UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Métodos para Caracterização de Desempenho de CPUs Industriais

por

ANDRÉ COSTI NÁCUL

Dissertação submetida a avaliação,
como requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. João Cesar Netto
Orientador

Porto Alegre, fevereiro de 2002

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Nácul, André Costi

Métodos para Caracterização de Desempenho de CPUs Industriais / por André Costi Nácul. — Porto Alegre: PPGC da UFRGS, 2002.

98 f.: il.

Dissertação (mestrado) — Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2002. Orientador: Netto, João Cesar.

1. Tempo Real. 2. CPU Industrial. 3. Avaliação de Desempenho. I. Netto, João Cesar. II. Título.

"If we knew what it was we were doing,
it would not be called research, would it?"
Albert Einstein

Agradecimentos

O primeiro agradecimento deve, obrigatoriamente, ser direcionado aos meus pais e meu irmão, que compreenderam os meus períodos de mau humor, os tempos que não estive junto com eles e os finais de semana que passei trabalhando em Porto Alegre enquanto eles estavam curtindo a nossa casa na praia. Eles estiveram sempre comigo e me deram todo o apoio que precisei para conseguir concluir o curso e, principalmente, esta dissertação. Devo a eles todas as chances que tenho de continuar estudando livre de maiores preocupações.

À Mirella, que me amou todos os dias, e me deu uns puxões de orelha quando eu estava meio devagar. Acho que nos completamos muito bem.

Ao professor João Netto, que me aguenta há quase 5 anos como orientando, desde os tempos de graduação e iniciação científica. Sempre recebi grandes oportunidades, conselhos e, principalmente, trabalho! Se tenho um bom currículo hoje, devo boa parte a ele.

Aos demais professores do pós graduação, que acreditam na Universidade Federal, e dedicam seu tempo para passar os conhecimentos que adquiriram aos alunos do curso. Em especial ao Luigi e ao Lubaszewski, que também me deram dicas valiosas, conselhos e ouvidos quando precisei discutir detalhes a respeito do meu futuro profissional quando eu terminasse esta etapa, e que ainda ajudaram a incrementar meu currículo com mais algumas publicações importantes, mesmo que eu não fosse seu aluno sob orientação direta.

Aos colegas do curso, que compartilharam as angústias, dúvidas, felicidades, frustrações e trabalhos dos últimos 2 (alguns mais) anos. Sei que corro o risco de esquecer alguém, mas obrigado ao Lisandro, Pilla, Kenzo, Lucc, Goiano, Cláudio.

Aos funcionários do Instituto de Informática, que ajudam a tornar a tarefa de ser aluno da pós graduação mais divertida e menos difícil.

Ao Instituto de Informática, lugar onde eu passei mais de 60% do meu tempo nos últimos 7 anos, que me acolheu desde a graduação em 95 até hoje. Me orgulho de dizer que faço parte desta instituição. Espero poder retribuir tudo que recebi daqui a alguns anos.

Ao pessoal da Altus, Gerbase, Rudy, Leonel, Rosana, Bordini e Brune, pela disposição em emprestar equipamentos, pelas dicas e pelo bom papo.

Por fim, um agradecimento aos órgãos financiadores desta pesquisa: ao CNPq, que me sustentou por dois anos com a bolsa de mestrado registrada sob número 131274/2000-7, à FINEP que financiou o projeto CT-Petro número 65.00.036300, do qual esta dissertação fez parte, e à FAPERGS, por apoio em projetos anteriores.

Sumário

Lista de Abreviaturas

CAN	*Controller Area Network*
CLP	Controlador Lógico Programável
COTS	*Commercial Off-The-Shelf Components*
CPGCC	Curso de Pós Graduação em Ciência da Computação
CPU	*Central Processing Unit*
EEPROM	*Electrically Eraseable Programmable Read-Only Memory*
I/O	*Input Output*
IEC	*International Electrotechnical Commission*
IEEE	*Institute of Electric and Electronic Engineers*
IETF	*Internet Engeneering Task Force*
IP	*Internet Protocol*
PC	*Personal Computer*
PPGC	Programa de Pós Graduação em Computação
Profibus	*Process Field Bus*
RAM	*Random Access Memory*
RFC	*Request For Comments*
TCP	*Transport Control Protocol*
UCP	Unidade Central de Processamento
UFRGS	Universidade Federal do Rio Grande do Sul

Lista de Figuras

Lista de Tabelas

Resumo

A caracterização de desempenho é uma atividade fundamental na área de controle industrial. Por se tratar, na maior parte das vezes, de aplicações de tempo real, a caracterização de desempenho torna-se ainda mais necessária e importante. Entretanto, atualmente não há nenhuma metodologia estabelecida para realizar esta caracterização. Não há nem mesmo um conjunto de parâmetros que devem ser avaliados em um equipamento de controle utilizado em processos industriais.

Para tentar suprir esta carência, este trabalho apresenta uma proposta de métricas e *workloads* para serem utilizados na avaliação de desempenho de sistemas de controle baseados em CLPs e CPUs Industriais. O processo de avaliação de desempenho é discutido em todas as etapas, desde o estudo da aplicação até a execução dos passos de caracterização de desempenho. Para ilustrar a aplicação das métricas, técnicas e procedimentos propostos, são avaliadas três CPUs Industriais, e os resultados são apresentados ao final do trabalho. Espera-se assim estar contribuindo para o estabelecimento de uma metodologia padronizada para avaliação de desempenho de equipamentos de controle industrial.

Palavras-chave: Tempo Real, CPU Industrial, Avaliação de Desempenho.

TITLE: "METHODS FOR PERFORMANCE PROFILING OF INDUSTRIAL CPUS"

Abstract

The performance profiling process plays a key role in the field of industrial control. Most of the times, these are real time applications, and then performance characterization is even more necessary and important. However, there is no current established methodology for performing this task. Not even a set of performance parameters that must be accounted for in an industrial control equipment is available.

In order to supply this demand, this work presents a set of metrics and workloads that can be used in performance analysis and profiling of industrial control systems based on Industrial CPUs and PLCs. The process of performance analysis is discussed as a whole, from the study of the application to the steps that must be executed in order to obtain performance data. To show the application of the metrics, techniques and procedures proposed here, three Industrial CPUs are characterized, and the results are reported by the end of this work. We expect to contribute to the establishment of a standard methodology for performance analysis of industrial control devices.

Keywords: Real-Time, Industrial CPU, Performance Analysis.

1 Introdução

Computação de Tempo Real é uma tecnologia presente em importantes áreas de aplicação, como aviação, controle industrial, usinas nucleares, telecomunicações e medicina, entre muitas outras. Grande parte desses sistemas é crítica, e a falha de um equipamento pode levar a conseqüências catastróficas [STA 96], [STA 96a].

Sistema de Tempo Real é caracterizado na literatura [BUR 97], [KRI 97], [OLI 2000],[STA 88] como sendo um sistema no qual o instante de tempo em que a resposta é apresentada é tão importante quanto o valor da resposta. Assim, para uma computação estar correta, não basta apenas que seu valor esteja logicamente certo, mas ele deve ser apresentado num instante de tempo determinado. Alternativamente, um sistema de tempo real pode ser definido como sendo um sistema de processamento de informações que deve responder a eventos de entrada externos em um intervalo de tempo finito e determinado. Nesse tipo de sistema, a falta de uma resposta pode ser tão grave quanto uma resposta incorreta.

Ao contrário dos sistemas tradicionais, onde um aumento de velocidade implica em um aumento de desempenho geral, no sistema de tempo real não existe preocupação com a velocidade de computação, e sim com as garantias de que os requisitos temporais de cada tarefa serão cumpridos. "Melhor do que ser veloz é ser previsível" [STA 88].

Entre as diversas áreas de aplicação de sistemas de tempo real, o controle industrial é de especial interesse neste trabalho. O controle de processos industriais é automatizado há bastante tempo com o uso dos controladores lógico programáveis, ou CPUs Industriais, que nada mais são que arquiteturas de processamento especialmente projetadas para suportar as condições de funcionamento encontradas em ambientes industriais, muito ruidosos, com grandes variações de temperatura, umidade e voltagem, onde falhas são catastróficas e a necessidade de precisão e garantias de funcionamento são fundamentais para o sucesso do processo controlado [BOU 96]. As CPUs Industriais interagem com o ambiente controlado através de sensores e atuadores, e o controle do processo industrial é especificado através de linguagens desenvolvidas para esse fim, como a lógica de relés e o diagrama de blocos de função [IEC 93].

Os primeiros equipamentos de controle industrial foram soluções proprietárias desenvolvidas nas décadas de 70 e 80. Porém, o uso de sistemas proprietários encarece a produção, dificulta a manutenção e impede a integração de tecnologias distintas. Nos anos 90, a necessidade de plataformas abertas, tanto para equipamentos de *hardware*, quanto para sistemas operacionais, linguagens de especificação de controle e redes de comunicação motivou a publicação de [CHR 2000].

As aplicações de controle industrial típicos, com o uso de CPUs Industriais, se encaixam na definição de sistema de tempo real. Elas possuem uma iteração com um ambiente físico e as respostas de estímulos gerados pelo ambiente devem ser apresentadas em um intervalo de tempo determinado, que tem relação com o tempo real decorrido no mundo físico. Diversas classificações são possíveis dentro dos sistemas de tempo real, e existe uma vasta literatura sobre o assunto. Com relação às tarefas do sistema, a aplicação pode ser considerada estática ou dinâmica. Um sistema de tempo real estático é aquele onde todas as tarefas já são conhecidas no início da execução do mesmo, e o conjunto de tarefas não muda ao longo do

tempo. Já o sistema de tempo real dinâmico envolve a admissão ou ativação de tarefas novas ao longo da execução da aplicação, e estas nem sempre são conhecidas no início da execução.

Outra classificação dos sistemas de tempo real está relacionada às características da execução das tarefas e aos instantes de tempo máximo para apresentação da resposta, ou *deadlines*, e a conseqüência do não cumprimento de algum *deadline*. Um sistema de tempo real pode ser do tipo determinístico, onde todas as tarefas são executadas sempre na mesma ordem e no mesmo instante de tempo relativo ao início da execução. Neste caso, é sempre possível concluir qual tarefa está sendo executada em cada instante de tempo. Ele pode ainda ser do tipo baseado em *deadlines*, onde a ordem de execução das tarefas não é fator importante, mas sim que todas elas apresentem a resposta esperada dentro do intervalo de tempo máximo determinado para cada uma. Por fim, o sistema pode ser mais flexível e aceitar um comportamento estatístico das tarefas, onde algumas vezes os instantes de resposta são perdidos, porém isso ocorre com uma freqüência conhecida e aceitável.

Finalmente, os sistemas de tempo real com *deadlines* permitem mais uma classificação, que envolve a gravidade da perda de um *deadline*. Neste caso, os sistemas são divididos em sistemas de tempo real rígidos, ou *hards*, sistemas de tempo real firmes, ou *firms*, e os sistemas de tempo real flexíveis, ou *soft*. O sistema de tempo real rígido é aquele que não aceita a perda de nenhum *deadline*, já que a ocorrência de um evento deste tipo pode ter conseqüências catastróficas, como uma explosão numa indústria química, por exemplo. Tipicamente, sistemas de controle industrial são sistemas de tempo real rígidos. Os sistemas de tempo real firmes são os que envolvem dois *deadlines*. O primeiro deadline é o instante desejado para apresentação da resposta, quando esta terá grande valor para o sistema. Se este deadline for perdido, não ocorrerá nenhuma catástrofe, porém a resposta passa a ter um valor mais baixo, até que após o segundo *deadline* ela não é mais útil ao sistema. O último sistema, o sistema de tempo real flexível, admite a perda de *deadlines*, sem que nenhuma conseqüência grave ocorra. Geralmente, a perda do deadline implica em perda da qualidade da solução, mas nem por isso ela será desprezada. Um exemplo típico de sistema flexível é uma vídeo-conferência, onde a perda do momento de apresentação de alguns *frames* da imagem torna a conferência de pior qualidade, mas ainda é útil para permitir a comunicação entre os envolvidos.

A previsibilidade do sistema de tempo real é um dos principais tópicos de pesquisa na área [STA 96], uma vez que deve-se garantir que os requisitos temporais de uma aplicação serão cumpridos, principalmente quando se tratam de aplicações de tempo real rígidas. Talvez não seja possível prever o instante exato da resposta, mas é necessário garantir que a tarefa estará concluída dentro dos limites temporais quando consideradas as condições mais pessimistas do ambiente onde o sistema está inserido. Em sistemas de tempo real estáticos, estudos sobre garantias de desempenho já foram desenvolvidos e estão estabelecidos. Porém, ainda segundo Stankovic [STA 96], "novas pesquisas sobre validação de desempenho são necessárias devido às limitações do estado da arte atual, que faz com que ele seja inadequado para sistemas de tempo real modernos, de grande porte e baseados em sistemas abertos, compostos por computadores, redes e sistemas operacionais".

Com o uso cada vez mais constante de componentes de prateleira (referenciados na literatura por COTS – commercial off-the-shelf components) na construção de arquiteturas computacionais [HAR 2000], [SED 2001], a necessidade do desenvol-

vimento de novas técnicas para avaliação de desempenho destes sistemas é reforçada. Mais do que isso, é preciso que sejam desenvolvidas técnicas que não envolvam instrumentação no código fonte do sistema operacional ou da aplicação, uma vez que os COTS são normalmente caixas pretas prontas para serem integradas aos produtos, onde o usuário tem pouco ou nenhum controle e conhecimento sobre os detalhes internos de implementação dos componentes.

Com o auxílio da Internet, todas as soluções se disseminam rapidamente, e o tempo gasto para construir um sistema completo é um ônus alto. Enquanto diminui o *time-to-market*, a construção de sistemas baseados em COTS aumenta a dependência em relação a outros revendedores e dificulta a avaliação da real capacidade e desempenho da arquitetura final [SED 2001]. Ao mesmo tempo, é difícil decidir entre dois ou mais fabricantes que oferecem produtos similares, pois os dados de desempenho que são apresentados aos usuários e que poderiam servir de parâmetro para uma avaliação foram obtidos (i) de maneira não padronizada e (ii) tentando ressaltar os pontos favoráveis de cada produto. Estas, por sinal, são as falhas mais comuns em processos de levantamento de desempenho [JAI 91].

Em sistemas de controle industrial, a tendência não é diferente. A utilização de subsistemas prontos para serem integrados é crescente, porém a dificuldade de avaliação de desempenho do produto final é um fator importante. No controle industrial, as aplicações são críticas e devem atender a requisitos temporais muitas vezes rígidos. Portanto, garantias de desempenho devem ser obtidas, através de técnicas analíticas, simulação ou medição direta.

Porém, a área de sistemas tempo real é muito ampla para que seja possível estabelecer um conjunto de parâmetros válidos para qualquer aplicação. A avaliação de desempenho é uma arte, e como todo trabalho de arte, não obtém sucesso caso seja desenvolvido mecanicamente. Para cada caso, deve ser elaborado um estudo individual que envolve as particularidades da aplicação [JAI 91].

Por mais surpreendente que possa parecer, ainda há uma ausência de padrões de comparação e garantias de desempenho quando trata-se de CPUs Industriais. Cada fabricante de equipamento utiliza suas próprias métricas, e as obtém a partir de avaliações não muito divulgadas. Com isso, é difícil comparar dois ou mais equipamentos no momento da especificação de um sistema de controle baseado em CPUs Industriais e controladores programáveis. Os resultados reportados pelos fabricantes tornam-se muito pouco úteis, já que não é possível determinar as condições sob as quais foram obtidos os resultados que são reportados. Novamente verifica-se a afirmação de Stankovic [STA 96] de que estudos a respeito de avaliação e garantias de desempenho ainda são tópicos carentes em sistemas de tempo real.

É neste contexto que se insere o trabalho proposto. A dissertação apresenta duas contribuições na área. A primeira delas é a construção de um conjunto de métricas que possam ser utilizadas para caracterizar o desempenho de uma CPU Industrial. A segunda contribuição é a determinação de métodos e cargas de teste para extração destas métricas nas diversas CPUs. Como forma de validação das métricas e das técnicas e programas de carga, será caracterizado o desempenho de um conjunto de CPUs, a título de estudo de caso.

Este trabalho está organizado da seguinte forma. O capítulo 2 apresenta o tópico de avaliação de desempenho, definindo a terminologia básica que será utilizada ao longo da dissertação, como forma de padronização de nomes e conceitos. O capítulo 3 aborda as CPUs Industriais, com discussões a respeito de seus componentes,

utilização e implementação. A seguir, o capítulo 4 detalha as métricas construídas e propostas neste estudo como forma de caracterização de desempenho de uma CPU Industrial. O capítulo 5 discute os *workloads* e instrumentação necessárias para a extração dos resultados da avaliação. O capítulo 6 apresenta os resultados da aplicação destes *workloads* para extração das métricas de um conjunto de CPUs Industriais, mostrando que as métricas e *workloads* propostos neste trabalho podem ser aplicados com sucesso na caracterização de CPUs Industriais. Por fim, o capítulo 7 apresenta as conclusões finais deste trabalho e os trabalhos futuros propostos a partir deste.

2 Avaliação de Desempenho

Um dos objetivos de um processo de avaliação de desempenho é ou comparar as características de dois (ou mais) sistemas ou verificar a influência da modificação de algum parâmetro ou componente no desempenho final de um determinado equipamento. O processo de avaliação de desempenho pode ser de custo relativamente baixo, e é uma alternativa viável quando se deseja verificar o impacto de uma modificação no sistema ou comparar alternativas de implementação sem que necessariamente um protótipo ou produto final esteja disponível. O conhecimento dos limites de um equipamento também pode ser listado como outra aplicação de um estudo de desempenho, desta feita com o produto final já disponível para que sua caracterização detalhada possa ser executada.

O processo de avaliação de desempenho nunca é exato. Ele é incapaz de especificar com precisão exata qual será o comportamento de um sistema. Qualquer que seja a técnica empregada, os resultados obtidos serão uma aproximação, com maior ou menor grau de fidelidade, do comportamento do sistema em seu ambiente e aplicação regular. Isso se deve às aproximações, simplificações e abstrações assumidas quando do processo de modelagem do sistema. Também a interferência dos equipamentos de medição, no caso da técnica de medição direta, influi na precisão dos resultados. Por fim, as condições a que o sistema será submetido durante a sua utilização regular dificilmente poderá ser reproduzido com alto grau de fidelidade. Todos esses fatores levam a resultados aproximados no processo de avaliação, mas nem por isso descartáveis. Caso a influência das técnicas de medição possa ser quantificada ou conhecida, os resultados são representativos do processo avaliado.

Este capítulo destina-se a apresentar os principais elementos envolvidos num processo de avaliação de desempenho, tais como métricas, *workloads* e técnicas para extração de parâmetros de um sistema, bem como a nomenclatura e classificações mais comumente utilizadas nos estudos desta área. O objetivo não é desenvolver uma abordagem extensa e detalhada, mas sim uma breve introdução necessária para o desenvolvimento e compreensão dos termos e decisões tomadas ao longo deste trabalho. Para leituras mais detalhadas a respeito de avaliação de desempenho e exemplos práticos, é possível encontrar vasta literatura disponível, como em Jain [JAI 91], Morris [MOR 82], Ferrari [FER 83] e Gunther [GUN 98].

2.1 Avaliação de Desempenho Passo a Passo

Processos de avaliação de desempenho são únicos e exigem que estudos de métricas e *workloads* sejam feitos caso a caso. Porém, as etapas a serem cumpridas no processo de avaliação são sempre as mesmas e devem ser seguidas cuidadosamente para evitar distorções e erros causados por problemas na condução do processo. Uma avaliação de desempenho deve seguir os seguintes passos, descritos por Jain [JAI 91]:

1. Estabelecimento dos objetivos da avaliação e definição das fronteiras do estudo. Uma das falhas mais comuns numa avaliação de desempenho é a falta de definição de objetivos, o que acaba levando a resultados inúteis. Uma clara definição dos objetivos do estudo ajuda a identificar métricas, *workloads* e métodos de avaliação.

2. Listagem dos serviços fornecidos pelo sistema avaliado e comportamento esperado. Não é possível avaliar o desempenho de um sistema desconhecido, pois falhas certamente aparecerão nas etapas seguintes de modelagem e projeto dos experimentos caso o sistema não esteja bem entendido e as respostas esperadas em cada caso não estejam listadas.

3. Seleção das métricas de desempenho. Esta é uma das etapas mais importantes. Métricas mal selecionadas levam a resultados sem significados práticos. Métricas em uma CPU Industrial podem ser o tempo de resposta a mudanças no estado do processo controlado ou tempo de execução de uma instrução do programa de controle especificado.

4. Listagem dos parâmetros do sistema e das situações de carga. Aqui devem ser listados os parâmetros que afetam o desempenho do sistema, bem como as situações a que o sistema é submetido. Entre parâmetros significativos, é possível listar a velocidade da CPU, largura do barramento de dados ou distâncias envolvidas na comunicação com os dispositivos sensores e atuadores.

5. Seleção dos fatores a serem variados nos experimentos. Os parâmetros a serem variados são os fatores, e os valores que eles assumem são chamados níveis.

6. Seleção das técnicas de avaliação e do *workload*. As técnicas mais comuns são métodos analíticos, simulação e experimentação.

7. Projeto dos experimentos. Esta etapa envolve a decisão do conjunto de experimentos que oferecem o máximo de resultados com o menor esforço. Pode-se dividir os experimentos em duas etapas. Na primeira delas, diversos fatores são variados em poucos níveis, para determinar quais deles influenciam mais significativamente nos resultados. Em seguida, o número de fatores é reduzido e a quantidade de níveis aumentada para avaliação do impacto nos resultados finais.

8. Análise e interpretação dos dados obtidos. Os resultados dos experimentos são diferentes a cada vez que uma rodada de testes é executada, e a variação pode ser significativa. Técnicas estatísticas devem ser aplicadas para obtenção dos resultados. Estes devem ser interpretados para que deles sejam tiradas conclusões significativas.

9. Apresentação dos resultados

2.2 Métricas e Técnicas de Avaliação

Num trabalho de avaliação de desempenho, deve ser escolhido um conjunto de critérios ou métricas para caracterização de um sistema ou comparação de desempenho entre dois ou mais equipamentos. As métricas representam os parâmetros significativos no processo de avaliação de desempenho, e dependem do sistema sendo avaliado e dos objetivos do estudo de desempenho. Conforme o estudo de desempenho, um conjunto diferente de métricas pode ser utilizado para um mesmo sistema.

A construção do conjunto de métricas deve ser feita com cuidado, de forma a contemplar todos os pontos que se deseja avaliar e ao mesmo tempo evitar a redundância entre as métricas. Ou seja, evitar que o mesmo parâmetro seja medido mais de uma vez por métricas distintas. Métricas redundantes implicam em mais trabalho e tempo de execução do processo de avaliação sem que algum resultado extra possa ser obtido.

Dependendo da utilização ou do que representa cada métrica, ela pode ser classificada em três diferentes conjuntos. A métrica pode ser do tipo "maior melhor", quando a métrica representa uma medida que um valor mais alto reflete um desempenho mais positivo. Por exemplo, uma métrica de vazão de dados por segundo em uma rede de comunicação para aplicações corporativas é do tipo "maior melhor". A métrica pode também ser classificada como "menor melhor", que significa que um valor mais baixo para a métrica representa um desempenho mais qualificado. Por exemplo, o tempo de espera do usuário pela apresentação da saída de um programa é do tipo "menor melhor", pois quanto menos tempo o usuário ficar esperando pela resposta, melhor é o desempenho do sistema. Por fim, as métricas ainda podem ser classificadas como "nominal melhor", quando um valor determinado é mais representativo de um melhor desempenho, e não é desejado nem um valor mais alto e nem um valor mais baixo. Como exemplo, pode-se citar a taxa de utilização de um sistema. Se a taxa de utilização é muito alta, significa que o sistema está sobrecarregado. Uma taxa de utilização muito baixa significa que o sistema está sendo sub utilizado, e que está super dimensionado para a aplicação atual. O melhor é que a taxa de utilização esteja entre 50% e 75%, quando não há sobrecarga e nem sub utilização.

A escolha da técnica a ser utilizada na avaliação também é uma etapa importante. Existem três técnicas de avaliação possíveis: modelagem analítica, simulação e medição direta. A técnica de medição direta exige que o sistema a ser avaliado esteja disponível ao menos como um protótipo, caso contrário não é possível medi-lo diretamente. As outras duas técnicas não dependem diretamente da existência do sistema real. Elas baseiam-se em modelagens em níveis diferentes de abstração do sistema a ser avaliado, e portanto podem ser executadas antes que um protótipo esteja disponível. Por outro lado, exatamente por representarem abstrações da realidade, simplificações são inseridas nos modelos, e então a precisão da avaliação pode ser em parte prejudicada. Tudo depende da fidelidade do modelo em relação às métricas que se deseja medir. O modelo não precisa ser fiel à realidade em todas as suas características, mas sim fiel em relação aos pontos que afetam as métricas do estudo. Muitas vezes, a fidelidade é difícil de ser obtida, e em outras vezes o processo de modelagem é bastante simples.

A técnica de medição direta é a que pode apresentar os resultados mais próximos do real, uma vez que avalia o sistema através de observações diretamente no sistema real em funcionamento. Ela exige o uso de monitores para extrair dados para o estudo, e tem um custo geralmente elevado quando comparado com as outras técnicas de avaliação. Exige a existência física do sistema a ser avaliado, e por esse motivo nem sempre pode ser aplicada. Novamente, a fidelidade dos resultados depende das métricas do estudo. Porém, aqui não se aplica a qualidade do modelo desenvolvido. O que importa é a precisão e a intrusão das ferramentas de monitoração das métricas, e a capacidade destes monitores de reportarem resultados com uma determinada precisão ou aproximação.

Avaliação analítica faz uma modelagem do sistema a ser avaliado, utilizando técnicas como teoria de filas, equações matemáticas ou redes de Petri, entre outros. É a técnica mais rápida e barata, porém exige que os analistas possuam um domínio matemático para poder modelar o sistema. É de baixo custo e não interage com o sistema alvo do estudo diretamente. Ao mesmo tempo, apresenta resultados menos precisos que a medição direta, já que é baseada numa abstração do sistema (modelo), que pode conter simplificações para facilitar a análise.

Por fim, a simulação também é uma técnica baseada em modelagem, e não interage diretamente com o sistema sendo avaliado. Apresenta um qualidade de resultados superior à técnica analítica, pois considera um modelo mais apurado do sistema, onde é possível especificar mais detalhes dentro do modelo. Ao mesmo tempo consome mais tempo no processo de avaliação, principalmente devido ao tempo de construção do modelo, que deve ser especificado em uma linguagem de simulação.

2.3 *Workloads*

Um *workload* de teste é uma carga a que é submetido um sistema durante o processo de avaliação de desempenho. Um *workload* pode ser real ou sintético. O *workload* real é aquele baseado na observação da execução do sistema em situações normais de funcionamento. Por isso mesmo, é difícil de ser repetido e não muito adequado para situações de teste de desempenho. Já o *workload* sintético é construído com características similares a um *workload* real, a partir de observações a respeito do mesmo, porém pode ser controlado e repetido diversas vezes. Portanto é mais adequando para estudos de desempenho. Os *workloads* sintéticos são em geral pequenos e fáceis de serem portados, e ainda facilmente modificáveis para inclusão ou remoção de recursos sem que sua funcionalidade seja significativamente alterada.

Os *workloads* sintéticos podem ainda ser divididos em três classes. Os *workloads* de aplicação são aqueles que se baseiam na execução de aplicações com entradas bem definidas e determinísticas. Por exemplo, um *workload* de aplicação pode ser a compilação de um programa determinado pelo compilador *gcc*, ou então a simulação de um circuito elétrico conhecido pelo simulador *SPICE*. Estes *workloads* são sintéticos, pois não representam necessariamente a aplicação real, com os parâmetros reais. Mesmo assim, são uma aplicação completa e bem estruturada. Outra classe de *workload* são os *workload* kernel, que não são uma aplicação completa e bem estruturada, mas sim uma parte importante do sistema que se quer avaliar. Por exemplo, o trecho de código utilizado para executar uma convolução de sinais pode ser um *workload* interessante quando se avalia um processador DSP. Apesar de não ser uma aplicação completa, a operação de convolução representa uma função bastante comum neste tipo de aplicação e que é responsável por grande parte do processamento do DSP. Por fim, existem os *workloads* sintéticos propriamente ditos, que são nada mais do que um conjunto de instruções sem uma função definida, que são agrupados apenas para avaliação das características de desempenho deste determinado conjunto. Exemplos clássicos deste tipo de *workload* são os aplicados pelos *benchmarks* Whetstone [CUR 76] e Dhrystone [WEI 84].

A escolha do *workload* é fundamental para o sucesso da avaliação de desempenho. De nada adiantam resultados que foram obtidos a partir de *workloads* des-

conhecidos ou pouco desenvolvidos, que não representam efetivamente as condições reais de funcionamento do sistema que está sendo avaliado. Pontos como quais as partes do sistema que são exercitadas pelo *workload* e qual o nível de detalhamento e representatividade do mesmo são fundamentais na escolha e construção de uma carga significativa e que possa extrair resultados coerentes e úteis do sistema que está sendo avaliado.

A composição do *workload* deve ser baseada nas características do sistema que está sendo testado e nas funções e operações desempenhadas por ele. Quando está-se avaliando um sistema de banco de dados, onde as operações básicas são transações de consulta e atualização, o *workload* deve ser composto por uma mistura destas duas operações. Não seria representativo apresentar dados como quantidades de chamadas de sistemas operacionais realizadas, ou instruções de máquina gastas por cada transação, uma vez que este não é o alvo da avaliação e estas informações estão em um nível de detalhamento maior do que o objetivo do estudo. Da mesma forma, a avaliação de uma solução de nível físico em uma rede de computadores deve apresentar a quantidade de bits que este nível físico é capaz de transmitir por segundo, e não faz sentido resultados que apresentem mensagens ou pacotes que foram trocados por segundo neste meio físico, pois estas operações encontram-se em níveis funcionais diferentes do que é executado pelo nível físico de uma rede, que só faz transmitir pacotes de um extremo a outro de uma conexão. A figura 2.1, extraída de [JAI 91] representa bem a noção de hierarquia de serviços em um sistema, mostrando as unidades básicas de operação executadas por cada um dos níveis especificados.

