

INSTITUTO FEDERAL  
MINAS GERAIS  
Campus Avançado Ipatinga

# *Automação Industrial*

## *Linguagens de programação para CLPs*

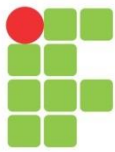
**Curso: Engenharia Elétrica**  
**Professor Sandro Dornellas**



## Linguagens de programação

Um item fundamental para utilização de um controlador lógico programável é a seleção da linguagem a ser utilizada, a qual depende de diversos fatores, entre eles:

- Disponibilidade da linguagem no CLP.
- Grau de conhecimento do programador.
- Solução a ser implementada.
- Nível da descrição do problema.
- Estrutura do sistema de controle.



## Linguagens de programação

Visando atender aos diversos segmentos da indústria, incluindo seus usuários, e uniformizar as várias metodologias de programação dos controladores industriais, a norma IEC 61131-3 definiu sintática e semanticamente cinco linguagens de programação:

|   |          |
|---|----------|
| Texto Estruturado (ST)                  | Textuais |
| Lista de Instruções (IL)                |          |
| Diagrama de Blocos e Funções (FDB)      | Gráficas |
| Linguagem <i>Ladder</i>                 |          |
| Seqüenciamento Gráfico de Funções (SFC) |          |

Linguagens segundo a norma IEC 61131-3.



## Linguagens textuais (IL e ST)

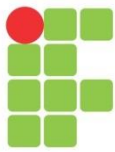
Linguagens textuais como Lista de Instruções e Texto Estruturado consistem em uma sequência de comandos padronizados correspondentes à funções específicas. Correções de programas são complexas e demandam mais tempo de aprendizado.

```
IF Posicao = 10 THEN
  Velocidade := 0 ;
  DesligaMotor := TRUE ;
ELSE IF Posicao = 20
  Velocidade := 50 ;
  DesligaMotor := FALSE ;
ELSE
  Velocidade := 100 ;
  DesligaMotor := TRUE ;
END_IF
```

(a) Exemplo código em ST.

|      |      |     |                               |
|------|------|-----|-------------------------------|
| 0001 | LD   | I1  | (*LE ENTRADA 1*)              |
| 0002 | ANDN | I2  | (*LÓGICA E NÃO*)              |
| 0003 | ST   | Q1  | (*COLOCA RESULTADO NA SAÍDA*) |
| 0004 |      |     |                               |
| 0005 | LD   | I3  |                               |
| 0006 | OR   | (   | IW4                           |
| 0007 | GE   | 500 | )                             |
| 0008 | ST   | Q2  |                               |
| 0009 |      |     |                               |

(b) Exemplo de código em IL.



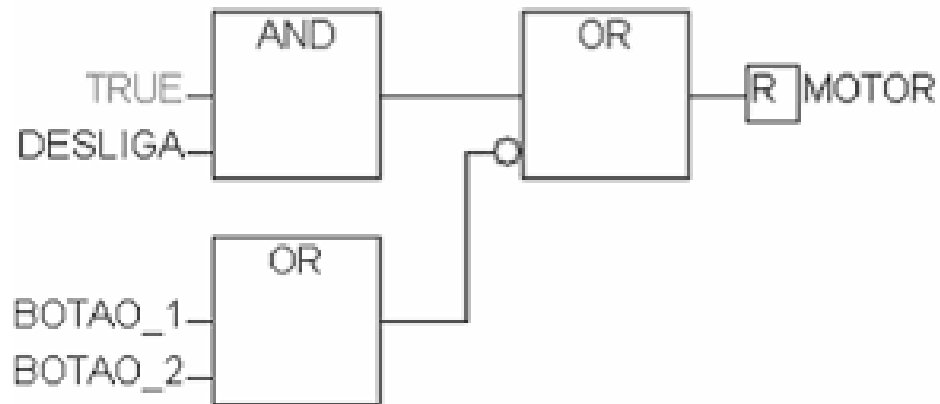
## **FBD - *Function Block Diagram***

Muito popular na Europa, cujos elementos são expressos por blocos interligados utilizando a linguagem de texto estruturado.

