

Fundamentos de Automação

Licenciatura em Engenharia Electrotécnica

Lógica Programável – 1ª Parte



ISEL
DEEA-SAR

Armando Cordeiro



Introdução

- A lógica programável utiliza sistemas electrónicos programáveis para fazer o tratamento dos dados.
- Ao contrário da lógica cablada, não é definida por um esquema em si mas por um programa introduzido numa memória.
- A lógica programável surgiu por volta dos anos 70 devido:
 - a grandes avanços registados nos circuitos electrónicos;
 - à necessidade de reduzir os custos dos sistemas cablados e resolver problemas complexos colocados pelas exigências de mercado.
- Os principais sistemas electrónicos programáveis usados na indústria são os **autómatos programáveis** (a.p.), os **microcontroladores** e os **terminais de interface humano-máquina**.
- Actualmente exige-se aos a.p. determinadas funções de controlo e de salvaguarda que visam garantir a segurança de pessoas e bens.



Vantagens

- **Menos componentes:** a lógica programável substitui um grande número de outros dispositivos.
- **Menos tempo de montagem:** resume-se à cablagem da alimentação das entradas e das saídas.
- **Menor espaço:** comando concentrado no autômato programável (a.p. ou PLC), ocupando menos espaço, podendo existir soluções distribuídas.
- **Maior fiabilidade:** Menos desgaste de peças em movimento (vantagem geral dos sistemas estáticos);
- **Maior flexibilidade:** as aplicações podem ser modificados com pouco esforço usando apenas um software de programação.
- **Manutenção facilitada:** a manutenção e a regulação ficam facilitadas.



Vantagens

- **Possibilidades de expansão:** ligação a unidades de expansão de entradas e saídas digitais, analógicas ou módulos de comunicações.
- **Elevado potencial de cálculo:** Velocidades de cálculo na ordem de 0,5 a 1,8 μ s/instrução (em determinadas aplicações pode não ser suficiente).
- **Funções matemáticas:** álgebra de inteiros e virgula flutuante. Cálculos trigonométricos avançados.
- **Funções de controlo:** Blocos com funções de controlo clássico (PID), Lógica difusa, etc.
- **Possibilidade de ligação remota:** Ligação a redes de comunicação digitais para troca de mensagens com redes de campo: CAN, profibus, DeviceNet, etc.
- **Os benefícios ultrapassam largamente os custos:** apesar dos custos de alguns equipamentos serem elevados, as potencialidades são muito grandes face à lógica cablada.

Lógica programável

■ Autômato

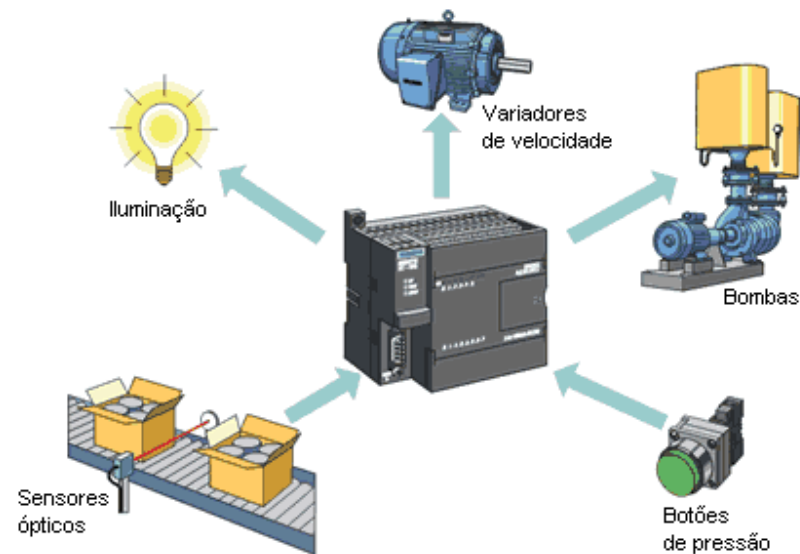
O **Autômato** é um equipamento electrónico robusto, que está preparado para trabalhar em ambientes industriais, programado pelo utilizador, com funcionamento cíclico e que é capaz de controlar processos.

Este controlo é efectuado através

- das entradas que dão informação ao autômato sobre o processo,
- do programa que analisa esta informação e
- das saídas que activam ou desactivam os equipamentos necessários.

Utilizados em aplicações comerciais, ou industriais, os autômatos actuam como controladores para máquinas e processos.

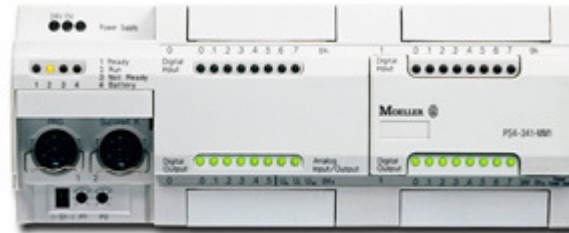
Monitorizam as entradas, tomam decisões e controlam as saídas de forma a automatizar máquinas e processos.



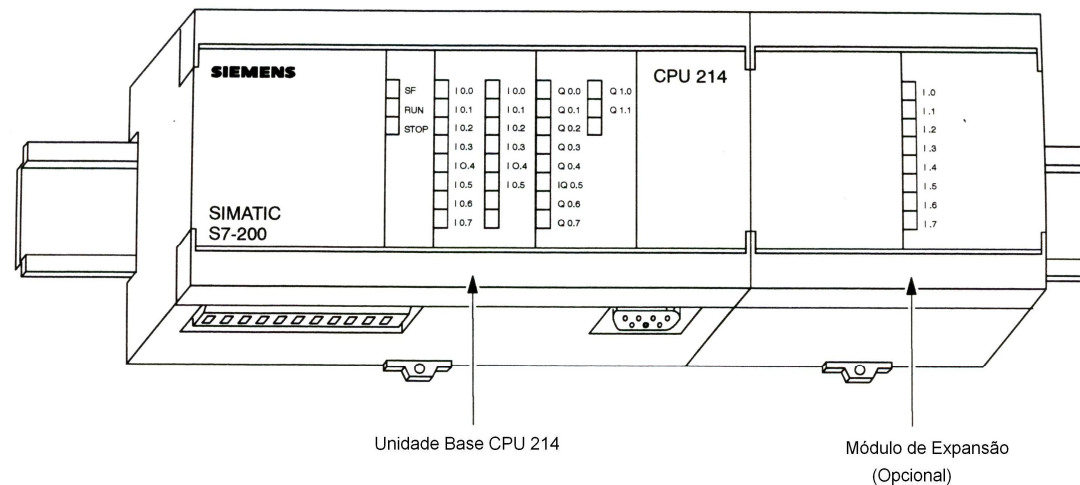
Lógica programável

■ Classificação dos Autômatos

- **Compactos** – Integram no mesmo conjunto todos os elementos necessários ao seu funcionamento (preços económicos e dedicados a pequenas aplicações).

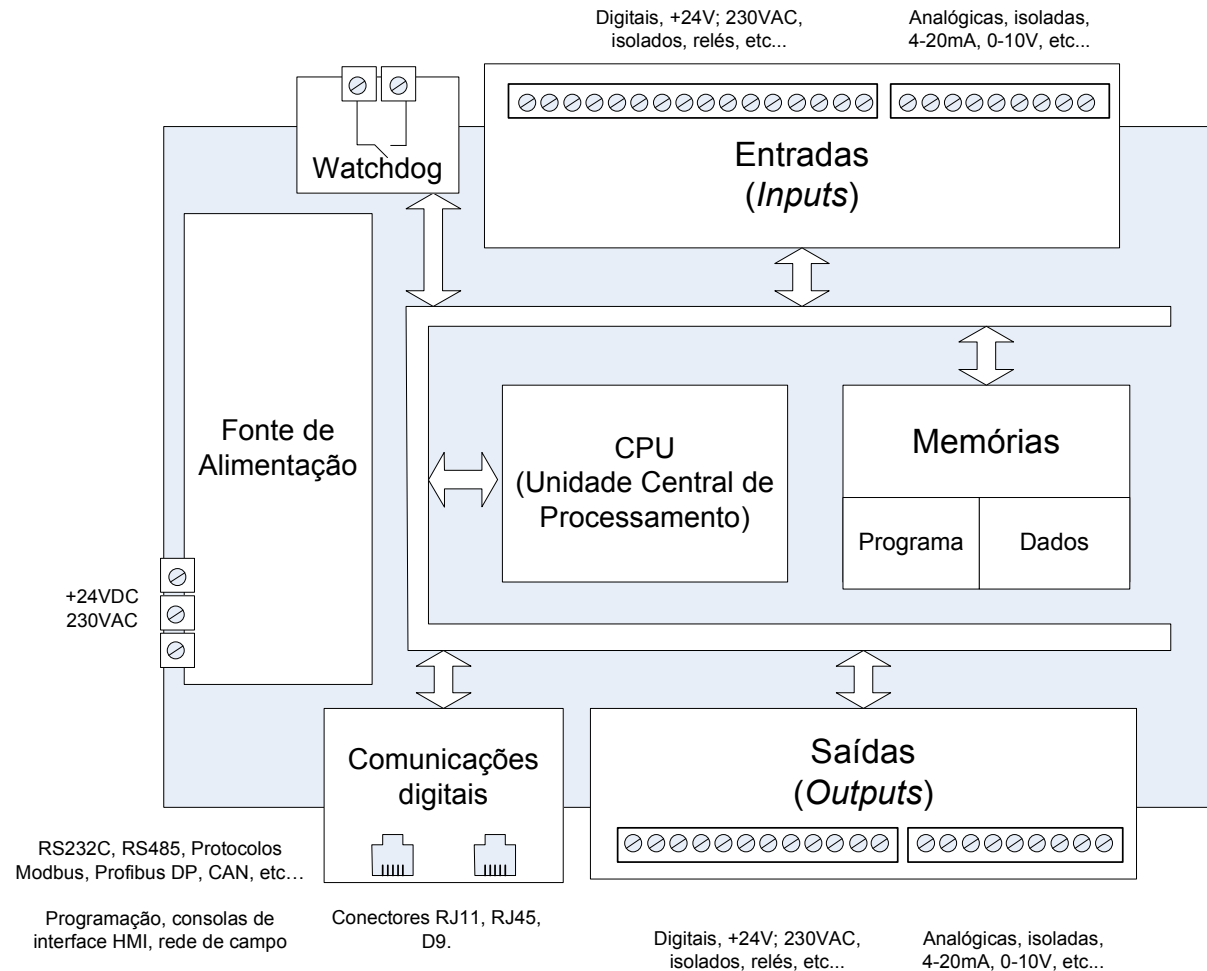


- **Modulares** – São compostos por diversos módulos que se associam de forma a obter a melhor configuração para cada aplicação.



Lógica programável

■ Arquitectura básica de um autómato programável





Lógica programável

- **Arquitectura básica de um autómato programável**
 - **CPU** – Unidade Central de processamento ou microprocessador é o elemento central do autómato. Executa o programa do utilizador, realizando o processamento de operações lógicas, aritméticas e de funções de controlo. Guarda o resultado destas operações em variáveis internas, na memória de dados.
 - **Memória** – Dentro do CPU existem várias áreas de memória, utilizadas para diferentes funções,
 - memória de programa – zona onde é armazenado o programa, efectuado pelo utilizador, que vai ser executado ciclicamente
 - memória de dados – zona dividida em sub-zonas especializadas de acordo com o tipo de dados a armazenar (valores das entradas e saídas, resultados das operações, temporizadores, etc)
 - memória de sistema operativo – zona onde se encontra o o programa que monitoriza o sistema (firmware)
 - memória de armazenamento - memória externa (EPROM, EEPROM, FLASH) onde se encontra o programa a executar pelo autómato.



Lógica programável

- **Arquitectura básica de um autómato programável**
 - **Comunicações digitais** – A interface de comunicação permite realizar a programação do autómato, estabelecer a sua ligação numa rede de comunicação digital ou a ligação a outros dispositivos, tais como outros autómatos e consolas de interface humano-máquina.
 - **Fonte de Alimentação** – Proporciona as tensões necessárias ao funcionamento do autómato. Estes podem ser alimentados a 24 VDC ou entre 80 e 240 VAC. Nos autómatos que possuem fonte de alimentação interna é vulgar a existência de uma tensão de saída de 24 VDC. Esta destina-se a alimentar directamente sensores e outros dispositivos de baixo consumo. Normalmente não têm capacidade para alimentar bobinas de contactores por isso, se se pretender alimentar actuadores com tensão de 24 VDC, é necessário existir uma fonte de alimentação externa.
 - **Watchdog** – temporizador que detecta se o tempo de execução do programa excede um determinado tempo máximo. Fecha ou abre um contacto exterior em caso de falha. Sistema de segurança básico.



Lógica programável

- **Arquitectura básica de um autómato programável**
 - **Entradas** — Asseguram a interligação directa do a.p. com o seu ambiente de trabalho. Os sensores colocados no processo interpretam as grandezas físicas (velocidade, caudal, temperatura, etc) transformando-as em sinais eléctricos normalizados, com significado lógico. Esta informação, sobre o estado dos sensores, é transmitida ao programa, através do módulo de entradas.
 - **Saídas** — Tal como as entradas, asseguram a interligação directa do a.p. com o seu ambiente de trabalho. As saídas transmitem directa ou indirectamente os sinais que irão activar ou desactivar os elementos do processo (motores, cilindros, etc). Geralmente estes elementos (actuadores) têm um consumo elevado e por isso as saídas do autómato não os podem actuar directamente, pois não têm capacidade para tal, sendo necessário um interface entre elas, designado por pré-actuador (contactores, electroválvulas, etc)

Lógica programável

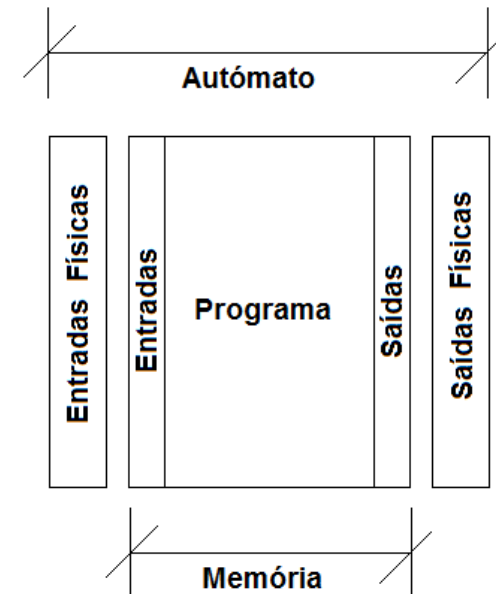
■ Princípio de funcionamento do Autómat Programável

Através das **entradas físicas** o autómato recebe informações do processo e guarda esta informação na **memória de dados**, na área correspondente às entradas.

Esta informação é processada pela CPU e de acordo com o programa altera (ou mantém) o estado das saídas, na **memória de dados**, na área das saídas.

