Capítulo 4

Linguagem de programação

No capítulo 1, foram apresentadas as linguagens de programação de um enfoque histórico. Agora serão discutidos alguns detalhes adicionais sobre as linguagens mais empregadas e será ampliada a discussão sobre a tentativa de padronização das linguagens pela norma IEC. Por sua importância e popularidade, a linguagem Ladder terá o capítulo 5 inteiramente dedicado a ela.

4.1 Norma IEC 61131

Em 1992, a International Electrotechnical Commission (IEC – Comissão Internacional de Eletrotécnica) publicou a primeira edição da norma IEC 61131, com o objetivo de estabelecer padrões para os controladores programáveis. Essa norma foi dividida em vários capítulos, possibilitando a definição de critérios para cada um dos tópicos relacionados aos CLPs.

Os capítulos são os seguintes:

- 61131-1 – Informações gerais sobre os CLPs.
- 61131-2 – Requisitos de *hardware*.
- 61131-3 – Linguagens de programação.
- 61131-4 – Guia de orientação ao usuário.
- 61131-5 – Comunicação.

São abordados aqui os padrões de linguagens de programação de CLPs apresentados na IEC 61131-3. Vale ressaltar que, mesmo que se trate de um capítulo à parte, a IEC 61131-3 especifica as semânticas e sintaxes para as linguagens de programação dos controladores definidos na IEC 61131-1 no que se refere aos conceitos gerais. A IEC 61131-5 trata da comunicação de dados internos, uma vez que esses tópicos estão indiretamente interligados.

A primeira edição da IEC 61131-3 foi publicada em dezembro de 1993, e a segunda, em 2003.

Essa norma define que, para um CLP estar de acordo com os padrões por ela estabelecidos, deve possuir ao menos duas linguagens gráficas e duas linguagens de texto para sua programação. Assim, é consenso entre os atuais fabricantes trabalhar com as seguintes linguagens:

- Textuais:
  IL – *instruction list* ou lista de instruções.
  ST – *structured text* ou texto estruturado.

- Gráficas:
  LD – *Ladder diagram* ou diagrama Ladder.
  FBD – *function block diagram* ou diagrama de blocos de funções.

Um quinto modo de programar alguns equipamentos de mercado é o SFC – *sequential function chart* ou sequenciamento gráfico de funções, que possui elementos para organizar programas de maneira sequencial e permite também o controle paralelo de processos.

Entre essas linguagens de programação, alguns parâmetros são definidos para que realmente haja certa compatibilidade entre os equipamentos. Tais parâmetros podem ser definidos como:

a) **Dados** – A norma prevê os seguintes tipos de dados:

- Grupo de bits: grupo de valores binários (*on*/*off*).
  - BOOL: 1 bit.
  - BYTE: 8 bits.
  - WORD: 16 bits.
  - DWORD: 32 bits.
  - LWORD: 64 bits.

- Inteiros: números inteiros e reais.
  - SINT: curto (1 byte).
  - INT: inteiro (2 bytes).
  - DINT: duplo inteiro (4 bytes).
  - LINT: longo inteiro (8 bytes).
  - U: não definido (adicionar um caractere U para o tipo de inteiro a ser utilizado).

- Real: ponto flutuante, ou seja, considera fracionários na utilização do número conforme a IEC 559 (1982).
  - REAL: 4 bytes.
  - LREAL: 8 bytes.

- Tempo: duração de *timers* e processos.
- Data e hora do dia.
  - DATE: data do calendário.
  - TIME_OF_DAY: hora local.
  - DATE_AND_TIME: data e hora local.

- *String*: caracteres que podem ser expostos em cotas únicas – normalmente para a transmissão de caracteres ASCII (*american standard code for information interchange*) para outros dispositivos.

