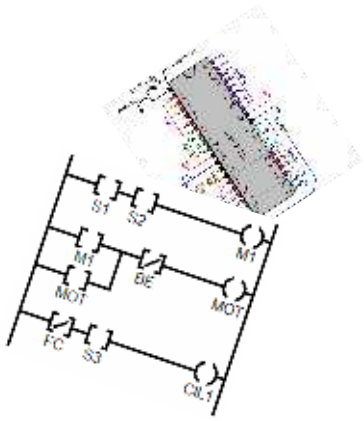
LINGUAGEM LADDER
p/ microcontroladores microchip PIC

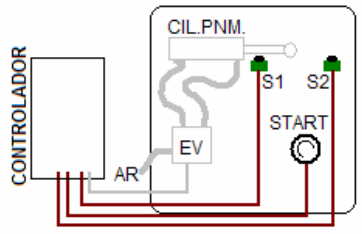
Autor: Daniel Corteletti
Centro Tecnológico de Mecatrônica SENAI



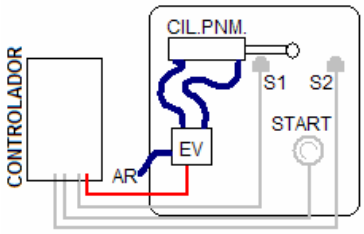
Ladder: É uma linguagem de programação gráfica, em forma de diagrama, que por ser de fácil criação e interpretação e representar ligações físicas entre componentes eletrônicos (sensores e atuadores), acaba sendo bastante utilizada em ambiente industrial.

Em um diagrama LADDER simples, podemos encontrar três tipos de elementos básicos:

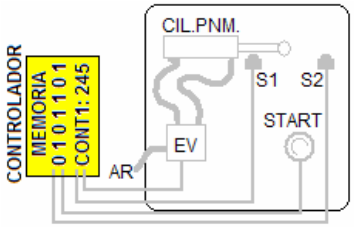
1) **CONTATO** (Contact): É o elemento que representa o sensor, ou seja, a entrada de sinal no bloco de controle lógico. Pode ser uma chave, um sensor reflexivo, um final de curso ou até mesmo o contato de algum relé auxiliar.



2) **BOBINA** (coiL): É o elemento atuador, ou seja, o elemento acionado ou desligado pelo bloco de controle lógico. Pode ser uma contactora, um motor, uma lâmpada, um atuador auditivo, etc...

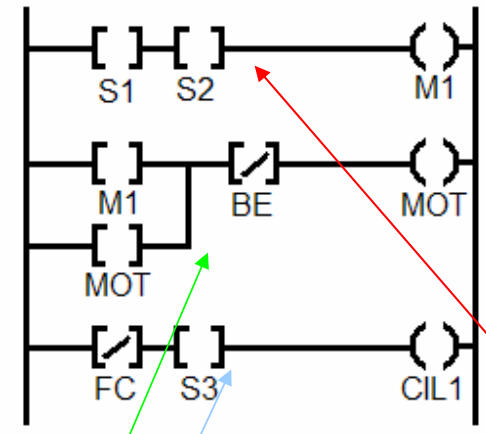


3) **MEMÓRIA** ou Relé Interno (Internal Relay): É a representação do estado de um contato ou bobina em memória, sem conexão direta com elementos externos.



Existem ainda outros elementos auxiliares que permitem realizar operações mais complexas, como temporização, contagem e manipulação de dados. Estes elementos serão discutidos na segunda parte deste tutorial.

Veja o exemplo de um diagrama LADDER:



Para este diagrama, temos o controle de 3 elementos, sendo estes M1, MOT e CIL1. Estes elemendos podem ser BOBINAS (ATUADORES) ou MEMÓRIAS (relés internos).

Os elementos S1, S2, BE, VC e S3 só aparecem ao lado esquerdo do diagrama, no formato de colchetes [ ], o que pressupõe que sejam sensores (entradas).

Na primeira linha, observamos que a regra do programa define que a saída M1 irá ativar somente se os sensores S1 e S2 estiverem AMBOS ligados.

Na segunda linha deste programa, observa-se que a regra determina que a saída MOT irá ligar se BE estiver DESLIGADO (a barra significa inversão) e se M1 ou MOT estiver acionado (ao menos um destes).

Na terceira linha, observa-se que o atuador CIL1 irá ativar caso o sensor FC estiver DESLIGADO (novamente observe a barra), e se o sensor S3 estiver acionado.

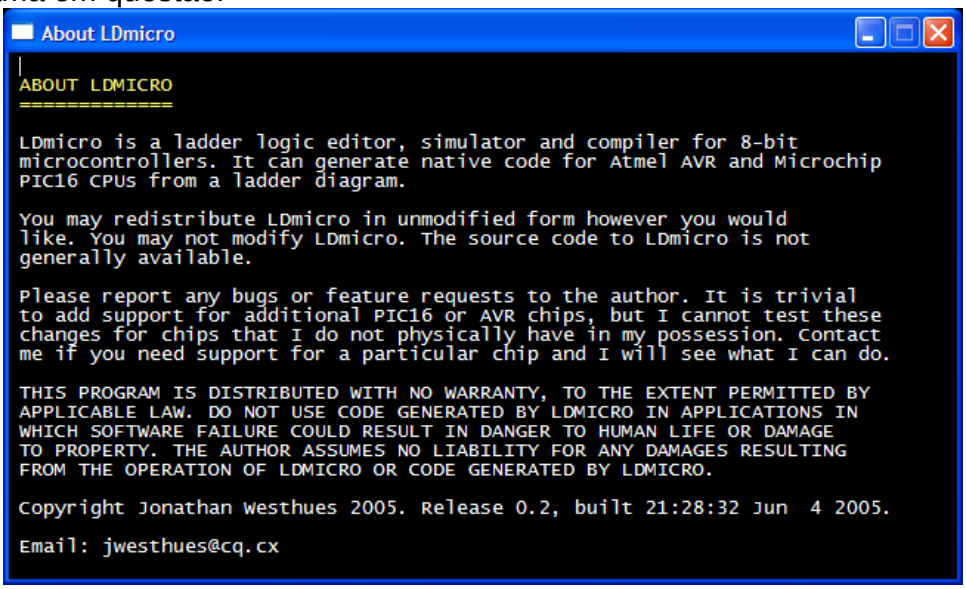
Existem ainda algumas outras regras importantes sobre programação LADDER:

1) Não é permitido (ou pelo menos recomendado) o uso de uma mesma bobina (saída) em mais de uma linha, pois as regras irão conflitar. Por exemplo, não poderíamos inserir no diagrama anteriormente representado mais uma linha que acionasse o atuador CIL1.
2) Existe a possibilidade, em algumas variações da linguagem, do uso do comando SET e RESET (liga e desliga) que determina em que momento um determinado atuador irá ligar ou desligar.
3) Existem blocos especiais que permitem temporizar, detectar pulso, borda, contagem e outros recursos. Isso pode variar conforme a linguagem utilizada.

LADDER PARA MICROCONTROLADOR PIC – O LDMICRO

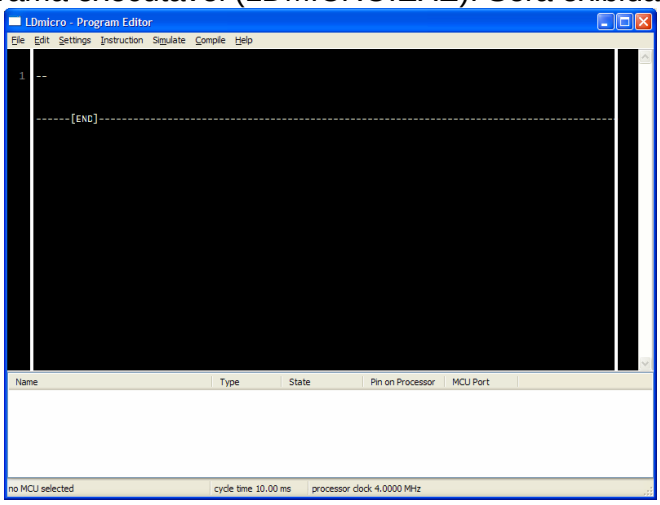
A linguagem LADDER nasceu na necessidade de facilitar a programação em ambientes industriais, remetendo para uma linguagem de alto nível e fácil de ser utilizada. No entanto existe um programa, (LDMICRO) de Jonathan Westhues, que permite a programação LADDER de microcontroladores, que viabiliza o estudo e implementação de controles de baixíssimo custo.