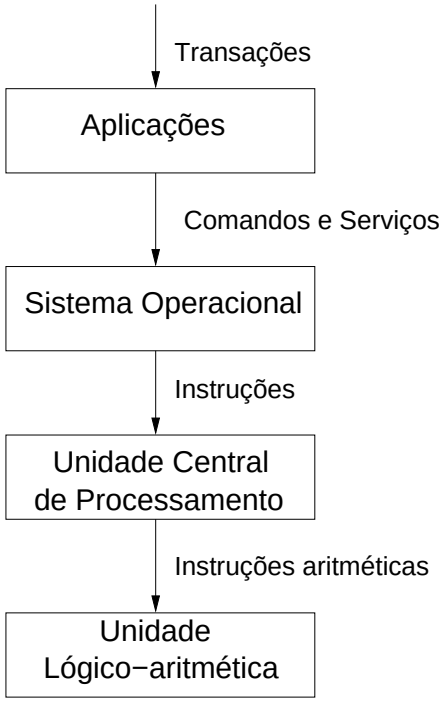


FIGURA 2.1 – A Hierarquia de Serviços de uma CPU

Outro ponto importante na elaboração de um *workload* é o nível de detalhe atingido por ele. Um *workload* simples pode ser composto apenas pela operação mais

freqüente no sistema. Apesar de fornecer uma informação a respeito do sistema sendo avaliado, ela é muito pouco detalhada e bastante restrita. Por outro lado, *workloads* compostos por um conjunto maior de operações distribuídas de acordo com a freqüência de ocorrência de cada uma no sistema já fornece informações mais detalhadas. Mais detalhes ainda são possíveis caso o *workload* inclua detalhes como contextualização das operações, executando-as na mesma seqüência aproximada de ocorrência no sistema real. Conforme aumenta o nível de detalhe do *workload*, mais representativos podem ser os resultados, porém mais complicado pode ser o processo de construção do *workload*, extração e análise de resultados.

Por fim, um *workload* deve ser facilmente repetível, de forma que os resultados obtidos em uma avaliação possam ser reproduzidos com pouca variação. *Workloads* que incluam variáveis randômicas ou fatores externos ao sistema prejudicam a característica de repetibilidade do mesmo.

2.4 Benchmarks

O processo de comparação de desempenho de dois ou mais sistemas através de medição direta é chamado de *benchmarking*, e os *workloads* utilizados no processo são denominados de *benchmarks* [JAI 91]. O processo de *benchmarking* é conduzido através de experimentos com *workloads* repetíveis e controláveis, de forma que eles possam ser rodados diversas vezes em cada um dos sistemas alvo da comparação ou avaliação, e da mesma forma em todos eles, tal que os resultados obtidos possam ser comparados.

A nomenclatura apresentada por Morris e Roth em [MOR 82] sugere que a unidade básica de qualquer benchmark é chamado de *benchmark kernel*. Um *benchmark kernel* pode estar em nível de instruções de máquina ou de operações de mais alto nível, como transações ou operações de ordenamento de dados, por exemplo. O próximo estágio na terminologia é a instrução de benchmark, composta por mais de um *benchmark kernel*. Uma instrução de benchmark é análoga a uma linha de código em um programa ou uma subrotina. Um conjunto de instruções de benchmark é o equivalente a um programa de computador, ou *workload*, e a coleção de todos os conjuntos de instruções de benchmark forma o benchmark propriamente dito. Em outras palavras, o *benchmark* é um conjunto de *workloads*.

A utilização de benchmarks para avaliação de sistemas é uma técnica bem aceita e largamente utilizada, uma vez que é capaz de reportar o comportamento do sistema em situações próximas do uso real. A utilização de *workloads* reais é praticamente impossível, porém a aproximação do resultado pode ser satisfatória a partir de *workloads* bem construídos e experimentos representativos e bem conduzidos. O benchmark é, com certeza, a técnica de avaliação de desempenho que é capaz de se aproximar o máximo dos resultados reais de desempenho do sistema considerado com um esforço pequeno quando comparado com as demais técnicas de avaliação. Estas exigem um esforço consideravelmente maior caso desejem obter resultados bastante próximos do mundo real, uma vez que terão que modelar diversos fatores que são normalmente ignorados numa modelagem para aproximações mais simples.

Por outro lado, a utilização de benchmarks exige a existência do sistema real a ser avaliado, ou ao menos um protótipo, uma vez que a técnica de benchmark

é baseada em medições diretas no sistema a ser avaliado. Ao mesmo tempo, o *workload* deve ser desenvolvido cuidadosamente, de forma que seja representativo dos *workloads* reais do sistema que está sendo avaliado . Por fim, a utilização de benchmarks quase sempre implica em resultados alterados pelas perturbações causadas pela instrumentação necessária para realizar as medições.

Os benchmarks mais tradicionais e antigos são o Whetstone [CUR 76] e o Dhrystone [WEI 84]. Atualmente, o benchmark mais utilizado é o SPEC [HEN 2000], formado por um conjunto de aplicações variadas que exercitam diversas partes de um computador pessoal ou estação de trabalho.

2.5 Resultados: Extração e Apresentação dos Dados

O produto final de um processo de avaliação de desempenho é o conjunto de resultados obtidos durante o processo e estes devem ser reportados de forma adequada. Para que seja possível chegar ao final do processo, ainda são necessárias duas etapas fundamentais e decisivas para a qualidade da avaliação: as técnicas para extração dos resultados e o conseqüente tratamento e apresentação dos dados obtidos.

A extração dos resultados de um sistema se dá através do uso de monitores. Monitores são ferramentas utilizadas para observar as atividades do sistema quando este está sendo exercitado pelo *workload* estabelecido para o processo de avaliação, de forma a caracterizar as métricas previamente estabelecidas. Tipicamente, um monitor observa as atividades do sistema e coleta estatísticas. Alguns monitores ainda são capazes de analisar os dados e apresentar algum tipo de resultado, ou ainda apontar possíveis pontos falhos e apresentar alguma sugestão para melhorias.

Na terminologia de monitores, alguns termos são freqüentemente utilizados, e serão citados aqui. Evento é o primeiro deles, e indica uma mudança no estado do sistema monitorado. Exemplos de eventos são o recebimento de um novo pacote de dados, o início de uma transação ou a conclusão de um ciclo de leitura de dados em um barramento. Um registro da seqüência de eventos que ocorreram no sistema, incluindo o tempo em que cada evento ocorreu, é chamado de *trace*. Ainda relacionado aos eventos existe o termo taxa máxima de entrada, que representa a máxima freqüência de ocorrência de eventos que o monitor é capaz de manipular e registrar corretamente, e a resolução, que informa qual a menor diferença de tempo que o sistema é capaz de registrar.

Na monitoração de sistemas com o objetivo de caracterização de métricas, é necessário levar em consideração a intrusão, ou perturbação, causada pelo equipamento de monitoração no comportamento normal do sistema avaliado. A perturbação é causada pelas modificações necessárias para que o monitor consiga observar o comportamento do sistema. É importante que a perturbação de um monitor seja mínima, ou então seja bem conhecida, de tal forma que seja possível descontar a influência do monitor nos valores obtidos em cada uma das métricas.

Monitores podem ser implementados por *software*, *hardware*, ou ainda de forma híbrida. Tipicamente, monitores de *software* são utilizados para registrar eventos em sistemas de alto nível, como um banco de dados. A cada ativação do monitor, uma seqüência de instruções é executada de forma a registrar a ocorrência do evento. Já monitores de *hardware* são equipamentos independentes do sistema avaliado,

utilizados para registros de eventos de baixo nível, e são conectados a ele através de *probes*.

Monitores de *software* são, tipicamente, mais intrusivos, apresentam uma menor resolução e possuem uma taxa máxima de eventos inferior a um monitor de *hardware*. Porém, ao mesmo tempo eles possuem uma capacidade de memória maior, o que possibilita o armazenamento de um *trace* de eventos mais longo e completo que um monitor de *hardware*.

Qualquer que seja a técnica de monitoração utilizada, ela irá gerar um conjunto de resultados a respeito do sistema monitorado, que precisa ser tratado para posterior apresentação dos resultados finais do processo de avaliação. Apesar de ser a última etapa a ser executada, a análise e apresentação dos dados é uma das mais importantes, afinal uma apresentação mal feita, ou mesmo a ausência da apresentação, faz com que os resultados do processo de avaliação de desempenho sejam ignorados e é como se a mesma não tivesse sido executada. A manipulação dos dados é feita com técnicas estatísticas. Valores como média, desvio padrão, intervalo de confiança, dispersão e distribuição são calculados, e serão utilizados posteriormente na apresentação.

Finalmente, a apresentação dos dados é feita através de gráficos, tabelas e histogramas, mostrando média, desvio padrão e intervalo de confiança de cada uma das métricas de interesse. É importante que não apenas os valores sejam explicitados nos resultados finais, mas também as condições sob as quais os *workloads* foram aplicados, bem como as configurações do equipamento medido, os fatores que foram variados e as condições das repetições executadas.

3 CPUs Industriais

Uma CPU Industrial, ou Controlador Lógico Programável (CLP), é um sistema de controle industrial, de tempo real, modular, de estado sólido, que opera sobre um programa armazenado em sua memória, normalmente escrito em lógica de relés, complementado por recursos de programação e monitoração [BOU 96].

Um "sistema" significa que o CLP é um pacote completo, com todas as partes projetadas e testadas para funcionarem juntas. Isto inclui componentes de *software* e *hardware*. Ao contrário de computadores genéricos, que incluem componentes de diferentes fornecedores, um CLP normalmente é fornecido por um único fabricante, que se certifica de que todas as partes funcionam corretamente juntas.

"Modular" indica que capacidades e funcionalidades extras podem ser adicionadas quando necessárias, bem como podem ser removidas para serem usadas em outro sistema.

"Industrial" reflete a adequação do equipamento às variações de temperatura, umidade, vibração e voltagem encontradas em ambientes industriais, e que ele foi projetado para funcionar 24 horas por dia, 7 dias por semana.

"Tempo real" significa que o controlador está sempre funcionando e interagindo com o mundo externo, composto de sensores e atuadores, e deve ser capaz de responder a um estímulo destes elementos em um intervalo de tempo determinado.

"Estado sólido" infere que não existem tubos de vácuo, equipamentos pneumáticos ou hidráulicos em um controlador programável. Normalmente, nem mesmo unidades de disquete são utilizadas.

O CLP é utilizado para controlar máquinas e/ou processos industriais por meio de uma lógica de programa e pontos de entrada e saída [JON 83]. O controlador programável é composto por dois elementos básicos, uma unidade central de processamento e uma interface de entrada e saída, e executa instruções lógicas, de seqüenciamento, de temporização e numéricas [MIY 96]. A figura 3.1 ilustra os componentes de um controlador lógico programável.

A unidade de processamento pode ser subdividida em memória, processador e fonte de alimentação. Normalmente, uma bateria estará presente para que o sistema possa manter o funcionamento mesmo em situações de queda de energia. A CPU lê dados dos sensores de entrada, executa o programa armazenado na memória e escreve as saídas apropriadas nos atuadores de saída. O processo contínuo de leitura de entradas, processamento e escrita de saídas é chamado de *ciclo de varredura*. Uma das mais importantes medidas de desempenho de um CLP é justamente o tempo que ele leva para realizar um ciclo de varredura completo, medida chamada de *tempo de ciclo*. A figura 3.2 mostra as etapas executadas pelo controlador programável durante o ciclo de varredura.

A interface de entrada e saída é a forma pela qual os dispositivos de campo se conectam à CPU Industrial. A finalidade desta interface é tratar os sinais de entrada e saída, realizando a conversão dos valores do domínio do dispositivo para o domínio do controlador. Como exemplo de dispositivos de campo pode-se citar sensores de temperatura, chaves e botões como dispositivos de entrada e motores, válvulas e solenóides como dispositivos de saída. Sob o ponto de vista da CPU Industrial, o ambiente controlado se comporta como um gerador de eventos em instantes de tempo desconhecidos e pouco previsíveis. Estes eventos devem ser recebidos pelos

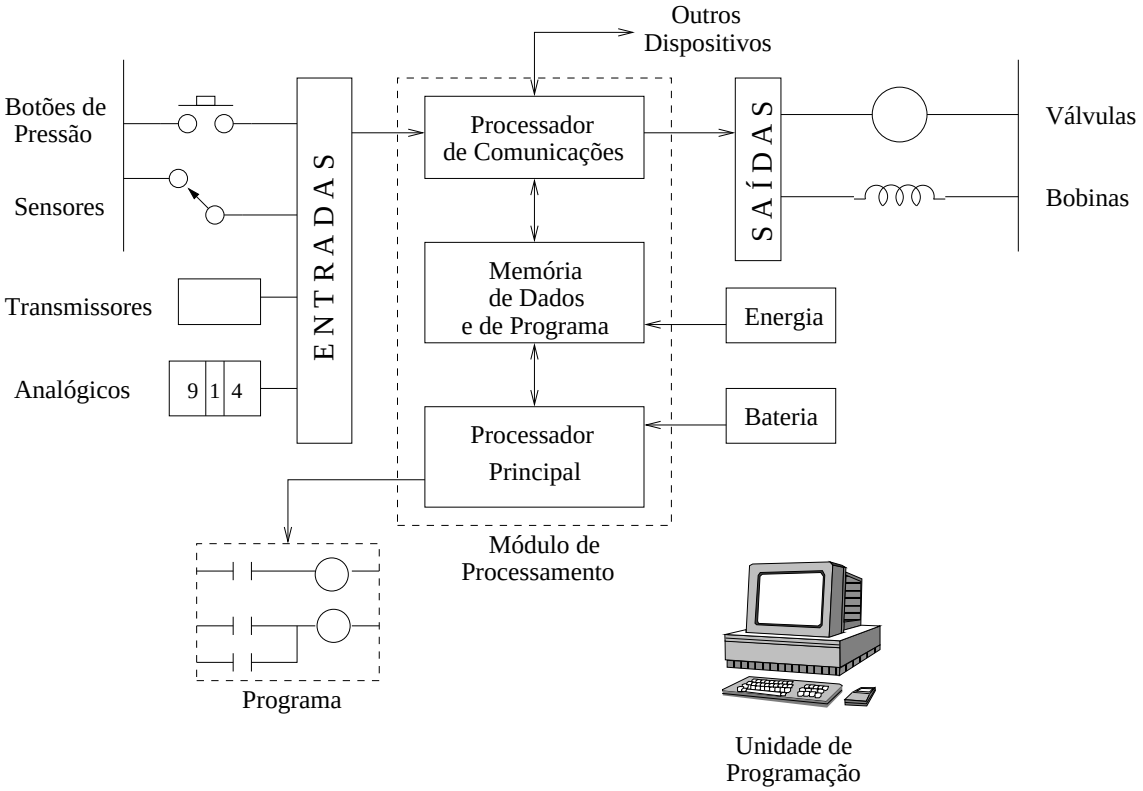


FIGURA 3.1 – Os componentes de um CLP típico

dispositivos de campo e repassados ao processo de controle para tratamento e reação.

O sistema de controle é passível de monitoração externa do programa de usuário, permitindo que sejam verificados os valores de operandos e de pontos de entrada e saída por equipamentos de supervisão, que comumente estão localizados em computadores tradicionais. Estes sistemas de supervisão recebem o nome de Sistemas Supervisórios.

Apesar de não fazer parte do controlador programável formalmente, uma unidade de programação é necessário para que se possa realizar a transferência do programa de controle para a memória do CLP.

As aplicações de CLPs na automação industrial são diversas, como por exemplo em indústrias químicas, automotivas e siderúrgicas, entre várias outras. O CLP é tratado como um elemento essencial no controle do processo industrial, e juntamente com os robôs, é responsável pelo controle de praticamente toda a linha de produção. Entre as grandes vantagens do uso de CLPs no controle de processos, pode-se citar a alta confiabilidade dos equipamentos, a flexibilidade do processo de controle, o custo reduzido e a modularidade, que facilita a expansão do sistema de controle [BOU 96].

Em algumas fábricas, vários controladores estão ligados em rede, cada um realizando uma pequena parte do controle. Nestes casos, é necessário observar o *overhead* causado pelos protocolos de comunicação, que podem degradar o desempenho total do sistema, aumentando o tempo do ciclo de varredura [KIM 92]. Para tentar suprir essa deficiência, as empresas desenvolveram controladores de maior capacidade, capazes de controlar mais pontos de entrada e saída sem afetar signifi-

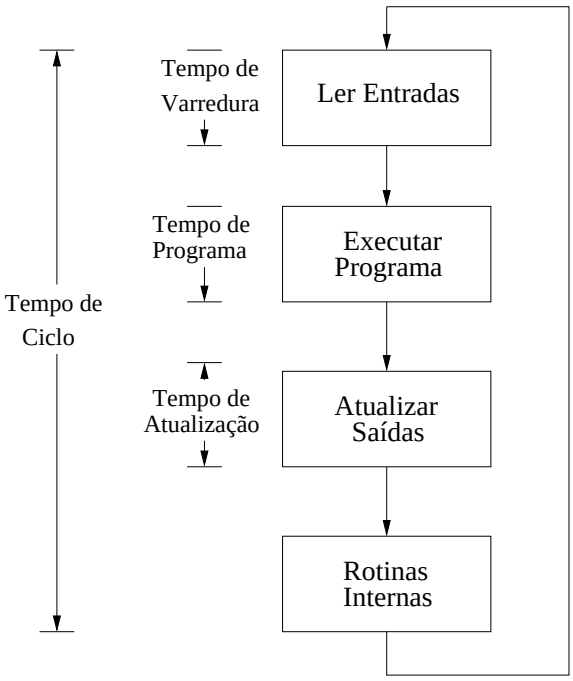


FIGURA 3.2 – As etapas do ciclo de varredura

cativamente o desempenho do processo de controle. No exemplo utilizado por Kim, Park e Kwon em [KIM 92], 28 controladores de grande porte são responsáveis pela linha de montagem da fábrica da Hyundai na Coréia, controlando mais de 65000 pontos de entrada e saída.

Hoje em dia, diversos microprocessadores de alta capacidade podem ser utilizados para a implementação de controladores programáveis. Porém, isso não garante um controlador programável de alto desempenho, principalmente porque controladores programáveis devem manipular bits, enquanto processadores genéricos manipulam bytes e palavras de 16 e 32 bits. Assim, ao utilizarem-se processadores genéricos para implementação de CLPs, são necessárias várias instruções para realizar uma única operação de bit [RHO 95]. Além disso, o acesso a memória é muito constante para leitura dos pontos de entrada durante o processamento da lógica, e processadores genéricos não estão preparados para lidar com grandes taxas de transferência de dados nos barramentos de acesso à memória. Diferenças nas linguagens de programação é outro fator limitante para o uso de microprocessadores genéricos na construção de CLPs. Portanto, o desenvolvimento de CPUs Industriais de Alto Desempenho passa por considerações arquiteturais de *hardware* e *software* principalmente, além de preocupações relacionadas a linguagens de programação e protocolos de comunicação.

3.1 CPUs Industriais: Um Sistema de Tempo Real

Por controlar ambientes industriais, interagindo com sensores e atuadores no mundo real, as CPUs Industriais fazem parte de um conjunto de aplicações especiais de computação: os *sistemas de tempo real*.

Sistema de Tempo Real é caracterizado na literatura [BUR 97], [KRI 97], [OLI 2000] como sendo um sistema no qual o instante de tempo em que a resposta é apresentada é tão importante quanto o valor da resposta. Assim, para uma computação estar correta, não basta apenas que seu valor esteja logicamente certo, mas ele deve ser apresentado num instante de tempo determinado. O instante de tempo máximo para apresentação da resposta é tipicamente chamado de *deadline*. Alternativamente, um sistema de tempo real pode ser definido como sendo um sistema de processamento de informações que tem que responder a eventos de entrada externos em um intervalo de tempo finito e determinado. Neste tipo de sistema, a falta de uma resposta pode ser tão grave quanto uma resposta incorreta.

Dentro dos sistemas de tempo real, tem-se uma subdivisão clássica, que classifica os sistemas em *soft real time* e *hard real time* [BUR 97]. Esta divisão está baseada na importância de se cumprir o intervalo determinado para apresentação da saída. Os *hard real time* são aqueles em que é absolutamente obrigatório que a resposta seja apresentada dentro do *deadline*. *Soft real time* são aqueles em que o tempo de resposta ainda é importante, porém o não cumprimento eventual de um *deadline* não compromete o funcionamento do programa ou sistema. Por exemplo, o sistema de controle de uma aeronave é *hard real time*, já que o não cumprimento dos *deadlines* pode levar a uma catástrofe. Já um sistema de vídeo conferência é um *soft real time*, visto que o atraso no envio de um pacote de áudio não vai influenciar decisivamente no processamento global dos dados nem no entendimento da conferência.

Alguns sistemas podem apresentar dois *deadlines*, um *hard* e um *soft*. Estes são classificados como *firm real time*. O *soft deadline* é o instante no qual a resposta é desejada, para uma utilização otimizada do sistema, mas que pode ser atrasado até o *hard deadline*, quando a resposta ainda tem importância para garantir a segurança e a integridade dos equipamentos. No intervalo de tempo entre o *soft* e o *hard deadline*, a importância da resposta diminui gradativamente, e passa a não ter nenhuma importância após o *hard deadline*.

Ao contrário do que possa parecer, um sistema de tempo real não implica necessariamente em computação de alto desempenho pelas métricas de alta velocidade de processamento [STA 88], e sim em alto desempenho quando consideradas as garantias de cumprimento de tempos de resposta para eventos gerados externamente. Performance é importante num sistema de tempo real, mas deve ser medida em termos das condições propiciadas pelo ambiente no qual a aplicação está inserida. Em sistemas de tempo real, computação de alto desempenho significa precisão computacional, eficiência e determinismo temporal [TOK 99]. Um sistema de alto desempenho pode ajudar, mas não soluciona por completo o projeto do sistema de tempo real. Fatores como arquitetura de *hardware*, sistemas operacionais e linguagens de programação são tão importantes quanto a carga computacional a que o sistema está sujeito [OLI 2000]. Dessa forma, grande parte dos esforços de pesquisa da área de tempo real estão concentrados em algoritmos de escalonamento, mecanismos de sincronização e acesso ao relógio, arquiteturas de memórias *cache* e definições de construções em linguagens de alto nível, entre outros.

No mundo do controle programável, quase a totalidade das aplicações é de tempo real, e destas a grande maioria é *hard real time*. É fácil imaginar que se a válvula de despressurização de um tanque não for aberta no instante correto, ele provavelmente irá explodir. Ou então se o aquecedor de uma mistura química não

for desligado quando a temperatura máxima for atingida, um acidente grave pode ocorrer. Os CLPs substituem os humanos em muitos ambientes onde a precisão é fundamental, e um pequeno desvio da especificação pode ser decisivo para a falha do processo de controle.

3.2 Arquitetura de Controle

Em um sistema de controle baseado em CLPs, existem três elementos facilmente identificáveis: o controlador propriamente dito, responsável por executar a lógica de controle que mantém os processos industriais em funcionamento, os pontos de entrada e saída, que amostram os dados do ambiente controlado através de sensores e alteram o mesmo ambiente pelos atuadores, e as redes de comunicação, que realizam a comunicação entre o controlador e os pontos de entrada e saída, entre controladores ou entre o controlador e um sistema de gerenciamento e supervisão.

O controlador propriamente dito pode ser dividido em componentes de hardware e *software*. Com arquiteturas tradicionalmente baseadas em microcontroladores, os controladores de grande porte têm migrado gradativamente para arquiteturas com microprocessadores, em virtude de melhorias na tecnologia do processo de fabricação e conseqüente aumento de velocidade. Junto com os microprocessadores e microcontroladores, existem as memórias, essenciais em todo sistema computacional. Em um controlador programável, existem tipicamente as memórias não voláteis, que mantêm seu conteúdo mesmo quando desenergizadas, e as memórias voláteis, mais velozes e de fácil escrita que as anteriores, porém que não retêm os dados em caso de desenergização. As primeiras são usadas para armazenar as partes essenciais do sistema, como o sistema operacional ou executivo e o programa de controle. Já a memória volátil é usada como memória de trabalho, servindo para espelhar os pontos de entrada e saída, para realização de cálculos matemáticos, para comunicação e para armazenamento de informações de estado do sistema.

Na camada de *software*, existem duas partes distintas: o programa de controle, que especifica a lógica de seqüenciamento do processo industrial, e o sistema operacional, que também recebe o nome de sistema executivo. O programa de controle é especificado pelo usuário, e é daí que vem uma das grandes vantagens do controlador programável quando comparado com o antigo painel de relés: a facilidade de reprogramação do equipamento para o controle de um novo processo. Para especificar a lógica de controle, existem algumas linguagens de programação mais tradicionalmente utilizadas, como a lógica de relés ou os blocos de função, e mais recentemente padronizações de linguagens por parte do IEC [IEC 93].

Já o sistema operacional é menos padronizado, e muito raramente sofre alterações ao longo da vida de um equipamento. Nos sistemas microcontrolados, é comum a existência de um sistema proprietário, que explora ao máximo as particularidades do microcontrolador e do *hardware* onde ele está inserido. Por outro lado, nos sistemas maiores, comumente baseados em microprocessadores, podem surgir também soluções com sistemas operacionais tradicionais, tanto de tempo real, como QNX ou VxWorks, quanto convencionais, como Windows NT e Linux, como forma de acelerar o desenvolvimento do controlador e obter um bom desempenho com processadores desktop, tanto Pentium quanto Sparc.

Os pontos de entrada e saída são o segundo elemento que compõe uma ar-

quitetura de controle de processos industriais. Os pontos de entrada e saída são a interface de comunicação do controlador com o mundo externo e é através deles que o controlador envia comandos e recebe informações a respeito do processo controlado. Existem dois tipos de pontos de entrada e saída: os digitais e os analógicos. Os pontos digitais representam condições booleanas do processo de controle, onde apenas um bit é suficiente para conter a informação desejada. Por exemplo, são pontos de entrada digital o estado de uma válvula, que pode estar aberta ou fechada, ou a posição de uma chave de segurança, que pode estar ligada ou desligada. Pontos digitais são facilmente amostrados, por não precisarem de nenhum processamento extra além da informação de estado do ponto, e são atualizados em cada varredura. Em contraste com os pontos digitais, existem os pontos analógicos, utilizados para receber e enviar informações de grandezas físicas do processo de controle, como a temperatura de um produto químico ou a velocidade de rotação de um motor. A amostragem de um ponto analógico é mais complexa que a de um ponto digital, pois envolve condicionamento e processamento da informação recebida, de forma a transformá-la do mundo analógico (físico), para o mundo do controle (digital) ou vice-versa. Para tanto, existe uma escala de conversão de valores, que depende do módulo que está fazendo a leitura ou escrita e do sensor ou atuador envolvido. Há também um valor máximo capaz de ser amostrado pelo equipamento, o chamado fundo de escala, assim como há também um valor mínimo que pode ser detectado pelo sistema. Pela complexidade de amostragem e também pela baixa velocidade de variação de grandezas analógicas, um ponto analógico não é amostrado em todos os ciclos de varredura.

O terceiro componente que faz parte da arquitetura de controle é a rede de comunicação. Em controladores menores, o controlador e os pontos de entrada e saída podem fazer parte de um mesmo equipamento, e então a rede de comunicação se restringe ao barramento interno da placa do controlador. Porém, em sistemas de médio e grande porte, os módulos de entrada e saída são distribuídos pela planta industrial, e a rede de comunicação é o meio físico onde trafegam os dados lidos pelos módulos e os comandos enviados pelo controlador. Neste caso, aparecem as redes de campo, um conjunto de protocolos e equipamentos de interligação de redes responsável por garantir a comunicação entre o controlador e a entrada e saída. Algumas vezes, o próprio módulo de entrada e saída pode realizar um processo de controle simplificado, e então este módulo é classificado como módulo de I/O inteligente.

Além da rede de campo, utilizada na comunicação entre o controlador e os pontos de entrada e saída, existem mais duas redes de comunicação que fazem parte da arquitetura de controle. A primeira delas é a rede de supervisão, responsável pela ligação entre o controlador programável e os equipamentos de supervisão. Através dos equipamentos de supervisão, também chamados de sistemas supervisórios, operadores humanos podem monitorar o funcionamento do processo controlado, verificando o estado do sistema em um nível de abstração mais alto, com representações gráficas para cada componente relevante do sistema. Por fim, existe a rede de comunicação entre os controladores programáveis nos processos de controle distribuído. Esta rede começou a se tornar mais comum nos últimos anos, com o avanço do controle distribuído e com o aumento da necessidade de controladores redundantes, recursos utilizados em sistemas de alta segurança, onde são necessários recursos para suportar falhas dos equipamentos controle.

Um exemplo típico de aplicação que necessita de um equipamento de alto desempenho são as plataformas de petróleo e os navios petrolíferos da Petrobrás. Num navio petrolífero, há diversos sub-sistemas a serem controlados, o principal deles representando a extração e tratamento do óleo, mas há também os sub-sistemas de energia, de incêndio, de gás e de dutos de transporte do petróleo. O número de pontos a ser controlado é grande, de 10000 a 12000 pontos analógicos e digitais, distribuídos ao longo da extensão do navio. Cada sub-sistema deve ser redundante, de forma a suportar falhas de um dos controladores responsáveis pelo mesmo. Assim como os controladores, as redes de comunicação e os controladores de entrada e saída também são duplicadas, para garantir a continuidade do serviço e a segurança do navio em casos de falhas da redes ou dos módulos de entrada e saída.

3.3 Arquitetura de Software

A figura 3.3, adaptada de [CHR 2000], mostra as tarefas executadas pelo sistema operacional de uma CPU Industrial. Os módulos em cinza são tipicamente de tempo real, enquanto os demais são processos comuns, sem restrições temporais. É possível observar que grande parte das tarefas são de tempo real, como a amostragem dos sensores, a execução do programa de controle e a base de dados presente no sistema.

