Universidade Federal de Santa Catarina
Curso De Pós-Graduação em Ciência da Computação

Um Ambiente de Desenvolvimento FORTH, para Sistemas Dedicados e Controle Difuso

por

Marcelo Moraes de Almeida

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação.
Orientador: Prof. Dr. João Bosco da Mota Alves

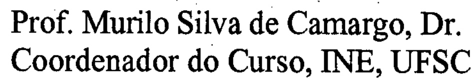
Florianópolis, novembro de 1996.

Um Ambiente de Desenvolvimento FORTH, para Sistemas Dedicados e Controle Difuso.

Marcelo Moraes de Almeida

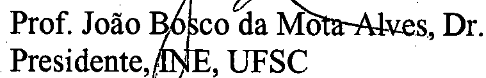
Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Especialidade Sistemas de Computação, e aprovada em sua forma final pelo Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

Prof. João Bosco da Mota Alves, Dr.
Orientador, INE, UFSC

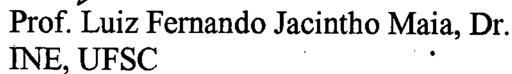


Prof. Murilo Silva de Camargo, Dr.
Coordenador do Curso, INE, UFSC

Banca Examinadora:



Prof. João Bosco da Mota Alves, Dr.
Presidente, INE, UFSC



Prof. Luiz Fernando Jacintho Maia, Dr.
INE, UFSC

Prof. Ubiratan Holanda Bezerra, Dr.
DEE, UFPa

AGRADECIMENTOS

Agradeço a meus pais, irmãos, e parentes por tudo que contribuíram para a concretização deste trabalho.

Ao Hermann (prof. Hermann Adolf Harry Lück, falecido em maio de 1995) pela oportunidade de realizar este trabalho e iniciar uma nova fase na minha vida profissional, como também pelas contribuições que tem prestado ao meu trabalho.

Ao prof. Bosco e prof. Maia (CPGCC/UFSC) que acreditaram no trabalho iniciado, e se propuseram a dar continuidade.

Aos colegas, professores, e funcionários da UFSC que acreditaram e incentivaram a continuidade deste trabalho, nos momentos difíceis que surgiram.

Ao pessoal do LABSOLDA, por darem oportunidade de aplicar o trabalho desenvolvido nesta dissertação, antes mesmo de tê-lo concluído.

SUMÁRIO

LISTA DE FIGURAS .... **VII**

LISTA DE QUADROS .... **IX**

RESUMO .... **XI**

ABSTRACT .... **XII**

I. INTRODUÇÃO **1**

I.1 INTRODUÇÃO 1

I.2 OBJETIVOS DO TRABALHO 4

I.3 ESTRUTURA DO TRABALHO 4

II. SISTEMAS PARA AUTOMAÇÃO E CONTROLE DE PROCESSOS **6**

II.1 SISTEMAS DE CONTROLE 6

II.1.1 Introdução 6

II.2 SISTEMAS DEDICADOS 9

II.2.1 Desenvolvimento de *Software* Dedicado 10

II.3 SISTEMAS EM TEMPO REAL 12

II.4 CONTROLE DIFUSO 13

II.4.1 Introdução ... 13
II.4.2 Controlador Lógico Difuso ... 14
II.5 LINGUAGEM FORTH ... 24
II.5.1 Introdução ... 25
II.5.2 A Arquitetura FORTH ... 27
II.5.3 O Dicionário FORTH ... 46
II.5.4 O Interpretador de Texto ... 47
II.5.5 Compilação ... 48
II.5.6 eFORTH ... 52

III. MODELO DO SISTEMA PROPOSTO ... 54
III.1 INTRODUÇÃO ... 54
III.2 ESTRUTURA DO SISTEMA ... 55
III.3 UTILIZAÇÃO DO SISTEMA ... 56
III.4 ALGORITMO E LINGUAGEM ... 58
III.4.1 O Algoritmo de Controle ... 58
III.4.2 Definição do Algoritmo ... 60
III.5 ESTRUTURA DO PROGRAMA DE CONTROLE ... 68

IV. IMPLEMENTAÇÃO E SIMULAÇÃO ... 70
IV.1 INTRODUÇÃO ... 70
IV.2 O SISTEMA SIMULADO ... 70
IV.3 MODELAGEM DIFUSA DO SISTEMA ... 72
IV.4 ESTRUTURA DO SISTEMA ... 73
IV.5 UTILIZAÇÃO DO SISTEMA ... 74
IV.6 O PROGRAMA DE CONTROLE ... 77
IV.6.1 Entrada de Dados ... 78
IV.6.2 Variáveis de Entrada: Universo de Discurso e Conjuntos Difusos ... 78
IV.6.3 Variável de Saída: Universo de Discurso e Conjuntos Difusos ... 79
IV.6.4 Fuzzyficação ... 79
IV.6.5 Base de Regras ... 80
IV.6.6 Defuzzyficação ... 80
IV.6.7 Saída de Dados ... 81

IV.6.8 *Loop* de Controle 81
IV.6.9 Resultados Obtidos 83

V. CONCLUSÕES E RECOMENDAÇÕES 89

V.1 PROPOSTA PARA FUTUROS TRABALHOS 89

LISTA DE FIGURAS

FIGURA II-A ESTRUTURA GERAL DE UM SISTEMA DE CONTROLE 7
FIGURA II-B ESTRUTURA GERAL DE UM SISTEMA COM CONTROLADOR DIGITAL (GOPAL, 1984). 8
FIGURA II-C COMPUTADOR PESSOAL E SISTEMA DEDICADO (TING, 1990). 9
FIGURA II-D ARQUITETURA BÁSICA DE UM SISTEMA DEDICADO 10
FIGURA II-E ESTRUTURA BÁSICA DE UM CONTROLADOR LÓGICO DIFUSO (CLD) 14
FIGURA II-F UNIVERSO DE DISCURSO 16
FIGURA II-G VARIÁVEL LINGUÍSTICA TEMPERATURA 17
FIGURA II-H FUNÇÕES DE PERTINÊNCIA MAIS COMUNS. 19
FIGURA II-I INFERÊNCIA DIFUSA 22
FIGURA II-J UTILIZAÇÃO DE SINGLETONS 23
FIGURA II-K CÁLCULO DO VALOR CRISP A PARTIR DE SINGLETONS. 23
FIGURA II-L INTERPRETADOR INTERNO PARA CÓDIGO DIRETAMENTE ALINHAVADO. 32
FIGURA II-M INTERPRETADOR INTERNO PARA CÓDIGO INDIRETAMENTE ALINHAVADO. 32
FIGURA II-N INTERPRETADOR INTERNO PARA CÓDIGO INDIRETAMENTE ALINHAVADO POR ÍNDICE 33
FIGURA II-O CÓDIGO INDIRETAMENTE ALINHAVADO. 35
FIGURA II-P ESTRUTURAÇÃO DE DEFINIÇÕES PARA A AVALIAÇÃO DO POLINÔMIO. 42
FIGURA II-Q O DICIONÁRIO FORTH. 48
FIGURA II-R EXEMPLO DE COMPILAÇÃO. 51
FIGURA II-S INTERPRETADOR FORTH. NAS LINHAS PONTILHADAS ESTÃO AS FUNÇÕES DE COMPILAÇÃO. 52
FIGURA III-A ESTRUTURA DO SISTEMA PROPOSTO. 56
FIGURA III-B FUNÇÕES DE PERTINÊNCIA MAIS COMUNS. 64

FIGURA IV-A GUINDASTE MOVIMENTADOR DE CARGAS.......... 71
FIGURA IV-B VARIÁVEIS LINGUÍSTICAS E CONJUNTOS DIFUSOS DO SISTEMA (ALTROCK, 1993).......... 72
FIGURA IV-C MATRIZ LINGUÍSTICA.......... 72
FIGURA IV-D ESTRUTURA DO SISTEMA.......... 73
FIGURA IV-E SISTEMA IMPLEMENTADO.......... 74
FIGURA IV-F HARDWARE DO SISTEMA DE CONTROLE - PLACA VPORT-25K.......... 75

LISTA DE QUADROS

QUADRO II-A 17
QUADRO II-B ESTRUTURA DE UM PROGRAMA EM LINGUAGEM DE ALTO NÍVEL 27
QUADRO II-C CÓDIGO DIRETAMENTE ALINHAVADO. 28
QUADRO II-D INTERPRETADOR INTERNO PARA CÓDIGO DIRETAMENTE ALINHAVADO. 28
QUADRO II-E INTERPRETADOR INTERNO ESCRITO DE MODO SIMPLIFICADO. 29
QUADRO II-F CODIGO INDIRETAMENTE ALINHAVADO 30
QUADRO II-G INTERPRETADOR INTERNO PARA CÓDIGO INDIRETAMENTE ALINHAVADO. 31
QUADRO II-H COMPARAÇÃO DOS INTERPRETADORES INTERNOS 34
QUADRO II-I EXEMPLO DE UTILIZAÇÃO DA PILHA DE PARÂMETROS. 36
QUADRO II-J PRÓLOGO PARA UMA CONSTANTE. 38
QUADRO II-K PRÓLOGO PARA UMA VARIÁVEL. 38
QUADRO II-L DEFINIÇÃO DE '@' - AT. 39
QUADRO II-M DEFINIÇÃO DE '!' - STORE. 39
QUADRO II-N PRÓLOGO PARA UMA ESTRUTURA DE DADOS "ARRAY". 40
QUADRO II-O EXEMPLO DE DESVIO CONDICIONAL DENTRO DE UMA DEFINIÇÃO 43
QUADRO II-P DESVIO CONDICIONAL. 43
QUADRO II-Q CHAMADA CONDICIONAL DE SUBROTINA. 44
QUADRO II-R PRÓLOGO PARA CHAMADA CONDICIONAL DE SUBROTINA. 44
QUADRO II-S CRIAÇÃO DE UMA DEFINIÇÃO FORTH CHAMADA POLINOMIO 50
QUADRO III-A ALGORITMO DE CONTROLE. 59

QUADRO III-B ETAPAS DO PROGRAMA DE CONTROLE. 59
QUADRO III-C LOOP DE CONTROLE EM FORTH........ 60
QUADRO III-D ETAPAS DO LEVANTAMENTO DE DADOS. 62
QUADRO III-E DECLARAÇÃO DAS FUNÇÕES DE PERTINÊNCIA........ 64
QUADRO III-F DEFINIÇÃO DE SINGLETON. 65
QUADRO III-G DECLARAÇÃO DA FUZZIFICAÇÃO........ 65
QUADRO III-H ALGORITMO DE AVALIAÇÃO DE REGRAS. 66
QUADRO III-I DECLARAÇÃO DE UMA REGRA. 67
QUADRO III-J DECLARAÇÃO DA DEFUZZYFICAÇÃO........ 67
QUADRO III-K PALAVRAS DO LOOP DE CONTROLE. 68
QUADRO III-L CONTEÚDO DO BLOCO DE DECLARAÇÕES. 69
QUADRO III-M CONTEÚDO DO BLOCO DE PROGRAMA. 69
QUADRO IV-A CONTEÚDO DOS BLOCOS DA ESTRUTURA DO SISTEMA........ 74
QUADRO IV-B DECLARAÇÃO DAS PALAVRAS DE INICIALIZAÇÃO E ENTRADA DE DADOS. 78
QUADRO IV-C DECLARAÇÃO DAS VARIÁVEIS LINGUÍSTICAS DE ENTRADA. 78
QUADRO IV-D DECLARAÇÃO DA VARIÁVEL LINGUÍSTICA DE SAÍDA. 79
QUADRO IV-E DECLARAÇÃO DA FUZZYFICAÇÃO........ 80
QUADRO IV-F BASE DE REGRAS. 80
QUADRO IV-G DECLARAÇÃO DA DEFUZZYFICAÇÃO. 80
QUADRO IV-H PALAVRAS DE SAÍDA DE DADOS........ 81
QUADRO IV-I DECLARAÇÃO DO LOOP DE CONTROLE. 82
QUADRO IV-J VALORES DE ENTRADA PARA SIMULAÇÃO. 83
QUADRO IV-K VALORES DE SAÍDA DO SISTEMA. 87

ANEXO A .... LISTAGEM eFORTH
ANEXO B .... DEFINIÇÕES FUZZY
ANEXO C .... PROGRAMA DE CONTROLE
ANEXO D........ DECLARAÇÕES DAS VARIÁVEIS
ANEXO E .... SISTEMA DE DESENVOLVIMENTO DA TOGAI
ANEXO F .... MICROCONTROLADOR NEC V25
BIBLIOGRAFIA

RESUMO

A necessidade de sistemas automáticos de controle nas indústrias e na vida moderna é crescente. A utilização dos computadores digitais para realizar tarefas de controle está consolidada, e em contínuo crescimento. As técnicas de controle convencionais como a modelagem matemática de sistemas continuam em emprego, mas disputam lugar com técnicas inovadoras, como o Controle Difuso e o emprego da Inteligência Artificial, tornando os sistemas de controle modernos ainda muito versáteis. O tamanho dos computadores foram reduzidos de modo a poderem ser encaixados em pequenos espaços nos equipamentos mecânicos como sensores, atuadores, etc, e também por necessidades de funcionamento. Os computadores se tornaram pequenos sistemas dedicados á tarefas específicas.

Baixo custo dos microcontroladores e a relativa facilidade de implementação de um algoritmo de controle difuso em um sistema digital, impulsionaram o emprego do controle difuso em sistemas de controle digitais, e hoje inúmeros sistemas controlados eletronicamente empregam está técnica Controle Difuso baseia-se na lógica Difusa, uma lógica que utiliza o conceito de "verdades parciais". Estas verdades parciais podem ser expressadas, se compararmos com a lógica Booleana, com a atribuição de valores intermediários entre zero e um.

A implementação de um algoritmo desta técnica de controle, voltado para sistemas dedicados de pequeno porte, e com baixo custo de desenvolvimento e componentes, encontraria uma quantidade muito grande de aplicações em equipamentos industriais, podendo também servir como um sistema de baixo custo para o aprendizado da técnica. Com este objetivo, desenvolvemos neste trabalho a implementação de um algoritmo de controle Difuso escrito em linguagem FORTH. Esta linguagem encontra sua maior área de aplicação, em sistemas de controle microprocessados. Versões existem para praticamente todos os microcontroladores e microprocessadores comerciais mais empregados, além de dispositivos especialmente projetos para sua execução. Na área de desenvolvimentos de sistemas de controle digital, onde hardware e software são desenvolvidos quase simultaneamente, a produtividade desta linguagem tem se mostrado incomparável. Deste modo, a implementação do algoritmo nesta linguagem não limita a implementação á plataforma de hardware utilizada, podendo ser levada a outras plataformas que utilizem outros microcontroladores. A implementação deste sistema, e a abordagem dos assuntos relacionados, são tratados neste trabalho.

ABSTRACT

The need for automatic control systems is growing in both industry environments and modern life. The use of digital computers for control tasks is consolidated and keeps growing. Conventional control technics like mathematics modeling are still in use, but have to compete with modern technics like Fuzzy Control and the use of Artificial Intelligence, which make versatile modern control systems. The size of computers was reduced in order to allow them fit in small places like sensors and other mechanical devices. Thus, computers became small systems dedicated to specific tasks. The small cost of microcontrollers and the relative facility of implementation of a fuzzy control algorithm in a digital system impelled the use of fuzzy control in digital control systems, and nowadays a great number of electronic control systems use this technics. The fuzzy control is based on fuzzy logic, which make use of "partial truths". These partial truths can be expressed, comparing with Boolean logic, by numbers between zero and one. The implementation of a fuzzy control algorithm to be used in small dedicated systems with low cost would have a great number of applications in industrial equipments and could also be used as educational system for learning this technics. Thus, we developed the implementation of such fuzzy control algorithm, using FORTH language. The Forth language is particularly suitable for microprocessed control systems. There are versions for practically any of the most common commercial microcontrollers and microprocessors and there are also devices developed specifically for run Forth. In the area of development of digital control systems, where hardware and software are developed almost simultaneously, the productivity of this language has been insuperable. The implementation of the algorithm in this language does not entail the implementation to the hardware platform used, having the possibility of being taken to another platform with another microcontroller. The implementation of this system and an approach in related subjects are presented in this work.

I. Introdução

I.1 Introdução

A necessidade crescente de produtos e serviços com qualidade e em maior quantidade, incrementou a necessidade de sistema automáticos computadorizados e sistemas de controle por computador, em todos os níveis e tipos de produção industrial. Se automação e controle por computador tornarem-se opções baratas para a indústria, produtos em larga escala de produção poderão ser equipados com tais dispositivos, trazendo qualidade, fácil operação, sofisticação, conservação, e menores custos de manutenção. Com a demanda destes sistemas aumentando, novos dispositivos de *hardware* poderão ser oferecidos pelos fabricantes, bem como novas técnicas de controle serão desenvolvidas.

O crescimento da indústria, além de depender de custos, depende também da rapidez com que os sistemas de automação são desenvolvidos, e do suporte técnico que seus equipamentos dispõem. Por tudo isso, a tecnologia e o conhecimento do desenvolvimento de sistemas de controle e automação deve ser absorvida e verificada dentro e fora das universidades.

A sistemática de desenvolvimento de sistemas para controle e automação difere do modo de desenvolver *software* para aplicações comerciais (automação comercial) (TZOU, LIM, MENON, e PALMER, 1993) ou do desenvolvimento de *software* para grandes sistemas de computadores, em termos de ferramentas empregadas e necessárias, e de ambiente de desenvolvimento e utilização do produto final, o programa executável. Devido as suas particularidades, o desenvolvimento de *software* para automação nem sempre pode fazer uso da engenharia de software para validar e certificar programas, principalmente quando o *hardware* é desenvolvido simultaneamente ao software, fato comum em projetos de automação.