Este software é muito versátil, não requer instalação (basta executar o arquivo ldmicro.exe em ambiente windows ou emulador compatível), e é de livre distribuição, como podemos ver na janela abaixo, extraída do próprio HELP do programa em questão:



O LDMICRO funciona da seguinte forma:

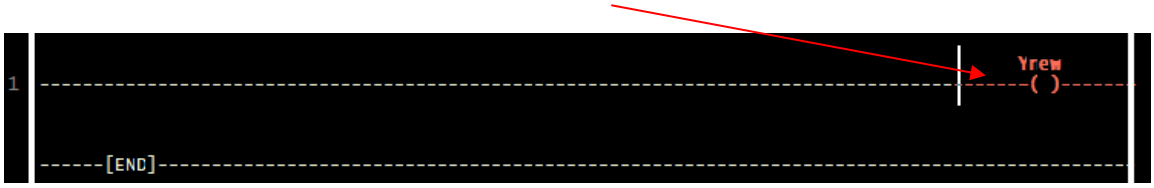
1) Inicie o programa executável (LDMICRO.EXE). Será exibida a seguinte tela:



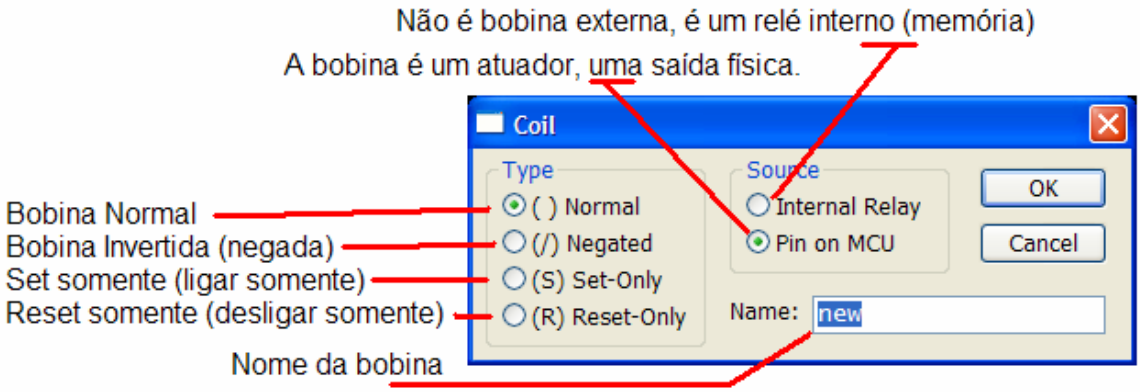
É neste ambiente que você pode gerar o programa LADDER para microcontrolador.

Para inserir uma bobina, pressione L.

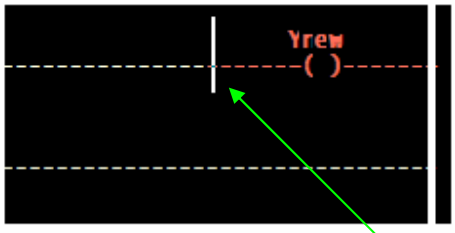
Você notará que será construída (ou complementada) a linha editada com a bobina indicada. É permitido inserir mais de uma bobina para a mesma linha.



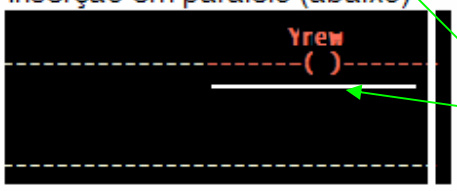
Clicando duas vezes sobre a bobina criada, será aberta a caixa de propriedade da bobina:



Inserção em série (ao lado)



Inserção em paralelo (abaixo)



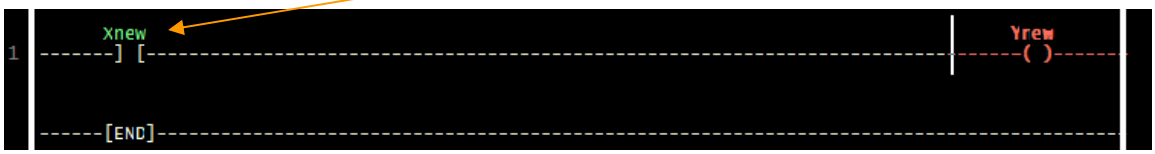
Se a bobina for definida (no campo Source) como INTERNAL RELAY, **o nome da bobina no diagrama ladder será precedido pela letra R**. Exemplo: Se o nome da bobina for new (como no exemplo acima), e se esta for definida como Internal Relay, será exibida como Rnew.

Se a bobina for definida como PIN ON MCU, o nome da bobina **será precedido pela letra Y** (no caso do exemplo, Ynew).

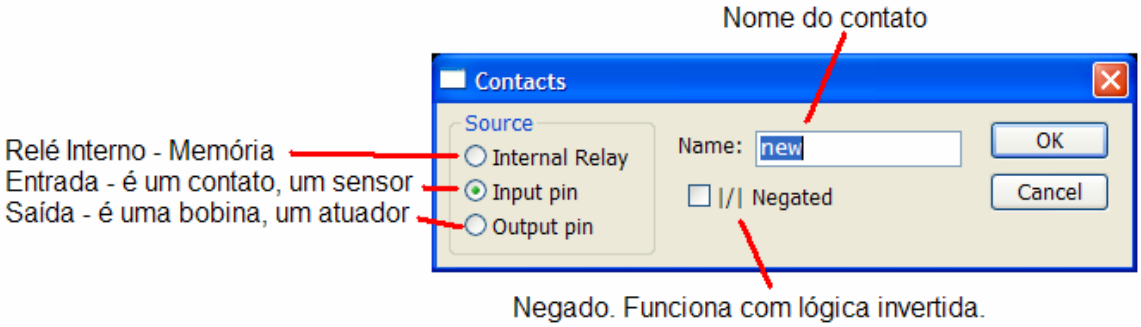
Observe que ao se inserir um contato ou bobina, será respeitada a posição do cursor (barra piscante) para definir o local da inserção. Ou seja, para inserir uma bobina ou contato abaixo de outra, posicione primeiro o cursor na posição horizontal.

Para inserir um contato:

Posicione o cursor no local desejado, e pressione C.



Note que surgirá um campo definido por colchetes --] [--- com o nome Xnew. Clique duas vezes sobre este item para abrir a caixa de propriedades do contato.



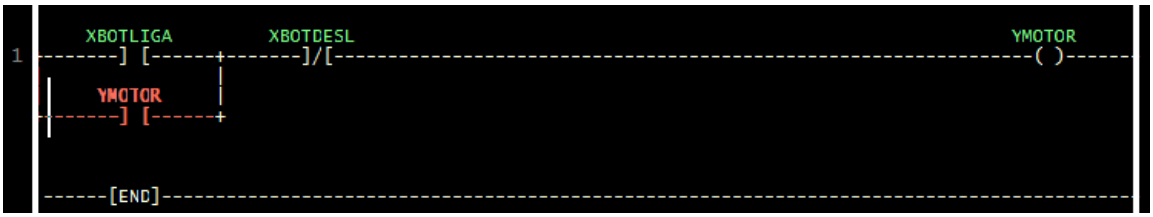
No campo source, você pode definir se o contato é um relé interno (memória). Para este caso, note que **o nome do contato será precedido pela letra R**. Se for definido como INPUT PIN (padrão), o contato é um sensor, uma entrada de sinal digital. Neste caso, **o nome do contato será precedido pela letra X** (como no exemplo acima: Xnew).

Se você desejar usar uma bobina como contato (isso é possível em ladder), basta ativar a opção **OUTPUT PIN**. Neste caso **o nome do elemento inserido será precedido pela letra Y**.

A caixa [/] define que a entrada funcionará negada (com **lógica invertida)**, ou seja, aciona zerando o contato, e desativa ligando o contato.

Prática:

Tente agora montar o seguinte diagrama LADDER usando os recursos acima citados:



Depois de editar este programa (**observe que os elementos usados são somente e exatamente XBOTLIGA, XBOTDESL, YMOTOR**). Não deve haver nenhum outro elemento no programa.

SALVANDO

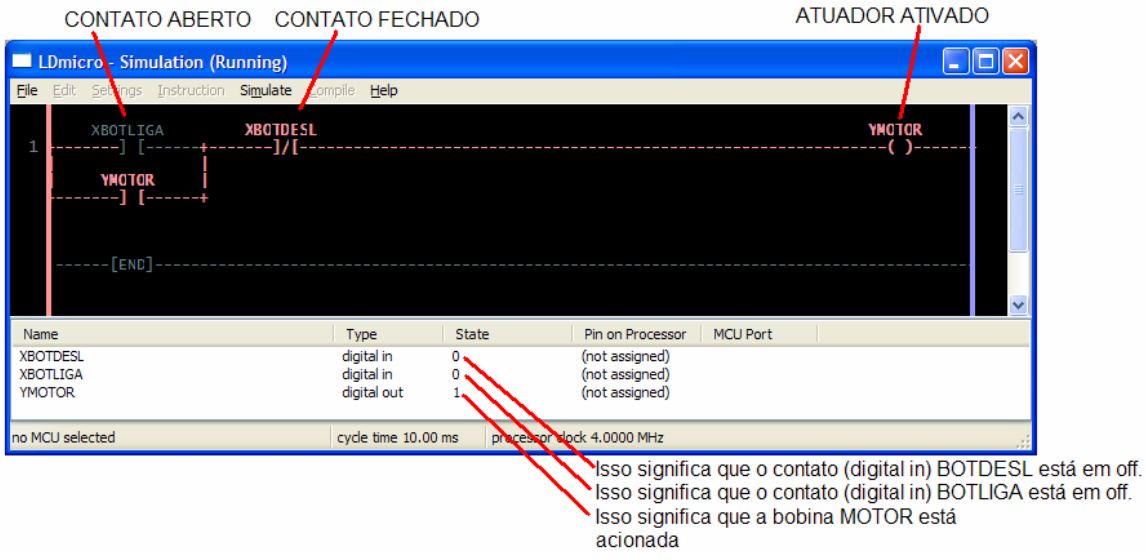
Após escrever seu programa, salve-o clicando em FILE -> SAVE AS... Salve como um arquivo com extensão LD.