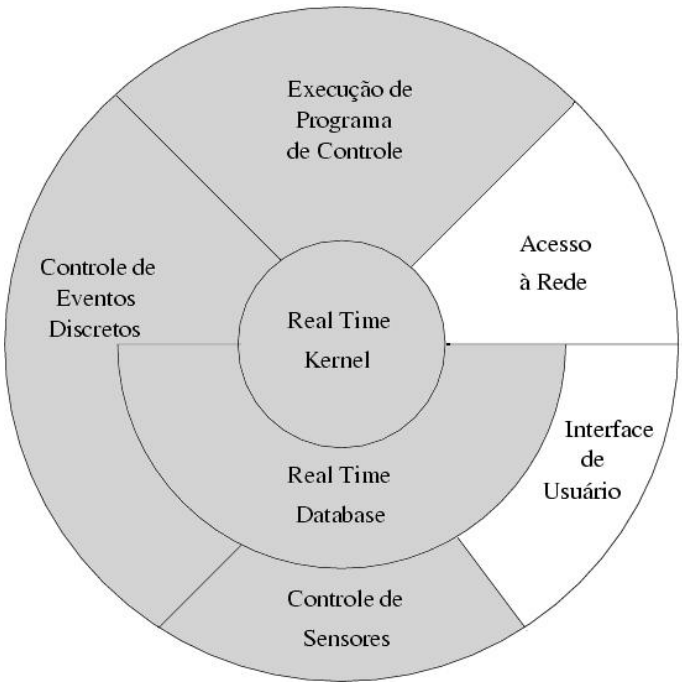


FIGURA 3.3 – Tarefas da CPU Industrial

A parte principal do equipamento é o kernel de tempo real. É através do kernel que as demais tarefas podem acessar os recursos de *hardware*. Esta deve ser a parte que deve concentrar os maiores esforços para garantir o comportamento determinístico do *software*, pois todas as demais funções da CPU Industrial são construídas sobre este módulo. É fundamental que o kernel seja de tempo real, em

virtude das características do equipamento e das demais tarefas responsáveis pelo sucesso do controle do processo industrial.

A base de dados de tempo real é a responsável por armazenar informações a respeito da configuração e estado do equipamento e compartilhar essas informações com as demais tarefas que necessitem destas informações para operarem corretamente. É nesta base de dados que estão armazenadas as informações sobre o conjunto de módulos de *hardware* conectados ao processador principal da CPU. É também na base de dados que são guardadas informações a respeito do estado de cada um dos pontos de entrada e saída, tanto analógicos quanto digitais. Tipicamente, as tarefas que necessitam de informações da base de dados são as que executam o programa de controle de processo e parte do sistema de controle de eventos discretos, que responde aos estímulos dos sensores, atualizando a base de dados com as mudanças no estado dos pontos de entrada e saída.

A execução do programa de controle é outra tarefa de tempo real. Esta é a principal função executada pela CPU, e certamente a tarefa que executa o programa de controle é de alta prioridade. O tempo de execução deve ser o menor possível, de forma a minimizar o tempo de reação do sistema a mudanças no ambiente controlado. Um tempo de reação muito longo pode levar a conseqüências indesejáveis, como por exemplo alterar o estado de uma chave no momento incorreto, quando o estado real do sistema não está mais corretamente atualizado na base de dados de tempo real, levando o processo controlado a um estado inconsistente.

O módulo de controle de eventos discretos é responsável por duas funções: eventos dos sensores e interrupções de *hardware*. Os eventos discretos gerados por alterações no estado dos pontos de entrada e saída devem ser reportados à base de dados de tempo real, para a atualização do estado dos mesmos. Já os eventos gerados por interrupção de *hardware* devem ser repassados para o kernel, que se encarrega de executar as rotinas associadas a cada interrupção. Tipicamente, estes eventos podem indicar a passagem de mais um ciclo do relógio da máquina, quando devem ser atualizados os estados dos temporizadores lógicos do sistema, ou a chegada de dados na porta serial, que devem ser lidos pelos módulos de comunicação e acesso à rede.

A interface de controle de sensores inclui drivers específicos para acesso a diferentes tipos de sensores e módulos de entrada e saída presentes no equipamento. É através desta interface que são adquiridas informações acerca do processo controlado que servirão para que os outros módulos do controlador possam tomar decisões e atuar no sistema. Esta interface não inclui sensores digitais simples, uma vez que estes são tratados pelo módulo de controle de eventos discretos. Neste módulo estarão presentes rotinas para acesso a dispositivos de funções mais complexas, como por exemplo o controle do movimento de um braço mecânico.

O processo de interface com o usuário é de baixa prioridade. A atualização das telas de informação pode ser postergada, uma vez que não faz parte das tarefas principais do controle. A interface serve apenas para informar aos usuários sobre o estado do processo que está sendo controlado, e em caso de sobrecarga do sistema a velocidade de atualização pode ser diminuída sem que grandes danos sejam causados.

Por fim, a tarefa de acesso à rede é a mais difícil de ser classificada. O determinismo temporal é mais dependente do meio físico e da estrutura da rede do que propriamente do *hardware* de controle e do sistema operacional utilizado. Atualmente, os controladores de alto desempenho têm utilizado redes Ethernet com fibra ótica

para realizar a interligação entre as diversas partes do sistema de controle, e esta rede não fornece garantias determinísticas no acesso ao meio. Assim, é impossível prever quando uma aplicação poderá acessar a rede, caso esta esteja sobrecarregada. Por outro lado, em redes como Profibus e Modbus, o controle de acesso ao meio é mais determinístico, permitindo que uma garantia de acesso seja fornecida. O tópico de redes de comunicação é vasto, e foi em parte discutido em [NAC 2000].

3.3.1 Alternativas de Implementação

A implementação da arquitetura de *software* pode ser desenvolvida de diversas maneiras. As mais comuns e tradicionais são o uso de sistemas monolíticos, também chamados de sistemas executivos, ou o uso de sistemas operacionais de tempo real, onde cada tarefa, ou grupo de tarefas, da figura 3.3 é codificada em um processo independente dentro do controlador, e estes trabalham de forma cooperativa para executar todas as tarefas de controle exigidas.

No sistema monolítico, todas as tarefas a serem executadas pela CPU Industrial são codificadas em um mesmo processo, de forma bastante integrada. A grande vantagem de sistemas monolíticos é o seu alto determinismo, já que apenas um processo está utilizando o processador e é possível precisar que etapa do controle será executada em cada instante no tempo. Por outro lado, uma falha de programação pode comprometer toda a arquitetura, uma fez que um bloqueio causado por uma condição imprevista trava a execução de todos os módulos.

Em contraste aos sistemas monolíticos, soluções baseadas em sistemas ou núcleos de tempo real também são utilizadas. Neste caso, as tarefas são implementadas como processos independentes, coordenadas entre si pela ação do processo (extra) de controle de escalonamento, que é executado pelo núcleo do sistema. Neste caso, ganha-se modularidade, extensibilidade e facilidade de manutenção e codificação, em detrimento de um pouco de determinismo e previsibilidade no sistema.

4 Métricas de Desempenho

O trabalho de avaliação de desempenho passa por diversas etapas, conforme já discutido no capítulo 2. A primeira etapa a ser executada é o conhecimento do objeto-alvo da avaliação, apresentado no capítulo 3. Em seguida, são necessárias as métricas a serem utilizadas durante a avaliação, o que será definido neste capítulo. As métricas já foram apresentadas anteriormente em [NAC 2000], com menos detalhes do que serão apresentados aqui.

Neste capítulo é formalizada a definição de cada métrica e as possíveis relações entre elas. A relação entre as métricas é importante para que seja possível avaliar os resultados obtidos com as diversas medidas efetuadas e classificá-las em corretas ou erroneas. Além disso, permite inferir o valor de algumas métricas a partir da extração de outras. A relação entre as métricas dá também uma estimativa, ainda que grosseira, dos valores que devem ser obtidos para cada medida efetuada. Por fim, as métricas que caracterizam os componentes de mais baixo nível da CPU Industrial, como *hardware* e sistema operacional, servem também para conhecer e estabelecer os fatores a serem variados durante o estudo de desempenho. Sabendo quais os fatores de baixo nível que influenciam as principais métricas da CPU, é possível avaliar o impacto que uma modificação nestes componentes terá no desempenho final do equipamento sob o ponto de vista das métricas propostas aqui.

O foco principal do trabalho é avaliar o comportamento das CPUs Industriais do ponto de vista das aplicações, ou seja, as métricas principais que envolvem este trabalho são as relacionadas ao desempenho do processo de controle. Certamente, estas métricas são influenciadas por diversos fatores, como a arquitetura do *hardware* de controle utilizado, os recursos e as características do sistema operacional, a rede de comunicação, as linguagens e o sistema de programação, que envolve tanto o ambiente quanto o compilador e a geração de código da aplicação de controle. Estes elementos também serão estudados e métricas que abordem cada um deles também são apresentadas neste capítulo.

As métricas aqui desenvolvidas são independentes do modelo de implementação utilizado para a arquitetura de *software* da CPU Industrial. Certamente, a técnica utilizada para implementação, seja ela o executivo cíclico ou o núcleo de tempo real multitarefa, pode influenciar nos valores a serem obtidos para cada métrica, porém não irão influenciar na definição de cada uma delas. O objetivo das métricas é exatamente comparar, entre outras variáveis, as implementações distintas de CPUs, e portanto não seriam úteis caso tivessem uma definição diferente para cada tipo de implementação.

A seguir, são apresentadas as métricas propostas para caracterização de desempenho de CPUs Industriais, que podem ser utilizadas também na comparação de diversos modelos distintos de CPU. A figura 4.1 mostra a hierarquia entre os diversos componentes que fazem parte da implementação de uma CPU Industrial. A relação entre as várias métricas que serão discutidas neste capítulo seguem uma estrutura baseada nesta hierarquia. Em tempo de execução, as partes que influem no desempenho do sistema estão relacionadas ao processo de controle, à rede de comunicação, sistema operacional e dispositivos de *hardware*. Em tempo de projeto, o compilador e a arquitetura de *software* também exercem um papel importante no desempenho final do sistema. As métricas relacionadas ao processo de controle

são as primeiras a serem detalhadas, pois este é o foco principal do trabalho. Em seguida, as demais métricas, que abordam o *hardware*, o sistema operacional, a rede, o compilador e a arquitetura de *software* também são abordadas.

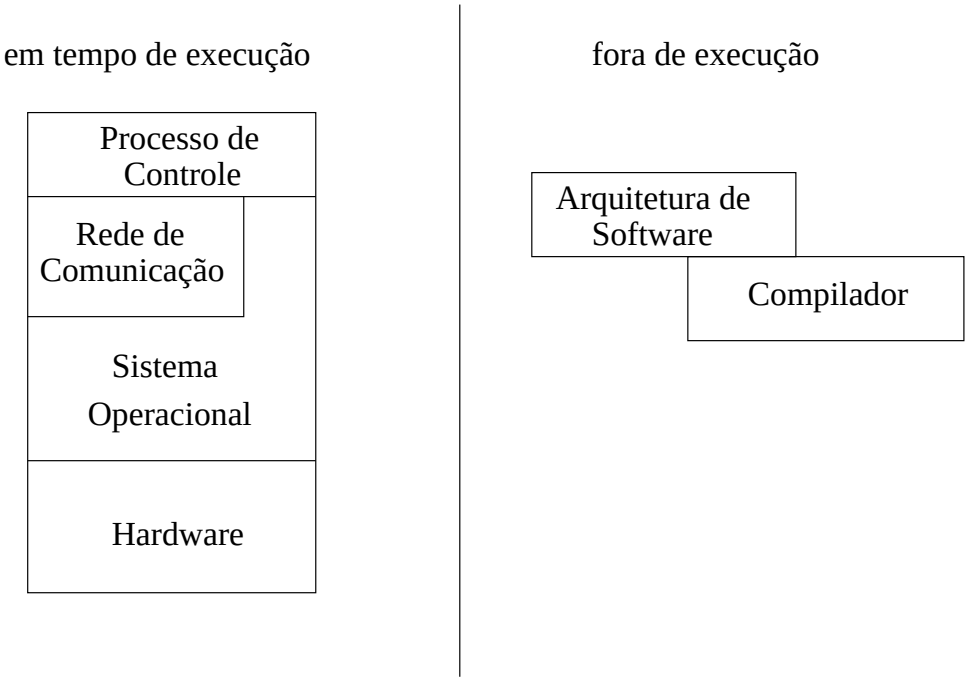


FIGURA 4.1 – Hierarquia de componentes

4.1 Processo de Controle

O processo de controle é a aplicação principal executada em uma CPU Industrial. Ele é um processo cíclico, infinito, que periodicamente amostra o estado do ambiente, através de sensores, executa a lógica de controle e atualiza o estado do ambiente novamente, modificando os valores dos atuadores. Na figura 3.3, que apresenta a arquitetura de *software* da CPU Industrial, o processo de controle é representado pela tarefa denominada "Execução do Programa de Controle".

As métricas importantes no processo de controle estão relacionadas com a velocidade e a *constância* com que a CPU é capaz de executar a lógica programada e também o tempo necessário para ler sensores e modificar atuadores. Assim, valores como tempo de ciclo, tempo de varredura, tempo de atualização e tempo de programa são os principais elementos no processo de avaliação de desempenho em nível de aplicação da CPU Industrial.

4.1.1 Tempo de Ciclo

O tempo de ciclo é uma das medidas de desempenho mais importantes de uma CPU Industrial. Ele avalia o tempo gasto pela CPU para processar um laço completo do processo de controle, o que inclui a execução do programa especificado pelo usuário e a leitura e escrita de entradas e saídas. A cada ciclo de execução, a

CPU amostra os pontos de entrada, executa o programa e calcula o estado de cada um dos pontos de saída.

A métrica de tempo de ciclo é do tipo "menor melhor". Assim, quanto menor for o tempo de ciclo, melhor será a avaliação da CPU Industrial. É importante que o tempo de ciclo seja o menor possível, pois assim pode-se assumir que as entradas não foram modificadas durante a etapa de execução do programa de controle. Como a CPU amostra os pontos de entrada no início de cada ciclo e calcula o estado das saídas com base nesta amostragem, um tempo de ciclo muito longo pode fazer com que alguns pontos de entrada tenham mudado de estado durante a execução do programa de controle, o que implicaria num estado das saídas diferente do que está sendo atualizado pela CPU ao final do ciclo. Isso pode levar o sistema controlado a um estado inconsistente, que pode ser perigoso para a segurança do ambiente.

Métrica 4.1 (Tempo de Ciclo) *é o intervalo de tempo decorrido entre duas ativações consecutivas do programa de controle.*

4.1.2 Tempo de Programa

O tempo de programa é uma métrica que mede a eficiência da CPU na interpretação e/ou execução das instruções de controle especificadas pelo usuário. O tempo de programa tem influência direta na métrica de tempo de ciclo, detalhada anteriormente, e é também uma métrica do tipo "menor melhor".

Métrica 4.2 (Tempo de Programa) *é o intervalo de tempo decorrido entre a execução da primeira instrução do programa de controle e a última instrução do programa de controle.*

A figura 4.2 mostra a composição do tempo de programa. A maior parte do tempo de programa é gasto com a execução do programa de controle propriamente dito. Ele é especificado numa linguagem própria, que não tem necessariamente uma equivalência 1:1 com as instruções do processador utilizado. Assim, as instruções básicas da linguagem de controle precisam ser traduzidas para instruções básicas do processador. Cada instrução de controle provavelmente será representada por mais de uma instrução *assembly* do processador. A instrução básica de controle será composta por 3 partes: a primeira delas envolve a leitura dos operandos da memória. Em seguida, é executada a operação de controle propriamente dita sobre estes operandos. Por fim, o resultado será escrito em uma posição de memória, que pode ser a mesma previamente lida.

As técnicas de estimativa de tempo de execução de programas embarcados como o *software* de controle executado pela CPU Industrial são as mais diversas. As técnicas variam em precisão das estimativas, custo e facilidade de obtenção dos resultados. Em geral, elas são divididas em [GIU 2001]:

- quanto ao nível de abstração, podem levar em conta análise de código fonte ou de código objeto;
- quanto às restrições de *software* às quais a técnica pode ser aplicada, por exemplo se considera ou não laços com condição de parada dinâmica, ponteiros, estruturas estáticas, etc ...;

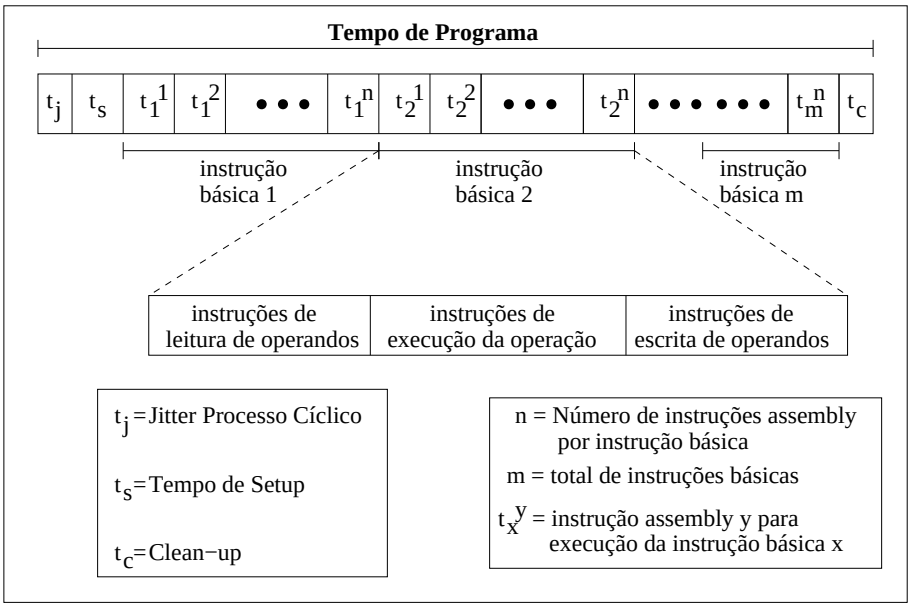


FIGURA 4.2 – Tempo de Programa

- custo de construção de modelos;
- precisão dos resultados;
- granulosidade da análise realizada;
- capacidade de lidar com estruturas de *hardware* como caches [LI 99] e pipelines;

Além do tempo para execução das instruções de controle, o tempo de programa ainda é composto por outros fatores. O primeiro e mais inconstante é o *jitter* dos processos cíclicos, inerente do sistema operacional. O *jitter* está relacionado à precisão dos timers e da capacidade do sistema operacional de ativar os processos em intervalos regulares constantes. Detalhes a respeito da composição do *jitter* estão representados na figura 4.5 e são discutidos na seção 4.3.

Também influem no tempo de programa um intervalo de *setup*, necessário para inicialização das variáveis do processo de execução do programa de controle, como resolução de referências para a área de imagem de memória e cópia de valores que devem ser preservados entre dois ciclos sucessivos. Da mesma forma, um tempo de finalização do processo é necessário e está representado pelo tempo de *clean up*, onde as estruturas de dados utilizadas são liberadas.

4.1.3 Tempo de Bit

O tempo de bit é uma medida que ilustra o tempo gasto pela CPU para executar uma instrução básica do programa de controle do usuário, que testa o estado de um ponto digital. O tempo de bit está representado na figura 4.2 como o tempo gasto para executar cada uma das m instruções básicas do programa de controle. Quanto menor o tempo de bit, menor o tempo de ciclo. Portanto, o tempo de bit também é uma métrica do tipo “menor melhor”.

Métrica 4.3 (Tempo de Bit) *é o tempo gasto para avaliação de uma instrução de controle digital, que lê o estado de um ponto digital (bit) e calcula um próximo estado, intermediário ou final, do programa de controle.*

4.1.4 Tempo de Reação

A métrica de tempo de reação representa o tempo necessário para que uma mudança no estado de uma das entradas se reflita no estado das saídas. O tempo de reação depende de três elementos: o tempo de varredura, o tempo de programa e o tempo de atualização. A figura 4.3 ilustra a composição do tempo de reação, uma métrica do tipo "menor melhor".

Métrica 4.4 (Tempo de Reação) *é o máximo tempo decorrido entre a mudança de um ponto digital de entrada e o efeito no ponto correspondente de saída. Se mais de um ponto de saída é influenciado pela mudança do valor da entrada, o tempo de reação deve ser computado levando em conta o último ponto de saída alterado pela mudança na entrada.*

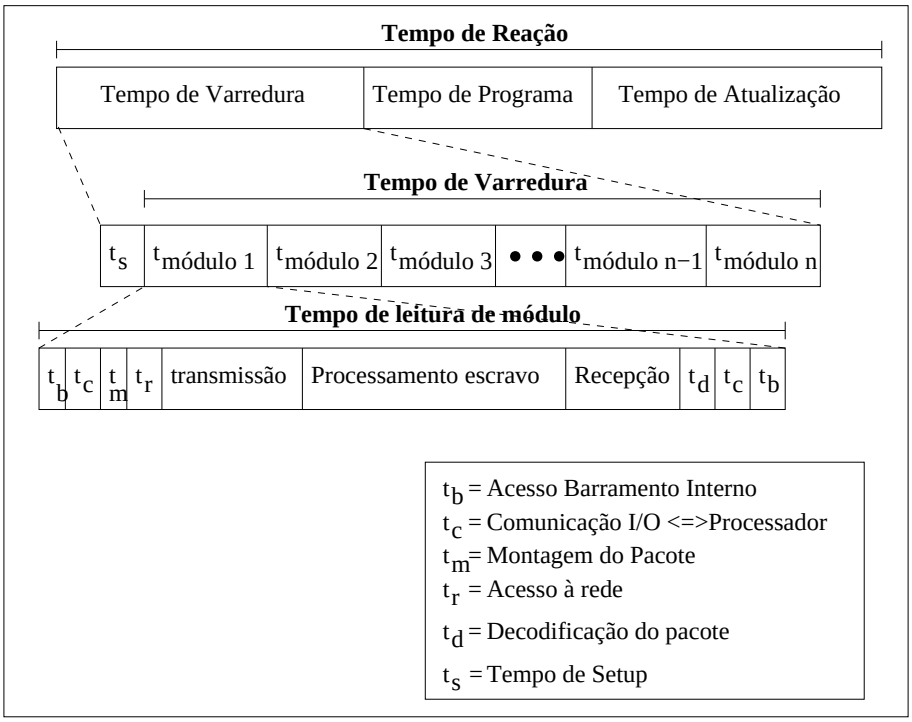


FIGURA 4.3 – Tempo de Reação

O tempo de varredura é o intervalo gasto para amostrar todos os pontos de entrada do sistema. Apenas após o término da varredura dos pontos de entrada é que tem início a execução do ciclo de controle. O tempo gasto na varredura das entradas depende diretamente do total de módulos de entrada a ser lido, onde um módulo pode ser composto por um ou mais pontos de entrada analógicos ou digitais.

Para cada módulo, o tempo de leitura é determinado pelo tempo de comunicação da CPU com o controlador de entrada e saída, o tempo gasto para montagem

da requisição de leitura, o envio da requisição propriamente dito, que pode envolver a rede de campo caso o ponto seja remoto, o tempo gasto pelo escravo para processar o pedido e enviar a resposta, o recebimento da resposta e sua conseqüente decodificação.

Além do tempo de leitura de um módulo, existe ainda um processamento necessário para acomodar a resposta do módulo em uma área adequada de memória, de forma que o processador possa acessá-la, e para verificar qual será o próximo módulo de entrada a ser lido. Pode existir, ainda, um intervalo mínimo que o barramento de entrada deve ser deixado em repouso antes que a próxima requisição de leitura de módulo seja feita.

Métrica 4.5 (Tempo de Varredura) *é o tempo decorrido entre a requisição de leitura do estado da primeira entrada do primeiro módulo de entrada da arquitetura de controle até o recebimento do estado da última entrada do último módulo de entrada da arquitetura de controle.*

O tempo de varredura é uma medida que varia bastante de aplicação para aplicação. Conforme a configuração de entradas é modificada, o tempo de varredura aumenta proporcionalmente. Alternativamente, o tempo de leitura de apenas um módulo de entrada apresenta um resultado mais significativo, que pode ser aplicado para diversas configurações de controle. Porém, ele não inclui a sobrecarga de processamento necessária para realizar o controle de sequenciamento de leitura dos módulos, e também não leva em consideração um eventual intervalo de repouso do barramento.

Métrica 4.6 (Tempo de Leitura de Entrada) *é o tempo decorrido entre a requisição de leitura do estado do primeiro bit de um módulo de entrada e o consequente recebimento da resposta do estado do ultimo bit do módulo de entrada.*

A segunda componente que influencia no tempo de reação é o tempo de programa, já discutido na seção 4.1.2. Por fim, a última componente do tempo de reação é o tempo de atualização, que envolve o tempo gasto para atualizar o estado dos pontos de saída. O tempo de atualização envolve as mesmas variáveis do tempo de varredura, e portanto não está detalhado na figura 4.3, e também é do tipo "menor melhor".

Métrica 4.7 (Tempo de Atualização) *é o tempo decorrido entre o envio do comando de atualização de estado do primeiro ponto de saída do primeiro módulo de saída da arquitetura de controle até o recebimento da confirmação da atualização do estado do último ponto de saída do último módulo da arquitetura de controle. Caso o protocolo de campo não inclua confirmação de atualização, o tempo de atualização é o tempo decorrido entre o comando de atualização de estado do primeiro ponto de saída do primeiro módulo de saída da arquitetura de controle até o envio do comando de atualização de estado do último ponto do último módulo de saída da arquitetura de controle.*

Da mesma forma que ocorre com o tempo de varredura, o tempo de atualização também varia bastante com a configuração de cada equipamento e ambiente de aplicação. Neste caso, o tempo de escrita de um módulo de saída também pode apresentar um resultado mais significativo e útil que o tempo de varredura propriamente dito.

Métrica 4.8 (Tempo de Escrita de Saída) *é o tempo decorrido entre o envio do comando de atualização de estado do primeiro bit de um módulo de saída até o recebimento da confirmação de atualização do último bit do mesmo módulo. Caso não haja confirmação no protocolo de escrita, o tempo de escrita de saída é o tempo decorrido entre o envio do comando de atualização de estado do primeiro bit até o envio do comando para atualização do estado do último bit do módulo de saída.*

4.2 Hardware

O *hardware* onde é implementada a CPU Industrial influencia no comportamento e desempenho do sistema, porém não de forma tão decisiva quanto o sistema operacional. Em termos práticos, o sistema operacional pode ser visto como uma "máquina virtual", que procura esconder os detalhes do *hardware* onde ele está sendo rodado da aplicação que o utiliza. Na CPU Industrial, não é diferente. Em vista disso, a maior parte das métricas de *hardware* são qualitativas, e não quantitativas. As métricas quantitativas são deixadas para avaliação do sistema operacional, que é quem efetivamente executa o código da CPU Industrial e utiliza os recursos de *hardware*.

4.2.1 Arquitetura e Conjunto de Instruções

Em termos de *hardware*, o processador exerce papel fundamental na caracterização de desempenho da CPU Industrial. Uma simples comparação de freqüências de funcionamento não é suficiente para determinar qual o processador mais adequado para a implementação da CPU. Mais do que a freqüência, é necessário conhecer a arquitetura interna do processador, sabendo se ele é uma máquina de 8, 16 ou até 32 bits, o tempo médio de ciclos de relógio para executar instruções, e também o conjunto de instruções disponíveis para o programador. Um conjunto reduzido de instruções tem a capacidade de ser executado mais rapidamente, porém serão necessárias mais instruções para executar a mesma função de um conjunto mais complexo de instruções de máquina. Portanto, a primeira métrica importante para avaliação do processador é a sua arquitetura interna, número de bits utilizados na arquitetura e o conjunto de instruções, juntamente com o tempo médio gasto na execução de cada instrução.

4.2.2 Barramento Interno

O Barramento Interno de conexão entre os dispositivos que compõem a CPU Industrial é essencial para a velocidade e eficiência da comunicação entre eles. Assim, é interessante avaliar a freqüência e a largura do barramento, o que refletirá a velocidade e capacidade de transmissão do mesmo. A velocidade do barramento, bem como sua largura interna, são métricas do tipo "maior melhor".

4.2.3 Troca a Quente

O suporte a troca a quente de módulos da CPU Industrial é uma funcionalidade muito importante para que o controle possa ser executado continuamente, sem paradas indesejadas para manutenção. A troca a quente significa que módulos de *hardware* podem ser adicionados e removidos sem que o sistema seja desligado.

A implementação de troca a quente não é uma tarefa simples, e portanto não são todas as CPUs que incluem esse suporte. A troca a quente implica em resolver problemas relacionados com a alimentação da placa, que deve ser tratada tanto quando ela está sendo inserida quanto quando ela está sendo removida. Inclui também necessidade de reconfiguração do sistema, que permite que uma nova placa seja inserida e configurada dinamicamente, de forma que todos os dispositivos sejam informados da ausência ou adição de um determinado módulo.

4.2.4 Memória

Os recursos de memória disponíveis e a sua organização também são importantes para a caracterização da CPU. A existência de um sistema de armazenamento estável é característica fundamental, bem como a presença de recursos que permitam a implementação de operandos retentivos, que não são perdidos em caso de falta de energia, permitindo que a CPU retome o funcionamento quando religada sem que nenhuma iteração ou configuração extra seja necessária. É importante destacar que nem sempre a memória estável pode ser utilizada como área para operandos retentivos. É o caso de memórias FLASH, que apresentam um limite máximo de ciclos de escrita durante sua vida útil.

A CPU utiliza dois tipos de memória: memória volátil, normalmente do tipo RAM, e memória não volátil, do tipo FLASH ou EEPROM. A memória não volátil substitui os discos rígidos, que são muito sensíveis e que seriam facilmente danificados num ambiente industrial. A figura 4.4 ilustra a ocupação de cada tipo de memória.

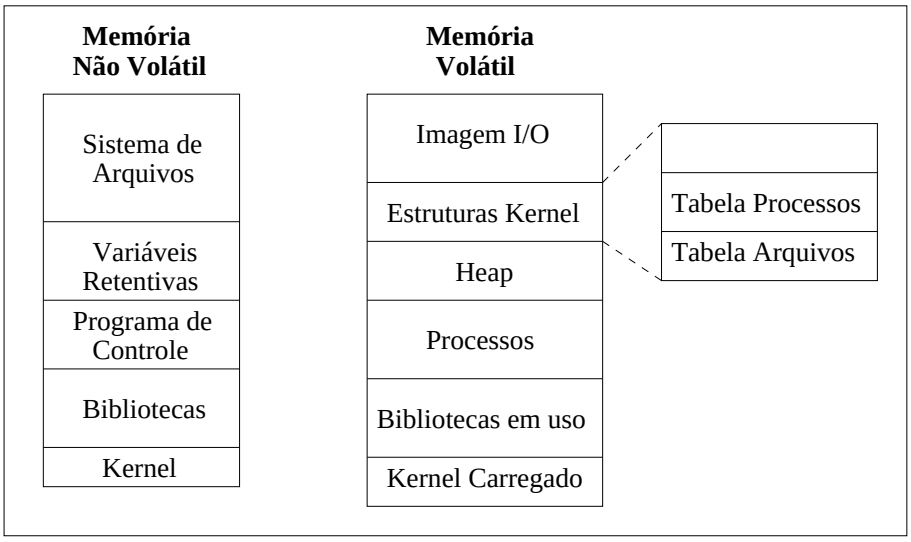


FIGURA 4.4 – Ocupação de memória

A memória não volátil serve como meio de armazenamento estável da CPU. Nela, estão presentes a imagem do sistema operacional, um sistema de arquivos, o programa de controle especificado pelo usuário, além das bibliotecas do sistema operacional e um espaço de armazenamento de variáveis retentivas.