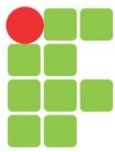
Permite um desenvolvimento hierárquico e modular do software, uma vez que podem ser construídos blocos de funções mais complexos a partir de outros menores e mais simples.



## FBD - *Function Block Diagram*



Exemplo de programação utilizando FBD.



## **SFC/Grafcet – *Sequential Function Chart***

Linguagem gráfica que permite a descrição de ações sequenciais, paralelas e alternativas existentes numa aplicação de controle.

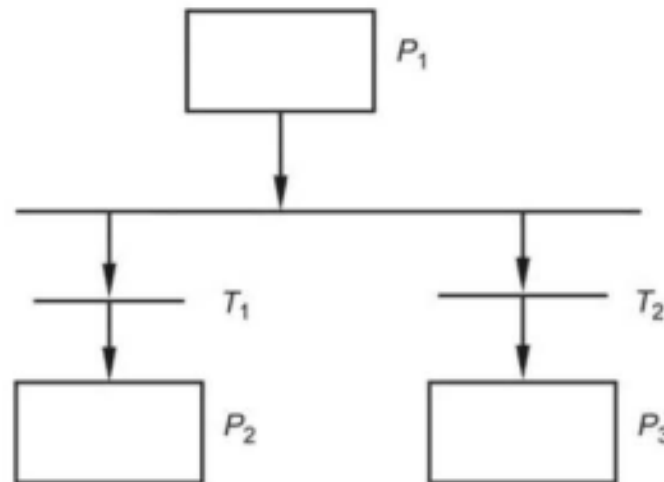
Isso permite uma visualização objetiva e rápida da operação e do desenvolvimento da automação implementada. É uma linguagem gráfica que se originou das Redes de Petri.

O SFC é programado em PASSOS P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub> (estados operacionais definidos) e TRANSIÇÕES T<sub>1</sub>, T<sub>2</sub> (condições definidas).



## SFC/Grafcet – *Sequential Function Chart*

Os passos contêm as ações booleanas, e as transições contêm os eventos necessários para autorizar a mudança de um passo a outro.



Exemplo de representação SFC.

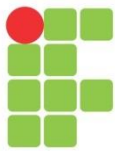




## ***LD – Ladder Diagram***

Linguagem gráfica baseada na lógica de relés e contatos elétricos para a realização de circuitos de comandos de acionamentos. Por ser a primeira linguagem utilizada pelos fabricantes, é a mais difundida e encontrada em quase todos os CLPs da atual geração.

Bobinas e contatos são símbolos utilizados nessa linguagem. Os símbolos de contatos programados em uma linha representam as condições que serão avaliadas de acordo com a lógica.

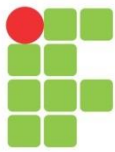


## LD – *Ladder Diagram*

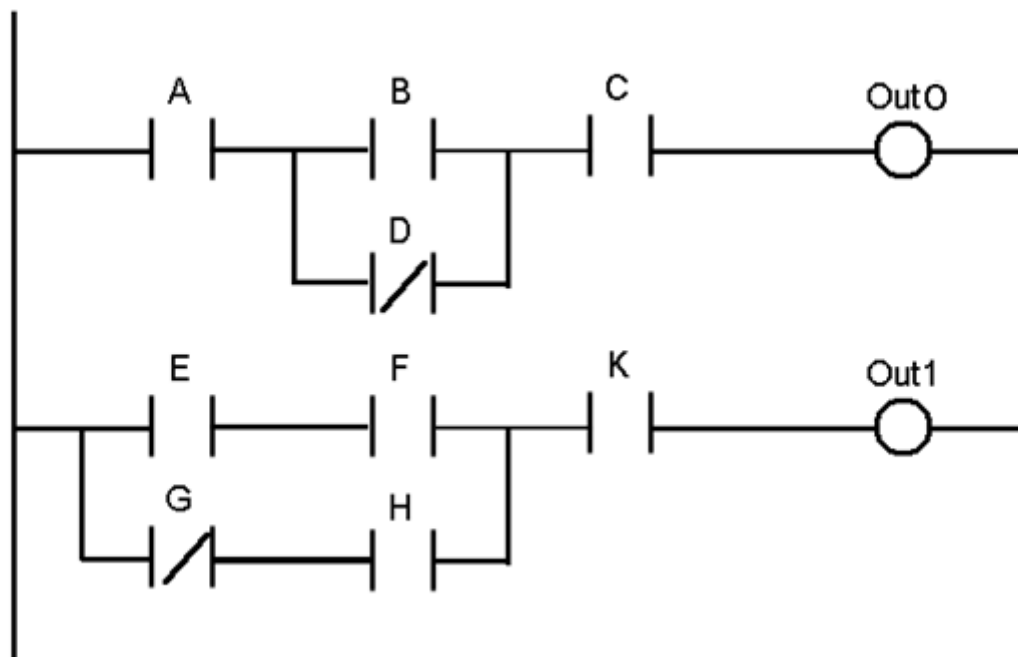
Como resultado determinam o controle de uma saída, que normalmente é representado pelo símbolo de uma bobina.

