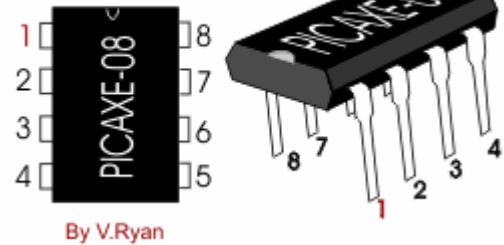


CURSO: "Sistemas de control Programado en la área de tecnología"

picaxe



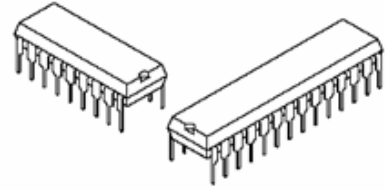
Ponentes: Marco A. López Grande/José Luis Fernández Souto

Febrero 2006

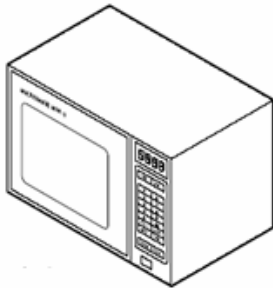
Índice:

<i>Capítulo</i>	<i>Página</i>
Introducción al sistema PICAXE	3
Tutorial 1. El sistema PICAXE	4
▪ Construcción Entrenador PICAXE-08	5
▪ Software de programación	7
Tutorial 2. Programando mediante flujogramas	11
Tutorial 3. Prácticas programación con Cyberpet	15
ANEXO1: Comandos BASIC PICAXE	24
ANEXO2: Esquemas circuitos de aplicación electrónicos	72
ANEXO3: Introducción a la programación	91

El microcontrolador PIC (microcontrolador programable) es a menudo descrito como un "ordenador en un chip". Es un circuito integrado que contiene memoria, unidades procesadoras y circuitos de entrada/salida, en una sola unidad.



Los microcontroladores son comprados en "blanco" y luego programados con un programa específico de control. Una vez programado, este microcontrolador es introducido en algún producto para hacerlo más inteligente y fácil de usar.



A manera de ejemplo, un horno de microondas puede utilizar un solo microcontrolador para procesar información proveniente del teclado numérico, mostrar información para el usuario en la pantalla y controlar los dispositivos de salida (motor de la mesa giratoria, luz, timbre y magnetrón).

Un microcontrolador puede a menudo reemplazar a un gran número de partes separadas, o incluso a un circuito electrónico completo. Algunas de las ventajas obtenidas con el uso de microcontroladores en el diseño de productos son:

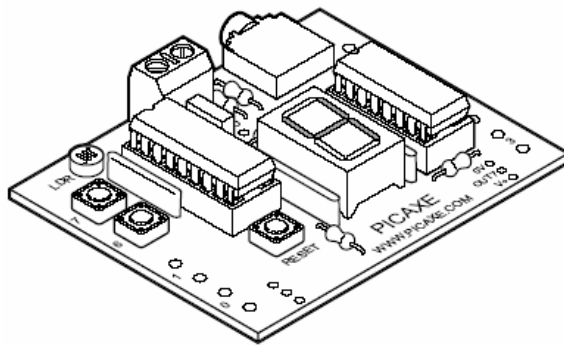
- aumento en la confiabilidad debido al menor número de partes.
- reducción en los niveles de existencia ya que un microcontrolador reemplaza varias partes.
- simplificación del ensamblaje del producto y productos finales más pequeños.
- gran flexibilidad y adaptabilidad del producto ya que las funciones del producto están programadas en el microcontrolador y no incorporadas en el hardware electrónico.
- rapidez en modificaciones y desarrollo del producto mediante cambios en el programa del microcontrolador, y no en el hardware electrónico.

Algunas de las aplicaciones que utilizan microcontroladores incluyen artefactos domésticos, sistemas de alarma, equipo médico, subsistemas de automóviles y equipo electrónico de instrumentación. Algunos automóviles modernos contienen mas de treinta microcontroladores - utilizados en una amplia variedad de subsistemas desde el control del motor hasta el cierre a control remoto!

En la Industria, los microcontroladores son usualmente programados utilizando programación en lenguaje C. Sin embargo, debido a la complejidad de este lenguaje, es muy difícil para estudiantes muy jóvenes de bachillerato el uso adecuado de dichos lenguajes.

El sistema PICAXE

El sistema "PICAXE" es un sistema de microcontrolador fácil de programar que utiliza un lenguaje BASIC muy simple, el cual la mayoría de los estudiantes pueden aprender rápidamente. El sistema PICAXE explota las características únicas de la nueva generación de microcontroladores de bajo costo FLASH. Estos microcontroladores pueden ser programados una y otra vez sin la necesidad de un costoso programador PIC.



El poder del sistema PICAXE radica en su sencillez. No necesita de ningún programador, borrador o complejo sistema electrónico - el microcontrolador es programado (con un simple programa en BASIC o un diagrama de flujo) mediante una conexión de tres alambres conectada al puerto serie del ordenador. El circuito operacional PICAXE utiliza únicamente tres componentes y puede ser ensamblado fácilmente en un tablero experimental para componentes electrónicos, en una placa corriente o en una placa PCB.

EL sistema PICAXE está disponible en dos variedades - 18 pines y 28 pines. El controlador PICAXE-28 provee 22 pines de entrada/salida (8 salidas digitales, 8 entradas digitales y 4 entradas analógicas). El sistema PICAXE-18 provee 8 salidas y 5 entradas.

Las características principales del sistema PICAXE son las siguientes:

- bajo costo, circuito de fácil construcción
- hasta 8 entradas, 8 salidas y 4 canales analógicos
- rápida operación de descarga mediante el cable serial
- Software "Editor de Programación" gratuito y de fácil uso
- lenguaje BASIC simple y fácil de aprender
- editor de diagramas de flujo incluido
- puede ser programado también mediante el software "Crocodile Technology"
- extenso número de manuales gratuitos y foro de apoyo en línea
- tablero experimental tutorial y tutoriales incluidos
- paquete de control remoto infrarrojo disponible
- paquete de servocontrolador disponible

Tutorial 1. El sistema PICAXE

El sistema PICAXE consiste en tres componentes principales:

El Software "Editor de Programación"

Este software debe ser ejecutado en un ordenador y permite utilizar el teclado del ordenador para escribir programas en un simple lenguaje BASIC. Los programas también pueden generarse dibujando diagramas de flujo. Alternativamente, el software "Crocodile Technology" puede ser utilizado para simular circuitos electrónicos completos, programándolos con diagramas de flujo. Por favor vea el apéndice de "Crocodile Technology" para mayor información.

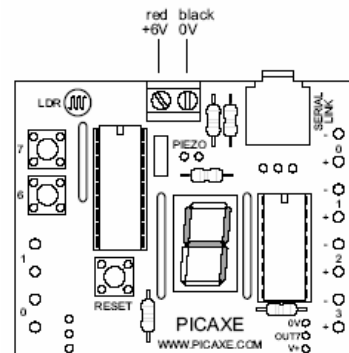
El cable serie

Este es el cable que conecta el sistema PICAXE al ordenador. El cable sólo necesita ser conectado durante la descarga de programas. No debe ser conectado cuando el PICAXE está siendo ejecutado debido a que el programa está permanentemente almacenado en el chip PICAXE - aún cuando la fuente de alimentación ha sido desconectada! Hay dos tipos de cables para descarga disponibles - al usar el tablero experimental tutorial cualquiera de los dos cables puede ser utilizado - los cuales se conectan ya sea a la cabecera de tres pines o al enchufe hembra estereo.

El chip PICAXE y el tablero electrónico

El microcontrolador PICAXE ejecuta programas que han sido descargados al mismo. Sin embargo, para operar, el chip debe ser montado en un tablero electrónico que provea una conexión al chip microcontrolador.

El tablero electrónico puede ser diseñado por el usuario en un circuito impreso, en una interfase prefabricada o, para ahorrar tiempo y por conveniencia, utilizar el tablero electrónico tutorial incluido. Este tutorial asume el uso del microcontrolador PICAXE-18 (18 pines) montado en el tablero electrónico tutorial.



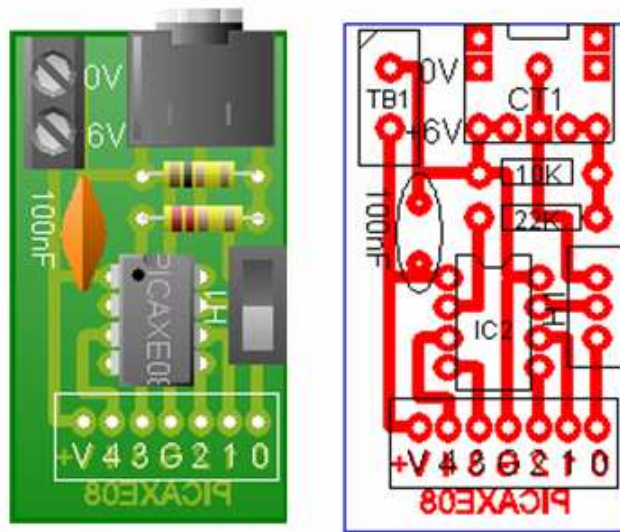
Resumen - Procedimiento de programación

1. Escriba el programa en el ordenador utilizando el software "Programming Editor".
2. Conecte el cable de descarga desde el ordenador al PICAXE.
3. Conecte el acumulador eléctrico (batería) al PICAXE.
4. Utilice el software "Editor de Programación" para descargar el programa. El cable de descarga puede ser removido posterior a la descarga.

El programa comenzará a ejecutarse en el PICAXE automáticamente. Sin embargo, el programa puede ser reiniciado en cualquier momento presionando el interruptor de reinicio.

Construcción del entrenador de PICAXE-08

1. PCB, disposición de componentes y vista real del montaje:



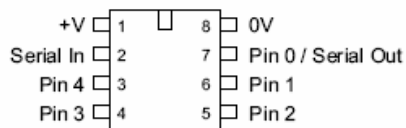
2. Listado de componentes:

Parte	Descripción
CT1	3.5mm stereo socket
TB1	Borne placa C.I. 2 terminales
R1	10k/0,25W
R2	22k/0,25W
C1	100nF polyester
IC2	Zócalo 8 pin
PIC	Microcontrolador PICAXE08
H1	Tira 3 pines
H2	Tira 7 pines
TB1	Portapilas 4xAA

3. Características PICAXE08:

Tipo de PICAXE	Nº Pines	Memoria (líneas)	Pines	Salidas	Entradas	A/D (b-baja)	Memoria Datos	Interrupciones
PICAXE-08	8	40	5	1-4	1-4	1b	128-prog	-

NOTA:

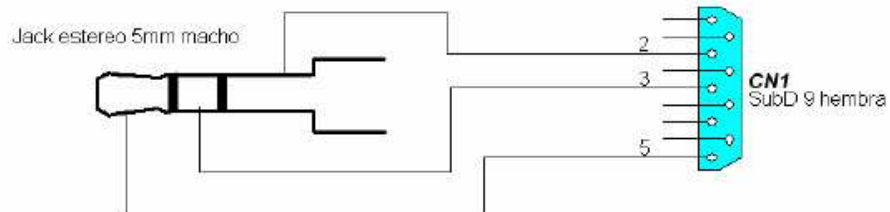


El número de líneas de memoria de programa es solo aproximado, ya que diferentes comandos requieren diferentes cantidades de memoria. El pin3 es solo de entrada y el pin0 es solo de salida. La alimentación de 3V a 5V. Se puede alimentar con 6V (4x1,5).

El conector H1 tiene un jumper que permite la programación del PICAXE-08 (jumper en parte superior) o bien poner el pin 0 como salida (jumper en parte inferior, tal y como figura en la imagen).

4. Cable de conexión:

Cable programación Picaxe (RS232 → Jack):



5. El software de programación:

El software de programación que se utilizará para programar, compilar y transferir es el diseñado por “Revolución Educación Ltd” es de uso libre (para uso exclusivo en educación).

Para instalarlo basta con ejecutar el archivo: “*Programming Editor.msi*”, o bien introducir el CDROM y seguir las instrucciones.

Antes de poder probar nuestro montaje deberemos realizar la configuración del software siguiendo los siguientes pasos:

1. Seleccionar el interface en idioma español
2. Configurar el puerto de conexión
3. Seleccionar el tipo de microcontrador (en esta primera práctica será PICAXE-08).
4. Conectaremos en cable de programación entre PC y Entrenador.
5. Escribiremos el código del programa de prueba y se lo enviaremos al microcontrolador.

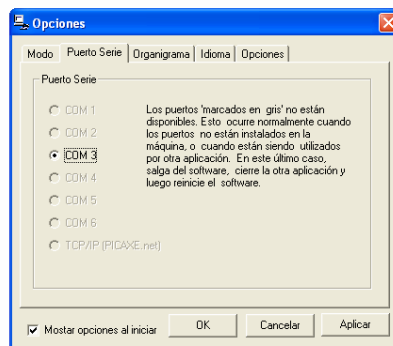
Comenzaremos con la configuración del software:

Al arrancar la aplicación aparece el cuadro de configuración (por opción) si no fuese el caso se encuentra dentro del menú principal: **Ver** → **Opciones**

En la ficha “**Idioma**” seleccionaremos el Castellano.

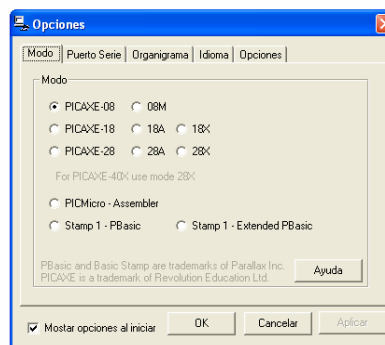


En la ficha “**Puerto Serie**” seleccionaremos el que utilizaremos (en el ejemplo es el COM3). Lo lógico es utilizar el COM1 o COM2 dependiendo si se tiene conectado ratón con conexión serie o un módem.



En la ficha “**Modo**” seleccionaremos el microcontrolador que programaremos, en nuestro caso el:

PICAXE-08.



Ahora corresponde realizar la edición del código, compilación y programación. Para poder comunicar “nuestras intenciones” al microcontrolador debemos hacerlo siguiendo el siguiente protocolo:

- Edición del código del programa: se puede realizar de dos métodos diferentes:
 - Mediante flujogramas: es un método más sencillo pero bastante limitado en sus posibilidades de programación. No

cabe la posibilidad de reutilizar parte de los flujogramas en otros programas.

- Mediante código de alto nivel como BASIC o C. El sistema PICAXE utiliza código BASIC. La ventaja de este tipo de programación es su potencia de programación y la posibilidad de reutilizar parte del código de un programa en otro, dado que muchas rutinas son válidas para muchas aplicaciones.
- Una vez editado el código del programa, bien utilizando comandos o bien flujogramas, deberemos codificarlo al lenguaje que utilizan los microcontroladores, o sea “1” y “0” . La aplicación encargada de realizar esa transformación recibe el nombre de *compilador*.
- Por último deberemos enviar esa información a la memoria de programa del microcontrolador, el dispositivo encargado de realizar esa operación recibe el nombre de *programador* o loader.

La ventaja del software y la “BIOS” que contienen los microcontroladores PIC del sistema PICAXE es que las operaciones anteriormente mencionadas las realizan la misma aplicación y de una forma sencilla sin tener que tener grandes conocimientos de ensamblador ni conocer y saber manejarlos diferentes dispositivos que permiten realizar la programación del PIC.

6. Vamos a realizar nuestra primera programación y comprobación de su correcto funcionamiento con un sencillo programa, se trata de encender y apagar un led de forma intermitente conectado a la salida de nuestra placa de prototipos. El programa lo realizaremos primeramente en código BASIC y posteriormente mediante flujogramas.

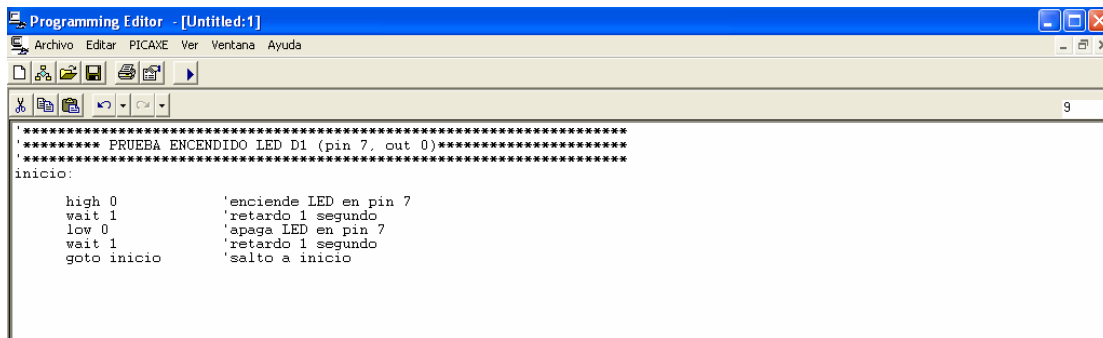
```
*****  
'***** PRUEBA ENCENDIDO LED D1 (pin 7, out 0)*****  
*****  
inicio:
```

```
high 0          'enciende LED en pin 7  
wait 1          'retardo 1 segundo  
low 0           'apaga LED en pin 7  
wait 1          'retardo 1 segundo  
goto inicio     'salto a inicio
```

Posteriormente se analizarán cada uno de los comandos que se están utilizando, que se pueden ir estudiando en el capítulo de programación.

Es necesario conectar un Led a través de una resistencia limitadora entre los terminales de salida 0 y GND y poner el jumper en función de salida (figura página 5)

Para ello arranquemos la aplicación “**Programming Editor**” y escribimos las líneas del código de la aplicación o bien utilizar la función copiar (ctrl + c) y pegar (ctrl + v) en la ventana del editor de código:



```
***** PRUEBA ENCENDIDO LED D1 (pin 7, out 0)*****  
*****  
inicio:  
  
  high 0      'enciende LED en pin 7  
  wait 1     'retardo 1 segundo  
  low  0      'apaga LED en pin 7  
  wait 1     'retardo 1 segundo  
  goto inicio 'salto a inicio
```

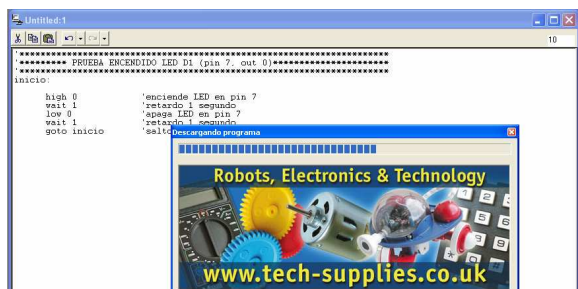
Ahora debemos compilar el código y programar el PIC, aquí es donde entra en funcionamiento el sistema tan sencillo de aprendizaje y puesta en marcha de dispositivos automatizados de forma autónoma utilizando este sistema.

Para realizar la operación, de forma conjunta, basta con tener conectado el cable de programación y el jumper en la posición de programación (jumper en los terminales de la parte inferior conmutados, tal y como figura en la página 5).

Ejecutamos en comando de compilación y programación:



También se puede ejecutar mediante el menú principal:
PICAXE → **Ejecutar** (o F5)



La transferencia del código compilado se envía a la memoria de programa del PIC visualizando su evolución mediante una barra azul (muy pocos segundos).

Se indicará el término de la transferencia así como el tamaño ocupado en la memoria interna:



Para comprobar su funcionamiento solo resta cambiar el jumper a la posición de salida del pin0, salida 7 (¡ojo con la denominación de “pin 0” y “salida 7”!, no se corresponde “Terminal físico” con “pin” definido por los creadores de PICAXE).

Si todo ha salido bien, el Led parpadeará intermitentemente con una candencia de un segundo.

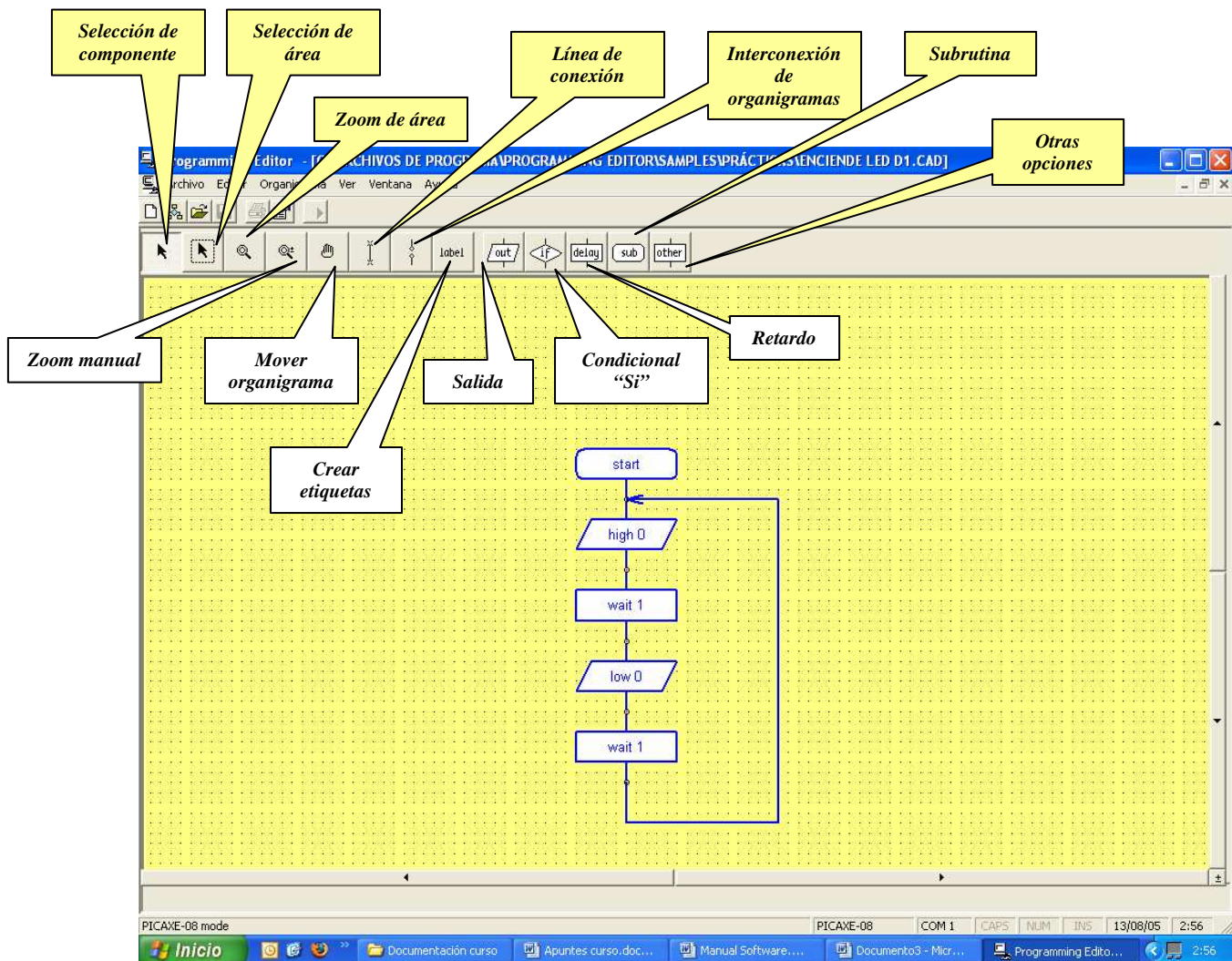
Tutorial 2. Programando mediante Organigramas (Flujogramas)

Los organigramas (también llamados flujogramas) son una herramienta muy útil que permiten representar gráficamente (dibujar) a los programas para hacerlos más fáciles de entender. El software Editor de Programación incluye un editor de organigramas que permite dibujar organigramas en la pantalla del ordenador. Estos organigramas se pueden convertir luego en código BASIC para descargarlos en el PICAXE. Los organigramas también pueden imprimirse y exportarse como figuras para incluirlos dentro de diagramas en la descripción de proyectos.

