PRÁTICAS DE AUTOMAÇÃO INDUSTRIAL:

ESPECIFICAÇÃO E PROGRAMAÇÃO DE SOLUÇÕES DE CONTROLO LÓGICO NO AMBIENTE DE TREINO “ITS PLC”

FICHA TÉCNICA

ISBN 978-989-96460-0-1

D.L. 313843/10

REAL GAMES LDA.
Rua Elísio de Melo, 39 - Piso 3
4000-196 Porto, Portugal
Email: info@realgames.pt
Internet: http://www.realgames.pt

Capa: Bruno Vigário e Nuno Silva, REAL GAMES LDA.

PRÁTICAS DE AUTOMAÇÃO INDUSTRIAL:

ESPECIFICAÇÃO E PROGRAMAÇÃO DE SOLUÇÕES DE CONTROLO LÓGICO NO AMBIENTE DE TREINO "ITS PLC"

António Pessoa de Magalhães

2ª Edição – 2012

TERMO DE DESRESPONSABILIZAÇÃO

Os problemas apresentados neste livro e as respetivas soluções têm unicamente propósitos didáticos. Em particular, o treino da programação de Controladores Lógicos Programáveis (PLCs) com recurso à plataforma "ITS PLC". Embora o autor e o editor creiam que as informações apresentadas estão corretas, em caso algum podem ser responsabilizados pela utilização dos programas fornecidos, ou de outros neles baseados, em aplicações das quais possam resultar danos ou prejuízos em pessoas ou bens.

Ao
João Rodrigo e ao José Diogo
que tanto gostam de "engenhocas"...

Índice

PREFÁCIO

Há muitos anos que suporto o ensino da programação de controladores lógicos programáveis (PLCs) em sistemas virtuais. Sem retirarem o devido espaço aos reais, os sistemas virtuais são uma solução de baixo custo, sem riscos para formadores e formandos, de instalação trivial e fáceis de multiplicar. E tudo isto mantendo os formandos em contacto com os equipamentos de controlo de interesse, como se de instalações reais se tratasse.

Mas, quando a estas virtudes se junta uma motivação acrescida por uma simulação extremamente fidedigna e interativa, que transporta para a sala de aula o ambiente e entusiasmo dos modernos jogos de computador, contagiando formadores e formandos, alcançam-se, então, condições muito favoráveis, quase únicas, a um ensino mais simplificado, célere, natural e eficiente.

Acredito assim que o software de treino ITS PLC é um meio privilegiado para alcançar um ambiente de grande entusiasmo, capaz de incutir nos alunos o desejo de realização, pensamento crítico e tenacidade que exibem à frente do ecrã de uma máquina de jogos. Por isso, aceitei de bom grado, com muito orgulho e empenho, o convite que a Real Games Lda. me dirigiu no sentido de produzir um "livro de exercícios" que acompanhasse o seu produto ITS PLC.

Foi uma tarefa complexa, mas, simultaneamente, muito interessante e enriquecedora que, não raras vezes, me fez sentir no ambiente de um jogo de computador. Seguindo precisamente a metodologia destes, elaborei um guião em que cada missão se inicia por pequenos desafios, fundamentais mas facilmente ultrapassáveis, evoluindo depois numa sequência de problemas em que cada um acrescenta sempre algo, mas não demasiado, ao anterior, proporcionando assim ao formando ganhos crescentes de conhecimentos e de autoconfiança. Mas, naturalmente que cada guião mais não é do que uma proposta de trabalho, que cada formador poderá, e deverá, adaptar aos seus interesses e formandos.

A apresentação das soluções em Texto Estruturado (*Structured Text*) será talvez uma surpresa para muitos, dado não ser a linguagem de programação mais comum. Há, contudo, duas fortes razões para tal escolha: por um lado, é uma linguagem de programação textual e de alto nível – logo, muito bem adaptada à comunicação com um público heterogéneo; por outro, contribui fortemente, mas com naturalidade, para a divulgação da norma IEC 61131-3. Quem programa habitualmente PLCs que não seguem esta norma nem empregam esta linguagem, terá certamente sentido pouco, e refletido ainda menos, sobre as virtudes de ambas. São exatamente esses os leitores que mais se pretende sensibilizar com esta iniciativa.

Reconhecendo que o Texto Estruturado não é a linguagem de eleição de muitos programadores, é muito possível que as soluções apresentadas venham a ser traduzidas nas mais diversas linguagens e dialetos de programação de PLCs. O interesse de tais traduções é compreensível, sendo até de prever a compilação e disponibilização das mesmas por iniciativa de algum grupo de interesse. Mas,

obviamente que as potencialidades do ITS PLC não se esgotam nessas traduções e, muito menos, nas propostas contidas neste texto. Haverá sempre espaço para o prazer e o desejo de descobrir novos problemas e novas soluções; ou seja, para novos jogos e formas de jogar o ITS PLC. Saibam, formadores e formandos, partilhar esses prazeres.

Porto, Setembro de 2011

António J. Pessoa de Magalhães

PARTE 1
APRESENTAÇÃO

Este livro propõe um conjunto de exercícios para o software educacional "ITS PLC", *Interative Training System for Programmable Logic Controllers*, produzido pela Real Games Lda. O seu principal propósito é rentabilizar o ambiente de treino oferecido por este produto, sugerindo planos de trabalho capazes de proporcionar uma aprendizagem aplicada, sólida, progressiva e coerente de técnicas de controlo lógico e correspondente programação de controladores lógicos programáveis – vulgarmente designados PLCs, conservando a sigla anglo-saxónica.

Embora as propostas aqui apresentadas estejam naturalmente centradas nos ambientes virtuais proporcionados pelo software ITS PLC, pretende-se que delas resultem ensinamentos cujo interesse e aplicação prática extravasem o mais possível esses mesmos ambientes. Assim, se no seu lado mais abrangente e vistoso, as instalações ITS PLC servem de mote a problemas lógicos razoavelmente latos e um tanto avançados na esfera da automação industrial, na sua vertente mais fundamental e prática são um ótimo pretexto para introduzir e discutir questões pontuais e elementares que vão ao encontro dos erros, dúvidas e dificuldades mais comuns de quem se inicia, ou pretende evoluir, no controlo de sistemas de eventos discretos e na programação de PLCs. Para a definição, apresentação, tratamento e encadeamento lógico destes problemas, em muito contribuiu a experiência relativamente longa do autor no ensino da programação de PLCs, uma atividade sempre apoiada tanto em sistemas reais como virtuais.

Conquanto o principal desafio do conjunto de exercícios propostos seja especificar e desenvolver um programa para controlar correta e elegantemente cada instalação ITS PLC, excelente é que tal feito seja, simultaneamente, um ponto de partida para o desenvolvimento de aplicações mais abrangentes e elaboradas no domínio da automação industrial. Por exemplo, aplicações que integrem consolas de interface homem-máquina (*Human-Machine Interface* – HMIs), sistemas de supervisão, controlo distribuído ou gestão de informação. Extremamente útil é também procurar soluções mais robustas e capazes de lidar com situações de falha ou insegurança. Propostas de trabalho orientadas em ambos os sentidos são incluídas neste texto. Têm em mente níveis de educação e treino mais elevados, devendo por isso os formadores adaptar tais linhas orientadoras aos objetivos dos seus cursos, equipamentos disponíveis e níveis de conhecimentos dos seus formandos.

Considerando unicamente os exercícios que têm como pano de fundo o controlo por PLC das cinco instalações virtuais ITS PLC, este livro propõe um total de sessenta problemas: doze por cada instalação. Para cada problema proposto é fornecida a respetiva solução na forma de um programa de PLC, devidamente comentado. Nos casos mais simples, os programas são antecedidos de uma explicação relativamente curta e informal; nos mais complexos, a solução é fundamentada numa especificação em linguagem *GRAFCET*[1] e de acordo com a norma IEC 60848, segunda edição.

[1]Seguindo a tendência da literatura especializada, o texto adota o termo *GRAFCET* para designar a linguagem de especificação *GRAphe Fonctionnel de Commande Étape Transition* e o vocábulo *grafcet* para designar um esquema gráfico que utiliza a linguagem *GRAFCET*.

Longe de serem as únicas possíveis, as especificações e as soluções apresentadas têm o propósito de serem modulares, genéricas e, desejavelmente, potenciadoras de inúmeros motivos de reflexão por parte de formadores e formandos.

É cada vez mais comum começar o projeto de um sistema sequencial pela sua especificação em *GRAFCET*. O facto de o *GRAFCET*, por vezes erradamente designado *Sequential Function Chart*, SFC, ser uma metodologia gráfica normalizada muito mais sucinta, objetiva e abrangente do que os diagramas de estado e os diagramas temporais, tem-no tornado numa ferramenta amplamente usada e bastante bem conhecida da generalidade dos programadores de PLCs. A tudo isso acresce ainda a relativa simplicidade do processo de tradução de um *grafcet* num programa genérico de PLC, uma questão que merece também o devido destaque neste livro. Consequentemente, adotar neste texto o *GRAFCET* como linguagem descritiva do comportamento de um sistema de controlo não mereceu qualquer hesitação.

Já a escolha da linguagem de programação em que as resoluções são apresentadas mereceu grande ponderação. Acabou por recair sobre o Texto Estruturado (*Structured Text* – ST), tal como semântica e funcionalmente previsto na norma IEC 61131-3, segunda edição. Para além de bem adaptada ao fim em vista, esta opção promove uma linguagem e uma norma de crescente importância no domínio dos PLCs, alargando, deste modo, o domínio didático do texto.

Porque se reconhece que o Texto Estruturado e a norma IEC 61131-3 nem sempre são bem conhecidos de quem se inicia na programação de PLCs, houve a preocupação de incluir neste livro um conjunto de informações orientadoras dos programadores menos familiarizados com esta linguagem e com a norma em causa. Essa informação precede a apresentação das soluções. E, falando em normas, interessante será também encontrar nos exercícios propostos espaço para promover a norma IEC 61499: "Blocos funcionais em sistemas distribuídos de controlo e medida de processos industriais". Embora tal meta saia um pouco dos objetivos primários deste texto, os formadores mais familiarizados com o tema saberão certamente encontrar o caminho certo para a alcançar.

Acredita-se, pois, que há inúmeros motivos de interesse neste livro, o qual está dividido em quatro partes organizadas do seguinte modo:

A Parte 1 – Apresentação – expõe as questões necessárias a uma correta e completa introdução do leitor ao ambiente de aprendizagem que o espera. Concretamente, começa por apresentar o enquadramento, a sequência, os objetivos e o público-alvo dos desafios lançados neste texto. Seguidamente, são feitas algumas considerações sobre o modo como o leitor deve encarar os desafios que lhe serão lançados e a sua previsível aprendizagem face aos seus conhecimentos prévios. Por fim, são listados os recursos necessários à realização e acompanhamento dos exercícios propostos.

A Parte 2 – Os Problemas – apresenta, como se de um jogo de computador se tratasse, os cinco grandes desafios, ou "missões", que o leitor deverá progressivamente resolver. Cada desafio corresponde ao controlo de uma instalação virtual ITS PLC e está organizado em doze exercícios de programação. Estes estão sequenciados em pequenas tarefas, de forma a potenciarem uma aprendizagem simples, natural e eficiente. No início de cada missão são dadas explicações sobre

o ambiente em que a mesma se desenrola, de modo que o leitor compreenda minimamente os problemas típicos da instalação industrial emulada e o interesse prático dos exercícios propostos. No fim de cada missão são lançadas propostas de trabalho destinadas a um público com conhecimentos mais avançados. Estas propostas estão organizadas segundo duas perspetivas: por um lado, a integração das aplicações com sistemas de supervisão e consolas de interface homem-máquina; por outro, melhorar a robustez dos programas desenvolvidos por inclusão de mecanismos lógicos de deteção e suporte de situações anómalas.

A Parte 3 – As Soluções – inicia-se com algumas considerações sobre a programação em Texto Estruturado e a norma IEC 61131-3. Esta introdução tem em mente os leitores menos familiarizados com estes temas, sugerindo-lhes também literatura, sítios na Internet e recursos de software julgados úteis. Feita essa breve apresentação, surgem então as soluções dos desafios lançados na Parte 2. Cada solução compreende a justificação dos procedimentos a programar, por vezes apoiada num *grafcet*, seguida do programa correspondente, devidamente comentado.

A Parte 4 – Epílogo – encerra o texto com um conjunto sumário de conclusões.

O Jogo

Bem-vindo ao jogo ITS PLC! Se não encontrou este livro absolutamente por acaso, sabe que o ITS PLC é um pacote de software desenvolvido pela Real Games Lda. que emula instalações industriais a controlar por PLC. Informações sobre este produto estão disponíveis no sítio www.realgames.pt.

É pois importante que, antes de continuar esta leitura, se familiarize minimamente com o software em causa, compreendendo perfeitamente os seus objetivos e potencialidades, assim como as funcionalidades e os meios de comando e de sensorização das instalações emuladas. Uma versão gratuita, não licenciada, serve para este primeiro contacto. Pode obtê-la no sítio da Real Games Lda., assim como o respetivo manual de utilização.

O principal objetivo deste texto é ajudá-lo a "dar vida" a cada um dos cinco cenários que compõem o ITS PLC. Para isso, o leitor, que é como quem diz, "o jogador", terá de interligar um PLC ao computador onde tem instalado o ITS PLC e programá-lo corretamente. Conhecendo a aplicação, facilmente o leitor depreenderá que este jogo lhe propõe cinco missões. Em lato senso, elas são:

- Automatizar uma instalação de *Sorting*;
- Automatizar uma instalação de *Batching*;
- Automatizar um Paletizador;
- Automatizar uma instalação *Pick and Place*;
- Automatizar um Armazém Automático.

O interesse prático e os objetivos exatos de cada missão serão revelados oportunamente. Para já, importa dizer que cada missão é composta por um conjunto de doze tarefas, a realizar pela ordem

proposta, de modo que o jogador se familiarize progressivamente com o sistema a automatizar e adquira as competências necessárias para o completo e efetivo cumprimento da missão atribuída. Cumprir uma tarefa significa subir um nível. Vencer o último nível significa completar a missão. Completar as cinco missões significa terminar o jogo e ser um "perito em programação de PLCs"!

A justificação e demonstração do interesse prático de cada missão são aspetos a que foi dado um particular cuidado. Assim, cada missão inicia-se com uma explicação dos aspetos físicos e funcionais da instalação emulada, permitindo compreender devidamente o interesse, os objetivos e as dificuldades da missão em causa no contexto de instalações e aplicações reais congéneres.

Cada tarefa tem um enunciado muito simples, e preciso, que inclui o cenário em que se desenrola, o objetivo a atingir e os sinais de I/O a considerar. Ao jogador são também fornecidas duas "pistas" para realizar cada tarefa. Utilizá-las faz naturalmente sentido. Mas começar por procurar as suas próprias pistas faz bastante mais!

Algumas tarefas visam o comando automático de apenas parte do equipamento disponível, tendo o jogador de comandar manualmente outras partes. Nesses casos, os enunciados explicam como e porquê. O mesmo acontece quando a verificação da correção dos programas desenvolvidos requer a simulação de avarias ou a imposição de um estado funcional a um ou mais atuadores.

Para cada tarefa proposta há uma resolução disponível. Concluída uma tarefa, o jogador deverá comparar a sua solução com a fornecida. Se acabar por concluir que, definitivamente, não é capaz de realizar uma tarefa, deve então consultar e estudar cuidadosamente a solução proposta, procurando entendê-la completamente antes de passar à tarefa seguinte. Mas é importante que não desista cedo demais! E, tanto em caso de sucesso como de insucesso, deve meditar nos comentários e justificações que acompanham a solução proposta.

