

Controladores Lógicos Programáveis

Prof. Marcelo Ricardo Stemmer

marcelo.stemmer@ufsc.br

DAS/CTC/UFSC

Sumário

1. Introdução aos CLPs

1.1. Definição

1.2. Áreas de aplicação

1.3. Histórico

2. Arquitetura e Características Gerais dos CLPs

3. Formas de IHM para CLPs

3.1. Interfaceamento por meio das E/S

3.2. Interfaceamento por meio de IHM dedicadas

3.3. Interfaceamento por meio de TP

3.4. Interfaceamento por meio de PC

Sumário

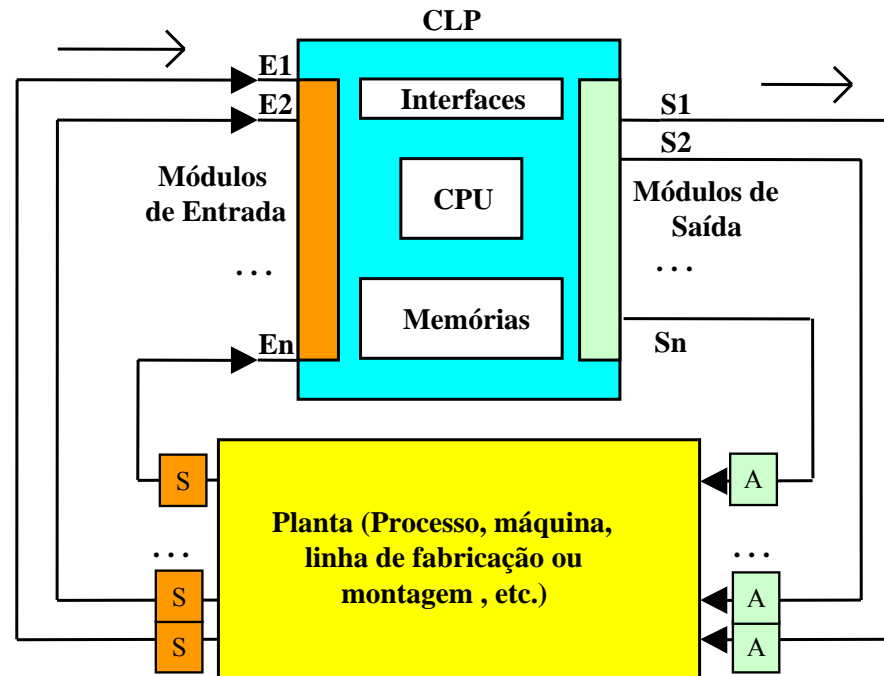
- 4. Módulos de Entrada e Saída**
- 5. Organização interna de memória**
- 6. Programação de CLPs**
 - 6.1. Introdução: programação segundo IEC 61131-3**
 - 6.2. Revisão de álgebra booleana**
 - 6.3. Linguagem LD**
 - 6.4. Linguagem GRAFCET/SFC**
 - 6.5. Linguagem IL**
 - 6.6. Linguagem FBD**
 - 6.7. Linguagem ST**
- 7. Normas e Padrões para CLPs**
- 8. Interligação de CLPs via rede**
- 9. Visão do mercado de CLP no Brasil**

Definição de CLP

- CLP (PLC, CP): equipamento eletrônico programável usado para controle, intertravamento, seqüenciamento e monitoração de máquinas ou processos.
- Possui hardware universal, aplicável a todos os tipos de processos.
- CLPs atuais baseados em microprocessadores ou microcontroladores.

Princípio de Funcionamento

- CLP lê sinais de entrada (sensores), executa lógicas programadas e fornece sinais de saída (atuadores), que atuam sobre a máquina ou processo.



Áreas de Aplicação

- Aplicação em quase todos os setores industriais envolvendo:
 - controle de processos;
 - automação da manufatura;
 - integração de sistemas de automatização;
 - linhas de fabricação e montagem;
 - automação predial;
 - controle de subestações de energia;
 - etc...

Áreas de Aplicação

- **Funções:**
 - **Controle:** PID industrial;
 - **Seqüenciamento:** definição da seqüência de operações em linhas de fabricação e montagem;
 - **Intertravamento:** ação y só pode ser executada se ação x foi concluída;
 - **Supervisão/monitoração:** visualização do andamento do processo, intervenção do operador.

Aplicações Usuais

- **Máquinas-ferramenta:** intertravamento e seqüenciamento das operações; controle de posição dos eixos, torque, velocidade de avanço, aceleração e outras;
- **Controlador PID:** controle de posição, rotação, velocidade, temperatura, pressão, vazão, força, potência e outras;
- **Seqüenciamento/intertravamento:** linhas de produção e montagem automatizadas.

Histórico

- CLPs concebidos para substituir lógicas de comando com relês.
- Inicialmente aplicados na indústria automobilística.
- Indústria automotiva: necessidade de alteração da lógica de comando das linhas de montagem com relês a cada alteração do modelo de carro.
- Primeiro usuário: General Motors.
- Mais tarde passaram a ter larga aplicação em todos os setores industriais.

Datas Importantes

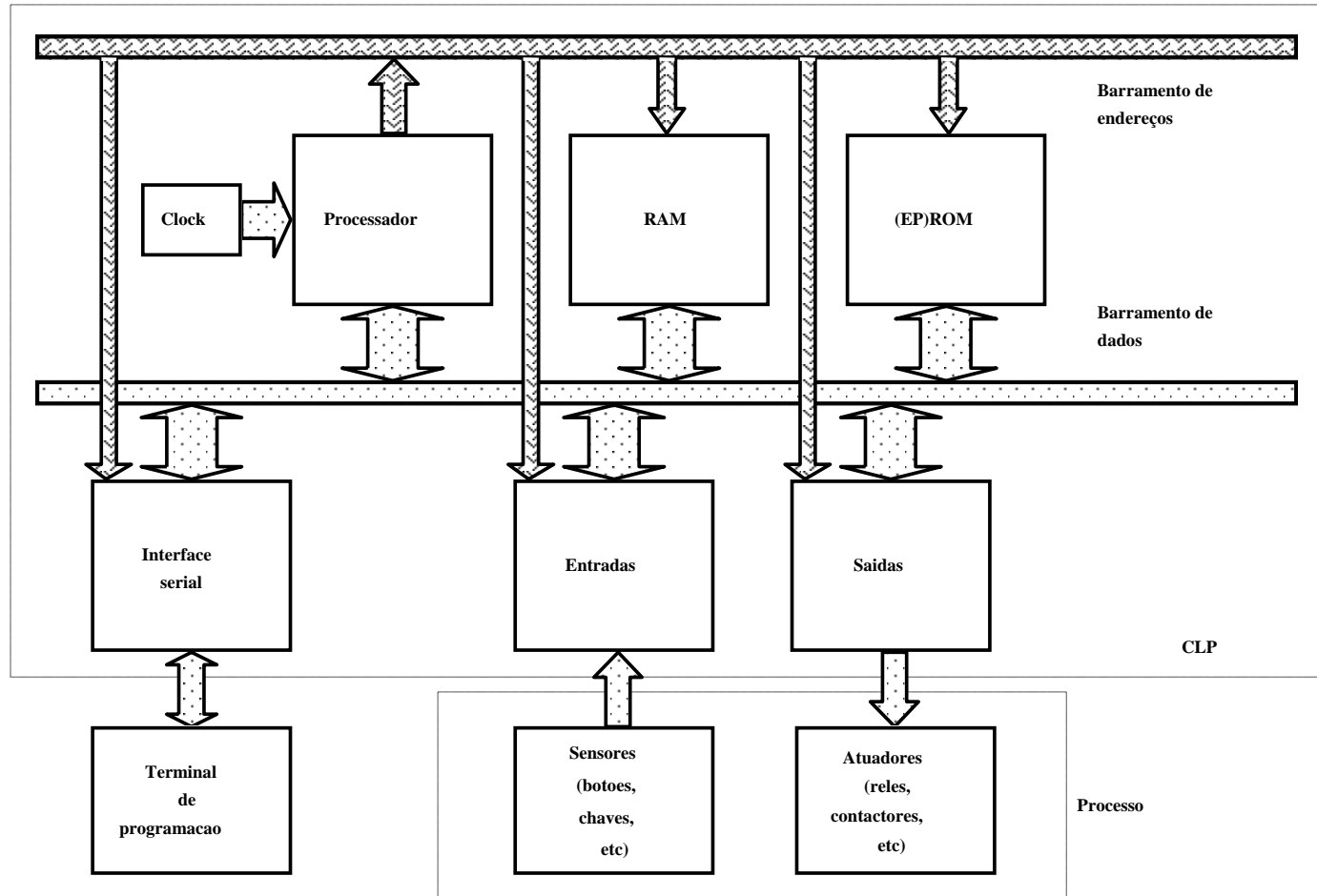
- **1968:** projeto do primeiro CLP para a General Motors pelo Eng. Dick Morley (fundador da MODICON, MOdular DIgital CONtroller), baseado em componentes discretos;
- **1971:** primeira aplicação de CLPs fora da indústria automobilística;
- **1975:** introdução do controlador PID analógico nos CLPs;
- **1977:** realização dos CLPs com microprocessadores em lugar de componentes discretos;
- **1978 em diante:** CLPs ganham larga aceitação industrial;

Vantagens dos CLPs

- Redução do tempo de implementação e alteração de lógicas por Software;
- Elevada confiabilidade;
- Dimensões e peso reduzido;
- Capacidade de integração com outros sistemas inteligentes;
- Construção robusta, adequada a ambientes industriais;
- Projeto modular, de fácil expansão.

Arquitetura Básica

- Construção eletro-mecânica apropriada para modularidade.
- Módulos tem acesso fácil aos barramentos.



Características Gerais

- CLPs atuais baseados em microcontroladores (microprocessadores com timers, DMA, I/O, etc, integrados no mesmo "chip").
- Exemplos de tipos utilizados: 8031, 8051, 80188, 80196, 80535, NEC V25, PIC, entre outros.
- Alguns CLPs utilizam ainda microprocessadores, como: Z80, 8085, 8088, 8086, 80286, 386, 486, Pentium, etc.

Características Gerais

- Programa do CLP introduzido através de TP, que só fica conectado ao CLP durante a entrada ou alteração de um programa.
- TPs são equipamentos simples, dotados de teclado e display.
- TPs mais sofisticados incluem monitor de vídeo, gravador de EPROM e software gráfico para programação e monitoração.
- Cada vez mais usados computadores do tipo PC, Laptops e Notebooks como TPs.

Características Gerais

- **Módulos de entrada:**
 - constituem um sistema de aquisição de dados.
 - Leitura de sinais digitais e analógicos.
 - Leitura de botões, chaves fim-de-curso, termopares, pressostatos, extensômetros, sensores de proximidade, encoders, etc.
 - Sinais com potência elevada isolados com optoacopladores.

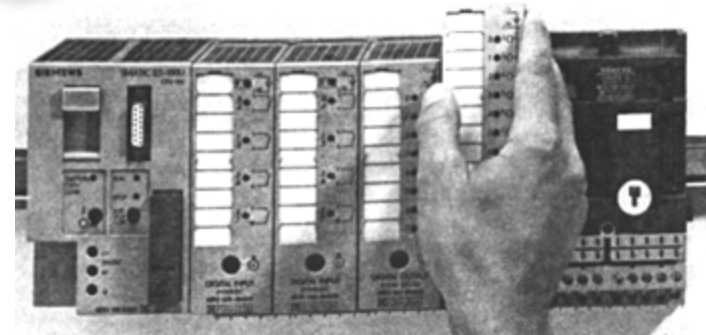
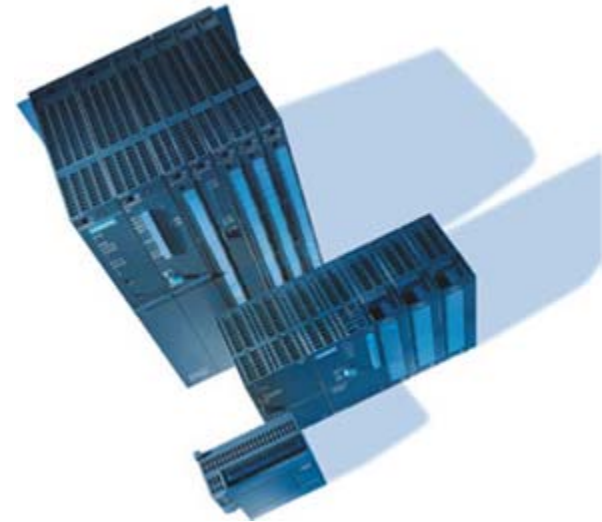
Características Gerais

- **Módulos de saída:**
 - fornecem sinais digitais ou analógicos.
 - energizam dispositivos de operação e sinalização como contactores, solenóides, motores, válvulas e atuadores diversos.
 - Saídas podem ser temporizadas, com regulagem dos retardos.

Características Gerais

- Os sinais de E/S podem ser de CC ou CA, em diferentes níveis de tensão e potência.
- Módulos montados em um "rack" de aço junto aos equipamentos que deverão controlar.
- Módulos acoplados por extensão dos barramentos de dados e endereços.

Formas Construtivas



IHM para CLPs

- É necessária uma interface entre o CLP e o operador humano para operação "on-line" e "off-line".
- Operação "on-line":
 - entrada de dados e comandos;
 - supervisão e visualização do andamento do processo sob controle do CLP.
- Operação "off-line":
 - entrada, atualização e correção dos programas de aplicação.

IHM para CLP

- Formas de Interfaceamento:
 - utilizando as entradas e saídas do CLP;
 - utilizando painéis dedicados para IHM (PI);
 - interligando o CLP a um TP/TI;
 - interligando o CLP a um PC.

IHM com E/S

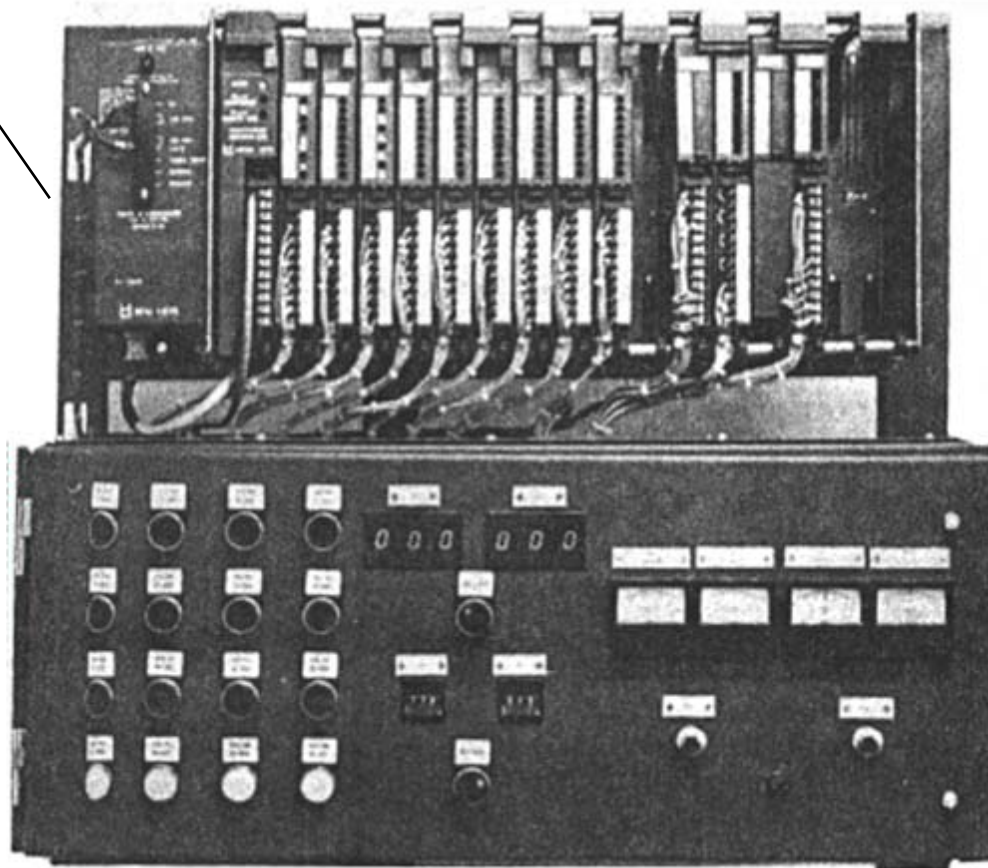
- Método utilizado para operação "on-line".
- Entrada de dados: referência para malhas de controle ou comandos do tipo ON/OFF.
- Dados introduzidos por botões, chaves seletoras, potenciômetros, etc., ligados à entradas pré-definidas do CLP.
- Técnica usada também para supervisão e visualização: lâmpadas, LEDS, indicadores digitais ou analógicos conectados à saídas pré-definidas do CLP.

IHM com E/S

- Desvantagens da técnica:
 - Inflexibilidade:
 - » alteração nas ligações de E/S requer alteração no programa do CLP.
 - » alteração do programa pode implicar na necessidade de alteração da fiação externa.
 - Disponibilidade de menos E/S para processo em si.

IHM com E/S

CLP

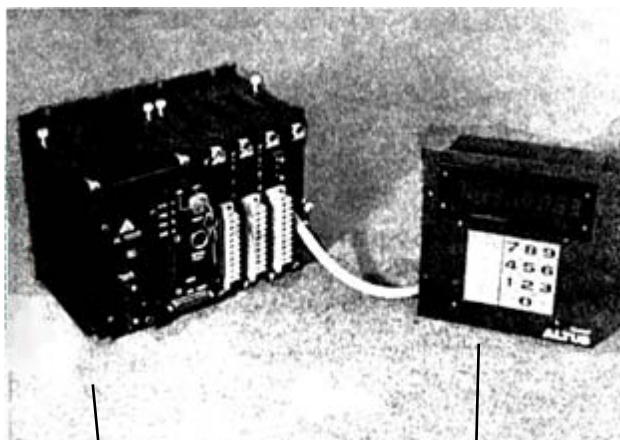


Painel
sinótico

IHM com PI

- PIs (Painéis Inteligentes) usam interface serial RS-232-C para a entrada de dados e visualização.
- PIs compostos de teclado (membrana), display (cristal líquido), RAM, EPROM, interface serial RS-232-C e um processador.
- PI é usado para a comunicação "on-line".
- Características:
 - fácil introdução e visualização de dados;
 - dimensões reduzidas e portabilidade;
 - cablagem mínima (um cabo de interface serial);
 - fácil alteração do modo de operação do PI por reprogramação da EPROM ou EAPROM.

IHM com PI



PI

CLP

PI mais sofisticado



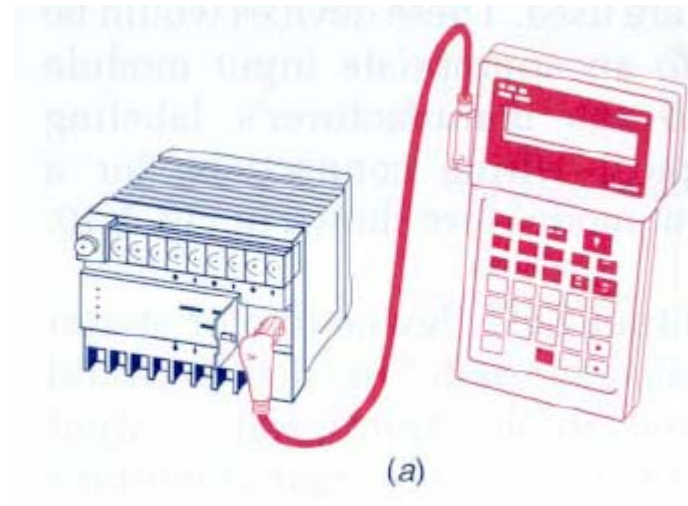
IHM com TP

- O TP (Terminal de Programação, ou TI) é usado para a introdução, alteração e depuração de programas (modo OFF-Line).
- TP concebido para utilização no chão de fábrica.
- TP é portátil e com bateria interna, de forma a independe de tomadas de energia nos locais de uso.
- TPs iniciais eram PIs com mais memória, teclado e display melhores, baseados em “single-board computers”.
- Tendência: substituição por PCs, especialmente portáteis.

IHM com TP



IHM com TP



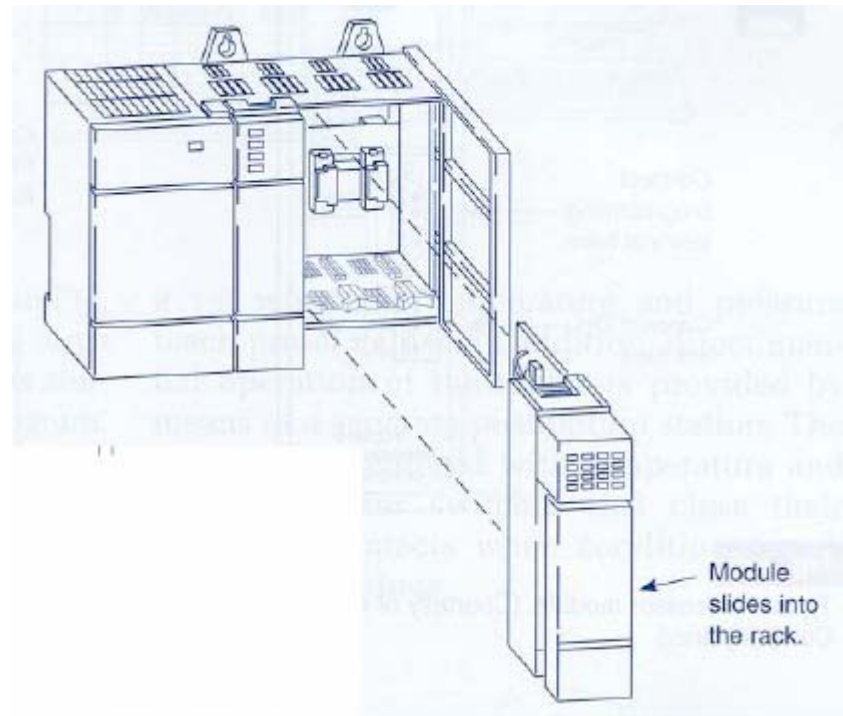
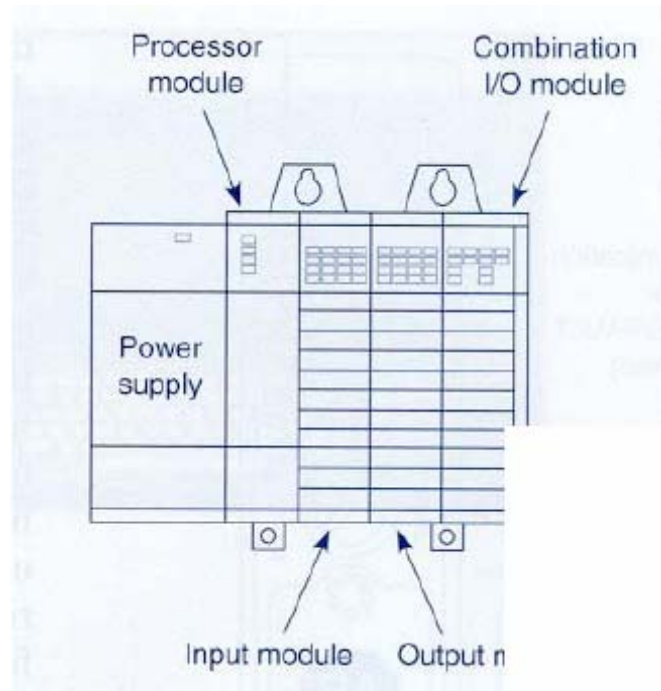
IHM com PC

- PCs cada vez mais usados para programação e visualização com CLPs, devido a:
 - Redução do custo;
 - Redução das dimensões e peso (Laptops, Notebooks, Palmtops).
- Softwares para CLPs em PCs incluem:
 - programação gráfico-interativa;
 - armazenamento de dados e programas do CLP em arquivos no disco do PC;
 - softwares supervisórios (SCADA):
 - » visualização do andamento do processo controlado pelo CLP na tela do PC;
 - » entrada de dados e parâmetros "on-line" para o CLP por intermédio do PC, incluindo a possibilidade de realização de cálculos complexos no PC e o envio dos resultados ao CLP;
 - » geração de relatórios e gráficos relativos ao andamento dos processos controlados pelo CLP no PC;

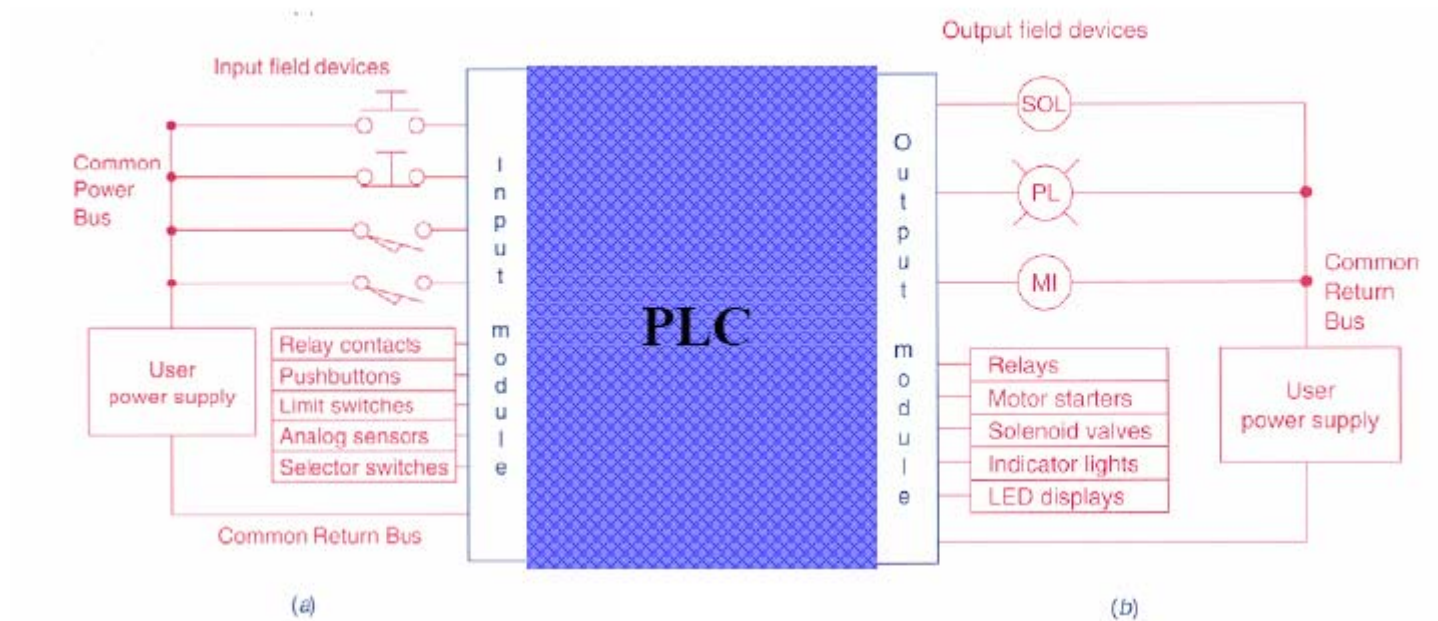
Módulos de E/S

- Módulo básico contém CPU, memórias RAM e EPROM, interface serial e eventualmente um pequeno número de entradas e saídas.
- CLP pode ser expandido pelo acréscimo de novos módulos com entradas e saídas adicionais de diferentes tipos.
- Módulos de E/S contém 2, 4, 8, 16, 32 circuitos montados em caixas isoladas contra poeira, óleo, umidade, altas temperaturas (usualmente até 60°C) e sobretensão (isolação galvânica).

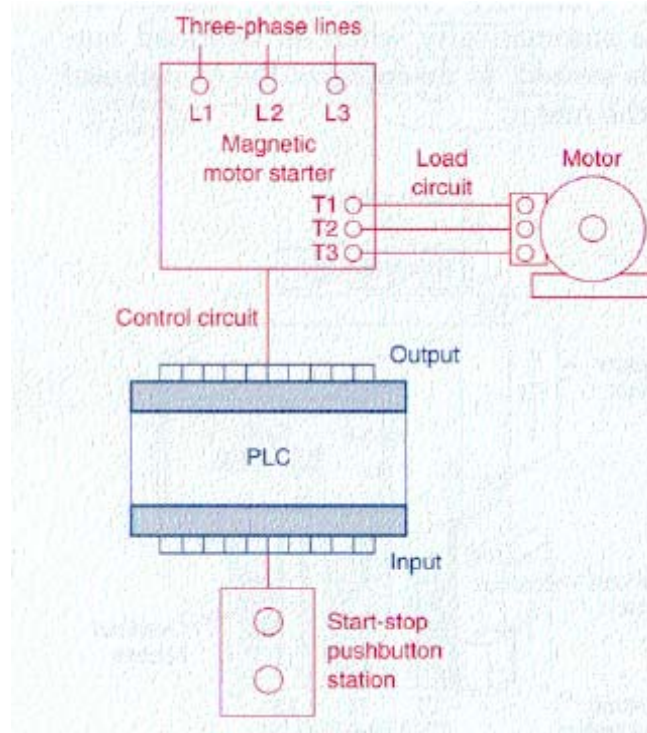
Módulos de E/S



Módulos de E/S



Módulos de E/S



Módulos de E/S

- **Módulos de E/S Analógicos:**

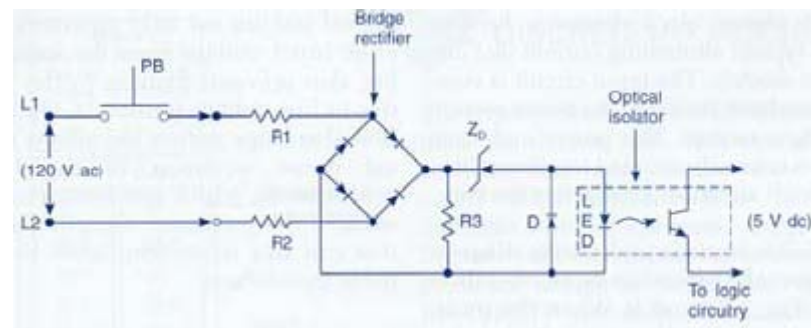
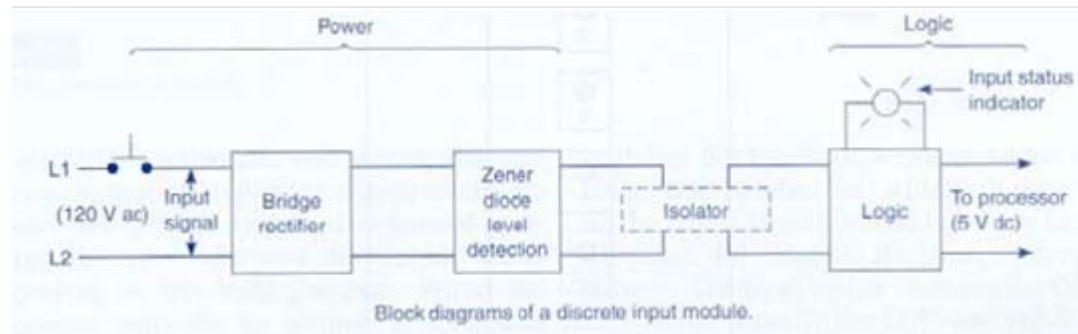
- 110V / 220V CA ($\pm 15\%$) – sinal de tensão
- 0 a 5V CC / 0 a 10V CC / ± 10 V CC – sinal de tensão
- 4 a 20mA / 0 a 20mA - sinal de corrente

- **Módulos de E/S Digitais:**

- 0 ou 5V CC (TTL)
- 0 ou 12V CC
- -12 ou +12V CC
- 0 ou 24V CC

Módulos de E/S

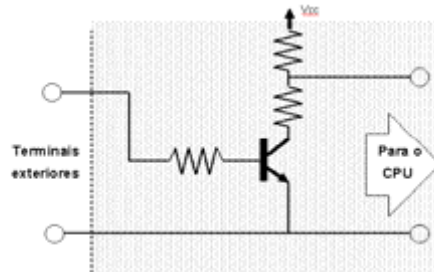
- **Entradas AC:**



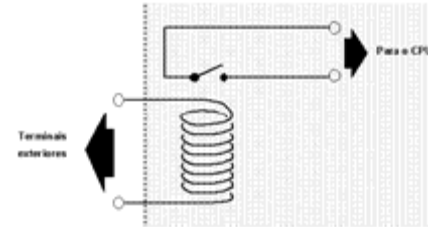
Módulos de E/S

- **Entradas DC:**

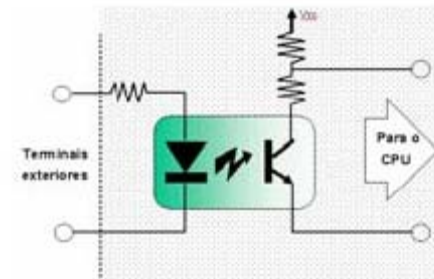
Transistor



Relé



Opto-coupler

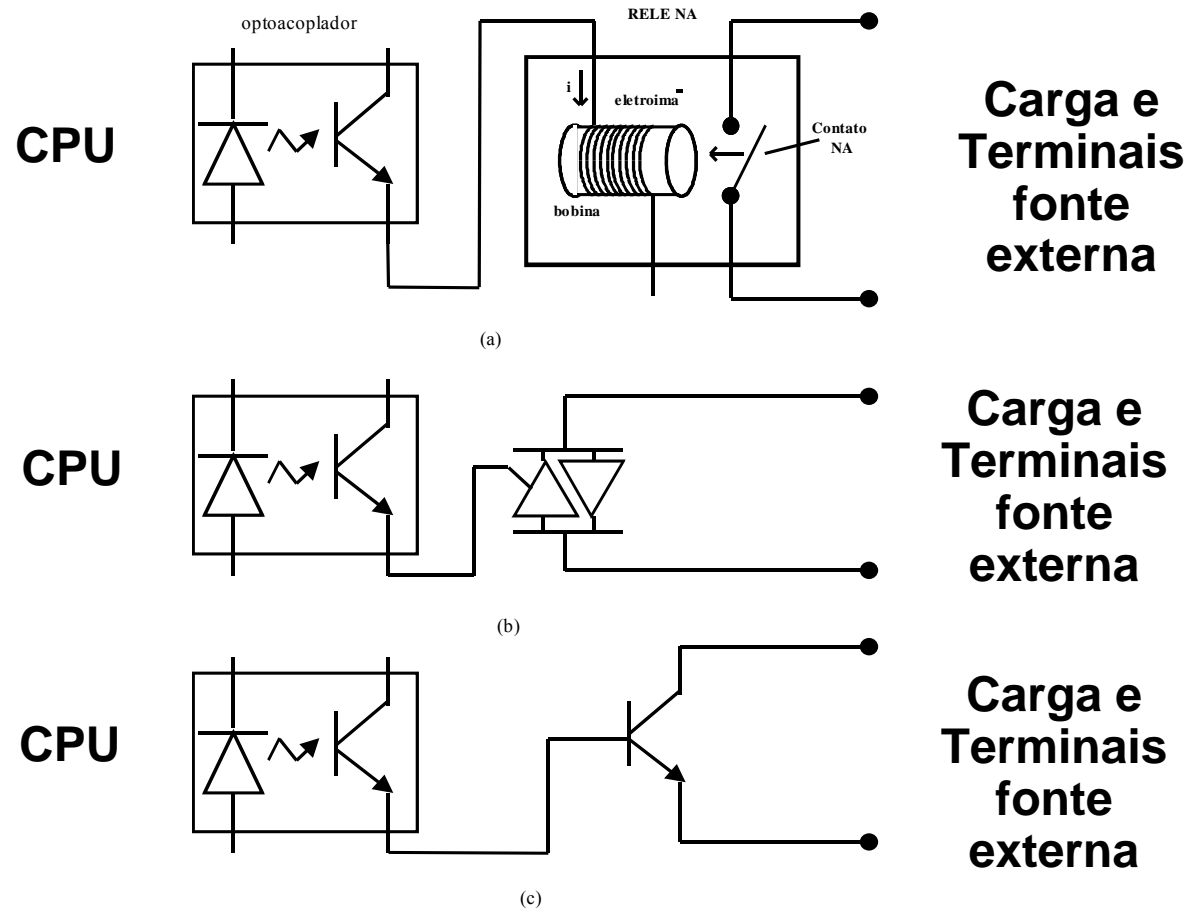


Módulos de E/S

- **Saídas CA e CC:**

- **Saídas a relê:** podem ser usadas em circuitos CA e CC, mas, como adotam comutação mecânica (relé NA), não permitem frequências de chaveamento muito altas.
- **Saídas a TRIAC:** também podem ser usados em circuitos CA e CC, mas permitem comutação rápida.
- **Saídas a Transistor coletor aberto:** são usadas em circuitos CC e permitem comutação rápida.

