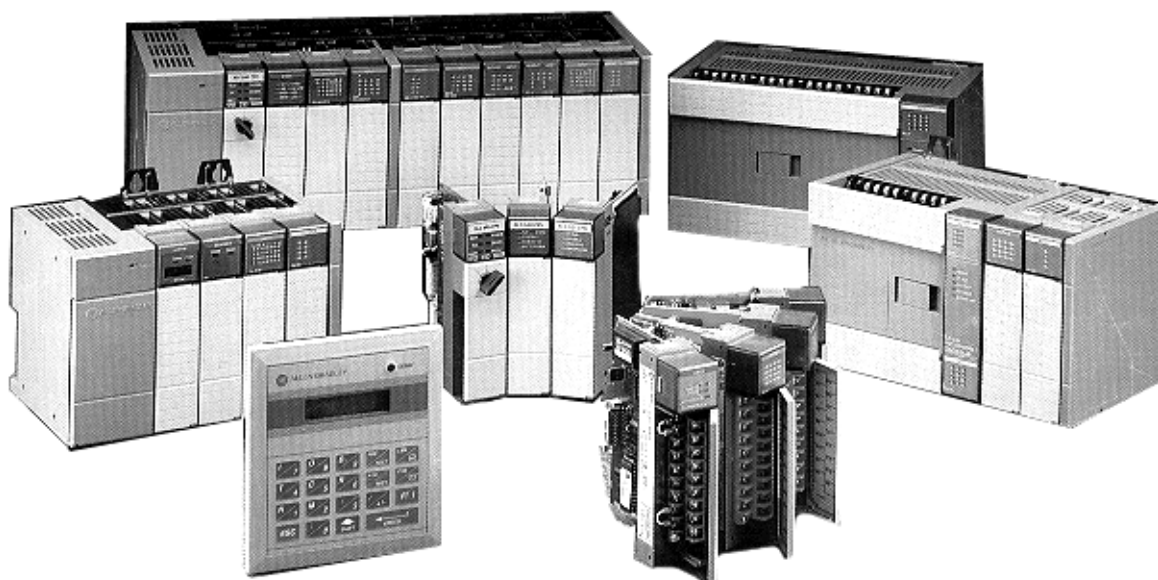


INTRODUÇÃO AOS CONTROLADORES LÓGICOS PROGRAMÁVEIS

1 Histórico dos CLPs

Segundo a NEMA (National Electrical Manufacturers Association), o Controlador Lógico programável (CLP) é definido como aparelho eletrônico digital que utiliza uma memória programável para o armazenamento interno de instruções específicas, tais como lógica, sequenciamento, temporização, contagem e aritmética, para controlar, através de módulos de entradas e saídas, vários tipos de máquinas e processos.

O desenvolvimento dos CLPs começou em 1968 em resposta a uma requisição da Divisão Hidráulica da General Motors. Naquela época, a General Motors passava dias ou semanas alterando sistemas de controles baseados em relés, sempre que mudava um modelo de carro ou introduzia modificações em uma linha de montagem. Para reduzir o alto custo de instalação decorrente destas alterações, a especificação de controle da GM necessitava de um sistema de estado sólido, com a flexibilidade de um computador, mais que pudesse ser programado e mantido por engenheiros e técnicos na fábrica. Também era preciso que suportasse o ar poluído, a vibração, o ruído elétrico e os extremos de umidade e temperatura encontrados normalmente num ambiente industrial. Abaixo alguns modelos de CLPs.



Os primeiros CLPs foram instalados em 1969, fazendo sucesso quase imediato. Funcionando como substitutos de relés, até mesmo estes primeiros CLPs eram mais confiáveis do que os sistemas baseados em relés, principalmente devido à robustez de seus componentes de estado sólido quando comparados às peças móveis dos relés eletromecânicos. Os CLPs permitiram reduzir os custos de materiais, mão-de-obra, instalação e localização de falhas ao reduzir a

necessidade de fiação e os erros associados. Os CLPs ocupavam menos espaço do que os contadores, temporizadores e outros componentes de controle anteriormente utilizados. E a possibilidade de serem reprogramados permitiu uma maior flexibilidade para trocar os esquemas de controle.

Talvez a razão principal da aceitação dos CLPs pela indústria foi que a linguagem inicial de programação era baseada nos diagramas de contato (ladder) e símbolos elétricos usados normalmente pelos eletricitistas. A maior parte do pessoal de fábrica já estava treinada em lógica ladder, adaptando-a rapidamente nos CLPs.

2 Por que usar um CLP?

“Deveríamos estar usando um controlador lógico programável?” Nos anos 70 e início dos 80, muitos engenheiros, gerentes de fábrica e projetistas de sistema de controle dedicaram grande parte de seu tempo a debater esta questão, tentando avaliar a relação custo-benefício.

Atualmente, aceita-se como regra geral que os CLPs se tornaram economicamente viáveis nos sistemas de controle que exigem mais de três relés. Considerando-se o baixo custo dos micro-CLPs e o fato dos fabricantes colocarem grande ênfase na qualidade e produtividade, a questão do custo deixa praticamente de existir. Além das reduções nos custos, os CLPs oferecem outros benefícios de valor agregado:

- ⇒ *Confiabilidade.* Depois de escrito e depurado, um programa pode ser transferido e armazenado facilmente em outros CLPs. Isto reduz o tempo de programação, minimiza a depuração e aumenta a confiabilidade. Como toda a lógica existe na memória do CLP, não existe qualquer possibilidade de cometer um erro lógico por conta de um erro de fiação. A única fiação necessária é para o fornecimento de energia para as entradas e saídas.
- ⇒ *Flexibilidade.* As modificações no programa podem ser feitas com pouca digitação. Os OEMs (fabricantes do equipamento original) podem realizar facilmente as atualizações no sistema, bastando enviar um novo programa em vez de um técnico. Os usuários finais podem modificar o programa em campo ou, por outro lado, os OEMs podem evitar que os usuários finais alterem o programa (o que é uma importante característica de segurança).
- ⇒ *Funções Avançadas.* Os CLPs podem realizar uma grande variedade de tarefas de controle, desde ações simples e repetitivas até a manipulação de dados complexos. Com a adoção dos CLPs, abrem-se muitas alternativas para os projetistas e simplifica-se o trabalho do pessoal de manutenção.
- ⇒ *Comunicações.* A comunicação com interfaces de operação, outros CLPs ou computadores facilita a coleta de dados e o intercâmbio de informações.
- ⇒ *Velocidade.* Como certas máquinas automatizadas processam milhares de itens por minuto e como os objetos são expostos aos sensores durante apenas uma fração de segundo, muitas aplicações de automação necessitam da capacidade de resposta rápida dos CLPs.

⇒ *Diagnóstico*. A capacidade de localização de falhas dos dispositivos de programação e o recurso de diagnóstico incorporado no CLP permite que os usuários localizem e corrijam rapidamente os problemas de software e hardware.

2.1 Outras Características

- ⇒ Hardware e/ou dispositivo de controle de fácil e rápida programação ou reprogramação, com a mínima interrupção na produção.
- ⇒ Capacidade de operação em ambiente industrial sem o apoio de equipamentos ou hardware específicos.
- ⇒ Sinalizadores de estado e módulos tipo plug-in de fácil manutenção e substituição.
- ⇒ Hardware ocupando espaço reduzido e apresentando baixo consumo de energia.
- ⇒ Possibilidade de monitoração do estado e operação do processo ou sistema, através da comunicação com computadores.
- ⇒ Compatibilidade com diferentes tipos de sinais de entrada e saída.
- ⇒ Capacidade de alimentar, de forma contínua ou chaveada, cargas que consomem correntes de até 2 A.
- ⇒ Hardware de controle que permite a expansão dos diversos tipos de módulos, de acordo com a necessidade.
- ⇒ Custo de compra e instalação competitivo em relação aos sistemas de controle convencionais.
- ⇒ Possibilidade de expansão da capacidade de memória.
- ⇒ Conexão com outros CLPs através de redes de comunicação

2.2 Aplicações Tradicionais

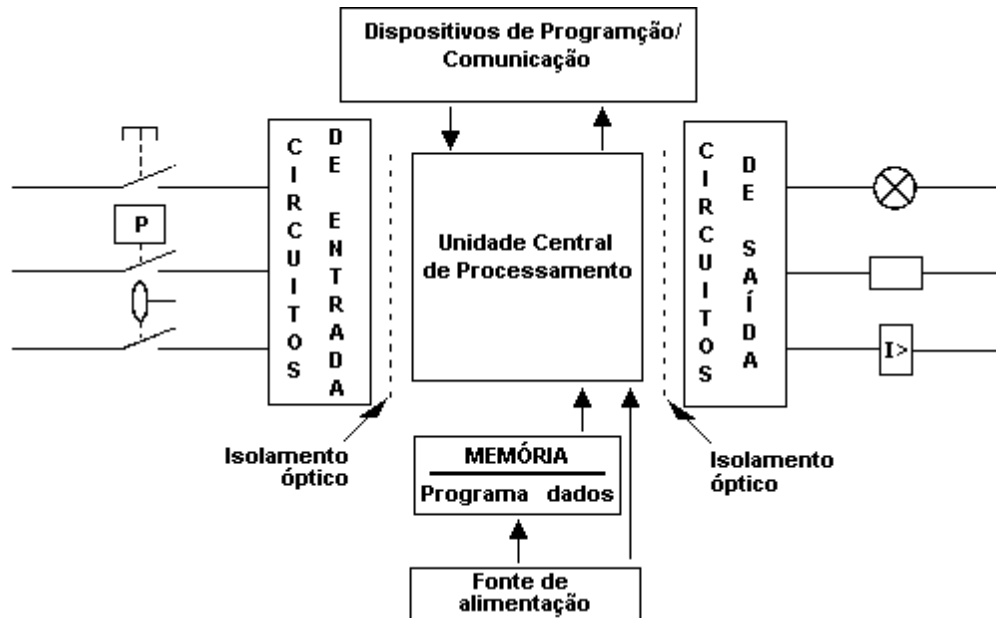
Seja qual for a aplicação, o uso do CLP permite aumentar a competitividade. Os processos que usam CLPs incluem: empacotamento, engarrafamento e enlatamento, transporte e manuseio de materiais, usinagem, geração de energia, sistemas de controle predial e de ar condicionado, sistemas de segurança, montagem automatizada, linha de pintura e tratamento de água. Os CLPs são utilizados nas mais diversas indústrias, incluindo alimentos e bebida, automotiva, química, plásticos, papel e celulose, farmacêutica e siderurgia/metalurgia. Basicamente qualquer aplicação que exija um controle elétrico pode usar um CLP.

3 Estrutura Básica de um CLP

A Estrutura básica de um controlador programável adveio do hardware básico de um computador. Podemos afirmar que um CLP é um computador para aplicações específicas.

Para entender como funciona um CLP, é necessário uma análise rápida de seus componentes. Todos os CLPs, dos micro aos grandes CLPs, usam os mesmos componentes básicos e estão estruturados de forma similar, como mostrado na figura abaixo. Os sistemas CLP consistem de :

1. Entradas
2. Saídas
3. Unidade Central de Processamento (Central Processing Unit – CPU)
4. Memória para o programa e armazenamento de dados
5. Fornecimento de alimentação
6. Dispositivo de programação



3.1 Entradas

Os terminais de entrada conectados no CLP formam a interface pela qual os dispositivos de campo são conectados ao CLP.

Os sinais recebidos por um módulo de entrada podem vir de dois tipos de sensores:

⇒ **Discretos:**

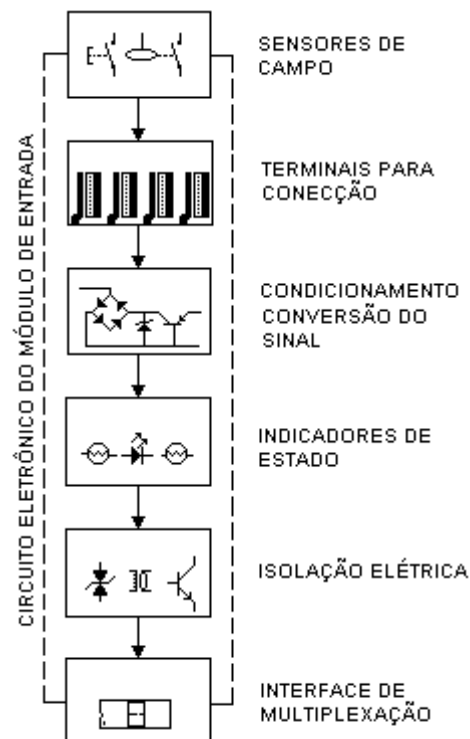
Chave limite; botoeira; chave de digitadora (thumbwheel); chave de pressão; fotocélula; contato de relé; chave seletora; teclado.

⇒ **Analógicos:**

Transdutor de pressão; transdutor de temperatura; célula de carga (strain gage); sensores de vazão; transdutores de vibração; transdutores de corrente; transdutores de vácuo; transdutores de força.

Os sinais elétricos enviados pelos dispositivos de campo ao CLP são normalmente de 120Vca ou de 24Vcc. Os circuitos de entrada no CLP recebem esta tensão vinda do campo e a “condicionam” de forma que possa ser utilizada pelo CLP. Tal condicionamento é necessário já que os componentes internos de um CLP operam a 5Vcc e devem, portanto, estar protegidos de flutuações de tensão. Para que os componentes internos fiquem eletricamente isolados dos

terminais de entrada, os CLPs empregam um isolador óptico, que usa a luz para acoplar os sinais de um dispositivo elétrico a outro.



A estrutura interna de um módulo de entrada pode ser subdividida em seis blocos principais, como mostrado na figura abaixo:

Tabela onde podemos ver a função de cada bloco:

Parte	Função
Sensores de campo	Informar ao controlador programável as condições do processo
Terminais para conexão dos sensores de campo	Permitir a interligação física entre os sensores de campo e o controlador programável.
Condicionamento e conversão do sinal de entrada	Converter os sinais de campo em níveis baixos de tensão, compatíveis com o processador utilizado.
Indicadores de estado das entradas	Proporcionar indicação visual do estado funcional das entradas contidas num módulo de entrada.
Isolação elétrica	Proporcionar isolação elétrica entre os sinais vindos do campo e os sinais do processador.
Interface/multiplexação	Informar ao processador o estado de cada variável de entrada.

Dependendo da natureza do sinal de entrada, podemos dispor dos seguintes tipos de módulos de entrada:

TIPO	CARACTERÍSTICAS
DIGITAL (AC)	12 Vac; 24 A 48 Vac; 110/127 Vac; 220/240 Vac
DIGITAL (DC)	120 Vdc com isolamento 12 Vdc; 12 a 24 Vdc com resposta rápida; 24 a 48 Vdc; 12 a 24 Vdc (lógica negativa) source; 12 a 24 Vdc (lógica positiva) sinking; 48 Vdc source; 48 Vdc sinking
ANALÓGICO	1 a 5 Vdc; 0 a 10Vdc; -10 a +10Vdc; 4 a 20mA.
ESPECIAL	TTL com suprimento; TTL com dreno; 5 a 30 Vdc selecionável; 5Vdc contador/ decodificador; 12 a 24Vdc codificador/ contador; termopar; código ASCII; código Gray; pulsos de alta velocidade.