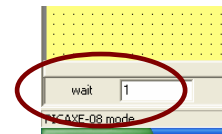
Instrucciones detalladas para dibujar/descargar un organigrama:

1. Conecte el cable PICAXE a uno de los puertos serie del ordenador. Recuerde tomar nota del puerto serie al cual conecta el cable (normalmente COM1 ó COM2).
2. Inicie el software "**Editor de Programación**".
3. En el menú desplegable seleccione **Ver** → **Opciones** para acceder a la pantalla de opciones (esta puede que aparezca automáticamente).
4. Haga clic en la lengüeta "**Modo**" y seleccione **PICAXE-08**.
5. Haga clic en la lengüeta "**Puerto Serie**" y seleccione el puerto serie al cual ha conectado el cable PICAXE. Haga clic en "**OK**".
6. Cree un nuevo organigrama haciendo clic en el menú **Archivo** → **Nuevo Organigrama**.
7. Dibuje el organigrama arrastrando los bloques requeridos a la pantalla y luego utilizando el ratón para dibujar flechas para conectar los bloques.
8. Cuando termine de dibujar el organigrama, puede convertirlo en un programa BASIC seleccionando el menú **Organigrama** → **Convertir Organigrama a BASIC**. Luego el programa BASIC puede descargarse en el PICAXE seleccionando en el menú **PICAXE** → **Ejecutar**.
9. Para imprimir o salvar el organigrama, utilice las opciones en el menú de Archivo. Para exportar el organigrama como figura, utilice el menú **Archivo** → **Exportar**. Para exportar la imagen a un documento de Word seleccione el archivo tipo **EMF**. Para exportar el organigrama a una página web use el archivo tipo **GIF**.

Este será el aspecto que tendrá el programa en BASIC editado en la práctica anterior, pero esta vez utilizando la función de organigrama, conozcamos su interface para poder realizar el organigrama:



- **Selección de componente:** utilizar este comando para seleccionar y mover bloques. Cuando se selecciona un solo bloque, su código BASIC puede editarse en la barra editora en la parte inferior de la ventana.



- **Selección de área:** utilizar este comando para seleccionar un conjunto de bloques seleccionados mediante la formación de un área (pinchar y arrastrar con botón izquierdo).
- **Zoom de área:** amplía una zona, previamente seleccionada, mediante la técnica de pinchar y arrastrar.
- **Zoom manual:** amplía una zona de forma manual. Para acercar hacer clic y mover el ratón hacia arriba. Para alejar hacer clic y mover el ratón hacia abajo.
- **Mover:** utilizar este comando para mover el organigrama completo alrededor de la pantalla.

- **Línea de conexión:** utilizar este comando para dibujar líneas entre los bloques. Se pueden hacer quiebres en las líneas haciendo clic una vez. Cuando la línea está cerca de un bloque, esta se pegará al punto de conexión del mismo.
- **Interconexión de organigramas:** este comando se utiliza cuando el organigrama es muy complejo y éste se puede simplificar en bloques más sencillos, utilizándolo posteriormente para la interconexión de los mismos de forma correspondiente.
- **Etiqueta:** utilizar este comando para añadirle etiquetas o títulos a los elementos del organigrama.
- **Salida/Si/Retardo/Subrutina/Otras opciones:** hacer clic en estos botones para ir al submenú de estos comandos y seleccionar el comando correspondiente.

Dibujando organigramas:

Para dibujar un organigrama hacer clic en uno de los bloques de menús de comandos (Salida/Si/Retardo/Otras opciones) de la barra de herramientas para ir al submenú de comandos requerido. Seleccione el comando deseado y luego hacer clic en la pantalla, en el lugar donde desea situar el comando. No tratar de colocar el bloque exactamente en posición en primera instancia, ponerlo en la pantalla en las cercanías del área donde desea ubicarlo y luego usar el comando *Seleccionar* para mover el bloque a la posición correcta.

Una vez que el bloque esté en posición, hacer clic en él de manera que sea resaltado. El código BASIC del objeto aparecerá en la barra editora en la parte inferior de la pantalla. Editar el código si lo requiere.

Uniendo bloques:

Para unir bloques, se debe acercarlos uno al otro hasta que se junten. Otra opción es dibujar líneas entre los mismos usando el comando línea en la barra de herramientas. Notar que sólo es posible unir la parte inferior de un bloque únicamente con la parte superior de otro (no se pueden conectar líneas con líneas). Además, sólo se permite sacar una línea de la parte inferior de conexión de cada bloque.

Para hacer diagramas ordenados, se pueden agregar quiebres a las líneas haciendo clic en las mismas. Al mover una línea cerca de un punto de conexión, la misma se pegará a este; para terminar la línea haga clic una vez más y la misma quedará en posición.

Las líneas no pueden moverse. Si se trata de mover una línea la misma será borrada y tendrá que crear una nueva línea.

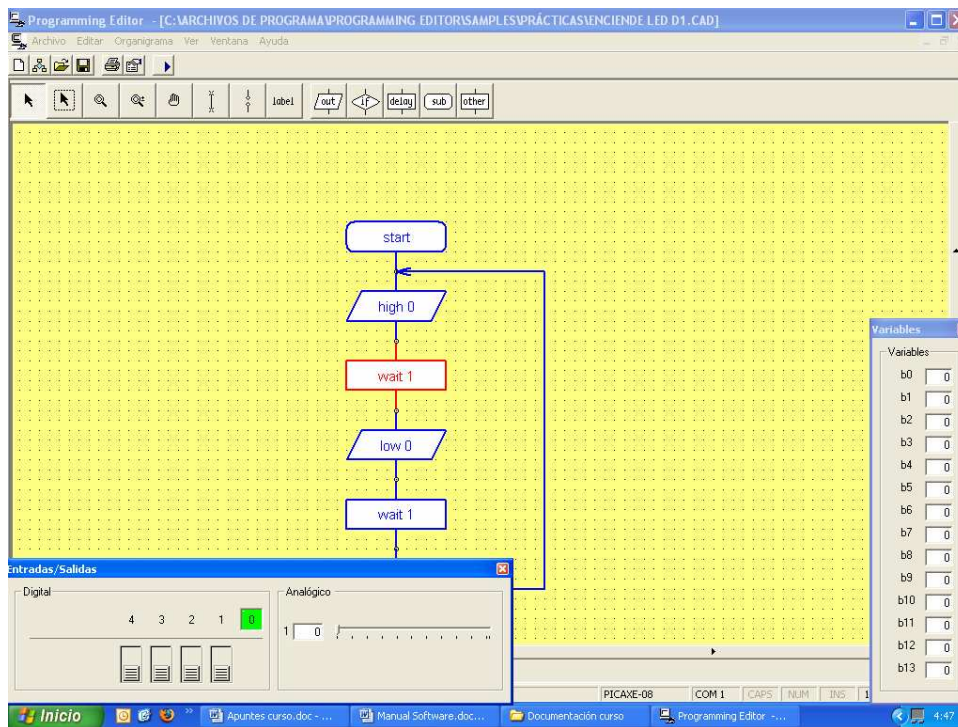
Simulación en pantalla:

Para simular el organigrama, haga clic en "*Simular*" en el menú Organigrama. El programa comenzará a ejecutarse en pantalla.

A medida que el programa se ejecuta, los bloques cuyos comandos están siendo ejecutados se irán resaltando en rojo. Las ventanas de "*Entradas/Salidas*" y "*Variables*" también aparecerán mientras se ejecuta la simulación. Para cambiar los valores de las entradas haga clic en el respectivo interruptor en pantalla (mostrado debajo del LED) ó utilice la barra deslizadora de entradas analógicas.

El tiempo de retardo entre un objeto y otro puede ser ajustado en las Opciones del Organigrama (menú **Ver** → **Opciones** → **Organigrama**).

Notar que algunos comandos representan acciones que no pueden ser simuladas en pantalla. En estos casos el comando es simplemente ignorado al ejecutar el organigrama.



Carga de los organigramas al PICAXE:

Los organigramas no se descargan directamente al microcontrolador. Primero el organigrama es convertido en un programa BASIC, el cual luego es cargado al PIC.

Para convertir un organigrama seleccionar "**Convertir**" en el menú Organigrama; el programa BASIC del organigrama será creado.

Aquellos bloques que no estén conectados a los bloques "**inicio**" ó "**sub**" en el organigrama, serán ignorados al momento de hacer la conversión. La conversión se detendrá si se encuentra un bloque no conectado; por lo tanto, utilizar siempre un bloque "**detener**" para terminar el diagrama antes de iniciar una simulación o de convertir el diagrama.

Notar que es posible convertir y descargar rápidamente un organigrama presionando dos veces la tecla **F5**.

Utilizando símbolos:

Entradas, Salidas y Variables pueden renombrarse utilizando la "*Tabla de Símbolos*" del menú Organigrama. Cuando un símbolo es renombrado el nuevo nombre aparecerá en los menús desplegables en la barra editora. No deben utilizarse nombres de comandos (por ejemplo switch o sound) como símbolos ya que esto puede generar errores en el programa BASIC convertido.

Guardar e imprimir organigramas:

Los organigramas pueden guardarse, imprimirse y exportarse como figuras (para insertarlos en documentos de procesadores de textos) utilizando el menú *Archivo*. Los organigramas pueden también copiarse en el portapapeles de Windows (para pegarlos luego a otras aplicaciones) utilizando el menú *Editar*.

Tutorial 3. Prácticas de programación con Cyberpet

Práctica 1:

Encendido y apagado de forma intermitente de un led conectado al pin 7 del PICAXE-08 (Salida: Pin0)

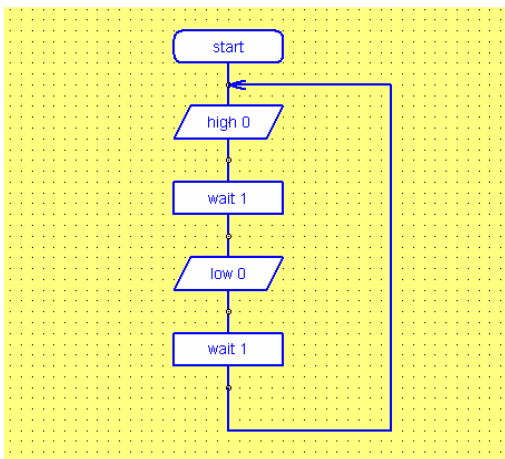
Código:

```
*****  
***** PRUEBA ENCENDIDO LED D1 (pin 7, out 0) *****  
*****
```

inicio:

```
high 0          'enciende LED en pin 7  
wait 1         'retardo 1 segundo  
low 0          'apaga LED en pin 7  
wait 1  
goto inicio    'salto a inicio
```

Flujograma:



Práctica 2:

Encendido y apagado de forma intermitente de un led conectado al pin 3 del PICAXE-08 (Salida: Pin4)

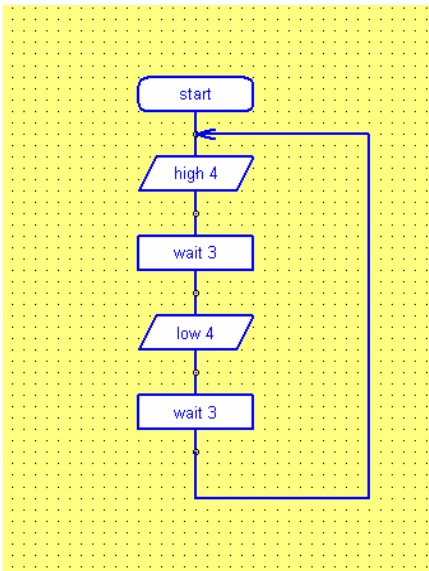
Código:

```
*****  
***** PRUEBA ENCENDIDO LED D2 (pin 3, out 4) *****  
*****
```

inicio:

high 4	'enciende LED en pin 3
wait 3	'retardo 1 segundo
low 4	'apaga LED en pin 3
wait 3	
goto inicio	'salto a inicio

Flujograma:



Práctica 3:

Encendido y apagado alternativamente de dos leds conectados al pin 3 y pin 7 del PICAXE-08 (Salida: Pin4 y Pin0).

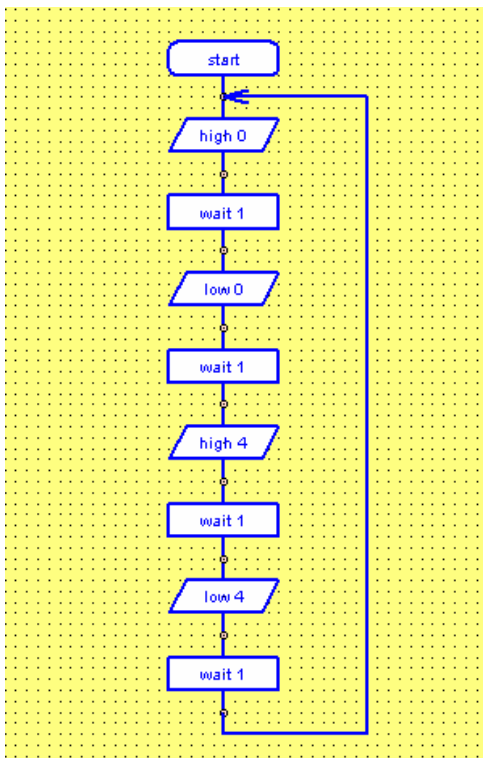
Código:

```
*****  
***** Leds Intermitentes alternativamente (pin 7, out 0)y (pin 3, Out 4) *****  
*****
```

inicio:

```
high 0          'enciende LED en pin 7  
wait 1          'retardo 1 segundo  
low 0           'apaga LED en pin 7  
wait 1  
high 4          'enciende LED en pin 3  
wait 1  
low 4           'apaga led en pin 3  
wait 1  
  
goto inicio     'salto a inicio
```

Flujograma:



Práctica 4:

Generación de 3 tonos diferentes a través del altavoz piezoeléctrico conectado al pin 5 del PICAXE-08 (Salida: Pin2).

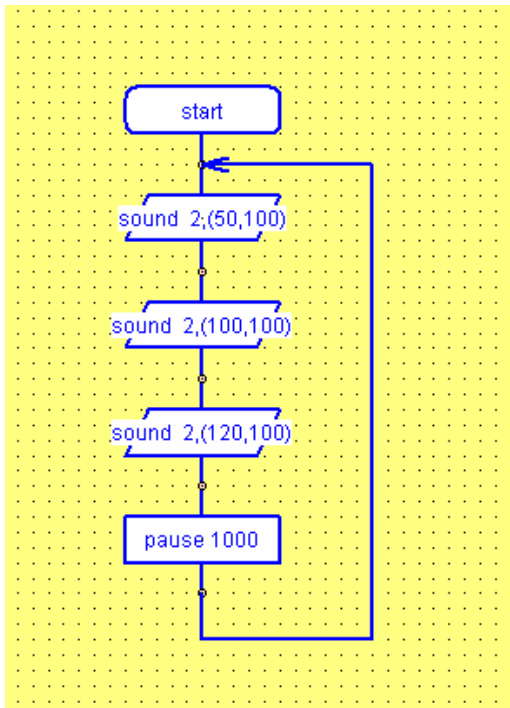
Código:

```
*****  
***** PRUEBA ALTAVOZ *****  
*****
```

inicio:

```
sound 2, (50, 100)      'sonido en altavoz piezo del pin 5, out 2 tono ((0..127), duración ms  
sound 2, (100, 100)    'otro tono  
sound 2, (120, 100)    'otro tono  
pause 1000             'pausa de 1000ms = 1 seg  
goto inicio
```

Flujograma:



Práctica 5:

Si se activa el interruptor conectado al pin 4 (Entrada: Pin3) enciende el led conectado al pin 7 del PICAXE-08 (Salida: Pin0).

Código:

```
*****  
***** PRUEBA INTERRUPTOR *****  
*****
```

inicio:

```
if input3 is on then led  
goto inicio
```

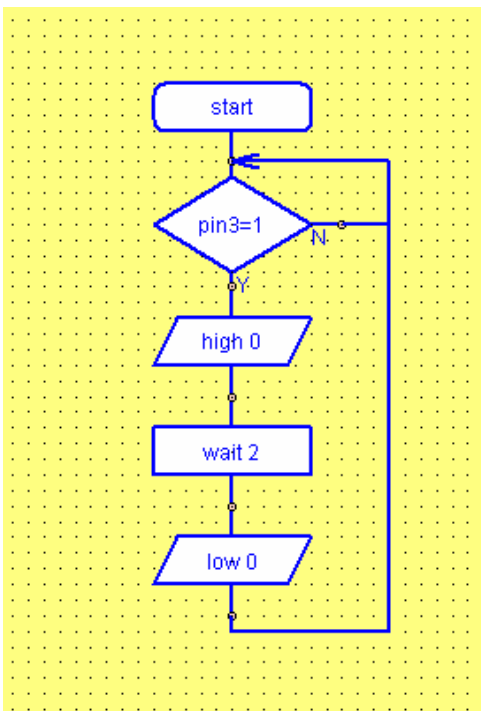
```
'si la entrada 3 es "1" salta a subrutina led sino a inicio  
'salto a inicio
```

led:

```
high 0  
wait 2  
low 0  
goto inicio
```

```
'enciende LED pin 7  
'encendido durante un retardo de 2 segundos  
'apagado LED pin 7
```

Flujograma:



Práctica 6:

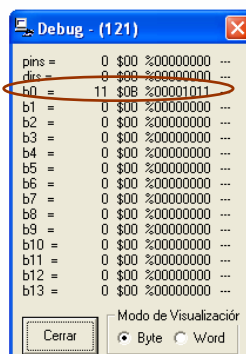
Tenemos conectada una LDR en el pin 6 (Entrada: Pin1) del PICAXE-08. Visualizaremos el valor que nos da el convertidor Analógico/Digital (DAC) a través del Debug de la aplicación (presenta el valor de las variables b0..b13, tanto en valor decimal como digital).

Código:

```
*****  
***** Chequeo de umbral de luz *****  
*****  
'LDR wn pin 6, input1  
  
**** Bucle principal ****  
  
inicio:  
    readadc 1,b0      'lee la señal analógica de LDR y carga en variable b0  
    debug b0         'transmite el valor b0 a la pantalla del PC  
    pause 100        'pausa  
    goto inicio      'saltar a inicio
```

Resultado:

Al cargar el programa al PICAXE-08 se abrirá el depurador (Debug) mostrando el valor de las variables b0..b13. Deberemos fijarnos en el valor que toma la variable b0 que es donde se almacena el valor generado por el convertidor DAC, anotaremos el valor justo en el momento que la intensidad luminosa que nos interesa detectar se presente. Cada muestreo será indicado visualmente a través del Led conectado al pin 7 del PICAXE-08 (Salida: Pin0).



Práctica 7:

Tenemos conectada una LDR en el pin 6 (Entrada: Pin1) del PICAXE-08. Si la intensidad de luz es alta (>120) se iluminará el Led 1, por el contrario, si la luz que incide sobre la LDR es baja (<70) se iluminará el Led 2.

Código:

```
*****
***** Detector luz máx y mín *****
*****
'LDR wn pin 6, input1
'LED D1 en pin7, out0
'LED D2 en pin3, out4

**** Bucle principal ****

inicio:
  readadc 1,b0
  if b0 > 120 then maximo
  if b0 < 70 then minimo
  low 0
  low 4
  goto inicio

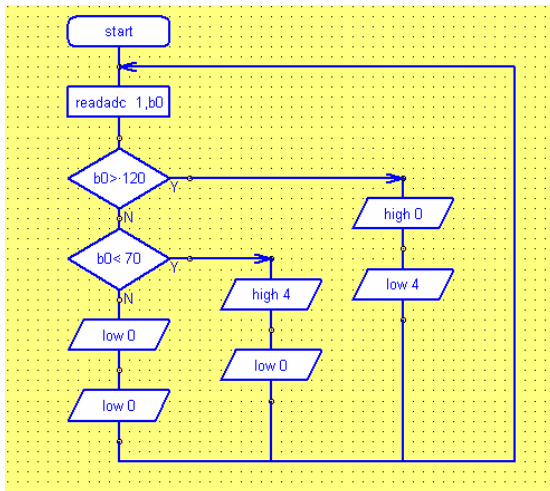
**** Acción límite máx ****

maximo:
  high 0
  low 4
  goto inicio

**** Acción límite mín ****

minimo:
  high 4
  low 0
  goto inicio
```

Flujograma:



Práctica 8:

Tenemos conectada una LDR en el pin 6 (Entrada: Pin1) del PICAXE-08. Debemos generar un haz de luz unidireccional que deberá incidir sobre la superficie sensible de la LDR. Si la luz incide no se dispara la alarma (Led y piezo activados), por el contrario si se corta el haz de luz (atravesado por un objeto móvil) esta se disparará activando una señal acústica y luminosa (altavoz piezoeléctrico y Led).

Código:

```
*****
***** BARRERA LUMINOSA. Alarma *****
*****
'LDR wn pin 6, input1
'LED D1 en pin7, out0
'ALTAVOZ piezoeléctrico en pin5, out2

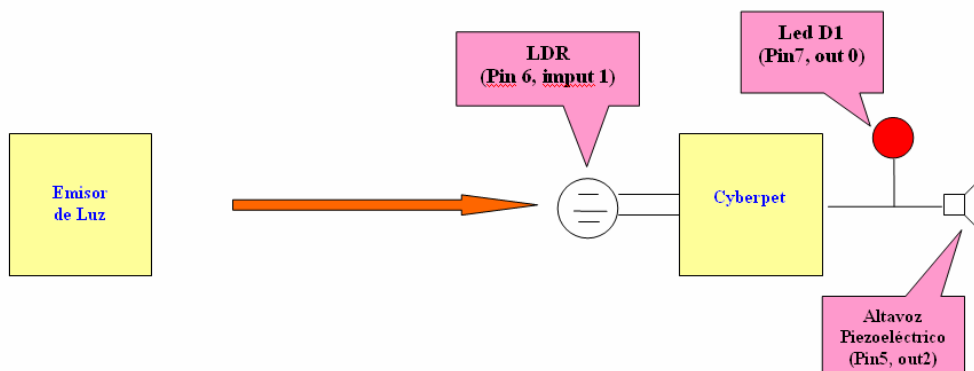
symbol light=b1

**** Bucle principal ****
'Este bucle detecta la presencia de intrusos y corta la barrera de luz

inicio:
    low 0
    readadc 1,light
    if light < 80 then alarma
    goto inicio

**** Activar alarma ****
'Esta rutina enciende el LED D1 y hace sonar el altavoz mientras exista corte de haz de luz

alarma:
    high 0
    sound 2,(50,300)
    readadc 1, light
    if light > 80 then inicio
    goto alarma
```



Práctica 9:

Control total de Cyberpet. Analizar el código y descubrir sus funciones a través de la interpretación del mismo.

Código:

```
*****
***** CIBERMASCOTA. Programa1 *****
*****
'LDR wn pin 6, input1
'PULSADOR en pin4, input3
'LED D1 en pin7, out0
'LED D2 en pin3, out4
'ALTAVOZ piezoeléctrico en pin5, out2

symbol light=b1

**** Bucle principal ****
'Este bucle hace parpadear los leds y verifica el estado del sensor de luz y del pulsador

inicio:

'Encender ambos led y leer valor de luz (light=b1)

    high 4
    high 0
    readadc 1,light

'Si el valor de luz está por debajo de 80 ir a dormir

    if light < 80 then bed

'Si el pulsador es presionado emitir sonido

    if pin3 = 1 then purr

'Hacer una pausa

    pause 500

'Apagar ambos LEDs y verificar estado del sensor nuevamente

    low 4
    low 0
    readadc 1,light
    goto inicio

**** Emitir sonido ****

purr:

    sound 2, (120,50,80,50,120,50)
    pause 200
    goto inicio

**** Rutina de ir a dormir cuando oscurece ****
'Si está oscuro apagar ambos leds y esperar hasta que haya luz nuevamente

bed:

    low 0
    low 4
    readadc 1, light
    if light > 80 then inicio
    goto bed
```

ANEXO1: Comandos BASIC PICAXE

COMENTARIOS

Usar comentarios. Aunque sea perfectamente obvio para ti, lo que estás escribiendo, alguien más puede leerlo (o tu mismo más tarde) y puede no tener idea de lo que habías hecho. Aunque los comentarios usan espacio en el archivo fuente, no lo hacen en el PICAXE.

Haz que los comentarios te digan algo útil acerca de lo que el programa está haciendo. Un comentario como “poner a 1 el pin 0” simplemente explica la sintaxis del lenguaje, pero no te dice nada acerca de la necesidad de hacerlo. “Encender el LED de alarma” puede ser más útil.

Un bloque de comentarios en el comienzo del programa y antes de cada sección de código puede describir que sucederá con más detalle que un simple espacio después de cada línea. No incluyas un bloque de comentarios en lugar de comentarios individuales de línea. Usar ámbos.

En el comienzo del programa que hará el programa, autor y fecha. Puede ser útil para listar informaciones de revisión y fechas. Especificar las conexiones de cada pin puede ser útil para recordar para que equipo en especial se programó.

