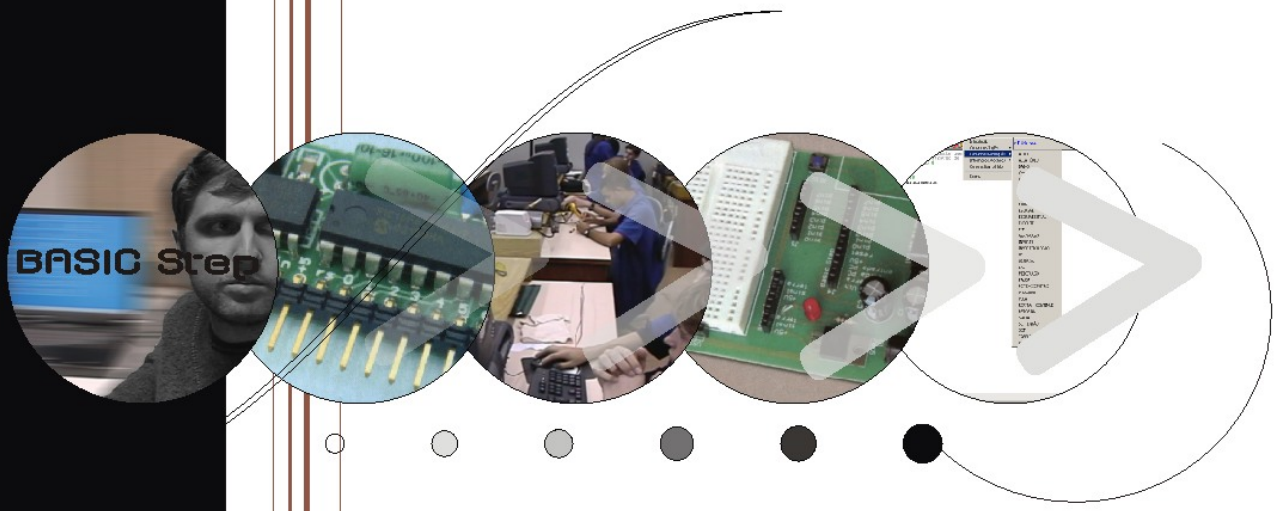
BASIC STEP



01010011100100011010011 Derli Luís Angnes

Introdução ao Microcontrolador BASIC Step

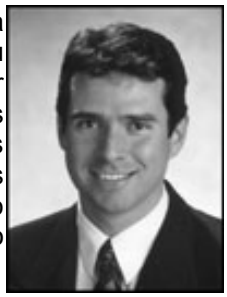
2003

Índice

Introdução 02
O que são microcontroladores e microprocessadores 03
Instruções e linguagem de programação 03
O BASIC Step 04
Compilador BASIC Step 05
Experimento 01: Conhecendo BASIC Step 06
Instruções do BASIC Step 07
Experimento 02 : Trabalhando com saídas 07
a) Sinalizador para saída de veículos 08
b) Seqüencial de 6 canais 09
c) Contador década 11
d) Contador década sem decodificador e com display anodo comum 11
e) Gerando tons musicais 12
f) Gerando tons musicais aleatoriamente 13
Experimento 03 : Trabalhando com entradas 13
a) Teclando tons musicais 13
b) Escolhendo um led 14
c) Conhecendo o comando potenciômetro 15
d) Temporizador de tempo ajustável 15
Experimento 04 : Trabalhando com display de cristal liquido 17
a) Inserindo mensagem no LCD 19
Exemplo de aplicação: Semáforo 19
Família BASIC Step 21
ANEXO – Instruções de comando do BASIC Step 22
1. Alto, liga, high 22
2. Aleatório, random 22
3. Baixo, desliga, low 23
4. Chave, button 23
5. Console, debug 25
6. Descança, nap 25
7. Dorme, sleep 26
8. Entrada, input 27
9. Escreve, write 28
10. Escreveserial, serout 28
11. Execute, gosub 31
12. Fim, end 31
13. Gerapulso, pulsout 32
14. Inverte, toggle 32
15. Invertedireção, reverse 32
16. Lê, read 33
17. Leserial, serin 33
18. Alto, liga, high 36
19. Medepulso, pulsim 36
20. Pausa, pause 37
21. Potenciômetro, pot 37
22. Procura, lookdown 38
23. Pula, branch 39
24. Repita-até-continue, for-to-next 40
25. Retorna, return 41
26. Saída, output 41
27. Se-então, if-then 42
28. Som, sound 43
29. Tabela, lookup 43
30. Vaipara, goto 44
31. Let 44

Introdução

BASIC Step é uma versão brasileira do famoso microcontrolador BASIC Stamp. O criador do BASIC Stamp tem uma história interessante:



"Em 1979 Chip Gracey (foto ao lado) teve sua primeira introdução à programação e eletrônica: o computador Apple II. Chip se interessou imediatamente na nova máquina, escreveu códigos em BASIC para mostrar gráficos e removeu a tampa do computador para ver os componentes eletrônicos. Esta experiência o levou rapidamente a examinar códigos fontes de videogames e outros aparelhos eletrônicos, a tentar usar estes aparelhos para outras finalidades. O hobby rapidamente se transformou em negócio quando ele passou a fabricar equipamentos para duplicar programas para o computador Comodore 64.

Os colégios não ofereciam cursos de hardware nem de software em 1982, e quando Chip se formou em 1986, a faculdade não parecia ser o melhor lugar para iniciar um novo negócio. Ao contrário ele e seu amigo Lance Walley fundaram a Parallax no seu apartamento. Os primeiros produtos foram digitalizadores de sons para o Aplle II e programadores para o 8051.

Os negócios cresceram lentamente até 1992 quando a Parallax lançou seu primeiro BASIC Stamp. A Parallax sabia que o BASIC Stamp seria especial – ele era a ferramenta que eles precisavam para os seus próprios projetos de hobby. O fato que o BASIC Stamp poderia criar a sua própria indústria, não era conhecido pelos fundadores da Parallax, mas rapidamente se tornou aparente que o pequeno computador tinha o seu grupo de entusiastas. Ele permitia a qualquer pessoa programar um microcontrolador, e dava a elas comandos poderosos de entrada e saída permitiam ligá-lo a outros componentes eletrônicos. Ao final de 1998 a Parallax tinha vendido 125.000 módulos BASIC Stamp e distribuía uma série completa de acessórios para ele através de mais de 40 distribuidores no mundo todo".

O preço em dólar do BASIC Stamp tornou inviável sua compra no Brasil. A Tato Equipamentos Eletrônicos criou o BASIC Step, a versão brasileira do famoso microcontrolador. O BASIC Step possui compilador próprio com instruções em português (TBASIC), além do tradicional PBASIC.

No curso, o aluno aprenderá a relacionar software e hardware, por isso é fundamental que o aluno conheça princípios de eletrônica analógica e digital para poder desenvolver todos os experimentos sem muita dificuldade.

Esse fascículo visa repassar noções de microcontrolador, a sua relação com o mundo e suas vantagens, em uma linguagem fácil e objetiva de forma extremamente agradável.

Grande parte dos exercícios pode ser implementada com uma placa de prototipagem e alguns componentes eletrônicos adicionais.

Bem Vindo ao Mundo dos Microcontroladores.

Derli Luís Angnes

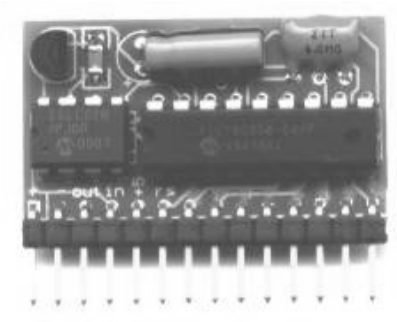
Santa Cruz do Sul/RS
angnes@zaz.com.br

O que são microcontroladores e microprocessadores

Ambos são computadores digitais que realizam operações em seqüência sem intervenção humana. As operações são programadas por um programador, que segue uma lista de instruções que compõe a linguagem de programação (Assembly, C, Java).

Os **microcontroladores** ou **µC** são conhecidos como computadores embutidos em circuito-integrado. Em um microcontrolador podemos encontrar memória, CPU, entradas e saídas. Alguns ainda possuem periféricos como conversores A/D e D/A, comparadores. Os microcontroladores chegam a custar muitas vezes mais barato do que um transistor. Existem uma quantidade grande de µC no mercado, veja alguns nomes a baixo:

- **Família 8051** – fabricante Intel
- **PIC** – fabricante Microchip
- **AVR** – fabricante Atmel
- **BASIC Stamp** – fabricante Parallax
- **BASIC Step** – fabricante Tato Equipamentos



BasicStep

Os µC estão presentes em agendas eletrônicas, telefones celulares, alarmes, CLP's, veículos, caixas eletrônicos, impressoras...

O **microprocessador** é um circuito integrado que possui uma poderosa CPU (Unidade Central de Processamento). É o microprocessador que processa as informações de uma memória ou de um periférico. Não é possível obter um computador somente com um microprocessador. Um exemplo clássico é o computador pessoal que agrega no interior do gabinete diversos componentes, dentre eles o microprocessador. Os microprocessadores são requisitados sempre que houver a necessidade de se processar inúmeros cálculos em um curtíssimo espaço de tempo. Exemplo de processadores: Z80, 8080, 80386, 80486...

Instruções e linguagem de programação

Programação é a seqüência de operações que o sistema deve executar para que a tarefa determinada seja realizada. Cada operação correspondente a uma instrução que pode ser interpretada e executada pelo computador. As instruções são constituídas por uma série de bits. Esses bits são decodificados e acionam as variáveis de controle internas ao sistema para que a operação correspondente à instrução seja realizada.

Cada microcontrolador adota uma linguagem de programação. As linguagens mais difundidas para microcontroladores são: Assembly e C. Todo programador deve conhecer o Set de Instruções de seu microcontrolador para poder realizar a programação do chip.

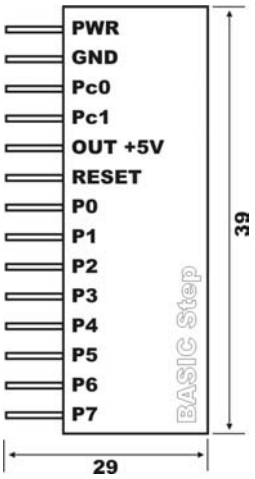
O código fonte do programa, ou seja, a escrita do programa precisa passar por um compilador, o compilador irá traduzir cada linha digitada em linguagem de máquina, para que o microcontrolador possa ser gravado. A gravação requer muitas vezes um equipamento de gravação. No mercado existem gravadores dedicados e universais que permitem a gravação de inúmeros chips's. O nosso BASIC Step só precisa de um cabo para ser programado.

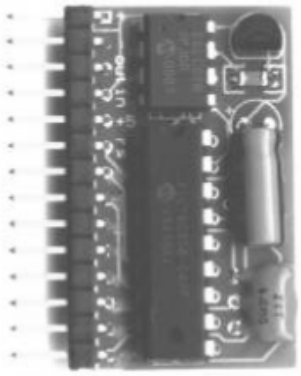
O BASIC Step

BASIC Step I é um microcontrolador extremamente fácil de utilizar, com comandos em português e inglês. Alimentação de 7,5V a 15V. Possui 8 entradas e saídas com capacidade de corrente de 25mA.Tamanho: 3,5 cm x 3,0 cm.

Hardware:

O Basic Step trabalha com um microcontrolador PIC16F628 e numa única placa engloba esse microcontrolador, uma memória com o interpretador PBASIC e um regulador de tensão. Pode ser alimentado com tensão de 7,5 a 15v. Possui 8 portas configuráveis para entrada ou saída. É programável diretamente pela serial do computador, por um cabo simples, por isso, não precisa de programadores caros ou difíceis de construir. Tudo isso numa minúscula placa. Outra vantagem é que ele se encaixa perfeitamente na matriz de contatos, simplificando os testes. Veja a pinagem do BASIC Step:

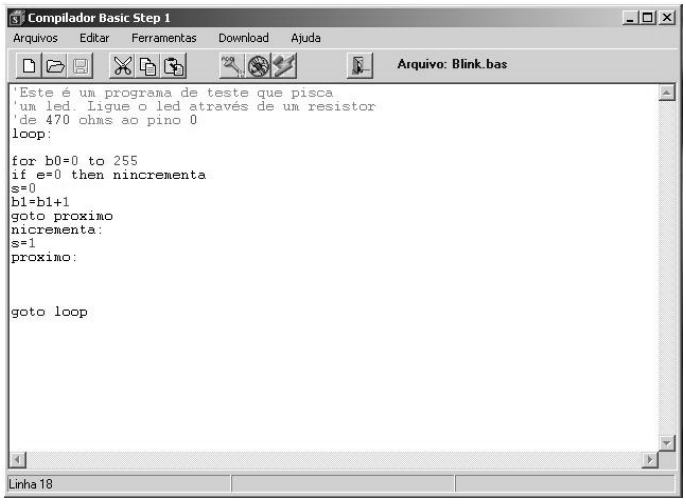




Pino	Função
PWR	Alimentação +7 a +15V
GND	0V - Terra
Pc0	Saída de sinais de progr.
Pc1	Entrada de sinais de progr.
OUT +5V	Saída de tensão +5V
RESET	Reseta com nível baixo
P0	I/O
P1	I/O
P2	I/O
P3	I/O
P4	I/O
P5	I/O
P6	I/O
P7	I/O

Software:

O BASIC Step é programado pela linguagem Tbasic ou Pbasic. Ambas são muito fáceis de utilizar, sendo que a diferença está no idioma: Pbasic é inglês e o Tbasic é português.



Ao lado o compilador BASIC Step.

O compilador permite edição de texto do código fonte, possui compilador, debug para procurar erros de digitação, download para transferência do programa ao microcontrolador e ajuda para consulta de instruções.

O software está na verão 0.9.22 e pode ser executado no Windows 95, Windows 98, Windows Me e Windows XP.

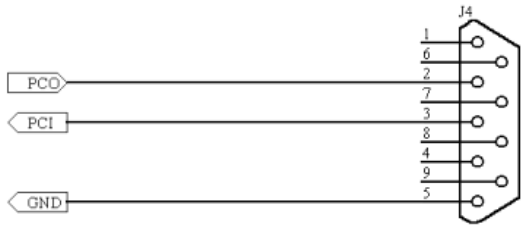
Programação:

Para programar o BASIC Step são necessários:

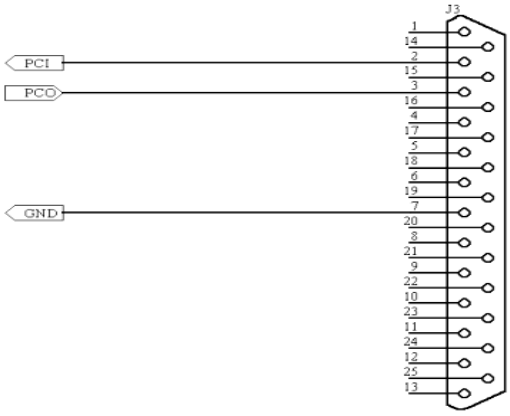
- Microcontrolador BASIC Step;
- Fonte CC para alimentação do µC;
- Compilador BASIC Step;
- Computador;
- Cabo de gravação;

- Opcionalmente uma placa de prototipagem StepLab.