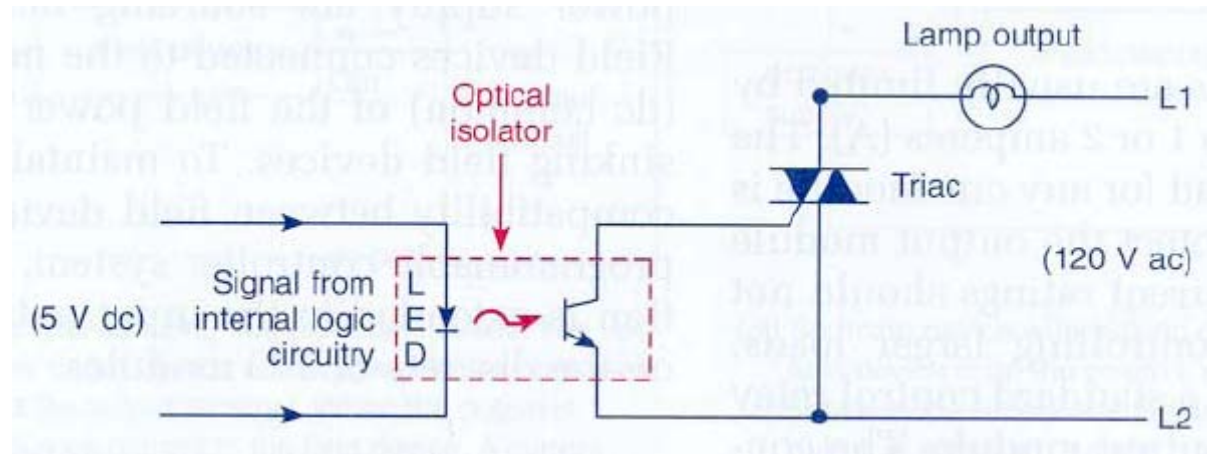
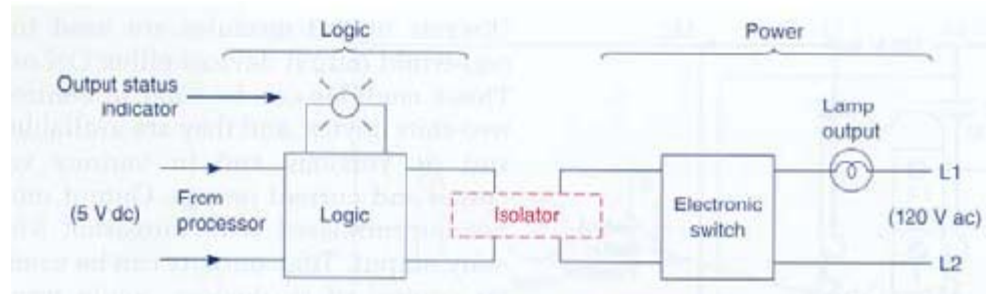
Módulos de E/S



- Saídas: (a) a relé; (b) a TRIAC; (c) a transistor coletor aberto

Módulos de E/S

- Saídas CA:



Módulos de E/S

- **Módulos de Comunicação:**
 - interfaces RS232C, RS422, RS423 ou RS485;
 - interfaces para LAN (Local Area Network);
 - geração de relatórios: incluem interface para impressora;
- **Módulos Especiais:**
 - controlador PID: usados no controle de temperatura, posição, vazão, pressão, velocidade, etc.
 - contadores de pulsos: usados na leitura de encoders digitais, medição de frequência, etc.
 - controle de motores de passo.

Módulos de E/S

- Em função do número de E/S pode-se classificar os CLPs em:
 - **Pequenos:** até 256 E/S, com custo da ordem de U\$300 a U\$5.000;
 - **Médios:** 256 até 1024 E/S, com custo da ordem de U\$5.000 a U\$30.000;
 - **Grandes:** mais de 1024 E/S, com custo da ordem de U\$30.000 até aproximadamente U\$80.000.

Organização de Memória

- RAM: protegida por baterias para impedir perda de programas e dados por falta de energia.
- EPROM ou EAPROM (Flash): utilizada para os programas "permanentes".
- Áreas de memória:

Grupo	Utilização
Memória do Sistema	Monitor do CLP Área de Trabalho do Monitor
Memória de Aplicação	Tabela de E/S Tabela de Dados Programa de Aplicação

Organização de Memória

- Monitor: inicialização, auto-testes, comunicação com periféricos (TP, PI, PC, impressora), carregamento e supervisão de programas (em EPROM).
- Área de trabalho do monitor: área de RAM para dados e variáveis do Monitor.
- Tabela de E/S: área de RAM usada para armazenar e atualizar valores das entradas e saídas (espelho de E/S).
- Tabela de dados: área de RAM que inclui os dados e variáveis do programa de aplicação (valores de contadores, temporizadores, constantes, etc).
- Área de programas de aplicação: área RAM para programas do usuário (opcionalmente em EPROM, se alterações esperadas pouco freqüentes).

Tabelas de E/S e de Dados

- **Variáveis** são designadas por **Endereços**, que são símbolos alfanuméricos, compostos de letras indicando o tipo de dado (entrada, saída, auxiliar, memória, etc) e de números que referenciam o módulo e a porta de I/O correspondentes ou simplesmente indexam os dados.
- **Constantes** são designadas por um prefixo (ex: KM ou #) e um valor numérico.

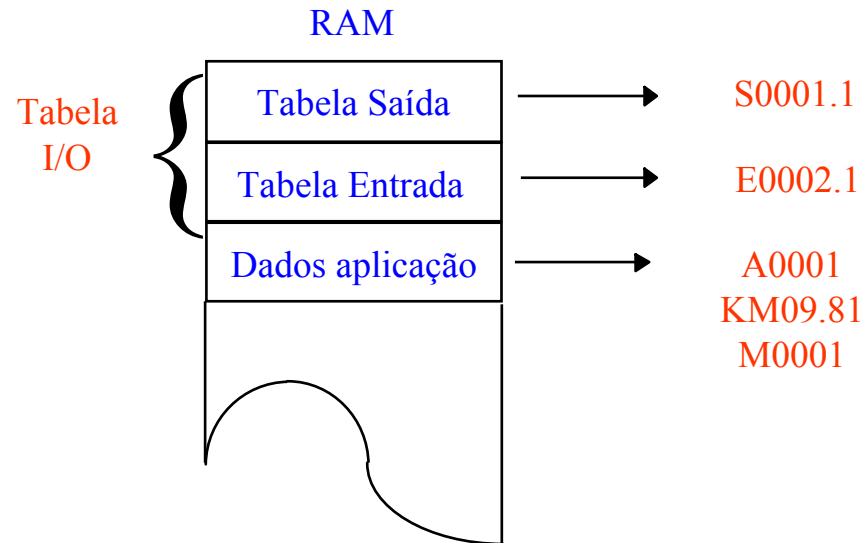
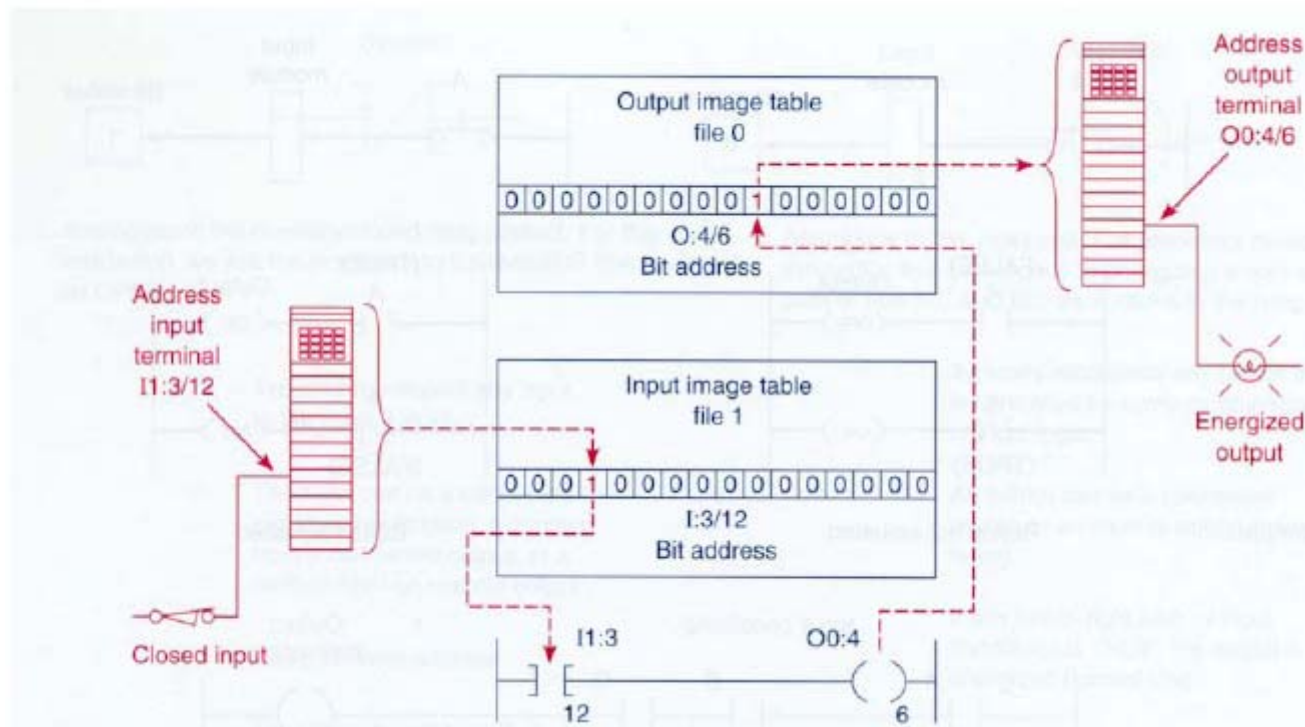


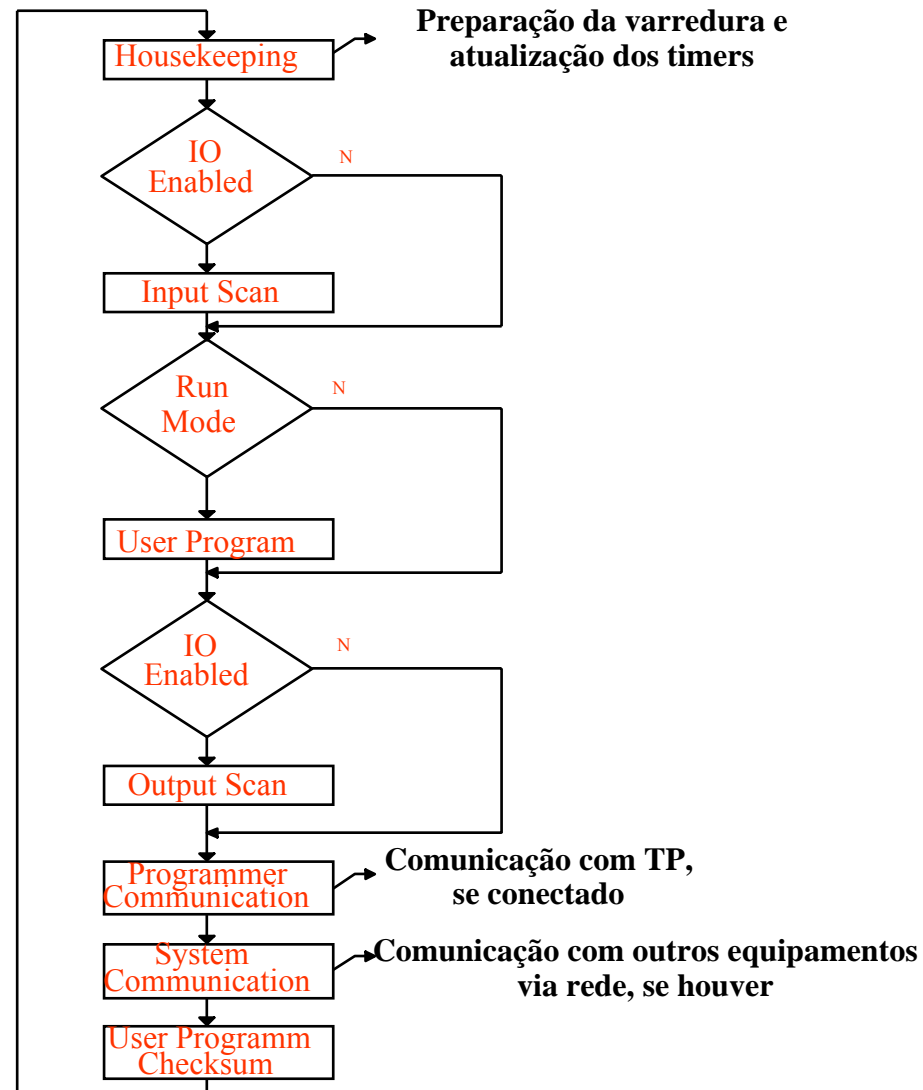
Tabela de E/S



Programação de CLPs

- Os programas de um CLP são sempre executados de forma cíclica (laço, "looping").
- Após a execução da última instrução, reiniciada a execução a partir da primeira linha: CICLO DE VARREDURA.
- No início do ciclo, todas as entradas são lidas e seu estado copiado na tabela de E/S.
- No final do ciclo a tabela de E/S é varrida e conteúdo das saídas copiado nas saídas físicas do CLP.

Ciclo de Varredura de um CLP

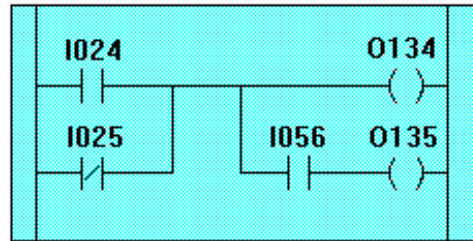


Configuração do CLP

- O CLP necessita ser configurado antes da utilização.
- A configuração é feita por meio de um arquivo contendo:
 - **Módulos de E/S:** número e tipo das entradas e saídas;
 - **Operandos:** número e tipo dos operandos e suas características (ex. retentividade);
 - **Parâmetros gerais:** tipo de CPU, endereço em rede, período de interrupções, etc.
 - **Parâmetros de Comunicação:** taxa de transmissão, paridade, etc.

Linguagens (IEC 61131-3)

1. **IL** (Instruction List): linguagem textual semelhante a Assembly;
2. **LD** (Ladder Diagram): linguagem gráfica criada para facilitar a implantação de lógicas originalmente realizadas com relês;
3. **FBD** (Function Block Diagram): linguagem gráfica usando blocos de lógica digital;
4. **ST** (Structured Text): linguagem textual de alto nível, baseada em PASCAL, ADA e C.
5. **GRAFCET** (Grafo de Comando Etapa-Transição) ou **SFC** (Sequential Function Chart): linguagem gráfica baseada nas Redes de Petri, boa para lógicas complexas;



ST Structured Text

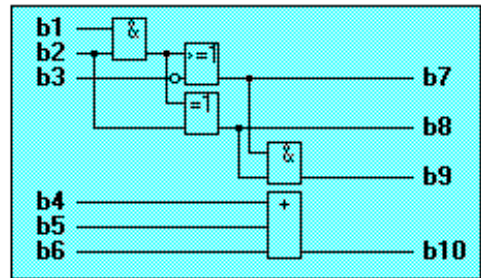
IL Instruction List

FBD Function Block Diagram

```

if [level <= level_max]
then
    out_valve := false;
    m_vlv := {vlv23 + dbh18} / 2;
else
    alarm_level := true;
end if;

```



start_cmd:	LD	bi101
	ADD	10
mul_ope:	MUL(i_bcmd
	SUB	bo100
)	
	ST	bcmd
	JMPC	mul_op

THE IEC 1131-3 LANGUAGES

Álgebra Booleana

- Sistemas digitais tem como bloco construtor básico as portas lógicas binárias.
- Álgebra booleana: lógica simbólica que mostra como portas lógicas operam.
- Portas lógicas operam com dois níveis de tensão, denominados nível "baixo" e nível "alto", cujos valores reais podem ser de 0V e +5V, -5V e +5V, -12V e +12V, 0V e +24V, etc.
- Uma variável booleana pode assumir somente dois valores correspondentes a tensão "baixa" e a "alta", representados por "0" e "1".
- Combinações básicas entre variáveis lógicas booleanas:
 - Conjunção (função lógica "E");
 - Disjunção (função lógica "OU");
 - Negação (função lógica "NÃO").

Álgebra Booleana

- **Conjunção (E):** $y = x1.x2$;

- **Tabela da verdade:**

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

- **Disjunção (OU):** $y = x1+x2$;

- **Tabela da verdade:**

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

- **Negação (NÃO):** $y = \overline{x}$

- **Tabela da verdade:**

x	y
0	1
1	0

Leis de Boole

- leis comutativas:

$$x_1.x_2 = x_2.x_1$$

$$x_1+x_2 = x_2+x_1$$

- leis associativas:

$$x_1.(x_2.x_3) = (x_1.x_2).x_3$$

$$x_1+(x_2+x_3) = (x_1+x_2)+x_3$$

- leis distributivas:

$$x_1.(x_2+x_3) = x_1.x_2+x_1.x_3$$

$$x_1+x_2.x_3 = (x_1+x_2).(x_1+x_3)$$

- leis da absorção:

$$x_1.(x_1+x_2) = x_1$$

$$x_1+x_1.x_2 = x_1$$

- tautologia:

$$x.x = x$$

$$x+x = x$$

- leis da negação:

$$x.\bar{x} = 0 \quad x+\bar{x} = 1 \quad \overline{(\bar{x})} = x$$

- Leis de "De Morgan":

$$\overline{x_1.x_2} = \bar{x}_1 + \bar{x}_2 \quad \overline{x_1 + x_2} = \bar{x}_1.\bar{x}_2$$

- Negação simples:

$$\bar{\bar{0}} = 1 \quad \bar{\bar{1}} = 0$$

- Operações com 0 e 1:

$$x.1 = x$$

$$x.0 = 0$$

$$x+0 = x$$

$$x+1 = 1$$

Álgebra Booleana

- A implementação de lógicas combinacionais é feita com auxílio das Tabelas da Verdade.
- A partir da Tabela da Verdade é obtida a expressão booleana que a representa:
 - Passo 1: procura-se na tabela as linhas cujas saídas tenham valor lógico 1;
 - Passo 2: para cada linha obtém-se a conjunção ("E") de todas as variáveis de entrada;
 - Passo 3: a expressão booleana de termos mínimos é dada pela disjunção ("OU") das expressões parciais obtidas no passo anterior.

Álgebra Booleana

- Exemplo:**

Linha	x1	x2	x3	y
1	0	0	0	0
2	0	0	1	0
3	0	1	0	1
4	0	1	1	0
5	1	0	0	1
6	1	0	1	0
7	1	1	0	1
8	1	1	1	0

Passo 1: linhas cujas saídas tem valor lógico 1 = 3, 5 e 7.

Passo 2: expressões parciais: $\overline{x_1} \cdot \overline{x_2} \cdot x_3$ $\overline{x_1} \cdot x_2 \cdot \overline{x_3}$ $x_1 \cdot \overline{x_2} \cdot \overline{x_3}$

Passo 3: expressão booleana de termos mínimos: $y = \overline{x_1} \cdot \overline{x_2} \cdot x_3 + \overline{x_1} \cdot x_2 \cdot \overline{x_3} + x_1 \cdot \overline{x_2} \cdot \overline{x_3}$

Usando as leis de Boole (ou Diagrama de Karnaugh) obtemos: $y = (\overline{x_1} + \overline{x_2}) \cdot x_3$

Álgebra Booleana

- Implementação das lógicas booleanas pode ser feita:
 - por Hardware:
 - » Com relês industriais
 - » Com portas lógicas digitais
 - por Software: técnica usada nos CLPs.
- Veremos agora as linguagens de programação de CLP conforme a norma IEC 61131-3.
- Obs: muitos CLPs (especialmente mais antigos) não seguem a norma e as instruções podem ser diferentes.

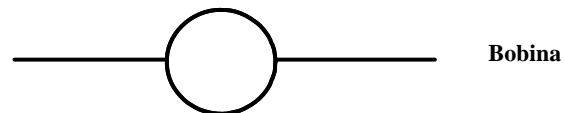
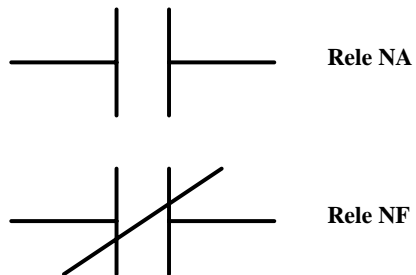
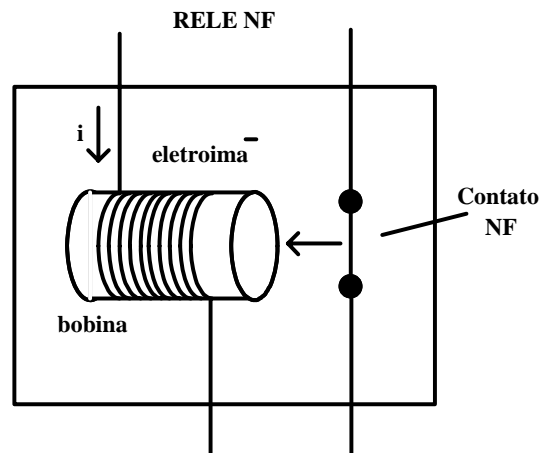
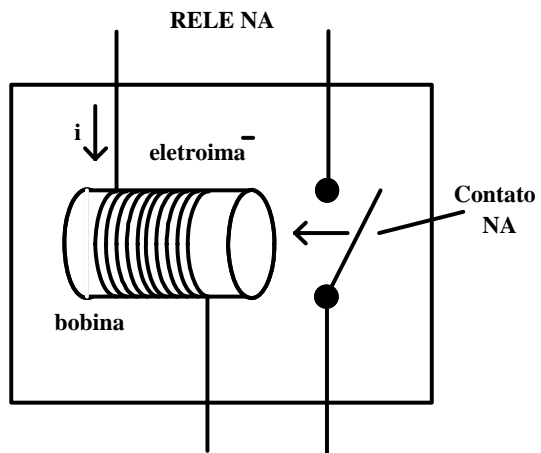
LD (Ladder Diagram)

- Criada para melhorar aceitação do CLP no mercado.
- Lógica de controle tradicional a relês era representada por "Diagrama de Escada" (ou “Diagrama de Relês”).
- Diagrama de Escada: associação de elementos de E/S, que representam função lógica de controle.
- LD: Linguagem gráfica que representa um circuito elétrico.

Elementos básicos de LD

- Três símbolos básicos:
 - Relês NA: contatos de saída abertos (OFF) enquanto não houver corrente na bobina;
 - Relês NF: contatos de saída fechados (ON) enquanto não houver corrente na bobina;
 - Bobinas: sinais de comando.

Elementos básicos de LD

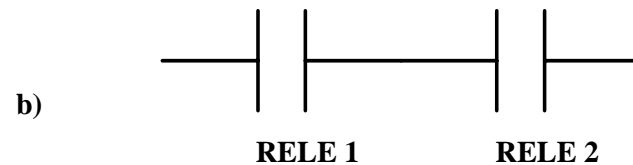
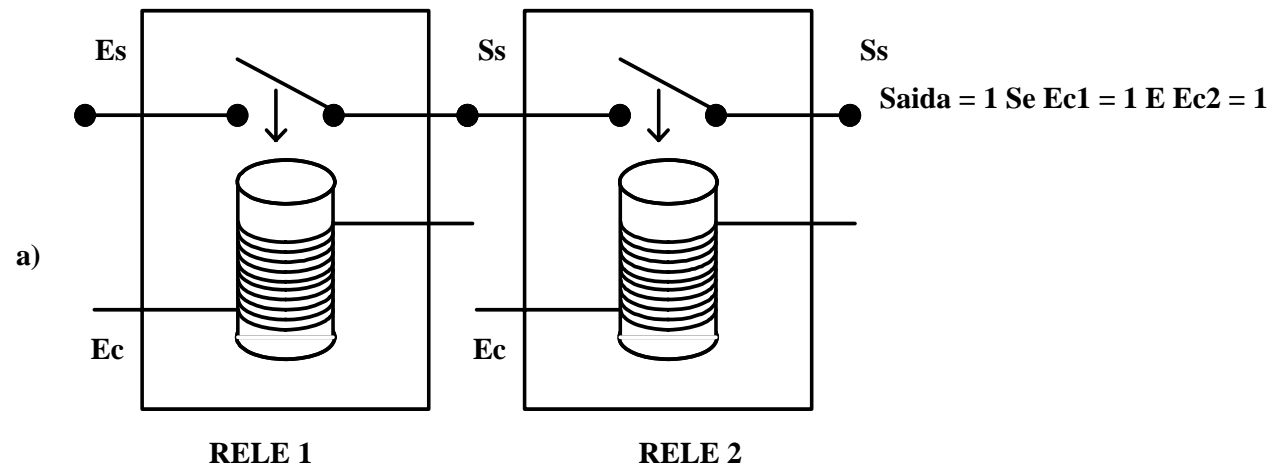


Elementos básicos de LD

- A cada elemento no Diagrama de Escada é associado um operando, identificado por letras e números.
- Relês NA e NF: usados para ler (sensorear) estado ou valor de um operado (entrada, saída ou interno).
- Bobinas: usadas para escrever (atuar) estado ou valor em um operando (entrada, saída ou interno).
- Além destes elementos básicos, existem representações próprias para temporizadores, contadores e outros elementos especiais de entrada e saída.
- Elementos básicos permitem reconstruir lógica de Boole (precisamos de E, OU e NÃO).

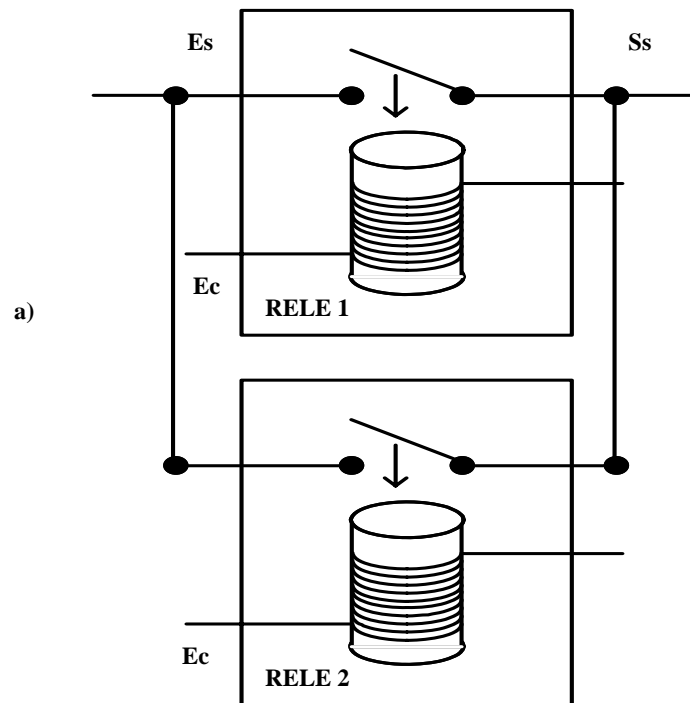
Função Lógica "E"

- Associação em série de relês:

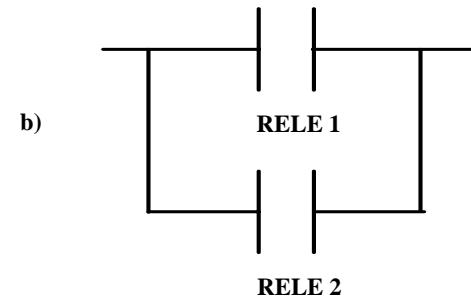


Função lógica "OU"

- Associação em paralelo de relês:

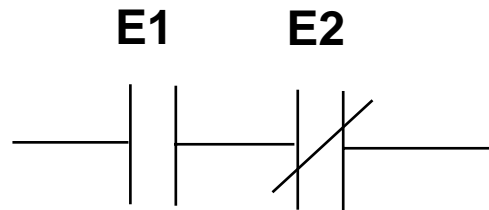


Saida = 1 Se $Ec1 = 1$ OU $Ec2 = 1$

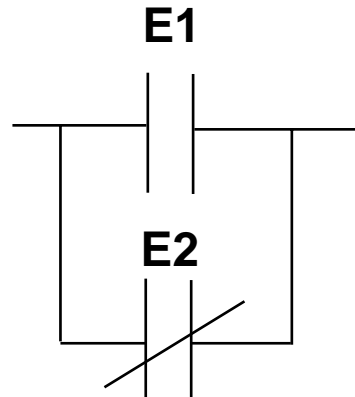


Função Lógica “NÃO”

- Relé NF: negação do sinal de entrada correspondente.



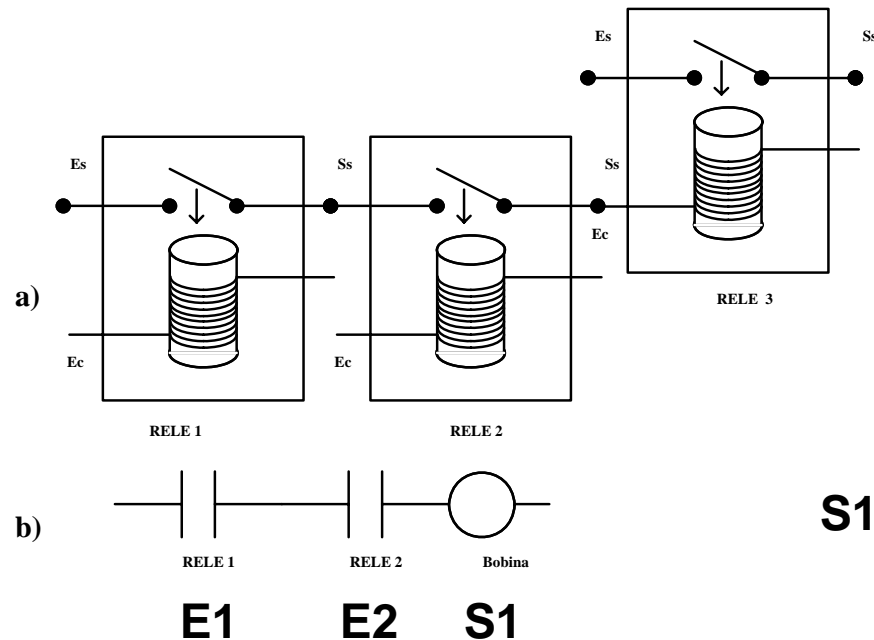
$E1 \cdot \bar{E2}$



$E1 + \bar{E2}$

Bobinas

- Bobina ligada em série com relês: sinal de saída será ativado se relês permitirem passagem de corrente até a bobina. Implica em escrever algo no operando.

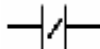


Estrutura de um programa em LD

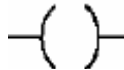
- Linguagem composta de símbolos gráficos.
- Os símbolos são interligados entre si para representar lógicas similares a um esquema elétrico.