Los comentarios comienzan con un apóstrofe (‘) o punto y coma (;) y continúa hasta el final de la línea. La palabra clave REM también puede ser usada para comentarios.

Ejemplo:

```
high 0          ‘poner pin0 a nivel alto
high0          ;poner pin0 a nivel alto
Rem poner pin0 a nivel alto
```

SIMBOLOS

Los símbolos nemotécnicos son palabras clave que identifican constantes, variables y direcciones de programa.

La asignación del símbolo se realiza poniendo el nombre del símbolo seguido del símbolo igual (=) y posteriormente la variable o constante.

Los símbolos pueden ser cualquier palabra que no sea la relativa a un comando.

Los símbolos pueden contener caracteres numéricos (e.j.: flash1, flash2, etc.) Pero el primer carácter no puede ser uno numérico (e.j.: 1flash)

El uso de símbolos no aumentan la longitud del programa.

Las direcciones de programa son asignadas siguiendo el símbolo con dos puntos (:).

Ejemplos:

```
symbol Led_Rojo = 7          ‘define Led-Rojo como una constante 7
symbol Contador = B0        ‘define Contador como una variable B0
let Contador = 200          ‘precarga de la variable Contador con el valor 200
```

```
inicio:                      ‘define una dirección de programa
high Led_Rojo                ‘pone a nivel alto “1” la salida 7
pause Contador               ‘espera de 0,2 segundos
low Led_Rojo                 ‘pone a nivel bajo “0” la salida 7
pause Counter               ‘espera de 0,2 segundos
goto inicio                  ‘salta hasta la dirección indicada en inicio
```

CONSTANTES

Las llamadas constantes pueden ser creadas de manera similar a las variables. Puede ser más conveniente usar un nombre de constante en lugar de un número constante. Si el número necesita ser cambiado, únicamente puede ser cambiado en un lugar del programa donde se define la constante. No pueden guardarse datos variables dentro de una constante.

Las constantes declaradas pueden ser de cuatro tipos: decimal, hexadecimal, binario y ASCII.

Los números decimales se escriben directamente sin ningún prefijo.

Los números hexadecimales se preceden del símbolo dólar (&).

Los números binarios se preceden del símbolo tanto por ciento (%).

Los valores ASCII se colocan entre comillas (“...”).

Ejemplos:

100	‘ número 100 en decimal
&64	‘ número 64 en hexadecimal
%01100110	‘ número binario: 01100110
“A”	‘ letra “A” en código ASCII (65)
“Hello”	‘ ”Hello” equivalente a “H”, “e”, “l”, “l”, “o”
B1 = B0 ^ \$AA	‘ operación XOR entre la variable B0 con AA en hex

VARIABLES

Tipos según sistema:

A continuación se indican las variables que se aplican a cada modelo de programación:

- PICAXE es usado al programar módulos PICAXE.
- El BASIC y Extended son usados al programar módulos Stamp.
- El ensamblador es el utilizado con código de ensamblador.

PICAXE

El sistema PICAXE da soporte a las siguientes variables:

Words: W0, W1, W2, W3, W4, W5, W6

Bytes: DIRS, PINS (solo PICAXE-08), INFRA, KEYVALUE
B0, B1, B2, B3, B4, B5, B6, B7, B8, B9, B10, B11, B12, B13

Bits: PIN0, PIN1, PIN2, PIN3, PIN4, PIN5, PIN6, PIN7 (conjuntamente igual PINS)
BIT0, BIT1, BIT2, BIT3, BIT4, BIT4, BIT5, BIT6, BIT7 (conjuntamente igual B0)
BIT8, BIT9, BIT10, BIT11, BIT12, BIT13, BIT14, BIT15 (conjuntamente igual B1)

In/Out añade los pseudo nombres:

INPUT0, INPUT1, etc. puede usarse en lugar de PIN0, PIN1, etc.
OUTPUT0, OUTPUT1, etc. puede usarse en lugar de 0, 1, 2, etc.

BASIC

El modo BASIC da soporte a las siguientes variables:

Words: *PORT*

W0, W1, W2, W3, W4, W5, W6

Bytes: *DIRS, PINS*

B0, B1, B2, B3, B4, B5, B6, B7, B8, B9, B10, B11, B12, B13

Bits: *DIR0, DIR1, DIR2, DIR3, DIR4, DIR5, DIR6, DIR7* (conjuntamente igual *DIRS*)

BIT0, BIT1, BIT2, BIT3, BIT4, BIT4, BIT5, BIT6, BIT7 (conjuntamente igual *B0*)

BIT8, BIT9, BIT10, BIT11, BIT12, BIT13, BIT14, BIT15 (conjuntamente igual *B1*)

Nota: B12 y B13 (W6) es usado dentro de las órdenes GOSUB como una pila. Por consiguiente no debería utilizarse como un registro de propósito general.

EXTENDED

El modo extendido da soporte a todas las variables de BASIC. Además:

INPUT0, INPUT1, etc . puede usarse en lugar de pin0, pin1 etcétera.

OUTPUT0, OUTPUT1, etc . puede usarse en lugar de 0, 1, 2 etcétera.

Nota: B12 y B13 (W6) es usado dentro de las órdenes GOSUB como una pila. Por consiguiente no debería utilizarse como un registro de propósito general.

ENSAMBLADOR

El modo ensamblador soporta los mismos modelos de variables que el modo extendido.

RESUMEN DE COMANDOS (A-0)

Sistemas:

La siguiente tabla muestra los comandos que se aplican a los siguientes sistemas:

- El BASIC y Extended se usan al programar módulos Stamp.
- El PIC16F84A se usan al hacer un download directamente en un PIC16F84A
PIC16F627
- PICAXE se usa al programar módulos PICAXE

Comando	BASIC	Extendida	PIC16F84A	PICAXE08	PICAXE18	PICAXE28
Backward		X	X	X	X	X
Branch	X	X	X	X	X	X
Button	X	X		X	X	X
Count						X (X)
Debug	X			X	X	X
Eeprom	X	X	X	X	X	X
End	X	X	X	X	X	X
For...next	X	X	X	X	X	X
Forward		X	X	X	X	X
Gosub	X	X	X	X	X	X
Goto	X	X	X	X	X	X
Halt		X	X	X	X	X
High	X	X	X	X	X	X
i2slave					X (X)	X (X)
If...then	X	X	X	X	X	X
If infra.. then					x(A)	X
If sensor.. then		X				
Infrain					X(A)	X
Input	X	X	X	X		
Keyin					X (A)	X (X)
Lcd		X	X			
Lookdown	X	X	X	X	X	X
Lookup	X	X	X	X	X	X
Low	X	X	X	X	X	X
Nap	X	X	X	X	X	X
Output	X	X	X	X		

RESUMEN DE COMANDOS (P-Z)

Sistemas:

La siguiente tabla muestra los comandos que se aplican a los siguientes sistemas:

- El BASIC y Extended se usan al programar módulos Stamp.
- El PIC16F84A se usan al hacer un download directamente en un PIC16F84A
PIC16F627
- PICAXE se usa al programar módulos PICAXE

Comando	BASIC	Extendida	PIC16F84A	PICAXE08	PICAXE18	PICAXE28
Pause	X	X	X	X	X	X
Peek					X	X
Poke					X	X
Pot	X	X		(use readadc)	(use readadc)	(use readadc)
Pulsin	X	X	X	X	X	X
Pulsout	X	X	X	X	X	X
Pwm	X	X	X	X		
Pwmout					X (X)	X (X)
Random	X	X	X	X	X	X
Readadc	(use pot)	(use pot)		X	X	X
Readadc10					X (X)	X (X)
Readi2c					X (X)	X (X)
Read	X	X	X	X	X	X
Readmem						X
Readowclk					X (A)	
Resetowclk					X (A)	
Readowsn					X (A)	X (X)
Readtemp					X (A)	X (A)
Readtemp12					X (X)	X (X)
Return	X	X	X	X	X	X
Reverse	X	X	X	X		
Serin	X	X		X	X	X
Serout	X	X	X	X	X	X
Sertxd					X (X)	X (X)
Servo					X (A)	X
Setint					X (A)	X (A)
Setfreq					X (X)	
Sleep	X	X	X	X	X	X
Sound	X	X	X	X	X	X
Switch on		X	X	X	X	X
Switch off		X	X	X	X	X
Symbol	X	X	X	X	X	X
Toggle	X	X	X	X	X	X
Wait		X	X	X	X	X
Write	X	X	X	X	X	X
Writemem						X
Writei2c					X (X)	X (X)

backward

BACKWARD motor

Hace que el motor conectado a la salida cambie de giro

- Motor: puede tomar el valor motor A o motor B

Nota: salidas 4, 5 (A) y salidas 6, 7 (B)

Ejemplo:

inicio:

forward A	'motor A en marcha
wait 5	'espera 5 segundos
backward A	'motor A cambio de giro
wait 5	'espera 5 segundos
halt A	'motor A parado
wait 5	'espera 5 segundos
goto inicio	'salto a inicio

branch

BRANCH offset,(dirección0, dirección1...direcciónN)

Causa que el programa salte a una posición diferente, basada en una variable indexada. Es similar al ON...GOTO de otros BASIC. La etiqueta que especifica la dirección, debe estar en la misma página de código que la instrucción BRANCH.

- offset: es una variable/constante que especifica que dirección usar (0, 1...n)
- direcciones: son etiquetas que especifican donde ir

Ejemplo:

rest:

let infra = 0	'resetear la variable infra
---------------	-----------------------------

inicio:

if infra > 0 then test	
goto inicio	'salta a test si la variable infra es > 0 si no vuelta a inicio

test:

branch infra, (btn0,btn1,btn2,btn3)	
goto rest	'ramifica a rutinas indicadas (btn0..btn3) si no 'resetea saltando a rest

btn0: etc.

btn1: etc.

btn2: etc.

btn3: etc.

button

BUTTON Pin, Down, Delay, Rate, Bvar, Action, Etiqueta

Lee Pin y opcionalmente ejecuta anti-rebote y autoprotección. Pin automáticamente se toma como entrada. Pin debe ser una constante entre 0-15 o una variable que contenga un número entre 0-15 (p.ej. B0) ó un número de pin (p.ej. PORTA,0)

- Down: estado del pin cuando se oprime el pulsador (0..1)
- Delay: contador de ciclos antes de que comience la auto-repetición (0..255). Si es 0, no se efectua anti-rebote ni auto-repetición. Si es 255 se eliminan rebotes, pero no auto-repitición
- Rate: valor de auto-repetición (0..255)
- Bvar: variable con tamaño de byte usada internamente para conteo de demoras y repeticiones. Debe ser inicializada a 0 antes de ser usada y no ser usada en cualquier lugar del programa
- Action: estado del pulsador al ser actuado
- Etiqueta: la ejecución comienza en esta etiqueta si es cierto Action

Ejemplo:

inicio:

```
button 0,0,200,100,b2,0,cont    'salta a cont a menos que pin0 = 0
toggle 1                        'cambia el estado del pin1
```

cont:

count

COUNT Pin, Period, Var

Cuenta el número de pulsos en un Pin, durante un período Period, y guarda el resultado en Var. Pin

- Pin: debe ser una constante (0..7) ó una variable que contenga un número de 0 a 15 (p.ej. B0) ó un numero de pin
- Period: valor numérico (0..65535) en ms
- Variable: variable tipo word, recibe el valor del conteo (0..65535)

Chequea el estado de Pin mediante un bucle y cuenta las transiciones de bajo a alto. Con un oscilador de 4 MHz chequea el estado del pin cada 20 μ s. Con un oscilador de 20MHz chequea el estado cada 4 μ s. De esto, se deduce que la mayor frecuencia de pulsos que puede ser contada es de 25 KHz con un oscilador de 4 MHz y de 125 KHz con un oscilador de 20 MHz si la frecuencia tiene un ciclo útil del 50% (los tiempos altos son iguales a los bajos).

Ejemplo:

inicio:

```
count 1, 500, w1                'cuenta los pulsos del pin1 en 5 segundos
debug w1                        'viasualiza los valores
goto inicio                     'salta a inicio (bucle)
```

debug

DEBUG Var

Visualiza la información contenida en la variable Var en la ventana del depurador durante la ejecución.

- Var: variable con un valor previamente cargado

Ejemplo:

inicio:

debug b1	‘visualiza el valor de b1
let b1 = b1+1	‘incrementa el valor de b1 en una unidad
pause 500	‘espera de 0,5 segundos
salto inicio	‘salta a inicio (bucle)

eeprom

EEPROM {location},(data,data,...)

Guarda constantes en un chip EEPROM. Si se omite el valor opcional Location, la primera declaración se guarda en la dirección 0 de la EEPROM y las subsiguientes en las siguientes direcciones del mismo. Si se indica un valor Location, éste indica la dirección de comienzo para guardar los datos..

- Location: es una constante numérica (0..255) que especifica donde empezar a almacenar los datos en la EEPROM. Si no existe una posición indicada el almacenamiento continúa a partir del último dato introducido. Si inicialmente no se especificó ninguna posición el almacenamiento comienza en la 0.
- Data: son constantes (0..255) que se almacenan en la EEPROM.

EEPROM solo trabaja con microcontroladores con EEPROM incorporado como el PIC16F84 y PIC16C84 , PICAXE-18A, 28, 28A, 28X. Dado que la EEPROM es una memoria no volátil, los datos permanecerán intactos aún sin alimentación.

Los datos se guardan en la EEPROM solo una vez, cuando el microcontrolador es programado, no cada vez que se ejecuta el programa. Se puede usar WRITE para colocar valores en la EEPROM en el momento de la ejecución.

Ejemplo:

EEPROM 5, (10,20,30)	‘guarda 10, 20, 30 a partir de la dirección 5
EEPROM 0, (“Hello Word”)	‘guarda el valor ASCII

end

END

Detiene la ejecución del proceso y entra en modo de bajo consume. Todos los pins de I/O permanecen en el estado en que se encuentran. END trabaja ejecutando una instrucción SLEEP continua dentro de un bucle.

Un END, STOP ó GOTO deben ser colocados al final de un programa para evitar pasar del límite de la misma u comience nuevamente.

Ejemplo:

inicio:

let b2 = 15	‘set b2 con valor 15
pause 2000	‘espera de 2 segundos
gosub flash	‘salta al procedimiento flash
let b2 = 5	‘set b2 con valor 5
pause 2000	‘espera de 2 segundos
gosub flash	‘salta al procedimiento flash
end	

flash:

for b0 = 1 to b2	‘define el bucle con el valor b2
high 1	‘salida 1 a nivel alto
pause 500	‘espera de 0,5 segundos
low 1	‘salida 1 a nivel bajo
pause 500	‘espera de 0,5 segundos
next b0	‘fin de bucle
return	‘retorno a linea siguiente de la llamada

for ... next

```
FOR contador = start TO end {STEP {-} inc}
{body}
```

El bucle FOR ... NEXT permite a los programas ejecutar un número de declaraciones {body} un número de veces, usando una variable como contador. Debido a su complejidad y versatilidad, es mejor describirla paso a paso.

El valor de **start** se asigna a la variable índice: **count**, que puede ser una variable de cualquier tipo.

Se ejecuta las instrucciones de **body**. Body es opcional y puede ser omitido (quizás por un bucle de demora).

El valor de **inc** es sumado a (ó restado si se especifica “-“) **count**. Si no se define un valor **steep**, se incrementa **count** en una unidad.

Si **count** no pasó **end** ó desbordó el tipo de variable, la ejecución vuelve al paso 2.

Si el bucle necesita contar más de 255 ($count > 255$), se debe usar una variable de tamaño Word.

Ejemplo:

```
bucle:
  for b0 = 1 to 20      'define el bucle con repetición de 1 a 20
    high 1             'salida 1 a nivel alto
    pause 500          'espera de 0,5 segundos
    low 1              'salida 1 a nivel bajo
    pause 500          'espera de 0,5 segundos
  next b0              'fin de bucle

  pause 2000           'espera de 2 segundos
  goto bucle           'salta a bucle para comienzo de nuevo
```

forward

FORWARD motor

Arranca motor hacia delante

- Motor: puede tomar el valor motor A o motor B

Nota: salidas 4, 5 (A) y salidas 6, 7 (B)

Ejemplo:

```
inicio:
  forward A            'motor A en marcha
  wait 5              'espera 5 segundos
  backward A          'motor A cambio de giro
  wait 5              'espera 5 segundos
  halt A              'motor A parado
  wait 5              'espera 5 segundos
  goto inicio         'salto a inicio
```

gosub

GOSUB etiqueta

*Salta a la subrutina indicada en **etiqueta**, guardando su dirección de regreso en la pila (snack). A diferencia del GOTO, cuando se llega a un RETURN, la ejecución sigue con la declaración siguiente al último GOSUB ejecutado.*

Se puede usar un número ilimitado de subrutinas en un programa y pueden estar anidadas. En otras palabras, las subrutinas pueden llamar a otra subrutina. Cada anidamiento no debe ser mayor de cuatro niveles.

Ejemplo:

inicio:

let b2 = 15	‘set b2 con valor 15
pause 2000	‘espera de 2 segundos
gosub flash	‘salta al procedimiento flash
let b2 = 5	‘set b2 con valor 5
pause 2000	‘espera de 2 segundos
gosub flash	‘salta al procedimiento flash
end	

flash:

for b0 = 1 to b2	‘define el bucle con el valor b2
high 1	‘salida 1 a nivel alto
pause 500	‘espera de 0,5 segundos
low 1	‘salida 1 a nivel bajo
pause 500	‘espera de 0,5 segundos
next b0	‘fin de bucle
return	‘retorno a linea siguiente de la llamada

goto

GOTO etiqueta

La ejecución del programa continua en la declaración de la etiqueta.

Ejemplo:

inicio:

high 1	‘salida 1 a nivel alto
pause 5000	‘espera de 5 segundos
low 1	‘salida 1 a nivel bajo
pause 5000	‘espera de 5 segundos
goto inicio	‘salta a inicio

halt

HALT motor

Para el motor indicado

- Motor: puede tomar el valor motor A o motor B

Nota: salidas 4, 5 (A) y salidas 6, 7 (B)

Ejemplo:

inicio:

forward A	'motor A en marcha
wait 5	'espera 5 segundos
backward A	'motor A cambio de giro
wait 5	'espera 5 segundos
halt A	'motor A parado
wait 5	'espera 5 segundos
goto inicio	'salto a inicio

high

HIGH pin

Pone a nivel alto el pin especificado y lo convierte automáticamente en salida. Pin puede ser una constante, 0 – 7, ó una variable que contenga un número de 0 – 7 (p. ej. B0) ó un número de pin (p. ej. PORTA.0)

Ejemplo:

bucle:

high 1	'pone salida 1 a nivel alto
pause 5000	'espera de 5 segundos
low 1	'pone la salida 1 a nivel bajo
pause 5000	'espera de 5 segundos
goto bucle	'salta a comienzo de bucle repetitivo

i2cslave

I2CSLAVE esclavo, velocidad, dirección

El comando i2cslave se utiliza para configurar los pines del PICAXE y definir el tipo de i2c. Si se utiliza únicamente un dispositivo i2c, entonces generalmente solo se necesita una orden i2cslave dentro del programa.

- Esclavo: es la dirección del esclavo i2c.
- Velocidad: marca la velocidad de transmisión. I2cfast: 400KHz o I2cslow: 100KHz.
- Dirección: es el i2cbyte o i2sword que indica la dirección a partir de la cual se guardará la información.

Después de que se utilice el comando i2cslave, se pueden utilizar los comandos **readi2c** y **writei2c** para acceder al dispositivo i2c. Se pueden utilizar para leer y grabar datos de una EEPROM serie usando un interface i2c de 2 hilos, como la 24LC01B ó similar. Esto permite guardar datos en una memoria externa no volátil, para que sean mantenidos aún sin conexión de alimentación. También se utilizan para poder comunicarse con otros dispositivos con interface i2c, como sensores de temperatura y convertidores A/D.

Dirección esclavo

La dirección del esclavo cambia para dispositivos diferentes (ver tabla inferior) del i2c.

Para las EEPROM del tipo 24LCxx la dirección es %1010xxxx.

Velocidad

La velocidad del bus i2c se selecciona a través de los comandos i2cfast para velocidad de 400KHz y i2cslow para 100 KHz.

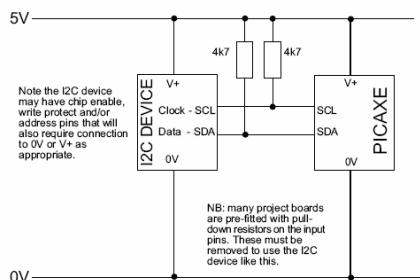
Dirección control

Los 7 bits superiores del byte de control contienen el código de control junto con la selección del chip e información adicional de dirección, dependiendo de cada dispositivo. El bit inferior es una bandera interna que indica si es un comando de lectura ó escritura y no se debe usar.

Por ejemplo, cuando comunicamos con un 24LC01B, el código de control es %1010 y no se usa la selección de chip, por lo que el byte de control será %10100000 ó \$A0. Algunos formatos de control se pueden ver en la siguiente tabla:

Dispositivo	Capacidad	Dirección esclavo	Velocidad	Tamaño dirección
24LC01B	EE 128	%1010xxxx	i2cfast	i2cbyte
24LC02B	EE 256	%1010xxxx	i2cfast	i2cbyte
24LC04B	EE 512	%1010xxbx	i2cfast	i2cbyte
24LC08B	EE 1k	%1010xbbx	i2cfast	i2cbyte
24LC16B	EE 2k	%1010bbbx	i2cfast	i2cbyte
24LC32B	EE 4k	%1010dddx	i2cfast	i2cword
24LC64	EE 8k	%1010dddx	i2cfast	i2cword
DS1307	RTC	%1101000x	i2cslow	i2cbyte
MAX6953	5x7 LED	%101dddx	i2cfast	i2cbyte
AD5245	Digital Pot	%010110dx	i2cfast	i2cbyte
SRF08	Sonar	%1110000x	i2cfast	i2cbyte
CMPS03	Compass	%1100000x	i2cfast	i2cbyte
SP03	Speech	%1100010x	i2cfast	i2cbyte

El tamaño de dirección enviado (byte o Word) viene determinado por el tamaño de la variable usada. Si se usa una variable con tamaño byte se envía una dirección de 8 bits. Si se envía una variable de tamaño Word, se envía una dirección de 16 bits. Asegurarse de utilizar una variable apropiada al dispositivo a comunicar.



if...then
if...and...then
if...or...then

IF variable ?? valor {AND/OR variable ?? valor ...} THEN etiqueta

Estructura de comparación condicional con salto a una dirección determinada (etiqueta) en caso de que se cumpla.

- *??: condicional =, <>, >=, <=, >, <*
- *Variable: valor de la comparación*
- *Valor: variable constante*
- *Etiqueta: dirección del salto en caso de cumplir el condicional*

Efectúa una o más comparaciones. Cada término variable puede relacionar un valor con una constante ú otra variable e incluye uno de los operadores listados anteriormente.

IF...THEN evalúa la comparación en términos de CIERTO o FALSO. Si lo considera cierto, se ejecuta la operación posterior al THEN. Si lo considera falso, no se ejecuta la operación posterior al THEN. Las comparaciones que dan 0 se consideran falso. Cualquier otro valor es cierto.

Ejemplo:

inicio:

if pin0 = 1 then flash	‘salta a flash si el valor del pin0 es un 1
goto inicio	‘sino salta a inicio

flash:

high 1	‘pone a valor alto la salida 1
pause 5000	‘espera de 5 segundos
low 1	‘pone a valor bajo la salida 1
goto inicio	‘salto a inicio

infrain

INFRAIN

Espera hasta que una señal de infrarrojos sea recibida.

Este comando se utiliza para esperar una señal infrarrojo nueva procedente del transmisor de infrarrojos (utiliza el protocolo de Sony). El programa se detiene hasta que reciba la señal infrarroja. El valor de la orden recibida se almacena en la variable predefinida “*infra*”.