| Instrução                              | Representação |
|--|---------------|
| Contato normalmente aberto — NA        | -     -       |
| Contato normalmente fechado — NF       | -   /   -     |
| Bobina                                 | -( )-         |
| Bobina inversa (acionada, desenergiza) | -(I)-         |
| Bobina set                             | -(S)-         |
| Bobina reset                           | -(R)-         |

Instruções para diagrama *Ladder*.



## LD – *Ladder Diagram*

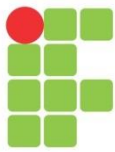


Exemplo de programação em diagrama *Ladder*.



## Vantagens da linguagem *Ladder*

- Possibilidade de uma rápida adaptação do pessoal técnico (semelhança com diagramas elétricos convencionais com lógica a relés);
- Possibilidade de aproveitamento do raciocínio lógico na elaboração de um comando feito com relés;
- Fácil recomposição do diagrama original a partir do programa de aplicação;
- Fácil visualização dos estados das variáveis sobre o diagrama Ladder, permitindo uma rápida depuração e manutenção do software;
- Documentação fácil e clara;

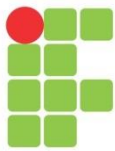


## Vantagens da linguagem *Ladder*

- Símbolos padronizados e mundialmente aceitos pelos fabricantes e usuários;
- Técnica de programação mais difundida e aceita industrialmente.

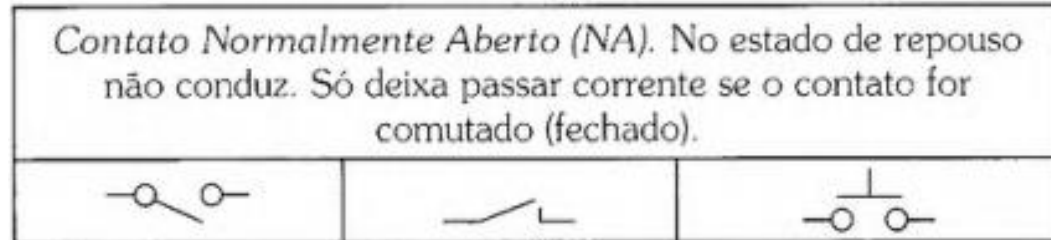
## Desvantagens

- Sua utilização em programas extensos ou com lógicas mais complexas é bastante difícil;
- Programadores não familiarizados com a operação de relés tendem a ter dificuldades com essa linguagem;
- Edição mais lenta.

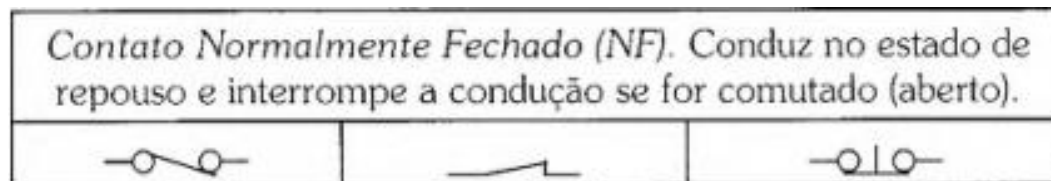


## Lógica de contatos

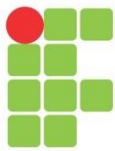
Um contato na linguagem *Ladder* equivale à uma chave (botoeira) elétrica, a qual pode estar em duas posições: aberta ou fechada.