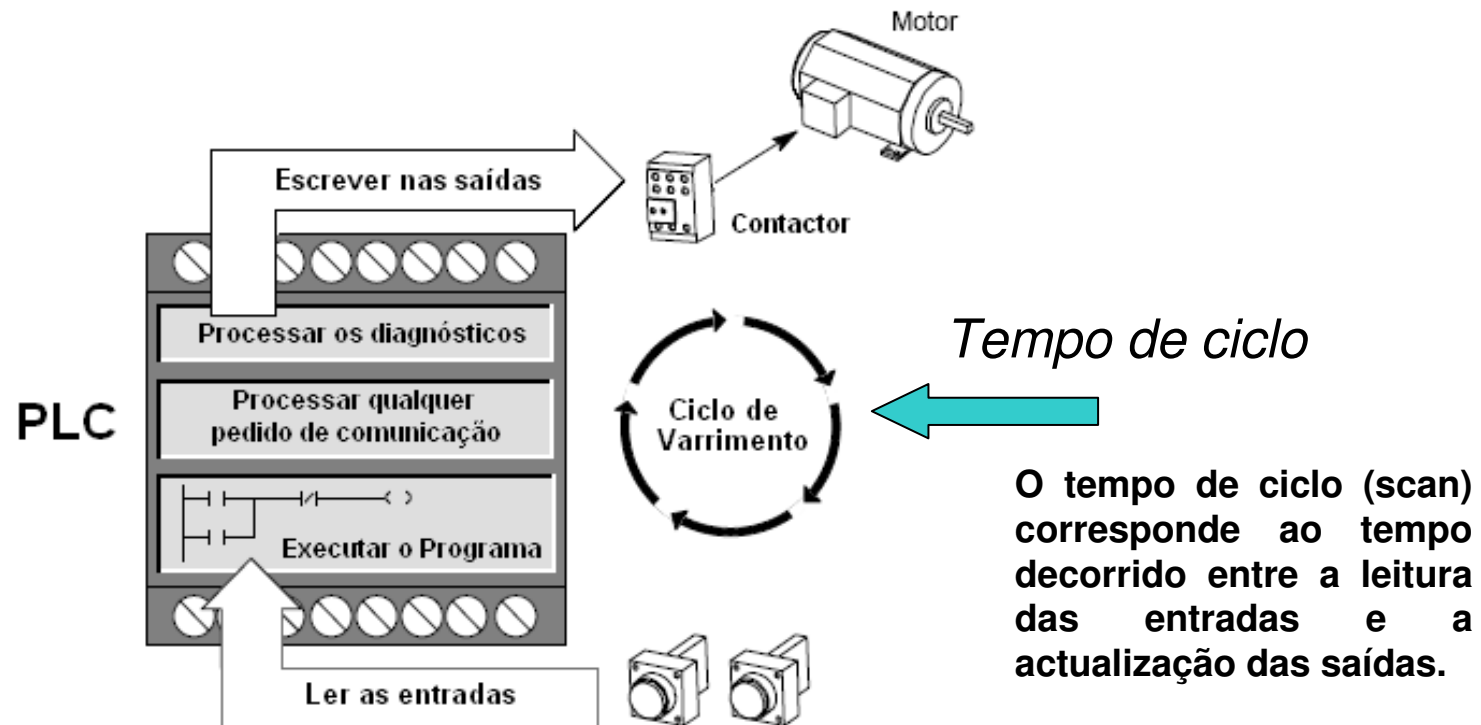
Uma vez executado todo o programa, o estado das saídas é passado para as **saídas físicas**, enviando ordens para os actuadores ou pré-actuadores, que por sua vez actuam sobre o processo.

A memorização prévia das variáveis de entrada destina-se a evitar alterações nas mesmas durante o decorrer do programa. O processamento é feito a partir do estado das variáveis memorizadas que é igual durante todo o ciclo. A actualização das saídas físicas dá-se no final do programa, quando todos os comandos a transmitir ao exterior já estão definidos.



Lógica programável

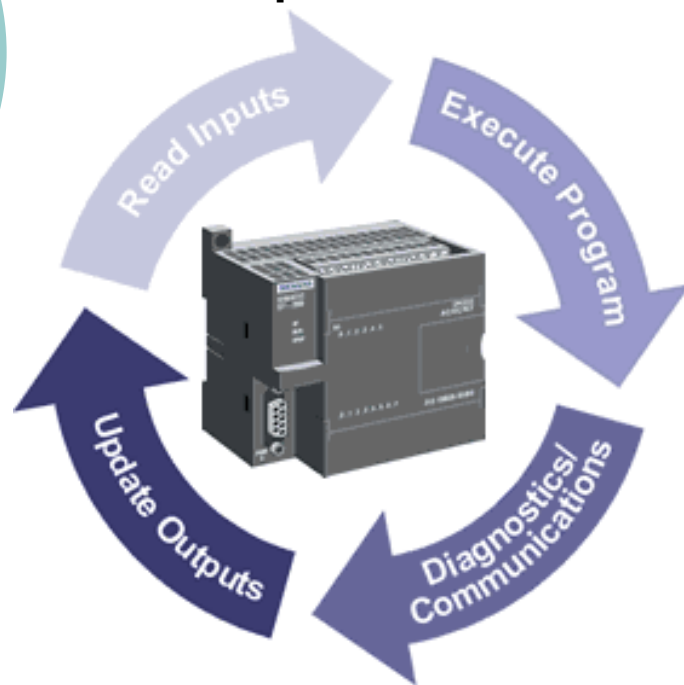
- Princípio de funcionamento do Autómat Programável



Os autómatos estão preparados para executarem ciclicamente uma sequência de tarefas, ou seja funcionam por fases que se repetem continuamente.

Lógica programável

■ Princípio de funcionamento do Autómat Programável



Fase 1: leitura das variáveis de entrada

O ciclo de scan começa com a leitura do estado das entradas físicas e respectivo armazenamento nas variáveis de entradas.

Fase 2: execução do programa

A informação presente nas variáveis de entradas, é processada pela CPU, que de acordo com o programa existente na memória actualiza o estado das variáveis de saídas, por cada instrução do programa.

Fase 3: diagnóstico de erros

Uma vez completa a execução do programa, a CPU executa uma série de tarefas internas de diagnósticos e comunicações.

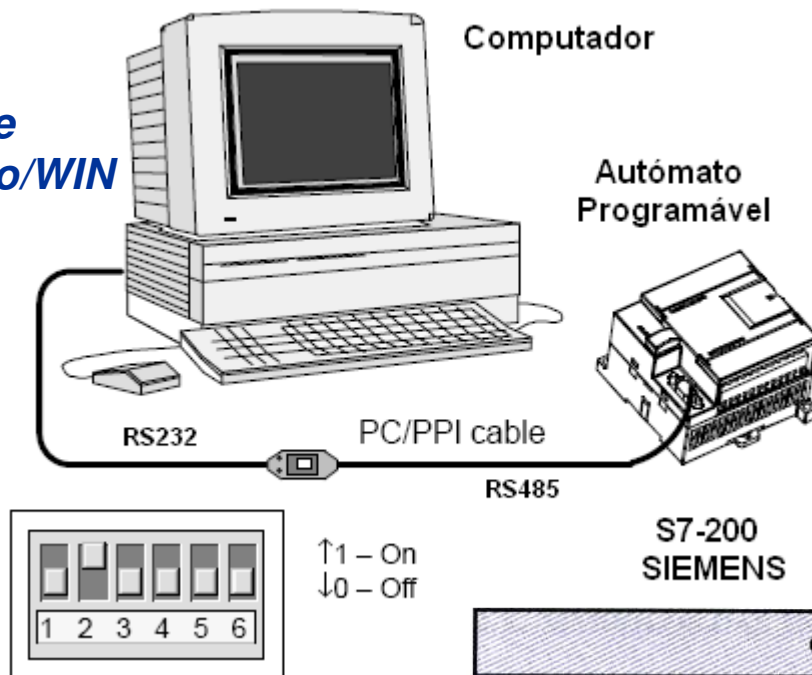
Fase 4: actualização das variáveis de saída

Caso não existam erros os valores das variáveis de saída, contidos na memória de dados, são transferidos para as saídas físicas e o autómato passa ao ciclo seguinte (recomeçando a fase 1 e assim sucessivamente).

Lógica programável

■ Programação do Autômato S7-200 da Siemens

Software
Step 7-Micro/WIN

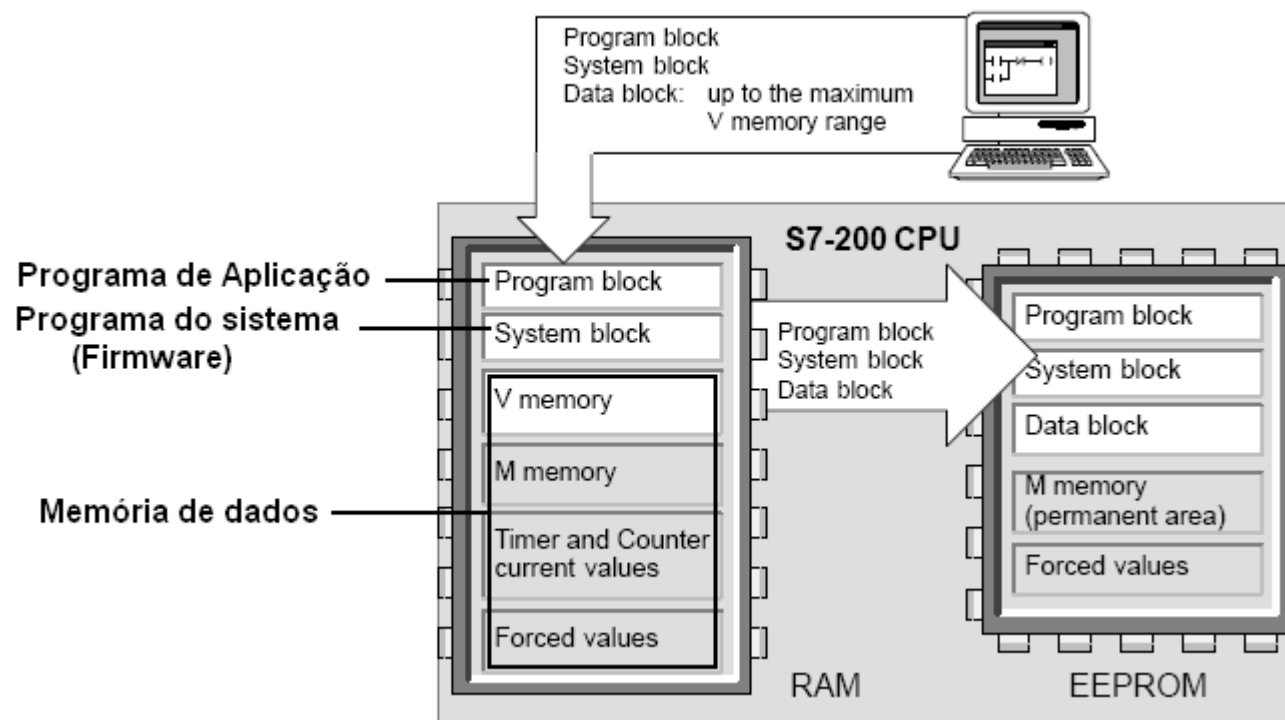


- **Computador Pessoal** — A programação é usualmente feita em computador pessoal, o que torna a programação e testes muito práticos.
- **Programador (PG)** — Trata-se de uma consola de programação dedicada vendida pelos fabricantes com software incluído.

Cabo de comunicação PC/PPI					
Interruptores DIP	123	Velocidade (kbits/s)	4	5	6
	000	115,2 – 38,4	1=10 bits 0=11 bits	1=DTE 0=DCE	1=RTS para XMT 0=RTS sempre
	001	19,2			
	010	9,6			
	011	4,8			
	100	2,4			
	101	1,2			

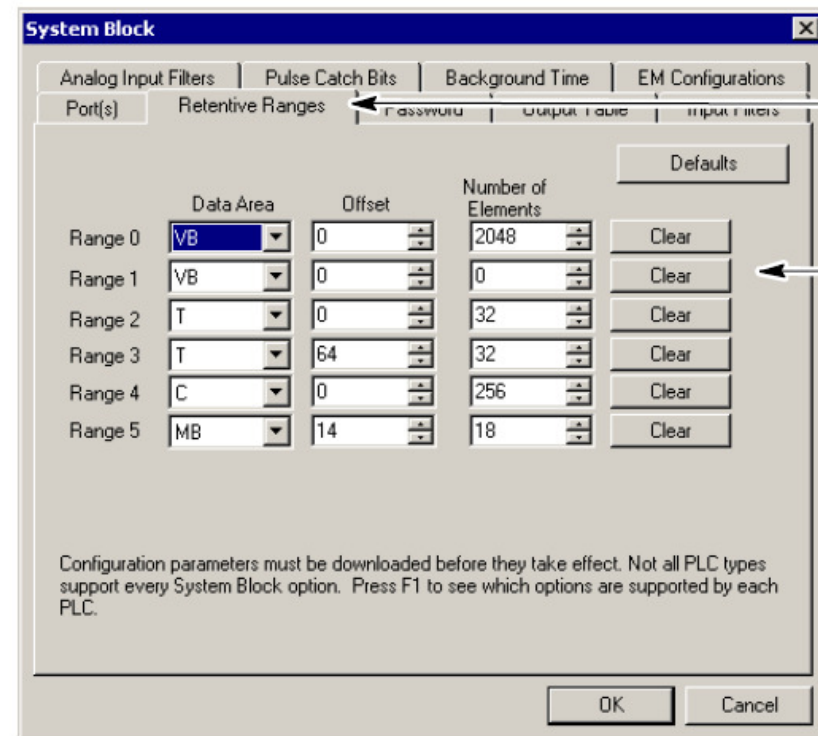
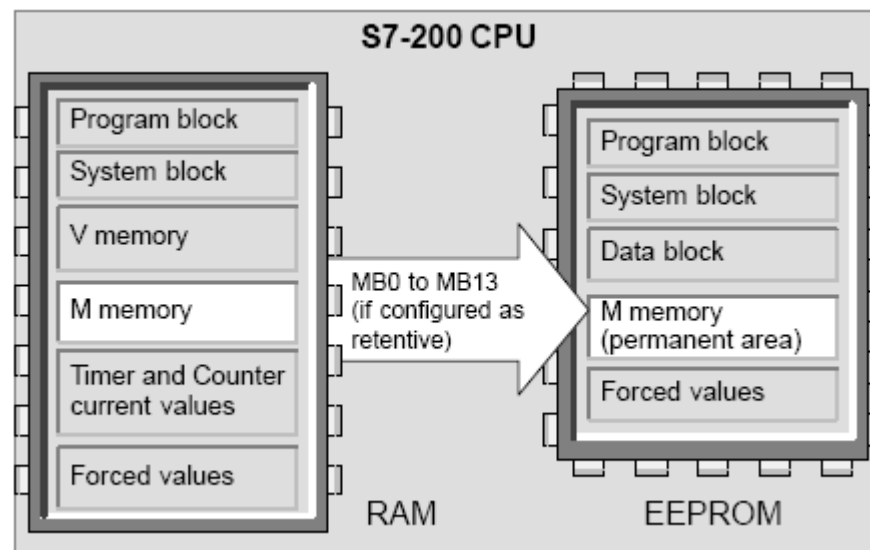
Lógica programável

- **Programação do Autômato S7-200 da Siemens**
 - **RAM e EEPROM** – Salvaguarda de determinadas áreas da memória RAM na EEPROM interna ou externa. A RAM pode armazenar os dados durante alguns dias e a EEPROM durante muitos anos (tipicamente >40 anos).




Lógica programável

- **Programação do Autômato S7-200 da Siemens**
 - A salvaguarda de outras áreas da memória RAM na EEPROM interna ou externa também pode ser efectuada através da programação.



Lógica programável

- **Programação do Autômato S7-200 da Siemens**
 - A norma **IEC61131-3** prevê cinco linguagens de programação diferentes, que podem ser combinadas dentro de uma aplicação:
 - **Três linguagens gráficas:**
 - Diagramas funcionais sequenciais (Grafcet) - (SFC);
 - Diagramas de Blocos Lógicos - (FBD);
 - Diagramas de Contactos (LD);
 - **Duas linguagens baseadas em texto:**
 - Linguagem Textual estruturada (ST);
 - Lista de Instruções (IL);
-  A norma **IEC61131-3** não prevê uma completa correspondência entre todas as linguagens de programação. Na prática, existe apenas um pequeno conjunto de funções que podem ser correctamente traduzidas de uma linguagem para outras.