3.2 Saídas

Os módulos de saída também são considerados como elementos de interface, pois permitem que o processador se comunique com o meio externo. A estrutura interna de um módulo de saída pode ser subdividida em sete blocos principais, relacionados a seguir:

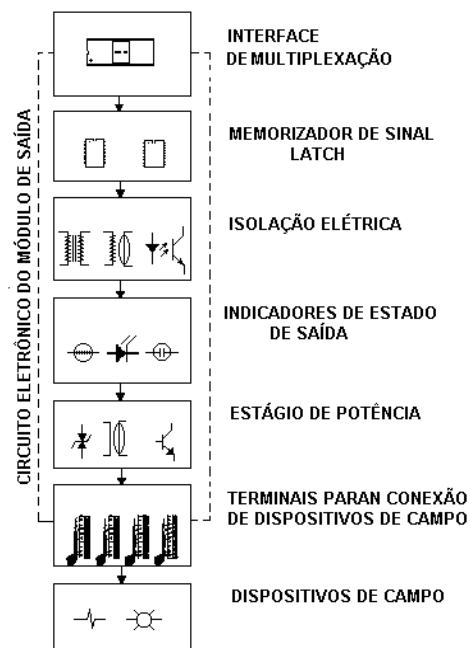


Tabela onde podemos ver a função de cada bloco:

Parte	Função
Interface/Demultiplexação	Recebe os sinais vindos do processador direcionando-os para as respectivas saídas.
Memorizador de sinal	Armazena os sinais que já foram multiplexados pelo bloco anterior.
Isolação elétrica	Proporciona isolação elétrica entre os sinais vindos do processador e os dispositivos de campo.
Indicadores de estado de saídas	Proporciona indicação visual do estado funcional das saídas contidas num módulo de saída
Estágio de Potência	Transforma os sinais lógicos de baixa potência vindos do processador em sinais de potência, capazes de operar os diversos tipos de dispositivos de campo
Terminais para conexão dos dispositivos de campo	Permite a conexão física entre CLP e os dispositivos de campo.
Dispositivos de campo	Consiste em dispositivos eletromecânicos que atuam no processo/equipamento, em função dos sinais de controle enviados pelo CP.

Dependendo da natureza dos dispositivos de campo e do tipo de sinal de controle necessário para comandá-los, podemos dispor dos seguintes tipos de módulos de saída:

TIPO	CARACTERÍSTICAS
Digital (AC)	12Vac; 24 a 48Vac; 120Vac; 220/240Vac; 120Vac com isolação.
Digital (DC)	12 a 60Vdc; 12 a 24Vdc com resposta rápida; 24 a 48Vdc; 12 a 24Vdc com suprimento; 12 a 24Vdc com dreno; 48Vdc com suprimento; 48Vdc com dreno.
Analógico	1 a 5Vdc; 0 a 10Vdc; -10 a +10Vdc; 4 a 20mA.
Especial	TTL com suprimento; TTL com dreno; 5 a 30Vdc selecionável; contato NA; contato NF; saída em ASCII; servo-motor; motor de passo.

Os circuitos de saída funcionam de maneira similar aos circuitos de entrada: os sinais emitidos pela CPU passam por uma barreira de isolamento antes de energizar os circuitos de saída.

Os CLPs utilizam vários circuitos de saída para energizar seus terminais de saída: relés, transistores e triacs.

⇒ *Relés*. Os Relés podem ser usados com alimentação alternada ou contínua. Os relés eletromagnéticos de CLPs tradicionais aceitam correntes de até alguns ampéres. Os relés suportam de forma melhor os picos de tensão porque contém uma camada de ar entre seus contatos que elimina a possibilidade de ocorrência de fuga. No entanto, são comparativamente lentos e sujeitos a desgaste com o tempo.

⇒ *Transistores*. Os transistores chaveiam corrente contínua, são silenciosos e não contém peças móveis sujeitas a desgaste. Os transistores são rápidos e podem reduzir o tempo de resposta, mas suportam cargas de, no máximo, 0,5A. Certos tipos especiais de transistores, os FETs (Transistores de Efeito de Campo) podem aceitar cargas maiores, normalmente de 1A.

⇒ *Triacs*. Os triacs chaveiam exclusivamente corrente alternada. Como os transistores, as saídas triacs são silenciosas, não tem peças móveis sujeitas a desgaste, são rápidas e transportam cargas de até 5A.

Obs. As saídas de estado sólido (triacs e transistor) podem ser danificadas e destruídas em caso de sobre tensão ou sobrecarga.

Os módulos de saída podem acionar os seguintes tipos de dispositivos de saída:

Discretos:

Controladores de motores, indicadores de painel, contator, válvula solenóide, display, bobina de relé, sistemas de alarme e segurança, sirena.

Analógicos:

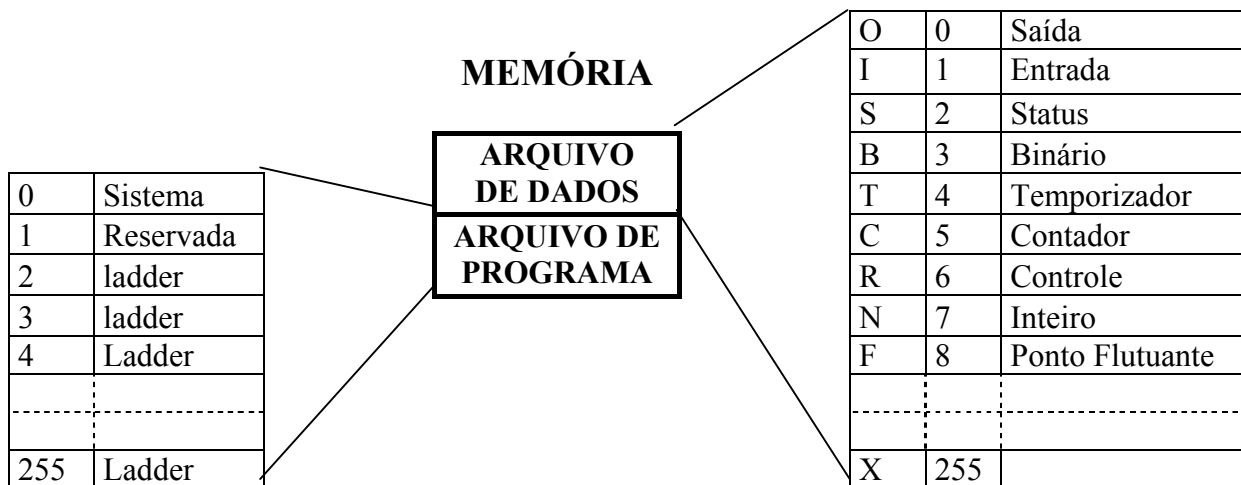
Acionadores AC, válvula de controle, acionadores DC.

3.3 Unidade Central de Processamento

A Unidade Central de Processamento (*Central Processing Unit* – CPU), formada por um microprocessador e um sistema de memória, é o principal componente do CLP. A CPU lê as entradas, executa a lógica segundo as instruções do programa de aplicação, realiza cálculos e controla as saídas, respectivamente.

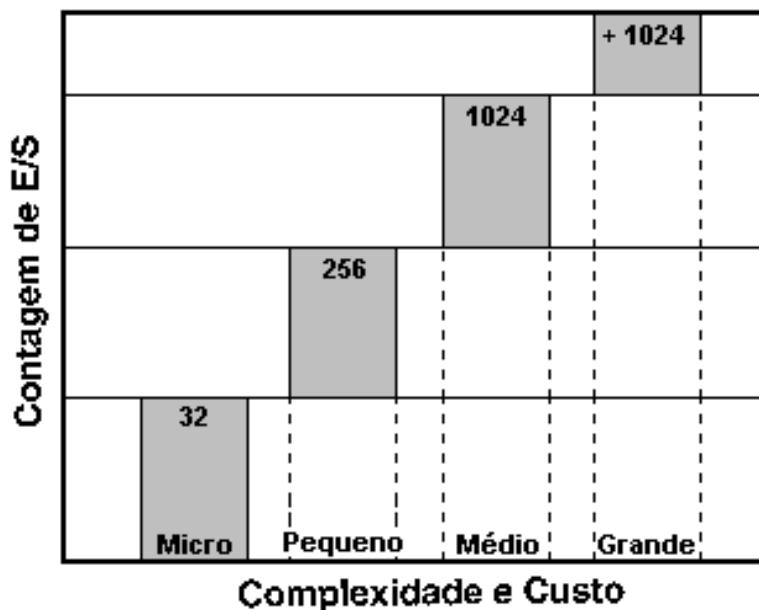
Os usuários de CLPs trabalham com duas áreas da CPU: *Arquivo de Programas e Arquivo de Dados*. Os Arquivos de Programa contêm o programa de aplicação do usuário, os arquivos de sub-rotina e as rotinas de falha. Os Arquivos de Dados armazenam dados associados com o programa, tais como status (condição) de entrada e saída, valores predefinidos e acumulados de contadores/temporizadores e outras constantes e variáveis. Juntas, estas duas áreas são chamadas de memória de aplicação ou memória do usuário. Veja a figura abaixo:

Ainda dentro da CPU encontra-se um programa executável ou *Memória do Sistema* que direciona e realiza as atividades de “operação”, tais como a execução do programa do usuário e a coordenação de varreduras das entradas e atualizações das saídas. A memória do sistema, programada pelo fabricante, não pode ser acessada pelo usuário.



4 O que caracteriza o tamanho do CLP?

Vários critérios são utilizados para classificar um CLP como micro, pequeno, médio ou grande, entre eles: funcionalidade, número de entradas e saídas, custo e dimensões físicas.



Os CLPs podem ser de *Estrutura Fixa* ou *Estrutura Modular*.

- ⇒ *Estrutura Fixa*. São unidades que já incluem o processador, a fonte de alimentação e as E/S reunidas em um só bloco.
- ⇒ *Estrutura Modular*. É aquele que tem componentes separados, porém interligados e podem ser expandidos com o acréscimo de mais módulos de E/S no chassi.

4.1 Método de Processamento

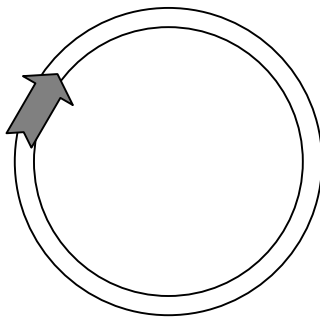
O processamento do programa do usuário de um CLP é geralmente um **processamento cíclico**.

⇒ Processamento cíclico

É a forma mais comum de execução que predomina em todas as CPU's conhecidas, e de onde vem o conceito de varredura, ou seja, as instruções de programa contidas na memória, são lidas uma após a outra sequencialmente do início ao fim, daí retornando ao início ciclicamente.

Um dado importante de uma CPU é o seu tempo de ciclo, ou seja, o tempo gasto para a execução de uma varredura. Este tempo está relacionado com o tamanho do programa do usuário (em média 1ms a cada 1.000 instruções de programa)

Para verificação do funcionamento da CPU, é estipulado um tempo de processamento, cabendo a um circuito chamado de *Watch Dog Timer*, supervisioná-lo. Ocorrendo a ultrapassagem deste tempo máximo, o funcionamento da CPU, será interrompido, sendo assumido um estado de erro.



Ciclo normal de um programa

O termo varredura ou scan, são usados para dar um nome a um **ciclo completo de operação** (*loop*). O tempo gasto para a execução do ciclo completo é chamado Tempo de Varredura, e depende do tamanho do programa do usuário, e a quantidade de pontos de entrada e saída.

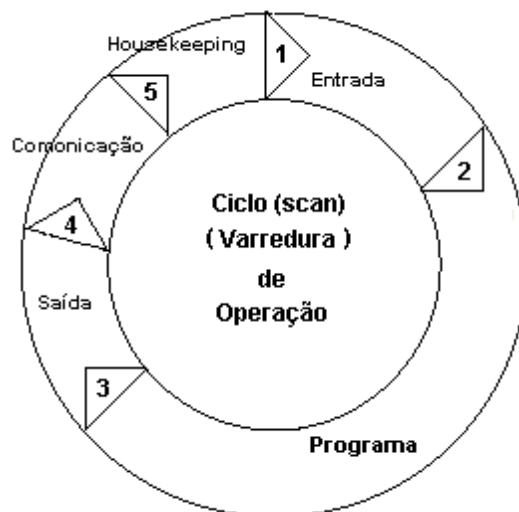
4.2 Ciclo de Operação

Todos componentes do sistema CLP são utilizados durante o ciclo de operação, que consiste de uma série de operações realizada de forma sequencial e repetida. Os elementos principais de um ciclo de operação são representados na figura abaixo.

1. *Varredura das entradas*. É o tempo necessário para que o controlador varra e leia todos os dados de entrada, isto é, examine os dispositivos externos de entrada quanto à presença ou ausência de tensão. O estado das entradas é armazenado temporariamente em uma região da memória denominada “tabela imagem de entrada”.
2. *Varredura do Programa*. É o tempo necessário para que o controlador execute as instruções do programa. Durante a varredura do programa, o CLP examina as instruções no programa ladder, usa o estado das entradas armazenado na tabela imagem de entrada e determina se

uma saída será ou não energizada. O estado resultante das saídas é armazenado em uma região da memória denominada “tabela imagem de saída”.

3. *Varredura das saídas.* É o tempo necessário para que o controlador varra e escreva todos os dados de saída. Baseado nos dados da tabela imagem de saída, o CLP energiza ou desenergiza seus circuito de saída que exercem controle sobre dispositivos externos.
4. *Comunicação.* É o momento do ciclo de operação no qual a comunicação se realiza com outros dispositivos, tais como um terminal portátil de programação, um computador, entre CLPs através de uma rede.
5. *Housekeeping / overhead.* É o tempo gasto no gerenciamento da memória e na atualização dos temporizadores e registros internos.



4.3 Tipos de Memória de Aplicação

Como o nome indica, os controladores lógicos programáveis têm memórias programáveis que permitem aos usuários desenvolver e modificar programas de controle. A memória é um espaço físico dentro da CPU onde podem ser armazenados e manipulados os arquivos de programas e arquivos de dados.

Há duas categorias de memória: *volátil* e *não-volátil*. A memória volátil pode ser alterada ou apagada facilmente, podendo-se ainda gravar ou ler tal memória. No entanto, em caso de falha de alimentação, o conteúdo programado poderá ser perdido, sendo necessário, portanto, ter um backup do programa.

A forma mais conhecida de memória volátil é a Memória de Acesso Aleatório (Random Access Memory- RAM). A memória RAM é relativamente rápida e oferece uma alternativa fácil para criar e armazenar os programas de aplicação do usuário. Em caso de interrupção do fornecimento de alimentação, os CLPs com memória RAM usam baterias ou capacitores de reserva para evitar a perda do programa.

A memória não volátil retém seu conteúdo programado, sem usar bateria ou capacitor, mesmo se houver interrupção do fornecimento de alimentação. A memória EEPROM (Electrically Erasable Programmable Read Only Memory – Memória de Leitura Eletricamente Apagável e Programável) é uma memória não volátil com a mesma flexibilidade da memória RAM, sendo programada por meio de um software de aplicação que roda em um computador pessoal ou por meio de um Terminal de Programação (Hand-Held).

Apesar das memórias RAM e EEPROM poderem salvar os programas aplicativos em caso de interrupção do fornecimento de alimentação, elas não salvam necessariamente os dados do processo, tais como os valores acumulados de um temporizador ou contador. Caso a retenção de dados de um processo seja importante em uma determinada aplicação, escolha um CLP que ofereça 100% de retenção dos dados. Em caso de falha de alimentação, este tipo de CLP salva automaticamente os dados do processo na memória EEPROM não volátil.

5 Dados, Memória e Endereçamento

A memória é um espaço físico e os dados são informações armazenadas neste espaço. A CPU funciona exatamente como um computador: ela manipula os dados usando dígitos binários, os bits. O bit é uma localização discreta dentro de uma pastilha de silício (chip). O bit pode estar submetido a tensão, sendo, portanto, lido como 1 (Energizado), ou não estar submetido a tensão e, então, seu valor será 0 (desenergizado). Portanto, os dados são um padrão de cargas elétricas que representam um valor numérico.

