1. CONCEPÇÃO DE UM AUTOMATISMO

1.1. CARACTERÍSTICAS DE UM AUTOMATISMO

É já difícil imaginar a nossa vida sem automatismos. A sua presença entrou de tal forma nos nossos hábitos, que muitas vezes nem damos conta de como nos facilitam a vida; eles estão presentes no controlo da luz da escada, no elevador, no portão de garagem, na máquina de lavar roupa, no sistema de bombagem de água e em muitos mais dispositivos de uso comum.

Na verdade, os automatismos, são dispositivos que permitem que determinado sistema funcione de uma forma autónoma (automaticamente), sendo a intervenção do operador reduzida ao mínimo indispensável.

Sendo bem concebido, um automatismo :

- Simplifica consideravelmente o trabalho do operador.
- Elimina ao operador as tarefas complexas, perigosas, pesadas ou indesejadas.

Quando o automatismo é aplicado a um processo industrial, também tem as seguintes vantagens :

- Facilita as alterações aos processos de fabrico.
- Melhora a qualidade dos produtos fabricados, mantendo uma constância das características dos mesmos.
- Aumenta a produção
- Permite economizar matéria prima e energia.
- Aumenta a segurança no trabalho.
- Controla e protege os sistemas controlados.

1.2. ESTRUTURA DE UM AUTOMATISMO

Podemos dividir um automatismo em três blocos:

- Entradas

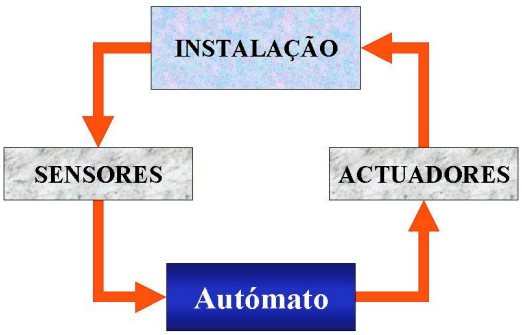
Neste bloco encontram-se todos os dispositivos que recebem informações do sistema a controlar. São em geral sensores, botoneiras, comutadores, fins de curso, etc.

- Saídas

Neste bloco temos todos os dispositivos actuadores e sinalizadores. Podem ser motores, válvulas, lâmpadas, displays, etc..

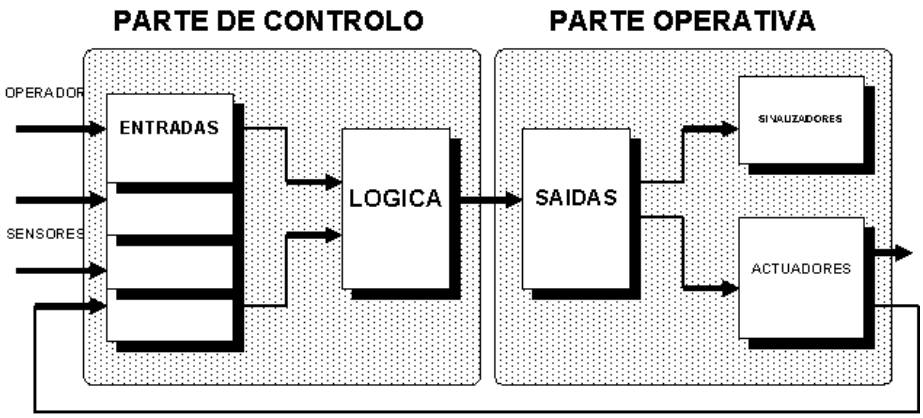
- Lógica

Neste bloco encontra-se toda a lógica que vai permitir actuar o bloco de saídas em função dos dados recebidos pelo bloco de entradas. É este bloco que define as características de funcionamento do automatismo. Pode ser constituído por relés, temporizadores, contadores, módulos electrónicos, lógica pneumática, electrónica programada, etc..



Podemos ainda definir como parte de controlo, o conjunto dos blocos de entradas e de lógica.

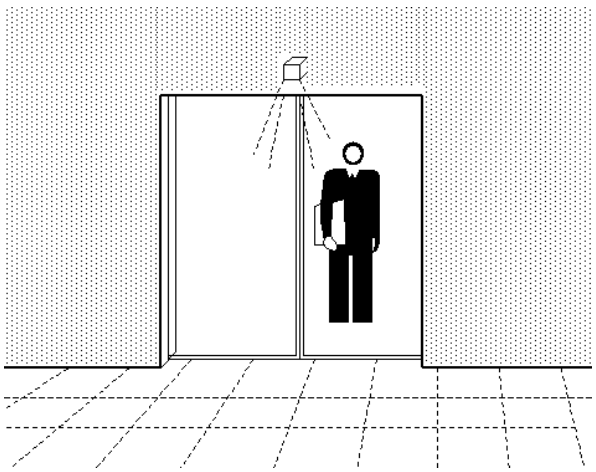
O bloco de saídas será a parte operativa.



Exemplo:

Automatismo de porta

Numa porta automática, o motor que acciona a abertura e fecho da mesma, constitui a parte operativa. O sensor de proximidade, os fins-de-curso, a chave de permissão e toda a lógica de exploração, constituem a parte de controlo.



1.3. IMPLEMENTAÇÃO DE UM AUTOMATISMO

A realização de um automatismo deve iniciar-se pela especificação do mesmo a três níveis:

1.3.1. Especificações funcionais

Visa-se com estas especificações, detalhar todo o funcionamento do sistema a automatizar. Nesta descrição deve pormenorizar-se toda a lógica que vai permitir a correcta exploração do sistema.

Voltando ao exemplo da porta automática, a especificação funcional poderia ser:

** Pretende-se que o automatismo de exploração deste sistema, permita comandar a abertura e fecho de uma porta de acesso.*

O comando poderá ser automático ou manual. Para o efeito, existirá no frontal do quadro de comando, um comutador de duas posições para a selecção do modo.

O modo manual usa duas botoneiras que permitirão respectivamente abrir e fechar a porta. Uma vez premida a botoneira de abertura, a porta abrir-se-á até que seja atingido um fim de curso que detecta o limite de abertura da mesma. Premindo a botoneira de fecho, a porta fechar-se-á até que seja premido um outro fim de curso que detecta o fecho.

O modo automático faz uso de dois sensores que detectam a proximidade de uma pessoa. Quado um deles é activado inicia-se a abertura da porta. Esta permanece aberta até que tenha passado um tempo que pode ir de 5 até 20 segundos, após o desaparecimento do sinal proveniente dos sensores de proximidade. Findo este tempo, a porta inicia o fecho. Se durante o fecho, o sensor de proximidade detectar a presença de uma pessoa, deverá interromper a operação e abrir de novo a porta.

1.3.2. Especificações tecnológicas

Com estas especificações, descreve-se o ambiente em que o sistema vai operar, assim como as características que os equipamentos a integrar deverão possuir, de forma a permitirem o perfeito desempenho do automatismo.

Concretizando com o exemplo anterior, poderia especificar-se:

** O sensor que detecta a proximidade de uma pessoa, dever ser um modelo para ser montado sobre a porta (um no interior e outro no exterior) e dever ser de infravermelhos passivo, com saída por transístor. A sua sensibilidade dever ser tal, que a saída deste só active quando estiver uma pessoa a menos de 2 metros da porta. O motor que aciona a abertura e fecho da porta, ser eléctrico, trifásico, ..,etc.*

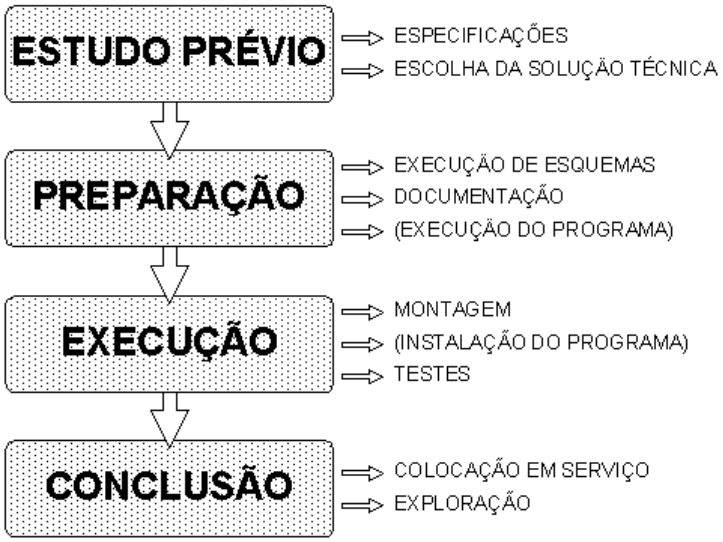
1.3.3. Especificações operativas

Dizem respeito à fiabilidade, segurança, disponibilidade, flexibilidade, manutenção, diálogo homem-máquina, etc.

Voltando ao exemplo anterior poderia especificar-se neste ponto:

** O comutador automático-manual dever ser um modelo com chave. Dever existir um contador de operações de abertura e fecho da porta, de forma a identificar o momento das operações de manutenção que deverão efectuar-se de 10.000 em 10.000 manobras...etc.*

A realização de um automatismo, implica a execução de uma série de tarefas interdependentes. A figura abaixo mostra a sequência de operações necessárias à implementação de um automatismo.



1.4. OPÇÕES TECNOLÓGICAS

São várias as opções tecnológicas disponíveis. Não se pode à partida dizer qual a melhor e pior, pois esta opção depende de várias condicionantes:

- Número de sistemas a automatizar
- Ambiente de trabalho
- Tipo de sinais de entrada/saída
- Função predominante
- Actuadores predominantes

Outra das condicionantes que deve estar sempre presente é a relação preço/performance. Uma determinada solução pode ser perfeita sob o ponto de vista puramente técnico, mas inviável sob o ponto de vista financeiro.

Ao nível das características de implementação, podemos definir dois grandes grupos :

- Lógica cablada
- Lógica programada

Na lógica cablada, o desempenho do automatismo é definido pela interligação física de elementos discretos. A alteração do processo, normalmente implica refazer a cablagem. Como exemplo, podemos referir um circuito eléctrico, composto por relés, relés biestáveis, temporizadores, etc..

Na lógica programada, o desempenho do automatismo é definido por uma série de instruções ou códigos que são programados no suporte de memória do sistema. Ao conjunto de instruções que definem o automatismo, chama-se programa. A alteração da lógica do processo a controlar apenas implica a alteração do programa. Como exemplo, temos o controlo numérico, autómato programável, micro-computador, etc..

Podemos definir basicamente três famílias tecnológicas; Electromagnética , Electrónica e Pneumática / Oleo-hidraulica.

A tecnologia electrónica é muito vasta e pode ser implementada através dos seguintes elementos:

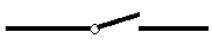
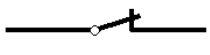
- Circuitos electrónicos dedicados
- Sistemas electrónicos standard (ex.:controlo numérico)
- Autómatos programáveis
- Micro e mini-computadores

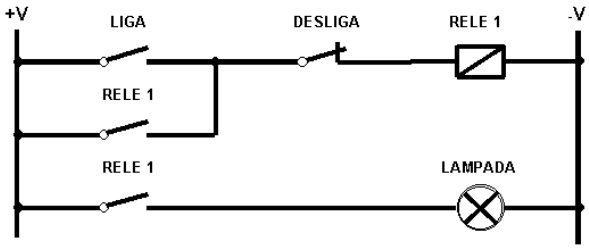
1.5. MÉTODOS GRÁFICOS PARA A DESCRIÇÃO DA LÓGICA DE UM AUTOMATISMO

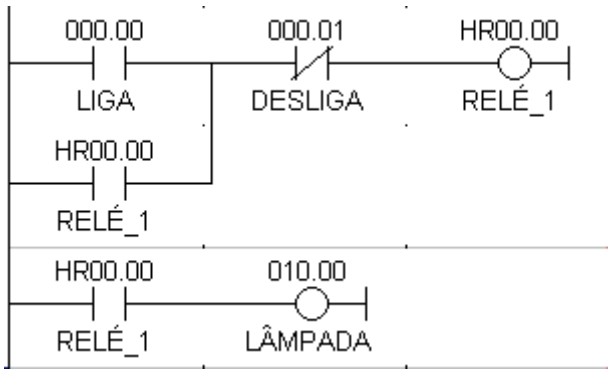
Ao nível das especificações funcionais, é possível traduzir numa linguagem gráfica toda a estrutura lógica do automatismo. Seguidamente apresentam-se algumas das mais usadas:

1.5.1. DIAGRAMA DE CONTACTOS

Este método de representação, implementa a sequência lógica usando contactos colocados em série e paralelo, tal como num esquema eléctrico. O símbolo de contacto é ligeiramente alterado, como se poder ver nas figuras abaixo.

CONTACTO (Acção)	SIMBOLO ELÉCTRICO	LINGUAGEM DE CONTACTOS
DIRECTO		
NEGADO		

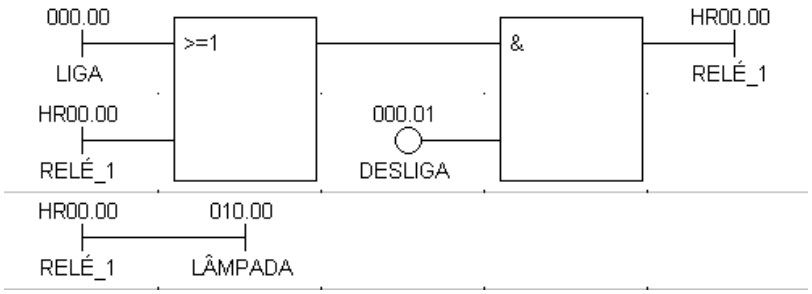




1.5.2. DIAGRAMA LÓGICO (LOGIGRAMA)

Esta representação do automatismo, implementa a lógica, usando circuitos lógicos "E" e "OU".

Para representar o mesmo circuito do exemplo anterior, teríamos:



1.5.3. GRAFCET

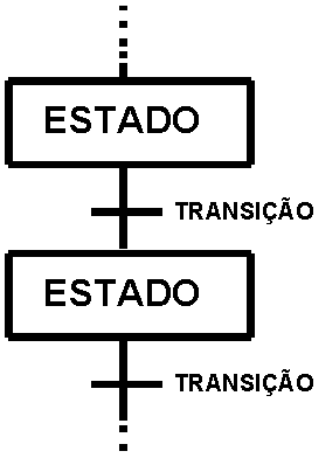
O GRAFCET foi criado com o objectivo de permitir a representação de processos complicados, de uma forma simples. É comparável ao uso dos fluxogramas, simplesmente estes são usados na programação de computadores, enquanto o GRAFCET é especialmente designado para máquinas e processos sequenciais. A principal diferença entre o fluxograma e o GRAFCET é que na estruturação em GRAFCET, todas as sequências possíveis têm de ser perfeitamente definidas. O facto de se ter de prever todas as situações é uma grande vantagem, já que sequências não previstas nunca poderão ocorrer.

O GRAFCET é uma representação gráfica das especificações funcionais de um sistema de controlo e pode ser aplicado a qualquer sistema lógico de controlo de processos industriais.

O nome GRAFCET tem origem numa abreviatura Francesa : GRAphe de Commande Etape-Transition (Gráfico para controlo de estado-transição). É originário de França tendo sido sugerido em 1977 pela AFCET (Association Francaise pour la Cibernetique Economique et Technologie) o seu uso como ferramenta de descrição das especificações de um controlo lógico.

Como já foi dito, um sistema pode geralmente ser dividido num bloco operativo e num bloco de controlo. O bloco operativo executa determinadas operações que são ditadas pelo bloco de controlo. A unidade de controlo, por sua vez, recebe feedback do bloco operativo por forma a manter-se actualizada da evolução do processo.

Quando se pretende usar o GRAFCET, devemos ter em consideração o bloco de controlo. Para o representar usamos uma sucessão alternada de ESTADOS e TRANSIÇÕES. Um processo é decomposto em estados que serão activados sucessivamente.



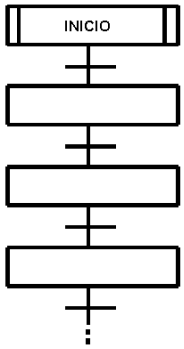
Um ESTADO representa as acções executadas pelo bloco operativo quando esse estado está activo.

A TRANSIÇÃO define as condições que vão permitir uma vez satisfeitas, a desactivação do estado que antecedente e a activação do estado seguinte.

Pode-se ter num sistema basicamente três tipos de processamentos:

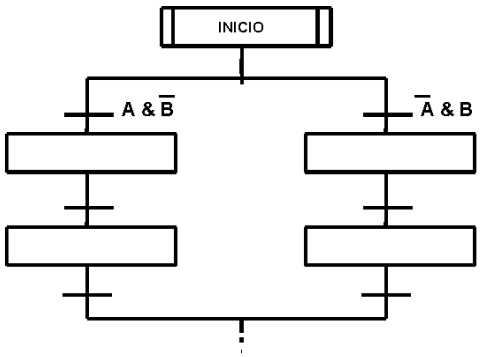
PROCESSAMENTO LINEAR

Os estados do processo estão dispostos em linha. Independentemente das condições do processo, este consta de uma sucessão de passos que se executam sempre, e na mesma sequência.



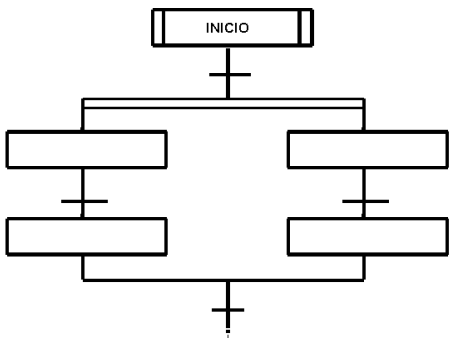
PROCESSAMENTO ALTERNATIVO

A determinada altura do processo, a sequência pode tomar caminhos alternativos de acordo com as condições estabelecidas. Estas condições, no entanto, têm de assegurar que só uma das alternativas se pode verificar.



PROCESSAMENTO PARALELO

Em determinado momento do processo, a sequência pode tomar dois ou mais caminhos que se vão executar em paralelo; ao ser executada a transição, em vez de se activar um só estado, activam-se dois ou mais em simultâneo. O número de estados nos vários ramos pode ser diferente, ou sendo igual, não implica que em todos os ramos a transição de estado se verifique em simultâneo.



Um processo sequencial geralmente consta de uma mistura de estes três tipos de processamento, podendo dar origem a intrincados diagramas.

2. AUTÓMATOS PROGRAMÁVEIS

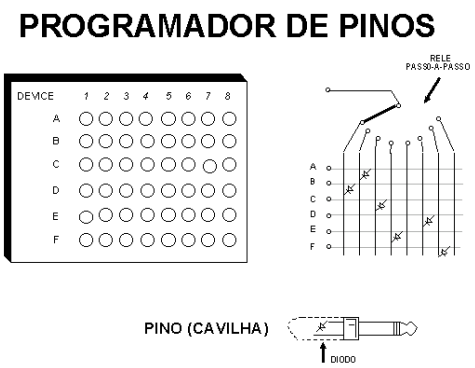
2.1. RESUMO HISTÓRICO

A ideia de criar sistemas flexíveis capazes de controlar processos, sempre norteou o espírito humano. Foi nos sistemas mecânicos que primeiro se desenvolveu este princípio.

À medida em que se foi conhecendo e desenvolvendo toda a tecnologia dos circuitos eléctricos, logo se verificou que facilmente se poderia alterar as características de um circuito de controlo, recorrendo basicamente a relés e comutadores. Desta forma, diversas combinações nos comutadores, davam origem a diferentes modos de funcionamento. Era o primeiro indício de "programação".

De acordo com as necessidades de controlo, foram-se desenvolvendo componentes tais como temporizadores, contadores, relés biestáveis e um sem número de outros componentes que iam integrando sistemas cada vez mais complexos.

Ainda hoje podemos admirar em alguns equipamentos "programadores de pinos". Estes programadores muito rudimentares, constavam de uma matriz de condutores que eram interceptados por uma cavilha ou pino que de acordo com a sua posição permitia definir a activação de determinado circuito.



Em paralelo ao desenvolvimento de circuitos eléctricos, apareceram também circuitos pneumáticos. Tal como nos circuitos eléctricos, nestes também foram desenvolvidos uma série de elementos que tinham desempenhos idênticos aos componentes eléctricos.

A invenção da Unidade Central de Processamento (1938), dá por sua vez origem ao microprocessador. É com base neste componente que em 1969-70 aparecem nos EUA os primeiros autómatos programáveis. Foi a indústria automóvel quem primeiro os utilizou. Na Europa, só dois anos depois é que começam a ser empregues na indústria.

Hoje é já difícil falar em automatização industrial, sem que se tenha de referir o autómato programável.

Sobre a tecnologia cablada, o autómato programável tem inúmeras vantagens:

- É muito mais fiável, pois o número de componentes mecânicos e de ligações é mínimo.
- O desenvolvimento do programa pode ser feito em paralelo com a montagem dos equipamentos. O sistema por lógica cablada, só pode ser montado depois do projecto estar completamente concluído.
- As alterações do automatismo só implicam alterações no programa. Na lógica cablada, qualquer alteração implica a alteração da cablagem e dos componentes.
- O espaço ocupado pelo autómato é constante e independente da complexidade da lógica do automatismo.

- Não requer stocks de equipamento de reserva tão elevados como nos sistemas por lógica cablada.

Sobre os sistemas controlados por microprocessador ou microcomputador, o autómato não requer a presença de um perito em informática ou em assembler para programar ou alterar um programa. A linguagem utilizada é standard e pode ser facilmente apreendida por pessoas com uma formação mínima.

Pode classificar-se a utilização dos autómatos programáveis em três classes:

- Controlo de máquinas ou automatismos simples e individualizados.
- Controlo e sincronização de diversas máquinas ou automatismos de uma linha de fabrico ou de um sistema complexo.
- Supervisão e controlo de uma unidade fabrico ou de um conjunto de sub-sistemas que podem eles mesmos serem também controlados por autómatos.

São hoje inúmeros os fabricantes de autómatos programáveis. No mercado português são cerca de 30 as marcas presentes, encontrando-se por isso, uma grande diversidade de modelos.

Quando se refere um autómato programável, é normal caracterizá-lo pelo número de Entradas+Saídas lógicas que este pode controlar.

No início, quando a linguagem de programação dos autómatos só incluía instruções booleanas, temporizadores e contadores, em função do número de entradas+saídas, era usual catalogá-los numa das três gamas:

Gama Baixa - E+S < 64
Gama Média - 64 < E+S < 256
Gama Alta - E+S > 256

Nesta divisão nunca devemos encarar os números apontados, como fronteiras absolutas, mas sim como valores de referência; cada fabricante tem fronteiras e conceitos próprios para a definição dos seus autómatos.

Hoje não é só o número de entradas+saídas que define a gama do autómato, mas também o tipo e número de instruções de programação disponíveis, a velocidade de processamento, as facilidades de comunicação e outras características, que definem as potencialidades do PLC.