    - WSTRING: permitir o envio de vários *strings*.
    - ARRAYS: múltiplos valores armazenados na mesma variável.
    - SUB RANGES: definir limites de valores para a entrada ou para a saída de dados – por exemplo, sinais de 4 a 20 mA.

b) **Variáveis** – Podem ser do tipo:

- Global: serve a todo o programa e não necessariamente só a uma parte dele.
- Local: serve somente a uma parte do programa.
- *I/O mapping*: mapeamento de todas as entradas e saídas em relação a posições de memória predefinidas.
- *External*: mapeamento definido exclusivamente como pontos de entrada e saída de dados.
- *Temporary*: usados momentaneamente durante a execução de parte do programa.

c) **Configuração** – Recursos de *hardware* e vínculos específicos para o processamento dos dados e dos programas.

- Recursos: reserva de memória ou índices de processamento para determinada parte do programa.
- Tarefas: podem seguir paralelas, sendo executadas simultaneamente pela UCP.
- Programas: podem ser executados ciclicamente, a cada determinado período ou quando ocorrer certo evento.

d) **Organização das unidades de programas** – Definidos pelas funções básicas, blocos básicos e possibilidade de criação de funções e blocos de acordo com a necessidade da programação.

- Funções-padrão como: ADD, SQRT, SIN, COS, GT, MIN, MAX, AND, OR, entre outras.
- Funções customizadas: campo no qual o programador pode criar funções ou utilizar mais de uma função preexistente para a criação de outra função em sua programação.
- Blocos de funções: padrões iguais aos apresentados nas funções, só que em linguagem de bloco.
- Blocos customizados: campo para a elaboração ou utilização de mais de uma função preexistente na criação de blocos. Podem ser compostos também por blocos comercializados por outros fabricantes ou empresas especializadas.
- Programas: programas e sub-rotinas específicos. Podem ser armazenados em funções ou blocos criados pelo programador e utilizados mais de uma vez na atual aplicação ou posteriormente em outros programas.

e) ***Links* externos** – Abrem espaço para o capítulo da IEC 61131-5, que trata especificamente dos formatos de comunicação e das facilidades que o padrão determina como necessários para a compatibilidade com a norma.

Vale ressaltar que essas normas possibilitaram que fabricantes de outros tipos de *hardware* compatibilizassem seus produtos. Hoje existem fabricantes de SoftPLCs, que nada mais são do que computadores que podem ser programados para atender a controles lógicos, obedecendo às normas definidas pela IEC. Trata-se de tecnologia muito recente, que ainda está sendo avaliada e certificada, porém, já é utilizada em alguns processos de automação de máquinas de pequeno porte.

Uma vez apresentadas as informações sobre os parâmetros que a IEC 61131-3 determina que um fabricante de *hardware* e *software* deve disponibilizar aos usuários, vamos ver agora as linguagens de programação definidas pela norma.

4.2 IL – lista de instruções

É basicamente a transcrição do diagrama de relés (Ladder), ou seja, a passagem de uma linguagem gráfica para uma linguagem escrita. Essa etapa foi importante nos primórdios do CLP, pois não existiam terminais gráficos como conhecemos atualmente, que permitem desenhar o diagrama Ladder na tela, usando o *mouse*. Antigamente os terminais de vídeo e os *displays* dos terminais de programação eram alfanuméricos; por isso, o programador precisava projetar o diagrama Ladder no papel e depois convertê-lo para a linguagem IL. Um compilador se encarregava de traduzir o IL para a linguagem de máquina (Assembler) do processador utilizado no CLP.

A tabela 4.1 apresenta as instruções mais comuns empregadas nessa linguagem.

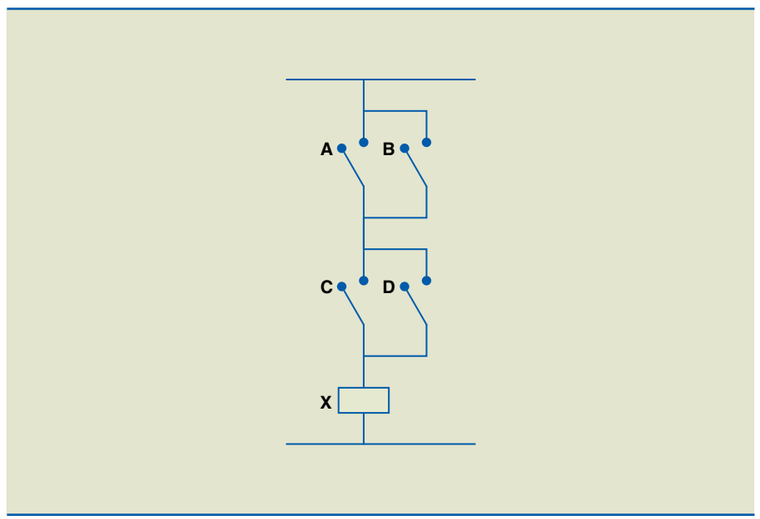
Tabela 4.1
Lista de comandos na linguagem IL contidas na IEC 61131-3