O Basic Step utiliza a comunicação serial do PC para gravação. O cabo utilizado é simples e a conexão para DB9 e DB25 estão indicados abaixo:



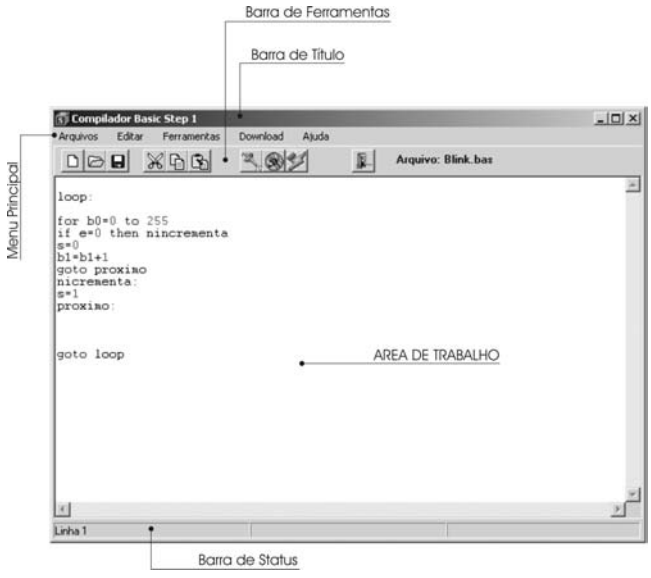
Cabo de gravação do BASIC Step com o PC através do conector DB9.



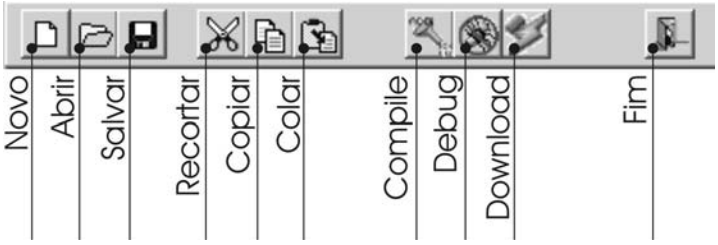
Cabo de gravação do BASIC Step com o PC através do conector DB25.

Compilador BASIC Step

Para instalar o compilador será necessário ter um computador: 100Mhz com 8MB de memória e 2MB de espaço no disco com um driver de disket. O compilador utiliza sistema operacional Windows 95 ou superior. Veja a tela principal:



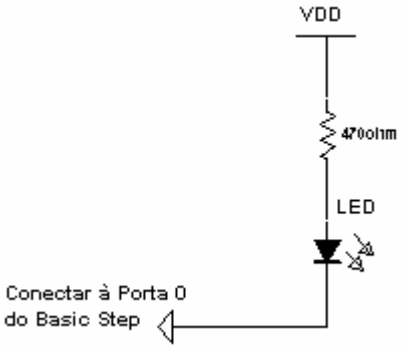
Barra de ferramentas:



Experimento 01: Conhecendo BASIC Step

Vamos conhecer como o BASIC Step trabalha, para isso é necessário escrever um programa. A princípio não vamos comentar as funções de cada instrução, pois o objetivo é compreender e se familiarizar com o modo de trabalho de gravação de um microcontrolador. Vamos lá: Conecte o cabo de gravação da placa StepLab à porta serial do seu computador. Monte agora o hardware e o software. O BASIC Step, o compilador e a placa Steplab podem ser adquiridos na home page do fabricante: www.tato.ind.br.

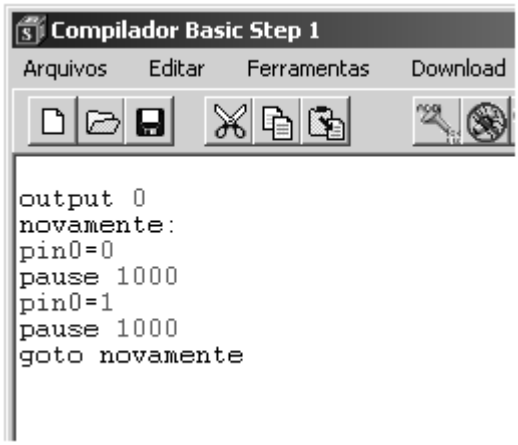
Hardware:



Monte o circuito ao lado em uma matriz de contatos ou na placa StepLab. Confira as ligações. Evite curto-circuito, pois um erro pode ocasionar a perda de um pino de I/O.

O VDD é + 5VCC.

Software:



Digite o mesmo texto da figura em seu compilador.

Depois de digitado, clique com mouse sobre o ícone "compile" para transformar seu programa em linguagem de máquina.

Caso surja uma mensagem de erro, após o clique no ícone compile, dê ok na tela de mensagem e revise todo o texto. O compilador está lhe dizendo que existe um erro no código fonte, um "Bug". Por isso "Debugar" um programa é fundamental antes da compilação final.

	Agora pressione novamente o ícone compile, ele irá compilar o programa.
	Agora faça o download para o µC. Verifique o cabo de gravação.
	Aguarde, o compilador está estabelecendo comunicação com o µC.

Veja o resultado! Caso o led não tenha piscado, revise o hardware e seu software.

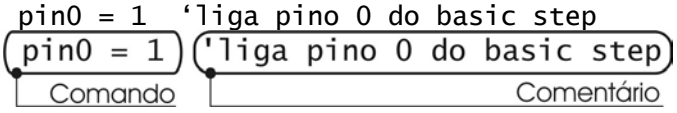
Instruções de comando do BASIC Step

Vamos utilizar a linguagem Pbasic para desenvolver nossos programas, o compilador também suporta a linguagem Tbasic, desenvolvida pelo fabricante. A linguagem Pbasic é muito semelhante à utilizada por outros microcontroladores: BASIC Stamp, 8052-Basic... Por essa razão, adotaremos o Pbasic como linguagem principal nos experimentos do curso.

Tabela de Instruções do BASIC Step			
Instruções em Português - Tbasic		Instruções em Inglês - Pbasic	
ALTO	LE	BRANCH	OUTPUT
ALEATÓRIO	LESERIAL	BUTTON	PAUSE
BAIXO	LIGA	DEBUG	POT
CHAVE	MEDEPULSO	EEPROM	PULSIN
CONSOLE	PAUSA	END	PULSAUT
DESCANÇA	POTENCIOMETRO	FOR...NEXT	PWM
DESLIGA	PROCURA	GOSUB	RANDOM
DORME	PULA	GOTO	READ
ENTRADA	REPITA...CONTINUE	HIGH	RETURN
ESCREVE	RETORNA	IF...THEN	REVERSE
ESCREVESERIAL	SAIDA	INPUT	SERIN
EXECUTE	SE...ENTÃO	LET	SEROUT
FIM	SOM	LOOKDOWN	SLEEP
GERAPULSO	TABELA	LOOKUP	SOUND
INVERTE	VAIPARA	LOW	TOGGLE
INVERTEDIREÇÃO		NAP	WRITE

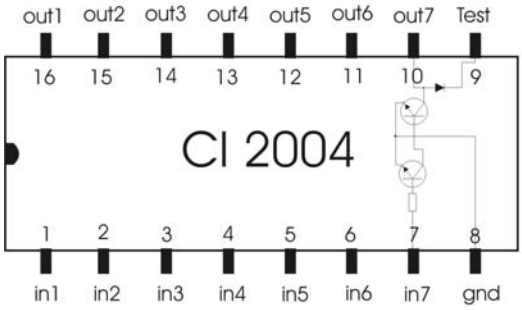
São 32 instruções a disposição do programador, em nosso curso não veremos todas por não haver necessidade. Como vemos na tabela acima, percebemos que o compilador suporta duas linguagens: o Tbasic e o Pbasic. Disponibilizamos todas as instruções no anexo. Os comentários podem ser escritos após o símbolo " ' ". Um comentário não é considerado pelo compilador, ele deve ser utilizado para comentar uma linha de programação. Exemplo:

```
pin0 = 1  'liga pino 0 do basic step
```



Experimento 02 : Trabalhando com saídas

Os pinos do BASIC Step quando estiverem trabalhando como saída drenam no máximo 25mA em sua saída, para cargas que necessitem de uma capacidade maior de corrente há a necessidade de ser adicionado um amplificador de corrente na saída (transistor, driver, tiristor, relé, contactora).

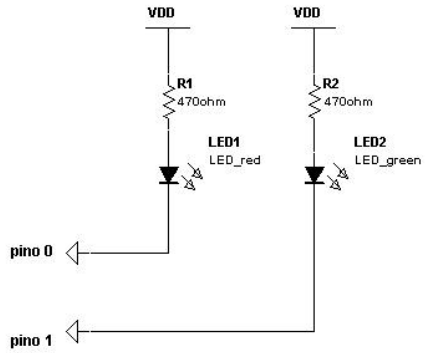


Ao lado um driver – 2004. O C.I. possui 7 transistores darlington NPN.
O C.I. 2004 pode ser utilizado com o BASIC Step, sempre que houver necessidade de elevação da corrente dos pinos. Não esqueça que o 2004 inverte o sinal (lógica invertida).
Para acionar cargas de baixa potência, como diodos emissores de luz, não se faz necessário utilizar o driver. Desde que não ultrapasse 25mA por pino.

a) Sinalizador para saída de veículos:

Vamos criar nosso segundo programa, um sinalizador de veículos:

Hardware:

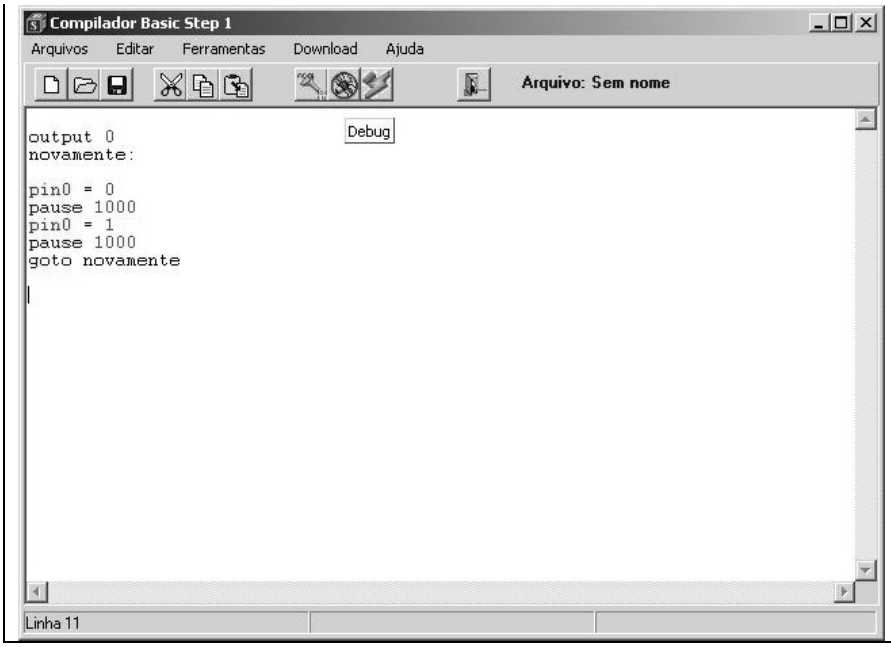


Monte o circuito ao lado na placa StepLab. Utilizando o 2004 e um relé poderíamos utilizar este circuito próximo a uma garagem.

Software:

```
'************************************
'*PROGRAMA: Sinalizador de veículo *
'*PROGRAMADOR: Derli L. Angnes     *
'*DATA: 02/11/02                   *
'*VERSÃO: 1.0                      *
'************************************

output 0        'o comando torna pino 0 uma saída
novamente:      'novamente: é um label, uma posição de memória

pin0 = 0        'estamos atribuindo a saída 0 o binário 0 - o led liga
pause 1000      'pausa a saída por 1000ms = 1seg.
pin0 = 1        'estamos atribuindo a saída 0 o binário 1 - led desl.
pause 1000      'pausa a saída por 1000ms = 1seg.
goto novamente 'goto manda saltar para label novamente
```

O resultado no compilador deve ficar igual à tela a baixo:



Agora revise o programa e corrija erros se houver.

Salve o programa no disket.

Depois compile o programa.

Por último, transfira o programa para o BASIC Step, clicando no ícone Download.

Aguarde a transferência!

Se seu programa estiver funcionando você deve ter observado que somente um led está oscilando, é necessário terminar o programa. Acrescente os comandos para fazer o segundo led piscar.
Uma outra forma de escrever o programa anterior é utilizando os comandos “high” e “low”. Veja:

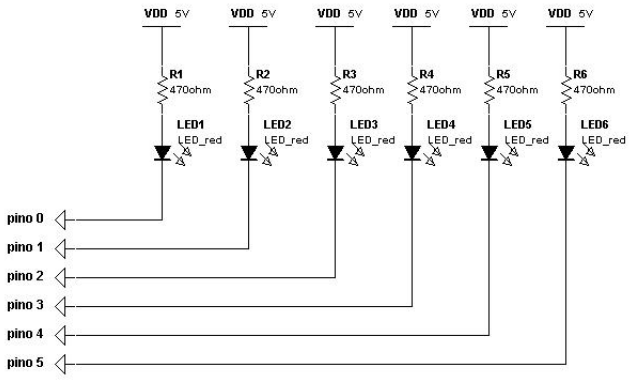
Software:

```
'************************************
'*PROGRAMA: Sinalizador de veículo *
'*PROGRAMADOR: Derli L. Angnes     *
'*DATA: 02/11/02                   *
'*VERSÃO: 2.0                      *
'************************************

novamente:     'novamente: é um label, uma posição

low 0          'pino 0 está em nível baixo
pause 1000     'pausa a saída por 1000ms = 1seg.
high 0         'pino 0 está em nível alto
pause 1000     'pausa a saída por 1000ms = 1seg.
goto novamente 'goto manda saltar para label novamente,entra em ciclo
```

Os comandos low e high atribuem nível lógico às saídas e ao mesmo tempo configura o pino declarado como uma saída. Para conhecer detalhadamente todos os comandos veja o anexo.

b) Seqüencial de 6 canais

Hardware:



Monte o circuito ao lado na placa StepLab. Vamos construir um seqüencial de 6 canais, lembra do C.I. 4017...

