Introduccion al Great Cow BASIC

Hola, bienvenidos a la ayuda de: "Great Cow BASIC". Este archivo de ayuda fue elaborado con la intencion de actuar como un respaldo de otros documentos, para clarificar tópicos que no hayan sido expuestos lo suficientemente claros en otros sitios.

Si eres nuevo en programación, deberás tratar:"The Great Cow Basic tutorial". Alli se explica todo paso a paso, y se asume que no hay conocimiento previo.

Si has programado en otro lenguaje, entonces los programas de demostración y referencias de comando podrían ser el mejor lugar para consultar.

Si hay alguna otra cosa en la que necesites ayuda, por favor visita: Great Cow BASIC forum.

Responder Preguntas Frecuentes

Cuales chips soporta GCBASIC?
Se espera que todos los chips de serie: 10/12/16/18. sean soportados Si usted encuentra alguno en el que GCBASIC no trabaje adecuadamente, por favor, email a: hconsidine@bigpond.com y efectúe su consulta.

Es GCBASIC sensitivo a cambios de minusculas-mayusculas?
No! Por ejemplo, Set, SET, set, SeT, etc., son todos tratados exactamente de la misma forma por GCBASIC.

Podria especificar el bit de una variable para alterarla usando otra variable? Set variable.otravariable no generará un error, pero no actuara como se espera. Lo correcto sería: ej. Set variable.3 on

Porque la facilidad x no es implementada?
Porque no hemos profundizado suficientemente en ella!. Si hay otras facilidades que les gustaria ver en GCBASIC, por favor, postúlela en la sección de "Discusion abierta" del forum de GCBASIC.

Usando GCBASIC

Necesita compilar un programa con "Great Cow Basic, pero no sabe donde comenzar? Pruebe estas instrucciones:

1. Primero, baje e instale GCBASIC. Es mejor si usted deja todo en el directorio predeterminado (carpeta)
2. Abra la carpeta que contiene GCBASIC. Esta es generalmente C:\Program Files\GCBASIC o para paises hispanoparlantes C:\Archivos de Programa\GCBASIC.
3. Abra la carpeta que contiene el programa que desea compilar. Este seguro de abrirlo en una nueva ventana. Arrastre el icono del archivo que desea compilar sobre "compile.bat". El compilador se abrira, compilara el archivo, y entonces se cerrara de nuevo.
4. Un archivo llamado "compiled.hex" se habra creado. Bájelo a su "Programador de prototipos"*, y usted estara listo para continuar!

Es posible crear una copia de "compile.bat" en un lugar mas conveniente, como por ejemplo como la carpeta donde guarda sus programas. compile.bat trabajara no importa donde este localizado, mientras todos los otros archivos sean dejados en el lugar donde estos fueron instaladas.

*Si usted no sabe como hacer esto, hay varios sitios en la red donde pueden ayudarlo. Pruebe: http://www.ic-prog.com/ como un punto de partida.

Inputs/Outputs

Configuration

Acerca de Configuracion About Configuration
Cada PIC tiene una palabra de configuracion. Esta es un area de memoria en el chip que guarda la preseleccion que gobierna la operacion del chip.

Los siguientes aspectos del chip son gobernados por la palabra de configuracion: (CONFIG word)
- Seleccion del Oscilador - correra el chip con oscilador interno o lo hara con uno externo?.
- Reset automatico - Debera el chip resetearse si el voltaje baja demasiado?. Detectara si esta corriendo la misma pieza de codigo constantemente?
- Proteccion de Codigo - Que areas de memoria deben permanecer escondidas una vez escritas?
- Uso de Pines - Cuales pines seran dispuestos para programacion, reiniciar el chip, o emitir señales de modulacion de ancho de pulso, (PWM)?

La exacta prefijacion de configuracion varia de acuerdo al tipo de chip. Para encontrar una lista de prefijacion valida, por favor consulte la hoja de data para el PIC que usted desea usar.
- Todo esto puede ser bastante confuso para usuarios nuevos - no obstante, GCBASIC pondra automaticamente algunas prefijaciones de configuracion, al menos que se desee lo contrario:
- **Programacion con bajo voltaje (LVP) es apagado.** Esto permite que el pin de programacion (usualmente B3 o B4) sea usado normalmente como de entrada y salida. (I/O pin).
- **Perro Guardian Timer (WDT) es apagado.** El perro guardian reinicia el chip si se esta corriendo la misma pieza de codigo una y otra ves seguido - esto causa problemas con algunas rutinas largas de retardo en GCBASIC.
- **Clear maestro (MCLR) sera desabilitado cuando sea posible.** En muchos chips nuevos esto permite que el pin MCLR (generalmente PORTA.5) puedan ser usados como pin de entrada y salida convencional. Tambien elimina la necesidad de usar la resistencia de polarizacion usada en el pin MCLR.
- **Un modo de oscilador sera seleccionado de acuerdo a las siguientes reglas:**

1. Si la velocidad de reloj es mayor de 4 Mhz, sera seleccionado el oscilador HS.
2. Si la velocidad de reloj es de 4 MHz o menos pero el chip puede soportar un oscilador externo, entonces el modo XT es seleccionado.
3. Si un chip no puede usar un oscilador externo, el oscilador interno es seleccionado.

Note que esas configuraciones pueden facilmente ser cambiadas individualmente cada vez que se necesite hacerlo. Por ejemplo, si el Watchdog Timer (Perro Guardian) se necesita, añadiendo la linea: #config WDT = ON
permitira que el perro guardian se inicie, sin afectar alguna otra configuracion.

Usando la Configuracion

Una vez que las opciones necesarias de CONFIG hayan sido determinadas, añadirlas al programa es facil. En una linea nueva tipee "#config" y entonces liste las opciones deseadas separadas por comas, como esta en la linea siguiente:

```
#config OSC = RC, BODEN = OFF
```

GCBASIC tambien soporta este formato en la serie de chips 10/12/16 como en la siguiente linea:

```
#config INTOSC_OSC_NOCLKOUT, BODEN_OFF
```

Para mas ayuda, vea:
#config Directive

Comandos

Lista de comandos de GCBASIC

Conversion Analógica/Digital
ReadAD
ADOff
EEPROM
EPRead
EPWrite
ProgramErase
ProgramRead
ProgramWrite
Control de Flujo
Call Every
Do
End
ExitSub
For
Gosub
Goto
IF
Repeat
Select
Wait
Pantalla de Cristal Liquido, LCD
PRINT
LOCATE
PUT
GET
CLS
LCDInt
LCDHex
LCDWord
Teclado
KeypadData
KeypadRaw
Modulacion de Ancho de Pulso, PWM
PWMOff
PWMOn
PWMOut
HPWM
Presentación de 7 segmentos
DisplayValue
DisplayChar
Comunicacion Serial RS232
InitSer
SerSend
SerReceive
Comunicacion Serial SPI
SPIMode
SPITransfer
Sonido
Tone

ShortTone
SoundOut
Ajustar Variables
ROTATE
SET
DIM
Comandos Varios
DIR
POT
PULSEOUT
PEEK
POKE
ReadTable
Dispositivos de Tiempo (Timers)
InitTimer0
InitTimer1
StartTimer
ClearTimer
StopTimer
Directivas del Compilador
#Chip
#Config
#Define
#Include
#Script
#Int
#StartUp
#Mem

Definiciones

Que es definir ("define")?
Una definición "define" es un tipo de directiva que le dice al compilador que encuentre una palabra dada y la reemplace con otra palabra o número. Una definicion "define" es útil para situaciones donde una rutina necesita ser alterada facilmente. Por ejemplo, una definición podría usarse para especificar la cantidad de tiempo que funcionara una alarma una vez iniciada.

es tambien posible usar definiciones "defines" para especificar puertos - estas definiciones pueden ser usadas para ayudar en la creación de código que pueda ser facilmente adaptado para correr en diferentes PICs con diferentes puertos.

GCBASIC hace uso considerable de definiciones "defines" internamente. Por instancia, el codigo LCD usa definiciones "defines" para poner los puertos que deben ser usados para comunicarse con el LCD.

Usando Definiciones
Para crear una definición es necesario usar la directiva #define. Aqui estan algunos ejemplos de definiciones:
#define Línea 34
#define Luz PORTB.0
#define LuzOn Set PORTB.0 on

Línea es una constante simple - GCBASIC encontrara "Linea" en el programa y la reemplazara con el numero 34. Esto podria ser usado en programas para seguimiento de linea, para hacer mas facil calibrar el programa para diferentes condiciones de iluminacion.

Luz es un puerto - representa un pin particular en el PIC. Esto podria ser util si el programa tiene muchas lineas de codigo que controlan Luz, *y* alli existe la posibildad de que el puerto al que Luz fue anexada necesitara cambiarse en el futuro.

LuzOn es una definicion usada para hacer al programa mas leible. En lugar de tipear "Set PORTB0 on" una y otra vez, podria ser posible escribir LuzOn y hacer que el compilador haga la dura tarea.

Variables

Acerca de Variables
Una variable es un area de memoria del microcontrolador (PIC), que puede ser usada para guardar un numero o una serie de letras. Eso es muy util para muchos propositos, como tomar una lectura de un sensor y actuar sobre el, o contar el numero de veces que un robot ha llevado a cabo un accion particular.

A cada variable se le debe dar un nombre, como "MiVariable o "CuentaPasos".
Escoger un nombre para una variable es facil - simplemente no incluya espacios o cualquier otro simbolo (salvo _), y este seguro que el nombre es al menos 2 caracteres (letras o numeros) de largo. Una variable siempre debe empezar por una letra.

Typos de Variables
Hay diferentes tipos de variables, y cada tipo puede almacenar una clase diferente de informacion. Estas son los tipos de variable que Great Cow BASIC
usa corrientemente:

Typo de Variable	Informacion que esta variable puede almacenar	Ejemplo de usos para este tipo de variable
Bit	Un bit (0 or 1)	Flags to track whether or not a piece of code has run
Byte	Un un numero entero entre 0 and 255	Almacenaje de proposito general de datos como contadores.
Word	Un un numero entero entre 0 and 65535	Almacenaje de numeros extra grandes
Array	Una lista de numeros enteros de un rango entre 0 to 255	Matrices
String	Una serie de letras, numeros y simbolos.	Mensages que deberan ser vistos en la pantalla

Usando variables
Las variables no necesitan ningún comando especial para colocadas - solo ponga el nombre de la variable en el comando cuando la variable sea necesitada.

Otros tipos de variables pueden ser usadas en una manera muy similar, excepto que ellas deban ser dimencionadas primero. Esto requiere el uso del comando DIM para decirle al Great Cow BASIC que esta manejando algo diferente a una variable byte.