O bit é a menor unidade disponível de memória. Normalmente, as CPUs processam e armazenam os dados em grupos de 16 bits, também conhecidos como “palavras”. Mas os usuários podem também manipular os dados ao nível de bits.

Cada palavra de dados possui uma localização física específica na CPU, chamada de “endereço” ou “registro”. Cada elemento do programa do usuário é referenciado com um endereço para indicar onde se localizam os dados para aquele elemento. Ao atribuir endereços para as E/S de um programa, observe que o endereço está relacionado ao terminal onde os dispositivos de entrada e saída estão conectados.

5.1 Partes de um Endereço

Os endereços são compostos de caracteres alfanuméricos separados por delimitadores. Os delimitadores incluem o dois pontos, o ponto, e a barra.

Os arquivos de Saída e Entrada possuem elementos de 1 palavra, onde cada elemento é especificado pelo número de slot e palavra.

Os Temporizadores e Contadores possuem três elementos de palavra.

Os arquivos de Status, Bit e Inteiro possuem elementos de 1 palavra.

Exemplos

N7:15 é um endereço de elemento, onde o dois pontos separa o Tipo e o Número do Arquivo (Arquivo Inteiro Núm.7) do elemento. Já que os arquivos de Inteiro possuem elementos de 1 palavra, o endereço N7:15 aponta para a palavra Núm.15 no arquivo de inteiro Núm. 7.

T4:7.ACC é um endereço de palavra, onde o ponto separa o elemento da palavra dentro do elemento. Já que os arquivos de Temporizador possuem elementos de 3 palavras, o endereço T4:7.ACC aponta para a palavra de Acumulador (terceira palavra) no elemento Núm. 7 do arquivo de Temporizador T4.

B3:64/15 é um endereço de bit, onde a barra separa o bit do elemento. Já que os arquivos de bits possuem elementos de uma palavra, o endereço B3:64/15 aponta para o bit Núm. 15 na palavra Núm. 64 no arquivo de Bits B3.

I:4.1/3 é um endereço de bits, onde a barra separa o bit da palavra, e o ponto separa a palavra do slot. Esta é uma maneira alternativa de endereçar terminais de E/S 16 e superiores. Outra maneira de representar este endereço é digitar I:4/19.

Apesar dos CLPs operarem de forma binária (1 e 0), eles também usam o binário para converter, aceitar e manipular dados de outros sistemas de numeração. Estes sistema incluem o decimal, o hexadecimal, o decimal codificado em binário (BCD) e o octal. Ver anexo 01.

Decimal	Hexadecimal	Binário	BCD	Octal
0	0	0000	0000	0
1	1	0001	0001	1
2	2	0010	0010	2
3	3	0011	0011	3
4	4	0100	0100	4
5	5	0101	0101	5
6	6	0110	0110	6
7	7	0111	0111	7
8	8	1000	1000	10
9	9	1001	1001	11
10	A	1010		12
11	B	1011		13
12	C	1100		14
13	D	1101		15
14	E	1110		16
15	F	1111		17

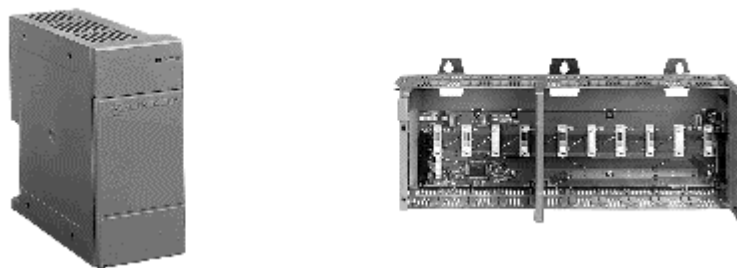
6 Fornecimento de alimentação

A fonte de alimentação fornece energia aos elementos internos do controlador, converte a tensão de entrada em uma forma utilizável e protege os componentes do CLP contra os picos de tensão.

Como a maior parte das instalações passa por flutuações de tensão na linha, as fontes de tensão do CLP são projetadas para manter a operação normal mesmo quando a tensão varia entre 10 e

15%. As quedas e surtos de tensão são causados por quedas da rede pública ou partidas/paradas de equipamentos pesados (tais como motores ou máquinas de solda). Em condições particularmente instáveis de tensão, talvez seja necessário instalar um estabilizador de tensão entre o CLP e a fonte primária de alimentação.

Outro fator que afeta o funcionamento do CLP é a interferência eletromagnética ou ruído elétrico. Apesar dos CLPs serem mais robustos que a maioria dos equipamentos eletrônicos (especialmente os PCs ou os controladores dedicados, que são às vezes usados no lugar dos CLPs), a interferência eletromagnética pode ser um problema. Neste caso, o CLP deve ser isolado por meio da instalação de um transformador de isolamento.

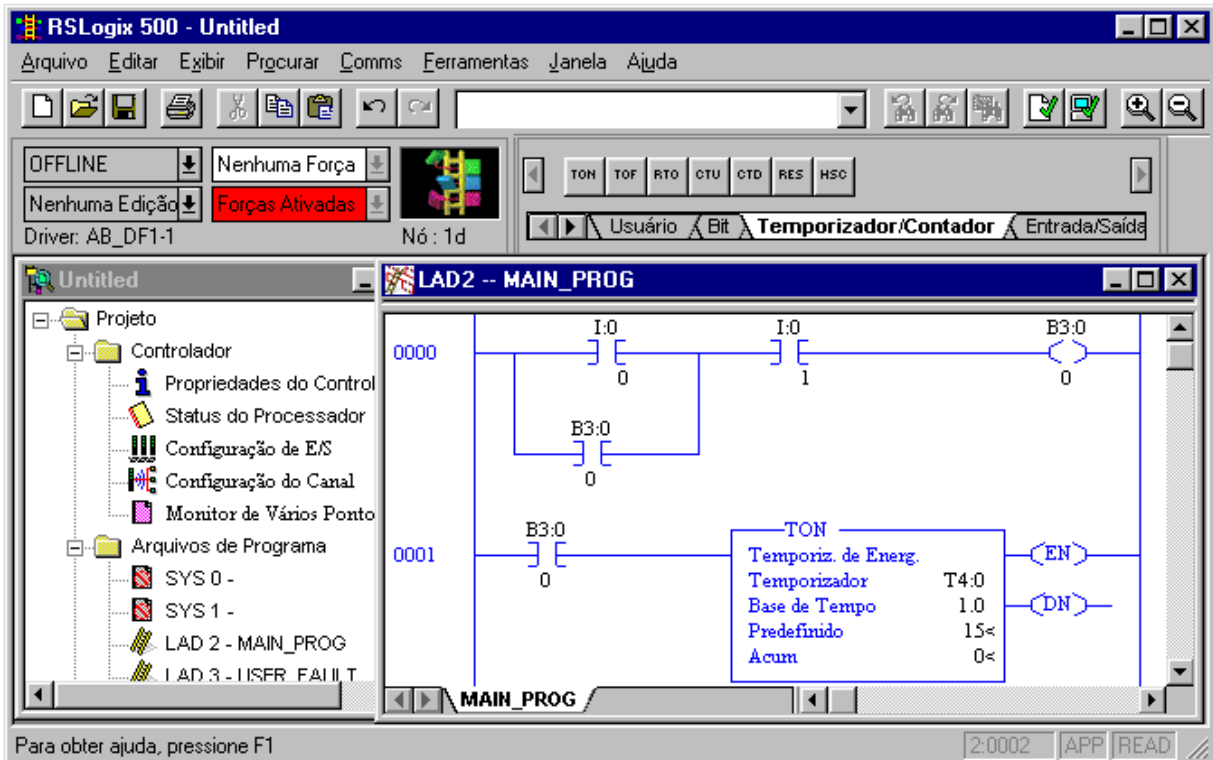


7 Dispositivos de Programação

Para inserir um programa em um CLP, os dois dispositivos mais utilizados são o computador pessoal (PC) e o Terminal Portátil de Programação (TPP).

O PC é usado para rodar o software de programação do CLP. Este software permite aos usuários criar, editar, documentar, armazenar e localizar as falhas dos diagramas ladder, gerando também relatórios impressos. As instruções dos softwares são baseadas em símbolos gráficos para as várias funções. Não é necessário o conhecimento das linguagens mais avançadas de programação para se usar o software, bastando um entendimento genérico dos diagramas elétricos funcionais (veja figura abaixo).

Apesar do TPP poder ser utilizados para programar o CLP, seu uso mais freqüente é na localização de falhas, pois é compacto e tem sua própria memória para armazenar os programas. Os terminais TPP são extremamente úteis quando se trata de localizar falhas em equipamentos nas fábricas, modificar programas e transferir programas a várias máquinas. A linguagem usada pelo TPP é uma forma gráfica de programação de lista de instruções, baseada nas instruções de lógica *ladder* do CLP (veja figura abaixo).



8 Linguagens de Programação

Um programa é uma série de instruções ou comandos que o usuário desenvolve para fazer com que o CLP execute determinadas ações. Uma linguagem de programação estabelece regras para combinar as instruções de forma que gerem as ações desejadas.

8.1 Classificação

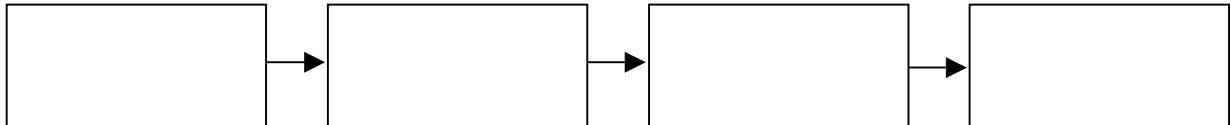
Podemos classificar as diversas linguagens utilizadas na programação de dispositivos microprocessados em dois grupos:

- Linguagem de baixo nível
- Linguagem de alto nível

8.1.1 Linguagem de baixo nível

A linguagem de máquina é considerada a de mais baixo nível, pois cada instrução é composta por combinações dos bits 0 e 1.

A linguagem Assembly é considerada de baixo nível, apesar das instruções consistirem em vocábulos simbólicos (mneumônicos). Neste tipo de linguagem, cada instrução do programa fonte corresponde a uma única instrução do programa objeto.



A linguagem de baixo nível apresenta - alguns inconvenientes no momento da sua utilização, pois requer do usuário conhecimento sobre a arquitetura do microprocessador.

8.1.2 Linguagem de alto nível

Uma linguagem de programação passa a ser de alto nível à medida que esta se aproxima da linguagem corrente utilizada na comunicação entre pessoas.

Apresenta uma estrutura rígida devido às regras utilizadas no momento da elaboração do programa. Uma única instrução em linguagem de alto nível (programa fonte), corresponderá a várias instruções em linguagem de máquina (programa objeto).

Como vantagens, temos:

- Não requer do usuário conhecimento sobre a arquitetura do microprocessador;
- Reduz o tempo gasto na elaboração de programas.

Como desvantagens, temos:

- O número de instruções do programa objeto só será conhecido após a compilação do programa fonte
- O tamanho dos programas em geral fica maior.

Os controladores programáveis utilizam linguagens de alto nível para a sua programação.

A seguir, temos alguns exemplos de utilização das linguagens de programação em função da aplicação.

NOME DA LINGUAGEM	USO
FORTRAN	Aplicações técnico-científicas
COBOL	Aplicações comerciais
PASCAL	Uso geral
BASIC	Uso geral
STEP 5	Programação de CLP SIEMENS/MAXITEC
AL3800	Programação de CLP ALTUS
MASTER TOOL	Programação de CLP ALTUS
PGM	Programação de CLP SISTEMA
SPW	Programação de CLP WEG
IPDS	Programação de CLP ALLEN-BRADLEY
SUCOS 3	Programação de CLP KLÖKNER

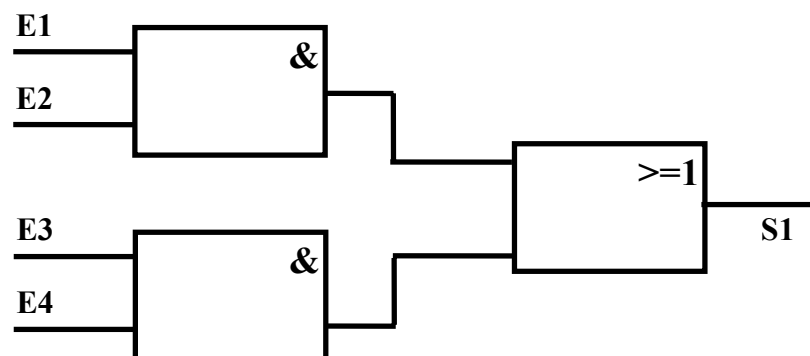
8.2 Linguagens de programação de CLPs

Normalmente podemos programar um controlador programável através de um software que possibilita a sua apresentação ao usuário em três formas diferentes:

- Diagrama de blocos lógicos;
- Lista de instruções
- Diagrama de contatos;

8.2.1 Diagrama de Blocos Lógicos

Mesma linguagem utilizada em lógica digital, onde sua representação gráfica é feita através das chamadas portas lógicas.



8.2.2 Lista de Instrução

Linguagem semelhante à utilizada na elaboração de programas para computadores.

```

: A  E1
: A  E2
: O
: A  E3
: A  E4
:=  S1

```

8.2.3 Diagrama de Contatos

Esta forma de programação, também é conhecida como: Diagrama de relés; diagrama escada ou diagrama *ladder*. Esta forma gráfica de apresentação está muito próxima a normalmente usada em diagramas elétricos. Vamos comparar um *hard-logic* (programação através de fios - comando) e o *soft-logic* (programação através de software – *ladder*) A figura abaixo permite comparar as três representações

8.3 Modos de operação

Normalmente o usuário, poderá dispor dos seguintes modos de operação:

8.3.1 Modo Programação - PROG

Esta posição habilita o controlador ao modo de programação. o controlador não “varre/executa” o programa ladder e as saídas são desenergizadas. É possível desenvolver a edição do programa on-line. O modo do controlador pode ser alterado somente através da posição da chave seletora.

O modo programação permite que o usuário altere a memória do processador da seguinte forma:

- acrescentando novos dados e/ou instruções;
- alterando as informações já gravadas na memória;
- apagando informações previamente gravadas.

As operações executadas, quando o sistema programador se encontra no modo programações, podem ocorrer de duas formas:

Off-line

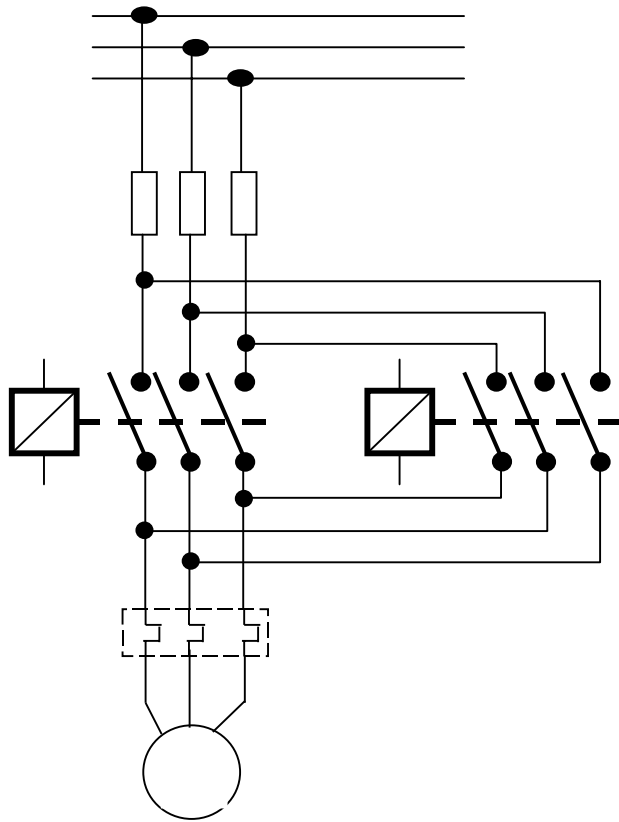
Neste modo de programação, o CLP poderá estar ou não em operação, pois o programa que estiver sendo desenvolvido no sistema de programação não será transferido para o CLP durante o seu desenvolvimento. Portanto, alterações ou apagamentos de programa não provocarão alterações nos dispositivos de saída.