Algumas ferramentas oferecem facilidades para tornar o ciclo de desenvolvimento e projeto de sistemas, mais fácil e rápido. Geralmente estas soluções apresentam custos elevados, principalmente quando proporcionam grande rapidez e agilidade no desenvolvimento. A necessidade crescente de automatizar as mais variadas tarefas e processos, a baixos custos e de forma rápida, muitas vezes sem muitos recursos, faz com que profissionais da área busquem suas próprias ferramentas e formas de desenvolvimento. Mesmo atualmente, quando *hardware* e software tendem a se tornar padronizados, são inúmeras as aplicações onde a interação entre o projetista e o protótipo (um circuito eletrônico, um *software*, e um processo a ser controlado) exigem pequenos ajustes. Verificações de detalhes do funcionamento de uma pequena parte da eletrônica ou das ações produzidas no *hardware* por uma simples rotina, são feitas de forma interativa, passo-a-passo, como no início dos tempos da automação.

Percebe-se na atividade de desenvolver sistemas para automação, que ocorre grande eficiência quando há uma boa interação entre o projetista e o sistema desenvolvido. Dessa forma, as idéias podem ser fácil e rapidamente testadas e seu funcionamento e desempenho verificados, tanto quanto ao equipamento quanto da solução proposta. Isto é comprovado pelas características oferecidas pelas ferramentas atuais para desenvolvimento de sistemas de automação, que possibilitam a utilização de simulações de *software* e alguns sinais de *hardware*, modelos matemáticos para comparação, e técnicas de desenvolvimento de *software*, tais como projeto estruturado ou orientado a objeto. Tudo isso para minimizar a possibilidade de erros e falhas quando for posto em funcionamento o sistema real. Nessa busca, algumas ferramentas se tornam muito sofisticadas e complexas, criando dificuldades adicionais para o projetista.

A tendência atual em termos de tecnologia de *hardware* é a padronização. As indústrias oferecem CPUs e circuitos com funções as mais diversas e especializadas possíveis, cabendo ao *software* oferecer ao projetista a possibilidade de adaptar o *hardware* padronizado as suas necessidades. Na verdade a flexibilidade do software é que possibilita a padronização do *hardware*. Sendo assim, a escolha de linguagens e ferramentas que mais se adaptem ao desenvolvimento de determinado projeto, deve ser cuidadosamente avaliada.

Linguagens como ADA, Smalltalk, e C têm sido empregadas no desenvolvimento de sistemas automáticos e de controle. Oferecem vantagens para o desenvolvimento de grandes

sistemas por facilitarem o trabalho em equipe, terem alto grau de estruturação e documentação. São bastante difundidas e amplamente utilizadas em grandes projetos. Linguagens que favorecem o desenvolvimento em equipe encontram nas universidades um bom meio de divulgação e disseminação, por permitir que equipes desenvolvam projetos conjuntamente, e também porque sua estruturação e documentação facilitam a evolução dos sistemas, em termos de programação. Devido a isto, alguns autores se referem a estas linguagens como "linguagens acadêmicas".

Quando pequenos sistemas são desenvolvidos, os procedimentos necessários entre a edição do programa e sua execução, as etapas do ciclo de desenvolvimento, podem tomar um tempo razoável. Muitas vezes o método de tentativa e erro deve ser adotado, e os passos entre editar e executar são repetidos muitas vezes, tornando as etapas intermediárias cansativas e até mesmo prejudiciais, uma vez que bloqueiam o fluxo de raciocínio do projetista. Esta crítica se justifica se for observada a grande quantidade de sistemas de desenvolvimento existentes que utilizam depuradores residentes, como programas monitores e linguagem BASIC, inclusive em *chips*.

Para KNAGSS (1993), no porte de pequenos sistemas, linguagens interpretadas encontram sua grande aplicação, merecendo destaque a linguagem FORTH, uma linguagem interpretada alinhavada, desenvolvida especialmente para aplicações de controle de alta velocidade (WILSON e MALINOWSKI, 1989), e que existe atualmente para todas as CPUs comercialmente mais utilizadas em computadores e microcontroladores (WOEHR, 1991), incluindo *Transputers* e DSPs (*digital signal processors*). Devido a não ser considerada como linguagem acadêmica, não tem divulgação muito ampla, mesmo assim, é empregada em projetos nos maiores centros de desenvolvimento, com *hardware* de última geração.

Assim como a automação tem crescido e evoluído, também novas técnicas de controle tem surgido, como o Controle Difuso, uma das mais ativas e produtivas áreas de pesquisa da lógica difusa (LEE,1990). Vários centros de pesquisa e indústrias estudam esta nova tecnologia, e resultados satisfatórios já atestam seu emprego para soluções de controle de sistemas (WILLIAMS, TOGAI, e BRUBAKER, 1991, SIBIGTROTH, 1992, VIOT, 1993, TAN e CHANG, 1994). As pesquisas prosseguem buscando divulgar a técnica e encontrar regras e métodos para sua aplicação e evolução(WILLIAMS, TOGAI, e BRUBAKER, 1991).

I.2 Objetivos do Trabalho

Aplicando a técnica característica da linguagem FORTH para desenvolvimento de sistemas dedicados à técnica de controle difuso, é apresentada neste trabalho uma forma interativa de desenvolver soluções de controle difuso para sistemas de pequeno porte. Deste modo possibilita-se ao projetista, obter rapidamente um "sentimento", ou entendimento, das implicações do modelo e das ações de controle difuso, sobre o comportamento do sistema controlado, e também entender como alterações no modelo podem resultar em um comportamento diferente do sistema controlado. A forma interativa de programação do FORTH proporcionará a fácil alteração dos dados do modelo difuso, favorecendo a experimentação, permitindo que seja atingida rapidamente a melhor condição de funcionamento.

Como a linguagem FORTH existe para praticamente todos os microcontroladores e microprocessadores mais empregados para controle dedicado e automação, a implementação do algoritmo nesta linguagem, torna possível a adaptação do sistema aqui proposto, à muitos outros sistemas diferentes, baseados em outras plataformas de *hardware*.

I.3 Estrutura do Trabalho

O trabalho está dividido em 5 capítulos. No capítulo I são introduzidos os tema e os objetivos do trabalho.

No capítulo II são abordados os sistemas utilizados em automação e controle de processos. Definem-se alguns termos e tipos de sistemas, e o seu modo particular de desenvolvimento. A técnica de controle difuso e o desenvolvimento de controladores difusos são apresentadas.

Finalmente, é analisada uma estrutura generalizada de implementação da linguagem FORTH, e uma versão específica empregada neste trabalho, a versão "eFORTH".

No capítulo III são apresentados o modelo do sistema proposto. Cria-se uma estrutura para desenvolvimento de sistemas e aplicações de controle difuso, apresentando seu método de utilização, e o algoritmo de controle. No final é feita uma visão geral sobre a estruturação e emprego do sistema.

No capítulo IV é realizada uma simulação de controle empregando o sistema proposto. Apresenta-se uma situação onde é desejado empregar um sistema de controle difuso, modela-se o sistema, propondo-se uma solução de controle e o emprego do sistema proposto. Cria-se o programa de controle descrevendo cada etapa deste processo. O programa é simulado no *hardware* de controle, e os dados são apresentados de modo a confirmar a correta execução do algoritmo.

No capítulo V são apresentadas as conclusões obtidas dos estudos realizados durante a efetivação do trabalho, e sobre sua utilização e melhoramentos possíveis.

No anexo A é apresentada a listagem do eFORTH, em assembler, com as modificações realizadas que permitiram o seu o funcionamento na placa do sistema dedicado utilizado neste trabalho. No anexo B são apresentadas as palavras FORTH criadas para a implementação de programas de controle. No anexo C estão as definições FORTH do programa que será simulado para validar as implementações deste trabalho, e também definições de auxílio para entrada e saída de dados da simulação. No anexo D é apresentado o programa de controle para a aplicação proposta no trabalho, e que será usado na simulação. No anexo E consta a listagem de um programa de controle difuso escrito em FPL®.

II. Sistemas para Automação e Controle de Processos

II.1 Sistemas de Controle

II.1.1 Introdução

A palavra "sistema" envolve essencialmente dois conceitos, segundo GOPAL (1984): (1) interação dentro de um conjunto dado ou escolhido de entidades, e (2) uma fronteira (real ou imaginária), separando entidades internas ao sistema, de entidades externas. A interação tanto pode ser entre as entidades internas ao sistema, como pela influência das entidades externas, ou seja, uma interação entre entidades internas e externas. A fronteira, porém, é considerada completamente flexível. Por exemplo, pode-se limitar uma parte componente de um sistema original e considerá-la como um sistema em si, enquanto outro pode extender a fronteira do sistema original, incluindo novas entidades. O presente estudo limita-se ao controle de sistemas físicos, com entidades físicas quantificáveis, que obedecem as leis fundamentais da física.

As entidades componentes de um sistema podem estar em desequilibrio umas em relação as outras, ou com o mundo ao seu redor. Sobre a influência de um estímulo externo, o "estado" do sistema estará mudando com o tempo de uma forma característica dependente do caráter do estímulo e do vínculo da interação.

Um sistema de controle computadorizado consiste de uma coleção de dispositivos controlados por um programa armazenado, composto de instruções, que atua como elemento regulador, em um *loop* realimentado (ALLWORTH, e ZOBEL, 1987). O elemento regulador geralmente é um microcontrolador ou microprocessador.

Em princípio, é possível alterar o estado de um sistema de uma forma previsível por meio da escolha apropriada de entradas, pelo menos dentro de limites razoáveis. Pode-se exercer

influência sobre o estado de um sistema através de uma manipulação inteligente de suas entradas. De modo geral, isto constitui um "sistema controlado." A teoria de controle convencional trata da formulação matemática de leis para ações de controle.

A figura ii-a mostra a estrutura geral de um sistema de controle. A saída do sistema, ou planta, é avaliada através da variável *s*, que reflete o seu estado. Ela pode ser referenciada como saída ou como variável controlada ou de estado.

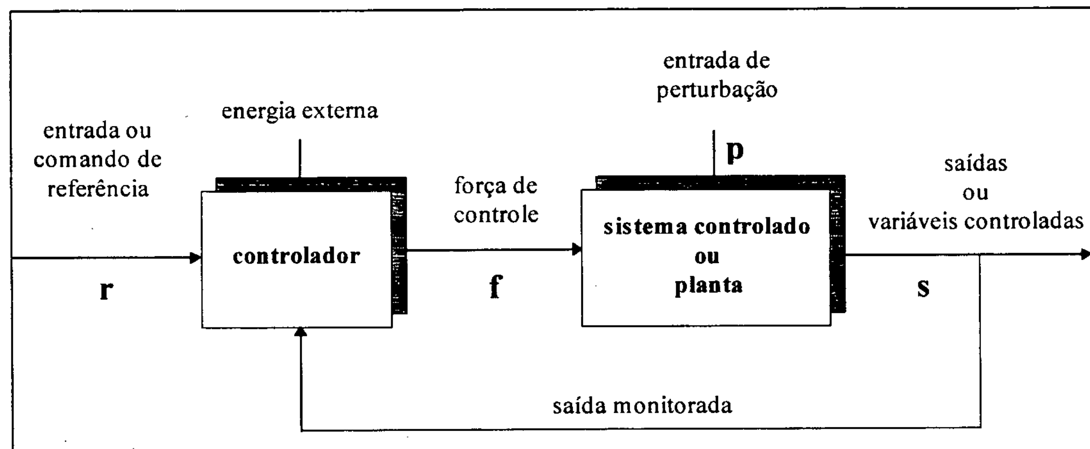


Figura II-A Estrutura Geral de um Sistema de Controle.

O controle do sistema é exercido através da força de controle *f* (sinal elétrico, por exemplo). Esta força é aplicada pelo controlador, que constitui a parte inteligente e de ação do sistema. O controlador determina a ação apropriada de controle baseado nas entradas do comando de referência *r*, e informações obtidas, via sensores de saída, referentes à saída monitorada *s*. A realimentação resulta em um fluxo de sinal ou dados, em ciclo fechado (*closed-loop*), empregando-se o termo "controle em malha fechada" (*closed-loop control*).

O diagrama de blocos genérico da figura ii-a também representa a entrada de perturbações *p*. Em algumas situações práticas, o controle é exercido de modo a evitar as influências prejudiciais de vários fatores classificados coletivamente de *perturbações*. Estes fatores podem ter origem externa, ou podem emanar do próprio sistema. A força perturbadora pode ser de natureza totalmente randômica ou previsível com algum grau de precisão. A complexidade de um controlador é função da complexidade da planta e da severidade dos requerimentos de controle.

Se as variáveis do sistema são funções contínuas do tempo, em todos os pontos, diz-se que o sistema é de tempo contínuo, e o controlador é um "controlador analógico", GOPAL (1984).

A complexidade de um controlador, necessária para implementar as leis de controle, é função da planta e da severidade dos requisitos de controle. O custo de um controlador analógico cresce acentuadamente com o aumento da complexidade da função de controle. Algumas funções podem ter complexidade tão grande a ponto de não ser possível implementar um controle apenas com elementos analógicos. Nestas situações, controladores digitais para aplicações especiais ou mesmo de aplicações gerais, onde um computador é o elemento central, o "coração" do sistema, é a opção ideal para exercer o controle (GOPAL ,1984). Computadores para aplicações gerais além de executarem a função de controle, podem realizar outras funções na planta ou no processo. Controladores digitais são versáteis, podendo-se modificar a função de controle com simples alterações de instruções em um programa. Os sinais que trafegam em controladores digitais são discretos no tempo, ou seja, são amostrados periodicamente. A estrutura geral de um sistema de controle digital é mostrada na figura ii-b.

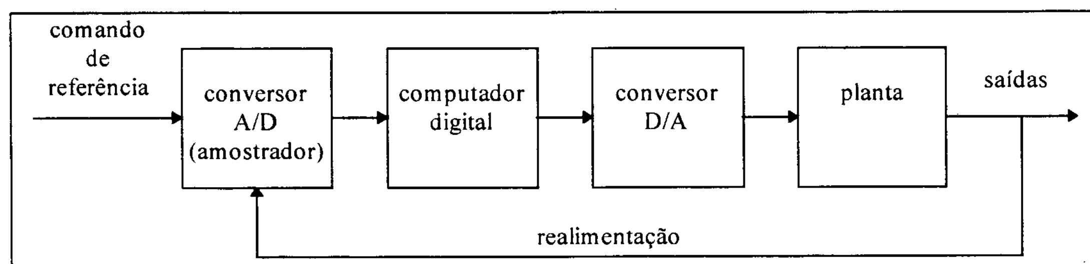


Figura II-B Estrutura Geral de um Sistema com Controlador Digital (GOPAL, 1984).

Sinais contínuos no tempo (analógicos) são convertidos pelo conversor A/D, para sinais discretos (digitais), na forma que o computador utiliza. O computador executa a função de controle implementada no programa armazenado, e a saída produzida é convertida para sinal contínuo no tempo, pelo conversor D/A, controlando a planta.

II.2 Sistemas Dedicados

Sistema Dedicado ou embutido como também é chamado, é o sistema que utiliza processador e *hardware* especial, dedicado à uma função específica de controle (TZOU, LIM, MENON, e PALMER, 1993). Diferente dos computadores pessoais, *desktops*, os sistemas dedicados geralmente não possuem monitores, unidades de disco e impressora, devido ao *hardware* ser minimizado para a tarefa de controle a que se dedica. A figura ii-c ilustra esta diferença, mostrando uma estrutura universal para um computador pessoal, e para um sistema dedicado.

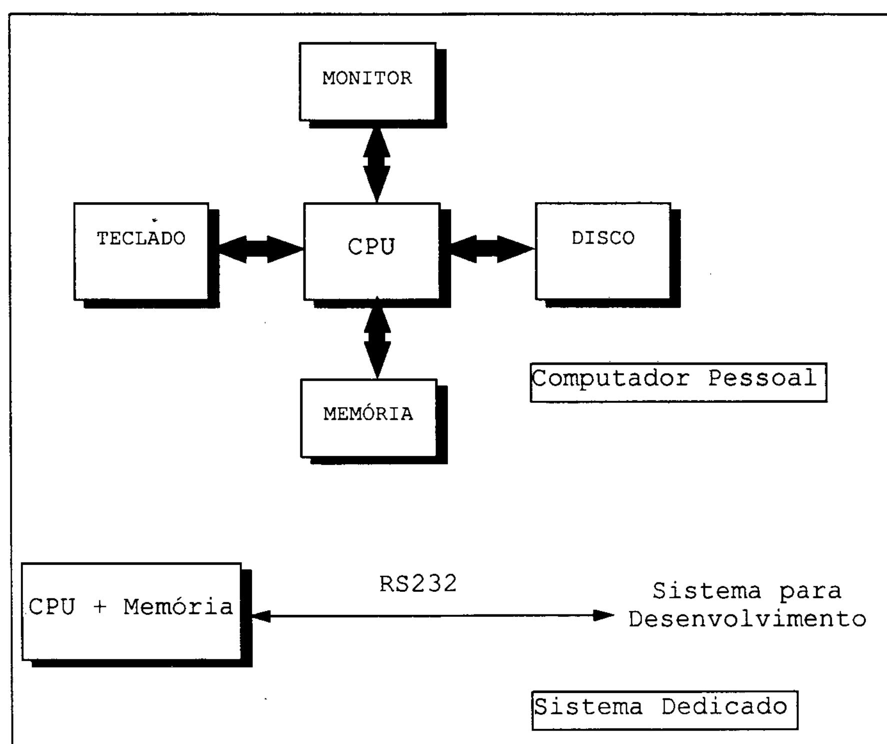


Figura II-C Computador Pessoal e Sistema Dedicado (TING, 1990).