Una característica de las variables es que es posible tener al microcontrolador chequeando una variable y solo correr una sección de codigo si es de un valor dado. Esto puede hacerse con el comando IF

Para mas ayuda, ver:
Declarando variables con DIM
Fijando Variables

Haciendole cosas a bits individuales de variables
- SET
- ROTATE

Chequeando variables y haciendo diferentes cosas basadas en sus valores:

IF
DO
FOR
Conditions

Arreglos

Acerca de Arreglos
Un arreglo (array) es un tipo de variable especial - una que puede almacenar varios valores a la vez. Es esencialmente una lista de números en la que los cuales pueden ser apuntados individualmente a traves del uso de un"indice". El índice es el valor en paréntesis inmediatamente despues del nombre del arreglo.

Ejemplos de nombres de arreglos son:

Arreglo/Indice	Significado
Pescado(10)	Elemento 10 de un arreglo llamado Pescado
DataLog(2)	El segundo numero en el arreglo llamado DataLog
ListadeBotones(Temp)	Un elemento en el arreglo "ListadeBotones" que es seleccionado de acuerdo al valor de la variableTemp"

Usando arreglos
Para usar un arreglo, se especifique su nombre, despues su indice. Los arreglos pueden ser usados en cualquier parte donde una variable normal pueda usarse.

Para mas ayuda, vea:
Declarando arreglos con DIM

Condiciones

En GCBASIC (y la mayor parte de los lenguajes de programación) una condicional es un palabra la que puede ser verdadera o falsa. Las condicionales son usadas cuando un programa debe hacer una decision..
Una condicional es generalmente dada como un valor o variable, un operador relativo (como as = or >), y otro valor o variable. Varias condicionales pueden ser conbinadas para formar una sola condicional a traves del uso de operadores logicos como AND y OR.

GCBASIC soporta estos operadores relativos.

Simbolo	**Significado**
=	Igual
<>	No Igual
<	Menor que
>	Mayor que
<=	Menor o Igual que
>=	Igual o mayor que

En adicion, estos operadores logicos pueden ser usados para combinar varias condicionales en una:

Nombre	**Abreviacion**	**Condicion verdadero si**
AND (si)	&	ambas condiciones son verdaderas
OR (o)	|	al menos una condicion es verdadera
XOR	#	una condicion es verdadera
NOT (no)	!	la condicion no es verdadera

NOT es un poco diferente de los otros operadoras logicos, En la que solo necesita otra condición. Otros operadores aritméticos pueden ser combinados en condicionales, para cambiar valores antes de que sean comparados, por ejemplo.

GCBASIC tiene dos condicionales hechas en el - TRUE (verdadero), el cual siempre es verdad, and FALSE (falso), el cual siempre es falso. Esos pueden ser usados para crear lazos infinitos.

Es tambien posible probar bits individuales en condiciones. Para hacer esto, especifique el bit a probar, entonces 1 or 0 (o ON y Off) respectivamente). En el presente no hay forma de combinar prueba de bits con otras condiciones - NOT, AND, OR y XOR no trabajaran.

Ejemplo de condicionales:

Condicion	**Comentarios**
Temp = 0	Condición es true si Temp = 0
Sensor <> 0	Condición es true si Sensor no es igual a 0
Lectura1 > Lectura2	True si Lectura1 es mayor que Lectura2
Modo = 1 AND Tiempo > 10	True si Modo es 1 y Tiempo es mayor que 10
Calor > 5 OR Humo > 2	True si Calor es ,mayor que 5 o Humo es mayor que 2

Luz >= 10 AND (NOT Tiempo > 7)	True si Luz es mayor o igual que 10, y Tiempo si es 7 o menos
Temp.0 ON	Verdadero si Temp bit 0 es on

Subrutinas

Acerca de Subrutinas

Una subrutina es un programa pequeño dentro del programa principal. Las subrutinas son usadas tipicamente cuando una operacion necesita ser repetida varias veces en diferentes partes del programa principal.

Hay dos usos principales para las subrutinas:
- Mantener los programas claros y faciles de leer.
- Reducr el tamaño de los programas permitiendo secciones comunes ser reusadas.

Cuando el programa del PIC llega a una subrutina el salva su localizacion en el presente programa antes de saltar al comienzo de esta, o llama la subrutina. Una vez que alcanza el final de la surutina el retorna al programa principal, y continua corriendo el codigo que dejo previamente.

Normalmente, es posible que las subrutinas puedan llamar otras subrutinas. Entonces hay limites al numero de veces que una subrutina pueda llamar a otra, esto varia de chip en chip, como se observa en la tabla siguiente:

Familia dePICs	Ancho de Instruccion	Numero de subs llamadas
10F*, 12C5*, 12F5*, 16C5*, 16F5*	12	1
12C*, 12F*, 16C*, 16F*, except those above	14	7
18F*, 18C*	16	31

Esos limites son debidos al monto de memoria en el PIC el cual salva la localizacion antes de brincar a una nueva subrutina. Algunos comandos de GCBASIC son subrutinas, de manera que usted podria permitirse de 2 a 3 llamadas a subrutinas mas de la que su programa pueda tener.

Otra comodidad de las subrutinas es que ellas se permiten aceptar parametros. Esos son valores que son pasados desde le programa principal a la subrutina cuando ella es llamada, y son retornados cuando la subrutina finaliza.

Usando Subrutinas

Llamar a una subrutina es muy sencillo - todo lo que se necesita es el nombre de la subrutina, y una lista de parámetros. Este código podra llamar a una subrutina llamada "Buzz" que no tiene parametros:

```
Buzz
```

Si la subrutina tiene parametros, entonces ellos deberan ser listados despues del nombre de la subrutina. Este podria ser el comando para llamar una subrutina llamada "MueveBrazo" esta tiene tres parametros:

```
MueveBrazo NuevaX, NuevaY, 10
```

Crear una subroutina casi siempre es tan simple. Debera haber una linea al comienzo la cual tiene "sub ", y entonces el nombre de la subrutina. Tambien, se necesita tener una linea al final de la subrutina que debe decir: "end sub". Para crear una subrutina llamada "Buzz", este es el codigo

requerido:

```
sub Buzz

'codigo para la subrutina va aqui

end sub
```

If the subroutine has parameters, then they need to be listed after the name. For example, to define the "MueveBrazo" sub used above, use this code:

```
sub MueveBrazo(BrazoX, BrazoY, BrazoZ) #NR

el codigo para la subrutina va aqui

end sub
```

En la subrutina de arriba, BrazoX, BrazoY and BrazoZ son todas variables. si la llamada de arriba es usada, las variables deberan tener esos valores al comienzo de la subrutina:

- BrazoX = NuevoX
- BrazoY = NuevoY
- BrazoZ = 10

"#NR" significa "No Return" (No Devolver). si es especificado, entonces GCBASIC no copiara los valores de los parametros hacia atras dentro de las viariables originales cuando la subrutina finaliza. Si "#NR" es removido del final de la linea, entonces GCBASIC copiara los valores hacia atras - en el ejemplo, NuevoX podria ser puesta como BrazoX y NuevoY tomaria el valor de BrazoY cuando la subrutina finalize.

Es posible usar variables tipo word (16-bit) como parámetros para las subrutinas - justo añada "As Word" al final del nombre del parámetro. Por ejemplo:

```
sub MueveBrazo(BrazoX As Word, BrazoY As Word, BrazoZ As Word)
...
end sub
```

Vea Tambien:
Funciones

Funciones

Acerca de Funciones
Funciones son un tipo especial de subrutina que puede retornar un valor. Esto significa que cuando el nombre de la función es usado en el lugar de la variable. GCBASIC llamara la función, tomara un valor de ella, y entonces pondrá el valor en la linea de código en el lugar de la variable.

Las funciones tambien sirven como parámetros - estos son tratados exactamente de la misma forma que los parámetros de las subrutinas.

Usando las Funciones

Este programa usa una función llamada PromedioAD para tomar dos lecturas análogas, y luego hacer una decision basada en el promedio.

```
'Seleccionar chip (pic)
#chip 16F88, 20

'Definir puertos
#define LED PORTB.0
#define Sensor AN0

'Direccionar  puertos.
dir LED out
dir PORTA.0 in

'Codigo principal
Main:
   set PORTB.0 off
   If PromedioAD > 128 Then set PORTB.0 on
   wait 10 ms
goto Main

function PromedioAD
'Get 2 readings
Temp = ReadAD(Sensor)
Temp = Temp + ReadAD(Sensor)

'Dividir por 2 usando rotate
rotate Temp right

'Retornar el  valor
PromedioAD = Temp

end function
```

Vea Tambien:
Subrutinas

Tablas de búsqueda

Acerca de Lookup Tables (tablas de busqueda)
una lookup table es una lista de valores que son almacenados en la memoria del programa dentro del chip (pic), los cuales pueden ser accedidos usando el comando: ReadTable.

La ventaja de lookup tables es que ellas son mas eficientes, comparadas a su equivalente del grupo de IF statements.

Usando Lookup Tables
Primero, la tabla debe ser creada. El código para crear una lookup table debe ser creado. El código para crear una lookup table es simple - una linea que tiene "Table" y el nombre de la tabla, una lista de numeros(hasta 254), y luego "End Table".

Una vez que la tabla es creada, el comando ReadTable es usado para leer la data de el. El comando ReadTable requiere el nombre de la tabla a ser leida, la localizacion del elemento a ser leido, y la variable que almacenara el numero cargado.

El elemento 0 de una tabla de busqueda (lookup table) almacena el tamaño de la tabla. si el comando ReadTable intenta leer mas alla del final de la tabla, un comportamiento inpredecible resultara.

Para mas ayuda, vea:
ReadTable

Scripts

ReadAD

Sintaxis:

variable = ReadAD (*puerto*)

Explicacion:

ReadAD es una función que puede ser usada para leer el convertidor analógico-digital interno que muchos PIC incluyen, *puerto* podria ser AN0, AN1, AN2, etc., hasta el número de entradas analogas conseguidas en el PIC en uso. Refiérase a la hoja de data del PIC para encontrar el número de puertos asequibles.

Ejemplo:

Este programa encenderá un LED conectado a PORTB.0 cuando el convertidor analógico digital (AD) en el pin AN0 devuelva un valor mayor de 120

```
dir PORTA.0 in
dir PORTB.0 out

set PORTB.0 off
if  ReadAD(AN0) > 120 then set PORTB.0 on
```

ADOff

Syntaxis:

ADOff

Explicación:

ADOff apaga el módulo de conversion analogico/digital del PIC, y pone todos los puertos análogos en modo digital.

Este comando deberia ponerse si usted experimenta problemas con puertos A o E.