Completada uma missão, isto é, terminada a automatização de um sistema, há ainda espaço para alargar os horizontes da mesma. Nesse sentido, o jogador é desafiado para metas mais ambiciosas enunciadas em dois "pacotes" de exercícios suplementares:

- ITS SUPER – *SUPERvisory Environments and Systems*;
- ITS DEEP – *DEpendable Environments Programming*.

O primeiro visa o desenvolvimento de soluções flexíveis e distribuídas por integração de recursos tecnológicos – tais como PLCs, consolas HMI, sistemas de supervisão e outros. O segundo encerra um conjunto de desafios destinados a melhorar a confiabilidade das soluções encontradas por inclusão de técnicas de deteção e suporte de situações anómalas. Mais do que exercícios muito específicos, com enunciados concretos e rígidos, as propostas contidas nestes dois "pacotes" são essencialmente linhas orientadoras de trabalhos mais latos, de nível mais elevado e que exigem mais tempo. Cabe por isso aos formadores adaptá-las aos equipamentos ao seu alcance e aos interesses e conhecimentos dos seus formandos. Uma possibilidade interessante é considerá-los como tema daqueles pequenos projetos de desenvolvimento, individuais ou em grupo, que habitualmente são propostos aos formandos de níveis mais avançados.

Os Jogadores

O software de treino ITS PLC é uma plataforma didática de programação de PLCs que, quando limitada a problemas simples, pode e deve ser utilizada por formandos que dão os primeiros passos nesta matéria. Porém, programar um PLC para comandar e dotar os cinco sistemas incluídos no pacote ITS PLC de funcionalidades um tanto elaboradas, é uma tarefa que exige um leque relativamente amplo de conhecimentos e competências. Concretamente, nas seguintes matérias:

- Sistemas lógicos: variáveis binárias e códigos binários; sistema de numeração binário, octal e hexadecimal; operadores lógicos e álgebra de Boole; elementos de memória; operações de *set* e *reset*; operações sobre registos; operações aritméticas, de rotação e de deslocamento;

- Especificação: representação funcional em diagramas temporais, diagramas de estado e *GRAFCET*;

- PLCs: modelo de funcionamento de um PLC; afetação e cablagem de entradas e saídas; registos e organização interna da memória; temporizadores e contadores; experiência mínima em programação de PLCs e na utilização de ferramentas de desenvolvimento e teste associadas; facilidade em ler e compreender o manual de um PLC.

Se tudo isto lhe é relativamente familiar, então considere-se apto a começar o jogo. E, se os seus conhecimentos destas matérias forem bastante razoáveis, acabará certamente por levá-lo até ao fim sem dificuldades de maior. Mas, se pelo contrário, não conhece ou não domina minimamente grande parte destes temas, então deve procurar melhorar um pouco os seus conhecimentos antes de aceitar os desafios que aqui se lançam. Há um infindável rol de literatura sobre estes assuntos. O seu formador saberá, seguramente, indicar-lhe a bibliografia mais adequada aos seus conhecimentos.

Para quem se prepara para começar a jogar, o conselho mais óbvio é o de que, tal como nos jogos de computador, "evite a batota". Ter dificuldades em encontrar uma solução e tê-la ao virar da página, pode não ser o melhor estímulo à perseverança. Mas, sempre que tiver a tentação de espreitar a resolução, lembre-se que perder a oportunidade de descobrir a sua própria solução – que, quem sabe, seria até mais interessante do que a proposta – não é o melhor contributo para uma boa aprendizagem.

Procure, pois, encontrar sempre as suas próprias soluções. Verificará que, desse modo, os seus conhecimentos evoluem e solidificam de uma forma muito natural e irreversível. Se optar por espreitar sistematicamente a solução sentirá que, na maioria dos casos, pouco tempo depois, já não se "lembrará do truque da solução", concluindo que, afinal, não aprendeu tanto quanto desejaria. Talvez nem tenha compreendido que a programação de PLCs vive da "lógica" e não de "truques"!

O Equipamento

Certamente tem a noção exata do equipamento que necessita para começar o jogo: um computador com o ITS PLC instalado e devidamente licenciado, e um PLC com um número mínimo de entradas e saídas. Verifique que tipo de entradas e saídas tem o seu PLC e certifique-se de que as ligou corretamente à placa de I/O que acompanha o produto. Consulte o manual de utilização do ITS PLC sobre esta questão. Tenha, aliás, este manual sempre por perto, pois vai precisar de consultar frequentemente o mapa de entradas e saídas das instalações a controlar.

Por perto deverá ter também o manual do seu PLC e do software de desenvolvimento associado. A ajuda em linha nem sempre é suficiente. Ter também à mão um ou outro livro sobre sistemas lógicos e especificação em *GRAFCET* (de preferência, a segunda edição da norma IEC 60848) é também uma boa ideia.

Mesmo um PLC relativamente modesto tem capacidade para controlar as instalações apresentadas. Mas, se tem um PLC relativamente sofisticado, não deixe de explorar as suas potencialidades, inventando novos exercícios ou outras formas de os resolver que as empreguem.

Se tem possibilidade de ligar dois monitores ao seu computador, deve fazê-lo. Use um para visualizar o ambiente ITS PLC e outro para analisar *online* toda a informação relativa à execução do programa no PLC recorrendo a uma ferramenta de *debugging*. Rapidamente compreenderá os benefícios desta estratégia.

Por último, se é uma pessoa que gosta de jogos de computador e se entusiasma facilmente com problemas lógicos, é aconselhável fazer-se acompanhar de uma garrafa térmica com o seu chazinho preferido. Talvez o dia, ou a noite, seja longo(a)...

Exposto o essencial sobre o ambiente do jogo ITS PLC, é tempo de avançar para a máquina de jogos!

PARTE 2

OS PROBLEMAS

Provavelmente está já à frente do seu computador, devidamente equipado e pronto a conhecer o primeiro desafio. Algumas breves notas antes de começar:

Conforme referido, os problemas são propostos em cinco módulos, autocontidos, seguindo a ordem por que as instalações são apresentadas no software ITS PLC: *Sorting*, *Batching*, Paletizador, *Pick and place* e Armazém automático. As missões não têm necessariamente de ser realizadas por esta ordem, mas aconselha-se a que o sejam. Isso porque, no sentido de evitar a duplicação de explicações detalhadas, a resolução de algumas tarefas remete para explicações prestadas em soluções anteriores. Mas, importante mesmo, é que, dentro de cada missão, tente resolver as tarefas pela ordem por que são apresentadas, uma vez que, no contexto da missão, cada tarefa complementa a anterior e será complementada pela seguinte. Quando achar que está a ser difícil evoluir numa missão, sugere-se, como primeiro passo, que a abandone temporariamente, sem consultar a solução, e passe à missão seguinte. Talvez esta lhe proporcione os ensinamentos e a inspiração que lhe permitirão retomar, mais tarde, a missão que deixou em suspenso.

Importante também é que se certifique sempre que compreendeu bem a questão que lhe é posta antes de pensar na respetiva resolução. Verifique também que o I/O indicado no enunciado é coerente com os objetivos da tarefa. Comece sempre por definir minimamente, mas de forma objetiva, os procedimentos a realizar. O recurso a um diagrama temporal, diagrama de estados ou *grafcet* é uma boa abordagem, especialmente nos casos mais complexos. Só depois se preocupe com o programa a desenvolver. Se o enunciado de algum problema não lhe parecer absolutamente claro, não use tal argumento para consultar as soluções. Procure resolver a questão tomando a interpretação que lhe parecer mais plausível e não a mais simples!

Por fim, dois conselhos práticos:

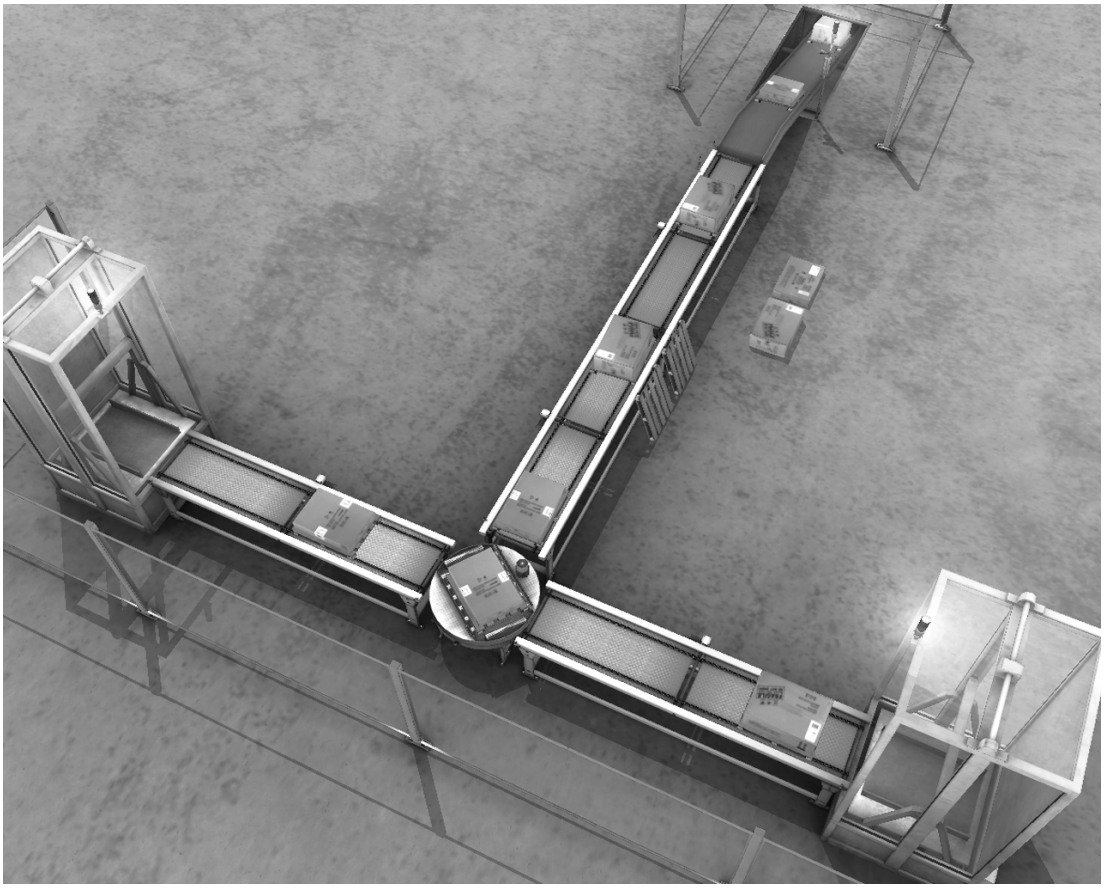
Ao longo das suas experiências vai certamente ter interesse em, por vezes, reiniciar o seu sistema; isto é, tanto a instalação virtual como o seu PLC. A instalação é limpa premindo o botão “Limpar” no painel do ITS PLC. Para reiniciar o PLC, sugere-se a seguinte dica: inclua no seu programa um procedimento para reinicializar as variáveis internas do PLC quando uma entrada não utilizada, por exemplo, um sensor ou uma botoneira, é forçada ou acionada. Pode, desse modo, e sempre que quiser, concretamente depois de premir “Limpar”, reiniciar o seu PLC sem o retirar do modo *run*. Pode, também, optar por utilizar o seletor “Manual/Auto” (Entrada 11) para o mesmo fim. Mas, nesse caso, o PLC reiniciará sempre que a instalação é lançada no modo automático. Tal situação nem sempre é a mais interessante.

Em qualquer caso, é conveniente que não coloque o seu PLC em modo *run* antes de colocar a instalação em modo automático, pois, nessa situação, as variáveis internas do programa do PLC podem evoluir para valores diferentes dos pretendidos para a situação inicial da instalação.

Feitas estas considerações, chegou finalmente o momento de conhecer os desafios que o esperam. Boa Sorte!

MISSÃO 1: AUTOMATIZAÇÃO DE UMA ESTAÇÃO DE TRANSPORTE E TRIAGEM DE MERCADORIAS EM PALETES

OBJETIVO: Encaminhar paletes do cais de entrada aos elevadores de saída, selecionando-as por alturas



Acerca desta Missão

O movimento de bens em transportadores automáticos, tais como tapetes rolantes e mesas de transferência, é uma tarefa muito comum em instalações industriais. Do ponto de vista funcional, um tapete rolante com um só sentido de movimentação é o elemento de transporte automático mais simples, podendo o seu estado, movimenta ou não mercadoria, ser representado por uma variável binária. Há, contudo, transportadores muito flexíveis, relativamente complexos, cujo leque de estados possíveis é obviamente muito mais vasto. É o caso dos transportadores sequenciais, como a mesa rotativa desta aplicação, que, servindo simultaneamente de elementos de transferência e triagem, têm até, por vezes, diversos pontos de entrada ou saída.

Um transportador tem à sua entrada algo que lhe fornece mercadoria; por exemplo, outro tapete, um operador ou um alimentador automático. À sua saída haverá algo que recebe mercadoria: um outro tapete, uma mesa de transferência, um cais de saída ou um operador. A missão de um transportador é transferir mercadorias de um ponto de partida a um ponto de chegada de forma eficiente. Eficiência pressupõe, tipicamente, um transporte de acordo com uma origem e um destino pré-definidos e realizado no menor tempo e com o menor consumo possível de energia. Ou seja, requer que:

- Um tapete não esteja em movimento caso nele não exista qualquer mercadoria;
- Um tapete que tenha mercadoria não esteja parado, a menos que tal seja absolutamente necessário;
- Os elementos de triagem encaminhem as mercadorias para os destinos corretos.

O principal objetivo desta missão é mostrar que, mesmo numa instalação complexa, o que de facto existe à entrada e à saída de um transportador pouco importa para o seu comando: importante é a sincronização do funcionamento de cada tapete com o dos alimentadores e transportadores que tem à sua entrada e à sua saída. É dessa compreensão que resulta a capacidade para desenvolver uma solução modular, apoiada num controlador centralizado ou distribuído, perfeitamente utilizável em sistemas de transporte automático reais de grandes dimensões.

Particularmente importante em qualquer sistema de transporte flexível é conseguir gerir toda a informação necessária ao correto encaminhamento das mercadorias. Para tal, a informação das mercadorias em trânsito, independentemente de ter origem em sistemas de identificação mais ou menos sofisticados, como leitores de códigos de barras ou de RFIDs, ou em vulgares sensores de proximidade, como no caso da presente aplicação, tem normalmente de seguir percursos e passar por processos de seriação e triagem muito semelhantes aos das próprias mercadorias.

Também importante num sistema de transporte é a abertura dos controladores locais à troca de informação com elementos de diálogo homem-máquina e sistemas de supervisão. Por último, mas também de importância vital, há a questão da deteção e suporte eficaz de situações erróneas que possam conduzir à degradação física da instalação ou dos bens transportados.

Esta missão toca em tudo isto, começando pelas coisas mais simples.