(a) Contatos elétricos NA.



(b) Contatos elétricos NF.



## Lógica de contatos

Os contatos representam os elementos de entrada (sensores, botoeiras, etc) do sistema de controle.









| Fabricante    | Contato Normalmente Fechado (NF) | Contato Normalmente Aberto (NA) |
|---------------|----------------------------------|---------------------------------|
| IEC 61131-3   |                                  |                                 |
| Allen-Bradley |                                  |                                 |
| Siemens S7    |                                  |                                 |
| GE Fanuc      |                                  |                                 |

Símbolos *Ladder* para contatos, utilizados por alguns fabricantes de CLPs.



## Bobinas (relés)

Bobinas na linguagem *Ladder* são equivalentes aos relés elétricos, quando circula corrente pelo circuito a bobina é energizada, caso contrário, é desernegezada. Representam os elementos de saída (atuadores) do sistema de controle.

| Fabricante      | Bobina   | Bobina negada   |
|-----------------|--|---|
| IEC 61131-3     |    |    |
| Allen-Bradley   |    | Não disponível  |
| GE Fanuc        |   |   |
| Modicon Quantum |  |  |
| Siemens S7      |  | Não disponível  |

Símbolos *Ladder* para bobinas, utilizados por alguns fabricantes de CLPs.



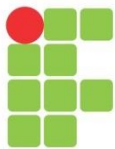


## Diagrama de contatos *Ladder*

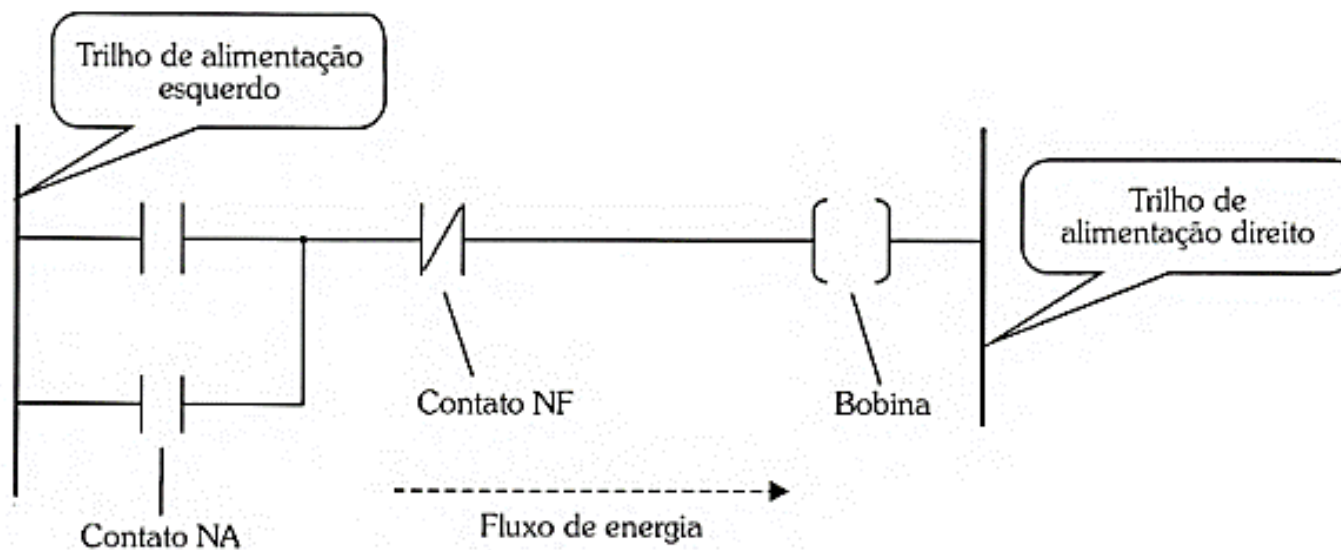
A ideia principal de um programa em linguagem Ladder é controlar o acionamento de saídas, dependendo da combinação lógica dos contatos de entrada.