Software:

```
'************************************
'*PROGRAMA: Seqüência de 6 canais  *
'*PROGRAMADOR: Derli L. Angnes     *
'*DATA: 02/11/02                   *
'*VERSÃO: 1.0                      *
'************************************

REPITA:              'label

LOW 0                'põe nível baixo na porta 0
HIGH 0               'põe nível alto na porta 0
PAUSE 100            'aguarda 100ms
LOW 0                'põe nível baixo na porta 0

LOW 1                'põe nível baixo na porta 1
HIGH 1               'põe nível alto na porta 1
PAUSE 100            'aguarda 100ms
LOW 1                'põe nível baixo na porta 1

LOW 2                'põe nível baixo na porta 2
```

```
HIGH 2              'põe nível alto na porta 2
PAUSE 100           'aguarda 100ms
LOW 2               'põe nível baixo na porta 2

LOW 3               'põe nível baixo na porta 3
HIGH 3              'põe nível alto na porta 3
PAUSE 100           'aguarda 100ms
LOW 3               'põe nível baixo na porta 3

LOW 4               'põe nível baixo na porta 4
HIGH 4              'põe nível alto na porta 4
PAUSE 100           'aguarda 100ms
LOW 4               'põe nível baixo na porta 4

LOW 5               'põe nível baixo na porta 5
HIGH 5              'põe nível alto na porta 5
PAUSE 100           'aguarda 100ms
LOW 5               'põe nível baixo na porta 5

GOTO REPITA         'retorna para o início do programa
```

Existe um modo de programar o mesmo programa de forma mais enxuta, veja:

Software:

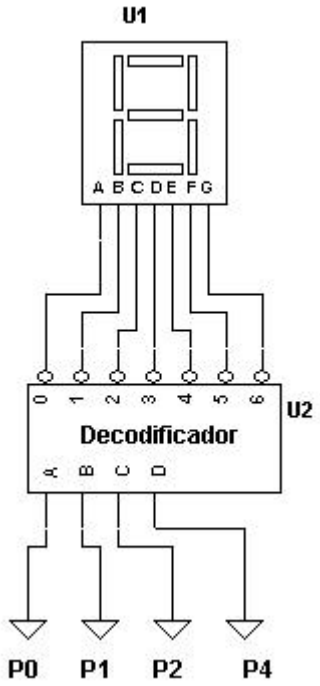
```
'************************************
'*PROGRAMA: Seqüêncial de 6 canais *
'*PROGRAMADOR: Derli L. Angnes     *
'*DATA: 02/11/02                   *
'*VERSÃO: 2.0                      *
'************************************

dirs=%11111111        'atribui todos os pinos como saída
symbol a = b0         'symbol declara variável a como sendo um byte
a =1                  'estamos atribuindo a variável a o valor 1
inicio:               'label inicio

if a = 64 then muda   'se a for igual a 64 então vá para muda
pause 100             'espera 100ms
pins=a                'atribui a porta (0a7) o valor da variável a
a=a*2                 'multiplica (a x 2) e atribui o resultado à a
goto inicio           'salta para o rótulo inicio

muda:                 'label muda
a= 1                  'muda o valor da variável a para 1
goto inicio           'salta para o rótulo inicio e entra em ciclagem
```

Informações sobre variáveis
Variáveis são muito utilizadas em programação, elas podem assumir valores numéricos e podem mudar seu valor ao longo do programa. Se houver necessidade de realizar cálculos num programa, o resultado sempre deve ser dado à uma variável. Para declarar variáveis devemos utilizar o comando "symbol".
Os nomes das variáveis podem ser de três tipos: 1bit, 1byte (8bits) ou 1word(16bits). Sempre devemos declarar as variáveis no menor tamanho possível para o dado que ela irá armazenar. O BAISIC Step tem um número limitado de variáveis. Elas são organizadas em 7 variáveis words (w0 até w6), 14 variáveis bytes (b0 até b13) e 16 variáveis bits (bit0 até bit15).
- 1 byte pode guardar valores de 0 até 255 - 1 word pode guardar valores de 0 até 65.535

Exemplos de atribuição de números nos sistemas: decimal, hexadecimal e binário
 	
`a = 15  'atribuição no sistema decimal (a = 15)`
`a = $f  'atribuição no sistema hexadecimal - $ (a = 15)`
`a = %11111111  'atribuição no sistema binário - % (a = 15)`

c) Contador década

Hardware:



Monte o circuito ao lado. Será necessário um display de 7 segmentos anodo comum e um decodificador BCD p/ código 7 segmentos.
Não esqueça de acrescentar 7 resistores entre a saída do decodificador e o display. O valor do resistor deve ser próximo de 470R.

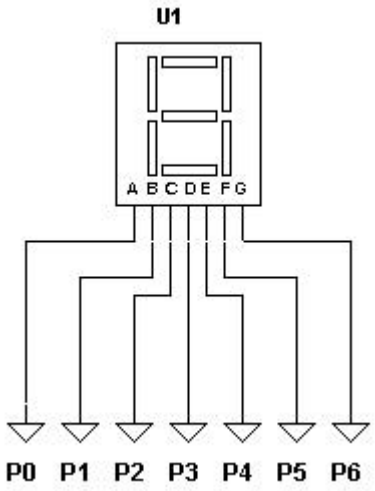
Software:

```
'************************************
'*PROGRAMA: Contador década         *
'*PROGRAMADOR: Derli L. Angnes      *
'*DATA: 02/11/02                    *
'*VERSÃO: 1.0                       *
'************************************

dirs=$ff              'habilita todos pinos como saída
SYMBOL conta = b0     'declara variável conta como sendo do tipo byte
inicio:               'label inicio
conta = 0             'variável conta é igual a 0

incrementa:           'label incrementa

IF conta >= 10 THEN inicio   'impede que o contador passe do valor 9

pins=conta            'atribui a porta o valor atual da variável conta
PAUSE 1000            'pausa por 1 segundo
conta = conta+1      'incrementa a variável conta em uma unidade
GOTO incrementa       'salta para label incrementa
```

d) Contador década sem decodificador e com display anodo comum

Uma das vantagens dos microcontroladores é a possibilidade de trabalhar com tabelas. Para acionar um display de 7 segmentos se faz necessário utilizar um decodificador que possua internamente uma tabela de conversão BCD para 7 segmentos. Vamos criar essa tabela via software, eliminando dessa forma o decodificador externo. O programa a seguir utiliza uma tabela contendo o código que irá imprimir no display uma contagem de 0 a 9. Poderíamos alterar a tabela para acionar um motor-de-passo.

Hardware:



Monte o circuito ao lado, não esqueça de por os resistores (7 x 470R).

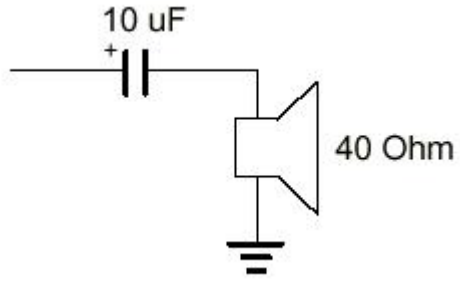
Software:

```
'************************************
'*PROGRAMA: Contador década         *
'*PROGRAMADOR: Derli L. Angnes      *
'*DATA: 02/11/02                    *
'*VERSÃO: 1.0                       *
'************************************
dirs=$ff          'habilita todos pinos como saída
rotate:           ' rótulo
FOR b2=0 TO 9     ' FOR faz b2 assumir valores de 0 a 9
PAUSE 1000        ' pára por 1 seg.
LOOKUP b2,(64,121,36,48,25,18,2,120,0,16),b3  'b2 pega um valor na
'tabela e repassa o dado para a porta, atravez da variável b3.
pins=b3           ' faz com que os dados de b3 passem para a porta
Next              ' retorna para o laço FOR para incrementar unidade
                  ' caso b2>9, o programa continua aqui
GOTO rotate       ' salta para o rótulo "rotate"
```

e) Gerando tons musicais

O BASIC Step possui um comando próprio para gerar tons musicais e ruídos. Todos comandos utilizados pelo BASIC Step podem ser consultados detalhadamente no help (ajuda) do compilador ou no anexo desse fascículo.

Hardware:



Monte o circuito ao lado, utilize a placa StepLab. Interligue o circuito ao pino 1 do BASIC Step. O falante pode ser de 8R/250mW.

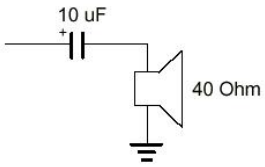
Software:

```
'************************************
'*PROGRAMA: Gera tons               *
'*PROGRAMADOR: Derli L. Angnes      *
'*DATA: 02/11/02                    *
'*VERSÃO: 1.0                       *
'************************************

repete:
SOUND 1,(20,100,56,100,100,100,120,100) 'gera tons musicais
SOUND 1, (250,140)                      'gera um ruído por um tempo
GOTO repete
```

f) Gerando tons musicais aleatoriamente

O BASIC Step possui um comando próprio para gerar notas musicais e ruído, SOUND.

Hardware:



Monte o circuito ao lado, utilize a placa StepLab. Interligue o circuito ao pino 1 do BASIC Step.

Software:

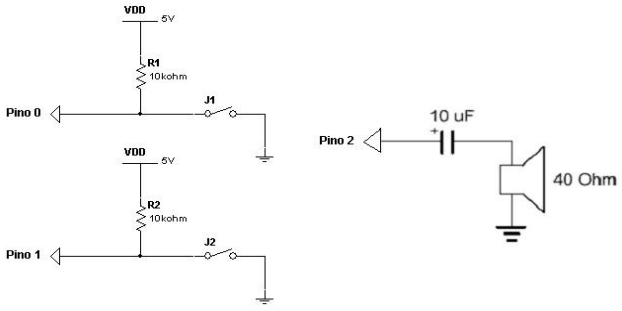
```
'************************************
'*PROGRAMA:Gera tons aleatoriamente*
'*PROGRAMADOR: Derli L. Angnes      *
'*DATA: 02/11/02                    *
'*VERSÃO: 2.0                       *
'************************************
loop:

RANDOM W1              'gera número aleatório de 16BIT's.
B4=b2/2                'pega-se somente os 8 bits LSB e divide por 2
SOUND 1,(b4,100)       'executa tons musicais aleatórios


GOTO loop              'repete o processo
```

Experimento 03 : Trabalhando com entradas

Para declarar um pino como entrada utiliza-se o comando input e para saída o comando output, porém se não for declarado, todos os pinos estão configurados intrinsecamente para serem entradas.

a) Teclando tons musicais

Hardware:



Monte o circuito ao lado, utilize a placa StepLab.

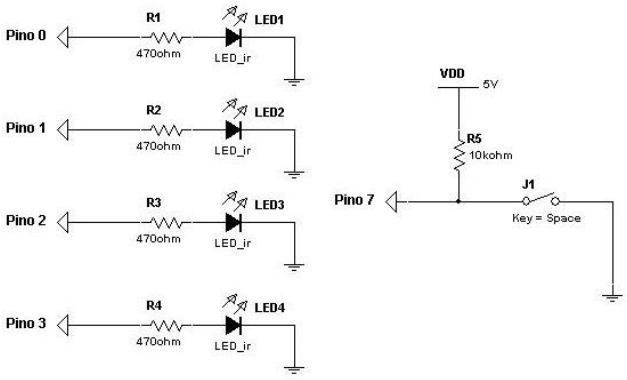
Software:

```
'************************************
'*PROGRAMA: Teclando tons musicais *
'*PROGRAMADOR: Derli L. Angnes     *
'*DATA: 02/11/02                   *
'*VERSÃO: 1.0                      *
'************************************

OUTPUT 2                  'pino 2 é saída
INPUT 0                   'pino 0 e 1 é entrada
INPUT 1

repete:

IF pin0=0 THEN agudo      'se o pino 0 for pressionado pula p/agudo
IF pin1=0 THEN grave      'se o pino 1 for pressionado pula p/grave
GOTO repete               'salta para repete

agudo:                    'rotina para gerar um ton agudo
SOUND 2,(120,150)
GOTO repete

grave:                    'rotina para gerar um ton grave
SOUND 2,(1,150)
GOTO repete
```

b) Escolhendo um led

Dentre 4 led's será possível selecionar um deles por intermédio de um único interruptor.

Hardware:



Monte o circuito ao lado, utilize a placa StepLab. Esse experimento utiliza o mesmo principio de alguns relógios digitais e outros equipamentos eletrônicos. Uma única tecla permite selecionar várias funções (led's).
O comando button é um pouco complexo, porém muito importante para quem pretende trabalhar com entradas sem sofrer problemas com repique de chaves.

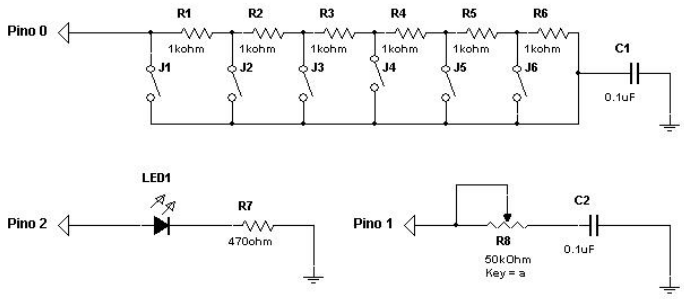
Software:

```
'************************************
'*PROGRAMA: Teclando tons musicais *
'*PROGRAMADOR: Derli L. Angnes     *
'*DATA: 02/11/02                   *
'*VERSÃO: 1.0                      *
'************************************

dirs=%01111111              'Pino 7 declarado entrada, demais saídas
let b2 = 0 : b3 = 0         'zeramento de variáveis
                            'obseve dois comandos em uma única linha
Loop:


BUTTON 7,0,5,1,b2,1,proximo 'quando o botão é acionado vai para
                            'rotina proximo
debug b2                    'mostra o estado de b2 em uma tela do PC
```

```
goto Loop                       'salta para loop

proximo:                        'rotina que irá incrementar os led's
b3=b3+1                         'incrementa em uma unidade
debug b3                        'mostra o esto de b3 na tela do PC
if b3<4 then tabela             'se b3 for menor que 4 salta para tabela

b3=0                            'se b3 for maior que 4 b3 será zerada

tabela:                         'rotina que contém a tabela de saída
lookup b3,(1,2,4,8),pins

goto loop                       'salta para o início do programa
```

c) Conhecendo o comando potenciômetro

Hardware:



Monte o circuito ao lado, utilize a placa StepLab.

Software:

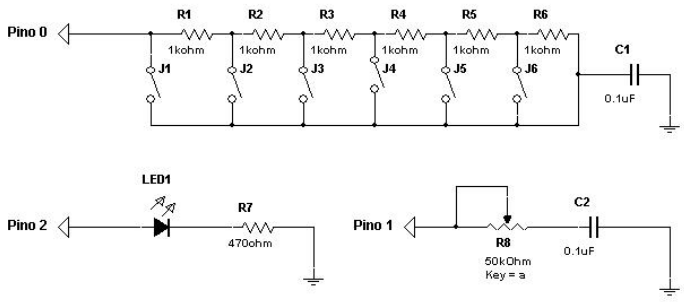
```
'************************************
'*PROGRAMA: Comando potenciômetro   *
'*PROGRAMADOR: Derli L. Angnes      *
'*DATA: 02/11/02                    *
'*VERSÃO: 1.0                       *
'************************************

abc:

POT 0,110,b2        'lê potenciômetro em pino 0 e carrega b2
POT 1,50, b3        'lê potenciômetro em pino 1 e carrega b3

DEBUG b2,b3         'mostra valor das variáveis na tela do PC

goto abc            'repete o processo
```

d) Temporizador de tempo ajustável

O comando potenciômetro possibilita ao programador trabalhar com valores analógicos. Ao variar a resistência elétrica de um potenciômetro iremos modificar o tempo de carga do capacitor, o BASIC Step calcula o tempo de carga do capacitor e repassa um valor numérico a uma variável. O programador pode realizar uma série de sentenças para tomar uma decisão – liga motor, liga lâmpada...
Com o comando POT poderíamos desenvolver um termômetro, termostato de forma simples.

Veja no próximo experimento um exemplo de temporizador. As chaves J1 a J6 permitem a escolha de tempos diferenciados, R8 modifica o tempo.

Hardware:



Monte o circuito ao lado, utilize a placa StepLab.

Caso o experimento não funcione será necessário reajustar valores do comando POT ou até mesmo das condições (if then).