A necessidade de minimizar se deve, em geral, a serem instalados em equipamentos onde o espaço é restrito, ou em gabinetes especiais (à prova de choques, de água, de altas temperaturas, pressão, sujeira, etc.) construídos especialmente para comportá-los. Sistemas dedicados equipam fornos aquecedores e de cozimento de alimentos, impressoras, controladores de disco magnético; sistemas de pilotagem em aviões e mísseis; sistemas de navegação em satélites e naves espaciais; injeção de combustíveis em automóveis, etc. O *software* executado

nestes sistemas é chamado de "*software* dedicado". A figura ii-d mostra a arquitetura básica de um sistema dedicado.

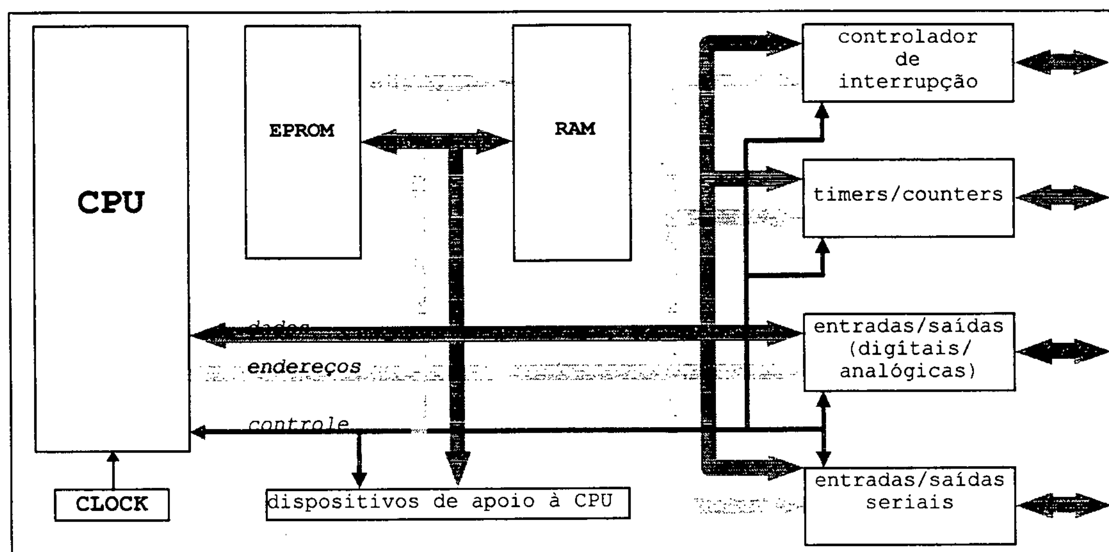


Figura II-D Arquitetura Básica de Um Sistema Dedicado.

A CPU (*central processor unit*) é o elemento que gerencia e controla o sistema. Aplicações que exijam processamento massivo de dados, requerendo maiores velocidades de processamento, utilizam microprocessador como CPU. Nestes casos, os blocos da figura ii-d são dispositivos discretos, cada bloco constituindo um *chip* do *hardware*. Nos sistemas dedicados normalmente são utilizados microcontroladores por simplificarem o *hardware*, e por serem dispositivos mais robustos para os fins a que estes sistemas se destinam. Microcontroladores comuns possuem em um mesmo encapsulamento, todos os blocos da figura ii-d. A tarefa de programação torna-se mais fácil, e o acesso aos dispositivos de entrada e saída mais rápido. O desenvolvimento de *software* dedicado tem uma técnica especial.

II.2.1 Desenvolvimento de *Software* Dedicado

A maioria dos *softwares* dedicados são escritos para determinado *hardware* de projeto específico (BROWN, 1994). Devido a isto, não há uma metodologia padronizada de desenvolvimento, ou pacotes de *softwares* comerciais prontos para todas as aplicações possíveis. Estes *software* são mais precisamente definidos por BROWN (1994), como *realtime-executives*, e não como sistemas operacionais, por possuirem características que vão além daquelas exigidas

de um sistema operacional. A finalidade de um sistema dedicado pode exigir funções do *software* dedicado muito diversas das funções comuns dos sistemas operacionais, como também dispensar algumas funções daquele.

Dois aspectos principais diferenciam o desenvolvimento de *software* dedicado do desenvolvimento comum de *software*: primeiro, o *software* dedicado geralmente não é desenvolvido no computador onde será executado quando estiver pronto, ou seja, no processador dedicado, sendo desenvolvido em outro computador (*cross development*) e posto para rodar no processador dedicado quando concluído; segundo, o *software* dedicado normalmente é desenvolvido paralelamente com o *hardware* dedicado. O fato de desenvolver *software* para um *hardware* que ainda não está pronto não permite que algumas práticas para teste de *software* sejam diretamente aplicáveis, obrigando o programador a utilizar técnicas e ferramentas menos confiáveis e eficientes. Este problema se agrava quando o sistema a ser desenvolvido é grande, necessitando de equipes de programadores.

Alguns aspectos limitam este modo de desenvolvimento no que diz respeito a produtividade e custos. Ferramentas ou sistemas de desenvolvimento para *cross development*, como assembladores, compiladores, *linkers*, simuladores, *shells*, nem sempre estão prontamente disponíveis para o microprocessador ou microcontrolador utilizado no projeto. Alguns sistemas podem ter altos custos comprometendo o projeto, ou serem lentos, comprometendo o tempo de implementação. Quando as ferramentas não possuem todas as necessidades, o programador dedica tempo à adaptá-las ao invés de dedicar-se ao projeto. O ciclo de testes torna-se um processo lento e pouco produtivo, devido a série de etapas necessárias até realizar um teste e obter conclusões. Etapas comuns são a edição do programa, compilação, descarga de programa do computador de desenvolvimento para a CPU onde o programa vai rodar, podendo ser via linha de comunicação ou gravação de EPROM, e depuração. Enquanto o *hardware* não estiver concluído ou utilizável, é necessário realizar os testes em simuladores, em geral lentos, principalmente quando o *clock* da CPU de simulação é muito próximo do *clock* da CPU simulada. As simulações também não refletem o comportamento exato do *hardware*. Um programador pode ter que simular grande quantidade de código, geralmente em assembler, e levar muito tempo até atingir o ponto de interesse. Mesmo tendo o *hardware* disponível, o ciclo

de testes pode continuar lento devido à descarga de código e à depuração em máquinas fisicamente diferentes e separadas.

Em termos do programa em si, suas funções, há muitas diferenças no procedimento de desenvolvimento. Programas desenvolvidos em computadores pessoais ou de maior porte, são escritos para serem executados na mesma máquina onde foram criados, e fazem uso dos recursos que o sistema, o conjunto *software* e *hardware*, oferece. Estes programas são executados "em cima" de programas (chamadas ao sistema ou rotinas básicas) que têm as funções mais básicas do sistema, como função. Os *softwares* dedicados geralmente realizam a tarefa de inicialização da CPU e dos dispositivos a ela conectados, sejam eles internos ou externos ao encapsulamento, para que então o programa de aplicação possa ser executado e faça uso dos recursos do sistema.

O termo *realtime-executive*, também empregado para designar o *software* dedicado, relaciona a execução de programas a temporizações, a eventos e ações do programa, relacionados com o tempo. Como categoria especial de sistemas, os sistema em tempo-real são abordados na próxima seção.

II.3 Sistemas em Tempo Real

O termo "tempo-real" foi originalmente introduzido para destinguir entre processamento em lote (*batch*) e computações onde haviam interações diretas entre o processador e o mundo real (BENNETT, e LINKENS, 1984). Atualmente é usado para indicar que a computação é "sincronizada" com eventos externos, e que o resultado de um processamento ou cálculo particular terá uma dependência de tempo , ou um tempo para ser executado.

Um sistema é considerado de tempo-real quando o correto processamento das informações depende não apenas da execução lógica correta, mas também do tempo que os resultados levam para serem produzidos. Tanto resultados incorretos produzidos pela execução lógica do processamento, como a produção dos resultados fora do tempo (ou margem de tempo), previamente especificado para que os resultados serem produzidos, implicam em falha do sistema. Exemplificando, um robô que tenha que pegar uma peça em uma esteira movendo-se

continuamente, se atrasar algum movimento, a peça não estará na posição onde ele deveria encontrá-la, causando uma atividade incorreta, falha.

Sistemas de tempo real são implementados tanto em grandes sistemas, com supercomputadores controlando processos de alta complexidade, como também em sistemas dedicados, para o controle de processos de menores proporções.

II.4 Controle Difuso

II.4.1 Introdução

O Controle Difuso baseia-se na lógica Difusa, um sistema lógico que se aproxima mais ao espírito humano de pensar e de se expressar em linguagem natural (LEE, 1990).

A lógica difusa é um extensão da lógica (booleana) convencional que foi estendida para trabalhar com o conceito de "verdade parcial" - valores verdadeiros entre "totalmente verdade" e "totalmente falso" (WILLIAMS, TOGAI, e BRUBAKER, 1991). Foi introduzida pelo Dr. Lotfi Zadeh nos anos 60, como um modo de modelar a incerteza na linguagem natural (Fuzzy FAQ).

A teoria de controle clássica parte de uma modelagem matemática para descrever o processo a ser controlado, exigindo um conhecimento detalhado do processo. Em alguns casos esta modelagem se torna muito complicada ou até mesmo impossível de ser feita (VIOT, 1993). Segundo WILLIAMS, TOGAI, e BRUBAKER, (1991), a lógica difusa pode agora ser aplicada a problemas lógicos e combinatórios complexos, para os quais se torna impossível a construção de modelos numéricos, devido a enorme quantidade de combinações possíveis. Para RABER (1994), o controle difuso começa a ter emprego em situações onde a utilização de algoritmos normais PD (proporcional e diferencial) e PID (proporcional, integral e diferencial) se tornam muito difiíceis de ajustar. Comenta ainda que o controle difuso pode ser utilizado com muito sucesso em qualquer processo considerado "difícil de ser controlado", onde grandes *overshoots* são observados, ou ocorre longo tempo de estabilização.

A técnica de modelagem e o controle Difuso manuseiam informações de forma qualitativa, permitindo expressar incertezas e falta de exatidão de modo mais conveniente ao conhecimento humano. Devido a não exigirem grande capacidade computacional, são facilmente implementáveis em microcomputadores e microcontroladores para controle de processos em tempo real. Segundo LEE (1990), a experiência mostra que os controladores lógicos difusos já apresentam resultados superiores aos obtidos com controladores que empregam algoritmos baseados nas técnica de controle convencional. A facilidade de implementação e a redução da complexidade de problemas está trazendo soluções para problemas antes intratáveis (WILLIAMS, TOGAI, e BRUBAKER, 1991; GOMIDE, GUDWIN, ANDRADE NETTO, 1993).

II.4.2 Controlador Lógico Difuso

Um Controlador Lógico Difuso (CLD) baseia-se na lógica Difusa para produzir uma estratégia de controle automático, a partir de uma estratégia de controle lingüística (LEE, 1990).

A figura ii-e mostra a estrutura básica de um controlador lógico difuso.

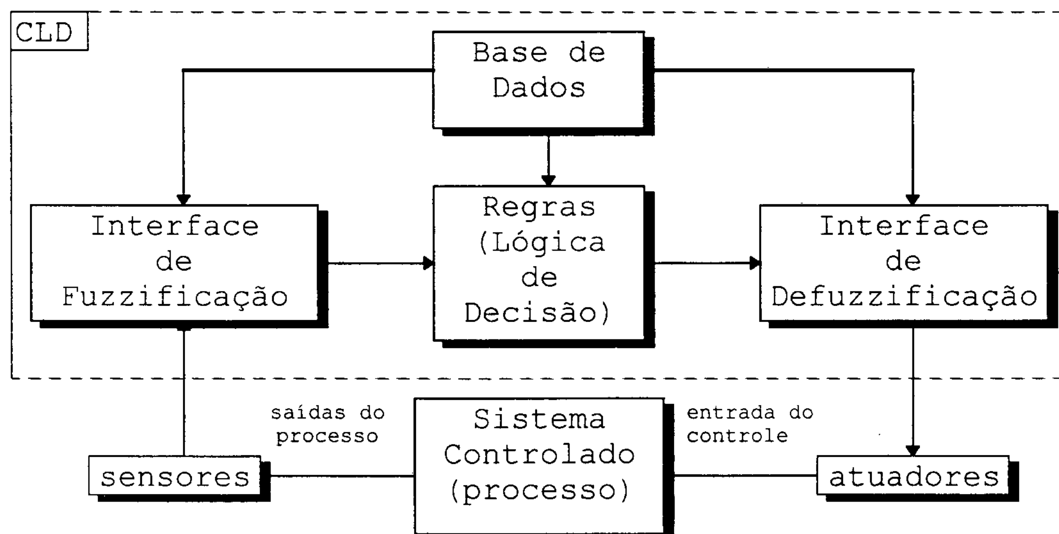


Figura II-E Estrutura Básica de um Controlador Lógico Difuso (CLD).

O emprego de um CLD é favorecido quando o processo a ser controlado é muito complexo para ser analisado por uma técnica quantitativa convencional, ou quando as fontes

disponíveis de informações são interpretadas qualitativamente, inexatas, ou incertas (LEE, 1990).

Sensores e atuadores dependem da aplicação, do processo que se pretende controlar. A "máquina de inferência Difusa" está implementada no CLD. Os sensores convertem grandezas como pressão, temperatura, umidade, velocidade, etc, em sinais elétricos que são entregues ao controlador, como variáveis de entrada a serem processadas. Os atuadores convertem as saídas do controlador na forma de energia necessária para atuar no processo controlado, seja na forma de energia elétrica de maior ou menor ordem de grandeza, ou eletro-mecânica.

Na seqüência enfoca-se as etapas do projeto de um CLD e da modelagem difusa.

II.4.2.1 Discretização/Normalização dos valores de entrada - Universo de Discurso

Os sinais lidos dos sensores são convertidos para números binários na entrada do CLD. Estes sinais são as variáveis de entrada do sistema, e devem ter seus valores de escala ajustados para o correto funcionamento do sistema.

Os valores limites que uma variável de entrada (e também para as variáveis de saída) pode assumir definem o seu "universo de discurso."

Supondo que uma variável de entrada tenha valores correspondentes a uma variação de temperatura, lida de sensores, e considerando-se um conhecimento previamente adquirido sobre o sistema, de onde se sabe que os valores podem variar entre 0 e 100 graus. Define-se um universo de discurso que vai de 0 a 100 graus, para uma variável que, estrategicamente, será chamada de 'temperatura'. Graficamente pode-se representar deste modo:

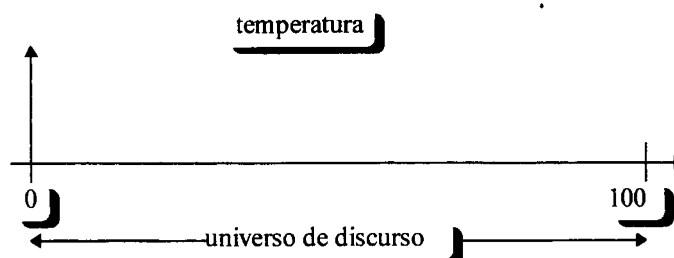


Figura II-F Universo de Discurso

Valores lidos entre 0 e 100 para esta variável, estarão dentro do seu universo de discurso. Deste modo, um valor que é contínuo no tempo é discretizado ou "quantificado em níveis", compondo o universo de discurso, de modo que possa ser manipulado e armazenado em memória pelo sistema. O número de níveis de quantização deve ser grande o bastante para expressar a variação do valor real, e pequeno bastante para não necessitar de muita memória. A escolha dos níveis de quantização tem influência essencial na precisão do controle que se deseja obter. Como exemplo, divide-se o universo de discurso de 'temperatura' em 100 partes, obtendo-se 100 níveis de variação para a variável 'temperatura'. Se forem armazenados os valores de cada nível em memória, serão necessárias 100 posições.

A quantização neste caso é linear. Pode ocorrer que uma quantização seja não linear, ou ambas, em toda ou em apenas determinada faixa de valores. É necessário um conhecimento *a priori* da variação do sinal de entrada, para estabelecer corretamente a quantização.

II.4.2.2 Conjuntos Difusos

Considerando o universo de discurso definido para a variável temperatura, será analisada uma variação hipotética de seus valores, dentro dos limites estabelecidos, para uma determinada aplicação:

-de 0 até 15 ou 25 graus no máximo, é considera como temperatura FRIA; -começando em 15 e até 75 ou 85 graus, é considera como temperatura MORNA, sendo em 50 graus o ideal; -começando em 75 ou 85 graus e em diante, é considera como temperatura QUENTE.

Quadro II-A

Obte-se então três termos lingüísticos para expressar temperatura, ou conforme a lógica difusa, três conjuntos difusos: FRIA, MORNA, e QUENTE.

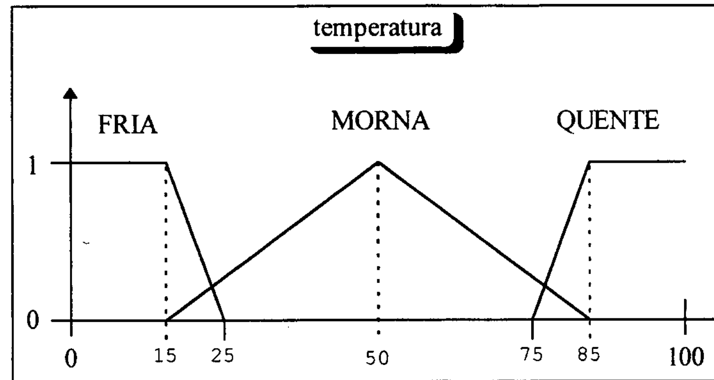


Figura II-G Variável Linguística Temperatura

Para valores entre 0 e 15 graus, a temperatura é considerada FRIA. De 15 a 25 graus existe uma incerteza a respeito de como classificá-la, pois ela começa a passar para um estado entre FRIA e MORNA que se confundem. Entre 25 e 75 graus a temperatura é considerada MORNA, atingindo aos 50 graus, a temperatura ideal. Entre 75 e 85 graus surge nova área de incerteza, entre MORNA e QUENTE, e de 85 graus até o limite superior, é QUENTE.