Ejemplo:

Inicio:

infrain	‘espera una nueva señal infrarroja
if infra = 1 then swon1	‘si el valor de infra es 1 salta a swon1
if infra = 2 then swon2	‘si el valor de infra es 1 salta a swon2
if infra = 3 then swon3	‘si el valor de infra es 1 salta a swon3
if infra = 4 then swoff1	‘si el valor de infra es 1 salta a swoff1
if infra = 5 then swoff2	‘si el valor de infra es 1 salta a swoff2
if infra = 6 then swoff3	‘si el valor de infra es 1 salta a swoff3
goto inicio	

swon1:

high 1
goto inicio

swon2:

high 2
goto inicio

swon3:

high 3
goto inicio

swoff1:

low1
goto inicio

swoff2:

low2
goto inicio

swoff3:

low3
goto inicio

input

INPUT pin

Convierte el Pin especificado en una entrada. Pin debe ser una constante entre 0-7, o una variable que contenga un número 0-7 (p. ej.: B0) o el nombre de un pin (p. ej.: PORTA.0)

Ejemplo:

inicio:

input 1	‘configura el pin1 como entrada
reverse 1	‘configura el pin 1 como salida
reverse 1	‘configura el pin1 como entrada
output1	‘configura el pin1 como salida

keyin

KEYIN

Espera hasta que se presione una tecla del teclado.

Este comando se utiliza para realizar una espera hasta que se accione una tecla del teclado (conectado directamente al PICAXE, no el teclado utilizado mientras se programa). Todo procesamiento se detiene hasta que lel accionamiento se reciba. El valor de la tecla pulsada se almacena en una variable predefinida “*keyvalue*”.

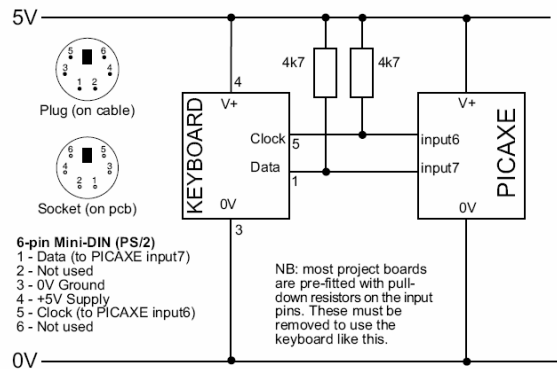
El valor almacenado en keyvalue estan en hexadecimal de forma que se debe tener en cuenta a la hora de programar anteponiendo el símbolo \$.

A continuación se muestra en detalle el valor que le corresponde a cada tecla:

KEY	CODE	KEY	CODE	KEY	CODE
A	1C	9	46	[54
B	32		OE	INSERT	E0.70
C	21	-	4E	HOME	E0.6C
D	23	=	55	PGUP	E0.7D
E	24	\	5D	DELETE	E0.71
F	2B	BKSP	66	END	E0.69
G	34	SPACE	29	PGDN	E0.7A
H	33	TAB	OD	U ARROW	E0.75
I	43	CAPS	58	LARROW	E0.6B
J	3B	LSHIFT	12	D ARROW	E0.72
K	42	LCTRL	14	R ARROW	E0.74
L	4B	LGUI	E0.1F	NUM	77
M	3A	LALT	11	KP/	E0.4A
N	31	RSHFT	59	KP*	7C
O	44	RCTRL	E0.14	KP-	7B
P	4D	RGUI	E0.27	KP +	79
Q	15	RALT	E0.11	KPEN	E0.5A
R	2D	APPS	E0.2F	KP.	71
S	1B	ENTER	5A	KP0	70
T	2C	ESC	76	KP1	69
U	3C	F1	05	KP2	72
V	2A	F2	06	KP3	7A
w	1D	F3	04	KP4	6B
X	22	F4	06	KP5	73
Y	35	F5	03	KP6	74

KEY	CODE	KEY	CODE	KEY	CODE
Z	1A	F6	0B	KP7	6C
0	45	F7	83	KP8	75
1	16	F8	0A	KP9	7D
2	1E	F9	01	J	5B
3	26	F10	09		4C
4	25	F11	78	'	52
5	2E	F12	07	•	41
6	36	PRNT SCR	??		49
7	3D	SCROLL	7E		4A
8	3E	PAUSE	??		

Forma de conectar el teclado al PICAXE:



Ejemplo:

inicio:

```

keyin
if keyvalue = $45 then swon1      'espera la activación de una tecla
if keyvalue = $16 then swon2      'pone salida 1 a valor alto
if keyvalue = $26 then swon3      'pone salida 2 a valor alto
if keyvalue = $25 then swoff1     'pone salida 3 a valor alto
if keyvalue = $2E then swoff2     'pone salida 1 a valor bajo
if keyvalue = $36 then swoff3     'pone salida 2 a valor bajo

```

swon1:

```

high 1
goto inicio

```

swon2:

```

high 1
goto inicio

```

swon3:

```
high 1  
goto inicio
```

```
swoff1:  
low 1  
goto inicio  
swoff2:  
low 2  
goto inicio  
swoff3:  
low 3  
goto inicio
```

keyled

KEYLED mascara

Activa los Leds del teclado.

- *Mascara: es una variable/constante que especifica que leds se activarán.*

Este comando se utiliza activar los leds, que monitorizan ciertas funciones, del teclado que se encuentre conectado directamente el PICAXE. El valor de *mascara* ajusta la operación de los leds de la siguiente forma:

- Bit 0: Scroll Lock (1=on, 0=off)
- Bit 1: Num Lock (1=on, 0=off)
- Bit 2: Caps Lock (1=on, 0=off)
- Bit 3-6: No usado
- Bit 7: Deshabilita Flash (1=on, 0=off)

El bit 7 si está habilitado los tres leds se encenderán intermitentemente cuando el comando Kevin detecte un pulsación de una tecla. Esta opción permite indicarnos que la pulsación de la tecla ha sido detectada por el PICAXE. Si se deshabilita no se indicará tal acción.

Ejemplo:

```
inicio:  
keyled %10000111      'todos los leds encendidos  
pause 500             'pausa de 0,5 seg  
keyled %10000000     'todos los leds apagados  
pause 500             'pausa de 0,5 seg  
goto inicio          'salta a inicio
```

lcd

LCD ({#}dato,{#}dato...)
LCD clear
LCD line1
LCD line2

Comando para controlar el módulo LCD con salida serie conectado al pin 7.

- *Clear: borra la pantalla LCD.*
- *Line1 :posiciona el cursor en la linea 1 para comenzar la escritura de dato.*
- *Line2: posiciona el cursor en la linea 2 para comenzar la escritura de dato.*
- *Dato: variable/constante (0-255) que se envían a la salida.*

Este seudo comando realmente analiza gramaticalmente comandos serout. Equivale em BASIC a serout 7, T2400, (datos).

Muestra ítems en un LCD con controlador Hitachi 44780 o equivalente. Estos LCD, usualmente, tienen un cabezal de 14 o 16 pines simples o duales en un extremo. Si el signo (#) está colocado antes de un ítem, la representación ASCII para cada dígito es enviada al LCD.

Un programa debe esperar, por lo menos, medio segundo antes de enviar el primer comando a un LCD. Puede tomar bastante tiempo a un LCD arrancar.

Los comandos son enviados al LCD, enviando un **\$FE** seguido por el comando. Algunos comandos útiles se muestran en la siguiente tabla:

Comando	Operación
\$FE, 1	Limpia visor
\$FE, 2	Vuelve a inicio (comienzo de la primera línea)
\$FE, \$0C	Cursor apagado
\$FE, \$0E	Subrayado del cursor activo
\$FE, \$0F	Parpadeo del cursor activo
\$FE, \$10	Mueve cursor una posición hacia la izquierda
\$FE, \$14	Mueve cursor una posición hacia la derecha
\$FE, \$C0	Mueve cursor al comienzo de la segunda línea

Ejemplo:

inicio:

LCD ("hola")	'escribe en el LCD la palabra "hola"
LCD line2	'posiciona cursor en la linea 2
LCD (" amigo")	'escribe en el LCD la palabra "amigo"
pause 2000	'espera 2 segundos
LCD clear	'borra contenido LCD
pause 500	'espera de 0,5 segundos
goto inicio	'salta a inicio

let

{LET} variable = {-}valor ?? valor ...

Asigna un valor a una variable. El valor puede ser una constante, otra variable o el resultado de una expresión. La palabra clave LET, por sí misma, es opcional.

?? puede ser:

+	:add
-	;subtract
*	;multiply (returns low word of result)
**	;multiply (returns high word result)
/	;divide (returns quotient)
//	;divide (returns remainder)
MAX	;make less than or equal to
MIN	;make greater than or equal to
&	;and
	;or (typed as SHIFT + \ on UK
^	;xor (typed as SHIFT + 6 on UK
&/	;and not (nand)
∕	;or not (nor)
^∕	;xor not (xnor)

Ejemplo:

inicio:

let b0 = b0 + 1	‘incrementa b0 en una unidad
sound 7, (b0,50)	‘emite sonido
if b0 > 50 then reset	‘si la variable es superior a 50 salta a reset
goto inicio	‘sino a inicio

reset:

let b0 = b0 max 10	‘resetea b0 con un valor 10 y 10 es el máximo valor
goto inicio	

lookdown

LOOKDOWN target,(valor0,valor1...valorN),variable

La declaración LOOKDOWN busca en una lista de 8 bits los valores (*value0...valueN*) que coincidan con un valor *target*. Si se encuentra, el índice de la constante es almacenado en *variable* así, 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.

Ejemplo:

serin 1,N2400,b0	‘obtiene un carácter hex de pin1 en ‘forma serie
lookdown bo,(“0123456789ABCDEF”),b1	‘convierte el character hex en b0 a un ‘valor decimal b1
serout 0,N2400,[#b1]	‘envía un valor decimal al pin0 de ‘forma serie

Otro ejemplo:

```
reset:
    let infra = 0                'resetea la variable infra
inicio:
    if infra > 0 then test      'salta a reset si se detecta una nueva señal
    goto inicio                'sino salta a inicio

test:
    lookdown infra, ("1234"),b0 'mete el carácter ASCII 1 a 4 en b0
```

lookup

LOOKUP offset,(dato0,dato1...datoN),variable

LOOKUP puede ser usado para obtener valores de una tabla de constantes de 8 bits. Si offset es cero, variable toma el valor del primer dato (dato0). Si es 1, variable toma el valor del segundo dato (dato1) y así sucesivamente. Si offset es mayor ó igual que el número de entradas en la lista de constantes, no se toma ninguna acción y variable permanece sin cambios.

Ejemplo:

```
for b0 = 0 to 5                'cuenta hasta 5
lookup b0,("hola"),b1         'obtiene el carácter b0 de la serie y lo deja en b1
serout 0,N2400,[b1]           'envía el carácter b1 al pin0 de forma serie
```

Otro ejemplo:

```
inicio:
    let b0 = 0 b0 + 1          'incrementa b0 en una unidad
    lookup b0,("1234"),b1     'mete el carácter ASCII 1 a 4 en b1
    if b0 < 4 then inicio     'si b0 es menor que 4 salta a inicio
    end                       'sino fin de programa
```

low

LOW pin

Coloca el pin especificado en valor bajo y automáticamente lo convierte en salida. Pin puede ser una variable/constante de 0-7.

Ejemplo:

```
inicio:
    high 1                    'pone el pin 1 a nivel alto
    pause 5000                'espera de 5 segundos
    low 1                     'pone el pin 1 a nivel bajo
    pause 5000                'espera de 5 segundos
    goto inicio               'salta a inicio del bucle
```

nap

NAP periodo

Coloca al microcontrolador en modo de bajo consumo por períodos de tiempo reducidos. Durante este NAP, se reduce al mínimo el consumo de energía. Los períodos indicados son solo aproximados, porque el tiempo se deriva del Watchdog Timer que está controlado por R/C y puede variar de chip a chip y también con la temperatura. Como NAP usa el Watchdog Timer es independiente de la frecuencia del oscilador.

- *Periodo: constante entre 0-7.*

El número de períodos está limitado a 7 y el tiempo correspondiente a cada período sigue la formula: $(2^{\text{periodo}}) \times 18 \text{ ms}$

Período	Demora (aprox.) en ms
0	18
1	36
2	72
3	144
4	288
5	576
6	1152
7	2304

Ejemplo:

inicio:

high 1 ‘pone salida pin1 a nivel alto
nap 4 ‘estado de bajo consumo durante: $(2^4) \times 18 \text{ms} = 288 \text{ ms}$
low 1 ‘pone salida 1 a nivel bajo
nap 7 ‘estado de bajo consumo durante: $(2^7) \times 18 \text{ms} = 2304 \text{ ms}$
goto inicio ‘salta a inicio

output

OUTPUT pin

Convierte el Pin especificado en una salida. Pin debe ser una constante entre 0-7, o una variable que contenga un número 0-7 (p. ej.: B0) o el nombre de un pin (p. ej.: PORTA.0)

Ejemplo:

inicio:

input 1 ‘configura el pin1 como entrada
reverse 1 ‘configura el pin 1 como salida
reverse 1 ‘configura el pin1 como entrada
output1 ‘configura el pin1 como salida

pause

PAUSE periodo

Detiene el programa por periodo milisegundos. Periodo tiene 16 bits, por lo que los retardos pueden ser hasta 65635 milisegundos (un poco más de un minuto). No coloca al microcontrolador en modo de bajo consumo como las otras funciones de retardo (NAP y SLEEP). Inclusive, consume mayor consumo, pero es más exacto. Tiene la misma precisión que el clock.

Ejemplo:

inicio:

high 1	'pone el pin 1 a nivel alto
pause 5000	'espera de 5 segundos
low 1	'pone el pin 1 a nivel bajo
pause 5000	'espera de 5 segundos
goto inicio	'salta a inicio del bucle

peek

PEEK direccion,variable

*Lee el registro del microcontrolador en la **dirección** y guarda la lectura en **variable**.*

- *Direccion: variable/constante que especifica la dirección del registro. Valores válidos entre 0-255.*
- *Variable: variable donde se guarda el dato del registro.*

Esto permite el uso de registros no definidos por b0-b13.

Es necesario tener precauciones en el uso de este comando. Para programadores inexpertos se recomienda el uso de los registros localizados entre \$50 a \$7F, que son registros de uso general.

Los registros indicados entre las direcciones \$00 a \$1F y \$80 a \$9F, son de función especial, que determinan como funciona el microcontrolador. Evitar el uso de estos registros a no ser que se sepa bien lo que se hace.

Las direcciones \$20 a \$7F y \$A0 a \$BF apuntan a registros de propósito general reservados para para el intérprete de instrucciones PICAXE. El uso de esos registros puede dar resultados inesperados y podría causar la no interpretación de instrucciones.

Utilizar los registros apuntados por las direcciones que van desde \$50 a \$7F (registros de propósito general) y de \$C0 a \$FF en el caso de PICAXE-28x.

Ejemplo:

peek 80,b1	'carga el contenido del registro apuntado por la dirección 80 en 'la variable b1
------------	---

peek, PORTA,b0	'toma el estado actual de PORTA y lo coloca en b0
----------------	---

poke

POKE direccion,variable

Guarda el valor de **variable** en el registro del microcontrolador indicado en **dirección**.

- *Dirección: variable/constante que especifica la dirección del registro. Valores válidos entre 0-255.*
- *Variable: variable que contiene el dato a cargar en el registro especificado.*

Lo indicado en el comando peek es válido para el comando poke, en cuanto a las precauciones a tomar en su uso.

Ejemplo:

poke 80,b1 ‘guarda el valor de b1 en el registro 80

poke \$85,0 ‘guarda 0 en el registro 85 en hexadecimal (set todo
 ‘PORTA como salidas)

Se puede acceder directamente a registros y bits sin necesidad de utilizar PEEK y POKE. Se recomienda usar el acceso directo y no los comandos mencionados.

TRISA = 0 ‘set todo PORTA como salidas
PORTA.0 = 1 ‘set a nivel alto el bit 0 de PORTA

pulsin

PULSIN pin,state,variable

Mire el ancho del pulso en **pin**. Si **state** es cero se mide el ancho de un pulso bajo. Si **state** es uno, se mide el ancho de un pulso alto. El ancho medido se coloca en **variable**. Si el flanco del pulso no llega, ó el ancho del pulso es demasiado grande para ser medido, **variable** = 0. Si se usa una variable de 8 bits, solo se usan los bits menos significativos de la medición de 16 bits.

- *Pin: variable/constante (0-7) que especifica en pin I/O que se usará.*
- *State: variable/constante (0 o 1). Configura el tipo de pulso a medir en unidades de 10µs (para un cristal de 4MHz).*
- *Variable: variable (1-65536) que contiene el dato a cargar en el registro especificado. Si el intervalo de espera ocurre (.65536s), entonces el resultado será 0.*

La resolución de PULSIN depende de la frecuencia del oscilador. Si se usa un oscilador de 4 MHz, el ancho de pulso se obtiene en incrementos de 10µs. Si se usa un oscilador de 20 MHz, el ancho de pulso tendrá una resolución de 2µs. Definir un valor de OSC no tiene efectos sobre PULSIN. La resolución siempre cambia con la velocidad del oscilador en uso.

Ejemplo:

pulsin 3,1,b1 'guarda la longitud del pulso introducido en el pin 1 en la variable b1

pulsout

PULSOUT pin,periodo

*Genera un pulso en **pin**, con un **periodo** especificado. El pulso se genera activando dos veces el pin, por lo que la polaridad del pulso depende del estado inicial del pin.*

- *Pin: variable/constante (0-7) que especifica em pin I/O que se usará.*
- *Período: variable/constante que especifica el período (0-65535)em unidades de 10µs (para um cristal de 4MHz).*

La resolución de PULSOUT depende de la frecuencia del oscilador. Si se usa un oscilador de 4 MHz, el período del pulso generado estará en incrementos de 10µs. Si se usa un oscilador de 20 MHz, período tendrá una duración de 2µs. Definir un valor de OSC no tiene efectos sobre PULSOT. La resolución siempre cambia con la velocidad del oscilador en uso.

Ejemplo:

inicio:

```
pulsout 4,150      'envía un pulso por el pin 4 de 1,5 ms de duración
pause 20           'pausa de 20 ms
goto inicio        'salta a inicio y repite formando un bucle
```

pwm

PWM pin,duty,cicle

*Envía un tren de pulsos modulados en ancho por el **pin** indicado. Cada ciclo de PWM está compuesto de 256 pasos. El ciclo útil **duty** para cada ciclo varía de 0 (0%) a 255 (100%). El ciclo PWM es repetido **cycle** veces.*

- *Pin: variable/constante (0-7) que especifica em pin I/O que se usará.*
- *Duty: variable/constante (0-255)especificando el ciclo útil que puede variar de 0 (0%) a 255 (100%).*
- *Cycle: variable/constante (0-255) que especifica el número de ciclos que se repetirá.Cada ciclo tiene un período de 5 ms (con un cristal de 4MHz).*

Cycle depende de la frecuencia del oscilador. Con un oscilador de 4 MHz, cada cycle será de aproximadamente 5 ms de largo. Con un oscilador de 20 MHz la duración del ciclo será de 1 ms aproximadamente. Definir un valor de OSC no tiene efecto sobre PWM. El tiempo de cycle siempre cambia con la velocidad del oscilador en uso.

Pin se convierte en salida justo antes de la generación del pulso y vuelve a ser entrada, cuando cesa. La salida de PWM en un pin tiene mucho ruido, y no tiene forma de onda cuadrada. Es necesario usar algún tipo de filtro para convertirla en algo útil. Un circuito R/C puede usarse como un simple convertidor A/D.

Ejemplo:

inicio:

pwm 4,127,20	'envía 20 pulsos por el pin 4 con un ciclo útil del 50%
pause 20	'pausa de 20 ms
goto inicio	'salta a inicio

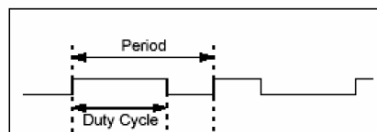
pwmout

PWMOUT pin,period,duty cycles

Genera una salida continua de pwm usando el módulo interno del pwm del microcontrolador.

Esta orden es diferente de otras ordenes semejantes de BASIC, el **pwmout funciona continuamente** hasta que otra orden de **pwmot** sea indicada. Por ello puede usarse, por ejemplo, para controlar la velocidad de un motor. Para detener la orden **pwmout**, enviar otra orden con un **period = 0**.

- **Pin:** variable/constante (0-7) que especifica em pin I/O que se usará (siempre 3 con 18X, 2 con 08M y 1 o 2 con el 28X/40X).
- **Period:** variable/constante (0-255) especificando el período del pulso.
- **Duty:** variable/constante (0-1023) especificando el ciclo de trabajo de la señal pwm.



El pwm period = (period + 1) x 4 x velocidad del oscilador

(velocidad del oscilador de 4 Mhz = 1/4000000)

El pwm duty cycle = (duty) x resonador speed

Notar que el valor de period y el valor de duty siguen la ecuación indicada anteriormente. Si desea una relación de 50:50 mientras se incrementa el período, entonces debe aumentar el valor del ciclo de trabajo apropiadamente. Un cambio de la velocidad del oscilador cambiará la fórmula.

La frecuencia del pwm = 1/(period pwm).

Como el comando utiliza el módulo interno del pwm del microcontrolador, hay ciertas restricciones para su uso:

1. El comando solo funciona con ciertos pines de los microcontroladores:
 - 28X/40X (1 y 2)
 - 18X (3)
 - 08M (2)
2. Duty cycles es un valor de 10 bits (0 a 1023).
3. El comando servo no se puede usar al mismo tiempo que el comando **pwmout** dado que utilizan la misma señal de reloj.
4. **Pwmout** deja de funcionar durante las ordenes de `, nap, sleep` o `end`.

Ejemplo:

inicio:

pwmot 2,150,100	'set pwm
pause 1000	'pausa de 1s.
goto inicio	'salta a inicio

random

RANDOM variable

Efectúa una iteración pseudos-aleatoria en variable.

- *Variable: variable de 16 bits donde se almacena el número aleatorio.*

El algoritmo pseudoaleatorio usado tiene un paso de 65535 (el único número que no produce es el cero).

Ejemplo:

inicio:

random b1	'carga en la variable b1 un valor aleatorio
let pins = b1	'pone el valor aleatorio en la salida pins
pause 100	'espera de 0,1 s
goto inicio	'salta a inicio

readadc

READADC canal,variable

El contenido del convertidor A/D (8 bits de resolución) indicado por canal es almacenado en variable.

- *Canal:variable/constante que especifica el canal (0-3)*
- *Variable: variable donde se almacena el byte leído.*

Ejemplo:

inicio:

readadc 1,b1	'lee el valor y lo almacena en b1
if b1 > 50 then flash	'salta a flash si b1 > 50
goto inicio	'sino salta a inicio

flash:

high 1	'pone a nivel alto el pin 1
pause 5000	'espera de 5 segundos
low 1	'pone a nivel bajo la salida 1
goto inicio	'salto a inicio

readadc10

READADC canal,wordvariable

El contenido del convertidor A/D (10 bits de resolución) indicado por canal es almacenado en variable.

- *Canal:variable/constante que especifica el canal (0-3)*
- *Wordvariable: variable tipo Word donde se almacena los bytes leídos.*

Ejemplo:

inicio:

readadc 1,w1	'lee el valor y lo almacena en w1
if w1 > 50 then flash	'salta a flash si b1 > 50
goto inicio	'sino salta a inicio

flash:

high 1	'pone a nivel alto el pin 1
pause 5000	'espera de 5 segundos
low 1	'pone a nivel bajo la salida 1
goto inicio	'salto a inicio

readi2c

READI2C location,(variable,...)

*Lee el i2c indicado en **location** y lo almacena en **variable(s)**.*

- *location: variable/constante que especifica la dirección mediante un byte del tipo Word la dirección del i2c.*
- *Variable(s): donde se almacenan los byte(s) del dato leído.*

Este comando se usa para leer los datos de byte de un dispositivo i2c.

Location define la dirección de inicio de la lectura de datos, aunque se logra también leer más de un byte secuencialmente (si el dispositivo i2c da soporte a lecturas secuenciales).

Location debe ser un byte que defina al 12cslave. Un comando del 12cslave tiene que haberse ejecutado con anterioridad de la ejecución del readi2c.

Si el hardware del i2c no fue configurado de forma correcta o el byte de localización no es el adecuado el valor almacenado en la variable será 255 (\$FF) indicándonos un error.

Ejemplo:

‘Un ejemplo de cómo usar el time clock DS1307
‘los datos son enviados/recibidos en código BCD
‘coloca la dirección del esclavo del DS1307

```
12cslave %11010000, i2cslow, i2cbyte
```

‘lectura de la hora y fecha y presenta en el depurador

inicio:

```
readi2c 0, (b0,b1,b2,b3,b4,b5,b6,b7)  
debug b1  
pause 2000  
goto inicio
```

read

READ location,variable

Lee la EEPROM incorporada en la dirección location, y guarda el resultado en variable.