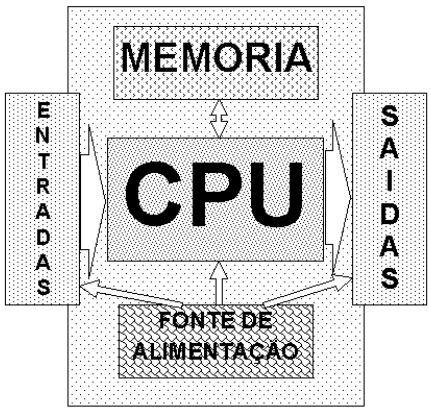
Quanto à sua apresentação, o autómato, pode ser:

- Compacto - Integra no mesmo bloco todos os elementos necessários à sua operação (entradas, saídas, CPU, memória e fonte de alimentação).

- Modular - É constituído por uma série de módulos que ligam a um bastidor (módulo do CPU, cartas de entrada, cartas de saída, memória, fonte de alimentação, etc.). Tem a vantagem de ser mais versátil que os modelos compactos. Como desvantagem geralmente aponta-se o seu custo, que é mais elevado, e o volume ocupado, que é também maior quando comparado com idêntico modelo compacto.

2.2. ESTRUTURA DE UM AUTÓMATO PROGRAMÁVEL

Do ponto de vista do utilizador, o autómato é uma "caixa preta" que processa informação.



Os Controladores Lógicos Programáveis (PLC's) podem apresentar aspectos físicos diferentes, diferentes performances e custos muito díspares; no entanto, os seus elementos constituintes são fundamentalmente os mesmos.

Sendo um equipamento capaz de controlar processos, naturalmente dispõe de dispositivos de aquisição e saída de informações. Sendo também um equipamento programável, integra um microprocessador e uma memória para guardar o programa. Para alimentar os circuitos atrás descritos, existirá também uma fonte de alimentação. Finalmente, para que possa ser introduzido o programa e para que possa existir um diálogo básico para o exterior, dispõe também a possibilidade de ligar dispositivos de programação.

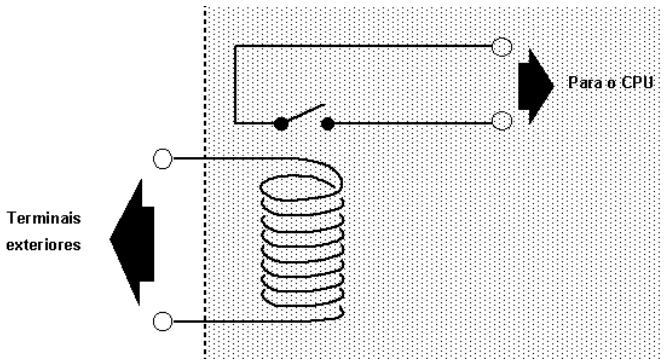
2.2.1. ENTRADAS / SAÍDAS

Sendo o autómato programável destinado ao controlo de processos, tem obrigatoriamente de adquirir dados referentes ao sistema a controlar e fornecer sinais de comando. Existem diversos tipos de entradas e saídas; nos parágrafos seguintes são apresentadas algumas das versões possíveis.

Normalmente o estado lógico das entradas e saídas é sinalizado por led's que quando ligados indicam a activação de determinada entrada ou saída.

ENTRADAS

Entradas usando relés

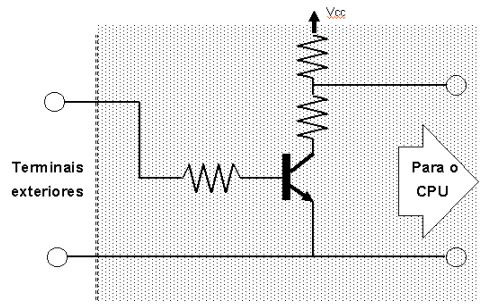


Nesta versão, existe um relé cuja bobine é excitada por uma tensão eléctrica aplicada na entrada do PLC. Os contactos do relé fornecem ao CPU um estado lógico correspondente ao estado da entrada. Como podemos ver no esquema seguinte, estando a entrada do PLC alimentada, implica que o contacto do relé se feche e conduza a informação aos circuitos de aquisição de sinais do CPU. Caso desapareça a tensão na entrada do PLC,o contacto do relé abre, e o valor lógico do circuito passa a zero. Este sistema garante um isolamento galvânico entre a entrada e CPU, já que o contacto do relé é isolado da bobine que o actua.

Este tipo de entrada tem a vantagem de poder aceitar sem problemas, tensões alternadas ou contínuas, introduzindo no entanto, um atraso considerável aos sinais lidos. O consumo de corrente na entrada é maior que nos circuitos usando semicondutores; este aspecto, pode ser de grande vantagem, quando se adquire um sinal que pode ser afectado pelas interferências induzidas no cabo que liga o sensor ao autómato.

Entradas por transístor

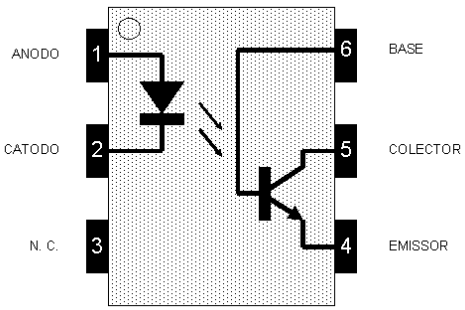
Este tipo de entrada usa um transístor e um conjunto de resistências para adquirir e converter os sinais na entrada, de forma a poderem ser lidos pelo CPU. Pode analisar-se na figura seguinte o funcionamento; Sempre que aparece um sinal positivo na "base" do transístor, o transístor conduz, fazendo com que Vo tenha um valor muito próximo de zero. Quando não há presença de tensão na entrada, Vo tem um valor próximo de Vcc.



Comparada com a entrada por relé, este tipo de entrada pelo facto de usar um semicondutor, reduz consideravelmente o atraso aos sinais de entrada e é mais fiável, uma vez que elimina sistemas mecânicos (relé) ; A corrente absorvida pela entrada é muito menor (na ordem da dezena de mA). Tem a desvantagem de não garantir isolamento eléctrico entre a entrada e o CPU.

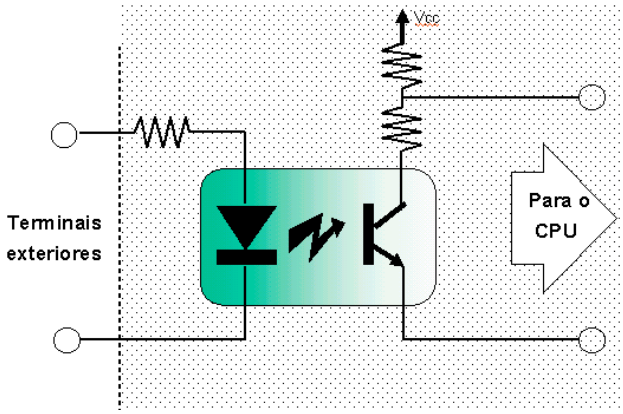
Entrada por acopulador óptico

Para superar a desvantagem de isolamento da entrada anteriormente descrita, esta pode ser isolada do CPU através do uso de um acopulamento óptico.



O acopulador óptico é constituído por um LED (díodo foto emissor) e um foto-transístor encapsulados num material isolante, conforme indicado na imagem. A transmissão da informação do estado da entrada para o CPU, é feita através do fluxo de fotões emitido pelo LED e recebido pelo foto-transístor. Ao ser atingido pelos fotões, o foto-transístor entra em condução. Como o meio de transmissão deste fluxo é dieléctrico, consegue-se um óptimo isolamento galvânico. A corrente de entrada deste circuito é também muito baixa.

Nas entradas por semicondutor atrás descritas, há que respeitar a polaridade do circuito de entrada. Esta pode ser PNP ou NPN. Ao escolher-se um sensor , este deverá estar em sintonia com a entrada onde vai ser ligado ; numa entrada PNP o comum tem de ser positivo.



Para ultrapassar esta condicionante, existem circuitos de entrada onde o isolador óptico dispõe de dois díodos led montados em

paralelo mas opostos na polaridade. Assim, qualquer que seja o sentido da corrente na entrada, é assegurada a polarização do fototransístor e consequentemente a detecção de um sinal activo na entrada.

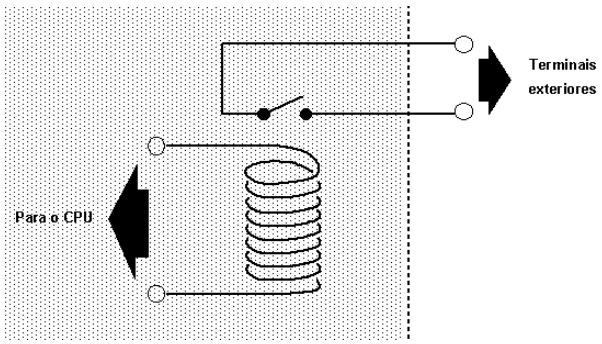
SAÍDAS

As saídas do autómato realizam uma função inversa à das entradas, ou seja, permitir a actuação de elementos que integram o sistema a controlar.

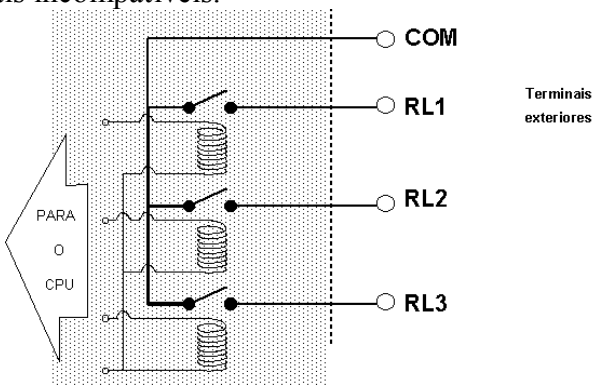
Podem ser de vários tipos:

Saída por relé

O sinal proveniente do CPU, ataca a bobine de um relé. Na saída do autómato, estão disponíveis os terminais do contacto do referido relé.

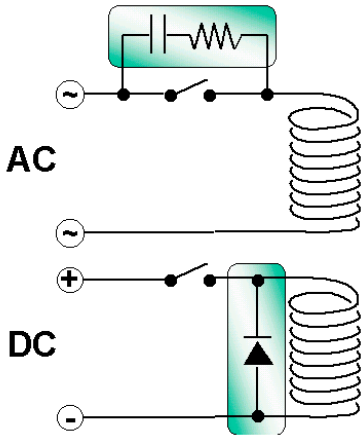


Regra geral, um autómato tem mais do que uma saída. Para que o número de terminais não seja exageradamente elevado, é frequente agruparem-se saídas, havendo para o efeito um terminal comum. Deve haver algum cuidado ao efectuar as ligações, pois deve assegurar-se que num mesmo comum não se juntem sinais incompatíveis.



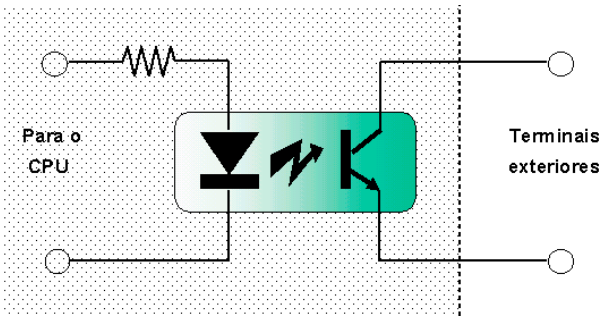
Este tipo de saída é o mais frequentemente usado, por ser o mais versátil. Pode comutar correntes contínuas ou alternadas, de tensões muito diversas. A sua frequência de resposta e o "bounce" são as suas principais desvantagens.

Para reduzir o desgaste do contacto do relé quando este comuta cargas indutivas, deve usar-se um dos dispositivos da figura abaixo, conforme se trate de corrente contínua ou alternada.



Saída por transístor

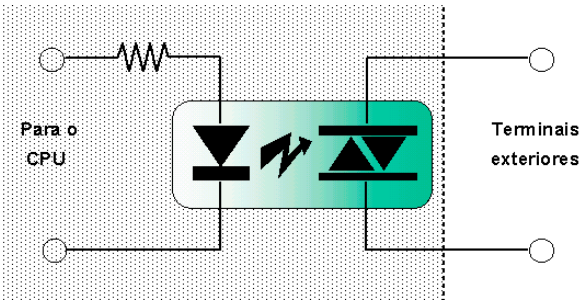
Este tipo de saída usa um transístor que recebe na "base" o sinal lógico proveniente do CPU; os terminais do "colector" e "emissor" são acessíveis do exterior, para ligação aos circuitos a controlar. Para que possa existir isolamento galvânico entre o CPU e os circuitos exteriores ao autómato, é frequente usar (em vez de um vulgar transístor) um fototransístor.



Este tipo de saída é usada quando os sinais a controlar são de corrente contínua, baixa tensão, baixas correntes e de frequência elevada.

Saída por triac

Nesta saída usa-se um triac como elemento activo na comutação das cargas. O sinal proveniente do CPU liga à "gate" do triac ou então activa o led de um foto-triac. Esta última opção é a mais usada por garantir um perfeito isolamento entre o CPU e os circuitos exteriores ao autómato.



A saída por triac usa-se na comutação de cargas trabalhando em corrente alternada. Tem a vantagem de poder comutar a frequências elevadas, não sofrendo desgaste significativo, quando comparado com um relé. Alguns fabricantes apresentam cartas de saídas por triac com a característica de comutarem cargas só quando a corrente alternada passa por zero; este pormenor faz reduzir a ocorrencia de ruidos parasitas na rede eléctrica, que normalmente ocorrem quando a comutação é realizada com relés.

2.2.2. CPU

É este o bloco que tem a função de ler os valores lógicos presentes nas entradas, executar as instruções que constituem o programa e transferir para as saídas as ordens provenientes dessas instruções. Tem ainda a seu cargo gerir todos os periféricos e diagnosticar defeitos que possam ocorrer internamente.

Na base de tudo isto, está um ou mais microcessadores que de uma forma sequencial vão executando instruções a velocidades extremamente elevadas.

A sequência descrita no primeiro parágrafo, é continuamente realizada. O tempo gasto para a realizar é designado como tempo de ciclo (ou tempo de scan) e é da ordem dos milisegundos. O tempo de ciclo depende de muitos factores dos quais se destacam:

- Velocidade de trabalho do microprocessador(s)
- Número de instruções do programa
- Tipo de instruções usadas no programa
- Número de periféricos

O CPU tem ainda circuitos de endereçamento de entradas e saídas, uma memória com o sistema operativo, interfaces para unidades externas, um circuito para a detecção de falhas de alimentação, e outros que interligam os anteriores.

Para sinalizar o estado de funcionamento do CPU, normalmente existem no frontal do mesmo, sinalizadores luminosos (led's). É comum encontrar as seguintes sinalizações:

- Presença de alimentação (POWER)
- Erro no CPU (ERROR ou FAILURE)
- Execução do programa (RUN)
- Falha da bateria de backup (BATTERY)

Como o CPU é um elemento vital de um autómato, exitem modelos que para garantirem uma grande fiabilidade de operação, dispõem de duplo CPU; quando um deles falha, o outro entra imediatamente em serviço, sem interromper o controlo do sistema.

2.2.3. Memória

É na memória que se encontra o programa a ser executado pelo autómato. A memória tem como função salvaguardar todas as instruções do programa, mesmo quando este não está a ser alimentado.

A memória caracteriza-se pela sua capacidade que pode ser expressa de três formas:
- Número de bits ou Kbts (1 Kbts = 1024 bits)
- Número de Bytes ou KB (1 Byte = 8 bits)
- Número de Words ou KW (1 Word = 16 bits)

Quanto à sua tecnologia podem ser :

- RAM (Random Access Memory)
Estas memórias têm a vantagem de poderem ser escritas e alteradas facilmente. São as mais usadas quando se está na fase de desenvolvimento do programa ou quando o sistema a controlar sofre frequentes alterações. Estas memórias perdem a informação quando a alimentação eléctrica das mesmas falha; por isso, obrigam ao uso de uma pilha de recurso que assegura a sua alimentação no caso de uma falha de energia.

- EPROM (Erasable Programable Read Only Memory)
Esta memória não perde a informação nela gravada no caso de falhar a tensão. Têm como desvantagem o facto de ser muito morosa qualquer alteração, mesmo sendo de um só bit. Antes de ser programada por um equipamento próprio, tem de ser apagada por exposição aos raios ultravioletas.

- EEPROM (Electricaly Erasable Programable Read Only Memory)
Esta memória não perde informação por falta de tensão de alimentação e pode ser apagada e escrita pelo autómato. Tem vantagens sobre os modelos anteriores, mas os inconvenientes de ter um número limitado de ciclos de escrita e do seu custo ser mais elevado que o de uma RAM.

-FLASHRAM
Esta memória de tecnologia muito recente, tem características semelhantes às EEPROM, permitindo também escrita e leitura no próprio ciscuito onde é usada. Limitada também pelo número de ciclos de escrita, apresenta vantagens sobre a EEPROM (uma delas, a velocidade de escrita).

2.2.4. Fonte de Alimentação

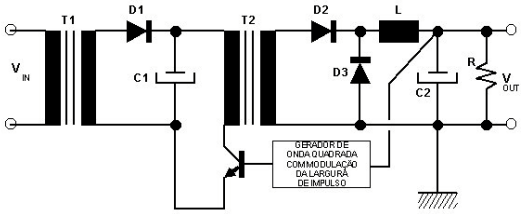
A fonte de alimentação tem por função fornecer as tensões adequadas ao funcionamento do CPU. Para esse efeito, é geralmente ligada aos 220v da rede, mas também existem modelos que aceitam tensões contínuas standard (ex.:24VDC).

Para além de fornecer tensões, a fonte de alimentação deve ter um bom comportamento no que diz respeito à filtragem de ruido e picos, muito frequentes nas instalações industriais.

Encontramos com grande frequência a equipar os autómatos, fontes de alimentação comutadas. Estas fontes reúnem para além das características atrás exigidas, as seguintes:

- Elevado rendimento
- Ocupam um pequeno volume
- Aceitam grandes variações na entrada

No esquema abaixo apresenta-se o esquema simplificado de uma fonte comutada (também conhecida como fonte switching).



Nos modelos alimentados a 220vac, por vezes, a fonte de alimentação tem uma saída auxiliar a 24vdc para alimentar os circuitos de entrada.

2.2.5. CONSOLA DE PROGRAMAÇÃO

O autómato programável não necessita desta unidade para o seu funcionamento. No entanto, a consola de programação constitui o equipamento mais acessível para introduzir ou alterar o programa que vai definir o funcionamento do automatismo e ainda monitorizar e alterar o estado das variáveis. Normalmente a consola só é necessária para a programação e colocação em serviço do autómato.

Esta unidade é basicamente é constituída por um teclado e um visor (display).

Conforme a sua natureza, poder permitir a programação em linguagem mnemónica (conjunto de abreviaturas e símbolos que constituem as instruções do programa), linguagem de contactos, logigrama, etc..

Há consolas mais sofisticadas que permitem guardar e ler programas gravados em suportes magnéticos, programar memórias EPROM, e dispor ainda de uma série de facilidades para o desenvolvimento de programas, tais como: monitorização em tempo real das condições lógicas de determinado bloco de instruções, atribuir nomes às variáveis, produzir listagens em impressora dos programas, etc.

Os autómatos podem ser também programados por computadores pessoais. Devido ao seu custo cada vez mais baixo, versatilidade e portabilidade, começam a ser cada vez mais usados na programação dos PLCs.

3. PROGRAMAÇÃO DE AUTÓMATOS

3.1. PRINCÍPIOS GERAIS

No capítulo anterior vimos o HARDWARE básico de um autómato; neste capítulo vamos analisar o SOFTWARE, ou seja, o conjunto de instruções e procedimentos que nos vão permitir implementar a lógica de controlo do automatismo.

Ao programarmos um autómato, estamos a definir a forma como as saídas deste são actuadas, em função de dados presentes nas entradas. Vamos então ver como podemos ter acesso às informações presentes nas entradas e como poderemos endereçar uma determinada saída.

Internamente e implementados pelo CPU existem bits, que não são mais do que posições de memória nas quais é possível reter uma informação lógica; ligado/desligado, verdadeiro/falso, ON/OFF ou 1/0. Estes elementos (aos quais também chamam relés por analogia com os circuitos electromagnéticos) estão normalmente associados em grupos de 16.

Um conjunto de 16 bits chama-se WORD (por vezes também se designa por CANAL).

Dentro de cada word os bits estão numerados de 00 a 15 ou de 0 a F (0,1,..,9,A,B,..,F) conforme a notação usada pelo fabricante. As words estão numeradas a partir de 00.

Os bits são endereçados pelo número da word em que se encontram e pela posição que ocupam nessa word (há também autómatos onde o endereçamento é contínuo, ou seja, começa em zero e segue a numeração decimal, independentemente de serem bits de entrada, saída ou internos).

Exemplo:
- Se pretender endereçar o 7º bit da 2ª word, então o seu endereço ser 106.
- Se pretender endereçar o 1º bit da 1ª word, então o seu enderço ser 000.
- Se pretender endereçar o 13º bit da 4ª word, então o seu endereço será 313 ou 3.C conforme a notação.

Num autómato há várias áreas de relés(bits) das quais se destacam:

- Relés de I/O
- Relés com retenção
- Relés de temporizadores e contadores
- Relés especiais

Os relés de I/O (input/output) são bits que podem estar associados a entradas ou saídas do autómato. Normalmente são em número superior ao número máximo de entradas+saídas possíveis.

Os bits que estão associados a entradas, têm o seu estado lógico definido pelo estado da entrada. Os bits associados a saídas reflectem nestas o seu estado.

Esta área de relés retém a informação enquanto o autómato se encontra alimentado. A partir do momento em que há uma falha na alimentação do autómato, todos os relés desta área tomam o

estado OFF, mantendo esse estado quando regressa a alimentação (os relés afectados pelas entradas tomam o valor presente nas mesmas).

Normalmente num mesmo canal (word) dispomos ou só de entradas ou só de saídas.

A assignação das entradas/saídas aos canais respectivos, depende de fabricante para fabricante e de modelo para modelo de autómato.

Os relés de retenção, contrariamente aos anteriores, retêm o seu estado mesmo quando há falha de alimentação do autómato. Estes relés não estão associados a entradas/saídas e são usados para guardar dados.

Os relés de temporizadores e contadores são relés cujo estado está associado a um determinado temporizador ou contador.

Nos relés especiais não é possível alterar directamente o seu estado. Este pode depender de funções que são executadas por programa ou pode ser determinado pelo CPU. São usados para monitorizar operações do PLC, aceder a impulsos de clock e sinalizar erros.

O programa que vai definir o automatismo, é constituído por numa série de instruções e funções onde são operandos os relés(bits) atrás mencionados. Estas instruções e funções serão introduzidas na memória do autómato através de um periférico destinado a esse fim e que poderá ser uma consola de programação ou software específico para PC.

Os autómatos têm basicamente dois modos de operação: RUN e PROGRAM.

- O modo RUN é o modo normal de funcionamento do autómato. Neste modo o CPU executa o programa contido na memória.
- Para se introduzir o programa, é preciso que o autómato se encontre no modo PROGRAM. Neste modo, o autómato não executa o programa.