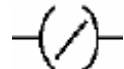
Entrada, Rele NA



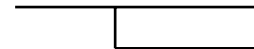
Entrada, Rele NF



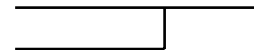
Saida, Bobina



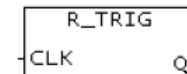
Saida complementada, Bobina



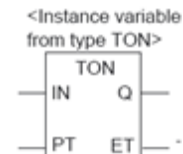
Abertura de ramo paralelo



Fechamento do ultimo ramo paralelo aberto



Rele de Pulso

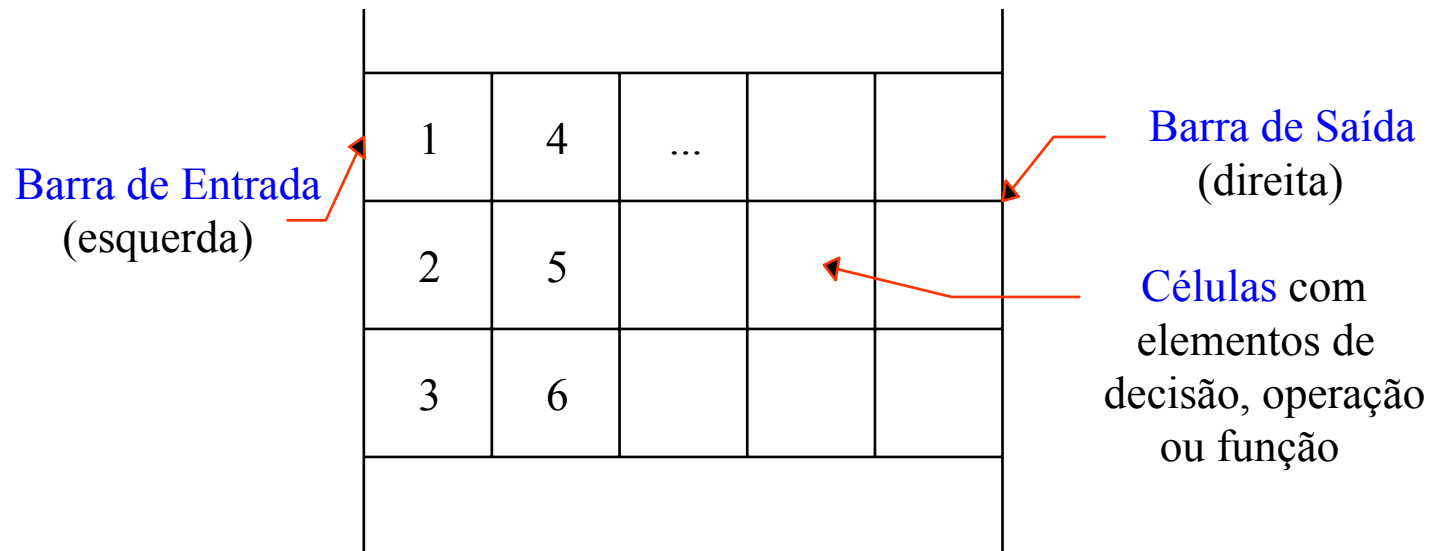


Temporizador

Estrutura de um programa em LD

- As lógicas são compostas de células:
 - **Barras de alimentação**: barras verticais simulando as linhas de alimentação do circuito;
 - **Ligações horizontais**: representam ligações elétricas entre células;
 - **Ligações verticais**: permitem interligar células em paralelo, definindo uma função "OU";
 - **Instruções e operandos**:
 - » **Elementos de decisão**: contatos NA, NF, etc;
 - » **Elementos de operação**: bobinas dos relês;
 - » **Blocos de função**: contadores, operações aritméticas, seqüenciadores, bobinas auxiliares, reles de pulso, etc.

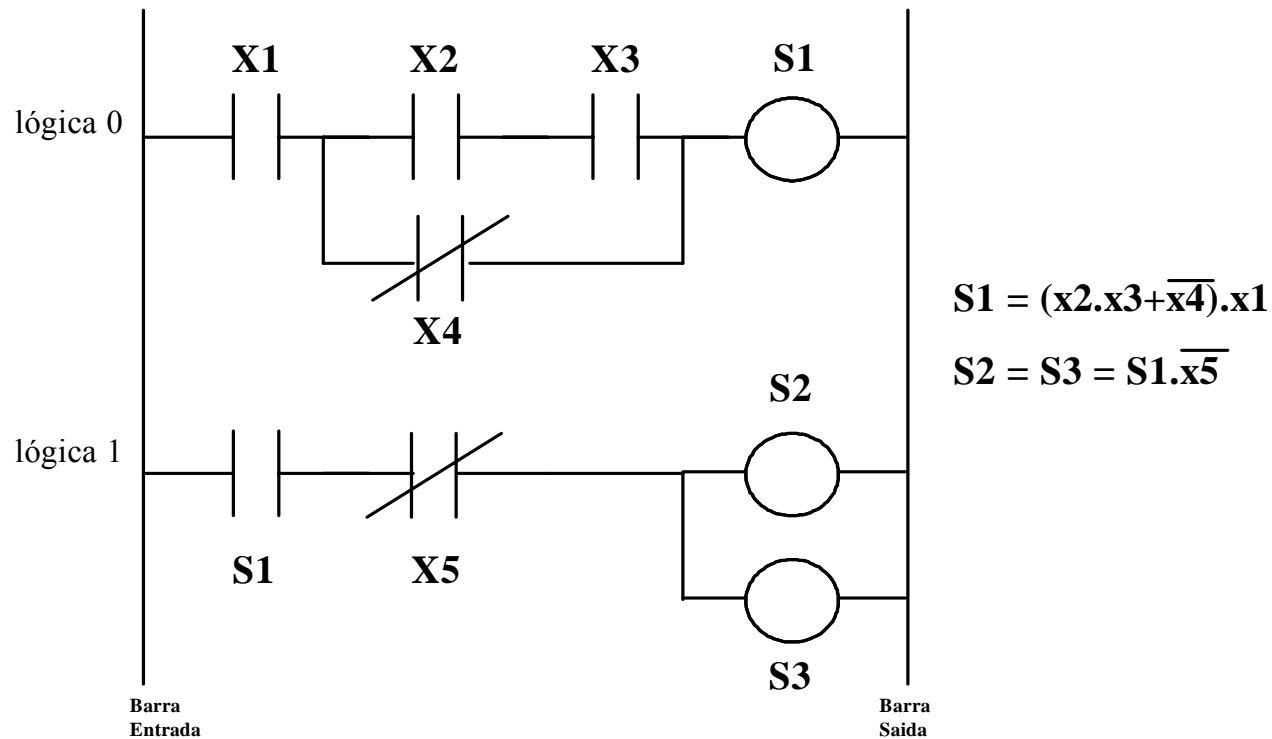
Estrutura de um programa em LD



- Cada lógica é processada coluna por coluna, na seqüência mostrada pela numeração das células.
- Programas compostos de várias lógicas são executados de cima para baixo, uma lógica após a outra, e repetidos ciclicamente após a execução da última lógica.

Exemplo de LD

- lógica de comando apresentada anteriormente para o circuito lógico digital em Diagrama de Escada (2 lógicas):



Edição do programa em LD

- Para a edição do programa o TP pode:
 - possuir um teclado dedicado com os símbolos indicados;
 - permitir a programação por meio de recursos gráficos de edição.



Operandos em LD

- **Operandos**: identificam variáveis e constantes utilizadas no programa aplicativo. Ex.: pontos de entrada e saída, memórias contadoras, etc.
- **Operandos retentivos**: valor conservado mesmo após o desligamento do CLP ou queda de energia.
- Operandos associados à entradas ou saídas digitais binárias são de apenas 1 bit.
- Operandos associados à entradas ou saídas analógicas podem ser de vários bits, em função da resolução dos conversores A/D e D/A (ex.: 8, 12, 16, 32 bits, etc.).
- Operandos internos (contadores, memórias auxiliares, resultados de operações matemáticas, etc.) podem ser de 1 bit, 8, 16 ou 32 bits ou mesmo real de ponto flutuante.

Operandos em LD

- Existem 3 tipos básicos de operandos:
 - **Operandos simples:** contém o valor atual de uma entrada (relé), saída (bobina) ou posição de memória (relé auxiliar) do CLP. São identificados por códigos alfanuméricos. Ex.: E02.3 (entrada 3 do bloco 02) ou S01.4 (saída 4 do bloco 01) ou A08 (auxiliar);
 - **Operandos constantes:** utilizados para definir valores fixos durante todo o programa. São identificados no programa por meio de códigos alfanuméricos. Ex.: #50;
 - **Operandos tipo tabela ou vetor:** são arranjos (arrays) de operandos simples. São identificados de forma semelhante aos casos anteriores. Ex.: TM0026.

Instruções em LD

-**Instruções**: são utilizadas para executar determinadas tarefas por meio de leitura e/ou alteração do valor dos operandos.

-Grupos principais:

- **relés**: leitura de valores de contatos e ajuste de valores de bobinas;
- **movimentadores**: entrada para memória, memória para saída, etc;
- **aritméticas**: somar, subtrair, multiplicar e dividir operandos de vários bits (bytes, words, etc.);
- **binárias**: E, OU e OU exclusivo entre operandos de vários bits;
- **contadores**: contagem incremental / decremental e temporização (delay);
- **conversão**: operações de conversão A/D, D/A, Binário / Decimal e Decimal / Binário;
- **teste**: operações de comparação (igual, maior, menor, etc);
- **comunicação**: envio e recepção de mensagens via rede;

Instruções em LD

- Instruções tipo Relé:

<Operand> --- ---	LD	<Operand> ---()	ST
		---(/)---	STN
<Operand> --- / ---	LDN	---(S)---	S (ou L = Latch)
		---(R)---	R (ou U = Unlatch)
--- NOT ---	NOT	<Operand> ---(#)---	Conector (armazena resultado parcial no operando)

Instruções em LD

- Instruções tipo Relé:

<Operand>

---(N)---

STF

Bobina Detectora de Transição Negativa: se houver flanco negativo na entrada, operando vai para 1. A saída contém o estado final da entrada.

<Operand>

---(P)---

STR

Bobina Detectora de Transição Positiva: se houver flanco positivo na entrada, operando vai para 1. A saída contém o estado final da entrada.

Instruções em LD

- Relés Detectores de Transições nos Operandos:

 **LDF**

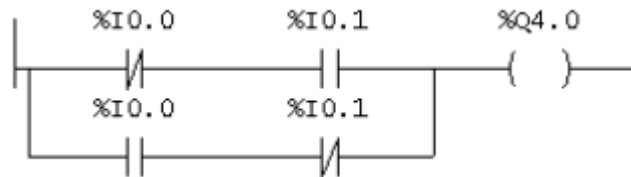
Compara o estado atual do **operando** com o estado da varredura anterior. Se houve uma transição negativa (falling edge), a saída é **1**.

 **LDR**

Compara o estado atual do **operando** com o estado da varredura anterior. Se houve uma transição positiva (rising edge), a saída é **1**.

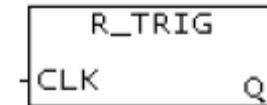
Instruções em LD

- Instruções tipo Relé:
- Não há instrução explícita para XOR, que pode ser obtido fazendo:

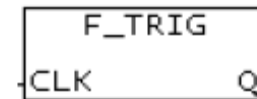


Instruções em LD

- **Relés de pulso:**
- R_TRIG: se entrada vai de 0 para 1, saída vai para 1 e fica em 1 até o fim do atual ciclo de varredura.

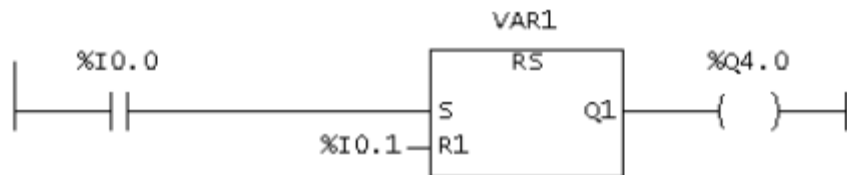


- F_TRIG: inverso (Falling Edge)

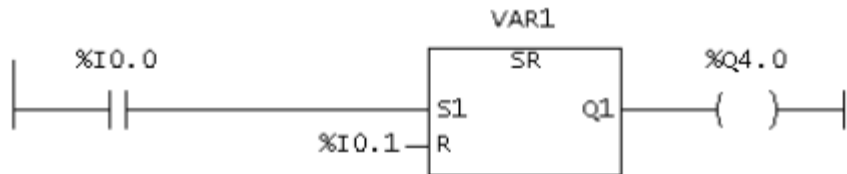


Instruções em LD

- Flip-Flops:



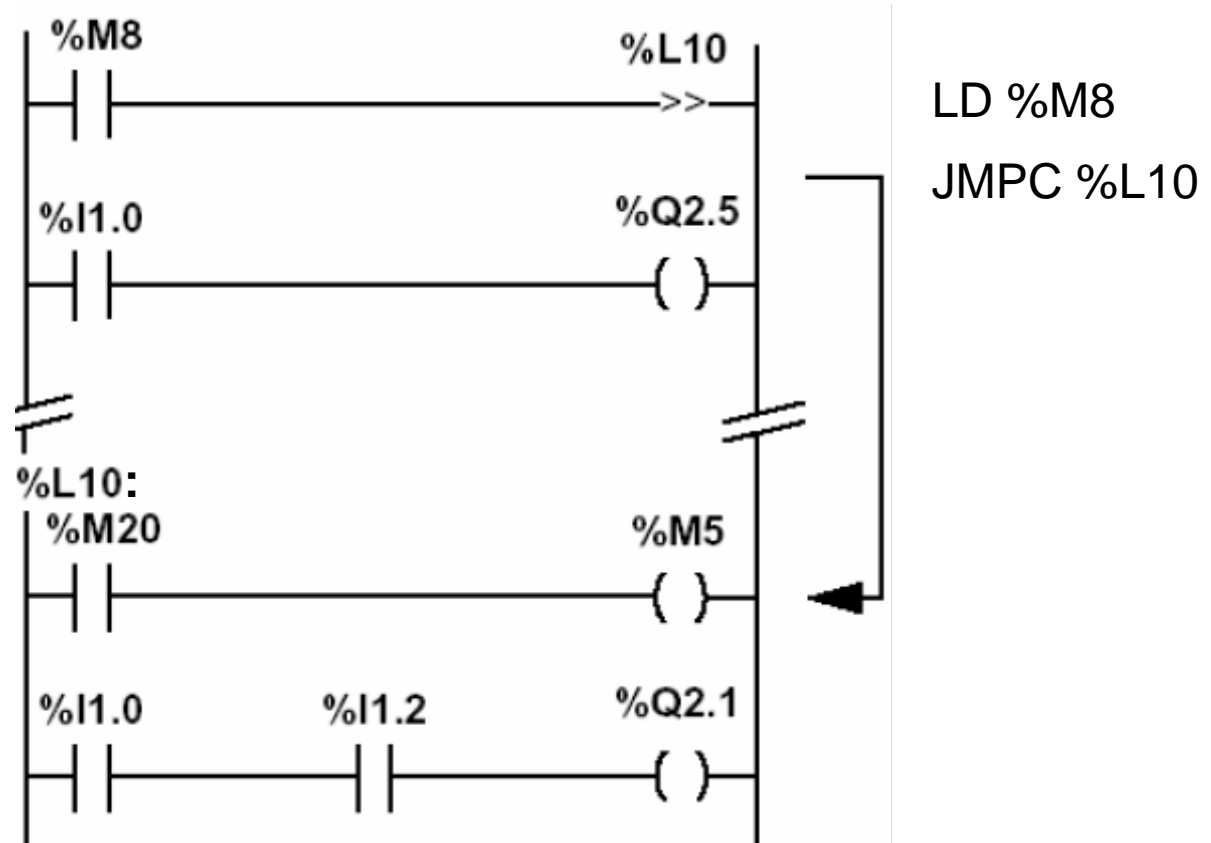
RS: se ambas as entradas estão em 1, prioriza Reset



SR: se ambas as entradas estão em 1, prioriza Set

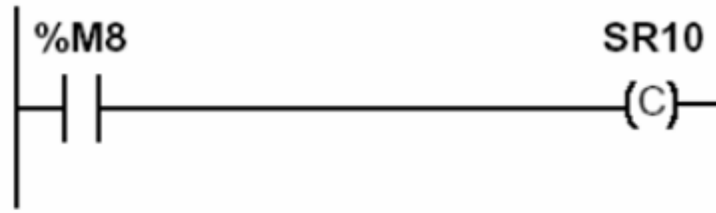
Instruções em LD

- Jumps (Bobinas de salto):



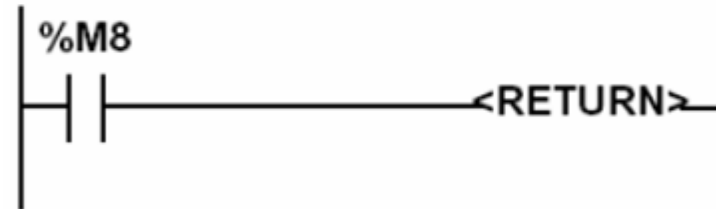
Instruções em LD

- Chamadas a sub-rotinas:



LD %M8
CAL SR10

...

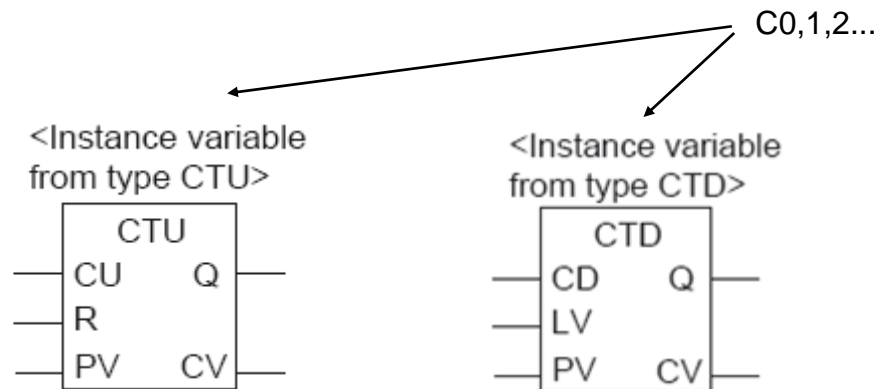


LD %M8
RET

Instruções em LD

- Instruções tipo Contador:

Contadores Simples (unidirecionais):



Incremental (Up)

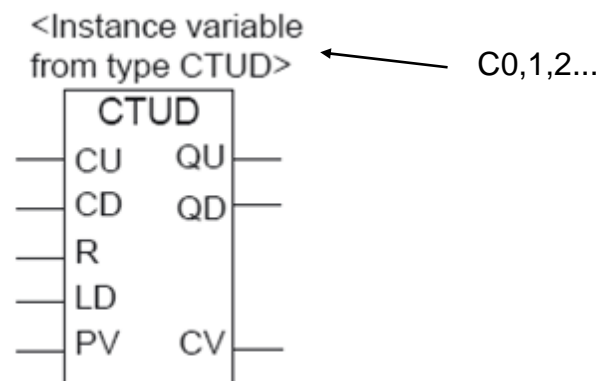
CU = Count Up
R = Reset
CV = Counter Value
Q = Status do contador ($CV \geq PV$)
PV = Limite de contagem

Decremental (Down)

CD = Count Down
LV = Reseta para valor inicial
CV = Counter Value
Q = Status do contador ($CV \leq 0$)
PV = valor inicial de contagem

Instruções em LD

Contador Bidirecional (incremental / decremental):



Up-Down

CU = Count Up

CD = Count Down

R = Reseta CV para 0

LD = Reseta CV para PV

CV = Counter Value

QU = Status do contador Up ($CV \geq PV$)

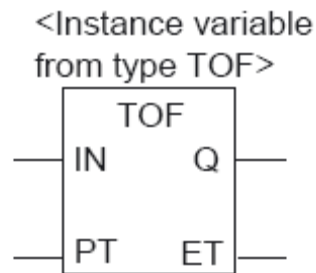
QD = Status do contador down ($CV \leq 0$)

PV = Limite de contagem

Instruções em LD

Temporizadores:

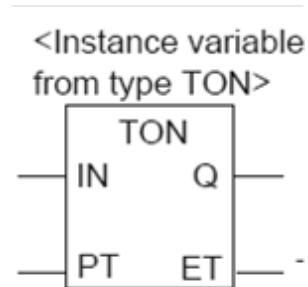
•TOF (Timer Off-Delay):



- Quando IN passa de 0 para 1, Q vai para 1.
- ET conta até atingir limite dado por PT.
- Q fica em 1 até o limite ser atingido e então passa para 0.
- Assim, a saída Q é desativada (OFF) com um delay.

Instruções em LD

- **TON (Timer On-Delay)**

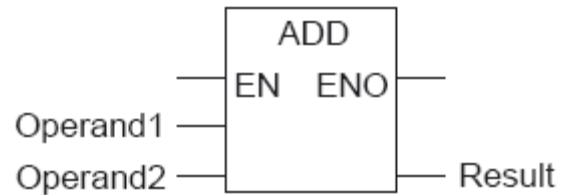


- Quando IN passa de 0 para 1, ET conta até atingir limite dado por PT.
- Q fica em 0 até o limite ser atingido e então passa para 1 se IN ainda estiver em 1.
- Assim, a saída Q é ativada (ON) com um delay.

Instruções em LD

-instruções aritméticas:

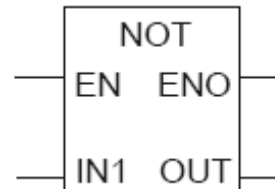
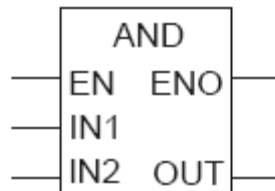
- ADD, SUB, DIV, MUL, MOD



Instruções em LD

- instruções binárias:

AND, OR, XOR, NOT



Instruções em LD

- Comunicação:

```

      +-----+
      |      SEND      |
      +-----+
      | REQ      NDR |---| BOOL
      | R        ERROR |---| BOOL
      | ID      STATUS |---| INT
      | R_ID    RD_1  |---| ANY
      | SD_1    :      |---| :
      | :      RD_m  |---| ANY
      | SD_n    |---|
      +-----+
  
```

```

      +-----+
      |      RCV       |
      +-----+
      | EN_R      NDR |---| BOOL
      | RESP     ERROR |---| BOOL
      | ID      STATUS |---| INT
      | R_ID    RD_1  |---| ANY
      | SD_1    :      |---| :
      | :      RD_n  |---| ANY
      | SD_m    |---|
      +-----+
  
```

Metodologia de programação

- Usar Diagrama de Karnaugh quando a aplicação não requer memória (armazenamento do estado do sistema).
- Usar Diagramas de Estado (máquinas de estado finito, FSM) quando a aplicação requer memória (maioria dos casos práticos).

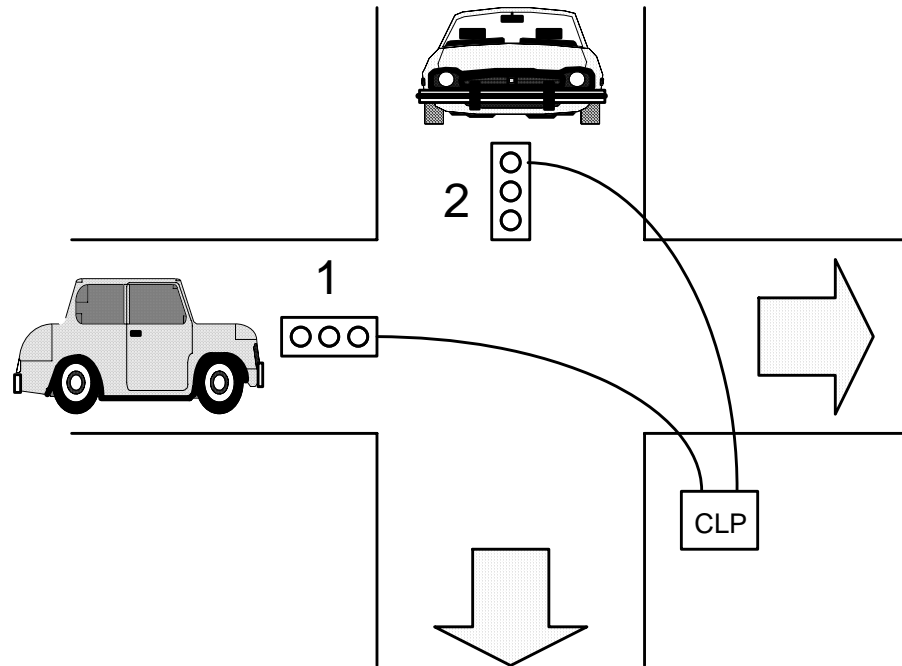
Metodologia de programação

- Máquina de estados pode ser vista como um modelo de comportamento de aplicativos.
- Máquina de estados composta de:
 - **Estados:** cada estado comporta-se como uma memória, armazenando informações sobre as saídas em um dado momento.
 - **Transições:** condição de mudança de um estado para outro.
 - **Saídas:** descreve atividade (ação) a ser realizada em um dado estado.

Metodologia de programação

- Passos:
 - Mapeamento de E/S do processo
 - Montagem da máquina de estados
 - Converter transições para LD
 - Converter ações para LD

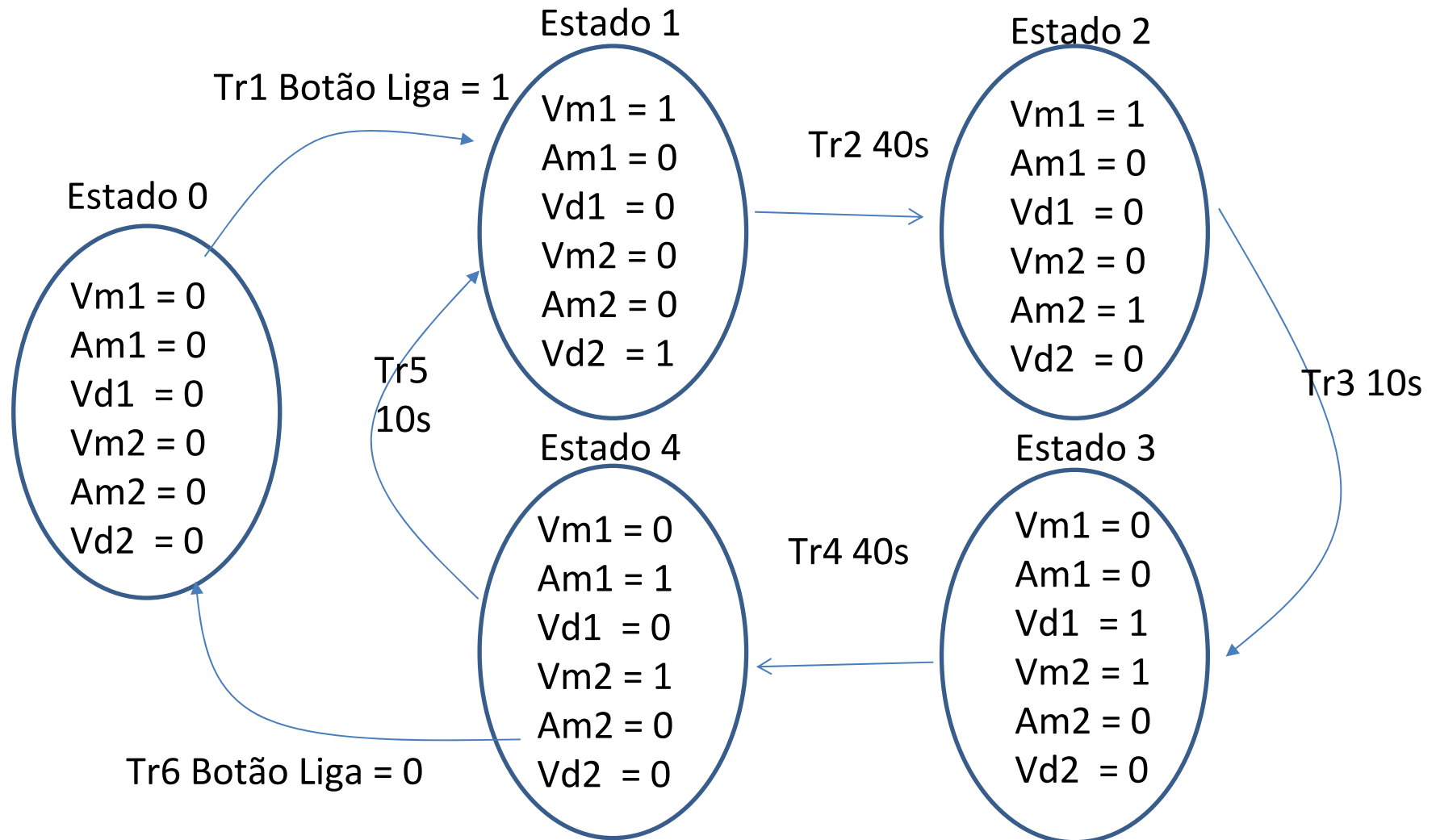
Exemplo 1: Comando de uma sinaleira



Mapeamento de E/S

Endereço de E/S no CLP	Função na sinaleira	Conexão Externa
Entrada 0	Habilita Sinaleiras	Chave ON/OFF
Saída 1	Sinaleira 1 Vermelha	Lâmpada
Saída 2	Sinaleira 1 Amarela	Lâmpada
Saída 3	Sinaleira 1 Verde	Lâmpada
Saída 4	Sinaleira 2 Vermelha	Lâmpada
Saída 5	Sinaleira 2 Amarela	Lâmpada
Saída 6	Sinaleira 2 Verde	Lâmpada

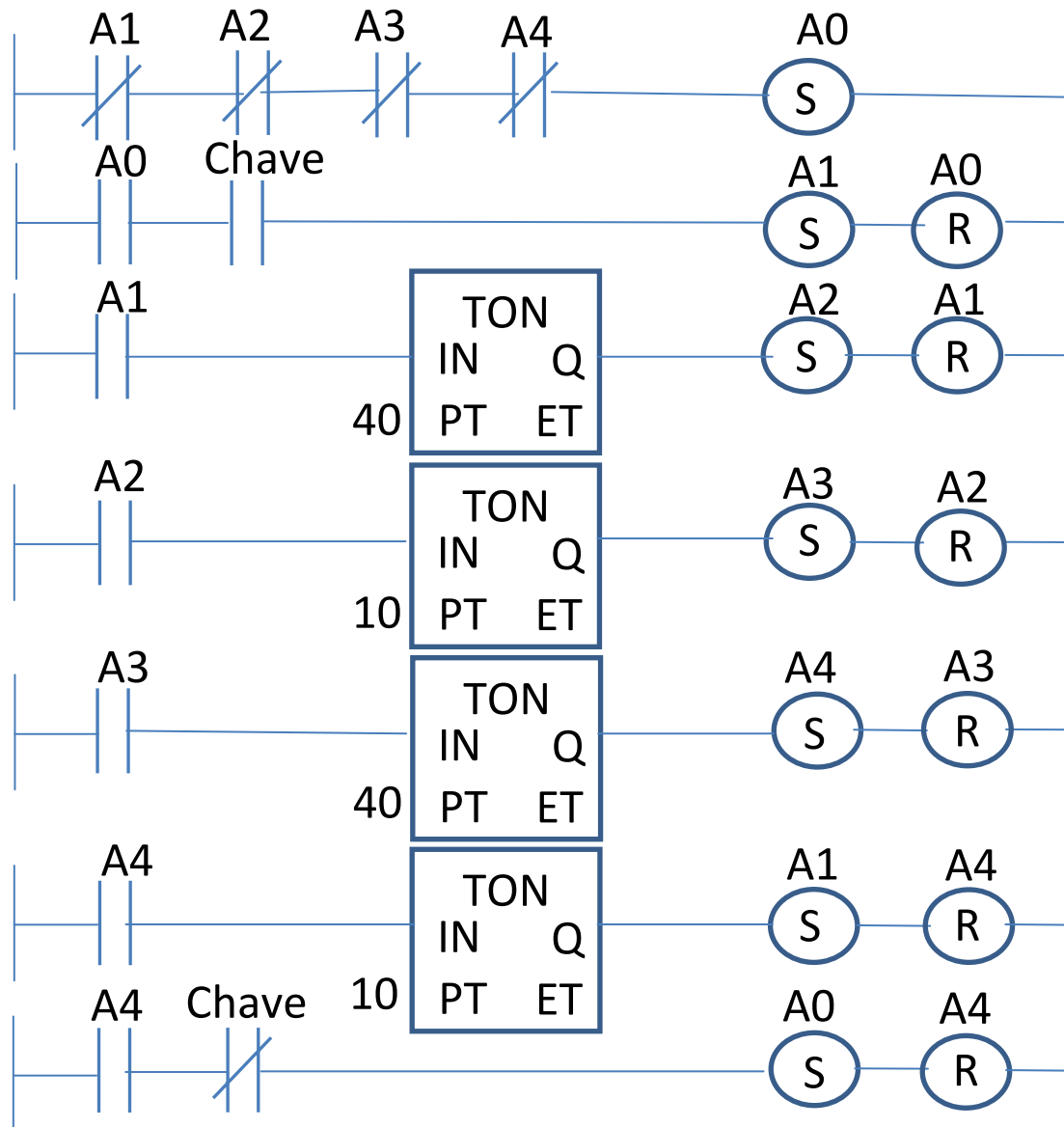
Máquina de estado



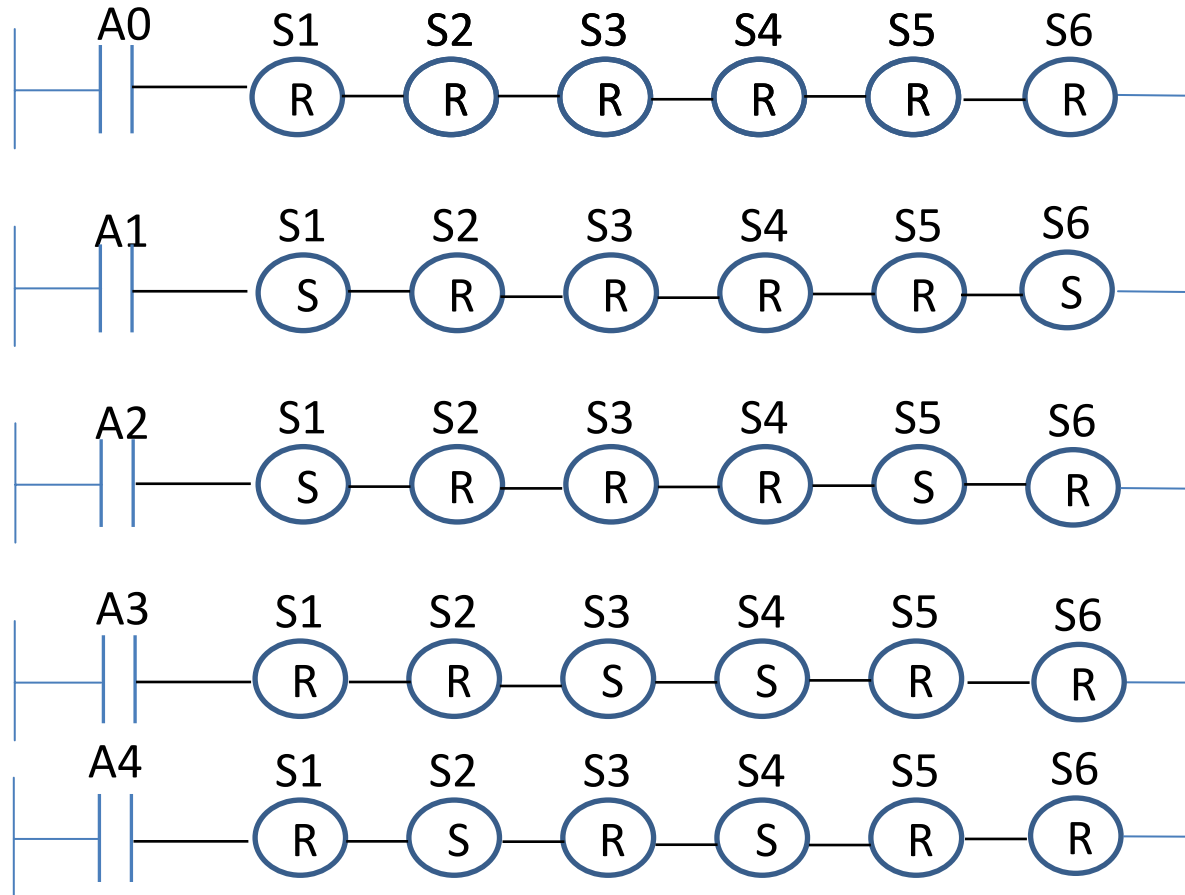
Conversão para LD

- Para cada estado, criar uma variável auxiliar (A_n), que vale 1 se o sistema está naquele estado, ou zero se não está.
- Para o exemplo, teremos 5 estados e portanto 5 variáveis auxiliares A_0 , A_1 , A_2 , A_3 e A_4 .
- Montar uma lógica para cada transição de estado.
- Montar uma lógica para cada ação (ativação de saída).