Ao mesmo tempo, a memória volátil é usada como área de trabalho. O sistema operacional é carregado na RAM, bem como o conjunto de bibliotecas necessário para a execução. Há também espaço para execução dos processos e armazenamento

das variáveis do mesmo, bem como uma área de rascunho (*heap*) e um bloco de estruturas do kernel, onde são mantidas as tabelas de processos e arquivos. Algumas CPUs implementam parte da memória estável utilizando RAM e baterias, de forma que seja possível armazenar os operandos retentivos em RAM e não passar pela limitação no total de ciclos de escrita de memórias FLASH.

4.3 Núcleo de Tempo Real

Na CPU Industrial, está presente um suporte de tempo real, que disponibiliza os recursos necessários para programação do *firmware* do equipamento. Geralmente, funções para gerenciamento de múltiplas tarefas, semáforos, timers e controle de memória são utilizadas para a implementação do suporte de mais baixo nível na CPU, integrando os processos de execução da lógica de controle, amostragem do ambiente e modificação de pontos de saída.

Num sistema de tempo real, é comum a existência de processos cíclicos, que devem ser ativados em intervalos de tempo regulares e executam uma rotina específica. Numa CPU Industrial, o processo de controle pode ser implementado por um processo cíclico, que é ativado constantemente para executar a rotina de controle especificada.

4.3.1 Jitter de Processos Periódicos

O processo cíclico, porém, é ativado em intervalos ligeiramente distintos. A diferença entre o intervalo de ativação especificado para o processo cíclico e o intervalo obtido é chamado de *jitter* de ativação. O *jitter* é resultado de diversos fatores, conforme ilustrado na figura 4.5.

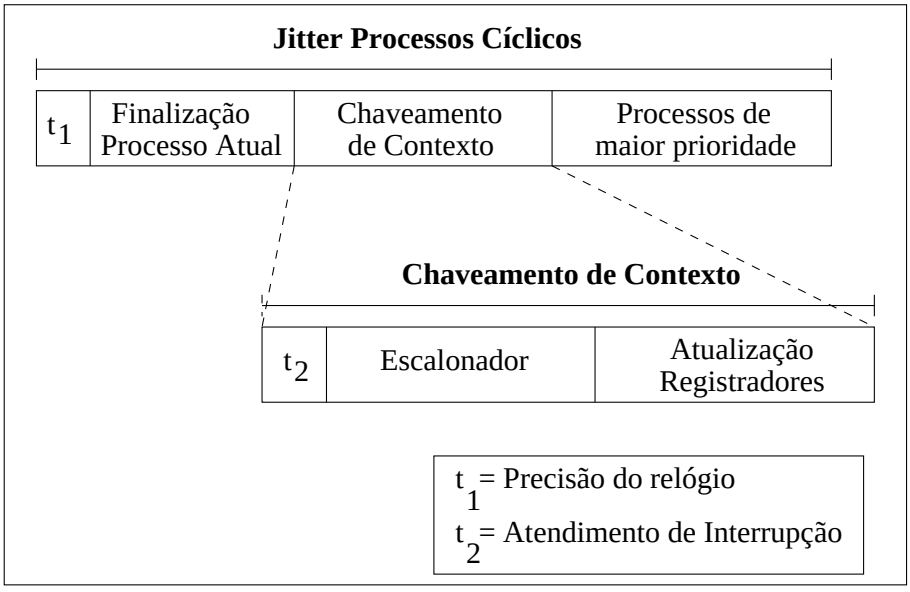


FIGURA 4.5 – *Jitter* dos Processos Cíclicos

O primeiro deles, difícil de ser removido, é causado pela imprecisão e pela granulosidade do relógio do sistema. Uma solução é utilizar relógios mais precisos,

os chamados relógios de tempo real. Mesmo estes têm uma precisão finita, e então alguma variação no intervalo de ativação ainda pode estar presente.

O *jitter* também é causado pela freqüência com que o processador é interrompido pelo relógio para contar a passagem de tempo, o que chamamos de *tick* de relógio. A cada tick, o processador é interrompido e a rotina de atendimento de relógio é executada, disparando os eventos programados para cada instante. O fato da contagem de tempo do processador ser discreta, e não contínua, gera uma variação nos tempos de ciclo, uma vez que o instante desejado para a ativação do processo provavelmente não irá coincidir com o instante de um *tick* do relógio do sistema. Outra forma de diminuir o *jitter*, portanto, é fazer com que o relógio do sistema gere ticks num intervalo de tempo menor. Porém, quanto menor o intervalo entre ticks, maior será a quantidade de interrupções de tempo que serão geradas e atendidas, fazendo com que o processador gaste muito tempo em tarefas de sistema operacional, como chaveamento de contexto e atendimento de interrupção, e menos tempo executando programas de usuário.

O *jiiter* é influenciado ainda pelo tempo gasto pelo escalonador, que deve determinar qual a tarefa cíclica a ser disparada no instante considerado. O tempo gasto pelo escalonador pode variar com fatores como a quantidade de processos no sistema, o número de processos prontos para ocuparem a CPU e as estruturas de dados internas ao escalonador. É preciso também esperar que a tarefa em execução no momento atinja um ponto onde ela possa ser preemptada, cedendo o processador à tarefa cíclica. Caso o momento possível de preempção esteja distante, isso irá implicar num *jitter* maior. Num sistema com escalonamento baseado em prioridades, a execução do processo cíclico ainda pode ser potergada até que nenhuma tarefa de maior prioridade esteja disputando o processador, contribuindo para um aumento ainda maior do *jitter*.

O *jitter* nunca pode ser removido sem uma fonte de referência temporal externa. A variação da constância temporal que afeta um sistema, seja ela positiva ou negativa, permanece indefinidamente até que um resincronismo seja feito através de uma outra fonte de informação de tempo de referência. Jitters subsequentes podem ser compensados, porém nunca removidos do sistema. O *jitter* é claramente uma métrica do tipo "menor melhor".

Métrica 4.9 (*Jitter* do processo cíclico) *é a variação do atraso de ativação observado nos processos periódicos ou cíclicos.*

4.3.2 Tempo de Chaveamento de Contexto

No sistema operacional, é importante o conhecimento do tempo gasto para realizar o chaveamento de contexto, ou seja, o tempo que o sistema operacional gasta para retirar o processo que está executando na CPU num determinado momento e colocar um novo processo para iniciar a execução. O tempo de chaveamento de contexto é importante por dois motivos. Caso o tempo de troca de contexto seja conhecido, é possível considerá-lo no período da tarefa cíclica e compensar o *jitter* causado pelo escalonador do sistema no instante de ativação de um ciclo. O tempo de chaveamento de contexto ainda é importante quando o escalonamento das tarefas do sistema de tempo real é calculado. Idealmente, os cálculos de escalonamento e os testes de escalonabilidade são feitos considerando-se a simplificação de tempo de chaveamento de contexto nulo. Porém, no caso de sistemas com uma grande carga

de ocupação do processador, com tarefas muito pequenas ou ainda onde a precisão deve ser grande, o tempo de troca de processos deve ser considerado nos cálculos do sistema. O tempo de chaveamento de contexto é também uma métrica do tipo "menor melhor".

Métrica 4.10 (Tempo de Chaveamento de Contexto) *é o intervalo de tempo decorrido entre a execução da última instrução anterior à preempção do processo que estava na CPU e a execução da primeira instrução do novo processo que foi colocado na CPU.*

4.3.3 Tempo para Criação de Processos

O tempo de criação de processos é outra medida importante no sistema de tempo real. O tempo gasto para a criação de uma nova tarefa no sistema pode influenciar negativamente no controle, compromentendo o atendimento de algum deadline. Ao se conhecer o tempo de criação de processos, pode-se decidir entre políticas de criação de tarefas no sistema de tempo real: em ambientes onde o tempo de criação de processos é crítico, possivelmente todas as tarefas serão criadas na inicialização do kernel, o que não causa sobrecarga durante a execução das tarefas. Em outros casos, quando o número máximo de tarefas ativas no sistema é restrito, a criação de tarefas pode ser feita dinamicamente, e mesmo as tarefas cíclicas podem ser criadas e destruídas uma vez por ciclo. Novamente, o tempo de criação de um processo pode influenciar no *jitter* e afetar o escalonamento e o cumprimento de todos os deadlines, e consequentemente deve também ser classificada como uma métrica do tipo "menor melhor".

4.3.4 Utilização de memória

Um sistema industrial é tipicamente um sistema embarcado, e é comum que ele inclua limitações de tamanho e custo de memória. Assim, é interessante que se possa avaliar a necessidade de memória de um determinado sistema operacional. Isso inclui o espaço de memória necessário para armazenar o próprio sistema operacional, além do espaço ocupado por cada processo, tanto pelo contexto quanto pelas áreas de trabalho de cada um deles. A forma de organização da memória de processos e sistema operacional (seção 4.2.4) é decisiva também no tempo gasto para o chaveamento de contexto. Uma troca de contexto que envolva apenas a atualização de um ponteiro para a área de descritor de outro processo será obviamente muito mais rápido do que a troca de contexto que inclua a cópia de conteúdos de memória, pilha, e tabelas de arquivos, por exemplo.

4.3.5 Latência de Interrupções

A latência de interrupção avalia a capacidade do sistema operacional de reagir a estímulos externos. A latência de interrupção mede o tempo gasto desde que uma interrupção foi gerada no processador até que a rotina de atendimento desta interrupção tenha começado a tratá-la.

A latência de interrupções é uma métrica "menor melhor". Uma baixa latência é importante em diversos aspectos. Uma baixa latência permite um tempo de reação curto à ocorrência de eventos no sistema, o que pode ser importante no caso de tratamento de exceções e na execução de terefas periódicas, diretamente influenciadas

pela latência no atendimento de interrupções do *timer*.

A latência de interrupções pode ser influenciada pelo programa de aplicação. Caso este desabilite o atendimento a interrupções por um período curto a fim de garantir que ele não será preemptado do processador, por exemplo, a latência será afetada. Da mesma forma, o próprio sistema operacional pode desabilitar as interrupções na execução de seções críticas, causando os mesmos efeitos indesejáveis. O ideal é que as interrupções fiquem desabilitadas pela mínima quantidade de tempo possível, de forma a afetar pouco a latência e principalmente a variação da latência, que é muito prejudicial no sistema de tempo real.

Métrica 4.11 (Latência de Interrupção) *é o intervalo de tempo decorrido desde a geração de uma interrupção no sistema até que a primeira instrução da rotina de atendimento da interrupção seja executada.*

4.4 Redes de Comunicação

A rede de comunicação exerce papel importante no processo de controle, uma vez que ela está presente nos diversos níveis da CPU. Existem arquiteturas de rede para comunicação com os dispositivos sensores e atuadores que estão no ambiente controlado, as chamadas redes de campo. As redes de supervisão e monitoramento também estão presentes, comunicando a CPU com os terminais de controle gerenciados pelos operadores do sistema. Por fim, ainda é possível conectar diversas CPUs através de uma rede dedicada para comunicação inter-CPU, de forma a permitir o processamento distribuído do controle industrial. Esta é a rede de controle. Portanto, o desempenho das redes de comunicação tem um papel decisivo no desempenho da CPU Industrial.

As métricas de redes são a parte mais abordada atualmente, onde é possível encontrar mais desenvolvimentos e grupos de trabalho envolvidos. No IETF, há o *Working Group* de *Benchmarking Methodology (bmwg)* dedicado exclusivamente à avaliação de desempenho de redes de computadores, de onde já foram geradas diversas RFCs. O foco principal deste grupo de trabalho do IETF são redes de uso geral [BRA 91],[BRA 99],[DUN 2001], sem preocupação especial com processos de controle ou comunicação de tempo real. Apesar de nenhuma delas abordar especificamente as características de determinismo e tempo real das redes, muitas idéias podem ser extraídas dos documentos disponíveis, como definições, nomenclaturas, métricas e técnicas para caracterização de desempenho. Existe ainda o *Working Group* denominado *IP Performance Metrics (ippm)*, onde estão sendo trabalhados outros documentos com definições de métricas específicas para caracterização de desempenho de redes IPs [PAX 98],[SHA 2001],[MAT 2001]. Novamente, estes também podem ser estudados e adequados para redes de controle industrial, onde atualmente a alternativa de utilização de Ethernet está em fase de estudos e definições [NAC 2000].

As métricas de rede são tanto quantitativas quanto qualitativas. Sob o ponto de vista de qualidades, é desejável conhecer principalmente o mecanismo de acesso ao meio. Os métodos clássicos de acesso ao meio físico incluem a técnica mestre-escravo, como usado no Profibus [PRO 99], por prioridades, como no CAN [CON 2000] ou mesmo por colisões, como é a política implementadas em redes Ethernet [GGH 2000].

Também é importante a caracterização do meio físico da rede, o que permite prever possíveis problemas com ruídos e interferências do ambiente.

As métricas quantitativas para caracterização das redes são diversas, e podem avaliar qualquer das três aplicações de redes em CPUs Industriais. Entre as mais importantes e representativas, pode-se listar a taxa nominal de transmissão de dados, o tempo de transmissão de dados, o tempo para acesso ao meio físico, que pode variar com os diversas políticas de acesso ao meio e mesmo serem inconstantes, e a eficiência dos protocolos utilizados, que representa a relação entre a quantidade de bytes de controle necessários para possibilitar a transmissão de um conjunto de bytes de dados efetivos.

4.4.1 Taxa Nominal de Comunicação de Dados

A Taxa nominal de comunicação de dados apresenta a velocidade máxima da rede de comunicação, em bytes por segundo. Ela mede a capacidade bruta de transmissão e recepção de dados pela rede, sem que nenhum pacote seja perdido. A taxa nominal é um parâmetro físico, que pode ser calculada analiticamente, mas que nem sempre pode ser atingida. Mesmo assim, é importante que seja levantada, e é uma métrica "maior melhor".

Métrica 4.12 (Taxa Nominal) *é a velocidade máxima de transmissão de dados pela rede, determinada pelas características físicas e de enlace do meio de transmissão.*

4.4.2 Tempo de Transmissão

O Tempo de Transmissão está diretamente relacionado com a velocidade da rede, e apresenta uma estimativa do tempo a ser gasto para enviar uma quantidade de bytes pela rede. Quanto menor o tempo de transmissão, melhor o desempenho da rede, e por conseqüência, do processo de controle e CPU Industrial.

Métrica 4.13 (Tempo de Transmissão) *é o intervalo de tempo desde que o primeiro bit do pacote foi colocado no meio físico da rede até que o último bit tenha sido retirado do meio físico*

4.4.3 Tempo de Acesso à Rede

O Tempo de Acesso à Rede é outra métrica importante. Não apenas o tempo gasto com a transmissão de dados no sistema deve ser considerado, mas também o tempo que cada dispositivo deve aguardar até que ele possa acessar o meio físico e inciar a transmissão dos dados deve ser estimado. Aqui, o importante é a constância do tempo de acesso, mas mesmo assim é possível classificar esta métrica como sendo do tipo "menor melhor".

O método de cálculo do tempo de acesso à rede varia conforme a política de controle de acesso ao meio. Num mecanismo com multiplexação temporal, onde cada intervalo de tempo é destinado a uma estação, o tempo de acesso ao meio é constante e depende apenas do número de estações na rede e do tempo que cada uma pode acessar a rede de forma contínua. Porém, casos com resolução de acesso por meio de deteção de colisão, como ocorre na rede Ethernet, são altamente não determinísticos, e só é possível ter uma estimativa aproximada do tempo médio de

espera para acesso ao meio físico. Numa rede com ordenamento por prioridades, o tempo de espera depende das atividades dos demais processos que também querem acessar a rede e da relação entre as suas prioridades. Já numa rede mestre/escravo, não há tempo de espera para acessar a rede, uma vez que ela está sempre disponível para o mestre ou para o escravo que deve enviar uma resposta à requisição anterior do mestre.

Métrica 4.14 (Tempo de Acesso à Rede) *é o intervalo de tempo decorrido entre o momento em que a estação está com os dados prontos para serem transmitidos até que o primeiro bit do pacote a ser transmitido seja colocado no meio físico da rede.*

4.4.4 Eficiência do protocolo de campo

A eficiência dos protocolos é uma medida importante, por dar uma noção da real utilização da rede para envio de dados efetivos, além dos dados de controle necessários pelos protocolos de comunicação. O ideal é que a eficiência de um protocolo seja próxima de 1, ou 100%, o que implica que nenhum dado de controle seria necessário para a comunicação entre as estações.

A eficiência do protocolo de campo pode ser medida de duas maneiras. A primeira delas leva em consideração a quantidade de dados que circula pela rede e a quantidade de dados úteis efetivamente obtidos durante uma comunicação. A equação 4.1 ilustra esse conceito.

$$E = \frac{dados\_úteis}{dados\_úteis + controle} \tag{4.1}$$

A eficiência do protocolo de campo também pode ser medida em relação ao tempo gasto para buscar as informações nos dispositivos escravos do campo. A equação 4.2 utiliza os tempos de processamento do escravo e o tempo de transmissão da rede para calcular uma eficiência para o protocolo de campo.

$$E = \frac{bytes\_lidos}{tempo\_proc\_escravo + tempo\_transmissão} \tag{4.2}$$

Na equação 4.1, tem-se uma medida de eficiência em percentual de bytes úteis. Na equação 4.2, a eficiência é apresentada em bytes por segundo.

4.4.5 A Rede de Supervisão

No aspecto de redes de comunicação, também é relevante o desempenho e a eficiência dos protocolos e da arquitetura de supervisão. A política de atendimento de transações de supervisão é decisivo tanto para os processos de supervisão quanto para o comportamento do processo de controle. Caso o processamento da lógica de controle e das requisições de supervisão sejam feitas pelo mesmo processador, uma sobrecarga de supervisão pode comprometer a constância do controle.

Do ponto de vista de supervisão, as métricas avaliam a política de atendimento de requisições e também o tempo gasto com cada uma delas. O número de atendimento de requisições por ciclo de controle apresenta uma estatística que permite avaliar a influência das requisições no tempo de ciclo. Um valor constante para esta métrica indica que as requisições de supervisão podem afetar o tempo de ciclo até

um limite máximo. Já o tempo gasto para o atendimento de uma requisição sugere a influência em termos de poder de processamento que é dedicada às requisições de supervisão. Com o tempo de atendimento de uma requisição e o número de requisições atendidas por ciclo, é possível estimar qual a variação máxima do tempo de ciclo devido à influência da rede de supervisão.

Métrica 4.15 (Número de requisições atendidas por ciclo) *é o número máximo de requisições de supervisão atendidas em um único ciclo de controle da CPU.*

Métrica 4.16 (Tempo para atendimento de requisição) *é o intervalo de tempo decorrido entre o recebimento de uma requisição de supervisão pela CPU e o aparecimento da resposta na rede.*

4.4.6 Rede de Controle

A rede de controle é a terceira componente da arquitetura de redes da CPU Industrial. Ela é utilizada para a comunicação entre CPUs, no caso da implementação de um sistema de redundância ou de um sistema de controle distribuído, onde as CPUs precisam se comunicar ou colaborar entre si.

Na rede de controle, pode-se avaliar as mesmas métricas gerais apresentadas anteriormente, bem como as métricas da rede de supervisão. A ressalva importante a ser feita é que as comunicações entre CPUs, através da rede de controle, deveriam ter uma prioridade maior de tratamento quando comparadas com as comunicações entre a CPU Industrial e o ambiente de supervisão.

4.4.7 Outras métricas

Um conjunto grande de métricas foi estabelecido pelo IETF nas RFCs 1242 [BRA 91] e 2544 [BRA 99] para avaliação de desempenho de dispositivos de interconexão de redes. CPUs Industriais não exercem a função de interconexão de redes, porém algumas métricas apresentadas nestes documentos são relevantes e podem ser estendidas para as aplicações de CPUs Industriais. As principais métricas estão detalhadas a seguir.

Métrica 4.17 (*Back-to-Back*) *é o tempo máximo que a CPU consegue processar o recebimento de requisições com o mínimo intervalo de tempo entre as requisições. O mínimo intervalo de tempo é determinado pelo meio físico e pelas definições da rede.*

Métrica 4.18 (Taxa de Perda de Pacotes) *representa o percentual de pacotes perdidos pelo equipamento devido à falta de capacidade de memória e processamento do mesmo quando este opera com uma determinada taxa de chegada de pacotes.*

Métrica 4.19 (Vazão) *é a velocidade máxima em bytes por segundo que o equipamento consegue processar requisições e enviar respostas sem que nenhuma requisição seja perdida por falta de recursos de memória ou processamento.*

4.5 Compilador e Ambiente de Programação

O compilador e o ambiente de programação exercem um papel importante no desempenho do sistema. Apesar de não ser utilizado efetivamente durante o funcionamento da CPU, um bom compilador pode gerar códigos mais eficientes e que explorem de forma adequada os recursos disponíveis no sistema. Da mesma forma, um bom ambiente de programação reduz a possibilidade de erros do programador, principalmente aqueles relacionados às dificuldades de utilização do ambiente de trabalho.

A avaliação do compilador e do ambiente de programação é feita com base em elementos qualitativos, muito mais que em pontos quantitativos. Neste caso, está-se interessado em levantar as características, funcionalidades e recursos disponíveis nos dois componentes, e não exatamente quanto tempo ele leva para executar e gerar um código pronto para ser utilizado.

Sob o ponto de vista do compilador, as métricas importantes reportam a capacidade do compilador em gerar código adequado ao processador em questão e conseguir explorar todos os recursos de *hardware* disponíveis de forma eficiente, além de executar otimizações no código de controle sem afetar o comportamento temporal da aplicação, o que é uma tarefa não trivial. O tempo gasto efetivamente para compilar a lógica de controle exerce papel secundário neste caso. Pode-se esperar um tempo pela compilação, desde que ele não seja exageradamente grande e que produza um código eficiente.

No ambiente de programação, as métricas qualitativas estão ligadas às facilidades de especificação do código e depuração do sistema. Devem ser avaliadas características como facilidades de depuração, possibilidade de simulação do sistema, sem que o *hardware* esteja fisicamente conectado ao ambiente e recursos de depuração diretamente no equipamento. Interessam ainda pontos como quantidades de linguagens suportadas, adequação do sistema às normas e padrões de programação para CPUs Industriais [IEC 93] e mecanismos para geração do código final a ser executado na CPU, que pode ser uma linguagem intermediária a ser interpretada ou diretamente código objeto nativo do processador integrado ao sistema operacional.

4.6 Arquitetura de Software

A avaliação de uma arquitetura de *software* é uma tarefa difícil e nem sempre pode ser feita apenas através de métricas quantitativas. Assim como o compilador do sistema, a arquitetura de*software* também é medida em termos qualitativos, além dos termos quantitativos.

Sob o ponto de vista de números, as métricas apresentadas até aqui avaliam a arquitetura de *software* desenvolvida, uma vez que as métricas levantam pontos como quantidade de memória ocupada por uma solução, quantidade de processos utilizados, necessidade e ocupação de outros recursos do sistema, como canais de comunicação, e barramentos, tempos de execução de programas, tempo de leitura e escrita de entradas e saídas e eficiência de comunicação. É possível também avaliar a forma de implementação de alguns módulos de são presença obrigatória numa arquitetura de *software* de uma CPU Industrial. Em qualquer implementação, haverá uma região de memória destinada a servir como imagem do ambiente controlado, refletindo o estado das entradas e saídas, que é chamado de Base de Dados ou *Ima-*

gem de I/O. Pontos como quantidade de memória gasta por ponto de entrada ou saída e velocidade de acesso à imagem podem ser considerados. É difícil afirmar quais os valores ideais para esses elementos, mas tipicamente quer-se que a base de dados seja tão rápida quanto possível, gastando o mínimo de memória necessário.

Por outro lado, as métricas qualitativas são mais sensíveis ao usuário e aos gerentes, projetistas e desenvolvedores da arquitetura. As principais métricas estão preocupadas com a implementação do algoritmo de controle, que deve atender plenamente aos requisitos estabelecidos na especificação do sistema, através das métricas especificadas para o processo de controle. É importante também avaliar a facilidade (ou dificuldade) de realizar manutenções periódicas no *software*, principalmente quando o programador principal de um determinado módulo não estiver mais disponível para executar esta tarefa. Neste ponto, fatores como modularização do sistema e boa documentação são essenciais.

4.7 Considerações Finais

Este capítulo apresentou as métricas a serem utilizadas na caracterização de desempenho de CPUs Industriais. O objetivo é caracterizar o desempenho da CPU sob o ponto de vista da aplicação de controle, de forma que se possa avaliar o desempenho do equipamento em sua utilização típica.

O capítulo apresentou também um conjunto de métricas que afetam os resultados das métricas principais do processo de controle, e estas estão relacionadas a aspectos de *hardware*, sistema operacional, redes, compiladores e arquitetura de *software*. Para facilitar a visualização da relação entre as métricas e também permitir a análise de potenciais otimizações na implementação da CPU Industrial, a tabela 4.1 apresenta a relação das métricas entre si, mostrando o quanto cada uma das métricas pode afetar o resultados das medições das métricas finais de controle.

A tabela 4.1 mostra que o tempo de ciclo (métrica 4.1) tem uma alta dependência em relação ao tempo de execução de cada instrução. O tempo de ciclo apresenta também uma influência média do *jitter* de ativação dos processos cíclicos (métrica 4.9). Esta é uma conclusão bastante direta, já que o tempo de ciclo é composto basicamente pelo tempo gasto para executar o programa de controle (que nada mais é que o número de instruções vezes o tempo de cada instrução). Além disso, o tempo de ciclo é definido como o tempo decorrido entre duas execuções consecutivas de uma instrução, e portanto o *jitter* para ativação de processos cíclicos tem ainda uma influência média no tempo de ciclo. Isso ocorre pois o atraso para ativação do processo de controle faz parte do tempo de ciclo: enquanto o processo cíclico não é reiniciado, o ciclo de controle anterior não foi terminado, caso tome-se por referência para o cálculo do tempo de ciclo a repetição da primeira instrução de controle, por exemplo. De forma análoga, qualquer que for a instrução escolhida como marco do fim do ciclo implicará na inclusão do atraso de ativação do processo. Porém, o atraso é um valor pequeno, e por isso a influência no tempo de ciclo não é tão significativa quanto o tempo de execução de instruções. Por fim, duas métricas que exercem uma influência significativa no tempo de ciclo são o tempo de varredura e o tempo de atualização. As demais métricas não exercem uma influência significativa no tempo de ciclo, e por isso não apresentam nenhuma caracterização neste caso.

O tempo de programa (métrica 4.2), por sua vez, também tem uma grande

influência do tempo de execução de instruções. Está é uma relação clara e direta, já que o programa de controle é composto apenas por instruções básicas de controle. No caso do tempo de programa, ao contrário do tempo de ciclo, o *jitter* dos processos cíclicos não exerce influência significativa. Isso ocorre pois o tempo de programa é definido apenas como o espaço de tempo entre a execução da primeira e da última instrução do programa de controle, independente do atraso experimentado para iniciar a execução do programa. Entretanto, no tempo de programa é possível notar uma influência do tempo de chaveamento de contexto. No caso de uma implementação do tipo executivo cíclico, esta componente é nula. Porém, no caso de uma implementação que utliza um núcleo de tempo real multitarefa, o tempo de chaveamento de contexto afeta o tempo total de programa, ainda que essa influência seja baixa, já que pode haver trocas de tarefas em execução durante o ciclo de controle, por exemplo para tratamento de interrupções da porta serial ou da rede de campo.

Os tempos de varredura (métrica 4.5) e atualização (métrica 4.7) apresentam um comportamento bastante semelhante. Estas métricas são afetadas em grande parte pelo desempenho da rede de campo. Elas apresentam uma relação forte com a vazão e eficiência da rede, e ainda uma relação média com o tempo de acesso à rede e a taxa de perda de pacotes. Por fim, há uma pequena componente destas métricas que são relacionadas com a velocidade do barramento interno de comunicação, já que é necessário transferir os dados lidos ou escritos do processador para os dispositivos de ligação com a rede de campo.

Por fim, o tempo de reação (métrica 4.4) é o que mais sofre influência das outras métricas. O tempo de reação é uma métrica redundante, que representa uma composição dos tempos de ciclo, varredura e atualização. Portanto, é afetada por todas as métricas que afetam estas últimas.

TABELA 4.1 – Relação entre métricas

	Tempo de Ciclo	Tempo de Programa	Tempo de Bit	Tempo de Reação	Tempo de Varredura	Tempo de Atualização
Tempo de Programa				alto		
Tempo de Varredura	alto			alto		
Tempo de Atualização	alto			alto		
Barramento Interno				baixo	baixo	baixo
Tempo de Execução de Instruções	alto	alto	alto			
Jitter de Processos Periódicos	médio			médio		
Chaveamento de Contexto		baixo	baixo			
Criação de Processos						
Latência de Interrupções				baixo		
Tempo de Transmissão/Vazão Máxima				médio	alto	alto
Tempo de Acesso à Rede				baixo	médio	médio
Eficiência do Protocolo de Campo				médio	alto	alto
Requisições Atendidas por Ciclo						
Tempo de Atendimento de Requisição						
Taxa de Perda de Pacotes					médio	médio

5 Workloads e Monitoração

O planejamento das cargas de medida, ou *workloads*, exercem um papel tão ou até mais importante do que as próprias métricas que estão sendo medidas. Um bom conjunto de métricas não representa nada caso a carga a que o sistema estava submetido no momento da medida não foi bem planejada ou, pior, é desconhecida. Assim como o planejamento das cargas utilizadas na medição, a especificação das técnicas e métodos a serem utilizados para realizar a extração das métricas também faz parte do processo de avaliação de desempenho.

Este capítulo apresenta um conjunto de *workloads* e técnicas de monitoração que pode ser utilizado na avaliação de sistemas industriais, tanto os baseados em componentes de prateleira quanto os totalmente desenvolvidos dentro de um mesmo projeto. Por querer servir para qualquer tipo de equipamento industrial, todas as técnicas apresentadas neste capítulo envolvem a utilização de medição direta como técnica de avaliação. Apesar de ser a mais cara de todas e necessitar de um protótipo do sistema para poder ser executada, a técnica de medição direta é a única alternativa disponível quando não é possível ter acesso a detalhes de implementação dos componentes, o que é comum nos componentes de prateleira utilizados em projetos de integração. Está-se interessado em caracterizar principalmente as métricas relacionadas ao processo de controle, que são as mais sensíveis ao usuário, e por isso os *workloads* desenvolvidos serão concentrados no nível de aplicação, ou controle.