Operador	Modificador	Tipo de dados	Descrição
LD	N	Diversos	Carrega valor do resultado
ST	N	Diversos	Armazena o resultado no local definido
S, R		BOOL	Seta ou resseta um valor (*lach* ou *flip-flop*)
AND, &	N, (	BOOL	Lógica booleana AND
OR	N, (	BOOL	Lógica booleana OR
XOR	N, (	BOOL	Lógica booleana OR EXCLUSIVE
ADD	(	Diversos	Soma matemática
SUB	(	Diversos	Subtração matemática
MUL	(	Diversos	Multiplicação matemática
DIV	(	Diversos	Divisão matemática
GT	(	Diversos	Compara maior que (>)

Operador	Modificador	Tipo de dados	Descrição
GE	(	Diversos	Compara maior ou igual a (≥)
EQ	(	Diversos	Compara igual a (=)
NE	(	Diversos	Compara diferente(<>)
LE	(	Diversos	Compara menor ou igual (≤)
LT	(	Diversos	Compara menor que (<)
JMP	C, N	LABEL	Salta para a linha de programa
CAL	C, N	NAME	Chama sub-rotina de acordo com nome
RET	C, N		Retorna da sub-rotina chamada
)			Conclui valor chamado

Dado o diagrama de comandos elétricos da figura 4.1, que já está na linguagem Ladder (exceto pela simbologia), vamos transcrevê-lo para a linguagem IL.

Figura 4.1
Exemplo de diagrama de comandos elétricos.



Analisando a figura, fazemos a seguinte leitura: havendo continuidade elétrica de A ou B e também de C ou D, X deve ser acionado.

Podemos traduzir essa lógica em lista de instruções segmentando o problema:

LD A – Carrega o valor de A em um acumulador.
LD B – Carrega o valor de B em um acumulador.

OR B – Executa a lógica booleana OR entre A e B e armazena o resultado em B.
LD C – Carrega o valor de C em um acumulador.
LD D – Carrega o valor de D em um acumulador.
OR D – Executa a lógica booleana OR entre C e D e armazena o resultado em D.
AND B – Executa a lógica booleana AND entre B e D (últimos acumuladores gravados) e armazena o resultado em B.
ST X – Armazena o valor de B em X.

Percebe-se que tal lógica utiliza uma única instrução por linha de programação, o que dificulta a elaboração de grandes programas. Desse modo, faz-se necessário um controle muito eficiente na utilização dos registradores e respectivas interfaces de entrada e saída para não haver falhas durante a confecção do programa.

4.3 ST – texto estruturado

É uma linguagem mais elaborada, considerada de alto nível, que usa o princípio de criação de sentenças para definir e informar ao CLP qual a lógica necessária em determinado ponto. Como possibilita a utilização de mais de uma instrução por linha, agiliza e facilita a tarefa dos programadores em projetos mais complexos.

Com estrutura similar à de linguagens de programação, como o C++ e o Pascal, permite o uso de comandos específicos para a definição de laços de controle, ou seja, funções ou operações lógicas que devem ser executadas até que determinado evento ocorra ou que determinada contagem seja atingida (funções REPEAT-UNTIL, DO-WHILE, entre outras).

Possibilita a utilização de instruções condicionais, referindo-se a reações preestabelecidas do programa para o caso de certos eventos ocorrerem, desde que previamente considerados (funções IF-THEM-ELSE, CASE), e também, por ser uma linguagem mais rica, o emprego de equações trigonométricas (SIN – função seno) e matemáticas (SQRT – raiz quadrada). Mesmo sendo uma linguagem mais fácil de ser compreendida e escrita, ainda demanda mão de obra especializada para a confecção e manutenção de programas.