Example:

```
'Este programa encendera un LED conectado al PORTA.1
#chip 16F877A, 20

'Pone todos los pines en puertos A y E a modo digital.
ADOff

'Enciende puerto A bit 1
SET PORTA.1 on
```

EPRead

Sintaxis:
EPRead (*lugar, lee*)

Explicación:
EPRead es usado para leer información de los datos del EEPROM, con los que muchos PIC's son equipados. *lugar* representa el sitio para leer la data y varía de chip a chip, *lee* representa la variable en la cual se almacenará la data que se va a leer del EEPROM.
Ejemplo:

```
'Este programa leerá y chequeará un valor del EEPROM
EPRead(0, Temp)
IF Temp < 255 THEN SET green ON
IF Temp = 255 THEN SET red ON
```

Para mas ayuda, vea:
EPWrite

EPWrite

Sintaxis:

EPWrite (lugar, datos)

Explicacion:

EPWrite es usado para escribir información al lugar de data del EEPROM, de manera que puedan acccederse mas tarde en la PC por un programmador, o por el comando EPRead. lugar representa el sitio para leer los datos, que varia de un PIC a otro, datos son los datos que deben ser escritos al EEPROM, y puede ser un valor o una variable.

Ejemplo:

```
'Este programa hará una medida de la entrada analogica 2 cada segundo,
y entonces escribira la lectura en el  EEPROM para acceso futuro.
DataPosition = 1
Principal:
   EPWrite(DataPosition, ReadAD(AN2))
   DataPosition = DataPosition + 1
goto Principal:
```

Para mas ayuda vea:

EPRead

ProgramErase

Sintaxis:

ProgramErase(lugar)

Explanation:

ProgramErase borra la información de la memoria de programa en PICs que tienen esta cualidad. El valor mas grande posible para *lugar* depende del monto de memoria de programa en el PIC usado, el cual es dado en la hoja de data del mismo.

Este comando debera ser llamado antes de escribir un bloque de memoria. es lento en comparación con otros comandos de GCBASIC. Note que el borra la memoria en bloques-de-32-bytes. Vea la hoja de data relevante. para mas información.

Este es un comando avanzado el cual deberia ser usado por desarrolladores de programa avanzados. Debe tenerse mucho cuidado con este comando, por que podria facilmente borrar el programa que esta corriendo en el PIC.

Ejemplo:

Para mas ayuda, ver:

ProgramRead
ProgramWrite

ProgramRead

Sintaxis:
ProgramRead (*lugar, leer*)

Explanation:
ProgramRead lee información desde la memoria de programa, en PICs que soportan esta cualidad..*lugar* y *leer* son ambas variables *word,* significa que ellas pueden leer valores por encima de 255,

El valor mas grande posible para *lugar* depende del monto de memoria de programa en el PIC, lo cual es dado en la hoja de data, leer es 14 bits de ancho, y por eso puede guardar valores hasta de 16383.
Este es un comando avanzado el cual deberia ser usado por desarrolladores avanzados.

Ejemplo:

Para mas ayuda, vea:
ProgramErase
ProgramWrite

ProgramWrite

Syntax:
ProgramWrite (*lugar, valor*)

Explanation:
ProgramWrite escribe información a la memoria de programa en chips que soportan esta cualidad. *lugar y valor* son ambas variables word

El valor mas grande posible para *lugar* depende del monto de memoria de programa en el PIC, lo cual es dado en la hoja de data. *valor* es 14 bits de ancho y por eso puede almacenar valores de hasta 16383.

Este es un comando avanzado el cual podria ser usado solo por programadores avanzados. ProgramErase debe ser usado para limpiar un bloque de memoria antes que ProgramWrite sea llamado.

Ejemplo:

Para mas ayuda, vea:
ProgramErase
ProgramRead

CALL EVERY

Todavia no implementado en GCBASIC

DO

Sintaxis:

DO [{WHILE | UNTIL} *condición*]
...
codigo de programa
...
LOOP [{WHILE | UNTIL} *condición*]

Explicación:

El comando DO causará que el codigó entre el DO y el LOOP corra repetidamente mientras la condición sea verdadera o hasta que la condición sea verdadera, dependiendo de que WHILE (mientras) o UNTIL (hasta) hayan sido especificados.

Note que el WHILE o UNTIL y la condición pueden ser especificadas a la vez, o no. Si ellas no son especificadas, el código correra repedidamente y sin fin.

Ejemplo:

```
'Este código hará parpadear una luz hasta que el boton sea presionado.

#define BOTON PORTA.0
#define luz PORTB.0

dir BOTON in
dir luz out

DO UNTIL BOTON PRESSED
   SET luz ON
   WAIT 1 s
   SET luz OFF
   WAIT 1 s
LOOP
```

Para mas ayuda, vea:

Condiciones

END

Sintaxis:

END

Explicación:

Cuando el comando END es usado, el programa para de correr.

Example:

```
    'Este programa encendera la luz roja, pero no la verde.
SET roja ON
END
SET verde ON
```

EXIT SUB

Sintaxis:

EXIT SUB

Explicación:

Este comando hará que el programa salga de la subroutina que se está ejecutando, como si llegara al final de la subrutina.

Ejemplo:

```
'Subrutina de Alarma para Ladrones
SUB Ladron(sensor)
    IF sensor = clear THEN EXIT SUB
    SET buzzer ON
    SET LIGHT ON
END SUB
```

FOR

Sintaxis:

FOR *contador* = *comienzo* TO *final* [STEP *incremento*]
...
codigo del programa
...
NEXT

Explicacion:

El comando FOR es ideal para situaciones donde una pieza de código necesita ser corrida un número de veces, y donde es necesario observar cuantas veces el código ha corrido. Cuando el comando FOR se empieza a ejecutar, *contador* es fijado a *comienzo, entonces cada vez que el programa hace un* lazo*, incremento* es añadido a *contador,* hasta que *contador* es igual a *fina*l. Entonces, el programa continua mas allá de NEXT.

STEP e *incremento son opcionales. si STEP no es especificada,* GCBASIC incrementara en 1 cada vez que el código se corra.

Ejemplo:

```
    'Este código hará destellar una luz verde 6 veces.

#chip 16F88, 20

#define LED PORTB.0
dir LED out

FOR LoopCounter = 1 to 6
   SET LED ON
   WAIT 1 s
   SET LED OFF
   WAIT 1 s
NEXT
```

Vease tambien:

REPEAT

GOSUB

Syntax:
GOSUB *Etiqueta*

Explicación:
El comando GOSUB es usado para saltar a una *Etiqueta* (como una subrutina), en una forma similar al GOTO. La diferencia es que RETURN puede ser usado para volver a la linea de código que sigue al GOTO. Es funcionalmente idéntica al comando CALL en assembler.

Ejemplo:

```
        'Este programa hara destellar un LED en PORTB bit 0 y sonará un beep
      'en PORTA bit 4, hasta que el robot se apague.
#chip 16F628A, 4     'Cambie esto según su PIC
#define Soundout PORTA.4

LazoPrincipal:
   'Encender Luz
   SET red ON
   WAIT 1 s
   SET red OFF
   WAIT 1 s

   'Sonido
   GOSUB TocarSonido
GOTO LazoPrincipal

TocarSonido:
 Tone 200, 10
 Tone 100, 10
RETURN
```

GOTO

Sintaxis:

GOTO etiqueta

Explanation:

El comando GOTO hará que el robot brinque a la línea especificada y continúe corriendo el programa desde allí. El comando GOTO es útil para hacer lázos infinitos.

Para definir una etiqueta, ponga el nombre de la etiqueta, solo en una línea y agregue dos puntos (:) al final

Ejemplo:

Este programa prenderá PORTB bit 0 hasta que el robot sea apagado.
Observe que el nombre de la etiqueta es LazoPrincipal

```
#chip 16F628A, 4 'Cambie esta linea para adaptar a su circuito.

LazoPrincipal:
   SET PORTB.0 ON
   WAIT 1 s
   SET PORTB.0 OFF
   WAIT 1 s
GOTO LazoPrincipal
```

IF

Sintaxis:

Forma corta*:*
IF *condición* THEN *comando*

Forma Larga*:*
IF *condición* THEN
...
código de programa
...
END IF

Explicación:

Se hace referencia al programa listado abajo.

El comando IF es el comando más comunmente usado para tomar decisiones. Si *condición* es true (verdadero), then (entonces) *comando (Forma corta)* o *codigo de programa (Forma larga)* correran, si la *condicion* es falsa (false), entonces el robot brincara al código localizado en la proxima línea (forma corta) o después del END IF (forma larga)..

Ejemplo:

```
'Alarma para ladrones
Comenzar:
   IF door = locked THEN SET green ON       'Forma corta
   IF door = broken THEN                    'Forma larga
      SET green OFF
      SET red ON
      SET buzzer ON
   END IF
GOTO Comenzar
```

Para mas ayuda, vea:

Condiciones

REPEAT

Sintaxis:

REPEAT *veces*
...
codigo del programa
...
END REPEAT

Explicación:

El comando REPEAT es ideal para situaciones donde una pieza de código necesita ser corrida un número de veces. Usa menos memoria y corre mas rápido que el comando FOR, y debería ser usado cuando no sea necesario contar cuantas veces el código ha corrido.

Ejemplo:

```
'Este código encenderá una luz verde 6 veces.

#chip 16F88, 20  ' cambie esta linea de acuerdo al PIC usado en su circuito

#define LED PORTB.0
dir LED out

REPEAT 6
   SET LED ON
   WAIT 1 s
   SET LED OFF
   WAIT 1 s
END REPEAT
```

Vea tambien:

FOR

SELECT

Sintaxis:

```
SELECT CASE variable

CASE valor1
   codigo1

CASE valor2
   codigo2

CASE ELSE
   codigo3

END SELECT
```

Explicación:
La estructura de control de SELECT CASE es usada para seleccionar y correr una sección particular de código, basado en el valor de *variable.*
Si *variable* iguala a *valor1* entonces *codigo1* correra. Una vez que *code1* ha corrido, el PIC saltara al comando END SELECT y continuara corriendo el programa. Si ninguna de las otras condiciones es TRUE (verdadera), entonces el codigo bajo la seccion CASE ELSE correra.

CASE ELSE es opcional, y el programa funcionara correctamente sin el.

es importante notar que **solo una sección de codigo correra** cuando se use SELECT CASE.

Ejemplo:

```
'Programa para leer el valor desde un potenciómetro y presentar una palabra
'basada en el resultado

#chip 18F4550, 20

'Conexiones para usar LCD, (Pantalla de Cristal Liquido)
#define LCD_IO 4
#define LCD_DB4 PORTD.4
#define LCD_DB5 PORTD.5
#define LCD_DB6 PORTD.6
#define LCD_DB7 PORTD.7
#define LCD_RS PORTD.0
#define LCD_RW PORTD.1
#define LCD_Enable PORTD.2

DIR PORTA.0 IN

do
   Temp = ReadAD(AN0) / 20
   cls
   select case Temp

      case 0
         Print "Nada!"
      case 1
```

```
            Print "Uno"
        case 2
            Print "Dos"
        case 3
            Print "Tres"
        case 4
            Print "Cuatro"
        case 5
            Print "Cinco"
        case else
            Print "Seis!"
    end select

            wait 250 ms
loop
```

Wait

Sintaxis:

Retardo de Largo Fijo: WAIT unidad de tiempo
Retardo *Condicional:* WAIT {WHILE | UNTIL} *condición*

Explicación:
El comando WAIT
causará que el programa espere por cualquiera de los montos de tiempo especificados (como ej. 1 segundo), o si una condición *mientras/hasta* es verdadera.