Mapeamento das transições



Mapeamento das ações



Exemplo 2: Comando de um Estacionamento

- Um estacionamento é composto de 2 andares, com 10 vagas para carros em cada andar.
- A quantidade de carros em cada andar é controlada através das fotocélulas FC1, FC2, FC3 e FC4.
- Sinaleira de entrada: lâmpada verde L1 e lâmpada vermelha L2, que só permite a entrada de um carro caso haja ao menos uma vaga no estacionamento.
- Sinaleira térreo: lâmpada verde L3, lâmpada vermelha L4.
- Sinaleira 1º andar: lâmpada verde L5 e lâmpada vermelha L6.

Exemplo 2: Comando de um Estacionamento

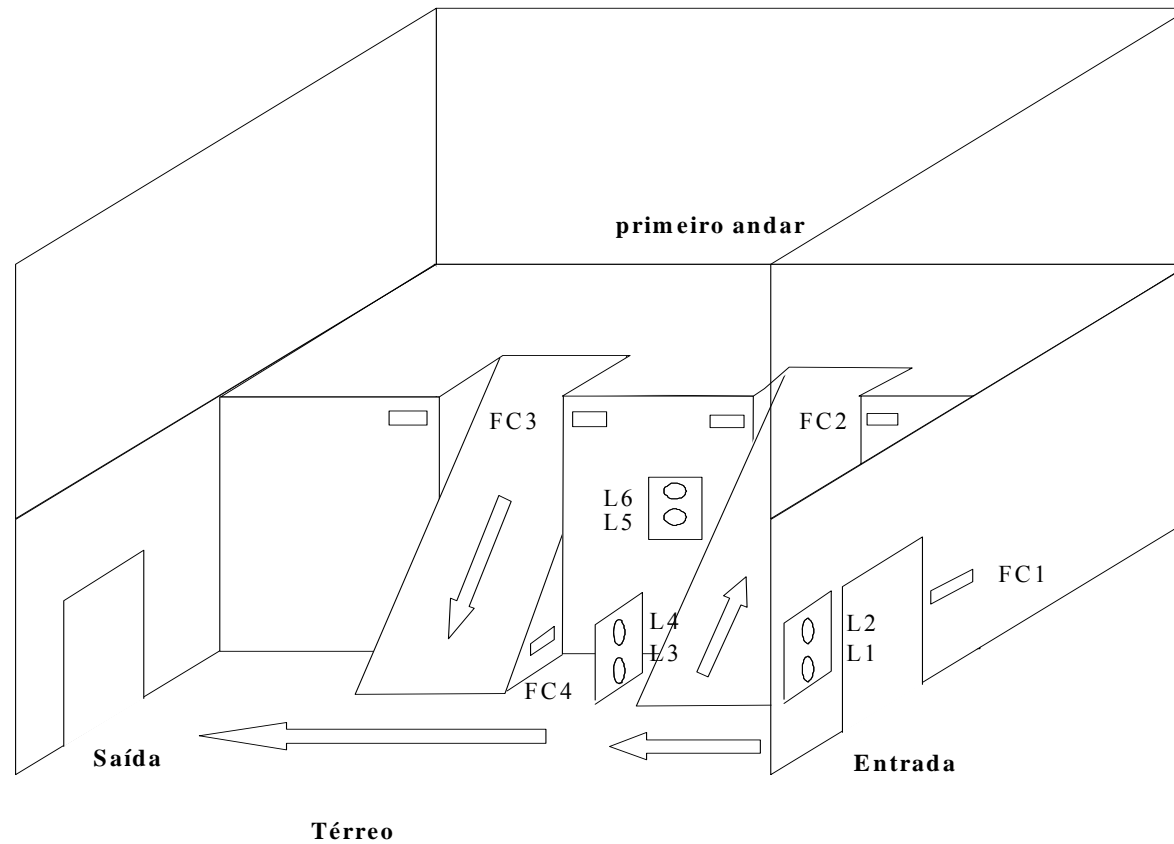
A operação transcorre da seguinte forma:

- Entrada de um veículo no estacionamento: a fotocélula FC1 é ativada e a quantidade de veículos no térreo e a quantidade total de carros no estacionamento são incrementadas em 1.
- Subida de um veículo ao primeiro andar: a fotocélula FC2 é ativada e o número de veículos no térreo é decrementado em 1 enquanto o número de veículos no primeiro andar é incrementado em 1.
- Lotação: Se o numero de carros no térreo chegar a 10, L4 acende. Se o número de carros no primeiro andar chega a 10, L6 acende. Se ambos os andares estão lotados, a lâmpada vermelha na entrada L2 acende.

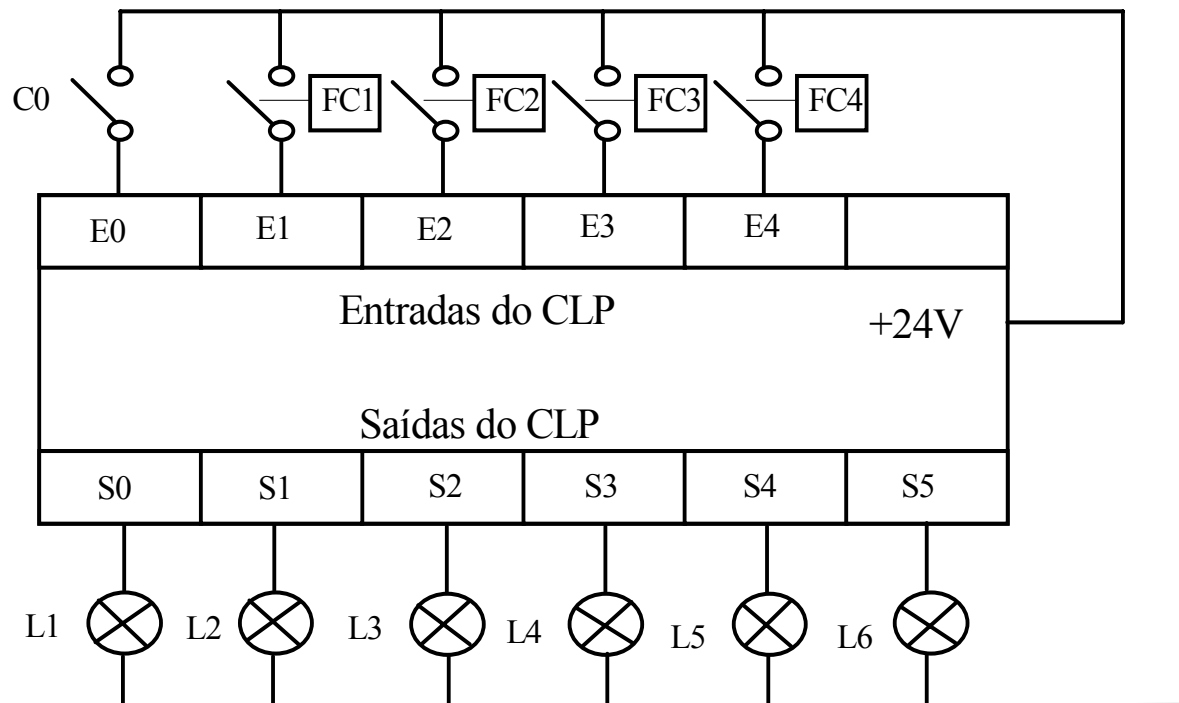
Exemplo 2: Comando de um Estacionamento

- Saída de um veículo do primeiro andar: a fotocélula FC3 é ativada e o número de carros no primeiro andar bem como o número total de carros no estacionamento são decrementados em 1.
- Saída de um veículo do térreo: a fotocélula FC4 é ativada e o número de carros do térreo bem como o número total de carros no estacionamento são decrementados em 1.
- O contador de carros pode ser resetado por meio do botão C0.

Exemplo 2: Comando de um Estacionamento



Exemplo 2: Comando de um Estacionamento



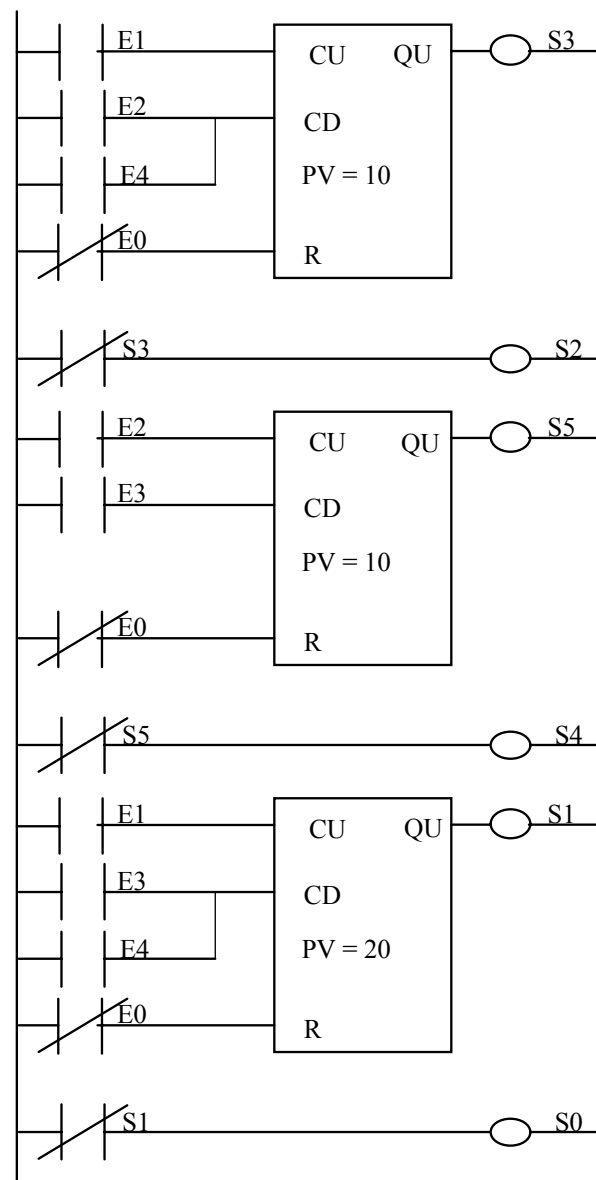
Exemplo 2: Comando de um Estacionamento

Entrada / Saída	Ligação externa	Comentários
E0	C0	Chave Resetar contador
E1	FC1	Fotocélula Entrada
E2	FC2	Fotocélula Subida
E3	FC3	Fotocélula Saída, 1. Pav.
E4	FC4	Fotocélula Saída, Térreo
S0	L1	Sinaleira Entrada, Verde
S1	L2	Sinaleira Entrada, Verm.
S2	L3	Sinaleira Térreo, Verde
S3	L4	Sinaleira Térreo, Verm.
S4	L5	Sinaleira 1. Pav., Verde
S5	L6	Sinaleira 1. Pav., Verm.

- poderíamos ainda incluir comando de portões de E/S...

E0 = C0
E1 = FC1
E2 = FC2
E3 = FC3
E4 = FC4

S0 = L1
S1 = L2
S2 = L3
S3 = L4
S4 = L5
S5 = L6



LD - Conclusão

- Vantagens:
 - possibilidade de uma rápida adaptação do pessoal técnico (semelhança com lógicas de relês);
 - possibilidade de aproveitamento do raciocínio lógico feito na elaboração de um comando feito com relês;
 - fácil recomposição do diagrama original a partir do programa de aplicação;
 - fácil visualização dos estados das variáveis sobre o diagrama de escada, permitindo depuração e manutenção do software rápidas;
 - documentação fácil e clara;
 - símbolos padronizados de maneira idêntica nas normas NEMA ICS 3-304 e IEC 61131-3, mundialmente aceitas pelos fabricantes e usuários.
 - Técnica de programação mais difundida na indústria.

LD - Conclusão

- Desvantagens:
 - Já existem linguagens de mais alto nível (ainda não muito difundidas).
 - Programadores não familiarizados com a operação de relês tendem a ter dificuldades com esta linguagem.
 - Edição mais lenta.
 - Lógicas complexas requerem uso de recursos adicionais, como a máquina de estados.

GRAFCET / SFC

- Para a programação de lógicas de comando mais complexas, uma linguagem de mais alto nível é conveniente.
- Em 1977, foi definida a linguagem GRAFCET pela AFCET (*Association Française pour la Cybernétique Economique e Technique*, Paris).
- GRAFCET = Grafo Funcional de Comando Etapa-Transição (*Graphe Fonctionnel de Commande Etape-Transition*).
- SFC = *Sequential Function Chart* (Linguagem de CLP baseada no GRAFCET).

GRAFCET / SFC

- Em 1988 foi publicado o padrão internacional IEC 848: *Preparation of function charts for control system*, baseado na linguagem Grafcet.
- A norma IEC 61131-3 introduziu pequenas modificações no padrão IEC 848 visando acoplar esta linguagem às demais previstas na nova norma.
- Adaptação da técnica de Redes de Petri.
- permite uma visualização dos estados pelos quais o sistema a comandar deve passar para realizar uma dada operação.
- Já incorpora a idéia de estados das máquinas de estado finito!

Elementos Básicos

- Etapa (SFC = STEP):
 - Representa um **estado parcial** do sistema.
 - Cada etapa tem um nome (ou número) único na aplicação.
 - Uma etapa pode estar ativada ou desativada em um dado momento.
 - Representa-se a ativação de uma etapa colocando uma ficha (token) em seu interior (flag ETAPAnome.X=1).
 - O estado global de um sistema em um dado momento é dado pelo conjunto de etapas ativadas ("marcagem" do grafo).



Etapa inativa



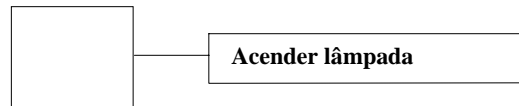
Etapa ativa



Etapa inicial

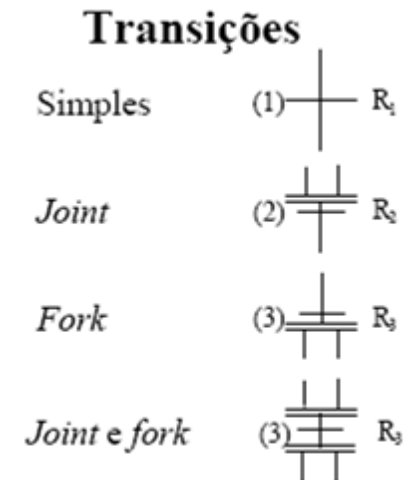
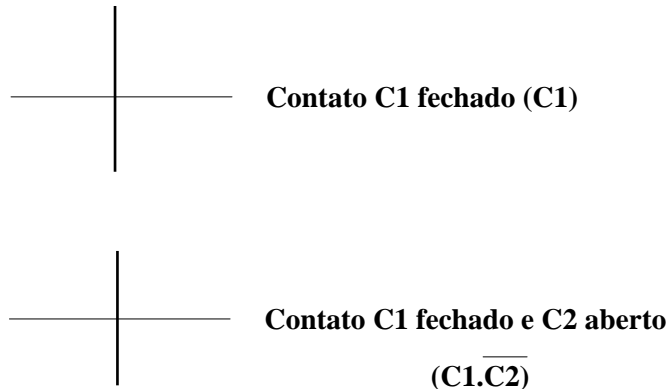
Elementos Básicos

- Ação:
 - Uma ação é sempre associada à uma etapa e só pode ser executada quando a etapa estiver ativada.
 - A execução de uma ação pode estar condicionada a uma função lógica de variáveis de entrada do CLP ou ao estado ativo ou inativo de outras etapas ($ETAPAn.X=1$).
 - Uma ação está usualmente relacionada à ativação ou desativação de saídas do CLP.
 - Uma etapa pode não ter nenhuma ação associada.



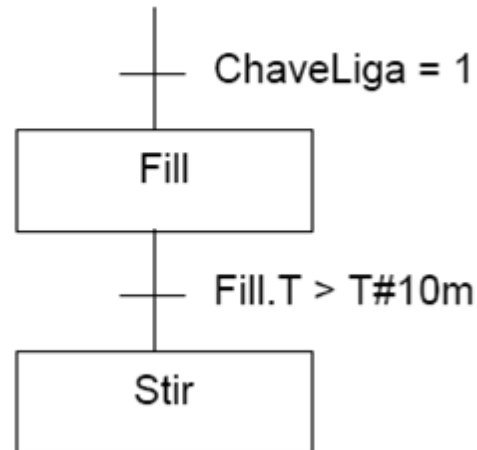
Elementos Básicos

- Transições e Receptividades:
 - Transições indicam a possibilidade de evolução entre etapas (mudança de estado).
 - A cada transição é associada uma condição lógica chamada receptividade, descrita ao lado do símbolo da transição.
 - A receptividade em geral é uma função associada ao estado das entradas do CLP.



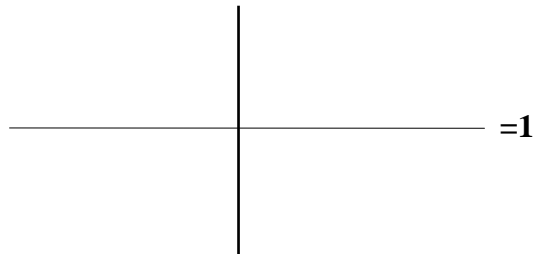
Elementos Básicos

- Uma receptividade especial é a que manipula a variável tempo.
- No exemplo, o grafo ficará no estado Fill por 10 minutos:



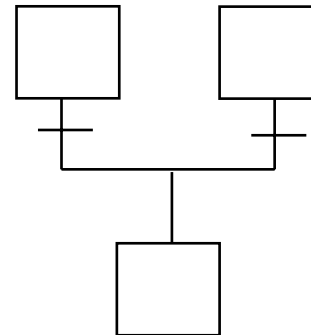
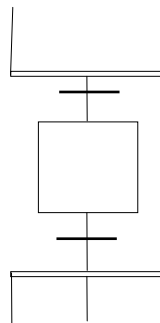
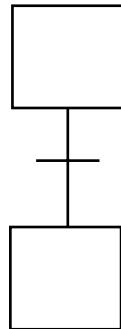
Elementos Básicos

- Caso não haja condição associada a receptividade, esta é chamada de "receptividade incondicional".



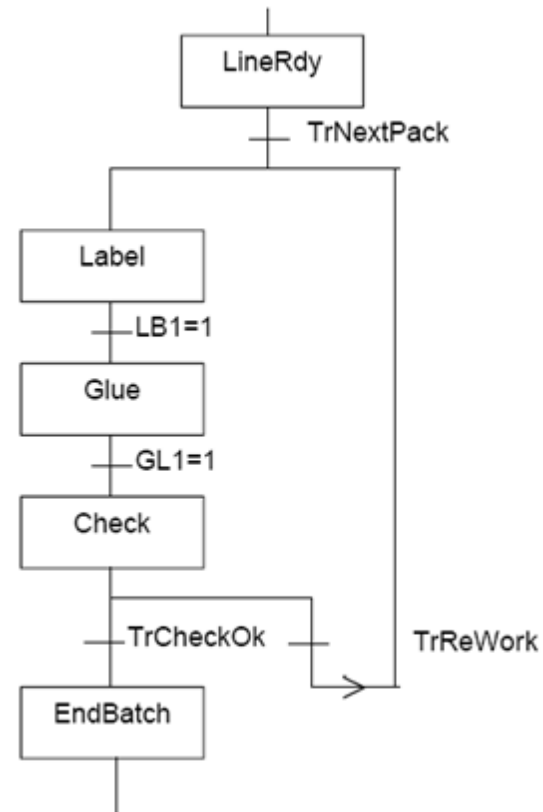
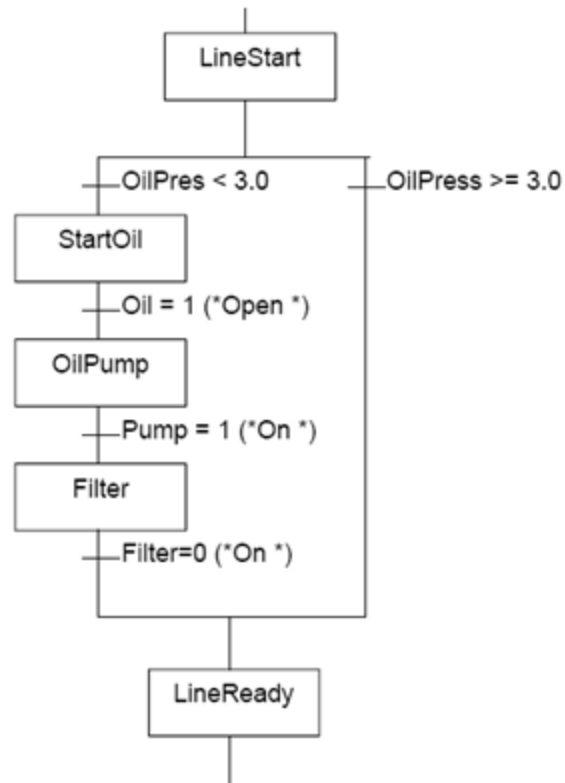
Elementos Básicos

- Arcos:
 - ligam sempre uma transição à uma etapa ou uma etapa à uma transição.
 - Uma etapa ou transição pode ter vários arcos.



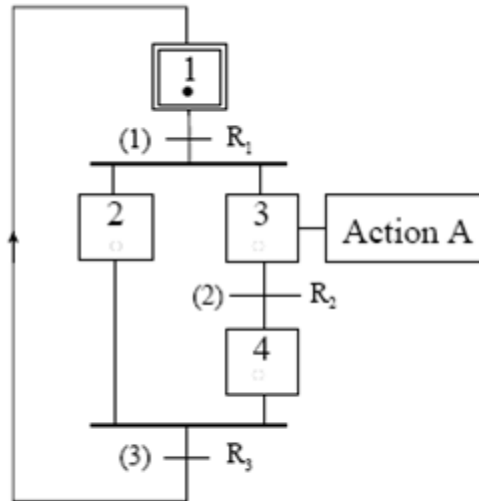
Elementos Básicos

- Grafos evoluem normalmente de cima para baixo
- Usa-se arcos direcionados para indicar outra direção de evolução do grafo



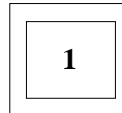
Elementos Básicos

- Exemplo simples de GRAFCET:



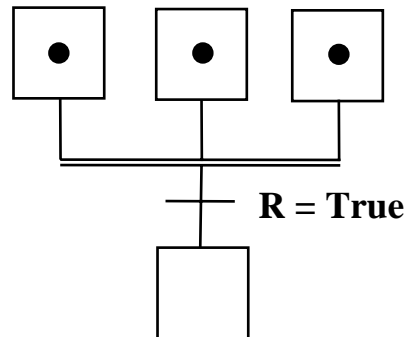
Regras de Evolução

- Cinco regras definem a evolução do estado (ou marcagem) do GRAFCET:
- Regra 1: Estado Inicial
 - As etapas ativadas na condição inicial devem ser assinaladas com um duplo quadrado.
 - As etapas são numeradas a partir daquela que representa a condição inicial (que recebe o número 1).



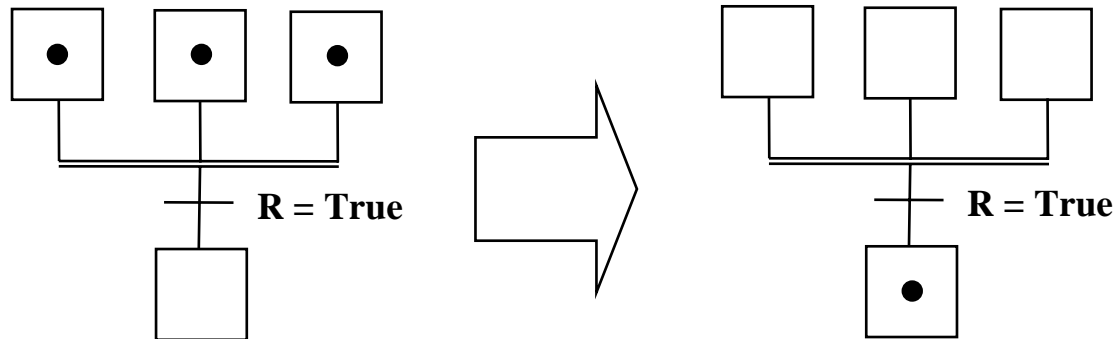
Regras de Evolução

- Regra 2: Disparo de uma transição
 - Uma transição será válida se todas as etapas de entrada desta estiverem ativadas.
 - Se a transição é válida e a receptividade a ela associada é verdadeira, a transição é dita disparável.
 - O "disparo", nestas circunstâncias, é obrigatório.



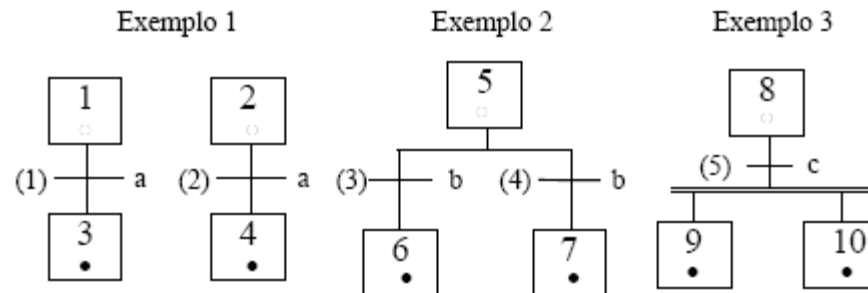
Regras de Evolução

- Regra 3: Evolução das etapas ativas
 - "Disparar" uma transição consiste em ativar todas as etapas de saída da transição e desativar todas as etapas de entrada.



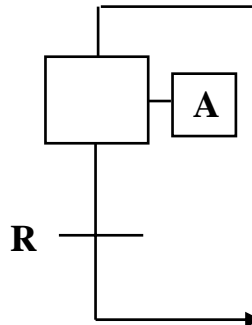
Regras de Evolução

- Regra 4: Simultaneidade do disparo
 - quando houver mais de uma transição simultaneamente disparável no grafo (paralelismo), todas deverão ser disparadas no mesmo instante.
 - Ações associadas a etapas assim ativadas são executadas em paralelo.



Regras de Evolução

- Regra 5: Ativação e desativação simultânea de uma etapa
 - Se uma etapa deve ser desativada e ativada simultaneamente, ela permanece ativada.
 - Temporizações ou contagens acionadas por ações associadas a esta etapa não seriam reinicializadas.



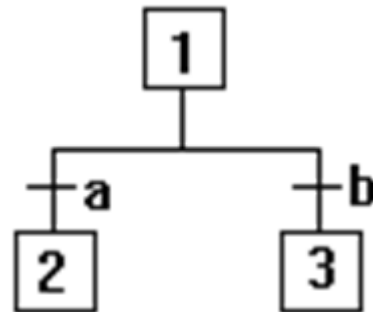
Estruturas Lógicas

Divergência em "OU":

IF ((etapa1.x=1) AND (a=1)) THEN (etapa2.x=1)

ELSE

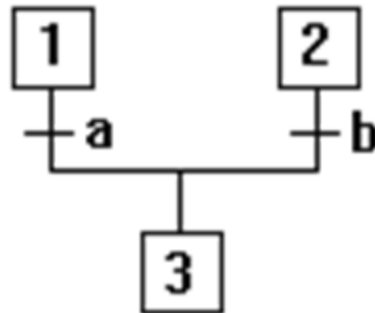
IF ((etapa1.x=1) AND (b=1)) THEN (etapa3.x=1).



Estruturas Lógicas

Convergência em "OU":

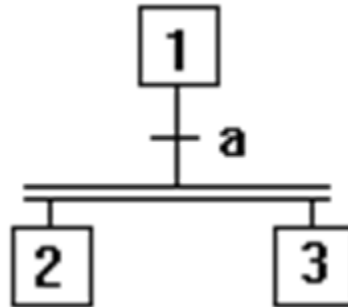
IF ((etapa1.x=1) AND (a=1)) OR ((etapa2.x=1) AND (b=1))
THEN (etapa3.x=1).



Estruturas Lógicas

Divergência em "E":

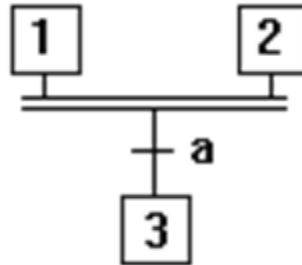
IF ((etapa1.x=1) AND (a=1))
THEN {etapa2.x=1); (etapa3.x=1)}.



Estruturas Lógicas

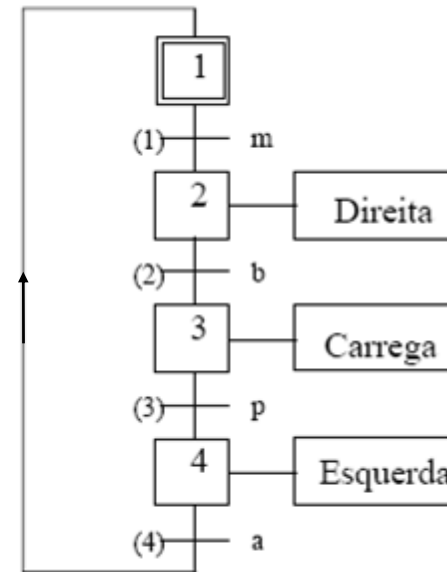
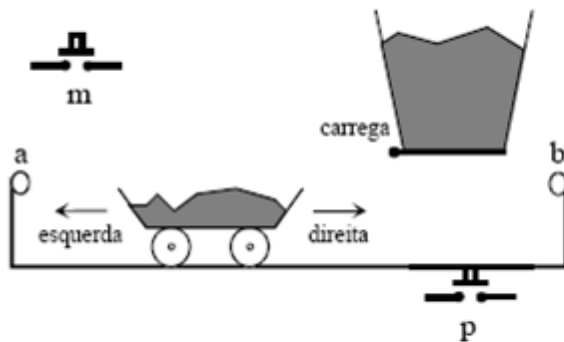
Convergência em "E":

IF ((etapa1.x=1) AND (etapa2.x=1) AND (a=1))
THEN (etapa3.x=1).



Exemplo simples

- Exemplo simples de modelagem em GRAFCET:



Ações na IEC 61131-3

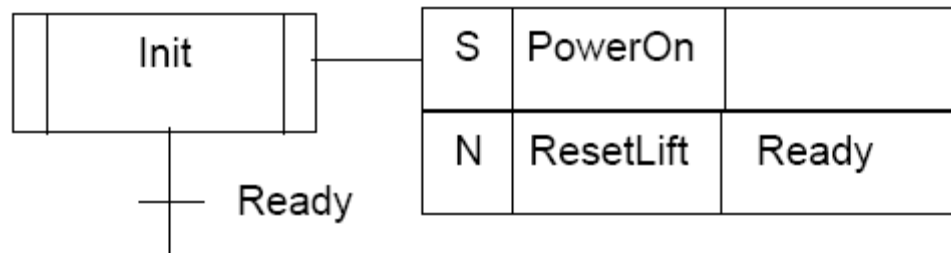
- As ações foram ampliadas na norma IEC 1131-3 e são representadas num retângulo contendo três campos:

Qualificador da Ação	Ação	Variável de Indicação
----------------------	------	-----------------------

- O qualificador da ação determina quando ela será executada.
- A variável de indicação (opcional) contém o nome de uma variável que é modificada dentro do corpo da ação e que indica que a ação foi completada.