Software:

```
'************************************
'*PROGRAMA: Comando potenciômetro  *
'*PROGRAMADOR: Derli L. Angnes     *
'*DATA: 02/11/02                   *
'*VERSÃO: 1.0                      *
'************************************

output 2

abc:                    'rotina principal
w4=0                    'atribui a w4 como sendo 0
low 2                   'faz pino 2 assumir valor 0
b2=0:b3=0               'zera variáveis b2 e b3
POT 0,110,b2            'lê potenciômetro em pino 0 e carrega b2
POT 1,50,b3             'lê potenciômetro em pino 1 e carrega b3

debug b2,b3             'mostra na tela do PC os valores de b2 e b3

if b2<=0 then t0        'se b2 satisfaz a situação salta para t0
if b2<=22 then t1       'se b2 satisfaz a situação salta para t0
if b2<=36 then t2       'se b2 satisfaz a situação salta para t0
if b2<=50 then t3       'se b2 satisfaz a situação salta para t0
if b2<=63 then t4       'se b2 satisfaz a situação salta para t0
if b2<=77 then t5       'se b2 satisfaz a situação salta para t0
if b2<=90 then abc      'se b2 satisfaz a situação salta para t0

goto abc

t0:                     'rotina do tempo 0
high 2                  'faz pino assumir nível 1
w4=b3*10                'aqui a variável W4 é o resultado de b3 x 10
pause w4                'aguarda o tempo acumulado em w4
low 2                   'faz pino assumir nível 0
goto abc                'retorna para abc

t1:                     'rotina do tempo 2
high 2
w4=b3*20
pause w4
low 2
goto abc

t2:                     'rotina do tempo 3
high 2
w4=b3*30
pause w4
low 2
goto abc

t3:                     'rotina do tempo 4
high 2
w4=b3*40
```

```
pause w4
low 2
goto abc

t4:                 'rotina do tempo 5
high 2
w4=b3*50
pause w4
low 2
goto abc

t5:                 'rotina do tempo 6
high 2
w4=b3*60
pause w4
low 2
goto abc
```

Experimento 04 : Trabalhando com display de cristal liquido

O display ao lado trabalha no formato serial em 2400bps, 8bits, sem paridade. Toda inicialização é automática e leva em torno de 400ms.
Ele possui 16 colunas e 2 linhas totalizando 32 posições (16x2). É alimentado com 5V com um consumo de 3mA.

O módulo de display LCD usado com o BASIC Step é simples de ser usado, para realizar a comunicação serial utilizamos apenas um pino do BASIC Step. A instrução utilizada para o envio de informações do Step para o display é feita através do comando SEROUT.

Especificações do LCD Serial:

- alimentação: 5v
- consumo: 3mA
- velocidade de comunicação: 2400bps

Funcionamento:

Os dados que serão enviados ao display só serão aceitos no formato serial (comando SEROUT do Basic Step), na velocidade de 2400bps, 8 bits, sem paridade, configurações essas que não podem ser mudadas. O display aceita tanto níveis RS-232, quanto TTL invertido.

Para serem enviados os comandos ao display, será usado o comando SEROUT, o qual será explicado abaixo:

```
serout x,y,z
```

O primeiro parâmetro (x), informa por qual pino do BASIC Step se deseja enviar sinais seriais, no caso, enviar sinais ao display.

O segundo parâmetro (y), é o modo como a porta serial do Step irá enviar os dados. Para uso com o display que necessita da velocidade de 2400bps, esse parâmetro fica assim:

```
serout x,N2400,z
```

O terceiro parâmetro (z), é à parte onde serão enviados os dados ao display.

Comandos:

Para que o display execute as suas funções devem ser enviados comandos que são reconhecidos por ele. Esses comandos são colocados na parte z da instrução serout, e devem vir entre parênteses.

Sempre antes de enviar uma instrução ao display, deve-se enviar o código 254, de acordo com o exemplo abaixo:

```
serout 2,N2400,(254,1)  'limpa o display ligado ao pino 2 do Step.
```

Veja a lista dos códigos e um exemplo de comando:

Comando	Código
Limpar o display	1
Retornar cursor ao inicio	2
Apagar o display	8
Ligar o display	12
Desligar cursor	12
Ligar cursor piscante	13
Ligar cursor fixo	14
Mover o display p/ a esquerda	24
Mover o display p/ a direita	28
Mover cursor p/ a esquerda	16
Mover cursor p/ a direita	20
Posicionar cursor (vide tabela de posições)	nº em DECIMAL, que representa a posição

Exemplo de comando:

```
...
serout 0,N2400,(254,1,2)        'limpa o display, coloca o cursor no
                                'inicio; display ligado ao pino 0
serout 0,N2400,("BASIC Step")   'mostra a mensagem "BASIC Step" na tela
                                'do display
...
```

****** Note que o comando 254, sempre vai antes de qualquer instrução a ser enviada ao display, com exceção das mensagens, que não devem vir com os 254.

Posições:

Usando os comandos apresentados na tabela abaixo, pode-se colocar o cursor em qualquer das 32 posições no display. Essas posições referem-se ao display 16X2.

LCD 16x2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Linha1	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
Linha2	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207

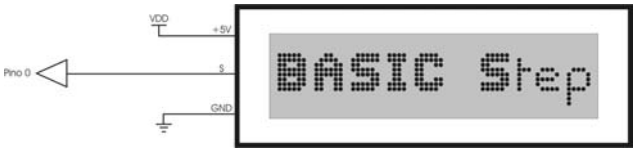
Exemplo de uso das posições:

```
...
serout 0,N2400,(254,1,192)      'posiciona cursor no início da 2ª linha
serout 0,N2400,("basic step")   'mostra a mensagem "basic step" na
                                'posição indicada acima
...
```

a) Inserindo mensagem no LCD

Esse programa conta de 0 a 255 e mostra cada valor no display.

Hardware:



Monte o circuito ao lado, utilize a placa StepLab. Tome cuidado ao conectar o display a placa StepLab. Utilize a saída de 5V.

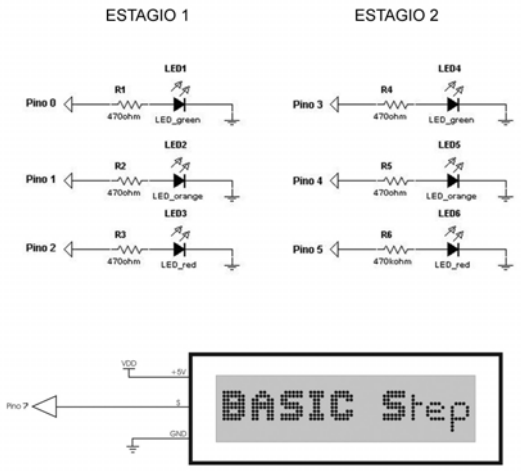
Software:

```
'************************************
'*PROGRAMA: Mensagem no LCD         *
'*PROGRAMADOR: Derli L. Angnes      *
'*DATA: 02/11/02                    *
'*VERSÃO: 1.0                       *
'************************************

symbol cmd=254                          'atribuição de valores
symbol clear=1
symbol home=2

b0=0
serout 0,N2400,(cmd,clear)              'limpa display
loop:
serout 0,N2400,(cmd,home)               'cursor no inicio
serout 0,N2400,("valor de b0=",#b0)     'mostra valor de b0 no display
b0=b0+1                                 'incrementa b0
pause 500
goto loop                               'repete o processo
```

Exemplo de aplicação: Semáforo

Iremos demonstrar uma aplicação prática envolvendo o BASIC Step, um semáforo de dois tempos. A idéia inicial do software é dos estudantes em eletrônica: Jonas e Marcelo da turma de 2001/2002 de aprendizagem industrial do SENAI de Santa Cruz do Sul/RS.

Hardware:



Monte o circuito ao lado, utilize a placa StepLab. Tome cuidado ao conectar o display a placa StepLab. Utilize a saída de 5V.

Jonas e Marcelo no Laboratório

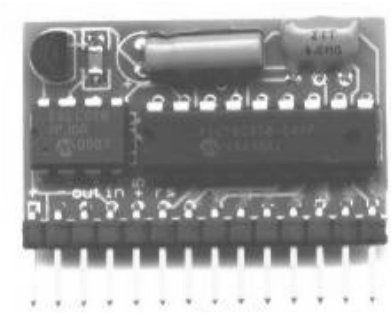
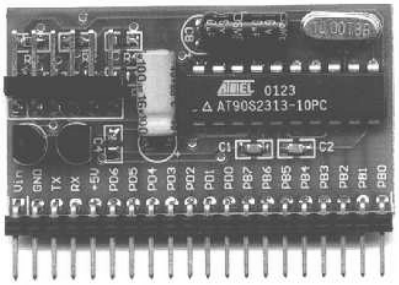
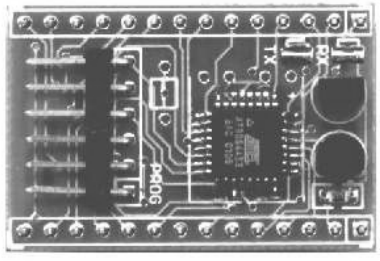
Software:

```
'***********************************
'*PROGRAMA: Semáforo de ponto único*
'*PROGRAMADOR: Jonas & Marcelo     *
'*DATA: 02/11/02                   *
'*VERSÃO: 1.0                      *
'***********************************

output 0            ''Verde
output 1            ''Laranja
output 2            ''Vermelho
output 3            ''Verde2
output 4            ''Laranja2
output 5            ''Vermelho2
output 7            ''LCD-vr(verde)-lj(laranja)-vm(vemelho)

inicio:

serout 7,n2400,(254,1)
serout 7,n2400,("Tempo 1  Tempo 2")


goto estagio1


goto inicio

estagio1:
pin0 = 0
pin0 = 1                  '' Liga o Verde
pin5 = 1                  '' Liga vermelho2
serout 7,n2400,(254,194)
serout 7,n2400,("Verd")
serout 7,n2400,(254,203)
serout 7,n2400,("Verm")
pause 15000
pin0 = 0                  '' Desliga verde
goto estagio2

estagio2:
pin1= 0
pin1 = 1                  ''Liga laranja
serout 7,n2400,(254,194)
serout 7,n2400,("Larj")
''''''''''
pause 500
pin5 = 0
pause 500
pin5 = 1
pause 500                  ''RESPONSÁVEL PELO PISCA-PISCA DO VERMELHO2
pin5 = 0
pause 500
pin5 = 1
pause 500
pin5 = 0
pause 500

pin1 = 0                   ''Desliga laranja
goto estagio3
''''''''''
estagio3:
pin2 = 0
pin2 = 1                   ''liga vermelho
pin5 = 0                   ''desliga verm.2
pin3 = 1                   ''liga verde2
serout 7,n2400,(254,194)
serout 7,n2400,("Verm")
```

```
serout 7,n2400,(254,203)
serout 7,n2400,("Verd")
pause 15000
serout 7,n2400,(254,203)
serout 7,n2400,("Larj")
pin3 = 0                    ''desliga verde2
pin4 = 1                    ''liga laranja2
''''''''''
pause 500
pin2= 0
pause 500
pin2 = 1
pause 500                   ''RESPONSÁVEL PELO PISCA-PISCA DO VERMELHO1
pin2 = 0
pause 500
pin2 = 1
pause 500
pin2 = 0
pause 500
''''''''''
pin4 = 0                    ''desliga laranja2
pin2 = 0                    ''desliga vermelho1

goto inicio                 ''Reinicia o programa
```

Família BASIC Step

O BASIC Step-I é ideal para iniciação e aprendizado em microcontroladores, mas chegamos em um momento que suas limitações, como 256 bytes de memória, são empecilho para projetos. O mercado apresenta outros membros da mesma família: BASIC Step-II e o Step-III.

BASIC Step-I		256 posições de memória (256bytes) 8 entradas/saídas Alimentação de 7,5V a 15V Tamanho: 3,5 x 3,0 cm
BASIC Step-II		2048 posições de memória (2k) Clock de 10 MHz 15 entradas/saídas Tamanho: 5cm x 2,7cm
BASIC Step-III		4096 posições de memória (4k) Clock de 8 MHz 20 entradas/saídas Conversor AD de 6 canais e 10 bits Tamanho: 3,5cm x 2,5cm

ANEXO – Instruções de comando do BASIC Step

COMANDO

ALTO pino

LIGA pino

HIGH pin

●Muda o pino especificado para nível alto (5V). Se o pino estiver programado como entrada, ele muda para saída.

►**Pin** é uma variável/constante (0-7) que especifica o pino de entrada/saída.

A instrução High é equivalente a:

```
output 3       ' Make pin 3 an output.
let pin3 = 1   ' Set pin 3 high.
```

Note que o comando Output aceita o número do pino (3), enquanto que Let necessita o nome do pino pin3. Então além de economizar uma instrução, o comando High permite fazer um pino saída alta utilizando somente números. Olhando o programa abaixo, imagine como seria difícil escrevê-lo usando Output e Let.

Isto mostra um problema com o High. Os programadores, às vezes, substituem os nomes dos pinos como pin3 pelo número do pino. Lembre-se que os nomes são variáveis bit. Como bit, eles podem armazenar 0 ou 1. A linha "High pin3" é uma linha válida, mas significa, "Pegue o estado do pin3. Se pin3 é 0, faça o pino 0 como saída em nível alto. Se pin3 é 1, faça o pino 1 saída em nível alto".

Programa exemplo:

```
' One at a time, change each of the pins to output and set it high.
for b2 = 0 to 7   ' Eight pins (0-7).
HIGH b2           ' Set pin no. indicated by b2.
pause 500         ' Wait 1/2 second between pins.
next              ' Do the next pin.
```

COMANDO

ALEATÓRIO

RANDOM wordvariable

●Gera um número pseudo-aleatório em wordvariable.

►**Wordvariable** é uma variável/constante (0-65535) usada pela rotina e que guarda o valor gerado. A cada execução do comando o número gerado é colocado na variável.

O Basic Step usa uma seqüência de 65535 números aleatórios para executar esta instrução. Quando Random executa, o valor em wordvariable determina em que lugar da seqüência pegar o próximo número. Se o número inicial for sempre o mesmo, a mesma seqüência de números será gerada. Apesar deste método não ser absolutamente aleatório, ele é bom o bastante para a maioria das aplicações.

Para obter um resultado verdadeiramente aleatório, você precisa adicionar um elemento de incerteza ao processo. Por exemplo, o programa pode executar Random continuamente até o usuário apertar um botão.

Programa exemplo:

```
loop:
RANDOM w1       ' Generate a 16-bit random number.
sound 1,(b2,10) ' Generate a random tone on pin 1 using the low
                ' byte of the random number b2 as the note num
goto loop       ' Repeat the process
```

COMANDO

BAIXO pino

DESLIGA pino

LOW pin

●Muda o pino especificado para nível 0. Se o pino estiver programado como entrada, ele muda para saída.

►**Pin** é uma variável/constante (0-7) que especifica o pino a usar.

A instrução Low é equivalente a:

```
output 3       ' Make pin 3 an output.
let pin3 = 0   ' Set pin 3 low.
```

Note que o comando Output aceita o número do pino (3), enquanto que Let necessita o nome do pino pin3. Então além de economizar uma instrução, o comando Low permite fazer um pino saída baixa utilizando somente números. Olhando o programa abaixo, imagine como seria difícil escreve-lo usando Output e Let.