TAREFA 1: Transporte automático de paletes isoladas no tapete de entrada

Cenário:	O tapete de entrada entra em movimento quando lhe chega uma palete, transporta-a até à mesa rotativa e para.
Objetivo:	Comandar automaticamente o tapete de entrada para transportar paletes isoladas.
Estado Inicial:	Tapete de entrada sem paletes.
Sinais de I/O:	Entradas: sensores 0 e 3.
	Saídas: atuador 1.
Procedimentos Manuais:	Comandar manualmente o alimentador, forçando e libertando o atuador 0, de modo a fazer chegar uma palete ao tapete de entrada somente quando este está vazio.
	Manter a mesa rotativa em carga, forçando o atuador 2, para dar saída às paletes provenientes do tapete de entrada, lançando-as para o chão.
	Remover caixas e paletes que se acumulem atrás da mesa rotativa, mantendo-a desobstruída.
Dicas:	Use uma variável binária, i.e., um *bit* de memória, para definir constantemente se o tapete deve estar ou não em movimento…
	A entrada e saída de paletes no tapete correspondem a transições lógicas em sensores…

TAREFA 2: Alimentação e transporte automático de paletes isoladas no tapete de entrada

Cenário:	O tapete de entrada funciona como na Tarefa 1. O alimentador é agora comandado automaticamente para só lançar uma palete no tapete de entrada quando este está vazio.
Objetivo:	Automatizar o alimentador, eliminando o comando manual da Tarefa 1.
Estado Inicial:	Tapete de entrada sem paletes.
Sinais de I/O:	Entradas: sensores 0 e 3 – ou outros, se achar mais conveniente.
	Saídas: atuadores 0 e 1.
Procedimentos Manuais:	Manter a mesa rotativa em carga, forçando o atuador 2, para dar saída às paletes provenientes do tapete de entrada, lançando-as para o chão.
	Remover caixas e paletes que se acumulem atrás da mesa rotativa, mantendo-a desobstruída.
	Forçar avarias ocasionais no alimentador, impedindo-o de fazer chegar paletes ao tapete de entrada, para verificar que este último efetivamente para quando não transporta qualquer palete.
Dicas:	Note que a transferência de uma palete do alimentador para o tapete de entrada requer o movimento simultâneo de ambos…
	Note que quando o tapete de entrada transporta uma palete, tal não implica necessariamente que o alimentador deva estar parado. Defina uma variável binária, “Busy_1”, que seja verdadeira quando o tapete de entrada não está em condições de receber uma palete…

TAREFA 12: Armazenamento e levantamento de mercadorias por valores de referência aleatórios

Cenário:	Quando a aplicação é lançada acende-se a luz da botoneira Reiniciar. Premindo esta botoneira inicia-se automaticamente o seguinte funcionamento cíclico: São sucessivamente armazenadas caixas em posições livres selecionadas aleatoriamente até que existam 35 caixas no armazém. Quando este número é atingido, são sucessivamente levantadas caixas de posições ocupadas, selecionadas aleatoriamente, até que existam apenas 15 caixas em armazém. Quando este número é atingido, é retomado automaticamente o processo de armazenamento definido anteriormente; depois, o de levantamento, e assim sucessivamente. A luz da botoneira Iniciar pisca durante os movimentos do transelevador como nas tarefas anteriores.
Objetivo:	Modo de demonstração baseado na movimentação de mercadorias segundo padrões aleatórios.
Estado Inicial:	Transelevador em qualquer posição mas sem carga. Armazém vazio.
Sinais de I/O:	Entradas: sensores 1, 3, 5, 6 e 7 e botoneiras Reiniciar e Emergência.
	Saídas: atuadores 0 a 7 e luz da botoneira Reiniciar.
Procedimentos Manuais:	Premir a botoneira Reiniciar para inicializar a instalação e a botoneira de Emergência para simular situações anómalas.
Dicas:	Pode voltar a registar as posições ocupadas numa tabela de *booleanos*…
	Pode gerar números aleatórios a partir do valor atual de um temporizador que é reiniciado periodicamente. Sempre que o número aleatório assim gerado não puder ser utilizado, considere o número imediatamente acima…

E agora que a quinta missão foi cumprida…

E eis-nos chegados ao fim da quinta e última missão! Parabéns por tão extraordinário feito. Foi sem dúvida uma missão muito interessante! Alguma vez tinha meditado sobre como gerar números aleatórios num PLC? A reflexão sobre códigos ASCII e BCD empacotado foi proveitosa? Conhece melhor as possibilidades do seu PLC nesta matéria? Esta missão, para além de visualmente muito graciosa, permitiu, de facto, lançar e dar resposta a interessantes desafios para qualquer programador de PLCs.

Está pois praticamente terminada esta missão e com ela o jogo ITS PLC. Restam as últimas propostas ITS SUPER e ITS DEEP. E nelas, e noutras que se possam imaginar, há realmente ainda muito espaço para descobrir mais desafios aliciantes em torno do armazém automático ITS PLC.

ITS SUPER – Integração de sistemas de supervisão e terminais de operação

A generalidade das tarefas propostas nesta missão ganha de facto bastante com a inclusão de uma consola de interface homem-máquina. Com um equipamento deste tipo poderá, por exemplo:

- Lançar as diversas ordens através da consola em alternativa às botoneiras virtuais;
- Receber códigos de erro mais detalhados do que o simples piscar de uma luz;
- Definir variáveis de interesse tais como “Lugar”, “Caixa_Id” e “Classe_Id”;
- Consultar e inicializar as tabelas de ocupação do armazém;
- Saber as mercadorias armazenadas de acordo com as suas classes.

Um sistema SCADA permite situações mais elaboradas e interessantes, tais como:

- Alterar as zonas reservadas às diferentes classes de produtos;
- Conhecer o tempo de permanência de cada mercadoria em armazém;
- Redistribuição de mercadorias, eliminando, por exemplo, casulos livres entre elas;
- Definição e seleção de diferentes critérios de levantamento de mercadorias;
- Permitir que determinadas mercadorias só possam ser solicitadas por determinados operadores;
- Criar e gerir filas de pedidos de armazenamentos e levantamentos;
- Registo dos tempos de funcionamento e indisponibilidade do armazém;

- Registo de situações de emergência e correspondente duração;
- Registo do histórico das operações;
- Registo de qualquer uma das situações anómalas propostas no pacote ITS DEEP que se segue.

ITS DEEP – Deteção e suporte de situações de erro

Tal como na missão 4, também alguns sensores foram ignorados nas soluções das tarefas propostas dado não terem um papel fundamental. Concretamente, os sensores 0, 2 e 4. Porém, são bastante interessantes para detetar situações de erro. Por exemplo:

- Defina um procedimento de alarme caso o sensor 0 esteja ativo quando o transelevador não é esperado na posição 51, e outro para o caso do mesmo sensor não estar ativo quando o transelevador é suposto estar no cais de entrada;
- Defina um procedimento de alarme caso os sensores 2 ou 4 não sejam ativados quando é dada ordem de avanço da plataforma, ou estejam ativos quando a plataforma é suposta estar recolhida.

Outros eventos anómalos cuja deteção é particularmente interessante nesta aplicação são os seguintes:

- Ordem de movimento do transelevador sem a plataforma recolhida;
- Falha na transferência de uma mercadoria do *chariot* de entrada para o transelevador:
- Falha na transferência de uma mercadoria do transelevador para o *chariot* de saída;
- Remoção imprevista de uma mercadoria do transelevador;
- Demora prolongada do transelevador a atingir a posição de destino;
- Ausência prolongada do *chariot* de entrada;
- Ausência prolongada do *chariot* de saída;
- Partida imprevista do *chariot* de entrada;
- Partida imprevista do *chariot* de saída;
- Avaria em qualquer sensor e identificação da mesma – saída sempre a 0 ou sempre a 1;
- Avaria em qualquer atuador e identificação da mesma – encravado a 0 ou a 1.

Propõe-se então que desenvolva mecanismos de deteção das anomalias acima listadas, e utilize as possibilidades de interação e de geração de avarias do ITS PLC para testar devidamente as suas propostas.

PARTE 3

AS SOLUÇÕES

Muito provavelmente, o secreto desejo do leitor, de que as soluções lhe fossem fornecidas na linguagem de programação do seu PLC, não se concretizará. Mas, com tantas marcas e modelos de PLCs, a probabilidade de tal acontecer era, de facto, muito baixa!

Na impossibilidade de apresentar as soluções na linguagem preferida de cada utilizador, optou-se por fazê-lo numa linguagem muito eficiente, de crescente aceitação, facilmente compreensível mesmo para quem jamais contactou com ela, facilmente traduzível noutras linguagens de programação de PLCs e extremamente expressiva e reveladora dos algoritmos de controlo empregues: o Texto Estruturado.

O Texto Estruturado e a Norma IEC 61131-3

O Texto Estruturado é uma linguagem de programação de PLCs moderna e contemplada na norma IEC 61131-3. A sua simplicidade e expressividade têm-lhe granjeado um número sempre crescente de adeptos entre programadores e fabricantes de PLCs.

Se é a primeira vez que contacta com esta linguagem, considere tal facto como mais um ponto de interesse deste livro de exercícios. Verá que, em pouco tempo, será mais um fã do Texto Estruturado. E, se conhece minimamente as vulgares linguagens de programação de alto nível, como o BASIC, Pascal ou C, vai constatar que o Texto Estruturado é, afinal, uma linguagem muito fácil de compreender e utilizar e, de certa forma, já sua conhecida.

Para muitos autores, o Texto Estruturado é a linguagem que permite uma programação mais rápida e eficiente dos modernos sistemas de controlo baseados em PLC. Conquanto tal seja obviamente discutível, a verdade é que os benefícios desta linguagem relativamente às mais clássicas são particularmente notórios em aplicações complexas. Mas um outro fator tem contribuído em muito para a eficiente conceção, realização e reengenharia de soluções baseadas em PLC: a norma IEC 61131. Se não conhece esta norma é tempo de a conhecer. Se trabalha ou estuda numa Universidade, encontrá-la-á certamente na biblioteca da sua instituição. Se não é o caso, a consulta de alguns dos inúmeros livros, artigos e sítios na Internet que dedicam grande atenção a esta norma, e em especial à Parte 3 (IEC 61131-3), podê-lo-ão ajudar. Apresentar as soluções dos exercícios propostos em Texto Estruturado serve assim também de pretexto para incitar o leitor a conhecer a norma IEC 61131, pese embora muitos dos seus aspetos fundamentais não se reflitam nas resoluções propostas.

Mas se é sua intenção continuar a utilizar uma linguagem de programação que "nada tem que ver com o Texto Estruturado" – esquemas de contactos, diagramas de blocos, lista de instruções – então, a norma IEC 61131-3 também é para si, já que ela contempla igualmente estas linguagens. Qualquer uma delas podia, por isso, ter sido utilizada para apresentar a resolução dos exercícios propostos, mantendo-se os mesmos propósitos de divulgação e conformidade com a norma. Não o

foram porque também subscrevemos a máxima de que o Texto Estruturado é a forma mais clara, simples e eficiente de expor um programa de PLC, sobretudo quando há que fazê-lo para um público heterogéneo.

A norma IEC 61131-3 inclui a linguagem gráfica SFC, *Sequential Function Chart*, com o propósito de estruturar, sequenciar e desenvolver um programa de controlo. Daí que o SFC não seja, só por si, uma alternativa ao Texto Estruturado ou a qualquer uma das outras linguagens previstas na norma. Contudo, a linguagem de programação SFC é um dos recursos mais interessantes e importantes da norma IEC 61131-3, sendo muitas vezes confundida com a linguagem de especificação *GRAFCET*, tal como definida na primeira versão da norma IEC 60848, em virtude da semelhança semântica e gráfica destas. O facto do *GRAFCET* ser uma importante porta de acesso à compreensão de alguns aspetos da norma IEC 61131-3, nomeadamente em termos de organização de código, é mais uma razão para a sua ampla utilização neste texto em detrimento de outras ferramentas e metodologias de especificação.

Se o PLC que utiliza é minimamente compatível com a norma IEC 61131-3 mas só suporta as linguagens mais clássicas, ou se, de qualquer modo, as pretende continuar a utilizar, então tem nestes exercícios um outro ponto de interesse: a tradução das soluções apresentadas para a "sua" linguagem de programação, mas usando a semântica e simbologia prevista na norma IEC 61131-3. Há, afinal, muitos motivos de interesse neste livro de exercícios!

Por último, se é adepto do Texto Estruturado e desenvolve habitualmente a sua programação de acordo com a norma IEC 61131-3, vai certamente considerar os programas apresentados deselegantes, ineficientes e até algo desconformes com o espírito da norma. Concordando em absoluto que o Texto Estruturado e a norma IEC 61131-3 permitem soluções bem mais interessantes do que as apresentadas – por exemplo, usando *enumerated data types* para lidar com evoluções de máquinas de estados, ou encapsulando e reutilizando sistematicamente código do utilizador em funções e blocos funcionais – há que dizer que tais abordagens lançariam, por certo, grandes confusões nos desconhecedores da norma e da linguagem de programação utilizada. E as soluções apresentadas pretendem ter dois méritos: primeiro, e mais fundamental, serem compreensíveis para qualquer programador de PLCs, mesmo que só minimamente habilitado; segundo, proporcionarem a cada leitor o exercício complementar de as traduzirem ou otimizarem para a(s) sua(s) linguagem(ns) de interesse. Consequentemente, se é fã do Texto Estruturado, encare as soluções apresentadas como pedaços de código a otimizar de acordo com a norma IEC 61131-3 e, já agora, também com a norma IEC 61499.

Literatura e Materiais de Apoio

Caso pretenda iniciar-se na norma IEC 61131 e no Texto Estruturado tem múltiplos pontos de partida:

Encontrar a norma IEC 61131 numa biblioteca não deverá ser difícil. Outra hipótese é adquiri-la junto da *International Electrotechnical Commission* – http://www.iec.ch.

A parte mais interessante da norma é a Parte 3, toda ela dedicada às Linguagens de Programação.

Tem a designação IEC 61131-3. É apresentada, discutida e exemplificada em diversos livros. Entre eles, destacam-se os seguintes:

- Robert W. Lewis – "*Programming Industrial Control Systems Using IEC 1131-3*" (IEE Control Engineering Series), 1998, ISBN: 0-85296-950-3;

- Karl-Heinz John – "*IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Aids to Decision-Making Tools*". 1995, Springer Verlag. ISBN 3-540-67752-6. Parte deste livro está disponível na Internet a partir da página pessoal do autor: http://www.fen-net.de/karlheinz.john/.

Um sítio obrigatório em matéria de normas e PLCs é o da PLCOPEN – www.plcopen.org. Múltiplas informações úteis podem aqui ser encontradas.

Uma consulta ao *Google*, ou a qualquer outro motor de busca, sobre o tema "*structured text programming*" conduz a uma lista infindável de sítios e artigos interessantes. Entre eles, o texto "*Automating Manufacturing Systems with PLCs*" por Hugh Jack, cuja leitura é absolutamente recomendada e disponível, à data desta publicação, no sítio claymore.engineer.gvsu.edu/~jackh/books/plcs.

Se não possui um PLC compatível com a norma IEC 61131-3, mas gostaria de experimentar a programação em Texto Estruturado, então sugere-se uma viagem ao mundo dos *Soft PLCs*. Muitos destes pacotes de software são totalmente compatíveis com a norma IEC 61131-3. Têm um custo elevado, mas os seus fabricantes disponibilizam versões de demonstração gratuitas, muito bem documentadas e exemplificadas, que, embora naturalmente limitadas, constituem ótimos pontos de partida para a programação segundo a norma IEC 61131-3. Sugere-se que conheça os seguintes produtos e fabricantes:

- ISaGRAF – http://www.isagraf.com;

- MULTIPROG – http://www.kw-software.com;

- CoDeSys – http://www.3s-software.com;

- TwinCAT PLC – http://www.beckhoff.com.

Se não conhece os *Soft PLCs*, então este é um excelente pretexto para, finalmente, contactar com este interessantíssimo meio de controlo de muitas das modernas instalações industriais. Mais um ponto de interesse deste livro de exercícios!

Acerca das Soluções

Tecidas algumas considerações sobre a norma IEC 61131-3 é, finalmente, tempo de passar à divulgação das soluções dos problemas propostos.