A partição do universo de discurso das variáveis de entrada e saída em conjuntos difusos, como afirma LEE (1990), segue um procedimento heurístico de tentativa e erro para encontrar o particionamento ótimo. A quantidade de conjuntos difusos determinará a granularidade do controle, ou seja, a relação entre variação na entrada e a resposta ou comportamento, do sistema de controle. A figura ii-g mostra todas as partes componentes de uma variável linguísitca, ou seja, universo de discurso, conjuntos difusos com seus nomes e funções de pertinência, e o nome da variável linguística, neste exemplo chamada de temperatura.

II.4.2.3 Fuzzyficação

Define-se "fuzzyficação" como o mapeamento dos valores entrados, lidos do mundo real, nos "conjuntos difusos". Este mapeamento resulta em valores de "grau de pertinência", que são os valores manipuláveis pela lógica difusa, dentro do sistema. Os valores de grau de pertinência

são extraídos de uma escala normalizada no eixo y. Para a figura ii-g, valores de temperatura lidos entre 0 e 15 graus, tem grau de pertinência 1 em relação ao conjunto difuso FRIA, e 0 para os demais conjuntos. Para valores lidos entre 15 e 25 graus, existem dois valores de grau de pertinência, que são descritos pela equação das retas utilizadas para descrever a variação da temperatura neste trecho. E assim por diante.

II.4.2.4 Funções de Pertinência

As funções de pertinência atribuem graus de pertinência, aos valores do universo de discurso. Há dois métodos de representar as funções de pertinência: numérico, e funcional. Numérico quando a função de pertinência é representada pôr um vetor numérico cuja dimensão depende do grau de discretização. Funcional quando é representada por uma função matemática.

Quanto a implementação destas formas de representação em sistemas dedicados, alguns fatores devem ser considerados. A representação em forma de vetor, ou tabela, tem a vantagem de trazer velocidade ao processamento. A pesquisa de um valor na tabela pode seguir uma procura binária ou mesmo uma rotina em assembler com instruções específicas, que em determinados processadores são extremamente rápidas. Porém, traz a desvantagem de necessitarem de mais espaço de memória, e de que se fixe o número de dígitos significativos para os valores representados. Desejando-se implementar o controle difuso em microcontroladores, onde a quantidade de memória é limitada em 4, ou 8 Kb de memória para código, a economia de memória deve ser considerada.

Na representação em forma de função, implementa-se a função no programa em forma de rotina. Esta forma reduz a quantidade de memória necessária para representar a função. Por outro lado exige mais capacidade computacional matemática do processador. Não dispondo de aritmética de ponto flutuante no sistema, será necessário que o projetista equacione a implementação do cálculo em aritmética de ponto fixo. A implementação dessa forma de representação em microcontroladores (onde a capacidade computacional é simples, reduzindo-se às operações matemáticas básicas) torna este processo muito lento, podendo prejudicar a tarefa de controle se for necessário que o cálculo seja executado repetidamente. Co-processadores aritméticos podem ser utilizados, mas encarecem significativamente o projeto. Essa opção é uma

hipótese a ser considerada se não comprometer os custos do projeto ou se houver melhoras na relação custos/benefícios.

Alguns exemplos de funções de pertinência mais utilizados em sistemas de controle difuso de pequena escala são mostrados na figura ii-h.

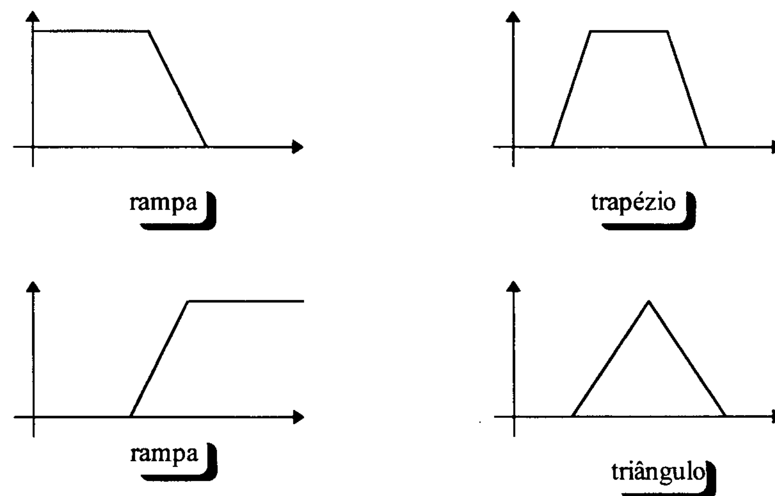


Figura II-H Funções de Pertinência mais comuns.

Verifica-se na literatura que estes tipos de funções de pertinência aplicados a maioria das soluções por controle difuso de sistemas de baixa complexidade com duas ou três variáveis de entrada e uma ou duas de saída, produzem bons resultados.

II.4.2.5 Base de Dados

A Base de Dados do sistema é composta pelos dados relativos ao universo de discurso e pelas funções de pertinência das variáveis de entrada e de saída. Tendo um algoritmo de controle difuso implementado no sistema, o ajuste, afinamento do controle, é feito apenas alterando esses valores, sem que sejam necessárias alterações no algoritmo.

II.4.2.6 Regras e Base de Regras

As regras têm a forma de declarações condicionais do tipo "SE - ENTÃO". O conjunto de regras de um sistema forma a sua "base de regras". A elaboração das regras tem implicação direta na performance de um CLD. Muitos estudos têm sido feitos mas não há métodos formais instituídos para extrair-se as regras de controle. A experiência e o conhecimento do processo são os principais fatores para uma escolha apropriada.

Tipicamente, as regras se relacionam à estados do processo. Exemplificando, uma regra que relaciona variação de temperatura com a abertura de uma válvula, pode ter a seguinte forma genérica:

`SE temperatura É fria ENTÃO válvula É aberta`

'Temperatura' é a variável linguística de entrada, e 'fria' é um de seus conjuntos difusos. O segmento '*SE temperatura É fria*' é chamado de "antecedente", referindo-se às condições (estados) atingidas, que antecedem uma decisão. 'Válvula' é a variável linguística de saída, e 'aberta' um de seus conjuntos difusos. O segmento '*ENTÃO válvula É aberta*' é o conseqüente, referindo-se ao resultado (conseqüência) da regra. No conseqüente estão as variáveis de controle do processo.

Esta é a forma mais simples para uma regra, pois existe apenas uma variável difusa de entrada no antecedente. Formas mais complexas podem referenciar duas variáveis de entrada, como por exemplo:

SE temperatura **É** fria **E** pressão **É** alta **ENTÃO** válvula **É** aberta

Nesta regra há duas variáveis linguísticas, "temperatura e pressão", e seus conjuntos difusos "fria e alta" respectivamente, e o operador 'E' entre os dois componentes do antecedente.

II.4.2.7 Operadores

Os operadores determinam qual dos valores de grau de pertinência, obtidos da fuzzyficação das variáveis linguísticas de entrada, será aplicado no conseqüente da regra. Os operadores mais comuns são o E (AND), e OU (OR). O operador "E" extrai o menor valor de grau de pertinência entre os componentes do antecedente. O operador "OU" extrai o maior valor de grau de pertinência.

II.4.2.8 Avaliação de Regras

A avaliação das regras é o procedimento de processar uma-a-uma as regras do sistema. Avalia-se o antecedente, obtendo-se um valor de grau de pertinência resultante conforme o operador empregado, que será atribuído à variável no conseqüente.

Normalmente existem várias regras em um sistema, e mais de uma pode referenciar a mesma variável de saída e o mesmo conjunto difuso no antecedente e no conseqüente. Um critério frequentemente utilizado, é o de atribuir à variável de saída o maior valor entre o valor resultante da avaliação da regra, e o valor atual da variável de saída, atribuído em regras anteriores. Quando utilizado este critério, a cada novo início de um procedimento de avaliação de regras, os valores das variáveis de saída devem ser inicializados para zero, de modo a evitar que resultados anteriores interfiram na avaliação em procedimento.

II.4.2.9 Agregação

A agregação consiste na obtenção de um conjunto de valores final para o sistema, a partir dos conjuntos difusos resultantes da análise individual de cada regra. A figura ii-i ilustra esta etapa de maneira mais clara.

II.4.2.10 Defuzzyficação

A defuzzyficação é a etapa final para a obtenção de um valor com significado "real" para o sistema controlado. No conjunto difuso resultante da agregação, são aplicados alguns métodos para obter-se o valor a ser atribuído ao sistema controlado. O método aplicado, como por exemplo o centro de massa ou o produto máximo, depende de algumas características do

sistema, principalmente no que se refere a sua capacidade computacional matemática. O valor exato, não difuso, obtido como resultado final, é chamado de valor *crisp*.

A figura ii-i ilustra graficamente estas etapas.

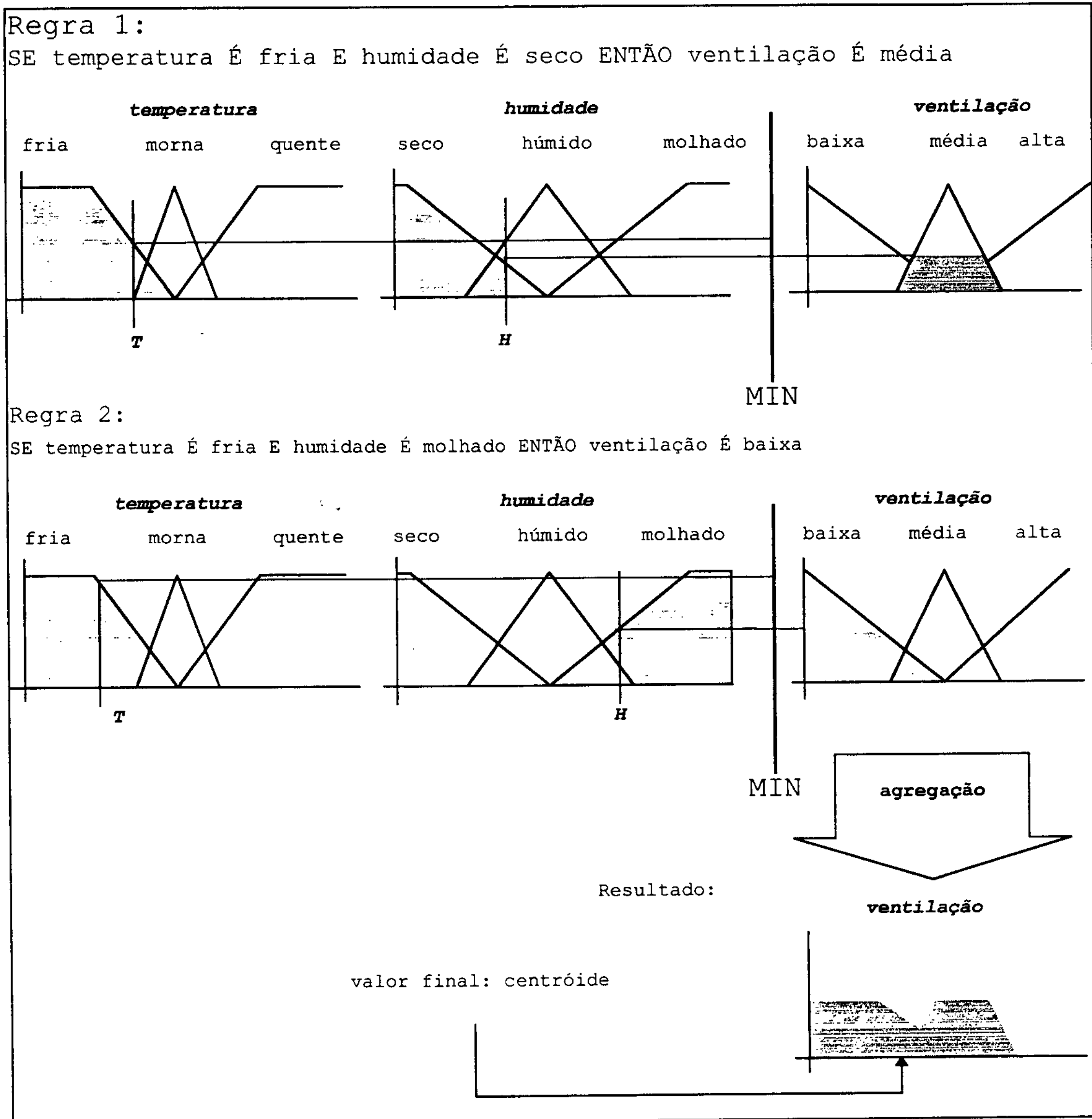


Figura II-I Inferência Difusa.

II.4.2.11 Singletons

O *singleton* é uma forma simplificada de representar conjuntos difusos, através de uma reta vertical passando pelo seu centro de massa. Em geral a representação por *singletons* é utilizada apenas para representação das variáveis linguísticas de saída, como mostrado na figura ii-j.

Regra 1:
SE temperatura É fria E humidade É seco ENTÃO ventilação É média

temperatura — fria, morna, quente
humidade — seco, húmido, molhado
ventilação — baixa, média, alta
T H
MIN

Regra 2:
SE temperatura É fria E humidade É molhado ENTÃO ventilação É baixa

temperatura — fria, morna, quente
humidade — seco, húmido, molhado
ventilação — baixa, média, alta
T H
MIN

agregação

Resultado:
ventilação
dois singletons truncados

Figura II-J Utilização de singletons.

Na figura ii-k é calculado o valor *crisp* a partir do resultado da agregação, quando são utilizados *singletons.*

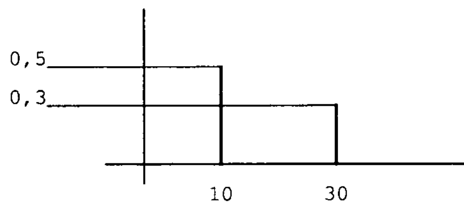


Figura II-K Cálculo do valor crisp a partir de singletons.

Dois *singletons* são apresentados. Um com valor 10 no universo de discurso da variável de saída, e outro com valor 30. Os valores 0,5 e 0,3 são valores de grau de pertinência resultantes da avaliação das regras. O valor *crisp* é calculado da seguinte forma:

$$\frac{0,5 \times 30 + 0,3 \times 10}{0,5 + 0,3} = 22,5$$

O valor *crisp* obtido é 22,5.

Conclui-se então uma visão geral sobre os conceitos básicos de controle difuso. A seguir é abordada a linguagem FORTH, onde as principais áreas de aplicação são controle e automação.

II.5 Linguagem FORTH

A linguagem FORTH foi criada por Charles F. Moore no início da década de 70. Moore construiu um sistema completo baseado nos conceitos de código alinhavado, para o controle em tempo-real, do telescópio do "Kitt Peak Observatory", USA. Na época utilizavam-se os computadores de quarta geração, e devido a isto Moore denominou seu sistema de "FORTH", uma espécie de abreviatura de "*Fourth*" (*fourth generation computer*), devido a utilização de apenas cinco letras para nomes de arquivo, permitido pelo computador onde o sistema foi desenvolvido.

Sistemas que sucederam ao seu também foram denominados FORTH por basearem-se praticamente nos mesmos conceitos e arquitetura, com algumas diferenças devido a aplicações específicas. Mas desde sua criação, a estrutura básica, arquitetônica, se mantém a mesma, tendo sido apenas adaptada aos novos microprocessadores e microcontroladores, e incorporados alguns novos conceitos de estruturas de dados. Após várias iniciativas de grupos de usuários no sentido de padronizar o FORTH, em 1994 o ANSI (*American National Standards Institute, USA*) deu a aprovação final (X3.215/1994) para a padronização, à proposta feita pelo *Forth Technical Committee*. Uma versão estendida do *Sun's Open Boot* já havia sido padronizada pelo

IEEE (1275) em 1994, e adotada pela IBM, Motorola, Apple, e outros fabricantes, para uso em PCI, VME, S-bus e outros barramentos e periféricos de próxima geração.

A arquitetura FORTH baseada no conceito de código alinhavado, cuja estrutura é praticamente a mesma nas mais diversas implementações, é abordada nesta seção.

II.5.1 Introdução

No início da década de 80 houve um crescente interesse por sistemas de *software* baseados no conceito de "código alinhavado" (KOGGE, 1982). O interesse se devia a necessidade de pequena quantidade de memória, 5 a 10 Kb, exigido por estes sistemas, e de características como:

* serem "conversacionais" como Lisp, Basic, e APL;
* ter as facilidades da compilação como as linguagens de alto nível, e programação estruturada;
* ter desempenho equivalente ou o mais próximo possível, ao da programação em código de máquina;
* ser escrita através de comandos básicos próprios, e altamente "portável";
* possibilitarem a combinação de sistemas, compiladores, ou código de aplicações;
* incluirem um sistema de memória virtual integrado, controlado pelo usuário, para os programas fontes e arquivos de dados;
* permitir que o usuário facilmente defina novos tipos de dados e estruturas; e
* poderem ser extendidas para incluir novos comandos escritos a partir dos comandos existentes, ou de linguagem de máquina.

FORTH é uma linguagem interpretada "alinhavada" (*threaded interpretive language - TIL*) que pode incluir todas estas necessidades, ou apenas as necessárias à aplicação que se destinar. É dita "alinhavada" devido ao modo como são encadeadas suas rotinas, chamadas de "palavras", constituindo o que em FORTH é chamado "vocabulário".