Isto mostra um problema com o Low. Os programadores, às vezes, substituem os nomes dos pinos como pin3 pelo número do pino. Lembre-se que os nomes são variáveis bit. Como bit, eles podem armazenar 0 ou 1. A linha "Low pin3" é uma linha válida, mas significa, "Pegue o estado do pin3. Se pin3 é 0, faça o pino 0 como saída em nível baixo. Se pin3 é 1, faça o pino 1 saída em nível baixo".

Programa exemplo:

```
'One at a time, change each of the pins to output and set it high.

for b2 = 0 to 7   ' Eight pins (0-7).
LOW b2            ' Reset pin no. indicated by b2.
pause 500         ' Wait 1/2 second between pins.
next              ' Do the next pin.
```

COMANDO

CHAVE pino, ????

BUTTO Npin,downstate,delay,rate,bytevariable,targetstate, address

●Debounce button input, perform auto-repeat, and branch to address if button is in target state. Button circuits may be active-low or active-high (see the diagram on the next page).

►**Pin** é uma variável/constante (0-7) que especifica o pino de I/O a ser usado.

►**Downstate** é uma variável/constante que especifica qual estado lógico é lido quando o botão é pressionado.

►**Delay** é uma variável/constante (0-255) que especifica quanto tempo o botão precisa ser pressionado antes que a repetição automática comece. O tempo é medido em ciclos da rotina Button. O Delay tem dois valores especiais; 0 e 255. Quando for 0, a rotina retorna o estado do botão sem debounce ou repetição automática. Quando for 255, a rotina faz debounce, mas não repetição automática.

►**Rate** é uma variável/constante (0-255) que especifica a taxa de repetição automática. A taxa é expressa em ciclos da rotina Button.

►**Bytevariable** é a área de trabalho da rotina. Ela precisa ser apagada antes de ser usada pela rotina Button pela primeira vez.

►**Targetstate** é uma variável/constante (0 ou 1) que especifica qual estado o botão tem que estar para ocorrer o desvio (0=não pressionada, 1=pressionada).

►**Address** é um label que especifica para onde desviar se o botão estiver no estado desejado.

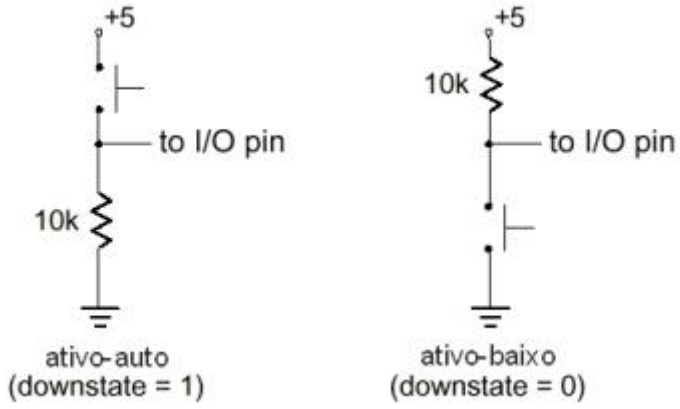
Quando você pressiona um botão ou muda uma chave, os contatos fecham ou abrem a conexão. Um pulso de ruído elétrico ocorre, já que os contatos repicam um contra o outro. A função debounce previne estes ruídos de serem interpretados como mais de uma ação do botão.

O comando Button permite ao Basic Step responder ao pressionamento de um botão do mesmo modo que o teclado do PC faz. Quando você aperta uma tecla, o caractere aparece no monitor. Se você mantém a tecla apertada por um curto periodo de tempo, uma rápida seqüência de caracteres aparece no monitor. A função repetição automática pode ser programada para funcionar do mesmo modo.

O comando Button foi criado para ser usada dentro de um loop. A cada interação do loop, o comando Button checa o estado do pino especificado. Quando ele encontra pela primeira vez o estado especificado em downstate, o comando elimina o ruído elétrico. Ele então de acordo com o valor de targetstate, desvia para address (targetstate=1) ou não (targetstate=0).

Se o botão é mantido em downstate, o comando Button conta o número de vezes que o loop executa. Quando esta contagem chega a delay, o comando executa novamente a ação especificada por downstate e address. A partir de então, se o botão permanece em downstate, o comando espera rate número de ciclos entre as ações.

O mais importante a lembrar sobre o comando Button, é que ele não interrompe a execução do programa. De modo a função delay repetição automática funcionarem, o comando Button deve ser executado dentro de um loop.



Programa exemplo:

```
' This program toggles (inverts) the state of an LED on pin 0 when the
' active-low switch on pin 7 is pressed. When the switch is held down, Button
' waits, then rapidly autorepeats the Toggle instruction, making the LED
' flash rapidly. When the switch is not pressed, Button skips the Toggle
' instruction. Note that b2, the workspace variable for Button, is cleared
' before its first use. Don't clear it within the loop.

let b2 = 0                    ' Button workspace cleared.
```

```
Loop:
BUTTON 7,0,200,100,b2,0,skip ' Go to skip unless pin7=0.
Toggle 0                     ' Invert LED.
...                          ' Other instructions.
skip:
goto Loop                    ' Skip toggle and go to Loop.
```

Ligação do LED para o exemplo

COMANDO

CONSOLE variável {variável}

DEBUG variable {,variable}

●Mostra a variável especificada (bit, byte ou word) numa janela na tela do PC conectado ao Basic Step. O comando Debug funciona somente após o programa estar rodando.

COMANDO

DESCANÇA período

NAP period

●Entra no modo sleep por um curto período de tempo. O consumo de energia é reduzido para algo em torno de 20uA, assumindo que não há cargas nas saídas.

►**Period** é uma variável/constante que determina a duração do modo sleep. A duração é de (2^2 period) x18ms. Period pode variar de 0 a 7, resultando nas durações mostradas abaixo.

Period Nap Length

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

Nap usa o mesmo mecanismo de desligamento/religamento de Sleep, com uma grande diferença. Durante Sleep o Basic step compensa as variações de velocidade do temporizador watchdog que serve como despertador. Como resultado, intervalos longos de sleep tem precisão de ±1%. O comando Nap é controlado pelo temporizador watchdog sem correção. variações em temperatura, tensão e fabricação do chip podem fazer a duração do nap variar em até -50 e +100% (por exemplo o comando NAP 0 pode variar de 9 a 36 ms).

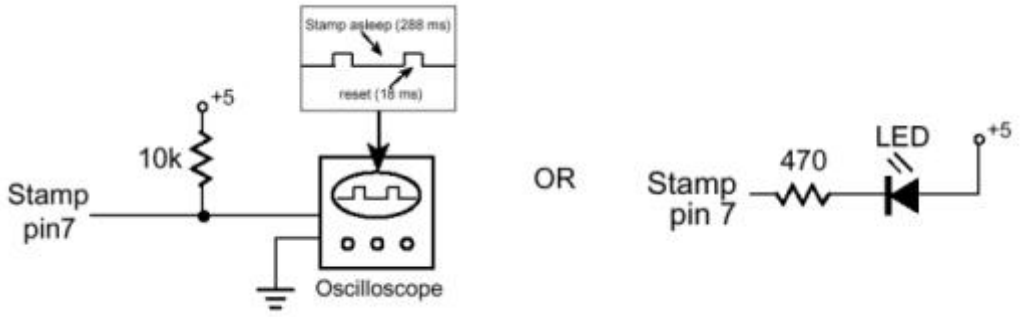
Se a sua aplicação está alimentando cargas durante o comando nap, a energia vai ser interrompida por um período de 18ms quando a Basic Step acorda. A razão é que o reset que acorda o chip também muda todos os pinos para entrada. Quando o PBASIC volta a ter controle do chip, ele restaura as direções corretas dos pinos.

Quando você usa End, Nap ou Sleep, certifique-se que as cargas podem suportar estas faltas de energia. Um modo simples de contornar isto é conectar resistores para o terra ou +5V conforme o necessário, para assegurar o contínuo fornecimento de corrente durante o reset.

O programa exemplo abaixo pode ser usado para demonstrar o efeito do comando nap usando um led e resistor ou um osciloscópio como mostrado na figura.

Programa exemplo:

```
' During the Nap period, the Stamp will continue to drive loads connected to

' pins that are configured as outputs. However, at the end of a Nap, all pins
' briefly change to input, interrupting the current. This program may be
' used to demonstrate the effect.
low 7         ' Make pin 7 output-low.
Again:
NAP 4         ' Put the Stamp to sleep for 288 ms.
goto Again    ' Nap some more.
```



COMANDO

BAIXO pino

DESLIGA pino

LOW pin

●Muda o pino especificado para nível 0. Se o pino estiver programado como entrada, ele muda para saída.

►**Pin** é uma variável/constante (0-7) que especifica o pino a usar.

A instrução Low é equivalente a:

```
output 3       ' Make pin 3 an output.
let pin3 = 0   ' Set pin 3 low.
```

Note que o comando Output aceita o número do pino (3), enquanto que Let necessita o nome do pino pin3. Então além de economizar uma instrução, o comando Low permite fazer um pino saída baixa utilizando somente números. Olhando o programa abaixo, imagine como seria difícil escrevê-lo usando Output e Let.

Isto mostra um problema com o Low. Os programadores, às vezes, substituem os nomes dos pinos como pin3 pelo número do pino. Lembre-se que os nomes são variáveis bit. Como bit, eles podem armazenar 0 ou 1. A linha "Low pin3" é uma linha válida, mas significa, "Pegue o estado do pin3. Se pin3 é 0, faça o pino 0 como saída em nível baixo. Se pin3 é 1, faça o pino 1 saída em nível baixo".

Programa exemplo:

```
‘One at a time, change each of the pins to output and set it high.

for b2 = 0 to 7   ' Eight pins (0-7).
LOW b2            ' Reset pin no. indicated by b2.
pause 500         ' Wait 1/2 second between pins.
next              ' Do the next pin.
```

COMANDO

DORME segundos

SLEEP seconds

●Entra no modo sleep pelo número especificado de segundos.

►**Seconds** é uma variável/constante (1-65535) que espacifica a duração do modo sleep em segundos. A duração do sleep pode variar de 2.3 segundos (veja a nota abaixo) até mais ou menos 18 horas. O consumo de energia é reduzido a 20uA, assumindo que não há outras cargas ligadas.

Nota: A resolução do Sleep é de 2.304 segundos. Sleep arredonda o valor para o múltiplo de 2.304 mais próximo. Sleep 1 cause uma pausa de 2.304 segundos, enquanto Sleep 10 uma de 11.52 segundos (5 x 2.304).

O comando Sleep permite ao Basic Step desligar-se e após um período de tempo especificado, religar-se. O despertador que acorda o Step é chamado watchdog timer. O watchdog é um oscilador interno ao chip. Durante a pausa, o Step periodicamente acorda e ajusta um contador para determinar quanto tempo ele está no modo Sleep. Se ainda não tiver passado o tempo especificado, ele volta para o Sleep.

Para assegurar intervalos precisos, o Step periodicamente compara o período do watchdog com um oscilador mais preciso. Períodos longos de Sleep têm uma precisão de ±1%. Se o Basic Step está ativando alguma carga durante o Sleep, a corrente será interrompida por aproximadamente 18ms quando o Step acorda a cada 2.3 segundos. A razão é que o reset que acorda o Step, também coloca todos os pinos como entrada e o Step leva 18ms para voltar os pinos para seus estados corretos.

Se você planeja usar os comandos End, Nap ou Sleep em seus programas, certifique-se que suas cargas podem tolerar estas faltas de energia momentâneas. O modo mais simples de contornar isto é conectar resistores para +5V ou terra conforme o necessário para suprir energia para as cargas.

Programa exemplo:

```
'SLEEP 3600 ' Sleep for about 1 hour.
goto xyz    ' Continue with program
            ' after sleeping.
```

COMANDO

ENTRADA pino

INPUT pin

●Muda o pino especificado para entrada. Isto desliga o driver de saída do pino, permitindo o seu programa ler qualquer estado que estiver presente no pino.

►**Pin** é uma variável/constante (0-7) que especifica qual pino usar.

Existem diversos modos de setar um pino como entrada. Quando o programa inicia, todos os pinos estão como entrada. As instruções de entrada (Pulsin, Serin) automaticamente mudam o pino especificado para entrada e o deixam neste estado. Escrevendo 0 no bit correspondente da variável dirs muda o pino para entrada. E também a instrução Input.

Quando um pino é setado como input, seu programa pode checar o seu estado lendo o seu valor. Por exemplo:

```
Hold: if pin4 = 0 then Hold ' Stay here until pin4 is 1.
```

O programa está lendo o estado do pino 4 que está sendo ativado por um circuito externo. Se nada estiver conectado ao pino 4, ele pode estar em qualquer estado (1 ou 0) e pode mudar de estado de modo aleatório.

O que acontece se o seu programa escreve num pino setado como entrada? O dato é escrito na porta de saída, mas como o driver de saída está desligado, não há mudança. Se o pino é mudado para saída, o último valor escrito na porta vai aparecer no pino. O programa seguinte explica como isto funciona:

Programa Exemplo:

```
' To see this program in action, connect a 10k resistor from pin 7 to +5V.
' When the program runs, a debug window will show you the state at pin 7
' (a 1, due to the +5 connection); the effect of writing to an input pin (none);
' and the result of changing an input pin to output (the latched state appears
' on the pin and may be read by your program). Finally, the program shows
' how changing pin 7 to output writes a 1 to the corresponding bit of dirs.

INPUT 7                                     ' Make pin 7 an input.
debug "State present at pin 7: ",#pin7,cr,cr
let pin7 = 0                                ' Write 0 to output latch.
debug "After 0 written to input: ",#pin7,cr,cr
output 7                                    ' Make pin 7 an output.
debug "After pin 7 changed to output: ",#pin7,cr
debug "Dirs (binary): ",#%dirs              ' Show contents of dirs.
```

COMANDO

ESCREVE posição,dado

WRITE location,data

●Guarda dados na memória EEPROM.

►**Location** é uma variável/constante (0-255) que determina em qual posição da memória armazenar o dado.

►**Data** é uma variável/constante (0-255) que vai ser guardada na memória.

A memória EEPROm é utilizada tanto para o armazenamento do programa (armazenado a partir do endereço 254, para baixo) e dos dados(a partir do 0 e para cima). Para garantir que seu programa não grava os dados em cima dele mesmo, leia a posição 255 da EEPROM antes de guardar qualquer dado. A posição 255 guarda o endereço da primeira instrução do seu programa. Então, seu programa pode utilizar qualquer espeço abaixo do valor no endereço 255. Por exemplo, se a posição 255 tem o valor 100, então o seu programa pode utilizar as posições 0-99 para os dados.

Programa exemplo:

```
read 255,b2           ' Get location of last
                      ' program instruction.loop:
b2 = b2 - 1           ' Decrement to next
                      ' available EEPROM location
serin 0,N300,b3       ' Receive serial byte in b3.
WRITE b2,b3           ' Store received serial
                      ' byte in next EEPROM location.
if b2 > 0 then loop   ' Get another byte if there's room.
```

COMANDO

ESCREVESERIAL

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

●Configura uma porta serial de saída e transmite dados.

►**Pin** é uma variável/constante (0-7) que especifica qual pino usar.

►**Baudmode** é uma variável/constante (0-15) que especifica o modo da porta serial. Baudmode pode ser o número indicado por # ou o símbolo mostrado na tabela. Os outros parâmetros de comunicação são setados para os valores mais utilizados: sem paridade, 8 bits de dados e um stop bit, geralmente abreviado como 8N1. Estes parâmetros não podem ser alterados.