Cada resolução compreende uma explicação e justificação, mais ou menos breve, dos procedimentos utilizados, seguida do respetivo programa em Texto Estruturado, devidamente comentado. Porque, na generalidade dos casos, as tarefas se vão complementando à medida que uma missão avança, a solução de uma tarefa é vulgarmente o ponto de partida para a seguinte. Consequentemente, excertos do código desenvolvido no contexto de uma tarefa tendem a ser reutilizados em tarefas posteriores. Contudo, os comentários que acompanharam a primeira apresentação de tais excertos de código são omitidos nas tarefas seguintes, conseguindo-se desse modo evitar fastidiosas repetições e dar, simultaneamente, maior espaço e visibilidade aos comentários centrais à tarefa em causa. Daí, volta-se a frisar, a importância de compreender devidamente a solução de uma tarefa antes de avançar para a seguinte.

No caso de soluções menos triviais ou propensas a alguma ambiguidade, os programas são desenvolvidos com base numa especificação prévia em *GRAFCET*, devidamente explicada. É importante referir que tais especificações seguem a segunda edição da norma IEC 60848. Dado que esta versão é relativamente recente, é provável que alguns leitores não conheçam ainda suficientemente bem as alterações introduzidas nesta versão, podendo daí resultar alguma estranheza relativamente a alguns aspetos semânticos, gráficos e funcionais de uns tantos *grafcets*; por exemplo, a afetação de ações a algumas transições. Embora, e a pensar exatamente nos leitores menos familiarizados com a versão em vigor, as soluções apresentadas apontem e alertem para algumas das inovações introduzidas na segunda edição da norma IEC 60848, tais esclarecimentos não dispensam a consulta da mesma. A promoção da segunda edição da norma IEC 60848 que, relativamente à primeira, se distanciou bastante da linguagem SFC, é outro propósito deste livro!

A declaração e inicialização das variáveis é um aspeto já de si importante em qualquer programa, mas é uma questão particularmente relevante quando se programa de acordo com a norma IEC 61131-3. Nesse contexto, a declaração das variáveis merece um espaço próprio nas soluções apresentadas, tal como deve também merecer uma atenção especial por parte do leitor. E isto porque, a declaração das variáveis compreende também a inicialização, implícita ou explicita, das mesmas com um valor de interesse para os programas que as utilizam. Como tal, a declaração de uma variável não deve ser vista como um mero formalismo de pouco interesse para as soluções apresentadas, mas antes como uma informação particularmente importante. Por exemplo, a atribuição do valor inicial TRUE a uma variável de estado revela, ou confirma, que a mesma simboliza uma etapa ou estado inicial.

Por questão de organização, as variáveis estão divididas em dois grupos: as que são comuns a todas as missões e as que apenas são válidas dentro cada missão. O primeiro grupo é composto exclusivamente pelas variáveis de I/O. São apresentadas no ponto seguinte. Mais à frente, já na esfera das soluções de cada missão, surge a declaração das variáveis e das instâncias de blocos funcionais utilizadas nas tarefas afetas à missão em causa.

Como nota final, importa sublinhar, mais uma vez, que as soluções apresentadas são apenas "soluções possíveis" e, de forma alguma, as únicas ou as melhores. Sugere-se, por isso, que o leitor as compare com as suas, tentando listar vantagens e inconvenientes relativos. A elaboração de um caderno de notas refletindo as conclusões pessoais sobre a resolução de cada tarefa é a última proposta deste livro de exercícios e, sem dúvida, uma das mais importantes para formadores e formandos.

Declaração das Variáveis de I/O

As variáveis que representam os parâmetros de entrada e de saída de um programa são globalmente designadas por variáveis de I/O. As variáveis de I/O são fundamentais em qualquer programa, já que são o meio de troca de informação entre este e os seus elementos periféricos, físicos ou lógicos, tais como sensores, atuadores, sinalizadores ou outras aplicações de software.

No caso da aplicação ITS PLC, a correta troca de informação entre os sistemas virtuais e o PLC externo é uma condição necessária ao seu correto funcionamento. Para tal, é necessário que ao PLC cheguem informações sobre o estado da instalação, na forma de variáveis de entrada do PLC, e que o estado da instalação se altere de acordo com ações de controlo adequadas, refletidas nas variáveis de saída do PLC. Esta troca de informação requer uma ligação física entre o PLC e a placa de interface USB que acompanha o produto. É dessa ligação física que resulta o mapeamento das variáveis de I/O do PLC nas variáveis de I/O dos sistemas virtuais ITS PLC.

A declaração das variáveis de I/O tem três propósitos: atribui-lhes nomes que podem ser usados na escrita do programa, definir os tipos de variáveis em jogo e afetá-las aos endereços físicos de I/O do PLC. Consequentemente, ao declarar as variáveis, cada programador tenderá a apelidá-las a seu gosto e a mapeá-las em função do PLC utilizado. De comum a todas as declarações, haverá apenas a afirmação de que cada ponto de I/O corresponde a uma variável *booleana*.

A norma IEC 61131-3 prevê a hipótese de, na declaração de uma variável, afetá-la a um espaço físico do PLC (memória, entrada ou saída). Esta hipótese, particularmente interessante na declaração das variáveis de I/O, reflete-se no emprego do atributo "AT". Assim, assumindo um PLC com módulos de entrada e saída de 8 *bits*, a declaração das variáveis de I/O pode ser feita do seguinte modo:

```
(*************************************************
      Declaração das Variáveis de Entrada e Saída
*************************************************)

VAR_INPUT

   In_0 AT %IX0.0 : BOOL; (* Sensor 0 *)
   In_1 AT %IX0.1 : BOOL; (* Sensor 1 *)
   In_2 AT %IX0.2 : BOOL; (* Sensor 2 *)
   In_3 AT %IX0.3 : BOOL; (* Sensor 3 *)
```

```
   In_4 AT %IX0.4 : BOOL; (* Sensor 4 *)
   In_5 AT %IX0.5 : BOOL; (* Sensor 5 *)
   In_6 AT %IX0.6 : BOOL; (* Sensor 6 *)
   In_7 AT %IX0.7 : BOOL; (* Sensor 7 *)
   In_8 AT %IX1.0 : BOOL; (* Sensor 8 *)
   In_9 AT %IX1.1 : BOOL; (* Sensor 9 *)
   In_10 AT %IX1.2 : BOOL; (* Sensor 10 *)
   In_11 AT %IX1.3 : BOOL; (* Seletor de Modo Manual/Automático *)
   In_12 AT %IX1.4 : BOOL; (* Botoneira Iniciar *)
   In_13 AT %IX1.5 : BOOL; (* Botoneira Parar *)
   In_14 AT %IX1.6 : BOOL; (* Botoneira Reiniciar *)
   In_15 AT %IX1.7 : BOOL; (* Botoneira Emergência *)

END_VAR

VAR_OUPUT

   Out_0 AT %QX0.0 : BOOL; (* Atuador 0 *)
   Out_1 AT %QX0.1 : BOOL; (* Atuador 1 *)
   Out_2 AT %QX0.2 : BOOL; (* Atuador 2 *)
   Out_3 AT %QX0.3 : BOOL; (* Atuador 3 *)
   Out_4 AT %QX0.4 : BOOL; (* Atuador 4 *)
   Out_5 AT %QX0.5 : BOOL; (* Atuador 5 *)
   Out_6 AT %QX0.6 : BOOL; (* Atuador 6 *)
   Out_7 AT %QX0.7 : BOOL; (* Atuador 7 *)
   Out_8 AT %QX1.0 : BOOL; (* Luz da botoneira Iniciar *)
   Out_9 AT %QX1.1 : BOOL; (* Luz da botoneira Reiniciar *)

END_VAR

(*************************************************
                  Fim da declaração
*************************************************)
```

A generalidade dos ambientes de programação de PLCs prevê procedimentos de declaração de variáveis mais ou menos semelhantes ao representado. Mas, admitindo que a declaração pode ser confusa para os leitores menos familiarizados com a norma IEC 61131-3, o que importa reter desta declaração é o seguinte:

- Em todas as tarefas, de todas as missões, as variáveis de entrada *booleanas* In_0, In_1, …, In_10 mapeiam, por esta ordem, os valores lógicos produzidos pelos sensores virtuais 0, 1, …, 10;

- Em todas as tarefas, de todas as missões, as variáveis de entrada *booleanas* In_11, In_12, …, In_15 mapeiam, por esta ordem, o estado do seletor de Modo e das botoneiras Iniciar, Parar, Reiniciar e Emergência;

- Em todas as tarefas, de todas as missões, as variáveis de saída *booleanas* Out_0, Out_1, …, Out_7 mapeiam, por esta ordem, os valores lógicos dos atuadores virtuais 0, 1, …, 7;

- Em todas as tarefas, de todas as missões, as variáveis de saída *booleanas* Out_8 e Out_9 mapeiam os valores lógicos dos sinalizadores luminosos das botoneiras Iniciar e Reiniciar, respetivamente.

A norma IEC 61131-3 possibilita a inicialização de uma variável aquando da sua declaração. Esta possibilidade não tem interesse para as variáveis de I/O, e nem sequer é possível, porque não faz sentido, para as variáveis de entrada. Por esse motivo, as variáveis de I/O não são inicializadas na respetiva declaração.

MISSÃO 1: AUTOMATIZAÇÃO DE UMA ESTAÇÃO DE TRANSPORTE E TRIAGEM DE MERCADORIAS EM PALETES

Variáveis e instâncias de blocos funcionais usadas na Missão 1

A cada tarefa de cada missão corresponde um programa que emprega um conjunto de variáveis e instâncias de blocos funcionais, cuja declaração é exigida pela norma IEC 61131-3. Porém, apresentar as soluções das diferentes tarefas seguindo este princípio, levaria a que o código de cada programa fosse antecedido de uma lista de variáveis e instâncias de blocos funcionais, por vezes extensa, com muitas delas já introduzidas e explicadas nas soluções de tarefas anteriores.

Para evitar sucessivas e fastidiosas declarações, que em pouco ou nada contribuiriam para uma melhor compreensão dos sucessivos programas e assim aligeirar o documento, optou-se por apresentar uma única declaração por missão, a qual reúne todas as variáveis e instâncias de blocos funcionais usadas nos doze programas da missão em causa. Essa declaração conjunta, que abre a apresentação das soluções de cada missão, é feita em conformidade com a norma IEC 61131-3 e deve ser tomada em devida conta na apreciação do código correspondente à solução de cada tarefa. Em particular, é de especial importância a observação do valor inicial atribuído a cada variável.

De acordo com a norma IEC 61131-3, a definição do valor inicial de uma variável permite atribuir-lhe um valor diferente daquele que seria o seu valor inicial por defeito, o qual é função do tipo de variável; por exemplo, variáveis *booleanas*, inteiras e reais iniciam-se, por defeito, em 0. Parecerá pois estranho que, na declaração de algumas variáveis, lhes sejam atribuídos os valores que teriam por defeito; por exemplo, é possível observar que muitas variáveis *booleanas* são inicializadas em FALSE. Tal "redundância" tem o propósito de revelar explicitamente o valor inicial de algumas variáveis particularmente importantes. Por exemplo, no caso de uma variável de estado, o seu valor inicial deve tipicamente refletir uma condição inicial prevista no cenário em que a tarefa se desenrola; logo, tem de ser inicializada com TRUE ou FALSE, em conformidade. Se essa inicialização for bem explícita, tanto melhor...

Assim, e muito concretamente, optou-se por inicializar explicitamente mesmo aquelas variáveis que implicitamente seriam inicializadas com os valores de interesse porque, mostra a experiência, quando a inicialização de uma variável não é explícita, alguns leitores têm tendência a deduzir que o valor inicial dessa variável é indiferente, não tomando desse modo consciência da importância do valor inicial que, por defeito, lhe é atribuído. Em coerência com este princípio, entendeu-se por bem inicializar explicitamente todas as variáveis aquando da sua declaração, exceto aquelas cujo valor inicial é, de facto, irrelevante.

Os casos em que a mesma variável deve ser inicializada com valores distintos em diferentes tarefas são devidamente salientados nos comentários que acompanham a respetiva declaração. Também são salientados os casos em que os valores iniciais são apenas exemplificativos; por

exemplo, parâmetros de configuração, tabelas de receitas, etc. Em qualquer caso, a declaração de uma variável encerra em comentário o significado da mesma, a codificação empregue e as tarefas em que é utilizada. As variáveis são agrupadas em função do seu contexto e, dentro de cada grupo, listadas por ordem alfabética. Tal permite enquadrar devidamente a funcionalidade das diferentes variáveis e encontrar com facilidade a declaração de qualquer uma delas. Por último, há que referir que a declaração das variáveis antecede a declaração das instâncias dos blocos funcionais.

Prestadas estas explicações, segue-se a declaração das variáveis empregues nas doze tarefas que compõem a Missão 1, a qual corresponde à seguinte listagem:

```
(*****************************************************
        Declaração das variáveis e instâncias de
    blocos funcionais IEC 61131-3 usadas na Missão 1
*****************************************************)

VAR
    (* Declaração e inicialização de variáveis *)

    (* Variáveis afetas ao comando do tapete de entrada *)

    Busy_1 : BOOL := FALSE; (* Tarefas 2-3, 5-12 *)
    (* Disponibilidade do tapete de entrada: 0 -> Disponível *)
    Count   : WORD := 16#8000; (* Tarefas 3, 5-12 *)
    (* Contador em anel, inicialmente a zero -> 8000H *)
    Fila : WORD; (* Tarefas 8-12 *)
    (* Fila das alturas das paletes no tapete de entrada *)
    Mem_1 : BOOL := FALSE; (* Tarefas 1-3, 5-12 *)
    (* Estado do tapete de entrada: 0 -> Sem paletes *)
    Mem_timer : BOOL; (* Tarefas 9-12 *)
    (* Memória para retenção do Timer_1 *)

    (* Variáveis afetas ao comando da mesa rotativa *)

    Busy_2 : BOOL := FALSE; (* Tarefas 5-12 *)
    (* Disponibilidade da mesa rotativa: 0 -> Disponível *)
    Carrega : BOOL := FALSE; (* Tarefas 4-12 *)
    (* Estado possível da mesa rotativa *)
    Descarrega : BOOL := FALSE; (* Tarefas 4-12 *)
    (* Estado possível da mesa rotativa *)
    Livre : BOOL := TRUE; (* Tarefas 4-12 *)
    (* Estado inicial da mesa rotativa *)
    Mem_timeout_mesa : BOOL := FALSE; (* Tarefas 11-12 *)
    (* Memória para retenção de Timeout_mesa *)
    Roda_carregada : BOOL := FALSE; (* Tarefas 4-12 *)
    (* Estado possível da mesa rotativa *)
    Roda_descarregada : BOOL := FALSE; (* Tarefas 4-12 *)
```

```
(* Estado possível da mesa rotativa *)
Sentido_descarga : BOOL := FALSE; (* Tarefas 6-12 *)
(* Se 0, a descarga é para o tapete da direita *)

(* Variáveis afetas ao comando do tapete de saída da direita *)

Busy_6 : BOOL := FALSE; (* Tarefas 7-12 *)
(* Disponibilidade do tapete de saída da direita: 0 -> Disponível *)
Mem_6 : BOOL := FALSE; (* Tarefas 5-12 *)
(* Estado do tapete de saída da direita: 0 -> Sem paletes *)
Mem_timeout_direita : BOOL := FALSE; (* Tarefas 11-12 *)
(* Memória para retenção de Timeout_direita *)

(* Variáveis afetas ao comando do tapete de saída da esquerda *)

Busy_5 : BOOL := FALSE; (* Tarefas 7-12 *)
(* Disponibilidade do tapete de saída da esquerda:0 -> Disponível *)
Mem_5   : BOOL := FALSE; (* Tarefas 6-12 *)
(* Estado do tapete de saída da esquerda: 0 -> Sem paletes *)
Mem_timeout_esquerda : BOOL := FALSE; (* Tarefas 11-12 *)
(* Memória para retenção de Timeout_esquerda *)

(* Variáveis afetas aos modos de funcionamento da instalação *)

Automatico : BOOL := FALSE; (* Tarefas 10-12 *)
(* Estado possível da instalação - inicial na Tarefa 10 *)
Encerramento : BOOL := FALSE; (* Tarefas 10-12 *)
(* Estado possível da instalação *)
Limpeza : BOOL := FALSE; (* Tarefas 11-12 *)
(* Estado possível da instalação *)
Parada : BOOL; (* Tarefas 10-12 *)
(* Instalação parada *)

Pronto : BOOL := FALSE; (* Tarefas 10-12 *)
(* Estado possível da instalação *)
Standby : BOOL := TRUE; (* Tarefas 11-12 *)
(* Estado possível da instalação - inicial nas Tarefas 11-12 *)

(* Instâncias de Blocos Funcionais *)

(* Deteção de transições descendentes *)

F_In_0 : F_TRIG; (* Tarefas 2-3, 5-12 *)
(* Transição descendente de In_0 *)
F_In_3 : F_TRIG; (* Tarefas 1-12 *)
(* Transição descendente de In_3 *)
F_In_7 : F_TRIG; (* Tarefas 4-12 *)
```

```
(* Transição descendente de In_7 *)
F_In_8 : F_TRIG; (* Tarefas 6-12 *)
(* Transição descendente de In_8 *)
F_In_9 : F_TRIG; (* Tarefas 5-12 *)
(* Transição descendente de In_9 *)
F_In_10 : F_TRIG; (* Tarefas 6-12 *)
(* Transição descendente de In_10 *)

(* Deteção de transições ascendentes *)

R_In_0 : R_TRIG; (* Tarefa 1 *)
(* Transição ascendente de In_0 *)
R_Limpeza : R_TRIG; (* Tarefas 11-12 *)
(* Transição ascendente de Limpeza *)
R_Parada : R_TRIG; (* Tarefas 11-12 *)
(* Transição ascendente de Parada – Instalação parou *)

(* Temporizadores *)

Timer_1 : TON; (* Tarefas 9-12 *)
(* Usado no comando do tapete de entrada *)
Pisca_encerra_auto_1 : TON; (* Tarefa 12 *)
(* Temporizador 1 do Pisca_encerra_auto *)
Pisca_encerra_auto_2 : TON; (* Tarefa 12 *)
(* Temporizador 2 do Pisca_encerra_auto *)
Pisca_encerramento_1 : TON; (* Tarefas 10-12 *)
(* Temporizador 1 do Pisca_encerramento *)
Pisca_encerramento_2 : TON; (* Tarefas 10-12 *)
(* Temporizador 2 do Pisca_encerramento *)
Pisca_limpeza_1 : TON; (* Tarefas 11-12 *)
(* Temporizador 1 do Pisca_limpeza *)
Pisca_limpeza_2 : TON; (* Tarefas 11-12 *)
(* Temporizador 2 do Pisca_limpeza *)
Timeout_direita : TON; (* Tarefas 11-12 *)
(* Timeout no comando do tapete de saída da direita *)
Timeout_entrada : TON; (* Tarefas 11-12 *)
(* Timeout no comando do tapete de entrada *)
Timeout_esquerda : TON; (* Tarefas 11-12 *)
(* Timeout no comando do tapete de saída da esquerda *)
Timeout_mesa : TON; (* Tarefas 11-12 *)
(* Timeout no comando da mesa rotativa *)

(* Contadores *)

Conta_paletes : CTD; (* Tarefa 12 *)
(* Contador de paletes *)
```

```
END_VAR

(*******************************************************
                    Fim da declaração
*******************************************************)
```