Cuando se usa un retardo de largo-fijo, hay una variedad de unidades, como:

Unidad	Largo de unidad	Rango de retardo
us	1 microsegundo	1 us - 255 us
10us	10 microsegundos	10 us - 2.55 ms
ms	1 millisegundo	1 ms - 255 ms
10ms	10 millisegundos	10 ms - 2.55 s
s	1 segundo	1 s - 255 s
m	1 minuto	1 min - 255 min
h	1 hora	1 hour - 255 hours

Las unidades de 10us y 10ms existen para trabajar alrededor del límite de las variables de 255.

Ejemplo:

```
'Este código esperará mientras un botón sea presionado, entonces
'se encendera una luz cada medio segundo.

Linterna:
   SET verde ON      'Enciende la luz
   WAIT 50 10ms            'Espera durante 50 x 10 (500) milliseconds
   SET verde OFF            'Apaga la luz
   WAIT 50 10ms            'Espera durante 50 x 10 (500) milliseconds
GOTO Linterna
```

Para mas ayuda, vea:
Condiciones

LCD
Relevant Constants

Esas constantes son usadas para el control de puesta a punto de las rutinas para el Display de Cristal Liquido (LCD) incluidas en GCBASIC. Para ponerlas, ponga una línea en el programa que diga #define para asignar un valor a la constante particular. **ej: #define LCD_IO 8**

Algunas constantes se requieren para modos de 4 y 8 bit, algunas son requeridas para el modo de 4 bits y otras para el modo de 8 bits. Cuando se usa el modo de 2 bits, solamente tres constantes deben ser colocadas - todas las otras deben ser ignoradas. Chequee la columna "Modos" para determinar si puede usar esa constante.

Nombre de Constante	Controles	Valor de normal	Modos
LCD_IO	El el modo E/S. Puede ser de 2, 4 ó 8.	8	2, 4, 8
LCD_DB	El pin de data usedo en modo de 2-bit.	N/A - ***Debe ser Puesto***	2
LCD_CB	El pin de reloj usedo en modo de 2-bit.	N/A - ***Debe ser Puesto***	2
LCD_RS	Specifica el pin de salida que es conectado al Register Select en el LCD.	N/A - ***Debe ser Puesto***	4, 8
LCD_RW	Specifica el pin de salida que es conectado al Read/Write en el LCD.	N/A - ***Debe ser Puesto***	4, 8
LCD_Enable	Specifica el pin de salida que es conectado al Enable en el LCD.	N/A - ***Debe ser Puesto***	4, 8
LCD_DATA_PORT	Puerto de salida usado para conectar al LCD data bus	N/A - ***Debe ser Puesto***	8 solamente
LCD_DB4	Pin de salida usado para conectar con el bit 4 del LCD data bus	N/A - ***Debe ser Puesto***	4 solamente
LCD_DB5	Pin de salida usado para conectar con el bit 5 del LCD data bus	N/A - ***Debe ser Puesto***	4 solamente
LCD_DB6	Pin de salida usado para conectar con el bit 6 del LCD data bus	N/A - ***Debe ser Puesto***	4 solamente
LCD_DB7	Pin de salida usado para conectar con el bit 7 del LCD data bus	N/A - ***Debe ser Puesto***	4 solamente

PRINT

Sintaxis:

PRINT *arreglo()*

Explicación:

El comando PRINT hará ver en el LCD, el contenido de un arreglo dado, empezando en la posición del cursor. *arreglo()* debera ser declarado como *string* antes de llamar este comando.

Ejemplo:

```
'Un programa 'Hola Mundo' para GCBASIC

'Configuracion general del PIC
#chip 16F877A, 20
#mem 368

'Conexiones del LCD
#define LCD_IO 8
#define LCD_DATA_PORT PORTC
#define LCD_RS PORTD.0
#define LCD_RW PORTD.1
#define LCD_Enable PORTD.2

PRINT "Hola Mundo"
```

Para mas ayuda vea:

Constantes Relevantes

LOCATE

Sintaxis:

LOCATE *linea, columna*

Explicación:

El comando LOCATE es usado para mover el cursor en el LCD a una localización dada.

Ejemplo:

```
'Un programa de "Hola Mundo" para  GCBASIC.
'Usa LOCATE para presentar "Mundo" en la segunda linea.

'Configuracion general de hardware
#chip 16F877A, 20   ' o el que usted este usando
#mem 368

'fijar conexiones del LCD
#define LCD_IO 8
#define LCD_DATA_PORT PORTC
#define LCD_RS PORTD.0
#define LCD_RW PORTD.1
#define LCD_Enable PORTD.2

'Rutina Principal
PRINT "Hola"
LOCATE 1, 5
PRINT "Mundo"
```

Para mas ayuda, vea:

Constantes relevantes

PUT

Sintaxis:

PUT Linea,Columna, Caracter

Explicación:

El comando PUT escribe el caracter ASCII dado a su localización en el LCD.

Ejemplo:

```
'A scrolling star for GCBASIC.

'Misc Settings
#define ScrollDelay 250 ms

'General hardware configuration
#chip 16F877A, 20
#mem 368

'LCD connection settings
#define LCD_IO 8
#define LCD_DATA_PORT PORTC
#define LCD_RS PORTD.0
#define LCD_RW PORTD.1
#define LCD_Enable PORTD.2

'Main routine
        FOR StarPos = 0 to 16
              If StarPos = 0 then
                PUT 0, 16, 32
         PUT 0, 0, 42
              end if
       if StarPos <> 0 then
          PUT 0, StarPos-1, 32
          PUT 0, StarPos, 42
             end if
             Wait ScrollDelay
         NEXT
```

Para mas ayuda, vea:

Costantes Relevantes

GET

Sitaxis

var = GET Linea Columna

Explicacion:

El comando GET funciona leyendo el código del carácter ASCII de un lugar en el LCD.

GET no se puede usar en el modo de 2 bits.

Para mas ayuda, vea:

PUT

Constantes Relevantes

CLS

Sintaxis:

CLS

Explicación:

El comando CLS limpia el contenido del LCD, y devuelve el cursor a la esquina superior izquierda de la pantalla.

Ejemplo:

```
        'Un programa Hola Mundo pulsante para GCBASIC
'Configuracion general del PIC
#chip 16F877A, 20    'U otro PIC cualquiera que soporte el numero de pines
#mem 368

'conexiones del LCD
#define LCD_IO 8
#define LCD_DATA_PORT PORTC
#define LCD_RS PORTD.0
#define LCD_RW PORTD.1
#define LCD_Enable PORTD.2

'Routina principal
Aqui:
 PRINT "Hola Mundo"
 Wait 1 sec
 CLS
 Wait 1 sec
goto Aqui
```

Para mas ayuda, vea:

Constantes Relevantes

LCDInt

Sintaxis:

LCDInt *valor*

Explicación:

El comando LCDint presentará el valor especificado en el LCD, en la corriente posición del cursor.

LCDInt es limitado a números entre 0 and 255 - para números mas grandes, por favor, use LCDWord.

Example:

```
'Un programa Medidor de Luz.

'Configuracion General del PIC
#chip 16F877A, 20
#mem 368
#define SensorDeLuz AN0

'Conexiones del LCD, seteo
#define LCD_IO 8
#define LCD_DATA_PORT PORTC
#define LCD_RS PORTD.0
#define LCD_RW PORTD.1
#define LCD_Enable PORTD.2

CLS
PRINT "Medidor de Luz:
locate 1,2
PRINT "Demo de GCBASIC"
wait 2 s

do while true
 CLS
 PRINT "Nivel de Luz"
 LCDInt(ReadAD(SensorDeLuz))
 wait 25 10ms
loop
```

Para mas ayuda, vea:

Constantes Relevantes

LCDHex

Sintaxis:

LCDHex *valor*

Explicación:

El comando LCDHex presentara el valor especificado en el LCD, en la posicion corriente del cursor, en formato hexadecimal.

Ejemplo:

'Un programa para contar del 1 al FF en una pantalla LCD.

```
'Configuracion general del hardware
#chip 16F877A, 20
#mem 368

'declaración de conexiones del LCD
#define LCD_IO 8
#define LCD_DATA_PORT PORTC
#define LCD_RS PORTD.0
#define LCD_RW PORTD.1
#define LCD_Enable PORTD.2

For Contador = 1 to 255
cls
LCDHex Contador
wait 25 10ms
next
```

LCDWord

Sintaxis:

LCDWord *valor*

Explicación:

El comando LCDWord presentará el valor especificado en el LCD, en la posicion corriente del cursor. LCDWord puede aceptar entradas de variables de 8 y 16 bits. permitiendo presentar valores de hasta 65535.

LCDWord usa mucha mas memoria y lleva mas tiempo correr que el LCDint. Si el número mas grande a presentar en pantalla es menos que 255, por favor use LCDInt en su lugar.

Ejemplo:

Por favor vea el ejemplo del LCDInt - los dos comandos tienen una sintaxis identica, y requiere poner las mismas constantes.

Para mas ayuda, vea:

Constantes Relevantes

Constantes Relevantes

Estas constantes son usadas para poner los comandos de control del Módulo de Modulación de Ancho de Pulso en el PIC. Para ponerlas, ponga una linea en el programa principal que diga #define, para asignar un valor a la constante particular.

Note que hay dos grupos de constantes: una para Hardware PWM y otro para Software PWM, Hardware PWM requiere el modulo CCP en el PIC. - Software PWM no necesita requerimientos referentes al PIC.

Hardware PWM

Esas constantes solo son requeridas para PWMOn,- HPWM y PWMOff no requieren ninguna constante para operar.

Nombre de Constante	Controles	Valor Normal
PWM_Freq	Especifica la frecuencia de salida del módulo PWM en KHz.	38
PWM_Duty	Pone el ciclo de trabajo de la salida del módulo PWM. Dado como porcentaje.	50%

Hardware PWM es asequible solo atraves de la pata "CCP1" or "CCP" en el PIC. Esta es una limitación de hardware de los microcontroladores PIC.

Software PWM

Nombre de Constante	Controles	Valor Normal
PWM_Delay	El periodo PWM. El largo de cualquier retardo usado puede ser multiplicado por 255. Si no se especifica valor, no se insertaran retardos. dentro de la rutina del PWM.	No definido - no retardo
PWM_Out*n*	El puerto fisico en el PIC que corresponde al canal n. n puede representar 1, 2,3 o 4.	No Definido

Mas de 4 canales son posibles, pero para eso la rutina en include\lowlevel\stdbasic.h debera ser alterada.