Ações na IEC 61131-3

- Ações com qualificadores e variável de indicação:

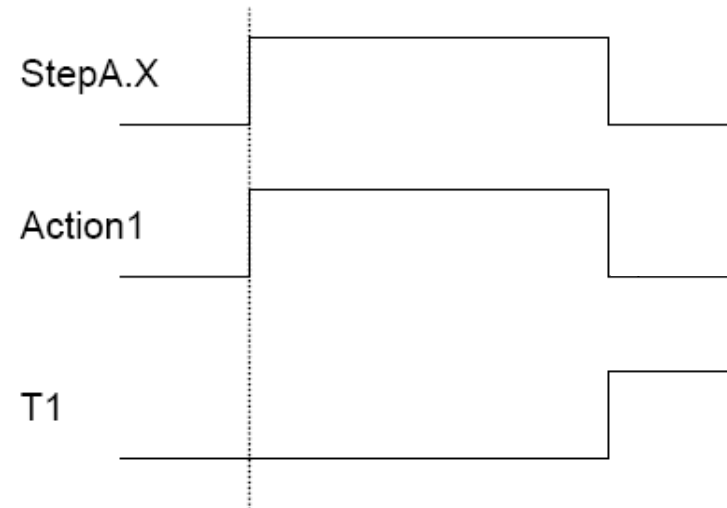
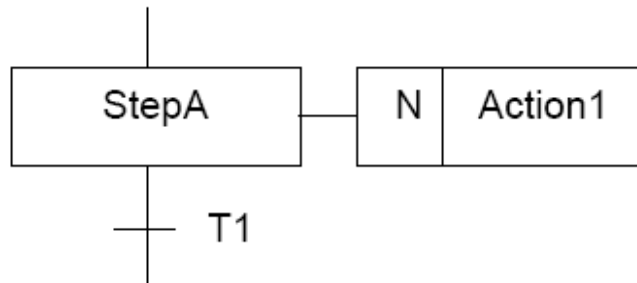


Qualificadores

Qualificador	Descrição
Nenhum	O mesmo que N
N	Non Stored (não armazenada)
R	Reset
S	Set (Stored)
L	Limited (limitada)
D	Delayed (atrasada)
P	Pulse
SD	Stored and Delayed
DS	Delayed and Stored
SL	Stored and Limited
P1	Pulse (rising)
P0	Pulse (falling)

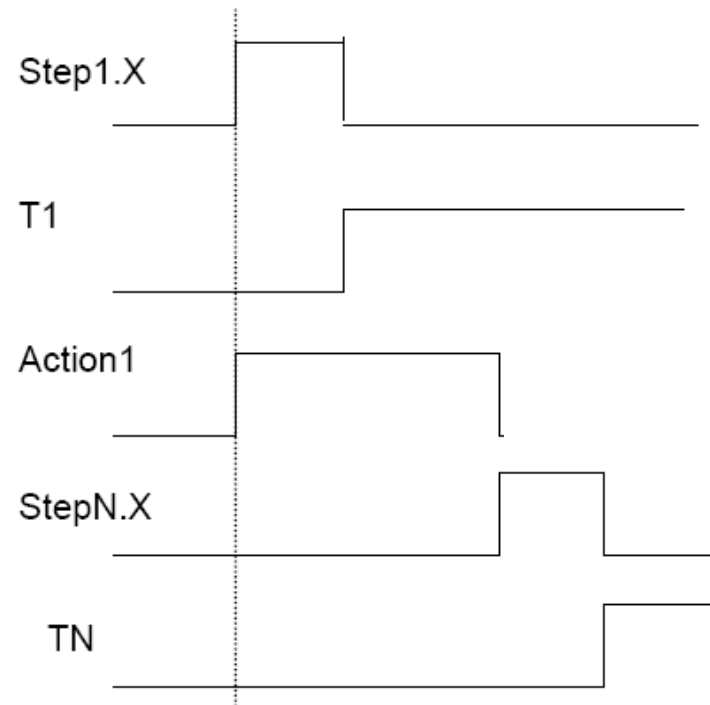
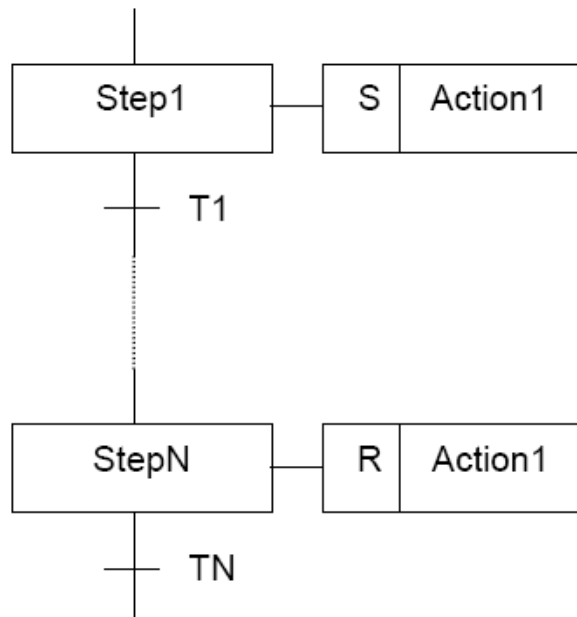
N - Ação não armazenada

- A ação executa continuamente enquanto o StepA está ativo (flag StepA.X = 1).



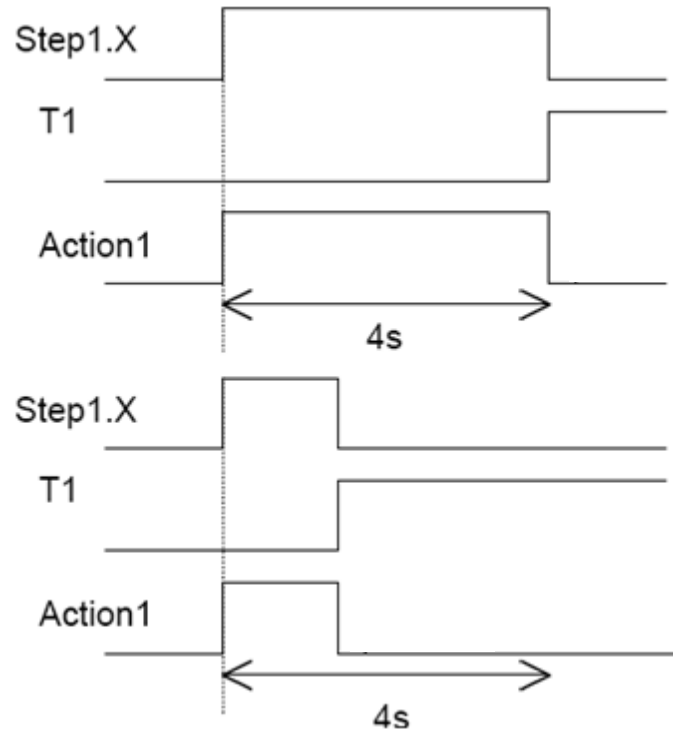
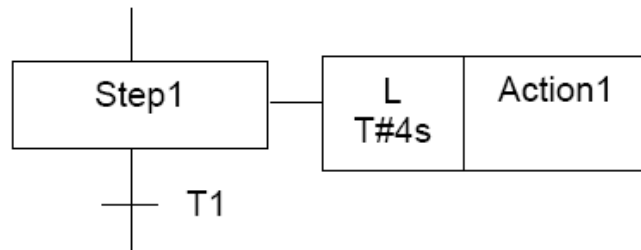
Set e Reset

- A ação começa a ser executada quando Step1 se torna ativo, e continua a ser executada até que o StepN é ativado.



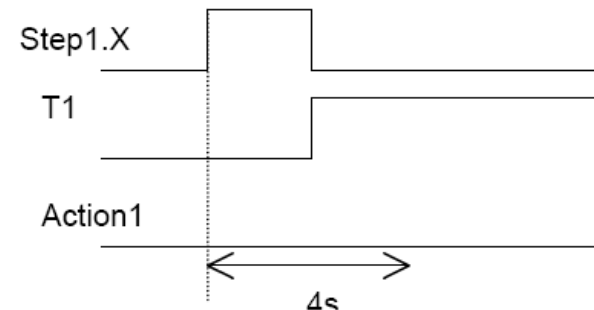
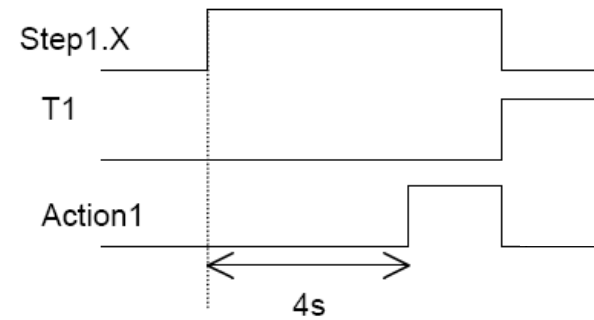
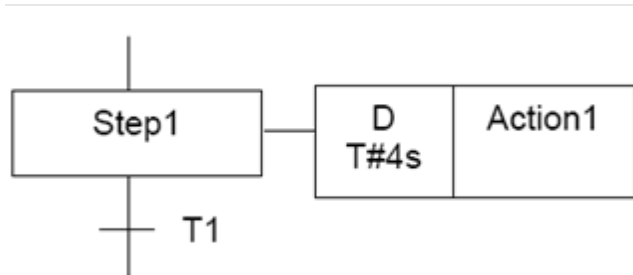
L - Ação limitada no tempo

- A ação executa durante o tempo estipulado a partir do instante de ativação da etapa.
- Se a etapa é desativada antes que o tempo tenha expirado, a ação interrompe sua execução.



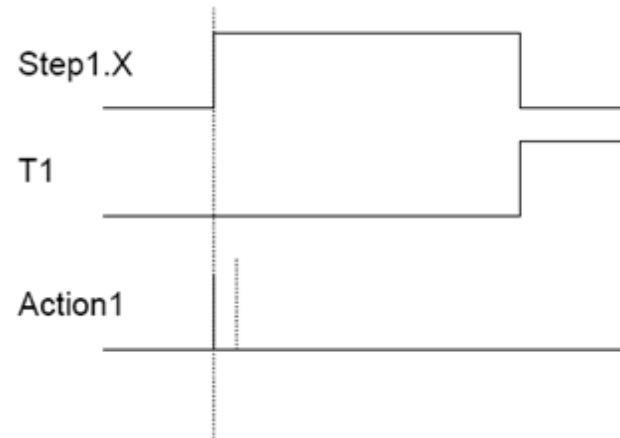
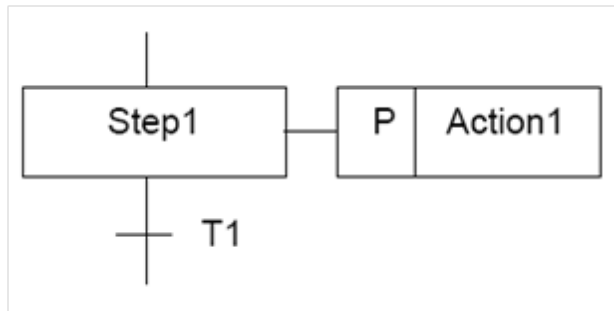
D - Ação com atraso de tempo

- A ação executa após um atraso de tempo estipulado, a partir do instante de ativação do estado.
- Se o estado é desativado antes que o tempo de atraso haja expirado, a ação não é executada.



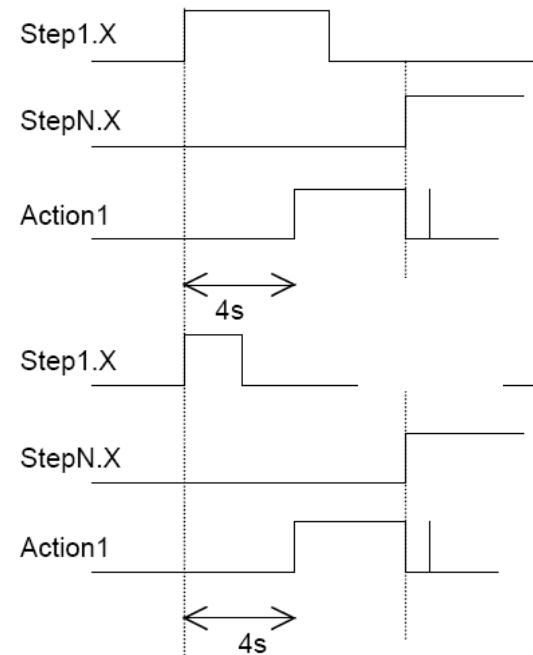
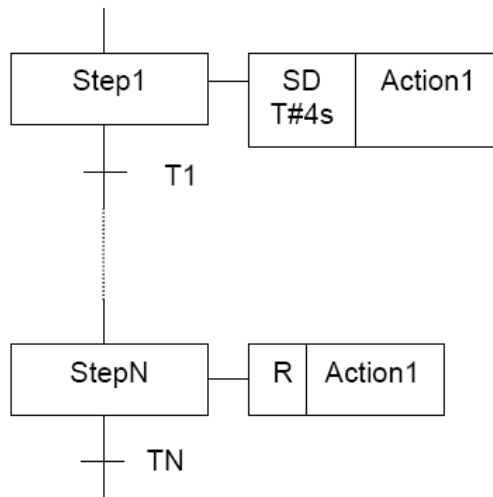
P - Ação pulsada

- A ação é executada uma única vez após o estado ter sido ativado.



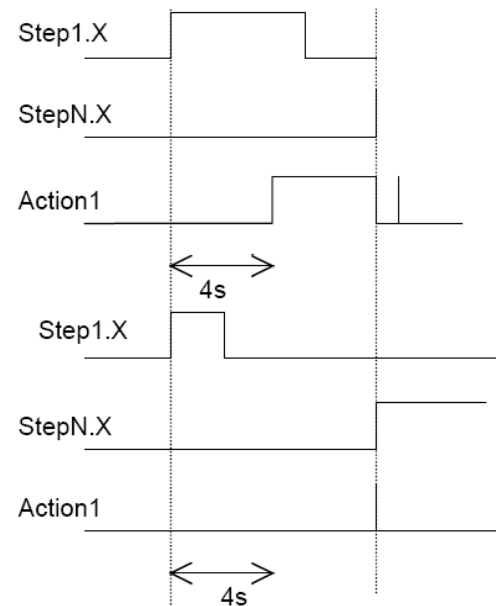
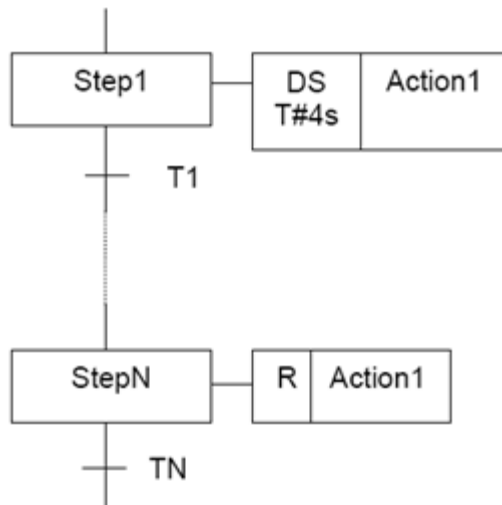
SD - Ação armazenada e atrasada

- Quando Step1 se torna ativo, a ação é armazenada, mas só começa a executar após o tempo ter se expirado.
- Independente da duração do Step1, a ação será executada até ser cancelada por uma ação com qualificador R referenciando a mesma ação (Action1).
- A ação não será executada se uma ação com qualificador R for executada antes do tempo de atraso expirar.



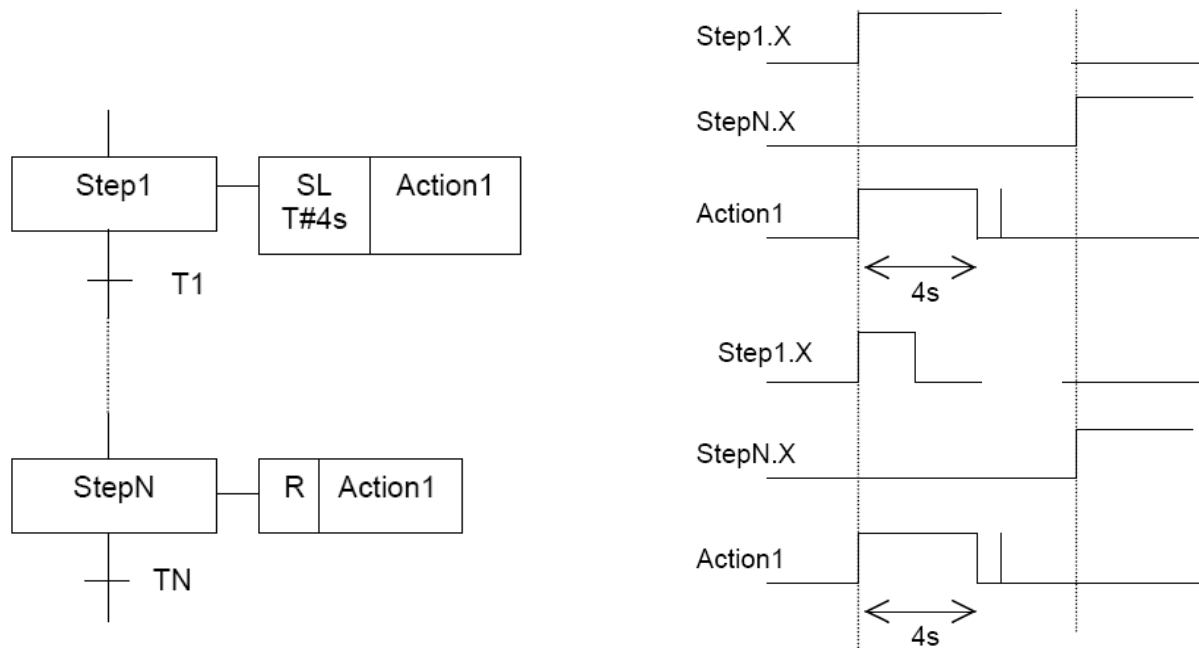
DS - Ação atrasada e armazenada

- No começo do Step1 o tempo de atraso começa a ser computado.
- Após o tempo ter expirado, a ação é armazenada e começa a ser executada.
- A ação continua a ser executada até ser novamente referenciada por uma ação com o qualificador R.
- Se o Step1 for desativado antes do tempo expirar, a ação não será armazenada e não será executada.



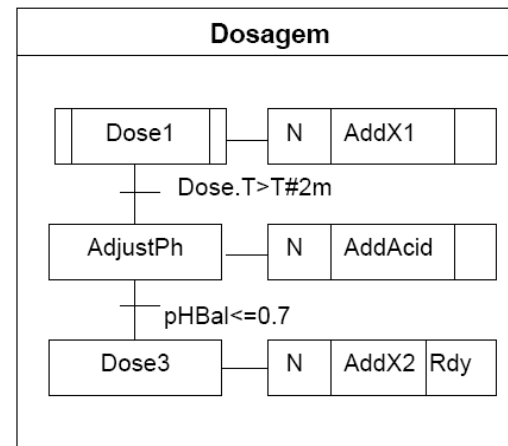
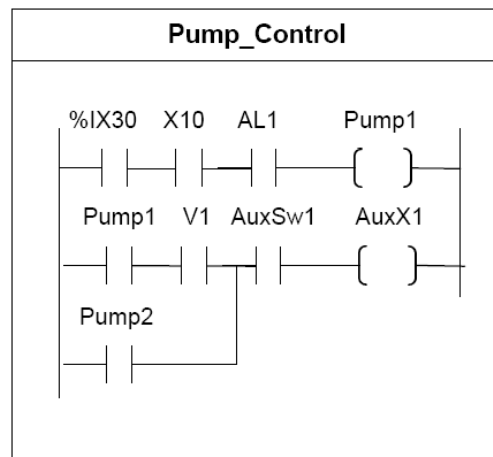
SL - Ação armazenada e limitada no tempo

- Quando o estado Step 1 se torna ativo, a ação é armazenada e começa a ser executada.
- Ela será executada até que o tempo T se esgote, independentemente do estado de Step1.
- Se a ação for desarmada por uma ação com qualificador R, antes do tempo se expirar, a ação será interrompida.



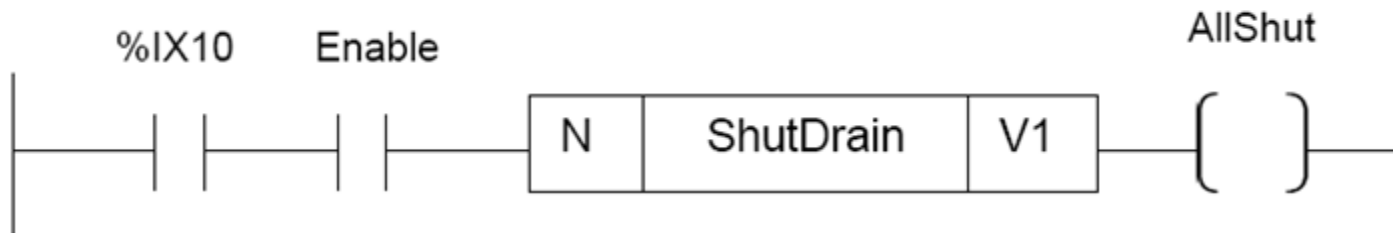
Ações em outras linguagens

- As ações ativadas pelas etapas podem ter a forma de sub-rotinas, descritas em qualquer uma das linguagens IEC 61131-3: ST, FBD, LD, IL ou GRAFCET.



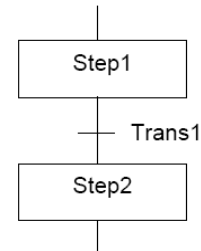
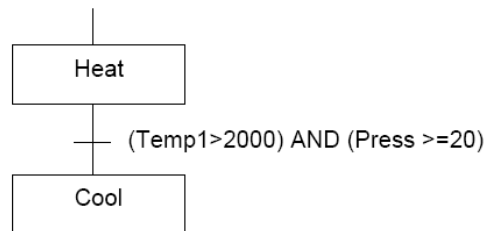
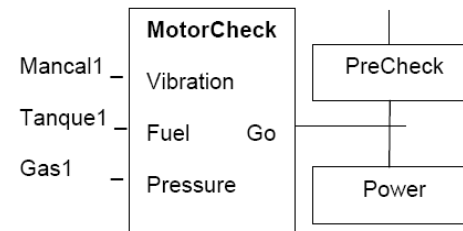
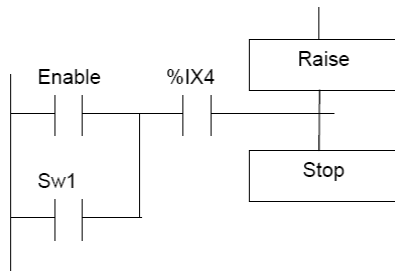
Ações em outras linguagens

- A norma prevê ainda que outras linguagens ativem ações tipo GRAFCET.



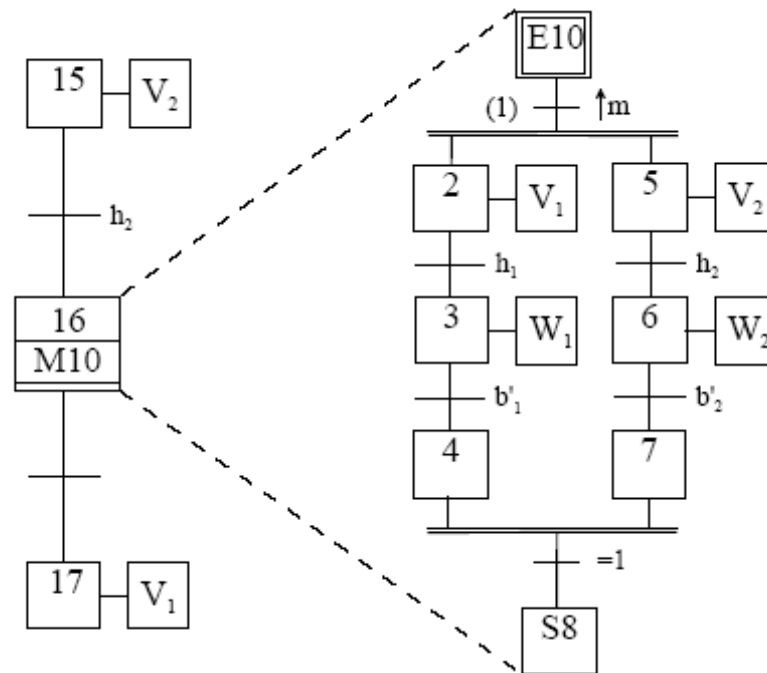
Transições na IEC 61131-3

- Também as transições podem estar definidas em qualquer linguagem da norma:



TRANSITION Trans1:
:= AB1 AND CX3 OR CX5 AND
(TX3 >= 100.2);
END_TRANSITION

Macro-Etapas



Análise e validação do GRAFCET

Um grafo bem projetado deve ter 5 propriedades:

1. Grafo Limitado e Seguro:

- Quando o número de fichas contido em qualquer etapa é inferior a um dado limite para qualquer marcagem possível, o grafo é dito *limitado*. Se este limite é “1”, o grafo é dito *seguro*. Uma etapa onde possam se acumular fichas em número ilimitado é indicativo de um erro de especificação.

2. Grafo Reinicializável:

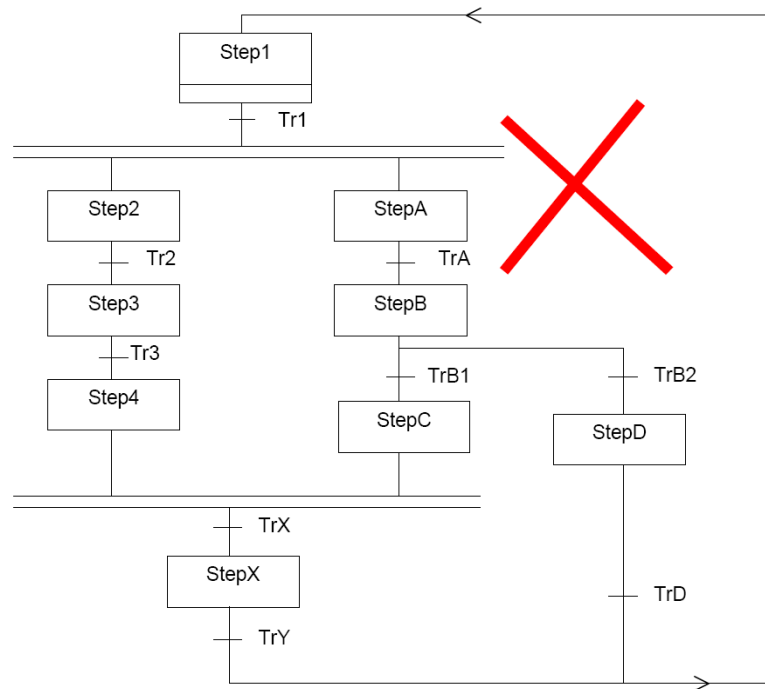
- Quando, para qualquer marcagem acessível atingida a partir da marcagem inicial, existe uma seqüência de disparo que permita o retorno a marcagem inicial, o grafo é dito reinicializável.

3. Grafo Vivo:

- Quando, para qualquer marcagem acessível M e qualquer transição T , existe uma seqüência de tiro disparável a partir de M que inclua T , o grafo é dito vivo. Isto equivale a dizer que não podem haver ramos do grafo por onde a ficha jamais passa.

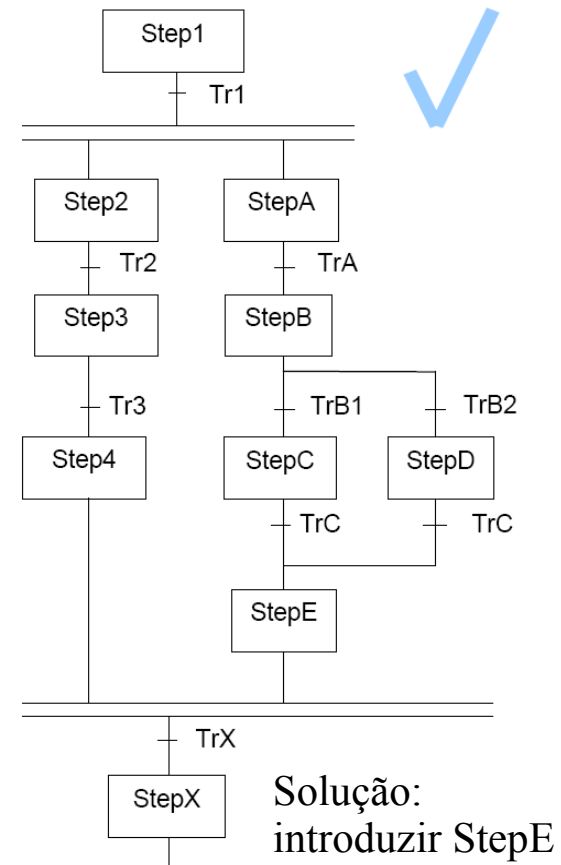
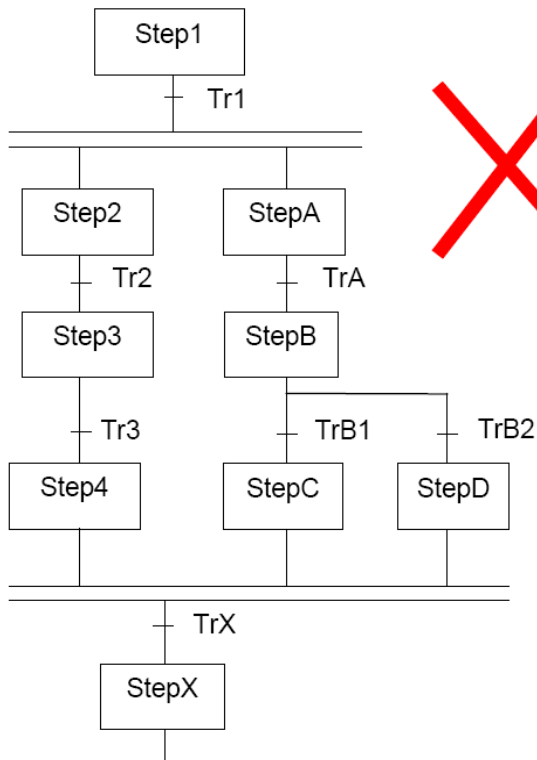
Análise e validação do GRAFCET

- Exemplo de grafo não seguro:
- Disparos sucessivos da transição TrB2 podem resultar em acúmulo de fichas no Step4.



Análise e validação do GRAFCET

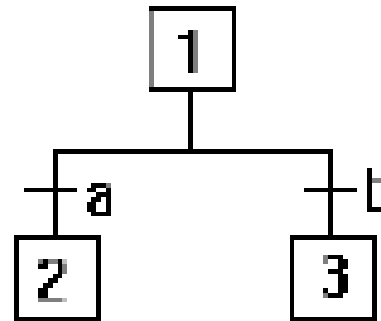
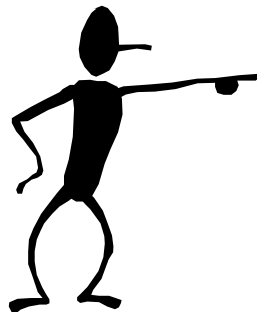
- Exemplo de grafo não vivo:
- StepX não é atingível, pois StepC e StepD são mutuamente exclusivos!



Análise e validação do GRAFCET

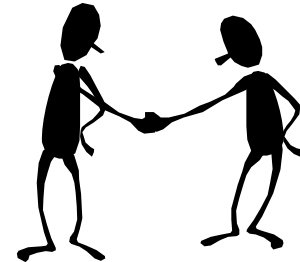
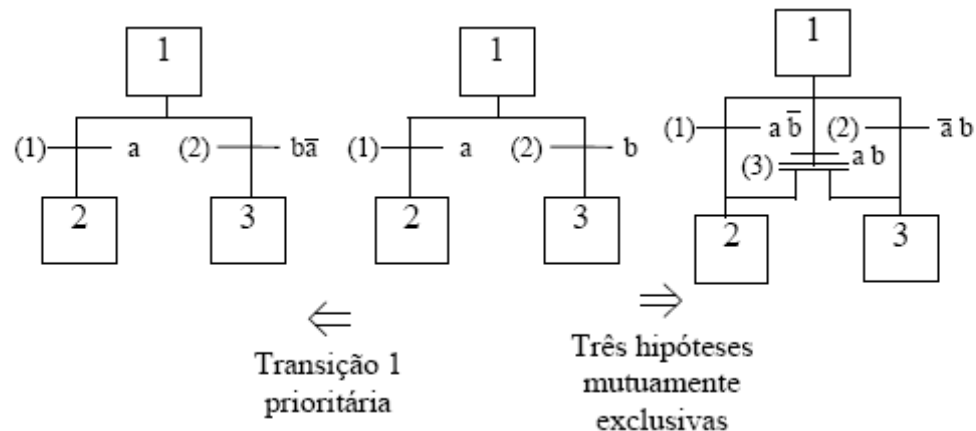
4. Grafo Determinista:

- um grafo é dito determinista quando todos os possíveis conflitos nele contidos estão resolvidos de forma inequívoca.
- Duas transições são ditas *em conflito* se existe uma marcagem acessível que sensibilize as duas ao mesmo tempo, de forma que o disparo de uma delas impeça o da outra.
- Normalmente a transição da esquerda é prioritária.



Análise e validação do GRAFCET

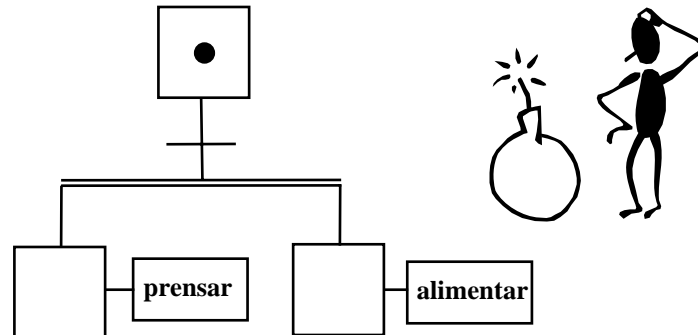
- Um *conflito* é dito *resolvido* se as etiquetas associadas às transições envolvidas não permitam nunca o valor “verdade” para as duas simultaneamente.



Análise e validação do GRAFCET

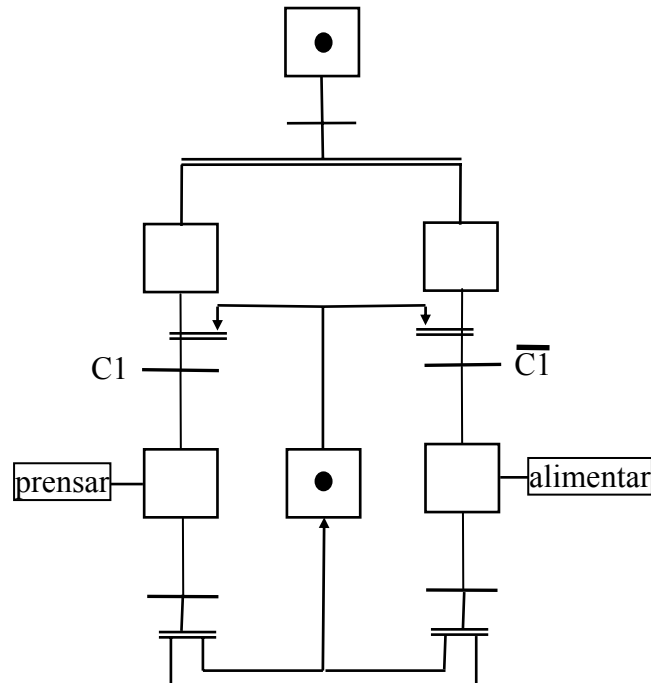
5. Grafo Determinado:

- um grafo é dito determinado se as ações associadas a etapas paralelas podem ser executadas simultaneamente.
- Duas etapas são ditas paralelas se existe uma marcagem acessível que ative as duas ao mesmo tempo.