Portanto, a medição e avaliação de desempenho apresentados neste trabalho consistirão no desenvolvimento de um *benchmark* para caracterização de desempenho de CPUs Industriais. A carga a que será submetido o sistema durante o processo de *benchmarking* está apresentada na seção 5.1, enquanto as técnicas para monitoração estão nas seções 5.2 e 5.3.

O benchmark apresentado será composto por programas em nível de aplicação, uma vez que se propõe a avaliar qualquer CPU Industrial e apresentar resultados que possam ser comparáveis entre si. Com programas em linguagens de alto nível, é possível eliminar possíveis distorções que poderiam ser causadas por otimizações de baixo nível no momento do porte do *workload* para as diversas plataformas alvo. Além disso, é desejável que o próprio utilizador do sistema possa executar o procedimento de avaliação, e neste caso o único recurso que está disponível a ele é a utilização de programas de controle para executar as medidas. O usuário final da CPU Industrial não consegue adicionar rotinas e modificações nos demais componentes internos do sistema, e por isso os *workloads* são baseados apenas na utilização de recursos disponíveis para programas de controle. Um *workload* especificado em linguagem de alto nível inclui ainda a vantagem de avaliar não apenas o desempenho do equipamento, mas também a capacidade do compilador e as diferenças de arquitetura do sistema, que são considerados automaticamente.

Por fim, os workloads serão compostos por programas de controle pois assim é possível refletir com maior precisão o comportamento que será obtido para cada instrução na utilização real do sistema. Ao contruir um workload com programas de controle, não há interferência nas rotinas internas da CPU, e assim a intrusão do método de medida nos resultados é a menor possível.

Em um sistema de medição direta, a minimização da intrusão dos sistemas de monitoração deve ser um objetivo básico, principalmente em sistemas com requisi-

tos temporais. Num sistema de controle de tempo real, esta é uma característica importante, já que uma intrusão para avaliação de desempenho poderia significar mudanças no comportamento temporal da aplicação e, por conseqüência, uma invalidação dos resultados obtidos durante a caracterização do sistema.

A construção dos *workloads* e especificação dos monitores para as CPUs Industriais deve explorar os recursos disponíveis na mesma, aproveitando as particularidades do método de funcionamento do ciclo de controle, como ilustrado na figura 3.2. A execução em ciclos da CPU Industrial auxilia, permitindo várias execuções consecutivas do *workload*, uma em cada ciclo. Porém, outras características da CPU podem dificultar o estabelecimento de um programa de avaliação e comparação entre CPUs. Muitas delas apresentam uma limitação de segurança no tempo máximo de ciclo, como forma de prevenir erros que possam comprometer o processo de controle ou erros causados por falhas que levem a um congelamento da CPU. Um *watchdog* é disparado nestes casos, causando a reinicialização do equipamento. Portanto, o tempo máximo de execução de um programa de controle para extração de métricas deve se limitar ao tempo máximo de ciclo estabelecido para a CPU, caso este exista.

5.1 *Workloads*

O *workload* a ser desenvolvido para esta avaliação está preocupado em extrair informações a respeito do funcionamento da CPU Industrial como um processador de controle industrial, e deve ser capaz de avaliar o desempenho do sistema inteiro, incluindo-se o processador propriamente dito e os dispositivos de entrada e saída, bem como a rede de comunicação entre CPUs Industriais e equipamentos de campo. Portanto, o *workload* não será desenvolvido para comparar apenas o processador ou o desempenho do sistema operacional, muito menos dependerá da existência ou não de um eventual processador dedicado para execução de programas de controle especificados em lógicas de relés. Certamente, este e outros recursos terão um impacto nos resultados da avaliação, porém não estarão sendo avaliados individualmente, e sua existência não será pré-requisito para a execução do *workload*.

O *workload* trata a CPU Industrial como um todo, e esta será vista como um "sistema de execução de ciclos de controle", onde estão incluidos os recursos para execução de programas, leitura e escrita de dispositivos do campo e ainda os mecanismos de comunicação entre as CPUs. O *workload* desenvolvido pode ser dividido logicamente em duas partes. Na primeira parte, procura-se estressar cada tipo de instrução individualmente, executando uma grande quantidade da mesma instrução em sequência e verificando o tempo gasto para cada tipo de instrução. O ciclo de controle em aplicações reais é composto por uma mistura das instruções destes diversos grupos, com uma distribuição diferente em cada aplicação de controle específica. Assim, avaliando-se o tempo de cada grupo de instrução é possível obter uma estimativa inicial do equipamento como um "executor de ciclos de controle".

É difícil estabelecer um programa único que aborde todas as situações onde a CPU Industrial é utilizada. A grande gama de aplicações onde uma CPU Industrial é utilizada faz com que as estruturas e características de cada programa desenvolvido seja única e distinta, e portanto encontrar uma seqüência de instruções que contemple essa variedade de programas de controle é uma tarefa ingrata. Devido a essa diversidade de seqüências, freqüências e organização de instruções, um progra-

ma único ou um conjunto de dois ou três programas reais que reportem o tempo gasto para executar cada um deles terão pouca aplicação para seus resultados.

Por outro lado, construir um programa sintético instrumentado que seja capaz de relatar o tempo gasto para execução de cada grupo de instruções (aritméticas, lógicas, contatos/relés, movimentação de operandos, etc ...) é mais recomendado e pode trazer resultados aplicáveis a praticamente todos os programas reais. Com informações a respeito das características de cada instrução, é possível avaliar diversos parâmetros no sistema:

- cálculo do tempo aproximado de programa para cada aplicação, dado que o tempo de cada instrução é conhecido e também a estrutura e freqüência de ocorrência das instruções é facilmente determinável.

- comparação entre várias CPUs Industriais, comparando-as em relação ao tempo de execução das instruções e possibilitando que a CPU mais adequada a uma determinada aplicação prática seja escolhida com mais facilidade e mais possibilidade de acertos, já que não estarão sendo comparados números genéricos a respeito do tempo de execução de um conjunto específico de programas e sim o tempo de execução de cada instrução individual.

- pela repetibilidade do *workload* e com a execução de várias rodadas de experimento, é possível determinar a variabilidade dos tempos de instrução e verificar o determinismo temporal de cada implementação de CPU Industrial.

É nesta direção que será desenvolvido o *workload* a ser utilizado neste processo de caracterização de desempenho de CPUs Industriais. A construção do *workload* foi feita com base na observação de uma amostra de programas e módulos de programa de controle, que tentavam caracterizar as estruturas e instruções mais frequentes neste tipo de aplicação. A seção 5.1.1 apresenta a análise das instruções, e em seguida o programa de teste de desempenho proposto será detalhado.

5.1.1 Análise de Programas Típicos

Com a finalidade de observar a construção dos programas de controle e a distribuição das instruções nos mesmos, foram selecionados um conjunto de programas a partir da base de projetos da empresa Altus, especializada no desenvolvimento de CPUs Industriais e de projetos de engenharia baseados neste tipo de equipamento. Foram selecionados 5 projetos, escolhidos ao acaso como forma de evitar algum vício na amostra. A tabela 5.1 mostra as características de alguns módulos dos projetos observados.

O primeiro programa observado foi projetado para a CPU AL2003. Ele é composto por 103 módulos de programa, sendo um módulo de configuração (módulo I-C000), um de inicialização (módulo I-E000), um de execução (módulo I-E001), seis módulos que implementam funções em assembly (módulos F) e 94 módulos de procedimentos de controle (módulos I-Pxxx) especificados em linguagem de diagrama de relés. Destes 94 módulos de procedimento, 70 deles eram estruturados da mesma forma, com as mesmas instruções, e que modificavam apenas as entradas e saídas manipuladas pelo módulo. Um destes 70 módulos é o módulo I-P006, que tem sua estrutura apresentada na tabela 5.1.

A segunda aplicação analisada também foi projetada para a CPU AL2003, e é composto por 41 módulos de programa. Destes, um módulo é para configuração do equipamento (II-C000), um de inicialização (II-E000), um módulo principal de execução (II-E001), 13 módulos de função escritos em relés e assembly e 25 módulos de procedimento. Dos módulos de procedimento, os módulos II-P000 a II-P004 são estruturados da mesma forma, assim como os módulos II-P030 a II-P032 também variam apenas nos operandos das instruções, que são iguais e organizadas da mesma maneira.

A próxima aplicação observada é composta por 12 módulos, 7 deles de procedimentos de controle. A aplicação foi desenvolvida para a CPU PL104, de menor capacidade de memória e processamento que a AL2003, alvo das duas primeiras observações.

A quarta aplicação executa na CPU AL2003, e é composta por 1 módulo de configuração, 3 de execução (inicialização, módulo principal e timer), 5 módulos de função e 12 módulos de procedimento. Por fim, a última aplicação que foi tomada como exemplo para análise foi desenvolvida para a CPU AL2002, de capacidade de memória e processamento um pouco inferior que a AL2003, composta por 13 módulos no total.

TABELA 5.1 – Características dos Programas

Programa	Módulo	# lógicas	RELES	CHP	MOV	ADD	OR	TEE
	I-E001	36	41	91	118			
I	I-P108	70	330		29	26		9
	I-P006	18	125		3	2		2
	I-P092	38	270		23			4
	II-E001	9	14	22				1
II	II-F032	7		3	4	5		
	II-P031	19	33	14	10		13	2
	II-P040	9	34	2	2			
	III-E001	4	3	7				
	III-P003	31	116		32	14	42	
III	III-P005	3	6		9			
	III-P002	4	2		4		6	
	III-P004	28			8	136		
	III-P007	4	5		6	4		
	IV-E001	7	20	11				1
IV	IV-P001	15	95					7
	IV-P002	10	60					8
	IV-P099	25	73	1	26	6		5
	V-E001	27	87	12	24			2
V	V-P005	46	184					
	V-P004	13	66		3			1
	V-P006	7	29		15			

A partir da observação da tabela 5.1, é possível verificar as seguintes características dos programas de controle especificados em linguagens de relés para as CPUs Industriais da Altus:

- os programas são divididos em *módulos*, e cada módulo é composto por uma quantidade variável de *lógicas*. Cada lógica é especificada por um conjunto de diferentes instruções de controle. Fazendo uma analogia com linguagens genéricas de alto nível, pode-se associar um módulo de controle a um procedimento em linguagem de alto nível, e cada lógica é equivalente a um bloco de instruções.

- boa parte das lógicas de um módulo é composta por apenas um tipo de instrução. Ao mesmo tempo, as lógicas que apresentam mais de um tipo de instrução de controle raramente excedem o total de três tipos distintos de instruções.

- cada módulo é responsável pela execução de uma parte específica do controle industrial, e as lógicas que compõem um módulo são bastante parecidas entre si.

- alguns programas apresentam vários módulos iguais entre si, onde a única variação está nos operandos das instruções. Esta característica é comum onde o ambiente controlado é composto por diversos subsistemas físicos similares, e uma única CPU Industrial controla todo o ambiente. Desta forma, cada módulo é desenvolvido para controlar um dos subsistemas, e assim vários módulos apresentam praticamente a mesma estrutura.

- a grande maioria das instruções encontradas em uma lógica são instruções de contatos (teste de operandos do tipo bit) e bobinas (escritas de operandos do tipo bit). Praticamente todos os módulos observados apresentam grande quantidade de instruções deste grupo.

- instruções do tipo movimentação de operandos também são bastante utilizadas, servindo para inicialização de estruturas de dados e para atribuição a variáveis.

- instruções para chamadas de procedimento e função são basicamente utilizados no módulo principal do programa de controle, o equivalente ao método `main()` em um programa em linguagem C.

- é muito difícil identificar, mesmo em dois módulos diferentes de um mesmo programa, uma seqüência de instruções comum. Ou seja, cada aplicação exige um desenvolvimento de aplicações novas, onde existe muito pouco reuso de módulos e estruturas anteriormente construídos.

Ao observar as características dos programas considerados, confirmou-se a idéia que a construção de um *workload* sintético é realmente a alternativa mais adequada, e isto não irá comprometer a qualidade e utilidade dos resultados. Apesar de não terem muitos trechos de instruções e mesmo organização comuns, programas reais compartilham a importante característica de lógicas que contêm instruções de uma única classe e ainda a alta quantidade de instruções de contato e bobina. Estas particularidades podem ser facilmente reproduzidas em *workloads* sintéticos, e a avaliação do tempo de execução de cada instrução pode contribuir significativamente na caracterização do desempenho de uma CPU Industrial.

Nota-se facilmente que a estrutura de programação de CPUs Industriais em linguagens de relés segue um modelo comum em linguagens de alto nível de alguns

anos atrás, como ALGOL e FORTRAN. Isso se deve ao fato de que a linguagem é antiga, tendo sido desenvolvida no final dos anos 60, e desde lá não passou por grandes modificações estruturais. Assim, é aceitável imaginar que as técnicas para programação e estruturação de código também não tenham evoluído muito.

Muitos esforços têm sido aplicados com o objetivo de atualizar os sistemas de programação, as linguagens e a estruturação dos programas de controle, através da norma de linguagens de programação IEC61131 [IEC 93]. Entretanto, não tem-se observado uma adesão significativa às novas técnicas de organização e linguagens [BON 97]. Portanto, a estruturação do benchmark de forma similar aos clássicos Whetstone [CUR 76] e Dhrystone [WEI 84] é a alternativa mais simples e direta para realizar o levantamento das métricas propostas.

5.1.2 O Programa de Avaliação

A partir da observação das características dos programas exemplo da seção 5.1.1, foi desenvolvido um programa capaz de avaliar e caracterizar o desempenho de CPUs Industriais. O programa deve ser baseado na execução de instruções de controle típicas, e deve ser capaz de ser repetido por um número indefinido de vezes.

Uma das conclusões importantes a que se chegou a partir da observação dos exemplos é que boa parte das lógicas de um módulo de controle é composta por apenas uma única classe de instruções, e na maior parte dos casos estas são instruções do grupo de relés e bobinas. Assim, foram desenvolvidas lógicas compostas exclusivamente por instruções deste grupo, com o objetivo de verificar o tempo gasto e o determinismo temporal na execução deste tipo de instrução. Outros grupos importantes de instruções são comuns em módulos de controle, como instruções de movimentação, soma/subtração e chamada de outros módulos de função e procedimentos, e estes também foram contemplados com lógicas dedicadas para avaliação do comportamento de cada uma. Portanto, a primeira parte do *workload* é composta por lógicas construídas com uma instrução única. São contempladas as seguintes instruções:

- Contatos e bobinas;
- Soma e subtração;
- Operações “E” e “OU” lógico;
- Movimentação de operandos;
- Chamada de funções e procedimentos

No caso das instruções de contatos e bobinas, são propostos dois casos de teste de desempenho. No primeiro deles, a lógica de contatos e bobinas é preenchida com o número máximo de instruções suportada pelo ambiente de programação utilizado nos estudos de caso, que limita em 32 contatos o total de instruções em uma única lógica. No segundo caso de teste, preencheu-se a lógica com o número mínimo de contatos para uma lógica da linguagem, composta por duas instruções: um contato como entrada e uma bobina como saída. Assim, é possível avaliar a influência dos espaços vazios no desempenho das instruções, bem como verificar, através de técnicas de regressão linear, se existe algum tempo necessário para inicialização da

lógica antes dela ser executada. Com os dois resultados ($T1$ e $T2$), é possível resolver o sistema apresentado na equação 5.1 e determinar o tempo real de uma instrução de contato (t_c) e o tempo de inicialização da lógica (t_{init}).

$$\begin{cases} 32t_c + t_{init} = T1 \\ 2t_c + t_{init} = T2 \end{cases} \tag{5.1}$$

Ao se utilizar lógicas com um único grupo de instruções, é possível verificar também o tempo de execução de instrução de cada um destes grupos. Caso seja possível obter o tempo gasto para a execução de uma lógica, basta dividir este tempo pelo total de instruções da lógica e chegar a um valor bastante aproximado do tempo real de execução de cada instrução.

A figura 5.1 ilustra um trecho do programa proposto. No trecho da figura, é caracterizado o tempo gasto para executar uma lógica com 32 instruções de contato/bobina. A lógica I é a responsável pela inicialização deste trecho de controle. É zerado o contador de repetições e capturado o instante do relógio. Na lógica II, é rodada a carga de interesse propriamente dita. No caso da figura 5.1, o que está sendo medido é o tempo para execução de uma lógica com 32 instruções de contato. A lógica III é responsável pelo laço de controle, desviando a execução novamente para a lógica II caso o número desejado de repetições ainda não tenha sido atingido. Por fim, a lógica IV só é atingida ao final de todas as repetições da lógica II, e ela registra a diferença entre o tempo de início do laço e o tempo atual, computando o tempo gasto para executar as repetições da lógica principal e o laço de controle.

Pequenas variações são feitas na seqüência da figura 5.1, para permitir que outras instruções sejam também testadas. A única diferença é a substituição da lógica II por outra lógica com o conjunto de instruções que se quer observar, por exemplo instruções de soma ou instruções de movimentação de operandos. O anexo A apresenta o programa completo proposto para avaliar as CPUs Industriais.

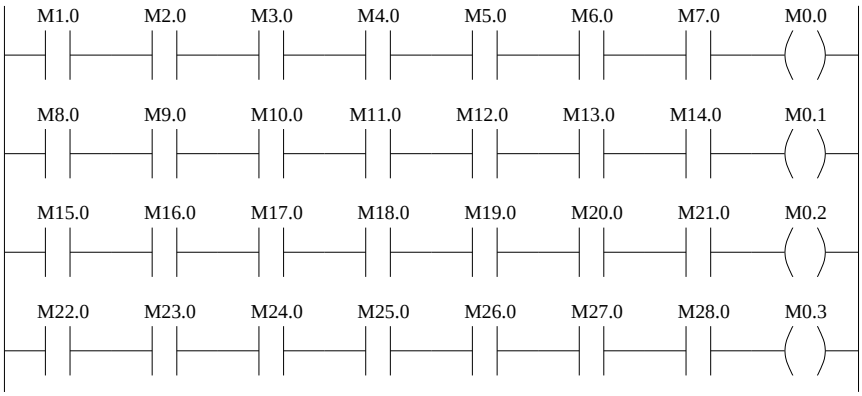
Para tentar dissolver os erros de medida devido à precisão dos monitores, as lógicas são executadas diversas vezes a cada rodada de avaliação. Com isso, tem-se um tempo significativo para medir, e provavelmente as imprecisões do relógio serão pouco representativas em comparação ao tempo total da execução. Porém, este procedimento resulta em um conjunto extra de instruções para controlar o número de vezes que cada lógica já foi executada, e acaba causando uma intrusão indesejada nos resultados.

Em face a esse problema, o *workload* tem ainda uma etapa de calibração, onde é avaliado o tempo gasto na execução das instruções do laço de controle para repetição da lógica que efetivamente possui as instruções que se deseja medir. A calibração é feita executando um laço sem nenhuma instrução, e o tempo para executar este laço é medido da mesma forma que é feito com as demais lógicas. No exemplo da figura 5.1, equivale à substituição da lógica II por uma lógica vazia. Este tempo é posteriormente descontado nos resultados finais obtidos em cada parte do *benchmark*. Além disso, o próprio tempo gasto com o laço de controle também pode ser usado como resultado do benchmark, já que é composto por um *mix* de instruções comum em sistemas de controle baseados em CPUs Industriais.

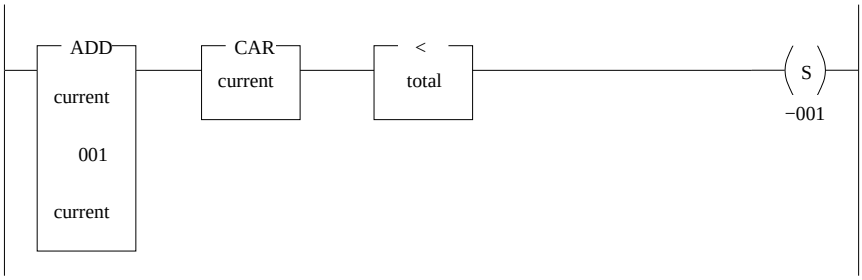
A segunda parte do workload é composta por um *mix* de instruções, distribuídas em várias lógicas, onde deseja-se medir o tempo total para execução do *mix*. A estrutura e conteúdo do *mix* de instruções é deixada em aberto para cada aplicação para a qual a CPU Industrial está sendo utilizada. Sugere-se que este seja

Lógica I − Inicialização da Variável de Tempo e do Contador de repetições

Lógica II − Instruções de Interesse do benchmark



Lógica III − Controle de total de repetições



Lógica IV − Armazenamento do Tempo total decorrido

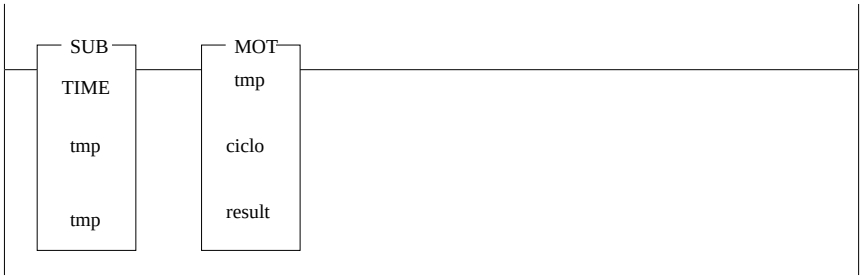


FIGURA 5.1 – Exemplo de medição do tempo de contato

desenvolvido caso a caso de acordo com a aplicação.

É importante ressaltar que aqui não se encontra um paradoxo ou indefinição. As aplicações de CPUs Industriais são vastas e, como já discutido anteriormente, é bastante difícil obter uma seqüência de instruções que seja representativa para a grande maioria das aplicações. Portanto, é pouco significativo desenvolver um *mix* de instruções genérico. Assim, propõe-se que para cada caso de aplicação mais restrita seja especificado um *mix* de instruções representativo, e que este tenha o tempo de execução medido da mesma forma que as outras partes do programa que foram desenvolvidas para avaliar o desempenho de cada tipo de instrução da CPU. Este processo é similar à construção de cada uma das partes do *benchmark* proposto aqui. Deve-se analisar as características da aplicação que está sendo utilizada, e construir um *workload* sintético do tipo kernel, que contemple as características dos núcleos do processo de controle. A partir dos tempos de instrução obtidos nas etapas anteriores do programa, é possível estimar o tempo que será necessário para executar o *mix*, e ao comparar este tempo com o tempo medido durante o programa é possível observar a precisão dos tempos de instrução medido anteriormente.

A aplicação do *mix* de instruções serve, portanto, a dois propósitos:

- verificar que os tempos de instrução medidos na primeira etapa do *benchmark* são realmente aplicáveis a programas próximos da aplicação real.

- utilizar um mesmo *mix* de instruções, significativo na aplicação real, para comparar as diversas alternativas de CPUs Industriais.

5.1.3 Valores de Entrada para o Programa

Os valores utilizados como parâmetro para as instruções do *workload* são os últimos elementos que ainda devem ser definidos para que seja possível executar o processo de *benchmarking*. Todas as execuções, em qualquer equipamento, devem ser feitas sob as mesmas condições, e isso inclui os valores de entrada para as instruções.

Nas partes do programa que envolvem estimativas de tempo de contato, é de interesse avaliar se o tempo de execução da instrução de contato depende do estado da variável que está sendo testada na instrução. Dependendo da implementação do sistema de execução dos programas de controle, o tempo para execução de uma seqüência de contatos que testam variáveis com valor em zero pode ser diferente da seqüência que testa valores de contato em um.

Portanto, serão executadas três rodadas independentes de carga, com diferentes valores nos parâmetros das instruções de contato. Na primeira rodada, todos os contatos serão executados com a entrada em *zero*. Na segunda rodada, todos os contatos serão executados com a entrada correspondente em *um*. Por fim, será executada uma terceira rodada, onde os contatos terão valores alternados entre zero e um, da seguinte forma: o primeiro contato da lógica recebe o valor *zero*, o seguinte recebe *um*, o próximo *zero*, e assim por diante.

Ao mesmo tempo, nas instruções de bloco, como instruções de soma e operações lógicas, não há grande influência dos parâmetros passados para a função, mas sim se ela está habilitada para execução em um determinado ciclo de controle ou não. Mesmo estando desabilitada, a execução do bloco pode consumir algum tempo de processamento, necessário para que o bloco identifique que o mesmo está desabilitado e que nada deve ser executado no ciclo corrente.

Para ter controle sobre a habilitação ou não de cada bloco de instruções, adicionou-se uma instrução de contato no início de cada lógica, de forma que o estado do contato possa determinar a condição de habilitado da instrução de bloco. Os resultados do *benchmark* devem reportar o tempo de execução de uma instrução de bloco quando a mesma está habilitada e desabilitada. Portanto, são necessárias duas rodadas de execução do *benchmark*, uma onde os blocos são efetivamente executados e outra onde os blocos estão desabilitados.

5.2 Monitor I – Geração de Ondas Quadradas

O primeiro mecanismo de monitoração que pode ser utilizado na caracterização de desempenho das CPUs Industriais é um monitor misto de *hardware* e *software*. O monitor de desempenho baseia-se na geração de sinais de onda quadrada em um ponto de saída, através de uma rotina de *software* a ser executada na CPU, utilizando um medidor de freqüências para observar o sinal quadrado gerado. O monitor permite obter uma estimativa dos tempos gastos em cada uma das etapas do ciclo de controle da CPU Industrial (figura 3.2).

O funcionamento da CPU Industrial envolve a execução de 4 macro etapas: leitura de entradas, execução de programa de controle, escrita de saídas e execução de rotinas auxiliares internas. O funcionamento é cíclico, e ao final da quarta etapa completa-se um ciclo e volta-se a executar a primeira etapa. Tendo em vista esse comportamento, é possível explorar um mecanismo que atualize um ponto de saída a cada ciclo. Medindo-se a freqüência de atualização deste ponto de saída é possível estimar o tempo de execução de um ciclo.

O método proposto nesta seção mede apenas o tempo de ciclo. Porém, é possível medir outras características indiretamente a partir do tempo de ciclo, modificando a arquitetura do sistema. Por exemplo, aumentando o número de módulos de entrada de 2 para 3 e depois de 3 para 4, pode-se estimar o tempo de leitura de um módulo de entrada a partir da variação do tempo de ciclo em cada caso de teste.

A figura 5.2 apresenta o trecho do programa de controle capaz de gerar uma onda quadrada. A cada varredura, o programa inverte o estado de um ponto de saída (na figura, é utilizado o ponto de saída S6.0), alternando o valor da saída entre 0 e 1. Logo, uma transição no ponto de saída, tanto de 0 para 1 quanto de 1 para 0, indica a conclusão de um ciclo de controle. Assim, pode-se medir a freqüência do sinal gerado no ponto de saída e concluir qual o tempo total de ciclo. O tempo de ciclo é igual à metade do período da onda quadrada gerada, já que para obter uma onda quadrada completa, são necessárias duas transições (de zero para um, e de um para zero), e portanto dois ciclos de controle.

Com o acréscimo e remoção de módulos, bem como com o aumento e diminuição do programa de controle, é de se esperar que a freqüência do sinal quadrado de saída se modifique. Por exemplo, ao incluir um módulo a mais de entrada numa arquitetura de controle, o tempo de varredura irá aumentar, e esse aumento corresponderá exatamente ao tempo necessário para ler o módulo de entrada que foi adicionado. Conseqüentemente, o tempo de ciclo também ficará maior, o que implica na diminuição da freqüência da onda que é gerada no ponto de saída. Assim, é possível obter uma estimativa do tempo de leitura de um módulo de entrada a partir da variação de freqüência do sinal quadrado, que deve-se apenas à adição de um

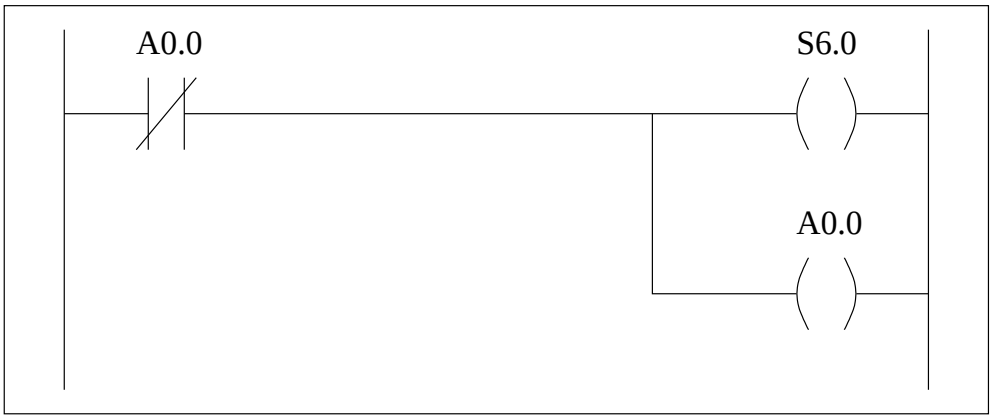


FIGURA 5.2 – Programa gerador de ondas quadradas

módulo de entrada. Para obter uma maior confiabilidade nos resultados, devem ser verificados os tempos de adição de vários módulos, um de cada vez. Assim, mede-se a variação da freqüência quando passa-se de um para dois módulos de entrada, e obtém-se o primeiro valor para leitura de módulo de entrada. Em seguida, adiciona-se mais um módulo, passando de dois para três módulos de entrada, e mede-se novamente a variação da freqüência. Para obter uma quantidade estatisticamente aceitável de resultados, deveria ser possível expandir o total de módulos de entrada até 10 módulos.

De forma similar, é possível estimar o tempo para escrita de um módulo de saída, e o tempo de execução de instruções do programa de controle do usuário. Para tanto, procede-se da mesma maneira, adicionando módulos de saída, que aumentarão o tempo de atualização e o tempo de ciclo, ou adicionando novas instruções ao programa de controle, que modificarão o tempo de programa e também o tempo de ciclo. Em seguida, verifica-se o reflexo das modificações na freqüência do sinal quadrado de saída, e assim é possível estimar o tempo para escrita de um módulo de saída ou para execução da instrução testada.