- *location: variable/constante que especifica la dirección mediante un byte (0-255).*
- *Variable: donde se almacena el byte leído.*

Al usar los módulos Stamp, la forma de almacenar los datos es desde la posición 0 hacia arriba mientras que el almacenamiento de programa se construye hacia abajo desde la posición 255. Al usar el PICAXE-08 y 18el almacenamiento de datos se construye hacia arriba desde la posición 0y el almacenamiento de programa es hacia abajo desde la posición 127.

Ejemplo:

inicio:

```
for b0 = 0 to 63          ‘inicio el bucle  
read b0,b1              ‘lee el dato de la memoria en la posición de bo y  
                          ‘almacena su valor en b1  
serout 7,T2400,(b1)     ‘transmite el valor del dato de b1 al LCD utilizando  
                          ‘transmisión serie  
next b0
```

readmem

READMEM location,variable

*Lee la memoria de programa FLASH en la posición **location** y lo vuelca en **variable**. Provee un espacio adicional de 256 bytes de almacenamiento para este comando en los dispositivos PIC 16F87x (incluyendo a PICAXE-28, 28^a). Este comando no está presente en el PICAXE-28X como el módulo del i2c puede servir para EEPROMs externas mayores.*

- *location: variable/constante que especifica la dirección mediante un byte (0-255).*
- *Variable: donde se almacena el byte leído.*

Ejemplo:

inicio:

```
for b0 = 0 to 255      'inicio el bucle
read b0,b1           'lee el dato de la memoria en la posición de bo y
                    'almacena su valor en b1
serout 7,T2400,(b1)  'transmite el valor del dato de b1 al LCD utilizando
                    'transmisión serie
next b0
```

readtemp/readtemp12

READTEMP pin,variable

READTEMP pin,wordvariable

*Lee la temperatura de un DS18B20, sensor digital de temperatura, conectado en **pin** y almacena su lectura en **variable**.*

- *Pin: es el pin de entrada del PICAXE que tiene conectado el sensor.*
- *Variable: donde se almacena el byte leído.*

READTEMP: la resolución de la lectura es de enteros de grado, y el sensor funciona de -55 °C hasta +125 °C. El bit 7 indicará si los valores de temperatura son positivos (0) o negativos (1).

READTEMP12: (para programadores avanzados). La temperatura viene expresada con una resolución de 0,125 °C y necesita 12 bits para su representación digital. El usuario debe interpretar los datos a través de cálculo matemático. Ver el dataste del DS18B20 (www.dalsemi.com) para más información en relación con Data Temperatura.

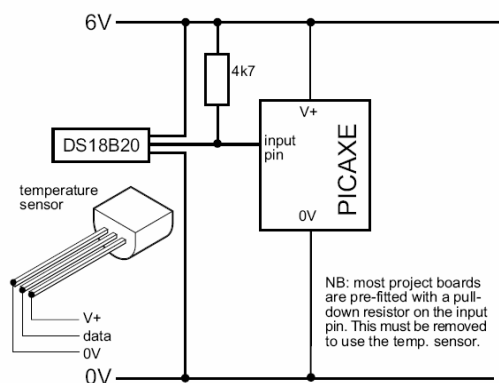
Ejemplo:

inicio:

```
readtemp 1,b1      'lee el valor del sensor en pin 1 y lo almacena en b1
if b1 > 127 then neg  'chequeo si el valor medido es negativo
serout, 7, T2400, (#b1) 'transmite el valor al LCD en formato serie
goto inicio        'salta a inicio
```

neg:

```
let b1 = b1 - 128  'ajuste del valor negativo en b1
serout 7,T2400, ("-") 'transmite el símbolo negativo
serout 7,T2400, (#b1) 'transmite el valor al LCD en formato serie
goto inicio
```



readowclk

READOWCLK pin

*Lectura de la hora indicada por un reloj de precisión DS2415 conectado en la entrada especificada por **pin**.*

- *Pin: es el pin de entrada (0-7) del PICAXE que tiene conectado el sensor.*

Este comando no esta disponible en el PICAXE-28X pero puede usarse un DS1307, reloj de precisión en tiempo real por i2c.

El DS2415 es un reloj en tiempo real de precisión que opera con una única línea. La información está contenida en 4 bytes (32 bits) y es muy preciso debido al uso de un cristal de cuarzo como clock del C.I. El registro de 32 bits puede aportar la información de 132 años en segundos. El dispositivo puede estar alimentado de forma independiente con 3 V. de forma que pueda trabajar aún cuando el controlador principal con el PICAXE esté parado. Si se utiliza ésta opción se debe utilizar la orden RESETOWCLK para activar el reloj de nuevo y de esta forma arrancar el reloj en tiempo real.

El comando READOWCLK lee los 32 bits del reloj y guarda el contenido de los 32 bits en la variable b10 (LBS) a b13 (MBS), también conocidas como w6 y w7.

Usando variables tipo byte:

- El valor b10 es el número de segundos
- El valor b11 es el número x 256 segundos
- El valor b12 es el número x 65536 segundos
- El valor b13 es el número x 16777216 segundos

Usando variables tipo Word:

- El valor en w6 es el número de segundos
- El valor en w7 es el número x 65536 segundos

Ejemplo:

inicio:

resetowclk 2 'reset el reloj en pin 2

bucle:

readowclk 2 'lectura del reloj en pin de entrada 2
debug b1 'visualiza el tiempo en depurador
pause 10000 'espera de 10 segundos
goto bucle 'repetición de lectura y presentación

resetowclk

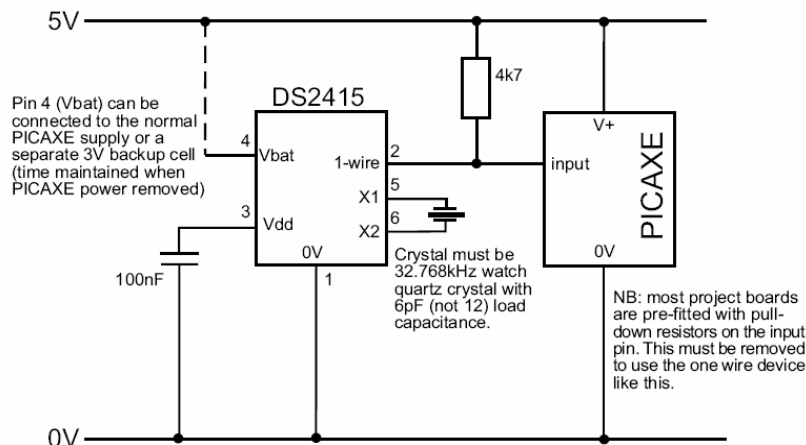
RESETOWCLK pin

Resetea el reloj DS2415, es decir pone a cero su contador.

- *Pin: es el pin de entrada (0-7) del PICAXE que tiene conectado el sensor*

Este comando pone a cero el tiempo en un chip reloj tipo DS2415 que está conectado a un pin de salida del PICAXE. También activa el cristal del reloj. Debe usarse al comienzo de cada conteo.

El método de conexionado del reloj de precisión con el PICAXE se muestra a continuación:



readownsn

READOWSN pin

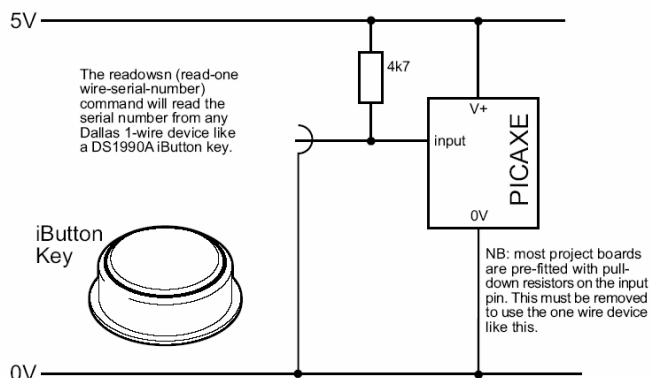
*Lectura serie de cualquier dispositivo Dallas 1-wire, conectado en el pin de entrada especificado por **pin**.*

- *Pin: es el pin de entrada (0-7) del PICAXE que tiene conectado el sensor.*

Este comando lee el valor de cualquier dispositivo Dallas de un Terminal i/o de comunicación (1-wire) que esté conectado al PICAXE en una de sus pines (especificado por **pin**), por ejemplo, sensor digital de temperatura: DS18B20, reloj de precisión: DS2415, iButton: DS1990A (http://www.esicomsa.com/product_1.html).

Al usar un dispositivo iButton cuyo número serie está grabado en el chip que contiene la envoltura del iButton y ejecutar el comando readownsn se realizará la lectura del número serie y guardará el código indicador de la familia en b6, el número de serie en b7..b12 y la suma de verificación en b13.

Nunca deberá utilizar los registros b6 a b13 para otros propósitos durante la ejecución del comando readownsn.



Ejemplo:

inicio:

let b6 = 0

‘reset el registro, con valor cero

‘bucle para leer el código hasta que b6 no sea igual a 0

bucle:

readownsn 2
if b6 = 0 then bucle

‘lectura serie del número en el pin de entrada 2
‘repite lectura hasta que b6 no sea 0

‘hace una comprobación simple de seguridad.
‘Lo normal es que b12 no contenga
‘nunca el valor FF, Si el valor FF existe quiere
‘decir que la lectura del dispositivo
‘ya fue efectuada o a ocurrido un cortocircuito

If b12 = \$FF then inicio

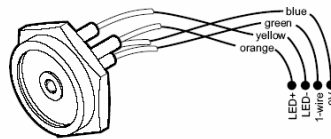
‘si el valor b12 es FF salta a inicio
‘todo esta correcto así que continúa
‘presenta el número en el depurador
‘espera de 1 segundo

debug b1
pause 1000

goto inicio

‘comienzo de programa

Conexiones del dispositivo iButton



return

RETURN

Vuelve desde una subrutina. Retoma la ejecución en la declaración que sigue al GOSUB que llamó la subrutina.

Ejemplo:

inicio:

let b2 = 15	‘asigna a la variable b2 el valor 15
pause 2000	‘espera de 2 segundos
gosub flash	‘llamada a subrutina flash
let b2 = 5	‘asigna a la variable b2 el valor 5
pause 2000	‘espera de 2 segundos
gosub flash	‘llamada a subrutina flash
end	

flash:

for b0 = 1 to b2	‘define el bucle para el tiempo de b2
high 1	‘pone la salida 1 a nivel alto
pause 500	‘espera 0,5 segundos
low 1	‘pone la salida 1 a nivel bajo
pause 500	‘espera 0,5 segundos
next b0	‘fin del bucle
return	‘vuelve a línea siguiente de llamada

reverse

REVERSE pin

*Si **pin** es entrada, lo convierte en salida. Si es salida, lo convierte en entrada.*

- *Pin: es el pin de entrada (0-7) del PICAXE.*

Ejemplo:

inicio:

```
input 1      'configura pin 1 como entrada
reverse 1    'configura pin 1 como salida
reverse 1    'configura pin 1 como entrada
output 1     'configura pin 1 como salida
```

serin

```
SERIN pin,baudmode,(qualifier,qualifier...)
SERIN pin,baudmode,(qualifier,qualifier...),{#}variable,{#}variable...
SERIN pin,baudmode ,{#}variable,{#}variable...
```

Recibe uno o más conjunto de datos en **pin**, en formato Standard asíncrono, usando 8 bit de datos, sin paridad y un bit stop (8N1). Pin automáticamente se convierte en entrada.

- *Pin:* es un pin de entrada del PICAXE (0-7).
- *Baudmode:* es una variable/constante (0-7) que especifica el modo en el que se transferirá el/los dato/s:

T2400	true input
T1200	true input
T600	true input
T300/T4800	true input
N2400	inverted input
N1200	inverted input
N600	inverted input
N300/N4800	inverted input

- *Qualifiers:* son variables/constants opcionales (0-255). La lista de datos a recibir puede estar precedida por uno o más **quqlifiers** encerrados entre corchetes. SERIN debe recibir estos bytes en un orden exacto, antes de recibir los datos. Si algún byte recibido no concuerda con el byte siguiente de la secuencia de calificación, el proceso de calificación comienza nuevamente (p.e. el próximo byte recibido es comparado con el primer byte de la lista de calificación).
- *Variable/s:* son variables/constantas opcionales (0-255). SERIN comienza a guardar datos en la variable asociada. Si el nombre de variable es único, el valor del carácter ASCII recibido es guardado en la variable. Si la variable es precedida por el signo #, SERIN convierte un valor decimal en ASCII y guarda el resultado en esa variable.

4800 y 9600 están sólo disponibles en el PICAXE-28X. El microcontrolador puede no llegar a seguir datagramas complicados a esa velocidad – se recomienda un máximo de 2400 para un reloj de 4 MHz.

Ejemplo:

inicio:

```

for b0 = 0 to 63           'comienzo bucle
  serin 6,T2400,b1        'recibe serie de datos
  write b0,b1             'guarda valor en b1
next b0                   'siguiente ciclo
    
```

serout

SEROUT pin,baudmode,({#}data, {#}data ...)

Envía uno o más conjunto de datos en **pin**, en formato Standard asíncrono, usando 8 bit de datos, sin paridad y un bit stop (8N1). Pin automáticamente se convierte en salida.

- *Pin: es un pin de salida del PICAXE (0-7).*
- *Baudmode: es una variable/constante (0-7) que especifica el modo en el que se transferirá el/los dato/s:*

T2400	True output	Always driven
T1200	True output	Always driven
T600	True output	Always driven
T300/T4800	True output	Always driven
N2400	Inverted output	Always driven
N1200	Inverted output	Always driven
N600	Inverted output	Always driven
N300/N4800	Inverted output	Always driven
OT2400	True output	Open driven
OT1200	True output	Open driven
OT600	True output	Open driven
OT300/OT4800	True output	Open driven
ON2400	Inverted output	Open source
ON1200	Inverted output	Open source
ON600	Inverted output	Open source
ON300/OT4800	Inverted output	Open source

- *Variable/s: son variables/constantes opcionales (0-255). Contienen el byte/s que se enviará por el pin de salida. Si la variable es precedida por el signo #, SEROUT convierte un valor decimal en ASCII y guarda el resultado en esa variable. Se puede enviar cadenas de texto, precedidas por ""("Hola").*

Ejemplo:

inicio:

for b0 = 0 to 63	‘comienzo bucle
read b0,b1	‘leer valor y cargar en b1
serout 7,T2400,(b1)	‘transmite valor de b1
next b0	‘siguiente ciclo

sertxd

SERTXD ({#}data, {#}data ...)

Envía uno o más conjunto de datos en **pin de programación**, en formato Standard asíncrono, usando 8 bit de datos, sin paridad y un bit stop (8N1) a 4800 baudios..

- *Variable/s: son variables/constantes opcionales (0-255). Contienen el byte/s que se enviará por el pin de salida. Si la variable es precedida por el signo #, SERTXD convierte un valor decimal en ASCII y guarda el resultado en esa variable. Se puede enviar cadenas de texto, precedidas por "" ("Hola").*

Ejemplo:

inicio:

for b0 = 0 to 63	‘comienzo bucle
read b0,b1	‘leer valor y cargar en b1
sertxd (b1)	‘transmite valor de b1
next b0	‘siguiente ciclo

servo

SERVO pin,pulse

Envía un pulso por el **pin** indicado para el control de un dispositivo de radio control tipo servo.

- *Pin: es un pin de salida del PICAXE (0-7).*
- *Pulse: es una variable/constante (75-225) que especifica la posición del servo.*

Este comando pone el **pin** a nivel alto durante un tiempo (x0.01 ms) cada 20ms indicado por **pulse**.

Generalmente el servo de RC requiere un pulso (0.75 a 2.25ms de duración) cada 20ms. Por consiguiente el comando **servo 1,75** moverá el servo a la posición de **0 grados**, con el comando **servo 1,225** se posicionará en el extremo opuesto **180 °**. Con el comando **servo 1,150** colocará el servo en la **posición central**.

El comando servo no se puede usar al mismo tiempo que **pwmout** dado que comparten un contador común.

No usar un valor de pulso menor de 75 o mayor de 255, dado que esto puede causar que el servo funcione incorrectamente. Debido a las tolerancias en la fabricación de los servos todos los valores son aproximados y requieren un ajuste fino por experimentación.

Ejemplo:

inicio:

servo 4,75	'mueve el servo a la posición inicial 0 grados
pause 2000	'espera 2 segundos
servo 4,150	'mueve el servo a la posición central
pause 2000	'espera 2 segundos
servo 4,225	'mueve el servo a la posición final 180 grados
pause 2000	'espera 2 segundos
goto inicio	'salta a inicio y repite ciclo

setint

SETINT input,mask

Interrupción en función de ciertas condiciones de entrada.

- *Input: es una variable/constante (0-7) que especifica las condiciones de entrada.*
- *Mask: es una variable/constante (75-225) que especifica la máscara.*

Este comando causa la interrupción del código en ejecución cuando se activa un pin, previamente configurado, de interrupción.

La interrupción es el método más rápido de atender a una petición de paro del microcontrolador en función de condiciones determinadas. Es el único tipo de interrupción disponible en el sistema PICAXE.

El pin de interrupción se comprueba entre la ejecución de cada línea de código del programa y inclusive durante cualquier orden de pausa.

Si la condición particular de entrada es cierta, un **gosub** para la subrutina de interrupción se ejecuta inmediatamente. Cuando remata la ejecución de la subrutina de interrupción, el programa continúa en la línea siguiente desde donde se produjo la llamada a la interrupción del programa principal.

La condición de entrada de interrupción es cualquier patrón de "0" y "1" en el "Terminal de entrada de interrupción", enmascarado por el byte **mask** (máscara). Por consiguiente cualquier bit enmascarado por un "0" en la máscara será ignorado.

Ejemplos:

*Para interrumpir con entrada **input1** a nivel **alto***

```
Setint %00000010, % 00000010
```

*Para interrumpir con entrada **input1** a nivel **bajo***

```
Setint %00000000, % 00000010
```

Para interrumpir con entrada *input0* a nivel alto, *input1* a nivel alto e *input2* a nivel bajo

Setint %00000011, % 00000111

Solo un patrón de entrada es admitido en cualquier momento. Para desactivar la interrupción ejecutar una orden *SETINT* con el *valor 0* como *byte de máscara*.

Notas:

- 1) Cada programa que usa la orden *setint* debe tener la correspondiente subrutina de interrupción (termina con el correspondiente retorno al programa principal con *return*) al final del programa principal.
- 2) Cuando ocurre una interrupción esta será atendida inmediatamente quedando inhabilitada cualquier interrupción del programa principal. Si se deseara ejecutar nuevamente una interrupción, esta deberá programarse dentro de la subrutina de interrupción en ejecución. Queda claro que después de ser atendida una interrupción y ejecutar la subrutina asociada quedan habilitadas las peticiones de interrupción del programa principal.
- 3) Si la condición de interrupción no es atendida dentro de la subrutina, una segunda llamada de interrupción puede ocurrir inmediatamente.
- 4) Después de ser ejecutado el código de la subrutina de interrupción, la ejecución del programa continúa en la siguiente línea del programa principal. Si la interrupción se produce durante la ejecución de un comando *pause*, después de la ejecución de la subrutina de interrupción y retorno al programa principal ignorará el tiempo que restaba de la ejecución del comando *pause* y continuará en la línea siguiente.

Ejemplo:

setint %10000000,%10000000 'activa interrupción cuando el pin7 esta a nivel alto

principal:

low 1	'conmuta la salida 1 a nivel bajo
pause 2000	'espera 2 segundos
goto principal	'repite bucle principal

interrupt:

high 1	'conmuta la salida 1 a nivel alto
if pin7 = 1 then interrupt	'salta a la subrutina interrupt hasta que se cumpla la condición indicada
pause 2000	'espera 2 segundos
setint %10000000,%10000000	'reactiva la interrupción
return	'termina la subrutina de interrupción y retorna al programa principal

En el ejemplo, la salida 1, conectada a un LED estará apagado y se pondrá a nivel alto durante 2 segundos cuando se active la interrupción (pin 7 a nivel alto), reactivando de nuevo la interrupción.

A continuación se muestra una variación del anterior ejemplo:

principal:

low 1	'conmuta la salida 1 a nivel bajo
pause 2000	'espera 2 segundos
if pin7 = 1 then sw_on	
goto principal	'repite bucle principal

sw_on:

high 1	'conmuta la salida 1 a nivel alto
if pin7 = 1 then sw_on	'salta a la subrutina sw_on hasta que se cumpla la condición indicada
pause 2000	'espera 2 segundos
return	'termina la subrutina de interrupción y retorna al programa principal

setfreq

SETFRQ freq

Establece la frecuencia interna del reloj para el microprocesador PICAXE-18X para 4 MHz (por defecto) o 8 MHz.

- *freqt: es un keyword m4 o m8*

Ejemplo:

setfreq m8	'frecuencia interna a 8 MHz
------------	-----------------------------

El cambio no ocurre hasta que el interruptor de arranque se active de nuevo, y el valor es almacenado en memoria interna siendo, ahora, el cambio permanente.

sleep

SLEEP seconds

*Coloca al microcontrolador en modo de bajo consumo (duerme) por un periodo de tiempo indicado en **seconds** segundos. Los retardos pueden ser de hasta 65535 segundos (aprox. 18 horas).*

- *seconds: variable/constante que especifica la duración del sleep (0-65535)*

SLEEP usa el WatchDog Timer, por lo que es independientemente de la frecuencia del oscilador utilizado. La resolución es de aproximadamente 2,3 s y la exactitud de aproximadamente un 99,9%, aunque puede variar de acuerdo al dispositivo y la temperatura.. El consumo se reduce en una relación 1/100.

Ejemplo:

inicio:

```
high 1          'salida 1 a nivel alto
sleep 10        'duerme durante 23 segundos aproximadamente
low 1           'salida 1 a nivel bajo
sleep 100       'duerme 230 segundos aproximadamente
goto inicio     'repite programa indefinidamente
```

symbol

SYMBOL symbolname = value

Asigna un **symbolname** (símbolo identificativo) con un valor **value** que lo identifica con más claridad en la elaboración e interpretación del código del programa.

- *Symbolname*: es una cadena de texto que debe comenzar con un carácter literal después se puede utilizar los caracteres numéricos (0..9)
- *Value*: es una variable que dan valor al nuevo símbolo

Ejemplo:

```
symbol LED_ROJO = 7      'define una constante symbol
symbol CONTADOR = B0     'define una constante symbol

let CONTADOR = 200      'precarga CONTADOR con el valor 200
```

inicio:

```
high LED_ROJO          'salida 7 a nivel alto
pause CONTADOR         'pausa de 0,2 segundos
low LED_ROJO           'salida 7 a nivel bajo
pause CONTADOR         'pausa de 0,2 segundos
goto inicio            'repite programa indefinidamente
```

sound

SOUND pin,(note,duration,note duration...)

Genera un tono y/o ruido blanco en el pin especificado. Pin se configura automáticamente como salida.

- *Pin*: es una variable/constante (0-7) que especifica el pin i/o a utilizar.
- *Note(s)*: es una variable o constante (0-255) que especifica la frecuencia del sonido a emitido.
 - *Note 0*: silencio
 - *Note (1-127)*: tonos
 - *Note (128-255)*: ruido blanco
- *Duration*: es una variable/constante (0-255) que determina la duración de la nota, en incrementos de 12 ms.

Los tonos y el ruido blanco están en una escala ascendente (p.e. 1 y 128 son las frecuencias menores, 129 y 266 las mayores). Note 1 es aproximadamente 78,74 Hz y note 127 es aproximadamente 10000Hz.

SOUND entrega como salida ondas cuadradas con nivel TTL. Gracias a las características del micro PIC, se puede manejar un altavoz a través de un condensador. El valor del condensador debe ser determinado en función de las frecuencias a usar y la carga del altavoz. Altavoces piezoeléctricos se pueden conectar directamente.

Ejemplo:

inicio:

```
let b0 = b0 + 1
sound 7,(b0,50)
goto inicio
```

```
'incrementa b0
'configura y emite sonido por pin 7
'repite programa indefinidamente
```

swith on

```
SWITH ON, pin
SWITHON, pin
```

Configura **pin** como salida a nivel alto.