Lógica programável

- **Programação do Autômato S7-200 da Siemens**
 - **O software de programação STEP 7 Micro/Win permite:**
 - **Programar em Diagramas de Contactos (LD);**
 - **Programar em Listas de Instruções (IL);**
 - **Programar em Diagramas de Blocos Lógicos (FDB);**
 - **Programar Diagramas funcionais sequenciais (SFC), não na forma como se conhecem os Grafcet, mas sim com recurso a blocos sequenciais.**
 - **Utilizar linguagem própria (SIMATIC) ou a recomendada pela IEC61131-3.**



Lógica programável

- **Organização da Memória do S7-200 da Siemens**
 - A memória de dados encontra-se organizada de acordo com as funções a realizar e estão previstas as seguintes áreas:

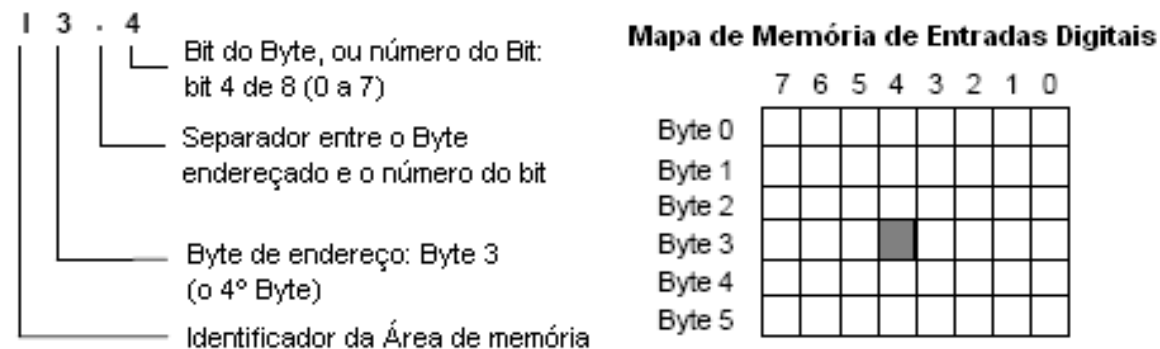
Designação	Código
Área das Entradas Digitais	<i>I</i>
Área das Saídas Digitais	<i>Q</i>
Área das Entradas Analógicas	<i>AI</i>
Área das Saídas Analógicas	<i>AQ</i>
Área da Memória Retentiva	<i>M</i>
Área das Memórias Especiais	<i>SM</i>
Área da Memória Variável	<i>V</i>
Área do Controlo Sequencial	<i>S</i>
Área dos Temporizadores	<i>T</i>
Área dos Contadores	<i>C</i>
Área da Memória Local	<i>L</i>

Nota: No caso da IEC61131 usa-se a notação acima indicada precedida do símbolo “%”.

Lógica programável

■ Endereçamento da Memória do S7-200 da Siemens

- Para aceder **aos bits da área de memória** I, Q, V, M, S e SM é necessário indicar o seguinte:



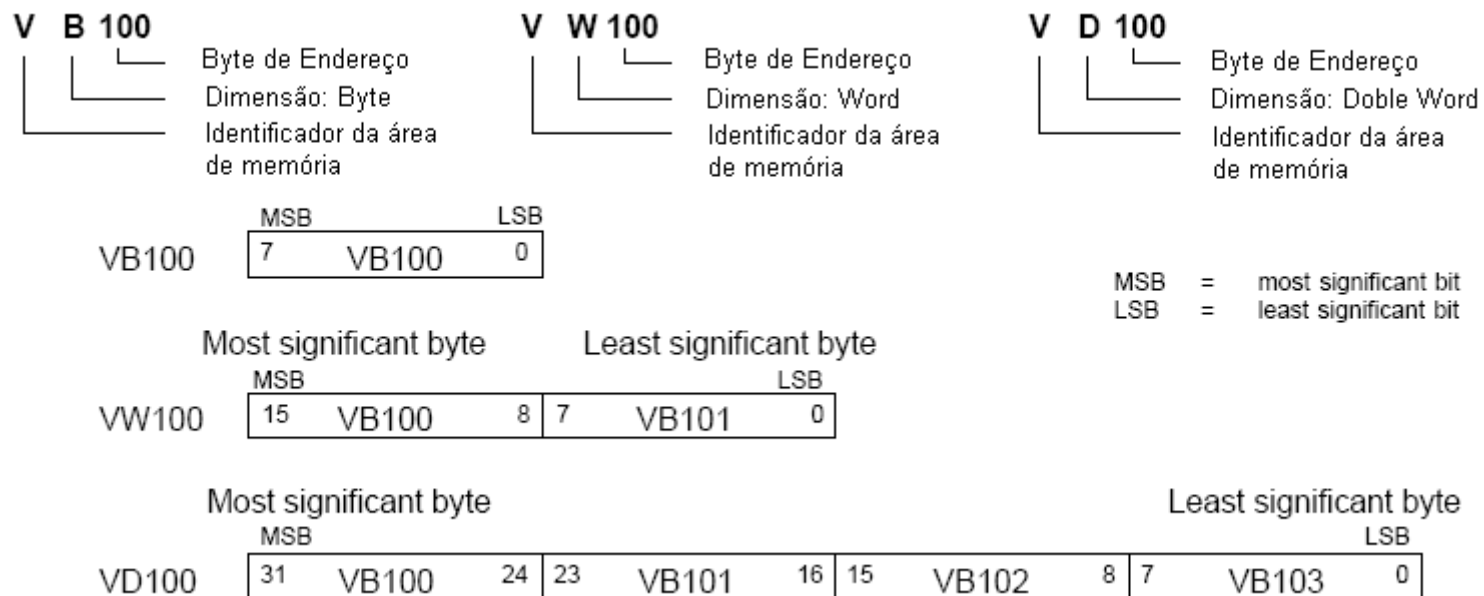
- Acesso aos **bits** das áreas de memória por modelo de CPU.

Access Method		CPU 221	CPU 222	CPU 224, CPU 226	CPU 226XM
Bit access (byte.bit)	I	0.0 to 15.7	0.0 to 15.7	0.0 to 15.7	0.0 to 15.7
	Q	0.0 to 15.7	0.0 to 15.7	0.0 to 15.7	0.0 to 15.7
	V	0.0 to 2047.7	0.0 to 2047.7	0.0 to 5119.7	0.0 to 10239.7
	M	0.0 to 31.7	0.0 to 31.7	0.0 to 31.7	0.0 to 31.7
	SM	0.0 to 179.7	0.0 to 299.7	0.0 to 549.7	0.0 to 549.7
	S	0.0 to 31.7	0.0 to 31.7	0.0 to 31.7	0.0 to 31.7
	T	0 to 255	0 to 255	0 to 255	0 to 255
	C	0 to 255	0 to 255	0 to 255	0 to 255
	L	0.0 to 59.7	0.0 to 59.7	0.0 to 59.7	0.0 to 59.7

Lógica programável

■ Endereçamento da Memória do S7-200 da Siemens

- Para aceder **aos bytes, palavras e palavras duplas da área de memória** é necessário indicar o seguinte (nem todas as áreas podem ser endereçadas por estes três métodos):



- Se a VD0 já estivesse utilizada e se necessitar de uma VW e de outra VD que endereços poderia utilizar?

Lógica programável

- **Endereçamento da Memória do S7-200 da Siemens**

- **Formato dos dados adotado**

Representação	Byte (B)	Palavra (W-Word)	Palavra dupla (D – Double Word)
Inteiros positivos	0 a 255 0 a FF	0 a 65535 0 a FFFF	0 a 4.294.967.295 0 a FFFF FFFF
Inteiros	-128 a + 127 80 a 7F	-32.768 a +32.767 8000 a 7FFF	-2.147.483.648 a +2.147.483.647 8000 0000 a 7FFF FFFF
Reais (IEEE 32-bit Virgula flutuante)	Não aplicável	Não aplicável	+1.175495E-38 a +3.402823E+38 -1.175495E-38 a -3.402823E+38

- **Acesso aos bytes da área de memória por modelo de CPU.**

Access Method		CPU 221	CPU 222	CPU 224, CPU 226	CPU 226XM
Byte access	IB	0 to 15	0 to 15	0 to 15	0 to 15
	QB	0 to 15	0 to 15	0 to 15	0 to 15
	VB	0 to 2047	0 to 2047	0 to 5119	0 to 10239
	MB	0 to 31	0 to 31	0 to 31	0 to 31
	SMB	0 to 179	0 to 299	0 to 549	0 to 549
	SB	0 to 31	0 to 31	0 to 31	0 to 31
	L	0 to 63	0 to 63	0 to 63	0 to 255
	AC	0 to 3	0 to 3	0 to 3	0 to 255

Lógica programável

■ Endereçamento da Memória do S7-200 da Siemens

- Acesso às **palavras** da área de memória por modelo de CPU.

Access Method		CPU 221	CPU 222	CPU 224, CPU 226	CPU 226XM
Word access	IW	0 to 14	0 to 14	0 to 14	0 to 14
	QW	0 to 14	0 to 14	0 to 14	0 to 14
	VW	0 to 2048	0 to 2048	0 to 5118	0 to 10238
	MW	0 to 30	0 to 30	0 to 30	0 to 30
	SMW	0 to 178	0 to 298	0 to 548	0 to 548
	SW	0 to 30	0 to 30	0 to 30	0 to 30
	T	0 to 255	0 to 255	0 to 255	0 to 255
	C	0 to 255	0 to 255	0 to 255	0 to 255
	LW	0 to 58	0 to 58	0 to 58	0 to 58
	AC	0 to 3	0 to 3	0 to 3	0 to 3
	AIW	None	0 to 30	0 to 62	0 to 62
	AQW	None	0 to 30	0 to 62	0 to 62

- Acesso às **palavras duplas** da área de memória por modelo de CPU.

Access Method		CPU 221	CPU 222	CPU 224, CPU 226	CPU 226XM
Double word access	ID	0 to 12	0 to 12	0 to 12	0 to 12
	QD	0 to 12	0 to 12	0 to 12	0 to 12
	VD	0 to 2044	0 to 2044	0 to 5116	0 to 10236
	MD	0 to 28	0 to 28	0 to 28	0 to 28
	SMD	0 to 176	0 to 296	0 to 546	0 to 546
	SD	0 to 28	0 to 28	0 to 28	0 to 28
	LD	0 to 58	0 to 58	0 to 58	0 to 58
	AC	0 to 3	0 to 3	0 to 3	0 to 3
	HC	0, 3, 4, 5	0, 3, 4, 5	0 to 5	0 to 5

Lógica programável

- **Endereços das principais Marcas Especiais – S7200 da Siemens**

Bits	Função
SM0.0	Bit que está sempre em On.
SM0.1	Bit que vai a “1”, no 1.º ciclo, após ordem de execução do programa.
SM0.2	Bit que se activa durante um ciclo se se perderem os dados remanescentes.
SM0.3	Bit que se activa durante um ciclo quando, se passa o autómato a modo Run, após retoma da alimentação.
SM0.4	Bit de relógio com T = 1 min (30 s em On – 30 s em Off).
SM0.5	Bit de relógio com T = 1 s (0,5 s em On – 0,5 s em Off).
SM0.6	Bit de relógio de ciclo; está On num ciclo e Off no ciclo seguinte.
SM0.7	Bit que indica a posição do selector do modo de funcionamento do autómato (Off=Term; On=Run).
SMB28	Byte que armazena o valor digital correspondente à posição do potenciómetro analógico 0.
SMB29	Byte que armazena o valor digital correspondente à posição do potenciómetro analógico 1.

- Podem ser consultados no apêndice D do manual do S7200.

Lógica programável

- Programação em IL e LD (segundo Step 7 Micro/Win).

(Lista de Instruções)		(Diagrama de contactos)	
Instrução		Símbolo	
LD			
LDN			
A			
AN			
O			
ON			
EU			
ED			
N			

(Lista de Instruções)		(Diagrama de contactos)	
Instrução		Símbolo	
ALD			
OLD			
=			
S			
R			

Instruções e símbolos lógicos básicos

Lógica programável

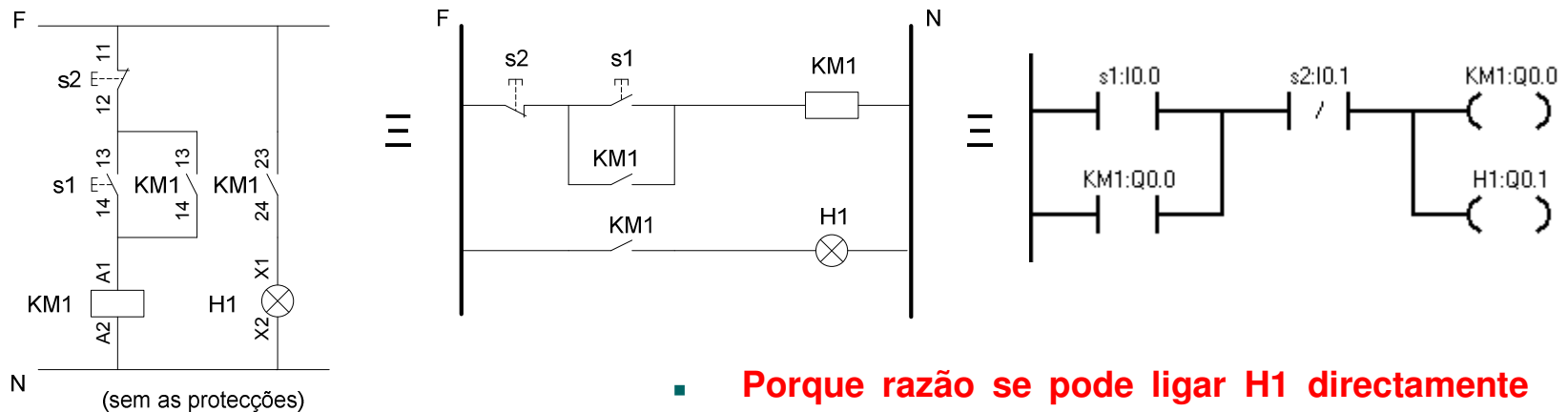
- Programação em IL e LD (segundo Step 7 Micro/Win).
- Significado das principais instruções.

Instrução	Código ³	Significado
Load	LD	Carrega um valor (início de uma rede).
Load Not	LDN	Carrega um valor invertido (início negado de uma rede).
And	A	Produto lógico (contacto série aberto).
And Not	AN	Produto lógico negado (contacto série fechado).
Or	O	Soma lógica (contacto paralelo aberto).
Or Not	ON	Soma lógica negada (contacto paralelo fechado).
Edge Up	EU	Na transição 0→1 é gerado, num <i>scan</i> , o valor lógico "1" (detecta flanco positivo).
Edge Down	ED	Na transição 1→0 é gerado, num <i>scan</i> , o valor lógico "1" (detecta flanco negativos).
Not	N	Inverte o valor.
And Load	ALD	Operação lógica <i>And</i> entre dois blocos lógicos.
Or Load	OLD	Operação lógica <i>Or</i> entre dois blocos lógicos.
Output	=	Atribui valor.
Set	S	Coloca no estado "1".
Reset	R	Coloca no estado "0".
No operation	NOP	Sem operação (instrução sem efeito no programa).
Stop	STOP	Finaliza imediatamente a execução do programa, alterando o modo de operação de RUN para STOP.
End Program	END	Fim do programa.

Lógica programável

- Programação em IL e LD (segundo Step 7 Micro/Win).

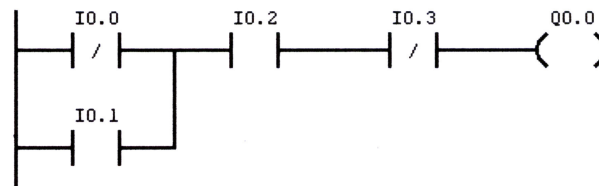
- Analogia entre esquemas eléctricos e diagramas de contactos



- Porque razão se pode ligar H1 directamente aos contactos de s1 e s2, no autómato e no esquema eléctrico não se pode?

- Diagrama Básico – Exemplo I

Diagrama de contactos



Lista de instruções

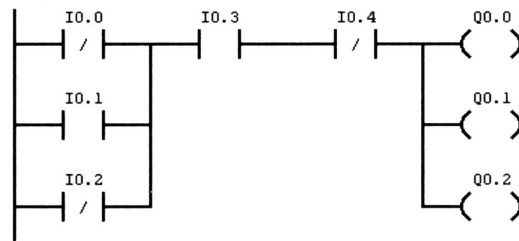
LDN	I0.0
O	I0.1
A	I0.2
AN	I0.3
=	Q0.0

Lógica programável

- Programação em IL e LD (segundo Step 7 Micro/Win).