5.2.1 Estimativa de Métricas

Com o método de geração de ondas quadradas, é possível estimar as seguintes métricas:

- tempo de ciclo: diretamente a partir do período da onda de saída. O tempo de ciclo é igual à metade de período da onda de saída.

- tempo para escrita de módulo de saída: pode ser deduzido a partir da medição de vários tempos de ciclo. Modificando a quantidade de módulos de saída, pode-se medir os tempos de ciclo de cada arquitetura e deduzir o tempo de escrita de um módulo a partir da diferença entre os tempos de ciclo medidos.

- tempo para leitura de módulo de entrada: análogo à medida do tempo de escrita de módulo de saída, através do acréscimo de módulos de entrada e posterior medição do tempo de ciclo.

- tempo de execução de uma instrução básica: calculado indiretamente a partir do tempo de ciclo ao se modificar o programa de controle utilizado. Modificando-se apenas o número de instruções básicas, todo o incremento ou decremento do tempo de ciclo será devido à mudança do programa de controle.

- tempo gasto em rotinas internas: sabendo-se o tempo gasto com leitura de módulos, execução de ciclo e escrita de módulos, pode-se descontar esses tempos do tempo de ciclo. O valor restante é o tempo gasto em rotinas internas da CPU Industrial.

5.2.2 Exemplo — QK801

Para exemplificar a técnica de monitoração apresentada nesta seção, foram medidos os tempos de leitura de módulos de entrada e de escrita de módulos de saída para a CPU Quark QK801 [ALT 98], fabricada e fornecida pela Altus.

A figura 5.3 mostra o sistema montado para realizar as medições apresentadas neste estudo de caso. Ele é composto por uma CPU Industrial da Altus, da família Quark, modelo QK801. Foram montados 3 módulos de entradas digitais e 4 módulos de saídas digitais, que foram adicionados e removidos da arquitetura para realizar as medições de tempos de ciclo. Posteriormente foram calculados os tempos de leitura/escrita dos módulos.

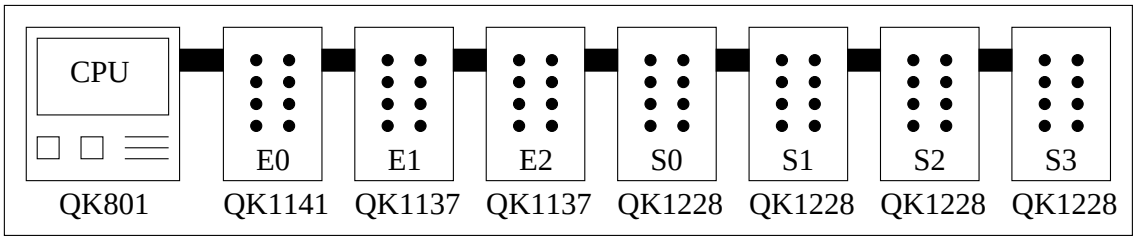


FIGURA 5.3 – Arquitetura do Exemplo — QK801

Como o objetivo é apenas ilustrar o funcionamento da técnica proposta, o programa que rodou na CPU QK801 neste exemplo era composto apenas pela sequência geradora de ondas quadradas, conforme ilustrado na figura 5.2. A tabela 5.2 mostra o conjunto de experimentos realizados e os resultados obtidos em cada um deles. Em cada linha da tabela, estão apresentados quais os módulos ilustrados na figura 5.3 são conectados em cada experimento, juntamente com a freqüência obtida no sinal de saída, o período da onda quadrada de saída e ainda o tempo de ciclo calculado para cada caso. Os módulos presentes em cada execução são marcados com um "X", e os que estão desconectados são deixados em branco na tabela. Na primeira execução, por exemplo, estão conectados a CPU e o módulo S0. A freqüência resultante do sinal quadrado é de 744 Hz, o que resulta em um tempo de ciclo de 0.672 ms.

Observando a tabela 5.2, é possível tirar algumas conclusões a respeito dos módulos de entrada e saída do QK801. Por exemplo, verifica-se que o tempo de acesso a módulos de entrada é um pouco menor que o dos módulos de saída.

Para calcular os tempos de acesso a módulos de entrada (E0 – E2) e saída (S0 – S3), é preciso verificar qual o impacto no tempo de ciclo causado pela adição

TABELA 5.2 – Resultados do QK801

CPU	E0	E1	E2	S0	S1	S2	S3	F(Hz)	T(ms)	Ciclo(ms)
X				X				744	1.344	0.672
X				X	X			659	1.517	0.758
X				X	X	X		591	1.692	0.846
X				X	X	X	X	531	1.883	0.941
X	X			X				648	1.543	0.756
X	X			X	X			582	1.718	0.859
X	X			X	X	X		529	1.890	0.945
X	X			X	X	X	X	485	2.061	1.030
X	X	X		X				577	1.733	0.866
X	X	X		X	X			525	1.904	0.952
X	X	X		X	X	X		481	2.079	1.039
X	X	X		X	X	X	X	444	2.252	1.126
X	X	X	X	X				520	1.923	0.961
X	X	X	X	X	X			478	2.092	1.046
X	X	X	X	X	X	X		441	2.267	1.133
X	X	X	X	X	X	X	X	409	2.444	1.222

de mais um módulo na arquitetura. Por exemplo, a única modificação da primeira arquitetura reportada na tabela para a segunda é a adição de um módulo de saída na arquitetura de controle. Ao mesmo tempo, o tempo de ciclo aumentou em 86μs. O tempo de ciclo sofre aproximadamente o mesmo acréscimo da segunda para a terceira topologia, e da terceira para a quarta (respectivamente 95μs e 88μs). A análise dos resultados segue para todos os outros experimentos realizados, calculando o aumento no tempo de ciclo cada vez que um novo módulo de saída é adicionado. A partir dos resultados, pode-se afirmar que o tempo de acesso a um módulo de saída é de aproximadamente 90μs.

O comportamento para os módulos de entrada é bastante parecido. Observa-se que o aumento de zero para um módulo de entrada tem um impacto no tempo de ciclo um pouco maior (aproximadamente 100μs) que nos outros casos testados, aumentando de um para dois e de dois para três a quantidade de módulos de entrada (aproximadamente 93μs em ambos os casos). Logo, é razoável imaginar que o tempo de varredura das entradas tem uma componente constante, que pode ser denominado de tempo de *setup*, que existe pelo simples fato de ter que executar uma varredura de entradas, e um tempo de leitura de um módulos de entradas, que varia de acordo com o total de módulos. Neste exemplo, todos os módulos de entrada são similares, e portanto o tempo de leitura de cada módulo é sempre o mesmo. Desta forma, este experimento apresenta resultados que são coerentes com a representação do tempo de varredura discutido na figura 4.3.

5.2.3 Conclusões

Esta técnica tem um custo bastante baixo, visto que a única instrumentação necessária é um freqüencímetro. Além disso, não é necessária nenhuma modificação no código do sistema operacional que executa na CPU. O programa de controle,

especificado pelo usuário e que pode ser modificado muito mais facilmente é que deve ser instrumentado com o trecho de programa extra da figura 5.2. O tempo necessário para montar o ambiente de teste é pequeno e o tempo para realizar as medidas efetivamente também é curto.

Por outro lado, a precisão dos resultados depende da qualidade do medidor de freqüências que foi utilizado. Igualmente, não é possível avaliar todas as métricas listadas na seção 4.1. Medidas como *jitter* de tempo de ciclo não podem ser extraídas apenas com o uso de freqüencímetro. Entretanto, com o auxílio de um osciloscópio ou um analisador lógico, estes tempos podem ser avaliados com maior precisão.

Outra característica desta forma de monitoração é que apenas o tempo de ciclo pode ser efetivamente medido. Todas as demais métricas são extraídas de forma indireta a partir da modificação da arquitetura do sistema e conseqüente alteração do tempo de ciclo.

5.3 Monitor II – Utilização de Timers

A outra técnica de monitoração utilizada na avaliação das CPUs Industriais neste trabalho envolve a instrumentação do código do programa de controle com rotinas de manipulação dos *timers* do sistema. Este é um sistema de monitoração de *software*. Em um equipamento de controle industrial, os timers são recursos bastante utilizados e as linguagens de programação padronizadas para estes dispositivos [IEC 93] dispõem de recursos para fácil acesso ao relógio. É comum que CPUs Industriais utilizem os chamados relógios de tempo real, que permitem uma maior precisão de medida e uma divisão de tempo mais refinada.

Utilizar *timers* para extração de métricas e observação do tempo decorrido para execução de tarefas é uma técnica que pode apresentar erros de medida significativos caso alguns cuidados não sejam tomados. São problemas inerentes à utilização de monitores de *software*. O primeiro deles diz respeito à granulosidade do relógio do sistema. O tempo é contado de forma discreta dentro do equipamento, sendo incrementado a intervalos de tempo constantes. Portanto, um registro de tempo do relógio do sistema contém uma distorção equivalente no máximo ao intervalo de incremento do relógio. Logo, esta não é uma técnica apropriada para medição de tempos próximos ao intervalo de incremento do relógio. Um *timer* que é incrementado a cada 50ms não é adequado para medir tempos inferiores a 3 segundos.

Para contornar o problema da granulosidade do relógio, o recurso mais comum é a repetição sucessiva do trecho de interesse da medição, de forma a aumentar o tempo a ser medido. Assim, o erro causado pela granulosidade do relógio é pouco significativo quando comparado com o tempo total medido. Porém, a adição de instruções para repetir o trecho a ser medido causa nova intrusão nas medidas, já que o resultado da medida é o tempo para executar não só as instruções de interesse, mas também as instruções de controle de repetição, além das próprias instruções de observação do relógio e ainda o tempo gasto com o atendimento da interrupção do relógio, que ocorre em tempos constantes.

A solução para este segundo problema é incluir um trecho de "calibração" do medidor, onde se avalia apenas o tempo para executar as intruções de controle e observação do relógio, utilizando um trecho vazio para ser medido. O tempo obtido

nesta medida representa a intrusão causada pelo monitor nos resultados medidos, e deve ser descontado de todos os resultados da avaliação.

Os tempos medidos podem ser colocados em tabelas internas do controlador, ficando assim disponíveis para serem consultados pelos avaliadores. A busca dos dados medidos pode ser feita pela da rede de supervisão, através de comandos de leitura de tabelas de memória, disponível em qualquer protocolo de supervisão utilizado em ambientes de gerenciamento de CPUs Industriais.

A figura 5.1 utiliza a técnica descrita nesta seção para extrair o tempo de execução da instrução de contato. Na figura, a lógica III mostra as instruções de controle para executar o trecho de interesse (lógica II) diversas vezes, de forma a aumentar o tempo a ser observado através do relógio do sistema e diminuir o erro de medida causado pela granulosidade do relógio. Ainda no exemplo da figura, as lógicas I e IV são responsáveis pela observação do relógio no início e final do experimento, armazenando o intervalo de tempo decorrido numa tabela interna da CPU.

5.3.1 Estimativa de Métricas

A monitoração das métricas utilizando *timers* permite avaliar todas as métricas relacionadas com o desempenho do programa de controle. As seguintes métricas podem ser avaliadas pela técnica de monitoração proposta:

- tempo de ciclo: o tempo de ciclo pode ser medido observando o relógio no início da execução do programa de controle em cada ciclo e calculando o intervalo de tempo decorrido desde a observação do relógio no ciclo anterior.

- tempo de instrução: o tempo gasto para executar uma instrução ou bloco de instruções pode ser obtido cercando com os *timers* um trecho com as instruções de interesse. Deve-se tomar cuidado para que o tempo de execução das instruções seja significativamente maior que a imprecisão da medida causada pela granulosidade do relógio.

- tempo de leitura de módulos de entrada: pode ser obtido indiretamente a partir de medidas do tempo de ciclo. Modificando-se a quantidade de módulos de entrada, o tempo de ciclo será acrescido de um tempo equivalente ao tempo de leitura dos novos módulos adicionados.

- tempo de escrita de módulos de saída: este tempo também pode ser calculado indiretamente a partir da variação do tempo de ciclo quando a quantidade de módulos de saída é modificada, da mesma forma que ocorre com o tempo de leitura de módulos de entrada.

5.3.2 Exemplo — AL2003

Novamente para exemplificar a técnica de monitoração, esta seção apresenta resultados para a avaliação de tempo de execução da instrução de contato na CPU AL2003 [ALT 98a] da Altus. O programa executado foi o mesmo ilustrado na figura 5.1. A granulosidade do relógio foi ajustada para 5 ms, e a lógica que contém as instruções de interesse foi executada 1000 vezes para cada medida, como forma de aumentar o tempo a ser medido e diluir o erro causado pela granulosidade do relógio.

O programa de avaliação foi executado 12 vezes, e calculou-se a média, desvio padrão e intervalo de confiança dos resultados. A tabela 5.3 resume os números deste experimento.

TABELA 5.3 – Resultados do AL2003

Tempo de execução do programa	168,400 ms
Tempo da etapa de calibração	106,416 ms
Tempo efetivo com instruções de contato	61,984 ms
Tempo médio de execução da instrução	1,937 μs
Desvio Padrão	0,0070
Intervalo de Confiança (95%)	0,0039

Pela tabela 5.3, pode-se ver que o tempo de controle para a repetição da lógica de contatos é significativo no tempo total de execução do programa. Se este valor não tivesse sido levado em consideração, o valor da instrução de contato seria bastante distorcido. Vê-se também que a CPU AL2003 tem um comportamento bastante regular para execução de instruções de contato. O desvio padrão é bastante baixo, o que significa que os resultados obtidos durante a experimentação foram bastante próximos entre si. Ao mesmo tempo, o intervalo de confiança para 95% de certeza do resultado real é uma faixa de valores bastante estreita. O intervalo de confiança está mostrando que, com 95% de chances, o tempo de contato em qualquer medição futura estará no intervalo de 1,937$\pm$0,0039 μs.

5.3.3 Conclusões

A técnica que utiliza timers do sistema para monitorar o desempenho do equipamento é capaz de extrair uma quantidade maior de métricas que o monitor de *hardware* apresentado anteriormente. Enquanto o monitor de ondas quadradas é capaz de medir apenas o tempo de ciclo e calcular as demais métricas de forma indireta, a utilização de *timers* permite avaliar o desempenho das diversas instruções que são executadas na CPU Industrial, com uma precisão relativamente boa, através de medidas diretas.

Porém, esta é uma técnica muito mais intrusiva que a geração de ondas quadradas. Isto é uma característica inerente aos monitores de *software* que exigem que o código de monitoração seja executado dentro da mesma CPU que está sendo avaliada. Portanto, para melhorar a qualidade dos resultados deve-se conhecer o grau de intrusão gerado pelo monitor. Num sistema de tempo real como a CPU Industrial, deve-se tomar um cuidado ainda maior com estas técnicas de monitoração, já que a intrusão causada por ela pode ser determinante na modificação do comportamento temporal do sistema, fazendo com que a situação que está sendo medida com os *timers* não represente o comportamento do sistema sem a intrusão dos *timers*. Outro cuidado a ser tomado é rodar os experimentos por um tempo longo, de forma que a granulosidade do relógio não cause um erro de medida significativo.

5.4 Relatório de Resultados

Com o *workload* e os monitores apresentados neste capítulo, pode-se caracterizar o desempenho de CPUs Industriais diretamente a partir da execução de programas de controle que rodam sob as mesmas condições que os programas de aplicações reais. As métricas apresentadas no capítulo 4, especialmente na seção 4.1, podem ser extraídas com estes *workloads* e monitores.

Os resultados obtidos em cada CPU devem incluir as seguintes informações a respeito do equipamento testado e das condições de teste:

- Tipo de Processador
- Velocidade do Processador
- Quantidade de Memória FLASH e RAM
- Número Máximo de Pontos de I/O Gerenciáveis pela CPU
- Precisão do relógio utilizado no monitor de *software*
- Linguagem de Programação utilizada no *workload*
- Resultados dos testes, conforme apresentado na tabela 5.4

A tabela 5.4 apresenta um resumo para a extração das métricas, mostrando como cada parte do programa de avaliação pode colaborar na caracterização destas, e ainda qual a técnica de monitoração mais apropriada em cada caso.

TABELA 5.4 – Métricas, *Workloads* e Monitores

Métrica	*Workload*	Monitor	Resultados
Tempo de Ciclo	*mix* de instruções	Ondas Quadradas	média e desvio padrão
Tempo de Programa	*mix* de instruções	*timers*	média e desvio padrão
Tempo de Instruções Básicas	Lógica com a instrução básica	*timers*	média, desvio padrão e intervalo de confiança
Tempo de Varredura e Atualização	*mix* de instruções	Ondas Quadradas	média e desvio padrão

5.5 O Procedimento de Avaliação

O procedimento para aplicação dos *workloads* e monitoração das métricas segue os passos necessários para instalação e execução de um programa de controle na CPU que está sendo caracterizada. Alguns cuidados e ações a serem seguidas são detalhadas a seguir.

- caso o equipamento não suporte a programação via IEC61131, deve-se portar o *benchmark* para o sistema de programação suportado, tomando os devidos cuidados para manter a estrutura do programa o mais próximo possível da estrutura original.

- após o porte, o *benchmark* pode ser instalado no equipamento a ser avaliado e em seguida executado. As métricas apresentadas na tabela 5.4 devem ser caracterizadas de acordo com os *workloads* e monitores especificados.

- para que os resultados sejam os mais realistas possíveis, é recomendado que a CPU esteja dedicada apenas à execução de instruções. Tarefas paralelas, como comunicação com sistemas de supervisão, devem ser evitadas, de forma a evitar intrusões nos resultados. Posteriormente, é possível avaliar o impacto destas tarefas no desempenho da CPU, desde que seja possível controlar a atividade e interferência de cada uma delas.

- a determinação do número de iterações dos laços de avaliação em uma execução do *benchmark* deve levar em conta a granulosidade do relógio, o tempo a ser medido pelo monitor de timer, quando este for utilizado, e o tempo máximo de ciclo, caso um sistema de cão de guarda esteja instalado para limitar o tempo máximo de ciclo.

- a etapa de calibração do *benchmark* deve ser executada sempre que o *benchmark* é rodado. O número de iterações do laço de calibração deve ser igual ao número de iterações do laço das instruções de interesse. Desta forma, o tempo obtido para execução da etapa de calibração oferece uma aproximação bastante aceitável do tempo a ser gasto com a parte de controle do *benchmark* quando se está avaliando o tempo de execução de uma instrução específica.

- As lógicas de interesse para avaliação dos tempos de execução das instruções de controle devem ser repetidos tantas vezes quanto forem possíveis, de forma a diluir os problemas de granulosidade e imprecisão do relógio. O tempo de execução de cada uma das etapas do *benchmark* deve ser consideravelmente (pelo menos 100 vezes) maior que a granulosidade do relógio.

- caso a CPU Industrial apresente um sistema de cão de guarda (*watchdog*)que limite o tempo máximo de um ciclo, cada execução do *benchmark* deve avaliar um conjunto restrito de instruções, de forma a não comprometer o tempo mínimo de execução de cada teste e, por conseqüência, também não comprometer a qualidade dos resultados obtidos.

6 Estudos de Caso

Este capítulo apresenta um estudo de desempenho de três sistemas de controle industrial, como forma de verificar a aplicabilidade do *benchmark* proposto na caracterização de desempenho destes equipamentos. Uma vez que esta metodologia de avaliação de desempenho foi desenvolvida como parte de um projeto CT-Petro em parceria com a empresa Altus, foram selecionadas duas CPUs Industriais desenvolvidas pela Altus que estavam à disposição nos laboratórios do Instituto de Informática. Além disso, avaliou-se o comportamento de um sistema de programação e controle com suporte à IEC61131, o ISaGRAF.

Enquanto as duas primeiras CPUs estudadas e avaliadas são CPUs Industriais tradicionais, que apresentam *hardware* especialmente desenvolvido para ser inserido no ambiente hostil encontrado no chão de fábrica, o sistema ISaGRAF é apenas uma CPU de *software*, chamado SoftPLC. O SoftPLC fornece um sistema de controle industrial que pode ser executado em equipamentos mais tradicionais, como um PC convencional. Ao utilizar um PC convencional para executar o sistema de controle industrial, consegue-se algumas vantagens, e também algumas desvantagens em relação à CPU Industrial tradicional. Por um lado, é mais fácil a integração de novos dispositivos de *hardware*, como placas de rede, expansões de memória ou até mesmo atualização de processadores. Entretanto, também é necessário o desenvolvimento de módulos de *hardware* extras para acesso aos módulos de entrada e saída. Além disso, muitos dos SoftPLC foram desenvolvidos para execução sobre sistemas operacionais não tempo real, como o Windows NT, e assim é possível enfrentar problemas de determinismo e confiabilidade que não são desejáveis neste tipo de aplicação e que não são encontrados em CPUs Industriais tradicionais.

6.1 Condução dos Experimentos

Os experimentos foram conduzidos no Laboratório de Automação de Sistemas, de forma assistida. Devido à limitação do tempo máximo de ciclo das CPUs Altus, o programa de avaliação teve que ser executado em várias etapas, cada uma delas exercitando uma das partes a que se propunha o algoritmo de avaliação. O número de repetições da lógica de interesse em cada execução também teve que ser limitada ao tempo máximo de ciclo admitido para cada CPU.

Os programas precisaram ser portados para cada um dos equipamentos utilizados, já que nenhum deles é totalmente compatível entre si. Apesar de usarem o mesmo sistema de programação, as CPUs AL2003 e QK801 não puderam compartilhar nem mesmo as descrições em diagramas de relés. Já o SoftPLC do ISaGRAF exigiu um porte completo do benchmark, adaptando as estruturas e rotinas anteriormente desenvolvidos para a família de CPUs Altus.

Durante a execução dos experimentos, as CPUs estavam isoladas, evitando assim qualquer perturbação da medição devido a rotinas de comunicação, por exemplo, ou qualquer outra tarefa que não estivesse diretamente ligada ao *workload*. No caso do ISaGRAF SoftPLC, que implementa uma CPU Industrial sobre um PC convencional, o sistema foi executado no modo dito “tempo real”.

Cada parte do programa de avaliação foi repetida 12 vezes, de forma a possibilitar o cálculo de média, desvio padrão e intervalo de confiança de cada uma das

métricas. No caso da CPU AL2003, por exemplo, foram executadas 12 rodadas de cada uma das partes do programa, resultando em 144 execuções de aproximadamente 5 minutos cada. Cada execução avaliou uma das 6 instruções básicas abordadas pelo *workload*, e a variação das condições de entrada levou ao total de 144 execuções. Portanto, apenas a avaliação da CPU AL2003 consumiu 720 minutos, ou 12 horas, sem incluir tempos de ajuste de parâmetros, configuração do equipamento e tratamento de resultados.

6.2 Altus AL2003

A CPU Industrial é atualmente a CPU de maior capacidade de processamento e gerenciamento de entradas e saídas da linha de controladores da Altus. O AL2003 é um sistema baseado no microcontrolador Intel 80251. Possui 128kb de memória RAM e 1Mb de memória FLASH. Dois canais de comunicação serial estão disponíveis no sistema, e a CPU é capaz de tratar aproximadamente 2048 pontos de entrada e saída digitais simultaneamente, com o auxílio de uma placa de expansão de entrada e saída. O projeto da CPU é modular, de forma que é possível adicionar uma placa de comunicação Ethernet e expansões de pontos de entrada e saída através de *hardware* extra conectado ao barramento de comunicação interno da CPU.

A CPU AL2003 é programada com *software* específico desenvolvido pela Altus e suporta apenas a linguagem de diagrama de relés para especificação do programa de controle. A ausência de suporte às demais linguagens da norma IEC61131 é uma limitação do programador, uma vez que este gera código objeto preparado para ser rodado diretamente pelo controlador 80251. Uma extensão futura ao programador Altus tornaria possível o suporte a IEC61131, bastando para isso que o programador pudesse gerar código objeto a partir de qualquer linguagem da norma.

A tabela 6.1 apresenta as características da CPU AL2003, como tipo e velocidade do processador, quantidade de memória e número de pontos gerenciáveis, entre outros. A tabela 6.2 apresenta os tempos de cada instrução avaliada para a CPU, com o tempo de execução, desvio padrão e intervalo de confiança de 95%. Todos os resultados da tabela 6.2 foram obtidos a partir da execução do *benchmark* proposto anteriormente, utilizando o monitor de *software* via timers. Por fim, a tabela 6.3 apresenta os tempos levantados para leitura e escrita de módulos de entrada e saída, respectivamente. Estes resultados foram extraídos utilizando o monitor de geração de ondas quadradas em um ponto de saída da CPU.

TABELA 6.1 – Características da CPU AL2003

Tipo de Processador	:	Intel 80251
Velocidade do Processador	:	20 MHz
Memória FLASH	:	1024 Kbytes
Memória RAM	:	128 Kbytes
Pontos de I/O	:	2048
Precisão do Relógio	:	5 ms
Linguagem de Programação	:	Diagrama de Relés

Observando os tempos de instrução do AL2003, nota-se que a CPU apresenta

um comportamento bastante regular e constante. Isso pode ser deduzido a partir da baixa variação obtida entre as diversas rodadas de avaliação executadas. A taxa de variação, que é a razão entre desvio padrão e média, é inferior a 1% em todas as instruções medidas, mostrando o determinismo da CPU. O determinismo também está representado no intervalo bastante restrito para confiança de 95%. Por exemplo, o tempo de contato medido para a CPU AL2003 é de 1980ns, com um intervalo de confiança de 95%. Isso significa que em 95% das vezes que se medir o tempo de contato, ele estará no intervalo de 1980±1.21ns. Nota-se também que há uma diferença no tempo de execução dos blocos de instrução quando estes estão habilitados ou desabilitados, como era de se esperar. Por outro lado, é indiferente para as instruções de contato se o parâmetro da mesma é *um* ou *zero*, já que a diferença nos resultados não é estatisticamente significativa.

Ao mesmo tempo, os módulos de entrada e saída não apresentam um comportamento tão constante quanto desejado, conforme pode ser visto na tabela 6.3. Apesar da leitura de um módulo ser feita rapidamente, o tempo gasto pode variar significativamente. Isso pode ser justificado pelo fato de que na AL2003, os módulos de entrada e saída são, na arquitetura avaliada, localizados em barramentos de extensão. Cada barramento de extensão é gerenciado por um módulo processado dedicado, e o ciclo de leitura/escrita é feito de forma independente da CPU de processamento. Assim, a requisição de leitura pode ficar aguardando o final da varredura do barramento de extensão pelo processador dedicado até que a resposta seja enviada. Dependendo do momento que a requisição é recebida no barramento de extensão, o tempo de espera pode ser maior ou menor.

TABELA 6.2 – Tempos de Instrução do AL2003

Instruçao	Condições	Tempo de Execução (ns)	Desvio Padrão	Intervalo de Confiança (95%)
Contato	Entradas em 1	1981	2.139	1.210
	Entradas em 0	1982	2.137	1.209
Soma	Habilitada	35489	7.156	4.049
	Desabilitada	16770	9.927	5.616
Lógica	Habilitada	32512	33.300	18.841
	Desabilitada	16655	10.119	5.725
Movimentação	Habilitada	29049	4.301	2.433
	Desabilitada	15757	3.390	1.918
Chamada de Função	Habilitada	44072	7.504	4.245
	Desabilitada	14157	37.070	20.974

TABELA 6.3 – Tempo de Leitura e Escrita de Entradas e Saídas do AL2003

	Média (μs)	Desvio Padrão
Módulo de Entrada Digital	100.37	38.889
Módulo de Saída Digital	114.00	39.175

6.3 Altus QK801

A CPU QK801 é uma CPU de menor capacidade que o AL2003, executando o controle de ambientes de pequeno e médio porte. A QK801 é baseada em microcontroladores Intel 8051. A CPU testada dispõe de 64kb de memória RAM e 64kb de memória FLASH. É capaz de gerenciar até 256 pontos de entrada e saída simultâneos, sem possibilidade de barramentos de expansão para conexão de *hardware* extra.

A programação do QK801 também é feita pelo mesmo software de programação do AL2003. Porém, no caso do QK801 é gerado código objeto para ser rodado no controlador 8051.

TABELA 6.4 – Características da CPU QK801

Tipo de Processador	:	Intel 8051
Velocidade do Processador	:	12 MHz
Memória FLASH	:	64 Kbytes
Memória RAM	:	64 Kbytes
Pontos de I/O	:	256
Precisão do Relógio	:	5 ms
Linguagem de Programação	:	Diagrama de Relés

A tabela 6.5 apresenta os resultados obtidos para escrita e leitura de módulos de entrada e saída. Conforme proposto no trabalho, estes valores foram monitorados através da geração de ondas quadradas em um ponto de saída. Ao contrário da CPU AL2003, o QK801 não possui barramento de expansão, e todos os módulos de entrada e saída estão conectados diretamente ao barramento local. Exatamente por isso, os resultados apresentam uma variação bem menor que os módulos de entrada e saída da AL2003, já que o mesmo processador é responsável por executar o programa de controle e por varrer os módulos de entrada e saída.

TABELA 6.5 – Tempo de Leitura e Escrita de Entradas e Saídas do QK801

	Média (μs)	Desvio Padrão
Módulo de Entrada Digital	95.44	3.086
Módulo de Saída Digital	87.44	2.790

6.4 ISaGRAF SoftPLC

O ISaGRAF é um ambiente completo para o desenvolvimento de aplicações de controle industrial, que está de acordo com a norma de programação de CPUs Industriais IEC61131 [IEC 93]. O ambiente ISaGRAF pode gerar dois tipos de saída para um programa de controle:

- um código C, que contém toda a estrutura necessária para implementar o controle, desde o programa propriamente dito até as rotinas internas que seriam

implementadas pela CPU Industrial, como atualização de entradas e saídas. O código C deve ser compilado e linkado com as bibliotecas ISaGRAF, de forma a gerar um programa executável para o controle do ambiente.

- o sistema especificado em uma linguagem intermediária, TIC, que pode ser interpretada pelos *kernels* ISaGRAF. Com a utilização de TIC, perde-se em desempenho, mas ganha-se em portabilidade de código. Qualquer ambiente que dispõe de um *kernel* ISaGRAF é capaz de interpretar o programa de controle, e o desenvolvimento de *kernels* é facilitado, já que a especificação da linguagem TIC e do protocolo de comunicação com o ISaGRAF para envio, recebimento e monitoração de programas são abertos.

A alternativa avaliada foi a geração de código em linguagem intermediária, que foi interpretado pelo *kernel* disponibilizado para Windows NT. Desta forma, "transformou-se" um PC em uma CPU Industrial, com todas as limitações conhecidas do NT e do PC para aplicações de tempo real, principalmente a baixa qualidade do relógio do sistema, de baixa precisão e pouca granularidade, e o grande não determinismo do sistema operacional. Tem-se então o SoftPLC, descrito no início deste capítulo.