O grupo de instruções FORTH possui uma arquitetura simples baseada em uma máquina abstrata, que é facilmente implementada por pequenos segmentos de código em qualquer arquitetura de microcontrolador ou microprocessador atual. As concepções de linguagens de alto nível e de sintaxe que possui, incluem a construção hierárquica de programas, estruturas de dados, programação sem GOTO, e tipos de dados definidos pelo usuário (PETER, 1982). Comandos de conversação para monitoração permitem uma interação direta com os objetos definidos pelo programador nos termos que ele definiu, e não apenas nos modos básicos e simples oferecidos pelas estruturas mais básicas do sistema. Entenda-se por modo de "conversação" como o modo onde um comando é executado na console, e o sistema o executa e imediatamente retorna o resultado da ação do comando. Praticamente todo o conjunto de comandos da linguagem pode ser executado neste modo, e funcionam do mesmo modo como se estivessem em um programa em execução. Programas e rotinas definidas pelo programador tornam-se automaticamente parte do sistema, podendo ser usadas em combinação com outros comandos tanto no modo de conversação como na definição de programas. Isto permite ao programador desenvolver *interfaces* orientadas à aplicações, compatíveis com comandos básicos da linguagem, e tão eficientes quanto estes. "Muitos argumentos existem para definir estes resultados como uma máquina, como uma linguagem, como uma arquitetura, ou como um sistema operacional", cita KOGGE (1982), declarando que se trata de "um pacote de *software* auto-extendível". Para WILSON, MALINOWSKI, e PAWLOWICZ, (1989), FORTH constitui não apenas uma linguagem de programação, mas também seu próprio ambiente de programação, ou seja, o equivalente a um sistema operacional e suas ferramentas de desenvolvimento associadas. Comparativamente, 10k de código em linguagem C escrito em 8 horas de trabalho, leva aproximadamente 8 horas para ser depurado (em um ambiente de desenvolvimento tipo Visual C++), enquanto 10k de código em FORTH, de complexidade semelhante, é escrito em 6 horas e leva aproximadamente 1 hora para ser depurado. É senso comum entre programadores que também utilizam FORTH, que a linguagem oferece grande facilidade para isolar e matar *bugs*, e consequentemente seus produtos finais estão menos sujeitos a erros.

II.5.2 A Arquitetura FORTH

Iniciando uma abordagem lógica da estruturação do FORTH, será analisado o produto final das técnicas modernas de desenvolvimento de *software*. Este produto final é uma hierarquia de procedimentos ou rotinas, cada uma com *interfaces* bem definidas e curtas, de conteúdo facilmente compreensível e manipulação clara de dados (encapsulamento). Pode-se ilustrar este produto com o programa genérico no quadro ii-b, que segue.

```
Aplicação:
            call Inicializar
            call LerEntradas
            call Processo
            call SairDados
            .
            .
            .
LerEntradas:
            call AcessarDispositivo
            call LerDados
            .
            .
            .
Processo:
            call Rotina_1
            call Rotina_2
            call Rotina_3
            ...
            call Rotina_n
```

Quadro II-B Estrutura de um programa em linguagem de alto nível.

No nível mais alto do programa, ele consiste de endereços de rotinas precedidos por uma instrução de chamada, no quadro ii-b generalizado pela instrução *call*. O primeiro passo lógico, ou generalisação, sugerido para simplificar esta estrutura, é a remoção desta instrução, transformando o programa numa lista de endereços, e escrevendo uma pequena rotina em linguagem de máquina, que percorra esta lista sequencialmente, executando saltos diretos, ou indiretos, para os endereços. Isto reduz a quantidade de memória necessária sem diminuir significativamente o desempenho, possibilitando outras generalizações.

Supondo-se que as rotinas de **Processo**, 'Rotina_1, Rotina_2, Rotina_3, até Rotina_n', no quadro ii-b, estejam em código de máquina puro, assembler, e que **Processo** faça parte de

uma lista de rotinas com conteúdo similar ao seu, e definidas com a forma mostrada no quadro ii-c.

```
Processo:

  * Ponteiro ← endereço de 'Lista_de_Rotinas';

  * executa rotina Próximo_da_Lista;

     Lista_de_Rotinas:

          * endereço de Rotina_1;

          * endereço de Rotina_2;

          * endereço de Rotina_3;

          * ...

          * endereço de Rotina_n;

     Fim_da_Lista:

          * executa a rotina Próximo_da_Lista;
```

Quadro II-C Código diretamente alinhavado.

As duas linhas iniciais formam o "prólogo" da definição de **Processo**. A primeira linha é chamada de **Ponteiro**, onde é carregado o endereço inicial da lista de rotinas. A seguir é executada a rotina do quadro ii-d, o interpretador de endereços, ou interpretador interno. Nesta rotina, o registrador ***W*** é carregado com o conteúdo apontado por **Ponteiro**, ou seja, o endereço de Rotina_1 que ali está armazenado (é o início da lista de rotinas). **Ponteiro** é então incrementado, passando a apontar o endereço de Rotina_2, e a rotina cujo endereço está em ***W***, Rotina_1, é executada. No final de cada rotina da lista da rotinas, Rotina_1 até Rotina_n, haverá uma chamada para a execução da rotina ***Próximo_da_Lista.***

```
* Próximo_da_Lista:

     * W ← conteúdo endereçado por Ponteiro;

     * incrementa Ponteiro;

     * executa a rotina do endereço que está em W.
```

Quadro II-D Interpretador Interno para código diretamente alinhavado.

Quando ao final de Rotina_1, a rotina ***Próximo_da_Lista*** for executada, o registro ***W*** é carregado com o endereço de Rotina_2, apontado por **Ponteiro**, que passará a apontar para Rotina_3, e a Rotina_2 será executada. E assim prossegue a execução de cada um dos endereços da lista de rotinas, até o final, quando novamente ***Próximo_da_Lista*** é chamada. Esta chamada dispara a interpretação da lista que seguir **Processo**.

Conforme KOGGE (1982), a literatura define o "alinhavamento" (*threading*) formado por uma sequência de subrotinas referênciadas apenas pelos seus endereços de execução imediata, ou direta, como "código diretamente alinhavado" (*direct threaded code - DTC*), e devido a isto a rotina do quadro ii-d é chamada de "interpretador de endereços" ou "interpretador interno". Sua implementação é dependente da arquitetura da CPU por utilizar seus registradores internos. Como é código adicional que deve ser executado no final de cada rotina do programa, a eficiência de sua implementação e execução pela CPU, é fundamental para o desempenho da implementação como um todo. Segundo LOELIGER (1981), o código do interpretador interno deve ter o menor tempo possível de execução, porque é isto que irá determinar quanto rápida uma TIL pode ser. Através da sequência de "interpretação" dos endereços de uma lista de endereços de rotinas, a máquina executa os procedimentos do programa.

Para efeito de simplificação, reescreve-se o quadro ii-d substituindo-se "***Ponteiro***" por ***I*** (de Interpretador), e "*conteúdo endereçado por **Ponteiro***", por "memória[***I***]", que significa "o conteúdo de memória endereçado por ***I*** ". Sendo ***W*** um registrador da máquina, '[***W***]' será o conteúdo desse registrador. O quadro ii-d pode então ser reescrito de maneira mais simples, no quadro ii-e.

```
* Próximo_da_Lista:

    * W ← memória[I]

    * I = I + 1;

    * executa [W].
```

Quadro II-E Interpretador interno escrito de modo simplificado.

A mesma técnica empregada pelo interpretador interno, é aplicada para estruturar níveis de hierarquia, através do uso de uma pilha para guardar valores de ***I***, e cada rotina de nível

superior na hierarquia é precedida de um procedimento que "empilha" o valor atual de ***I***, ou seja, a próxima rotina da lista atualmente em interpretação, e o inicializa com o endereço da nova lista a ser interpretada. Para isto ter efeito, deverá haver no final de cada rotina da lista, um procedimento que recupere da pilha o valor de ***I***, que apontava para a rotina chamadora (de nível mais alto na hierarquia). Como a pilha é usada para retornar à rotina chamadora, ela é chamada de "pilha de retorno" (*return stack*). O quadro ii-f mostra o algoritmo.

Esta técnica inverte a posição das informações que "dizem" à máquina, que ela está "entrando" em uma nova rotina. Na abordagem clássica, os códigos associados com o endereço da rotina que será executada, é interpretado antes do início da execução na nova rotina. Em FORTH, o código do prólogo tem a mesma função, mas é invocado após a entrada na lista que será interpretada. Conforme KOGGE (1982), o que está sendo definido é um mecanismo para que uma rotina chamada determine que tipo de rotina ela é, e como salvar as informações para retorno. Na abordagem clássica, o procedimento de chamada de rotina é que fornece estas informações. Em FORTH, a máquina não sabe em que tipo de rotina está entrando, até estar lá (KOGGE, 1982). Esta característica de auto-definição é a chave para algumas vantagens e capacidades das TILs.

```
* Entrada:
      * salva I na pilha;
      * I ← endereço de Nova_Lista;
      * executa a rotina Próximo_da_Lista;
Nova_Lista:
      * endereço de Rotina_1;
      * endereço de Rotina_2;
      ...
      * endereço de Rotina_n;
Retorno:
      * recupera valor anterior de I;
      * executa a rotina Próximo_da_Lista;
```

Quadro II-F Codigo indiretamente alinhavado.

Do modo como foi definido, cada nova rotina constituída de uma lista de endereços de rotinas, deve ter uma "cópia" do código do prólogo no seu início. Para evitar estas duplicações, o código do prólogo constitui uma rotina a parte, independente, e no início de cada nova lista é colocado apenas o seu endereço inicial, e não a rotina completa. Toda rotina que usar o mesmo prólogo, terá o mesmo endereço no início. A existência desse endereço torna necessário um procedimento diferente do interpretador interno. Ao invés de saltar diretamente para o início da nova rotina, o interpretador interno deve saltar indiretamente, através desse endereço. O novo interpretador interno é mostrado no quadro ii-g, sendo *X* um registrador da máquina.

Esta técnica inverte a posição das informações que "dizem" à máquina, que ela está "entrando" em uma nova rotina. Na abordagem clássica, os códigos associados com o endereço da rotina que será executada, é interpretado antes do início da execução na nova rotina. Em FORTH, o código do prólogo tem a mesma função, mas é invocado após a entrada na lista que será interpretada. Conforme KOGGE (1982), o que está sendo definido é um mecanismo para que uma rotina chamada determine que tipo de rotina ela é, e como salvar as informações para retorno. Na abordagem clássica, o procedimento de chamada de rotina é que fornece estas informações. Em FORTH, a máquina não sabe em que tipo de rotina está entrando, até estar lá (KOGGE, 1982). Esta característica de auto-definição é a chave para algumas vantagens e capacidades das TILs.

```
* Próximo_da_Lista:
      * W ← memória[I];
      * I = I + 1;
      * X ← memória[W];
      * salta para [X].
```

Quadro II-G Interpretador interno para código indiretamente alinhavado.

A seguir são mostrados os tipos de interpretadores internos possíveis, para cada tipo de código alinhavado.

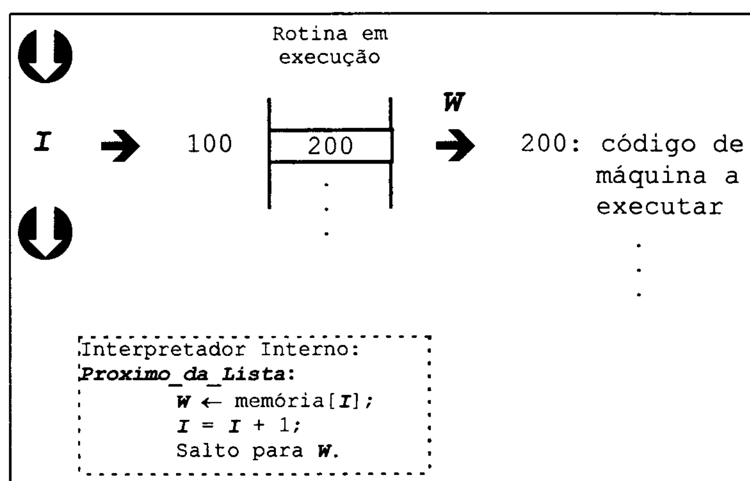


Figura II-L Interpretador interno para código diretamente alinhavado.

Na figura ii-l uma rotina é executada, e o ponteiro ***I*** tem o valor 100, cujo conteúdo é 200. Analisando o funcionamento do interpretador interno, é carregado em ***W*** o conteúdo apontado por ***I***, 200, ***I*** é incrementado, passando a apontar o próximo endereço da lista. A execução do programa segue então para a rotina que inicia no endereço 200. A execução do código é feita de modo direto, pelo endereço 200.

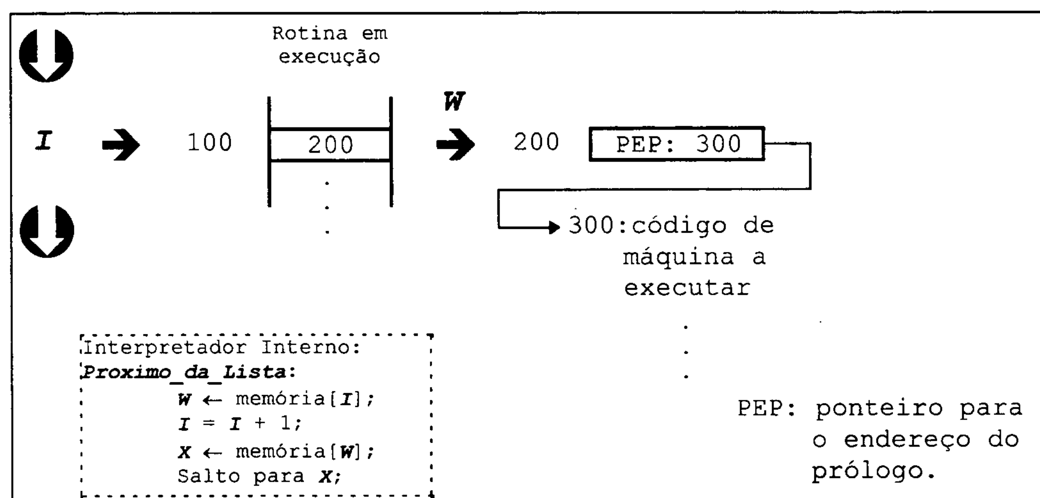


Figura II-M Interpretador interno para código indiretamente alinhavado.

Na figura ii-m, é executada uma rotina similar a anterior. O ponteiro ***I*** aponta o endereço 100, cujo conteúdo é 200. ***W*** recebe o conteúdo apontado por ***I***, 200, e ***I*** é incrementado, passando a apontar o endereço seguinte da lista. Na posição 200 está o prólogo desta rotina, que aponta para o endereço 300, início do código a ser executado, que é carregado no registro ***X***, para onde a execução do programa é transferida. Este "alinhavamento" do código é dito "indireto" porque a execução do prólogo não é feita diretamente através do endereço apontado

por *I*, mas indiretamente através de um ponteiro "apontado por *I*". Neste tipo de alinhavamento sempre haverá no PEP, um ponteiro para o endereço de execução do prólogo da rotina, e não o endereço para execução "direta" do código.

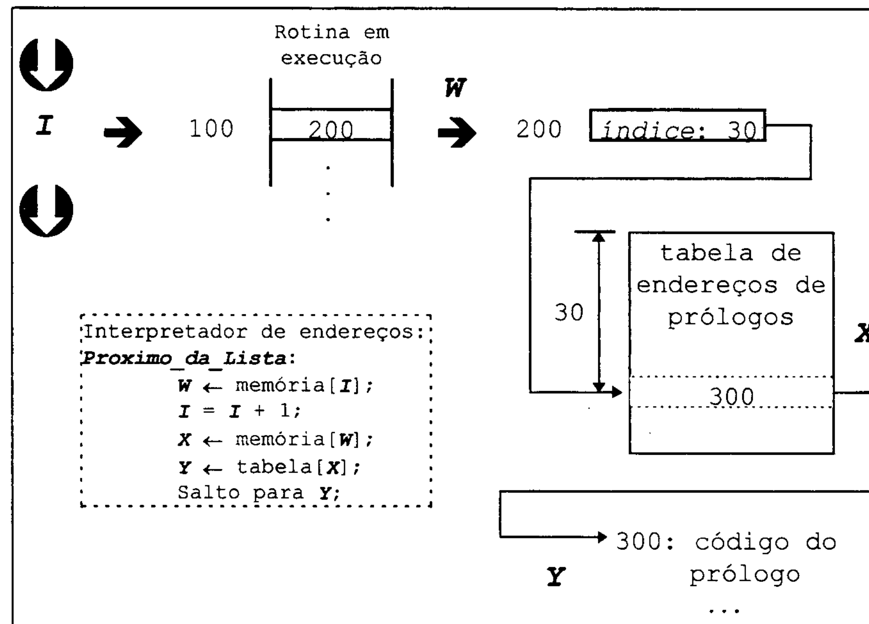


Figura II-N Interpretador interno para código indiretamente alinhavado por índice.

Na figura ii-n, há outra forma possível para o alinhavamento de código. O ponteiro *I* tem o endereço 100 da lista, *W* é carregado com 200, e *I* é incrementado, passando a apontar o próximo endereço da lista. Na posição 200 há um índice para uma tabela de prólogos. Este índice consiste de um deslocamento (*off-set*) dentro da tabela de endereços de prólogos, de posição previamente estabelecida no sistema. O registrador *X* é carregado com o valor do índice, 30, e finalmente o registrador *Y* é carregado com o conteúdo (um endereço) da posição inicial da tabela, somada ao valor do *índice*. A execução do programa prossegue então para o endereço 300, onde inicia o código do prólogo.

No quadro ii-h são analisados os tipos de interpretadores internos, para código diretamente alinhavado - CDA; código indiretamente alinhavado - CIA; e código indiretamente alinhavado por índice - CIAI, em termos de registradores utilizados.

tipo de interpretador interno	registradores de interpretação *I W X y*	endereço final do código executado
CDA	100 200 --- ---	200
CIA	100 200 300 ---	300
CIAI	100 200 30 300	300

Quadro II-H Comparação dos interpretadores internos.