►**Data** são variáveis byte/constantes (0-255) que serão transmitidas. Se forem precedidas pelo símbolo #, os dados são transmitidos como cadeias de caracteres de até 5 caracteres. Sem o #, os dados são transmitidos como um byte único.

#	Simbolo	Baud Rate	Polaridade	Modo de saída
0	T2400	2400	true	always driven
1	T1200	1200	true	always driven
2	T600	600	true	always driven
3	T300	300	true	always driven
4	N2400	2400	inverted	always driven
5	N1200	1200	inverted	always driven
6	N600	600	inverted	always driven
7	N300	300	inverted	always driven
8	OT2400	2400	true	open drain (driven high)
9	OT1200	1200	true	open drain (driven high)
10	OT600	600	true	open drain (driven high)
11	OT300	300	true	open drain (driven high)
12	ON2400	2400	inverted	open source (driven low)
13	ON1200	1200	inverted	open source (driven low)
14	ON600	600	inverted	open source (driven low)
15	ON300	300	inverted	open source (driven low)

O comando Serout faz do pino especificado uma porta serial de saída com as características definidas por baudmode. Ele transmite o dado em uma entre dois modos:

- Um byte único.

- Uma cadeia de caracteres de com 1 a 5 caracteres.

Aqui estão alguns exemplos:

```
SEROUT 0,N2400,(65)
```

Transmite um byte com valor 65 através do pino 0 a 2400 bauds, polaridade invertida. Se você receber este byte em um PC rodando um programa de comunicação, o caractere "A' irá aparecer na tela, porque 65 é o código ASCII de "A".

```
SEROUT 0,N2400,(#65)
```

Transmite a cadeia de caracteres "65" através do pino 0 a 2400 bauds, polaridade invertida. Se você receber este byte em um PC rodando um programa de comunicação, o texto "65" irá aparecer na tela. Quando o valor é precedido pelo símbolo "#", Serout automaticamente converte o valor para uma cadeia de caracteres.

Quando você deve usar o símbolo #? Se você está enviando dados deo Basic Step para um PC onde as pessoas irão ler o resultado, use #. Se você está enviando dados para outro Basic Step ou para um computador para futuro processamento, é mais eficiente omitir o #.

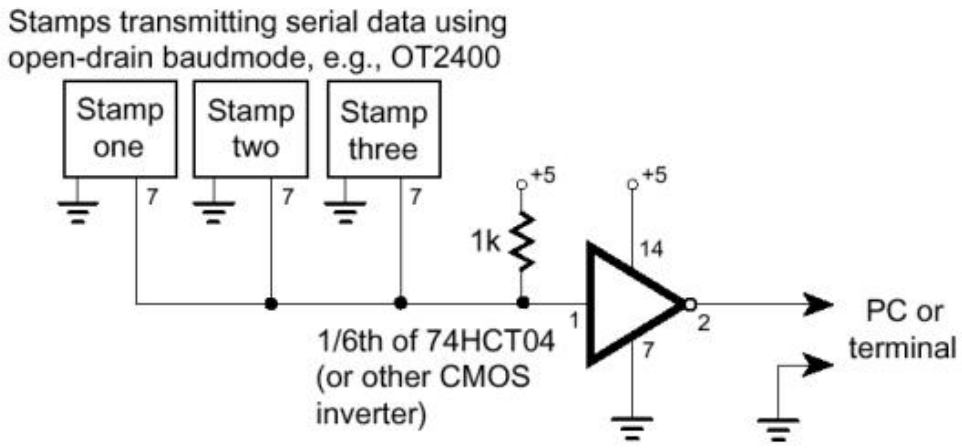
Enviando dados para o PC: Para enviar dados serialmente do Basic Step para o PC, você precisa alguns fios, um conector e um programa de comunicação. Ligue o conector conforme mostrado no esquema abaixo e utilize polaridade invertida. Embora a saída do Basic Step possa variar entre 0 e 5 volts (e não entre +10V e -10V conforme o recomendado para RS-232), a maioria dos PCs recebem os dados sem problemas.

Se você não possui um programa de comunicação, pode digitar e rodar o seguinte programa em QBASIC para configurar a porta serial e receber os dados do Basic Step.

QBASIC Program to Receive Data:

```
' This program receives data transmitted by the Stamp through the PC's
' COM1: serial port and displays it on the screen. To end the program,
' press control-break. Note: in the line below, the "0" in "CD0,CS0..." is a zero.

OPEN "COM1:2400,N,8,1,CD0,CS0,DS0,OP0" FOR INPUT AS #1
CLS
Again:
theChar$ = INPUT$(1,#1)
PRINT theChar$;
GOTO Again
```

Opções open-drais/open-source. As oito últimas opções de configuração do comando Serout começam com "O" de open-drais ou open-source. O diagrama abaixo mostra como utilizar o modo open-drain para conectar dois ou mais Basic Step a um único PC formando uma rede. Você poderia também usar o modo open-source, mas o resistor deveria ser ligado à terra e o inversor substituído por um buffer não inversor.



Para entender porque você deve usar o modo "open" numa rede, considere o que aconteceria se você não usasse. Quando nenhum dos Steps estão transmitindo, todos os pinos estão em estado alto. Como todos estão em 5V, nenhuma corrente flui entre os pinos. Então um Basic step transmite e muda sua saída bara nível baixo. Com a sua saída em nível baixo, ha um curto entre o +5V E O TERRA. A corrente flui entre os pinos, possivelmente danificando-os.

Se todos os Basic Steps são configurados para open-drais, o caso é diferente. Quando os Steps não estão transmitindo seus pinos estão como entrada, efetivamente desconectando-os da linha serial. O resistor para o +5V mantém a linha em nível alto. Quando um Step transmite, ele faz a linha serial ir para nível baixo. Quase nenhuma corrente flui pelos pinos dos outros

Basic Steps, pois estão como entrada. Mesmo se dois Steps transmitem simultaneamente, eles não podem danificar um ao outro.

Programa Exemplo:

```
abc:
pot 0,100,b2         ' Read potentiometer on pin 0.
SEROUT 1,N300,(b2)   ' Send potentiometer
                     ' reading over serial output.
goto abc             ' Repeat the process.
```

COMANDO

EXECUTE endereço

GOSUB address

●O comando Gosub armazena o endereço da instrução seguinte a ele, desvia para address, e continua a execução lá. O próximo comando Return faz o programa voltar para o endereço armazenado, continuando a execução do programa na instrução seguinte ao gosub.

►**Address** é um label que especifica para onde desviar. Até 16 GOSUBS são permitidos por programa. Até 4 GOSUBs podem ser aninhados. O PBASIC armazena os dados referente ao gosub na variável w6. Certifique-se que seu programa não escreva em w6 a não ser que todos os Gosubs tenham terminado, e não espere que o valor em w6 esteja intacto após um Gosub.

Se uma série de instruções é usada em mais de um ponto de seu programa, você pode transformas estas instruções em uma sub-rotina. Então, onde você precisar inserir este código, você pode simplesmente escrever Gosub label (onde label é o nome de sua sub-rotina).

Programa exemplo:

```
' In this program, the subroutine test takes a pot measurement, then performs
' a weighted average by adding 1/4 of the current measurement to 3/4 of a
' previous measurement. This has the effect of smoothing out noise.

for b0 = 1 to 10
GOSUB test               ' Save this address & go to test.
serout 1,N2400,(b2)      ' Return here after test.
next                     ' Again until b0 > 10.
end                      ' Prevents program from running into test.
test:
pot 0,100,b2             ' Take a pot reading.
let b2 = b2/4 + b4       ' Make b2 = (.25*b2)+b4.
let b4 = b2 * 3 / 4      ' Make b4 = .75*b2.
return                   ' Return to previous address+1 instruction.
```

A instrução Return no fim de test manda o programa para a instrução imediatamente seguinte ao Gosub, neste caso Serout.

Certifique-se que o seu programa não alcance uma sub-rotina a não ser com o use de um Gosub. No exemplo o que aconteceria se o end fosse removido? Após o loop, a execução do programa continuaria em test. Quando ele encontrasse o Return, ele retornaria para o comando Serout porque este foi o último endereço retornado. O loop For...Next executaria para sempre.

COMANDO

FIM

END

●Entra no modo sleep. O Basic Step acorda quando é desligado e ligado novamente ou o PC conecta. O consumo é reduzido para aproximadamente 20uA, assumindo que não há cargas nas saídas.

Se você deixa os pinos do Basic Step como saída em nível alto ou em nível baixo, duas coisas acontecem:

- As cargas continuarão a consumir corrente da fonte do Basic Step.

- A cada 2.3 segundos, a corrente destas cargas serão interrompidas por um período de aproximadamente 18 milisegundos(ms).

A razão para esta pequena interrupção a cada 2.3 segundos tem a ver com o circuito do chip do interpretador PBASIC. Ele tem um oscilador chamado "watchdog timer" que pode resetar o processador periodicamente fazendo o Basic Step sair do modo sleep. Uma vez acordado, o Basic Step checa seu programa para verificar se ele deve continuar acordado ou voltar para o modo sleep. Após uma instrução End, o Basic Step tem ordem de permanecer no modo sleep.

Infelizmente o watchdog timer não pode ser desligado, assim o Basic Step fica no modo sleep em intervalos de 2.3s. Após este período, o watchdog reseta o chip. Entre outras coisas o reset passa todos os pinos para o modo de entrada. E firmaware PBASIC leva aproximadamente 18ms para voltar os pinos para seus estados anteriores e voltar ao modo sleep.

Se você utiliza os comandos End, Nap ou Sleep em seus programas, tenha certeza que as suas cargas podem tolerar estas quedas de energia. A solução mais fácil é conectar resistores de pull-up ou pull-down conforme o necessário para garantir o contínuo fornecimento de corrente durante o período de reset.

COMANDO

GERAPULSO pino,tempo

PULSOUT pin,time

●Gera um pulso invertendo o estado do pino por um determinado período de tempo. Se o pino estiver como entrada quando Pulsout for executado, ele será mudado para saída.

►**Pin** é uma variável/constante (0-70 que especifica o pino a ser usado.

►**Time** é uma variável/constante (0-65535) que especifica a duração do pulso em múltiplos de 10us.

Programa exemplo:

```
abc:
PULSOUT 0,3    ' Invert pin 0 for 30
               ' microseconds.
pause 1        ' Pause for 1 ms.
goto abc       ' Branch to abc.
```

COMANDO

INVERTE pino

TOGGLE pin

●Muda o pino especificado para saída e muda o seu estado.

►**Pin** é uma variável/constante (0-70 que especifica qual pino usar.

Programa exemplo:

```
for b2 = 1 to 25
TOGGLE 5           'Toggle state of pin 5.
Next
```

COMANDO

INVERTEDIREÇÃO pino

REVERSE pin

●O comando Reverse inverte a direção de um determinado pino. Se o pino é uma saída, ele passa a ser entrada e vice-versa.

►Pin é uma variável/constante (0-70 que especifica qual pino usar.

Veja os comandos Input e Output para mais informações sobre a direção dos pinos.

Programa exemplo:

```
dir3 = 0  ' Make pin 3 an input.
REVERSE 3 ' Make pin 3 an output.
REVERSE 3 ' Make pin 3 an input.
```

COMANDO

LE posição,variável

READ location,variable

● Le o valor de um aposição da memória EEPROM e o guarda em uma variável.

► **Location** é uma variável/constante (0-255) que especifica qual posição de memória deve ser lida.

► **Variable** é a variável na qual o valor lido vai ser guardado.

A memória EEPROM é usada tanto para o armazenamento do programa quanto para o armazenamento dos dados. Para garantir que o seu programa não sobre-escreve a si mesmo, leia a posição 255 da memória antes de gravar qualquer dado. A posição 255 armazena o endereço da última posição de memória utilizada pelo programa. Tendo este endereço, o seu programa pode utilizar qualquer endereço abaixo dele. Por exemplo, se a posição 255 tem o valor 100, o seu programa pode utilizar as posições 0-99 para dados.

Programa exemplo:

```
READ 255,b2         ' Get location of last program instruction.
loop:
b2 = b2 - 1         ' Decrement to next available EEPROM location
serin 0,N300,b3     ' Receive serial byte in b3
write b2,b3         ' Store received serial byte in next EEPROM location
if b2 > 0 then loop ' Get another byte if there's room left to store it.
```

COMANDO

LESERIAL

SERIN pin,baudmode,(qualifier,qualifier,...)

SERIN pin,baudmode,{#}variable,{#}variable,...

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

●Configura uma porta serial de entrada e espera por qualificador opcional ou por variáveis.

►**Pin** é uma variável/constante (0-7) que especifica qual pino usar.

►**Baudmode** é uma variável/constante (0-7) que especifica o modo da porta serial. Baudmode pode ser o número ou o símbolo mostrado na tabela abaixo. Os outros parâmetros são deixados nos seus valores mais comuns: sem paridade, 8 bits de dados, um stop bit, 8N1. Estes valores não podem ser mudados.

►**Qualifiers** são variáveis/constantes opcionais (0-255) que devem ser recebidos na ordem especificada para a execução continuar.

►**Variables** (opcional) são variáveis utilizadas para guardar o dado recebido (0-255). Se qualquer qualifier for especificado, ele deve ser recebido antes do valor a ser guardado na variável. Se o sinal # preceder o nome da variável, então o comando Serin irá converter

números em forma de texto (como os digitados no teclado) para um valor a ser armazenado na variável.

O comando Serin faz o pino especificado uma entrada serial com as características dadas por baudmode. Ele recebe um caractere por vez e faz uma das duas coisas:

- Compara com um qualifier.

- Guarda numa variável.

Em ambos os casos, o Basic Step não irá fazer nada até todos os caracteres sejam recebidos na ordem correta e todas as variáveis tenham sido preenchidas. Um único comando Serin pode incluir tanto variáveis para preencher quanto qualifier.

Aqui estão alguns exemplos:

#	Símbolo	Baud Rate	Polaridate
0	T2400	2400	true
1	T1200	1200	true
2	T600	600	true
3	T300	300	true
4	N2400	2400	inverted
5	N1200	1200	inverted
6	N600	600	inverted
7	N300	300	inverted

Aqui estão alguns exemplos:

```
SERIN 0,T300,b2
```

Para a execução do programa até que um byte seja recebido serialmente (polaridade normal, 300 baud) pelo pino 0. Guarda o byte recebido na variável b2 e continua. Por exemplo, se o caracter "A" fosse recebido, Serin iria guardar o valor 65 (o código ASCII de "A") em b2.

```
SERIN 0, T1200,#w1
```

Para a execução do programa até que uma cadeia numérica seja recebida serialmente (polaridade normal, 1200 baud) pelo pino 0. Guarda o valor da cadeia numérica em w1. Por exemplo, se o seguinte texto fosse recebido"XYZ: 576%" Serin iria ignorar "XYZ:" pois são caracteres não numéricos. Ele iria receber os caracteres "5", "7" e "6" até o primeiro caractere não numérico "%". Serin então converteria a cadeia de caracteres "576" no valor 576 e o guardaria em w1. Se o símbolo # antes de w1 fosse omitido, Serin iria receber apenas o primeiro caractere "X" e guardaria o seu código ASCII, 88, em w1.

```
SERIN 0,N2400,("A")
```

Para a execução do programa até que um byte seja recebido serialmente (polaridade invertida, 2400 baud) pelo pino 0. Compara o byte recebido com 65, o código ASCII de "A". Se for igual, continua o programa. Se não for igual, recebe outro caractere e repete a comparação. O programa não irá continuar até que um "A" seja recebido. Por exemplo, se Serin recebesse "LIMIT 65A" a execução do programa não continuaria até que o "A" final fosse recebido.

```
SERIN 0,N2400,("SESAME"),b2,#b4
```

Para a execução do programa até que a cadeia de caracteres recebida serialmente (polaridade invertida, 2400 baud) seja igual a "SESAME". Uma vez que o caractere tenha sido recebido, guarda os dois próximos bytes em b2. Então recebe uma cadeia numérica, converte em um valor e guarda em b4. Por exemplo, se a cadeia recebida fosse "...SESESAME!*****19*" o "...SE" seria ignorado, o "SESAME" seria aceito, o valor 33 (código de "!") seria colocado em b2, o caractere "*" seria ignorado e guardaria os valores "1"e "9" em b4.