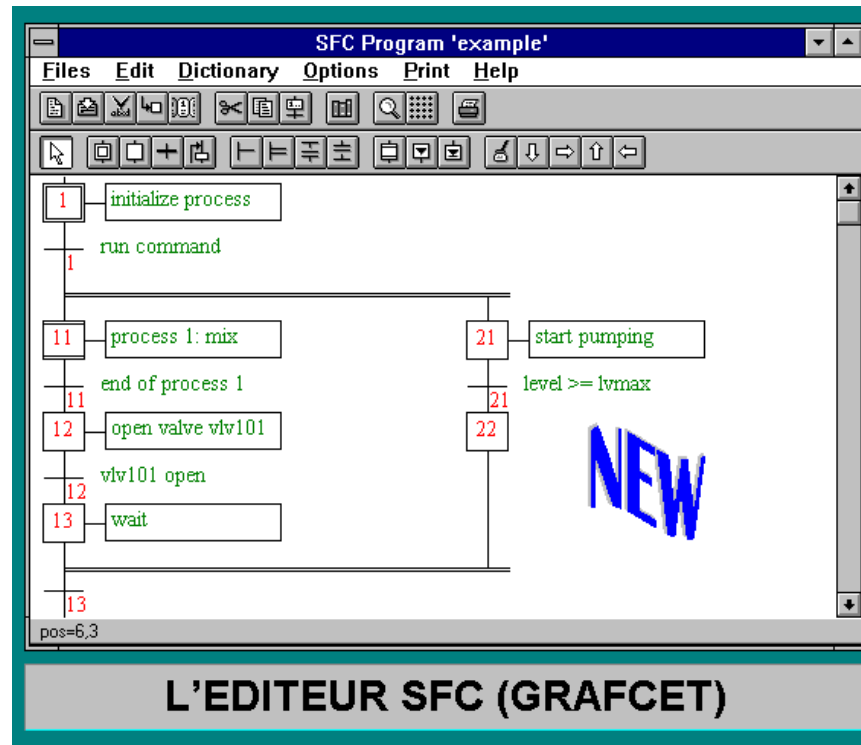
Análise e validação do GRAFCET

- Um grafo não determinado indica a necessidade de definir algum intertravamento faltante entre ações. Um intertravamento pode ser representado por uma exclusão mútua:



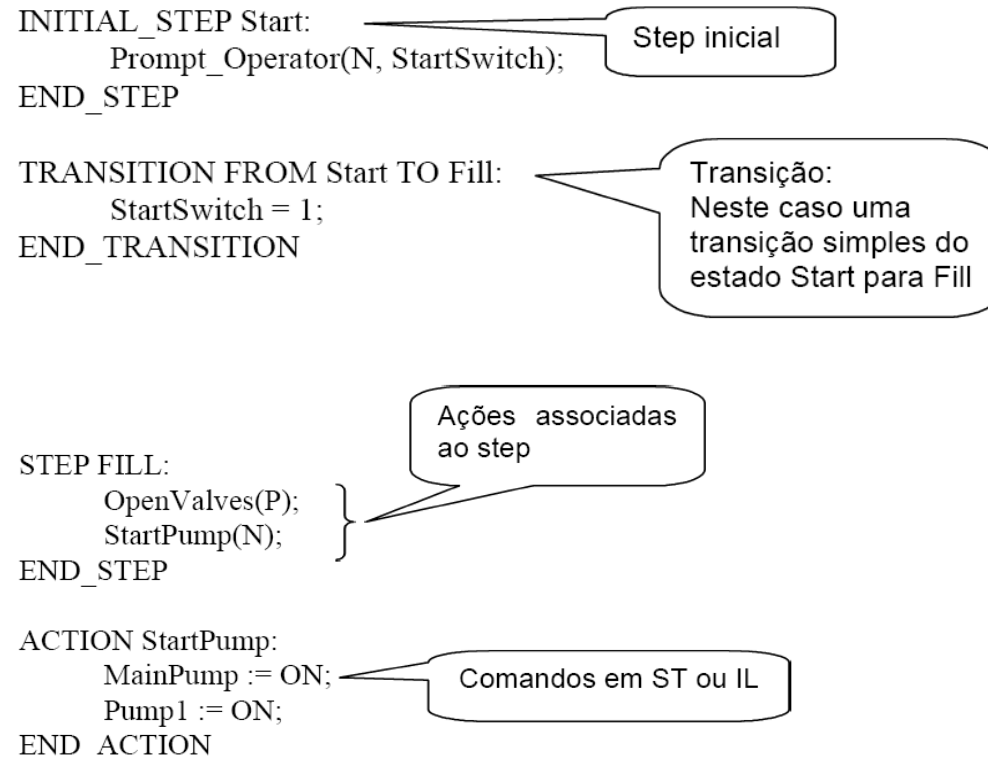
Edição do GRAFCET

- O GRAFCET pode ser programado com o auxílio de recursos gráfico-interativos ou por meio de uma linguagem textual que descreva o grafo.



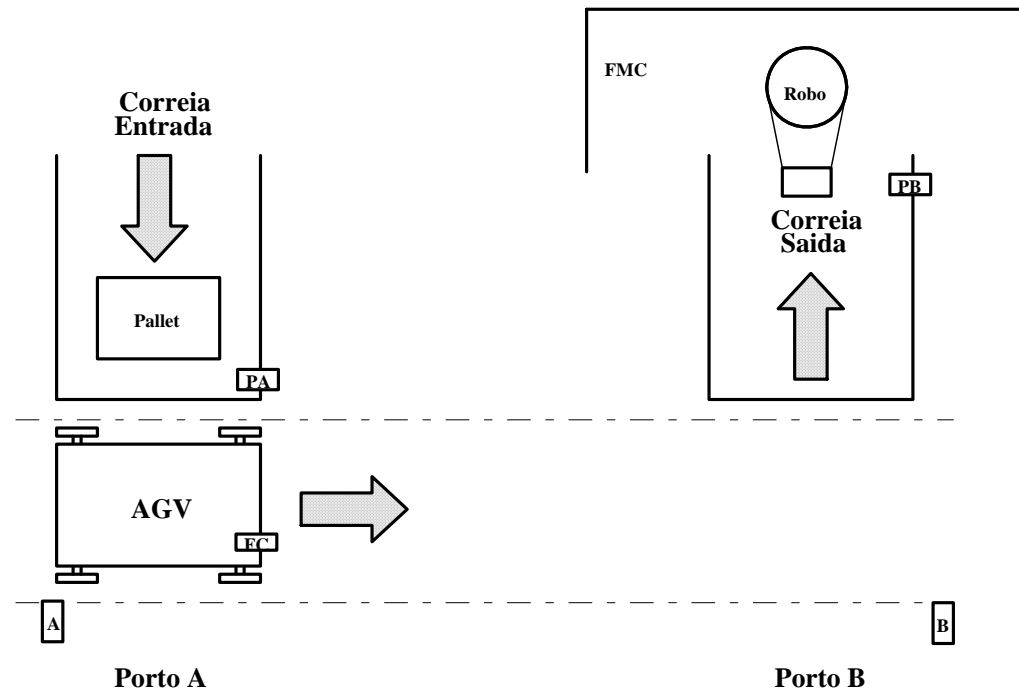
Edição do GRAFCET

- Grafcet em modo texto:



Exemplo de programação

- AGV deve transportar pallets do porto A ao porto B.

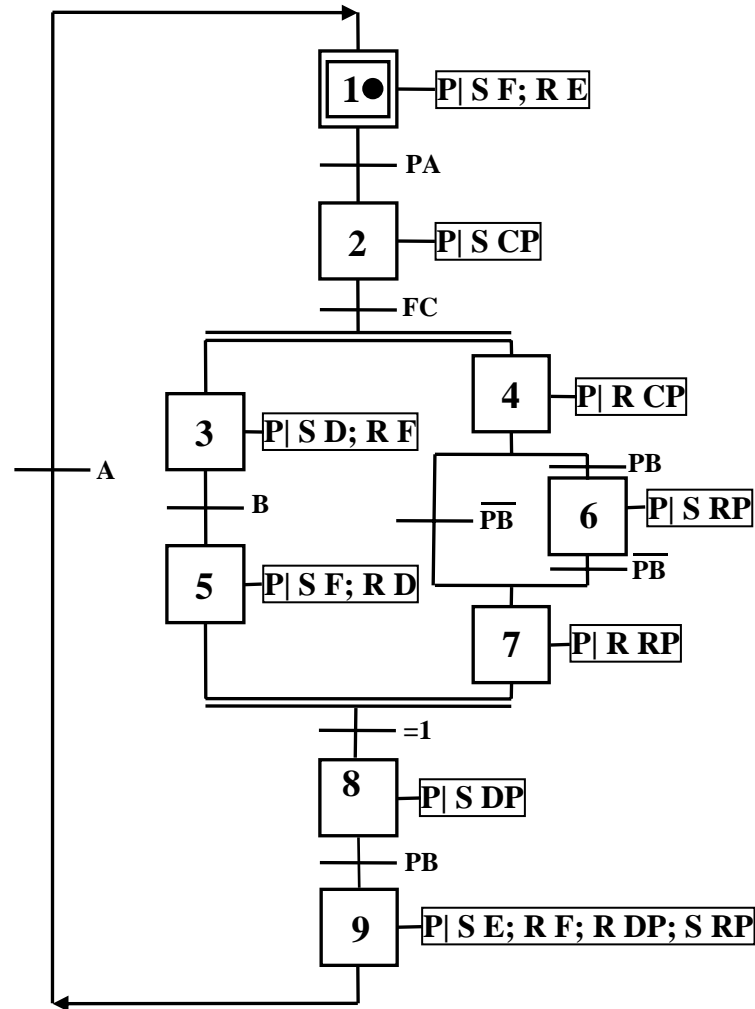


Exemplo de programação

- Sinais de Entrada: (Sensores)
 - PA = Pallet em A
 - PB = Pallet em B
 - FC = Fim de carregamento do Pallet no AGV
 - A = AGV no porto A
 - B = AGV no porto B
- Sinais de Saída: (Atuadores)
 - D = AGV se desloca para a Direita
 - E = AGV se desloca para a Esquerda
 - F = Freia AGV
 - CP = Carrega Pallet no AGV
 - DP = Descarrega Pallet do AGV
 - RP = Remove Pallet (Robô)

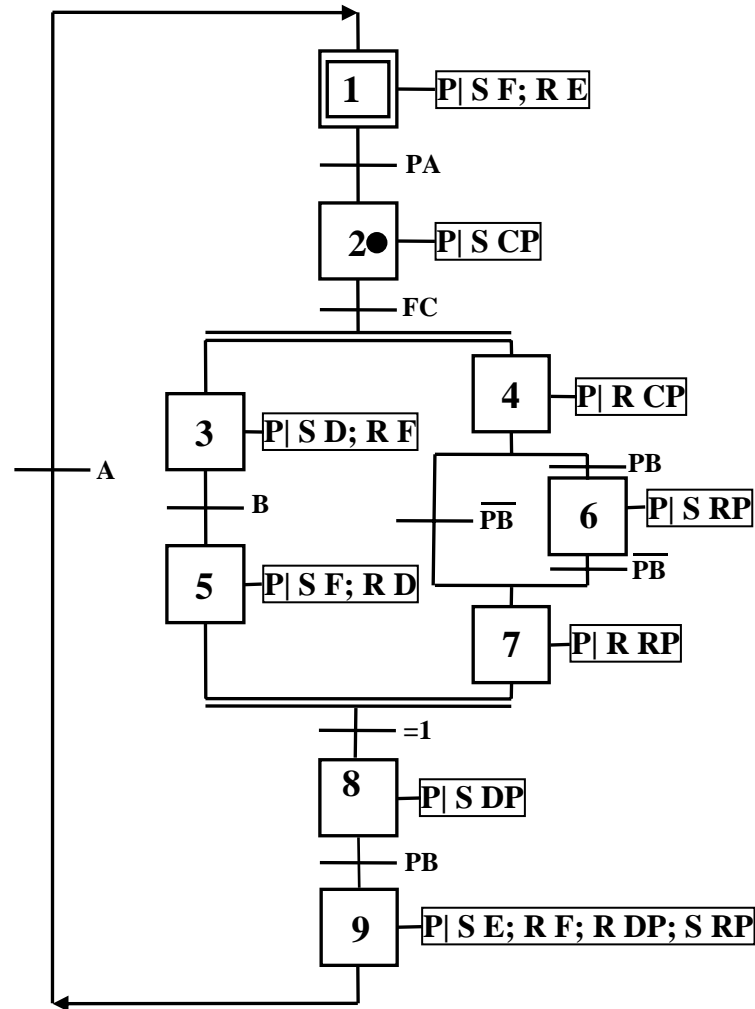
Exemplo de programação

- Solução:



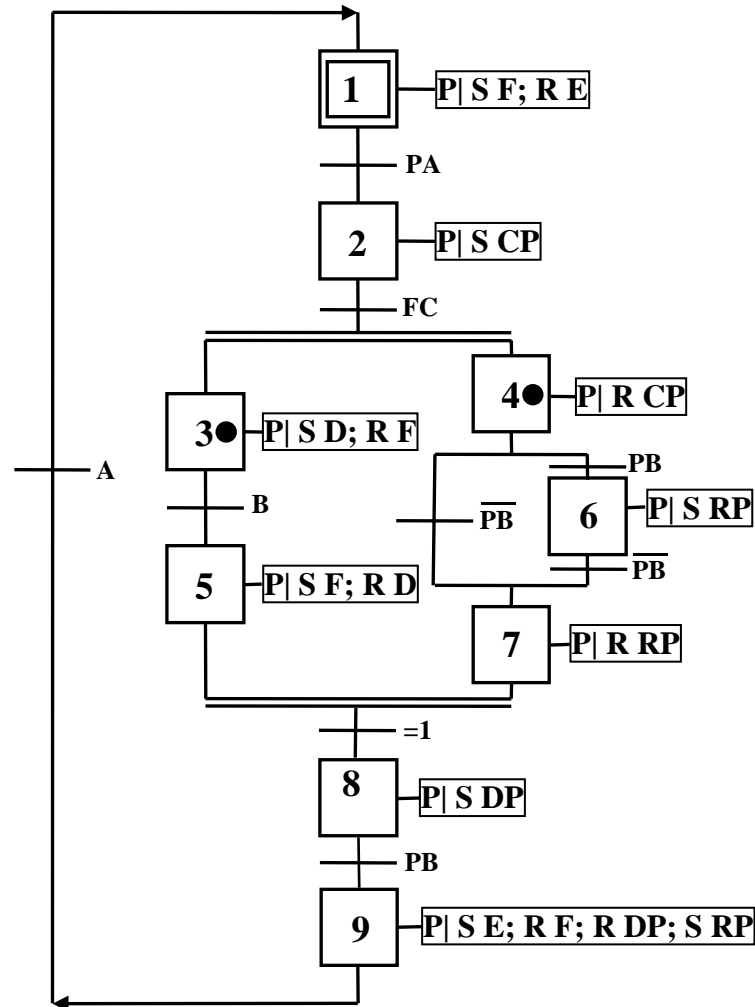
Exemplo de programação

- Solução:



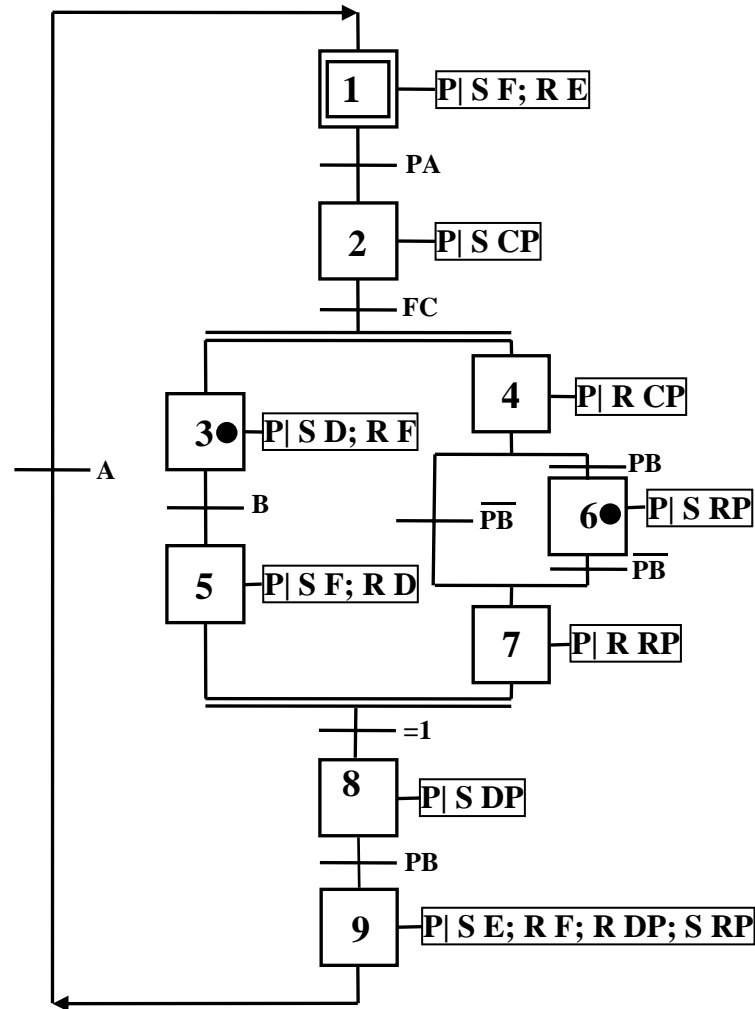
Exemplo de programação

- Solução:



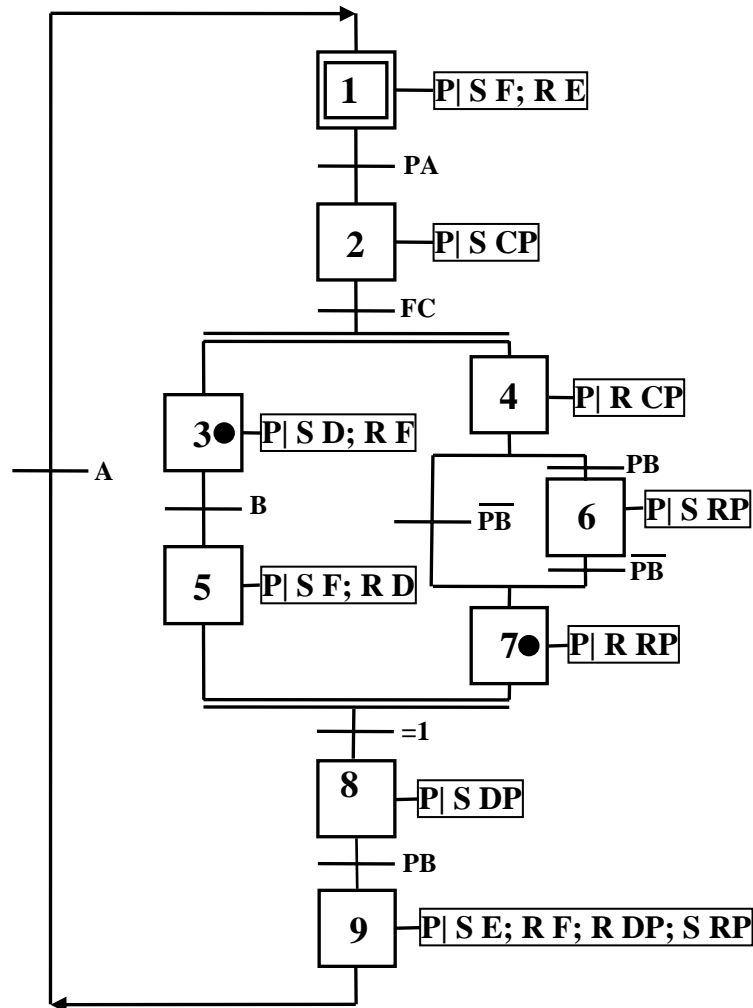
Exemplo de programação

- Solução:



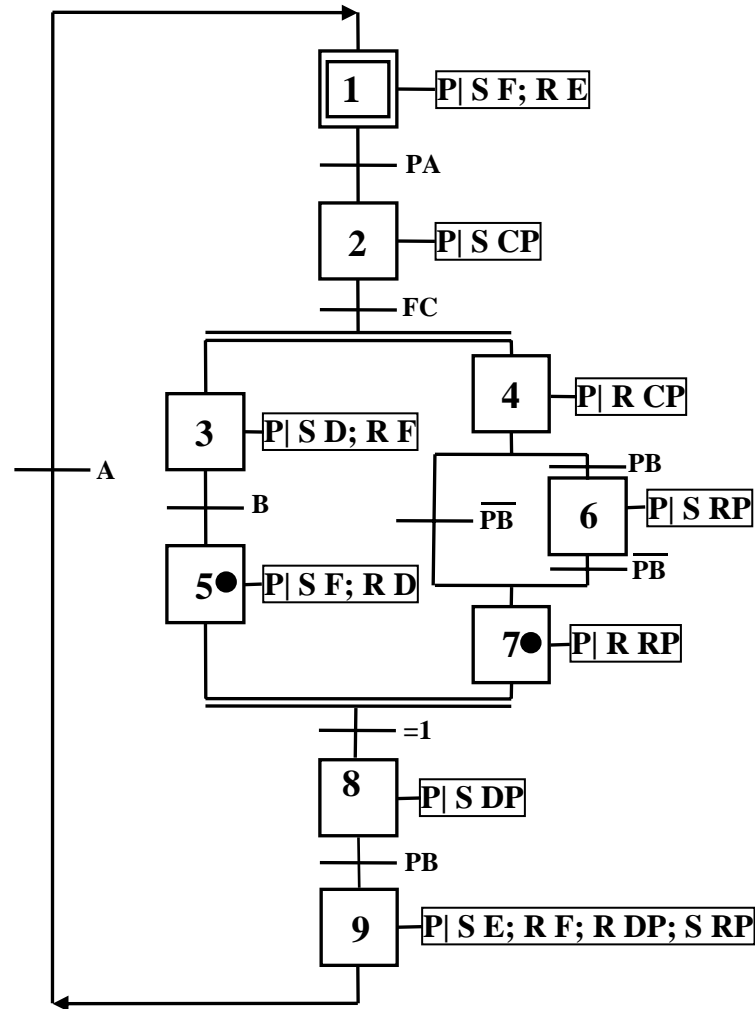
Exemplo de programação

- Solução:



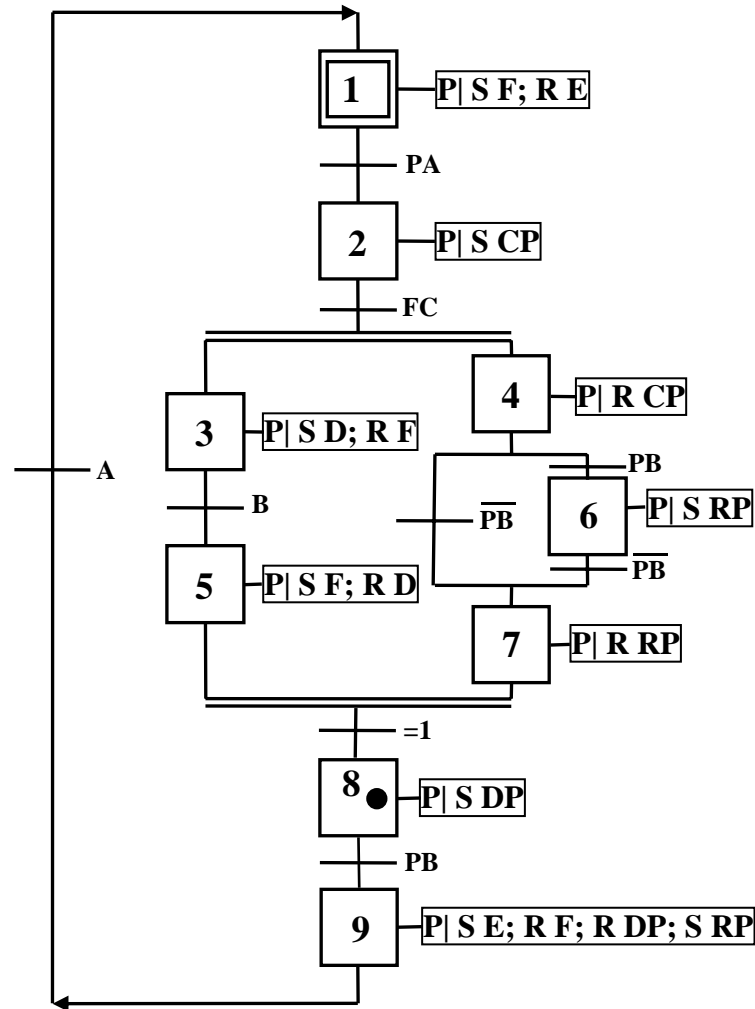
Exemplo de programação

- Solução:



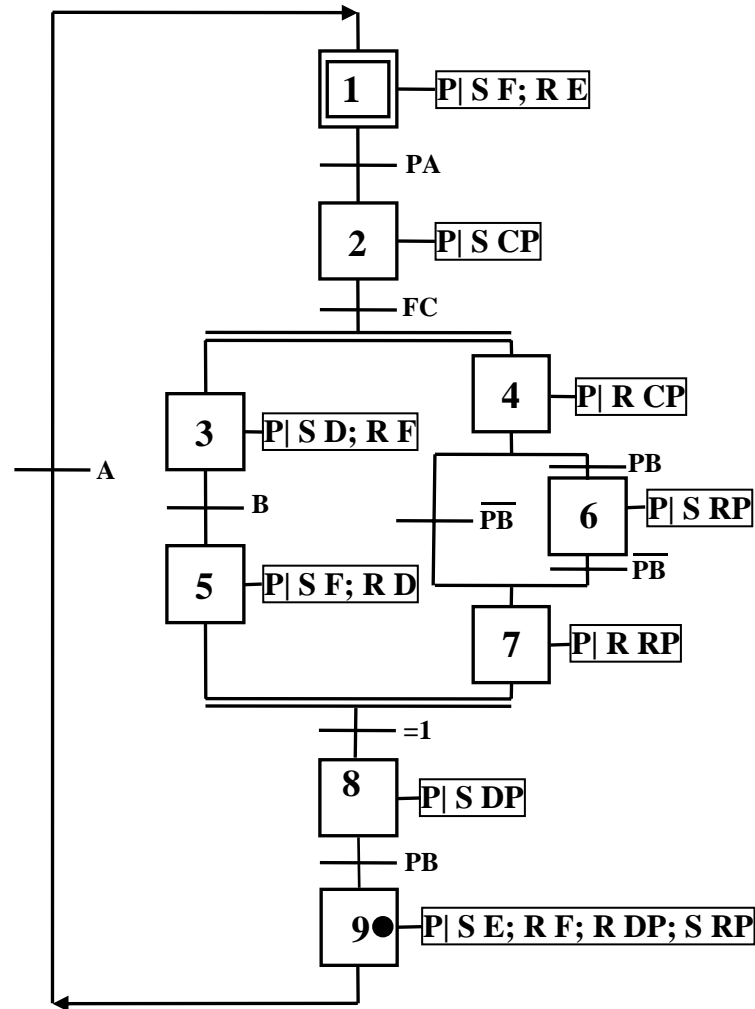
Exemplo de programação

- Solução:



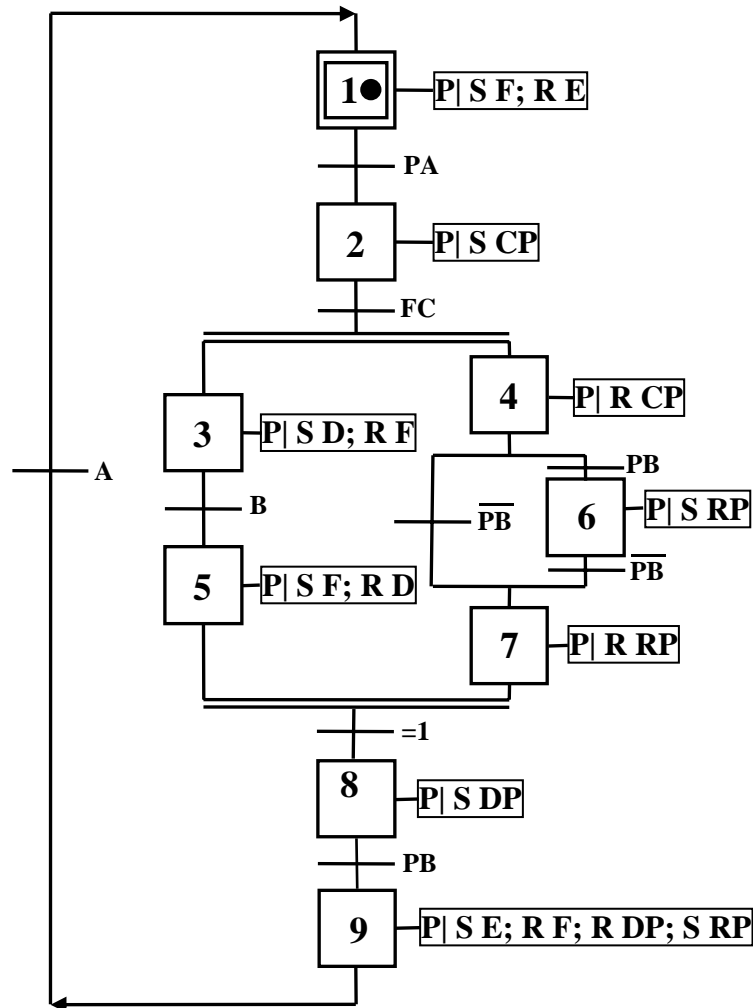
Exemplo de programação

- Solução:



Exemplo de programação

- Solução:



Exercício

- **Propor solução para este problema usando máquina de estados finitos!**

GRAFCET - Conclusão

- Alguns links sobre GRAFCET (SFC)
- Resenha histórica:
 - <http://www.ecsi.org/ecsi/Doc/OtherDoc/SLDL/PDF/caspi.pdf>
 - http://www.lurpa.ens-cachan.fr/grafcet/groupe/gen_g7_uk/geng7.html
- Tutorial:
 - http://asi.insa-rouen.fr/~amadisa/grafcet_homepage/tutorial/index.html
 - http://www-ipst.u-strasbg.fr/pat/autom/grafce_t.htm
- Simulador:
 - http://asi.insa-rouen.fr/~amadisa/grafcet_homepage/grafcet.html
 - <http://www.automationstudio.com>
- Bibliografia:
 - Programação de Autômatos, Método GRAFCET, José Novais, Fundação Calouste Gulbenkian
 - Petri Nets and GRAFCET: Tools for Modelling Discrete Event Systems, R. DAVID, H. ALLA, New York : PRENTICE HALL Editions, 1992
 - Norme Française NF C 03-190 + R1 : Diagramme fonctionnel "GRAFCET" pour la description des systèmes logiques de commande

GRAFCET - Conclusão

- Vantagens do uso do GRAFCET:
 - simplicidade de interpretação;
 - abrangência, podendo representar qualquer sistema de controle lógico industrial;
 - formalização rigorosa, permitindo análise e documentação precisas;
 - facilidade de decomposição do sistema em subsistemas funcionais;
 - facilidade de representação de sistemas com paralelismos, intertravamentos, ações seqüenciais, etc.
 - Pode ser usada como ferramenta para gerar programa em outra linguagem, como LD (em lugar da máquina de estados).

IL (Instruction List)

- Primeira linguagem de programação de CLP.
- Linguagem define mnemônicos como “assembly”.
- Mnemônicos representam operações lógicas booleanas e comandos de transferência de dados.
- Originalmente cada fabricante oferecia seu próprio conjunto de mnemônicos.
- Mnemônicos unificados pela norma IEC 61131-3.

Elementos básicos de IL

- Repertório básico de comandos:
 - AND <operando>: operação lógica E entre acumulador e operando;
 - ANDN <operando>: operação lógica E entre acumulador e negação do operando;
 - ANDR <operando>: operação E do flanco ascendente (rising) com o acumulador;
 - ANDF <operando>: operação E do flanco descendente (falling) com o acumulador;
 - AND LD: executa E entre acumulador e topo do stack;
 - OR <operando>: operação lógica OU entre acumulador e operando;
 - ORN <operando>: operação lógica OU entre acumulador e a negação do operando;
 - ORR <operando>: operação OU do flanco ascendente (rising) no operando com o acumulador;
 - ORF <operando>: operação OU do flanco descendente (falling) no operando com o acumulador;
 - OR LD: executa OU entre acumulador e topo do stack;
 - XOR <operando>: operação lógica OU Exclusivo entre acumulador e operando;

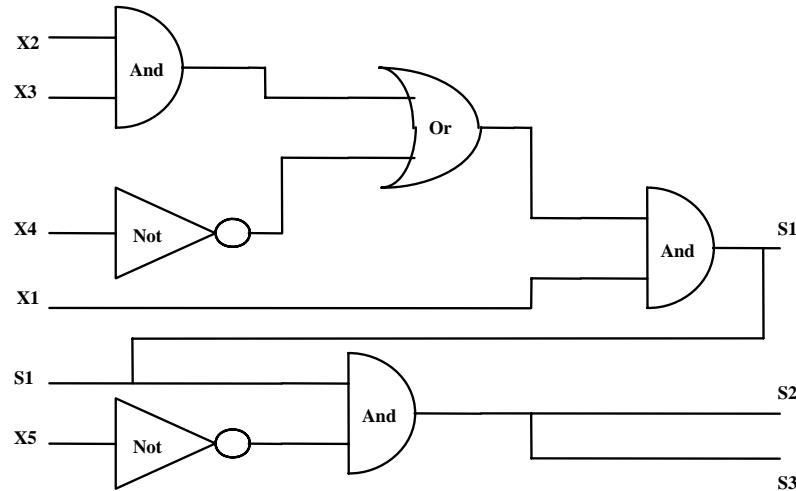
Elementos básicos de IL

- Repertório básico de comandos (cont):
 - LD <operando>: carrega conteúdo do operando no acumulador;
 - LDN <operando>: carrega conteúdo negado do operando no acumulador;
 - LDR <operando>: carrega 1 no acumulador se houver um flanco positivo (rising) no operando.
 - LDF <operando>: carrega 1 no acumulador se houver um flanco negativo (fallig) no operando;
 - ST <operando>: armazena conteúdo do acumulador no operando indicado;
 - STN <operando>: armazena conteúdo negado do acumulador no operando indicado;
 - S <operando>: seta operando em 1, se o conteúdo do acumulador for 1;
 - R <operando>: reseta operando em 0, se o acumulador contiver 1;
 - TM <n> #T: executa um retardo com temporização definida por “T”.