Um diagrama de contatos é composto de duas barras verticais que representam os polos positivo e negativo de uma bateria.

Uma linha no diagrama é composta de um conjunto de condições de entrada (representado por contatos NA e NF) e uma instrução de saída no final da linha (representada pelo símbolo de uma bobina).



## Diagrama de contatos *Ladder*

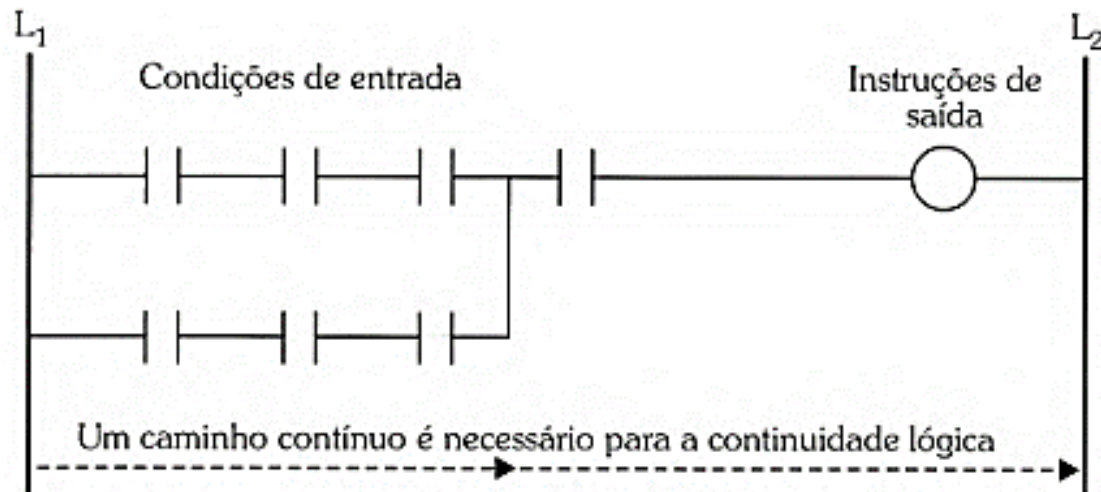


Estrutura típica de uma linha (degrau) em linguagem *Ladder*.



## Diagrama de contatos *Ladder*

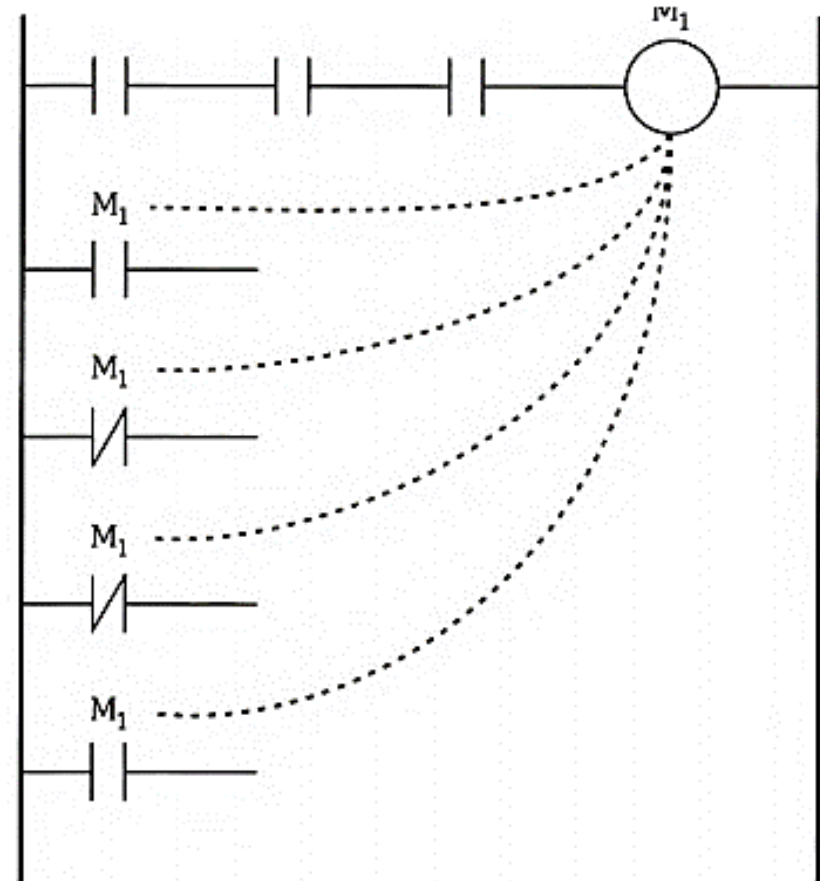
Uma linha é verdadeira, ou seja, energiza uma saída ou um bloco funcional, quando os contatos permitem um fluxo "virtual de eletricidade", ou seja, existe uma continuidade entre a barra da esquerda e a da direita.