Estes modos são normalmente seleccionados através de um comutador que se pode encontrar no frontal do autómato ou na consola de programação.

Antes de introduzir um programa através da consola, deve converter-se o esquema de contactos numa lista de instruções entendidas pelo autómato. Há no entanto dispositivos que permitem a programação directa em esquema de contactos (por ex. Software para PC).

O programa é introduzido nos endereços de memória do programa. Cada endereço contém uma instrução, os seus parâmetros de definição e todos os parâmetros requeridos por essa instrução. Os endereços de memória do programa (linhas do programa) começam em 0 e estão limitados pela capacidade da memória de programa.

Cada fabricante de autómatos tem formas diferentes de levar a cabo a programação de um PLC e por isso as suas especificidades; As áreas de relés têm designações diversas, as instruções e funções têm mnemónicas e códigos diferentes, e a sequência de teclas na consola para levar a cabo a programação, difere de marca para marca. No entanto, conhecendo um modelo, facilmente nos integramos noutro pela simples consulta do respectivo manual, já que a lógica de programação dos sistemas existentes no mercado não difere no essencial.

Para podermos abordar com objectividade a programação de um autómato vamos exemplificar com o autómato SYSMAC CPM1 da OMRON.

FICHA TÉCNICA

Modelo : CPM1-20CDR-A
Autómato compacto

Entradas : 12 digitais(24 Vdc)
Saídas : 8 por relé (2A/24Vdc-220Vac)
Alimentação : 220 Vac
Consumo : 60 W max.
Expansibilidade : até 90 entradas/saídas.
Memória de programa : FLASHRAM - 4kB

Conjunto de instruções : 135
Relés internos:
Sem retenção : 68 x 16
Com retenção : 20 x 16
Words de dados : 1024 com retenção
Temporizadores e Contadores: 127
Velocidade de processamento: 0,72 a 16,2 µS / instrução

.Interface série RS-232C em opção (CPM1-CIF01)
.Uma entrada de contagem rápida (2KHz bidireccional).
.Duas entradas de interrupt
.Dois registos analógicos

Dimensões : 180 x 90 x 85 mm
Peso : 800 gr.

Este autómato dispõe das seguintes áreas de memória:

Área	**Símbolo**	**Gama**
Relés Internos	IR	Words:000 a 019 e 200 a 231 Bits :00.00 a 019.15 e 200.00 a 231.15
Relés Especiais	SR	Words:232 a 255 Bits :232.00 a 255.15
Relés Auxiliares	AR	Words:AR00 a AR15 Bits :AR00.00 a AR15.15
Relés com Retenção	HR	Words:HR00 a HR19 Bits :HR00.00 a HR19.15
Temporizadores e Contadores	TC	TC000 a TC127
Relés de Comunicação	LR	Words:LR00 a LR15 Bits :LR00.00 a LR15.15
Relés Temporários	TR	TR00 a TR07 (Só Bits)
Memória de Dados	DM	Leitura/Escr.:DM0000 a DM1023 Só leitura :DM6144 a DM6655
Memória de Programa	UM	4 Kbytes

Relés Internos - São usados para controlar os pontos de entradas/saídas, outros bits, temporizadores, contadores e para guardar temporariamente dados. As entradas neste autómato ocupam 12 bits do canal 0. As saídas ocupam 8 bits do canal 10.

Relés Especiais – Disponibilizam sinais de clock, flags, bits de control e status do sistema.

Relés Auxiliares - Contêm bits e flags para funções especiais. Retêm o seu estado durante a ausência de alimentação.

Memória de Dados - São usados para memorização e manipulação de dados. Retêm os dados durante a ausência de alimentação.

Relés com Retenção - São usados para guardar e memorizar dados quando o autómato é desligado.

Relés de Temporizadores e Contadores - São como operandos das instruções LD(NOT), AND(NOT) e OR(NOT) informam o estado dos contadores e temporizadores com o mesmo endereço.

Relés de Comunicação – A sua principal função está associada ao estabelecimento de comunicações para troca de dados automática com outros autómatos. Na ausência desta função, podem ser usados como relés de trabalho.

Relés Temporários - São usados para guardar de forma temporária estados de condições de execução. Estes bits só podem ser usados nas instruções LD e OUT.

Memória de Programa - É usada para guardar o conjunto de instruções que constitui o programa do autómato. O número máximo de instruções que pode ser introduzido nesta memória, depende do tipo de instruções usadas.

RELÉS ESPECIAIS

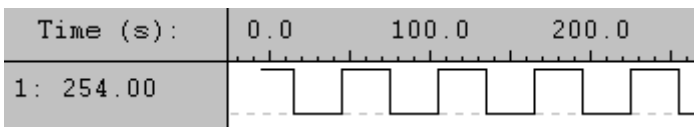
A área de relés especiais é uma continuação da área de relés internos e ocupa os endereços 232.00 até ao 255.15. Seguidamente enumeram-se alguns dos relés especiais mais relevantes.

253.13 - Relé sempre a ON.

253.14 - Relé sempre a OFF

253.15 - Relé de 1º scan. Este relé encontra-se a ON durante a execução do primeiro scan, passando em seguida a OFF.

254.00 - Clock com período de 1 minuto. Este relé está continuamente a oscilar (ON-OFF) sendo o período da oscilação de 1 minuto e o duty-cycle de 50%(o tempo a ON é igual ao tempo em OFF).



254.01 - Clock com período de 0,02 segundos.

255.00 - Clock com período de 0,1 segundos.

255.01 - Clock com período de 0,2 segundos.

255.02 - Clock com período de 1 segundo.

255.03 - Flag de Erro (ER) na execução de uma instrução.

255.04 - Flag de Carry (CY).

255.05 - Flag de "maior que". Este relé altera seu estado sempre que é executada uma função de comparação.

255.06 - Flag de "igual a". Este relé altera seu estado sempre que é executada uma função de comparação.

255.07 - Flag de "menor que". Este relé altera seu estado sempre que é executada uma função de comparação.

3.1.2. CONSOLA DE PROGRAMAÇÃO

3.1.3. TECLAS NUMÉRICAS

São as teclas brancas numeradas de 0 a 9. Estas teclas são usadas para introduzir valores numéricos ou alfanuméricos (recorrendo à tecla SHIFT). Também são usadas associadas à tecla FUN para programar instruções especiais.

TECLA CLR

Esta tecla é usada para limpar o display. Usa-se também quando é necessário limpar do visor a mensagem "PASSWORD". Para isso, deve digitar-se a sequência CLR + MONTR.

TECLAS DE OPERAÇÃO

Estas teclas amarelas são usadas na edição do programa. Deste grupo destacam-se três pela sua frequência de uso.

As teclas com as setas permitem incrementar ou decrementar o endereço da memória do programa de forma a visualizar as várias instruções em memória.

A tecla WRITE permite validar as instruções de programa escritas na consola.

À frente será explicada a função das restantes teclas.

TECLAS DE INSTRUÇÕES

Exceptuando a tecla SHIFT, as restantes teclas cinzentas servem para introduzir as instruções do programa. A tecla SHIFT permite aceder às funções superiores das teclas com dupla função.

Cada uma das restantes teclas cinzentas tem assignada uma função indicada com uma abreviatura, que em seguida se explica.

FUN - Permite seleccionar uma função especial.

SFT - Instrução SHIFT REGISTER (também pode ser programada com FUN+10).

NOT - Permite negar o estado de um relé (bit).

AND - Instrução AND ("E" lógico).

OR - Instrução OR ("OU" lógico).

CNT - Instrução CONTADOR.

LD - Instrução LD usada para iniciar uma condição ou bloco lógico.

OUT - Instrução de OUTPUT. Permite transferir um valor lógico para um relé.

TIM - instrução TEMPORIZADOR.

TR - Especifica um relé temporário.

LR - Especifica um relé ou canal de LINK.

HR - Especifica um relé ou canal com retenção de memória.

DM - Especifica um canal DATA MEMORY.

CH - Especifica um canal (IR, SR, HR, AR ou LR).

CONT - Especifica um contacto de um relé (IR, SR, HR, AR ou LR).

- Especifica uma constante numérica

* - Especifica um endereçamento indirecto, quando usado com DMs.

SELECTOR DE MODO

A consola de programação está equipada com um comutador para controlar o modo do autómato. O modo seleccionado determina a operação do autómato, assim como as funções possíveis com a consola de programação.

O modo RUN é o modo usado para a normal execução do programa. Neste modo é possível a monitorização de dados, mas a sua alteração não é permitida.

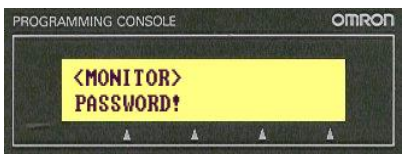
No modo MONITOR o programa é executado tal como acontece no modo RUN, mas permite a monitorização das instruções do programa "on-line", assim como monitorizar e alterar dados. Este modo é normalmente usado na fase de teste e afinação de um programa.

No modo PROGRAM o programa não é executado. Este modo é destinado à alteração ou limpeza de áreas de memória, assim como à programação, alteração ou limpeza da memória de programa.

3.1.4. OPERAÇÃO DA CONSOLA DE PROGRAMAÇÃO

INTRODUÇÃO DA PASSWORD

Para aceder às funções da consola, é necessário introduzir uma password. Esta password é igual em todos os modelos e não pode ser alterada. A sua função é não permitir que "curiosos" possam alterar o conteúdo do programa ou o funcionamento do autómato (pressupõe-se que quem conhece a password tem também outros conhecimentos sobre o funcionamento deste equipamento).



Sempre que aparece no ecrã a mensagem "PASSWORD" deve digitar-se a seguinte sequência de teclas:

LIMPEZA DA MEMÓRIA

Para se inicializar a memória do autómato, deve executar-se a seguinte sequência de teclas:

Após a execução desta sequência, foram limpas do seu conteúdo a área de memória e todas as áreas de relés com retenção.

LIMPEZA DE MENSAGENS DE ERRO

Quaisquer mensagens de erro que se encontrem em memória, devem ser apagadas (Presume-se que as causas que originaram o aparecimento destas mensagens, foram eliminadas).

Para visualizar uma mensagem deve digitar-se a seguinte sequência:

Para anular a presente mensagem e visualizar a seguinte (se houver) deve premir-se novamente a tecla MONTR.

OPERAÇÕES DE EDIÇÃO

INTRODUÇÃO DE INSTRUÇÕES

Uma vez o programa convertido em mnemónicas, pode iniciar-se a sua introdução na memória do autómato. As instruções do programa só podem ser introduzidas com o autómato em modo PROGRAM.

A primeira instrução de um programa (LD) deve ser programada no endereço 00000 (este endereço aparece no canto superior esquerdo do ecrã); as outras instruções ocuparão os endereços sucessivos.

Tal como já foi dito, após ter-se digitado uma linha do programa, deve validar-se esta, premindo a tecla WRITE; o endereço do programa é incrementado automaticamente, possibilitando a introdução de uma nova linha de instrução.

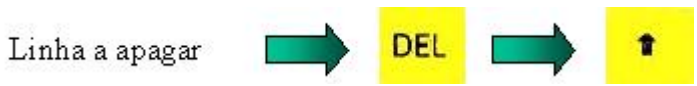
INSERÇÃO DE INSTRUÇÕES

Pode acontecer um esquecimento ou querer acrescentar a um programa já introduzido, uma nova instrução. Para o fazer, deve posicionar-se no visór da consola de programação a linha de instrução que vai preceder a nova instrução; para o efeito use as teclas com as setas. Em seguida deve fazer-se a seguinte sequência de teclas:

O ecrã apresenta agora a mesma instrução, mas com o endereço incrementado; após a sequencia anterior, o autómato duplicou a linha de instrução que estava a ser visualizada. Deve agora escrever-se a nova instrução, premindo WRITE no final.

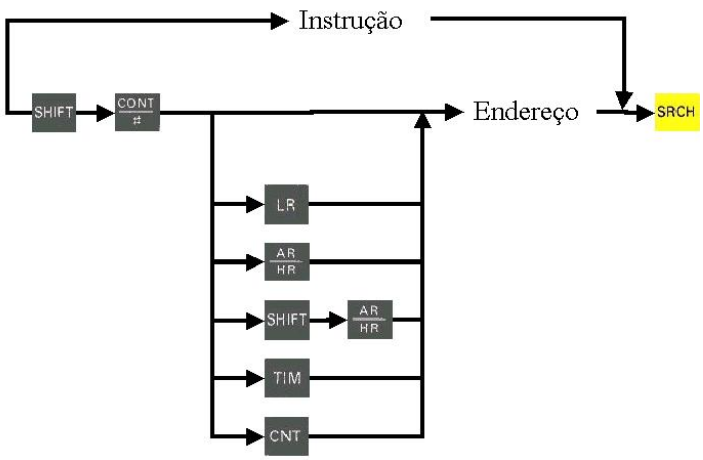
ELIMINAÇÃO DE INSTRUÇÕES

Para retirar uma linha de instrução ao programa já introduzido, deve posicionar no visor da consola de programação a linha que pretende eliminar. Em seguida deve executar a seguinte sequência de teclas:



BUSCA DE INSTRUÇÕES

É possível procurar ou saber o número de ocorrências de determinada instrução ou relé, sem ter de percorrer todo o programa (recorrendo às teclas com setas). Para tal, bastar executar a sequência de teclas abaixo descrita, tendo atenção de que esta deve ser iniciada estando o visor limpo (só aparece no canto superior esquerdo o endereço 00000).

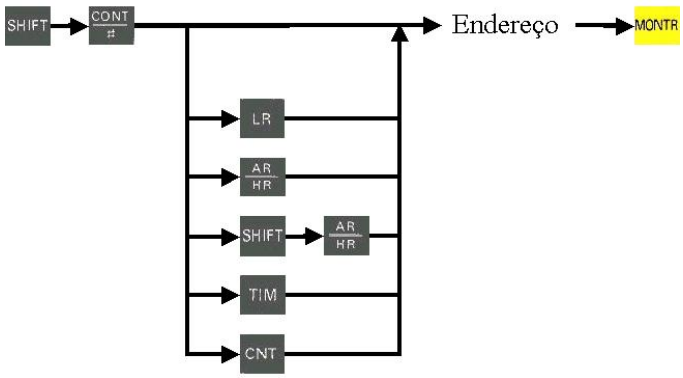


Quando se faz SRCH, automaticamente é procurada no programa a instrução ou relé. Se existir, então aparecer no visor a linha do programa que contém a primeira ocorrência. Para procurar novas ocorrências, bastar premir a tecla SRCH. Quando não houver mais ocorrências da instrução ou relé em causa, então aparecerá a última linha do programa, que deverá ser END(01).

MONITORIZAÇÃO

ESTADO DE UM RELÉ

É possível monitorizar e alterar o estado de um relé. Para tal é necessário executar a seguinte sequência de teclas, após ter limpo o ecrã premindo a tecla CLR.

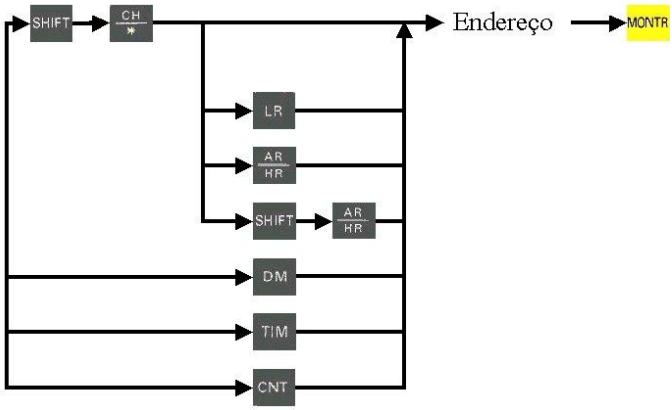


Depois de executada esta sequência, no ecrã aparece o estado do relé em causa (ON ou OFF), tal como no exemplo que se segue. Visualiza-se aqui o estado do relé 200.00 que em princípio estará a OFF.

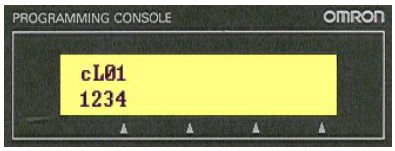
Sempre que se pretender, pode alterar-se o estado de um relé usando a consola de programação. Para tal basta após monitorizar o seu estado, premir as teclas SET ou RESET, conforme se queira colocar a ON ou a OFF respectivamente. Há que ter em atenção o facto de a consola não ter predominância sobre o programa ou entradas físicas; se forçar a ON um relé associado a uma entrada que está no momento a OFF, este continuará a OFF. Há no entanto forma de forçar o estado de um relé nas condições do exemplo anterior. Para tal bastar anteceder o comando SET ou RESET com a tecla SHIFT.

CONTEUDO DE UMA WORD (CANAL)

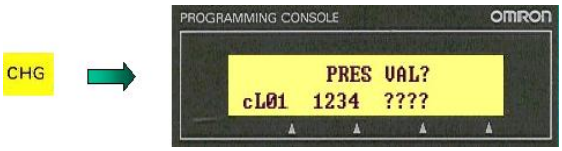
É possível monitorizar e alterar o conteúdo de uma word. Para tal é necessário executar a seguinte sequência de teclas, após ter limpo o ecrã premindo a tecla CLR.



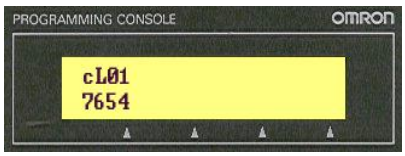
Depois de executada esta sequência, no ecrã aparece o conteúdo do canal em causa, tal como no exemplo que se segue. Visualiza-se neste caso o canal LR01, cujo conteúdo no exemplo é 1234.



Pode alterar-se o conteúdo do canal premindo a tecla CHG e digitando o novo valor seguido de WRITE. Aproveitando o exemplo,



Digitando 7654 seguido de WRITE, alteraria o conteúdo do canal LR01 para o valor digitado.



3.2. INSTRUÇÕES DE TRATAMENTO LÓGICO

3.2.1. Instruções Básicas (LD, OUT, END)

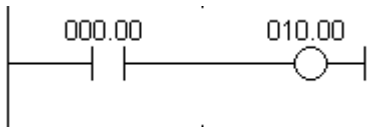
Num esquema de contactos, temos a possibilidade de colocar relés em série, em paralelo, operar com relés negados e/ou várias combinações entre estas hipóteses.

Vamos começar por analisar a programação de um circuito puramente académico, mas que nos vai servir como primeira abordagem à programação.

Imaginemos um circuito controlado por um autómato cuja lógica é a seguinte:

- O estado da saída 10.00 é dado pelo estado da entrada 0.00.

Em linguagem de contactos, teríamos a seguinte linha lógica:



A codificação em linguagem mnemónica do esquema anterior, será :

Endereço	Instrução	Dados	
00000	LD	000.00	<WRITE>
00001	OUT	010.00	<WRITE>
00002	END(01)		<WRITE>

O que seria em linguagem corrente, qualquer coisa como:

.Pega no estado do bit 000.00
.Coloca no bit 010.00
.Fim

Para programarmos este simples programa, deveremos proceder à seguinte sequência:

- Instalar a consola de programação no autómato

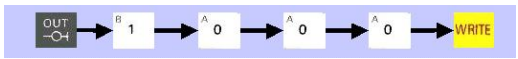
- Ligar a alimentação

- Colocar o comutador existente na consola no modo PROGRAM

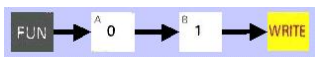
- Digitar a seguinte sequencia de PASSWORD: 

Limpar a área de memória de programa do autómato usando a seguinte sequência:

- Digitar cada linha da lista de instruções do programa. Não esquecer de validar premindo a tecla WRITE.



- A última linha do programa deverá ser : END(01)



- Usando as teclas com as setas, percorra as linhas do programa, confirmando as instruções introduzidas.

- Se tudo está conforme, passe o comutador de estado do autómato para o modo RUN.

- Não havendo anomalias, o sinalizador de RUN no frontal do autómato deve acender.

Nesta fase, o trabalho de programação está concluído, restando somente testar o correcto desempenho do programa implementado no autómato. Para o efeito, introduza um sinal de 24Vdc na entrada 0.00. O led sinalizador da entrada dever acender-se e com ele também deve acender o led referente à saída 10.00.

Para a programação deste simples exemplo, usam-se três instruções:

LD - é usada para iniciar uma linha lógica ou bloco com o status do bit especificado.
OUT - transfere o resultado das condições lógicas que antecedem esta instrução para o bit especificado.
END(01) - Indica o fim do programa. É imprescindível o seu uso.

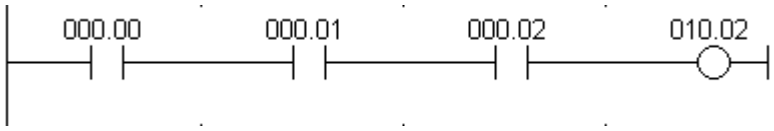
Para implementar circuitos lógicos, dispomos ainda das seguintes instruções:

AND - realiza um E lógico com o bit especificado
OR - realiza um OU lógico com o bit especificado
NOT - nega o estado do bit ao qual está associado
AND LOAD - realiza um E lógico entre dois blocos lógicos.
OR LOAD - realiza um OU lógico entre dois blocos lógicos.

3.2.2. Instrução AND

Pretende-se implementar um circuito lógico que activa a saída 10.02 do autómato, só se as entradas 0.00 e 0.01 e 0.02 estiverem activas (a ON).

O esquema de contactos correspondente será:



Em linguagem mnemónica teríamos:

Endereço	Instrução	Dados
00000	LD	000.00
00001	AND	000.01
00002	AND	000.02
00003	OUT	010.02
00004	END(01)	

O autómato ao interpretar cada linha do programa, vai executar as funções de uma forma sequencial e vai guardando de imediato o resultado lógico resultante da execução.

Exemplificando com o programa anterior e admitindo que os bits tomavam o seguinte estado,

0.00 - ON
0.01 - OFF
0.02 - ON

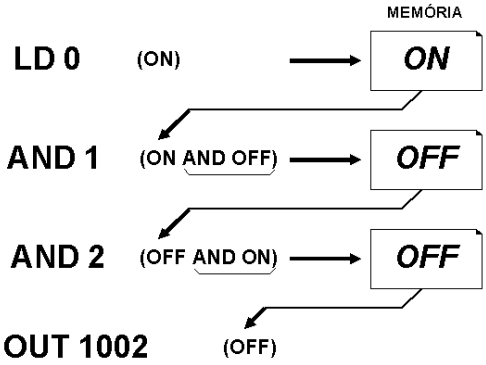
teríamos:

LD 000.00 — 0.00 está a ON. Guarda o valor ON na memória.

AND 000.01 — Executa a função AND do estado de 0.01 com o estado em memória; neste caso, OFF e(AND) ON resulta OFF. Guarda o valor OFF na memória.

AND 000.02 — Executa a função AND do estado de 0.02 com o estado em memória; neste caso, ON e(AND) OFF resulta OFF. Guarda o valor OFF na memória.