Resolução da Tarefa 1

De acordo com o enunciado, o tapete de entrada deve movimentar-se quando nele existe uma palete. Então, admitindo que é possível representar o estado do tapete por uma variável binária (ou memória), "Mem_1", tal que Mem_1 = TRUE se existe uma palete no tapete, o comando deste transportador passa simplesmente pela expressão Out_1 = Mem_1.

Dado que não existe um sensor que indique se o tapete tem ou não uma palete, o valor lógico de Mem_1 terá de ser deduzido; concretamente, com base na observação da entrada e saída de paletes no tapete. Assim, a entrada de uma palete deve fazer Mem_1 = TRUE – ou seja, o *set* de Mem_1; a saída de uma palete deve fazer Mem_1 = FALSE – ou seja, o *reset* de Mem_1. Mem_1 deve ser inicializada em FALSE uma vez que é dito que inicialmente não há paletes no tapete de entrada.

Em termos lógicos, a entrada e saída de paletes no tapete refletem-se na transição ascendente de In_0 (↑In_0 = 1) e na transição descendente de In_3 (↓In_3 = 1), respetivamente. O *set* e o *reset* de Mem_1 devem pois ser motivados por estas transições.

O programa correspondente à presente tarefa é o seguinte:

```
(*****************************************
           Missão 1 - Tarefa 1
*****************************************)

(**** DETEÇÃO DE EVENTOS RELEVANTES ****)

R_In_0 (CLK := In_0);
(* R_In_0.Q é 1 quando In_0 tem uma transição ascendente. Sinaliza que uma palete
chegou ao tapete de entrada *)

F_In_3 (CLK := In_3);
(* F_In_3.Q é 1 quando In_3 tem uma transição descendente. Sinaliza que uma palete
abandonou o tapete de entrada *)

(**** COMANDO DO TAPETE DE ENTRADA ****)

(* Definição da variável de estado, Mem_1, em função dos sinais de entrada *)

IF R_In_0.Q THEN (* Quando chega uma palete *)
```

```
    Mem_1 := TRUE; (* Set de Mem_1 *)
END_IF;

IF F_In_3.Q THEN (* Quando sai uma palete *)
    Mem_1 := FALSE; (* Reset de Mem_1 *)
END_IF;

(* Definição da variável de saída, Out_1, em função do estado do tapete de entrada *)
Out_1 := Mem_1;

(*****************************************
            Fim do Programa
*****************************************)
```

Resolução da Tarefa 2

Este problema é bastante simples, mas encerra ideias de importância fundamental para a generalidade das tarefas da presente missão – na realidade, para a generalidade dos problemas de transporte de mercadorias em passadeiras.

É muito importante notar que o transporte de uma palete tem três fases. Veja-se para o caso do tapete de entrada:

- A carga da palete – requer o movimento simultâneo do alimentador e do tapete de entrada. Inicia-se quando a palete é detetada pelo sensor 0 e termina quando o mesmo sensor deixa de a detetar. Assim, durante esta fase, o valor lógico de In_0 é 1;

- O transporte, propriamente dito – requer unicamente o movimento do tapete de entrada. Durante esta fase, a palete não é detetada pelos sensores colocados nos extremos do tapete, pelo que In_0 = In_3 = 0;

- A descarga da palete – requer o movimento simultâneo do tapete de entrada e dos rolos da mesa rotativa. Inicia-se quando a palete é detetada pelo sensor 3 e termina quando o mesmo sensor deixa de a detetar. Assim, durante esta fase, o valor lógico de In_3 é 1.

Porque a carga do tapete de entrada corresponde à descarga do alimentador – tal como a descarga do tapete de entrada corresponde à carga da mesa rotativa – então, estando o tapete de entrada indisponível, isto é, transportando ele uma palete, o alimentador não pode descarregar. Ou seja, o alimentador não pode passar uma palete para além da posição do sensor 0. Só o poderá fazer quando o tapete de entrada ficar disponível, isto é, quando nele não existir qualquer palete.

Uma solução genérica, para problemas deste tipo, é aceitar que um tapete A gera o sinal Busy_A = 1 quando, por qualquer razão, está indisponível para receber paletes. Ao ler este sinal, um tapete B, que alimente A, fica a saber que não deve movimentar paletes para além do seu limite. Só o poderá fazer quando o tapete A produzir o sinal Busy_A = 0.

A adaptação deste princípio ao problema em causa é muito simples: o controlador do tapete de entrada faz o *set* de Busy_1 após carregar uma palete – isto é, quando sente uma transição descendente de In_0 – e o *reset* de Busy_1 sempre que uma palete é descarregada – isto é, quando sente a transição descendente de In_3. Compete ao comando do alimentador ler o sinal Busy_1 de modo que, chegando uma palete ao fim do alimentador e estando o tapete de entrada ocupado, o movimento seja suspenso até que o tapete de entrada fique disponível.

Outro aspeto importante é que, por questões de eficiência, o tapete de entrada não deve parar após descarregar uma palete caso, nessa altura, já exista outra palete no final do alimentador. Assim, as condições de *set/reset* de Mem_1 têm de ser modificadas relativamente à Tarefa 1. Uma hipótese é só fazer o *reset* na transição descendente de In_3 se, nesse instante, In_0 = 0; outra, é fazer o *set* quando In_0 = 1 e torná-lo dominante sobre o *reset*. A solução apresentada segue a segunda hipótese.

```
(****************************************
           Missão 1 - Tarefa 2
****************************************)

(**** DETEÇÃO DE EVENTOS RELEVANTES ****)

F_In_0(CLK := In_0);
(* Palete foi carregada no tapete de entrada - ou seja, foi descarregada do
alimentador *)
F_In_3(CLK := In_3);

(**** COMANDO DO TAPETE DE ENTRADA ****)

(* Definição de Mem_1 e de Busy_1 *)

IF F_In_3.Q THEN (* Quando uma palete abandona o tapete *)
    Mem_1 := FALSE; (* Reset de Mem_1 *)
    Busy_1 := FALSE; (* Reset de Busy_1 *)
END_IF;

IF F_In_0.Q THEN (* Quando uma palete é carregada *)
    Busy_1 := TRUE; (* Set de Busy_1 *)
END_IF;

IF In_0 THEN (* Havendo palete no fim do alimentador *)
    Mem_1 := TRUE; (* Set de Mem_1 *)
END_IF;

(** NOTA IMPORTANTE:
Note que o set de Mem_1 é agora para In_0 = 1 e dominante sobre o
reset – uma vez que a instrução de set surge depois da de reset. Então, se F_In_3.Q e
```

```
In_0 forem ambos verdadeiros, Mem_1 fica com o valor lógico TRUE que entregará à saída
Out_1. Isso garante que o tapete de entrada continua em movimento, sem transições a
zero, caso já exista uma palete à saída do alimentador quando a anterior abandona o
tapete de entrada **)

(* Definição da saída Out_1 – como na Tarefa 1 *)
Out_1 := Mem_1;

(**** COMANDO DO ALIMENTADOR ****)

(* Alimentador não está em movimento se tapete de entrada ocupado e há uma palete no
final do alimentador *)
Out_0 := NOT Busy_1 OR NOT In_0;
(* O mesmo que: Out_0 := NOT (Busy_1 AND In_0) *)

(*************************************
            Fim do Programa
*************************************)
```

Resolução da Tarefa 3

As diferenças a introduzir no comando do tapete de entrada são muito poucas relativamente à Tarefa 2. Concretamente:

- Mem_1 só vai a 0 quando uma palete é descarregada e o tapete fica vazio;
- Busy_1 só vai a 1 quando passa a haver duas paletes no tapete.

O alimentador continua a ser comandado como na Tarefa 2.

Para saber quantas paletes existem em cada momento no tapete de entrada é necessário um contador crescente/decrescente. A esmagadora maioria dos PLCs inclui blocos de contagem deste tipo – normalmente designados CTUD – que, se capazes de sinalizar, de algum modo, os dois valores de interesse, 0 (tapete vazio) e 2 (lotação do tapete completa), servem os interesses da presente tarefa.

Outra hipótese é construir um contador à medida da aplicação. Neste caso, faz sentido – e ver-se-á mais à frente porquê –, construir um contador muito peculiar: um "contador em anel". Por definição, incrementar um contador em anel traduz-se em rodar o seu conteúdo uma posição para a esquerda; decrementá-lo é sinónimo de o rodar uma posição para a direita.

No presente caso, o contador em anel será materializado num registo de 16 *bits* – que é a extensão típica dos registos dos PLCs – denominado "Count". O código do número 0 corresponde, por convenção, à palavra que tem o *bit* mais a esquerda a 1 e todos os outros a 0. Assim, o conteúdo do registo Count denotará uma contagem com a evolução representada na Figura M1T3, pese embora, nesta tarefa, só evolua entre 0 e 2, a que correspondem os códigos 16#8000 e 16#0002,

respetivamente – o prefixo 16# significa que se está a usar notação hexadecimal. A rotação para a esquerda, incremento, é obtida pela instrução ROL (Count, 1). A rotação para a direita, decremento, é obtida pela instrução ROR (Count, 1).

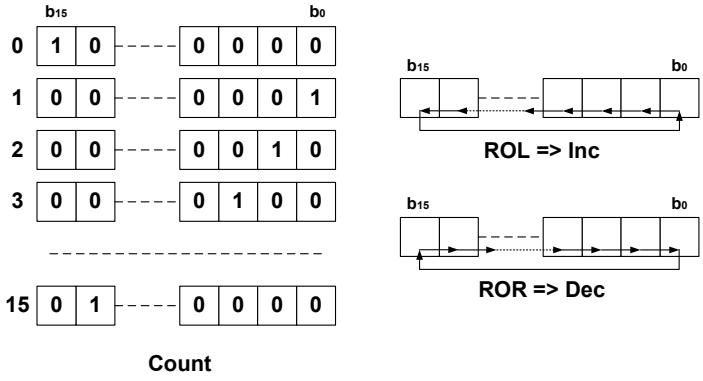


Figura M1T3 – *Evolução de um contador em anel*

As, por ora, misteriosas virtudes deste contador serão desvendadas numa futura tarefa; concretamente, quando for necessário conhecer a altura da palete descarregada na mesa rotativa e o tapete de entrada puder transportar um qualquer número de paletes. Mas, pode-se já adiantar, que tais virtudes têm que ver com o facto de o contador empregar uma codificação que, para valores diferentes de 0, espelha muito bem a fila de paletes no tapete de entrada: o 1 indica a posição da palete mais antiga (ou seja, a mais avançada) e os 0's à sua direita as paletes que a seguem...

Quanto às efetivas diferenças entre esta tarefa e a anterior, que têm basicamente que ver com os procedimentos relacionados com o contador, elas são as seguintes:

O registo Count é inicializado a 16#8000 (conferir na declaração das variáveis usadas na Missão 1) correspondendo à inicialização da contagem em 0. Sempre que uma caixa entra no transportador o registo é rodado uma posição para esquerda – usando a instrução ROL (Count, 1). Se, após a contagem, exibir o valor 2, é feito o *set* de Busy_1. Sempre que In_3 transita de 1 para 0, o registo Count é rodado uma posição para a direita – ROR (Count, 1). Se a contagem chegar a 0 é feito o *reset* de Mem_1.