Levando em conta o exemplo apresentado na figura 4.1, podemos definir a lógica de programação em linguagem estruturada da seguinte forma:

X: = (A OR B) AND (C OR D)

Ou seja, X é o resultado da operação booleana AND de dois resultados distintos: lógica OR entre A e B e lógica OR entre C e D.

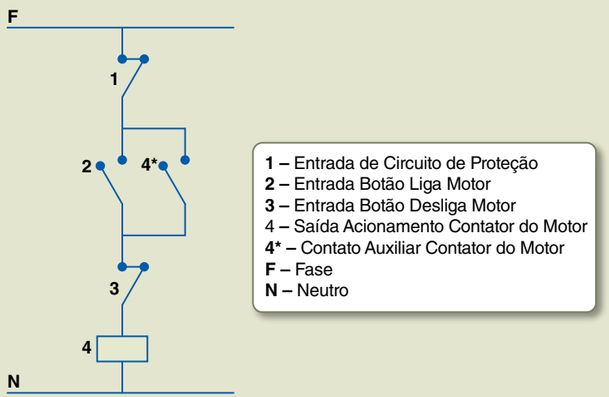
4.4 FBD – diagrama de blocos funcionais

Utilizada na programação de CLPs, é uma linguagem gráfica baseada na interligação de blocos funcionais previamente disponibilizados pelos fabricantes ou que permite ao próprio programador construir os blocos a serem utilizados. As entradas e saídas são conectadas a esses blocos criando malha de interco-

nexões que possibilita a obtenção dos mesmos resultados de outras lógicas de programação.

Uma das grandes vantagens dos blocos funcionais é a reutilização de blocos dentro de um programa. Suponha que um projeto use vários motores, todos com o mesmo princípio de funcionamento, conforme lógica predefinida de acionamento de um motor em partida direta (figura 4.2). Uma vez construído o bloco funcional de partida do motor, ele poderá ser utilizado várias vezes no programa, adotando entradas e saídas distintas, que, por sua vez, controlarão motores distintos.

Figura 4.2
Diagrama de comandos elétricos da ligação de um motor.



A figura 4.3 mostra como configurar esse bloco de acionamento de motor e a figura 4.4 exemplifica a utilização de um mesmo bloco em mais de um motor, considerando essa distinção de variáveis.

Figura 4.3
Montagem de um bloco de nome "BLOCO MOTOR" baseado em blocos primários AND e OR.

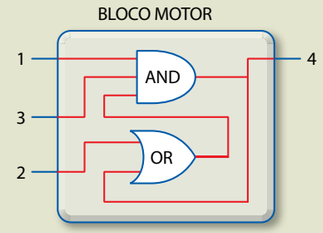
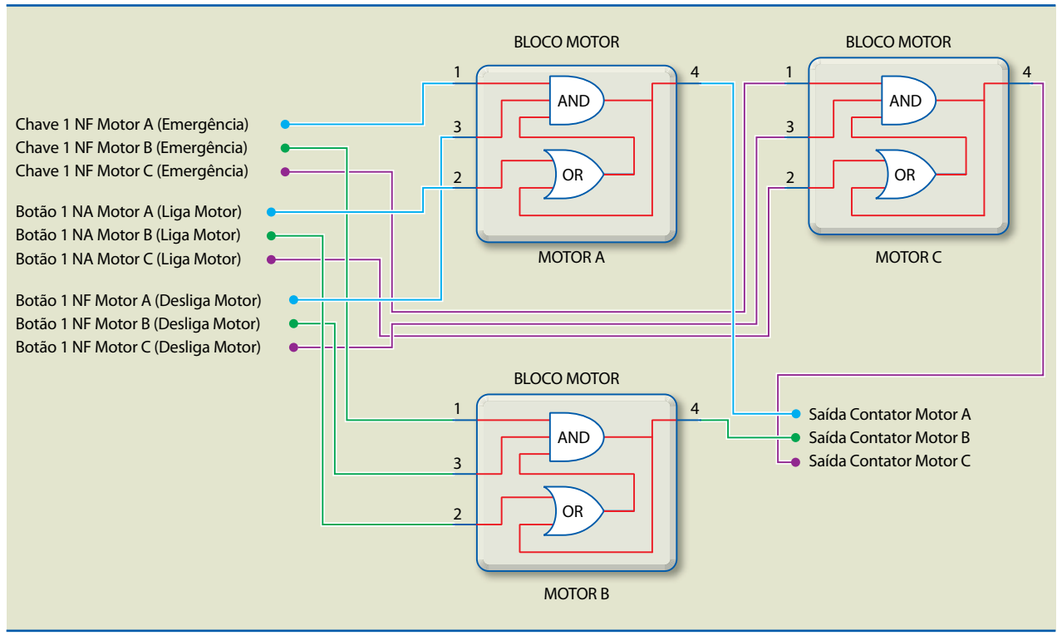