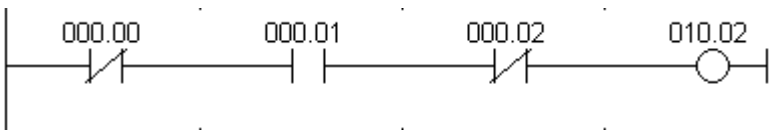
OUT 010.02 — Actualiza o estado lógico do bit 10.00 com o estado lógico contido na memoria (neste caso OFF).



3.2.3. Instrução NOT

Pegando no exemplo anterior, pretende-se que a saída 10.02 fique activa se as entradas 0.00 e 0.02 estiverem a OFF e a entrada 0.01 estiver a ON (considera-se que uma entrada está a ON quando o led que a sinaliza, se encontra ligado).

A representação em linguagem de contactos, será:



Em linguagem mnemónica seria:

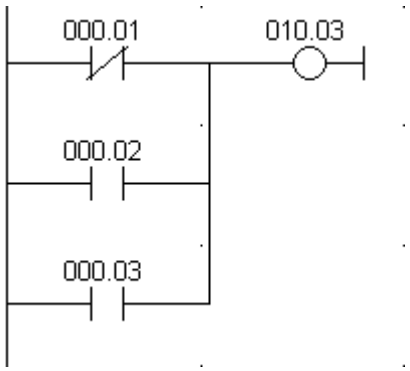
Endereço	Instrução	Dados
00000	LD NOT	000.00
00001	AND	000.01
00002	AND NOT	000.02
00003	OUT	010.02
00004	END(01)	

A função NOT pode ser usada conjuntamente com as funções LD, AND, OR e OUT.

3.2.4. Instrução OR

Pretende-se implementar um circuito lógico que active a saída 10.03 quando a entrada 0.01 estiver a OFF ou quando as entradas 0.02 ou 0.03 estiverem a ON.

O esquema de contactos correspondente ao especificado será:



A codificação em linguagem mnemónica será:

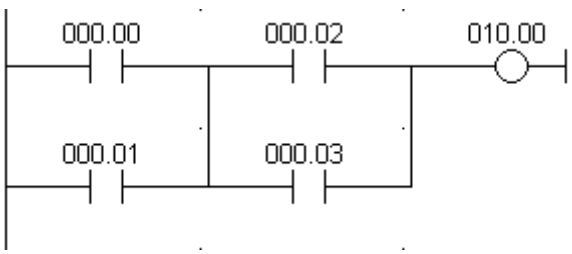
Endereço	Instrução	Dados
00000	LD NOT	000.01
00001	OR	000.02
00002	OR	000.03
00003	OUT	010.03
00004	END(01)	

3.2.5. Instrução AND LOAD

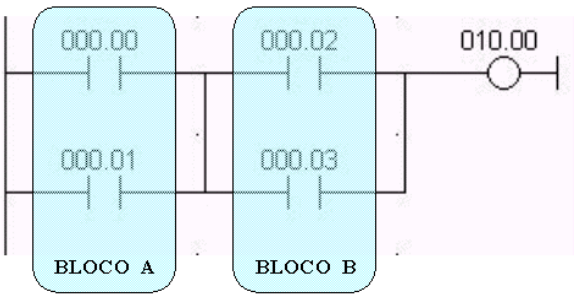
A instrução AND LOAD permite colocar em série dois blocos lógicos, ou seja, permite realizar um E lógico entre dois blocos.

Pretende-se implementar um circuito, capaz de activar a saída 10.00 sempre que a entrada 0.00 ou 0.01 estiverem a ON e as entradas 002 ou 003 estiverem também a ON.

O esquema de contactos será:



A programação deste esquema de contactos implica que façamos um tratamento diferente do circuito. Assim vamos dividir o circuito em dois blocos A e B.

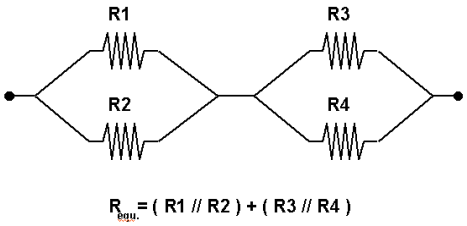


A linguagem mnemónica correspondente será:

Endereço	Instrução	Dados
00000	LD	000.01
00001	OR	000.02
00002	LD	000.03
00003	OR	000.04
00004	AND LD	
00005	OUT	010.00
00006	END(01)	

Tudo se passa como se estivéssemos a programar dois blocos lógicos independentes, e os ligássemos no fim em série (com a instrução AND LD).

Para mais fácil compreensão, vamos fazer a analogia da programação deste circuito com o cálculo de resistências equivalentes. Vamos supor que cada bit do esquema acima é uma resistência eléctrica.



Para calcularmos a resistência equivalente do circuito, teríamos de calcular primeiro o paralelo da resistência 0.00 com a resistência 0.01, seguidamente calcular o paralelo da resistência 0.02 com a resistência 0.03 e então finalmente calcular a série das resistências equivalentes anteriormente calculadas.

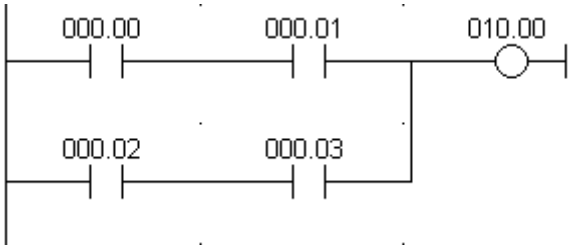
3.2.6. Instrução OR LOAD

A instrução OR LOAD permite colocar em paralelo dois blocos lógicos, ou seja, permite realizar um OU lógico entre dois blocos.

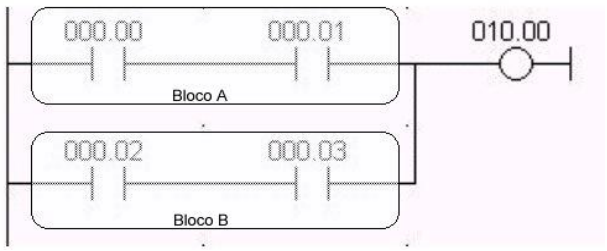
Exemplo:

Pretende-se implementar um circuito lógico capaz de activar a saída 10.00 sempre que as entradas 0.00 e 0.01 ou as entradas 0.02 e 0.03 estiverem simultaneamente a ON.

O esquema de contactos será:



A programação deste esquema de contactos implica, tal como no caso anterior, um tratamento diferente do circuito. Vamos dividir o circuito em dois blocos A e B.

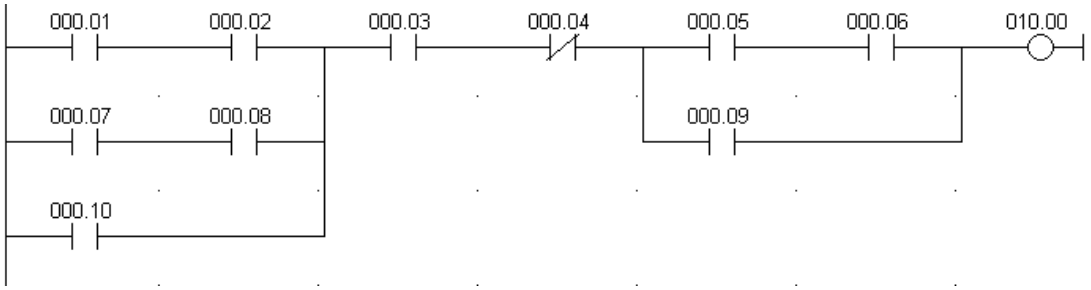


A linguagem mnemónica correspondente será:

Endereço	Instrução	Dados
00000	LD	000.01
00001	AND	000.02
00002	LD	000.03
00003	AND	000.04
00004	OR LD	
00005	OUT	010.00
00006	END(01)	

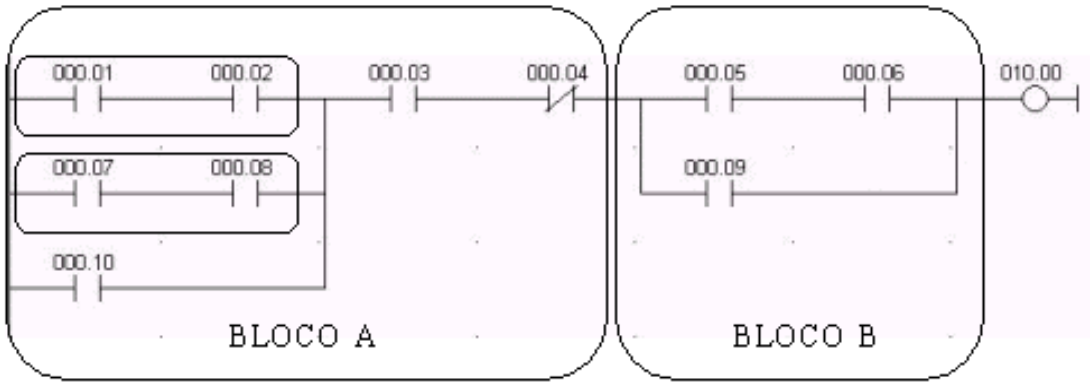
Tudo se passa como se estivéssemos a programar dois blocos lógicos independentes, e os ligássemos no fim em paralelo (com a instrução OR LD).

Exemplo 1



No diagrama anterior, há necessidade de criar vários blocos lógicos para se poder traduzir correctamente em linguagem mnemónica o esquema em causa. Note-se que uma instrução AND LD ou OR LD junta só dois blocos lógicos.

Sempre que definimos um bloco lógico, é necessário começá-lo com uma instrução LOAD.

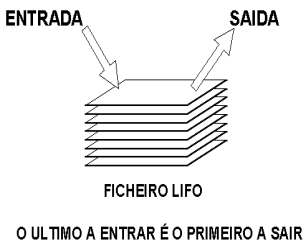


A linguagem mnemónica correspondente será:

Endereço	Instrução	Dados
00000	LD	000.01
00001	AND	000.02
00002	LD	000.07
00003	AND	000.08
00004	OR LD	
00005	OR	000.10
00006	AND	000.03
00007	AND NOT	000.04
00008	LD	000.05
00009	AND	000.06
00010	OR	000.09
00011	AND LD	
00012	OUT	010.00
00013	END(01)	

No início do capítulo indicou-se que o autómato guardava os resultados de cada operação lógica, resultantes da execução do programa, numa memória; na verdade esta memória não é única, mas faz parte de um ficheiro LIFO (last in, first out) ou seja um ficheiro onde o último valor a ser guardado, é o primeiro a sair (Tal como um monte de fichas, a primeira ficha a ser colocada no monte, é a última ser retirada).

Sempre que se inicia um bloco lógico, é adicionada uma ficha ao ficheiro, que guarda os valores lógicos decorrentes da execução do programa do bloco em causa. Sempre que o autómato encontra uma instrução AND LD ou OR LD, usa as duas últimas fichas colocadas no ficheiro e executa a operação lógica AND ou OR usando o valor lógico contido nessas fichas.

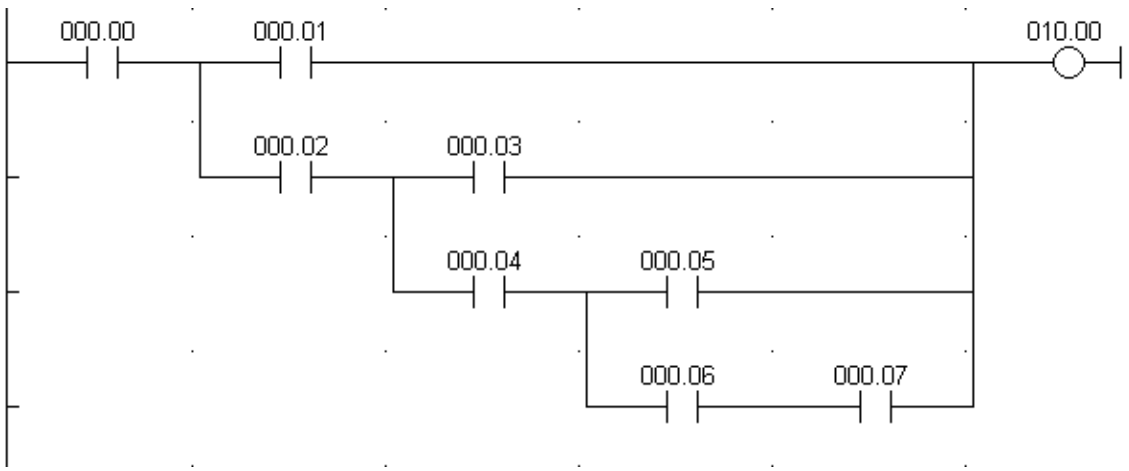


Neste autómato, o número máximo de fichas (de posições de memória) é oito.

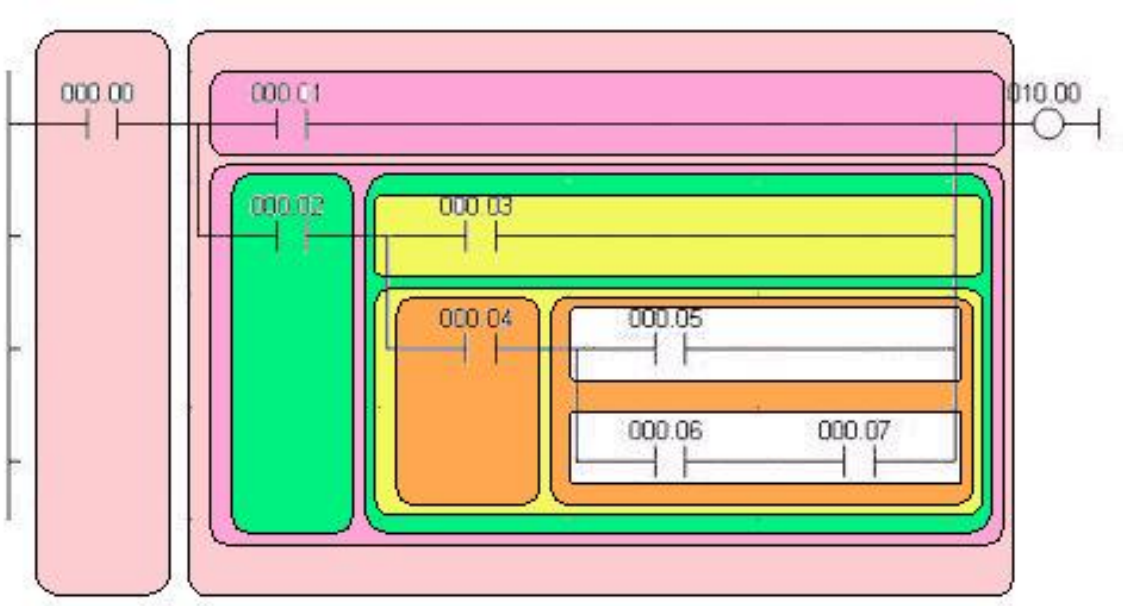
Tal como na tradução de um texto, a conversão de linguagem de contactos para mnemónica, dever ser tão fiel quanto possível. Se a codificação for bem feita, então a partir do código é possível refazer o esquema de contactos. Para tal, a leitura do esquema de contactos dever fazer-se de cima para baixo e da esquerda para a direita.

Exemplo 2

No exemplo seguinte, pretende-se a linguagem mnemónica correspondente ao esquema de contactos.



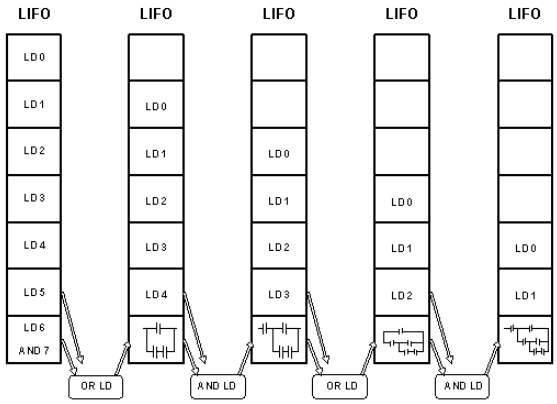
Tal como no exemplo anterior, vai ser necessário criar vários blocos lógicos que irão depois ser associados usando instruções AND LOAD e OR LOAD. Para facilitar a codificação, na figura seguinte assinalam-se os diversos blocos em que se deve subdividir o esquema.



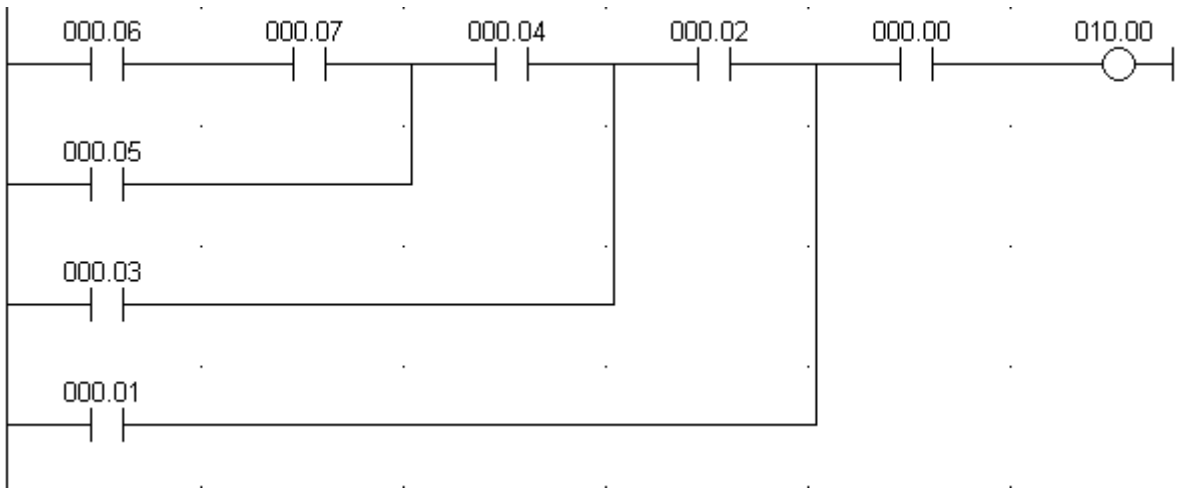
A linguagem mnemónica correspondente será:

Endereço	Instrução	Dados
00000	LD	000.00
00001	LD	000.01
00002	LD	000.02
00003	LD	000.03
00004	LD	000.04
00005	LD	000.05
00006	LD	000.06
00007	AND	000.07
00008	OR LD	
00009	AND LD	
00010	OR LD	
00011	AND LD	
00012	OR LD	
00013	AND LD	
00014	OUT	010.00
00015	END(01)	

Para melhor se entender o desempenho do ficheiro LIFO (anteriormente descrito) e a sua relação com as instruções AND LOAD e OR LOAD, mostra-se na figura seguinte a evolução dos valores lógicos correspondentes ao exemplo 2.



O exemplo exposto apresenta-se um pouco confuso no que respeita à sua codificação em linguagem mnemónica. No entanto, respeitando o esquema tal e qual ele se apresenta, a solução indicada é a única correcta. Quando não é necessário respeitar o desenho do esquema de contactos, podemos introduzir pequenas alterações, de forma a simplificar a sua programação. O esquema de contactos que se segue é um bom exemplo disso; com algumas alterações no esquema, e sem alterar a lógica, é possível reduzir o trabalho e o número de instruções para a programação do esquema inicialmente proposto neste exemplo.



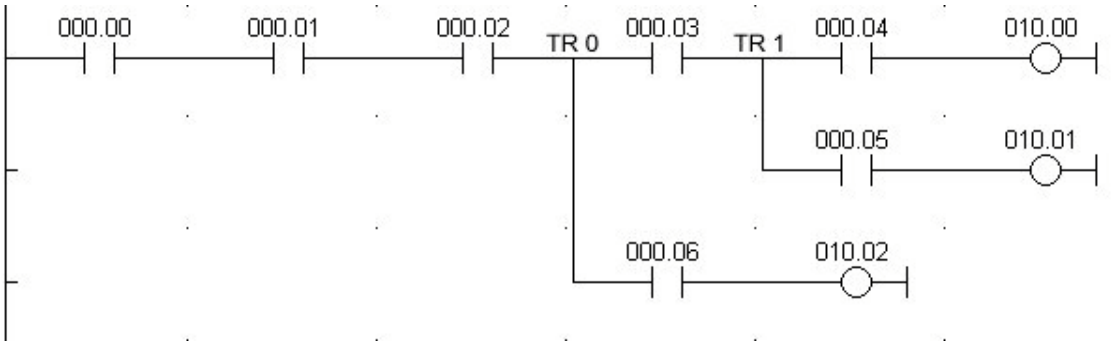
A linguagem mnemónica correspondente ao esquema anterior, seria:

Endereço	Instrução	Dados
00000	LD	000.06
00001	AND	000.07
00002	OR	000.05
00003	AND	000.04
00004	OR	000.03
00005	AND	000.02
00006	OR	000.01
00007	AND	000.00
00008	OUT	010.00
00009	END(01)	

3.2.7. Uso de TRs

Conforme já foi anteriormente referido, existem neste autómato 8 relés temporários (TR). Estes não devem ser usados como relés internos, sendo o seu uso só destinado à conversão de um esquema de contactos para linguagem mnemónica. O TR (relé temporário) serve para guardar temporariamente o estado lógico de um ponto do circuito onde este se ramifica. Esse valor lógico será posteriormente lido para programar o restante circuito, reduzindo assim o número de instruções.

Exemplo:



A linguagem mnemónica correspondente será:

Endereço	Instrução	Dados
00000	LD	000.00
00001	AND	000.01
00002	AND	000.02
00003	OUT	TR0
00004	AND	000.03
00005	OUT	TR1
00006	AND	000.04
00007	OUT	010.00
00008	LD	TR1
00009	AND	000.05
00010	OUT	010.01
00011	LD	TR0
00012	AND	000.06
00013	OUT	010.02
00014	END(01)	

Em cada malha podem usar-se até oito relés temporários (do TR0 ao TR7) e não é necessário seguir qualquer ordem sequencial no seu uso. No mesmo programa, podem usar-se os mesmos TR's quantas vezes for necessário.

3.2.8. Instrução INTERLOCK

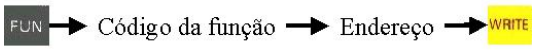
A instrução INTERLOCK (IL(02)) está sempre associada à instrução INTERLOCK CLEAR (ILC(03)), sendo esta última sempre antecedida pela primeira.

A instrução INTERLOCK é sempre antecedida de uma condição lógica que define a actuação da instrução IL(02). Quando o resultado da condição lógica que antecede IL(02) é OFF, todas as instruções OUT contidas entre esta instrução e a instrução ILC(03) tomam o estado OFF, independentemente do estado das condições que lhes dão origem; a todos os temporizadores é feito o reset. Se a condição que antecede a instrução IL(02) estiver a ON, a parte do programa entre IL(02) e ILC(03) não é afectada.

