Entrada/Salida del BS2. Este Pin se direcciona como salida automáticamente.

Explicación
La función TOGGLE pone el Pin especificado como salida e invierte el estado, es decir, cambia de 0 a 1 y de 1 a 0. Realmente TOGGLE trabaja midiendo el estado actual del Pin en cuestión y luego invierte el estado.

El programa de abajo muestra un LED parpadeando.



Ejemplo 1

```
' {$STAMP BS2}          ' STAMP directive

Again:
  PAUSE 500             ' Espera (1/2 segundo)
  TOGGLE 0              ' Invierte la dirección del Pin 0
GOTO Again              ' Repite para siempre
```

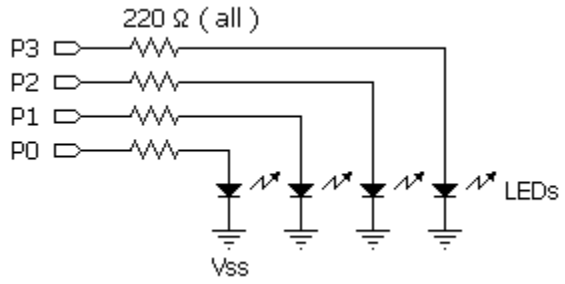
El siguiente ejemplo muestra una función equivalente a TOGGLE, esta función puede resultar útil cuando queramos resultados invertidos, en salidas de pulsadores momentáneos.

Ejemplo 2

```
' {$STAMP BS2}          ' STAMP directive

DIRS = $FFFF            ' Direcciona el puerto como salida
OUTS = $0000            ' Todas las salidas en LOW

Main:                   ' Etiqueta de referencia
  OUT0 = ~OUT0          ' Invierte el estado actual
  DEBUG BIN16 OUTS,CR   ' Imprime el estado de las salidas
  PAUSE 500             ' Espera (1/2 segundo)
GOTO Main
```

El siguiente ejemplo se conectan 4 LEDs y se muestra un efecto visual muy interesante utilizando la función TOGGLE. En principio todos los LEDs están apagado a medida que el contador avanza de 0 a 3, cada LED queda encendido, luego se invierte el ciclo y los LEDs comienzan a

apagarse individualmente. Esto también se puede apreciar por la pantalla a través de DEBUG.



Ejemplo 3

```
' {$STAMP BS2}          ' STAMP directive

4 leds  VAR     NIB

DIRS = $FFFF            ' Direcciona el puerto como salida
OUTS = $0000            ' Todas las salidas en LOW

Main:                           ' Etiqueta de referencia
  FOR 4 leds = 0 TO 3           ' Contador de 0 a 3
    TOGGLE 4leds                ' Invierte la salida de cada LED
    DEBUG BIN16 OUTA,CR         ' Imprime el estado de las salidas del puerto A
    PAUSE 500                   ' Espera (1/2 segundo)
  NEXT                          ' Continua con el contador
GOTO Main                       ' Retorna a Main
```

WRITE

WRITE Localización, DatoTipo

Función
Puede escribe datos en la memoria EEprom en tiempo de ejecución.

- **Localización** puede ser una variable/constante/expresión de (0-2047) que indica la dirección en la memoria EEprom del BS2, la memoria EEprom del BS2 contiene 2048 direcciones en la que puede almacenar un BYTE por dirección. Por lo general es de tipo WORD.

- **DatoTipo** puede ser una variable/constante/expresión especifica el valor a almacenar. Por lo general es de tipo BYTE.

Limites

Dirección inicial	0
Dirección final	2047

Explicación
En realidad la sentencia WRITE y DATA realizan la misma función almacenar datos permanentemente en la memoria EEprom. La diferencia consiste en que DATA graba los datos durante la programación y WRITE graba los datos después de la programación, es decir, en tiempo de ejecución. Esto lo convierte en una valiosa herramienta. Supóngase que usted desea una aplicación que tome muestra de temperatura cada cierto tiempo y las guarde en la memoria permanentemente y luego usted puede visualizarlas.

La función WRITE se complementa con la función READ, READ lee los datos después de almacenados con la sentencia DATA o WRITE.