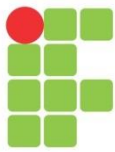




## Repetição de contatos

Os programas em *Ladder* podem ter quantos contatos normalmente abertos ou fechados desejar. Isso significa que um mesmo contato pode ser repetido várias vezes.

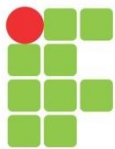




## Repetição de bobinas

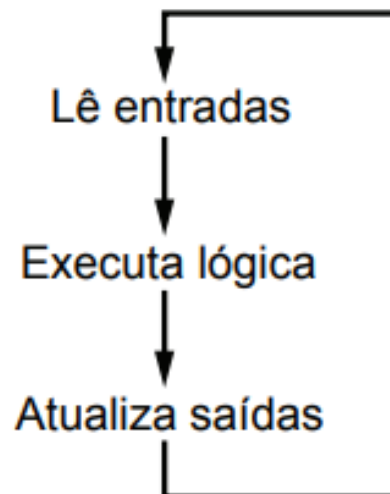
Embora alguns modelos de CLP permitam que uma mesma saída (bobina) seja repetida, é desaconselhável fazê-lo porque a repetição de uma saída em degraus diferentes vai tomar muito confusa a lógica do programa e, por consequência, dificultar o entendimento de quem assumir a manutenção desse programa.

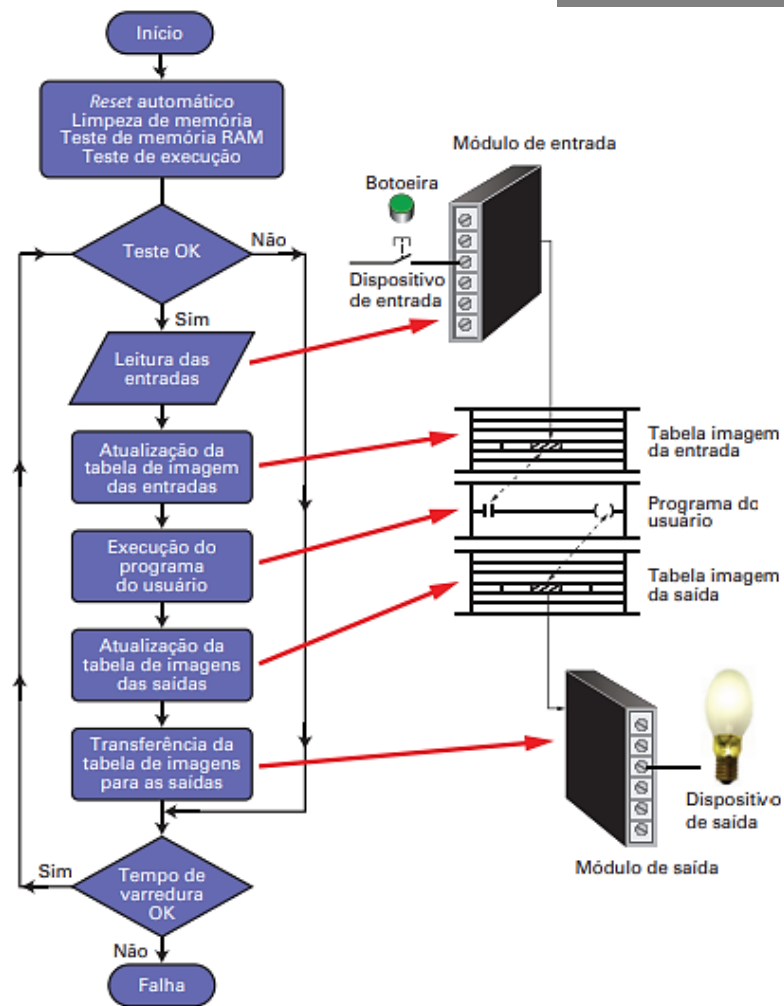
Recomenda-se, portanto, que uma bobina (saída) não seja repetida.