PWMOff

Sintaxis:

PWMOff

Explicacion:

Este comando desabilitará la salida de modulo PWM en el Pic.

Ejemplo:

```
'Este programa abrira una señal de PWM de 76 Khz, con un ciclo de trabajo de
      '80%. Emitira una señal por 10 segundos, entonces parara.
#define PWM_Freq 76       'Poner frequencia en KHz
#define PWM_Duty 80       'Poner ciclo de trabajo en 80 %
PWMOn           'Encender el PWM
WAIT 10 s         'Esperar 10 segundos
PWMOff          'Apagar el PWM
```

Para mas ayuda, vea:

PWMOn

PWMOn

Sintaxis:

PWMOn

Explicación:

Este comando permitirá la salida del módulo PWM en el PIC.

Ejemplo:

```
        'Este programa generará una señal de PWM de 76 Khz, con un ciclo de trabajo
        'de 80%. Emitirá la señal por 10 segundos, luego parará.
#define PWM_Freq 76       'Poner frequencia en KHz
#define PWM_Duty 80              'Poner Ciclo de trabajo a 80 %
PWMOn           'Enciende el módulo PWM
WAIT 10 s          'Espera 10 segundos
PWMOff          'Apaga el módulo PWM
```

Para mas ayuda, vea:

PWMOff

PWMOut

Sintaxis:

PWMOut *canal, ciclo de trabajo, ciclos*

Explicación:

Este comando usa una rutina de software de PWM, incluida en GCBASIC para producir señales de PWM en el puerto seleccionado del PIC. Esta rutina no requiere del modulo PWM interno del PIC.

canal pone el canal donde la señal de PWM será generada. Esto debera ser definido previamente poniendo las constantes SoftPWM1, SoftPWM2, etc. El máximo número de canales que se tiene es 4.

ciclo de trabajo especifica el ciclo de trabajo de PWM, el cual cubre de 0 to 255. 255 corresponde a 100%, 127 a 50%, 63 a 25%, y asi sucesivamente.

ciclos es usado para poner la cantidad de pulsos PWM a suplir. Esto es útil para situaciones en la cual un pulso de largo especifico se requiera. La formula para calcular el tiempo tomado para un ciclo es:

$T_{CICLO} = (28 + 10C)T_{OSC} + 255PWM_Retardo,$

Donde C es el numero de canales usados y Tosc el el largo de tiempo necesitado para ejecutar una instruccion en el PIC (0.2 us en un PIC de 20 MHz., 1 us en un PIC de 4 Mhz). PWM_Retardo es el largo del tiempo especificado usando la constante PWM_Delay.

Ejemplo:

```
        'Este programa controla el brillo de un LED en PORTB.0
        'usando la rutina de PWM y un potenciometro.

'Seleccione el modelo de PIC
#chip 16F877A, 20

'Define el Puerto PWM
#define PWM_Out1 PORTB.0

'Define las direcciones del puerto
dir SoftPWM1 out    'PWM
dir PORTA.0 in    'Potenciometro

'Rutina principal
do
    PWM_Out1, ReadAD(AN0), 100     '100 cicles es un valor puramente 'arbitrario aqui.
loop
```

HPWM (Modulacion de Ancho de Pulso Por Hardware)

Sintaxis:

HPWM *canal, frequencia, ciclo de trabajo*

Explicacion:

Este comando permite trabajar con el módulo de hardware PWM dentro del PIC para generar una forma de onda PWM de una frecuencia y ciclo de trabajo dada. Una vez que este comando es llamado, el PWM se emitira hasta que el PWMOff sea llamado. Si usted solo necesita una frecuencia y ciclo de trabajo en particular, usted deberia usar PWMOn y las constantes PWM_Freq and PWM_Duty instead.

canal es 1 o 2, y corresponde a los pines CCP1 and CCP2 respectivamente. En chips con solo un puerto CCP, el pin CCP o CCP1 es siempre usado, y *canal* es ignorado. (deberia ser puesto en 1 de cualquier manera para permitir futuras mejoras en PICs mas poderosos.)

frequency fija la frecuencia de salida PWM. Es medida en KHz. El valor maximo permitido es 255 KHz. El valor minimo varía dependiendo de la velocidad de reloj del PIC. 1 KHz es el mínimo en PICs de 16 MHz o por debajo y 2 Khz es el mas bajo posible en PICs de 20 MHz. En situaciones que no requieran una frecuencia PWM especifica, la frecuencia PWM seria igual a aproximadamente 5/100 de la velocidad de reloj del PIC (ej. 40 Khz en un chip de 20 Mhz, 16 Khz en un chip de 8 MHz). Esto da la mejor resolución de ciclo de trabajo posible

1/500 de la velocidad del clock del PIC (ej. 40 Khz en un chip de 20 MHz., 16 Khz. en un chip de 8 MHz.). Esto da la mejor resolucion de ciclo de trabajo posible.

ciclo de trabajo especifica la relacion deseada de ciclo de trabajo en la señal de PWM, esto da un rango de 0 a 255 donde 255 es el 100% del ciclo de trabajo

Ejemplo:

```
'Este programa alterará el brillo de un LED usando
      'Hardware PWM.

'Seleccione el modelo PIC y su velocidad
#chip 16F877A, 20

'Ponga el pin CCP1 en modo de salida
DIR PORTC.2 out

'Código principal
do
   'Encienda el brillo durante  2.5 segundos
   For Brillo = 1 to 255
      HPWM 1, 40, Brillo
      wait 10 ms
   next
   'Baje el brillo durante 2.5 segundos.
   For Brillo = 255 to 1
      HPWM 1, 40, Brillo
      wait 10 ms
   next
loop
```

Para mas ayuda, vea:

PWMOff

Constants Relevantes

Para usar las rutinas de presentación de 7-Segmentos suministradas en GCBASIC, es necesario poner estas constantes.

Nombre de Constante	Controles	Valor normal
DisplayCount	Especifica el número de dígitos de 7 segmentos que se van a usar (de 1 a 4 digitos)	1
DisplayPort*n*	Controla en *n* el puerto de salida del PIC, usado para conectar a los 7 segmentos: a, b, c, d, e, f, g.	N/A - ***Debe ser puesto***
DispSelect*n*	Este comando es usado para seleccionar el display n. para dirigir la salida al digito que se debe presentar en el momento cuando varios digitos sean 'multiplexados'	ninguno

Para usar esas rutinas, los digitos de 7 segm. deben estar conectados asi:

pin del Puerto del PIC	Segmento
0	a
1	b
2	c
3	d
4	e
5	f
6	g

DisplayValue

Sintaxis:
DisplayValue (*display, datos*)

Explicación:
Este comando presentara el valor dado en un display de LEDs de 7 segmentos. display es el numero del display en uso, y data es el valor entre 0 y 9 a presentar.

Ejemplo:

```
'Este programa contará de 0 a 9 en un display de LEDs.
'El display deberá ser conectado al PORTB

#define DisplayPortA PORTB

for Contador = 0 to 9
 DisplayValue 1, Contador
 Wait 1 sec
next
```

Para mas ayuda, vea:
Constantes Relevantes

DisplayChar

Sintaxis:
DisplayChar (*display, caracter*)

Explicación:
Este comando presentará un caracter ASCII dado en un display de LEDs de 7 segmentos. *display* es el número del digito a usar, y *caracter* es el caracter ASCII a presentar.

Ejemplo:

```
'Este programa presentara "Hola" en un display de LEDs de 7 segmentos
'El display debera conectarse al PORTB

#define DisplayPortA PORTB

DIM Mensaje(10)
Mensaje() = "Hola "
for Contar = 0 to 4
 DisplayChar 1, Mensaje(Contar)
 Wait 25 10ms
next
```

Para mas ayuda, vea :
Constantes Relevantes

InitSer

Sintaxis:

InitSer (*channel, rate, start, data, stop, parity, invert*)

Explicación:

Este comando pondrá la comunicación serial. Los parámetros son como sigue:

- *channel* es 1, 2 or 3, y se refiere a los puertos en entrada/salida que son usados en para la comunicación.
- *rate* es la rata de bits, los cuales son dados por la letra r y luego la rata deseada en bps. Unidades aceptables son r300, r600, r1200, r2400, r4800, r9600 and r19200.
- *start* da el número de bits de arranque, el cual es usualmente 1. Para hacer que el PIC espere por el bit de arranque antes de proceder a recibir, añada 128 a *start*. (Nota: puede ser deseable el uso aqui de la constante WaitForStart)
- *data* dice al programa cuantos bits de data hay que enviar o recibir. En la mayoría de las situaciones eso es 8, pero puede oscilar entre 1 y 8, inclusive.
- *stop* es el numero de bits de parada. Si el bit de start (bit 7) es on, entonces este número es ignorado.
- *parity* se refiere a paridad, un sistema de prueba de errores usado por muchos dispositivos. El puede ser inpar (en lo cual debe siempre haber un numero inpar de bits altos), o par (donde el numero de bits altos siempre seran pares), o ningunos (para sistemas que no usan parity).
- *invert* puede ser cualquiera "*normal*" o "*invert*". If it in "*invert*", entonces los bits altos seran cambiados a bajos y los bajos a altos.

Ejemplo:

```
'Uso de ejemplo de comunicacion con Lego RCX:
InitSer(1,r2400,1+WaitForStart,8,1,odd,invert)
```

Para mas ayuda, ver::

Constantes Relevantes

Constantes Relevantes

Esas constantes son usadas para controlar la puesta a punto de las rutinas de comunicacion serial RS232. Para ponerlas, ponga #define en una linea en el programa y asígnele un valor a la constante en particular.

Constant Name/s	Controls	Valor Normal
Ser_Links	El número de enlaces seriales soportados. Al ponerlo a un número mas bajo, reduce el monto de codigo bajado al PIC	3
SendALow, SendBLow, SendCLow	Estos son para definir los comandos usados para enviar un bit low(0) en canales seriales A, B y C respectivamente	nada **(debe ser puesto por el usuario)**
SendAHigh, SendBHigh, SendCHigh	Estos son para definir los comandos usados para enviar un bit high(1) en canales seriales A, B y C respectivamente	nada **(debe ser puesto por el usuario)**
RecALow, RecBLow, RecCLow	La condición que es verdadera cuando un bit bajo esta siendo recibido	Sys232Temp.0 OFF **(debe ser puesto por el usuario)**
RecAHigh, RecBHigh, RecCHigh	La condición que es verdadera cuando un bit alto esta siendo recibido	Sys232Temp.0 ON **(debe ser puesto por el usuario)**

SerSend

Sintaxis:
SerSend (*canal, datos*)

Explicación:
Este comando enviará un byte dado por *datos,* usando el canal RS232 referido a *canal* de acuerdo a las reglas planteadas usando InitSer.