Este modo de programação é o mais seguro, pois o programa só será transferido para o CLP quando o mesmo estiver parado.

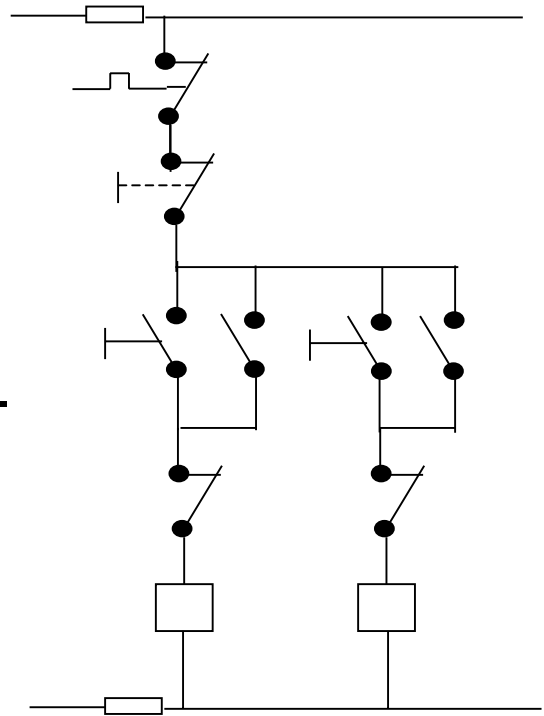
On-line

O modo de programação on-line permite que se alterem dados e/ou instruções na memória do processador, com o CLP em operação. Portanto, qualquer alteração efetuada no programa será executada imediatamente pelo processador.

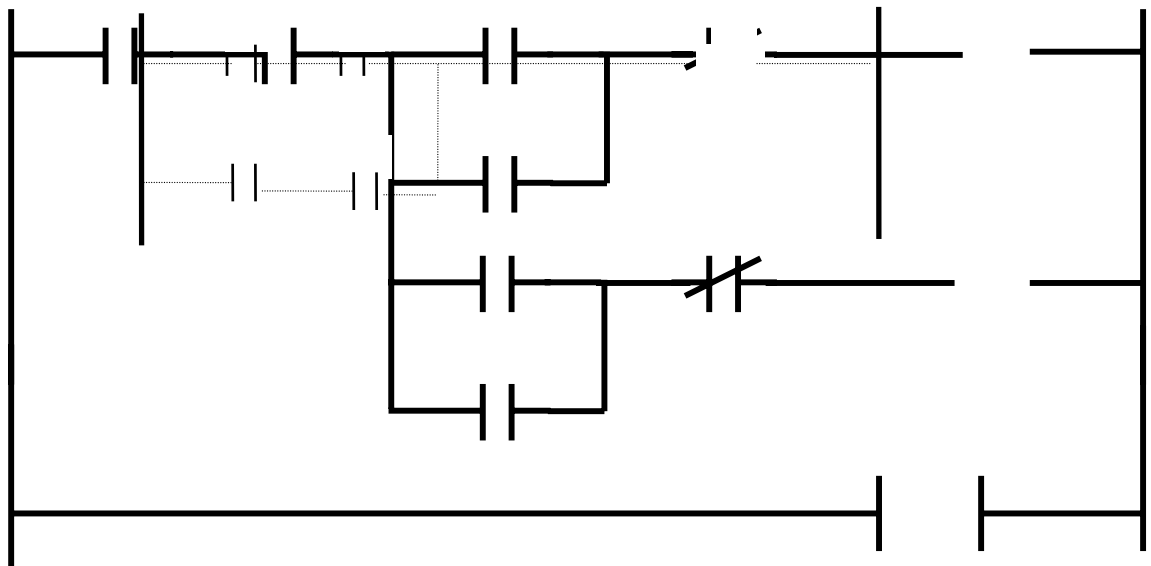
FORÇA



COMANDO



LADDER



Modo Execução – RUN

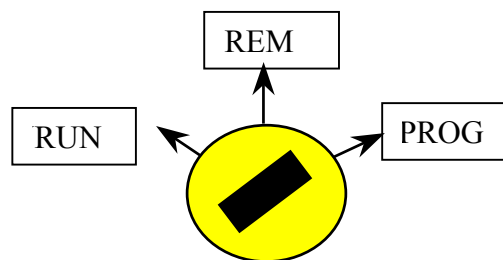
Esta posição habilita o controlador ao modo de operação. O controlador “varre/executa” o programa ladder, monitora dispositivos de entrada e saída e ativa os pontos forçados de E/S habilitados. O modo do controlador pode ser alterado somente através da posição da chave seletora. Não é possível desenvolver a edição do programa on-line.

Para mudar o modo do controlador para RUN, gire a chave seletora de PRPG ou REM para RUN. Ao selecionar a chave no modo RUN, não é possível utilizar uma interface de operação/programação para alterar o modo do controlador.

Modo Remoto - REM

Esta posição habilita o controlador ao modo remoto. Modos REM/RUN, REM/PROG ou REM/TEST. O modo do controlador pode ser alterado através da posição da chave seletora ou mudando o modo através de uma interface de programação/operação. É possível desenvolver a edição de programa on-line nessa posição.

Quando a chave seletora estiver na posição REM, é possível utilizar uma interface de programação/operação para mudar o modo do controlador.



8.4 Modelos de Arquitetura de CLPs

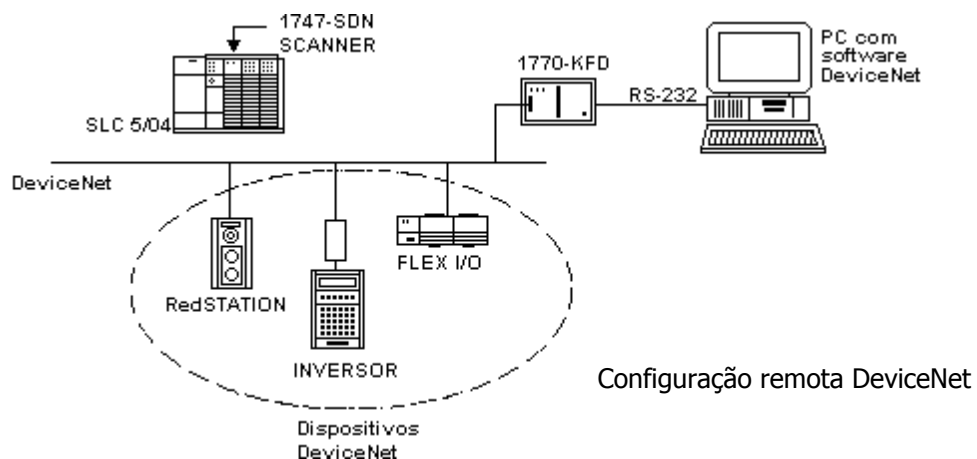
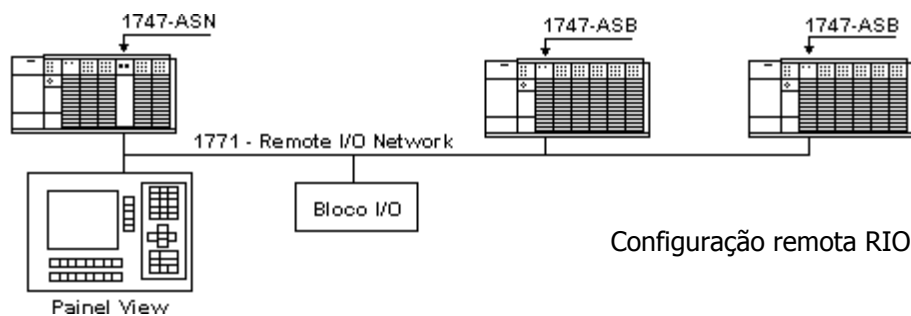
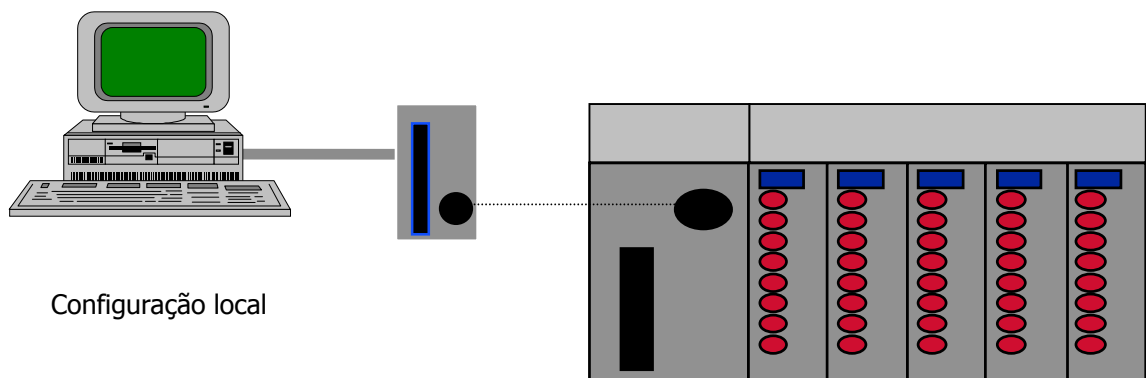
A arquitetura de um CLP, está ligada a maneira como os módulos de I/O estão ligados a CPU. A arquitetura, também chamada de configuração, representa a disposição como estão conectados os diversos módulos de I/O, podendo ser classificado como:

- **Configuração local**
- **Configuração remota: RIO e DeviceNet**
- **Configuração em rede**

Entende-se como **configuração local**, aquela em que os módulos de I/O, estão montados no mesmo rack da CPU ou a no máximo 15 metros de distância do mesmo. Os processadores SLC-500 podem controlar até 3 chassis ou 30 slots.

Entende-se como **configuração remota RIO**, aquela em que os módulos I/O, estão montados fora do rack da CPU em distâncias acima de 15 metros. Para tal finalidade são necessários módulos especiais para interligação de racks remotos (Scanner Remote I/O 1747-SN). Cada Scanner 1747-SN suporta 4 chassis de 256 E/S cada um. Podem ser conectados a uma rede RIO (Remote I/O), no máximo 16 dispositivos. A distância máxima para este tipo de configuração gira em torno de 3000 metros.

Entende-se como **configuração Remota DeviceNet**, aquela que conecta os dispositivos de chão de fábrica diretamente ao sistema de controle, reduzindo o número de interfaces de E/S e a fiação associada. A Rede de comunicação DeviceNet é uma rede completamente aberta e possui o suporte de fabricantes de sensores, atuadores e dispositivos de controle.



Em uma configuração típica, o Scanner DeviceNet 1747-SDN atua como uma interface entre os dispositivos e os controladores SLC 500. O Scanner comunica os dispositivos DeviceNet através da rede para:

1. Ler as entradas de um dispositivo;
2. Escrever as saídas para um dispositivo;
3. Descarregar os dados de configuração;
4. Monitorar o status operacional de um dispositivo.

Comprimento máximo da Devicenet: 500 m.

Entende-se como **configuração em rede**, aquela em que diversas CPU's e módulos I/O, estão montados e interligados por um cabo de comunicação. A rede utilizada para os controladores SLC 500 é chamada de DH-485. A rede DH-485 foi projetada de modo a passar informações entre os dispositivos instalados na planta. A rede monitora os parâmetros do processo e das CPUs, o status das CPUs e do processo e os programas de aplicação de modo a suportar a aquisição e monitoramento de dados, o carregamento/d Descarregamento do programa e o controle de supervisão.

A rede DH-485 oferece:

1. Interconexão de 32 dispositivos;
2. A habilidade de adicionar ou remover nós sem interromper a rede;
3. Comprimento máximo da rede: 1219 m.

9 PROGRAMAÇÃO DO CLP EM LADDER

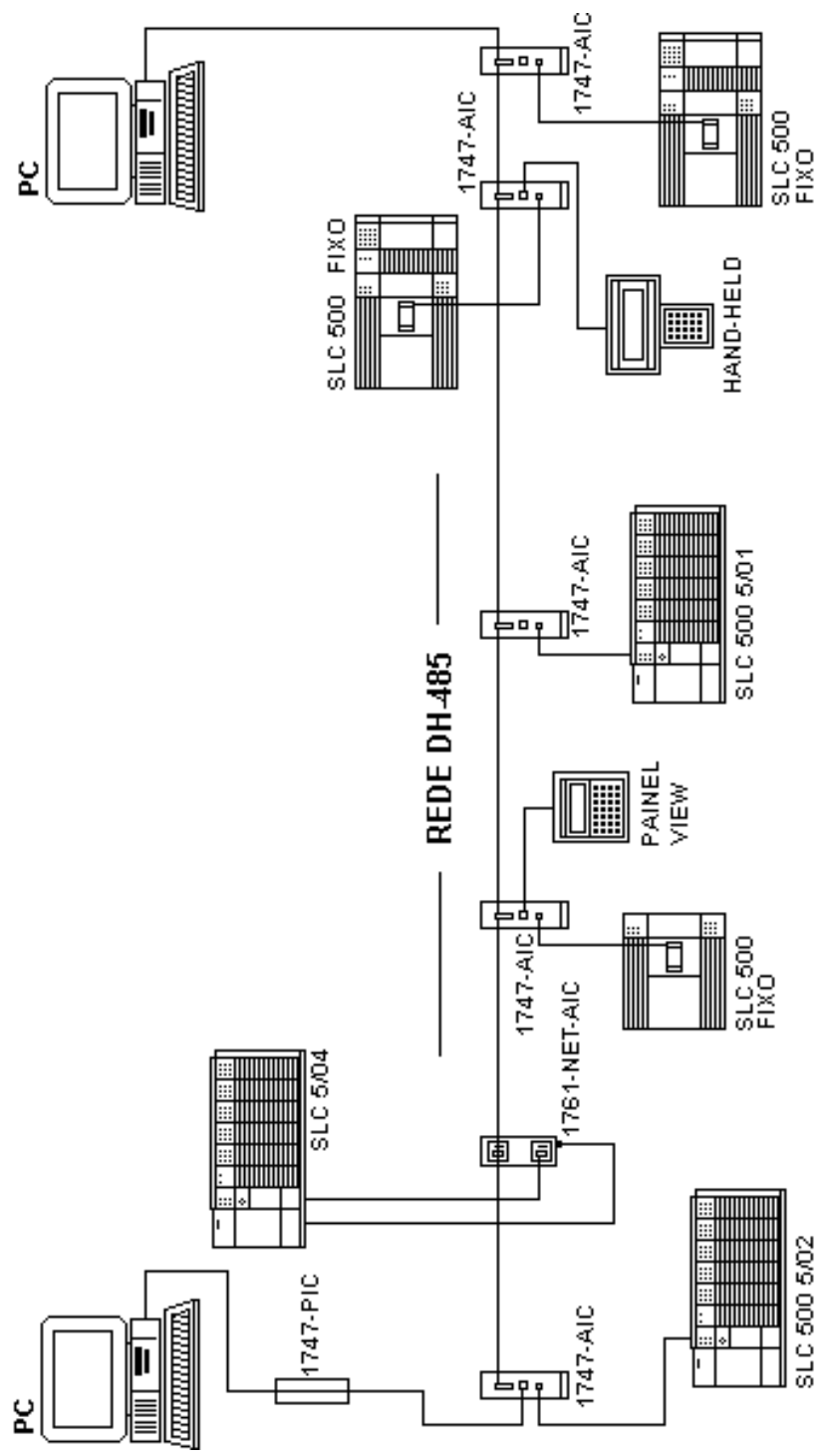
A partir de agora teremos informações gerais sobre as Instruções Básicas e explicações como elas funcionam. Cada uma dessas Instruções Básicas inclui informações como:

- Simbologia
- Como se usa a Instrução

9.1 Instruções de Bit

As Instruções de Bit são as seguintes:

- Examinar se Energizado (XIC)
- Examinar se Desenergizado (XIO)
- Energizar Saída (OTE)
- Energizar Saída com Retenção (OTL)
- Desenergizar Saída com retenção (OTU)
- Monoestável sensível a Borda de Subida (OSR)



9.1.1 Instruções de “Examinar”

São duas as Instruções de Examinar:

- Examinar se Energizado (XIC)
- Examinar se Desenergizado (XIO)

a) Examinar se Energizado (XIC)

Formato da Instrução XIC



Quando um dispositivo de entrada fecha seu circuito, o terminal de entrada conectado ao mesmo indica um estado energizado, que é refletido no bit correspondente do arquivo de entrada. Quando o controlador localiza uma instrução com o mesmo endereço, ele determina que o dispositivo de entrada está energizado (1), ou fechado, e ajusta a lógica da instrução para verdadeira. Quando o dispositivo de entrada não mais fecha seu circuito, o controlador verifica que o bit está desenergizado (0) e ajusta a lógica dessa instrução para falsa (tabela 1A).