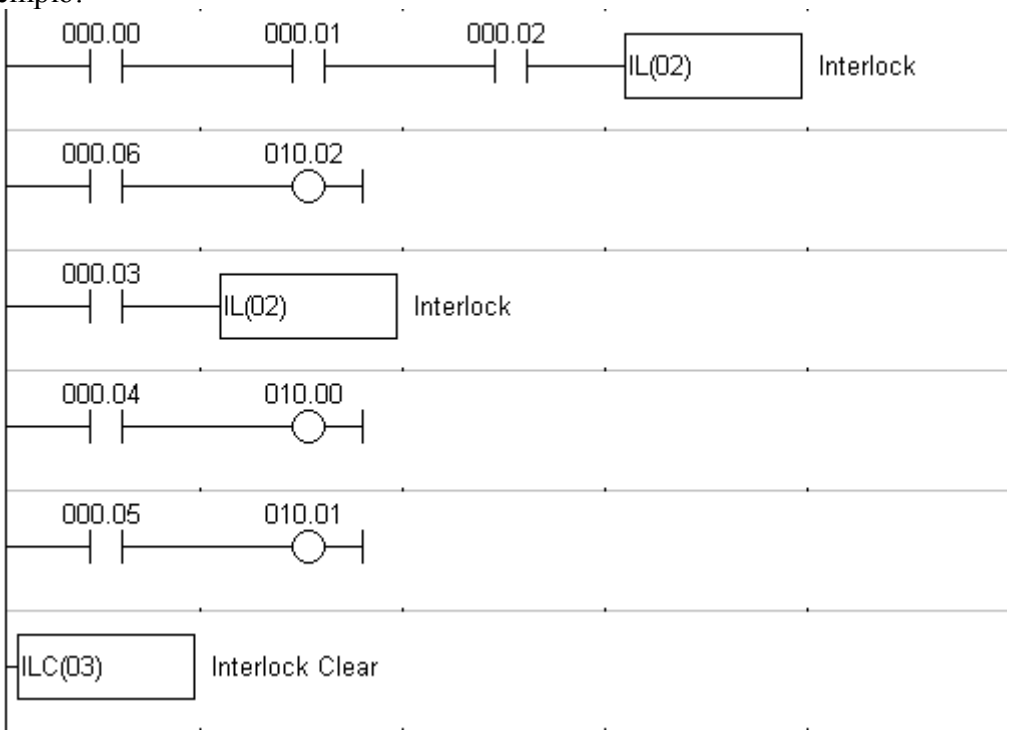
Podem usar-se várias funções IL(02) com uma só função ILC(03).

ATENÇÃO: Os relés encravados por funções KEEP(11) não são afectados por esta instrução.

Como já repararam, as instruções IL(02) e ILC(03) são representadas junto com um número entre parêntesis. Este número é o código das funções e é usado em conjunto com da tecla FUN, para programar as mesmas no autómato (tal como a instrução END(01)).



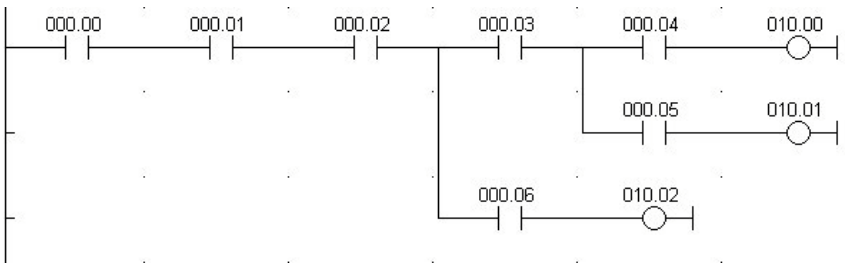
Exemplo:



A linguagem mnemónica correspondente será:

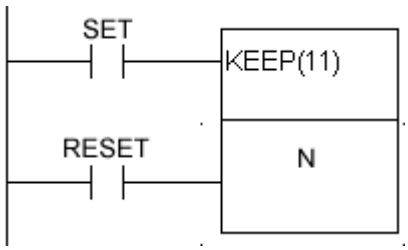
Endereço	Instrução	Dados
00000	LD	000.00
00001	AND	000.01
00002	AND	000.02
00003	IL(02)	
00004	LD	000.06
00005	OUT	010.02
00006	LD	000.03
00007	IL(02)	
00008	LD	000.04
00009	OUT	010.00
00010	LD	000.05
00011	OUT	010.01
00012	ILC(03)	
00013	END(01)	

A instrução INTERLOCK pode ser usada como alternativa aos TR's; o circuito anterior tem um desempenho idêntico ao esquema que se segue:

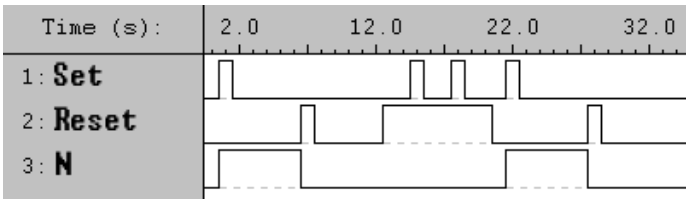


3.2.9. Instrução KEEP(11)

A instrução KEEP(11), permite definir um relé como biestável, sendo o seu estado definido por duas condições lógicas; uma de SET e outra de RESET. O relé especificado na instrução ficará activo desde que a condição de SET tenha tomado o valor ON. O relé só desactivará quando existir um valor ON na condição de RESET. Caso haja simultaneidade das duas condições a ON, é a condição de RESET a predominante.

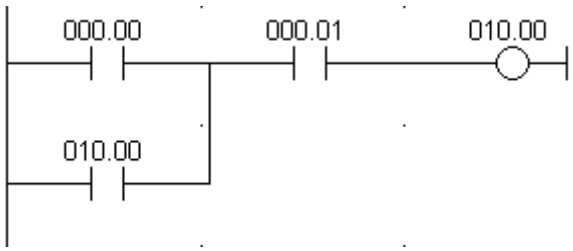


Abaixo apresenta-se um diagrama de tempos exemplificativo.



Exemplo:

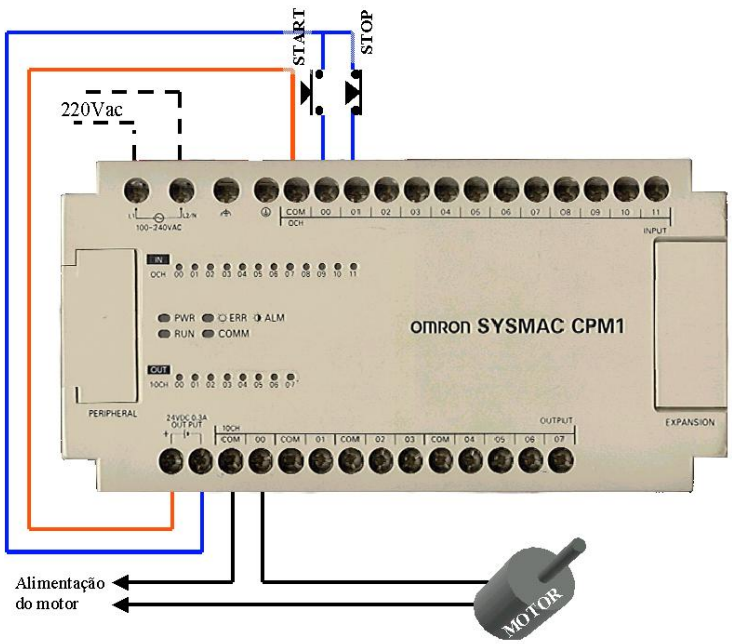
É frequente encontrar comandos de motores que dispõem de duas botoneiras; uma para ligar e outra para desligar. Implementando um circuito semelhante usando um autómato, poderíamos ter o seguinte diagrama de contactos:



Ao relé 0.00 seria ligada a botoneira de ARRANQUE e ao relé 0.01 a botoneira de PARAGEM. O motor seria comandado pelo relé 10.00.

Normalmente, por questões de segurança, os circuitos de comando de desactivação de equipamentos, usam botoneiras com contacto normalmente fechado. Se por qualquer razão houver uma ruptura nas ligações dos comandos, isso implica a interrupção do circuito e consequentemente a imediata paragem dos equipamentos. Por essa razão, vamos usar no exemplo uma botoneira de paragem normalmente fechada.

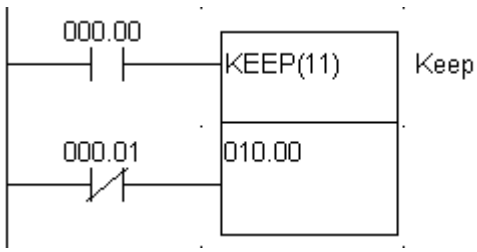
A ligação dos vários elementos ao autómato seria como se ilustra na figura que se segue.



Em linguagem mnemónica teríamos o seguinte programa:

Endereço	Instrução	Dados
00000	LD	000.00
00001	OR	010.00
00002	AND	000.01
00003	OUT	010.00
00004	END(01)	

Usando a instrução KEEP(11) poder-se-ia programar o mesmo controlo de uma forma mais simples:



A codificação em linguagem mnemónica seria:

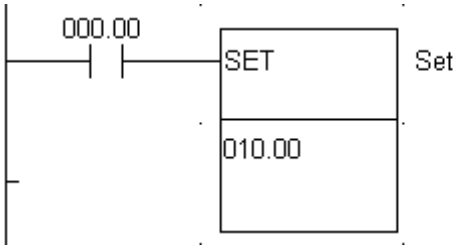
Endereço	Instrução	Dados
00000	LD	000.00
00001	LD NOT	000.01
00002	KEEP(11)	010.00
00003	END(01)	

Nota: Sempre que existam vários blocos lógicos a controlar uma instrução, estes são programados em primeiro lugar e só depois é programada a instrução em causa.

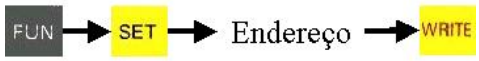
3.2.10. Instruções SET e RESET

Em alternativa à instrução KEEP(11) que congrega as condições de activação e desactivação de um bit, existem duas instruções que permitem manipular o estado de um bit, em circunstâncias semelhantes. Essas instruções são SET e RESET.

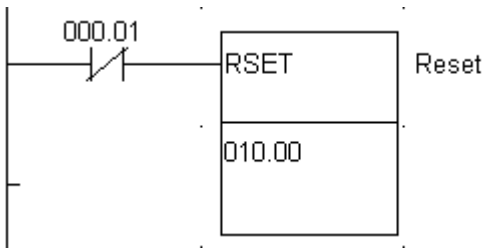
Quando a condição lógica que antecede a instrução SET vai a ON (ilustrada no exemplo seguinte pelo bit 0.00) , o bit manipulado pela instrução (10.00) é também levado a ON, mantendo-se nesse estado.



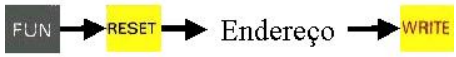
Para programar a função SET usando a consola de programação, executar a seguinte sequência de teclas:



Situação semelhante acontece com a instrução RESET, pois quando a condição lógica que antecede esta instrução vai a ON (ilustrada no exemplo pelo bit 0.02) , o bit manipulado (10.00) é em simultâneo levado a OFF, permanecendo nesse estado.

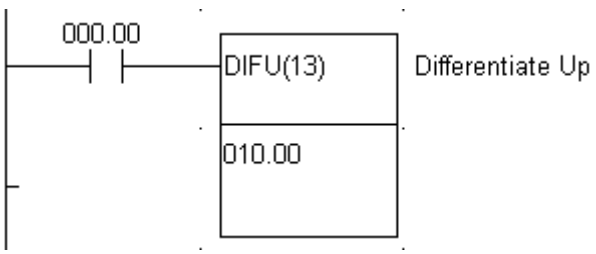


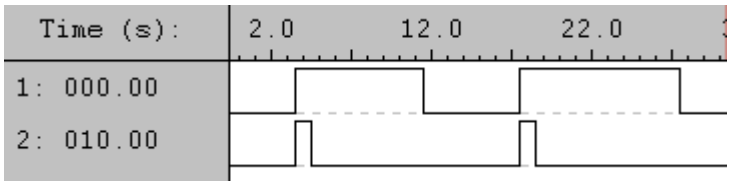
Para programar a função RESET usando a consola de programação, executar a seguinte sequência de teclas:



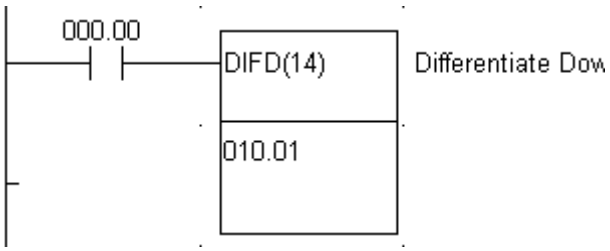
3.2.11. Instruções DIFU(13) e DIFD(14)

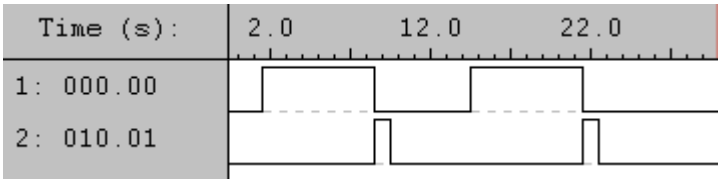
A instrução DIFU(13) permite activar um relé durante um ciclo de scan, sempre que a condição lógica que antecede a instrução, transita do estado OFF para ON.





A função DIFD(14) permite activar um relé durante um ciclo de scan, sempre que a condição lógica que antecede a instrução, transita de um estado ON para OFF.



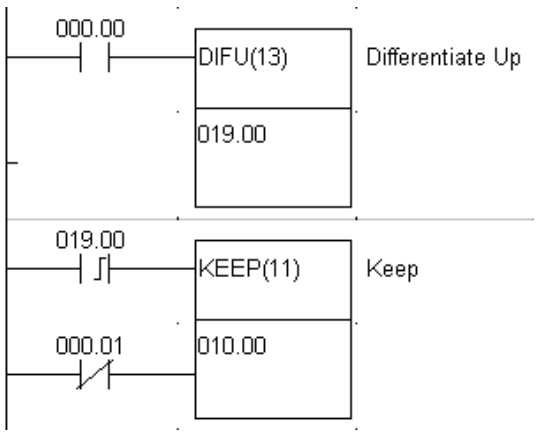


Retomando o exemplo apresentado na instrução anterior (comando de um motor), vamos requintá-lo por forma a aumentar a segurança do comando, aproveitando para o efeito a instrução DIFU(13).

Vamos admitir que a botoneira de arranque encrava, mantendo o circuito fechado. Quando o operador pretender parar o motor, este realmente pára, mas só enquanto a botoneira de paragem estiver premida; logo que o operador larga a botoneira, o arranque do motor é inevitável, pois o sinal de arranque está presente.

Usando a instrução DIFU(13) podemos conseguir um sinal muito breve num relé auxiliar, que por sua vez activa a instrução KEEP(11). Desta forma, a ordem de ligar o motor só acontece quando houver uma transição de OFF para ON no sinal de arranque.

O esquema de contactos que se segue mostra o comando do motor já dotado deste melhoramento.



A codificação em linguagem mnemónica seria:

Endereço	Instrução	Dados
00000	LD	000.00
00001	DIFU(13)	019.00
00002	LD	019.00
00003	LD NOT	000.01
00004	KEEP(11)	010.00
00005	END(01)	

3.3. TEMPORIZADORES E CONTADORES

Ao falarmos das áreas de relés no início do capítulo, fez-se referência à área de TC. Esta área é partilhada por Temporizadores e Contadores. A cada número TC podemos associar um só temporizador ou contador.

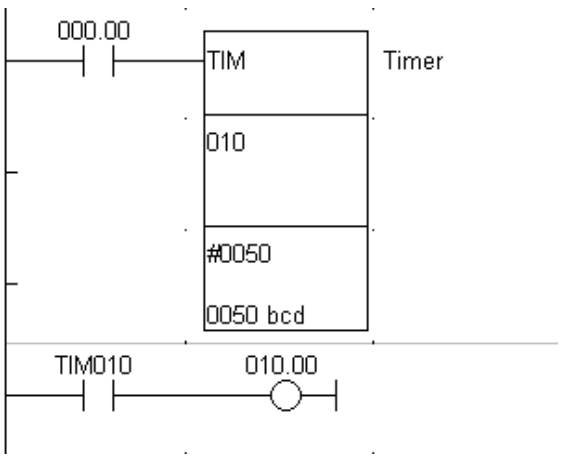
3.3.1. TEMPORIZADORES

Para definir um temporizador existem duas instruções disponíveis: TIM e TIMH(15).

A instrução TIM permite definir um temporizador de atraso à operação com a precisão de 0.1 segundo podendo este ter um alcance máximo de 999.9 segundos. O valor de PRESET (tempo inicial) pode ser especificado por uma constante ou pelo conteúdo de uma word. Associado a cada temporizador existe um contacto TIM N (sendo N o número do temporizador).

A instrução TIM é sempre antecedida por uma condição lógica, que estando a ON activa o temporizador; este começa a decrementar o tempo pre-seleccionado e quando atinge o zero, fecha o contacto TIM N. Se a condição lógica passar a OFF, implica o RESET do temporizador e consequentemente a abertura do contacto TIM N.

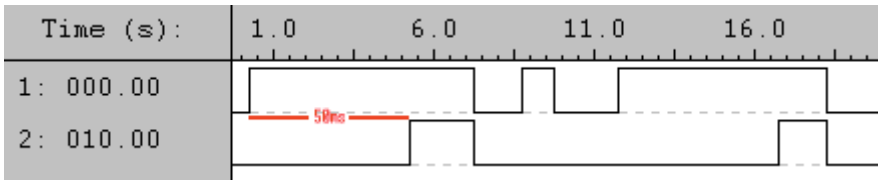
No exemplo seguinte pode ver-se a representação em linguagem de contactos da instrução TIM.



A codificação em linguagem mnemónica seria:

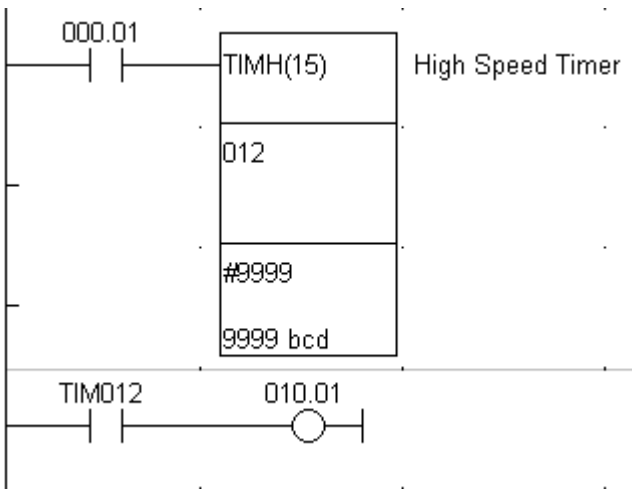
Endereço	Instrução	Dados
00000	LD	000.00
00001	TIM	10
		#0050
00002	LD	TIM 10
00003	OUT	010.00
00004	END(01)	

O diagrama de tempos correspondente será:



A instrução TIM programa-se usando uma tecla própria para o efeito, existente na consola de programação.

Para programar a instrução TIMH(15) é necessário usar a tecla FUN e o código 15. Esta instrução permite implementar um temporizador idêntico ao implementado pela instrução TIM, com a diferença de que este tem uma precisão de 0.01 segundo e um alcance máximo de 99.99 segundos. O contacto deste temporizador tem a designação TIM N tal como na instrução TIM.



Usando a instrução TIM ou TIMH podem implementar-se temporizações de atraso à desoperação, temporizações mistas (atraso à operação e desoperação) e temporizações por impulso.

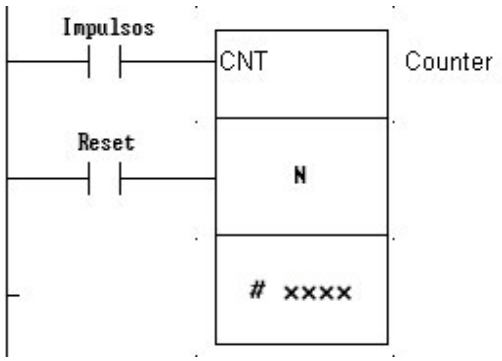
3.3.2. CONTADORES

Há várias métodos para implementar contadores. Neste capítulo vamos apenas abordar a programação da instrução CNT e CNTR(12).

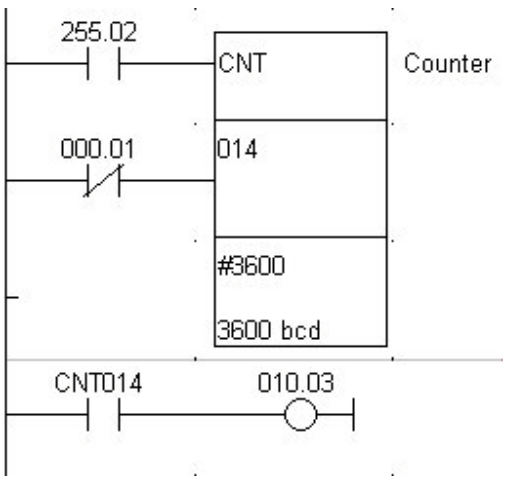
A instrução CNT permite a programação de um contador decrescente. Este é identificado com um número, tal como acontece nos temporizadores. É especificado também o valor de PRESET que pode ser uma constante ou o valor contido numa word.

A instrução CNT está associada a duas condições lógicas; Na primeira quando acontece uma transição de estado de OFF para ON, faz decrementar em uma unidade o conteúdo do contador. Na segunda, sempre que está a ON, faz o RESET ao contador e consequentemente o conteúdo do contador toma o valor de PRESET. A cada contador está associado um contacto CNT N (sendo N o número do contador), que vai a ON sempre que o contador toma o valor ZERO. Quando o contador atinge o valor zero, permanece nesse valor até que seja efectuado o RESET ao contador.

Na figura seguinte vê-se a representação de um contador em linguagem de contactos:



Um pormenor importante de se referir, é que ao contrário dos temporizadores, os contadores retêm o seu conteúdo, mesmo após a falha de alimentação do autómato. Tirando partido deste pormenor, pode implementar-se um temporizador com retenção do tempo decorrido, usando para o efeito um contador e um relé de clock da área de relés especiais, conforme se mostra no exemplo seguinte:



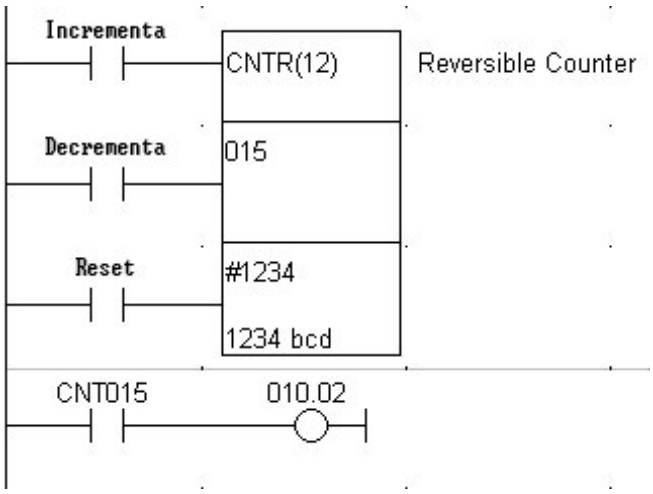
A codificação em linguagem mnemónica seria:

Endereço	Instrução	Dados
00000	LD	255.02
00001	LD NOT	000.01
00002	CNT	14
		#3600
00003	LD	CNT 14
00004	OUT	10.03
00005	END(01)	

A instrução CNTR(12) permite programar um contador reversível. Tal como na instrução CNT, este é identificado com um número. É especificado também o valor de PRESET que pode ser uma constante ou o valor especificado por um canal.