SIMULANDO

Com o programa salvo, para simular o programa, clique em ***SIMULATE → SIMULATION MODE***, e posteriormente em ***SIMULATE → START REAL TIME SIMULATION***.

A partir deste momento, observe no painel da parte inferior da janela o estado dos contatos e das bobinas. **Basta dar um DUPLO CLICK** sobre o item para mudar seu estado.

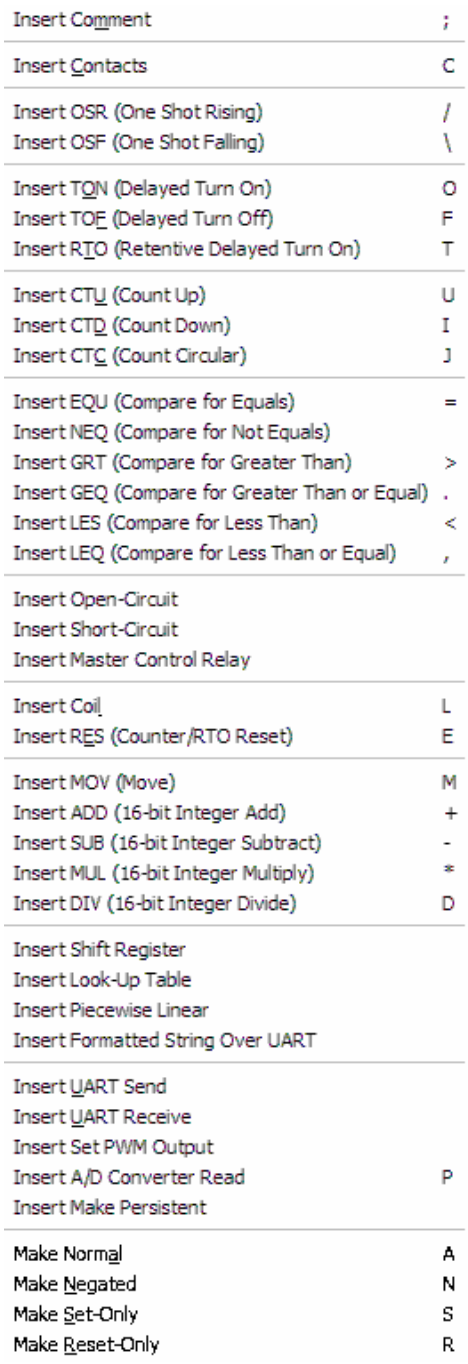
Teste alterando o estado dos sensores, e veja se o programa funciona.



COMPILANDO

Para gerar um arquivo HEX a partir deste programa, basta seguir estes passos:

1) Clique em ***SETTINGS → MICROCONTROLER*** e defina qual o microcontrolador a ser utilizado. Para melhor funcionamento, clique em ***SETTINGS → MCU PARAMETERS*** e defina o valor do cristal de clock utilizado. O padrão é 4MHz.
2) Agora de um duplo clique sobre cada elemento DIGITAL IN ou DIGITAL OUT da parte inferior da janela, **associando** cada CONTATO ou BOBINA a um pino do microcontrolador.
3) Agora clique em ***COMPILE → COMPILE AS..*** e indique o nome do arquivo a ser gerado. IMPORTANTE: Não esqueça de colocar a extensão HEX. Ex: **PROG.HEX**. Caso você não informe a extensão, ficará mais difícil achá-la depois com o programa de gravação (EPIC, ICPROG, etc...)



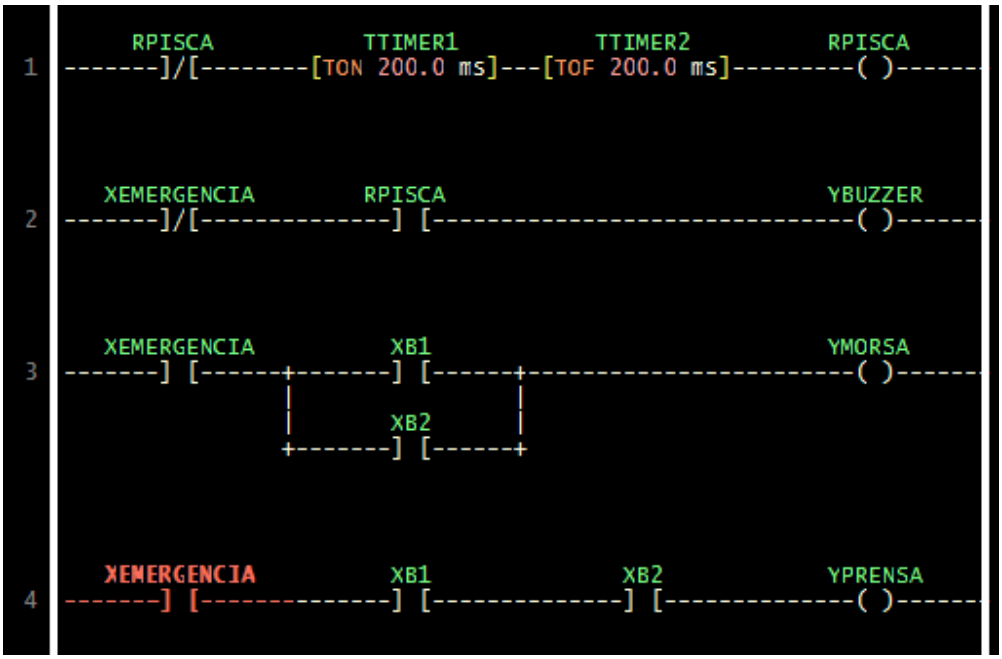
Comandos mais usados:

Inserir nova linha	shift V ou shift 6
Inserir um comentário	ponto e vírgula
Detecta borda subida	/
Detecta borda descida	\
Temporizar para desligar	F
Temporizar para ligar	O
Temporizar para ligar retentivo	T
Contador incremental	U
Contador decremental	I
Contador circular	J
Compara igualdade	=
Compara se é maior	>
Compara se é menor	<
Compara se é maior ou igual	.
Compara se é menor ou igual	,
Insere BOBINA	L
Insere Contato	C
Insere reset de contador	E
Carrega variável c/ valor	M
Insere operação soma	+
Insere operação subtração	-
Insere operação multiplic.	*
Insere operação de divisão	D
Leitura de analógico	P

Exercícios:

Tente executar os seguintes programas LADDER no microcontrolador:

1)



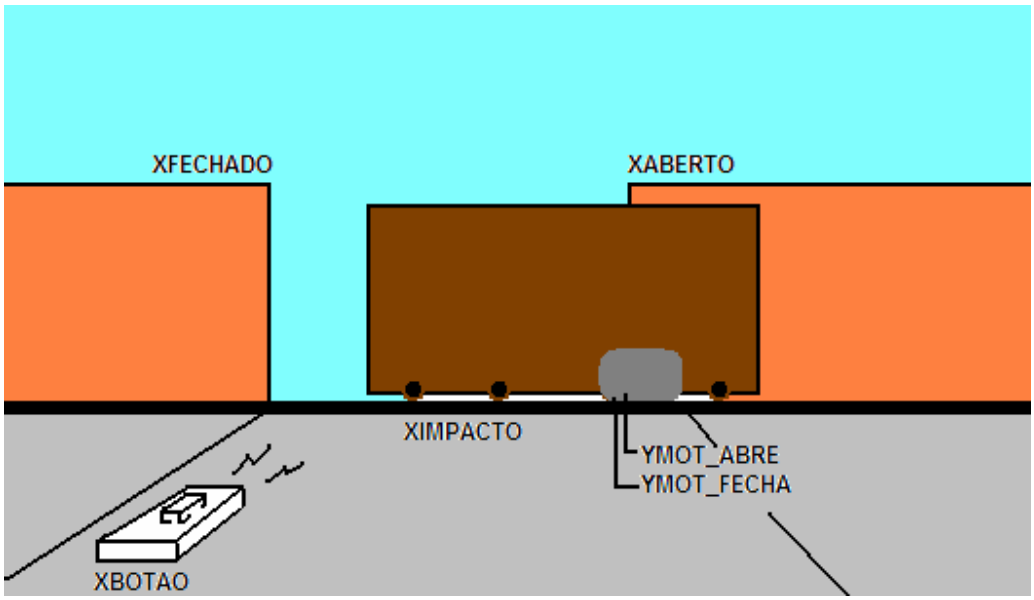
Obs: Para inserir uma linha, use SHIFT + V
Observe os nomes CORRETOS dos itens envolvidos:

XB1 : Botoeira 1 do sistema bi-manual.
XB2 : Botoeira 2 do sistema bi-manual.
XEMERGENCIA: Botão emergência. Se OK, está em 1. Pressionado em 0.
YMORSA : Atuador MORSA, que prende a peça. Liga com um bot. press.
YPRENSA : Atuador PRENSA, somente liga quando press. os 2 botões.
YBUZZER : Alerta sonoro. Deve indicar emergência acionada (em zero).
RPISCA : Relé auxiliar que ficará piscando a cada 400 ms.