O *kernel* ISaGRAF foi executado num equipamento com processador AMD K6-2, de 450Mhz, com as características listadas na tabela 6.6. Os resultados obtidos para execução de cada instrução no SoftPLC estão mostrados na tabela 6.7.

TABELA 6.6 – Características do ISaGRAF SoftPLC

Tipo de Processador	:	AMD K6-2
Velocidade do Processador	:	450 MHz
Memória FLASH	:	N/A
Memória RAM	:	128 Mbytes
Pontos de I/O	:	N/A
Precisão do Relógio	:	10 ms
Linguagem de Programação	:	IEC 61131

Observando os resultados na tabela 6.7 mostram um *kernel* ISaGRAF extremamente constante, ao contrário do que se imaginava que fosse acontecer numa CPU Industrial baseada num PC convencional. O fato do *kernel* ter sido configurado para rodar em modo "tempo real" pode justificar em parte a constância. Entretanto, para algumas instruções, todas as rodadas de execução apresentaram *exatamente* o mesmo resultado, o que leva à suposição do estabelecimento de um sincronismo entre o equipamento sendo medido e o método de medição.

Na comparação de desempenho contra a CPU AL2003, verifica-se resultados bastante superiores no *kernel* ISaGRAF. Porém, deve-se tomar cuidado ao afirmar que o ISaGRAF é "melhor" que o AL2003, já que o SoftPLC do ISaGRAF foi executado em um processador muito mais veloz e com uma arquitetura muito mais moderna que o microcontrolador 80251 que é utilizado pela AL2003.

No caso do ISaGRAF SoftPLC, não foi possível avaliar os tempos de leitura e escrita de entradas e saídas, já que o sistema foi executado em um PC que não dispunha de nenhum módulo de entrada ou saída.

TABELA 6.7 – Tempos de Instrução do ISaGRAF SoftPLC

Instrução	Condições	Tempo de Execução (ns)	Desvio Padrão	Intervalo de Confiança (95%)
Contato	Entradas em 1	59	0.121	0.682
	Entradas em 0	59	0.000	0.000
Soma	Habilitada	333	0.000	0.000
	Desabilitada	195	0.000	0.000
Lógica	Habilitada	501	1.287	0.728
	Desabilitada	195	0.000	0.000
Movimentação	Habilitada	253	0.360	0.204
	Desabilitada	196	0.643	0.364
Chamada de Função	Habilitada	528	0.000	0.000
	Desabilitada	196	0.000	0.000

6.5 Comparação de Desempenho

A comparação do desempenho das CPUs Industriais avaliadas tomando com base os valores absolutos obtidos durante o processo de caracterização de desempenho é obviamente favorável ao ISaGRAF. Com um processador de muito maior capacidade em relação aos outros dois equipamentos, o ISaGRAF apresenta números largamente superiores.

Porém essa comparação é injusta. Com um sistema rodando a 450Mhz, contra uma CPU de 20Mhz e outra de 12Mhz, este resultado é mais do que normal. Para tentar normalizar os números e possibilitar uma comparação relativa entre as CPUs, esta seção apresenta resultados normalizados em relação à freqüência do processador. Assim, é possível expressar cada uma das métricas com base no número de ciclos de máquina necessários em cada uma delas. Sabe-se que mesmo esta comparação não é a mais adequada quando tratam-se de processadores tão distintos quanto um microcontrolador 80251 e um AMD K6-2, porém ela já permite tratar com valores menos distorcidos pela diferença de velocidade do processador.

A tabela 6.8 apresenta os números obtidos para a comparação de desempenho relativo. Para chegar a cada um dos valores da tabela, dividiu-se os tempos obtidos para execução de cada instrução ou o tempo para leitura/escrita de módulos pelo período do ciclo de máquina do processador. Assim, os resultados apresentados na tabela 6.8 estão expressos em ciclos de máquina.

Observando a tabela 6.8, é possível verificar a qualidade e eficiência de cada implementação de CPU. Claramente, o SoftPLC da ISaGRAF leva bastante vantagem sobre a CPU AL2003, gastando aproximadamente um terço do total de ciclos gastos pela última para executar cada uma das instruções. Esse desempenho reflete o que era esperado, já que o K6-2 é um processador mais moderno e com mais recursos de unidades de processamento e pipeline que o 8051.

TABELA 6.8 – Resultados Normalizados pela Freqüência do Processador

Métrica	AL2003	QK801	ISaGRAF
Freqüência do Processador	20Mhz	12Mhz	450Mhz
Tempo do Ciclo de Máquina	50ns	83.3ns	2.2ns
Instrução de Contato	39.6		26.8
Instrução de Soma Habilitada	709.8		151.3
Instrução de Soma Desabilitada	335.4		88.6
Instrução Lógica Habilitada	650.2		227.7
Instrução Lógica Desabilitada	333.1		88.6
Instrução de Movimentação Habilitada	581		115
Instrução de Movimentação Desabilitada	315.1		88.6
Chamada de Função Habilitada	881.4		240
Chamada de Função Desabilitada	283.1		89.1
Leitura de Módulo	2007.4	1145.7	
Escrita de Módulo	2280	1049.7	

7 Conclusão

Este trabalho estudou o desempenho de CPUs Industriais, um tópico que apresenta carência na área de controle industrial. Nos dias atuais, onde padrões abertos para soluções de *software*, *hardware* e redes de comunicação são cada vez mais utilizados, a caracterização de desempenho destes equipamentos mostra-se de uma importância cada vez maior. O objetivo do trabalho engloba a caracterização de CPUs Industriais para aplicações de controle de processos, fornecendo uma ferramenta que possa ser utilizada na avaliação da adequação de uma determinada arquitetura para resolver um problema de controle específico. Ao mesmo tempo, a ferramenta fornece um procedimento padronizado que pode ser utilizado para comparação de diversas implementações distintas de CPUs Industriais.

Entre as principais contribuições desta dissertação estão:

- Elaboração e proposta de um conjunto de métricas para avaliação de desempenho de sistemas de controle baseados em CPUs Industriais. Foram abordadas métricas em diversos níveis de abstração, desde o nível de controle de processos até o compilador e arquitetura de *software*, passando por elementos de *hardware*, sistemas operacionais e redes. As métricas propostas para caracterização de desempenho foram discutidas no capítulo 4.

- Proposta de um sistema de avaliação sintético, capaz de caracterizar as métricas construídas durante o trabalho. O *workload* sintético foi proposto a partir da observação de um conjunto de programas de controle, e está detalhado no capítulo 5.

Este estudo era necessário, já que não havia até então uma forma padrão para levantamento de desempenho das CPUs Industriais. Nem mesmo havia uma técnica padrão para o conhecimento de uma CPU. Esta é uma lacuna importante quando lida-se com aplicações de tempo real, onde o conhecimento do comportamento do sistema é premissa fundamental para o sucesso do mesmo.

A experiência na utilização destes equipamentos facilitou as primeiras etapas do processo, como o conhecimento do sistema e a delimitação do objeto de estudo. O processo de estabelecimento de métricas exige bastante esforço e reflexões, e as mesmas passaram por várias revisões até que tomassem o formato final apresentado neste trabalho. A partir da observação de um conjunto de programas reais de controle fornecidos pela Altus, é proposto um programa sintético capaz de caracterizar o desempenho de CPUs Industriais com base nas métricas propostas. Discute-se também os monitores de desempenho a serem utilizados para extração de cada uma das métricas e finalmente são estudados o desempenho de 3 equipamentos de controle industrial, como forma de mostrar a aplicação do procedimento de *benchmarking* proposto.

Entre as principais métricas desenvolvidas, destacam-se as métricas de avaliação do processo de controle, listadas a seguir:

- Tempo de Ciclo

- Tempo de Programa

- Tempo de Bit

- Tempo de Reação
- Tempo de Varredura
- Tempo de Leitura de Módulos de Entrada
- Tempo de Atualização
- Tempo de Escrita de Módulos de Saída

A tabela 7.1 resume as métricas apresentadas no trabalho, mostrando a influência das métricas de mais baixo nível nas métricas de controle.

A partir da observação de programas de controle, confirmou-se a idéia de que o tempo de bit, ou tempo de execução de uma instrução de contato, é a principal métrica neste tipo de sistema, já que os programas escritos para CPUs Industriais apresentam grande quantidade de instruções deste tipo. Entre as técnicas apresentadas por Jain [JAI 91] para otimização de desempenho de sistemas computacionais, a primeira e talvez mais importante é "otimize o caso comum", que na utilização de CPUs Industriais significa o tempo de execução de uma operação de contato.

O tempo de ciclo é a outra métrica importante a ser destacada no conjunto de métricas propostas, já que ele influencia em grande parte das outras métricas. A influência do tempo de ciclo na métrica de tempo de reação é uma observação relevante, o que determina a qualidade do projeto de controle desenvolvido e também as limitações de tempo impostas pelo processamento para interação da CPU com o ambiente controlado propriamente dito. Com um tempo de ciclo mais curto, é possível reagir mais rapidamente a mudanças no ambiente, aumentando a segurança do processo de controle, o que é uma característica fundamental.

A análise das características dos programas utilizados em CPUs Industriais permitiu ainda confirmar as hipóteses de que as aplicações desenvolvidas para controle industrial não contêm muitos trechos em comum entre si, o que permite muito pouco reaproveitamento e reutilização de módulos de uma solução em outra. A menos da divisão em diversos módulos, onde cada módulo executa uma função específica de controle, há pouco, ou nada, de similar em aplicações distintas. Isso dificulta o desenvolvimento de *workloads* reais, e a alternativa de programas sintéticos é a única que se mostra viável.

Dessa forma, é apresentado um programa sintético capaz de caracterizar o desempenho da CPU. Os blocos principais do *workload* foram desenvolvidos a partir dos resultados da análise dos programas de controle, e contemplam as principais classes de instruções utilizadas em CPUs Industriais, tais como instruções de contato, de soma, de operações lógicas, de movimentação de operandos e de chamada de sub rotinas. Além disso, é sugerido um *mix* de instruções, de forma a confirmar os resultados obtidos com os blocos de instruções individuais e ao mesmo tempo permitir a avaliação de outras métricas, como o tempo de ciclo e o tempo de varredura e atualização. Devido à grande diferença entre as diversas aplicações de CPUs Industriais, o *mix* de instruções deve ser desenvolvido especialmente para cada caso de aplicação. O detalhamento do programa sintético, bem como os passos necessários para a sua execução e conseqüente obtenção dos resultados, estão apresentados em detalhes no anexo A e na seção 5.5.

Finalmente, discutiram-se técnicas a serem utilizadas para extração das métricas durante o processo de avaliação de desempenho. Foram apresentadas duas

TABELA 7.1 – As métricas desenvolvidas no trabalho

Tempo de Ciclo	alta	Tempo de Execução de Instruções
		Tempo de Varredura
		Tempo de Atualização
	média	Jitter de Processos Periódicos
	baixa	
Tempo de Programa	alta	Tempo de Execução de Instruções
	média	
	baixa	Chaveamento de Contexto
Tempo de Bit	alta	Tempo de Execução de Instruções
	média	
	baixa	Tempo de Chaveamento de Contexto
Tempo de Reação	alta	Tempo de Programa
		Tempo de Varredura
		Tempo de Atualização
	média	Eficiência do Protocolo de Campo
		Taxa de Transmissão / Vazão Máxima
		Jitter de Processos Periódicos
	baixa	Latência de Interrupções
		Tempo de Acesso à Rede
		Barramento Interno
Tempo de Varredura	alta	Tempo de Transmissão / Vazão Máxima
		Eficiência do Protocolo de Campo
	média	Tempo de Acesso à Rede
		Taxa de Perda de Pacotes
	baixa	Barramento Interno
Tempo de Atualização	alta	Tempo de Transmissão / Vazão Máxima
		Eficiência do Protocolo de Campo
	média	Tempo de Acesso à Rede
		Taxa de Perda de Pacotes
	baixa	Barramento Interno

técnicas para serem utilizadas neste trabalho. A primeira delas consiste na geração de um sinal de onda quadrada em um ponto de saída da CPU. Ao medir-se a freqüência do sinal, é possível estimar o tempo de ciclo instantâneo da CPU. Além disso, é possível obter-se diversas outras métricas indiretamente a partir do tempo de ciclo, modificando-se a arquitetura da CPU ou o tamanho do programa de controle. As variações que serão causadas na freqüência do sinal na saída por estas mudanças levam à obtenção destas métricas. A segunda técnica para monitoração das métricas envolve a utilização de *timers* de *software* no próprio equipamento para estimar o tempo gasto com determinadas operações ou tarefas de interesse.

7.1 Limitações e Aplicações

A principal limitação do *benchmark* apresentado aqui é a possível imprecisão dos resultados, como ocorre com processos de avaliação de desempenho baseados em medição direta e programas sintéticos. Mesmo assim, os dados extraídos a partir da execução do *benchmark* são úteis para o conhecimento das características do sistema e da sua aplicabilidade para o controle de um determinado ambiente.

Utilizando-se os jargões de Sistemas de Tempo Real, pode-se afirmar que os resultados obtidos não fornecem um *upper-bound*, ou um limite máximo para o tempo de execução de um programa de controle qualquer. Porém, o procedimento proposto é capaz de informar um tempo médio de execução, que pode ser utilizado com considerável confiança, como mostram os cálculos de intervalo de confiança dos estudos de caso.

De qualquer forma, não é o objetivo do trabalho obter um detalhamento apurado a respeito da característica de cada instrução, e nem um limite máximo para o tempo de execução, e sim uma visão geral das características do sistema que forneça uma estimativa das métricas e de onde possam estar possíveis pontos a serem otimizados na implementação de uma CPU Industrial.

7.2 Trabalhos Futuros

A continuidade deste trabalho pode se dar em várias direções. É interessante desenvolver um processo de análise mais apurado das características de programas de controle no ambiente industrial, já que a análise apresentada nesta dissertação foi pequena e baseada em poucos exemplos. Estimam-se que as características gerais apresentadas aqui devam se confirmar quando mais aplicações forem consideradas, mas este é um fato que deve ser trabalhado e confirmado.

O aprofundamento da análise de aplicações de controle pode levar também ao aperfeiçoamento do *workload* sintético apresentado no trabalho. A análise de novas situações, cargas e instruções a serem abordadas durante o processo de avaliação de desempenho também é deixada a título de trabalhos futuros.

A definição de *workloads* e técnicas de medição para as demais métricas apresentadas no capítulo 4 é mais um ponto a ser explorado. Uma parte das métricas para cada camada foi apresentada, e se optou por caracterizar apenas as relacionadas ao processo de controle pois estas incluiriam os efeitos das demais, além de serem as mais visíveis diretamente pelos usuários e as mais sentidas pelo ambiente controlado. Porém, ainda é de interesse que se possam analisar todas as partes da CPU Industrial individualmente, o que poderia levar à detecção de eventuais pontos a serem melhorados em cada um dos níveis de implementação do sistema de controle.

Ainda relacionado ao tema de métricas, também pode ser desenvolvido um estudo que permita um conhecimento mais profundo a respeito da influência de cada métrica de mais baixo nível nas métricas de controle. Seria interessante poder atribuir pesos a cada uma das métricas, que poderiam representar a influência, ou relação entre duas métricas.

Por fim, a avaliação de sistemas de controle distribuídos também poderia ser explorada. É visível o aumento das aplicações distribuídas no ambiente industrial, e nestes casos as operações de comunicação entre CPUs Industriais exercem um papel chave no desempenho do sistema. O aprofundamento das métricas de comunicação

apresentadas neste trabalho, bem como a estruturação de um *workload* para caracterização da influência da rede no processo controlado também são temas de interesse em trabalhos futuros relacionados ao trabalho descrito aqui.

Anexo 1 O Programa de Medição

Este anexo apresenta o programa desenvolvido para avaliação de desempenho dos tempos de instrução de CPUs Industriais. A seqüência de instruções gráficas apresentada corresponde ao programa para CPUs da Altus, e foi executado nas CPUs AL2003 e QK801. Para outros equipamentos, o programa deve ser adaptado caso seja necessário.

O programa deve ser transferido para a CPU utilizando as ferramentas apropriadas disponibilizadas pelo fabricante do equipamento. Os passos para transferência do programa são os mesmos que devem ser seguidos para transmissão de qualquer programa de controle, e podem variar entre diferentes CPUs. Consulte o manual do usuário do equipamento que está sendo avaliado para obter o procedimento correto.

Após a transferência, o programa deve ser colocado em execução, novamente seguindo o mesmo procedimento indicado para execução de programas de controle na CPU sendo avaliada. Para realizar a efetiva extração dos dados de desempenho de um dado equipamento, as seguintes variáveis do programa de avaliação devem ser ajustadas da forma apropriada:

- `TOT_CAL`: variável que indica quantas repetições do laço de calibração devem ser executadas a cada medição. Para facilidade de cálculos, aconselha-se que este valor coincida com os valores ajustados nas demais variáveis `TOT_XXX` listadas abaixo. Quanto maior o valor ajustado para `TOT_CAL`, maior a quantidade de repetições que serão executadas entre duas medições consecutivas de relógio. Assim, aumenta-se a precisão das medidas, porém aumenta também o tempo de execução do programa de avaliação.

- `TOT_MAX`: número de repetições da lógica com o número máximo de contatos que será medido pelo programa. Indica o número de instruções a ser executada entre duas medições consecutivas de relógio. Cada repetição corresponde à execução de 32 contatos.

- `TOT_MAX`: número de repetições da lógica com o número mínimo de contatos que será medido pelo programa. Indica o número de instruções a ser executada entre duas medições consecutivas de relógio. Cada repetição corresponde à execução de 2 contatos.

- `TOT_ADD`: número de repetições da lógica com instruções de soma que será medido pelo programa. Indica o número de instruções a ser executada entre duas medições consecutivas de relógio. Cada repetição corresponde à execução de 4 somas e 3 subtrações.

- `TOT_LOG`: número de repetições da lógica com instruções de operações lógicas que será medido pelo programa. Indica o número de instruções a ser executada entre duas medições consecutivas de relógio. Cada repetição corresponde à execução de 4 operações AND e 3 operações OR.

- `TOT_MOV`: número de repetições da lógica com movimentação de operandos que será medido pelo programa. Indica o número de instruções a ser executada entre duas medições consecutivas de relógio. Cada repetição corresponde à execução de 14 movimentações de operandos.

- `TOT_CHP`: número de repetições da lógica com chamada de procedimento que será medido pelo programa. Indica o número de instruções a ser executada entre duas medições consecutivas de relógio. Cada repetição corresponde à execução de 14 chamadas de procedimento.

- `EN_ADD`: variável booleana que controla a condição de execução das instruções de soma. Caso a variável esteja em `FALSE`, as operações estão desabilitadas, caso contrário elas estarão habilitadas. Deve ser utilizada para extrair as métricas de tempo de execução de instrução de soma quando as mesmas estão habilitadas ou desabilitadas.

- `EN_LOG`: variável booleana que controla a condição de execução das instruções lógicas. Caso a variável esteja em `FALSE`, as operações estão desabilitadas, caso contrário elas estarão habilitadas. Deve ser utilizada para extrair as métricas de tempo de execução de instrução de instruções lógicas quando as mesmas estão habilitadas ou desabilitadas.

- `EN_MOV`: variável booleana que controla a condição de execução das instruções de movimentação de operandos. Caso a variável esteja em `FALSE`, as operações estão desabilitadas, caso contrário elas estarão habilitadas. Deve ser utilizada para extrair as métricas de tempo de execução de instrução de movimentação de operandos quando as mesmas estão habilitadas ou desabilitadas.

- `EN_CHP`: variável booleana que controla a condição de execução das instruções de chamada de procedimento. Caso a variável esteja em `FALSE`, as operações estão desabilitadas, caso contrário elas estarão habilitadas. Deve ser utilizada para extrair as métricas de tempo de execução de instrução de chamada de procedimentos quando as mesmas estão habilitadas ou desabilitadas.

- `DO_RUN`: variável booleana que indica que o programa deve começar a executar as medições. Deve ser ajustada para `TRUE` após todas as demais variáveis terem sido especificadas. Quando esta variável está em `FALSE`, o programa de avaliação está sendo executado, porém nenhuma medida é feita e todas as lógicas, à exceção da primeira, são ignoradas. Quando ajustado para `TRUE`, a medição dos tempos e a repetição das lógicas é executada, conforme especificado pelas variáveis descritas anteriormente. Ao final de uma execução completa de avaliação, a variável `DO_RUN` é ajustada para `FALSE` automaticamente, para indicar que os dados estão disponíveis para serem coletados.

O resultado da execução do programa de avaliação estará disponível nas estruturas `CLK_CAL`, `CLK_MAX`, `CLK_MIN`, `CLK_ADD`, `CLK_LOG`, `CLK_MOV` e `CLK_CHP`. Estas apresentarão o tempo médio gasto em cada conjunto de repetições dos grupos de instruções.

A.1 Exemplo de utilização

Para ilustrar a utilização passo a passo do programa de avaliação proposto, detalhou-se o procedimento necessário para execução na UCP AL2003. Os passos a serem seguidos para obtenção dos resultados são os seguintes:

1. Conectar a unidade de programação ao AL2003 através do cabo de comunicação serial.

2. Com o auxílio da unidade de programação da CPU AL2003, a ferramenta *MasterTool* fornecida pela ALTUS, realizar o envio do programa de avaliação para o AL2003, com o comando "*Enviar Programa*".

3. Com o recurso de monitoração de variáveis do *MasterTool*, observar os valores de todas as variáveis de controle do programa de avaliação.

4. Neste exemplo, deseja-se medir o tempo de instrução de soma/subtração. Portanto, deve-se ajustar as variáveis `TOT_CAL` e `TOT_ADD` para o valor 1000, de forma a executar 1000 vezes o laço de calibração e o laço de instruções de soma/subtração.

5. Ajustar todas as demais variáveis `TOT_XXX` para o valor '0', já que a única métrica de interesse para o exemplo é o tempo de soma/subtração.

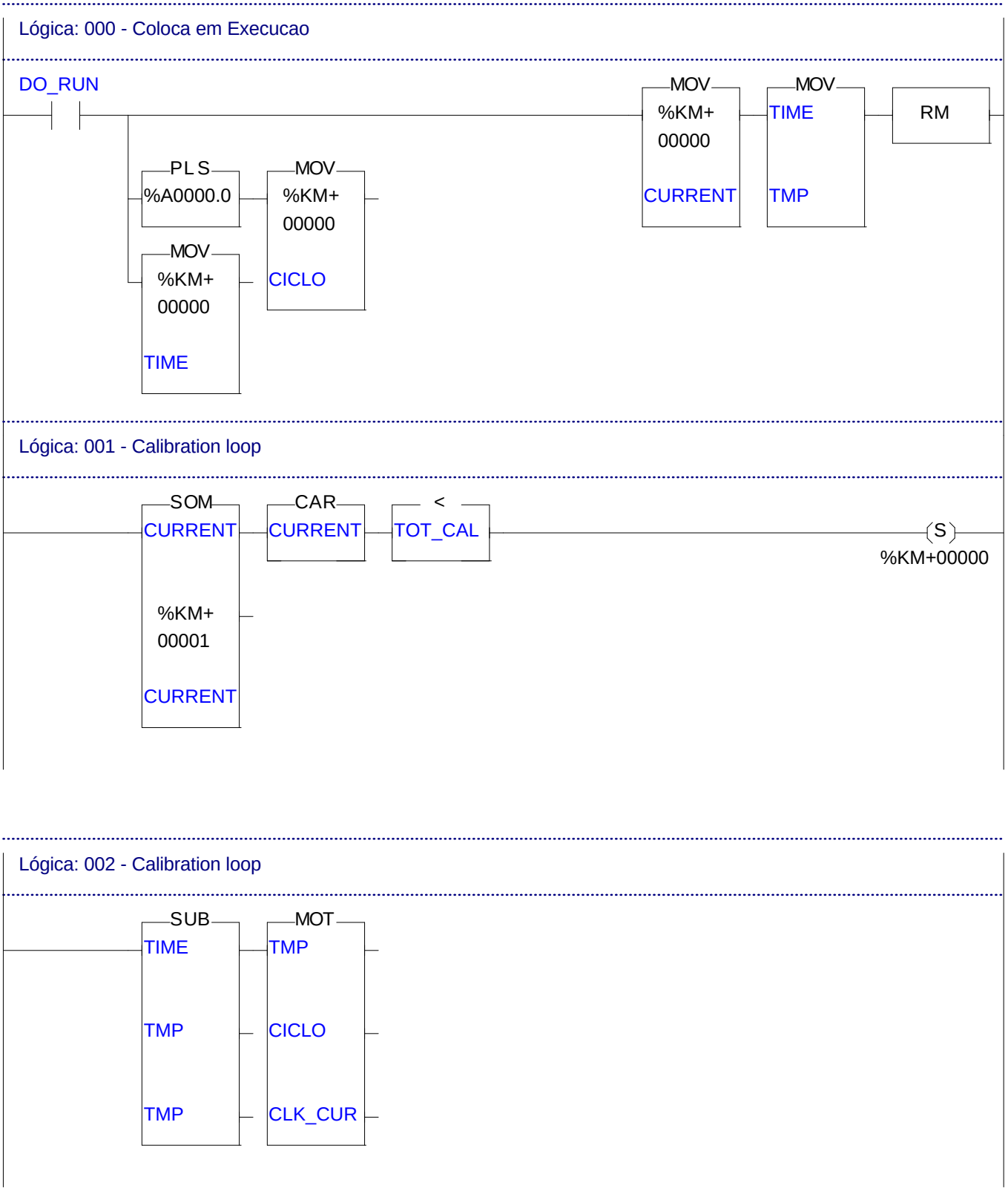
6. Ativar o início das medições modificando o valor da variável `DO_RUN` de '0' para '1'.

7. Aguardar o término das medições, o que é indicado pelo retorno da variável `DO_RUN` para o valor '0'.

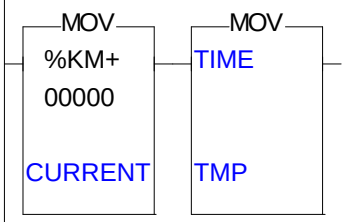
8. Para obter o resultado para a execução atual, observar o valor das variáveis `CLK_CAL` e `CLK_ADD` com a ferramenta *MasterTool*. A primeira conterá o tempo gasto com as instruções de controle do laço de medição, enquanto a segunda conterá o tempo gasto com o controle do laço e execução das instruções de interesse. O tempo gasto com as instruções de soma/subtração será obtido pela subtração de `CLK_ADD` e `CLK_CAL`.

9. Os passos anteriores devem ser repetidos até que o intervalo de confiança de 95% seja menor que 1% do tempo da instrução, desde que o limite mínimo de 10 execuções tenha sido atingido.