O código indiretamente alinhavado é relativamente mais lento que o diretamente alinhavado, por utilizar mais registros e mais acessos à memória, até executar o prólogo. Em compensação, tem menos dependência da máquina onde está implementado, como também são mais flexíveis. Se o código for transportado para outra máquina, reescreve-se e recompilam-se apenas as rotinas de nível mais baixo, em assembler, e alteram-se os endereços dos PEP, mantendo-se intacto o código de mais alto nível. No código diretamente alinhavado, devido ao código de alto nível ter o endereço direto do prólogo e de outras rotinas que usa, alterações em quaisquer endereços de rotinas básicas desencadeiam a re-compilação das definições que a utilizam.

Para aplicações onde a velocidade é necessária, o interpretador interno é implementado no microcódigo do processador, como no microprocessador Harris™ RTX2000®, onde todo um sistema FORTH está implementado no microcódigo.

Na figura ii-o é mostrada a execução de uma lista de endereços, onde os nomes "Ler_Entradas" e "Processo", representam os endereço de duas rotinas.

A rotina "Ler_Entradas" é composta dos endereços das rotinas "Entrada (um prólogo), Abrir_Arquivo, Ler_Dados, Fechar_Arquivo, e Saída (outro prólogo)". Cada uma destas rotinas tem também a sua composição de endereços, como por exemplo a rotina "Ler_Dados", que inicia com o endereço do prólogo "Entrada", tem uma série de endereços, e acaba com o endereço do prólogo "Saída".

A rotina "Processo" é composta dos endereços das rotinas "Entrada (prólogo), Rotina_1, Rotina_2, Rotina_3, etc, e Saída (prólogo)". "Rotina_1" esta escrita em assembler.

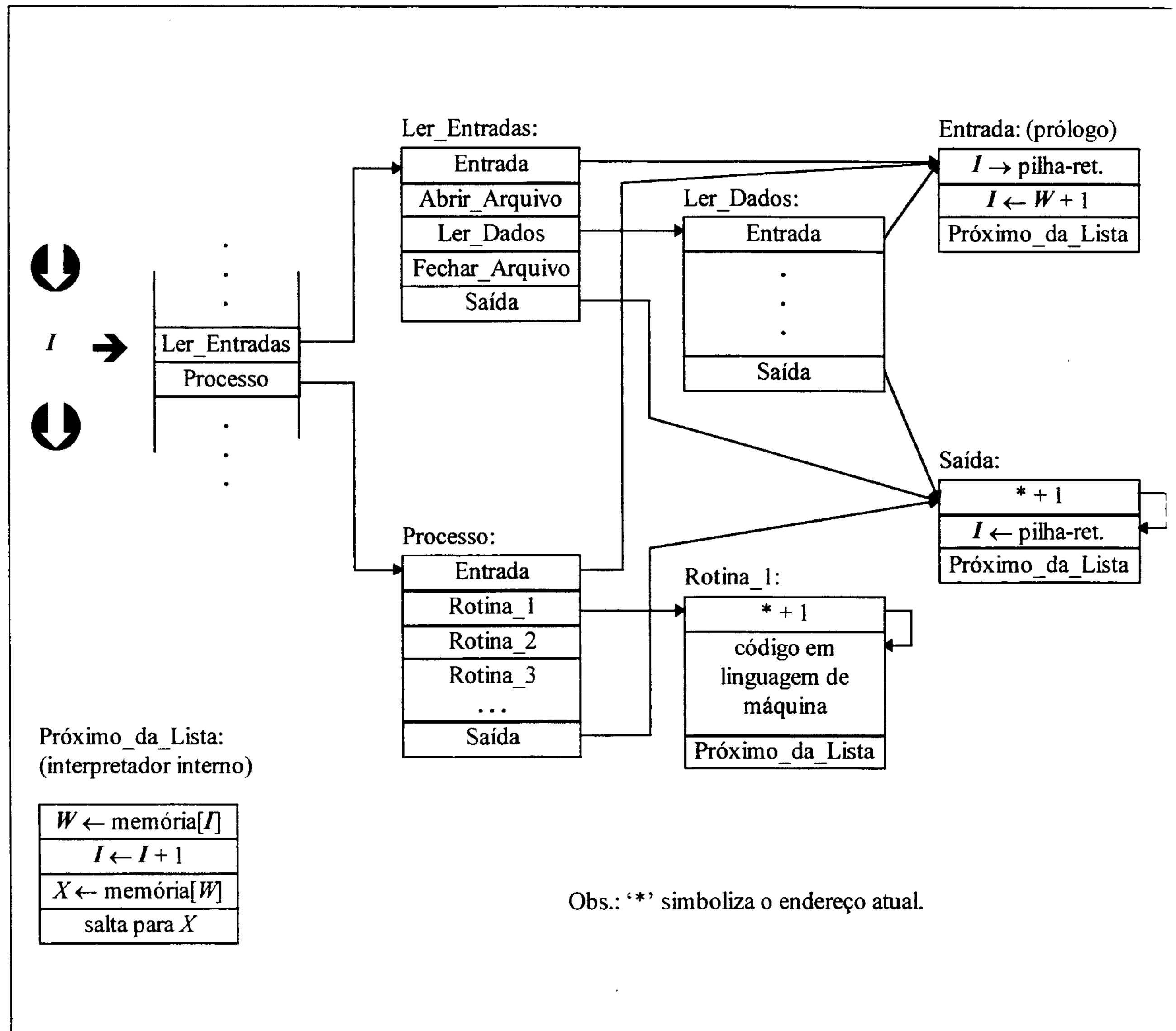


Figura II-O Código Indiretamente Alinhavado.

O prólogo *Entrada*, que faz com que o interpretador interno assuma a interpretação da lista, e salve o ponteiro ***I***, na pilha de retorno. O prólogo *Saída* finaliza a interpretação da lista, recuperando o ponteiro ***I***, voltando a interpretar a lista anterior. Um aspecto importante a destacar, é que a grande maioria das linguagens geram código totalmente dependente da máquina (CPU) onde rodam. Nesta linguagem, somente os níveis mais baixos da linguagem (geralmente um núcleo bem reduzido) tem dependência direta da máquina e da arquitetura. Todos os outros níveis usam endereços, favorecendo o transporte de programas para outras arquiteturas diferentes, praticamente sem alterações.

A partir destas concepções pode-se ter uma idéia inicial da arquitetura de um conjunto de instruções FORTH, onde cada instrução consiste apenas de um código de operação: o endereço da rotina. Para completar esta arquitetura é necessário um modo de passar parâmetros. O mecanismo utilizado para isto é uma pilha. Rotinas que necessitem dados de entrada os receberão do topo desta pilha, e dados produzidos pela rotina, são colocados no seu topo. Esta pilha de dados é chamada de "pilha de parâmetros" (*parameter stack*). Para exemplificar simplificadamente, são somados dois números, *n1* e *n2*, produzindo o valor *n3*. O quadro ii-i mostra o algoritmo necessário para a soma, na arquitetura proposta.

```
SOMA
        * [ponteiro para o interpretador interno];
        * colocar n1 na pilha;
        * colocar n2 na pilha;
        * somar n1 e n2;
        * colocar n3 na pilha;
        * executa a rotina Próximo_da_Lista;
```

Quadro II-I Exemplo de utilização da pilha de parâmetros.

A vantagem da utilização de duas pilhas, segundo KOGGE (1982), está na simplicidade da implementação, e na redução da complexidade conceitual que é oferecida ao programador. Dados são sempre passados através de pilha da parâmetros, e informações referentes ao sequenciamento de execução das rotinas, através da pilha de retorno.

A existência de pilha para dados favorece a utilização da notação polonesa reversa - NPR - para especificar sequências de operações, como a do exemplo de soma de *n1* e *n2*, no quadro ii-i. Na NPR os operandos vêm antes das operações, e a avaliação da expressão procede da esquerda para a direita, uma operação a cada vez. Não há parênteses nem precedência de operadores. Algumas calculadoras usam a NPR para a solução de cálculos matemáticos. Como exemplo, considere o polinômio:

$$At^2 + Bt + C = (At + B)t + C$$

Escrito na forma da NPR:

$A\,t\,{}^*B+t\,{}^*C+$

Lendo-se a expressão da esquerda para a direita, interpreta-se da seguinte maneira:

* ponha A na pilha;

* ponha t na pilha;

* multiplique os dois elementos no topo da pilha (A e t), deixando o resultado no topo;

* ponha B na pilha;

* some os dois elementos no topo da pilha (At e B), deixando o resultado no topo;

* ponha t na pilha;

* multiplique os dois elementos no topo da pilha ((At + B) e t), deixando o resultado no topo;

* ponha C na pilha;

* some os dois elementos no topo da pilha (((At + B)t) e C), deixando o resultado no topo;

Uma vez obtidos os valores de A, B, C, e t, o cálculo do valor final pode ser obtido com a sequência de nove passos, ou nove endereços quando usado código alinhavado, exatamente na ordem especificada acima. Não é necessário usar registros de uso geral, acumuladores, ou outros artifícios quaisquer. Esta simplificação é a chave para manter uma interface de usuário concisa, como será descrito a seguir.

Áreas de armazenamento como vetores, variáveis, constantes, etc, são estruturadas na arquitetura da linguagem, através de prólogos com rotinas especiais, que executam diferentes procedimentos para cada tipo de estrutura. Exemplificando, uma rotina que ponha na pilha de parâmetros o valor de uma constante, deve ter dois elementos: o PEP (ponteiro para o endereço do prólogo específico para constantes), e o valor da constante na posição de memória seguinte. O código do prólogo para constantes executa a sequência de tarefas do quadro ii-j.

```
CONSTANTE

      * Z ← memória[W + 1];

      * Z → pilha de parâmetros;

      * continuar a execução na rotina Próximo_da_Lista;
```

Quadro II-J prólogo para uma Constante.

Neste prólogo, o registro Z recebe o conteúdo da posição de memória seguinte ao PEP na estrutura da constante, que é o dado armazenado, e após coloca-o na pilha. Podem ser definidas tantas constantes quantas forem necessárias, e todas terão a mesma estrutura, o endereço desse prólogo, e o valor constante que armazenam.

Variáveis possuem estrutura similar, devendo ser possível alterar o dado armazenado. O prólogo para estas estruturas, mostrado no quadro ii-k, não põe o dado armazenado na pilha, mas o seu endereço, ficando a cargo do procedimento seguinte no programa, a manipulação do conteúdo, seja para leitura do dado armazenado, ou para armazenamento de um novo dado.

```
VARIÁVEL

      * Z ← W + 1;

      * Z → pilha de parâmetros;

      * continuar a execução na rotina Próximo_da_Lista;
```

Quadro II-K Prólogo para uma Variável.

Neste prólogo o registro Z recebe o endereço do dado armazenado, e coloca-o na pilha. Em FORTH os procedimentos de leitura e escrita são denominados ou simbolizados pôr '@' (*at*) e '!' (*store*) respectivamente. No quadro ii-l está o algoritmo de '@'.

```
AT

    * endereço da instrução seguinte;

    * Z ← dado no topo da pilha de parâmetros;

    * Y ← memória[Z];

    * Y → pilha de parâmetros;

    * executar a rotina Próximo_da_Lista;
```

Quadro II-L Definição de '@' - AT.

No início da definição está o PEP, apontando para o seu próprio código. O valor do topo da pilha (um endereço) é retirado e colocado no registro Z. O conteúdo da memória endereçada por Z é lido e carregado no registro Y, e então colocado na pilha de parâmetros. Estes dois últimos passos podem ser reduzidos para um único. Considerando que a pilha de parâmetros normalmente é mapeada em memória convencional, uma transferência direta de memória para memória é uma atividade que envolve muitos ciclos de máquina nas arquiteturas de CPUs mais comuns. Devido a isto é mais comum utilizar transferência via registros.

No quadro ii-m é mostrado o algoritmo de '!'- *STORE.* Sua rotina remove dois operandos da pilha de parâmetros: um valor , e um endereço para sua armazenagem.

```
STORE

    * endereço da instrução seguinte;

    * Z ← dado no topo da pilha de parâmetros;   (endereço)

    * Y ← dado no topo da pilha de parâmetros;   (dado)

    * memória[Z] ← Y;                            (salva)

    * executar a rotina Próximo_da_Lista;
```

Quadro II-M Definição de '!' - STORE.

Os dois dados são lidos da pilha, sendo o endereço carregado em Z, e o dado carregado em Y, e armazenado na posição de memória endereçada por Z. A execução procede na rotina do interpretador interno.

Um conceito importante que pode ser observado das definições de CONSTANTE e VARIÁVEL, é que o acesso a estruturas de dados pode ser definido em uma rotina de prólogo, endereçada pelo PEP, e trabalhar sobre uma área de dados que inicie em W+1 (o endereço seguinte ao PEP). Segundo KOGGE (1982), isso permite que qualquer tipo de estrutura de dado seja definida. Exemplificando, uma rotina para manipulação de *array* pode utilizar um dado no topo da pilha de parâmetros como um índice, a posição W+1 como dimensão do *array*, e a posição W+2 como início da área de dados do *array*. O algoritmo do prólogo endereçado pelo PEP da definição do *array*, é sugerido no quadro ii-n.

```
ARRAY
    * Z ← dado no topo da pilha de parâmetros;   (índice)
    * Se memória[W+1] ≤ Z, então indica erro;
    * Y ← W + 2 + Z;
    * Y → pilha de parâmetros;
    * continuar a execução na rotina Próximo_da_Lista;
```

Quadro II-N Prólogo para uma estrutura de dados "Array".

Pode-se estender essas definições para a criação de novas estruturas de dados conforme as necessidades do programador, que podem ser integradas ao resto da arquitetura da linguagem de modo claro, através da pilha de parâmetros, que é comum a toda a arquitetura.

Como ilustração dos conceitos empregados na linguagem, e para melhor entendimento do funcionamento da arquitetura da linguagem como um todo, é mostrada na

figura ii-p, a hierarquia de procedimentos e código, necessária para a avaliação do polinômio,

$$At^2 + Bt + C = (At + B)t + C.$$

Assume-se que o valor de *t* está no topo da pilha de parâmetros, que A é uma constante, B uma variável, C é o terceiro elemento de um *array*, e Z é uma variável onde o resultado será colocado.

Da esquerda para a direita na figura ii-p, são mostrados os dados no topo da pilha de parâmetros, a sequência de definições (programa) que o programador deve escrever para a

avaliação do polinômio, a estrutura de cada definição usada, e seus prólogos. A estrutura mostrada nessa figura, ilustra o modo como o programa é encadeado na memória do computador, e o algoritmo de cada rotina.

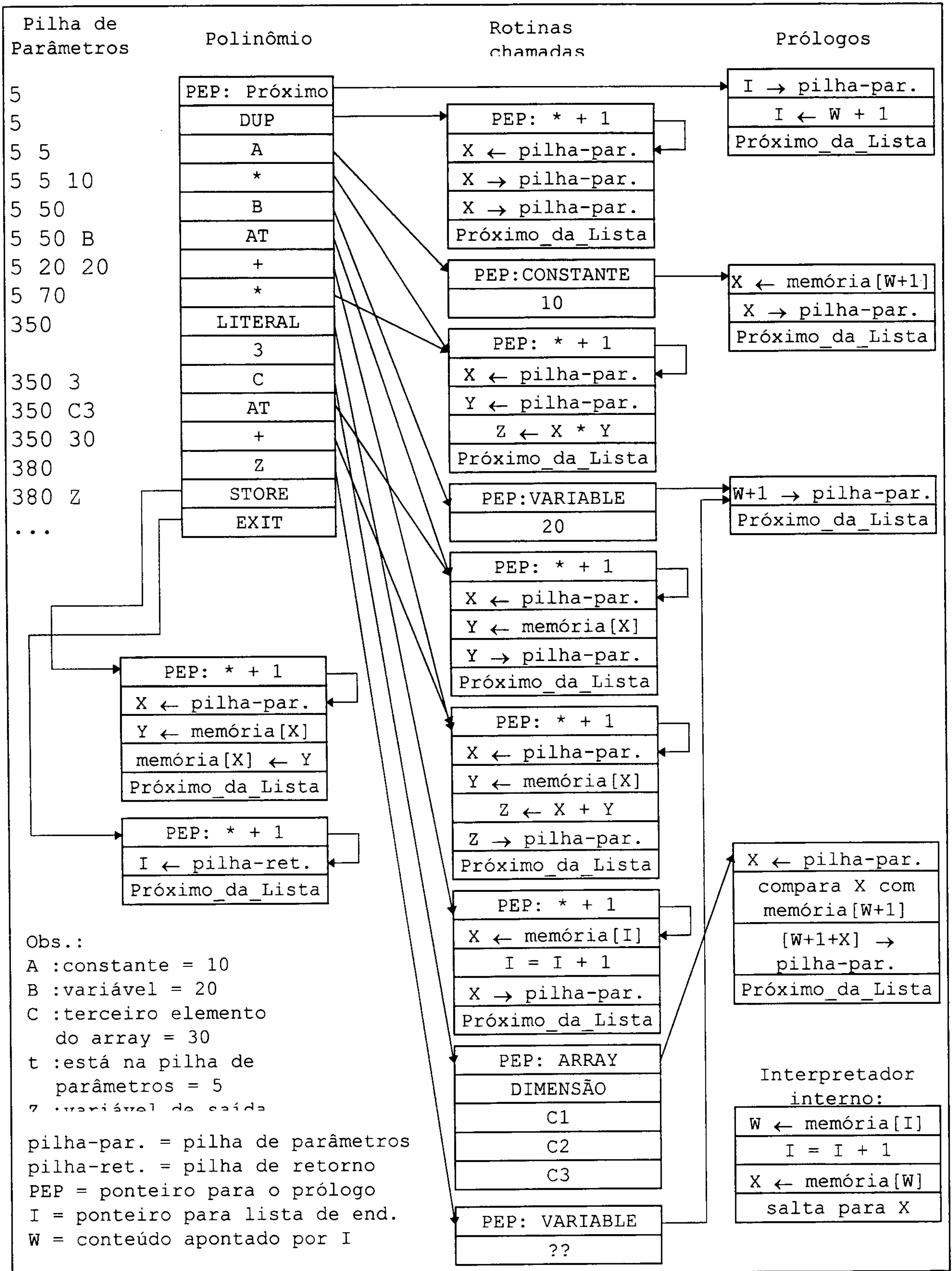


Figura II-P Estruturação de definições para a avaliação do Polinômio.