Tabela 1.A

Lógica da Instrução XIC

Estado do Bit	Instrução XIC
0	Falsa
1	Verdadeira

b) Examinar se Desenergizado (XIO)

A figura abaixo ilustra o formato da Instrução Examinar se Desenergizado

Formato da Instrução XIO



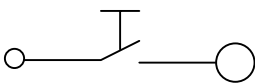
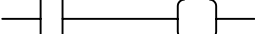
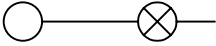
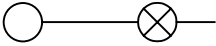
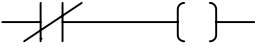
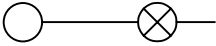

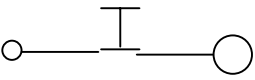
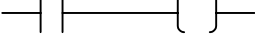
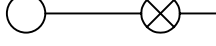
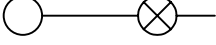
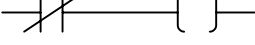
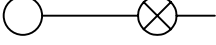

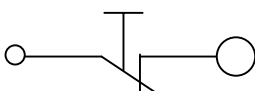
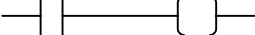
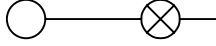
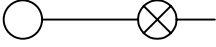
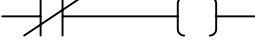
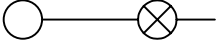

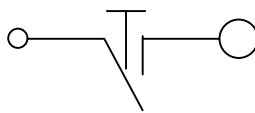
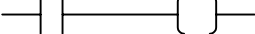
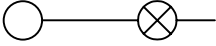
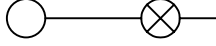
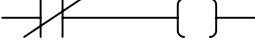
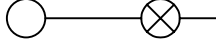

Quando um dispositivo de entrada não é acionado, o terminal de entrada conectado a ele indica um estado desenergizado, que é refletido no bit correspondente do arquivo de entrada. Ao localizar uma instrução XIO com o mesmo endereço, o controlador determina que a entrada está desenergizada (0) e ajusta a lógica da instrução para verdadeira. Quando o dispositivo é acionado, o controlador ajusta a lógica dessa instrução para falsa (tabela 1.B).

Tabela 1.B

Lógica da Instrução XIO

Estado do Bit	Instrução XIO
0	Verdadeira
1	Falsa

Instruções de condição e seus resultados:

	Dispositivo De Entrada	Entrada		de Saída	da Saída
Botão NA - Não Ativado					
					
Botão NA - Ativado					
					
Botão NF - Não Ativado					
					
Botão NF - Ativado					
					

9.1.2 Instruções Energizar / Desenergizar Saída

As instruções Energizar/desenergizar Saída são empregadas para energizar ou desenergizar um bit específico.

Essas instruções são as seguintes:

- Energizar Saída (OTE)
- Energizar Saída com Retenção (OTL)
- Desenergizar Saída com Retenção (OTU)

a) Energizar Saída (OTE)

A figura abaixo ilustra o formato da instrução Energizar Saída (OTE).

Formato da Instrução OTE

—()—

O estado de um terminal de saída é indicado através de um bit específico do arquivo de saída. Ao ser estabelecida uma lógica verdadeira na linha de programa que contém a instrução OTE, o controlador energiza o respectivo bit (1), fazendo com que o terminal de saída seja energizado e o dispositivo de saída conectado a este terminal seja acionado. Caso essa lógica verdadeira não seja estabelecida, o controlador desenergiza o bit (0), a instrução OTE é desabilitada e o dispositivo de saída associado é desenergizado.

A instrução OTE é não-retentiva e a mesma é desabilitada quando:

- o controlador for alterado para o modo Operação ou Teste, ou quando a alimentação é restaurada;
- ocorrer um erro grave.

Deve-se observar que uma instrução OTE habilitada em uma área de subrotina permanecerá habilitada até que haja uma nova varredura na área de subrotina.

b) Energizar Saída com Retenção e Desenergizar saída com Retenção

A figura abaixo ilustra o formato das instruções Energizar Saída com Retenção (OTL) e Desenergizar Saída com Retenção (OTU).

Formato das Instruções OTL E OTU

—(L)—

—(U)—

Essas instruções são instruções de saída retentiva e geralmente, são utilizadas aos pares para qualquer bit da tabela de dados controlado pelas mesmas. Também podem ser empregadas para inicializar valores de dados em nível de bit.

Quando se determina um endereço para a instrução OTL que corresponde ao endereço de um terminal do módulo de saída, o dispositivo de saída conectado a este terminal será energizado assim que o bit na memória for energizado. O estado habilitado deste bit é determinado pela lógica da linha anterior às instruções OTL e OTU.

Caso a lógica verdadeira seja estabelecida com instruções de entrada, a instrução OTL é habilitada. Se a mesma não for estabelecida e o bit correspondente na memória não tiver sido energizado previamente, a instrução OTL não será habilitada. Entretanto, se a lógica verdadeira for estabelecida previamente, o bit na memória será retido energizado e assim permanecerá, mesmo após as condições da linha terem se tornado falsas.

Uma instrução OTU com o mesmo endereço da instrução OTL rearma (desabilita ou desenergiza) o bit na memória. Quando uma lógica verdadeira é estabelecida, a instrução OTU desenergiza seu bit correspondente na memória.

O programa de aplicação pode examinar um bit controlado pelas instruções OTL e OTU sempre que necessário.

9.1.3 Monoestável Sensível a Borda de subida (OSR)

A figura abaixo ilustra o formato da instrução Monoestável Sensível à Borda de Subida (OSR).

Formato da instrução OSR-

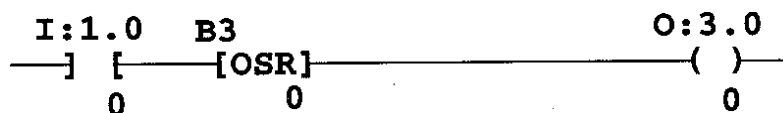
— **[OSR]** —

Essa instrução torna a linha verdadeira durante uma varredura com uma transição de falsa para verdadeira da condição anterior à atual da linha.

As aplicações para esta instrução incluem iniciar eventos acionados por um botão de comando, como por exemplo, “congelar” valores exibidos muito rapidamente (LED).

A figura abaixo a seguir, exhibe a utilização da instrução OSR.

Exemplo 1 de Instrução OSR para controlador SLC-5/03



Na figura anterior quando a instrução de entrada passa de falsa para verdadeira, a instrução OSR condiciona a linha de forma que a saída fique verdadeira durante uma varredura do programa. A saída passa a falsa e assim permanece durante várias varreduras até que a entrada realize uma nova transição de falsa para verdadeira.

O Controlador Micrologix permite utilizar uma instrução OSR por saída em uma linha.

Importante: Recomenda-se não utilizar um endereço de saída juntamente com a instrução OSR, devido a pequena duração do tempo de uma varredura.

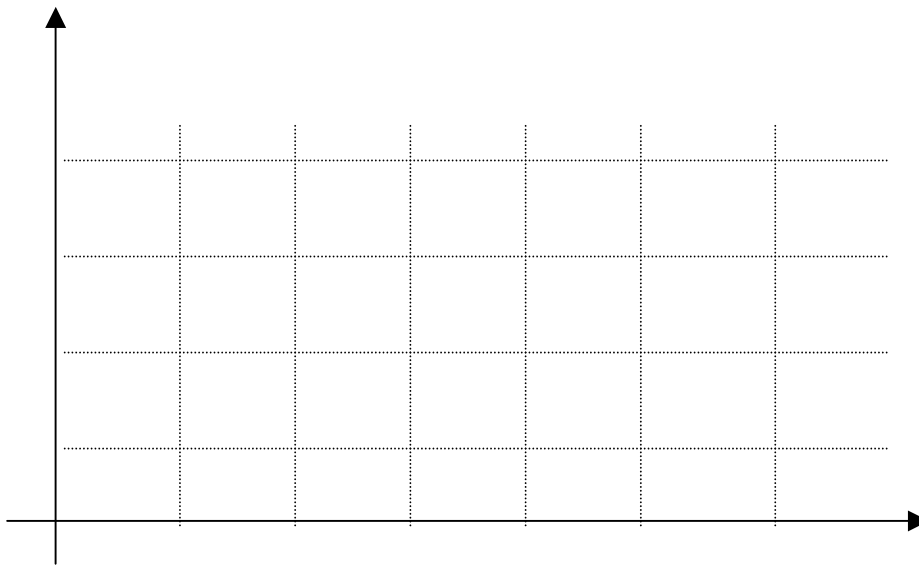
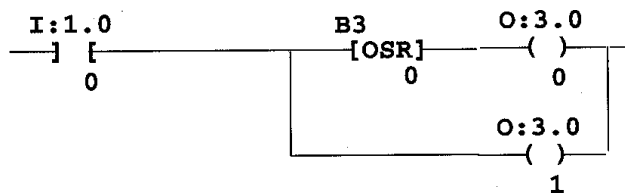


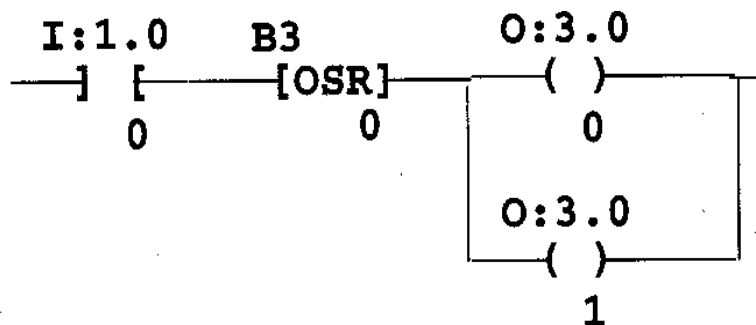
Diagrama ilustrativo da Instrução OSR

Uso da Instrução OSR em *Branch* (Paralelo)

No exemplo da figura abaixo, a instrução OSR **não** poderá ser usada dentro de uma Branch (paralelo).



No exemplo da Figura abaixo, a linha é verdadeira, porque a instrução OSR esta fora do Branch.



9.1.4 Instruções de Temporizador

As instruções de temporizador são as seguintes:

- Temporizador na Energização (TON)
- Temporizador na Desenergização (TOF)

- Temporizador Retentivo (RTO)

Essas instruções encontram-se descritas nas seções a seguir.

- Temporizador na Energização (TON): conta intervalos de base de tempo quando a instrução é verdadeira. A base de tempo é selecionada entre 0,01s ou 1,0s para os Controladores SLC-5/03;
- Temporizador na Desenergização (TOF): conta intervalos de base de tempo quando a instrução é falsa. A base de tempo é selecionada entre 0,01s ou 1,0s para os Controladores SLC-5/03.
- Temporizador Retentivo (RTO): este temporizador retém o seu valor acumulado quando a instrução se torna falsa.

As instruções de Temporizador e Contador requerem três palavras do arquivo de dados. A palavra **0** é a palavra de controle que contém os bits de estado da instrução. A palavra **1** é o valor Predefinido. A palavra **2** corresponde ao valor acumulado. Figura abaixo:

	15	14	13	
Word 0	EN	TT	DN	Internal Use
Word 1	Preset Value			
Word 2	Accumulator Value			

Addressable Bits	Addressable Words
EN = Bit 15 Enable	PRE = Preset Value
TT = Bit 14 Timer Timing	ACC = Accumulated Value
DN = Bit 13 Done	

ACC. Para os temporizadores, o valor acumulado é o número atual de intervalos temporizados que transcorreram; para contadores, é o número de transições de falso para verdadeiro que ocorreram.

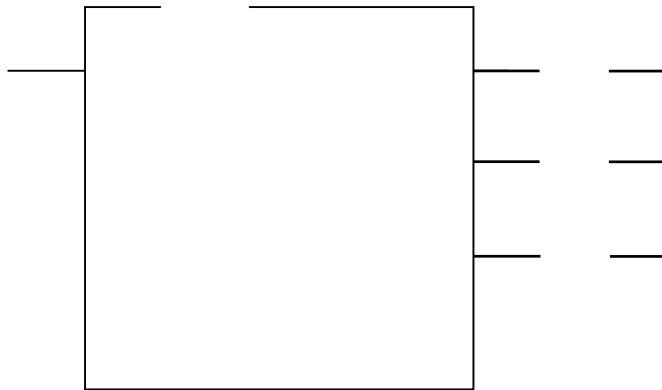
PRE. O valor Predefinido é o valor inserido para controlar a temporização ou contagem da instrução.

Quando o valor acumulado for igual ou maior que o valor Predefinido, o bit de estado será energizado. Pode-se utilizar este bit para controlar um dispositivo de saída.

Os valores Predefinido e acumulado para temporizadores variam de 0 a +32.767 e os valores para contadores variam de -32.768 a +32.767. Se o valor acumulado ou Predefinido do temporizador for um número negativo, ocorrerá um erro de *run-time*, causando falha no controlador.

a) Temporizador na Energização - TON

A figura a seguir ilustra o formato da instrução de temporizador na Energização (TON)
Formato da Instrução TON



A instrução de Temporizador na Energização (TON) inicia a contagem dos intervalos da base de tempo quando a condição da linha se torna verdadeira. À medida que a condição da linha permanece verdadeira, o temporizador incrementa seu valor acumulado (ACC) a cada varredura até atingir o valor Predefinido (PRE). O valor acumulado é zerado quando a condição da linha for falsa independente do temporizador ter ou não completado a temporização.