Figura 4.4
Utilização de vários "BLOCO MOTOR" dentro de um programa em diagrama de blocos funcionais.



Para fixar o conceito de programação em diagrama de blocos funcionais, são apresentados nas figuras 4.5 e 4.6 outros exemplos de programas de fabricantes distintos.

Figura 4.5
Exemplo de programa em diagrama de blocos funcionais.

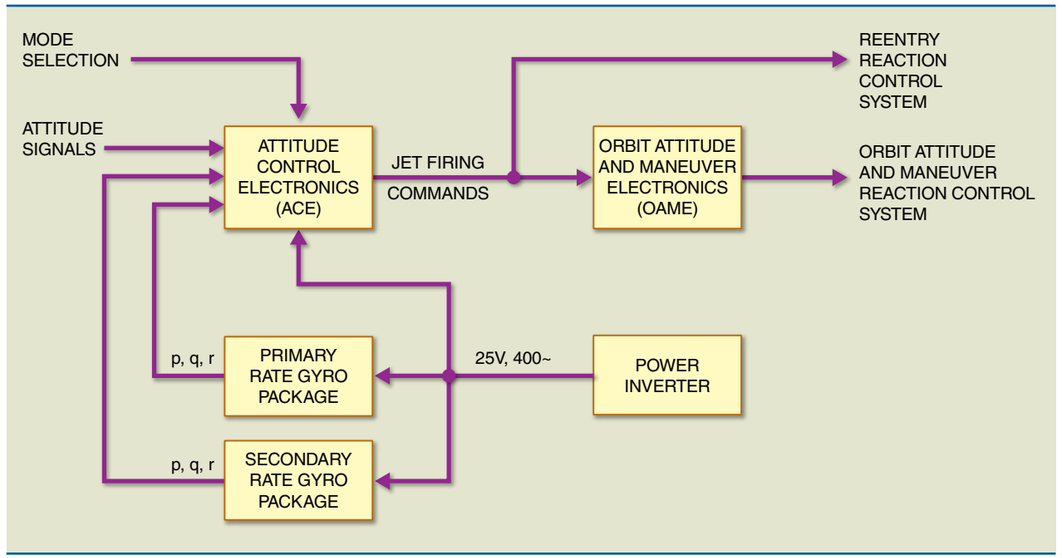
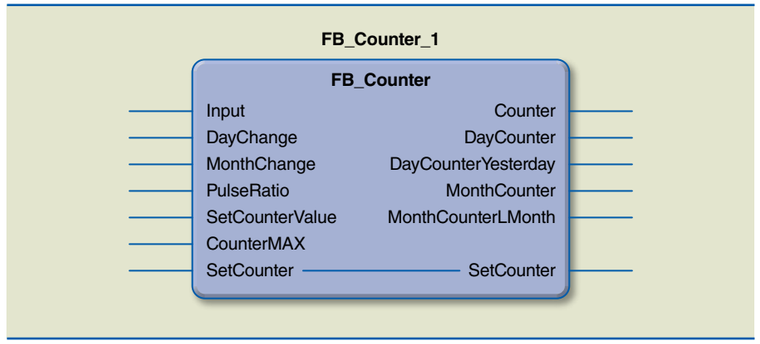


Figura 4.6
Exemplo de bloco de função criado por programador.