En el siguiente ejemplo se pueden gravar 7 dígitos del 0 al 9, los cuales pueden corresponder a un numero telefónico. Con dos pulsadores usted tiene un menú interactivo donde selecciona los dígitos y luego los graba en la memoria EEprom permanentemente.

Ejemplo 1

```
' -----[ Title ]-----------------------------------------------------------
'
' File..... Write 01.bs2
' Purpose... Aprender el uso del comando WRITE
' Author.... Diego Pulgar

' -----[ Program Description ]---------------------------------------------
'Este programa contiene 2 push botón conectados a D7 y D6
'los cuales funcionan como un menu interactivo
'D7 = Leer & D6 = Grabar

' -----[ I/O Definitions ]--------------------------------------------------
'
```

```
' -----[ Constants ]-------------------------------------------------------
'

' -----[ Variables ]-------------------------------------------------------
'
Indice    VAR       BYTE
Caracter  VAR       BYTE
Valor               VAR       BYTE
Cont                VAR       NIB

' -----[ EEPROM Data ]-----------------------------------------------------
'
'
' -----[ Initialization ]---------------------------------------------------

OUTS = $FFFF
DIRS = $0000

DEBUG CLS, "---------Programa Interactivo para Grabar Datos---------",CR,CR
DEBUG "Presione D7 para Grabar",CR
DEBUG "Presione D6 para Leer",CR

' -----[ Main Code ]-------------------------------------------------------
'
Main:
  IF IN0 = 0 THEN Leer
  IF IN1 = 0 THEN Grabar
  PAUSE 50
GOTO Main

Grabar:
  DEBUG CLS : PAUSE 50
  DEBUG "Presione D7 para Grabar",CR
  DEBUG "Presione D6 para Incrementar el valor",CR
  DEBUG 2,2,(8+Cont),"Dato No.",DEC1 (Cont+1),"-(",DEC1 Valor,")"

Grabar_Debounce:
  IF IN1 = 0 THEN Grabar Debounce

S Grabar:
  PAUSE 50
  IF IN0 = 0 THEN Incre
  IF IN1 = 0 THEN Grabar_Write
GOTO S Grabar

Grabar Write:
  WRITE Cont, Valor
  GOSUB Save
  IF Cont = 6 THEN FIN_Grabar_Write
  Cont = Cont + 1
  DEBUG 2,2,(8+Cont),"Dato No.",DEC1 (Cont+1),"-(",DEC1 Valor,")"

Write Debounce:
  IF IN1 = 0 THEN Write_Debounce
GOTO S_Grabar

Incre:
  Valor = (Valor + 1)//10
  DEBUG 2,2,(8+Cont),"Dato No.",DEC1 (Cont+1),"-(",DEC1 Valor,")"
Incre_Debounce:
  IF IN0 = 0 THEN Incre_Debounce
  GOTO S Grabar

Leer:
```

```
DEBUG 2,2,7,"Leyendo los datos",CR
FOR Indice = 0 TO 6
  READ Indice, Caracter
  DEBUG 2,2,(8+Indice),"Dato No.",DEC1 Indice,"-(",DEC1 Caracter,")"
NEXT

FIN_Grabar_Write:
DEBUG 2,2,16,"Archivo Completo..."

END
' -----[ Subroutines ]------------------------------------------------------
'
Save:
  DEBUG 2,20,4, "Grabando"
  FOR Indice = 0 TO 3
    DEBUG "." : PAUSE 250
  NEXT
  PAUSE 250
  DEBUG 2,20,4, REP " "\12
RETURN
```

XOUT

XOUT Mpin,Zpin,[House\Commando{Ciclos}{,House\Commando{\Ciclos}…}]

Función

Envía un código X-10 por la red eléctrica de 120AC/60Hz. A través de un dispositivo que sirve como interface entre la línea AC y el microcontrolador.

- **Mpin** puede ser una variable/constante/expresión de (0-15) del puerto de entrada / salida del BS2 para enviar la señal (modulada) X-10, a través del dispositivo de interface. Este pin es ajustado como salida automáticamente.

- **Zpin** puede ser una variable/constante/expresión de (0-15) del puerto de entrada / salida del BS2 que recibe la señal (cruce por cero) X-10, a través del dispositivo de interface. Este pin es ajustado como entrada automáticamente.