A arquitetura do sistema FORTH permite a definição de várias formas de tomada de decisão e processamento condicional. O desvio na ocorrência de zero é a forma mais simples e comum. Supondo-se uma sequência de programa como mostrado no quadro ii-o, onde BRANCH0 - *branch on zero* na terminologia FORTH - desviará a execução do programa para o endereço DESVIO, se houver zero no topo da pilha. Define-se este desvio condicional através de um prólogo, preferencialmente em assembler, para a rotina BRANCH0, como mostrado no quadro ii-p.

```
        .
        .
        .
I →  BRANCH0    (rotina que decide o rumo da execução)
     DESVIO     (endereço para desvio)
        .
        .
        .
```

Quadro II-O Exemplo de desvio condicional dentro de uma definição.

Quando o ponteiro ***I*** encontra o BRANCH0, o seu prólogo é executado.

```
BRANCH0
     * endereço da instrução seguinte;
     * Z ← dado no topo da pilha de parâmetros;
     * Se Z = 0
          então: I = I + memória[I];     (usa DESVIO)
          senão: I = I + 1;              (pula DESVIO)
     * executa a rotina Próximo_da_Lista;
```

Quadro II-P Desvio Condicional.

Na execução do prólogo, o valor no topo da pilha é checado. Se for zero, a execução desta lista de endereços é desviada para outra lista, cujo endereço é simbolizado por DESVIO. Não sendo zero, o ponteiro ***I*** é incrementado passando a apontar o endereço seguinte a DESVIO.

Outra forma, particularmente útil em *loops*, é a saída condicional de rotinas. O endereço no topo da pilha de retorno é carregado em ***I***, se o valor no topo da pilha de parâmetros for zero, abortando a lista em execução. Caso contrário, nada muda na pilha de retorno. Outra forma de mudar a sequência de interpretação de uma lista, é a chamada condicional de subrotinas. Supondo a sequência do quadro ii-q, onde ***I*** aponta 'CALL0' - *call if zero* em FORTH - e 1F58 é um endereço em hexadecimal.

```
      .
      .
      .
I →   CALL0
      1F58
      .
      .
      .
```

Quadro II-Q Chamada condicional de subrotina.

O prólogo em assembler para a instrução 'CALL0' é mostrado no quadro quadro ii-r.

```
CALL0
     * endereço da instrução seguinte;
     * Z ← dado no topo da pilha de parâmetros;
     * Se Z = 0
          então: I + 1 → pilha de retorno;
                 I ← memória[I];
          senão: I = I + 1;
     * executar a rotina Próximo_da_Lista;
```

Quadro II-R Prólogo para chamada condicional de subrotina.

O endereço 1F58 só será executado se o valor no topo da pilha de parâmetros for zero. O endereço ***I*** + 1 é salvo na pilha de retorno para que ao final da subrotina, a lista original volte a ser executada. Se o valor no topo da pilha não for zero, a execução da lista prossegue normalmente, sem considerar o endereço 1F58.

As construções vistas até aqui, com algumas variações, formam estruturas similares como IF-THEN-ELSE, e também formas de *loop* como DO-WHILE e UNTIL-DO. *Loops* iterativos (como por exemplo DO K=1,10) também são definidos, usando variáveis e outras formas de construções, para armazenar o contador de iterações do *loop*. FORTH usa a pilha de retorno para armazenar o contador de iterações e os limites. Antes de entrar no *loop*, os valores apropriados são colocados na pilha de retorno; após cada iteração, os valores colocados no topo da pilha de retorno são incrementados de um valor apropriado e comparados com os limites. Se o *loop* ainda deve ser executado, os valores modificados ainda permanecem na pilha de retorno. Se o *loop* estiver finalizado, os valores são removidos da pilha. Este tipo de estrutura é inerentemente reentrante, e pode ser "aninhados" em vários níveis.

Segue um sumário parcial da arquitetura FORTH até aqui analisada:

* *é um mecanismo de acesso indireto a códigos;*
* *utiliza duas pilhas;*
* *possui estruturas de dados definívies pelo usuário;*
* *suas operações são extensíveis;*
* *possui uma estrutura hierarquizada; e*
* *há possibilidade de implementação direta de estruturas de linguagens de alto nível.*

Uma aplicação construída nesta arquitetura será constituída de:

* *um grupo reduzido de pequenas rotinas de prólogos;*
* *um grupo básico de rotinas escritas especificamente para a arquitetura da máquina (CPU);*
* *um grupo de rotinas construídas a partir de ponteiros para prólogos e sequências de endereços apontando para rotinas deste grupo, ou para o grupo básico.*

Quanto a performance, há uma diferença de entre o último grupo de rotina, e o grupo básico. No último grupo de rotinas há um custo em termos de tempo de execução devido a interpretação, e a execução dos códigos dos prólogos, e também devido a utilização de registros da CPU para a passagem de parâmetros. Estas perdas têm relação direta com a arquitetura da

CPU onde o sistema está implementado. KOGGE (1982) mostra que testes comparativos de desempenho entre a execução de código alinhavado e código escrito diretamente em assembler resultam para o primeiro em um acréscimo de 100% no tempo de execução quando em processadores de 8 bits, e de apenas 20% quando em processadores de 16 bits. Considerando um FORTH implementado para microprocessadores Intel da família 8086, cuja arquitetura de instruções e registradores não sofreu alterações nos últimos anos, tendo melhorado o desempenho devido a maiores velocidades de execução de instruções que influenciam tanto ao assembler quanto ao código alinhavado, estes dados ainda podem ser considerados. Há uma compensação destas perdas pela diminuição no tamanho do programa, e pelo ganho em termos de independência da máquina. As únicas partes dependentes da máquina em uma aplicação, são os códigos dos prólogos e o grupo básico de rotinas, que geralmente são relativamente fáceis de serem transportados para máquinas diferentes, e em pequena quantidade. Pode ocorrer que o "corpo" da aplicação, constituído de endereços, seja transportado intacto para outra máquina.

Nas seções seguintes serão analisados aspectos relativos à compilação e interatividade.

II.5.3 O Dicionário FORTH

O que torna uma coleção de rotinas em código alinhavado, interativa com um programador, é um mecanismo que transforma nomes simbólicos em uma definição constituída de um PEP e um "corpo" (conteúdo). Um mecanismo simples implementado em FORTH, consiste na inclusão de informações antes do campo do ponteiro para o endereço do prólogo (PEP). Estas informações consistem de três ítens:

* um nome simbólico para a rotina na forma de uma *string* de caracteres;

* um ponteiro para outro nome de rotina;

* informações para fins de compilação.

Os ponteiro servem para encadear rotinas, formando uma lista encadeada, que em FORTH é chamada de "dicionário". Iniciando em um nome simbólico, a procura de uma rotina

(pelo seu nome) no dicionário, retorna um ponteiro para o seu PEP e corpo. Na figura ii-q é mostrada parte de uma forma possível para estruturar um dicionário FORTH.

II.5.4 O Interpretador de Texto

A combinação do dicionário com um interpretador de texto (também chamado de interpretador externo) formam a base para interação com o programador FORTH. O interpretador de texto é uma rotina simples que recebe caracteres de um terminal; quando um nome é digitado, a rotina procura no dicionário por um nome igual. Se encontrar, executa a rotina com este nome. Caso contrário, o interpretador de texto verifica se não se trata de um valor numérico. Se for, joga o valor para a pilha de parâmetros. Se não for, envia uma mensagem de erro por se tratar de uma situação impossível. Após isto o interpretador de texto volta a esperar que algo seja entrado (digitado).

O interpretador de texto é extremamente simples, e dois aspectos chave dão ao sistema uma capacidade verdadeiramente interativa. Primeiro, sua rotina é projetada para que quando uma rotina definida pelo usuário for executada, nada haverá na pilha de parâmetros ou na pilha de retorno, que tenha sido colocado por ela, deixando-as livres e para total uso e controle do usuário. Segundo, rotinas são executadas na ordem em que elas são entradas, da esquerda para e direita. Desse modo, quando a notação Polonesa reversa é usada para expressar um cálculo e um processamento, há uma equivalência exata na sequência com que os nomes das rotinas devem ser entrados no interpretador de texto. Exemplificando, a sequência "10 20 30 * + PRINT" põe 10, 20, e 30 na pilha de parâmetros, multiplica 20 e 30, soma 10 ao produto, e a rotina PRINT mostrar o resultado. Ao contrário da maioria dos sistemas interativos, este não requer uma significante análise sintática sobre todas as linhas do texto, antes da execução iniciar (KOGGE, 1982), facilitando inclusive a compilação.

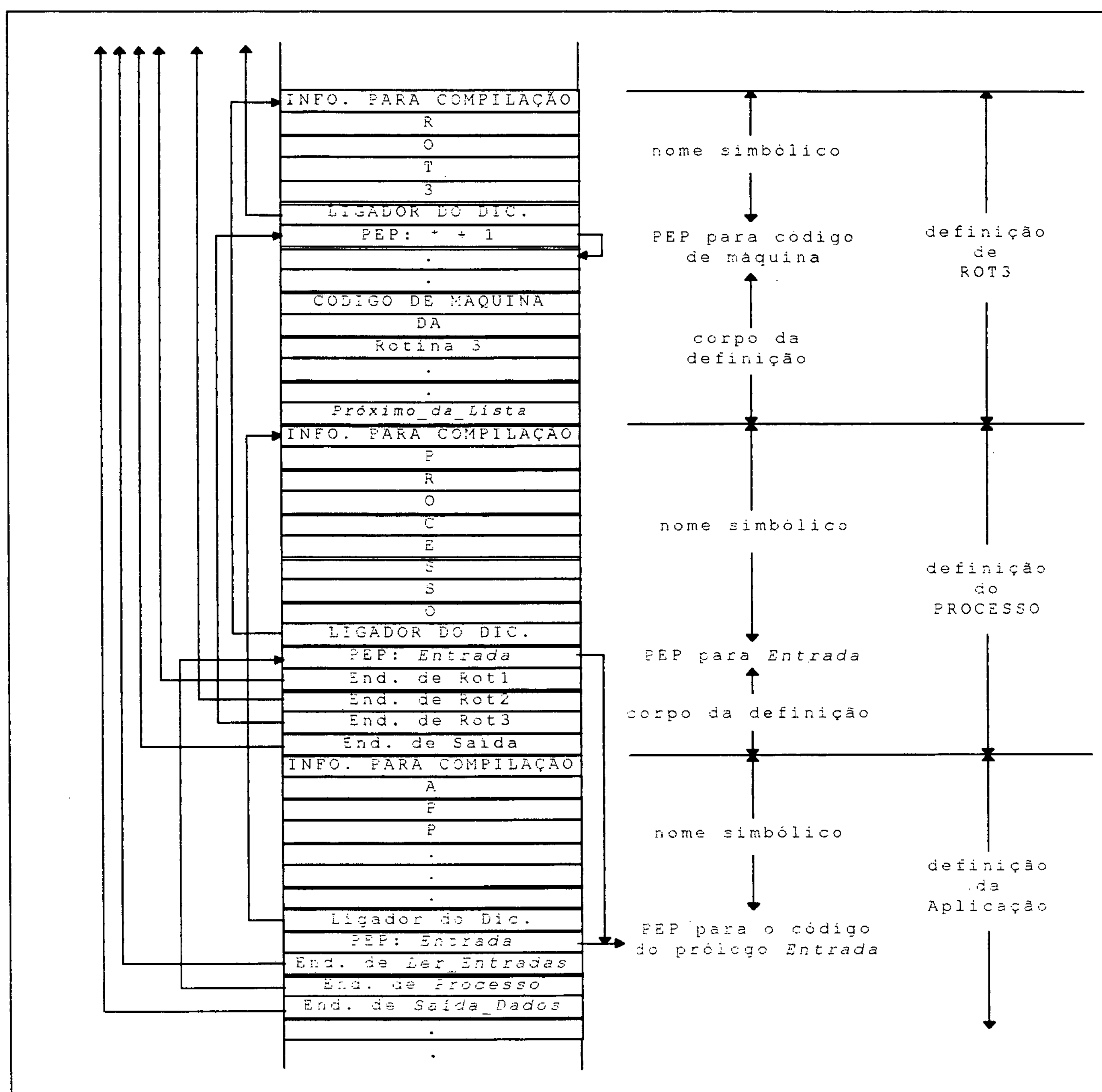


Figura II-Q O Dicionário FORTH.

II.5.5 Compilação

Como o sistema não necessita de análise sintática, a compilação é feita em apenas uma passagem (*one-pass compile*). Basicamente, quando a compilação é disparada por um comando do usuário (a partir do interpretador de texto), é incluído um novo nome de rotina no dicionário. E ao invés de executar os nomes de rotinas subsequentemente entrados, apenas serão armazenados os endereços de seus PEPs em posições sucessivas no corpo da nova rotina. A

detecção de um comando específico completa a compilação, fazendo o sistema retornar ao estado normal de interpretação de texto.

O modo de compilação é implementado em FORTH através de uma modificação no interpretador de texto, de modo que existam dois comportamentos ou "estados", implementados em rotinas diferentes, e um modo de executar a transição entre eles. O estado '0' corresponde ao estado de execução, onde a detecção de um nome de rotina no *buffer* de entrada do sistema, provoca a imediata execução de seu PEP e do seu conteúdo. No estado '1', o estado de compilação, a detecção de nomes de rotinas no *buffer* de entrada não causa sua imediata execução, mas simplesmente a inclusão de uma cópia do endereço de seus PEPs para o corpo da nova rotina em definição.

Na maioria das implementações FORTH, duas rotinas especiais que fazem o controle dos estados, têm os nomes simbólicos ' : ' e ' ; '. A rotina com nome simbólico ' : ' (em FORTH chamada de "definição"), quando executada, usa o próximo nome simbólico que estiver no *buffer* de entrada não como uma palavra a ser procurada no dicionário, mas como uma informação necessária para iniciar uma nova definição nele. Esse nome simbólico será o nome da nova definição. Esta nova definição é então encadeada com as demais definições do dicionário da maneira apropriada, e lhe é atribuída um PEP que aponte para prólogo de *Entrada.* A definição do corpo da rotina está iniciada, e o estado do interpretador externo modificado para '1' (compilação).

Após a execução de ' : ', o interpretador externo analisa os nomes no *buffer* de entrada, procura pela existência de cada um deles no dicionário, e (com certas exceções) adiciona uma copia do endereço de seus PEPs na nova definição, na forma de uma compilação incremental.

A rotina disparada por ' ; ' (em FORTH chamada de "fim-de-definição") é um exemplo de um grupo de instruções que é executada imediatamente mesmo quando no estado de compilação. O modo de diferenciar estas definições, é através da existência de um *bit* precedente sinalizador, que existe em cada definição do dicionário. Se o *bit* é '1', implica que a definição deve ser executada em ambos os estado; se o *bit* for '0', implica que ela deverá ser executada somente no estado de execução.

O *bit* precedente está em '1' na definição de ' ; '. Assim, quando é encontrada no *buffer* de entrada, sua rotina sempre será executada. Sua execução causa o término da definição corrente, adicionando ao seu final, o endereço da rotina *Saída* (usada nos exemplos anteriores), e mudando o estado do interpretador para '0'.

Como exemplo, o texto no quadro ii-s mostra a definição FORTH para a estruturação da rotina de avaliação do polinômio da

figura ii-p.

```
: POLINOMIO DUP A * B @ + * 3 C @ + Z ! ;
```

Quadro II-S Criação de uma definição FORTH chamada POLINOMIO

O nome da nova definição é POLINOMIO (a linguagem não permite acentuação). Assume-se que uma definição prévia tenha atribuído valores para A, B, C, e Z, também previamente definidos. Na figura ii-r são mostradas as mudanças no estado e no dicionário durante a compilação de POLINOMIO.