Ejemplo:

```
'Este programa enviará un byte usando PORTB.2, el valor que
'depende de cuando un botón es presionado.

#define RecAHigh PORTB.2 ON
#define RecALow PORTB.2 OFF

InitSer (1, r9600, 1+WaitForStart, 8, 1, none, normal)
Main:
 if Button pressed then Temp = 0
 if Button released then Temp = 100
 SerSend (1, Temp)
 Wait 1 s
goto Main
```

Para mas ayuda, ver:
Constantes Relevantes
InitSer

SerReceive

Sintaxis:
SerReceive (*canal, salida*)

Explicación:
Este comando leerá un byte del canal RS232 dado por *canal* de acuerdo
con las reglas planteadas usando InitSer. y guardar el byte recibido en la variable *salida*.

Ejemplo:

```
'Este programa leera un byte desde PORTB.2, y encenderá el Led_verde
      'si el byte es mayor de 50

#define RecAHigh PORTB.2 ON
#define RecALow PORTB.2 OFF

InitSer (1, r9600, 1+WaitForStart, 8, 1, none, normal)
SerReceive (1, Temp)
if Temp > 50 then set Led_verde on
```

Para mas ayuda, vea:
Constantes Relevantes
InitSer

SPIMode

Sintaxis:

SPIMode *Modo*

Explicación:

SPIMode pone el modo del módulo SPI dentro del PIC chip. Estos son los modos posibles SPI:

Nombre del Mode	Descripción
MasterSlow	Modo maestro, el clock SPI es 1/64 de la frecuencia del PIC.
Master	Modo maestro, el clock SPI es 1/16 de la frecuencia del PIC.
MasterFast	Modo maestro, el clock SPI es 1/4 de la frecuencia del PIC.
Slave	Modo esclavo
SlaveSS	Modo esclavo, con el pin Seleccion de Esclavo permitida.

Vea También:

SPITransfer

SPITransfer

Sintaxis:
SPITransfer *tx, rx*

Explicación:
Este comando simultaneamente envia y recibe un byte de datos usando el protocolo SPI. El se comporta diferente dependiendo de que el PIC haya sido puesto para actuar como maestro o esclavo.
Cuando se opera como master, SPITransfer iniciará una trasferencia. La data en *fx* sera enviada al esclavo, a la vez que el byte que esta almacenado en el esclavo sera leido en *rx*
When operating as a master, SPITransfer will initiate a transfer. The data in *tx* will be sent to the slave, whilst ***** que es whilst ******* the byte that is buffered in the slave will be read into *rx.*
En el modo esclavo, el comando SPITransfer pausará el programa hasta que una trasferencia sea iniciada por el maestro. En ese punto, el mandara una data en *fx* **** que es whilst **** a la vez que lee la transmision desde el maestro dentro de la variable *rx*.
In slave mode, the SPITransfer command will pause the program until a transfer is initiated by the master. At this point, it will send the data in *tx* whilst reading the transmission from the master into the *rx* variable.

Ejemplo:
Hay dos ejemplos para este comando - Uno corre en el PIC esclavo y otro en el maestro. Una lectura es tomada de un sensor en el esclavo, y enviada a traves de el maestro el cual presenta la data en la pantalla LCD.

Programa esclavo:

```
'Seleccione modelo de PIC chip y configuración
#chip 16F88, 20
#config MCLR_OFF

'Poner direcciones para los pines SPI
dir PORTB.2 out
dir PORTB.1 in
dir PORTB.4 in
'Poner direccion del pin análogo
dir PORTA.0 in

'Poner el modo del SPI a esclavo
SPIMode(Slave)

'Permitir a otro PIC, iniciar LCD
Wait 1 sec

'Lazo principal - tomar una lectura, y entonces esperar para enviarla a traves.
do
'Note que rx es 0 - esto es porque no se ha recibido data.
SPITransfer(ReadAD(AN0),0)
loop
```

Programa Maestro:

```
'Configuracion general de hardware.
#chip 16F877A, 20
#mem 368
```

```
'poner conexiones del LCD
#define LCD_IO 8
#define LCD_DATA_PORT PORTC
#define LCD_RS PORTD.0
#define LCD_RW PORTD.1
#define LCD_Enable PORTD.2

'Poner direcciones de SPI
dir PORTC.5 out
dir PORTC.4 in
dir PORTC.3 out

'Poner el modo de SPI a maestro, con reloj rápido
SPIMode(MasterFast)

'Lazo principal
do
'Leer un byte desde el esclavo
'No datos a enviar, entonces tx is 0
SPITransfer(0, Temp)

'Presentar  data
if Temp > 0 then
  CLS
  Print "Luz: "
  LCDInt Temp
  Temp = 0
end if

'Esperar para dar tiempo al LCD de presentar el valor dado.
wait 100 ms
loop
```

Constantes Relevantes

Esas constantes son usadas para controlar las rutinas de generación de tono. Para ponerlas, ponga una linea en el programa principal que use #define para asignar un valor a la constante particular.

Nombre de Constante	Controles	Valor normal
SoundOut	El pin de salida para producir el sonido.	N/A - ***Debe ser puesto***

Tone

Sintaxis:
Tone (*Frequencia, Duración*)

Explicación:
Este comando producirá el tono especificado y por la duración especificada.
Frequencia es medida en Hz, y la *Duración* es en unidades de 10 ms.

Por favor note que este comando no produce la frecuencia exacta especificada.
Mientras sea suficientemente precisa para "bips" sonidos que señalan errores y pequeñas piezas de musica monofónica, no debería usarse para nada que requiera una frecuencia precisa.

Eemplo:

```
'Programa de muestra para producir una nota DO (440 Hz)
'en el  PORTB.1

#chip 16F877A, 20
#mem 368

#define SoundOut PORTB.1
dir SoundOut OUT

do
 Tone 440, 1000
loop
```

Para mas ayuda, vea:
Constantes Relevantes

ShortTone

Sintaxis:
ShortTone (Frecuencia, Duración)

Explicación:
Este comando producira el tono especificado por la duración especificada.
Frequencia es medida en unidades de 10 Hz, y Duracion es en unidades de 1 ms.

Por favor note que este comando no produce la frecuencia exacta especificada.
Mientras sea suficientemente precisa para "bips" sonidos que señalan errores y pequeñas piezas de música monofónica, no debería usarse para nada que requiera una frecuencia precisa.

Ejemplo:

```
'Programa de muestra para producir un tono en PORTB bit 1.
'basado en la lectura de un LDR en an0 (usualmente 'PORTA bit 0).

#chip 16F88, 20

#define SoundOut PORTB.1
dir SoundOut OUT

do
 ShortTone ReadAD(an0), 100
loop
```

Para mas ayuda, vea:
Constantes Relevantes

Fijando Variables

Sintaxis:

Variable = data

Expicación:

Variable tomará el valor de data. Data puede ser cualquiera de un valor fijo (como 157), otra variable o una suma.

Si *data* es de un valor fijo, debe ser un entero entre 0 y 255 inclusive.

Si *data* es una suma, entonces podrá tener cualquiera de los siguientes operandos:

- + (Sumar)
- - (Restar)
- * (Multiplicar)
- & (y)
- | (o)
- # (xor)
- ! (no)
- = (igual)
- <> (no iqual)
- < (menor que)
- > (mayor que)
- <= (menor que o igual)
- >= (mayor que o igual)

Los 6 operandos finales son para probar condiciones. Ellos retornaran FALSO(0) - "FALSE(0)" si una condicion es falsa, o VERDADERO(255) - "TRUE(255)" si la condición es verdadera.

GCBASIC también entiende el orden de las operaciones. Si hay varios operandos presentes, ellos deben ser procesados en este orden:

1. Paréntesis
2. Multiplicación
3. Suma/Substracción
4. Y/O/Xor/No (And/Or/Xor/Not)

Hay dos modos en los cuales las variables pueden ser usadas - modo byte y modo word GCBASIC automáticamente usará un modo diferente para cada calculación, dependiendo del tipo de variable usada. Si una variable tipo byte es usada, el modo byte será seleccionado, si es una variable tipo word, el modo word sera usado; si una variable tipo byte es usada pero el cálculo envuelve números mas grandes de 255, el modo word puede ser usado añadiendo [WORD] al final de la linea.

Si usted prefiere, puede añadir "LET" al comienzo de la línea, eso no afectará al programa, pero es incluido para compatibilidad con otros dialectos BASIC.

Ejemplo:

```
'Este programa es para ilustrar como poner variables.
Cachilapo = 46        'Pone la variable Cachilapo a 46
Animal = Cachilapo    'Pone la variable Animal al  valor de la variable Cachilapo
Oso = 2 + 3 * 5       'Pone la variable Oso al resultado de 2 + 3 * 5, 17.
Oveja = (2 + 3) * 5   'Pone la variable Oveja al resultado de (2 + 3) * 5, 25.
Animal = 2 * Oso    'Pone la variable Animal al doble que el valor de Oso.
LargeVar = 321        'LargeVar debe ser puesta como word - vea: DIM.
Temp = LargeVar / 5 [WORD] 'Note el uso de [WORD] para asegurar que el cálculo
                    'sea hecho correctamente
```

ROTATE

Sintaxis:

ROTATE *variable* {LEFT | RIGHT} [SIMPLE]

Explicación:

El comando ROTATE rotará una variable en una dirección específica.

Esta tabla describe la operación del comando ROTATE:

Comando	***variable***	**STATUS.C**
Valores al comienzo:	b'00110011'	1
ROTATE variable LEFT	b'01100110'	0
ROTATE variable RIGHT	b'10011001'	1
ROTATE variable LEFT SIMPLE	b'01100110'	N/A
ROTATE variable RIGHT SIMPLE	b'10011001'	N/A

Ejemplo:

```
'Este programa rotará la variable Patron dos lugares a la derecha.
Patron  = 15
'Valor de Patron en binario: 00001111
ROTATE Patron RIGHT SIMPLE
'Valor de Patron en binario: 10000111
ROTATE Patron RIGHT SIMPLE
'Valor de Patron en binario: 11000011
```

SET

Sintaxis:

SET *variable.bit* {ON | OFF}

Explicación:

El propósito del comando SET es cambiar los bits individuales en "on" y "off" o sea encendido y apagado.

El comando SET es muy útil para controlar puertos de salida, pero también puede ser usado para conformar variables.

A menudo cuando se controlan puertos de salida, SET es usado en conjunción con constantes. Esto es porque es algunas veces visto ejemplos como "SET green ON". En este caso, green es una constante que ha sido definida en alguna parte, la cual se refiere a un puerto de salida.

Ejemplo:

```
'DESTELLAR UN LED programa de muestra para GCBASIC
'Controla un LED en el PORTB bit 0.

'Fijar modelo de PIC chip y opciones de configuracion
#chip 16F84A, 20
#mem 68
#config HS_OSC

'Poner deireción de pin
dir PORTB.0 OUT

'Rutina principal
do
 set PORTB.0 ON
 Wait 1 sec
 set PORTB.0 OFF
 Wait 1 sec
loop
```

DIM

DIM tiene dos usos, ambos envuelven variables grandes, el puede ser usado para definir arreglos "arrays", y para declarar variables mas grandes que 1 byte.