- **House** puede ser una variable/constante/expresión de (0-15) que especifica el código de la unidad representada por los valores de (0-15) los cuales representan las letras de la (A a la P) el los dispositivos.

- **Comando** puede ser una variable/constante/expresión de (0-30) que especifica el comando a enviar. Los valores de (0-15) corresponde con el código de la unidad de (1-16) y del (17-30) son códigos de control los cuales se describen en la tabla de códigos de control.

- **Ciclos** es opcional puede ser una variable/constante/expresión de (1-255) especifica él numero de veces que se transmitirá el código X-10. Si no se especifica el transmite dos veces por defecto.

Limites

Compatible con los módulos de interface	PL-513 y TW-523
Nota especial	El comando XOUT detiene la ejecución mientras transmite. Si el modulo no esta energizado. El BS2 se detiene hasta que se energice.

Explicación

El comando XOUT le da el control absoluto sobre los módulos X-10. XOUT se usa para enviar información de control a dispositivos X-10. Los módulos X-10 están disponibles en muchos lugares y de distintos proveedores.

Los módulos X-10 controlan dispositivos del hogar como luces, tomacorrientes, breaker y alarmas. Se conectan directamente a la red eléctrica de 120AC/60Hz interrumpiendo el dispositivo a controlar.

Todos los módulos X-10 se comunican a través de red eléctrica de 120AC/60Hz, la cual aprovechan como el medio de transmisión. Si por ejemplo usted quiere controlar unas luces distantes solo tiene que colocar un modulo X-10 a las luces a controlar. Luego de otro punto distante y con otro modulo X-10 conectado a la red del servicio eléctrico usted puede enviarle ordenes a dicho modulo o a varios módulos. Esto evita el re-alambrar los dispositivos a automatizar. El concepto X-10 esta estrechamente ligado al tema de hogares inteligentes.

Puede buscar más información sobre este tema en la dirección: http://www.x10.com/homepage.htm

Para que el BS2 pueda controlar los dispositivos X-10 se requiere una interfase que se conecte a la red eléctrica de 120AC/60Hz. Unos de los módulos de interface es el TW-523 para comunicaciones de dos vías y el PL-513 para comunicaciones de una vía. Estos dispositivos tienen la interfase a la línea de alimentación y aíslan el microcontrolador de la línea de AC. Como X-10 está patentado, esta interfase también cubre el licenciamiento.

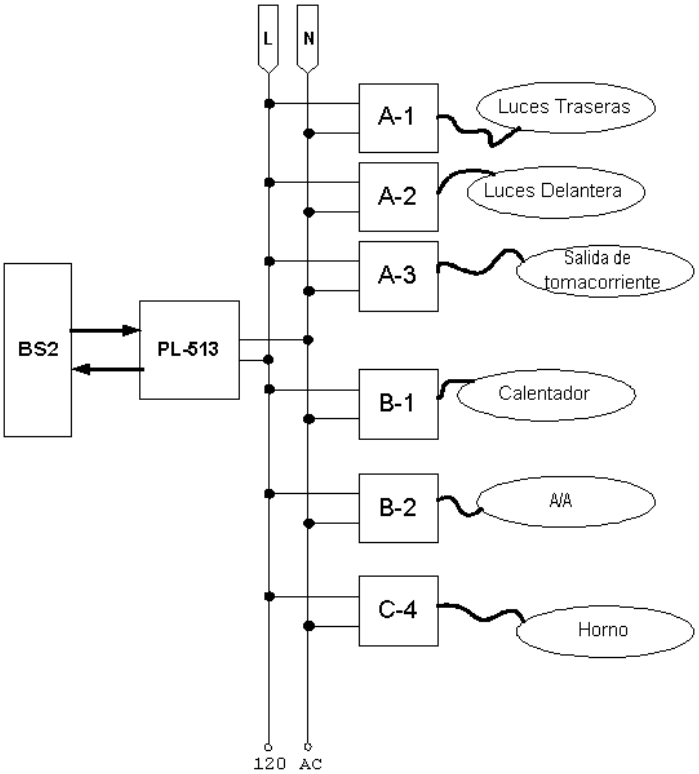


Figura 8.25 diagrama en bloques de la interconexión entre los modulos X-10 y el BS2