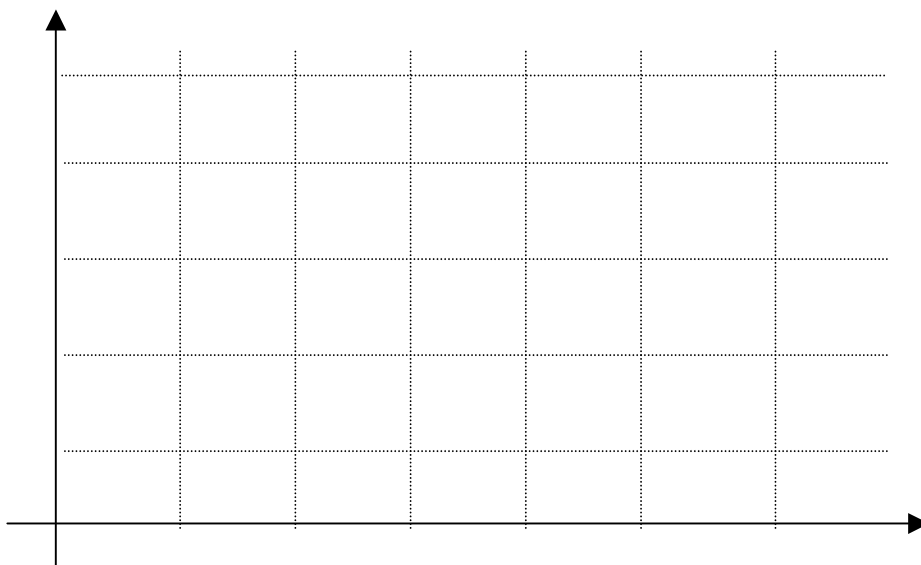


Diagrama ilustrativo da Instrução TON

O bit de executado (DN) é energizado quando o valor acumulado é igual ao valor Predefinido e é desenergizado quando a condição da linha se torna falsa. O bit de temporizado (TT) do temporizador é energizado quando a condição da linha é verdadeira e o valor acumulado é menor que o valor Predefinido. Quando o bit de executado é energizado ou a condição da linha é falsa, esse bit é desenergizado. O bit de habilitação (EN) do temporizador é energizado quando a condição da linha é verdadeira. Caso contrário, esse bit é desenergizado.

Se o controlador for passado do modo Operação ou Teste para Programação, ou então, se a alimentação for perdida enquanto uma instrução TON está contando o tempo sem ainda ter atingido o valor Predefinido, ocorre o seguinte:

- os bits de habilitação e temporizado permanecem energizados.
- o valor acumulado permanece o mesmo.

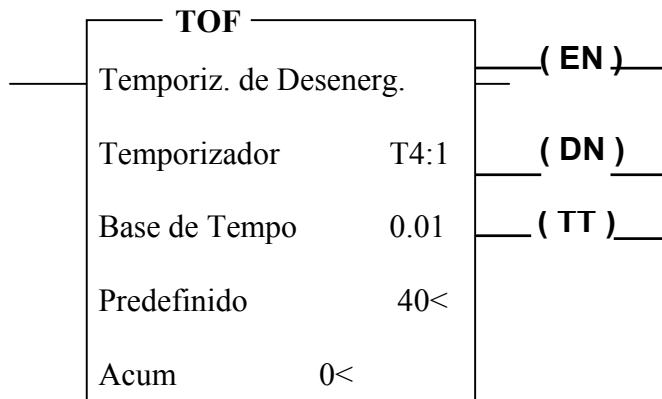
Quando o controlador retorna ao modo Operação ou Teste, pode acontecer o seguinte:

- se a linha for verdadeira, o valor acumulado é zerado e os bits de habilitação e temporizado permanecem energizados.
- se a linha for falsa, o valor acumulado é zerado e os bits de controle são desenergizados.

b) Temporizador na Desenergização - TOF

A figura a seguir ilustra o formato da instrução de Temporizador na Desenergização (TOF)

Formato da Instrução TOF



A instrução de Temporizador na Desenergização (TOF) inicia a contagem dos intervalos da base de tempo quando a linha realiza uma transição de verdadeira para falsa. À medida que a condição da linha permanece falsa, o temporizador incrementa o seu valor acumulado (ACC) a cada varredura até atingir o valor Predefinido (PRE). O valor acumulado é zerado quando a condição da linha for verdadeira, independente do temporizador ter realizado a temporização.

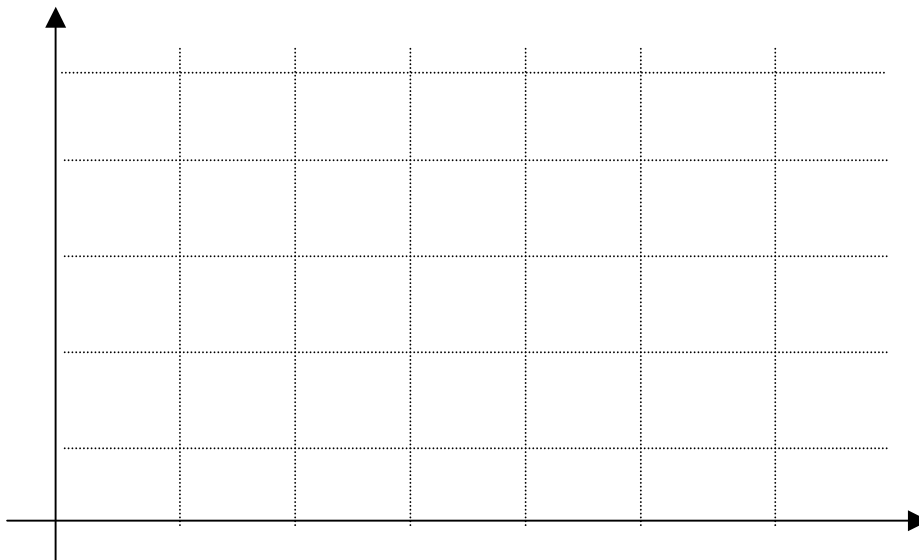
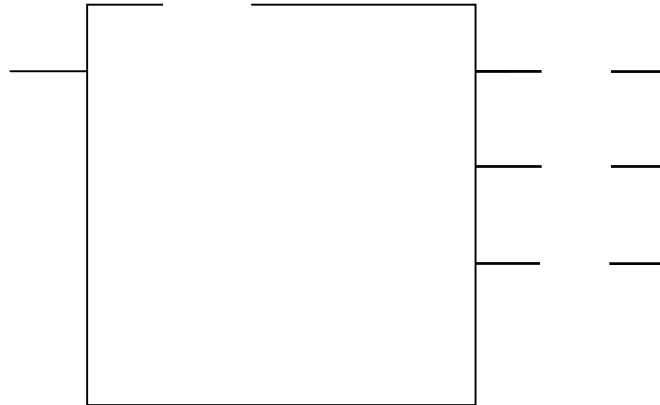


Diagrama ilustrativo da Instrução TOF

c) Temporizador Retentivo - RTO

A figura a seguir ilustra o formato da instrução RTO.

Formato da Instrução RTO



A instrução RTO inicia a contagem dos intervalos da base de tempo quando a condição da linha se torna verdadeira. À medida que a condição da linha permanece verdadeira o temporizador incrementa o seu valor acumulado (ACC) a cada varredura até atingir o valor Predefinido (PRE). O valor acumulado é retido quando:

- a condição da linha se torna falsa
- o controlador é alterado de Operação ou Teste para Programação
- o controlador perde a alimentação (desde que seja mantida a bateria de back-up)
- ocorre uma falha

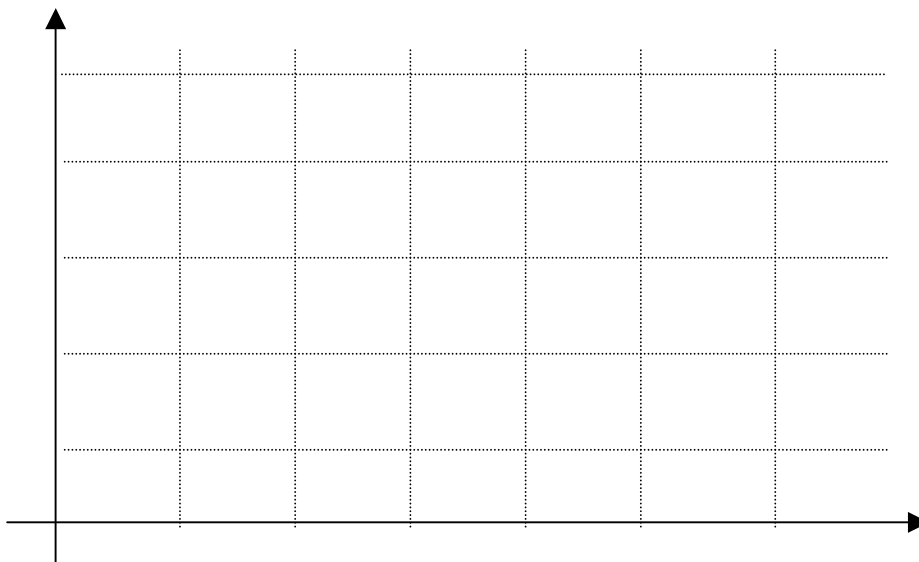


Diagrama ilustrativo da Instrução RTO

Quando o controlador retorna ao modo Operação ou Teste e/ou a condição da linha passa a verdadeira, a temporização continua a partir do valor acumulado retido. Ao reter o seu valor acumulado, o temporizador retentivo mede o período em que a condição da linha está verdadeira. Pode-se utilizar essa instrução para energizar ou desenergizar uma saída dependendo da lógica do programa.

Os bits de estado da instrução RTO operam como descrito a seguir:

- bit de executado (DN) é energizado quando o valor acumulado é igual ao valor Predefinido. No entanto, esse bit não é desenergizado quando a condição da linha se torna falsa, ele só é desenergizado quando a instrução RES é habilitada.
- bit de temporizado (TT) da instrução de Temporizador Retentivo é energizado quando a condição da linha é verdadeira e o valor acumulado é menor que o valor Predefinido. Quando a condição da linha passa a falsa ou quando o bit de executado é energizado, o bit de temporizado é desenergizado.
- bit de habilitação (EN) é energizado quando a condição da linha é verdadeira e é desenergizado quando a condição se torna falsa.

O valor acumulado deve ser zerado pela instrução **RES**. Quando a instrução RES (reset), com o mesmo endereço da instrução RTO, for habilitada, o **valor acumulado é zerado** e os **bits de controle são desenergizados**.

OBS: A instrução RES de contador/temporizador não deve ser empregada com a instrução TOF.

9.1.5 Instruções de Contador Crescente (CTU) E Decrescente (CTD)

As Instruções de Contador são as seguintes:

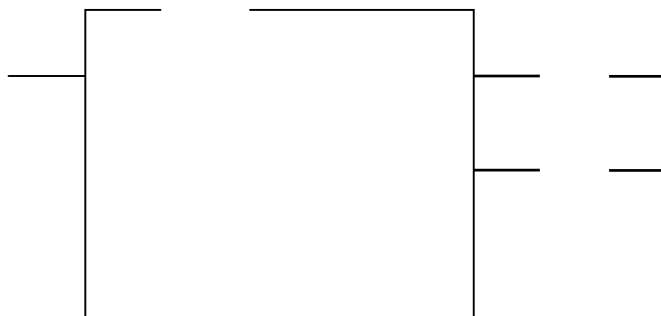
- Contador Crescente (CTU)
- Contador Decrescente (CTD)
- Rearme (RES)

Estas Instruções encontram-se descritas a seguir.

a) Contador Crescente (CTU):

A contagem é incrementada a cada transição de falso para verdadeiro.

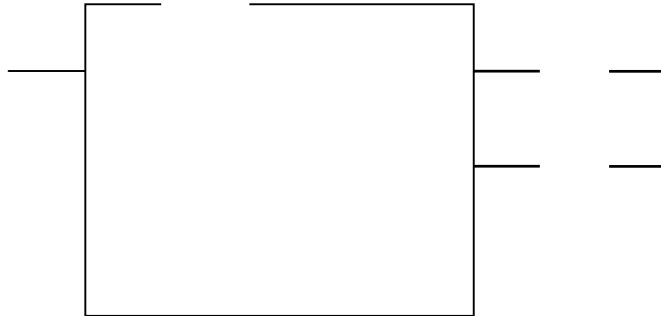
Formato da instrução CTU



b) Contador Decrescente (CTD):

A contagem é decrementada a cada transição de falso para verdadeiro.

Formato da Instrução CTD



As instruções de Contador Crescente (CTU) e Contador Decrescente (CTD) contam as transições de falsa para verdadeira, as quais podem ser causadas por eventos que ocorrem no programa, tais como peças que passam por um detetor.

c) Rearme (RES):

Esta instrução zera o valor acumulado e os bits de estado de um Temporizador e Contador. Quando a Instrução **RES** é habilitada, é **zerado** o valor acumulado (**ACC**) no Temporizador na Energização (TON), no Temporizador na Desenergização (TOF), no Temporizador Retentivo (RTO), no Contador Crescente (CTU) e no Contador Decrescente (CTD) que tenham o mesmo endereço da instrução (RES).

As Instruções de Contador requerem três palavras do arquivo de dados. A palavra 0 é a palavra de controle e contém os bits de estado da Instrução. A palavra 1 é o valor de preset. A palavra 2 é o valor do acumulador. A palavra de controle contém seis bits de estado como representado na figura a seguir.

	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Word 0	CU	CD	DN	OV	UN	UA										Internal Use
Word 1	Preset Value															
Word 2	Accumulated Value															

Addressable Bits

- CU = Count up enable
- CD = Count down enable
- DN = Done bit
- OV = Overflow bit
- UN = Underflow bit
- UA = Update accum. value

Addressable Words

- PRE = Preset
- ACC = Accum

Os valores acumulado e Predefinido são armazenados como números inteiros (os valores negativos são armazenados na forma de complemento de 2).

Quando as condições da linha para uma instrução CTU passam de falsa para verdadeira, o valor acumulado é incrementado de um, desde que haja uma varredura entre essas transições. Quando isto ocorre sucessivamente até que o valor acumulado se torne igual ao valor Predefinido, o bit de executado (DN) é energizado, permanecendo nesse estado se o valor acumulado exceder o valor Predefinido.

O **bit 15** da palavra de controle da instrução de Contador é o bit de habilitação de Contador Crescente (CU). Esse bit é energizado quando a condição da linha é verdadeira e desenergizado quando a condição da linha se torna falsa ou uma instrução RES, com o mesmo endereço da CTU, é habilitada.

A instrução CTU pode contar além de seu valor Predefinido. Quando a contagem ultrapassa o valor Predefinido e atinge $(32.767 + 1)$, ocorre uma condição de overflow. Isso é indicado quando o **bit 12**, bit de overflow (OV), é energizado.

Pode-se desenergizar o bit de overflow habilitando-se uma instrução RES com o mesmo endereço da instrução CTU. Também é possível desenergizá-lo, decrementando a contagem para um valor menor ou igual a 32.767 com uma instrução CTD.

Quando o bit de overflow (OV) é energizado o valor acumulado atinge -32.768 e continua a contagem crescente a partir daí.

As instruções CTD também contam as transições da linha de falsa para verdadeira. O valor acumulado do contador é decrementado a cada transição de falsa para verdadeira. Quando ocorre um número suficiente de contagens e o valor acumulado se torna menor que o valor Predefinido, o bit de executado (bit 13) do contador é desenergizado.

O **bit 14** da palavra de controle da instrução de Contador é o bit de habilitação de Contador Decrescente (CD). Esse bit é energizado quando a condição da linha é verdadeira e é desenergizado quando a condição da linha se torna falsa (contador decrescente desabilitado) ou a instrução apropriada de desenergização é habilitada.

Quando a instrução CTD conta além do seu valor Predefinido e atinge $(-32.768 - 1)$, o bit de underflow (**bit 11**) é energizado. Pode-se desenergizar esse bit, habilitando-se a instrução RES apropriada. Pode-se também desenergizá-lo, incrementando a contagem para um valor maior ou igual a -32.768 com uma instrução CTU com o mesmo endereço da instrução CTD.

Quando o bit de underflow (UN) é energizado, o valor acumulado atinge +32.767 e continua a contagem decrescente a partir daí.

As instruções CTU e CTD são retentivas. O valor acumulado (ACC) é retido depois que a instrução CTU ou CTD passa a falsa e quando a alimentação do controlador é removida e, a seguir, restaurada.