Sintaxis:

DIM *variable* AS *type* [Variables > 1 byte]
DIM *array*(*size*) [Arreglos -"Arrays"]

Explicación:

El comando de variables DIM, es usado para informar al GCBASIC de variables mas grandes de 1 byte.

El comando de arreglos DIM también dispone variables de arreglos. Cuando se usa para esto, *size* debe ser una constante de hasta 80.

Ejemplo:

```
'Este programa dispondrá un arreglo y una variable word

dim DataList(10)
dim Reading as word

Reading = 21978
DataList(1) = 15
```

DIR

Sintaxis:

DIR *port.bit* {IN | OUT} ***(Forma Individual)***
DIR *port* {IN | OUT | *DirectionByte*} ***(Forma de Puerto Entero)***

Explicación:

El comando DIR es usado para fijar la dirección de los puertos en el PIC chip. La forma individual fija la direccion de un pin a un tiempo, mientras que la forma de puerto entero fija todos los bits de un puerto.

En la forma individual, especifique el puerto y el bit (ej. PORTB.4), luego la dirección, la cual es cualquiera de IN o OUT.
I
La forma entera del puerto es similar a la instrucción TRIS usada por algunos PIC chips. Para usarla, de el nombre del puerto (ej. PORTA), y luego el byte que sera escrito en la variable TRIS. *Esta forma de comando es para aquellos que estan familiarizados con la arquitectura interna del PIC chip.*

Para la serie de chips 10/12, use la forma entera del puerto. Especifique "GPIO" o "IO" como el puerto.

Ejemplo:

```
'Este programa fija PORTA bits 0 y 1 a entrada "in", y el resto a salida "out"
      ' El también pone todo el PORTB como salida, exceptuando B1.
'la forma idividual e usada para PORTA:
DIR PORTA.0 IN
DIR PORTA.1 IN
DIR PORTA.2 OUT
DIR PORTA.3 OUT
DIR PORTA.4 OUT
DIR PORTA.5 OUT
DIR PORTA.6 OUT
DIR PORTA.7 OUT
'La forma de puerto entero es usada para B:
DIR PORTB b'00000010'

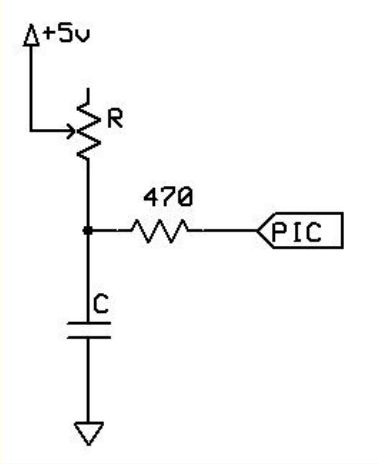
'La forma de puerto entero es useda para C:
DIR PORTC IN

      'La forma de puerto entero es usada para la serie de PICs 10/12
      DIR GPIO b'00001100'
```

POT

Sintaxis:
POT *pin, salida*

Explicación:
POT hace posible medir un elemento análogo como una resistencia, con un puerto digital, con la adición de un pequeño condensador. Este es el circuito requerido:



El comando trabaja usando el PIC para descargar el condensador, entonces mide el tiempo que le toma al condensador cargarse de nuevo atraves de la resistencia.

El valor del condensador debe ser ajustado dependiendo del tamaño de la resistencia variable. El tiempo de carga necesita ser aproximadamente 2.5 ms cuando la resistencia esta al valor máximo. Para un típico potenciómetro de 50 kohms. o LDR, un condensador de 50 nf, (.05 uF.) es requerido.

Este comando debería ser usado cuidadosamente. Cada vez que es insertado, 20 palabras de memoria de programa son usadas en el PIC chip, lo cual es alrededor de 15 veces el tamaño del comando SET.

pin es el puerto conectado al circuito. La dirección del pin debera dealt con el comando POT.

output es el nombre de la variable que recibirá el valor.

pin es el puerto conectado al circuito. La dirección del pin deberá estar de acuerdo con el comando POT.
output es el nombre de la variable que recibirá el valor.

Ejemplo:

```
'Este programa hara sonar un 'bip' cada vez que una sombra sea detectada
'Un potenciómetro es usado para ajustar el umbral.

#chip 16F628A, 4
#config INTOSC_OSC_NOCLKOUT, MCLRE_OFF

#define Ajustar PORTB.0
#define LDR PORTB.1
#define SoundOut PORTB.2

dir SoundOut out
```

```
Main:
   POT Ajustar, Umbral
   POT LDR, NivelLuz
   if NivelLuz > Umbral then
      Tone 1000, 100
   end if
goto Main
```

Vea tambien:
http://www.cvs1.uklinux.net/cgi-bin/calculators/time_const.cgi (Util para calcular el valor del condensador. (No asociado con GCBASIC.)

PULSEOUT

Sintaxis:

PULSEOUT *pin,* tiempo

Explicación:

El comando PULSEOUT pondra el pin especificado en 1, esperara por el monto de tiempo especificado y entonces pondra el pin especificado en 0 de nuevo. El pin es especificado de la misma forma que lo es para el comando SET, y el tiempo como el mismo usado en el comando WAIT.

Ejemplo:

```
'Este programa hara parpader un LED un segundo,  en GPIO.0 usando PULSEOUT
#chip 12F629, 4
#config INTRC_OSC_NOCLKOUT

DIR GPIO.0 OUT
Main:
 PULSEOUT GPIO.0, 1 sec       'Encender LED por 1 Seg.
 Wait 1 sec                        'Esperar 1 Seg. con el LED apagado
goto Main
```

PEEK

Sintaxis:

Variable_de_Salida = PEEK (*localización*)

Explicación:

El comando PEEK es usado para leer información desde el RAM dentro del PIC chip.

localización es una variable tipo 'word' y puede tener un valor de hasta 511, que es el límite del RAM en la serie 16 de PIC chips.

Ejemplo:

```
'Este programa leerá y chequeará un valor de PORTA
Temp = PEEK(5)
IF Temp.Z ON THEN SET green ON
IF Temp.Z OFF THEN SET red ON
```

POKE

Sintaxis:

POKE (*localización, Valor*)

Explicación:

El comando POKE es usado para escribir información en el RAM dentro del PIC chip.

localización es una variable tipo 'word' y puede tener un valor de hasta 511, que es el limite del RAM en la serie 16 de PIC chips.

Ejemplo:

```
'Este programa pondrá todos los pines de PORTB en alto
POKE (6, 255)
```

ReadTable

Sintaxis:

ReadTable NombreTabla*, Item, Salida*

Explicación:

El comando ReadTable es usado para leer informacion desde Tablas 'lookup tables'. *NombreTabla* es el nombre de la tabla que se va a leer, *item* es la linea de la tabla a leer, y Salida es la variable para escribir el valor copiado de la tabla.

Item es 1 para la primera linea de la tabla, 2 para segunda y asi en lo sucesivo. Item 0 da el tamaño de la tabla. Se debe tener cuidado en asegurarse que el programa no llegue a leer mas alla del final de la tabla, o efectos extraños podran observarse.

Ejemplo:

```
'Datos del PIC
#chip 16F88, 20
#config MCLR_OFF

'Hardware Settings
#define LED PORTB.0
dir LED out

'Rutina principal
ReadTable TimesTwelve, 4, Temp
set LED off
if Temp = 48 then set LED on

'Lookup table named "TimesTwelve"
Table TimesTwelve
 12
 24
 36
 48
 60
 72
 84
 96
 108
 120
 132
 144
End Table
```

Para mas ayuda, vea:

Lookup Tables

InitTimer0

Sintaxis:

InitTimer0 fuente*, prescaler*

Explicación:

InitTimer0 pondrá timer 0 de acuerdo con los valores dados. *fuente* podrá ser: Osc o Ext y *prescaler* podria ser uno de los siguientes valores dados:

- PS0_1/2
- PS0_1/4
- PS0_1/8
- PS0_1/16
- PS0_1/32
- PS0_1/64
- PS0_1/128
- PS0_1/256

Ejemplo:

Vea InitTimer1 para un ejemplo.

InitTimer1

Sintaxis:

InitTimer1(*source, prescaler)*

Explicación:

InitTimer1 pondrá timer 1 de acuerdo con los valores dados. *fuente* podrá ser: Osc, Ext o ExtOsc. y *prescaler* podría ser uno de los siguientes valores dados:

- PS1_1/1
- PS1_1/2
- PS1_1/4
- PS1_1/8

Ejemplo:

```
'Este ejemplo medirá el tiempo que un Switch esté  en on
#chip 16F819, 20
#define Switch PORTA.0

        dir Switch In
        DataCount = 0
        InitTimer1 (Osc, PS1_1/8)
Start:
        ClearTimer(1)
        Wait until Switch On
        StartTimer(1)
        Wait until Switch Off
        StopTimer(1)

        EPWrite(DataCount, TMR1H)
        EPWrite(DataCount + 1, TMR1L)

        DataCount = DataCount + 2

Goto Start
```

StartTimer

Sintaxis:

StartTimer(*TimerNo)*

Explicación:

StartTimer es usado para empezar el timer especificado. Note que no se puede usar start Timer 0 - Este timer siempre corre.

Ejemplo:

```
'Este ejemplo medirá el tiempo que un Switch esté  en on
#chip 16F819, 20
#define Switch PORTA.0

        dir Switch In
        DataCount = 0
        InitTimer1 (Osc, PS1_1/8)
Start:
        ClearTimer(1)
        Wait until Switch On
        StartTimer(1)
        Wait until Switch Off
        StopTimer(1)

        EPWrite(DataCount, TMR1H)
        EPWrite(DataCount + 1, TMR1L)

        DataCount = DataCount + 2

Goto Start
```

ClearTimer

Sintaxis:
ClearTimer(*TimerNo)*

Explicación:
ClearTimer es usado para poner a cero el timer especificado

Ejemplo:

```
'Este ejemplo medirá el tiempo que un Switch esté  en on
#chip 16F819, 20
#define Switch PORTA.0

        dir Switch In
        DataCount = 0
        InitTimer1 (Osc, PS1_1/8)
Start:
        ClearTimer(1)
        Wait until Switch On
        StartTimer(1)
        Wait until Switch Off
        StopTimer(1)

        EPWrite(DataCount, TMR1H)
        EPWrite(DataCount + 1, TMR1L)

        DataCount = DataCount + 2

Goto Start
```

StopTimer

Sintaxis:
StopTimer(*TimerNo)*

Explicación:
StopTimer es usado para parar el timer especificado. Note que no se puede usar para parar el Timer 0 - este siempre esta corriendo.