- Diagrama Básico – Exemplo II

Diagrama de contactos

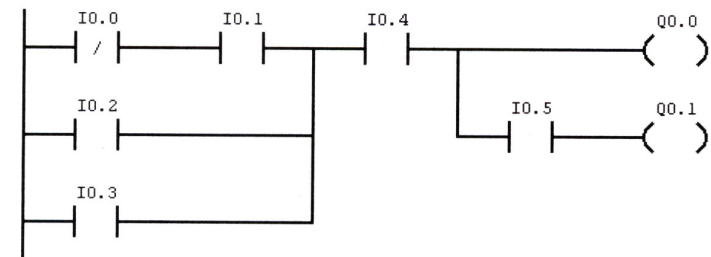


Lista de instruções

LDN	I0.0
O	I0.1
ON	I0.2
A	I0.3
AN	I0.4
=	Q0.0
=	Q0.1
=	Q0.2

- Diagrama Básico – Exemplo III

Diagrama de contactos



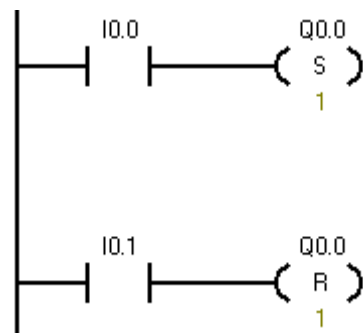
Lista de instruções

LDN	I0.0
A	I0.1
O	I0.2
O	I0.3
A	I0.4
=	Q0.0
A	I0.5
=	Q0.1

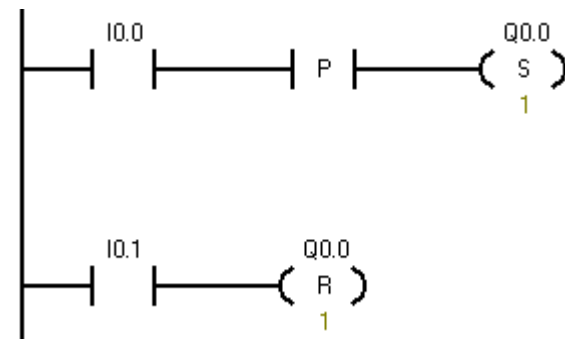
Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).

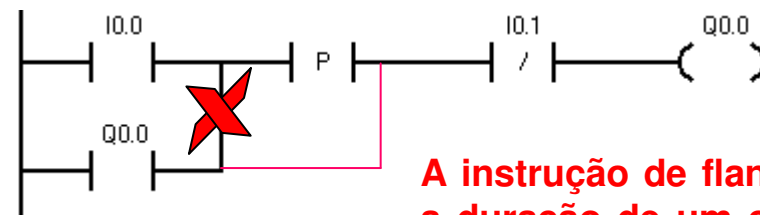
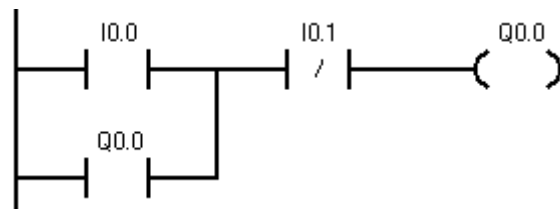
- Instruções SET e RESET



- Utilização de flancos



- Circuito equivalente

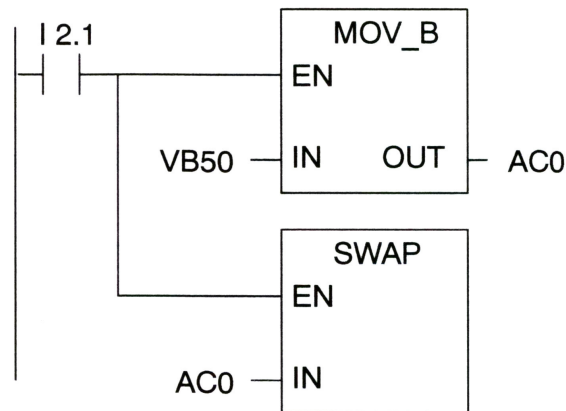


A instrução de flanco tem a duração de um ciclo de scan (entre 10 a 50 ms), logo a auto-alimentação tem de a incluir

Lógica programável

■ Programação em LD (segundo Step 7 Micro/Win).

■ Instruções de Transferência



- **MOV_B** (Move Byte)
- **MOV_W** (Move Word)
- **MOV_DW** (Move Double Word)
- **MOV_R** (Move Real)
- **SWAP** (Troca os bytes mais e menos significativos apenas para Words)

Se for necessário escrever um valor numa variável, consoante o tipo da variável (Byte, Word ou Double-word) utiliza-se a função MOVE correspondente.

Suponha que o programa que está a implementar no autómato precisa de inicializar os bits V0.0, V0.1 e V0.2 a 0, o bit V1.0 a 1, a double word MD0 a 15, o inteiro VW2 a 4 e todas as saídas a zero, excepto a primeira que fica a 1.

Como faria?



Lógica programável

- **Programação em LD (segundo Step 7 Micro/Win).**
- **Temporizadores (Timers)**
 - **Tipos de Temporizadores**
 - **TONR** – Temporizador memorizado com atraso ao trabalho;
 - **TON** – Temporizador com atraso ao trabalho;
 - **TOF** – Temporizador com atraso ao repouso;
 - **TP** – Temporizador por impulsos.
 - **Características**
 - **PT (*Preset time*)** – Entre 0 e 32767 e programado pelo utilizador, indica qual o tempo que se pretende contar ;
 - **ET (*Elapsed time*)** – Entre 0 e 32767; indica o tempo decorrido;
 - **IN (*Input*)** – Dá início ao processo de contagem (Binária);
 - **Q (*Output*)** – Indica que o temporizador atingiu o tempo programado

Lógica programável

■ Endereços dos Temporizadores– S7200 da Siemens

- Estão disponíveis 256 temporizadores com diferentes características. Cada um deles só pode contar 32767 intervalos de tempo. A duração da cada intervalo de tempo é igual à base de tempo do temporizador.

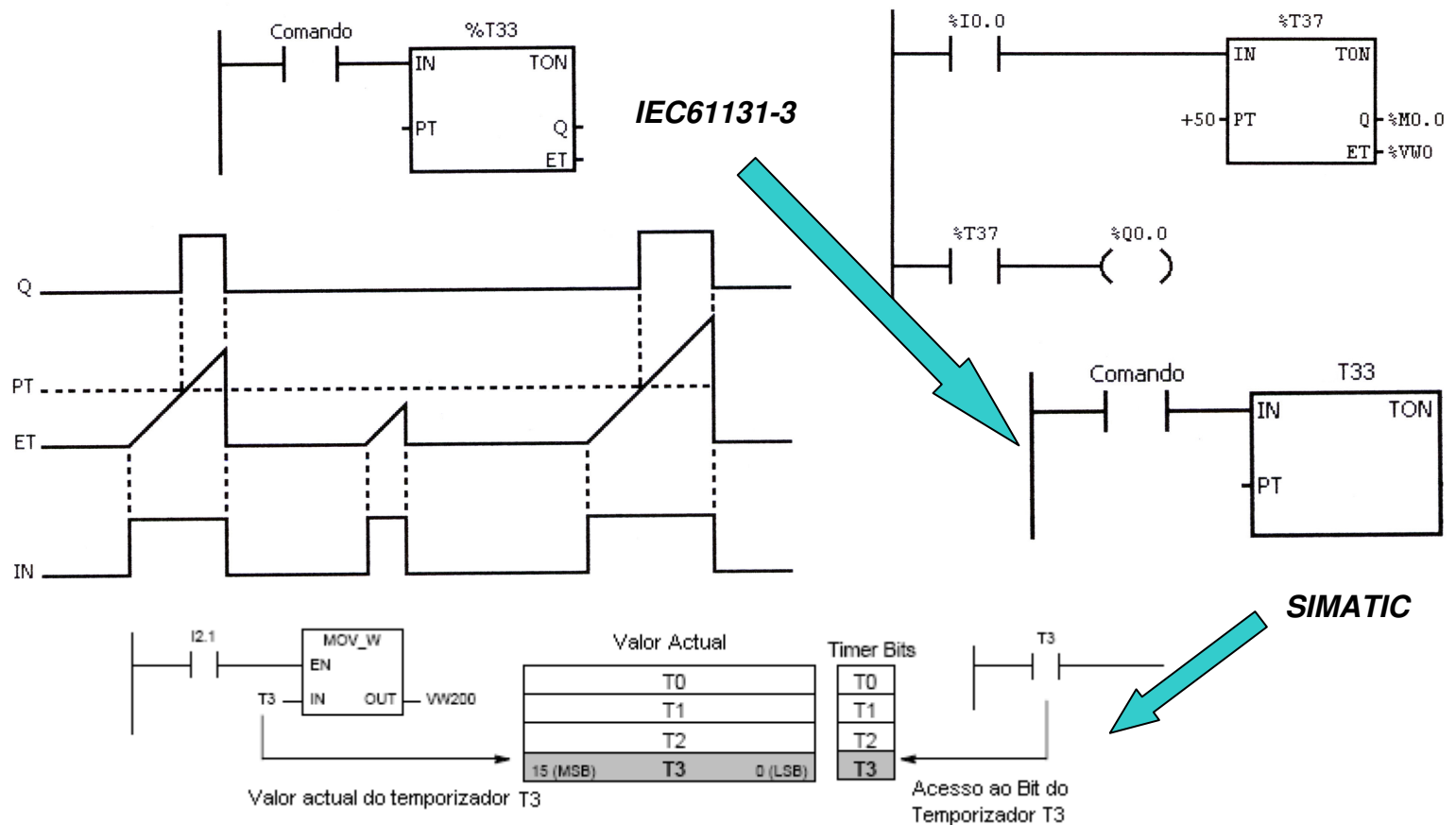
Tipo de temporizador	Base de tempo	Valor máximo (15 bits)	Nº do temporizador
TONR (total de 64)	1 ms	32,767 s	T0 e T64
	10 ms	327,67 s	T1 a T4, T65 a T68
	100 ms	3276,7 s	T5 a T31, T69 a T95
TON, TOF, TP (total de 192)	1 ms	32,767 s	T32 e T96
	10 ms	327,67 s	T33 a T36, T97 a T100
	100 ms	3276,7 s	T37 a T63, T101 a T255

- **Determinação do pre-set time (PT), em função da base de tempo (BT)**
- $PT = \text{Tempo [ms]} / BT$; Para, por exemplo, 3 segundos, ficará:
 - $PT = 3000/1=3000$ para BT de 1 milisegundo
 - $PT = 3000/10=300$ para BT de 10 milisegundos
 - $PT = 3000/100=30$ para BT de 100 milisegundos

Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
- **TON** – Temporizador com atraso ao trabalho

Quando a contagem de tempo (ET) for maior ou igual ao preset time (PT) o bit de saída fica activo.
Se o tempo de activação da entrada for menor que ET, o bit de saída não chega a ficar activo

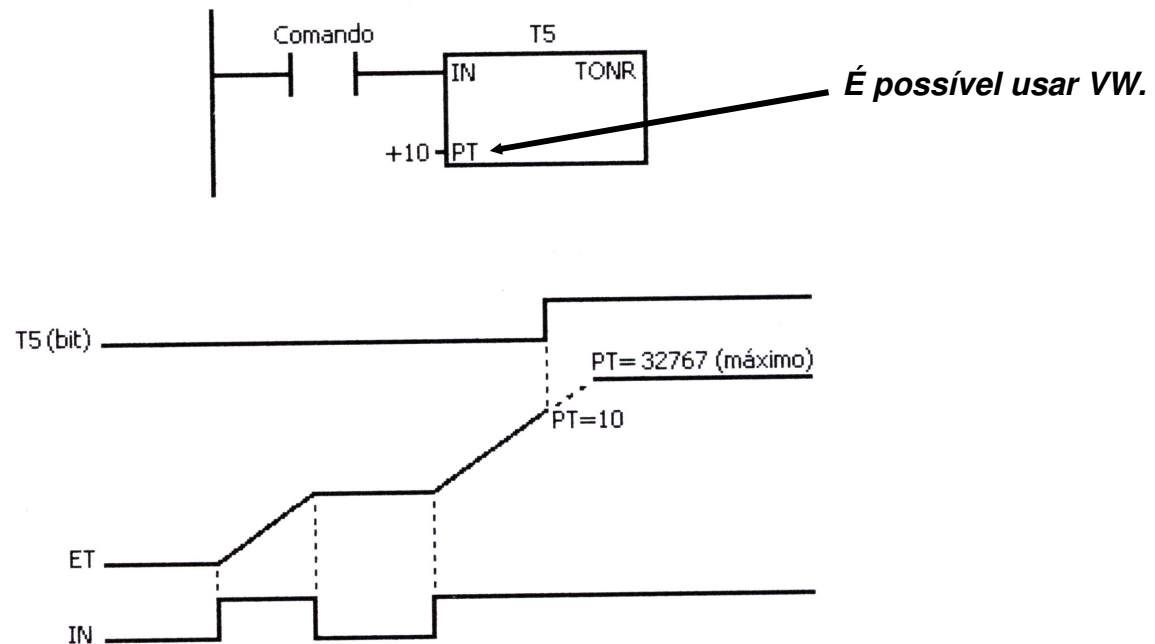


Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
 - **TONR** – Temporizador memorizado com atraso ao trabalho

Quando a entrada está activa o temporizador conta tempo, mas quando a entrada fica inactiva, a contagem não volta a zero, como no temporizador TON. O temporizador retém o tempo de contagem.

Para levar a contagem novamente a zero é necessário sujeitar o temporizador a uma instrução de reset.

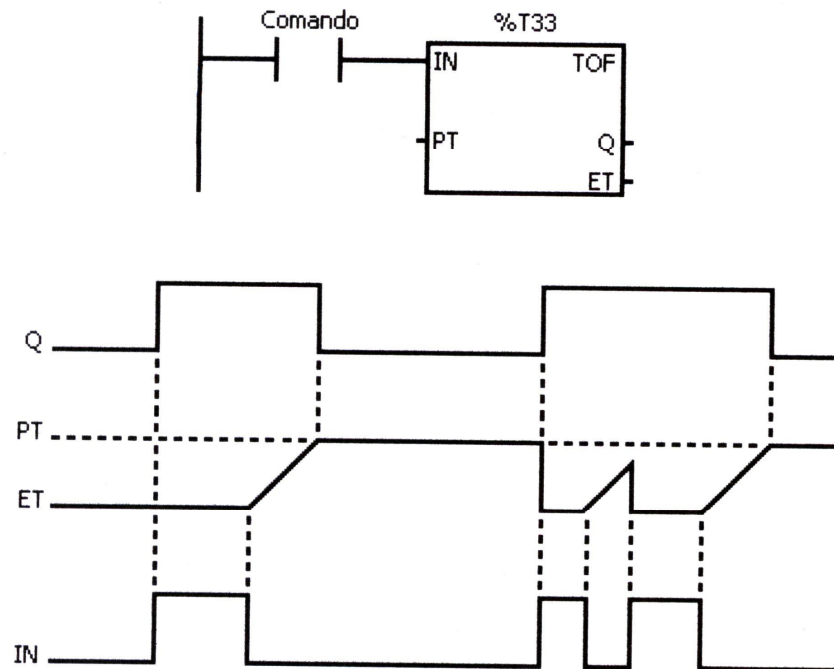


Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
 - **TOF** – Temporizador com atraso ao repouso

Quando a entrada está activa o bit de saída do temporizador fica instantaneamente activo e quando a entrada fica inactiva, é iniciada a contagem de tempo.

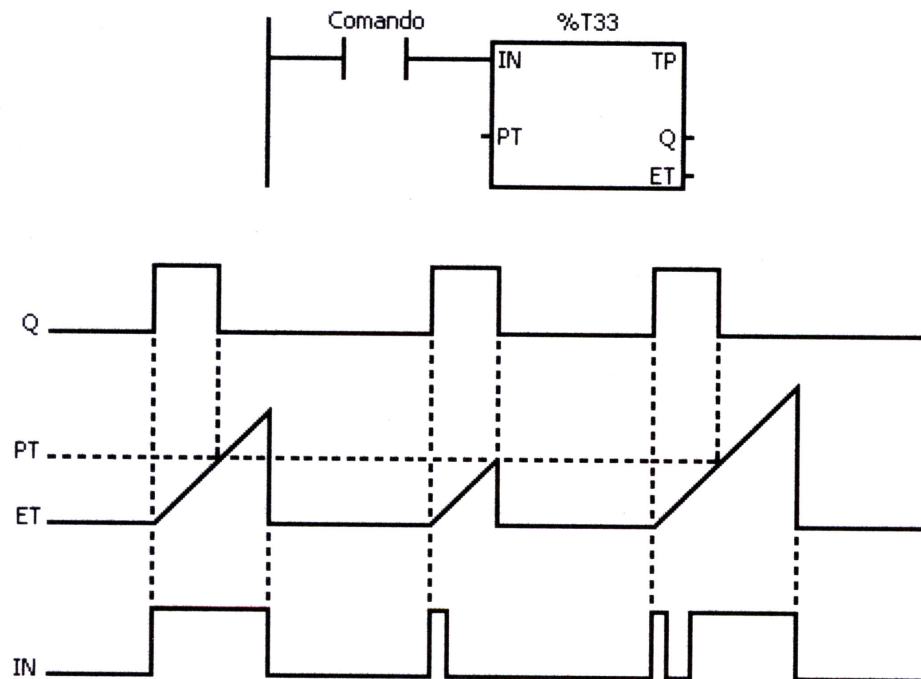
No fim desta contagem o bit de saída é levado a zero.



Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
 - **TP** – Temporizador por impulsos

Neste tipo de temporizador, independentemente do tempo de activação da entrada, quer seja maior ou menor que PT, a saída está activa sempre com o mesmo intervalo de tempo.





Lógica programável

- **Programação em LD (segundo Step 7 Micro/Win).**
 - **Exercícios com temporizadores**
 1. Implemente um TON que fique activo ao fim de 10 seg se a entrada I0.0 estiver activa, mas que fique activo ao fim de 20 seg se I0.1 estiver activa;
 2. Considere um motor equipado com ventilador (ventilação forçada) que se liga em simultâneo com o motor, mas que permanece activo durante 30 seg após se desligar o motor. Programe o temporizador;
 3. Admita uma campainha que toca enquanto se tiver o dedo no botão de pressão respectivo, mas com um máximo de 5 seg. Programe um temporizador para este efeito;
 4. Suponha que um tanque de mistura é cheio com vários componentes, um dos quais durante 10 seg. Quer o enchimento se faça de modo contínuo quer haja interrupções (por exemplo devido a uma paragem de emergência) programe um temporizador para obter sempre este tempo;
 5. Um sistema de alarme tem uma sinalização sonora intermitente, ou seja está activa durante 2 minutos, inactiva por 3 minutos e assim sucessivamente até se desligar o alarme. Implemente este funcionamento com temporizadores.

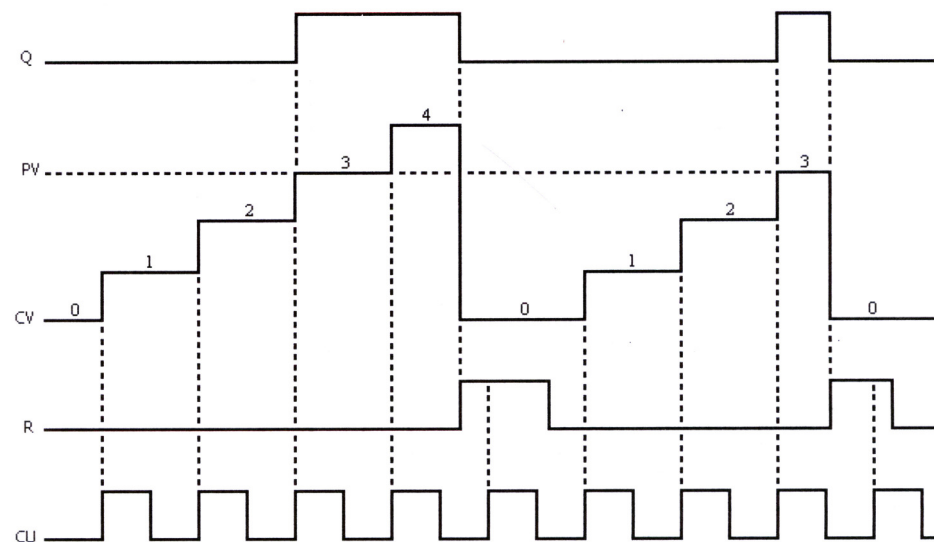
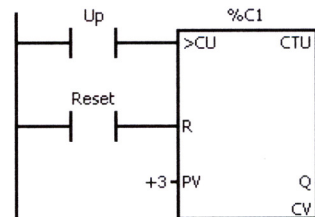


Lógica programável

- **Programação em LD (segundo Step 7 Micro/Win).**
- **Contadores (Counters)**
 - **Tipos de Contadores**
 - **CTU** – Contador Ascendente;
 - **CTD** – Contador Descendente;
 - **CTUD** – Contador Ascendente/Descendente.
 - **Características**
 - **PV (Preset value)** – entre 0 e 32767 e programado pelo utilizador;
 - **CV (Current value)** – entre 0 e 32767; indica o valor actual da contagem;
 - **CU** – Incrementa o contador no flanco ascendente;
 - **CD** – Decrementa o contador na flanco descendente;
 - **R (Reset)** – Coloca o conteúdo do contador a zero;
 - **LD (Load)** – Coloca o conteúdo do contador com o valor de PV;
 - **Q (Output)** – Saída do contador Ascendente ou Descendente;
 - **QU e QD** – Saídas do contador Ascendente/Descendente;

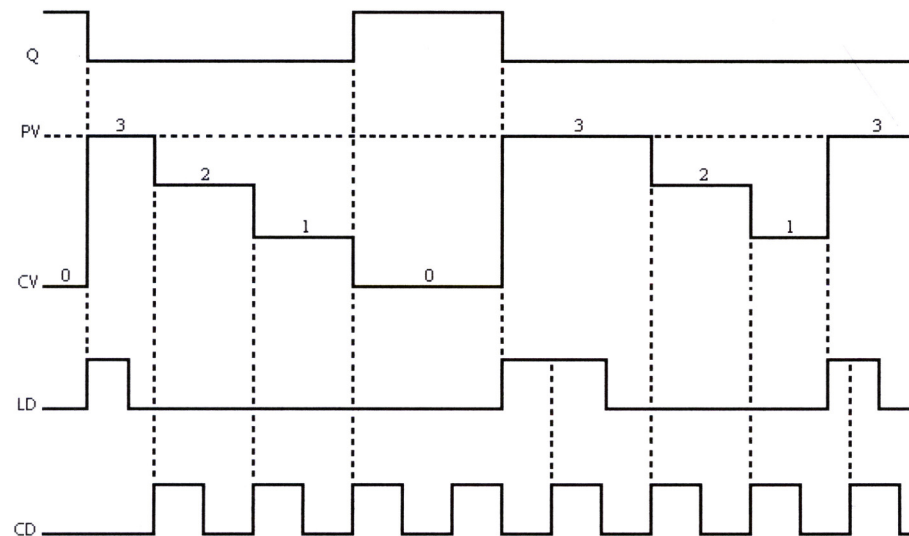
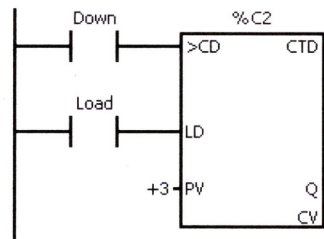
Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
 - CTU – Contador Ascendente



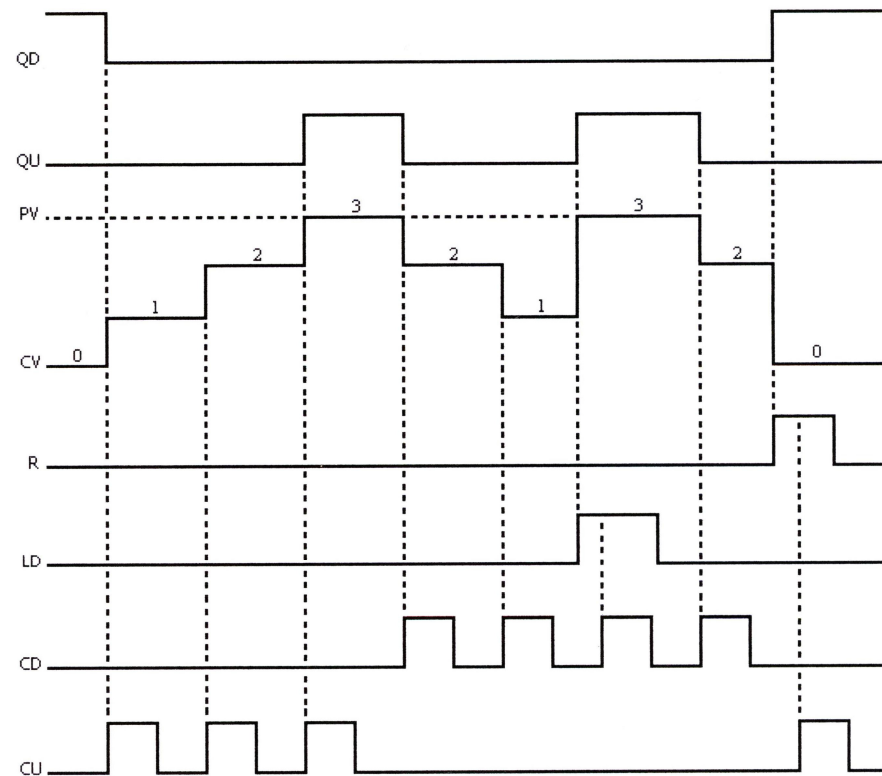
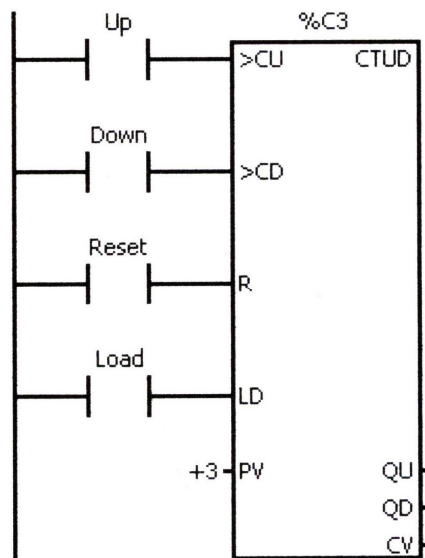
Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
- **CTD** – Contador Descendente



Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
 - CTUD – Contador Ascendente/Descendente

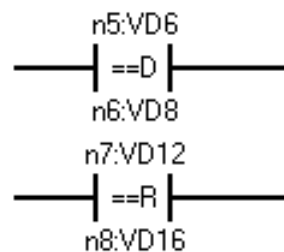
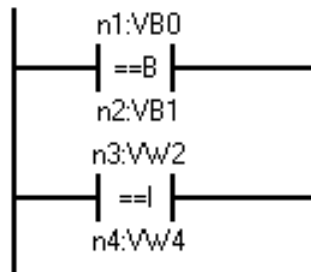


Lógica programável

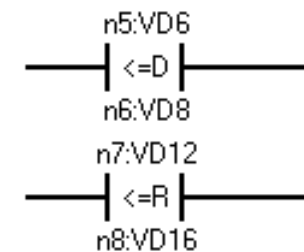
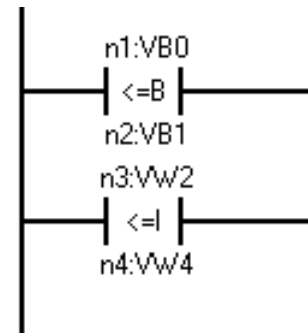
- **Programação em LD (segundo Step 7 Micro/Win).**

- **Instruções de Comparação**

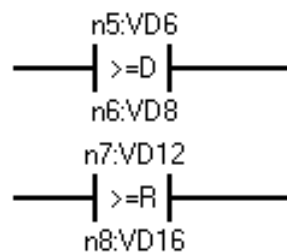
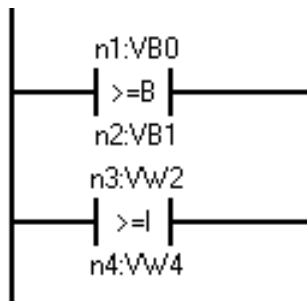
O contacto fecha quando $n1=n2$



O contacto fecha quando $n1 \leq n2$



O contacto fecha quando $n1 \geq n2$



- **B** = Byte
- **I** = Inteiro (ou Word)
- **D** = Duplo Inteiro (double Word)
- **R** = Real



Lógica programável

- **Programação em LD (segundo Step 7 Micro/Win).**

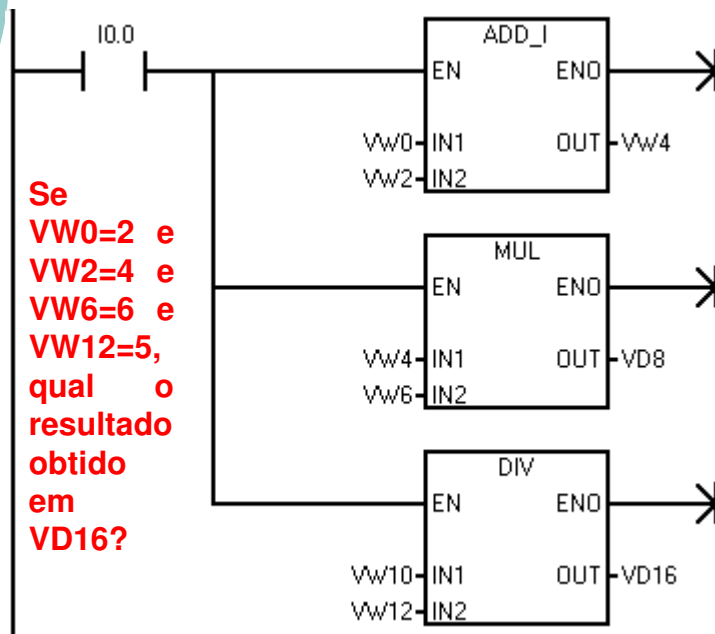
- **Exercícios**

1. **Implemente um sistema de contagem de peças, que por cada 12 peças active uma sinalização luminosa durante 2 segundos e que fique pronto para contar novas peças. Estas 12 peças são colocadas em caixas e por cada 6 caixas existe outra sinalização luminosa que é activada durante 5 segundos.**
2. **Programe um sistema para manutenção que activa uma sinalização luminosa caso a máquina supervisionada atinga as 10000 (dez mil) horas de funcionamento e que só desactiva a sinalização após a manutenção feita.**
3. **Implemente o sistema de alarme intermitente anteriormente visto (2 minutos a tocar, 3 minutos de silencio), mas agora só com um temporizador e um comparador.**

Lógica programável

■ Programação em LD (segundo Step 7 Micro/Win).

■ Operações aritméticas com inteiros



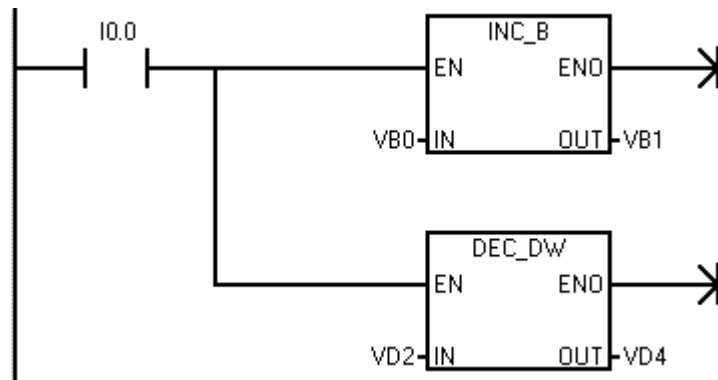
Bits afectados:

- SM1.0 – Resultado Nulo;
- SM1.1 - Overflow;
- SM1.2 – Resultado negativo;
- SM1.3 - Divisão por zero.