Os estados energizado ou desenergizado dos bits de executado, overflow e underflow também são retentivos. Esses bits de controle e o valor acumulado são zerados quando a instrução RES é habilitada.

Cada contagem é retida quando as condições da linha se tomam falsas e, assim permanece até que uma instrução RES, com o mesmo endereço da instrução de contador, seja habilitada.

Cada instrução de contador possui um valor Predefinido e acumulado, e uma palavra de controle associada.

d) Como o Contador trabalha

A figura 21, demonstra como o controlador trabalha.

O valor da contagem deve estar entre, (-32768 a 32767). Se o valor do Contador vai acima de 32 767 ou abaixo de -32 768 o status do Contador acusará overflow (OV) ou underflow (UN) e o bit é setado.

O Contador pode ser resetado a zero usando a Instrução (RES) .

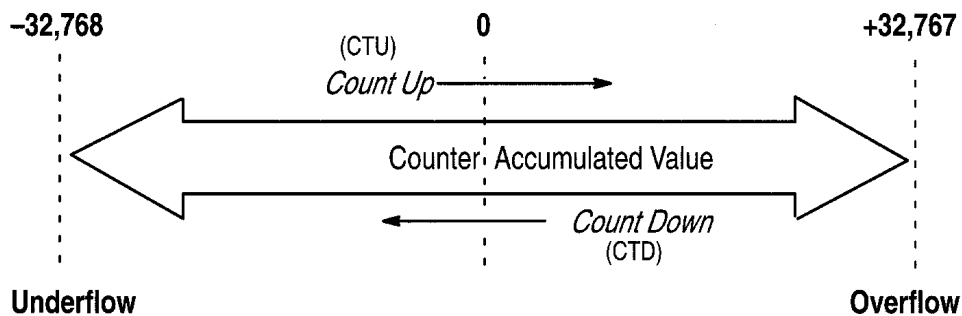


Figura 21

9.1.6 Instruções de Comparação

As instruções de Comparação são usadas para testar pares de valores de forma a condicionar a continuidade lógica de uma linha.

a) Igual a (EQU)

Testa se dois valores são iguais. Se a *Origem A* e *Origem B* são iguais, a lógica da linha é verdadeira.

Origem A deve ser um endereço. **Origem B** pode ser uma constante do programa ou um endereço.

A figura a seguir apresenta o formato da instrução



b) Diferente (NEQ)

Testa se o primeiro valor não é igual ao segundo. Se Origem A e Origem B são diferentes, a lógica da linha é verdadeira.

Origem A deve ser um endereço. **Origem B** pode ser uma constante do programa ou um endereço.

A figura a seguir apresenta o formato da instrução NEQ

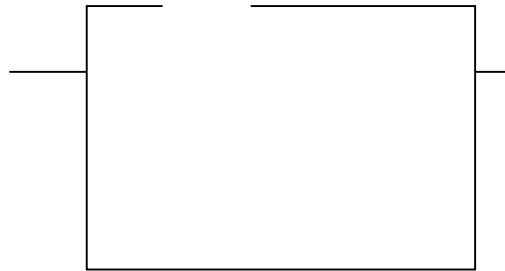


c) Menor que (LES)

Testa se o primeiro valor é menor que o segundo. Se a Origem A é menor que o valor da Origem B a lógica da linha é verdadeira.

Origem A deve ser um endereço. **Origem B** pode ser uma constante do programa ou um endereço.

A figura a seguir apresenta o formato da instrução



d) Menor ou igual a (LEQ)

Testa se o primeiro valor é menor ou igual ao segundo. Se o valor da Origem A é menor ou igual Origem B, a lógica da linha é verdadeira.

Origem A deve ser um endereço. **Origem B** pode ser uma constante do programa ou um endereço.

A figura a seguir apresenta o formato da instrução



e) Maior que (GRT)

Testa se o primeiro valor é maior que o segundo. Se o valor da Origem A é maior que o valor da Origem B, a lógica da linha é verdadeira.

Origem A deve ser um endereço. **Origem B** pode ser uma constante do programa ou um endereço.

A figura a seguir apresenta o formato da instrução



f) Maior ou igual a (GEQ)

Testa se o primeiro valor é maior ou igual ao segundo. Se o valor da Origem A é maior ou igual ao valor da Origem B, a lógica da linha é verdadeira.

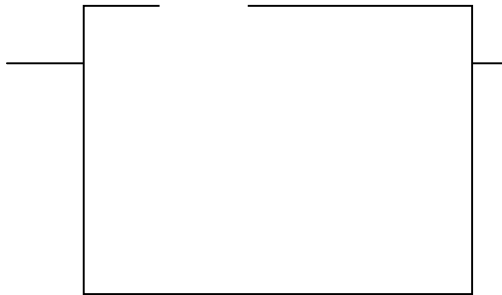
Origem A deve ser um endereço. **Origem B** pode ser uma constante do programa ou um endereço.

A figura a seguir apresenta o formato da instrução



g) Teste de Limite (LIM)

Exemplo:



LIM testa se o valor de **Teste** está dentro ou fora da faixa especificada por **Limite Inferior** (Lim Inf) e **Limite Superior** (Lim Sup).

Para testar se o valor de **Teste** está dentro da faixa, o **Limite Inferior** deve ter um valor igual a ou menor que o **Limite Superior**. A instrução será verdadeira quando o valor de Teste estiver entre os limites ou for igual a um dos limites. Se o valor de Teste estiver fora dos limites, a instrução será falsa.

Para testar se o valor de **Teste** está fora da faixa, o **Limite Inferior** deve ter um valor maior que o **Limite Superior**. A instrução será verdadeira quando o valor de Teste estiver fora dos limites ou for igual a um dos limites. Se o valor de Teste estiver entre os limites, a instrução será falsa.

Fornecendo Parâmetros

Dependendo de como você define o parâmetro **Teste**, os parâmetros **Limite Inferior** e **Limite Superior** podem ser um endereço de palavra ou uma constante de programa. Veja abaixo.

Se Teste for	Limite Inferior	Limite Superior
Constante	Endereço de Palavra	Endereço de Palavra
Endereço de Palavra	Endereço de Palavra ou Constante	Endereço de Palavra ou Constante

9.1.7 Instruções Matemáticas

As Instruções Matemáticas consideram um par de valores e realizam a operação desejada. O resultado é colocado em uma localização separada.

Parâmetros das Instruções

Origem - Endereços dos valores em que a operação matemática será executada. Podem ser endereços de palavra ou constantes de programa. Se a instrução tiver dois operandos Origem, não é possível introduzir constantes de programa nos dois operandos.

Dest - Endereço destino referente ao resultado da operação.

a) Adição (ADD)

Adiciona o valor **Origem A** ao valor **Origem B** e armazena o resultado no destino.

A figura a seguir apresenta o formato da instrução



b) Subtração (SUB)

A instrução **SUB** subtrai o valor **Origem B** do valor **Origem A** e armazena o resultado no destino (Dest).

A figura a seguir apresenta o formato da instrução



c) Multiplicação (MUL)

A instrução **MUL** multiplica o valor origem A pelo valor origem B e armazena o resultado no destino (Dest).

A figura a seguir apresenta o formato da instrução



d) Divisão (DIV)

A instrução **DIV** divide o valor origem A pelo valor origem B e armazena o resultado arredondado no destino (Dest).

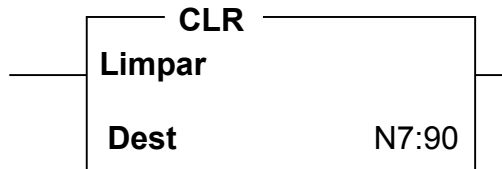
A figura a seguir apresenta o formato da instrução



e) Zeramento (CLR)

Zera todos os bits de uma palavra

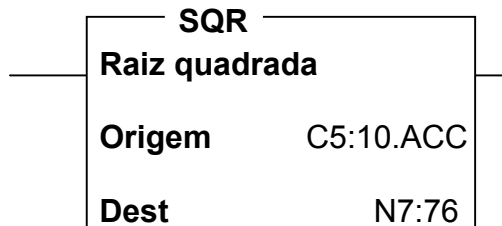
A figura a seguir apresenta o formato da instrução



f) Raiz Quadrada (SQR)

Calcula a raiz quadrada do valor Origem e coloca o inteiro resultante no destino Dest.

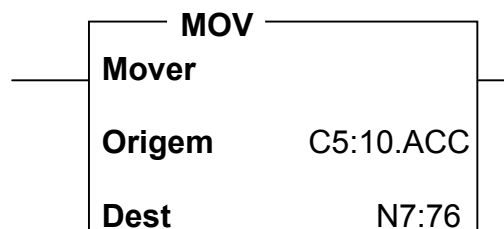
A figura a seguir apresenta o formato da instrução



g) Mover (MOV)

Move o valor da origem para o destino;

A figura a seguir apresenta o formato da instrução



ANEXO 1

SISTEMAS DE NUMERAÇÃO.

Todos nós, quando ouvimos pronunciar a palavra números, automaticamente a associamos ao sistema decimal com o qual estamos acostumados a operar. Este sistema está fundamentado em certas regras que são base para qualquer outro.

Vamos, portanto, estudar estas regras e aplicá-las aos sistemas de numeração *binária*, *octal* e *hexadecimal*. Estes sistemas são utilizados em computadores digitais, circuitos lógicos em geral e no processamento de informações dos mais variados tipos. O número decimal **573** pode ser também representado da seguinte forma:

$$573_{10} = 500 + 70 + 3 \quad \text{ou} \quad 573_{10} = 5 \times 10^2 + 7 \times 10^1 + 3 \times 10^0$$

Isto nos mostra que um dígito no sistema decimal tem, na realidade, dois significados. Um, é o valor propriamente dito do dígito, e o outro é o que está relacionado com a posição do dígito no número (peso). Por exemplo: o dígito **7** no número acima representa 7×10 , ou seja 70, devido a posição que ele ocupa no número. Este princípio é aplicável a qualquer sistema de numeração onde os dígitos possuem "pesos", determinados pelo seu posicionamento. Sendo assim, um sistema de numeração genérico pode ser expresso da seguinte maneira:

$$N = d_n \cdot B^n + \dots + d_3 \cdot B^3 + d_2 \cdot B^2 + d_1 \cdot B^1 + d_0 \cdot B^0$$

Onde:

N = Representação do número na base B

d_n = Dígito na posição n

B = Base do sistema utilizado

n = Valor posicional do dígito

por exemplo, o número **1587** no sistema decimal é representado como:

$$N = d_3 \cdot B^3 + d_2 \cdot B^2 + d_1 \cdot B^1 + d_0 \cdot B^0$$

$$1587_{10} = 1 \cdot 10^3 + 5 \cdot 10^2 + 8 \cdot 10^1 + 7 \cdot 10^0$$

Sistema de Numeração Binário

O sistema binário utiliza dois dígitos (**base 2**), para representar qualquer quantidade. De acordo com a definição de um sistema de numeração qualquer, o número binário **1101** pode ser representado da seguinte forma:

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$1101_2 = 8 + 4 + 0 + 1 = 13_{10}$$

Note que os índices foram especificados em notação decimal, o que possibilita a conversão binária-decimal como descrito acima.

Através do exemplo anterior, podemos notar que a quantidade de dígitos necessários para representar um número qualquer, no sistema binário, é muito maior quando comparada ao sistema decimal. A grande vantagem do sistema binário reside no fato de que, possuindo apenas dois dígitos, estes são facilmente representados por uma chave aberta e uma chave fechada ou, um relé ativado e um relé desativado, ou, um transistor saturado e um transistor cortado; o que torna simples a implementação de sistemas digitais mecânicos, eletromecânicos ou eletrônicos.

Em sistemas eletrônicos, o dígito binário (**0 ou 1**) é chamado de BIT, enquanto que um conjunto de 8 bits é denominado BYTE.

• Conversão Binário - Decimal

A conversão de um número do sistema binário para o sistema decimal é efetuada simplesmente adicionando os pesos dos dígitos binários 1, como mostra o exemplo a seguir

a) 11010_2

b) 1100100_2

Solução:

$$\begin{aligned} \text{a) } 11010_2 &= 1.2^4 + 1.2^3 + 0.2^2 + 1.2^1 + 0.2^0 \\ 11010_2 &= 16 + 8 + 0 + 2 + 0 \\ 11010_2 &= 26_{10} \end{aligned}$$

$$\begin{aligned} \text{b) } 1100100_2 &= 1.2^6 + 1.2^5 + 0.2^4 + 0.2^3 + 1.2^2 + 0.2^1 + 0.2^0 \\ 1100100_2 &= 64 + 32 + 0 + 0 + 4 + 0 + 0 \\ 1100100_2 &= 100_{10} \end{aligned}$$

Conversão Decimal - Binário

Para se converter um número decimal em binário, divide-se sucessivamente o número decimal por 2 (base do sistema binário), até que o último quociente seja 1. Os restos obtidos das

divisões e o último quociente compõem um número binário equivalente, como mostra o exemplo a seguir.

Converter os seguintes números decimais em binário.

a) $23_{10} \quad 23_{10} = 10111_2$

b) $52_{10} \quad 52_{10} = 110100_2$

• **Adição com números Binários**

A adição no sistema binário é efetuada de maneira idêntica ao sistema decimal. Devemos observar, entretanto, que o transporte (vai um) na adição em binário, ocorre quando temos **1+1**. A tabela abaixo ilustra as condições possíveis para adição de Bits.

A + B	S O M A	TRANSPORTE
<i>0 + 0</i>	<i>0</i>	<i>não ocorre</i>
<i>0 + 1</i>	<i>1</i>	<i>não ocorre</i>
<i>1 + 0</i>	<i>1</i>	<i>não ocorre</i>
<i>1 + 1</i>	<i>0</i>	<i>ocorre</i>

Observe, nos exemplos seguintes, como é efetuada uma adição em binário.

Adicionar os seguintes números binários.

a) $101110_2 + 100101_2$

b) $1001_2 + 1100_2$

Solução:

a)
$$\begin{array}{r} 111 \\ 101110 \\ +100101 \\ \hline 1010011 \end{array}$$

b)
$$\begin{array}{r} 1 \\ 1001 \\ +1100 \\ \hline 10101 \end{array}$$

OBSERVAÇÃO:

O termo transporte (*vai um*), utilizado para indicar o envio de um dígito para a posição imediatamente superior do número é chamado de *CARRY* em inglês. Este termo será utilizado a partir de agora, em lugar de "*transporte*", por ser encontrado na literatura técnica.

Solução:

$$\begin{array}{r} \text{a) } 1\ 0\ 0\ 1 \\ 0\ 1\ 1\ 0 \rightarrow 1^\circ \text{ complemento} \\ + \quad \underline{1} \\ 0\ 1\ 1\ 1 \rightarrow 2^\circ \text{ complemento} \end{array}$$

$$\begin{array}{r} \text{b) } 1\ 1\ 0\ 1 \\ 0\ 0\ 1\ 0 \rightarrow 1^\circ \text{ complemento} \\ + \quad \underline{1} \\ 0\ 0\ 1\ 1 \rightarrow 2^\circ \text{ complemento} \end{array}$$

No exemplo **a** o número 9_{10} ($1\ 0\ 0\ 1_2$) tem como segundo complemento $0\ 1\ 1\ 1_2$. O segundo complemento é a representação negativa do número binário, ou seja, -9_{10} representado como sendo $0\ 1\ 1\ 1_2$.

A subtração binária através do segundo complemento, é realizada somando-se o subtrator com o segundo complemento do subtraendo, como mostra o exemplo a seguir.