A instrução CNTR(12) tem associadas três condições lógicas. Na primeira condição lógica, uma transição de OFF para ON faz incrementar o valor do contador. Na segunda condição, uma transição de OFF para ON faz decrementar uma unidade ao valor do contador. A terceira condição lógica faz o RESET ao contador, sempre que esteja a ON. O RESET neste contador faz com que o seu conteúdo vá a zero.

Associado ao CNTR(12) há um contacto de um relé que é designado tal como nos contadores anteriormente descritos (CNT X ;sendo X o nº atribuído ao contador). Esse contacto vai a ON sempre que há uma transição de 0 para o valor de preset ou deste para 0.



3.4. INSTRUÇÕES DE TRATAMENTO DE DADOS

Nos parágrafos anteriores, vimos instruções que operam com condições lógicas. Vamos agora ver instruções que operam sobre words, ou seja, sobre a informação contida num conjunto de 16 bits. Num bit, podemos ter apenas dois estados: ON ou OFF. Num conjunto de 16 bits, vai ser possível codificar mais informação.

Antes de apresentar algumas instruções de tratamento de dados, vamos analisar os sistemas numéricos.

3.4.1. SISTEMAS NUMÉRICOS

O sistema numérico que melhor conhecemos e que usamos diàriamente é o SISTEMA DECIMAL. Tal como o nome indica, este sistema numérico dispõe de dez símbolos, uados na seguinte sequência: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9. Quando se esgotam os símbolos, repete-se a sequência, colocando o símbolo seguinte à esquerda. Se o símbolo à esquerda é um 0, então este não se representa. Para melhor explicar esta mecânica, vamos ver o exemplo abaixo. Neste exemplo, representam-se os 0 à esquerda só para que seja mais fácil compreensão.

```
0 0 0
0 0 1
0 0 2
.....
0 0 9   Esgotados os símbolos, repete-se
0 1 0   a sequência colocando à esquerda
0 1 1   o símbolo seguinte.
0 1 2
.....
0 1 9
0 2 0
.....
0 9 9
1 0 0
.....
```

Compreendida esta mecânica, facilmente perceberemos o que se passa com os outros sistemas numéricos.

Num circuito eléctrico, facilmente podemos definir dois estados: ligado/desligado ou com tensão/sem tensão. Podem-se então usar estes dois estados como base de um sistema numérico; um sistema binário. Para simplificar a representação dos estados ligado e desligado usam-se os símbolos 1 e 0 respectivamente.

Podemos codificar valores numéricos numa base binária, usando a mecânica que já foi anteriormente explicada para o sistema decimal.

0 0 0	=	0
0 0 1	=	1
0 1 0	=	2
0 1 1	=	3
1 0 0	=	4
1 0 1	=	5
1 1 0	=	6
1 1 1	=	7

À frente da representação binária, colocou-se o valor decimal correspondente.

Para converter um valor binário em decimal, somam-se os pesos dos dígitos.

$$\text{Valor} = 1^{\text{o}}\ \text{dígito} \times 2^{0} + 2^{\text{o}}\ \text{dígito} \times 2^{1} + 3^{\text{o}}\ \text{dígito} \times 2^{2} + \ldots$$

$$\text{Valor} = 1^{\text{o}}\ \text{dígito} \times 1 + 2^{\text{o}}\ \text{dígito} \times 2 + 3^{\text{o}}\ \text{dígito} \times 4 + \ldots$$

Para que a conversão de um valor binário em decimal seja mais fácil, existe um sistema de codificação que usa quatro bits para codificar valores de 0 a 9, desaproveitando as restantes 6 combinações possíveis. Esta forma de codificar valores numéricos em binário chama-se BCD.

Para codificar o valor decimal 947, temos a seguinte codificação BCD:

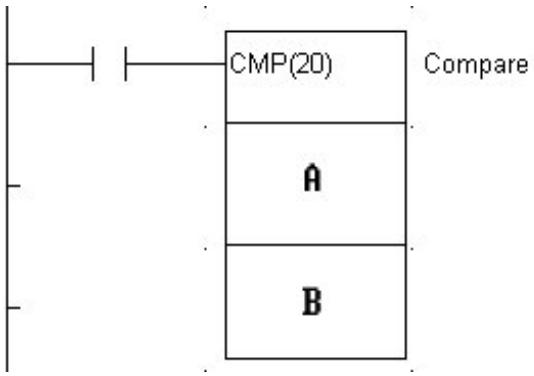
15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	0	0	0	1	0	0	1	0	1	0	0	0	1	1	1
0				9				4				7			

Note-se que a codificação BCD pelo simples facto de desperdiçar 6 combinações do binário, reduz o valor máximo codificado numa word: 9999 em BCD, contra 65535 em Binário.

3.4.2. Instrução CMP(20)

Esta instrução permite comparar dois valores numéricos sendo o resultado dado pelo estado de três relés especiais.

A instrução CMP(20) é sempre antecedida por uma condição lógica que quando está a ON permite a execução da comparação.



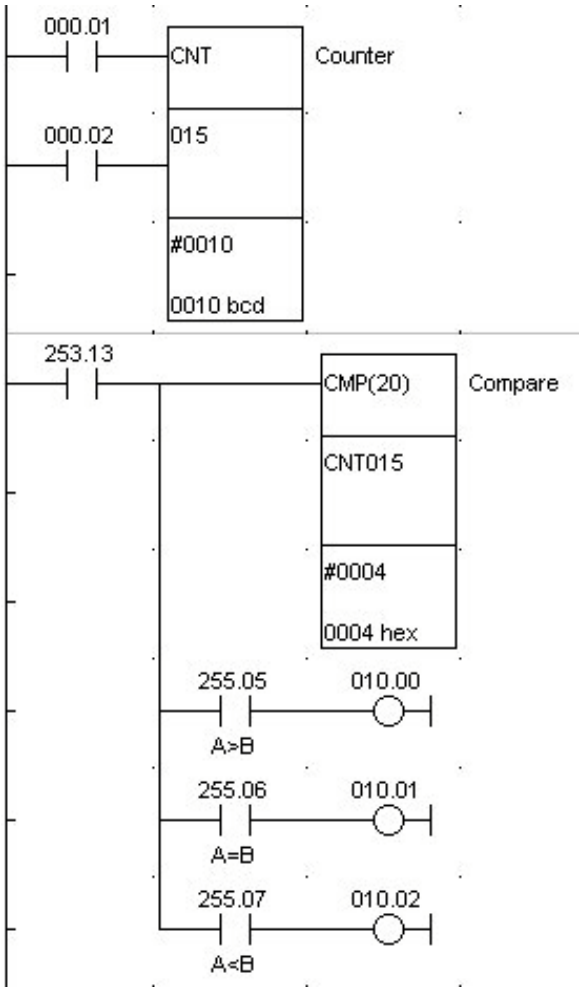
Sempre que esta instrução é executada, é comparado o valor contido em A, com o valor contido em B. A e B podem conter uma constante ou o conteúdo de uma word, temporizador ou contador.

- Se A>B então o relé 25505 vai a ON.

- Se A=B então o relé 25506 vai a ON.

- Se A<B então o relé 25507 vai a ON.

No exemplo que se segue, compara-se sempre (usa-se para o efeito um relé sempre a ON – 253.13) o conteúdo do contador 15 com a constante 4. Conforme o valor do contador 15 teremos activo o relé 10.00 se o contador tiver um valor maior que 4; o relé 10.01 passará a ON se o contador 15 tiver o valor 4; ser o relé 10.02 a irá a ON se o CNT15 tiver um valor inferior a 4.

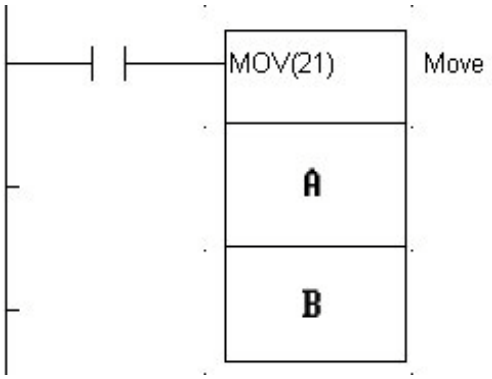


A codificação em linguagem mnemónica seria:

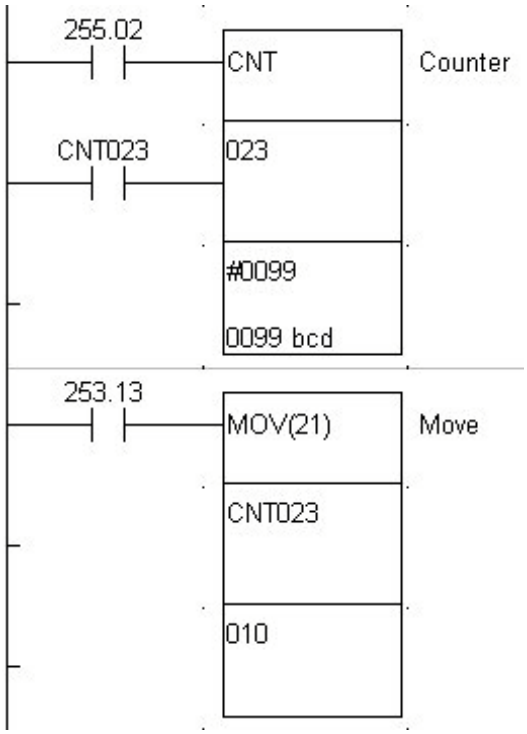
Endereço	Instrução	Dados
0000	LD	000.01
0001	LD	000.02
0002	CNT	15
		#0010
0003	LD	253.13
0004	OUT	TR0
0005	CMP(20)	
		CNT15
		#0004
0006	AND	255.05
0007	OUT	010.00
0008	LD	TR0
0009	AND	255.06
0010	OUT	010.01
0011	LD	TR0
0012	AND	255.07
0013	OUT	010.02
0014	END(01)	

3.4.3. Instrução MOV(21)

A instrução MOVE permite copiar o valor contido em A para o destino expresso em B, sempre que a condição lógica que antecede esta instrução esteja a ON."A" pode ser um canal, um temporizador/contador ou uma constante. "B" pode ser um canal ou temporizador/contador.



No exemplo que se segue, a instrução MOV(21) copia o conteúdo do contador 23 para o canal de saídas (canal 10). O contador decrementa ao ritmo do impulso gerado pelo relé 255.02. Quando o contador chega a 0 automaticamente é resetado e volta ao valor inicial. Para que a instrução MOV(21) seja sempre executada, é antecedida por um relé sempre a ON (253.13).

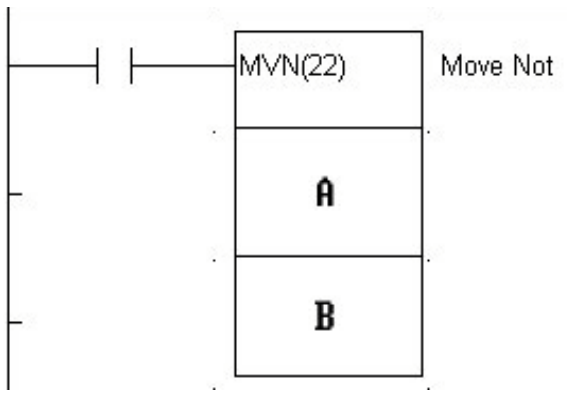


A codificação em linguagem mnemónica seria:

Endereço	Instrução	Dados
0000	LD	255.02
0001	LD	CNT23
0002	CNT	23
		#0099
0003	LD	253.13
0004	MOV(21)	
		CNT23
		002
0005	END(01)	

3.4.4. Instrução MOVN(21)

A instrução MOVE NOT permite copiar o conteúdo negado de A para o destino expresso em B, sempre que a condição lógica que antecede esta instrução esteja a ON.



Tal como na instrução MOV(21), em "A" pode ser especificado um canal, um temporizador/contador ou uma constante e em "B" pode ser especificado um canal ou temporizador/contador.

3.4.5. Instrução SFT(10)

A instrução SHIFT permite implementar um registo deslocamento começando na word A e acabando na word B.

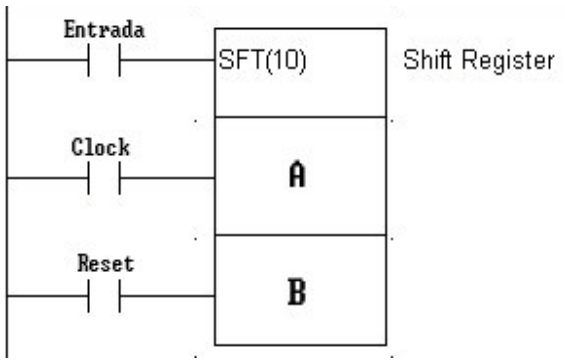
A word A deverá ter um endereço menor ou igual à word B e as duas devem pertencer à mesma área de memória.

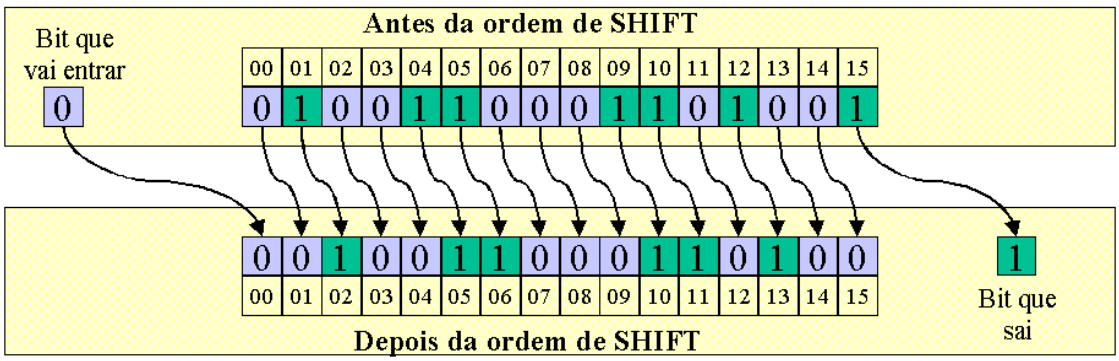
Esta instrução é controlada por três condições lógicas:

- A primeira, define o estado do bit que "entra" no canal A. No momento do deslocamento, o bit 0 da word A terá o estado lógico desta condição.

- A segunda, define o momento do deslocamento, que se verifica sempre que houver uma transição de OFF para ON nesta condição.

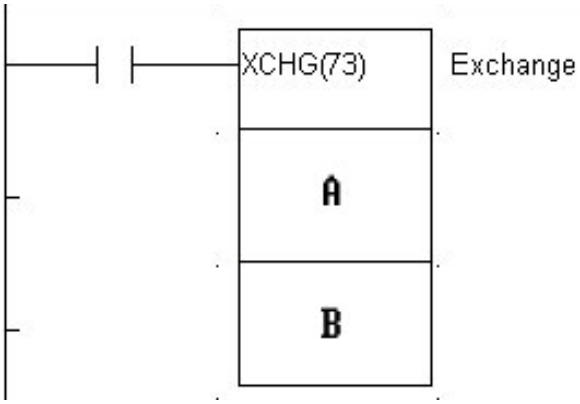
- A terceira, é a condição de RESET. Quando o seu estado é ON, os bits das words afectadas pela instrução SHIFT, são forçados a OFF.



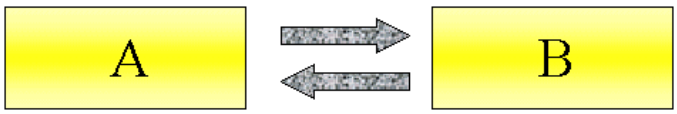


3.4.6. Instrução XCHG(73)

Sempre que a condição que antecede a instrução é verdadeira, XCHG(73) troca o conteúdo dos dois canais operandos desta instrução.



Por cada scan efectuado com a condição de execução activa, esta instrução coloca o conteúdo do canal especificado em A no canal especificado em B e o conteúdo de B em A.



3.5. PROGRAMAÇÃO DE PROCESSOS SEQUENCIAIS

São inúmeras as situações de comando onde as tarefas a executar se repetem sempre na mesma sequência ou em sequências pré-determinadas. No autómato dispomos de diversas "ferramentas" para levar a cabo a programação de processos sequenciais;

- A primeira forma de o fazer é usando a função KEEP(11) ou uma forma mais primária de "encravar" relés de controlo, recorrendo a encravamentos discretos ou recorrendo às funções SET/RESET.

- A segunda alternativa passa pelo uso da função SFT(10). Esta função está particularmente adaptada para a programação de sequências lineares.

- Existe uma outra alternativa que é usando duas funções dedicadas à implementação de GRAFCET: STEP(08) e SNXT(09).

Em todos estes métodos há um factor comum: existe um relé por cada estado da sequência, que quando está a ON indica a activação desse estado. Quando se verifica a condição de transição, o relé que está a ON passa a OFF e em simultâneo o relé que sinaliza o estado seguinte passa a ON. Este método é semelhante a uma corrida de estafetas; existe um testemunho que transita de atleta para atleta, conferindo a cada um deles a legitimidade da corrida.

Para concretizar cada método, vamos usar um exemplo muito simples, mas que vai permitir um melhor esclarecimento da mecânica usada.

No nosso exemplo vamos implementar o controlo de três electroválvulas A, B e C. Num determinado processo, estas electrovávulas trabalham na sequência que se descreve a seguir:

1º estado - Nenhuma electroválvula está ligada
A transição para o estado seguinte acontece quando a botoneira *Bot* for premida.

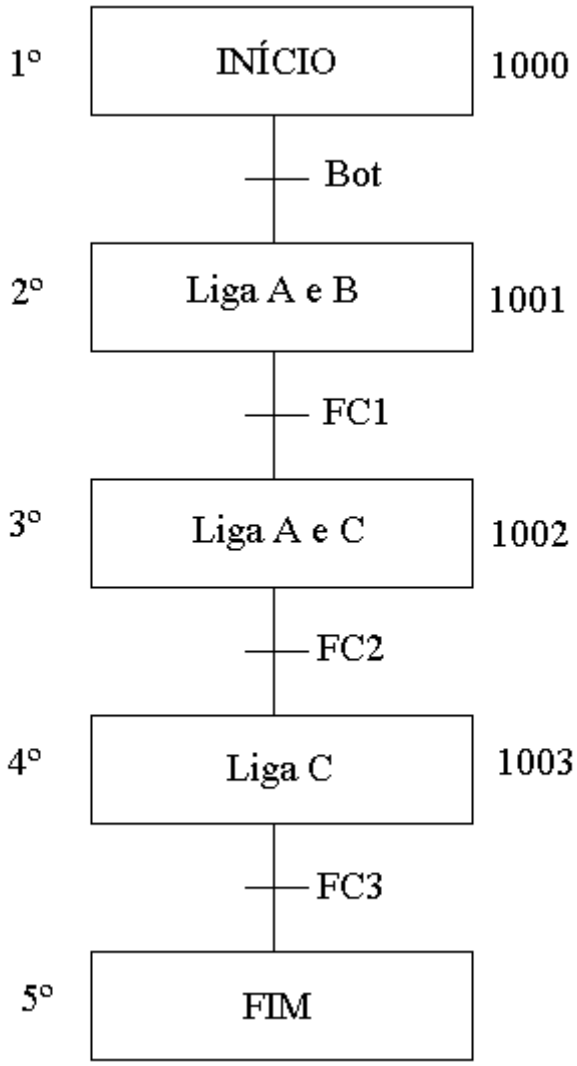
2º estado - As electroválvulas A e B estão activadas.
A transição para o estado seguinte acontece quando o fim-de-curso FC1 for actuado.

3º estado - As electroválvulas A e C estão activadas.
A transição para o estado seguinte acontece quando o fim-de-curso FC2 for actuado.

4º estado - A electroválvula C está activada.
A transição para o estado seguinte acontece quando o fim-de-curso FC3 for actuado.

5º estado - Nenhuma electroválvula está ligada.
Estado final.

Em grafcet teríamos qualquer coisa como:

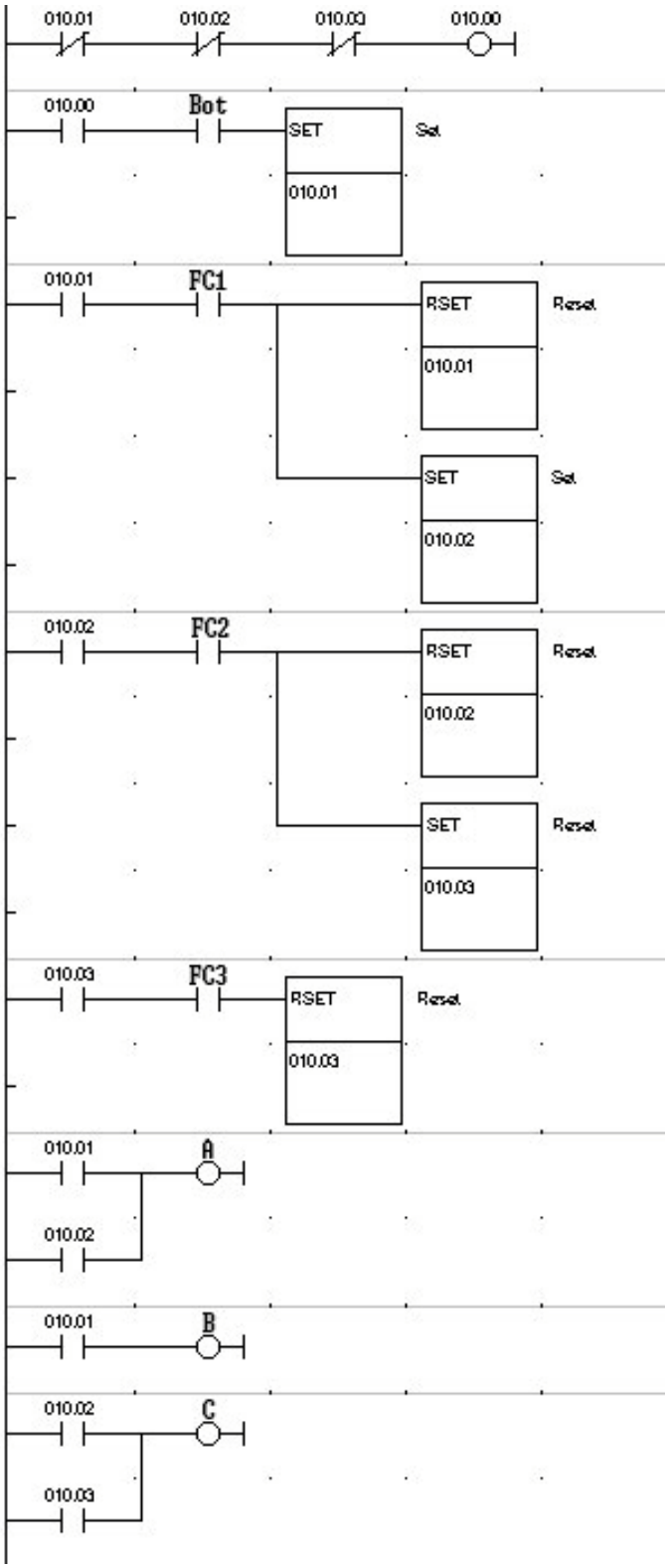


3.5.1. IMPLEMENTAÇÃO USANDO AS FUNÇÕES SET / RESET

Usando funções SET e RESET pode-se implementar o problema acima descrito, criando um relé (auxiliar) que sinaliza o estado que se encontra activo. Pode por exemplo, atribuir-se aos relés 10.00, 10.01, 10.02 e 10.03 essa tarefa (o 5º estado não necessita de relé de controlo). Sabemos entretanto que o relé 10.00, que sinaliza o estado 1, fica activo até que o sinal da botoneira fique presente. Nesse momento o relé 10.00 passa a OFF e em simultâneo é activado o relé 10.01 que sinaliza o estado 2. O mesmo procedimento seria usado para activar os restantes relés da sequência.