2) Tente adicionar um sensor de peça no sistema acima. Caso a peça não seja detectada, a morsa não deve ligar.

3) Agora adicione também um sinal sonoro indicando se um botão foi pressionado e a peça não foi colocada.

2) Tente criar o esquema ladder para um portão de garagem. Usem os seguintes elementos:



XBOTAO : Botão do controle remoto.

XABERTO: Sensor de final de curso que determina que o portão está aberto

XFECHADO: Sensor de final de curso que determina que o portão está fechado

XIMPACTO: Sensor de impacto. Detecta que o portão colidiu em algo.

YMOT_ABRE: Motor que move o portão no sentido de abrir.

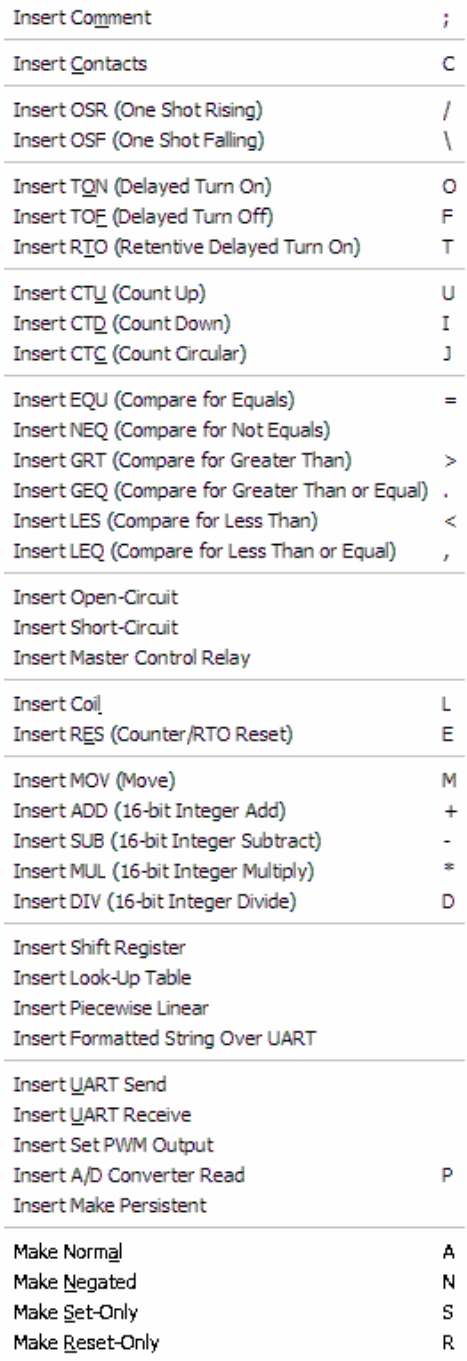
YMOT_FECHA: Motor que move o portão no sentido de fechar.

Use sua criatividade. Simule o programa no ambiente ladder, e na estação c/ microcontrolador PIC. Bom trabalho.

Para maiores informações:
Prof. Daniel Corteletti
daniel@mecatronica.org.br

PARTE 2: DESCRIÇÃO DOS COMANDOS LDmicro

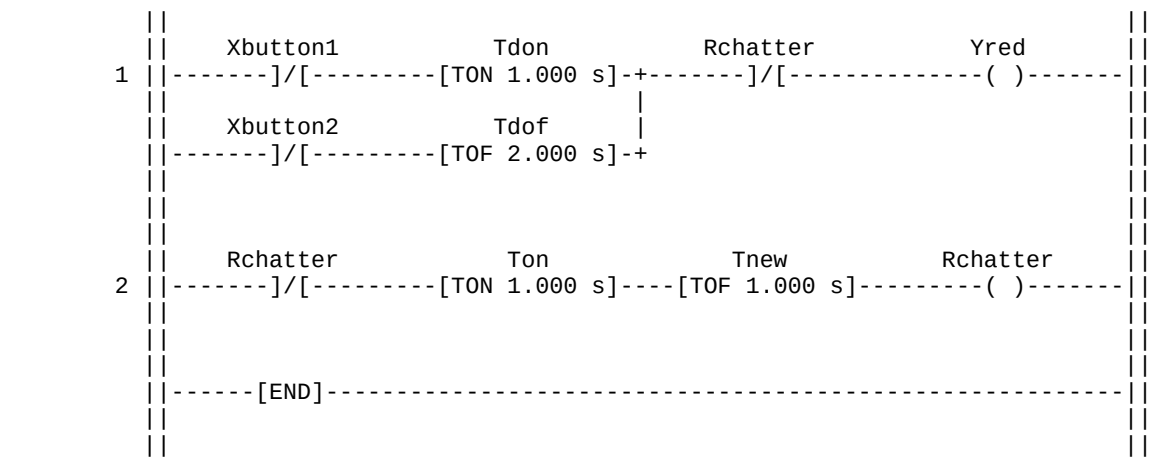
1. Inserção de comentário
2. Inserção de contato
3. Detecção de borda de subida (pulso)
4. Detecção de borda de descida (pulso)
5. Temporização para ligar
6. Temporização para desligar
7. Temporização retentiva para ligar
8. Contador incremento
9. Contador decremento
10. Contador circular
11. Comparação – igual
12. Comparação – diferente
13. Comparação – maior
14. Comparação – maior ou igual
15. Comparação – menor
16. Comparação – menor ou igual
17. Circuito aberto
18. Circuito fechado
19. Relé de controle principal (geral)
20. Inserir bobina
21. Inserir reset de contador / timer
22. Movimentação de dados (atribuição)
23. Adição (16 bits)
24. Subtração (16 bits)
25. Multiplicação (16 bits)
26. Divisão (16 bits)
27. Registrador de deslocamento
28. Tabela (look-up)
29. Tabela de valores (associação linear)
30. String formatada pela serial
31. Insere saída pela serial
32. Insere entrada pela serial
33. Ativa PWM
34. Insere leitura A/D
35. Define valor como persistente em EEPROM



Tradução do HELP do LDmicro, documento de Jonathan Westhues, realizada por Daniel Corteletti, em dezembro de 2007

Introdução
==========

LDmicro gera código nativo para certos microcontroladores Microchip PIC16 e Atmel AVR. Usualmente, o programa para estes microcontroladores é escrito em uma linguagem de programação como assembly, C ou BASIC. Um programa em uma destas linguagens compreende uma série de estruturas complexas e adequadas à arquitetura do processador em questão.
Os PLCs (controladores lógico-programáveis), por outro lado, na sua maioria são programados em ‘ladder logic’, uma simples linguagem de programação que se parece com isso:



(TON é um temporizador para ativação, e TOF é um temporizador para desacionamento. Os elementos representados por colchetes (**--] [--**) são entradas, como contatos de relés. Os elementos (**---( )---**)são saídas, e podem ser, entre outras coisas, bobinas de relés a serem acionados. Existem muitas referências sobre esta linguagem na internet, dentre as quais podemos citar:

* O programa é apresentado em um formato gráfico, e não como uma lista textual de elementos. Muitas pessoas acham muito mais fácil entender este tipo de programação, principalmente os principiantes.

* Em níveis mais básicos, o programa se parece muito com um diagrama elétrico de reles, com contatos e bobinas. O programa torna-se bastante intuitivo para programadores que conhecem teoria de circuitos elétricos.

* O compilador ladder fará os cálculos necessários para manter atualizadas as saídas. Você não terá que escrever código para determinar quando as saídas serão recalculadas e nem especificar a ordem e que estes cálculos serão executados; A ferramenta PLC fará isso por você.