## Princípio de Funcionamento

*Loop infinito:*







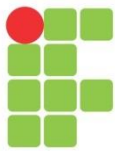
## Princípio de Funcionamento

A CPU realiza a leitura de todos os pontos de entrada e armazena-os na tabela de imagem das entradas.

Cada ponto de entrada corresponde a uma posição de memória específica (um bit de um determinado byte/word).

A tabela de imagem das entradas é acessada pela CPU durante a execução do programa de aplicação.





## Princípio de Funcionamento

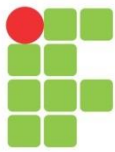
Após a execução deste segmento em um determinado scan, a leitura das entradas será realizada apenas no scan seguinte, ou seja, se o status (condição) de um determinado ponto de entrada mudar após a leitura das entradas, ele só terá influência na execução do programa de aplicação no scan seguinte, quando será percebida tal alteração.



## Princípio de Funcionamento

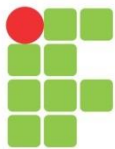
Neste segmento, a CPU executa as instruções do programa de aplicação, que definem a relação entre a condição das entradas e a atuação das saídas, ou seja, definem a lógica de controle a ser realizada.

Constrói, assim, uma nova tabela de imagem das saídas, gerada a partir da lógica executada.



## Princípio de Funcionamento

Após a execução do programa de aplicação, o conteúdo da tabela de imagem das saídas, construída de acordo com a lógica executada, é enviado aos pontos de saída correspondentes.



|                  |   | BIT |   |   |   |   |   |   |   |
|------------------|---|-----|---|---|---|---|---|---|---|
|                  |   | 7   | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| B<br>y<br>t<br>e | 0 | 0   | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
|                  | 1 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|                  | 2 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|                  | 3 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Ender. de  
Bit

I0.0

I0.1

I0.7

I1.0

Ender. de  
Byte

IB0 = Byte 0

IB1 = Byte 1

Ender. de  
Word

IW0 = Byte 0 + Byte 1

IW1 = Byte 1 + Byte 2

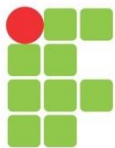
IW2 = Byte 2 + Byte 3

Forma de Endereçamento: I byte.bit



|                  |   | BIT |   |   |   |   |   |   |   |
|------------------|---|-----|---|---|---|---|---|---|---|
|                  |   | 7   | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| B<br>y<br>t<br>e | 0 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|                  | 1 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|                  | 2 | 0   | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
|                  | 3 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Q2.3



|                  |   | BIT |   |   |   |   |   |   |   |
|------------------|---|-----|---|---|---|---|---|---|---|
|                  |   | 7   | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| B<br>y<br>t<br>e | 0 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|                  | 1 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|                  | 2 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|                  | 3 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|                  | 4 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|                  | 5 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|                  | 6 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|                  | 7 | 0   | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

MO.0

MB1

MW2

MW1

MW0

MW2

MD0 - Bytes 0, 1, 2 e 3

MD4 - Bytes 4, 5, 6 e 7