Resumindo, a condição de transição quando satisfeita faz o SET do relé que sinaliza o estado seguinte. Em simultâneo a condição de transição faz o RESET do relé que sinaliza o estado presente.

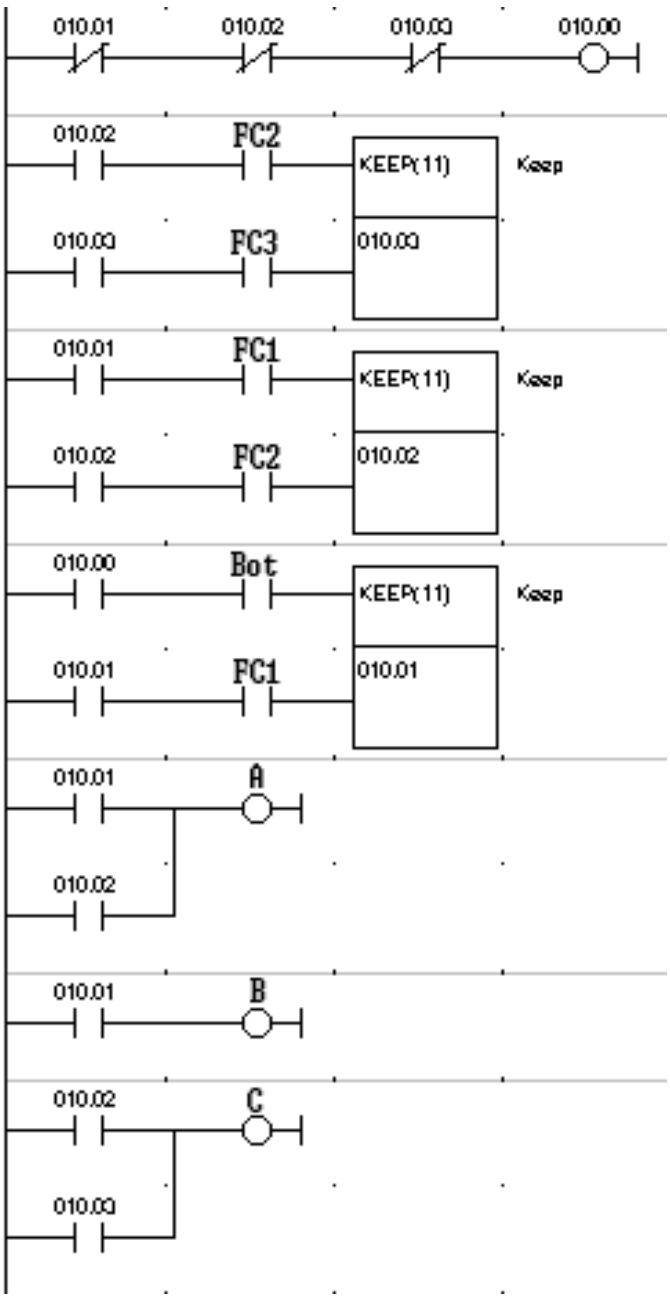
Abaixo apresenta-se o esquema de contactos respectivo.



3.5.2. IMPLEMENTAÇÃO USANDO A FUNÇÃO KEEP(11)

Usando funções KEEP(11) pode-se também implementar o problema anterior usando o mesmo princípio. Resumindo, a condição de transição quando satisfeita faz o SET do relé que sinaliza o estado seguinte e em simultâneo, a condição de transição faz o RESET do relé que sinaliza o estado presente.

Abaixo apresenta-se o esquema de contactos respectivo.



3.5.1. IMPLEMENTAÇÃO USANDO A FUNÇÃO SFT(10)

Este mesmo problema pode ser implementado usando a função SHIFT REGISTER (SFT(10)). Para tal, no canal que contem os relés de controlo, vai assegurar-se que em qualquer instante está um só bit a ON. Ao usar a instrução SFT(10) para implementar uma sequência, convém que os relés, que servem de controlo aos estados, sejam contíguos. Assim, vamos usar os relés do canal 10 para implementar o controlo (tal como no exemplo anterior).

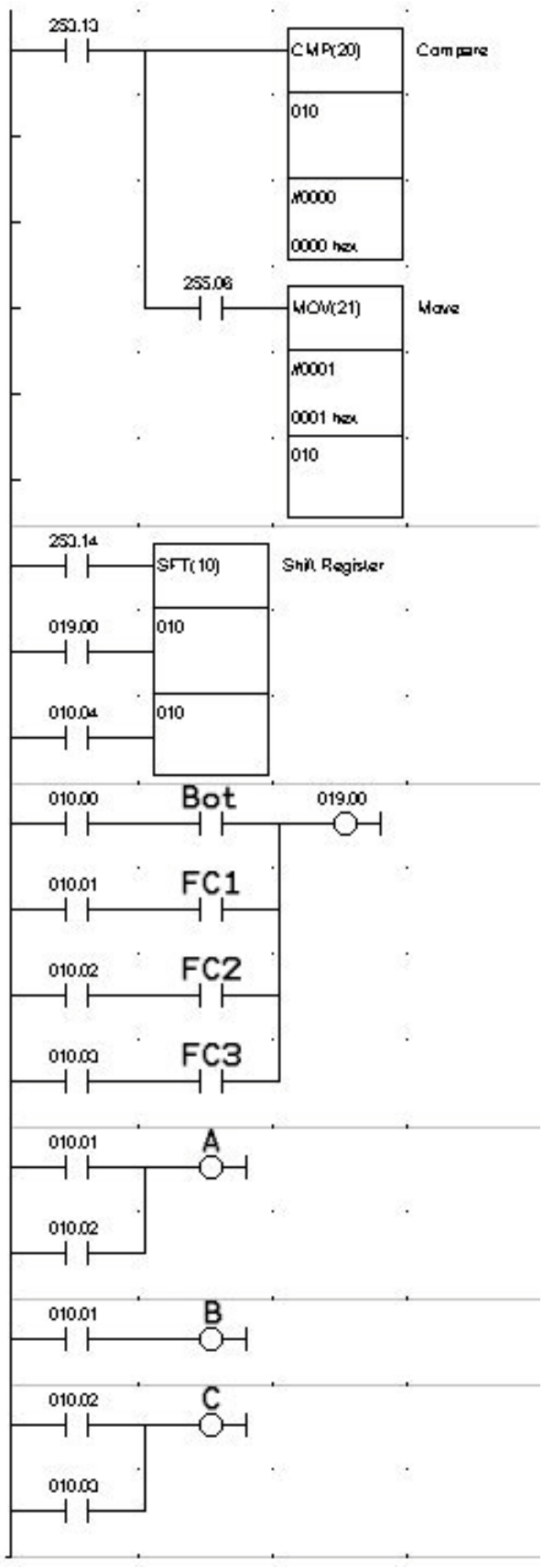
Para nos assegurarmos de que não existe nenhum relé a ON nesse canal quando activamos o 1º estado, faz-se uma comparação do canal 10 com #0000 (constante zero). Se o resultado da comparação for igual, significa que não há nenhum relé a ON no canal e por isso pode-se activar o 1º estado (estado inicio). A activação do relé 10.00 (relé que controla o primeiro estado) pode ser feita executando a instrução MOV(21) e transferindo #0001(constante um) para o canal 10.

Em seguida são programadas as condições de transição, que sempre que são verdadeiras activam a ON o relé auxiliar 19.00. Este relé é usado na instrução SFT(10) na entrada de clock, o que quer dizer que sempre que este relé vai a ON, vai activar o deslocamento dos bits do canal 10. Desta forma, consegue-se a desactivação de um relé e a activação do imediatamente seguinte.

Note-se que o relé 10.04, correspondente ao estado final, é também usado no comando da instrução SFT(10). Quando este relé vai a ON, activa o RESET do canal controlado pelo shift register, levando a OFF todos os relés desse canal, e consequentemente originando a inicialização do processo.

No final são programadas as saídas de comando. Agrupam-se a cada saída os relés de controlo de estados, de forma a activar a saída de acordo com o grafcet.

Este método de programação é bastante eficaz em sequências predominantemente lineares, com muitos estados.



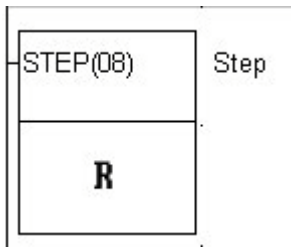
3.5.2. IMPLEMENTAÇÃO USANDO AS FUNÇÕES STEP(08) e SNXT(09)

Existem em alguns autómatos, instruções dedicadas à implementação de grafcet. No autómato OMRON CPM1 elas são as funções STEP e STEP NEXT.

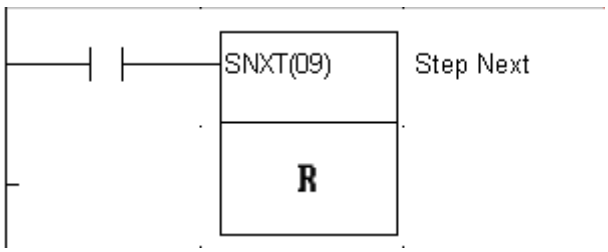
A função STEP(08) é programada directamente do barramento esquerdo e como tal não é antecedida por qualquer condição lógica. Esta função pode ser programada de duas formas: com um relé como operando e sem relé.

Quando usada com um relé (R), a função STEP(08) define o início de uma secção do programa designada ESTADO.

Quando usada sem relé de operando, a função STEP(08) define o fim da implementação de grafcet no programa. A sua representação esquemática é como a figura seguinte documenta.



Para activar um estado existe a instrução STEP NEXT (SNXT(09)), à qual está associado um relé (relé de controlo). SNXT(09) é sempre antecedida por uma condição lógica, tal como se pode constatar pela representação gráfica seguinte.



Sempre que a condição lógica que antecede SNXT(09) vai a ON, a instrução faz activar o estado definido pelo relé especificado e em simultâneo desactiva o estado presente (se existir).

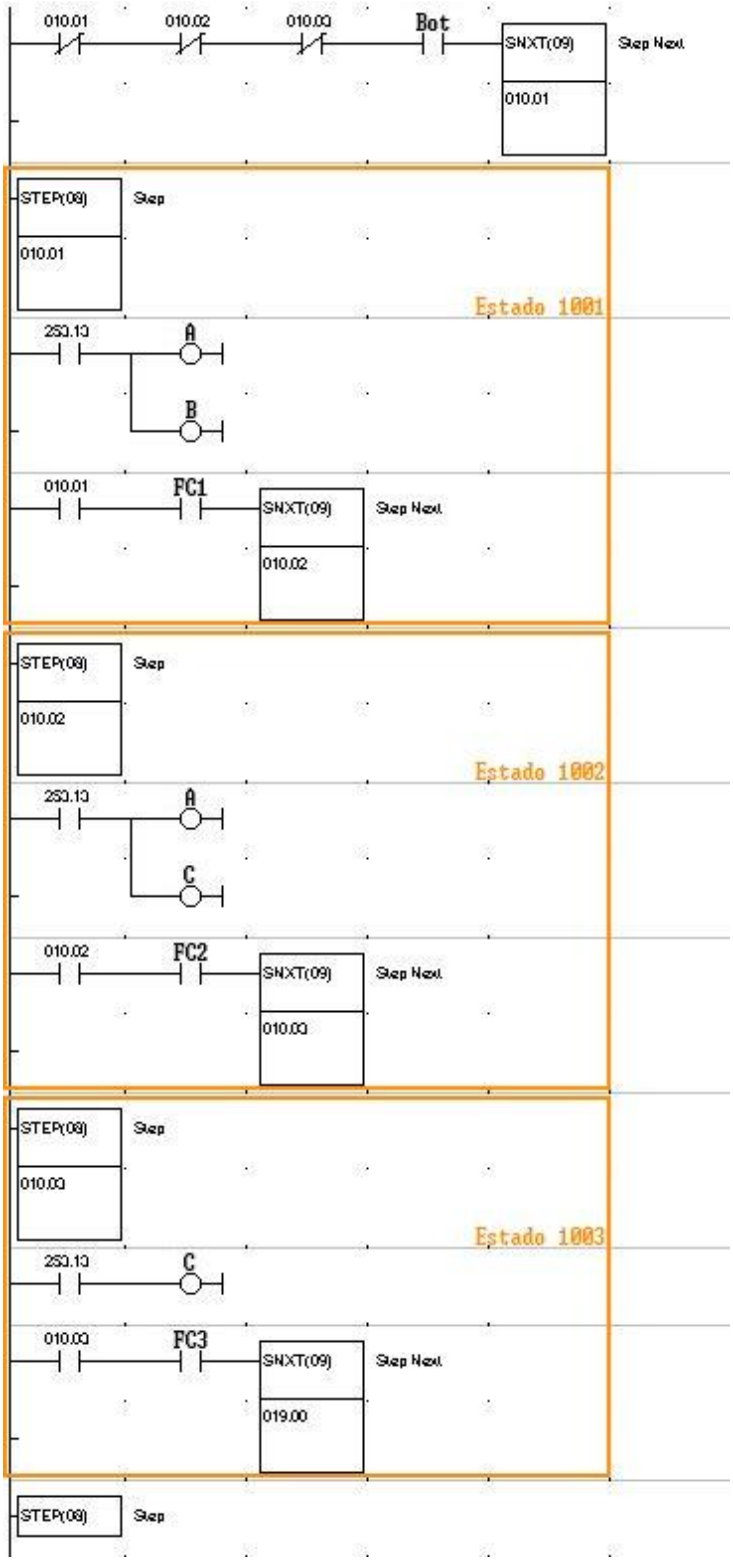
A secção de programa que se encontra entre a instrução STEP(08) e a instrução SNXT(09) só é executada, se o relé de controlo dessa secção estiver a ON. Estando a OFF, as instruções que se encontram entre STEP(08) e SNXT(09) são ignoradas pelo autómato (é como se não existissem).

Para concretizar a aplicação destas instruções, vamos implementar o processo sequencial que nos tem vindo a servir de exemplo.

A solução apresentada usando funções STEP(08) e SNXT(09) poderá parecer à primeira vista um pouco confusa, pois aparecem várias instruções OUT para uma mesma saída e também aparece uma instrução SNXT(09) tendo por operando um relé que não parece ter nehuma relação com o programa.

- O facto de aparecerem várias instruções OUT para a mesma saída (A e C) justifica-se pelo facto de o autómato só "ver" uma de cada vez. Como já foi dito, apenas as instruções que estão entre STEP(08) e SNXT(09) do estado activo, são tidas em conta pelo autómato; todas as outras contidas noutros estados não activos são pura e simplesmente omitidas, tudo se passando como se não existissem no programa. Para reforçar esta ideia, repare-se que na solução apresentada, as saídas estão condicionadas por um relé sempre a ON (253.13).

- A última instrução SNXT(09) tem por operando o relé 19.00. Este relé não tem qualquer função activa no programa (podia ser outro qualquer) e é usado porque por imposição da função SNXT(09) somo obrigados a especificar um; a sua função é a de receber o "testemunho" do último estado activo.



3.6. INSTRUÇÕES DE CÁLCULO ARITMÉTICO

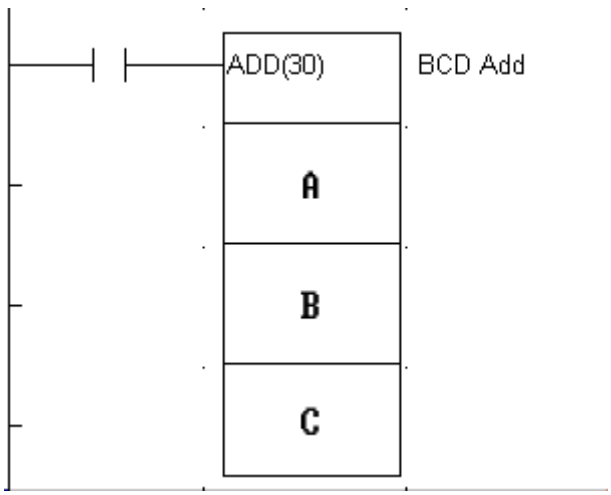
Encontramos com frequência nos autómatos mais recentes instruções que permitem a execução de operações aritméticas, embora esta não seja uma máquina vocacionada para o cálculo numérico. Dependendo do autómato, poderemos encontrar as operações básicas (soma, subtracção, multiplicação e divisão) ou outras mais sofisticadas, como sendo o cálculo em virgula flutuante, funções trigonométricas, raiz quadrada, etc.. A base numérica também pode ser diversa: binário, BCD, etc..

Vamos neste capítulo analisar o uso das operações aritméticas básicas em BCD e outras funções associadas.

3.6.1. SOMA EM BCD

O autómato que estamos a usar dispõe de duas funções soma em BCD.

A primeira que vamos analisar é a função ADD(30). Esta função permite adicionar dois valores numéricos A e B e coloca o resultado da adição no canal especificado em C. Os valores numéricos especificados em A e B podem ser constantes ou o conteúdo de um canal, contador ou temporizador.



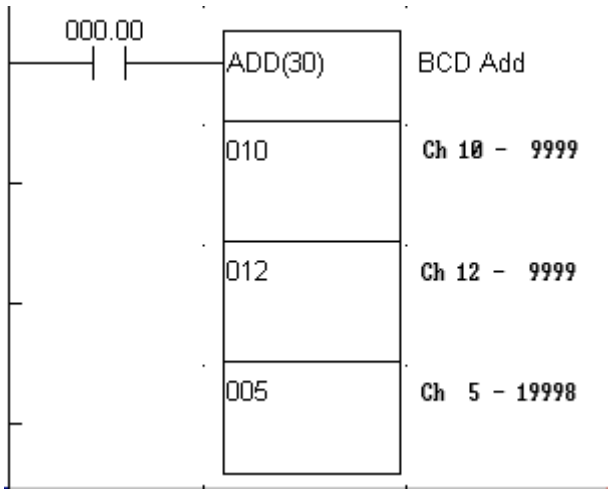
Nota: Esta instrução afecta o relé de CARRY.

3.6.2. RELÉ DE CARRY

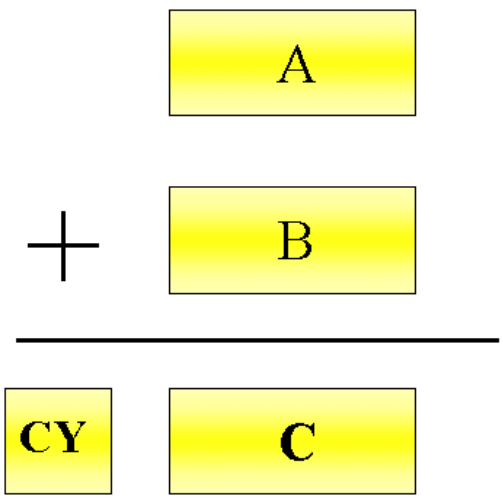
Já foi dito que num canal ou word pode codificar-se em BCD um valor numérico decimal até 9999.

Vamos analisar o que se passa quando for executada a função soma atrás descrita, tendo como operadores o canal 10 e o canal 12. O canal 10 contém o valor #9999 e o canal 12 contém o valor #9999.

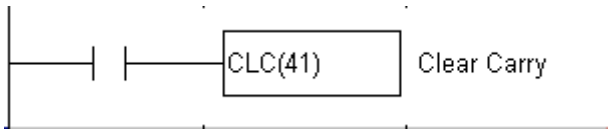
Quando a operação soma for executada, o canal 5 recebe o resultado da operação, tal como no exemplo abaixo.



Neste exemplo, o resultado da operação é 19998, e como tem 5 dígitos, não é possível colocar todo o valor no canal 5 (canal do resultado); O canal 5 recebe o valor 9998 e a presença do dígito mais significativo (1) é sinalizada por um relé especial que passa a ON e que se chama RELÉ DE CARRY.



O relé de carry (abreviadamente CY) tem no autómato CPM1 o endereço 255.04. Este relé uma vez a ON, só passa a OFF se for executada a função CLC(41).

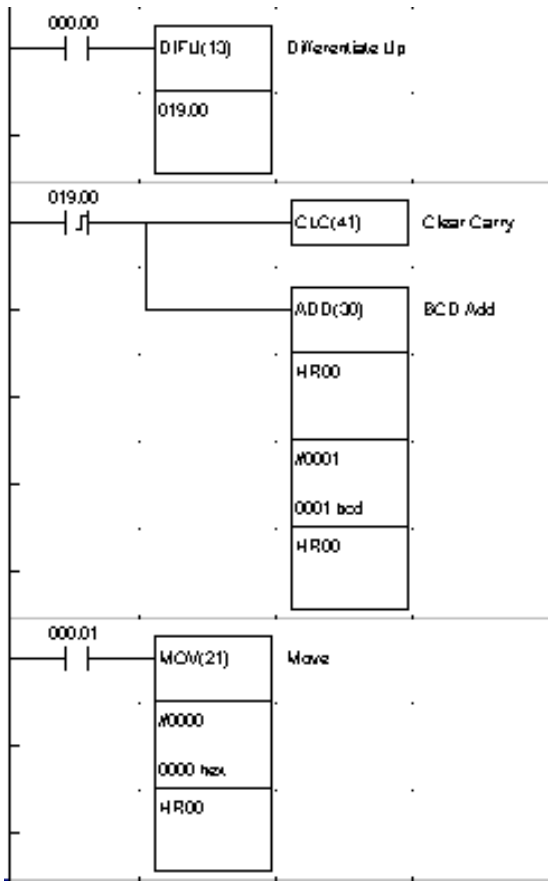


Esta função só é executada se a condição lógica que a antecede estiver a ON.

Para concretizar o que foi apresentado, vamos ver como poderia ser implementado um contador ascendente.

Neste exemplo, usa-se o relé 0.00 como entrada de impulsos e o relé 0.01 é usado para fazer o reset ao contador. Para que a função de adição seja executada uma só vez quando o relé 0.00 passa a ON, recorre-se à função DIFU(13) para detectar a transição e activar um relé auxiliar durante um só ciclo de scan. Esse relé é depois usado para controlar a execução da função ADD(30).