4.5 LD – diagrama Ladder

A grande maioria das aplicações atuais em CLPs adota a programação Ladder, assunto que será tratado mais detalhadamente no capítulo 5 deste livro.

4.6 SFC – sequenciamento gráfico de funções

Também é uma linguagem gráfica de programação muito poderosa. Proporciona uma representação das sequências do processo controlado na forma de um diagrama. O SFC é utilizado para dividir um problema de controle, permitindo uma visão geral do processo e facilitando o diagnóstico. Outra grande vantagem é o suporte para sequências alternativas e paralelas, tornando possível que sub-rotinas que servem ao interesse do processo controlado sejam executadas de maneira paralela, sem a necessidade de parada da lógica principal de controle. Em resumo, o SFC vai além de uma programação gráfica usada em CLPs: é uma forma de estruturar a lógica e a sequência de eventos desejadas em um processo a ser automatizado.

O SFC é elaborado com blocos funcionais dispostos como um fluxograma, possibilitando a confecção e o estudo dos processos por meio de ações e transições que devem ocorrer. Isso permite que um processo seja aberto ao menor nível de análise até que se tenha o modelo desejado mapeado em detalhes.

Também conhecido como GRAFCET, o SFC é baseado no conceito de análise binária das redes de Petri, levando em consideração, para ações futuras, os atuais estados de variáveis monitoradas. De forma prática, podemos analisar o funcionamento de uma lógica produzida em SFC observando a figura 4.7.

Na lógica apresentada na figura 4.7, os retângulos representam os passos a serem executados no controle do processo e, entre alguns retângulos, a condição necessária para que se chegue ao novo passo. Desse modo, podemos garantir que determinado passo nunca ocorra sem que uma transição esteja concluída.

Analisando o exemplo da figura 4.7, para que o passo 1 seja concluído, é necessário que sua resposta seja positiva, ou seja, o tanque está cheio. Enquanto o tanque estiver vazio, ele permanecerá monitorando essa etapa do processo.

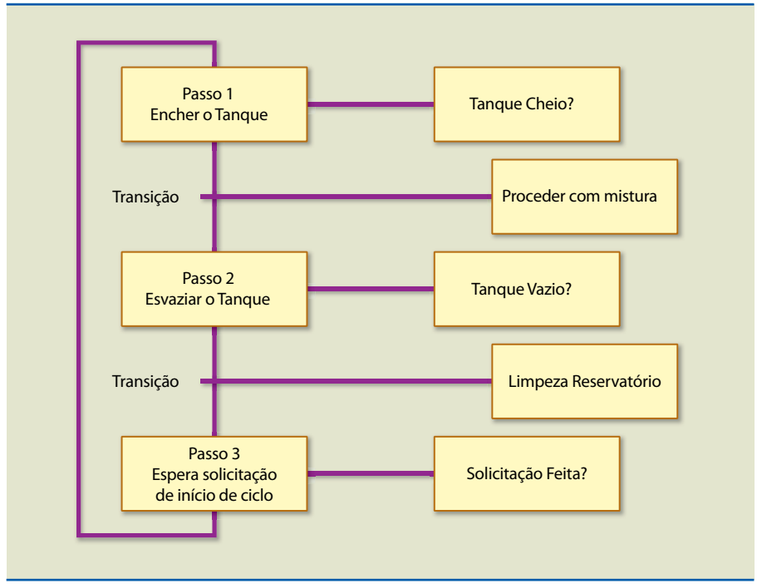


Figura 4.7
Exemplo de lógica em SFC.

Cada um dos blocos poderá ser programado na linguagem que for mais conveniente ao programador, pensando nas seguintes facilidades:

- Gerar o código do programa.
- Garantir que outras pessoas compreendam o programa.
- Fazer manutenção e alterações no *software*.

No entanto, vale ressaltar que, se a tarefa envolver lógica simples, poderá ser conveniente o uso do diagrama Ladder; e, se contiver muitas fórmulas matemáticas, será mais conveniente uma linguagem do tipo texto estruturado.