LDmicro compila lógica ladder para código nativo de PIC16 e AVR. São suportados :
* PIC16F877
* PIC16F628
* PIC16F876 (não testado)
* PIC16F88 (não testado)
* PIC16F819 (não testado)
* ATmega128
* ATmega64
* ATmega162 (não testado)
* ATmega32 (não testado)
* ATmega16 (não testado)
* ATmega8 (não testado)

Poderia ser facilmente suportado uma faixa maior de microcontroladores PIC16 e AVR, mas eu não tenho como testar todos os modelos. Se você tiver um modelo em particular que deseja implementar, entre em contato comigo que verei o que posso fazer.

Usando LDmicro, você poderá escrever um diagrama ladder. Você pode sumular a lógica em tempo real no seu PC. E quando você estiver certo que o programa está correto, você pode associar os pinos do microcontrolador as entradas e saídas da lógica ladder. Após é possível gerar o código HEX e transferi-lo para a memória ROM do microcontrolador usando um programador adequado.

LDmicro é desenvolvido para ser similar à maioria dos sistemas de programação de PLCs comerciais existentes. Há algumas exceções, e algumas coisas não são padrão industrial. Leia atentamente a descrição destas instruções, e algumas serão familiares. Este documento assume que há um conhecimento básico em lógica ladder e de estruturas de um software de PLC (o ciclo de execução: ler entradas, processar, atualizar saídas)

CARACTERÍSTICAS ADICIONAIS

É também possível gerar código C ANSI. Você poderá usar com um processo envolvendo um compilador C, mas você será responsável por desenvolver o “runtime”. Isso significa que o LDmicro somente gera código para uma função PlcCycle(). Você deve implementar a chamada a função PlcCycle a cada ciclo de execução, e você deve implementar toda a parte de leitura e escrita nas entradas e saídas digitais. Veja os comentários gerados no programa fonte para maiores detalhes.

E finalmente, LDmicro pode gerar bytecode independente do processador para uma máquina virtual desenhada para rodar lógica ladder. Eu disponibilizei um exemplo de implementação de um interpretador (máquina virtual) escrita em linguagem C. Este pode então rodar em qualquer plataforma onde você consiga instalar a máquina virtual. Isso significa que você pode usar em aplicações onde você deseja usar ladder com uma “linguagem script”, para padronizar o programa. Veja os comentários nos exemplos de interpretador para detalhes.

OPÇÕES DE LINHA DE COMANDO

ldmicro.exe é geralmente executado sem opções de linha de commando. Isso signfica que você pode crier um atalho para o programa, ou simplesmente executá-lo diretamente da área de trabalho ou pasta onde está salvo.

Se o LDmicro for executado em linha de comando, passando-se como parâmetro o nome de um arquivo ld, (como ‘ldmicro.exe asd.ld'), então o programa será iniciado abrindo o arquivo indicado (no caso `asd.ld’). Se não existir, será será exibida uma mensagem de erro. Isso significa que você pode associar a extensão ld ao LDmicro, e assim abrir LD os arquivos diretamente ao clicar sobre eles.

Se o LDmicro é chamado passando-se argumentos pela linha de comando, como ‘ldmicro.exe /c src.ld dest.hex', ele irá tentar compilar o arquivo `src.ld` e gerar o arquivo `dest.hex`. LDmicro irá sair após a compilação, e possíveis mensagens de erro serão exibidas via linha de comando. Isso é útil quando se deseja usar o LDmicro como compilador em modo linha de comando.

PRINCIPIOS

Se você rodar LDmicro sem argumentos, ele iniciará vazio, pronto para edição de um novo programa. LDmicro não importa formatos de outras ferramentas ladder.

Se você não executar um programa existente então você iniciará a construção de um programa novo, com uma linha (degrau) vazia. Então você pode adicionar uma instrução: por exemplo, você pode adicionar um conjunto de contatos (Instruction -> Inser Contacts) denominados 'Xnew'. 'X' significa que o contato é ligado fisicamente a uma entrada do microcontrolador. Você poderá associar o pino correto posteriormente, após escolher o microcontrolador e finalizar os contatos do diagrama. A primeira letra do nome de um elemento indica que tipo de objeto se trata. Por exemplo:

* Xnome – entrada. É um contato.
* Ynome – Saída. É uma bobina.
* Rnome – 'relé interno'. É um bit de memória
* Tnome – é um timer.
* Cnome – É um contador
* Anome – é um inteiro lido do conversor A/D
* nome - é uma variável inteira de uso geral

Escolha o resto do nome de acordo com a função do objeto, tendo certeza que o nome é único no programa. Objetos com o mesmo nome referem-se ao mesmo objeto (lembrando que Xnome e Ynome não são o mesmo nome, e portanto são objetos diferentes). Por exemplo, se você tiver um erro ao ter um TON delay chamado 'Tdelay' e um TOF delay chamado 'Tdelay' no mesmo programa, haverá problema pois os elementos utilizarão o mesmo "contador" no programa. Por outro lado, às vezes isso é necessário, como no caso de haver um RTO chamado 'Tdelay' e uma instrução RES (reset) associada a mesma variável 'Tdelay' para causar o reset deste temporizador retentivo.

O nome das variáveis pode ser composto por letras, números e "underscores" (_). O nome das variáveis não podem iniciar com números, e é diferenciado maiúsculas de minúsculas.

As instruções para variáveis (MOV, ADD, EQU, etc.) podem trabalhar em variáveis com qualquer nome. Isso quer dizer que estas instruções também podem acessar contadores e acumuladores. Isso pode ser bastante útil, principalmente no caso de comparação de contadores ou temporizadores com uma determinada faixa de valores.

Variáveis são sempre inteiros de 16 bits com sinal. Isso significa que a faixa suportada é de -32768 a 32767. Você pode atribuir a estas variáveis valores constantes, caracteres ASCII ('A', 'z') sempre colocando o caractere entre aspas simples.

Na parte de baixo da tela, você poderá ver uma lista de todos os objetos do programa. Esta lista é automaticamente gerada a partir do programa. Não é necessário atualizada manualmente. Muitos objetos não precisam de nenhuma configuração. Objetos do tipo 'Xname', 'Yname' e 'Aname' sempre precisam ser associados a pinos do microcontrolador. Mas primeiro, escola o microcontrolador a ser utilizado, usando (Senttings -> Microcontroller). Então associe seus pinos de I/O clicando duas vezes sobre o item da lista.

Você pode modificar o programa inserindo ou apagando instruções. O cursor piscante no programa indica a instrução e o ponto de inserção. Se ele não estiver piscando, então pressione <TAB> ou clique em uma instrução. Quando estiver selecionada, a instrução pode ser apagada, podem ser inseridas novas instruções (à direita, esquerda) em série ou (acima e abaixo) em paralelo. Algumas operações não são permitidas. Por exemplo, não podemos inserir instruções à direita de uma bobina (coil).

O programa inicia com um único degrau (linha). Você pode adicionar mais linhas selecionando as opções "Insert Rung Before" e "Insert Rung After" no menu EDIT.

Assim que você terminar de escrever um programa, você pode testá-lo na simulação, e então gerar o arquivo HEX para transferir à memória ROM do microcontrolador.

SIMULAÇÃO

Para entrar no modo simulação, escolha SIMULATE -> SIMULATION MODE ou pressione CTRL + M. O programa irá ficar diferente em modo de simulação (outras cores, e sem cursor). As instruções ativas (energizadas) aparecerão em uma cor mais intensa, e as instruções inativas aparecerão em uma cor mais fraca. Pressionando a barra de espaço, a simulação ocorrerá em um ciclo. Caso queira realizar a simulação em modo contínuo, escolha SIMULAR -> START REAL-TIME SIMULATION ou pressione <Ctrl + R>. As linhas do programa irão ser atualizadas conforme as modificações provenientes da lógica do programa ocorrerem.

Você pode mudar o estado das entradas do programa pressionando um “duplo click” na lista na parte inferior da janela, ou dando um “duplo click” sobre o contato na janela do programa.

COMPILANDO PARA CÓDIGO NATIVO

O objetivo final do processo é gerar um arquivo HEX, que pode ser transferido para a memória do microcontrolador. Primeiramente, precisamos selecionar o tipo de microcontrolador, na opção SETTINGS -> MICROCONTROLER. Então você pode associar um pino de I/O para cada objeto. Faça isso clicando duas vezes sobre cada elemento “not assigned” da lista que há na parte inferior da tela. Surgirá uma janela de seleção contendo os pinos não alocados até o momento.

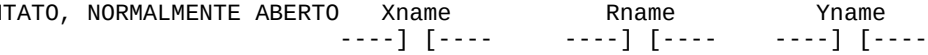
Então você deverá selecionar o tempo de ciclo desejado para o processo em questão, e também definir qual o CLOCK do sistema para que os temporizadores sejam incrementados corretamente. Geralmente o tempo de ciclo padrão de 10ms é suficiente para a maioria das aplicações. Estas opções são modificadas no menu SETTINGS -> MCU Parameters...