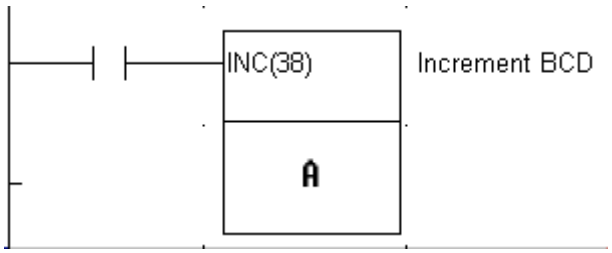
Quando se trabalha com números positivos, é boa norma fazer executar a função CLC(41) antes de executar uma função aritmética. Desta forma limpa-se o relé de CARRY (é colocado a OFF) que poderia ter sido colocado a ON por outra operação aritmética existente no programa e que caso estivesse activo falsearia o resultado da operação a executar.



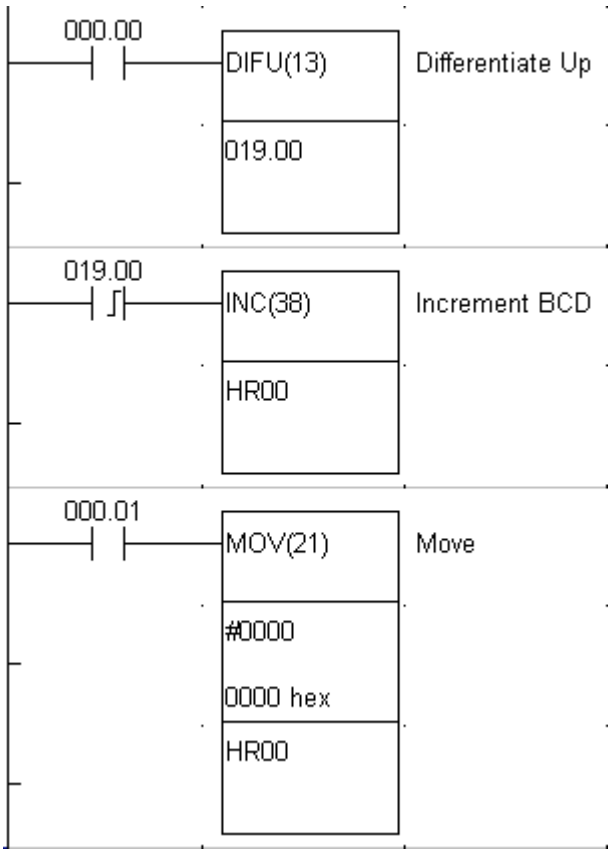
3.6.3. Instrução INC(38)

A instrução INC(38) deriva de um caso particular da adição em BCD. Sempre que a condição de execução está activa, esta instrução faz incrementar uma unidade ao conteúdo do canal especificado em A, em cada scan.

ATENÇÃO: O conteúdo de A deve ser BCD.

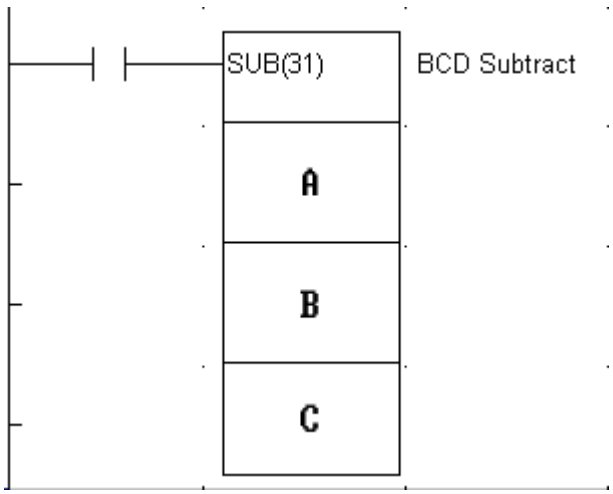


Usando esta instrução podemos simplificar o contador ascendente que foi dado como exemplo na exposição da instrução ADD(30), como abaixo se mostra.



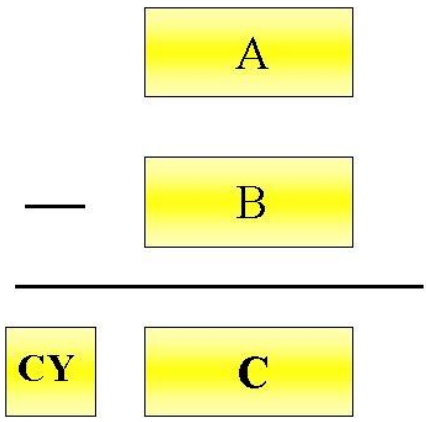
3.6.4. SUBTRACÇÃO EM BCD

A função SUB(31) permite subtrair ao valor contido em A o valor contido em B e coloca o resultado no canal especificado em C.

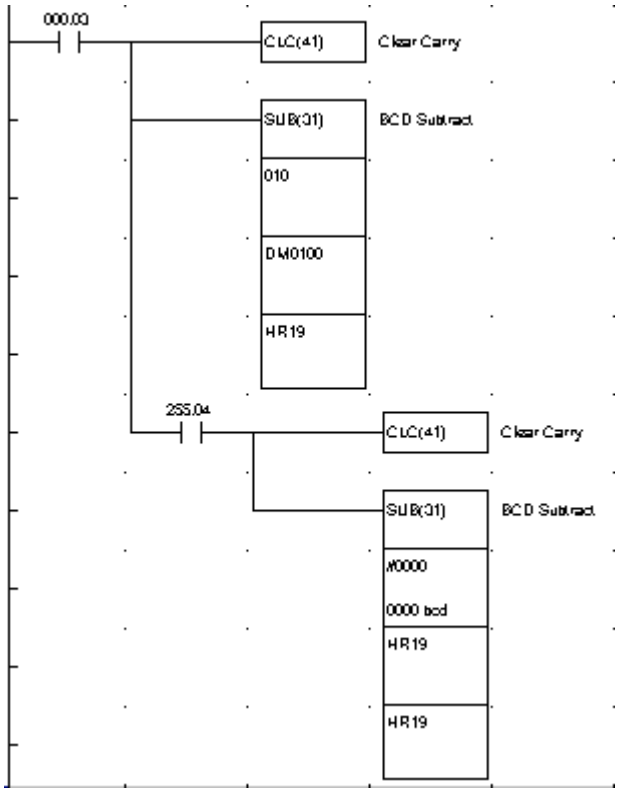


Tal como na função ADD(30) os valores numéricos especificados em A e B podem ser constantes ou o conteúdo de um canal, contador ou temporizador.

No caso da operação de subtracção, o resultado nunca excede quatro dígitos. Há no entanto a possibilidade de o resultado ser negativo (quando A for menor que B). Esta ocorrência é assinalada pelo mesmo relé de carry atrás mencionado. O CY vai a ON se o resultado da operação for menor que zero e em C é colocado o "complemento a 10" da subtracção.



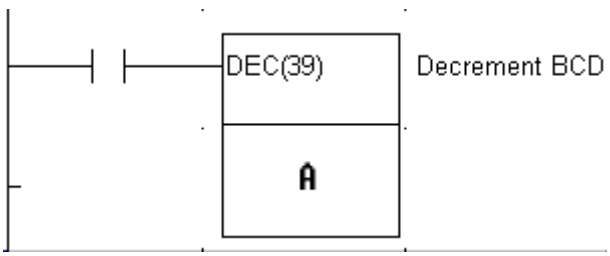
Para se obter o valor absoluto da subtracção quando o resultado é menor que zero, deve-se subtrair o resultado obtido a zero (como no exemplo seguinte).



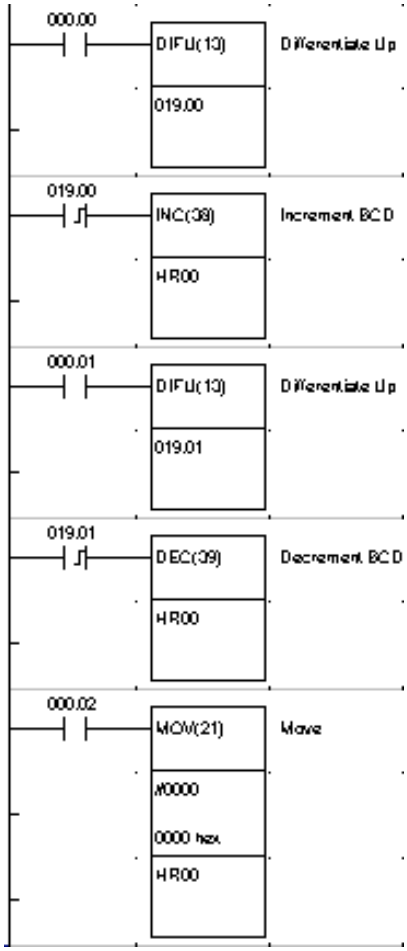
3.6.5. Instrução DEC(39)

Tal como acontecia com a instrução INC(38), a instrução DEC(39) deriva de um caso particular da subtracção em BCD. Sempre que a condição de execução está activa, esta instrução faz decrementar uma unidade ao conteúdo do canal especificado em A, em cada scan.

ATENÇÃO: O conteúdo de A deve ser BCD.

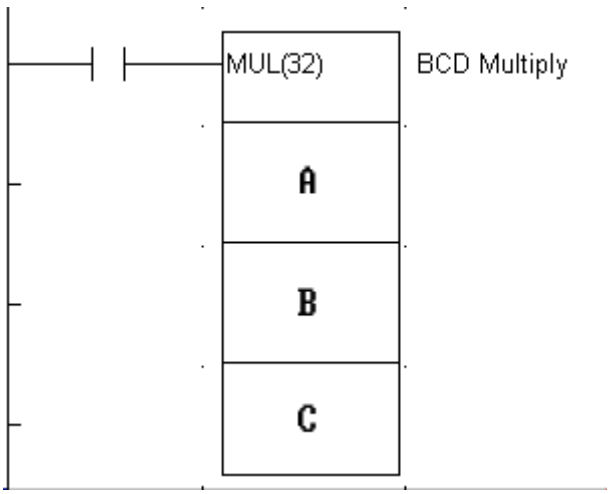


Usando as instruções já expostas, é possível implementar de outra forma, um contador bidireccional com funcionamento idêntico ao contador reversível (instrução CNTR(12)), como poderemos ver no diagrama de contactos que se segue.

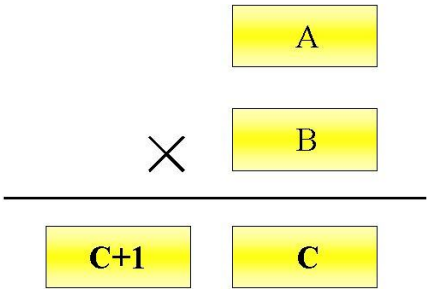


3.6.6. MULTIPLICAÇÃO EM BCD

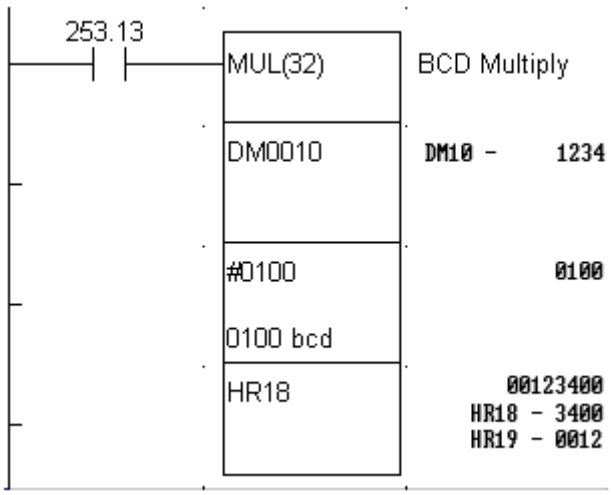
A função que permite efectuar o produto de dois valores numéricos BCD, é a função MUL(32).



Sempre que a condição lógica que antecede a função está a ON, o valor em A é multiplicado pelo valor contido em B e o resultado da operação é colocado no canal especificado em C e no imediatamente seguinte. Os quatro dígitos menos significativos do resultado são colocados no canal especificado em C e os quatro restantes dígitos mais significativos são colocados no canal C+1.

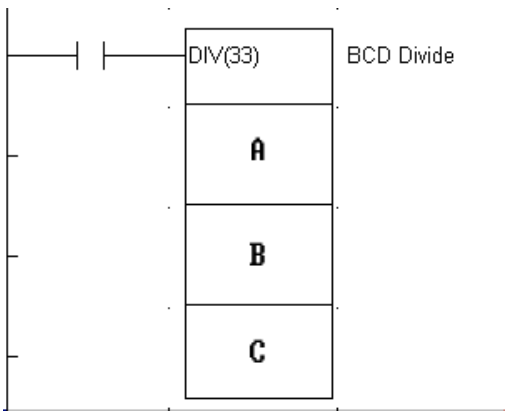


No exemplo que se segue, faz-se o produto do conteúdo do canal DM10 (neste exemplo é 1234) pela constante 100. O resultado é colocado no canal HR18.

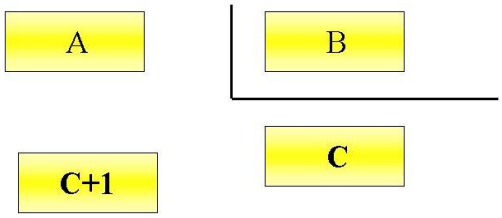


3.6.7. DIVISÃO EM BCD

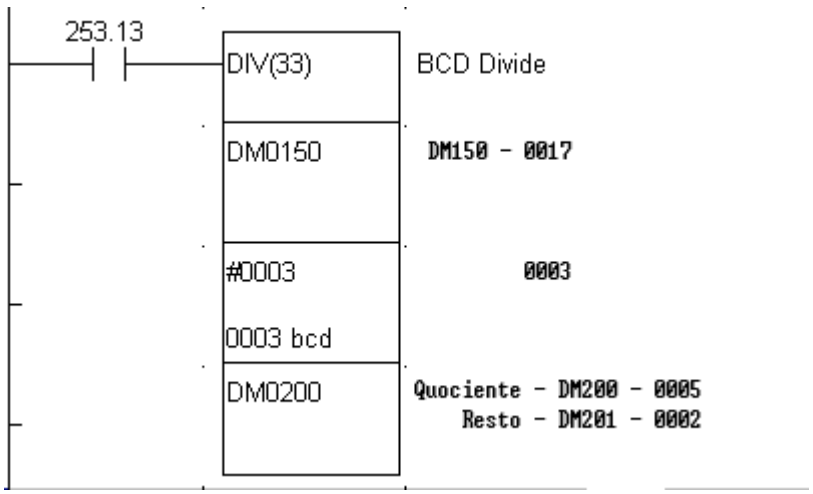
A função DIV(33) permite efectuar o quociente de dois valores numéricos BCD.



Sempre que a condição lógica que antecede a função está a ON, o valor em A é dividido pelo valor contido em B e o resultado da operação é colocado no canal especificado em C e no imediatamente seguinte. O quociente é colocado no canal especificado em C e o resto é colocado no canal C+1.



No exemplo que se segue, faz-se o quociente do conteúdo do canal DM150 (neste exemplo é 17) pela constante 3. O resultado é colocado no canal DM200.



Nas operações aritméticas apresentadas anteriormente, apenas é possível trabalhar com operandos de 4 dígitos (uma word) em BCD. Há no entanto situações que obrigam a trabalhar valores maiores que 9999 e para o fazer temos de recorrer à codificação em Binário.

Em Binário é possível codificar numa word um valor que pode ir até 65535, contra 9999 em BCD.

Existem também no autómato CPM1 as instruções aritméticas básicas para operar com valores binários, de forma análoga às instruções aritméticas BCD.

ADB(50) - soma binária

SBB(51) - subtracção binária

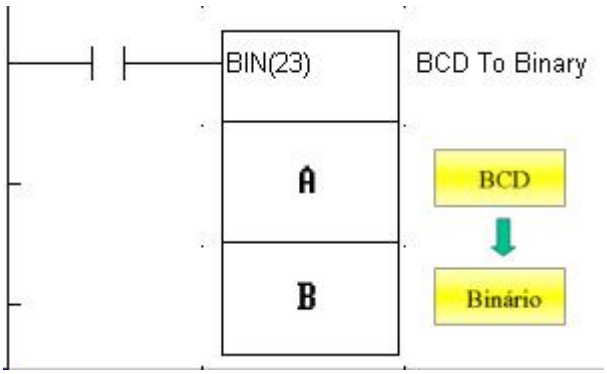
MLB(52) - multiplicação binária

DVB(53) - divisão binária

Há a acrescentar ainda duas instruções que permitem converter valores entre as duas bases numéricas já enunciadas.

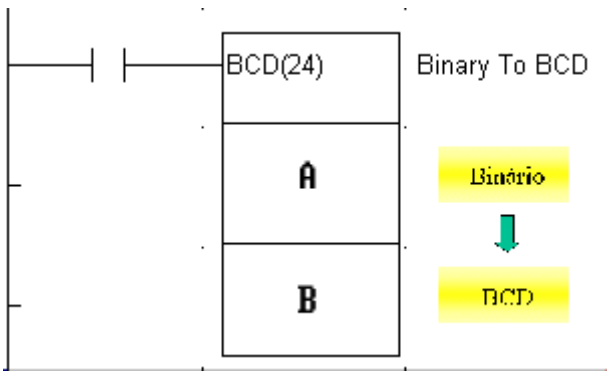
3.6.8. Instrução BIN(23)

Sempre que a condição de execução que a antecede esteja a ON, esta instrução permite converter o conteúdo BCD contido num canal em A para Binário, sendo o resultado da conversão colocado no canal especificado em B.



3.6.9. Instrução BCD(24)

Esta instrução permite converter o conteúdo na base Binária contido num canal em A para BCD, sendo o resultado da conversão colocado no canal especificado em B.



3.7. FUNÇÕES DE EXECUÇÃO DIFERENCIAL

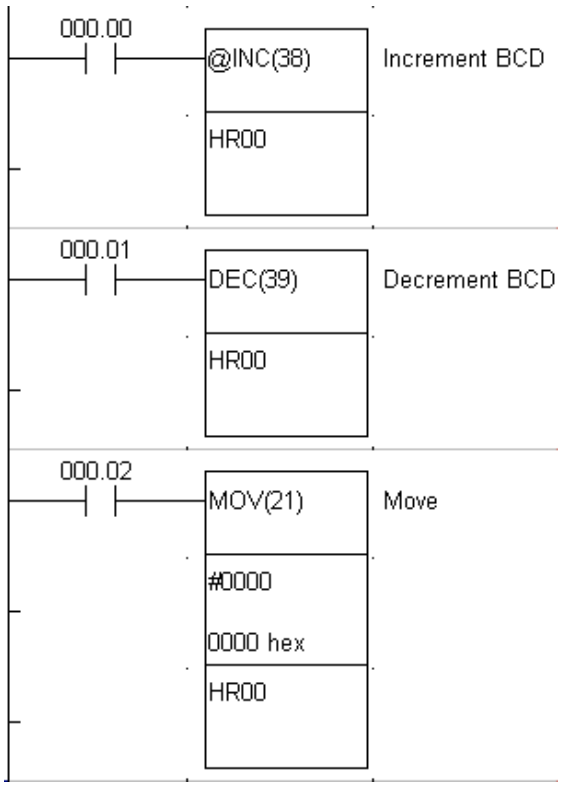
Vimos no capítulo 3.2 uma função que permite detectar uma transição de OFF para ON: DIFU(13). Até agora era a única forma de permitir a execução controlada de uma instrução (o

contador ascendente usando a função ADD(30) é disso um bom exemplo) e assegurar que a mesma era executada uma só vez.

Há no entanto uma alternativa que permite dotar uma instrução da capacidade de ser executada, só quando houver uma transição de OFF para ON na condição de execução. Essa alternativa consiste em adicionar o símbolo @ à instrução.

Para adicionar o símbolo @ à instrução que se pretende programar, deve premir-se a tecla NOT após digitar a tecla FUN e o código da função. Desta forma, a instrução em causa só será executada uma vez e sempre que existir uma transição de OFF para ON na condição lógica que a antecede.

Concretizando com um exemplo, vamos ver como se poderia também programar o contador bidireccional apresentado aquando da instrução DEC(39).

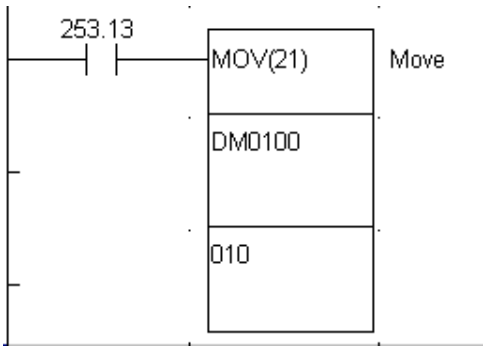


3.8. ENDEREÇAMENTO INDIRECTO

Esta facilidade pode ser encontrada em alguns autómatos. O uso do endereçamento indirecto permite dotar os programas de uma grande flexibilidade e reduzir a extensão dos mesmos.

Normalmente, quando é especificado o conteúdo de um canal da área de DMs numa instrução, a instrução é executada considerando o valor contido nesse DM. Por exemplo, quando se

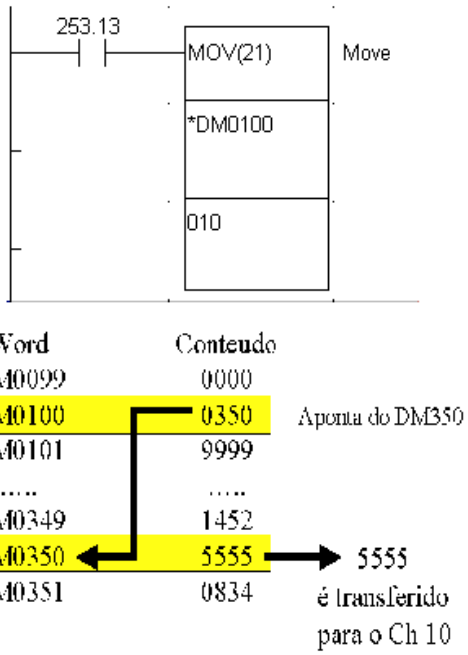
executa a instrução MOV(21) tendo o DM100 como primeiro operando e o CH2 como segundo operando, o conteúdo do DM100 é copiado para o canal 2.



Se o DM100 contém o valor 350, então o CH10 tomará esse valor.

É no entanto possível usar endereços indirectos de DMs como operandos de muitas instruções. Para especificar um endereço indirecto de um DM, deve-se anteceder o endereço (ex. DM100) por um asterisco (*DM100). Desta forma, o conteúdo do DM não contém o valor a processar pela instrução; Em vez disso o conteúdo do DM, contém o endereço de um outro DM, encontrando-se neste último o valor a ser considerado pela instrução.

Retomando o exemplo dado anteriormente, se especificássemos *DM100 (em vez de DM100) e o conteúdo do DM100 fosse o mesmo (350), o canal 10 tomaria o valor do DM apontado pelo conteúdo de DM100, neste caso o DM350.



O endereçamento indirecto pode ser usado em todas as instruções que aceitem DMs como operandos.

NOTA: O conteúdo do DM contendo o apontador, deve estar compreendido entre 0 e 1999, ou seja, à área de DM existentes.