Considerações sobre velocidade: O comando Serin é rápido o suficiente para receber múltiplos bytes de dados, não importando o qual rápido o computador manda os dados. Entretanto se o programa recebe os dados, armazena e os processa, o tempo deste processamento pode fazer o próximo comando Seri perder dados ou recebê-los de forma incorreta. Use um dos métodos a seguir para contornar isto:

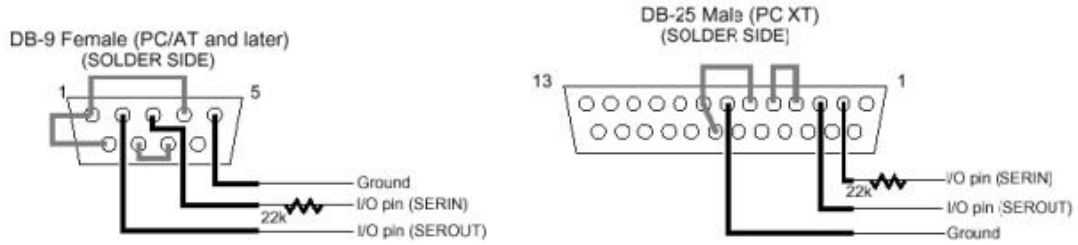
- Aumente o número de stop bits de 1 para 2 (ou mais, se possível)

- Reduza o baud rate

- Se possível adicione um tempo entre a transmissão de cada dado.

- Reduza a quantidade de processamento que o Step tem que executar

Recebendo dados do PC: Para mandar dados serialmente do PC para o Basic Step, tudo que você precisa é um resistor de 22K, alguns fios, um conector e um programa de comunicação. Ligue os fios conforme o diagrama abaixo. Os fios mostrados em cinza desabilitam o controle de fluxo da porta serial, o que normalmente necessitaria de conexões adicionais para operar. O controle de fluxo não é necessário aqui, pois você não irá transmitir um volume muito grande de dados.



Quando você escreve programas para receber dados seriais usando este tipo de conexão, certifique-se de especificar inverted no comando Serin. Se você não tem um programa de comunicação, você pode digitar o seguinte programa em QBASIC para configurar a porta serial (2400,8N1) e transmitir os caracteres digitados no teclado. O QBASIC é o compilador basic que vem com o MS-DOS versão 5 ou superior.

QBASIC Program to Transmit Data:

```
' This program transmits characters typed at the keyboard out the PC's
' COM1: serial port. To end the program, press control-break.
' Note: in the line below, the "0" in "CD0,CS0..." is a zero.

OPEN "COM1:2400,N,8,1,CD0,CS0,DS0,OP0" FOR OUTPUT AS #1
CLS
Again:
theChar$ = INKEY$
IF theChar$ = "" then Goto Again
PRINT #1,theChar$;
GOTO Again
```

Programa exemplo para o Basic Step:

```
' To use this program, download it to the Stamp. Connect
' your PC's com1: port output to Stamp pin 0 through a 22k resistor
' as shown in the diagram. Connect a speaker to Stamp pin 7 as
' shown in the Sound entry. Run your terminal software or the QBASIC
' program above. Configure your terminal software for 2400 baud,
' N81, and turn off hardware handshaking. The QBASIC
' program is already configured this way. Try typing random
' letters and numbers--nothing will happen until you enter
' "START" exactly as it appears in the Stamp program.
' Then you may type numbers representing notes and
' durations for the Sound command. Any non-numeric text
```

```
' you type will be ignored.

SERIN 0,N2400,("START")     ' Wait for "START".
sound 7,(100,10)            ' Acknowledging beep.
Again:
SERIN 0,N2400,#b2,#b3       ' Receive numeric text and
                            ' convert to bytes.
sound 7,(b2,b3)             ' Play corresponding sound.
goto Again                  ' Repeat.
```

COMANDO

Alto pino

LIGA pino

HIGH pin

●Muda o pino especificado para nível alto (5V). Se o pino estiver programado como entrada, ele muda para saída.

►**Pin** é uma variável/constante (0-7) que especifica o pino de entrada/saída.

A instrução High é equivalente a:

```
output 3        ' Make pin 3 an output.
let pin3 = 1    ' Set pin 3 high.
```

Note que o comando Output aceita o número do pino (3), enquanto que Let necessita o nome do pino pin3. Então além de economizar uma instrução, o comando High permite fazer um pino saída alta utilizando somente números. Olhando o programa abaixo, imagine como seria difícil escreve-lo usando Output e Let.

Isto mostra um problema com o High. Os programadores, às vezes, substituem os nomes dos pinos como pin3 pelo número do pino. Lembre-se que os nomes são variáveis bit. Como bit, eles podem armazenar 0 ou 1. A linha "High pin3" é uma linha válida, mas significa, "Pegue o estado do pin3. Se pin3 é 0, faça o pino 0 como saída em nível alto. Se pin3 é 1, faça o pino 1 saída em nível alto".

Programa exemplo:

```
' One at a time, change each of the pins to output and set it high.
for b2 = 0 to 7    ' Eight pins (0-7).
HIGH b2            ' Set pin no. indicated by b2.
pause 500          ' Wait 1/2 second between pins.
next               ' Do the next pin.
```

COMANDO

MEDEPULSO pino,estado,variável

PULSIN pin,state,variable

●Muda o pino especificado para entrada e mede a duração de pulsos em unidades de 10us.

►**Pin** é uma variável/constante (0-7) que especifica o pino a usar.

►**State** é uma veriável/constante (0 ou 1) que especifica qual flanco deve ocorrer para iniciar a medida.

►**Variable** é a variável usada para guardar o resultado da medida. Variable deve ser uma variável word numa faixa de 1 a 65535 ou variável byte com faixa de 1 a 255.

Muitas propriedades analógicas (voltagem, resistência, capacitância, freqüência, ciclo de trabalho) podem ser medidas em termos de duração de pulsos. Isto faz o comando Pulsin uma boa forma de fazer a conversão analógico-digital.

Você pode imaginar o comando Pulsin como um cronômetro que é disparado por uma mudança de estado (0 ou 1) no pino especificado. Quando o estado no pino muda para o

estado especificado em Pulsin, o cronômetro começa a contar. Quando o estado do pino muda novamente, o cronômetro para.

Se o estado do pino não muda (mesmo se ele já estiver no estado especificado na instrução Pulsin) o cronômetro não dispara. O comando Pulsin espera o máximo de 0.65535 segundos pelo disparo e então retorna 0 em variable.

A variável pode ser do tipo word ou byte. Se for word, o valor retornado por Pulsin pode variar de 1 a 65535 unidades de 10us. Se a variável for do tipo byte, o valor retornado pode variar de 1 a 255 unidades de 10us. Internamente, Pulsin sempre utiliza o timer de 16 bits. Quando o seu programa especifica variável byte, Pulsin guarda os 8 bits menos significativos do contador interno. Pulsos de duração maior que 2550us darão falsos valores quando usados com variável byte. Por exemplo, um pulso de 2560us lido com variável word returna 256, mas com variável byte, retorna 0.

Programa exemplo:

```
PULSIN 4,0,w2       ' Measure an input pulse on pin 4.
serout 1,n300,(b5) ' Send high byte of 16-bit pulse measurement
. . .               ' over serial output.
```

COMANDO

PAUSA milisegundos

PAUSE milliseconds

●Para a execução do programa por um determinado período especificado em milisegundos.

►**Milliseconds** é uma variável/constante (0-65535) que especifica quantos milisegundos o programa vai parar.

A pausa causada pelo comando é tão precisa quando o ressonador cerâmico usado pelo Basic Step, +-1%. Quando utilizar o comando Pause em aplicações que precisem de precisão de tempo, tenha em mente a relativa baixa velocidade do interpretador BASIC (mais ou menos 2000 instruções por segundo). Este é o tempo necessário para o chip ler e interpretar as instruções armazenadas na EEPROM.

Visto que o PBASIC leva 0.5 milisegundos para ler a instrução PAUSE e 0.5 milisegundos para ler a instrução seguinte, os loops envolvendo o comando Pause levam, no mínimo, 1 milisegundo a mais do que o especificado pelo comando. Se você está escrevendo loops de tempo longos, tenha em mente a precisão de 1% do oscilador.

Programa exemplo:

```
abc:
low 2       ' Make pin 2 output low.
PAUSE 100   ' Pause for 0.1 second.
high 2      ' Make pin 2 output high.
PAUSE 100   ' Pause for 0.1 second.
goto abc
```

COMANDO

POTENCIOMETRO pino,escala,variável

POT pin,scale,variable

●O comando Pot lê um potenciômetro, termistor, foto-célula ou outro resistor variável. O pino especificado no comando Pot deve ser ligado em um lado do resistor, o outro lado deve ser ligado através de um capacitor à terra. A medida da resistência é feita contando-se o tempo que o capacitor leva para descarregar através do resistor. Se o pino especificado estiver como entrada, ele será mudado para saída.

►**Pin** é uma variável/constante (0-7) que especifica qual pino usar.

►**Scale** é uma variável/constante (0-255) usada para calibrar o resultado de 16 bits. A leitura de 16 bits é multiplicada por scale/256, assim um valor de scale de 128 irá reduzir em 50% o valor do resultado, um valor de 64 irá reduzir a 25% e assim por diante. A opção Alt-P (explicada abaixo) permite achar o melhor valor para uma determinada combinação de resistor e capacitor.

►**Variable** é usado para guardar o resultado final da leitura. Internamente, a instrução Pot calcula o valor em 16 bits o qual é reduzido a 8 bits. O quanto o valor interno deve ser reduzido varia com o valor do resistor usado.

Encontrando o melhor valor de scale:

- Para encontrar o melhor valor de scale, conecte o resistor a ser usado e conecte o Step ao PC.

- Pressione Alt-P quando estiver rodando o software de edição do Basic Step. Uma janela especial de calibração irá aparecer, permitindo achar o melhor valor.

- A janela pergunta em qual pino o resistor está conectado. Selecione o pino apropriado (0-7).

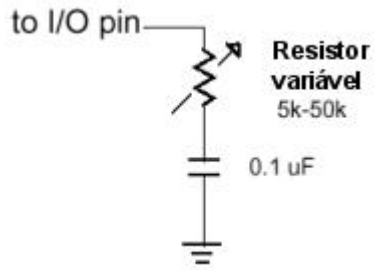
- O editor grava um pequeno programa no Basic Step (este programa irá apagar qualquer programa existente).

- Outra janela aparecerá, mostrando dois números: escala e valor. Ajuste o resistor para o menor o menor número possível seja mostrado em escala (assumindo que você pode facilmente ajustar o resistor, como num potenciômetro).

Uma vez que você tenha achado o menor valor, está pronto. Este número deve ser usado para scale na instrução Pot.

Programa exemplo:

```
abc:
POT 0,100,b2       ' Read potentiometer on pin 0.
serout 1,N300,(b2) ' Send potentiometer reading
                   ' over serial output.
goto abc           ' Repeat the process.
```



COMANDO

PROCURA alvo,(valor0,valor1,...valorN), variável

LOOKDOWN target,(value0,value1,...valueN),variable

●Procura entre value0,value1, etc um que seja igual a target. Se encontrar um igual, o valor á armazenado em variable. Se não for encontrado, variable não é alterado.

►**Target** é a variável/constante procurada.

►**Value0, value1**,... é uma lista de valores. O valor de target é comparado com estes valores.

►**Variable** armazena o resultado da procura.

A habilidade do comando Lookdown em converter uma sequencia arbitrária de valores em uma sequência ordenada (0,1,2,...) faz dele um parceiro perfeito para o comando Branch. Usando Lookdown e Branch juntos, você pode criar uma instrução SELECT...CASE

Programa Exemplo:

```
' Program receives the following one-letter instructions over a serial
' linkand takes action: (G)o, (S)top, (L)ow, (M)edium, (H)igh.

Get_cmd: serin 0,N2400,b2    ' Put input value into b2.
LOOKDOWN b2,("GSLMH"),b2     ' If b2="G" then b2=0 (see note)

' If b2="S" then b2=1
' If b2="L" then b2=2
' If b2="M" then b2=3
' If b2="H" then b2=4

branch b2,(go,stop,low,med,hi) ' If b2=0 then go

' If b2=1 then stop
' If b2=2 then low
' If b2=3 then med
' If b2=3 then hi

goto Get_cmd ' Not in range; try again.
go: ... ' Destinations of the
stop: ... ' Branch instruction.
low: ...
med: ...
hi: ...

' Note: In PBASIC instructions, including EEPROM, Serout, Lookup and
' Lookdown, strings may be formatted several ways. The Lookdown command
' above could also have been written:
' LOOKDOWN b2,(71,83,76,77,72),b2 ' ASCII codes for "G","S","L"...
' or
' LOOKDOWN b2,("G","S","L","M","H"),b2
```

COMANDO

PULA desvio(endereço0, endereço1 ..)

BRANCH offset,(address0,address1,...addressN)

●Vai para o endereço especificado em offset (se estiver dentro da faixa)

►**Offset** é uma variável/constante que especifica o endereço para desviar (0-N).

►**Addresses** são labels que especificam para onde desviar.

O comando Branch funciona como o comando ON x GOTO encontrado em outros BASICs.É útil quando você deseja alguma coisa como isto:

```
if b2 = 0 then case_0 ' b2=0: go to label "case_0"
if b2 = 1 then case_1 ' b2=1: go to label "case_1"
if b2 = 2 then case_2 ' b2=2: go to label "case_2"
```

Você pode usar o BRANCH para organizar isto em um comando único.

```
BRANCH b2,(case_0,case_1,case_2)
```

Isto funciona exatamente como o exemplo IF...THEN. Se o valor não estiver dentro da faixa (neste caso se b2 é maior que 2), BRANCH não faz nada. O programa continua com a instrução seguinte ao BRANCH.

BRANCH pode ser combinado com a instrução LOOKDOWN para criar uma instrução SELECT CASE simples. Veja LOOKDOWN para um exemplo.