Agora você pode gerar código para seu programa. Escolha COMPILE -> COMPILE, ou COMPILE -> COMPILE AS... caso deseje gerar o programa HEX com outro nome. Se não houver problemas ou erros, o LDMICRO irá gerar o arquivo HEX para posterior programação do microcontrolador escolhido.

Use um programador qualquer adequado ao seu hardware para transferir o arquivo HEX para o microcontrolador. Lembre de definir os bits de configuração (fuses). Para o PIC16, os bits de configuração estarão incluídos no arquivo HEX, e muitos programas de programação vão selecionar estes padrões adequadamente de forma automática. Para processadores AVR, você deve definir os bits de configuração manualmente.

INSTRUÇÕES

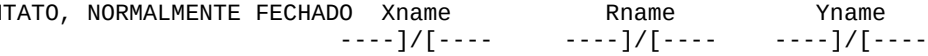
> CONTATO, NORMALMENTE ABERTO



Tecla de atalho : C
Pode ser aplicado à : Pinos de entrada, Pinos de saída e Relés internos

Se o sinal da instrução (contato) é falso, a saída de sinal também será falsa. Interprete como uma chave com conexões em contato normalmente aberto, que quando acionado fecha o circuito.

> CONTATO, NORMALMENTE FECHADO

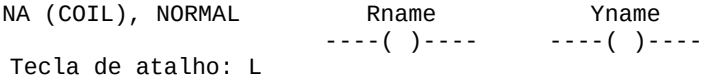


Tecla de atalho : C
Pode ser aplicado à : Pinos de entrada, Pinos de saída, Relés internos
Se o sinal entrar de entrada for falso, a saída é verdadeira. Se o sinal de entrada for verdadeiro, a saída é falsa. Entenda como uma chave com conexões em

contato normalmente fechado, que quando é acionada irá abrir o circuito, interrompendo a passagem de sinal.

Esta instrução é similar a anterior, com a diferença da opção |\|Negated ativada.

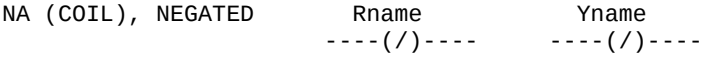
> BOBINA (COIL), NORMAL



Tecla de atalho: L
Aplica-se à: Relé interno e pinos de saída

Se o sinal vindo das instruções é verdadeiro, então a saída em questão irá ser ativada. Se o sinal é falso (circuito aberto), a saída será desativada. Esta instrução sempre estará mais a direita possível no diagrama ladder. Podem ser inseridas em paralelo, mas não em série.

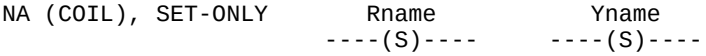
> BOBINA (COIL), NEGATED



Tecla de atalho: L
Aplica-se à: Relé interno e pinos de saída

Se o sinal que vai à instrução é verdadeiro, então a saída em questão é desativada. Caso o sinal de entrada seja falso (circuito aberto), a saída será ativada. Similar à instrução anterior com o valor invertido.

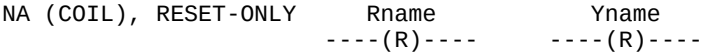
> BOBINA (COIL), SET-ONLY



Tecla de atalho: L
Aplica-se à: Relé interno e pinos de saída

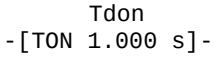
Com esta opção, estabelece-se que a saída em questão irá ter seu estado modificado para ATIVADO quando a entrada de sinal for verdadeira, e não desativará quando o sinal de entrada for falso.

> BOBINA (COIL), RESET-ONLY



Se o sinal que chega a instrução for verdadeiro, então o relé interno ou pino de saída será desativado. Caso contrário, o pino de saída ou relé interno não terá seu estado modificado. Esta instrução somente poderá mudar o estado do pino de LIGADO para DESLIGADO, e geralmente isso é utilizado em combinação com uma ação BOBINA SET-ONLY. Esta instrução deve ser a mais a direita possível.

> RETARDO NA ENERGIZAÇÃO (TEMPORIZADOR)



Quando o sinal que vai para a instrução comutar de DESLIGADO para LIGADO (borda de subida), a saída permanecerá falsa por N segundos antes de ser ativado. Quando o sinal de entrada mudar de LIGADO para DESLIGADO, a saída irá para DESLIGADO imediatamente. O temporizador é reiniciado sempre que a entrada estiver em nível DESLIGADO. O tempo (N) é configurável. No exemplo, o valor é de 1 segundo)

O nome ‘Tdon’ irá contar a partir de zero em unidades de scantimes. A instrução de saída TON irá proporcionar saída em nível alto sempre que o contador de ciclos for igual ou superior ao delay escolhido. É também possível manitular esta variável contadora ‘Tdon’, por exemplo com instruções de manipulação de dados como a instrução MOV.

> RETARDO NO DESLIGAMENTO (TURN OFF DELAY)

```
                                        Tdoff
                                    -[TOF 1.000 s]-
```

Quando o sinal vindo da instrução muda de LIGADO para DESLIGADO, o sinal de saída ainda permanecerá ativado por N segundos antes de desativar. Quando o sinal vindo da instrução mudar de DESLIGADO para LIGADO, a saída irá mudar para LIGADO imediatamente. O temporizador é reiniciado sempre que a entrada mudar para nível DESLIGADO. A entrada precisará estar desativada pelo tempo determinado para que a saída mude para estado FALSO. O tempo (N) é configurável.

A variável ‘Tdoff’ é uma variável contadora crescente que inicia em zero, e incrementa em unidades de tempo de ciclo. É possível manipular esta variável de tempo através de instruções de manipulação de dados, como a instrução MOV.

> TEMPORIZADOR RETENTIVO (RETENTIVE TIMER)

```
                                          Trto
                                      -[RTO 1.000 s]-
```

Esta instrução está associada ao tempo de duração do pulso da entrada. Se a entrada permanecer ligada por mais de N segundos, a saída será ativada. Em outro caso, a saída permanecerá falsa. A entrada precisa permanecer acionada por, no mínimo, N segundos para ativar a saída, que uma vez ativada, assim o permanecerá até que a variável ‘Trto’ em questão seja reiniciada, ou através de uma instrução manual de reset.

É possível manipular o estado da variável contadora de tempo através de instruções de manipulação de memória, como a instrução MOV ou RESET.

> RESET

```
                          Trto            Citems
                     ----{RES}----    ----{RES}----
```

Esta instrução reinicia um timer ou contador. Retardos do tipo TON e TOF são automaticamente reiniciados quando as suas entradas mudarem de estado, mas o temporizador retentivo não. O temporizador RTO, contadores CTU e CTD não são reiniciados automaticamente, e devem ser reiniciados manualmente usando a instrução RES. Quando a entrada é verdadeira, o contador ou timer associado é reiniciado. Esta instrução necessita ser a instrução mais a direita no degrau ladder produzido.

> BORDA DE SUBIDA (ONE-SHOT RISING)

```
                                          _
                                    --[OSR_/ ]--
```

Esta instrução produz, por padrão, saída DESLIGADA. Se o sinal de entrada mudar durante o processo de DESLIGADO para LIGADO a saída será ativada. Isso gera um pulso de um ciclo na saída, e pode ser usado para disparar eventos baseados na borda de subida de um sinal.

> BORDA DE DESCIDA (ONE-SHOT FALLING)

```
                                          _
                                    --[OSF \_]--
```

Esta instrução produz, por padrão, saída DESLIGADA. Se o sinal de entrada mudar durante o processo de LIGADO para DESLIGADO a saída será ativada. Isso gera um pulso de um ciclo na saída, e pode ser usado para disparar eventos baseados na borda de descida de um sinal.

> CIRCUITO FECHADO, CIRCUITO ABERTO (SHORT CIRCUIT, OPEN CIRCUIT)

```
                          ----+----+----      ----+     +----
```

A condição de saída de um CIRCUITO FECHADO é sempre igual a sua condição de entrada, e o estado do sinal da saída de um CIRCUITO ABERTO é sempre desligado. Isso pode ser útil para depurar o programa em situações de análise e teste.

> RELÉ PRINCIPAL (MASTER CONTROL RELAY)

```
                          -{MASTER RLY}-
```

Por padrão, todos os degraus tem condição de entrada LIGADA. Este relé principal realiza controle esta entrada, e pode desativar estas entradas. Dentro do ciclo do programa, podem ser adicionadas instruções para ativação e desativação parciais do MASTER RLY, sendo isso bastante útil para depuração.

> MOVE

```
                         {destvar :=  }      {Tret :=     }
                        -{ 123     MOV}-    -{ srcvar  MOV}-
```

Quando o sinal de entrada desta instrução for LIGADO, a variável de destino será carregada com o valor da variável de origem ou da constante. Se o sinal de entrada estiver DESLIGADO, nada acontece com a variável. Você pode utilizar a instrução MOVE com qualquer variável, incluindo contadores e temporizadores.