O código do programa que resolve esta tarefa é o seguinte:

```
(*****************************************
          Missão 1 - Tarefa 3
*****************************************)

(**** DETEÇÃO DE EVENTOS RELEVANTES ****)

F_In_0(CLK := In_0);
```

```
F_In_3(CLK := In_3);

(**** COMANDO DO TAPETE DE ENTRADA ****)

(* Se tem pelo menos uma palete: Mem_1 = TRUE Se tem duas paletes: Busy_1 = TRUE *)

IF F_In_0.Q THEN (* Quando é carregada uma palete *)
    Count := ROL(Count, 1); (* Contador incrementado *)

    IF (Count = WORD#16#2) THEN (* Se Contador em 2 *)
        Busy_1 := TRUE; (* Set de Busy_1 *)
    END_IF;
END_IF;

IF F_In_3.Q THEN (* Quando é descarregada uma palete *)
    Busy_1 := FALSE; (* Reset de Busy_1 *)
    Count := ROR(Count, 1); (* Contador decrementado *)

    IF (Count = WORD#16#8000) THEN (* Se Contador em 0 *)
        Mem_1 := FALSE; (* Reset de Mem_1 *)
    END_IF;
END_IF;

IF In_0 THEN (* Quando há palete no fim do alimentador *)
    Mem_1 := TRUE; (* Set de Mem_1 *)
END_IF;

Out_1 := Mem_1;

(**** COMANDO DO ALIMENTADOR ****)
(* Como na Tarefa 2 *)

Out_0 := NOT Busy_1 OR NOT In_0;

(***************************************
              Fim do Programa
***************************************)
```

Resolução da Tarefa 4

O *grafcet* apresentado na Figura M1T4 mostra as cinco etapas (ou estados) a considerar no comando da mesa rotativa. Correspondem à seguinte evolução:

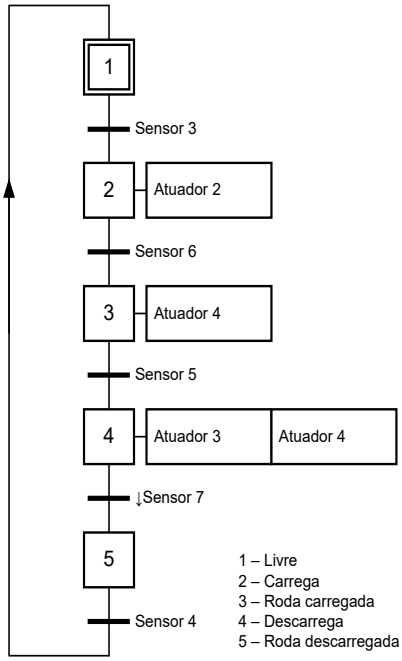


Figura M1T4 – *Grafcet do comando da mesa rotativa na Tarefa 4*

- Inicialmente a mesa está "Livre", isto é, apta a receber uma palete;
- Estando "Livre" e havendo uma palete no final do tapete de entrada (In_3 = 1) transita para "Carrega";
- Quando a palete está devidamente posicionada na mesa rotativa (In_6 = 1), transita de "Carrega" para "Roda carregada";
- De "Roda carregada" transita para "Descarrega" quando a mesa atinge a posição de descarga (In_5 = 1);
- De "Descarrega" transita para "Roda descarregada" quando a palete abandona a mesa rotativa (↓In_7 = 1);
- De "Roda descarregada" volta a "Livre" quando a mesa atinge a posição de carga (In_4 = 1).

Considerando que a descarga das paletes é feita para o tapete de saída da direita, as saídas (deduzidas das ações) relacionam-se com as etapas da seguinte forma:

- Carrega –> Out_2 = 1;
- Roda carregada –> Out_4 = 1;
- Descarrega -> Out_3 = Out_4 = 1.

Isto significa que:

- Out_2 = 1 quando Carrega;
- Out_3 = 1 quando Descarrega;
- Out_4 = 1 quando Roda carregada ou Descarrega.

A dedução de um programa de controlo que imponha à mesa rotativa o comportamento descrito no *grafcet* da Figura M1T4 passa pela tradução dos procedimentos enunciados em expressões lógicas.

Há duas metodologias para traduzir um *grafcet* num programa de PLC: a síncrona e a assíncrona. A metodologia síncrona assegura com naturalidade muitas das regras do *GRAFCET* dado que não permite que um *grafcet* evolua mais do que uma etapa em cada varrimento (ciclo de execução do programa), mesmo em situações instáveis. É, contudo, mais morosa de programar. A metodologia assíncrona tem uma programação mais expedita, mas algumas situações delicadas têm de ser devidamente reconhecidas e acauteladas. Por exemplo, situações de instabilidade que resultem na transposição de múltiplas transições num único varrimento só são aceitáveis se assegurarem que nenhuma ação importante deixa de ser realizada.

Detalhes sobre este interessante assunto, curiosamente pouco tratado pela literatura atual, transcendem os objetivos deste texto. Importante é referir que todas as soluções apresentadas que envolvem a tradução de *grafcets* para código de PLC têm por base a metodologia assíncrona. Nos casos em que um *grafcet* encerre aspetos peculiares a que a sua tradução tem de dar especial atenção, eles serão devidamente apontados.

A tradução do *grafcet* da Figura M1T4 para um programa de PLC não oferece dificuldades de maior. Concretamente, corresponde ao código que se apresenta de seguida, o qual ilustra bastante bem os procedimentos típicos deste tipo de tradução.

```
(*************************************
            Missão 1 - Tarefa 4
*************************************)

(**** DETEÇÃO DE EVENTOS RELEVANTES ****)

F_In_7(CLK := In_7);
```

```
(* Palete abandona a mesa rotativa pela direita *)

(**** COMANDO DA MESA ROTATIVA ****)

(* Transição de estados: transposição das transições e consequente ativação e
desativação das etapas *)

IF Livre AND In_3 THEN (* Livre -> Carrega *)
    Livre := FALSE;
    Carrega := TRUE;
END_IF;

IF Carrega AND In_6 THEN (* Carrega -> Roda carregada *)
    Carrega := FALSE;
    Roda_carregada := TRUE;
END_IF;

IF Roda_carregada AND In_5 THEN (* Roda carregada -> Descarrega *)
    Roda_carregada := FALSE;
    Descarrega := TRUE;
END_IF;

IF Descarrega AND F_In_7.Q THEN (* Descarrega -> Roda descarregada *)
    Descarrega := FALSE;
    Roda_descarregada := TRUE;
END_IF;

IF Roda_descarregada AND In_4 THEN (* Roda descarregada -> Livre *)
    Roda_descarregada := FALSE;
    Livre := TRUE;
END_IF;

(* Nota importante:
A variável de estado Livre é inicializada em TRUE dado que denota uma etapa inicial.
Todas as outras variáveis de estado são inicializadas em FALSE. Conferir na declara
ção das variáveis *)

(* Definição das saídas (ações) em função das etapas ativas *)

Out_2 := Carrega;
Out_3 := Descarrega;
Out_4 := Roda_carregada OR Descarrega;

(****************************************
            Fim do Programa
****************************************)
```

Resolução da Tarefa 5

O tapete de saída tem capacidade para transferir paletes a uma cadência superior à da mesa rotativa. Logo, em condições normais, está sempre pronto a receber uma palete que a mesa rotativa lhe queira enviar e nunca transporta mais do que uma palete em simultâneo. Assim, o comando do tapete de entrada desenvolvido na Tarefa 1 pode ser transposto para o comando do tapete de saída.

Por seu turno, a sincronização do tapete de entrada com a mesa rotativa passa pelos princípios empregues na Tarefa 2 para sincronizar o alimentador com o tapete de entrada. Ou seja:

- A mesa rotativa deve gerar o sinal Busy_2 = 1 quando está indisponível para receber paletes – o que acontece desde que carrega uma palete (isto é, ocorre a transição descendente de In_3) até que volta a ficar livre (isto é, a etapa Livre torna-se ativa);
- O movimento do tapete de entrada deve ser inibido quando uma palete chega ao sensor 3 e a mesa rotativa está indisponível para a receber.

O comando do alimentador não sofre alterações.

A presente tarefa tem pois uma solução globalmente simples que corresponde ao seguinte código:

```
(*************************************
           Missão 1 - Tarefa 5
*************************************)

(**** DETEÇÃO DE EVENTOS RELEVANTES ****)

F_In_0(CLK := In_0);
F_In_3(CLK := In_3);
F_In_7(CLK := In_7);
F_In_9(CLK := In_9); (* Palete abandona tapete de saída da direita *)

(**** COMANDO DO TAPETE DE SAÍDA DA DIREITA ****)

(* Mem_6 indica o estado do tapete de saída da direita: Se tem uma palete, então
Mem_6 = TRUE *)

IF F_In_9.Q THEN (* Quando uma palete abandona o tapete *)
    Mem_6 := FALSE; (* Reset de Mem_6 *)
END_IF;

IF In_7 THEN (* Quando há palete para carregar *)
    Mem_6 := TRUE; (* Set de Mem_6 *)
END_IF;
```

```
(* Tapete de saída da direita ligado se Mem_6 = TRUE *)
Out_6 := Mem_6;

(**** COMANDO DA MESA ROTATIVA ****)

IF Livre AND In_3 THEN
    Livre := FALSE;
    Carrega := TRUE;
END_IF;

IF Carrega AND In_6 THEN
    Carrega := FALSE;
    Roda_carregada := TRUE;
END_IF;

IF Roda_carregada AND In_5 THEN
    Roda_carregada := FALSE;
    Descarrega := TRUE;
END_IF;

IF Descarrega AND F_In_7.Q THEN
    Descarrega := FALSE;
    Roda_descarregada := TRUE;
END_IF;

IF Roda_descarregada AND In_4 THEN
    Roda_descarregada := FALSE;
    Livre := TRUE;
END_IF;

(* Definição do sinal Busy_2 *)
IF F_In_3.Q THEN (* Quando uma palete é carregada na mesa rotativa *)
    Busy_2 := TRUE; (* Set de Busy_2 *)
END_IF;

IF Livre THEN (* Quando mesa rotativa está Livre *)
    Busy_2 := FALSE; (* Reset de Busy_2 *)
END_IF;

(* Definição das saídas *)
Out_2 := Carrega;
Out_3 := Descarrega;
Out_4 := Roda_carregada OR Descarrega;

(**** COMANDO DO TAPETE DE ENTRADA ****)

IF F_In_0.Q THEN
```

```
    Count := ROL(Count, 1);

    IF (Count = WORD#16#2) THEN
        Busy_1 := TRUE;
    END_IF;
END_IF;

IF F_In_3.Q THEN
    Busy_1 := FALSE;
    Count := ROR(Count, 1);

    IF (Count = WORD#16#8000) THEN
        Mem_1 := FALSE;
    END_IF;
END_IF;

IF In_0 THEN
    Mem_1 := TRUE;
END_IF;

(* Redefinição de Out_1: tapete de entrada ligado se tem pelo menos uma palete e pode
movimentá-la *)
Out_1 := Mem_1 AND (NOT Busy_2 OR NOT In_3);

(**** COMANDO DO ALIMENTADOR ****)
(* Como na Tarefa 2 *)

Out_0 := NOT Busy_1 OR NOT In_0;

(***************************************
            Fim do Programa
***************************************)
```

Resolução da Tarefa 6

As soluções das Tarefas 1 e 5 permitem definir sem dificuldades o comando do tapete de saída da esquerda.

Já permitir que a mesa rotativa descarregue para a direita ou para a esquerda tem naturalmente reflexos no seu comando. O *grafcet* da Figura M1T6, que cobre este novo cenário, revela exatamente quais:

- A recetividade da transição 4/5 passa a ser a disjunção de dois eventos;
- As expressões das saídas Out_2 e Out_3 passam a incluir o sentido de descarga.

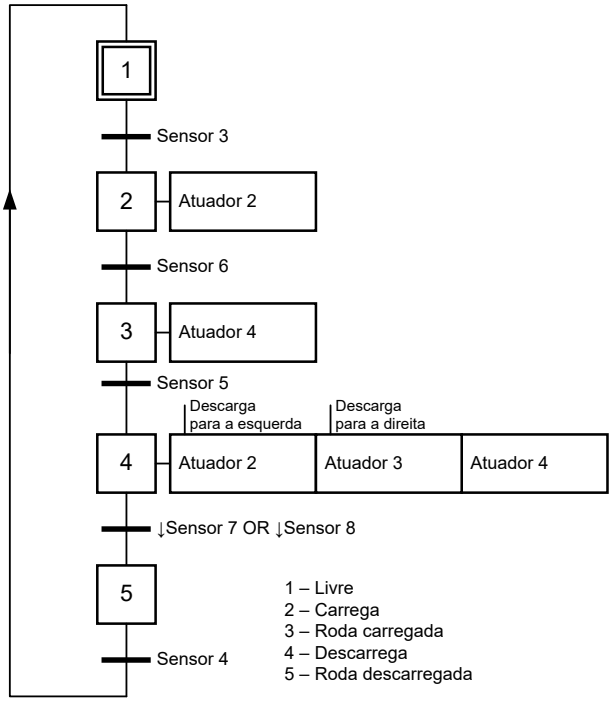


Figura M1T6 – *Comando da mesa rotativa com definição do sentido de descarga*

A primeira modificação é trivial de programar. A segunda passa pelo seguinte:

Uma vez que só existem dois tapetes de descarga, o sentido de descarga pode ser representado por uma variável binária. Chamemos-lhe "Sentido_descarga" e assuma-se que Sentido_descarga = 0 significa descarga para o tapete da direita.

Nesta base, as expressões de saída para o comando da mesa rotativa passam a ser as seguintes:

- Out_2 = 1 se Carrega OR Descarrega AND Sentido_descarga;
- Out_3 = 1 se Descarrega AND NOT Sentido_descarga;
- Out_4 = 1 se Roda carregada OR Descarrega.

Alternar o sentido de descarga requer que o valor lógico de Sentido_descarga seja complementado cada vez que uma palete passa por um determinado ponto. Porque, de futuro, o sentido de descarga será função das alturas das paletes, parâmetro esse que será reconhecido pelo comando do tapete de entrada e transmitido ao comando da mesa rotativa sempre que nesta é carregada uma palete, faz sentido utilizar a transição descendente de In_3 para complementar Sentido_descarga e incluir esse procedimento no comando do tapete de entrada.

É importante notar que, nesta situação, a variável Sentido_descarga está a ser complementada antes da descarga da palete num tapete de saída. Consequentemente, se Sentido_descarga for inicializada a 0, a primeira palete é descarregada para o tapete da esquerda.

```
(****************************************
           Missão 1 - Tarefa 6
****************************************)

(**** DETEÇÃO DE EVENTOS RELEVANTES ****)

F_In_0(CLK := In_0);
F_In_3(CLK := In_3);
F_In_7(CLK := In_7);
F_In_8(CLK := In_8); (* Palete abandona mesa rotativa pela esquerda *)
F_In_9(CLK := In_9);
F_In_10(CLK := In_10); (* Palete abandona tapete de saída da esquerda *)


(**** COMANDO DO TAPETE DE SAÍDA DA ESQUERDA ****)

(* Mem_5 indica o estado do tapete: se tem palete, Mem_5 = 1 *)

IF F_In_10.Q THEN (* Quando uma palete abandona o tapete *)
    Mem_5 := FALSE; (* Reset de Mem_5 *)
END_IF;

IF In_8 THEN (* Quando há palete para carregar *)
    Mem_5 := TRUE; (* Set de Mem_5 *)
END_IF;

(* Tapete de saída da esquerda ligado se tem uma palete *)
Out_5 := Mem_5;
```

```
(**** COMANDO DO TAPETE DE SAÍDA DA DIREITA ****)
(* Como na Tarefa 5 *)

IF F_In_9.Q THEN
    Mem_6 := FALSE;
END_IF;

IF In_7 THEN
    Mem_6 := TRUE;
END_IF;

Out_6 := Mem_6;

(**** COMANDO DA MESA ROTATIVA ****)

IF Livre AND In_3 THEN
    Livre := FALSE;
    Carrega := TRUE;
END_IF;

IF Carrega AND In_6 THEN
    Carrega := FALSE;
    Roda_carregada := TRUE;
END_IF;

IF Roda_carregada AND In_5 THEN
    Roda_carregada := FALSE;
    Descarrega := TRUE;
END_IF;

(* A descarga pode agora dar-se para qualquer um dos lados. Logo: *)

IF Descarrega AND (F_In_7.Q OR F_In_8.Q) THEN
    Descarrega := FALSE;
    Roda_descarregada := TRUE;
END_IF;

IF Roda_descarregada AND In_4 THEN
    Roda_descarregada := FALSE;
    Livre := TRUE;
END_IF;

IF F_In_3.Q THEN
    Busy_2 := TRUE;
END_IF;

IF Livre THEN
```

```
    Busy_2 := FALSE;
END_IF;

(* Definição das saídas em função do estado das etapas e do sentido de descarga *)
Out_2 := Carrega OR Descarrega AND Sentido_descarga;
(* Out_2 a 1 quando a Descarga é para a esquerda *)
Out_3 := Descarrega AND NOT Sentido_descarga;
(* Out_3 a 1 quando Descarga é para a direita *)
Out_4 := Roda_carregada OR Descarrega;

(**** COMANDO DO TAPETE DE ENTRADA ****)

IF F_In_0.Q THEN
    Count := ROL(Count, 1);

    IF (Count = WORD#16#2) THEN
        Busy_1 := TRUE;
    END_IF;
END_IF;

IF F_In_3.Q THEN
    Sentido_descarga := NOT Sentido_descarga;
    (** Definição do sentido de descarga:
    O sentido de descarga é alterado sempre que uma palete é descarregada do tapete
     de entrada para a mesa rotativa. Se Sentido_descarga = 0, a descarga é para a
    direita. **)

    Busy_1 := FALSE;
    Count := ROR(Count, 1);

    IF (Count = WORD#16#8000) THEN
        Mem_1 := FALSE;
    END_IF;
END_IF;

IF In_0 THEN
    Mem_1 := TRUE;
END_IF;

Out_1 := Mem_1 AND (NOT Busy_2 OR NOT In_3);

(**** COMANDO DO ALIMENTADOR ****)
(* Como na Tarefa 2 *)

Out_0 := NOT Busy_1 OR NOT In_0;
```

```
(*****************************************
          Fim do Programa
*****************************************)
```

Resolução da Tarefa 7

A indisponibilidade dos elevadores em situação de alarme (isto é, quando In_15 = 0, já que a botoneira é normalmente fechada) reflete-se sobre o comando dos tapetes de saída como um sinal de entrada do tipo "Busy", no sentido em que os informa de que não devem movimentar paletes para além da posição dos seus sensores avançados.