Detalles del protocolo X-10

El formato X-10, es el mismo utilizado por las compañías de transmisión de energía eléctrica mejor conocido como Power Lan Carrier PLC. El sistema X-10 fue adecuado para operar en las redes eléctricas de 120AC/60Hz. Trabaja imponiendo una señal portadora de 120 Khz, durante los cruces por cero de la línea de 60Hz. En este momento la señal portadora de 120 KHz envía 11 bits en apenas 50 uS. Esto se realiza de una manera tan veloz que es equivalente a que los dispositivos X-10 se comunicaran sin la energía eléctrica, es como si la línea de 120AC/60Hz existiera solamente para los módulos X-10. esto gracias a que los 60Hz de la red eléctrica son relativamente lentos en comparación con los 120 Khz de la señal portadora.

XOUT solamente procesa datos en el momento en que la línea de AC pasa por cero (en ese momento recibe ZeroPin). En la práctica, primero se envía un comando especificando el número de módulo X-10, seguido por un comando especificando la función deseada. Algunas funciones operan en todos los módulos, por lo que el número de módulo es innecesario.

Los módulos X-10

Cada día la gama de productos X-10 es más diversa desde alarmas, cámaras de seguridad, control de dispositivos, música ambiental, etc. Pero todos los módulos tienen algo en común que es código House y el numero de la unidad. House contiene de la letra (A-P) y unidad del (1-16). Realizando combinaciones podemos tener hasta 256 dispositivos.

UNIT																
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
HOUSE																
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	

Figura 8.26 combinación del código HOUSE con el numero de la unidad

A cada dispositivo le asignamos una letra y un numero de unidad esta combinación lo hace único. Entones cuando queramos controlar una unidad especifica solo tendremos que llamarla por su nombre.

Por lo general en un hogar no es necesario tener más de 16 unidades, por lo que se le asigna una misma letra a todos y luego es varia él numero de unidad a cada dispositivo.

Figura 8.27 diversos dispositivos X-10 de interfase

El modulo PL-513

El interfase PL-513 requiere 4 conexiones. En la figura 8.28 se puede apreciar un modulo PL-513, en la figura 8.29 el diagrama de conexión hacia el BS2, se puede ver que el pin 1 del modulo PL-513 es colector abierto por lo que requiere polarizarlo con +5V a través de una resistencia en serie de 10K ohms.

Figura 8.28 módulo PL-513

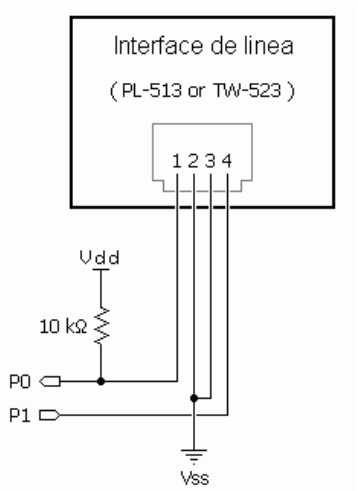


Figura 8.29 conexión del PL-513 al BS2

PL-513	Al BS2
1	Zpin*
2	GND
3	GND
4	Mpin

*Este pin debe ser conectado a +5V a través de una resistencia de 10K

Tabla de códigos de control		
KeyCode	**Valor**	**Función**
unitOn	%10010	Enciende el módulo
UnitOff	%11010	Apaga el módulo
UnitsOff	%11100	Apaga todos los módulos
LightsOn	%10100	Enciende todos los módulos de luz
Dim	%11110	Menos brillo al módulo de luz
Bright	%10110	Más brillo al nódulo de luz

He aquí un ejemplo utilizando el comando XOUT:

```
Zpin        CON   0
Mpin        CON   1
House_A     CON   0
Unit_2      CON   1

XOUT Mpin, Zpin, [House_A\Unit_2]         ' Alerta al Modulo A-2
XOUT Mpin, Zpin, [House_A\UnitOn]         ' Enciende al Modulo A-2
```