- **ADD_I** - Adiciona 2 Inteiros e obtém 1 inteiro (IN1+IN2=OUT)
- **ADD_DI** - Adiciona 2 inteiros duplos e obtém 1 inteiro duplo
- **SUB_I** - Subtrai 2 Inteiros e obtém 1 inteiro
- **SUB_DI** - Subtrai 2 inteiros duplos e obtém 1 inteiro duplo
- **MUL** - Multiplica 2 inteiros e obtém 1 inteiro duplo
- **MUL_I** – Multiplica 2 inteiros e obtém 1 inteiro
- **MUL_DI** – Multiplica 2 inteiros duplos e obtém 1 inteiro duplo
- **DIV** – Divide 2 inteiros e obtém 1 inteiro duplo, onde o inteiro mais significativo é resto e o menos significativo é o quociente
- **DIV_I** – Divide 2 inteiros e obtém 1 inteiro. O resto não é guardado
- **DIV_DI** – Divide 2 inteiros duplos e obtém 1 inteiro duplo. O resto não é guardado

Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
 - Operações de incremento/decremento



- **INC_B** - Incrementar Byte (ex: IN+1=OUT)
- **INC_W** - Incrementar Word (ou inteiro)
- **INC_DW** - Incrementar Double Word (ou inteiro duplo)
- **DEC_B** - Decrementar Byte (ex: IN-1=OUT)
- **DEC_W** - Decrementar Word (ex:)
- **DEC_DW** (Decrementar Double Word)

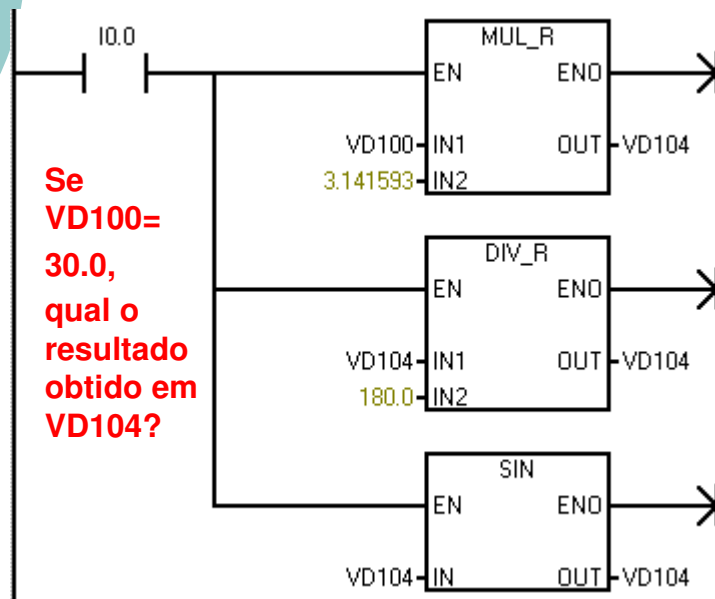
Bits afectados:

- SM1.0 - Resultado a Zero;
- SM1.1 - Overflow

Lógica programável

■ Programação em LD (segundo Step 7 Micro/Win).

■ Operações aritméticas com reais



Bits afectados:

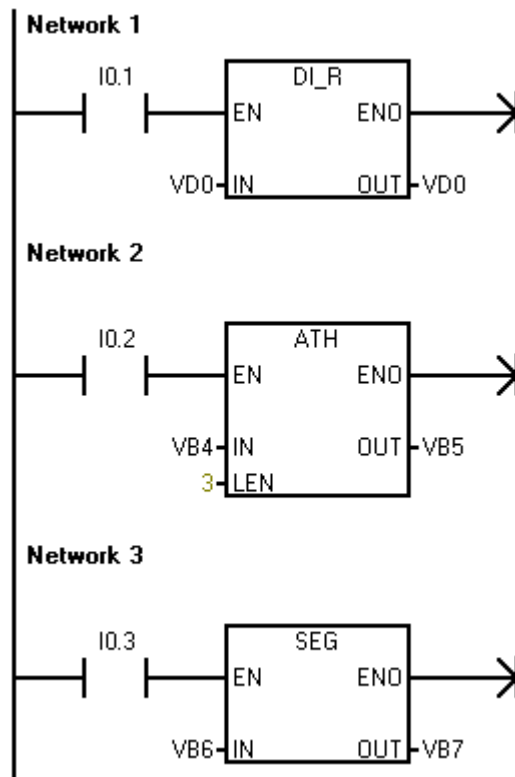
- SM1.0 – Resultado Nulo;
- SM1.1 - Overflow;
- SM1.2 – Resultado negativo;
- SM1.3 - Divisão por zero.

- **ADD_R** – Adição de reais
- **SUB_R** – Subtracção de reais
- **MUL_R** - Multiplicação de reais
- **DIV_R** - Divisão de reais
- **SQRT** – Obtem a raiz quadrada da entrada
- **SIN** - Obtem o Seno da entrada, que é dada em radianos
- **COS** - Obtem o Coseno da entrada que é dada em radianos
- **TAN** - Obtem a tangente da entrada que é dada em radianos
- **LN** - Obtem o logaritmo da entrada
- **EXP** - Obtem o valor de e elevado à entrada. Para se elevar um número real a outro número real é necessário combinar a função EXP com a função LN. Por exemplo para se obter 2 elevado ao cubo será: $2^3 = \text{EXP}(3 * \text{LN}(2))$. (Atenção: O valor é aproximado, não é exactamente 8.)

Lógica programável

■ Programação em LD (segundo Step 7 Micro/Win).

■ Operações de Conversão



Bits afetados:

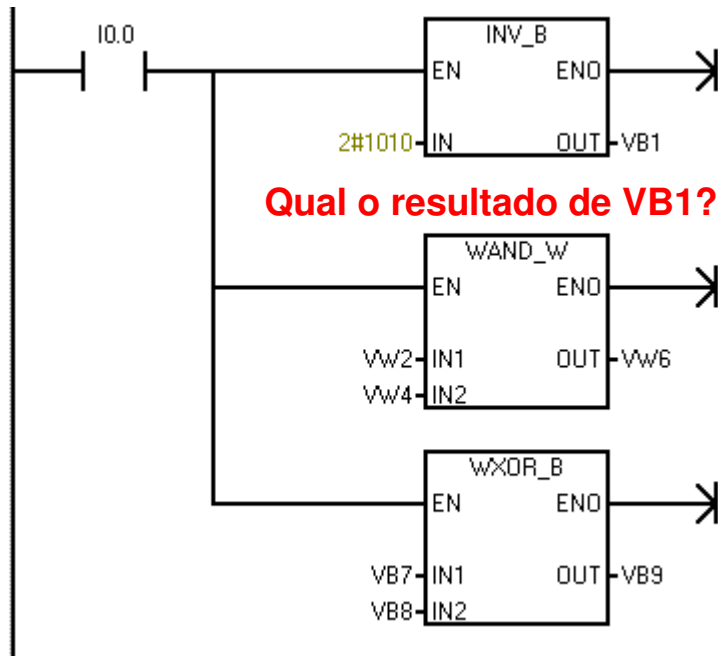
- SM1.1 (Overflow).

- **B_I** – Byte para Inteiro
- **I_B; I_DI; I_S** – Inteiro para Byte, (se ≤ 255); para Inteiro Duplo; para String
- **DI_I; DI_R; DI_S** – Inteiro duplo para Inteiro (se ≤ 65535); para Real; para String
- **BCD_I** - BCD para Inteiro
- **I_BCD** - Inteiro para BCD
- **ROUND** – Real para duplo inteiro com arredondamento
- **TRUNC** – Real para duplo inteiro, descartando a parte fracionária
- **R_S** – Real para string
- **ITA; DTA; RTA** – Inteiro para ASCII; Duplo Inteiro para ASCII; Real Para ASCII
- **ATH** – ASCII para HEX
- **HTA** – HEX para ASCII
- **S_I; S_DI; S_R** – String para Inteiro; para Duplo Inteiro; para Real
- **SEG** -Descodificador de sete segmentos

Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).

- Operações lógicas



Qual o resultado de VB1?

- **INV_B** – Complemento(ou inverso) do byte
- **INV_W** – Complemento da word
- **INV_DW** – Complemento da double word
- **WAND_B** – AND Byte (bit a bit)
- **WAND_W** - AND Word
- **WAND_DW** - AND Double Word
- **WOR_B** – OR Byte
- **WOR_W** - OR Word
- **WOR_DW** - OR Double Word
- **WXOR_B** – XOR Byte
- **WXOR_W** - XOR Word
- **WXOR_DW** - XOR Double Word

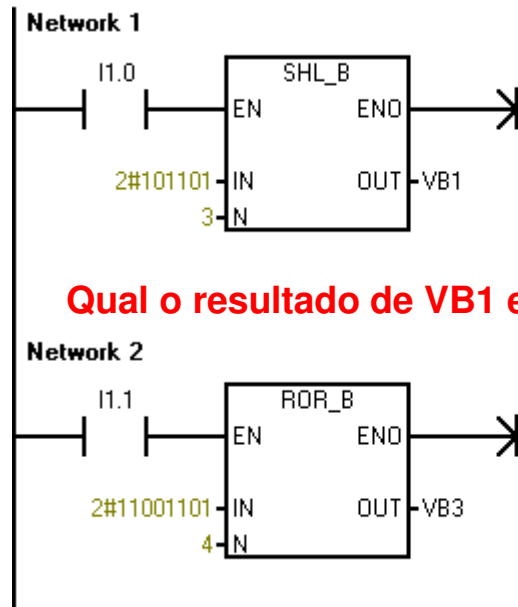
Bits afectados:

- **SM1.0** – Resultado Nulo

Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).

- Operações de rotação e deslocamento de bits



Qual o resultado de VB1 e de VB3?

Bits afectados:

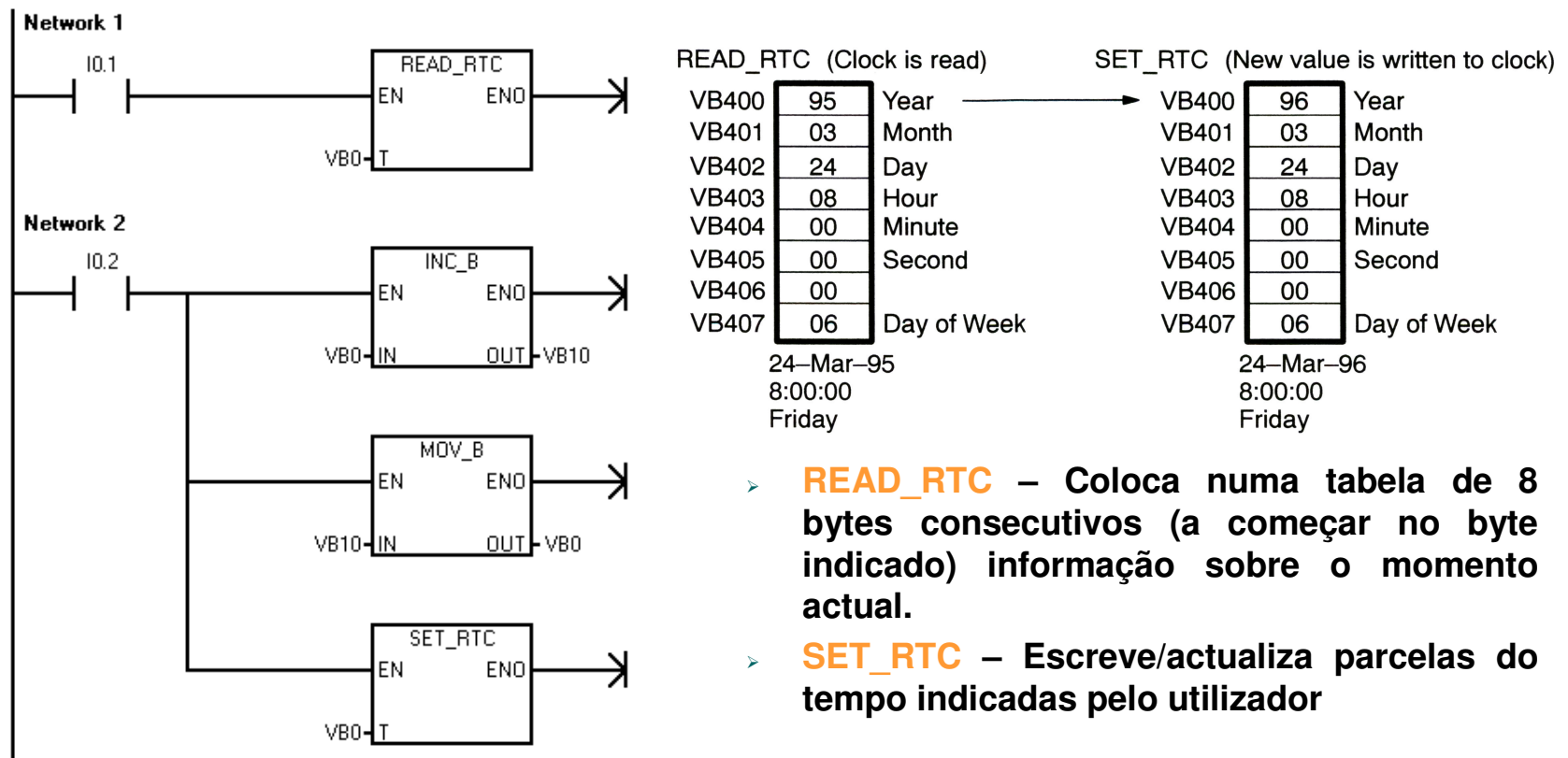
- SM1.0 – Resultado Nulo
- SM1.1 - Overflow

- **SHL_B; SHL_W; SHL_DW** – Desloca para a esquerda N bits do byte; da word; da double_word. Os N bits à esquerda são esmagados e as N novas posições de bits à direita são preenchidas a 0
- **SHR_B; SHR_W; SHR_DW** – Desloca para a direita N bits do byte; da word; da double word
- **ROL_B; ROL_W; ROL_DW** – Roda para a esquerda N bits do byte; da Word; da Double Word. Os N bits À esquerda são reescritos nas N novas posições à direita
- **ROR_B; ROR_W; ROR_DW** – Roda para a direita N bits do byte; da word; da Double_Word

Lógica programável

■ Programação em LD (segundo Step 7 Micro/Win).

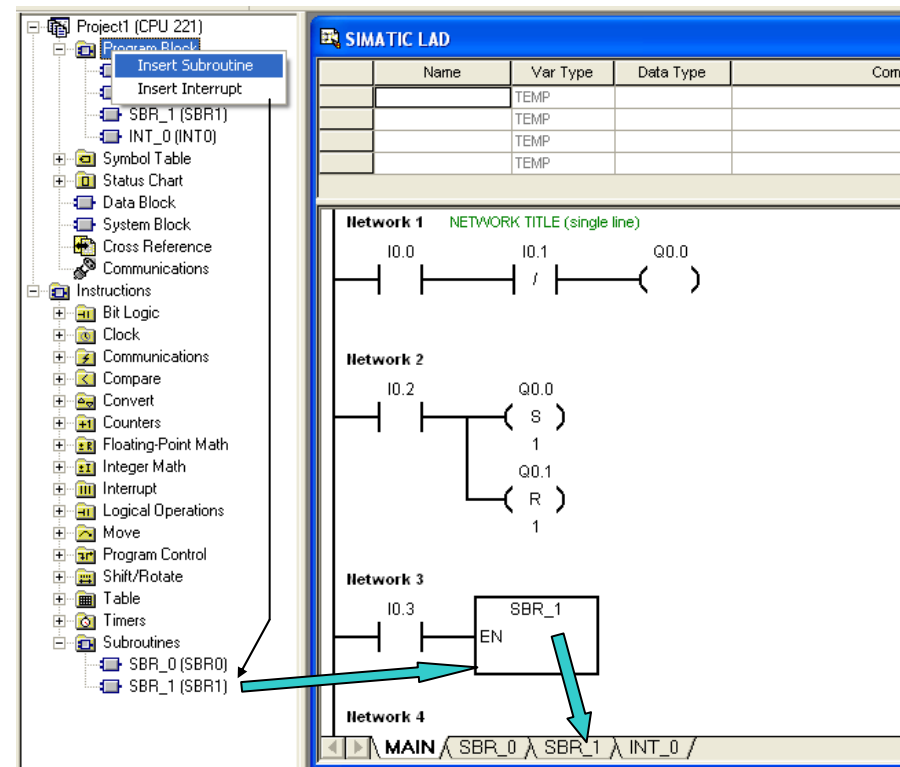
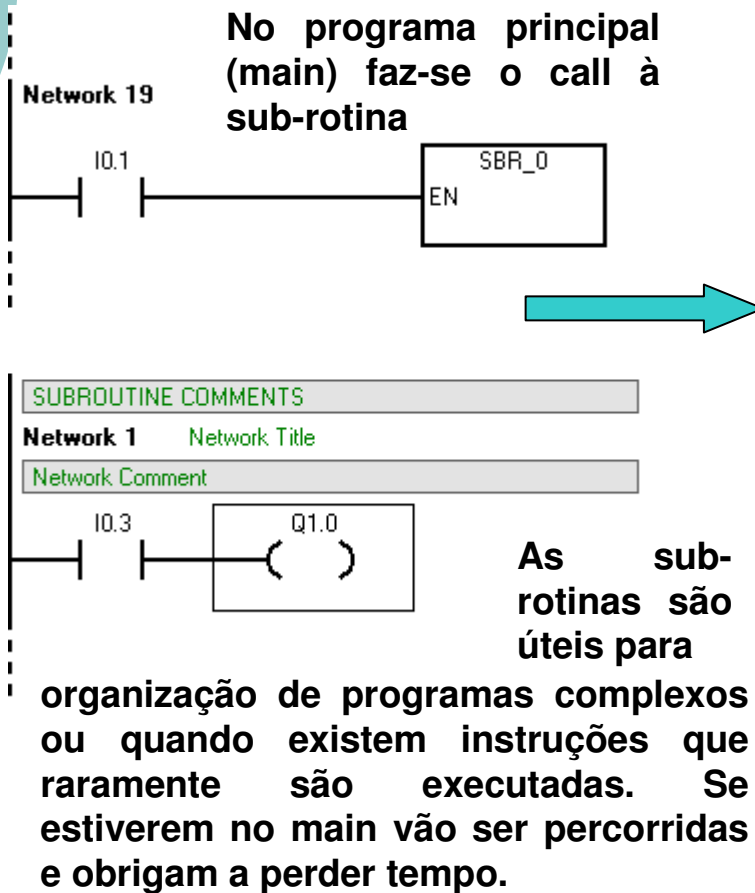
■ Instruções de relógio em tempo real



Na primeira segunda-feira de todos os meses actue uma saída durante 30 segundos.