Por outro lado, pretender-se que um tapete de saída não transporte mais do que uma palete em simultâneo, obriga o respetivo comando a gerar um sinal do tipo "Busy" a ser lido pelo controlador da mesa rotativa.

Assim, ao comando desenvolvido anteriormente para cada tapete de saída, tem de ser acrescentado o suporte ao sinal de alarme e a geração de um sinal Busy.

Por seu turno, a mesa rotativa passa a só poder descarregar se o tapete de destino estiver disponível. Tal introduz uma pequena alteração ao *grafcet* da Figura M1T6: as ações Atuador 2 e Atuador 3, que na etapa 4 da Figura M1T6 estavam apenas condicionadas ao sentido de descarga, passam a estar também condicionadas à disponibilidade do tapete de destino – ver Figura M1T7.

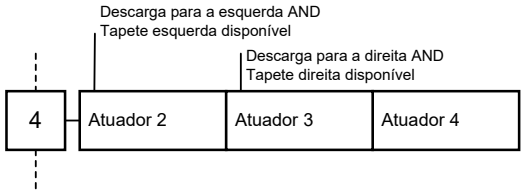


Figura M1T7 – *Alteração do comando da mesa rotativa relativamente à Tarefa 6*

O código correspondente à Tarefa 7 é então o seguinte:

```
(*****************************************
          Missão 1 - Tarefa 7
*****************************************)

(**** DETEÇÃO DE EVENTOS RELEVANTES ****)

F_In_0(CLK := In_0);
F_In_3(CLK := In_3);
F_In_7(CLK := In_7);
```

```
F_In_8(CLK := In_8);
F_In_9(CLK := In_9);
F_In_10(CLK := In_10);

(**** COMANDO DO TAPETE DE SAÍDA DA ESQUERDA ****)

IF F_In_10.Q THEN (* Quando uma palete abandona o tapete *)
    Mem_5 := FALSE; (* Reset de Mem_5 *)
    Busy_5 := FALSE; (* E fica disponível para carregar *)
END_IF;

IF F_In_8.Q THEN (* Quando é carregada uma palete *)
    Busy_5 := TRUE; (* Tapete fica indisponível *)
END_IF;

IF In_8 THEN (* Quando há palete para carregar *)
    Mem_5 := TRUE; (* Set de Mem_5 *)
END_IF;

(* Tapete de saída da esquerda ligado se tem uma palete e não há alarme e palete no
seu final *)
Out_5 := Mem_5 AND (NOT In_10 OR In_15); (* Notar que alarme corresponde a In_15 = 0
*)

(**** COMANDO DO TAPETE DE SAÍDA DA DIREITA ****)

IF F_In_9.Q THEN (* Quando uma palete abandona o tapete *)
    Mem_6 := FALSE; (* Reset de Mem_6 *)
    Busy_6 := FALSE; (* E fica disponível para carregar *)
END_IF;

IF F_In_7.Q THEN (* Quando é carregada uma palete *)
    Busy_6 := TRUE; (* Tapete fica indisponível *)
END_IF;

IF In_7 THEN (* Quando há palete para carregar *)
    Mem_6 := TRUE; (* Set de Mem_6 *)
END_IF;

(* Tapete de saída da direita ligado se tem uma palete e não há alarme e palete no
seu final *)
Out_6 := Mem_6 AND (NOT In_9 OR In_15); (* Notar que alarme corresponde a
In_15 = 0 *)

(**** COMANDO DA MESA ROTATIVA ****)

IF Livre AND In_3 THEN
```

```
    Livre := FALSE;
    Carrega := TRUE;
END_IF;

IF Carrega AND In_6 THEN
    Carrega := FALSE;
    Roda_carregada := TRUE;
END_IF;

IF Roda_carregada AND In_5 THEN
    Roda_carregada := FALSE;
    Descarrega := TRUE;
END_IF;

IF Descarrega AND (F_In_7.Q OR F_In_8.Q) THEN
    Descarrega := FALSE;
    Roda_descarregada := TRUE;
END_IF;

IF Roda_descarregada AND In_4 THEN
    Roda_descarregada := FALSE;
    Livre := TRUE;
END_IF;

IF F_In_3.Q THEN
    Busy_2 := TRUE;
END_IF;

IF Livre THEN
    Busy_2 := FALSE;
END_IF;

(* Definição das saídas em função das etapas, do sentido de descarga e da
disponibilidade dos tapetes de saída *)

Out_2 := Carrega OR Descarrega AND Sentido_descarga AND NOT Busy_5;
(* Descarga para a esquerda se tapete disponível *)
Out_3 := Descarrega AND NOT Sentido_descarga AND NOT Busy_6;
(* Descarga para a direita se tapete disponível *)
Out_4 := Roda_carregada OR Descarrega;

(**** COMANDO DO TAPETE DE ENTRADA ****)
(* Como na Tarefa 6 *)

IF F_In_0.Q THEN
    Count := ROL(Count, 1);
```

```
    IF (Count = WORD#16#2) THEN
        Busy_1 := TRUE;
    END_IF;
END_IF;

IF F_In_3.Q THEN
    Sentido_descarga := NOT Sentido_descarga;
    Busy_1 := FALSE;
    Count := ROR(Count, 1);

    IF (Count = WORD#16#8000) THEN
        Mem_1 := FALSE;
    END_IF;
END_IF;

IF In_0 THEN
    Mem_1 := TRUE;
END_IF;

Out_1 := Mem_1 AND (NOT Busy_2 OR NOT In_3);

(**** COMANDO DO ALIMENTADOR ****)
(* Como na Tarefa 2 *)

Out_0 := NOT Busy_1 OR NOT In_0;

(***************************************
            Fim do Programa
***************************************)
```

Resolução da Tarefa 8

Relativamente à Tarefa 7, o problema passa "apenas" por atualizar a variável Sentido_descarga, não alternadamente, mas sim em função da altura da palete que entra na mesa rotativa. Tudo o resto se mantém.

A informação sobre a altura de uma palete é dada pelo valor lógico de In_2 no instante em que esta é carregada no tapete de entrada, ou seja, quando In_0 tem uma transição descendente. Porém, sendo a informação sobre a altura de uma palete adquirida quando esta entra no tapete de entrada, mas só utilizada para definir Sentido_descarga quando a mesma é descarregada na mesa rotativa, tal informação, que corresponde a um simples *bit*, tem de ser devidamente armazenada.

Permitir o transporte simultâneo de várias paletes no tapete de entrada, obriga à existência de outros tantos *bits* para armazenamento das respetivas alturas. Gerir devidamente a utilização desses *bits* é o problema a resolver.

Uma forma muito simples de manter a informação relativa às alturas das paletes em trânsito no tapete de entrada devidamente atualizada e ordenada, é guardá-la num registo denominado "Fila" e proceder do seguinte modo:

Sempre que uma palete entra no tapete de entrada, o conteúdo de Fila é deslocado para a esquerda uma posição, colocando-se no *bit* mais à direita a informação sobre a altura da palete que chegou: 1 se é alta, 0 se é baixa. Tal procedimento pode ser conseguido com a instrução SHL(Fila, 1) – que desloca o registo Fila para a esquerda uma posição colocando um 0 no *bit* mais à direita – forçando depois o *bit* mais à direita a 1 se a caixa que entrou é alta.

O facto do contador Count ter sido materializado por um contador em anel – também deslocado para a esquerda (embora, na realidade, fosse rodado) sempre que uma caixa é carregada no tapete de entrada –, revela agora o seu verdadeiro interesse: para contagens diferentes de zero, a posição do 1 no registo Count corresponde à posição em que, no registo Fila, se encontra a informação da altura da caixa mais antiga no tapete de entrada. Consequentemente, o resultado da operação lógica "Count AND Fila" executada quando uma palete é descarregada, (mas antes de decrementar Count), revela a sua altura. Repare-se:

Tendo Count o seu *bit k* a 1 e todos os outros a zero, então da operação "Count AND Fila" resulta uma palavra que tem zeros em todos os *bits* diferentes de *k*, surgindo neste o valor lógico situado na posição *k* de Fila. Logo, se da operação "Count AND Fila" resultar um valor diferente de zero, a palete mais antiga é alta. Nesta operação lógica o valor de Count funciona como uma máscara que apenas deixa ver o *bit k* de Fila. Foi para conseguir esta máscara da forma mais simples possível que se utilizou um contador em anel!

A Figura M1T8 ilustra as ideias expostas para dois casos: no primeiro, há só uma palete no tapete e é alta. No segundo, há duas paletes e a mais antiga é baixa. Note-se que o produto lógico "Count AND Fila" resulta sempre num valor diferente de zero se a palete mais antiga é alta, independentemente da extensão da fila de paletes transportadas no tapete de entrada. Significa isto que esta estratégia vai continuar válida quando for removida a restrição do tapete de entrada não poder transportar mais de duas paletes em simultâneo.

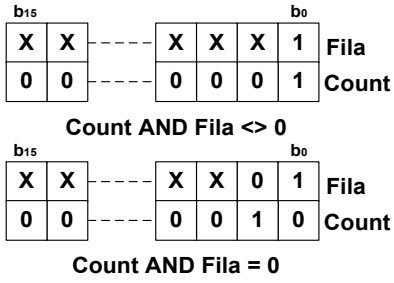


Figura M1T8 – *Dedução da altura da palete que abandona o tapete de entrada*

O programa correspondente à solução da Tarefa 8 tem então o seguinte código:

```
(***************************************
           Missão 1 - Tarefa 8
***************************************)

(**** DETEÇÃO DE EVENTOS RELEVANTES ****)

F_In_0(CLK := In_0);
F_In_3(CLK := In_3);
F_In_7(CLK := In_7);
F_In_8(CLK := In_8);
F_In_9(CLK := In_9);
F_In_10(CLK := In_10);

(**** COMANDO DO TAPETE DE SAÍDA DA ESQUERDA ****)
(* Como na Tarefa 7 *)

IF F_In_10.Q THEN
    Mem_5 := FALSE;
    Busy_5 := FALSE;
END_IF;

IF F_In_8.Q THEN
    Busy_5 := TRUE;
END_IF;

IF In_8 THEN
    Mem_5 := TRUE;
END_IF;

Out_5 := Mem_5 AND (NOT In_10 OR In_15);

(**** COMANDO DO TAPETE DE SAÍDA DA DIREITA ****)
(* Como na Tarefa 7 *)

IF F_In_9.Q THEN
    Mem_6 := FALSE;
    Busy_6 := FALSE;
END_IF;

IF F_In_7.Q THEN
    Busy_6 := TRUE;
END_IF;

IF In_7 THEN
    Mem_6 := TRUE;
```

```
END_IF;

Out_6 := Mem_6 AND (NOT In_9 OR In_15);

(**** COMANDO DA MESA ROTATIVA ****)
(* Como na Tarefa 7 *)

IF Livre AND In_3 THEN
    Livre := FALSE;
    Carrega := TRUE;
END_IF;

IF Carrega AND In_6 THEN
    Carrega := FALSE;
    Roda_carregada := TRUE;
END_IF;

IF Roda_carregada AND In_5 THEN
    Roda_carregada := FALSE;
    Descarrega := TRUE;
END_IF;

IF Descarrega AND (F_In_7.Q OR F_In_8.Q) THEN
    Descarrega := FALSE;
    Roda_descarregada := TRUE;
END_IF;

IF Roda_descarregada AND In_4 THEN
    Roda_descarregada := FALSE;
    Livre := TRUE;
END_IF;

IF F_In_3.Q THEN
    Busy_2 := TRUE;
END_IF;

IF Livre THEN
    Busy_2 := FALSE;
END_IF;

Out_2 := Carrega OR Descarrega AND Sentido_descarga AND NOT Busy_5;
Out_3 := Descarrega AND NOT Sentido_descarga AND NOT Busy_6;
Out_4 := Roda_carregada OR Descarrega;

(**** COMANDO DO TAPETE DE ENTRADA ****)

IF F_In_0.Q THEN (* Quando uma palete é carregada: *)
```

```
    Count := ROL(Count, 1); (* Incrementa contador *)

    IF (Count = WORD#16#2) THEN
        Busy_1 := TRUE;
    END_IF;

    Fila := SHL(Fila, 1); (* Ajusta a Fila *)
    (* Desloca uma posição para a esquerda introduzindo um 0 *)

    IF In_2 THEN
        Fila := Fila OR WORD#1;
        (* Força depois bit 0 a 1 se a palete que entrou é alta *)
    END_IF;
END_IF;

IF F_In_3.Q THEN (* Quando sai uma palete, verifica a sua altura *)
    Busy_1 := FALSE;

    IF (Count AND Fila) <> WORD#0 THEN
    (* Se a palete que sai é alta... *)
        Sentido_descarga := TRUE;
        (* ...será descarregada para a esquerda *)
    ELSE
        Sentido_descarga := FALSE;
        (* senão, para a direita *)
    END_IF;

    Count := ROR(Count, 1); (* O contador é agora decrementado *)

    IF (Count = WORD#16#8000) THEN
        Mem_1 := FALSE;
    END_IF;
END_IF;

IF In_0 THEN
    Mem_1 := TRUE;
END_IF;

Out_1 := Mem_1 AND (NOT Busy_2 OR NOT In_3);

(**** COMANDO DO ALIMENTADOR ****)
(* Como na Tarefa 2 *)

Out_0 := NOT Busy_1 OR NOT In_0;

(***************************************
            Fim do Programa
***************************************)
```

Resolução da Tarefa 9

Permitir que o tapete de entrada possa transportar tantas paletes quantas as possíveis, significa deixar de ter o sinal Busy_1 dependente do contador. Porém, este sinal não pode ser simplesmente retirado, dado que o tapete de entrada nem sempre está em condições de receber paletes do alimentador. Concretamente, não o pode fazer quando na sua posição mais avançada tem uma palete que a mesa rotativa não está em condições de receber, já que nessa situação tem necessariamente de suspender o seu movimento.