Ejemplo:

```
'Este ejemplo medirá el tiempo que un Switch esté  en on
#chip 16F819, 20
#define Switch PORTA.0

        dir Switch In
        DataCount = 0
        InitTimer1 (Osc, PS1_1/8)
Start:
        ClearTimer(1)
        Wait until Switch On
        StartTimer(1)
        Wait until Switch Off
        StopTimer(1)

        EPWrite(DataCount, TMR1H)
        EPWrite(DataCount + 1, TMR1L)

        DataCount = DataCount + 2

Goto Start
```

Teclado (keypad)
Constantes Relevantes:

Esas constantes son usadas para poner los controles necesarios para el leer el código del Teclado en GCBASIC. Para ponerlos, ponga una línea en el programa principal y usando #define asigne un valor a la constante particular.

Nombre de Constante	Controles	Valor normal
KeypadPort	El puerto en el PIC donde se va a conectar el Teclado.	N/A **(debe ser puesto por el usuario)**

Para usar esas rutinas, el teclado debe ser conectado como sigue:

Pin del puerto del PIC	Conector del Teclado
0	Fila1
1	Fila2
2	Fila3
3	Fila4
4	Columna 1
5	Columna 2
6	Columna 3
7	Columna 4

Un teclado de 3 x 3 trabajara con esas rutinas, pero todos los pines que no se usan deben ser conectados a +5 Volts.

KeypadData

Sintaxis:
variable = KeypadData

Explicacion:
Esta función retornará un valor correspondiente a la tecla que es presionada en el teclado. Note que si dos o mas teclas son presionadas, solo un valor será retornado.

variable puede tener uno de los siguientes valores:

Valor	Nombre de Constante	Tecla Presionada
0		0
1		1
2		2
3		3
4		4
5		5
6		6
7		7
8		8
9		9
10	KEY_A	A
11	KEY_B	B
12	KEY_C	C
13	KEY_D	D
14	KEY_STAR	Asterisk/Star (*)
15	KEY_HASH	Hash (#)
255	KEY_NONE	None

Ejemplo:

```
'Programa para ver en el LCD el valor de la ultima tecla presionada.
#chip 18F4550, 20

'Poner conexiones del LCD
#define LCD_IO 4
#define LCD_DB4 PORTD.4
#define LCD_DB5 PORTD.5
#define LCD_DB6 PORTD.6
#define LCD_DB7 PORTD.7
#define LCD_RS PORTD.0
#define LCD_RW PORTD.1
#define LCD_Enable PORTD.2

'Poner conecciones del Teclado.
#define KeypadPort PORTB
```

```
'Lazo principal
do
'Agarrar tecla
Temporal = KeypadData
'Si una tecla es presionada, presentarla en el LCD.
IF Temporal <> 255 Then
  CLS
  LCDInt Temporal
  Wait 100 ms
END IF
loop
```

Para mas ayuda, vea:
Constantes Relevantes

KeypadRaw

Sintaxis:

variablegrande = KeypadRaw

Explicación:

Esta función retornara un valor de 16 bits, en el que cada bit corresponde a una tecla en el teclado. Si una tecla es presionada su correspondiente bit tendra el valor de 1, y si es soltada ese bit volvera a ser 0.

Esta tabla presenta el bit correspondiente a cada tecla:

Bit	Posicion de Tecla (row, col)	Símbolo de cada Tecla
15	1,1	1
14	1,2	2
13	1,3	3
12	1,4	A
11	2,1	4
10	2,2	5
9	2,3	6
8	2,4	B
7	3,1	7
6	3,2	8
5	3,3	9
4	3,4	C
3	4,1	*
2	4,2	0
1	4,3	#
0	4,4	D

Ejemplo:

```
'Programa para ver el status de Teclado usando LEDs
#chip 16F877A, 20

'Poner conecciones del Teclado.
#define KeypadPort PORTB

'LEDs
#define LED1 PORTC
#define LED2 PORTD
dir LED1 out
dir LED2 out

'Declarar KeyStatus como variable de 16 bits
dim KeyStatus As Word
```

```
'Lazo principal
do
'Agarrar tecla
KeyStatus = KeypadRaw
'Display
LED1 = KeyStatus_H        'High Byte
LED2 = KeyStatus_L        'Low Byte
loop
```

Para mas ayuda, vea:
Constantes Relevantes

#chip

Sintaxis:

#chip *modelo, velocidad*

Explicación:

La directiva #chip es usada para especificar el modelo de chip (PIC) y velocidad a la que GCBASIC debera compilar. *modelo* es el modelo del PIC chip - algo alrededor de las lineas del "PIC16F819". *velocidad* del chip en MHz, y es requerido para los retardos y rutinas de PWM.

#config

Sintaxis:

#config *opcion1, opcion2, ... , opcionn*

Explicación:

La directiva #config es usada para especificar las opciones de configuración del chip. Hay una detallada explicación de #config en la Sección de Configuracio- nes de la ayuda

Vea Tambien:

Configuration

#define

Sintaxis:

#define *Encontrar Reemplazar*

Explicación:

#define buscará atraves del programa para *Encontrar,* y reemplazar con el valor dado para *Reemplazar*

Vea Tambien:

Defines

#ifdef

Sintaxis:

#ifdef *Constante | Valor Constante | Variable(NombreVariable)*
...
#endif

Explicación:

La directiva #ifdef es usada para permitir selectivamente secciones de código. Hay tres manera en las cuales puede usarse.
La ventaja de usar #ifdef es preferible que una serie equivalente de IF por la cantidad de código que es bajado al PIC chip. #ifdef controla que codigo es compilado y bajado. IF controla que es corrido de una vez en el PIC chip. #ifdef deberia ser usado cuando alguna vez el valor de una constante deba ser chequeado.

(Nota: El código en las siguientes secciones no se compilará, por que no se han puesto las directivas del PIC chip y los comandos DIR. Se entiende que solo son listados como ejemplo)

Permitiendo código si una constante es definida

Syntax Example:

```
#define Blink1

#ifdef Blink1
   PulseOut PORTB.0, 1 sec
   Wait 1 sec
#endif
#ifdef Blink2
   PulseOut PORTB.1, 1 sec
   Wait 1 sec
#endif
```

Este codigo hará pulsar PORTB.0, pero no a PORTB.1. Esto es debido a que Blink1 ha sido definido, pero Blink2 no. Si la linea:

#define Blink2

fuera añadida el comienzo del programa, entonces ambos pines podrian pulsar. El valor de una constante definida no es importante y puede dejarse fuera de la linea de #define.

Permitiendo ejecución de código si una constante es definida y tiene un valor dado

Ejemplo de Sintaxis

```
#define PinsToFlash 2

#ifdef PinsToFlash 1,2,3
   PulseOut PORTB.0, 1 sec
#endif
#ifdef PinsToFlash 2,3
   PulseOut PORTB.1, 1 sec
#endif
#ifdef PinsToFlash 3
   PulseOut PORTB.2, 1 sec
#endif
```

Este programa usa una constante llamada PinsToFlash que controla cuantas luces son pulsadas. PORTB.0 es pulsado cuando PinToFlash es igual a 1, 2 o 3, PORTB.1 es pulsado cuando PinToFlash iguala 2 o 3, y PORTB.2 es pulsado cuando PinsToFlash es 3.

Permitiendo código si un sistema de variables es definido

Ejemplo de Syntaxis

```
#ifdef NoVar(ANSEL)
    SET ADCON1.PCFG3 OFF
    SET ADCON1.PCFG2 ON
    SET ADCON1.PCFG1 ON
    SET ADCON1.PCFG0 OFF
#endif
#ifdef Var(ANSEL)
    ANSEL = 0
#endif
```

La sección de código de arriba ha sido copiada directamente de a-d.h. Es usado para no permitir la funcion A/D en pines, de manera que puedan ser usados en forma de puertos de entrada y salida digital. Si ANSEL no es declarada como un sistema de variables para un chip en particular, entonces el programa usa ADCON1 para controlar los modos de puerto. If ANSEL es definido, entonces el chip es nuevo y sus puertos pueden ser fijados a digital llevando ANSEL a cero.

Permitiendo código si un sistema de bits es definido

Similar a arriba, excepto con Bit and NoBit en el lugar de Var y NoVar respectivamente.

Vea Tambien:
Defines
#define

#include

Sintaxis:

#include N*ombreDeArchivo*

Explicación:

#include le dice a GCBASIC que abra otro archivo, leerl todas sus subrutinas y constantes, y luego copiarlas dentro del programa corriente.

Hay dos formas de include - absoluta y relativa.

Absoluta es usada para referirse a archivos en el directorio C:\Program Files\GCBASIC\include. El nombre del archivo es especificado entre los simbolos < y >. Por instancia, para incluir el archivo "srf04.h", la directiva es:

```
#include <srf04.h>
```

Relativa es usada para leer archivos en la misma carpeta como el programa corrientemente seleccionado. Nombres de archivos son dados incluidos entre comillas, como:

```
#include "mycode.h"
```

cuando mycode.h es el nombre del archivo que sera leido.

No es esencial que el nombre de archivo incluido termine en .h - la cosa importante es que el nombre dado a GCBASIC sea el nombre exacto del archivo a ser incluido

Aquellos que estan familiarizados con #include en assembler o C deberian tener en mente que #include en GCBASIC trabaja diferente al #include en la mayoria de los otros lenguajes - el codigo no se inserta en el lugar de #include, pero seria preferible al final del corriente programa.

#script

#int

#startup

Sintaxis:

#startup NombreSubrutina*SubName*

Explicación:

#startup es usado en archivos incluidos para insertar rutinas de inicializacion automáticamente. Si una definición o subrutina desde el archivo es usada en el programa, entonces la subrutina especificada sera llamada.

Hay algunas limitaciones en esta directiva. Puede solo ocurrir una vez dentro de un archivo, y no se pueden especificar parametros para la subrutina que esta siendo llamada.

#mem

Sintaxis:

#mem TamañoDelRam

Explicación:

La directiva #mem es usada para especificar el monto de RAM en bytes en el PIC chip seleccionado. Debe ser declarado antes que los arreglos y constantes de cadenas sean usados.

El monto de memoria en el PIC puede ser encontrado en la hoja de data en la seccion "Device Overview".

NOTA: GCBASIC 0.9.3.0 Y MAS ALTO PODRA AUTOMATICAMENTE DETERMINAR EL MONTO DE RAM EN UN PIC CHIP. ESTA DIRECTIVA ES OBSOLETA Y DEBERA SER REMOVIDA.

Solucionando Problemas

Problema	Causas Comunes	Mas Asistencia
No puedo compilar un programa	Hay un error en el programa. Es Great Cow BASIC reclamando acerca de una linea particular de código?	Foro de GCBASIC
	Great Cow BASIC no ha sido instalado correctamen- te - reinstalelo.	Foro de GCBASIC
	Hay un problema con Great Cow BASIC	Envie un 'email' con su programa a: w_cholmondeley@users.sourceforge.net