Esta instrução precisa ser a mais a direita no degrau de instruções.

> OPERAÇÃO ARITMÉTICA (ARITHMETIC OPERATION)

```
                          {ADD  kay  :=}       {SUB  Ccnt :=}
                         -{ 'a' + 10   }-     -{ Ccnt - 10  }-

                          {MUL  dest :=}       {DIV  dv :=  }
                         -{ var * -990 }-     -{ dv / -10000}-
```

Quando o sinal de entrada desta instrução é verdadeiro, a variável de destino é carregada com o resultado da expressão. Isso pode ser realizado também em variáveis de contagem e temporizadores. Estas instruções aritméticas utilizam um processamento aritmético de números 16 bits com sinal. Cada ciclo de processamento desencadeia uma operação aritmética, caso o sinal de entrada esteja verdadeiro. Se você usar esta instrução para incrementar ou decrementar uma variável, e isso estiver associado a um sinal de entrada (tipo um botão), use a instrução de borda de subida ou descida para que o incremento ou decremento ocorra somente em uma operação por pulso de entrada. A divisão é truncada, e não arredondada. 8 / 3 resulta em 2.

Esta instrução precisa ser a mais a direita no bloco (degrau) em uso.

> COMPARAÇÃO (COMPARE)

```
                        [var ==]        [var >]        [1 >=]
                       -[ var2 ]-      -[ 1   ]-      -[ Ton]-

>                       [var /=]       [-4 <   ]       [1 <=]
                       -[ var2 ]-     -[ vartwo]-     -[ Cup]-
```

Se o sinal de entrada para esta instrução é DESLIGADO, então a saída também é DESLIGADA. Se o sinal de entrada for LIGADO, então a saída será LIGADA se e somente se a instrução de comparação for verdadeira. Esta instrução pode ser usada para comparação (igual, maior que, maior e igual, diferente, menor que, menor e igual) entre variáveis, ou para comparar uma variável com uma constante numérica de 16 bits.

> (CONTADOR) COUNTER

```
                             Cname          Cname
                          --[CTU >=5]--  --[CTD >=5]--
```

Um contador incremental (CTU, conut up) ou decrementa (CTD, count down) é associado à deteção de borda de uma determinada condição de entrada. O sinal de saída desta instrução é verdadeiro caso o valor do contador tenha atingido o valor limite.

Você pode utilizar instrução de incremento (CTU) e decremento (CTD) com variáveis de mesmo nome, e a instrução RES para reiniciar o valor do mesmo.

> CONTADOR CIRCULAR (CIRCULAR COUNTER)

```
                Cname
            --{CTC 0:7}--
```

Um contador circular trabalha como um contador normal, exceto pelo fato de, após atingir o limite, ele reinicia voltando ao valor inicial. Por exemplo, o contador acima, se incrementado, contará 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, ...

Isso pode ser muito útil se combinado com estruturas condicionais na variável ‘Cname’, podendo ser criado o efeito de seqüenciador. CTC conta na borda de subida do sinal de entrada, e esta instrução deve ser a mais a direita no bloco (degrau) ladder.

> SHIFT REGISTER

```
 {SHIFT REG   }
-{ reg0..3    }-
```

Um SHIFT REGISTER (Registrador de deslocamento) é associado a um grupo de variáveis. Por exemplo, este registrador pode ser associado com as variáveis ‘reg0’, ‘reg1’, ‘reg2’ e ‘reg3’.

A cada pulso (borda de subida) na entrada desta instrução, ocorre o deslocamento (reg3 ← reg2, reg2 ← reg1, reg1 ← reg0, e reg0 permanece inalterado).

Observe que esta instrução consome uma área significativa de memória, e deve ser usada com cuidado.

Ela precisa ser a instrução mais à direita no degrau em uso.

> LOOK-UP TABLE

```
 {dest :=     }
-{ LUT[i]     }-
```

Uma tabela look-up é um conjunto de n dados ordenados. Quando a condição de entrada for LIGADO, a variável de destino ‘dest’ será carregada com o valor da variável de origem na ocorrência ‘i’. O índice (i) inicia em zero. Esta instrução deve ser a mais a direita no degrau em uso.

> PIECEWISE LINEAR TABLE

```
 {yvar :=     }
-{ PWL[xvar]  }-
```

Esta é a melhor forma de aproximar uma complicada função de transformação não linear. Pode ser útil, por exemplo, se você tiver que associar os valores lidos a uma curva de calibração baseado no valor de entrada de um sensor.

Você pode entrar com dados na tabela associando com um valor linear desejado. O valor será atribuído a variável xvar, e isso define o valor da variável yvar.

Se você atribuir a variável xvar um valor entre dois pontos lançados na tabela, o valor de yvar será computado em função da proximidade deste dois valores.

Os pontos precisam ser especificados em ordem crescente em xvar. Em alguns casos, por questões matemáticas e de limitações do processamento aritmético de 16 bits, o LDmicro poderá gerar mensagens de erro relativas a conversão. Neste caso, adicione mais dados à tabela.

Ex: Isso irá produzir um erro, se lançado estes dois pontos:

```
(x0, y0)    = (  0,   0)
(x1, y1)    = (300, 300)
```

Que pode ser corrigido desta forma:

```
(x0, y0)    = (  0,   0)
(x1, y1)    = (150, 150)
(x2, y2)    = (300, 300)
```

Em casos extremos pode ser necessário usar mais de cinco ou seis pontos. Adicionando mais pontos o código fica maior e mais lento para ser executado.

A instrução precisa ser a mais à direita do degrau em uso.

> A/D CONVERTER READ Aname
--{READ ADC}--

LDmicro pode gerar código para usar o conversor A/D embutido em certos microcontroladores. Se a condição de entrada para esta instrução é LIGADO, então será obtida uma simples amostra do conversor A/D e isso é armazenado na variável chamada ‘Aname’. Testa variável pode ser manipulada com operações genéricas, como comparação e atribuição.

Associe o pino à variável ‘Axxx’ da mesma forma que é associado um pino de entrada/saída digital, clicando duplamente na parte inferior da tela.

Para todos os microcontroladores suportados atulamentente, Entradas de 0 Volts corresponde ao valor 0, e uma entrada na tensão máxima do sistema (Vdd) corresponde ao valor 1023 (AD de 10 bits). O software não irá permitir que você associe pinos que não sejam entradas anlógicas às mesmas.

Esta instrução precisa ser a mais a direita no degrau em uso.

> SET PWM DUTY CYCLE duty_cycle
-{PWM 32.8 kHz}-

LDmicro pode gerar código para usar o periférico PWM embutido em certos microcontroladores. Se a condição de entrada desta instrução for verdadeira, então o duty cycle do periférico PWMW é definido com o valor da variável duty_cycle. O duty cycle precisa ser um número entre 0 e 100, onde 0 corresponde a “sempre desligado”, e 100 corresponde a “sempre ligado”. (Se você está familiarizado com o procedimento que os periféricos PWM utilizam, deve ter percebido que o LDmicro fará a conversão proporcional do número 0 a 100 em valores binários correspondentes para os períodos de clock necessários).

Você pode especificar a freqüência, em Hz. No entanto, a freqüência que você especificar pode não ser a exata a ser utilizada, dependendo dos divisores internos do microcontrolador e da freqüência de clock utilizada por este. Se você tentar definir um valor fora da faixa permitida, o LDmicro irá alertá-lo.

Esta instrução precisa ser a mais a direita no degrau em uso.

> MAKE PERSISTENT saved_var
--{PERSIST}--

Quando a instrução de entrada é LIGADA, isso fará com que determinada variável seja salva na EEPROM. Isso significa que o valor persistirá quando o sistema for desativado (desconectado da energia elétrica).

Não é necessário declarar o local onde a informação será gravada, isso ocorrerá de forma automática, e a variável será automaticamente carregada quanto o sistema for reiniciado.

Se o sistema abusar deste recurso, gravando muito freqüentemente na eeprom, então este recurso pode ser danificado, pois muitos sistemas garantem um limite de 100000 gravações somente.

Esta instrução precisa ser a mais a direita no degrau em uso.

> UART (SERIAL) RECEIVE var
--{UART RECV}--

LDmicro pode gerar código para usar a UART embutida em certos microcontroladores. Nos AVRs com várias UARTs, sometne a UART1 (não UART 0) será suportada. Configure a taxa de transferência (baud rate) usando a opção “Settings → MCU Parameters. Certas taxas podem não ser aceitas em certas faixas de freqüência de clock. LDmicro irá alertá-lo, neste casos.