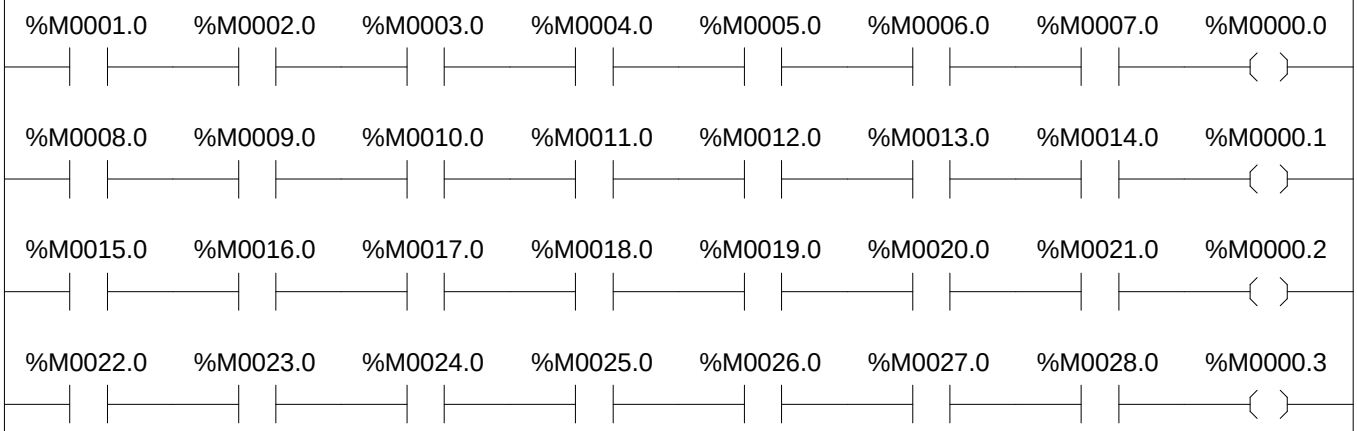
A.2 Código Fonte



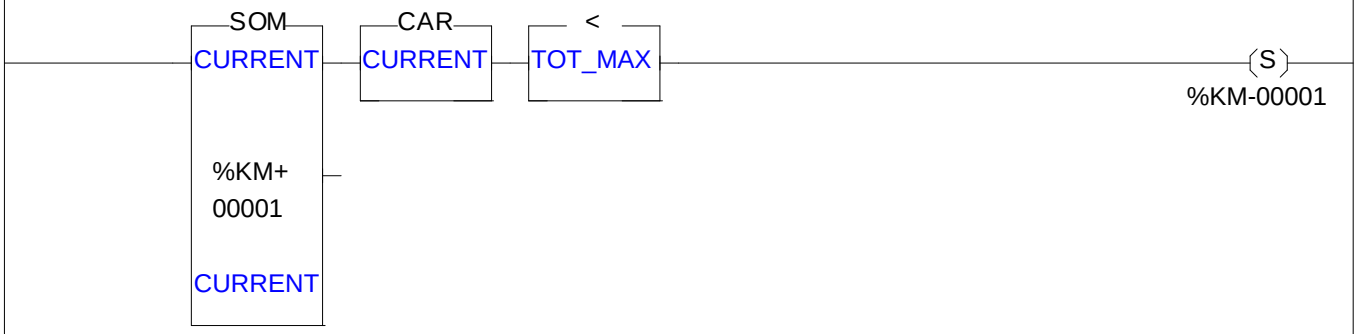
Lógica: 003 - Timers para lógica completa



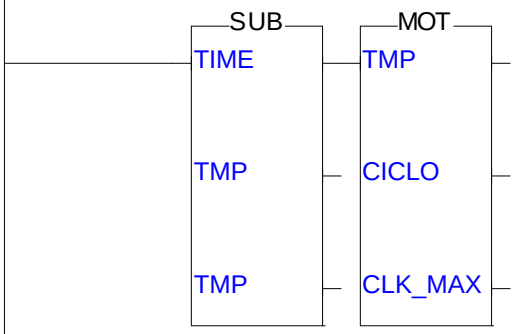
Lógica: 004 -



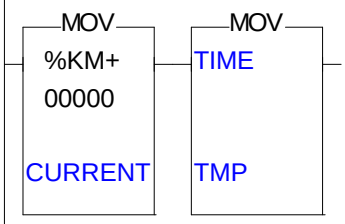
Lógica: 005 - Loop para Lógica Completa



Lógica: 006 - Loop para Lógica Completa

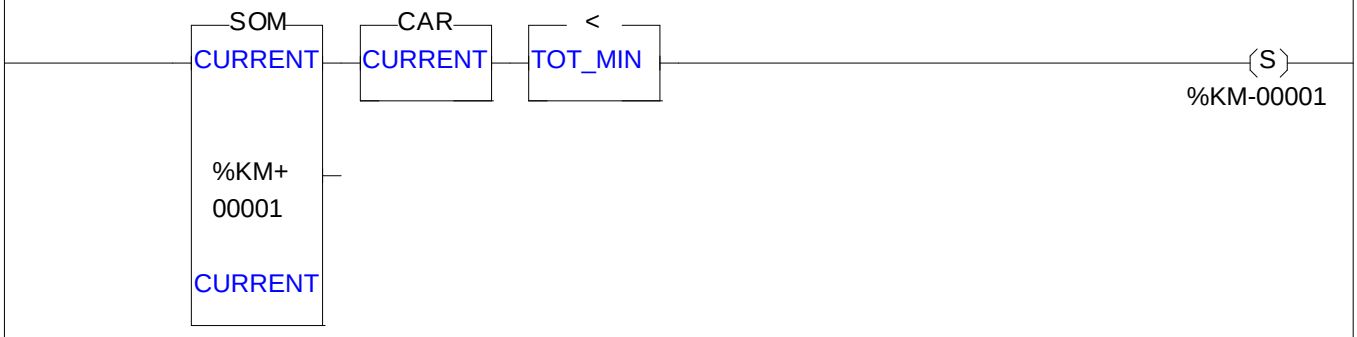


Lógica: 007 -

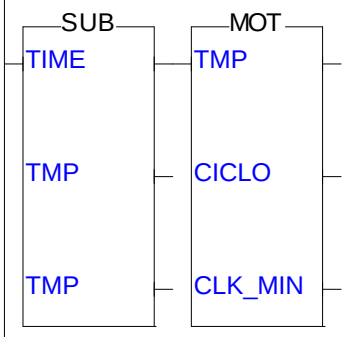


Lógica: 008 - Lógica com mínimo de contatos

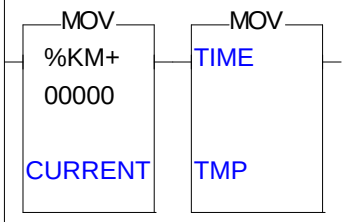
Lógica: 009 - Lógica com mínimo de contatos



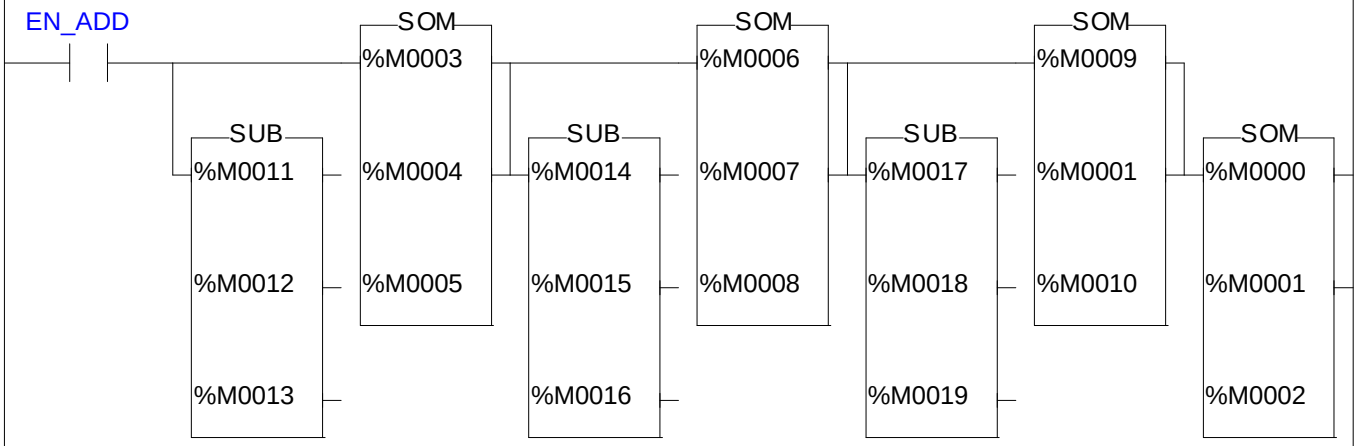
Lógica: 010 -



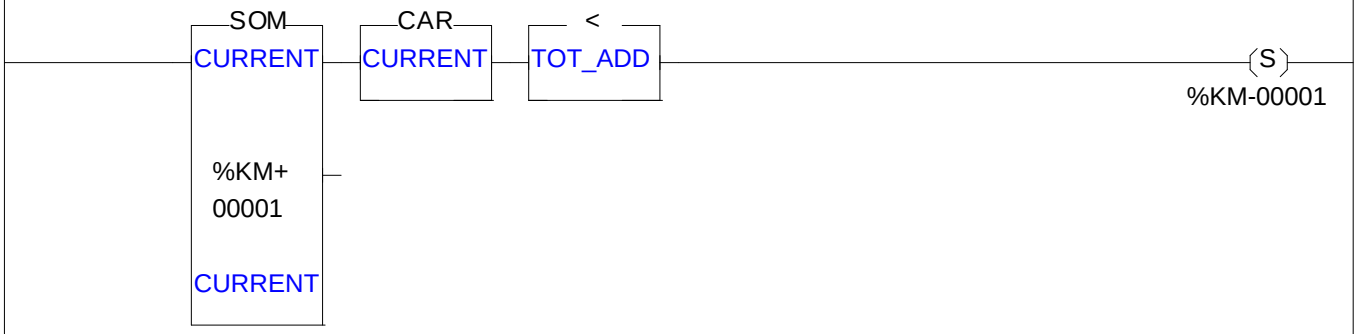
Lógica: 011 -



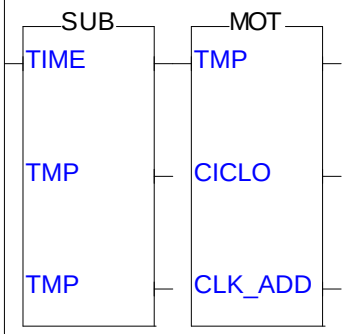
Lógica: 012 - Lógica com soma/subtração



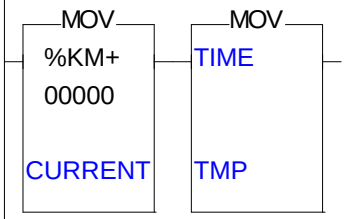
Lógica: 013 - Lógica com soma/subtração



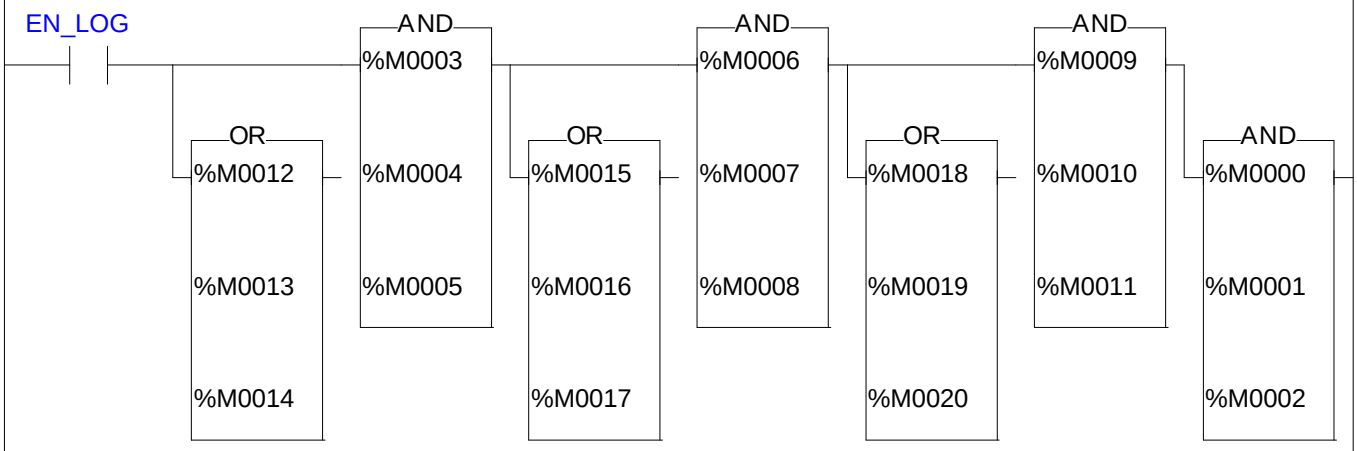
Lógica: 014 -



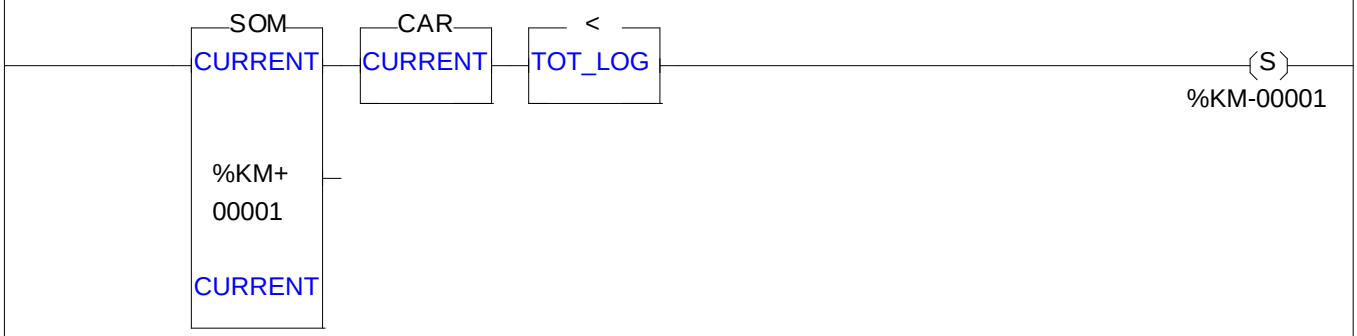
Lógica: 015 -



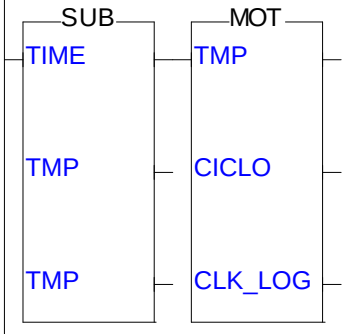
Lógica: 016 - Lógica com and/or



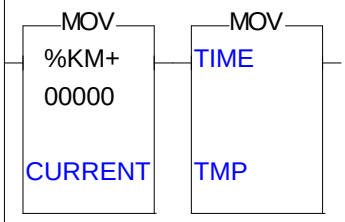
Lógica: 017 - Lógica com and/or



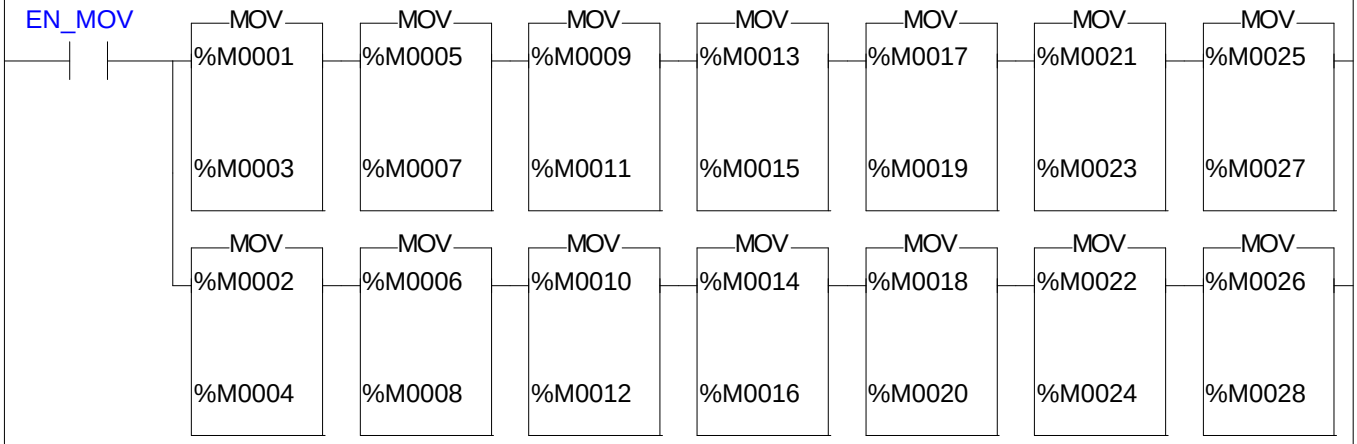
Lógica: 018 -



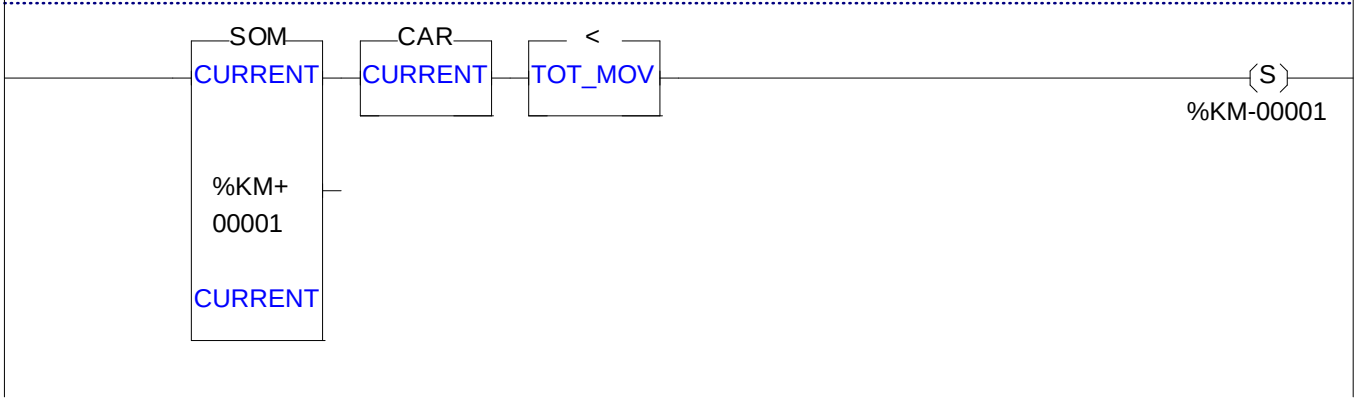
Lógica: 019 -



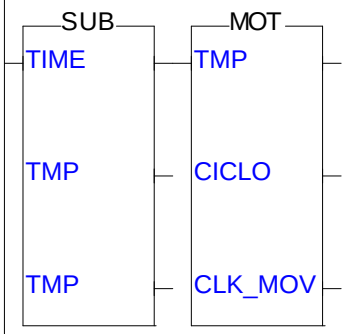
Lógica: 020 - Lógica com MOV



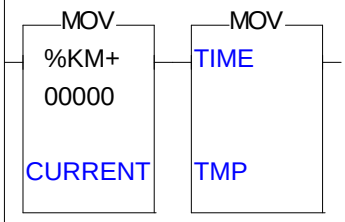
Lógica: 021 - Lógica com MOV



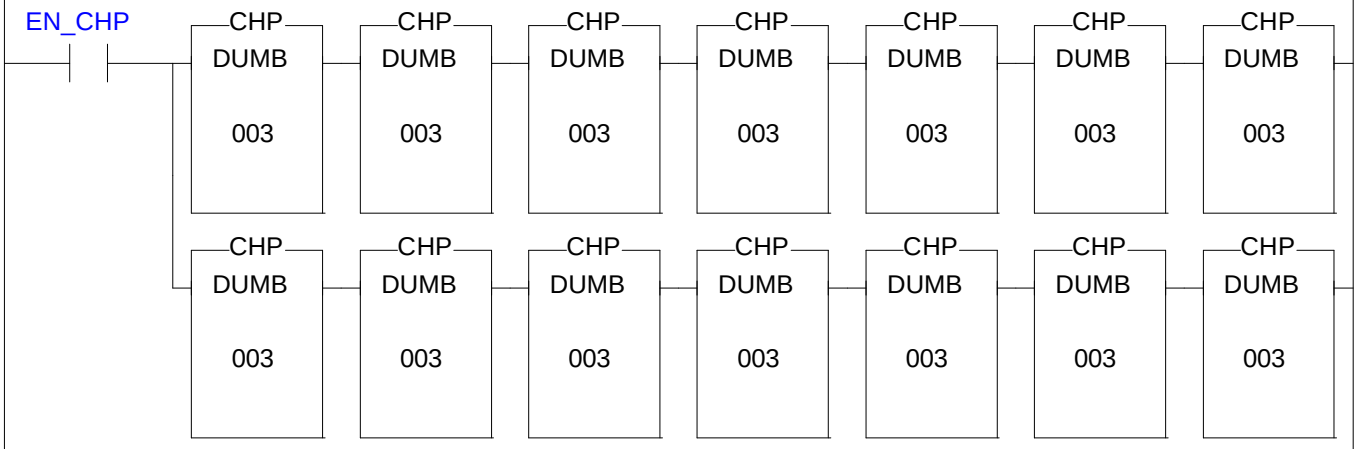
Lógica: 022 -

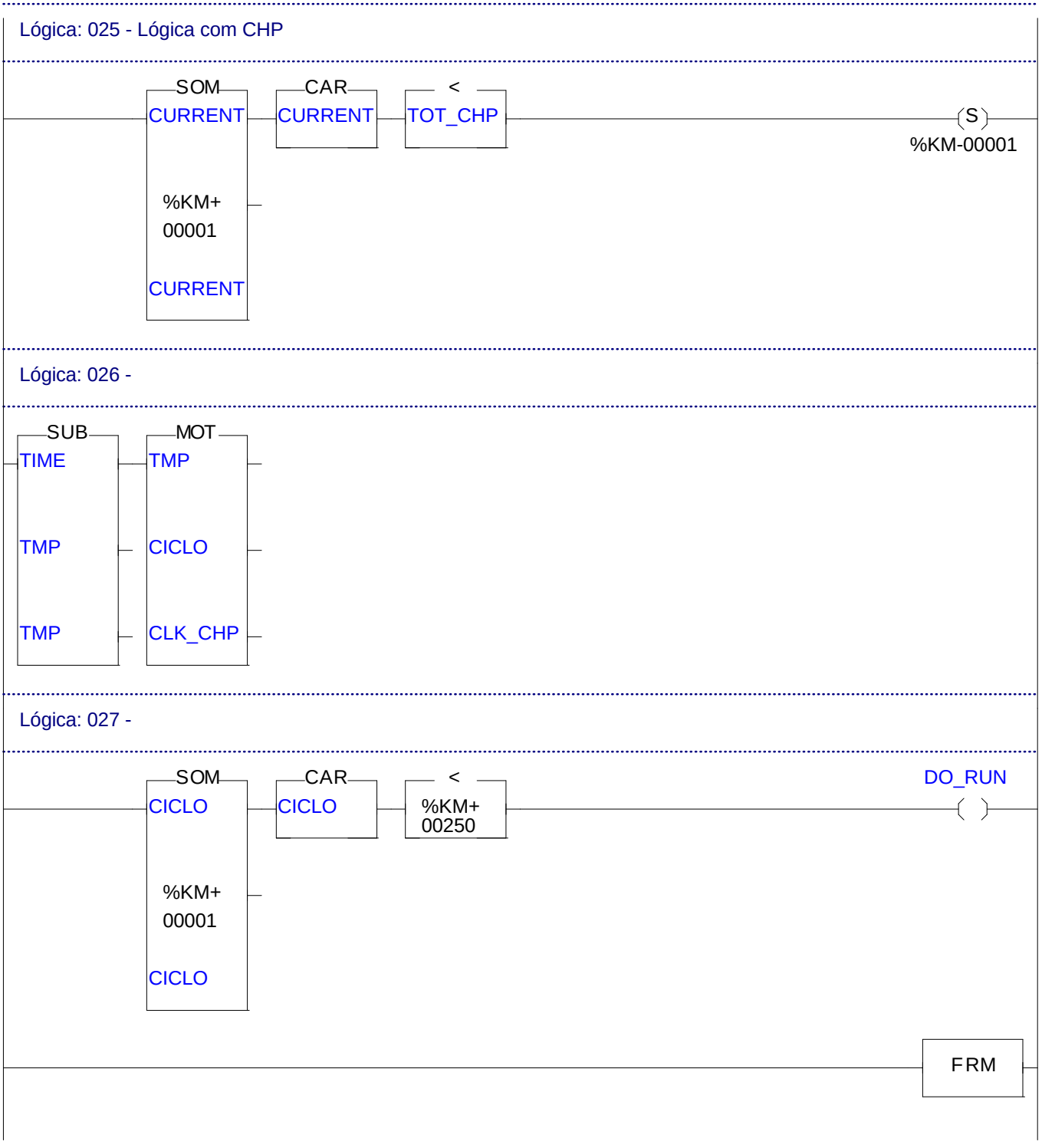


Lógica: 023 -

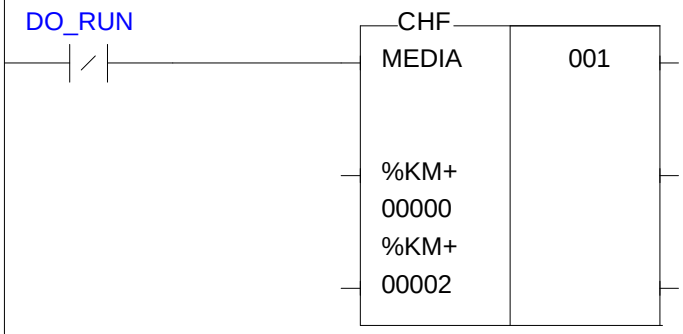


Lógica: 024 - Lógica com CHP




Lógica: 025 - Lógica com CHP
SOM
CURRENT
%KM+
00001
CURRENT
CAR
CURRENT
<
TOT_CHP
S
%KM-00001
Lógica: 026 -
SUB
TIME
TMP
TMP
MOT
TMP
CICLO
CLK_CHP
Lógica: 027 -
SOM
CICLO
%KM+
00001
CICLO
CAR
CICLO
<
%KM+
00250
DO_RUN
FRM

Lógica: 028 - Computar resultados



Bibliografia

[ALT 98] ALTUS SISTEMAS DE INFORMÁTICA S.A. **Características Técnicas CPU QK801**. Porto Alegre, 1998. (CT-6106-100.0 rev. K).

[ALT 98a] ALTUS SISTEMAS DE INFORMÁTICA S.A. **Características Técnicas CPU AL2003**. Porto Alegre, 1998. (CT-6104-111.0 rev. F).

[BON 97] BONFATTI, F; MONARI, P; SAMPIERI, U. **IEC1131-3 Programming Methodology**. França: CJ International, 1997. 377p.

[BOU 96] BOUCHARD, J. E. Programmable Logic Controllers. In: ALBERT, C. L; COGGAN, D. A (Ed.). **Fundamentals of Industrial Control – Pratical Guides for Measurement and Control**. EUA: Instrumental Society of America, 1996. p.537–623.

[BRA 99] BRADNER, S; MCQUAID, J. **Benchmarking Methodology for Network Interconnection Devices**, RFC2544. [S.l.], 1999.

[BRA 91] BRADNER, S. **Benchmarking Terminology for Network Interconnection Devices**, RFC1242. [S.l.], 1991.

[BUR 97] BURNS, A; WELLINGS, A. **Real Time Systems and Programming Languages**. 2nd. ed. Harlow, England: Addison-Wesley, 1997. 611p.

[CHR 2000] CHRYSLER CORPORATION; FORD MOTOR COMPANY; GENERAL MOTORS CORPORATION. **Requirements of Open, Modular Architecture Controllers for Applications in the Automotive Industry**. Disponível em: <http://www.arcweb.com/omac/Techdocs/omacv11.htm> Acesso em: dez. 2000.

[CON 2000] CONTROLLER IN AUTOMATION. **Controller in Automation (CiA) — Controller Area Networks (CAN)**. Disponível em: <http://www.can-cia.de> Acesso em: nov. 2000.

[CUR 76] CURNOW, H. J; WICHMANN, B. A. A Synthetic Benchmark. **The Computer Journal**, New York, v.19, n.1, p.43–49, Feb. 1976.

[DUN 2001] DUNN, J; MARTIN, C. **Terminology for Frame Relay Benchmarking**, RFC3133. [S.l.], 2001.

[FER 83] FERRARI, D; SERAZZI, G; ZEIGNER, A. **Measurement and Tuning of Computer Systems**. New Jersey, EUA: Prentice Hall, 1983. 523p.

[GGH 2000] GGH MARKETING COMMUNICATIONS. **Industrial Ethernet**:The Online Industrial Ethernet Book Resource. Disponível em: <http://ethernet.industrial-networking.com> Acesso em: ago. 2000.

[GIU 2001] GIUSTO, P; MARTIN, G; HARDCOURT, E. Reliable Estimation of Execution Time of Embedded Software. In: PROCEEDINGS OF DESIGN, AUTOMATION AND TEST IN EUROPE, 2001, Orlando, EUA. **Anais...** New York: IEEE Computer Society, 2001. p.580–587.

[GUN 98] GUNTHER, N. **The Pratical Performance Analyst**. New York, EUA: McGraw-Hill, 1998. 432p.

[HAR 2000] HARINATH, R et al. Experiences with an Object Oriented Framework for Distributed Control Applications. **ACM Computing Surveys**, New York, v.32, n.1, Mar. 2000.

[HEN 2000] HENNING, J. L. CPU Performance in the New Millenium. **Computer**, New York, v.33, n.7, p.28–35, July 2000.

[IEC 93] IEC. **Programmable controllers — Part 3**: Programming languages, n.1131-3. Suiça, 1993.

[JAI 91] JAIN, R. **The Art of Computer Systems Performance Analysis — Techniques for Experimental Design, Measurement, Simulation, and Modeling**. New York, EUA: John Wiley & Sons, 1991. 685p.

[JON 83] JONES, C. T; BRYAN, L. A. **Programmable Controllers — Concepts and Applications**. EUA: International Programmable Controls Inc., 1983. 329p.

[KIM 92] KIM, J; PARK, J; KWON, W. H. Architecture of a Ladder Solving Processor for Programmable Controllers. **Microprocessors and Microsystems**, Oxford, v.16, n.7, p.369–379, Sept. 1992.

[KRI 97] KRISHNA, C; SHIN, K. **Real Time Systems**. EUA: McGraw-Hill, 1997. 446p.

[LI 99] LI, Y.-T. S; MALIK, S; WOLFE, A. Performance Estimation of Embedded Software with Instruction Cache Modeling. **ACM Transactions on Design Automation of Eletronic Systems**, New York, v.4, n.3, p.257–279, July 1999.

[MAT 2001] MATHIS, M; ALLMAN, M. **A Framework for Defining Empirical Bulk Transfer Capacity Metrics**, RFC3148. [S.l.], 2001.

[MIY 96] MIYAGI, P. E. **Controle Programável — Fundamentos do Controle de Sistemas a Eventos Discretos**. São Paulo: E. Blücher, 1996. 194p.

[MOR 82] MORRIS, M; ROTH, P. **Computer Performance Evaluation — Tools and Techniques for Effective Analysis**. New York, EUA: Van Nostrand Reinhold Company, 1982. 260p.

[NAC 2000] NACUL, A. C. **Métricas de Desempenho de CPUs Industriais**. 2000. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[OLI 2000] OLIVEIRA, R. S de; FARINES, J.-M; SILVA FRAGA, J da. **Sistemas de Tempo Real**. São Paulo, Brasil: IME-USP, 2000. 201p.

[PAX 98] PAXSON, V et al. **Framework for IP Performance Metrics**, RFC2330. [S.l.], 1998.

[PRO 99] PROFIBUS NUTZERORGANISATION E.V. **Profibus Technical Description**. Karlsruhe, [Alemanha], 1999. (Profibus Brochure, n.4002) Disponível em: <http://www.profibus.com/data/technic> Acesso em: 15 nov. 2000.

[RHO 95] RHO, G. S et al. Implementation of a RISC Microprocessor for Programmable Logic Controllers. **Microprocessors and Microsystems**, Oxford, v.19, n.10, p.599–607, Dec. 1995.

[SED 2001] SEDIGH-ALI, S; GHAFOOR, A; PAUL, R. A. Software Engineering Metrics for COTS-Based Systems. **Computer**, New York, v.34, n.5, p.44–50, May 2001.

[SHA 2001] SHALUNOV, S; TEITELBAUM, B; ZEKAUSKAS, M. **A One-way Delay Measurement Protocol**. [S.l.]: IETF, 2001. Internet Draft – Expires April 2002.

[STA 88] STANKOVIC, J. A. Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems. **Computer**, New York, v.21, n.10, p.10–19, Oct. 1988.

[STA 96] STANKOVIC, J et al. **Real-Time Computing — A Critical Enabling Technology**. Disponível em: <ftp://ftp.cs.umass.edu/pub/ccs/spring/critenable.ps>. Acesso em: 2000.

[STA 96a] STANKOVIC, J. Real-Time and Embedded Systems. **ACM Computing Surveys**, New York, v.28, n.4, p.751–763, Dec. 1996. Strategic Directions in Computing Research — Real Time Working Group. Disponível em: <http://www-ccs.cs.umass.edu/sdcr/rt.ps>. Acesso em: 22 dez. 2000.

[TOK 99] TOKHI, M. Special Issue on High Performance Real-Time Computing — Editorial. **Microprocessors and Microsystems**, Surrey, v.23, n.6, p.307, Nov. 1999.

[WEI 84] WEICKER, R. P. Dhrystone: a Synthetic Systems Programming Benchmark. **Communications of the ACM**, New York, v.27, n.10, p.1013–1030, Oct. 1984.