Lógica programável

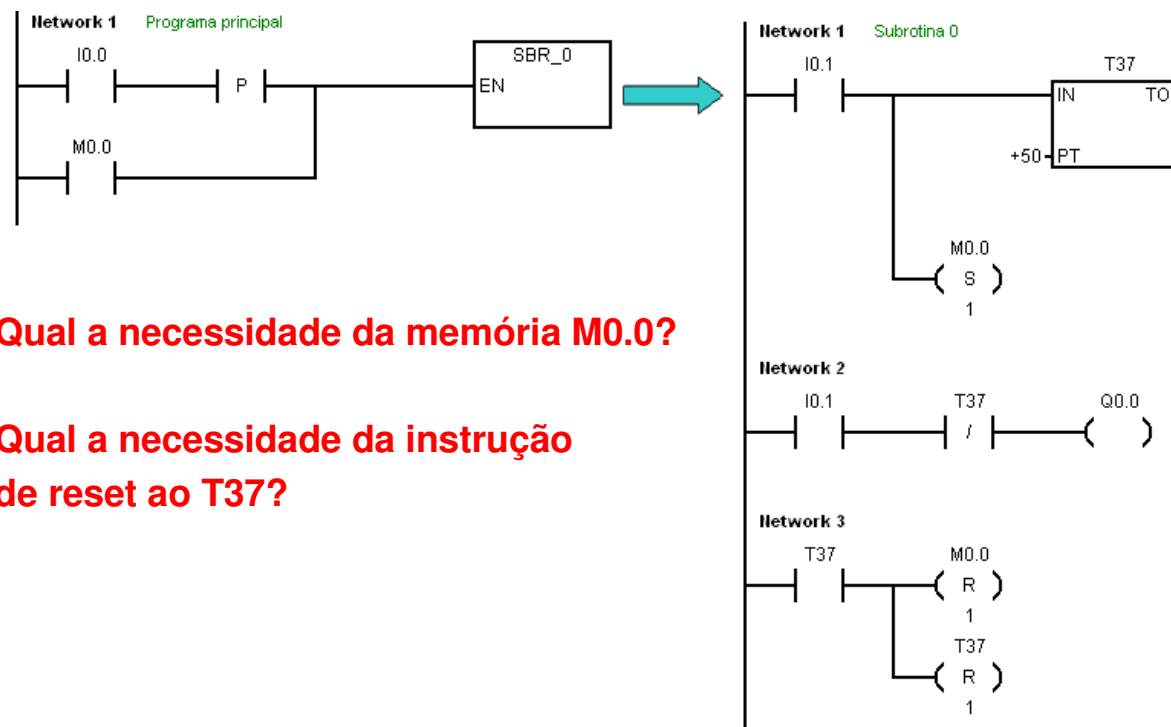
- Programação em LD (segundo Step 7 Micro/Win).
- Instruções de subrotinas



Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).

- Subrotinas



Qual a necessidade da memória M0.0?

Qual a necessidade da instrução de reset ao T37?

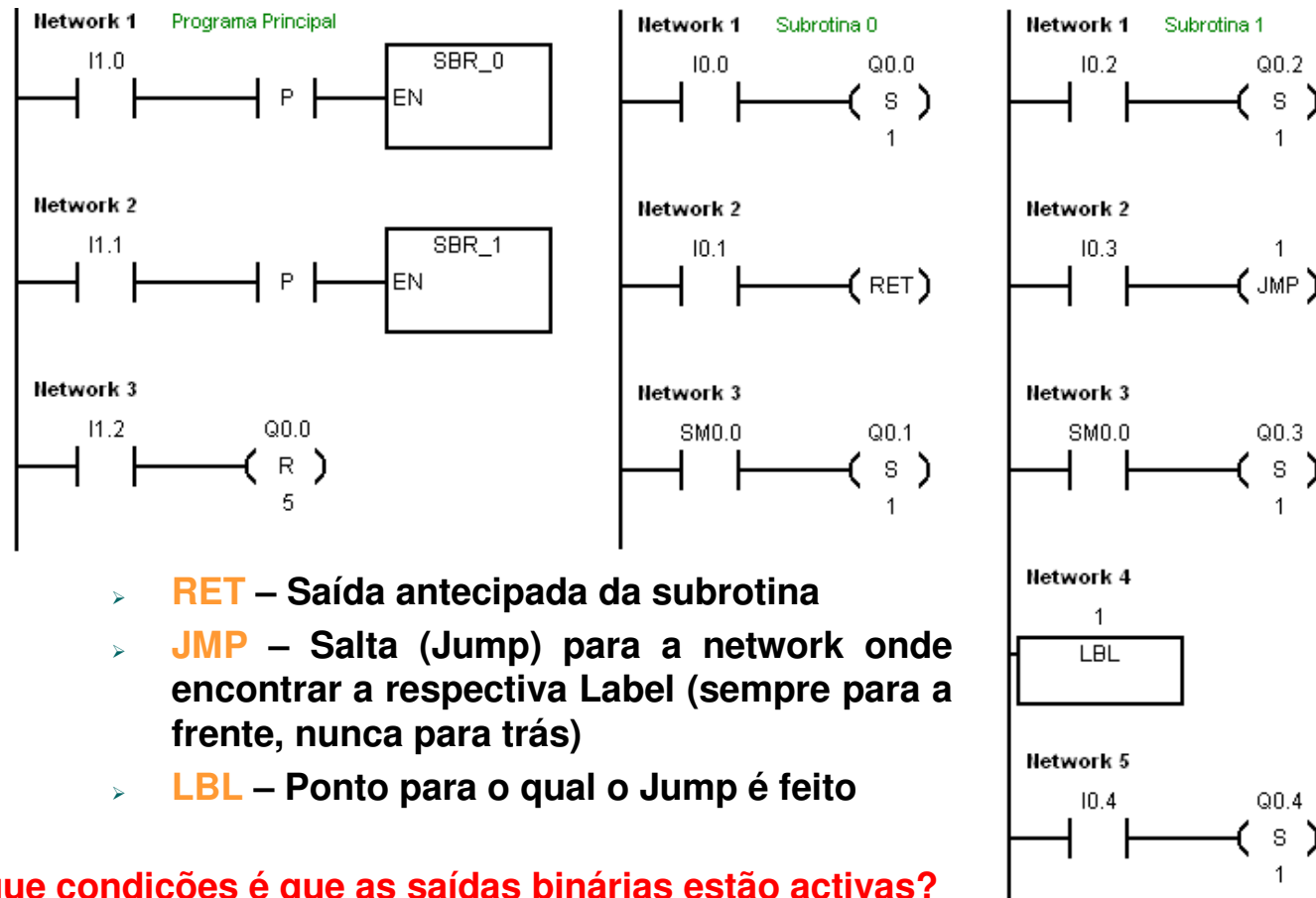


As subrotinas são executadas, normalmente, apenas durante um ciclo de varrimento e algumas variáveis (saídas binárias, memórias, contadores e temporizadores) poderão não funcionar como esperado!

Lógica programável

■ Programação em LD (segundo Step 7 Micro/Win).

■ Subrotinas



- **RET** – Saída antecipada da subrotina
- **JMP** – Salta (Jump) para a network onde encontrar a respectiva Label (sempre para a frente, nunca para trás)
- **LBL** – Ponto para o qual o Jump é feito

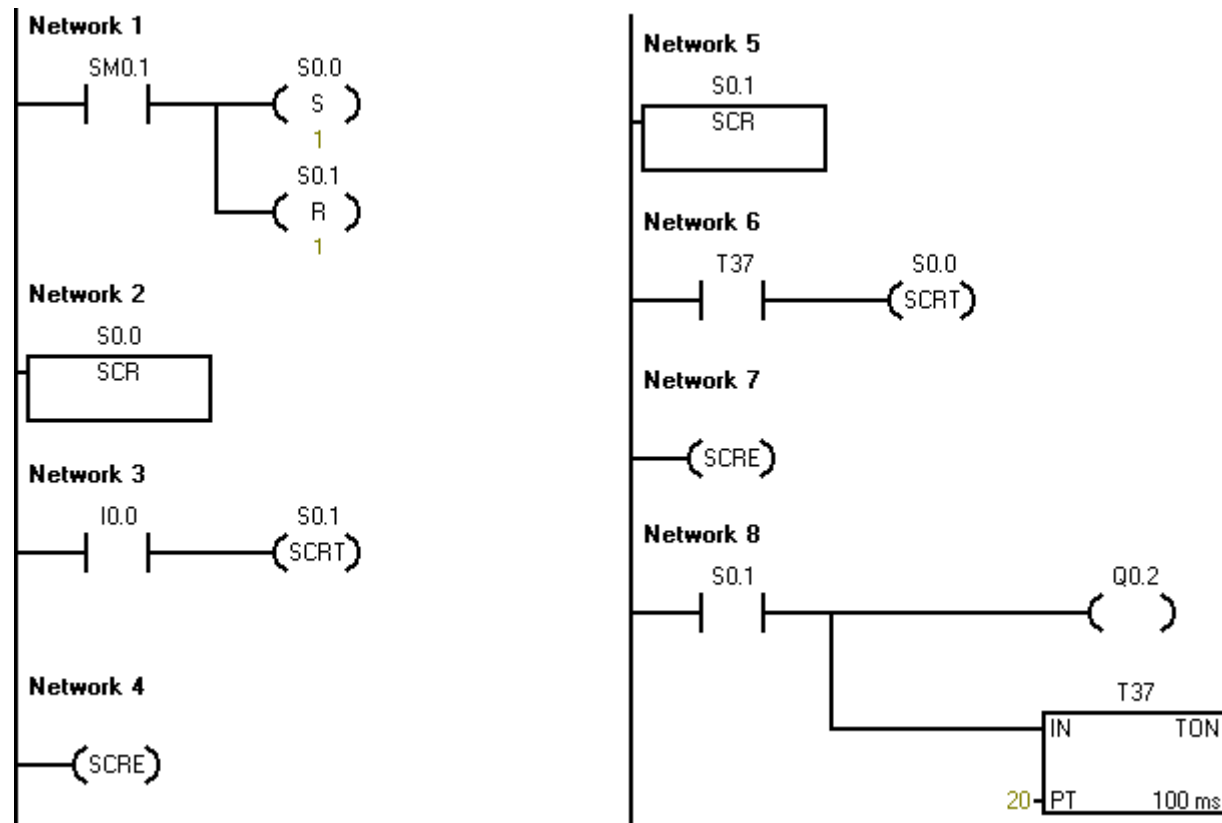
Em que condições é que as saídas binárias estão activas?

Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
 - Utilização de Instruções de controlo sequencial – Permite utilizar a linguagem de Diagramas de Contactos para produzir um efeito semelhante aos Diagramas funcionais sequenciais (GRAFCET).

- Exemplo:

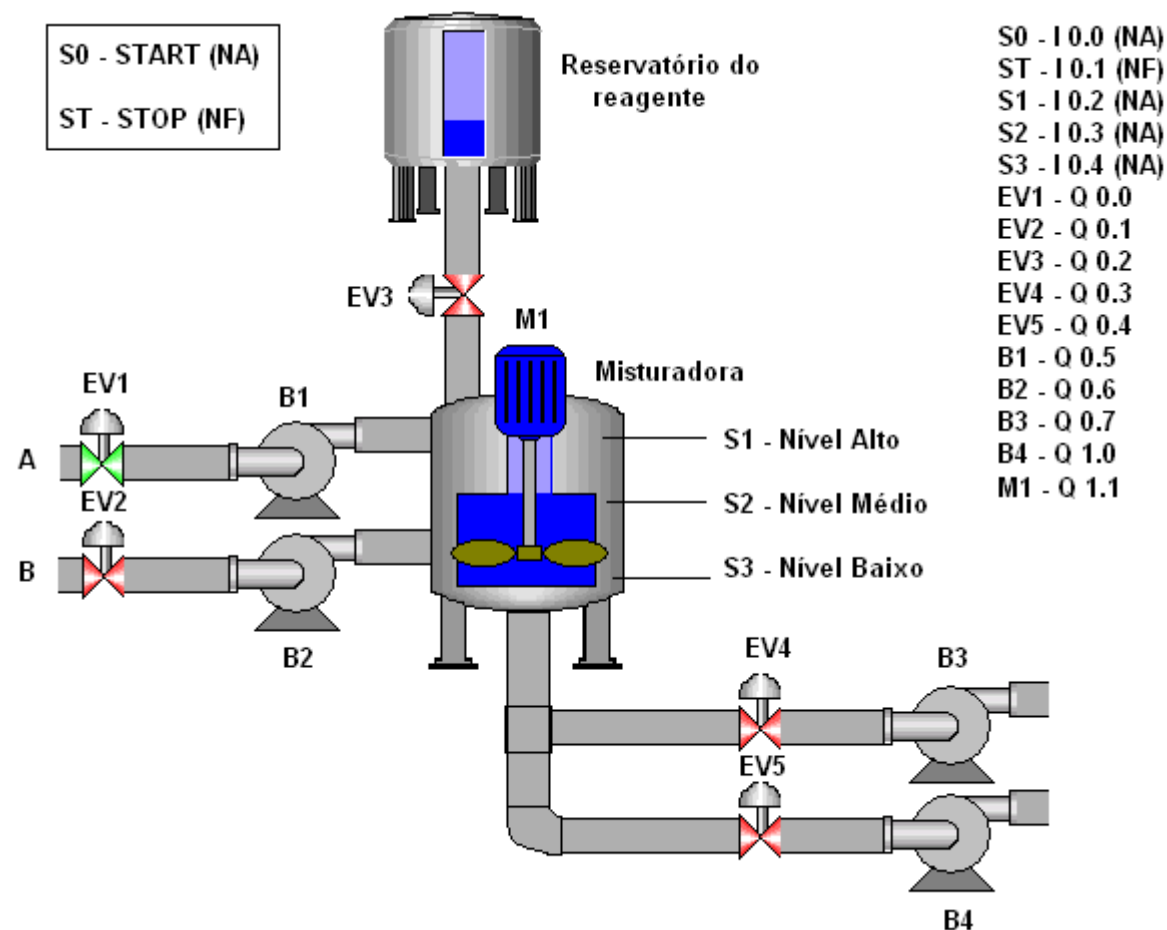
- **SCR** – Sequence Control Relay. Activa a etapa
- **SCRT** – Sequence Control Relay. Transita de etapa, activando a próxima.
- **SCRE** – Sequence Control Relay. Fecha a etapa.



Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).

- Exercício de Aplicação I





Lógica programável

- **Programação em LD (segundo Step 7 Micro/Win).**
 - **Exercício de Aplicação I - Descrição**
 - O botão de pressão S0 dá início ao processo começando por abrir EV1 e arrancar B1. O enchimento da misturadora com o líquido A é feita até ao nível S2 (Nível médio).
 - Quando o nível S2 é atingido dá-se o enchimento final da misturadora com o líquido B através de EV2 e B2. O enchimento final termina no nível S1.
 - Depois da misturadora cheia, EV3 vai adicionar reagente durante 10 seg.
 - Finalizado este processo o motor M1 vai agitar convenientemente a mistura durante 1 minuto.