Programa Exemplo:

```
Get_code:
serin 0,N2400,("code"),b2          ' Get serial input.
                                   ' Wait for the string "code",
                                   ' then put next value into b2.
BRANCH b2,(case_0,case_1,case_2)   ' If b2=0 then case_0
                                   ' If b2=1 then case_1
```

```
                                    ' If b2=2 then case_2
goto Get_code                       ' If b2>2 then Get_code.
case_0: ...                         ' Destinations of the
case_1: ...                         ' Branch instruction.
case_2: ...
```

COMANDO

REPITA variável = início ATÉ fim {PASSO {-} incremento}...CONTINUE {variável}

FOR variable = start TO end {STEP {-} increment}...NEXT {variable}

●Começa um loop For...Next. Variable é setada com o valor de start. O código entre For e Next é então executado. Variable é então incrementado/decrementado de increment (se o valor de increment não é dado, a variável é incrementada de 1). Se variable não alcançou nem ultrapassou o valor de end, as instruções entre For e Next são executadas novamente. Se variable é igual ou maior que end, então o programa continua com a instrução seguinte ao Next. O loop é executado pelo menos uma vez, não importando quais valores são dados para start e end.

Seu programa pode ter quantos For...Next forem necessários, mas eles não podem estar aninhados em mais de 8 níveis (em outras palavras, seu programa não pode ter mais de 8 loops dentro de outros loops).

►**variable** é uma variável do tipo bit, byte ou word e é usada como contadora interna. Start e end são limitadas pelo tamanho de variable (variáveis bit podem contar de 0 a 1, variáveis byte de 0 a 255 e variáveis word de 0 a 65535).

►**Start** é uma variável/constante que especifica o valor inicial de variable.

►**End** é uma variável/constante que especifica o valor final de variable.

►**Increment** é uma variável/constante opcional pelo qual o contador é incrementado ou decrementado (se negativo). Se o valor de step não é dado, variable é incrementado de 1.

►**variable** (após o Next) é opcional. Ele é usado para identificar a qual de uma séria de For...Next um Next particular pertence.

Programa exemplo:

```
Programmers most often use For...Next loops to repeat an action a
fixed number of times, like this:
FOR b2 = 1 to 10  ' Repeat 10 times.
debug b2          ' Show b2 in debug window.
NEXT              ' Again until b2>10.
```

Não subestime o fato de que todos os parâmetros de For...Next podem ser variáveis. O seu programa pode não apenas estabelecer estes valores, mas também modificá-los enquanto o loop está sendo executado. Aqui está um exemplo onde o valor de step aumenta a cada loop.

```
let b3 = 1
FOR b2 = 1 to 100 STEP b3   ' Each loop, add b3 to b2.
debug b2                    ' Show b2 in debug window.
let b3 = b3+2               ' Increment b3.
NEXT                        ' Again until b2>15.
```

Se você rodar este programa, vai notar alguma coisa familiar com os números na janela de debug (1,4,9,16,25,36,49 ...). eles são todos números elevado ao quadrado (1^2=1, 2^2=4, 3^2=9, 4^2=16, etc.), mas o nosso programa utiliza somente adições, e não multiplicação, para calculá-los. Este método de cálculo é creditado a Sir Isaac Newton.

Existe um sério problema na estrutura do For...Next. PBASIC utiliza matemática de 16 bits para incrementar/decrementar a variável contadora e comparar com o valor de end. O valor máximo

de uma variável de 16 bits é 65535. Se você adiciona 1 a 65535, obtém 0 (o registrador de 16 bits estoura e retorna a 0, como o odômetro de um carro quando ultrapassa o máximo que ele pode indicar).

Se você escreve um loop For...Next no qual o valor de step é maior que a diferença entre o valor de end e 65535, este estouro vai fazer com que o loop execute mais do que o esperado. Tente o seguinte:

```
FOR w1 = 0 to 65500 STEP 3000  ' Each loop add 3000 to w1.
debug w1                       ' Show w1 in debug window.
NEXT                           ' Again until w1>65500.
```

O valor de w1 aumente de 3000 a cada interação do loop. Quando ele se aproxima do valor final, uma coisa interessante acontece: 57000, 60000, 63000, 464, 3464... Ele ultrapassou o valor de end e continuou executando. Isto é porque 63000+3000 excede a capacidade máxima de um registrador de 16 bits. Quando o valor estoura e vai para 464 ele passa no teste "w1>65500" usado pelo comando Next para determinar quando terminar o loop.

O mesmo problema pode acontecer quando o valor de step for negativo a maior (em valor absoluto) que a diferença entre end e 0.

COMANDO

RETORNA

RETURN

●Retorna de uma subrotina. O comando return desvia a execução do programa para o endereço seguinte ao comando Gosub mais recente, a execução do programa continua daquele ponto.

O comando Return não eceita parâmetros. Para mais informações sobre como usar subrotinas, veja o comando Gosub.

Programa exemplo:

```
for b4 = 0 to 10
gosub abc          ' Save return address and then branch to abc.
next
abc:
pulsout 0,b4       ' Output a pulse on pin 0.
                   ' Pulse length is b4 x 10 microseconds.
toggle 1           ' Toggle pin 1.
RETURN             ' Return to instruction following gosub.
```

COMANDO

SAIDA pino

OUTPUT pin

●Faz o pino selecionado funcionar como saída.

►**Pin** é uma variável/constante (0-7) que especifica qual pino utilizar.

Quando o programa começa, todos os pinos são entrada. Se você deseja aticar uma carga, como um led ou um circuito lógico, você precisa configurar o pino correspondente como saída.

As instruções de saída (High, Low, Pulsout, Serout, Sound e Toggle) mudam automaticamente o pino selecionado para saída e o deixam neste estado. Embora tecnicamente não seja uma instrução de saída, o comando Port também muda os pinos para saída. Escrevendo 1 em um bit particular da variável Dirs faz o pino correspondente uma saída.

Quando um oino é configurado como saída, você pode mudar o seu estado escrevendo um valor nele uo na variável Pins. Quando um pino muda para saída, ele pode estar no estado 0 ou 1, dependendo do valor previamente escrito no pino. Para garantir o estado de um pino use

os comandos High ou Low para mudá-lo para saída ou escreva o estado desejado antes de mudá-lo para saída.

Programa exemplo:

```
' To see this program in action, connect a 10k resistor from pin 7 to the +5
' power-supply rail. When the program runs, a debug window will show you the
' the state at pin 7 (a 1, due to the +5 connection); the effect of writing
' to an input pin (none); and the result of changing an input pin to output
' (the latched state appears on the pin and may be read by your program).
' Finally, the program will show how changing pin 7 to output wrote
' a 1 to the corresponding bit of the variable Dirs.

input 7                                 ' Make pin 7 an input.
debug "State present at pin 7: ",#pin7,cr,cr
let pin7 = 0                            ' Write 0 to output latch.
debug "After 0 written to input: ",#pin7,cr,cr
OUTPUT 7                                ' Make pin 7 an output.
debug "After pin 7 changed to output: ",#pin7,cr
debug "Dirs (binary): ",#%dirs
```

COMANDO

SE variável ?? valor {AND/OR variável ?? valor} ENTÃO endereço

IF variable ?? value {AND/OR variable ?? value...} THEN address

●Compara variable com value e desvia se o resultado for verdadeiro.

►**??** é um dos seguintes operadores; =(igual), <> (diferente), > (maior), < (menor), >= (maior ou igual), <= (menor ou igual)

►**variable** é a variável que será comparada.

►**value** é uma variável/constante para comparação.

►**Address** é um label que especifica para onde desviar se o resultado da comparação for verdadeiro.

Diferentemente de outros tipos de BASIC, este comando If...Then, somente pode ir para determinado endereço. Ele não suporta comandos do tipo "IF x>30 THEN x=0". Para executar o mesmo comando em PBASIC faça:

```
IF x <= 30 THEN skip    ' If x is less than or equal
let x = 0               ' to 30, don't make x=0.
skip: ...               ' Program continues.
```

A menos que x > 30, o programa pula a instrução"let x = 0".

O If...Then do PBASIC pode efetuar duas ou mais comparações ao mesmo tempo com o auxílio de And e Or. Ele trabalha da esquerda para a direita e não aceita parênteses para mudar a ordem da comparação. Pode ser difícil entender uma comparação composta. Sugerimos que faça um teste da lógica da comparação usando o comando Debug como mostrado no programa a seguir.

Programa exemplo:

```
' Evaluates the If...Then statement and displays the result in a debug window.
let b2 = 7                      ' Assign values.
let b3 = 16
IF b3 < b2 OR b2 = 7 THEN True ' B3 is not less than b2, but
                                ' b2 is 7: so statement is true.
debug "statment is false"       ' If statement is false, goto here.
end
True:
debug "statement is true"       ' If statement is true, goto here.
End
```

COMANDO

SOM pino,(nota,duração,nota,duração,...)

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

●Muda o pino especificado para saída, gera um trem de ondas quadradas com a duração dada. O pino de saída deve ser conectado como mostrado no diagrama. Você pode substituir o capacitor por um resistor de 220 ohms, mas o autofalante vai consumir mais energia.

►**Pin** é uma variável/constante (0-7) que especifica o pino a usar.

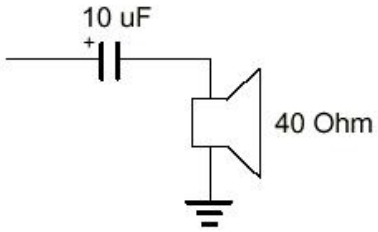
►**Note(s)** é uma variável/constante (0-255) que especifica a freqüência da nota. O valor 0 representa silêncio pela duração especificada. Valores de 1 a 127 são sons crescentes. De 128 a 255 são ruído branco crescentes.

►**Duration(s)** é uma variável/constante (1-255) que determina a duração da nota (em passos de 12ms).

As notas produzidas podem variar de 94.8Hz (1) a 10.550Hz (127). Se você precisa determinar a freqüência correspondente a uma determinada nota, ou precisa achar o valor para uma determinada nota, use as equações abaixo.

Exemplo de programa:

```
for b2 = 0 to 256
SOUND 1,(25,10,b2,10) ' Generate a constant tone (25)
                      ' followed by an ascending tone
                      ' (b2). Both tones have the
next                  ' same duration(10).
```



COMANDO

TABELA índica(valor0,valor1,...valorN),variável

LOOKUP offset,(value0,value1,...valueN),variable

●Procura o dado especificado por offset e armazena em variable. Por exemplo, se os valores fossem 2, 13, 15, 28, 8 e o offset fosse 1, então variable receberia o valor "13", visto que "13" é o segundo valor da lista (o primeiro valor é #0, o segundo #1, etc). Se offset for maior que o número de valores da lista, então variable não é modificado.

►**Offset** é o índice do número a ser lido

►**Value0, value1,** ... é a tabela de valores, até 256 elementos.

►**Variable** guarda o resultado do comando.

Muitas aplicações requerem que o computador calcule um valor de saída baseado num valor de entrada. Quando a relação é simples, como out = 2*in", não há problemas. Mas e quando a relação não é tão óbvia? Em PBASIC você pode usar o comando Lookup.

Por exemplo, motores de passo funcionam de um modo estranho. Eles requerem um padrão de 1s e 0s para controle da corrente de 4 bobinas. A seqüência aparece na tabela abaixo:

Step#	Binary	Decimal
0	1010	10
1	1001	9

2 0101 5
3 0110 6

Repetindo a seqüência, faz o motor girar. O programa abaixo mostra como usar Lookup para gerar a seqüência.

Programa exemplo:

```
' Output the four-step sequence to drive a stepper motor w/on-screen simula-tion.

let dirs = %00001111            ' Set lower 4 pins to output.
Rotate:
for b2 = 0 to 3
LOOKUP b2,(10,9,5,6),b3         ' Convert offset (0-3)
                                ' to corresponding step.
let pins = b3                   ' Output the step pattern.
LOOKUP b2,("|/-\"),b3           ' Convert offset (0-3)
                                ' to "picture" for debug.
debug cls,#%pins," ",#@b3       ' Display value on pins,
next                            ' animated "motor."
goto Rotate                     ' Do it again.
' Note: In the debug line above, there are two spaces between the quotes.
```

COMANDO

VAIPARA endereço

GOTO address

●Desvia para address, e continua a execução do programa naquele ponto.

►**Address** é um label que especifica para onde desviar.

Programa exemplo:

```
abc:
pulsout 0,100    ' Generate a 1000-µs pulse on pin 0.
GOTO abc         ' Repeat forever.
```

COMANDO

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

●Determina o valor de uma variável e/ou faz uma operação lógica com a variável. toda a matemática e lógica é feita em nível de word (16 bits).

A instrução "Let" é opcional. Por exemplo, a instrução "A=10" é idêntica a "Let A=10".

►**??** é um do seguintes operadores:

+ soma
- subtração
* multiplicação (produz a word baixa do resultado)
** multiplicação (produz a word alta do resultado)
/ divisão (produz o quociente)
// divisão (produz o resto)
min mantém variable maior ou igual a value
max mantém variable menor ou igual a value
& AND lógico
| OR lógico
^ XOR lógico
&/ AND NOT lógico
|/ OR NOT lógico
^/ XOR NOT lógico

►**Variable** é a variável que vai ser atribuído o valor.

►**Value** é uma variável/constante que afeta variable.

Quando você escreve um programa que faz operações matemáticas, tenha em mente as limitações das variáveis do PBASIC; Todos os números são inteiros e positivos, bit pode representar 0 ou 1; bytes, 0 a 255 a word 0 a 65535/ O PBASIC não tem números de ponto flutuante (como 3.14), números negativos (-73), ou números maiores que 65535.

Na maioria das aplicações, estas limitações não são sérias. Por exemplo, suponha que você precise medir temperaturas de -50 a +200F. Assumindo o valor de 0 para -50 e 65535 para +200F você teria uma resolução de 0.0038!

A restrição de números inteiros não significa que você não possa fazer cálculos complexos com o Basic Step. Você apenas precisa improvisar. Suponha que você precise usar a constante pi (3.1415...) no seu programa. Você gostaria de escrever:

```
Let w0 = b2 * 3.14
```

No entanto "3.14" é um número de ponto flutuante, que o Basic Step não entende. Existe uma alternativa. Você poder expressar tais quantidades como frações. Por exemplo, o valor 1.5 é equivalente à fração 3/2. Com um pouco de esforço, você pode achar frações equivalentes para a maioria dos números de ponto flutuante. No nosso exemplo, a fração 22/7 é muito próxima do valor de pi. Para fazer a operação "Let w0 = b2* 3.14" nós podemos usar:

```
Let w0 = b2 * 22 / 7
```

PBASIC efetua as operações da esquerda para a direita. Você não pode usar parêntesis para alterar esta ordem, como em outros BASICs. E ainda, não há precedência de operadores. Muitos BASICs calculam "2+3*4" como 14, pois eles calculam "3*4" primeiro e depois adicionam o 2. Como o PBASIC calcula da esquerda para a direita, ele obtém o resultado 20, visto que ele calcula "2+3" e depois multiplica por 4. Quando estiver em dúvida, faça um exemplo e utilize o comando Debug para ver o resultado.

Programa Exemplo:

```
pot 0,100,b3    ' Read pot, store result in b3.
LET b3=b3/2     ' Divide result by 2.
b3=b3 max 100   ' Limit result to 0-100.
' Note that "LET" is not necessary.
```

NOTA:

Espero ter contribuído ou auxiliado no ensino deste pequeno notável microcontrolador brasileiro. O BASIC Step representa para o Brasil uma tecnologia nacional, um microcontrolador com suas potencialidades e facilidades de utilização. Iniciativas como a do fabricante do BASIC Step deveria se tornar tendência em nosso mercado, uma nação deterá poder financeiro quando investir em tecnologias, ou melhor, quando exportá-las.

Derli Luís Angnes