- *Pin: es una variable/constante (0-7) que especifica el pin i/o a utilizar.*

Ejemplo:

inicio:

```
switch on 7
wait 5
switch off 7
wait 5
goto inicio
```

```
'configura pin 7 como salida a nivel alto
'espera 5 segundos
'configura pin 7 como salida a nivel bajo
'espera 5 segundos
'repite programa indefinidamente
```

swith off

```
SWITH OFF, pin
SWITHOFF, pin
```

Configura **pin** como salida a nivel bajo.

- *Pin: es una variable/constante (0-7) que especifica el pin i/o a utilizar.*

Ejemplo:

inicio:

switch on 7	'configura pin 7 como salida a nivel alto
wait 5	'espera 5 segundos
switch off 7	'configura pin 7 como salida a nivel bajo
wait 5	'espera 5 segundos
goto inicio	'repite programa indefinidamente

toggle pin

TOGGLE pin

Invierte el estado del pin especificado. Pin es configurado automáticamente como salida.

- *Pin: es una variable/constante (0-7) que especifica el pin i/o a utilizar.*

Ejemplo:

inicio:

toggle 7	'invierte salida 7
pause 1000	'espera 1 segundo
goto inicio	'repite programa indefinidamente

wait

WAIT seconds

*Pausa de ejecución del código del programa de **seconds** segundos.*

- *Seconds: es una constante (0-65) que especifica la duración de la pausa en segundos.*

Ejemplo:

inicio:

switch on 7	'configura pin 7 como salida a nivel alto
wait 5	'espera 5 segundos
switch off 7	'configura pin 7 como salida a nivel bajo
wait 5	'espera 5 segundos
goto inicio	'repite programa indefinidamente

write

WRITE Address,value

Graba valores value en la EEPROM incorporada, en la dirección especificada por Address.

- *Address: es una variable/constante que especifica el byte de la dirección de memoria (0-255).*

- *Value: es una variable/constante que indica el valor del dato a escribir en la memoria.*

Al usar los módulos STAMP, el almacenamiento de datos es hacia arriba de 0 mientras el almacenamiento de programas se el construye hacia abajo de 255. Al usar el PICAXE-08 y 18, almacenamiento de datos se construye hacia arriba de la dirección 0 mientras el almacenamiento de programas se construye hacia debajo de la dirección 127. La posición de 255(127) de más alto menos está disponible para el usuario para el almacenamiento de datos.

Al programar el PICAXE-18A, 28, 28A, 28X esta orden es la que se trazó un mapa para la memoria FLASH separada (la posición de programa está separada) de datos. Por consiguiente comprobar el límite superior de la memoria de datos, como lo 28 y 28A tenga 64 bytes de memoria de datos (posiciones válido 0-63), el 28X tiene 128 bytes y el 18A, 18X tienen los bytes 256 completos.

La duración de la grabación es de 10 ms aproximadamente.

Ejemplo:

inicio:

```
for b0 = 0 to 63           'comienzo del bucle
  serin 6,T2400,b1        'recibe el dato via serie por pin 6 a 2400
                           'baudios y almacena en variable b1
  write b0,b1             'escribe el valor del dato en memoria
next b0
```

writemem

WRITEMEM Address,value

*Graba valores value en la EEPROM interna, en la dirección especificada por **Address**.*

- *Address: es una variable/constante que especifica el byte de la dirección de memoria (0-255).*
- *Value: es una variable/constante que indica el valor del dato a escribir en la memoria.*

Los PIC16F87X (incluidos los PICAXE-28, 28A) poseen un espacio adicional de almacenamiento de 256 bytes. Este comando no está disponible en el PICAXE-28X utilizando en su lugar una EEPROM externa utilizando la tecnología i2c.

Ejemplo:

inicio:

```
for b0 = 0 to 255        'comienzo bucle
  serin 6,T2400,b1        'recibe los datos vía serie
  writemem b0,b1         'escribe valor dato en b1
next b0                  'siguiente valor
```

writei2c

WRITEI2C Address,(variable,...)

*Graba valores value en la EEPROM interna, en la dirección especificada por **Address**.*

- *Address: es una variable/constante que especifica el byte de la dirección de memoria (0-255).*
- *Variable: es una variable que contiene el dato byte(s) para ser grabados.*

Esta orden se usa para escribir datos para el dispositivo i2c. Address define la dirección del inicio de los datos de escritura, cabe la posibilidad de escribir más de un byte secuencialmente (si el dispositivo del i2c soporta escritura secuencial – tener cuidado al usar EEPROMS que a menudo disponen de limitaciones de escritura).

Se debe tener en cuenta que la mayoría de las EEPROMS requieren unos 10ms para efectuar la escritura del dato, con lo cual se deberá tener en cuenta este retraso cuando se programe la ejecución del comando i2cwrite. Si no se tiene en cuenta este tiempo de retardo en la grabación los datos pueden corromperse.

Antes de usar este comando se debe haber utilizado previamente el comando **i2cslave**.

Si el hardware del i2c no esta correctamente configurado, o se han equivocado los datos del comando **i2cslave**, no se generará ningún error. Por ello se debe asegurarse que los datos han sido correctamente grabados por la orden **readi2c**.

Ejemplo:

‘Ejemplo de como usar el timer (reloj en tiempo real) DS1307
‘Los datos son enviados/recibidos en formato BCD
‘Se deben definir los symbol: seconds, mins, etc...(p.e. symbol seconds = b0)
‘La dirección determinada del esclavo del DS1307 i2cslave %11010000, i2cslow, i2cbyte

‘Escribir la hora y fecha con 11:59:00 y 25/12/05

Inicio:

```
let seconds =$00           '00 en formato BCD
let mins     =$59           '59 en formato BCD
let hour     =$11           '11 en formato BCD
let day      =$03           '03 en formato BCD
let date     =$25           '25 en formato BCD
let month    =$12           '12 en formato BCD
let year     =$03           '03 en formato BCD
let control  = %00010000    'habilitar salida a 1 Hz

writei2c 0,(seconds, mins, tour, day, dat, month, year,control)
end
```

El siguiente listado da un breve resumen de los diferentes comandos disponibles para los microcontroladores PICAXE.

Comandos PICAXE (válidos para todos los tamaños):

Salida	high, low, toggle, pulsout, let pins =
Sonido	sound
Entrada	if...then, readadc, pulsins, button
Serie	serin, serout
Flujo de Programa	goto, gosub, return, branch
Bucles	for...next
Matemáticas	let (+, -, *, **, /, //, max, min, &, , ^, &/, /, ^/)
Variables	if...then, random, lookdown, lookup
Memoria de datos	eeeprom, write, read
Tiempos de retardo	pause, wait, nap, sleep, end
Miscelaneos	symbol, debug

Comandos específicos del PICAXE-08:

Configuración Ent/Sal	input, output, reverse, let dirs =
PWM	pwm

Comandos específicos del PICAXE-18/18A:

RAM	peek, poke
Memoria de datos	writemem, readmem
Control de servo	servo (18A únicamente)
Infrarrojo	infrain (18A únicamente)
Interrupción	setint (18A únicamente)
Temperatura	readtemp (18A únicamente)
Número de serie de 1 cable	readownsn (18A únicamente)
Reloj de 1 cable	readowclk, resetowclk (18A únicamente)
Teclado	keyin, keyled (18A únicamente)

Comandos específicos del PICAXE-28/28A/28X:

RAM	peek, poke
Control de Servo	servo
Infrarrojo	infrain
Interrupción	setint (28A / 28X únicamente)
Temperatura	readtemp (28A / 28X únicamente)
Número de serie de 1 cable	readownsn (28X únicamente)
Reloj de 1 cable	readowclk, resetowclk (28X únicamente)
Teclado	keyin, keyled (28X únicamente)
I2C	i2cin, i2cout, i2cfamily (28X únicamente)

INSTRUCCIONES EN ENSAMBLADOR:

Nemónico	Operandos	Descripción
BYTE-ORIENTED FILE REGISTER OPERATIONS		
ADDWF	f,d	Add W and f
ANDWF	f,d	AND W and f
CLRF	f	Clear f
CLRWF	-	Clear W
COMF	d	Complement f
DECWF	f,d	Decrement f
DECFSZ	f,d	Decrement f, Skip if 0
INCF	f,d	Increment f
INCFSZ	f,d	Increment f, Skip if 0
IORWF	f,d	Inclusive OR with f
MOVF	f,d	Move f
MOVWF	f	Move W to f
MOVWF	f	Move f to W
NOP	-	No Operation
RLF	d	Rotate Left f through Carry
RRF	f,d	Rotate Right f through Carry
SUBWF	f,d	Substract W from f
SWAPF	f,d	Swap nibbles in f
XORWF	f,d	Exclusive OR W with f
BIT-ORIENTED FILE REGISTER OPERATIONS		
BCF	f,d	Bit Clear f
BSF	f,d	Bit Set f
BTFSC	f,d	Bit Test f, Skip if Clear
BTFSS	f,d	Bit Test f, Skip if Set
LITERAL AND CONTROL OPERATIONS		
ADDLW	k	Add literal and W
ANDLW	k	AND literal with W
CALL	k	Call Subroutine
CLRWDT	-	Clear Watchdog Timer
GOTO	k	Go to address
IORLW	k	Inclusive OR literal with W
MOVLW	k	Move literal to W
RETFIE	-	Return from interrupt
RETLW	k	Return with literal in W
RETURN	-	Return from Subroutine
SLEEP	-	Go into standby mode
SUBLW	k	Substract W from literal
XORLW	k	Exclusive OR literal with W

Donde:

- f= file register name
- k= value of literal
- b= bit number (0-7)
- d= result position (W or f)

CONTACTOS:

Software

La última versión del software de compilación/simulación/programación: *Programming Editor*
La podrás encontrar en :

www.picaxe.co.uk

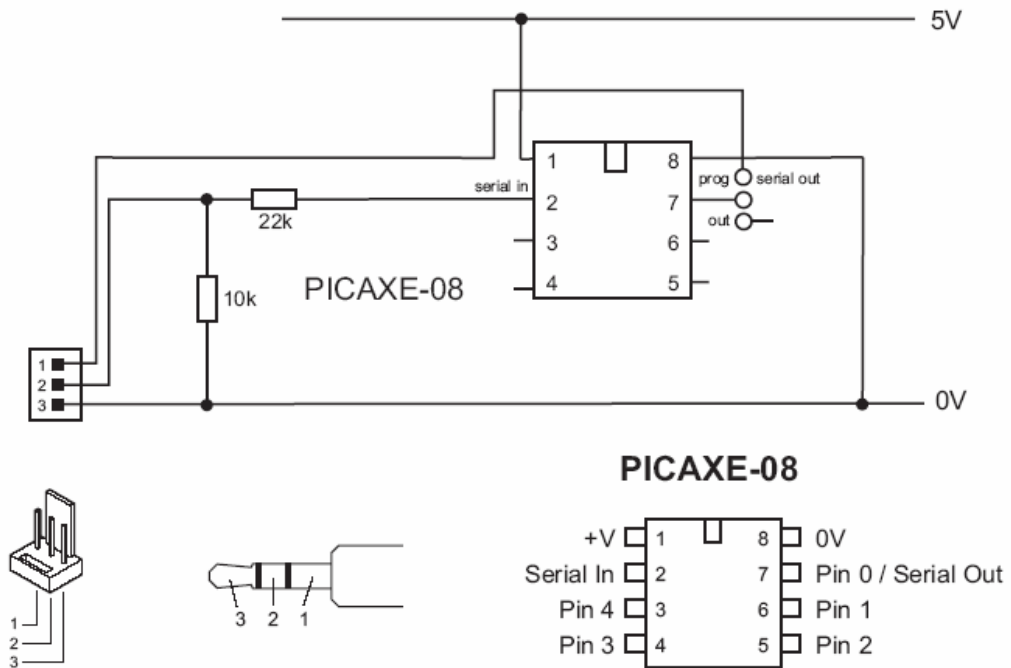
También encontrarás foros sobre proyectos diversos y soporte técnico.

Dirección de contacto

Revolución Education Ltd
4 Old Dairy Business Centre, Melcombe Road, Bath, BA2 3LR
<http://www.rev-ed.co.uk/>

ANEXO2: Esquemas electrónicos

Conexión típica de un PICAXE-08:



Aplicación PICAXE-08: Cyberpet:

Esquema

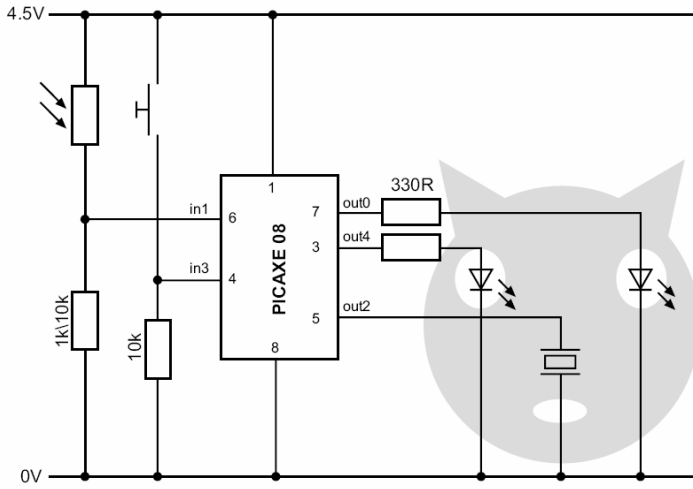
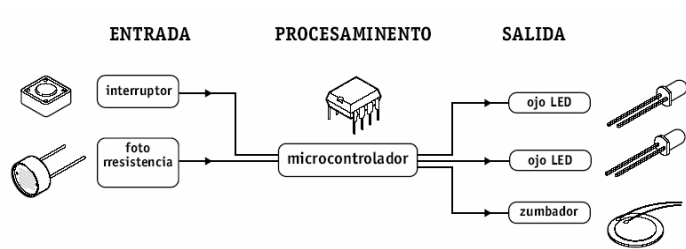
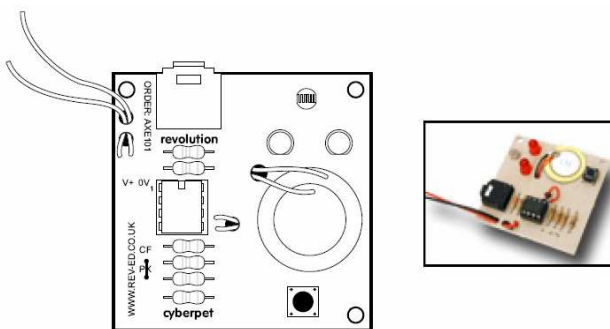


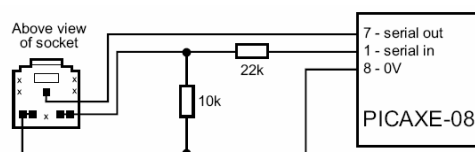
Diagrama de entradas/salidas:



Placa de Circuito Impreso

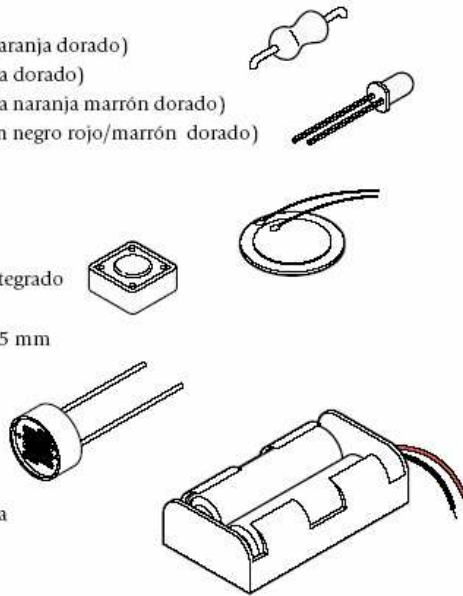


Conexión para la programación del Cyberpet



Para la construcción del circuito se necesitará:

- R1 y R2 resistencias de 10k(marrón negro naranja dorado)
- R3 resistencia de 22k (rojo rojo naranja dorado)
- R4 y R5 resistencias de 330R (naranja naranja marrón dorado)
- R6 resistencia de 1k/10k (marrón negro rojo/marrón dorado)
- LED1 y LED2 LEDs rojos de 5 mm
- PZ zumbador electrónico
- LDR fotorresistencia miniatura
- SW1 interruptor miniatura de 6 mm
- IC1 conector de 8 pines para circuito integrado
- PX microcontrolador PICAXE-08
- CT1 conector de descarga PICAXE de 3.5 mm
- BT1 clip de la batería
- BT1 caja de baterías de 4.5 V (3 x AA)
- PCB tablero de circuito impreso



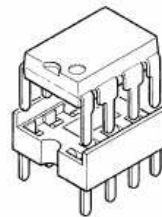
cables (en caso que esté conectando la fotorresistencia y los LEDs mediante cables)

Herramientas:
soldador y soldadura
alicate

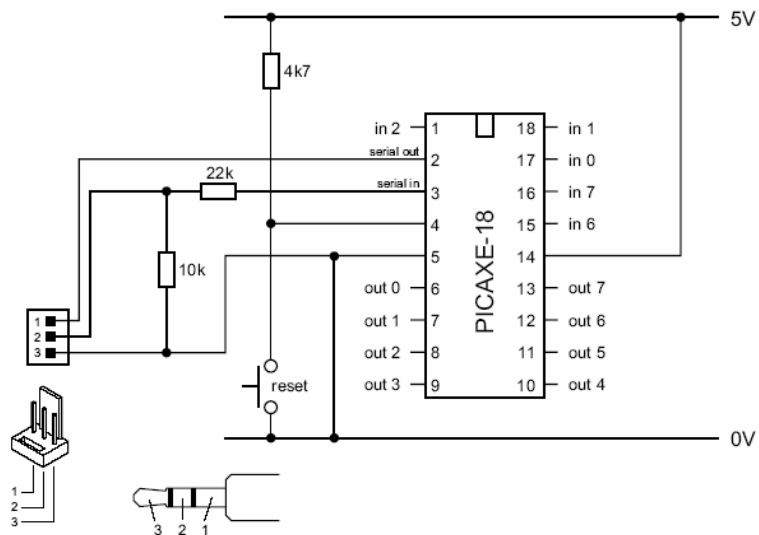
Codigo de Colours de las Resistencias			
Negro	0	0	Negro x1
Marron	1	1	Marron x10
Rojo	2	2	Rojo x100
Naranja	3	3	Naranja x1000
Amarillo	4	4	Amarillo x10,000
Verde	5	5	Verde x100,000
Azul	6	6	Azil x1,000,000
Violeta	7	7	
Gris	8	8	
Blanco	9	9	

			Plateado ±10%
			Dorado ±5%

Ejemplo:
azul, gris, marron, dorado
= 680R ±5%

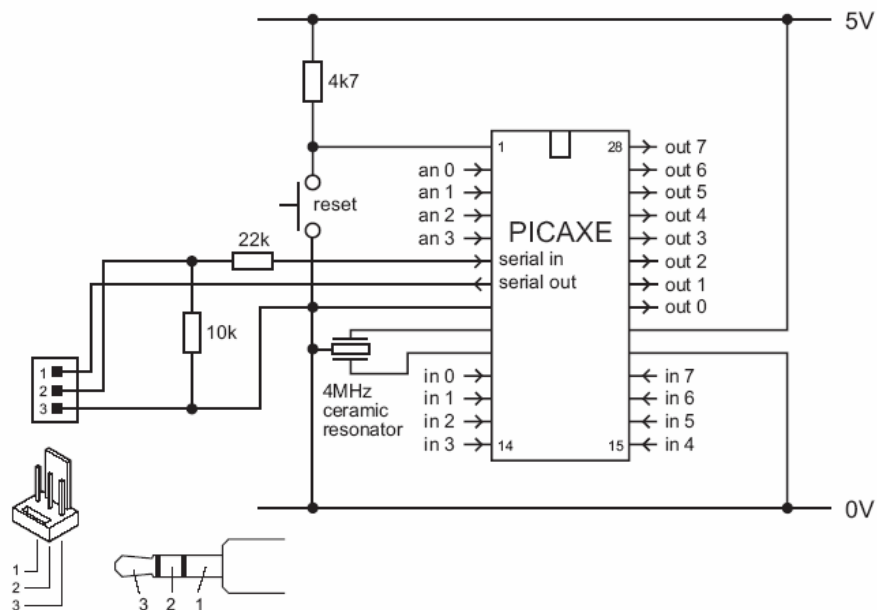


Conexión típica de un PICAXE-18:



The 4k7 resistor is used to pull the PICAXE microcontrollers reset pin (pin 4) high. If desired, a reset switch can also be connected between the reset pin (pin 4) and 0V. When the switch is pushed the PICAXE microcontroller 'resets' to the first line in the program.

Conexión típico de un PICAXE-28:



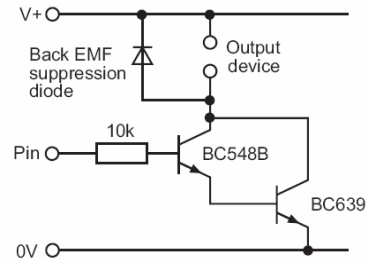
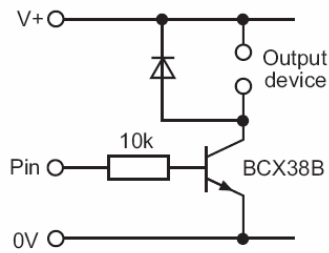
The 4MHz 3-pin ceramic resonator sets the 'clock speed' of the PICAXE microcontroller. To ensure correct timings it is essential that a 4MHz device is used. Each PICAXE microcontroller is supplied with a suitable 3 pin ceramic resonator. These devices are not polarised and so can be connected either way around.

The 4k7 resistor is used to pull the PICAXE microcontrollers reset pin (pin 1) high. If desired, a reset switch can also be connected between the reset pin (pin 1) and 0V. When the switch is pushed the PICAXE microcontroller 'resets' to the first line in the program.

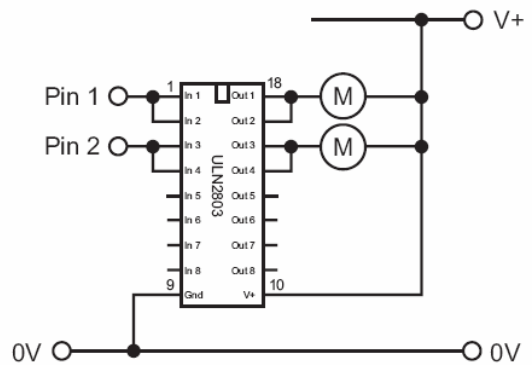
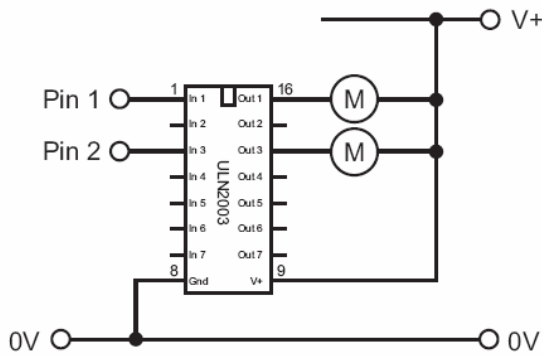
Note that the microcontroller has two 0V connections (pin 8 and pin 19). These pins are internally connected within the microcontroller.

Circuitos electrónicos de aplicación con PICAXE

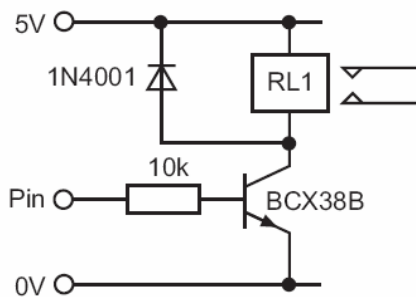
Interface típico de amplificación para dispositivo de salida:



Control de motores a través de un Darlington Driver en C.I.:



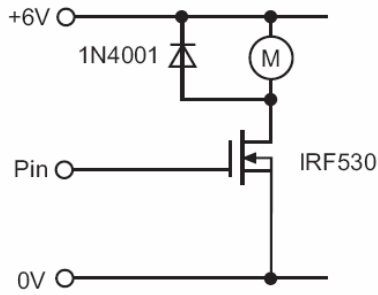
Control de salida con relé:



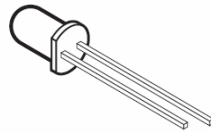
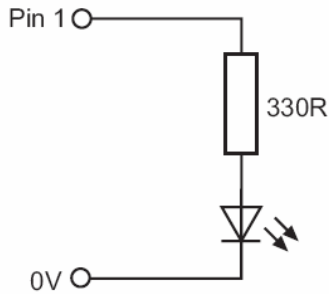
El transistor se elegirá en función de la corriente máxima de la bobina del relé.