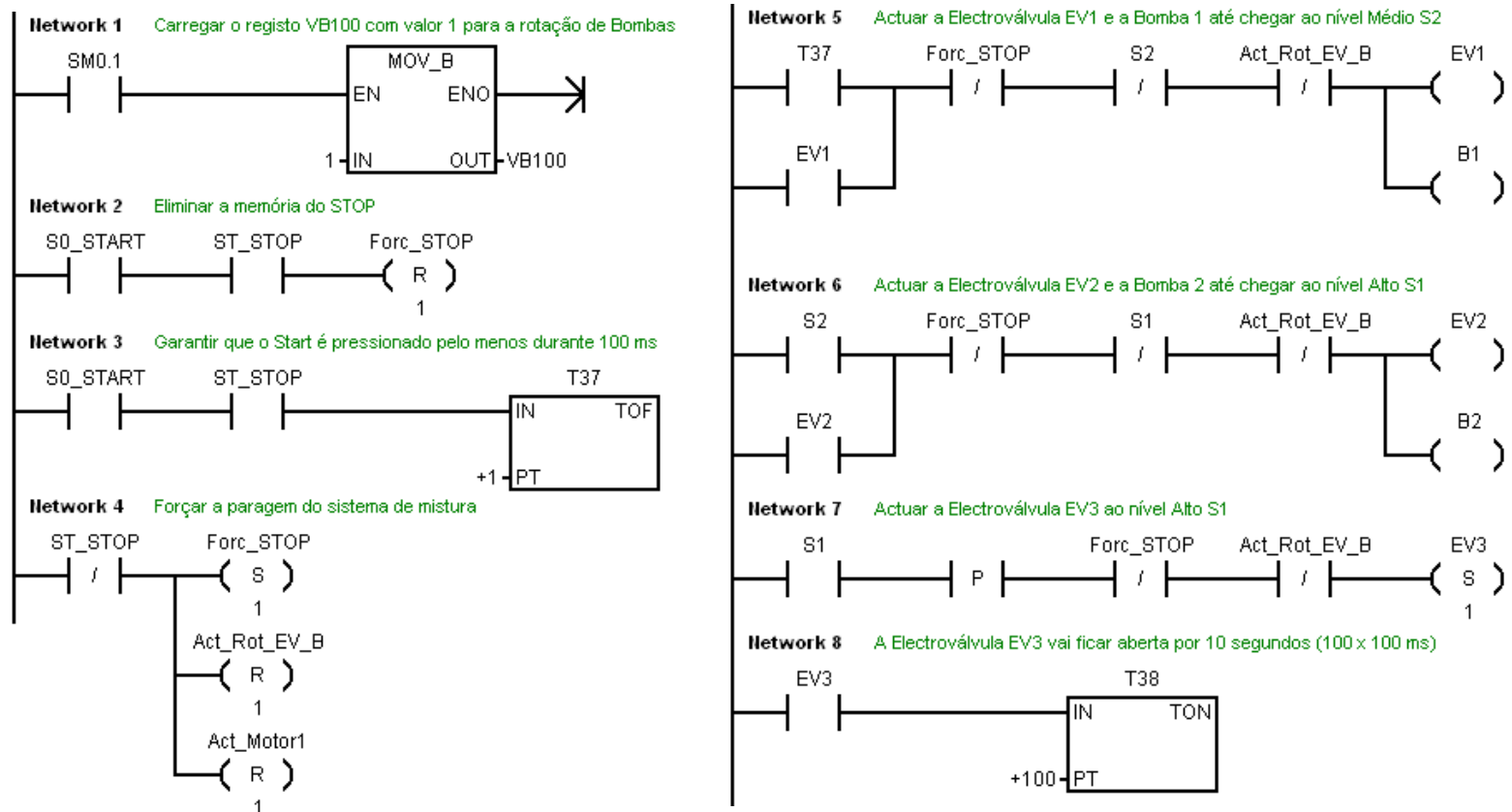


Lógica programável

- **Programação em LD (segundo Step 7 Micro/Win).**
 - **Exercício de Aplicação I – Descrição (continuação)**
 - Decorrido este tempo, uma das válvulas EV4 ou EV5 e Bombas B3 ou B4 vão funcionar e drenar o líquido do reservatório. Estas válvulas e bombas funcionam em regime de rotação. Dá-se prioridade inicial a EV4 e B3.
 - O processo termina com o esvaziamento da misturadora (S3) e só volta a começar pressionando o botão de pressão S0.
 - O botão ST pára o processo em qualquer altura.
 - Deverá ainda existir um contador para o número de vezes que o processo foi efectuado.

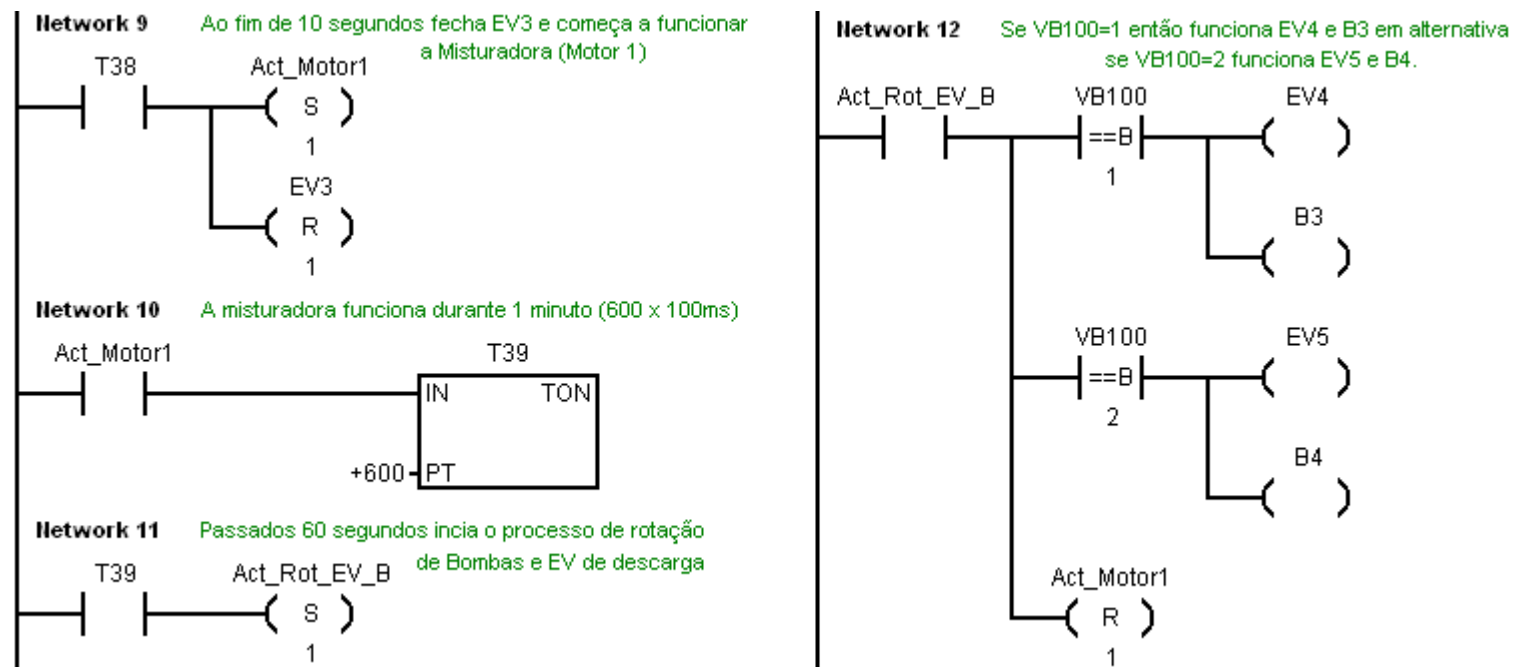
Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
- Exercício de Aplicação I – Solução (SIMATIC)



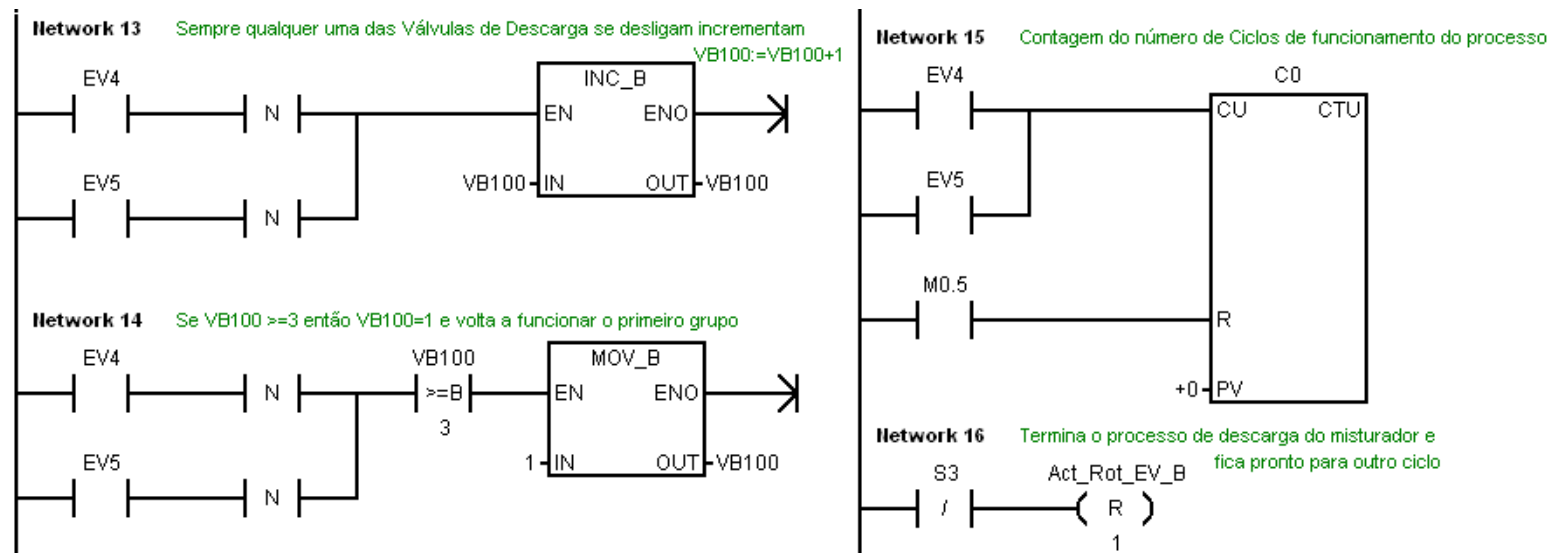
Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
- Exercício de Aplicação I – Solução (SIMATIC)



Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
 - Exercício de Aplicação I – Solução (SIMATIC)



Lógica programável

- Programação em LD (segundo Step 7 Micro/Win).
- Exercício de Aplicação I – Solução (SIMATIC)

Tabela de símbolos ou mnemónicas

	Name	Address	Comment
1	S0_START	I0.0	Comando de START
2	ST_STOP	I0.1	Comando de STOP
3	Act_Motor1	Q1.1	Actuar Motor M1 (Misturadora)
4	Forc_STOP	M0.2	Forçar o STOP em todo o programa
5	S1	I0.2	Nível Alto do Tq da Misturadora
6	S2	I0.3	Nível Médio do Tq da Misturadora
7	S3	I0.4	Nível Baixo do Tq da Misturadora
8	EV1	Q0.0	Actuar electroválvula EV1
9	EV2	Q0.1	Actuar electroválvula EV2
10	EV3	Q0.2	Actuar electroválvula EV3
11	EV4	Q0.3	Actuar electroválvula EV4
12	EV5	Q0.4	Actuar electroválvula EV5
13	B1	Q0.5	Actuar Bomba 1
14	B2	Q0.6	Actuar Bomba 2
15	B3	Q0.7	Actuar Bomba 3
16	B4	Q1.0	Actuar Bomba 4
17		M0.3	Actuar a Rotação de bombas e de Electroválvulas de Descar-

Tabela de referências cruzadas

	Element	Block	Location	Context
1	S0_START	MAIN (OB1)	Network 2	- -
2	S0_START	MAIN (OB1)	Network 3	- -
3	ST_STOP	MAIN (OB1)	Network 2	- -
4	ST_STOP	MAIN (OB1)	Network 3	- -
5	ST_STOP	MAIN (OB1)	Network 4	- -
6	S1	MAIN (OB1)	Network 6	- -
7	S1	MAIN (OB1)	Network 7	- -
8	S2	MAIN (OB1)	Network 5	- -
9	S2	MAIN (OB1)	Network 6	- -
10	S3	MAIN (OB1)	Network 16	- -
11	EV1	MAIN (OB1)	Network 5	-()
12	EV1	MAIN (OB1)	Network 5	- -
13	EV2	MAIN (OB1)	Network 6	-()
14	EV2	MAIN (OB1)	Network 6	- -
15	EV3	MAIN (OB1)	Network 7	-(S)
16	EV3	MAIN (OB1)	Network 8	- -
17	EV3	MAIN (OB1)	Network 9	-(R)
18	EV4	MAIN (OB1)	Network 12	-()

Lógica programável

- Programação em IL (segundo Step 7 Micro/Win).
- Exercício de Aplicação I – Solução (SIMATIC)

```
NETWORK 1      //Carregar o registo VB100
LD      SMO.1   com valor 1 para a rotação de Bombas
MOVB    1, VB100
NETWORK 2      //Eliminar a memória do STOP
LD      SO_START
A       ST_STOP
R       Forc_STOP, 1
NETWORK 3      //Garantir que o Start é pressionado
LD      SO_START           pelo menos durante 100 ms
A       ST_STOP
TOF     T37, +1
NETWORK 4      //Forçar a paragem do sistema
LDN     ST_STOP           de mistura
S       Forc_STOP, 1
R       Act_Rot_EV_B, 1
R       Act_Motor1, 1

NETWORK 5      //Actuar a Electroválvula EV1 e a
LD      T37           Bomba 1 até chegar ao nível Médio S2
O       EV1
AN      Forc_STOP
AN      S2
AN      Act_Rot_EV_B
=       EV1
=       B1
NETWORK 6      //Actuar a Electroválvula EV2 e a
//           Bomba 2 até chegar ao nível Alto S1
LD      S2
O       EV2
AN      Forc_STOP
AN      S1
AN      Act_Rot_EV_B
=       EV2
=       B2
```

Lógica programável

- Programação em IL (segundo Step 7 Micro/Win).
- Exercício de Aplicação I – Solução (SIMATIC)

```
NETWORK 7      //Actuar a Electroválvula EV3
LD      S1      ao nível Alto S1
EU
AN      Forc_STOP
AN      Act_Rot_EV_B
S      EV3, 1
NETWORK 8      //A Electroválvula EV3 vai ficar
LD      EV3      aberta por 10 segundos (10 x 100 ms)
TON      T38, +100
NETWORK 9      //Ao fim de 10 segundos fecha EV3 e
LD      T38      começa a funcionar a Misturadora (M1)
S      Act_Motor1, 1
R      EV3, 1
NETWORK 10     //A misturadora funciona durante
LD      Act_Motor1      1 minuto (600 x 100ms)
TON      T39, +600
```

```
NETWORK 11     //Passados 60 segundos incia o
LD      T39      processo de rotação de Bombas e EV
S      Act_Rot_EV_B, 1      de descarga
NETWORK 12     //Se VB100=1 então funciona
                        EV4 e B3 em alternativa se VB100=2
LD      Act_Rot_EV_B      funciona EV5 e B4.
LPS
AB=     VB100, 1
=      EV4
=      B3
LRD
AB=     VB100, 2
=      EV5
=      B4
LPP
R      Act_Motor1, 1
```

Lógica programável

■ Programação em IL (segundo Step 7 Micro/Win).

■ Exercício de Aplicação I – Solução (SIMATIC)

```
NETWORK 13      //Sempre qualquer uma das Válvulas de Descarga :
LD      EV4      desligam incrementam VB100:=VB100
ED
LD      EV5
ED
OLD
INCB      VB100

NETWORK 14      //Se VB100 >=3 então VB100=1 e volta a
//              funcionar o primeiro grupo
LD      EV4
ED
LD      EV5
ED
OLD
AB>=      VB100, 3
MOVB     1, VB100

NETWORK 15      //Contagem do número de Ciclos de
//              funcionamento do processo
LD      EV4
O        EV5
LD      MO.5
CTU      CO, +0

NETWORK 16      //Termina o processo de descarga do misturador
//              e fica pronto para outro ciclo
LDN      S3
R        Act_Rot_EV_B, 1
```