Dai que Busy_1 seja agora formulado da seguinte forma:

```
Busy_1 = Busy_2 AND In_3;
```

Esta pequena alteração é suposta responder aos objetivos da presente tarefa, melhorando bastante a eficiência da instalação. Porém, se o leitor a experimentar, verificará que duas paletes podem agora encostar-se no tapete de entrada, sendo percebidas como uma só pelo sensor 3 e resultando daí problemas sérios durante a rotação da mesa rotativa.

O problema acontece quando o tapete de entrada é forçado a suspender o seu movimento, ficando uma palete marginalmente para lá do sensor 0. Uma vez que o alimentador pode avançar uma palete até ao sensor 0, esta vai ficar muito próxima da que se encontra no tapete de entrada. Iniciando o alimentador e o tapete de entrada os seus movimentos em simultâneo, as paletes acabam por se encostar.

Este problema é parcialmente resolvido de uma forma muito simples: caso exista uma palete no início do tapete de entrada quando Busy_2 retorna a zero, então o retorno a zero do sinal Busy_1 será atrasado até que essa palete avance para lá da posição definida pelo sensor 1. Há, finalmente, uma utilidade para o sensor 1! Faça-se então o *set* de sinal Busy_1 para Busy_2 AND In_3 e o seu *reset* para NOT Busy_2 AND NOT In_1.

Mas, experimentando repetidamente esta solução, verifica-se que um outro problema subsiste: pode acontecer que o tapete de entrada fique indisponível durante a carga de uma palete, ficando esta numa posição em que já é detetada pelo sensor 1, mas ainda suficientemente apoiada no alimentador a ponto de requerer o movimento deste para abandonar a posição em que ficou parada. Neste cenário, a estratégia pensada não resulta: se o movimento do tapete de entrada desacompanhado do do alimentador não é suficiente para remover a palete da sua posição, então o sensor 1 não deixa de detetar a palete, pelo que o sinal Busy_1 não volta a zero, impedindo que o alimentador inicie o seu movimento.

A situação resolve-se recorrendo a um sinal de *timeout*; ou seja, se um dado tempo após a transição a zero de Busy_2, o sensor 1 continua a detetar uma palete, então será forçada a transição a zero do sinal Busy_1. O código que se apresenta de seguida mostra como isso pode ser feito. Foi utilizada uma temporização de 5 segundos para a geração do sinal de *timeout* que se verificou adequada ao fim em vista.

```
(***************************************
          Missão 1 - Tarefa 9
***************************************)

(**** DETEÇÃO DE EVENTOS RELEVANTES ****)

F_In_0(CLK := In_0);
F_In_3(CLK := In_3);
F_In_7(CLK := In_7);
F_In_8(CLK := In_8);
F_In_9(CLK := In_9);
F_In_10(CLK := In_10);

(**** COMANDO DO TAPETE DE SAÍDA DA ESQUERDA ****)
(* Como na Tarefa 7 *)

IF F_In_10.Q THEN
    Mem_5 := FALSE;
    Busy_5 := FALSE;
END_IF;

IF F_In_8.Q THEN
    Busy_5 := TRUE;
END_IF;

IF In_8 THEN
    Mem_5 := TRUE;
END_IF;

Out_5 := Mem_5 AND (NOT In_10 OR In_15);

(**** COMANDO DO TAPETE DE SAÍDA DA DIREITA ****)
(* Como na Tarefa 7 *)

IF F_In_9.Q THEN
    Mem_6 := FALSE;
    Busy_6 := FALSE;
END_IF;

IF F_In_7.Q THEN
    Busy_6 := TRUE;
END_IF;

IF In_7 THEN
    Mem_6 := TRUE;
```

```
END_IF;

Out_6 := Mem_6 AND (NOT In_9 OR In_15);

(**** COMANDO DA MESA ROTATIVA ****)
(* Como na Tarefa 7 *)

IF Livre AND In_3 THEN
    Livre := FALSE;
    Carrega := TRUE;
END_IF;

IF Carrega AND In_6 THEN
    Carrega := FALSE;
    Roda_carregada := TRUE;
END_IF;

IF Roda_carregada AND In_5 THEN
    Roda_carregada := FALSE;
    Descarrega := TRUE;
END_IF;

IF Descarrega AND (F_In_7.Q OR F_In_8.Q) THEN
    Descarrega := FALSE;
    Roda_descarregada := TRUE;
END_IF;

IF Roda_descarregada AND In_4 THEN
    Roda_descarregada := FALSE;
    Livre := TRUE;
END_IF;

IF F_In_3.Q THEN
    Busy_2 := TRUE;
END_IF;

IF Livre THEN
    Busy_2 := FALSE;
END_IF;

Out_2 := Carrega OR Descarrega AND Sentido_descarga AND NOT Busy_5;
Out_3 := Descarrega AND NOT Sentido_descarga AND NOT Busy_6;
Out_4 := Roda_carregada OR Descarrega;

(**** COMANDO DO TAPETE DE ENTRADA ****)

IF F_In_0.Q THEN
```

```
    Count := ROL(Count, 1);
    Fila := SHL(Fila, 1);

    IF In_2 THEN
        Fila := Fila OR WORD#1;
    END_IF;
END_IF;

IF F_In_3.Q THEN
    IF (Count AND Fila) <> WORD#0 THEN
        Sentido_descarga := TRUE;
    ELSE
        Sentido_descarga := FALSE;
    END_IF;

    Count := ROR(Count, 1);

    IF (Count = WORD#16#8000) THEN
        Mem_1 := FALSE;
    END_IF;
END_IF;

IF In_0 THEN
    Mem_1 := TRUE;
END_IF;

(* Geração do Sinal Busy_1 *)

IF Busy_2 AND In_3 THEN (* Se não pode movimentar paletes fica indisponível para
receber *)
    Busy_1 := TRUE; (* Set de Busy_1 *)
END_IF;

(* Se a mesa rotativa está disponível e o tapete de entrada não tem uma palete no
sensor 1, ou o Timer_1 chegou ao fim, então pode voltar a receber paletes do
alimentador *)

IF NOT Busy_2 AND NOT In_1 OR Timer_1.Q THEN
    Busy_1 := FALSE; (* Reset de Busy_1 *)
    Mem_timer := FALSE; (* Desliga o temporizador *)
END_IF;

(* Se não puder ficar disponível ao mesmo tempo que a mesa rotativa – por existir uma
palete no início do tapete de entrada... *)

IF NOT Busy_2 AND Busy_1 THEN
    Mem_timer := TRUE;
```

```
END_IF;

(* ...Lança um Timer de 5 segundos para limitar o atraso do movimento do alimentador
em relação ao do tapete de entrada *)

Timer_1(IN := Mem_timer, PT := T#5s);

IF In_0 THEN
    Mem_1 := TRUE;
END_IF;

Out_1 := Mem_1 AND (NOT Busy_2 OR NOT In_3);

(**** COMANDO DO ALIMENTADOR ****)
(* Como na Tarefa 2 *)
Out_0 := NOT Busy_1 OR NOT In_0;

(***************************************
            Fim do Programa
***************************************)
```

Resolução da Tarefa 10

A instalação passou a ter três estados (ou fases) possíveis, cuja evolução está representada no *grafcet* da Figura M1T10a:

- Automático – Funcionamento contínuo como na Tarefa 9;
- Encerramento – Alimentador não fornece paletes e os demais transportadores transferem as paletes ainda em trânsito;
- Pronto – A instalação não funciona por escassez de paletes.

O curioso deste funcionamento é que, excetuando o alimentador, todos os transportadores, em todos dos modos, se comportam da forma descrita na Tarefa 9! É por simples "escassez de paletes" – ou seja, pelo facto do alimentador deixar de fornecer paletes – que a instalação entra na fase de encerramento, acabando depois inativa. Fazer a instalação regressar ao modo automático, é permitir que o alimentador volte a fornecer paletes que os diferentes transportadores encaminham devidamente.

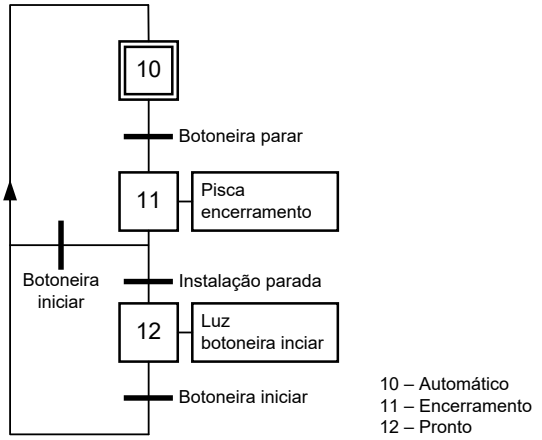


Figura M1T10a – *Evolução e sinalização dos modos de funcionamento da instalação*

Assim, o código do comando do alimentador – que permaneceu imutável desde a Tarefa 2 – é o único que requer alteração em relação à tarefa anterior. Considerando que, caso exista uma palete na zona do sensor 0 no momento em que é dada ordem de encerramento, essa palete é ainda enviada para o tapete de entrada, o comando do alimentador passa a ser dado por:

```
Out_0 = (NOT In_0 OR NOT Busy_1) AND (In_0 OR NOT Encerramento) AND NOT
Pronto;
```

Ou, equivalentemente:

```
Out_0 = (NOT In_0 OR NOT Busy_1) AND (In_0 OR Automatico);
```

As variáveis “Automático”, “Encerramento” e “Pronto” são variáveis internas que tomam o valor lógico 1 quando a instalação está no respetivo modo de funcionamento, permitindo programar facilmente o *grafcet* da Figura M1T10a. Reconhecer que a instalação está parada é reconhecer que nenhum transportador contém paletes:

```
Parada = NOT Mem_1 AND Livre AND NOT Mem_5 AND NOT Mem_6;
```

Para sinalizar o encerramento da instalação é necessário realizar um pisca com as características temporais desejadas que só funcione quando a variável Encerramento exibe o valor lógico 1.

A forma mais simples de realizar um pisca é através de dois temporizadores *on-delay* com a configuração mostrada na Figura M1T10b:

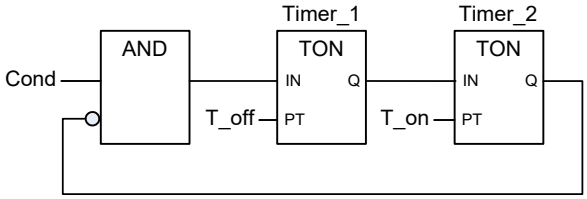


Figura M1T10b – *Pisca construído com dois temporizadores on-delay*

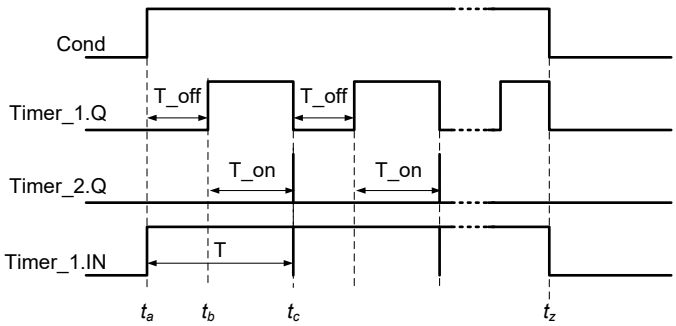


Figura M1T10c – *Diagrama temporal para o pisca da Figura M1T10b*

A compressão deste esquema passa pela compreensão do diagrama temporal representado na Figura M1T10c:

Considere-se que, inicialmente, a condição de interesse genérica, Cond, que deverá fazer funcionar o pisca, está a 0. Isso faz com que as entradas e saídas dos temporizadores estejam também em 0.

O primeiro temporizador, Timer_1, é armado ao tempo *ta*, ou seja, quando a condição Cond – no caso desta tarefa será a variável "Encerramento" – toma o valor lógico 1. Mantendo-se Cond a 1, a saída de Timer_1 vai a 1 ao tempo *tb* = *ta* + T_off, fazendo Timer_2 iniciar a respetiva temporização. Consequentemente, mantendo-se Cond a 1, a saída do segundo temporizador vai a 1 ao tempo *tc* = *tb* + T_on, ou seja, um tempo T_on após o primeiro, desarmando-o. O desarme do primeiro temporizador desarma o segundo que, por sua vez, rearma o primeiro. Significa isto que, ao tempo *tc* retomam-se as condições observadas em *ta*, pelo que o ciclo se repetirá com uma periodicidade T = T_on + T_off.

Assim, a saída do primeiro temporizador alternará com um período T = T_off + T_on e um *duty cycle* = T_on/T até ao tempo *tz*, instante em que a variável de interesse, Cond, passa a exibir o valor lógico 0.

Assumindo-se que, no caso da presente tarefa, o pisca será realizado por dois temporizadores designados por Pisca_encerramento_1 e Pisca_encerramento_2, então a sinalização do estado da instalação através da luz da botoneira Iniciar é feita pela seguinte instrução:

```
Out_8 = Parada OR Pisca_encerramento_1.Q;
```

O programa correspondente à Tarefa 10 é o seguinte:

```
(****************************************
           Missão 1 - Tarefa 10
****************************************)

(**** DETEÇÃO DE EVENTOS RELEVANTES ****)

F_In_0(CLK := In_0);
F_In_3(CLK := In_3);
F_In_7(CLK := In_7);
F_In_8(CLK := In_8);
F_In_9(CLK := In_9);
F_In_10(CLK := In_10);

(**** EVOLUÇÃO E SINALIZAÇÃO DOS MODOS DE FUNCIONAMENTO ****)
(* Evolução das varáveis de estado *)

IF Automatico AND NOT In_13 THEN (* Automático -> Encerramento *)
    Encerramento := TRUE;
    Automatico := FALSE;
END_IF;

IF Encerramento AND Parada THEN (* Encerramento -> Pronto *)
    Pronto := TRUE;
    Encerramento := FALSE;
END_IF;

IF In_12 THEN (* Quando a Botoneira Iniciar é premida *)
    Automatico := TRUE; (* Encerramento ou Pronto -> Automático *)
    Encerramento := FALSE;
    Pronto := FALSE;
END_IF;

(* Deteção de instalação inativa *)
Parada := NOT Mem_1 AND Livre AND NOT Mem_5 AND NOT Mem_6;

(* Materialização do Pisca de Encerramento *)
Pisca_encerramento_1(IN := Encerramento AND NOT Pisca_encerramento_2.Q, PT := T#1s);
Pisca_encerramento_2(IN := Pisca_encerramento_1.Q, PT := T#1s);

(* Sinalização do Modo *)
```

```
Out_8 := Pronto OR Pisca_encerramento_1.Q;

(**** COMANDO DO TAPETE DE SAÍDA DA ESQUERDA ****)
(* Como na Tarefa 7 *)

IF F_In_10.Q THEN
    Mem_5 := FALSE;
    Busy_5 := FALSE;
END_IF;

IF F_In_8.Q THEN
    Busy_5 := TRUE;
END_IF;

IF In_8 THEN
    Mem_5 := TRUE;
END_IF;

Out_5 := Mem_5 AND (NOT In_10 OR In_15);

(**** COMANDO DO TAPETE DE SAÍDA DA DIREITA ****)
(* Como na Tarefa 7 *)

IF F_In_9.Q THEN
    Mem_6 := FALSE;
    Busy_6 := FALSE;
END_IF;

IF F_In_7.Q THEN
    Busy_6 := TRUE;
END_IF;

IF In_7 THEN
    Mem_6 := TRUE;
END_IF;

Out_6 := Mem_6 AND (NOT In_9 OR In_15);

(**** COMANDO DA MESA ROTATIVA ****)
(* Como na Tarefa 7 *)

IF Livre AND In_3 THEN
    Livre := FALSE;
    Carrega := TRUE;
END_IF;

IF Carrega AND In_6 THEN
```