Se a condição de entrada desta instrução for DESLIGADO, então nada irá ocorrer. Caso contrário a instrução tentará receber um simples caracter da UART. Se nenhum caracter for lido, então a condição de saída será FALSO. Se um caractere for lido, então o mesmo será armazenado na variável ‘var’ e a condição de saída da instrução será LIGADO (por um único ciclo de execução).

```
> UART (SERIAL) SEND              var
                              --{UART SEND}--
```

Se a entrada desta instrução estiver DESLIGADA, então nada acontecerá. Se a condição estiver LIGADA, a instrução irá enviar um simples caractere na UART. O valor ASCII do caractere a ser enviado deve ter sido previamente armazenado na variável ‘var’. A opção de saída do degrau será LIGADA enquanto a transmissão estiver ocorrendo, e DESLIGADO quando o processo terminar.

Lembre que o caractere toma algum tempo para ser transmitido. Verifique a condição de saída para se certificar que o processo de transmissão do primeiro caractere terminou antes de enviar um segundo.

Veja também a instrução FORMATTED STRING OVER UART, que é muito mais prática e fácil para enviar grandes trechos de dados.

```
> FORMATTED STRING OVER UART                  var
                                      -{"Pressure: \3\r\n"}-
```

Quanto esta instrução é utilizada (com sinal de entrada LIGADO), ela começa a enviar uma seqüência de caracteres (STRING) através da porta serial. Na string, onde houver a seqüência \3, será substituída pela variável em questão. O \3 significa que o valor irá tomar exatos 3 caracteres; Por exemplo, se a variável ‘var’ estiver comportando o valor 35, então a string exata que será enviada seriá: ‘Pressure: 35\r\n’ (observe o espaço extra antes do número). Se, por outro lado, a variável ‘var’ poderá assumir número de mais dígitos, como por exemplo o número 1432, então deverá ser mudada esta definição. Será necessário usar o ‘\4’.

Se a variável em questão poderá assumir valores negativos, use ‘\-3d’ ou ‘\-4d’. O dígito aparecerá somente quando os valores forem negativos. Para valores positivos, o sinal será substituído por um espaço.

Se múltiplas strings formatadas forem acionadas simultaneamente (ou se uma for acionada antes de outra terminar), ou se estas instruções estiverem sendo usadas simultaneamente com instruções UART TX, o resultado poderá ser indefinido.

Isso é permitido para que possa ser enviada uma simples string de dados, e pelo programa, disparar, em seqüência, os dados em outra linha.

Use o caractere especial ‘\\’ para exibir uma contrabarra. Abaixo a lista de caracteres especiais que podem ser usados:

* \r   -- carriage return (retorno de carro. Volta para primeira coluna)
* \n   -- newline (Nova linha.)
* \f   -- formfeed (Alimenta formulário. Geralmente usado como clear)
* \b   -- backspace (Volta um caractere)
* \xAB -- character with ASCII value 0xAB (hex) (exibir carac. especial)

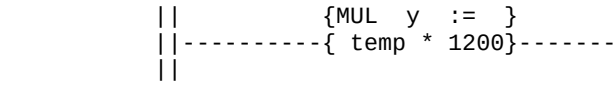
A saída desta instrução é verdadeira enquanto ela estiver transmitindo dados. Esta instrução consome muita memória de programa, e deve ser usada com cuidado. Na implementação atual do LDMicro, esta função não é muito otimizada consumindo muito recurso de processamento.

Observações ao usar recursos matemáticos

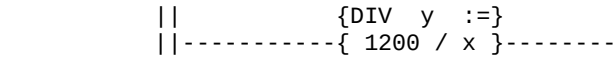
Lembre que o LDmicro realiza somente instruções matemáticas com inteiros de 16 bits. Isso significa que o resultado final de qualquer cálculo que for realizado deverá ser um valor inteiro entre -32768 e 327667. Isso também quer dizer que valores intermediários de cálculos também devem permanecer nesta faixa.

Por exmeplo, supondo que você queira calcular y = (1/x)*1200, onde x é um valor entre 1 e 20. Então y poderá valer 1200 a 60, armazenados em uma variável inteira de 16 bits. Para isso ser calculado, teoricamente há duas maneiras de se escrever o código:

```
||         {DIV  temp  :=}          ||
||---------{ 1 / x       }----------||
||                                   ||
```



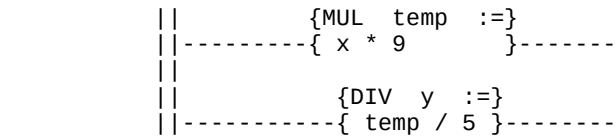
Ou pode realizar a divisão em um passo único :



Matematicamente, as duas são equivalents. Mas se você testá-las, poderá observar que a primeira sempre dará o incorreto resultado de y = 0. Isso é porque a variável ‘temp’ somente armazenará valores inteiros, e o resultado do primeiro degrau (1 / x) geralmente será um número menor que 1 (para x = 3, tempo será 0,3333...) e isso não é possível de se armazenar em uma variável inteira.

Se você estiver tendo problemas nos resultados de suas equações, verifique portanto os valores intermediários, observando se nenhum valor gerado irá resultar em dados não armazenáveis em variáveis de 16 bits com sinal.

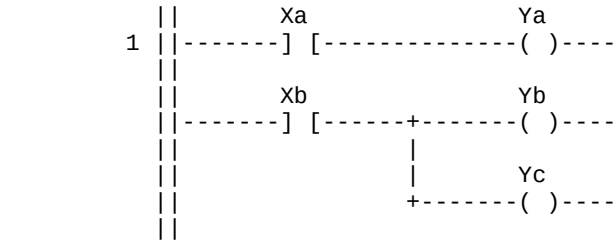
Quando você precisar multiplicar uma variável por uma fração, use isso usando respectivamente as instruções de multiplicação e divisão. Por exemplo, para multiplicar y por 1.8 * x, calcule: y = (9/5)*x (lembrando que 9/5 = 1.8). E, no código ladder, faça a multiplicação ocorrer antes da divisão.



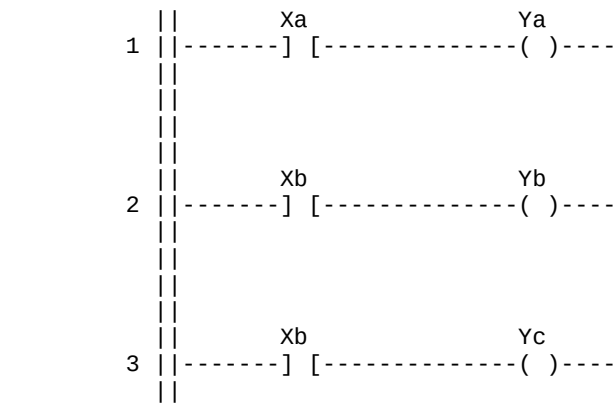
Isso funcionará para qualquer x < (32767 / 9), ou x < 3640. Para valores maiores de x, a variável ‘temp’ irá sair de seu limite.

Estilos de Codificação

É permitido múltiplas bobinas em paralelo em um simples degrau. Isso significa que você pode fazer coisas como isso:



No lugar disso:



Isso permite que, em teoria, você possa escrever programas em um único degrau gigante. Na prática isso pode não ser uma boa idéia, porque os degraus ficarão mais complexos e mais difíceis de serem editados, lidos e interpretados.

Mesmo assim, isso pode ser uma boa idéia para grupos de mesma relevância lógica, tratando-as como um simples degrau.

* * *

BUGS

LDmicro não gera um código HEX muito eficiente. É um pouco lento para ser executado e desperdiça muita memória RAM e ROM. Em contrapartida, LDmicro com um microcontrolador de porte médio (modelos PIC ou AVR) pode fazer tudo que um pequeno PLC pode, o que compensa esta baixa otimização de recursos.

O comprimento máximo do nome das variáveis é limitado.

Programas danificados (arquivo .ld) podem executar código inesperado.

Por favor, reporte bugs adicionais ou requisitos ao autor.

Agradecimentos:

* Marcelo Solano, for reporting a UI bug under Win98
* Serge V. Polubarjev, for not only noticing that RA3:0 on the PIC16F628 didn't work but also telling me how to fix it
* Maxim Ibragimov, for reporting and diagnosing major problems with the till-then-untested ATmega16 and ATmega162 targets
* Bill Kishonti, for reporting that the simulator crashed when the ladder logic program divided by zero
* Mohamed Tayae, for reporting that persistent variables were broken on the PIC16F628
* David Rothwell, for reporting several user interface bugs and a problem with the "Export as Text" function

COPIA, TERMO DE ISENÇÃO DE RISCOS e DIREITO AUTORAL

Traduzido e adaptado do texto original de **Jonathan Westhues** por Daniel Corteletti

Jonathan Westhues

Rijswijk -- Dec 2004
Waterloo ON -- Jun, Jul 2005

Cambridge MA -- Sep, Dec 2005
Feb, Mar 2006
Feb 2007