Se puede también combinar dos instrucciones en una:

```
'Enciende al Modulo A-2
XOUT Mpin, Zpin, [House_A\Unit_2\2, House_A\UnitOn]
```

He aquí un ejemplo controlándole la intensidad a una lámpara:

```
Zpin        CON   0
Mpin        CON   1
House_B     CON   1
Unit_5      CON   4
' Alerta al Modulo B-5
XOUT Mpin, Zpin, [House_B\Unit_5]

' Reduce la intensidad de una lampara
XOUT Mpin, Zpin, [House_B\UnitOff\2, House_B\Dim\10]
```

Mapa de caracteres ASCII

Dec	Hex	Carácter	Dec	Hex	Carácter	Dec	Hex	Carácter
32	20	<espacio>	64	40	@	96	60	`
33	21	!	65	41	A	97	61	a
34	22	"	66	42	B	98	62	b
35	23	#	67	43	C	99	63	c
36	24	$	68	44	D	100	64	d
37	25	%	69	45	E	101	65	e
38	26	&	70	46	F	102	66	f
39	27	'	71	47	G	103	67	g
40	28	(	72	48	H	104	68	h
41	29	)	73	49	I	105	69	i
42	2A	*	74	4ª	J	106	6A	j
43	2B	+	75	4B	K	107	6B	k
44	2C	,	76	4C	L	108	6C	l
45	2D	-	77	4D	M	109	6D	m
46	2E	.	78	4E	N	110	6E	n
47	2F	/	79	4F	O	111	6F	o
48	30	0	80	50	P	112	70	p
49	31	1	81	51	Q	113	71	q
50	32	2	82	52	R	114	72	r
51	33	3	83	53	S	115	73	s
52	34	4	84	54	T	116	74	t
53	35	5	85	55	U	117	75	u
54	36	6	86	56	V	118	76	v
55	37	7	87	57	W	119	77	w
56	38	8	88	58	X	120	78	x
57	39	9	89	59	Y	121	79	y
58	3A	:	90	5ª	Z	122	7A	z
59	3B	;	91	5B	[	123	7B	{
60	3C	<	92	5C	\	124	7C	|
61	3D	=	93	5D	]	125	7D	}
62	3E	>	94	5E	^	126	7E	~
63	3F	?	95	5F	_	127	7F	delete

Palabras Reservadas

BASIC STAMP II			
ABS	DTMFOUT	LOWNIB	REP
AND	END	LSBFIRST	REVERSE
ASC	FOR	LSBPOST	SBIN
BELL	FREQOUT	MSBPRE	SBIN1..SBIN16
BKSP	GOSUB	MAX	SDEC
BIN	GOTO	MIN	SDEC1..SDEC5
BIN1..BIN4	HEX	MSBFIRST	SERIN
BIT	HEX1..HEX4	MSBPOST	SEROUT
BIT0..BIT15	HIGH	MSBPRE	SHEX
BRANCH	HIGHBIT	NAP	SHEX1..SHEX4
BRIGHT	HIGHNIB	NCD	SHIFTIN
BUTTON	HOME	NEXT	SHIFTOUT
BYTE	IHEX	NIB	SIN
CLS	IHEX1..IHEX4	NIB0..NIB3	SKIP
CON	IF	NOT	SLEEP
COS	IN0..IN5	OR	STEP
COUNT	INA	OUT0..OUT15	STOP
CR	INB	OUTA	STR
DATA	INC	OUTB	SQR
DCD	IND	OUTC	TAB
DEBUG	INH	OUTD	THEN
DEC	INL	OUTH	TO
DEC1..DEC5	INPUT	OUTL	TOGGLE
DIG	INS	OUTPUT	UNITOFF
DIM	ISBIN	OUTS	UNITON
DIR0..DIR15	ISBIN1..ISBIN16	PAUSE	UNITSOFF
DIRA	ISHEX	RCTIME	VAR
DIRB	ISHEX1..ISHEX4	REV	WAIT
DIRC	LIGHTSON	PULSIN	WAITSTR
DIRD	LOOKDOWN	PULSOUT	WORD
DIRH	LOOKUP	PWM	WRITE
DIRL	LOW	RANDOM	XOR
DIRS	LOWBIT	READ	XOUT