Un BD135 (IC máx: 0,5A) es válido para la mayoría de los tipos de relé.

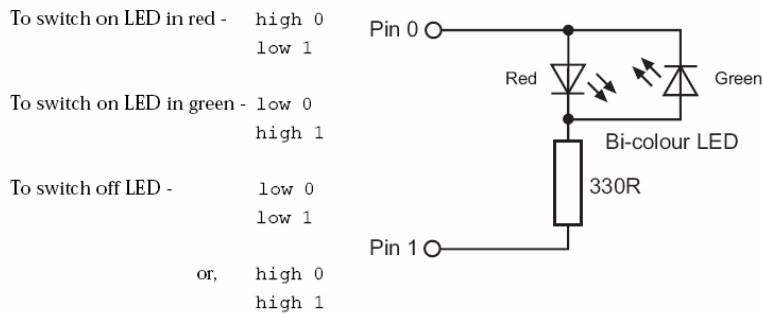
Control directo de un motor de consumo elevado con MOSFET:



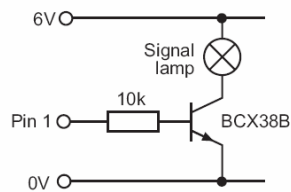
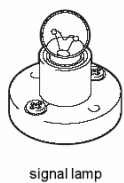
Control de salida para un LED:



Control de un sistema indicador Bicolor con LEDES (Rojo, Verde):



Control de una salida para una lámpara incandescente de baja tensión:

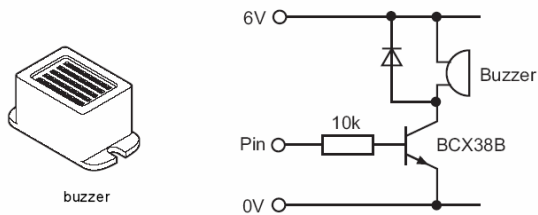


El transistor se elegirá en función de la corriente nominal de la lámpara.

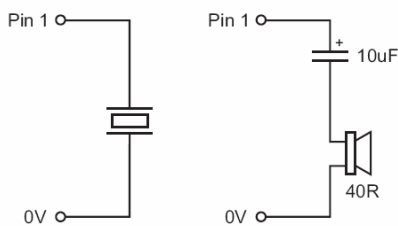
Un BD135 (IC máx: 0,5A) es válido para la mayoría de las lámparas de baja tensión.

Control de una salida para un zumbador:

Válido el versátil transistor: BD140



Control de una salida para un altavoz piezoeléctrico o electrodinámico de baja potencia:



To produce a note of pitch 100, length 50 on pin 1 -

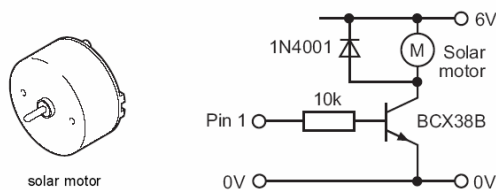
```
sound 1, (100,50)
```

To produce a varying noise using variable b1 -

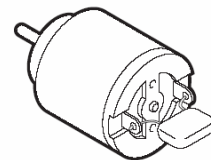
```
for b1 = 1 to 100
  sound 1, (b1,25)
next b1
```

Control de una salida para motor de bajo consumo (solar):

Válido el versátil transistor: BD140

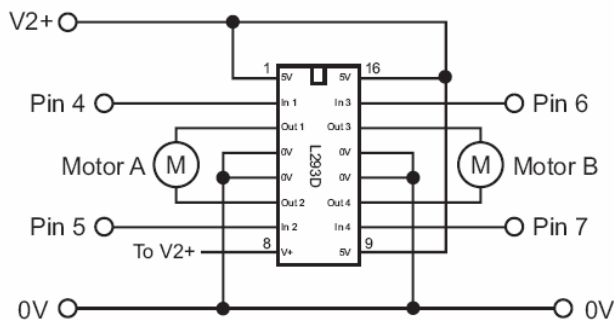


Para reducir los ruidos parásitos que producen la conmutación de las escobillas (ruidos el teléfono y barras en TV) se recomienda soldar en paralelo con los bornes de conexión un condensador cerámico de 220nF.

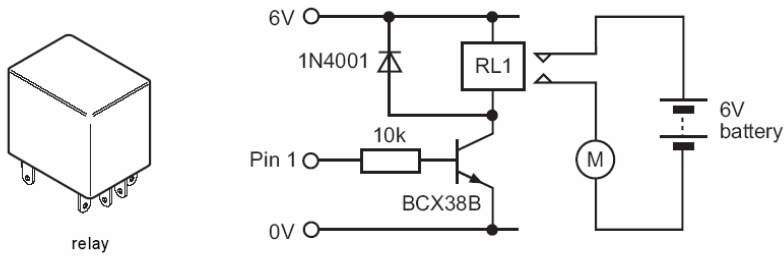


Control de dos motores con un Driver en C.I. "inteligente":

Para su correcto manejo consultar Datasheet.

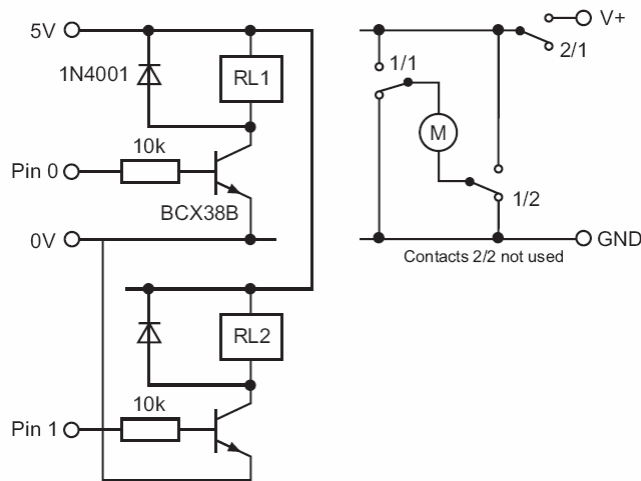


Control de un motor con tensiones diferentes de alimentación del circuito de control automatizado y control de potencia:

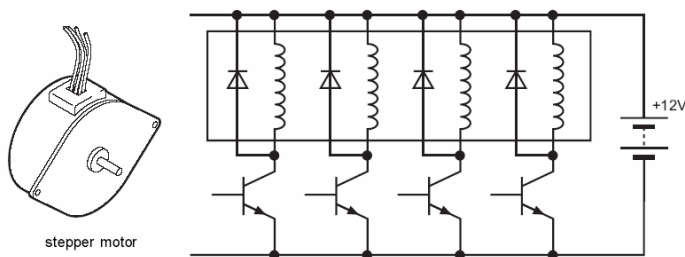


Este sistema aísla eléctricamente el circuito de control del circuito de potencia.

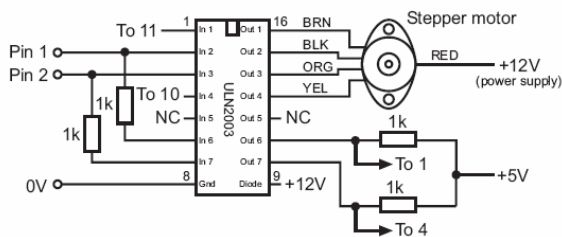
Control de la inversión de giro de un motor:



Control de un motor paso a paso (PAP) unipolar:



Step	Coil 1	Coil 2	Coil 3	Coil 4
1	1	0	1	0
2	1	0	0	1
3	0	1	0	1
4	0	1	1	0
1	1	0	1	0



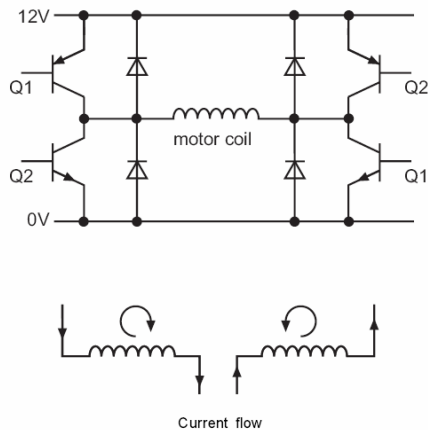
Step	Coil 1	Coil 3	Change
1	1	1	
2	1	0	coil 3
3	0	0	coil 1
4	0	1	coil 3
1	1	1	coil 1

N.B. colours of stepper motor leads may vary

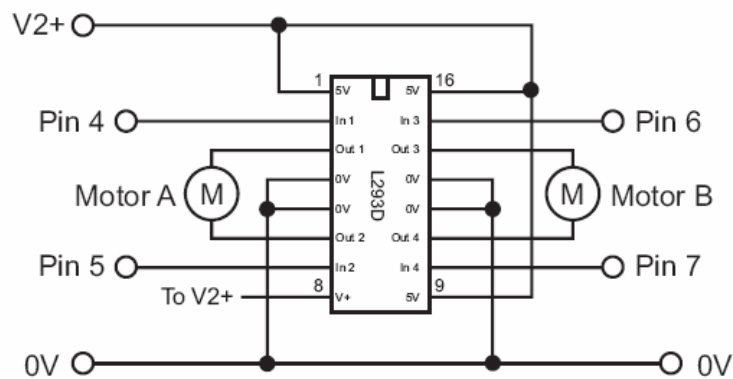
```

steps:  toggle 1           \ Toggle pin 1
        pause 200         \ Wait 200 ms
        toggle 2           \ Toggle pin 2
        pause 200         \ Wait 200ms
        goto steps        \ Loop
    
```

Control de un motor paso a paso (PAP) bipolar:



Step	Q1	Q2	Q3	Q4
1	1	0	1	0
2	1	0	0	1
3	0	1	0	1
4	0	1	1	0
1	1	0	1	0



```

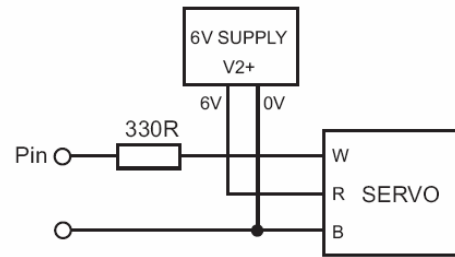
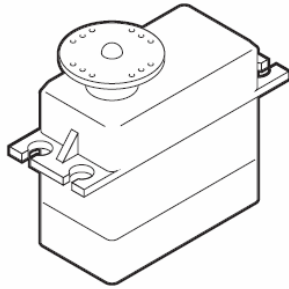
main:  for b3 = 0 to 99           \ start a for...next loop
        gosub lstep              \ call left step sub-procedure
    next b3                       \ next loop
    for b3 = 0 to 99             \ start a for...next loop
        gosub rstep              \ call left step sub-procedure
    next b3                       \ next loop

lstep: let b1 = b1 + 1           \ add 1 to variable b1
        goto step                 \ goto the lookup table

rstep: let b1 = b1 - 1           \ subtract 1 from variable b1

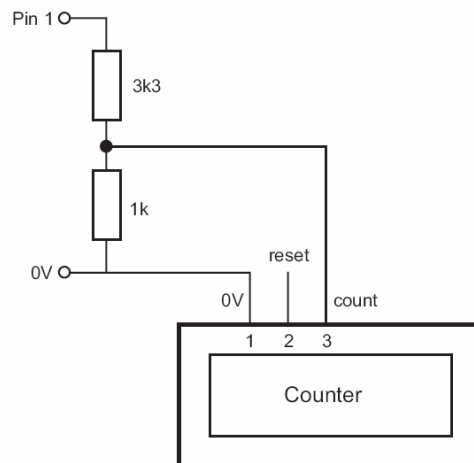
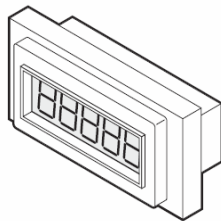
step:  let b1 = b1 & 2           \ mask lower two bits of b1
        lookup b1, (%1010,%1001,%0101,%0110),b2 \ lookup code into b2
        let pins = b2            \ output b2 onto control lines
        return
    
```

Control de un servomotor:



```
loop: servo 4,75      ` move servo to one end
  pause 2000         ` wait 2 seconds
  servo 4,150        ` move servo to centre
  pause 2000         ` wait 2 seconds
  servo 4,225        ` move servo to other end
  pause 2000         ` wait 2 seconds
  goto loop          ` loop back to start
```

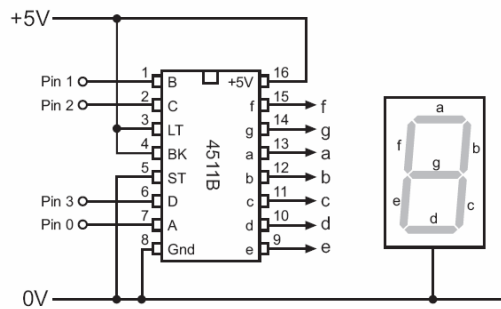
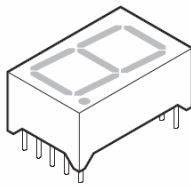
Control de un módulo de conteo:



Para incrementar conteo: Pulsout 1,100

Para resetear poner conectar una salida del PICAXE al pin 2 (Reset)

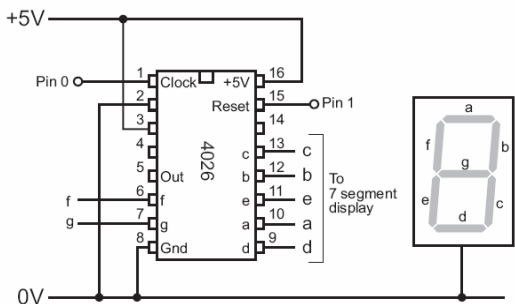
Control de un display:



This code example counts through the digits 0 to 9

```

main: for b1 = 0 to 9      \ Set up a for...next loop using variable b1
    let pins=b1           \ Output b1 onto the four data lines
    pause 1000            \ Pause 1 second
next b1                    \ Next
goto main                 \ Loop back to start
    
```



Con este C.I. tenemos la posibilidad de resetear el conteo a través del pin 1. Poniendo en estado alto el pin 1 en Display se pone a 0.

This code example uses sub-procedure 'clock' to display the digit '4', which is stored in the variable b1.

This is the sub-procedure

```

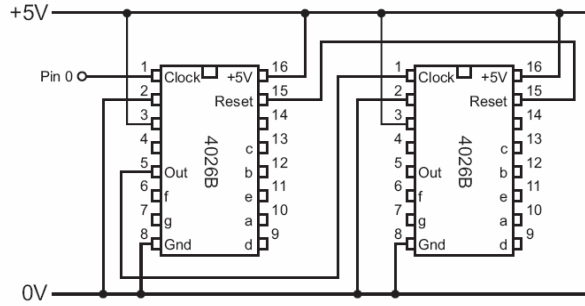
clock: pulsout 0,10      \ reset display to 0
    if b1 = 0 goto endclk \ if b1 = 0 then return
    for b3 = 1 to b1      \ start a for...next loop
        pulsout 1,10     \ pulse clock line
    next b3              \ next loop
endclk: return          \ return from sub-procedure
    
```

This is the main code

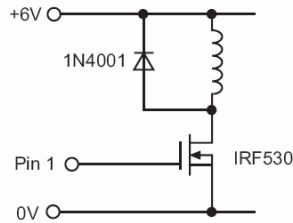
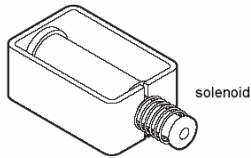
```

main: let b1 = 4         \ give variable b1 the value 4
    gosub clock          \ call sub-procedure
    pause 1000          \ wait 1 second
    goto main           \ loop
    
```

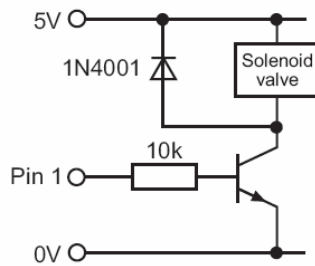
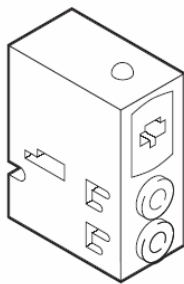
Control de dos Displays.



Control de un solenoide (cerraduras electrónicas):

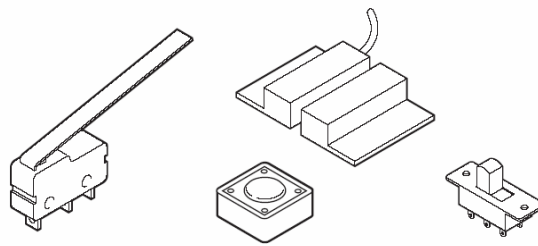


Control de electroválvula (sistemas electroneumáticos):

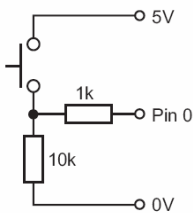


To switch the solenoid on - high 1
 To switch the solenoid off - low 1

Control de dispositivos de entrada (pulsador):

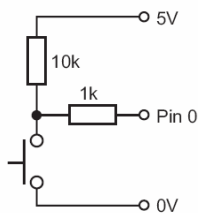


Salida "1" al accionar el pulsador



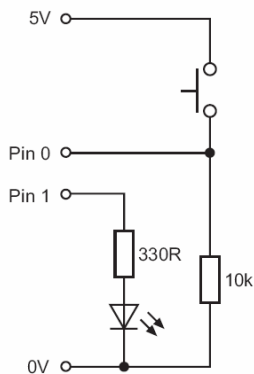
```
Goto 'jump' when switch is open:   if pin0 = 0 then jump
Goto 'jump' when switch is closed: if pin0 = 1 then jump
```

Salida "0" al accionar el pulsador



```
Goto 'jump' when switch is open:   if pin0 = 1 then jump
Goto 'jump' when switch is closed: if pin0 = 0 then jump
```

Control de dispositivos de entrada (pulsador con acción determinada):

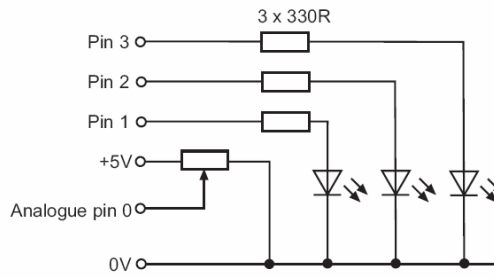
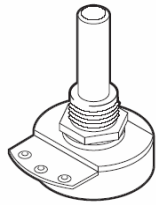


```
init: let b0 = 0
main: if pin 1 = 1 then add
      goto main
add:  let b0 = b0 + 1
      if b0 < 5 then main
      high 1
      goto main

init: let b0 = 0
main: if pin 1 = 1 then add
      goto main
add:  pause 100 'short delay here
      let b0 = b0 + 1
      if b0 < 5 then main
      high 1
      goto main
```

Indicador de acción después de la activación del pulsador según el resultado (ver código de ejemplo). Comprobar que al pulsar 5 veces el pulsador el LED se apaga.

Control de dispositivos de entrada (resistencia variable):



```
main:  readadc 0,b1      ' read value on pin0 into variable b1
       if b1<75 then light1  ' if b1 is less than 75 then light 1
       if b1<175 then light2 ' if b1 is less than 175 then light 2
       goto light3        ' if b1 is greater than 175 then light 3

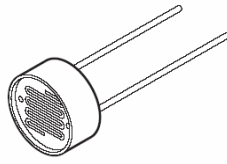
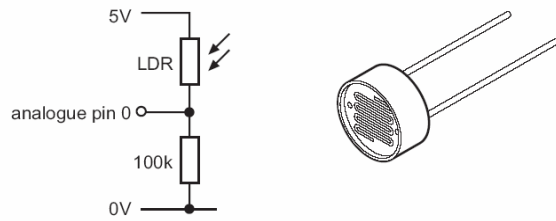
light1: high 1           ' switch on LED 1
       low 2             ' switch off LED 2
       low 3             ' switch off LED 3
       goto main        ' loop

light2: low 1           ' switch off LED 1
       high 2           ' switch on LED 2
       low 3            ' switch off LED 3
       goto main       ' loop

light3: low 1           ' switch off LED 1
       low 2            ' switch off LED 2
       high 3           ' switch on LED 3
       goto main       ' loop
```

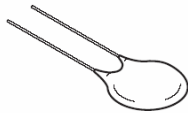
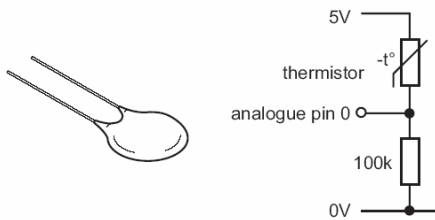
En función de la posición del potenciómetro se iluminarán los LEDS 1, 2 y 3.

Control de dispositivos de entrada (LDR):



```
main: readadc 0,b1          ` read the value
    if b1<50 then light1    ` range 0-50 = 50
    if b1<100 then light2  ` range 50-100 = 50
    if b1<145 then light3  ` range 100-145 = 45
    if b1<175 then light4  ` range 145-175 = 30
    goto main
```

Control de dispositivos de entrada (NTC):



```
main: readadc 0,b1          ` read the value
    if b1<50 then light1    ` range 0-50 = 50
    if b1<100 then light2  ` range 50-100 = 50
    if b1<145 then light3  ` range 100-145 = 45
    if b1<175 then light4  ` range 145-175 = 30
    goto main
```

Símbolos:

Switch (latching) 	Switch (non-latching) 																
Light dependent resistor 	Photodiode 																
Microphone 	Thermistor 																
Ammeter 	Voltmeter 																
Ceramic resonator 	Resistor 																
Variable resistor 	Potentiometer 																
Capacitor 	Capacitor (polarised) 																
Semiconductor diode 	Zener diode 																
Multiplication factors and symbols <table border="1"> <tr> <td>M</td> <td>mega</td> <td>1 000 000</td> <td>(10⁶)</td> </tr> <tr> <td>k</td> <td>kilo</td> <td>1000</td> <td>(10³)</td> </tr> <tr> <td>m</td> <td>milli</td> <td>0.001</td> <td>(10⁻³)</td> </tr> <tr> <td>μ</td> <td>micro</td> <td>0.000 001</td> <td>(10⁻⁶)</td> </tr> </table>		M	mega	1 000 000	(10 ⁶)	k	kilo	1000	(10 ³)	m	milli	0.001	(10 ⁻³)	μ	micro	0.000 001	(10 ⁻⁶)
M	mega	1 000 000	(10 ⁶)														
k	kilo	1000	(10 ³)														
m	milli	0.001	(10 ⁻³)														
μ	micro	0.000 001	(10 ⁻⁶)														

NPN transistor 	PNP transistor 																																																		
Field effect transistor (FET) 	Thyristor 																																																		
Operational amplifier 	Schmitt inverter 																																																		
AND gate 	NAND gate 																																																		
OR gate 	NOR gate 																																																		
NOT gate 	XOR gate 																																																		
Resistor colour codes <table border="1"> <tr> <td>Black</td> <td>0</td> <td>0</td> <td>Black x1</td> <td>Silver ±10%</td> </tr> <tr> <td>Brown</td> <td>1</td> <td>1</td> <td>Brown x10</td> <td>Gold ±5%</td> </tr> <tr> <td>Red</td> <td>2</td> <td>2</td> <td>Red x100</td> <td></td> </tr> <tr> <td>Orange</td> <td>3</td> <td>3</td> <td>Orange x1000</td> <td></td> </tr> <tr> <td>Yellow</td> <td>4</td> <td>4</td> <td>Yellow x10,000</td> <td></td> </tr> <tr> <td>Green</td> <td>5</td> <td>5</td> <td>Green x100,000</td> <td></td> </tr> <tr> <td>Blue</td> <td>6</td> <td>6</td> <td>Blue x1,000,000</td> <td></td> </tr> <tr> <td>Violet</td> <td>7</td> <td>7</td> <td></td> <td></td> </tr> <tr> <td>Grey</td> <td>8</td> <td>8</td> <td></td> <td></td> </tr> <tr> <td>White</td> <td>9</td> <td>9</td> <td></td> <td></td> </tr> </table> <p>Example shown: blue, grey, brown, gold = 680R ±5%</p>		Black	0	0	Black x1	Silver ±10%	Brown	1	1	Brown x10	Gold ±5%	Red	2	2	Red x100		Orange	3	3	Orange x1000		Yellow	4	4	Yellow x10,000		Green	5	5	Green x100,000		Blue	6	6	Blue x1,000,000		Violet	7	7			Grey	8	8			White	9	9		
Black	0	0	Black x1	Silver ±10%																																															
Brown	1	1	Brown x10	Gold ±5%																																															
Red	2	2	Red x100																																																
Orange	3	3	Orange x1000																																																
Yellow	4	4	Yellow x10,000																																																
Green	5	5	Green x100,000																																																
Blue	6	6	Blue x1,000,000																																																
Violet	7	7																																																	
Grey	8	8																																																	
White	9	9																																																	

tech supplies Electronic Components
Pictures and symbols

Buzzer 	Loudspeaker
Lamp 	Light emitting diode (LED)
Motor 	Solenoid
Relay 	Piezo sounder
Primary cell 	Battery
a.c. supply 	Transformer
Earth 	Fuse

Robots, games and parts for projects are available from:

www.tech-supplies.co.uk

Características de PICAXE:

Resumen de Microcontroladores PICAXE...