Subtraia os seguintes números em binários.

$$\text{a) } 13_{10} - 7_{10}$$

$$\text{b) } 6_{10} - 9_{10}$$

Solução:

$$\begin{array}{l} \text{a) } 13_{10} = 1\ 1\ 0\ 1_2 \\ 7_{10} = 0\ 1\ 1\ 1_2 \end{array}$$

Calculando o 2° complemento de 7_{10} ($0\ 1\ 1\ 1_2$), temos:

$$\begin{array}{r} 0\ 1\ 1\ 1 \\ 1\ 0\ 0\ 0 \rightarrow 1^\circ \text{ complemento} \\ + \quad \underline{1} \\ 1\ 0\ 0\ 1 \rightarrow 2^\circ \text{ complemento} \end{array}$$

$$\begin{array}{l} \text{logo:} \\ 13 = 1\ 1\ 0\ 1 \\ \underline{-7} = \underline{+1\ 0\ 0\ 1} \\ 6 \quad 0\ 1\ 1\ 0 \end{array}$$

OBSERVAÇÃO:

Sempre que houver carry do bit mais significativo, ele deverá ser desprezado.

$$\begin{array}{l} \text{b) } 6_{10} = 0\ 1\ 1\ 0_2 \\ 9_{10} = 1\ 0\ 0\ 1_2 \end{array}$$

Calculando o 2° complemento de 9_{10} ($1\ 0\ 0\ 1_2$), temos:

$$\begin{array}{r} 1\ 0\ 0\ 1 \\ 0\ 1\ 1\ 0 \rightarrow 1^\circ \text{ complemento} \\ + \quad \underline{1} \\ 0\ 1\ 1\ 1 \rightarrow 2^\circ \text{ complemento} \end{array}$$

Se no resultado da soma (1 1 0 1) não existe carry, devemos achar o 2º complemento deste número e acrescentar o sinal negativo.

$$\begin{array}{r}
 1\ 1\ 0\ 1 \\
 0\ 0\ 1\ 0 \rightarrow 1^\circ \text{ complemento} \\
 + \quad \underline{1} \\
 0\ 0\ 1\ 1 \rightarrow 2^\circ \text{ complemento}
 \end{array}
 \qquad
 \begin{array}{l}
 \text{então:} \\
 6 - 9 = -3 \\
 \text{ou seja:} \\
 -0\ 0\ 1\ 1
 \end{array}$$

OBSERVAÇÃO:

Podemos achar o 2º complemento de um número binário a partir da seguinte regra: conservamos o primeiro *bit 1* (*um*) menos significativo e efetuamos o 1º complemento dos bits mais significativos (bits da esquerda)

Sistema de Numeração Hexadecimal

O sistema hexadecimal, ou sistema de base 16, é largamente utilizado nos computadores de grande porte, tais como, IBM system 360, IBM system 370, IBM 1130, Honeywell 200, RCA spectra 70, entre outros. Neste sistema são utilizados 16 símbolos para representar cada um dos dígitos hexadecimais, conforme a tabela a seguir:

Nº DECIMAL	DÍGITO HEXADECIMAL	Nº BINÁRIO
0	0	0 0 0 0
1	1	0 0 0 1
2	2	0 0 1 0
3	3	0 0 1 1
4	4	0 1 0 0
5	5	0 1 0 1
6	6	0 1 1 0
7	7	0 1 1 1
8	8	1 0 0 0
9	9	1 0 0 1
10	A	1 0 1 0
11	B	1 0 1 1
12	C	1 1 0 0
13	D	1 1 0 1
14	E	1 1 1 0
15	F	1 1 1 1

Note que as letras **A, B, C, D, E, F** representam dígitos associados às quantidades **10,11,12,13,14 e 15** respectivamente.

- **Conversão Hexadecimal - Decimal**

Novamente aplicamos para o sistema hexadecimal a definição de um sistema de numeração qualquer. Assim temos:

$$N = \mathbf{dn.16^n} + \dots + \mathbf{d2.16^2} + \mathbf{d1.16^1} + \mathbf{d0.16^0}$$

Para se efetuar a conversão, basta adicionar os membros da segunda parcela da igualdade, como ilustra o exemplo a seguir:

a) 23_{16}

b) $3B_{16}$

S o l u ç ã o :

a) $23_{16} = 2 \times 16^1 + 3 \times 16^0$
 $23_{16} = 2 \times 16 + 3 \times 1$
 $23_{16} = 35_{10}$

b) $3B_{16} = 3 \times 16^1 + 11 \times 16^0$
 $3B_{16} = 3 \times 16 + 11 \times 1$
 $3B_{16} = 59_{10}$

Observe que o dígito hexadecimal "**B**", no exemplo (b), é equivalente ao número 11 decimal, como mostra a tabela apresentada anteriormente.

- **Conversão Decimal - Hexadecimal**

A conversão decimal-hexadecimal é efetuada através das divisões sucessivas do número decimal por 16, como demonstrado no exemplo a seguir.

a) 152_{10}

b) 249_{10}

S o l u ç ã o :

a) 152

b) 249

Números Decimais Codificados em Binário (BCD)

Como já foi discutido anteriormente, os sistemas digitais, em geral, trabalham com números binários. Com o intuito de facilitar a comunicação homem-máquina, foi desenvolvido um código que representa cada dígito decimal por um conjunto de 4 dígitos binários, como mostra a tabela seguinte:

EXERCÍCIOS

01 – Desenvolva um programa para ligar e desligar uma lâmpada utilizando um botão liga NA (verde) e um botão desliga NF (vermelho). Use instruções de bit: XIC e OTE.

02 - Desenvolva um programa para ligar e desligar uma lâmpada utilizando um botão liga NA (verde) e um botão desliga NF (vermelho). Use instruções de bit: XIC, XIO, OTL e OTU.

03 - Desenvolva um programa para ligar e desligar uma lâmpada utilizando apenas o botão liga NA (verde). Use instruções de bit: XIC, XIO, OTE e arquivo B3.

04 - Desenvolva um programa para ligar e desligar uma lâmpada utilizando apenas o botão liga NA (verde). Use instruções de bit: XIC, XIO, OSR, OTE e arquivo B3.

05 - Desenvolva um programa para ligar três lâmpadas em sequência quando o botão liga NA (verde) for acionado por três vezes consecutivas, e desligar, as três lâmpadas ao mesmo tempo, quando o botão desliga NF (vermelho) for acionado. Use instruções de bit: XIC, XIO, OSR, OTL e OTU.

06 - Desenvolva um programa para ligar três lâmpadas em sequência quando o botão liga NA (verde) for acionado por três vezes consecutivas, e desligar, as três lâmpadas ao mesmo tempo, quando o botão desliga NF (vermelho) for acionado. Use instruções de bit: XIC, XIO, OSR, OTE e arquivo B3.

07 - Desenvolva um programa para ligar três lâmpadas em sequência quando o botão liga NA (verde) for acionado por três vezes consecutivas, e desligar, as três lâmpadas ao mesmo tempo, quando o botão liga NA (verde) for acionado pela quarta vez. Use instruções de bit: XIC, XIO, OSR, OTE e arquivo B3.

08 - Desenvolva um programa para ligar um motor quando o botão liga NA (verde) for acionado, sendo que o mesmo desligará automaticamente após 10s ou quando o botão desliga NF (vermelho) for acionado. Uma lâmpada verde deverá estar acesa sinalizando o motor desligado e uma vermelha o motor ligado. Use instruções XIC, XIO, OTE, arquivo B3 e de temporização TON.

09 - Desenvolva um programa para ligar um motor quando o botão liga NA (verde) for acionado, sendo que o mesmo desligará automaticamente após 10s ou quando o botão desliga NF (vermelho) for acionado. Uma lâmpada verde deverá estar acesa sinalizando o motor desligado e uma vermelha o motor ligado. Use instruções XIC, XIO, OTE, arquivo B3 e de temporização TOF.

10 - Desenvolva um programa para ligar um motor quando o botão liga NA (verde) for acionado. O motor funcionará obedecendo o seguinte ciclo de operação: 10s ligado e 5s

desligado. O ciclo de operação será interrompido quando o botão desliga NF (vermelho) for acionado. Uma lâmpada verde deverá estar acesa sinalizando o motor desligado e uma vermelha o motor ligado. Use instruções XIC, XIO, OTE, arquivo B3 e de temporização TON e TOF.

11 - Desenvolva um programa que simule o funcionamento de uma sinaleira de forma que a Lâmpada verde fique acesa por 12s, a Amarela por 3s e a vermelha por 15s. O ciclo será iniciado quando o botão liga NA (verde) for acionado e terminado quando o botão desliga NF (vermelho) for acionado. Quando o ciclo for terminado a Lâmpada amarela deverá piscar em intervalos de 3s. Use instruções XIC, XIO, OTE, arquivo B3 e de temporização TON ou TOF.

12 - Desenvolva um programa para ligar um motor quando o botão liga NA (verde) for acionado, sendo que o mesmo desligará automaticamente após 10s ou quando o botão desliga NF (vermelho) for acionado. Uma lâmpada verde deverá estar acesa sinalizando o motor desligado e uma vermelha o motor ligado. Use instruções XIC, XIO, OTE, arquivo B3 e de temporização RTO.

13 - Desenvolva um programa para ligar um motor quando o botão liga NA (verde) for acionado. Após 10 voltas o motor deverá desligar automaticamente ou quando o botão desliga NF (vermelho) for acionado. Uma lâmpada verde deverá estar acesa sinalizando o motor desligado e uma vermelha o motor ligado Use instruções XIC, XIO, OTE, arquivo B3 e de contagem CTU e RES.

14 - Desenvolva um programa para ligar um motor quando o botão liga NA (verde) for acionado. Após 10 voltas o motor deverá parar automaticamente ou quando o botão desliga NF (vermelho) for acionado. Uma lâmpada verde deverá estar acesa sinalizando o motor desligado e uma vermelha o motor ligado. Quando o motor for desligado o acumulado do contador deverá ser zerado. Use instruções XIC, XIO, OTE, arquivo B3 e de contagem CTD e RES.

15 - Desenvolva um programa para ligar um motor quando o botão liga NA (verde) for acionado. O motor funcionará obedecendo o seguinte ciclo ininterrupto de operação: 10 voltas e 5s desligado. O ciclo de operação será interrompido quando o botão desliga NF (vermelho) for acionado. Uma lâmpada verde deverá estar acesa sinalizando o motor desligado e uma vermelha o motor ligado. Use instruções XIC, XIO, OTE, arquivo B3 e de temporização TON ou TOF e CTU ou CTD.

16 - Desenvolva um programa para ligar um motor quando o botão liga NA (verde) for acionado. O motor funcionará obedecendo o seguinte ciclo ininterrupto de operação: 10 voltas no sentido horário e 5s desligado / 10 voltas no sentido anti-horário e 5s desligado. O ciclo de operação será interrompido quando o botão desliga NF (vermelho) for acionado. Uma lâmpada verde deverá estar acesa sinalizando o motor desligado e uma vermelha o motor ligado. Use instruções XIC, XIO, OTE, arquivo B3 e de temporização 2TON ou 2TOF e 2CTU ou 2CTD.

17 - Desenvolva um programa para ligar um motor quando o botão liga NA (verde) for acionado. O motor funcionará obedecendo o seguinte ciclo ininterrupto de operação: 10 voltas no sentido horário e 5s desligado / 10 voltas no sentido anti-horário e 5s desligado. O ciclo de operação será interrompido quando o botão desliga NF (vermelho) for acionado. Uma lâmpada

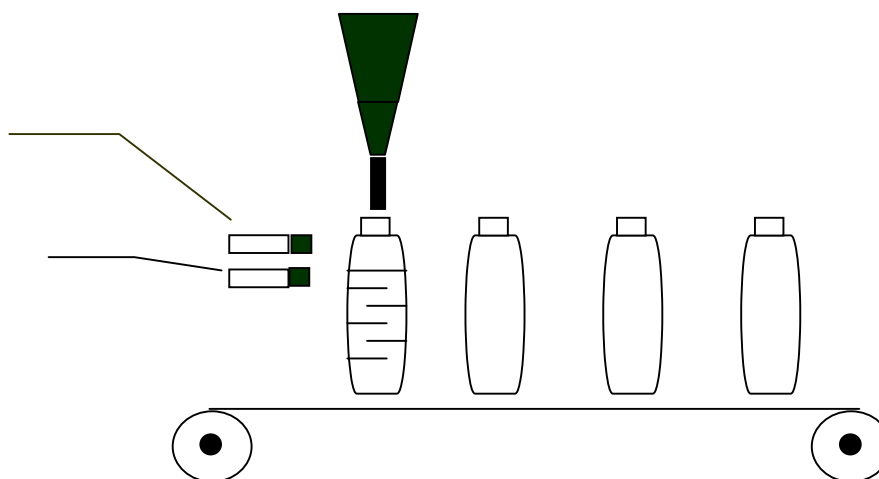
verde deverá estar acesa sinalizando o motor desligado e uma vermelha o motor ligado. Use instruções XIC, XIO, OTE, arquivo B3 e de temporização 1TON ou 1TOF e 1CTU ou 1CTD.

18 - Desenvolva um programa para ligar e desligar uma lâmpada em intervalos de 3s.

O ciclo será iniciado quando o botão liga NA (verde) for acionado e terminado quando o botão desliga NF (vermelho) for acionado. Use instruções XIC, XIO, OTE, arquivo B3 e de temporização 1TON e de comparação a escolher.

19 - Desenvolva um programa que simule o funcionamento de uma sinaleira simples de forma que a Lâmpada verde fique acesa por 12s, a amarela por 3s e a vermelha por 15s. O ciclo será iniciado quando o botão liga NA (verde) for acionado e terminado quando o botão desliga NF (vermelho) for acionado. Quando o ciclo for terminado a Lâmpada amarela deverá piscar em intervalos de 3s. Use instruções XIC, XIO, OTE, arquivo B3 e de temporização 1TON ou 1TOF e de comparação a escolher.

20 - Desenvolva um programa para controlar o enchimento de garrafas com produto químico. Quando o botão liga NA (verde) for acionado a esteira é ligada e movimenta as garrafas até o bico de enchimento, a garrafa é detectada através de um sensor, a esteira deve para e abrir a válvula do bico de enchimento para encher a garrafa, o nível é detectado através de um sensor, depois de detectado o nível devemos esperar por 10s e acionar a esteira novamente para recomençar o ciclo, encher a próxima garrafa. Devemos contar também a quantidade de garrafas cheias (10 garrafas). Após a contagem, deverá ser acionado um alarme e o ciclo só recomençará se o botão de reconhecimento de alarme for acionado. Quando o botão desliga NF (vermelho) for acionado o ciclo será interrompido. Use instruções XIC, XIO, OTE, OTL, OTU, arquivo B3, de temporização 1TON ou 1TOF e 1CTU ou 1CTD.



21 - Desenvolva um programa que simule o funcionamento de uma sinaleira dupla de forma que a Lâmpada verde fique acesa por 12s, a amarela por 3s e a vermelha por 15s. O ciclo será iniciado quando o botão liga NA (verde) for acionado e terminado quando o botão desliga NF (vermelho) for acionado. Quando o ciclo for terminado a Lâmpada amarela deverá piscar em intervalos de 3s. Use instruções XIC, XIO, OTE, arquivo B3 e de temporização 1TON ou 1TOF e de comparação a escolher.

22 - Desenvolva um programa que converta Graus Fahrenheit em Celcius, aplicando a fórmula $C = \lfloor 5(F - 32) \rfloor / 9$. Quando a temperatura estiver entre 15 e 35 graus Celcius a lâmpada verde estará acesa e quando a temperatura estiver fora desta faixa, abaixo de 14 ou acima de 36 graus Celcius a lâmpada vermelha acenderá. Use Instruções de bit, matemáticas e de comparação. Não se esqueça de usar o arquivo N7 para entrada e armazenamento de dados.