Elementos básicos de IL

- **Linguagem inclui ainda mnemônicos para:**
 - **Comparações:** GT, LT, EQ, GE, LE, NE;
 - **deslocamentos e rotações à esquerda e à direita:** SHR, SHL, ROR, ROL;
 - **Op. aritméticas:** ADD, SUB, MUL, DIV, MOD;
 - **Saltos:** JMP, JMPC, JMNC
 - **Sub-rotinas:** CAL, RET
 - **Contadores:** operações de contagem simples (incremental), contagem bidirecional (incremental / decremental) e temporização (delay);
 - **Conversão:** operações de conversão A/D, D/A, Binário / Decimal e Decimal / Binário;
 - **Comunicação:** envio e recepção de mensagens via rede;

IL – Exemplo simples



$$S1 = (X2.X3 + \overline{X4}).X1$$

$$S2 = S3 = S1.\overline{X5}$$

Programa do CLP:

```
LD      x2 // coloca x2 no acumulador
AND     x3 // acum = acum AND x3
ORN     x4 // acum = acum OR NOT x4
AND     x1 // acum = acum AND x1
ST      s1 // armazena acum em s1
ANDN    x5 // acum = acum AND NOT x5
ST      s2 // armazena acum em s2
ST      s3 // armazena acum em s3
```

IL - Jumps

- **Uso de Jumps (saltos):**

LD I1.0

JMPC L1

LD I1.1

AND I1.2

ST Q2.0

L1:

LD M20

ST M5

LD I1.3

OR I1.4

ST Q2.1

Executa jump para
label L1 se I1.0 for 1



IL - Nesting

- **Comandos aninhados (nesting):**

```
LD      I1.0
OR (    I1.1
ANDN    I1.2
AND     I1.3
)
ST      Q1.0
```

- Programa carrega I1.0 no acumulador e faz OR com resultado da operação entre parênteses (I1.1 AND NOT I1.2 AND I1.3)

IL - Stack

- **Uso da pilha (stack):**

```
LD      x1
AND     x2
LD      x3
AND     x4
OR      LD
ST      y1
```

Solução alternativa (nesting):

```
LD      x1
AND     x2
OR (    x3
AND     x4
)
ST      y1
```

- Programa empurra resultado do primeiro AND para a pilha, executa segundo AND e faz OR com o topo da pilha, isto é, $y = x1.x2 + x3.x4$

IL - Contadores

- **Uso de contadores:**

```
LD VAR1 // carrega variável no acumulador
ST C8.PV // coloca valor limite de contagem PV no contador C8
...
...
LD I1.0 // lê entrada a ser contada
CU C8 // incrementa (Count Up) contador (se transição 0->1)
LD C8.Q //carrega status do contador (vale 1 se limite atingido)
...
```

IL - Timers

- **Uso de temporizadores (timers):**

LD VAR1 // carrega variável no acumulador

ST T1.PT // coloca tempo limite PT no timer T1

...

LD I1.0 // lê entrada que dispara timer (transição 0->1)

IN T1 // insere no timer T1

...

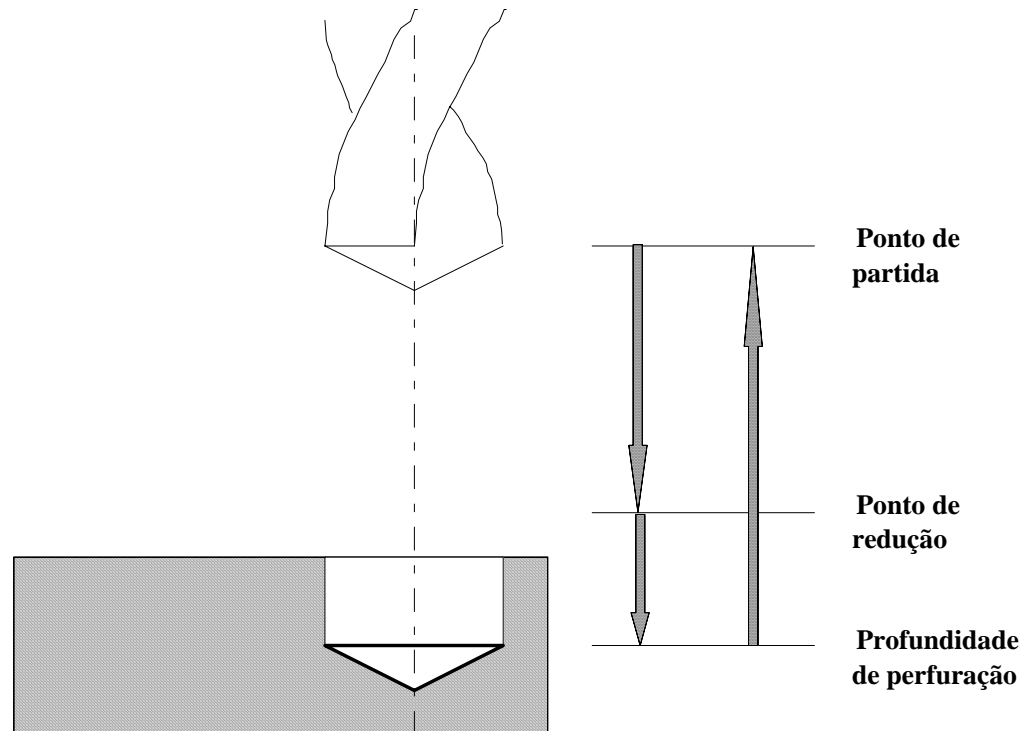
LD T1.Q // carrega status do timer (vale 1 se tempo atingido)

STEP-7 (Siemens)

Grupo	Operação	Descrição
operadores lógicos	U UN O ON	lógico E (Und) lógico E NÃO (Und Nicht) lógico OU (Oder) lógico OU NÃO (Oder Nicht)
memória	S R	Seta E/S/Variável Reseta E/S/Variável
contadores	S R FR ZV ZR	Seta contador Reseta contador libera contador (FReizesten) contagem p/ frente contagem p/ trás
teste de bits	P PN SU RU	Testa se bit = 1 (Pruefen) Testa se bit = 0 Seta incondicional Reseta incond.
carregar e transferir	L T LC	Carrega (Laden) Transfere (Transferieren) Carrega codificado
comparações	= < > >< <= >=	igual menor maior diferente menor ou igual maior ou igual
operações matemáticas	+ - x :	adiciona subtrai multiplica divide
Pulos	SPA SPB	Pula absoluto (Springe Absolut) Pula condicional (Springe Bedingt)
Deslocamento	SLW SRW RLD RRD	Desloca a esquerda Desloca a direita rotaciona a esquerda rotaciona a direita

Exemplo de IL: Furadeira

- Implementação de parte da lógica de comando de uma furadeira:



Exemplo de IL: Furadeira

- **Sinais de Entrada:**
 - **x1: comando de partida (operador);**
 - **x2: furadeira na posição de partida;**
 - **x3: ponto de redução alcançado;**
 - **x4: profundidade de perfuração alcançada.**
- **Sinais de Saída:**
 - **y1: ligar/desligar avanço rápido;**
 - **y2: ligar/desligar avanço lento com rotação;**
 - **y3: ligar/desligar retrocesso rápido.**

Exemplo de IL: Furadeira

- Operação básica:
 - Se a furadeira estiver na posição de partida ($x_2=1$) e um comando de partida foi dado ($x_1=1$), ligar avanço rápido ($y_1=1$) até atingir o ponto de redução ($x_3=1$).
 - Neste ponto, desligar o avanço rápido ($y_1=0$) e ligar o avanço lento com rotação ($y_2=1$).
 - Ao atingir a profundidade de perfuração desejada ($x_4=1$), desligar o avanço lento com rotação ($y_2=0$) e ligar o retrocesso rápido ($y_3=1$), até retornar a posição de partida ($x_2=1, y_3=0$).

Exemplo de IL: Furadeira

- Programa do CLP:

LD	x1 //	SE comando de partida
AND	x2 //	E posição de partida
S	y1 //	ENTÃO liga avanço rápido
LD	y1 //	SE avanço rápido ligado
AND	x3 //	E ponto de redução
R	y1 //	ENTÃO desl. avanço rápido
S	y2 //	e liga avanço lento com rotação
LD	y2 //	SE avanço lento com rotação ligado
AND	x4 //	E profundidade de perfuração atingida
R	y2 //	ENTÃO desl. avanço lento com rotação
S	y3 //	e liga retrocesso rápido
LD	y3 //	SE retrocesso rápido ligado
AND	x2 //	E pos. partida atingida
R	y3 //	desliga retrocesso rápido

IL - Conclusão

- **Vantagens:**

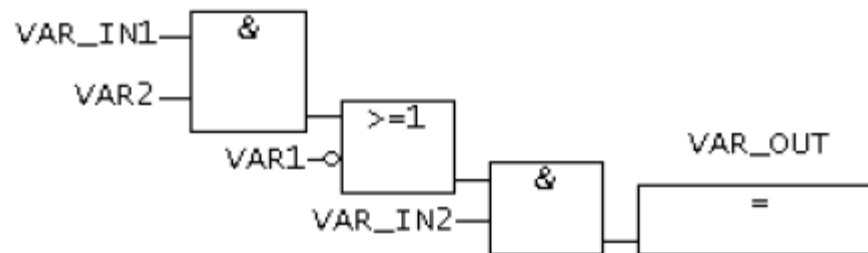
- correspondência entre comandos da linguagem e as instruções assembly do processador, facilitando uma estimativa do tempo de execução do programa;
- documentação mais compacta do que a equivalente com relês.

- **Desvantagens:**

- necessidade de familiarização do operador com álgebra booleana;
- necessidade de uma certa familiarização com programação em assembly;
- alterações trabalhosas nas lógicas já implementadas.
- Difícil implementar lógicas complexas!

FBD

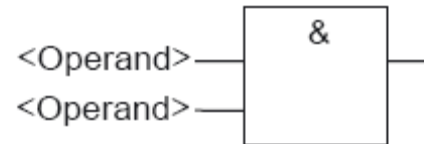
- FBD = Function Block Diagram.
- Linguagem gráfica prevista na norma IEC 61131-3.
- Utiliza blocos de função semelhantes aos de álgebra booleana.
- Vê sistemas em termos do fluxo de sinais entre elementos de processamento.



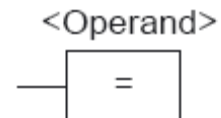
Blocos FBD

- Função AND:

- Bloco básico:



- Atribuição:



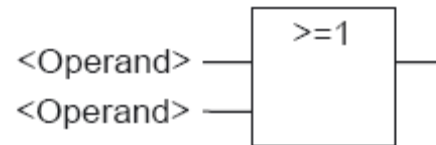
- Exemplo:



Blocos FBD

- Função OR:

- Bloco básico:



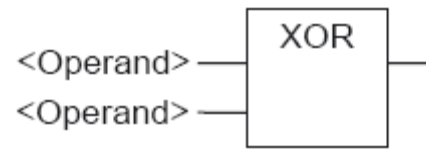
- Exemplo:



Blocos FBD

- Função XOR:

- Bloco básico



- Exemplo:



Blocos FBD

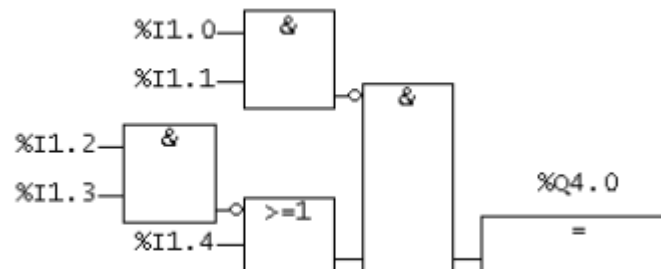
- Negação de entrada:

- Bloco básico:

<Operand>

—o|

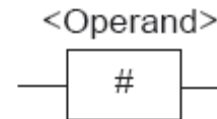
- Exemplo:



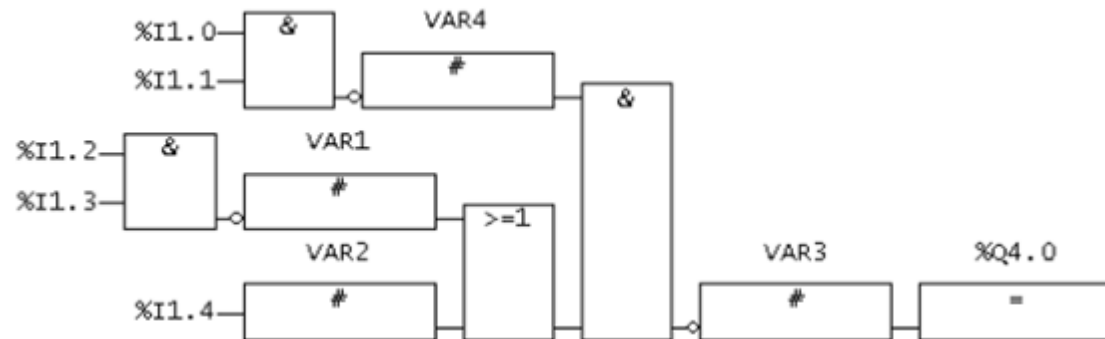
Blocos FBD

- Conector:
- Armazena resultado parcial de uma função em um operando

- Bloco básico:



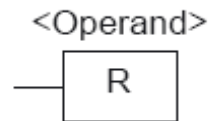
- Exemplo:



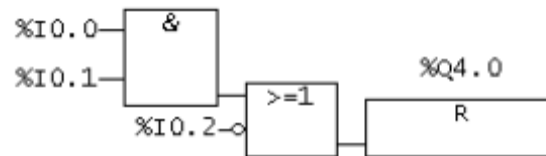
Blocos FBD

- Reset:

- Bloco básico:



- Exemplo:



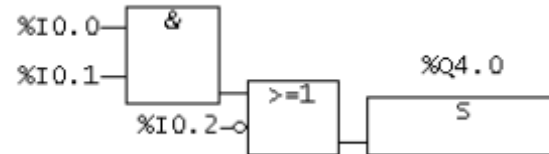
Blocos FBD

- Set:



- Bloco básico:

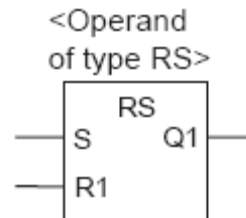
- Exemplo:



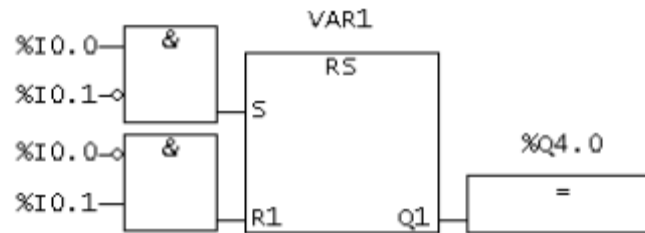
Blocos FBD

- Flip-Flop RS:
- Prioridade para Reset: sinal 1 em ambas as entradas causa reset.

- Bloco básico:



- Exemplo:



Se %I 0.0 = 1 e %I 0.1 = 0, VAR1 é resetada e a saída %Q 4.0 é **0**.

Se %I 0.0 = 0 e %I 0.1 = 1, VAR1 é setada e a saída %Q 4.0 é **1**.

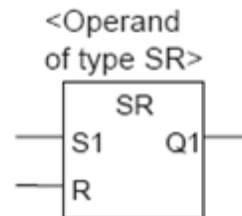
Se ambas as entradas são **0**, nada muda.

Se ambas as entradas são **1**, reset tem prioridade (VAR1 é resetada e %Q 4.0 é **0**).

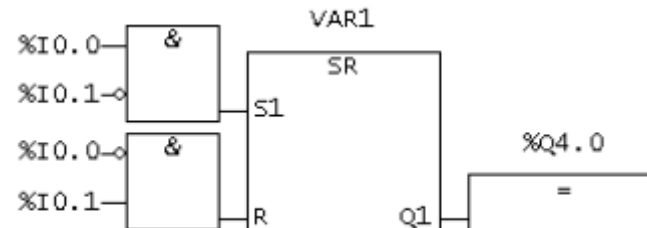
Blocos FBD

- Flip-flop SR:
- Prioridade para Set: sinal 1 em ambas as entradas causa set.

- Bloco básico:



- Exemplo:



Se %I 0.0 = 1 e %I 0.1 = 0, VAR1 é setada e %Q 4.0 é **1**.

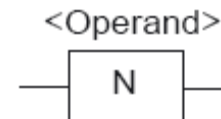
Se %I 0.0 = 0 e %I 0.1 = 1, VAR1 é resetada e %Q 4.0 é **0**.

Se ambas as entradas são **0**, nada muda.

Se ambas as entradas são **1**, set tem prioridade (VAR1 é setada e %Q 4.0 é **1**).

Blocos FBD

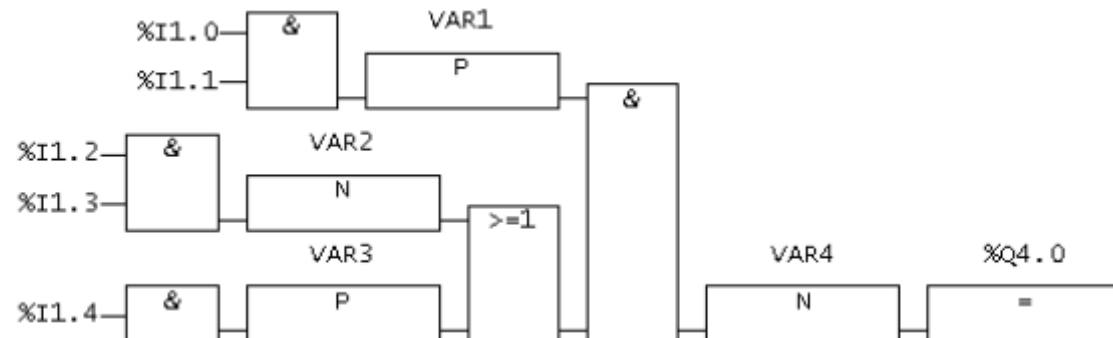
- Varredura de flanco Negativo (scan edge 1->0)
- Como Bobina Detectora de Transição Negativa
- Se houve flanco negativo, operando vai para 1



- Varredura de flanco Positivo (scan edge 0->1)
- Como Bobina Detectora de Transição Positiva
- Se houve flanco positivo, operando vai para 1

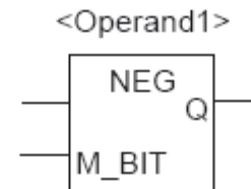


- Exemplo:

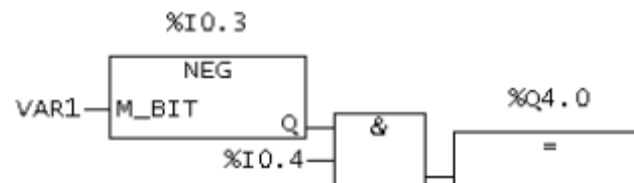


Blocos FBD

- Detecção de Flanco Negativo no Operando (descendente):
- M_BIT indica a variável onde o estado anterior do operando1 foi salvo
- Bloco básico:



- Exemplo:



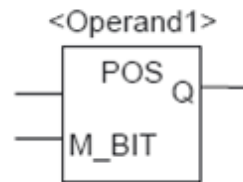
A saída %Q 4.0 é **1** se:

- entrada %I 0.3 tem um flanco negativo
- E a entrada %I 0.4 = **1**.

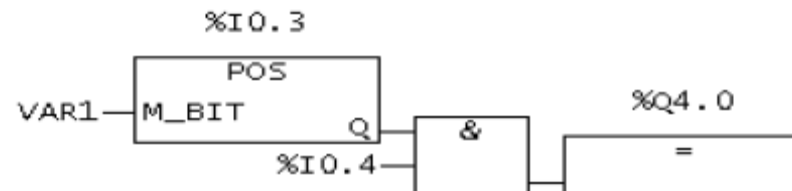
Blocos FBD

- Detecção de Flanco Positivo no Operando (ascendente):

- Bloco básico:



- Exemplo:

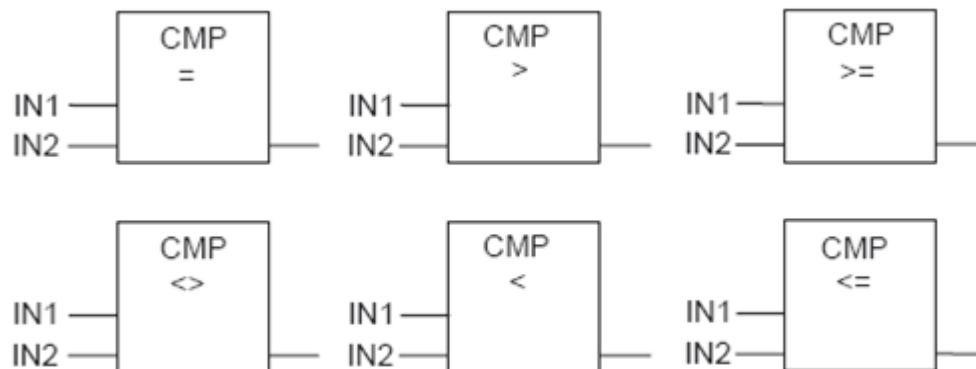


A saída %Q 4.0 é **1** se:

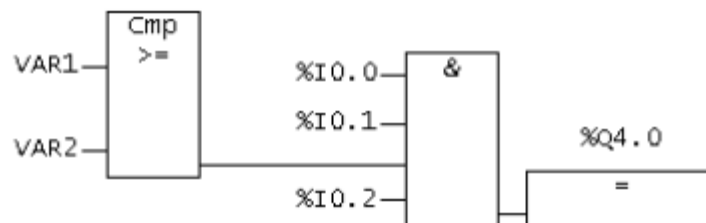
- entrada %I 0.3 tem um flanco positivo
- E a entrada %I 0.4 = **1**.

Blocos FBD

- Comparações:

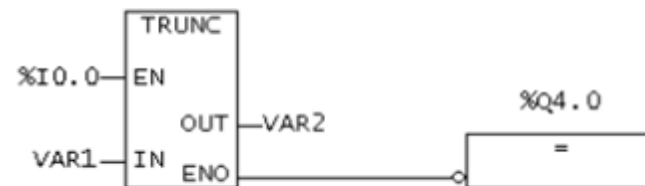
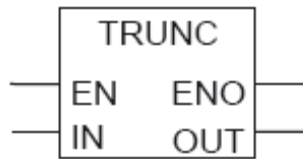


- Exemplo



Blocos FBD

- Conversões:
- TRUNC, BOOL_TO_BYTE, BYTE_TO_BOOL, BYTE_TO_SINT, BYTE_TO_UINT, INT_TO_REAL, etc.



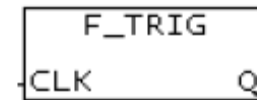
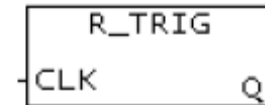
Var1 é real de ponto flutuante

Var2 conterá inteiro de 32 bits

%Q4.0 será 1 se ocorreu erro

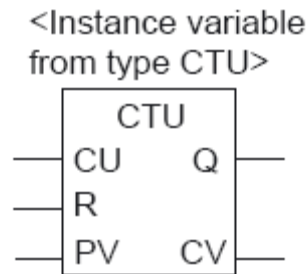
Blocos FBD

- Relés de Pulso:
- R_TRIG: Se a entrada passa de 0 para 1, a saída fica em 1 por um ciclo de varredura (Rising).
- F_TRIG: se a entrada passa de 1 para 0, a saída fica em 1 por um ciclo de varredura (Falling).



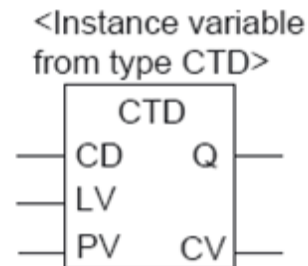
Blocos FBD

- Contadores:



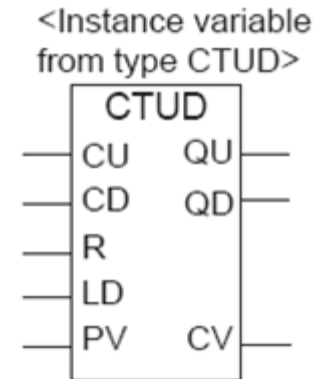
Up

CU = Count Up
 R = Reset
 CV = Counter Value
 Q = Status do contador ($CV \geq PV$)
 PV = Limite de contagem



Down

CD = Count Down
 LV = Reseta para valor inicial
 CV = Counter Value
 Q = Status do contador ($CV \leq 0$)
 PV = valor inicial de contagem

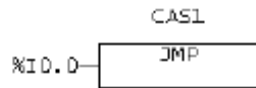


Up-Down

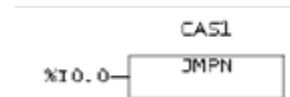
CU = Count Up
 CD = Count Down
 R = Reseta CV para 0
 LD = Reseta CV para PV
 CV = Counter Value
 QU = Status do contador Up ($CV \geq PV$)
 QD = Status do contador down ($CV \leq 0$)
 PV = Limite de contagem

Blocos FBD

- Salto (Jumps):



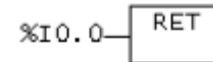
Jump para label
cas1 se entrada = 1



Jump para label
cas1 se entrada = 0



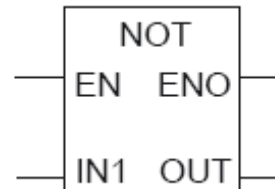
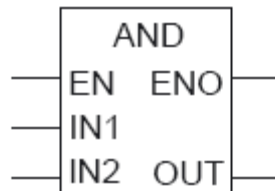
Label alvo
do jump



return

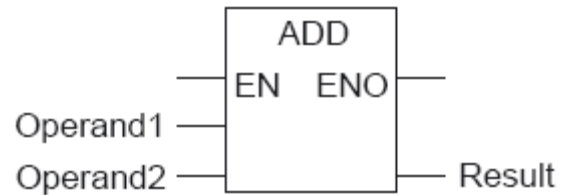
Blocos FBD

- Lógica não binária:
- AND, OR, XOR, NOT
- Exemplos:



Blocos FBD

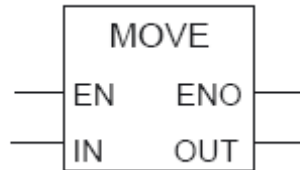
- Operações aritméticas:
- ADD, SUB, DIV, MUL, MOD
- Exemplo:



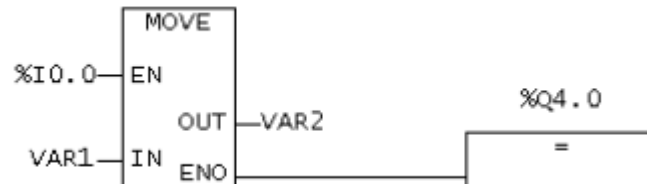
Blocos FBD

- Mover (copiar):

- Bloco básico:



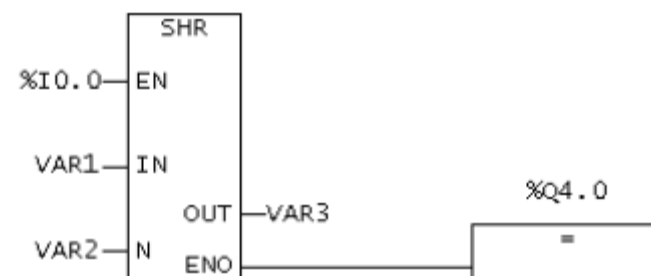
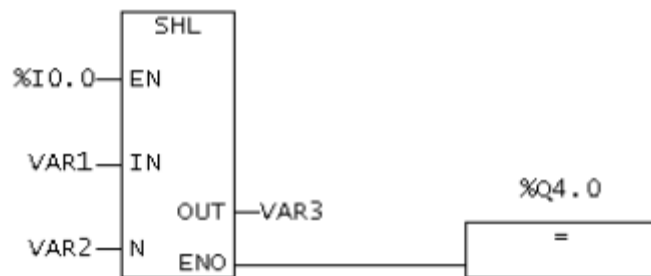
- Exemplo:



A operação é executada se %I 0.0 = 1.
O conteúdo de VAR1 é copiado em VAR2.
%Q 4.0 é 1 se a operação for executada.

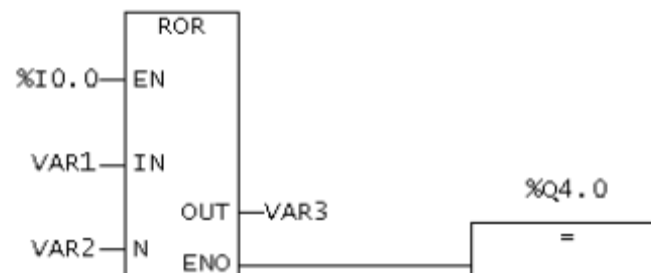
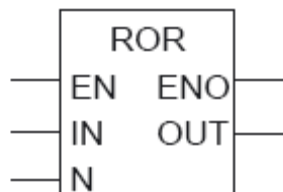
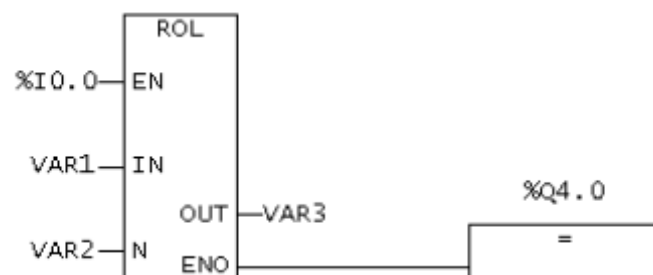
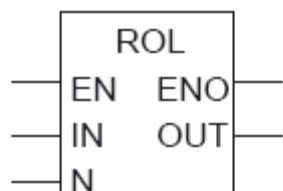
Blocos FBD

- Shifts (deslocamentos):
- N contém número de bits a deslocar de IN para OUT
- EN habilita entrada (enable)
- ENO habilita saída
- Exemplos (SHL e SHR):



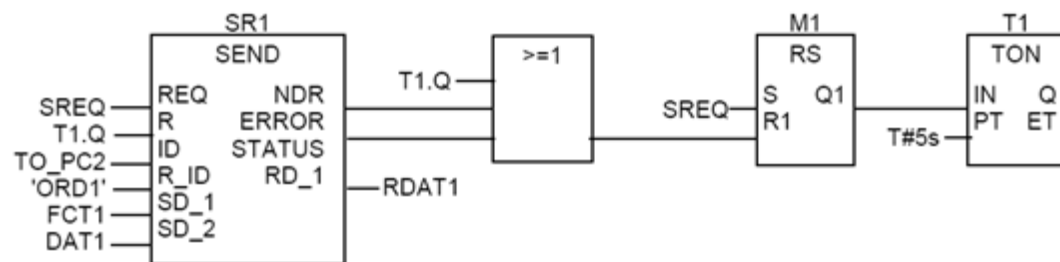
Blocos FBD

- Rotate (rotações):

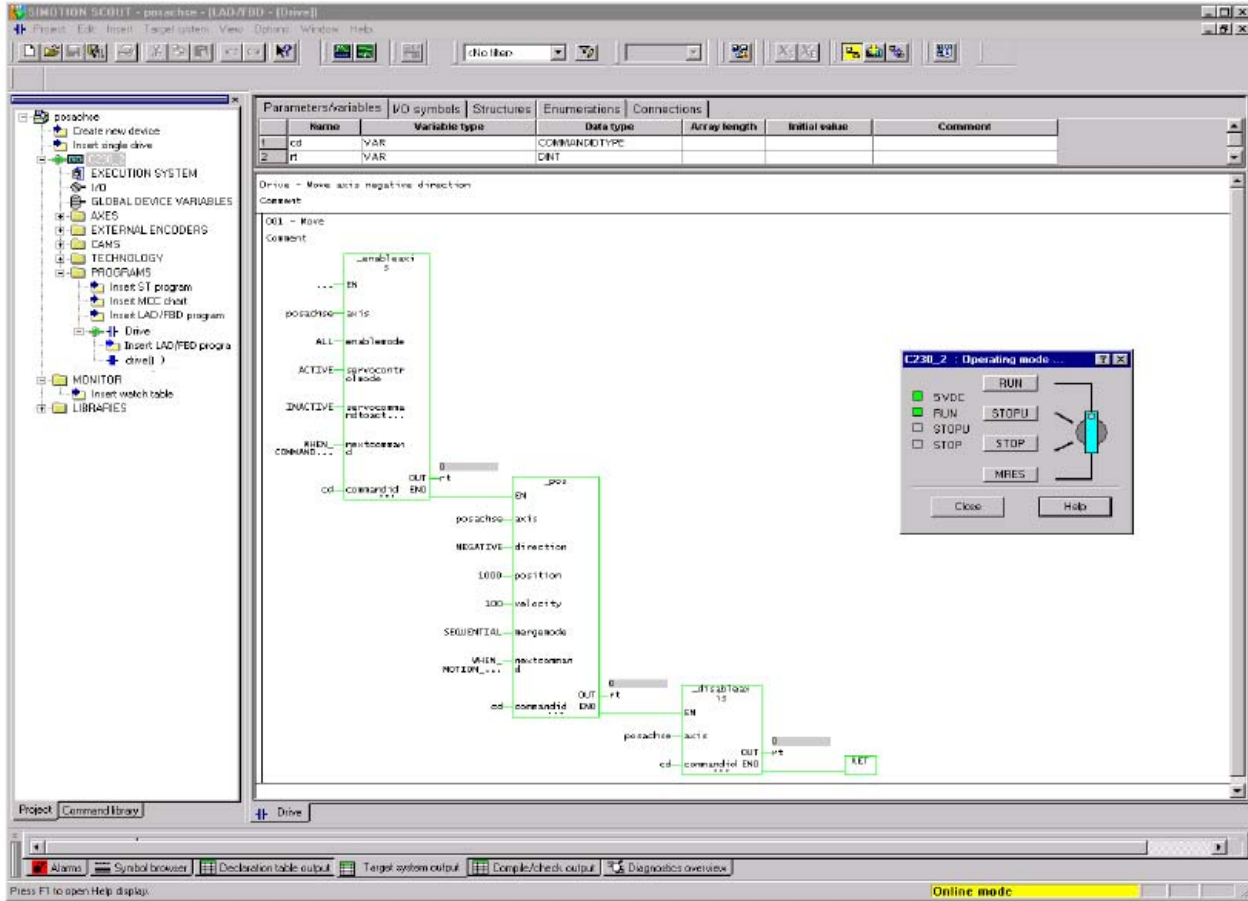


Blocos FBD

- Comunicação:

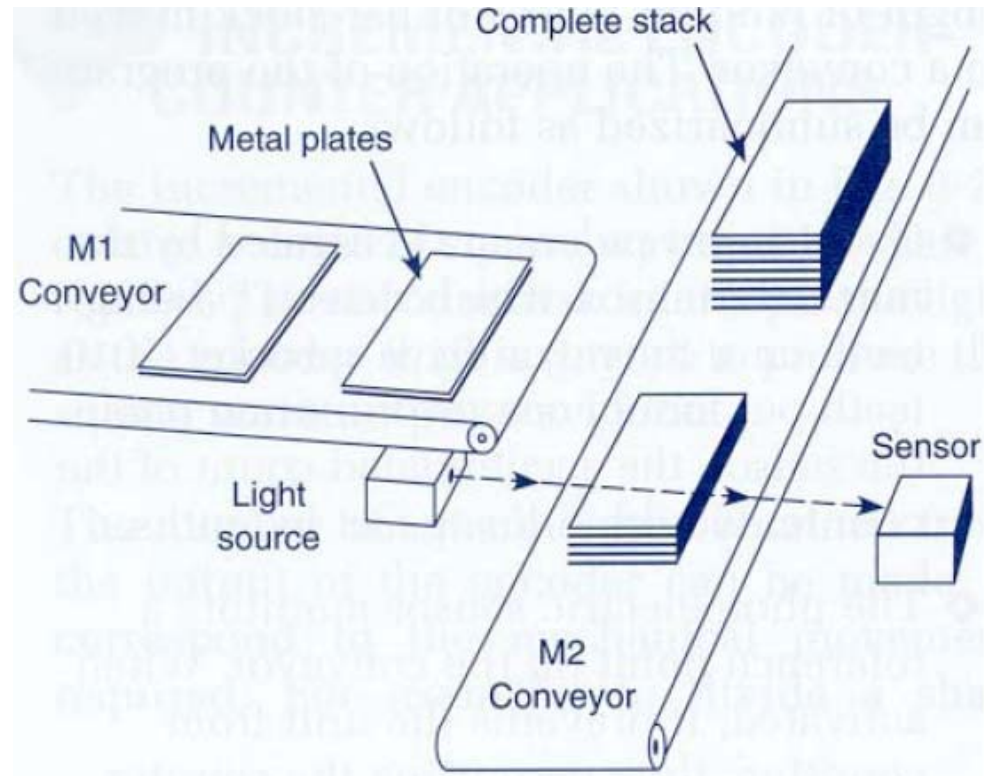


Editor FBD



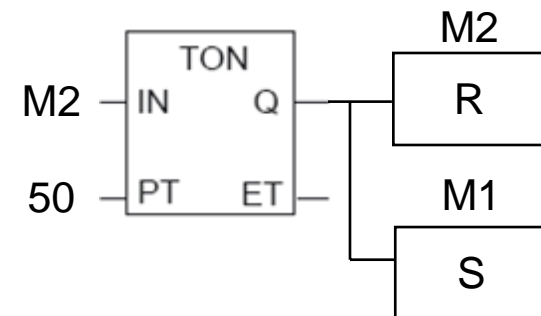
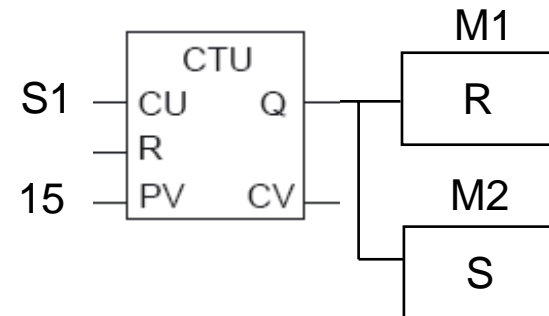
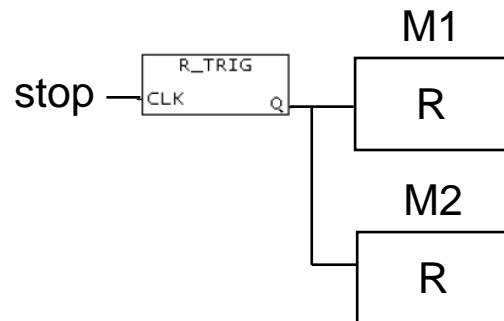
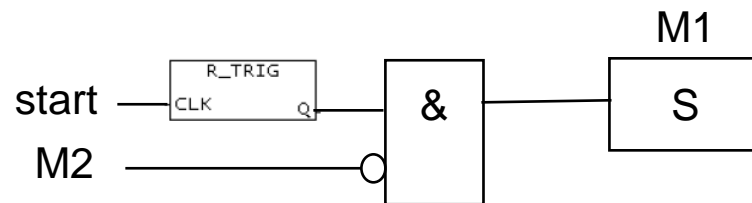
FBD - Exemplo

- Botão *start* inicia operação e *stop* interrompe
- Após botão *start*, M1 começa a funcionar
- Placas que caem são contadas por sensor de luz, que envia pulsos para entrada S1
- Após 15 placas, M1 pára e M2 começa a funcionar
- M2 pára após 5 segundos
- Seqüência reinicia



FBD - Exemplo

- Solução:



FBD - Conclusão

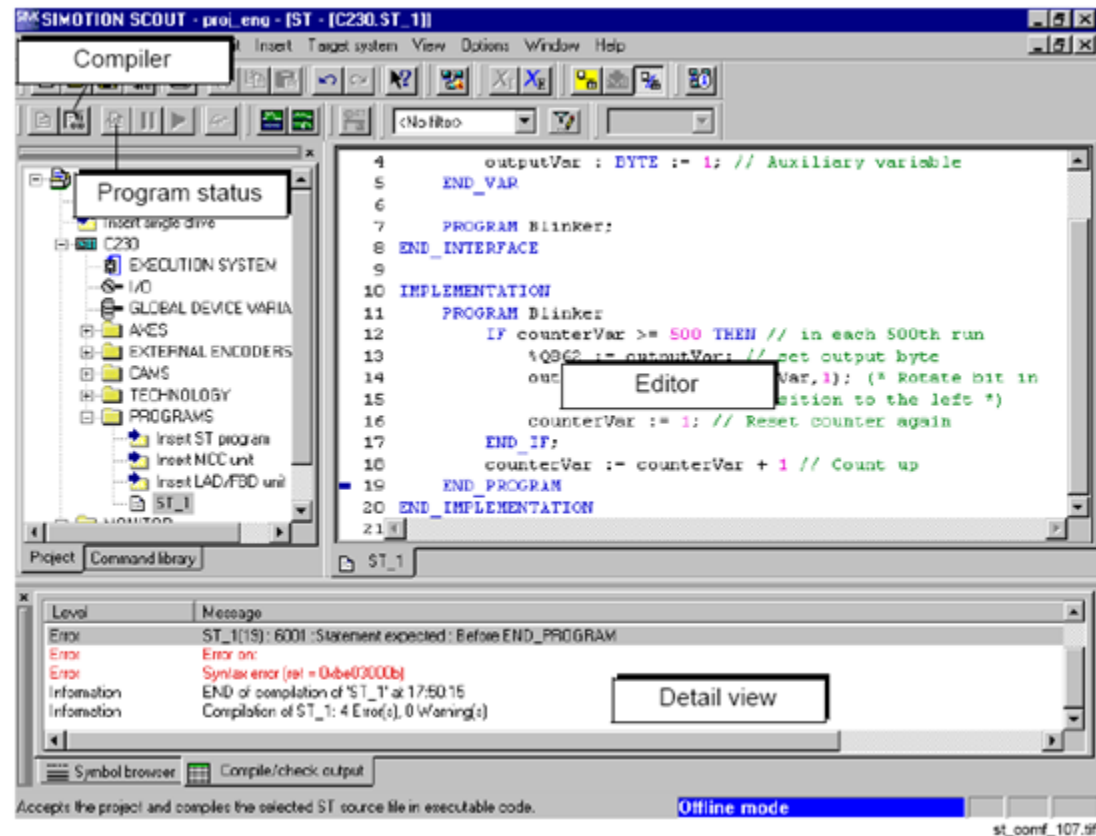
- FBD possui basicamente a mesma capacidade de expressão lógica de IL e LD
- Alguns fabricantes possuem ambientes de desenvolvimento capazes de converter automaticamente programas entre estas linguagens.
- FBD é mais recente e menos difundida do que IL e LD.

ST

- ST (Structured Text) é uma linguagem de alto nível baseada em PASCAL, ADA e C.
- ST e IL são as linguagens textuais (não gráficas) previstas na norma IEC 61131-3.
- ST requer um ambiente de desenvolvimento semelhante ao de outras linguagens de alto nível, como C/C++ e PASCAL.
- O ambiente inclui: editor, compilador, depurador e linker.

ST- Ambiente de Desenvolvimento

- Ex.: Simotion ST (Siemens)



ST - Sintaxe

- **Interface e implementação:**

```
INTERFACE
  VAR_GLOBAL
    counterVar : INT  := 1;  // Counter variable
    outputVar  : BYTE := 1;  // Auxiliary variable
  END_VAR

  PROGRAM Blinker;
END_INTERFACE

IMPLEMENTATION
  PROGRAM Blinker
    IF counterVar >= 500 THEN  // in every 500th pass
      %QB62 := outputVar;    // set output byte
      outputVar := ROL(outputVar,1); (* rotate bit in
                                     byte one digit to the left *)
      counterVar := 1;        // reset counter
    END_IF;
    counterVar := counterVar + 1; // increment counter
  END_PROGRAM
END_IMPLEMENTATION
```

ST - Sintaxe

- **Uso de comentários:**

```
// This is a one-line comment.  
a := 5;
```

```
// This is an example of a single-line  
// comment used several times in succession.  
b := 23;
```

```
(* The example above is easier to maintain  
   as a multiple-line comment.  
   *)  
c := 87;
```

ST - Sintaxe

- **Declaração de variáveis e arrays, inicialização:**

```
VAR
    // Declaration of a variable ...
    var1 : REAL := 100.0;
    // ... or if there are several variables of the same type:
    var2, var3, var4 : INT := 1;
    var5 : REAL := 3 / 2;
    var6 : INT := 5 * SHL(1, 4)
    myC1 : C1 := GREEN;
    array1 : ARRAY [0..4] OF INT := [1, 3, 8, 4, 0];
    array2 : ARRAY [0..5] OF DINT := [6 (7)];
    array3 : ARRAY [0..10] OF INT := [2 (2(3),3(1)),0];
        // corresponds to [2(3),3(1),2(3),3(1)),0]
        // Initialization, as follows:
        // Array elements 0, 1 with 3;
        // Array elements 2, 3, 4with 1;
        // Array elements 5, 6 with 3;
        // Array elements 7, 8, 9with 1;
        // Array element 10 with 0
    myAxis : PosAxis := TO#NIL;
END_VAR
```

ST - Sintaxe

- **IF-THEN-ELSE - ELSIF**

```
IF A=B THEN  
    n:= 0;  
END_IF;
```

```
IF temperature < 5.0 THEN  
    %Q0.0 := TRUE;  
ELSIF temperature > 10.0 THEN  
    %Q0.2 := TRUE;  
ELSE  
    %Q0.1 := TRUE;  
END_IF;
```

ST - Sintaxe

- **CASE:**

```
TW := BCD_TO_INT(THUMBWHEEL);  
TW_ERROR := 0;  
  
CASE TW OF  
  1,5:  DISPLAY := OVEN_TEMP;  
  
  2:    DISPLAY := MOTOR_SPEED;  
  3:    DISPLAY := GROSS - TARE;  
  4,6..10: DISPLAY := STATUS(TW - 4);  
ELSE   DISPLAY := 0 ;  
       TW_ERROR := 1;  
  
END_CASE;  
QW100 := INT_TO_BCD(DISPLAY);
```

ST - Sintaxe

- **FOR:**

```
J := 101 ;  
FOR I := 1 TO 100 BY 2 DO  
  IF WORDS[I] = 'KEY' THEN  
    J := I ;  
    EXIT ;  
  END_IF ;  
END FOR ;
```

- **WHILE:**

```
WHILE Index <= 50 DO  
  Index := Index + 2;  
END_WHILE;
```

ST - Sintaxe

- **REPEAT-UNTIL:**

```
Index:= 1;  
REPEAT  
    Index := Index + 2;  
UNTIL Index > 50  
END_REPEAT;
```

- **EXIT (em laços):**

```
SUM := 0 ;  
FOR I := 1 TO 3 DO  
    FOR J := 1 TO 2 DO  
        IF FLAG THEN EXIT ; END_IF  
        SUM := SUM + J ;  
    END_FOR ;  
    SUM := SUM + I ;  
END_FOR ;
```

ST - Sintaxe

- **Funções:**

```
FUNCTIONname : function_data_type
    // Declaration section
    // Statement section
END_FUNCTION
```

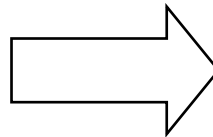
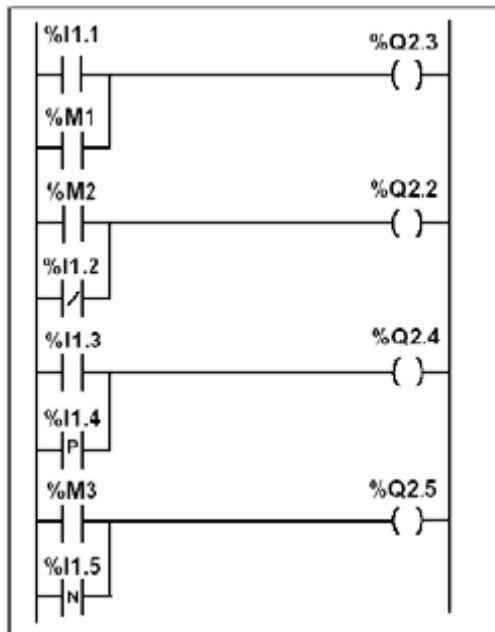
- **Espera temporizada:**

```
INTERFACE
    PROGRAM waitTime;
END_INTERFACE

IMPLEMENTATION
    PROGRAM waitTime
        VAR
            retVal :DINT;
        END_VAR;
        retVal := _waitTime(timeValue := T#10s);
    END_PROGRAM
END_IMPLEMENTATION
```


ST - Sintaxe

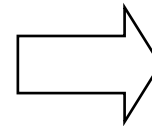
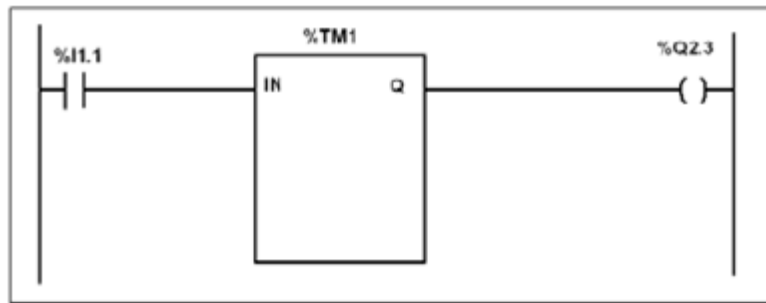
- Exemplos de programação básica:



```
%Q2.3:=%I1.1 OR %M1;  
%Q2.2:=%M2 OR (NOT%I1.2);  
%Q2.4:=%I1.3 OR (RE%I1.4);  
%Q2.5:=%M3 OR (FE%I1.5);
```

ST - Sintaxe

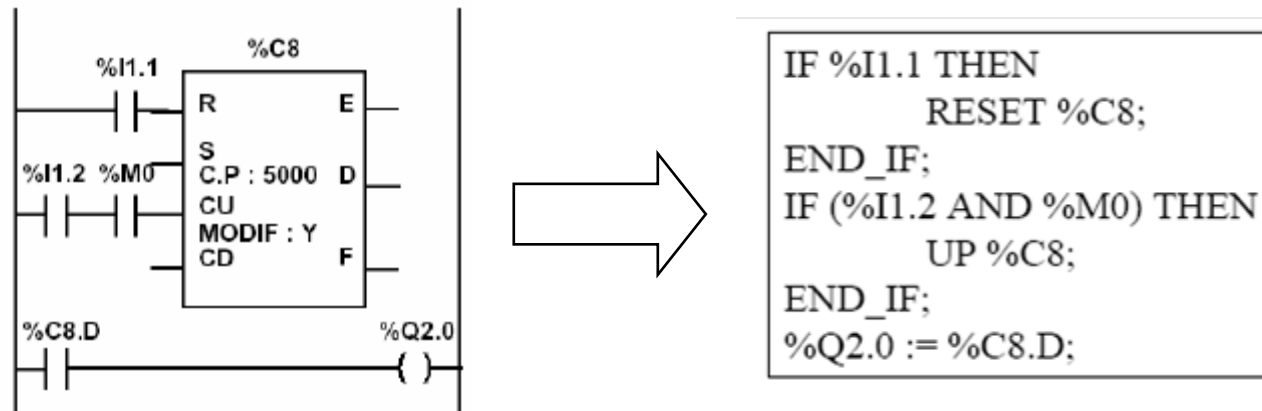
- **Uso de timers:**



```
IF %I1.1 THEN  
    START %TM1;  
END_IF;  
%Q2.3 := %TM1.Q
```

ST - Sintaxe

- **Uso de contadores:**



ST - Sintaxe

- **Processamento numérico, FE (Falling Edge), RE (Rising Edge):**

```
%Q2.2 := %MW50 > 10;  
IF %I1.0 THEN  
    %MW10 := %KW0 + 10;  
END_IF;  
IF FE %I1.2 THEN  
    INC %MW100;  
END_IF;
```

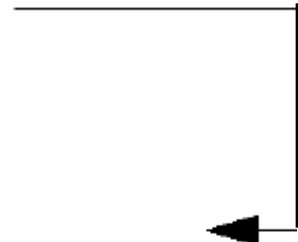
```
IF %M0 THEN  
    %MW0 := %MW10 + 100;  
END_IF;  
IF %I3.2 THEN  
    %MW0 := SQRT(%MW10);  
END_IF;  
IF RE %I3.3 THEN  
    INC %MW100;  
END_IF;
```

ST - Sintaxe

- **Jumps:**

```
IF %M8 THEN
    JUMP %L10;
END_IF;
%Q2.5:=%I1.0;

-----
%L10:
    %M5:=%M20;
    %Q2.1:=%I1.0 AND %I1.2;
```

A diagram illustrating a jump instruction. A horizontal line extends from the right side of the 'JUMP %L10;' instruction, then turns 90 degrees downward, and finally turns 90 degrees leftward to point at the label '%L10:'.

```
graph LR
    JUMP["JUMP %L10;"] --> L10["%L10:"]
```

ST - Conclusão

- ST ainda é pouco difundida no Brasil.
- Uso deverá crescer.
- Tem todas as vantagens das linguagens de alto nível.

Padronização de CLPs

- Diversas entidades internacionais preocupam-se com a criação e o aperfeiçoamento de normas e padrões para CLPs.
- Normas importantes:
 - NEMA ICS 1-109 "tests and test procedures": define uma série de testes aplicáveis no projeto, produção e aplicação de CLPs;
 - NEMA ICS 2-230 "Components for solid-state logic systems": descreve testes e equipamentos para verificação de imunidade a ruído;
 - IEEE std 518-1977 "Guide for installation of electrical equipment to minimize electrical noise inputs to controllers from external sources": ruídos, suas fontes e métodos de redução;

Padronização de CLPs

- NEMA ICS 3-304 e IEC 61131 (partes 1 até 5):
 - Normas mais completas sobre CLPs.
 - Abordam confiabilidade, redundância, recuperação de falhas, sistemas de comunicação, etc;
 - Enumeram estruturas, tipos de programação e subsistemas;
 - Estabelecem parâmetros máximos e mínimos de operação;
 - Descrevem testes padronizados para caracterizar e verificar as especificações dos fabricantes.

CLPs em Rede

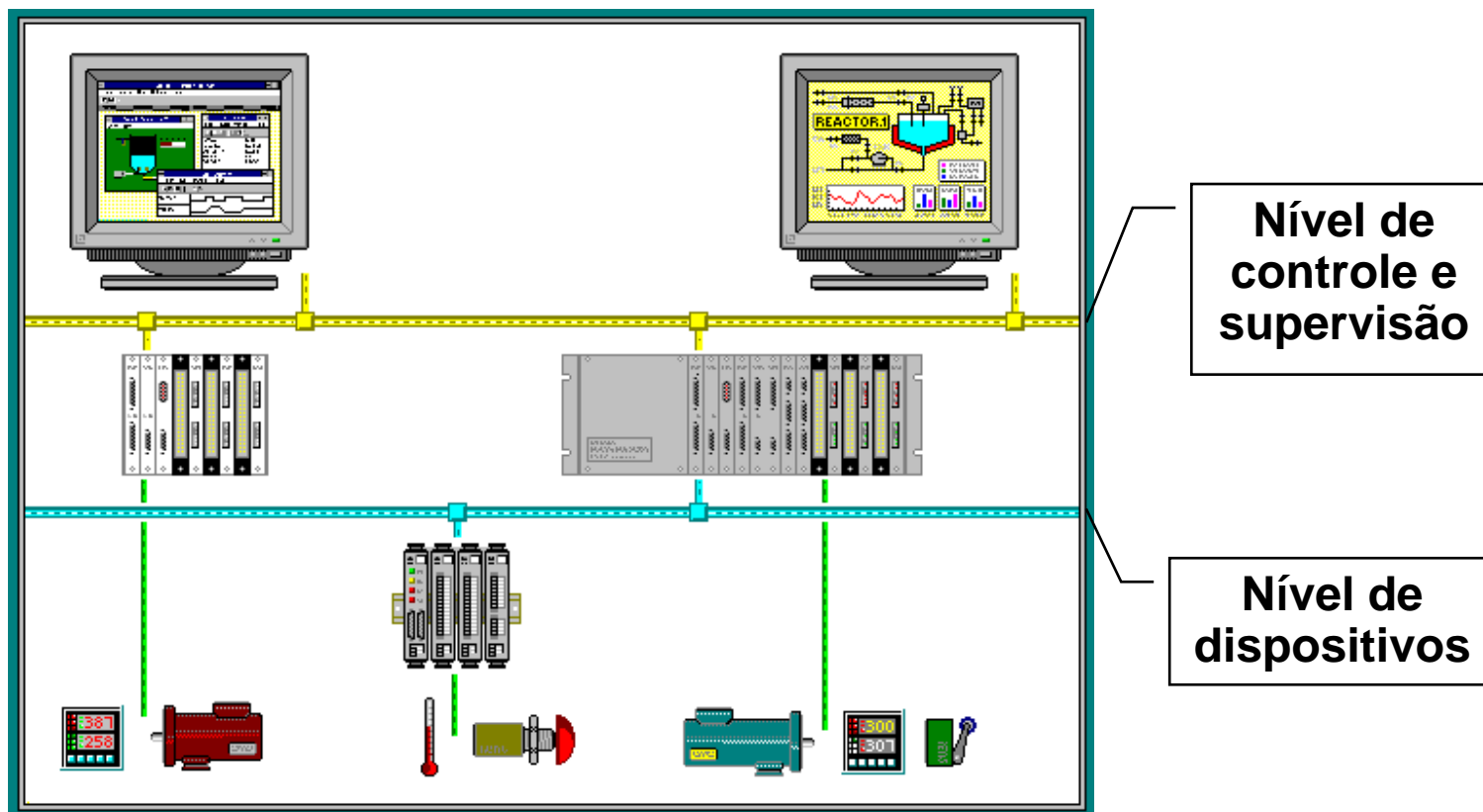
- Modelo CIM (Computer Integrated Manufacturing) requer integração de todos os níveis hierárquicos de automação.
- Hoje todos os fabricantes de componentes para automação (CNC, CLP, PC, RC, Sensores e Atuadores, etc.) fornecem interfaces para rede.
- A comunicação entre CLPs e outros equipamentos requer padronização de interfaces e protocolos;

CLPs em Rede

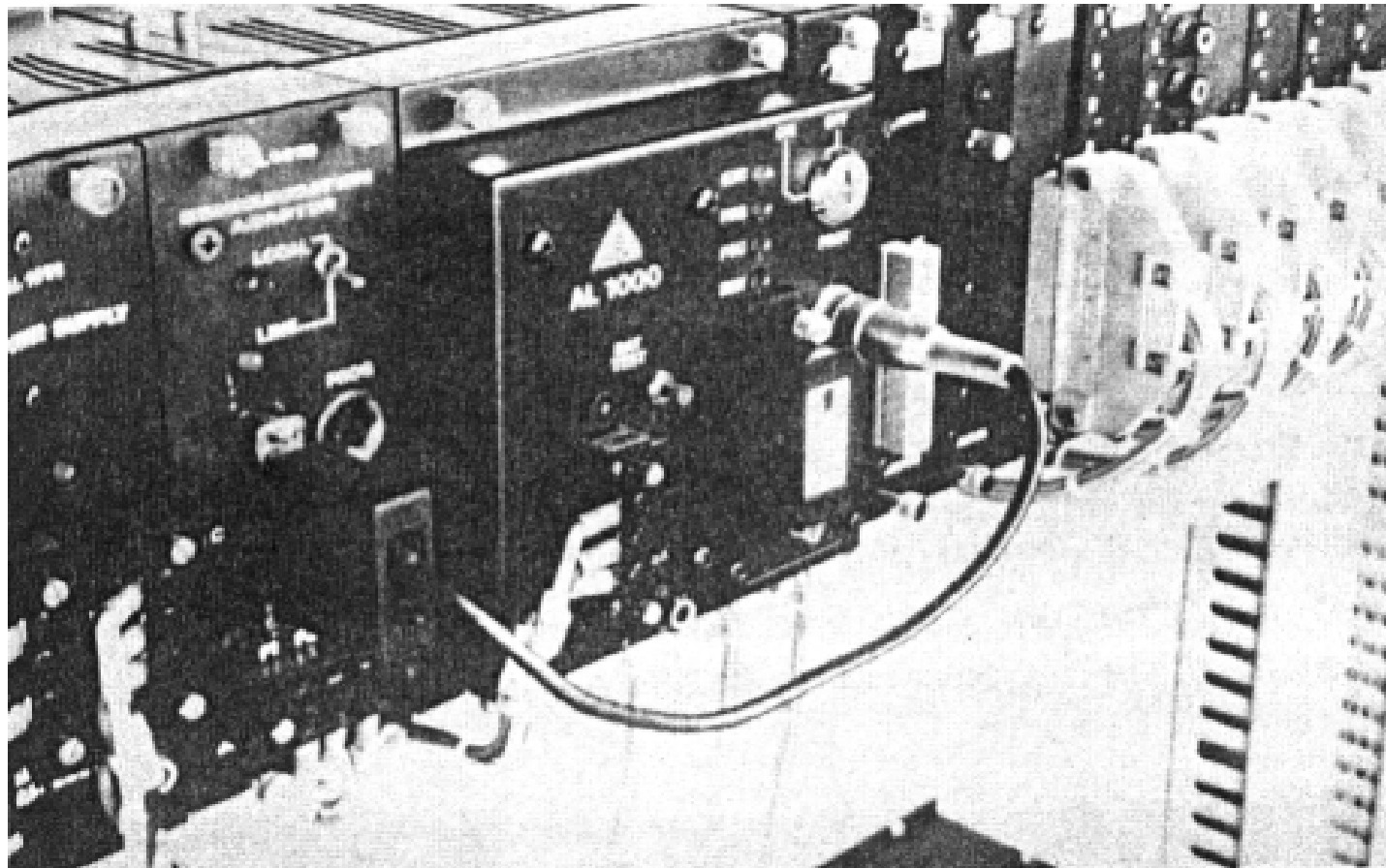
- Muitos fabricantes oferecem redes "proprietárias" para esta finalidade.
- Tendência é seguir padrões abertos para redes de chão de fábrica:
 - FIP;
 - PROFIBUS;
 - FF.
- Disponíveis também:
 - Ethernet, Interbus-s, CAN, DeviceNet, MAP, MAP/EPA, Mini-MAP, Token-bus, Token-ring, LON, SERCOS, ASI-Bus, Modbus, etc.
- Interfaces de comunicação para CLP padronizadas pela norma IEC 61131-5.

CLPs em Rede

→ Aplicação: SDCD (Sistema Digital de Controle Distribuído)



Módulo de conexão em rede para CLP



Fabricantes no Brasil

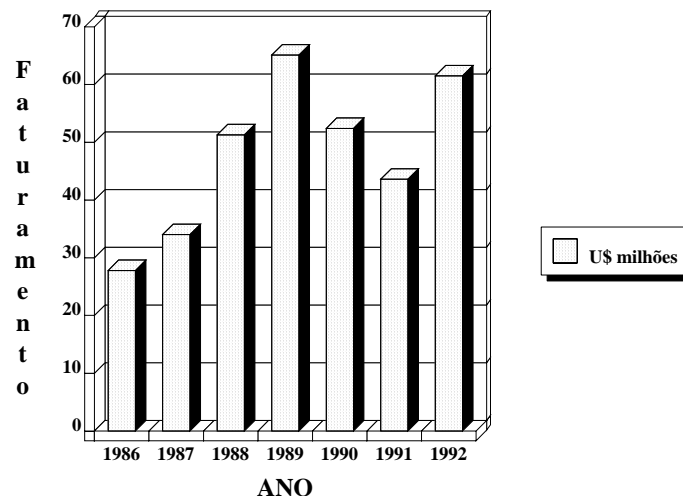
- No Brasil, mais de 20 empresas estão produzindo CLPs, com as mais variadas especificações.
- Enquanto alguns fabricantes desenvolveram tecnologia própria, grande parte deles adota tecnologia importada.

Exemplos de fabricantes

Fabricante	Modelo (exemplos)	E	S	Max E+S	Mem.	Proc.
ATOS	MPC-504	20	12	32		8040
	MPC-514	32	16	48		8040
	MPC-264	16	16	128		8035
ALTUS (Bosch)	AL-1000	32	32	256	16K	8031
	AL-600	16	8	256		80C32
BCM						
Cambridge	CAMCON-80	26	20	256		
	CAMCON-100					
Chronos	CPC-1000					
Contrap	CA-1200	16	16	220	32K	Z80-2
CGR	MPI-256			256	8K	
	MPI-4096			4096	16K	
Digicon	D-20	12	8	40		
	CP/DIG-80	256	256	512	2K	8085
	CP/DIG-80-10	256	256	512	4K	8085
Engeleto	1484	4	4	512	8K	
FANUC (GE)	PLC90-30 modelo 311	320	320		3K	80188
	PLC90-30 modelo 331	512	512		8K	80188
Geotron	Geotron CLP					
Herge	HSC-1000			512	16K	8085
Italtolt	MPC-85			512	14K	8085
	MPC-130			128	3K	8748
Maxitec (Siemens)	MXT-110	8	8	256	4K	
	MXT-130	512	512	3072	24K	
Metal Leve (Allen-Bradley)	CLP2mini			128	1K	
	CLP2/20			896	8K	
Pulse	CLP-16	8	8	16	4K	
	CLP-40	8	8	40	4K	
	CLP-80	8	8	80	4K	
Sinomatic (Klöckner-Möller)	PS-22	8	8	288	4K	
SCI	CP-80	8	8	256	12K	8080A
Villares	Vilologic-500	256	256	512	1K	
WEG (AEG)	CPW-A500			4096		8086
	CPW-A200	12	12	336	8K	
	CPW-A100	12	12	336	8K	

O Mercado de CLP no Brasil

- Fonte: SOBRACON
- Volume de vendas:
 - U\$ 27,8 milhões em 1986
 - U\$ 61,5 milhões em 1992
- Número de unidades vendidas:
 - 3.324 em 1990
 - 5.579 em 1992



O Mercado de CLP no Brasil

- Taxas de Uso:
 - 69,14% das empresas usuárias de componentes para automação industrial utilizam CLPs;
 - 44,44% dos CLPs instalados são de pequeno porte;
 - 28,40% são de médio porte;
 - 19,75% são de grande porte.

O Mercado de CLP no Brasil

- 42,67% dos CLPs instalados são usados na área de controle de processos;
- 40,00% na área de manufatura;
- 4,00% são usados em ensino;
- 2,68% na integração de sistemas de automatização industrial.
- Os restantes são aplicados em áreas diversas como:
 - controle de demanda;
 - laboratórios de teste;
 - linhas de fabricação de máquinas;
 - automação predial;
 - controle de subestações de energia.
- Origem: 48,70% dos CLPs usados são de fabricação nacional.

Considerações Finais

- CLP usados em larga escala em automação.
- CLP tem ainda que enfrentar concorrentes:
 - IC (Industrial Computer, versão industrial do PC):
 - » Ainda muito caros;
 - » Usados quando processo a controlar requer cálculos complexos.
 - Sensores e atuadores inteligentes:
 - » Atualmente usados junto com um CLP (via rede);
 - » Tendência de se tornar cada vez mais autônomos (dispensar CLP?).



Doich MegaPAC Rugged Multi-slot Portable PC



Considerações Finais

- PAC (Programmable Automation Controller):
 - Nova geração (ou sucessor) do PLC a partir de 2002
 - Uso em 20% das aplicações mais complexas
 - Possui arquitetura modular aberta
 - Funcionalidade mista PLC/PC
 - Suporte Multitarefa Tempo Real (ex.: VxWorks)
 - Interoperabilidade
 - Integração em rede mais fácil
 - Interfaces Ethernet e USB
 - Suporta TCP/IP, XML, OPC, SQL, etc.
 - Memória não volátil tipo Flash
 - Processador de Ponto Flutuante
 - Programação preferencial em alto nível:
 - » Abordagem baseada em PLC: ST, GRAFCET
 - » Abordagem baseada em PC: C/C++, LabVIEW RT