Tipo de PICAXE	CI	Memoria (lineas)	Pines	Salidas	Entradas	A/D (b = baja)	Memoria Data	Interrupt
PICAXE-08	8	40	5	1-4	1-4	1b	128-prog	-
PICAXE-18	18	40	13	8	5	3b	128-prog	-
PICAXE-18A	18	80	13	8	5	3	256	Si
PICAXE-28	28	80	20	8	8	4	64+256	-
PICAXE-28A	28	80	20	8	8	4	64+256	Si
PICAXE-28X	28	300	20	8	8	4	128+256	Si

PICAXE-28X available mid 2003

Notes:

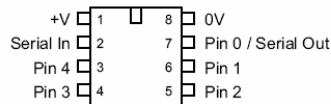
El número de líneas de memoria de programa es sólo un aproximado, ya que diferentes comandos requieren diferentes cantidades de memoria.

El PICAXE-28 tiene pines separados para el Conversor A/D, en todos los demás los Conversores A/D están combinados con los pines de entrada.

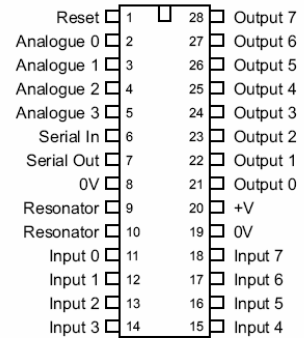
Los microcontroladores PICAXE-18 y PICAXE-28 requieren una resistencia de 4k7 en el pin de reinicio.

El microcontrolador PICAXE-28 requiere un resonadorexterno de cerámica de 4MHz y 3 pines.

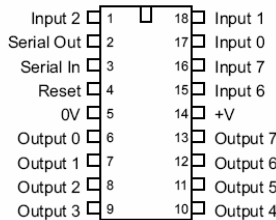
PICAXE-08



PICAXE-28



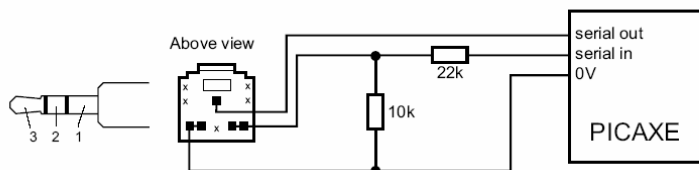
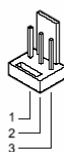
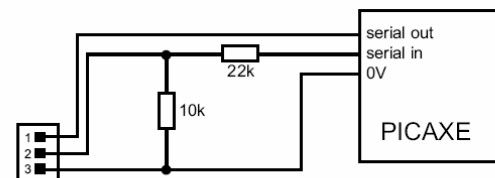
PICAXE-18



www.picaxe.co.uk
(c) Rev-Ed Ltd 2002

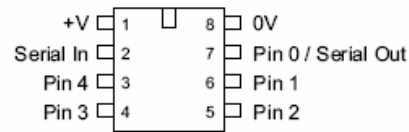
Circuito Serie de Descarga:

El circuito serie de descarga para todos los microcontroladores PICAXE es (conexiones estereo o en línea) el siguiente:



- Input = Entrada
- Output=Salida
- Serial In = Entrada Serie
- Serial Out = Salida Serie
- Resonator = Resonador
- Reset = Reiniciar
- Analogue = Analógica

Descripción de la función de cada pin en el PICAXE-08:



Terminal	Descripción	Notas
1	Alimentación positiva (+V)	De 3V – 5V. Admite (4x1,5V)
2	Entrada Serie	Usada para la descarga de programas
3	Pin 4	Entrada o Salida (E/S)
4	Pin 3	Entrada siempre (E)
5	Pin 2	Entrada o Salida (E/S)
6	Pin 1	Entrada o Salida (E/S) y entrada analógica
7	Pin 0	Salida siempre (S). Tambien se una para salida serie
8	Negativo (GND)	Conexión a la tensión de 0V

Tipo de PICAXE	Nº Pines	Memoria (líneas)	Pines	Salidas	Entradas	A/D (b-baja)	Memoria Datos	Interrupciones
PICAXE-08	8	40	5	1-4	1-4	1b	128-prog	-

ANEXO3: Introducción a la programación

1. INTRODUCCIÓN A LA PROGRAMACIÓN.

Los pasos a seguir a la hora de elaborar un programa deberían ser los siguientes:

- Hacer un estudio del problema: datos necesarios, posibles soluciones, errores...
- Realizar un análisis del proceso que seguiremos para solucionar el problema.
- Elaborar un pseudo-código, es decir, un programa en el que las órdenes estarían expresadas en nuestro propio idioma y no en un lenguaje de programación.
- Crear el código en un determinado lenguaje.
- Probar el programa y depurar los errores.

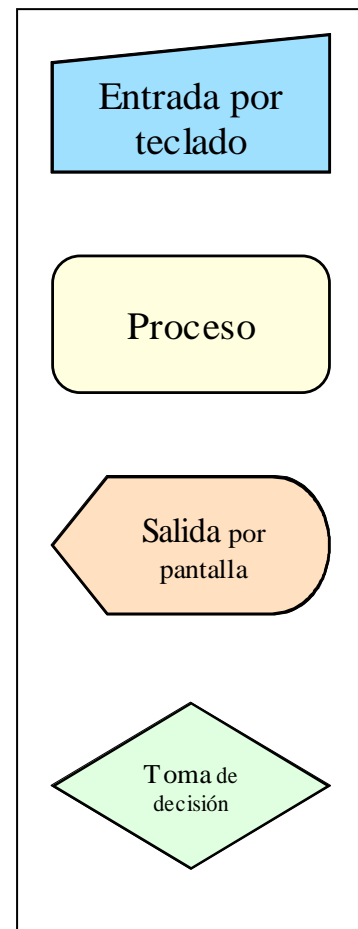
1.1. Análisis del proceso.

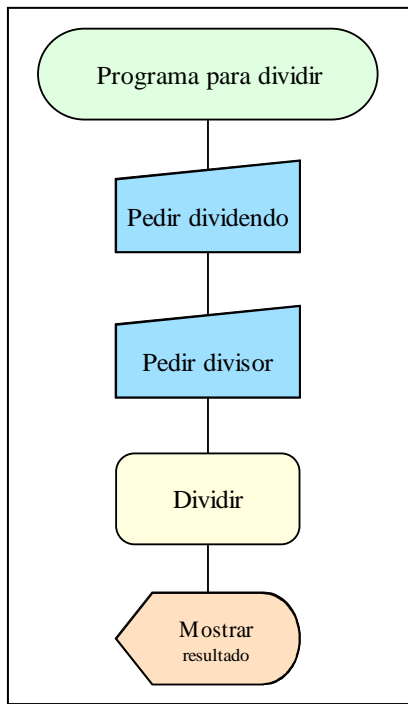
Antes de comenzar a escribir el código de nuestro programa (sea en el lenguaje que sea) lo primero que debemos realizar es un análisis del proceso que queremos realizar. Habitualmente este análisis se expresa en forma de organigrama empleando los símbolos que podemos ver en la figura de la derecha.

Supongamos que deseamos crear un programa que realice una división de dos números. Lógicamente los pasos que deben seguirse serían los siguientes:

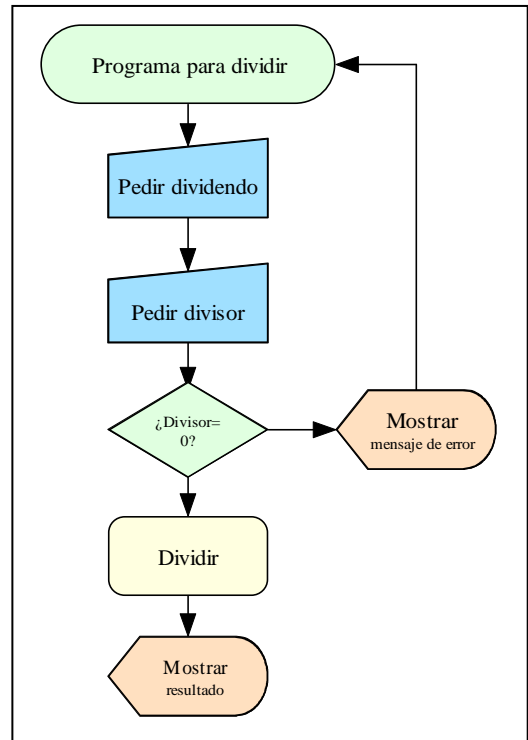
- Pedir el dividendo.
- Pedir el divisor.
- Efectuar la operación.
- Mostrar el resultado

Estos pasos mostrador en forma de organigrama podrían tener el siguiente aspecto:





Este diagrama de flujo tiene un pequeño inconveniente y es que sería posible que el usuario intentará dividir entre 0, con lo que se generaría un error. Podríamos evitarlo si planteamos el programa de la siguiente forma:



- **Actividad 1:** Dibuja el organigrama de un programa que resuelva ecuaciones de segundo grado con soluciones dentro de los números reales.

1.2. Pseudo-código.

El pseudo-código sirve para hacer un programa, no válido en ningún lenguaje particular pero que servirá para una posterior codificación en cualquiera de ellos. Conociendo el significado de cada uno de los símbolos empleados en los organigramas, crear el pseudo-código es algo sencillo. Partiendo del anterior organigrama nuestro primer programa podría ser algo así:

Pedir dividendo

Pedir divisor

Si divisor = 0 entonces

Mostrar ERROR

Ir a Pedir divisor

En otro caso

Calcular cociente

Calcular resto

Mostrar cociente

Mostrar resto

- **Actividad 2:** Escribe el pseudo-código de un programa que resuelva ecuaciones de segundo grado con soluciones dentro de los números reales.

1.3. Manipulación de la Información.

a) Tipos de datos: numéricos, caracteres y booleanos.

Cuando creamos un programa éste deberá trabajar con datos. La forma de manipular y tratar estos datos puede depender del tipo de lenguaje de programación que estemos empleando, pero en general todos suelen dividir los datos en tres grandes grupos: numéricos, caracteres y booleanos.

- **Numéricos:** Como su nombre indica hacen referencia a datos que representan un determinado valor numérico. Los tipos numéricos suelen dividirse en grupos según distintos criterios. Por una parte tenemos los tipos enteros y los tipos decimales. Los primeros se usan para trabajar con números que no tienen parte fraccionaria, mientras que los segundos son útiles en caso de que esa parte sí exista. Al tratar los números enteros hay que hacer otra división: con signo y sin signo. Los primeros siempre son positivos, mientras que los segundos son positivos y negativos.

- **Caracteres:** En los programas es habitual tratar con datos como el nombre de una persona, su dirección o su empresa. Éstos no son números sino sucesiones de caracteres.

En realidad los caracteres son tratados por el ordenador como combinaciones de bits, es decir como combinaciones de unos y ceros. La diferencia con un dato numérico está en que cuando el ordenador "sabe" que esa combinación de bits representa un carácter no la interpreta como un número sino que lo "consulta" una tabla para ver que carácter representa esa combinación de unos y ceros.

La tabla de caracteres más conocida es la llamada ASCII. En ella las combinaciones de bits equivalentes a los números 32 a 126 tienen asociado un carácter visible.

- **Booleanos:** Aparte de números y caracteres, prácticamente todos los lenguajes cuentan con otros tipos de datos capaces de representar de distintas maneras las combinaciones de bits que se almacenan en las celdillas de memoria. Uno de estos tipos es el booleano, que sólo puede contener dos valores posibles: "verdadero" y "falso".

Este tipo de dato se utiliza para representar datos que suelen tener dos estados. ¿Tienen permiso de conducir? ¿Está el interruptor abierto?.

- **Actividad 3:** Indica a qué tipo de datos de los vistos anteriormente pertenecerían los siguientes.

- DNI
- NIF
- Estado de un interruptor
- Velocidad del viento
- Provincia de residencia
- Valor de una resistencia
- Nota de un examen
- Nota de la evaluación
- Repetir curso

En el caso concreto de la programación con picaxe, no manejaremos datos de tipo carácter, sólo numéricos y boléanos.

1.4. Constantes y variables.

Cada celdilla de memoria de un ordenador tiene asignada una dirección, un número que sirve para hacer referencia al dato que contiene dicha celdilla. Aunque la mayoría de los lenguajes de programación disponen de los elementos necesarios para poder trabajar con direcciones de memoria, lo más usual es utilizar una representación simbólica, un nombre o identificador, ya que las personas trabajamos mucho mejor con palabras que con números.

En cualquier lenguaje de programación actual podemos asociar un identificador, a una celdilla o conjunto de celdillas de memoria, de tal forma que a partir de ese momento podemos usar dicho identificador en lugar de las direcciones correspondientes.

Realmente el enlace entre los identificadores simbólicos y las direcciones de memoria lo efectuaría el compilador o intérprete que utilizásemos, de tal manera que nosotros, como programadores, podríamos olvidarnos de las direcciones físicas de las celdillas en que están contenidos nuestros datos.

Al crear un identificador, para así poder acceder a una o más celdillas de memoria y trabajar con ellas, la mayoría de lenguajes nos permitirán declarar ese identificador como una constante o como una variable.

Una constante toma un valor inicial que ya no podrá modificarse durante la ejecución del programa. Un valor como el del número π podrá alojarse en la memoria utilizando una constante ya que no cambiará a lo largo del programa.

En contraposición, **una variable** es un identificador mediante el cual podrán efectuarse operaciones de lectura y escritura de las celdillas de memoria que representa. Si decidiésemos usar un identificador para poder alojar en una celdilla de memoria la temperatura, que vamos tomando mediante un equipo de medida, deberíamos usar una variable, ya que el valor de esta irá cambiando de cuando en cuando.

Algunos lenguajes de programación, como por ejemplo Pascal, exigen que declaremos todas las variables que vamos a emplear al principio del programa, mientras que otros permiten declarar variables en cualquier punto del programa como sucede en la variante de BASIC empleada para programación con PICAXE.

■ *Utilización de variables en programación de PICAXE.*

En el Basic empleado por los PICAXE se deben nombrar las variables empleando la letra "b" seguida de un valor numérico: b0, b1, b2... hasta b13.

Como puede este sistema hace difícil recordar el significado de cada variable podemos emplear el comando "symbol" al inicio de cada programa para asignar nombre distintos a las variables. Por ejemplo:

```
symbol contador = b0  
symbol estado = b1  
symbol entrada1 = b9  
symbol salida2 = b5
```

Para asignar un valor a una variable empleamos la instrucción "let" de la siguiente forma:

```
let nombrevariable = valor numérico
```

ejemplos:

```
let b0 = 3  
let contador = 5
```

También se puede modificar el valor de una variable mediante operadores matemáticos de la siguiente forma:

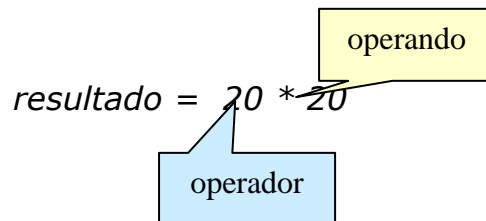
```
let contador = contador + 2
```

De esta forma incrementamos el valor de la variable "contador" en dos unidades cada vez que se ejecute esta instrucción.

1.5. Operandos y operadores

Para conseguir que un programa haga algo útil, aparte de almacenar y recuperar valores de la memoria también será necesario manipular esos valores con el fin de obtener resultados. Supón que tienes que calcular el total de una factura. Deberías tomar los importes parciales almacenados en varias celdillas de memoria y luego sumarlos para obtener un total.

Con el fin de manipular los datos o tomarlos como base para actuar de alguna manera se crean expresiones que pueden ser de distintos tipos: aritméticas, relacionales y lógicas principalmente. En dichas expresiones intervienen dos elementos fundamentalmente: los operandos y los operadores. El ejemplo más simple lo constituiría una multiplicación:



Si ejecutas esta instrucción obtendrás el resultado de la operación.

Como hemos dicho antes los operadores se agrupan básicamente en tres categorías: aritméticos, relacionales y lógicos.

Veamos algunos de los más habituales en cada una de estas categorías:

Aritméticos		Relacionales		Lógicos	
+	Suma	<	Menor	And	Si
-	Resta	>	Mayor	Or	O
*	Multiplicación	<>	Distinto	Not	Negación
/	División	=	Igualdad		

1.6. Control de flujo de un programa.

Actualmente todos los ordenadores funcionan en base a una arquitectura conocida como Neumann, llamada así en honor a su creador, el matemático Louis von Neumann.

Lo revolucionario de la idea de Neumann, propuesta en los años 40 cuando todavía no existían microprocesadores, está en la forma de instruir al ordenador acerca de lo que debe hacer. Hasta ese momento los ordenadores eran máquinas con un programa fijo encargado de realizar una tarea concreta, de tal manera que la única entrada por parte de los usuarios eran los datos que, tras ser procesados, generaban unos resultados. Es como funciona hoy en día una calculadora. La idea de von Neumann era radicalmente distinta: el programa ejecutado por el ordenador no sería un concepto fijo, sino que podría introducirse en su memoria igual que se introducían hasta ese momento los datos.

De esta forma los ordenadores comenzaron a utilizar la arquitectura von Neumann y se convirtieron en máquinas multipropósito.

■ Ejecución del código.

Cuando se pone en marcha la ejecución de un programa, el proceso que sigue el ordenador es, básicamente, el siguiente: se toma una instrucción, se procesa y ejecuta, se toma la siguiente instrucción y así sucesivamente.

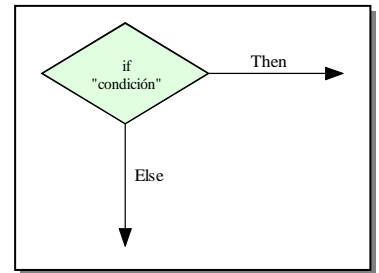
Si un programa se ejecutase siempre secuencialmente, de principio a fin, lo cierto es que sería complicado lograr algo más que resultados simples.

Para lograr que un programa pueda ser útil, este debe ser flexible, ejecutarse de distinta forma en función de algunas variables. Son necesarias dos estructuras básicas en programación: los condicionales y los bucles. A continuación aprenderemos algo más sobre ellos.

■ Estructuras condicionales

Todos los lenguajes de programación cuentan con una estructura condicional simple, con una salida afirmativa y otra negativa como la representada en la figura.

Planteamos una condición, si ésta se cumple (then) el programa ejecuta una serie de instrucciones, en caso contrario (else) el programa ejecuta otras distintas. En el caso del lenguaje Basic usado por los Picaxe no existe la cláusula else. Si la condición se cumple se ejecuta la acción indicada, en caso contrario se continúa con el programa.



Ejemplo de un programa en BASIC:

```
symbol pulsador = input3
```

inicio:

```
if pulsador is on then apagar  
if pulsador is off then encender
```

apagar:

```
low 4  
goto inicio
```

encender:

```
wait 2  
high 4  
goto inicio
```

- **Actividad 3:** El programa del ejemplo tiene un pequeño fallo. ¿Sabes cuál es?. ¿Cómo lo solucionarías?
- **Actividad 4:** Escribe un programa para controlar una luz en función de un pulsador. Si el pulsador está abierto la luz estará apagada, si el pulsador está cerrado la luz estará encendida. Al soltar el pulsador la luz permanecerá encendida durante 3 segundos más.

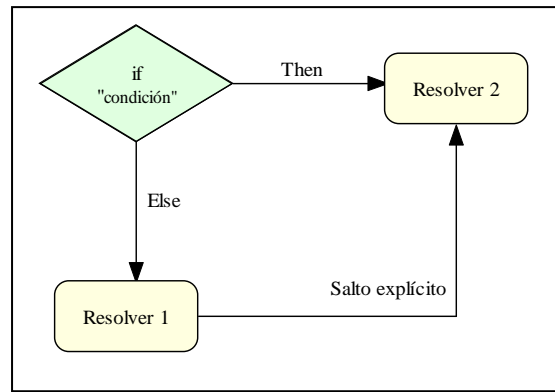
La condicional IF también se puede combinar con los operadores lógicos AND y OR para fijar más de una condición a la vez. Ejemplo:

```
if pin0 = 0 and pin1 = 0 and pin2 = 1 then enciende
```

- **Actividad 5:** Escribe un programa que permita emplear el PICAXE como una puerta lógica OR de dos entradas.
- **Actividad 6:** Escribe un programa que simula el funcionamiento de un timbre. Mientras accionemos el pulsador sonará un sonido de dos tonos diferentes. Al soltar el pulsador se detiene el sonido.

■ Bifurcaciones.

Tanto las condicionales como los bucles, según hemos podido ver, lo que hacen es desviar la ejecución del programa desde un punto hasta otro. Es lo que se conoce como una bifurcación en el flujo de ejecución de un programa. Los saltos, no obstante, están controlados por el condicional o el propio bucle, no siendo nosotros, de forma explícita, los que indicamos a dónde hay que pasar el control del programa.



Los saltos explícitos, de un punto a otro del programa, pueden utilizarse en situaciones donde sea necesario cambiar de un proceso a otro totalmente distinto.

En BASIC esto podemos hacerlo de una forma muy simple. Basta con escribir el nombre del procedimiento al que queremos saltar.

Ejemplo de un programa en BASIC:

output 0

principal:

```
let b2 = 0
gosub parpadea
wait 2
let b2 = 4
gosub parpadea
wait 2
goto principal
```

parpadea:

```
for b1 = 1 to 3
  high b2
  pause 500
  low b2
  pause 500
next b1
return
```

- **Actividad 7:** Escribe un programa que permita controlar un faro que debe emitir dos grupos de destellos 3 y 5 destellos separados 2 segundos entre si.
- **Actividad 8:** Escribe un programa para controlar la velocidad de parpadeo de un led en función del estado de un pulsador. Si el pulsador esta accionado parpadea deprisa, en caso contrario parpadea despacio.

■ Estructuras de repetición.

Gracias a las estructuras condicionales ya podemos controlar qué sentencias de nuestro programa son ejecutadas en cada caso, de tal manera que dicha ejecución no es completamente secuencial. Con una condición (If, Si) podemos saltarnos la ejecución de una o más sentencias, pero en realidad no nos sirve si precisamos la realización reiterada de una o más sentencias.

Supón que deseamos realizar un programa para que la tortuga avance y gire hasta que se cumpla una determinada condición. ¿Podemos hacerlo simplemente con una condicional?. La respuesta es no. Necesitamos una nueva estructura conocida como bucle.

Un bucle estará formado por una o más sentencias que deben repetirse, para lo cual se delimitan con las órdenes adecuadas para hacer posible la ejecución reiterada. Generalmente un bucle siempre tiene asociada una expresión condicional cuyo resultado "verdadero" o "falso", condiciona que el bucle se siga ejecutando o no.

El tipo de bucle más común, existente en todos los lenguajes de programación, tiene una sintaxis similar a esta:

*mientras condicional
sentencias*

La instrucción que debemos usar con nuestro PICAXE es la siguiente:

```
for nombrevariable = valor to valor step incremento  
    acciones a ejecutar  
next nombrevariable
```

Ejemplo:

```
symbol contador = b0
```

```
inicio:
```

```
    for contador = 1 to 10  
        high 4  
        wait 2  
        low 4  
        wait 3  
    next contador
```

```
end
```

En este ejemplo se repetirán 10 veces las instrucciones contenidas dentro del bucle.

- **Actividad 9:** Escribe un programa que genere 5 tonos que se incrementen secuencialmente mediante el altavoz piezoeléctrico conectado al pin 5 del PICAXE-08.
- **Actividad 10:** Modifica la **actividad 4** de forma que al soltar el pulsador la luz parpadee 5 veces antes de apagarse.