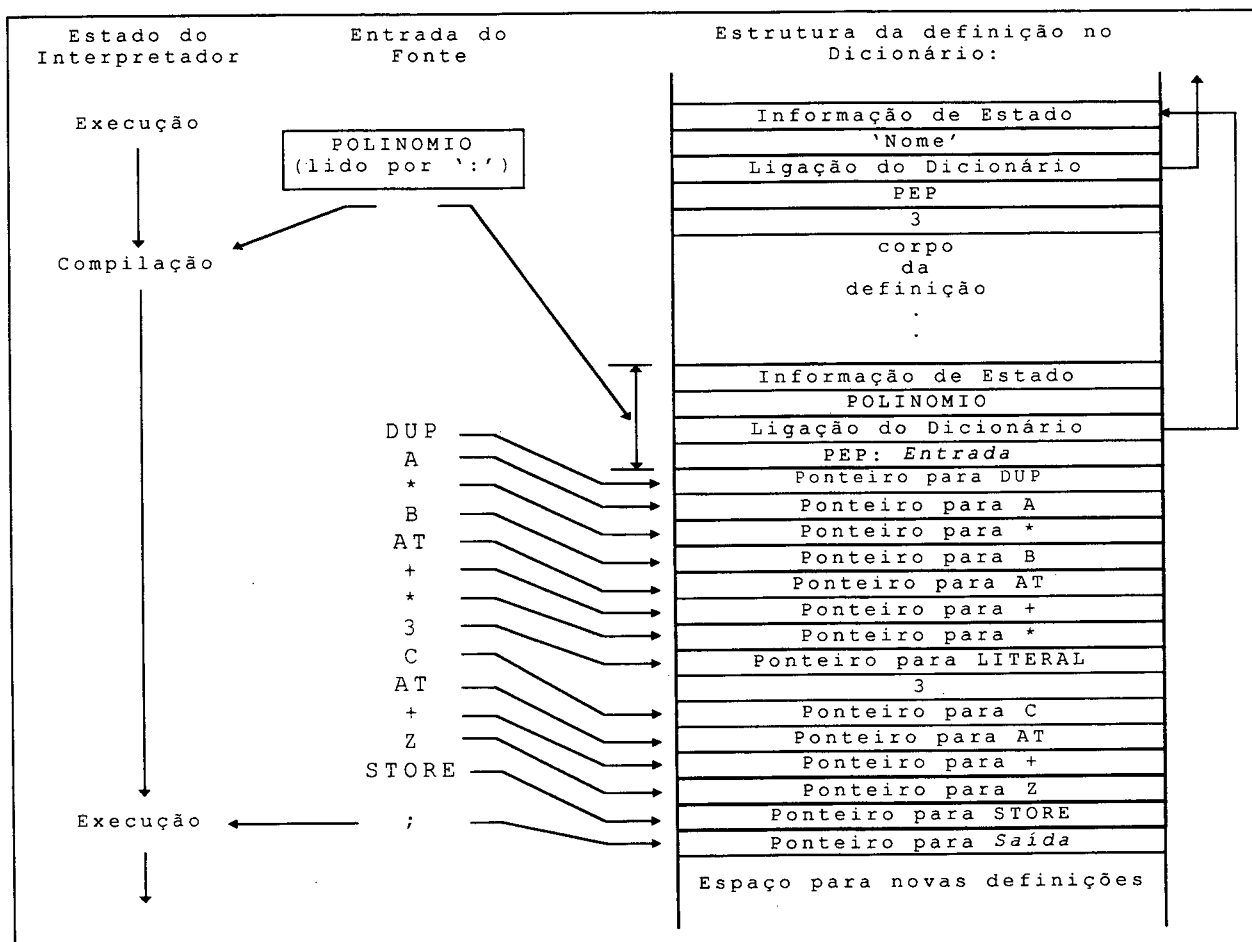


Figura II-R Exemplo de Compilação.

Finalizando, na figura ii-s é mostrado um diagrama de fluxo do interpretador de texto. A parte dentro da área pontilhada corresponde as etapas básicas de compilação. Todos os procedimentos mostrados na figura, podem ser definidos (ou redefinidos dinamicamente pelo usuário) através de uma definição do tipo ": ... ;".

De maneira geral, FORTH baseia-se nos conceitos analisados. Para aplicações específicas são feitas implementações especiais, conforme a necessidade de mais velocidade ou de reducão de memória ou flexibilidade do sistema.

A seguir é abordado um modelo da linguagem FORTH criada especialmente para sistemas dedicados, e com alto grau de "portabilidade".

II.5.6 eFORTH

Conforme TING (1990), o eFORTH é o nome de um modelo de FORTH projetado para ser "portável" para um grande número de processadores recentes e potentes que já existem, e para os que se tornarão disponíveis num futuro próximo. Segue o modelo figFORTH (*forth interest group* - grupo de usuários FORTH), mas sem as limitações de ter como plataforma de *hardware* um processador de 8 bits de dados, e 16 bits para a pilha, nem as limitação de ambiente de programação que existiam quando o modelo figFORTH foi concebido, no início dos anos 80.

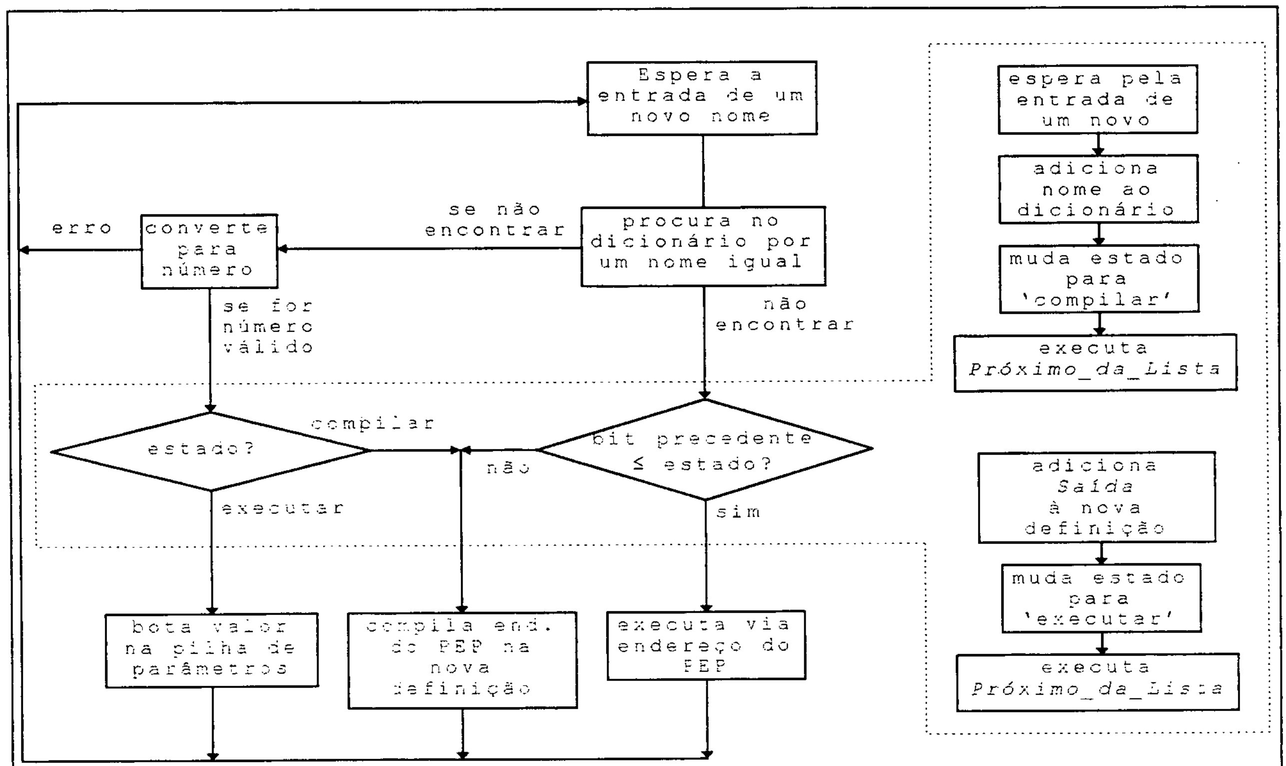


Figura II-S Interpretador FORTH. Nas linhas pontilhadas estão as funções de compilação.

eFORTH foi originalmente implementado em um IBM-PC, no formato MASM (*Microsoft Macro-Assembler*), no ambiente DOS, que lhe fornece os serviços de entrada e saída serial. É um sistema FORTH completo e funcional, com as seguintes características :

* 31 palavras em linguagem de máquina, e 193 em alto nível (FORTH);

* código diretamente alinhavado;

* dicionários separados para nomes e código;

* variáveis do sistema são definidas como variáveis do usuário, para facilitar o funcionamento em EPROM (se for desejado);

* palavras de entrada e saída vetoradas;

* transferência de arquivos através de linha serial;

* palavras especiais para detecção e indicação de erros;

* apenas uma estrutura de repetição (FOR-NEXT);

* segue o padrão FORTH proposto pelo ANSI - *American National Standards Institute*;

* detecção de palavras somente de execução, quando em modo interpretado;

* inclui palavras especiais para depuração;

* flexibilidade para mapeamento em memória.

O sistema é relativamente pequeno, sendo necessários 5 kbytes para o dicionário de código, e 2 kbytes para o dicionário de nomes. Este conjunto de características torna o eFORTH uma base ótima e flexível para o desenvolvimento de *software* dedicado. A listagem do seu código é mostrada no anexo A.

III. Modelo do Sistema Proposto

III.1 Introdução

Sistemas de pequeno porte são sistemas com pequeno número de variáveis de entrada monitoradas e pequeno número de saída para exercer controle no sistema, geralmente duas ou três de entrada e uma ou duas de saída. A necessidade computacional para exercer o controle do sistema pode ser atendida por uma arquitetura baseada em microcontrolador, formando um *hardware* simples e de pequenas dimensões, e com baixo custo.

Para o desenvolvimento de aplicações de controle são oferecidos comercialmente *hardware* de aplicação geral, na forma de "*kits* de desenvolvimento", que necessitam apenas de circuitos de "*interfaceamento*" de sinais de entrada e saída para formar, juntamente com o *software*, o sistema de controle (ou sistema dedicado).

Uma vez construído o "*interfaceamento*" necessário para a utilização do *kit* empregado, o projeto prossegue com o desenvolvimento do programa de controle. Alguns *Kits* de desenvolvimento oferecem linguagens ou programas de depuração residentes nas memórias da placa de controle. Estes programas residentes ou monitores como também são chamados, podem executar comandos no *hardware* de controle, quando conectados via linha de comunicação, geralmente uma linha de comunicação de dados serial, com um terminal de computador. Comandos similares aos comandos de depuração do programa *Microsoft debug* utilizado nos PCs IBM compatíveis, podem ser executados no terminal, assim como também transferir código de programas para serem executados no *hardware* de controle.

Programas monitores favorecem a depuração de programas por permitirem acesso aos registros do microcontrolador (ou microprocessador), como também à memória. Alguns oferecem ainda a possibilidade de executar instruções assembler. Isto é extremamente útil mas não é possível desta forma, criar programas com centenas de linhas de código. Além disso, é

comprovado que é mais produtivo escrever programas em linguagens de mais alto nível. A linguagem FORTH tem ampla utilização em sistemas de controle (dedicados), por oferecer as mesmas vantagens dos programas monitores em relação a depuração de programas, e as vantagens da programação em linguagem de alto nível. A linguagem FORTH situa-se entre o assembler e linguagens de mais alto nível como C e FORTRAN. É considerada pelos programadores como uma linguagem "próxima ao *hardware*" pelo tipo de comandos que possui para a manipulação deste, e pela sua estruturação.

O emprego ideal da linguagem ocorre quando ela está residente no *hardware* de desenvolvimento (modo interpretado), e através de um terminal são executados seus comandos e escritos os programas. Empregando um microcomputador como sistema de desenvolvimento, emulando terminal de comunicação, e fornecendo o meio de armazenamento e edição de programas, forma-se um sistema de baixo custo e ótimo desempenho para o desenvolvimento de sistemas dedicados. No microcomputador editam-se, armazenam-se e transferem-se via linha serial os programas ao *hardware* de desenvolvimento. Pode-se ainda manipular e depurar o programa através de um emulador de terminal.

Nas seções seguintes o sistema é descrito, onde um sistema dedicado, mais a linguagem FORTH, e a técnica de controle difuso são integradas, na intenção de obter um sistema para desenvolvimento de aplicações de controle difuso de baixo custo.

III.2 Estrutura do Sistema

A estrutura básica do sistema é mostrada na figura iii-a.

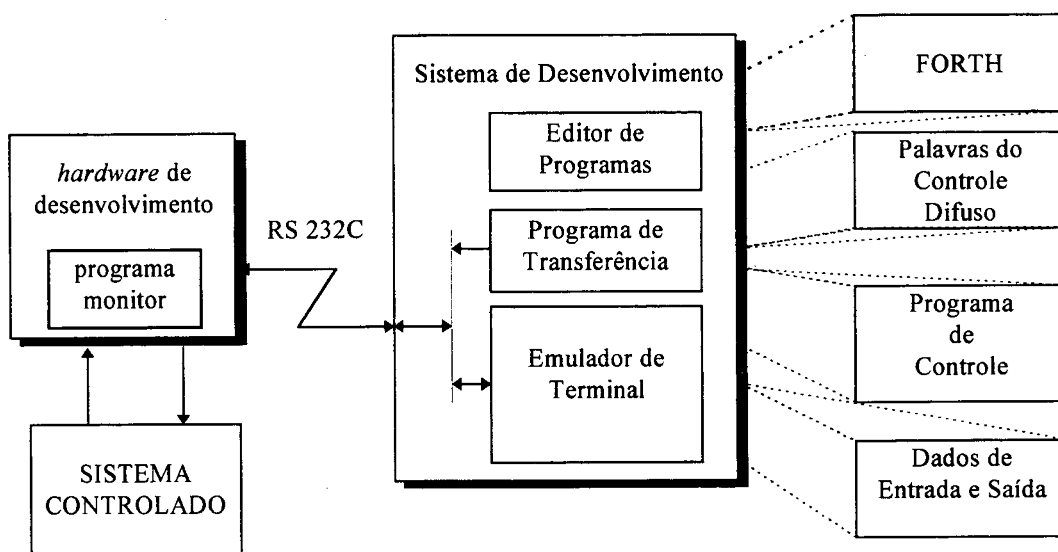


Figura III-A Estrutura do Sistema Proposto.

O *hardware* de desenvolvimento deve atender a requisitos como simplicidade e dimensões reduzidas, de modo que possa ser instalado em gabinete pequeno e isolado de perturbações ambientais como sujeira, vibrações, e radiações eletromagnéticas; é desejável possuir entradas e saídas paralelas de dados que permitam acoplar o sistema à sensores e atuadores que fornecerão à CPU os sinais monitorados (variáveis de entrada) e sinais de controle (variáveis de saída); possuir linha de comunicação serial compatível com as linhas de comunicação utilizadas em PCs (RS 232C), para transferência de programas e dados bem como para entrada de comandos e monitoração de saídas durante a execução do programa. Quando em desenvolvimento de uma aplicação, o *hardware* de desenvolvimento estará localizado junto ao sistema controlado, conectado ao sistema de desenvolvimento via linha de comunicação serial.

O sistema de desenvolvimento deve oferecer a possibilidade de editar e arquivar programas, transferí-los ao *hardware* de desenvolvimento, e emular terminal de comunicação. No lado do *hardware* de desenvolvimento, um programa monitor residente deve fazer a comunicação com o sistema de desenvolvimento.

A linguagem de programação utilizada para implementar o algoritmo de controle, como também as definições do algoritmo (rotinas), e o programa de controle, serão armazenados no

PC e transferidos ao *hardware* de desenvolvimento quando o sistema é inicializado. Os dados de entrada (no *hardware* de desenvolvimento) são fornecidos pelo teclado do sistema de desenvolvimento, e os dados de saída (provenientes do *hardware* de desenvolvimento) serão recebidos pelo programa emulador de terminal e mostrados na tela do PC, podendo armazená-los se necessário. O programa de controle rodará em *loop*, exercendo a função de controlar o sistema.

III.3 Utilização do Sistema

A utilização do sistema em situação real de desenvolvimento deve ter as seguintes etapas:

1- alimentação do sistema: PC e unidade remota (com linha de comunicação já conectada);

2- o sistema FORTH é transferido para a unidade remota e colocado em execução;

3- o algoritmo de controle difuso implementado em FORTH é transferido à unidade remota;

Neste momento dois procedimentos podem ser adotados: com o FORTH rodando no *hardware* de desenvolvimento e as definições do algoritmo difuso constando do seu dicionário, tanto pode-se enviar para o *hardware* um bloco de palavras de controle, ou até mesmo um programa completo, que tenham sido previamente editadas e arquivadas para testar e verificar seu funcionamento, como pode-se editar diretamente no FORTH em execução no *hardware* de desenvolvimento, via emulador de terminal, as palavras que irão formar o programa de controle. Cria-se uma nova palavra, e testa-se seu funcionamento. Procede-se desta maneira iterativa até obter o programa completo. Generalizando, os passos seguintes serão:

4- editar/transferir programa de controle;

5- disparar o programa de controle;

6- fornecer/coletar dados, ou alterá-los.

O procedimento ideal é utilizar um editor de texto "paralelamente" a um emulador de terminal. Definida uma nova palavra, ela é transferida e executada no *hardware* de desenvolvimento via emulador de terminal. Se não produzir o comportamento desejado, redefine-se a palavra até alcançar o que se espera. Procede-se desta maneira, definindo novas palavras e utilizando as já definidas e testadas, até obter o programa completo e funcional.

III.4 Algoritmo e Linguagem

A maior parte dos algoritmos para controle difuso tem sido implementados em C e Pascal. Algoritmos implementados nestas linguagens, como no programa *FuzzyFan* por Viot (1993), são programas que através dos recursos da linguagem, permitem ao programador implementar uma solução específica para determinado problema de controle. Tanto empregando controle difuso ou não. A linguagem em si não oferece recursos específicos para que sejam criados programas de controle de qualquer tipo. A FPL® - *Fuzzy Programming Language* - foi escrita especificamente para a criação de programas de controle difuso. É oferecida pelo fabricante com um sistema de desenvolvimento de *software* completo, gerando código para o DFP™ - *digital fuzzy processor* - FC110. Este processador possui instruções assembler específicas para programas de controle difuso em seu "micro-código." Programas escritos nesta linguagem utilizam comandos específicos para a execução de controle difuso. O DFP™ FC110 possui caracterísitcas de microcontrolador, com dispositivos de entrada e saída internos ao *chip*, e também unidade aritmética para cálculos com ponto flutuante.

Esta característica do sistema e mais especificamente da linguagem, fazem com que o projetista que busca resolver um problema de controle através da lógica/controle difuso, se preocupe com a melhor aplicação da técnica e com a solução, despreocupando-se com o "modo" com que irá implementá-la na linguagem que escolher. Seu raciocínio envolve-se com a técnica de controle difuso, com a modelagem do sistema, e com a eficiente aplicação da técnica para produzir o resultado desejado.

Visando alcançar estes mesmos resultados, e expandir para microcontroladores de uso comum e aplicações gerais em controle, e baseando-se nesta linguagem para definir em FORTH, linguagem "portável" e amplamente empregada na criação de novas linguagens, um conjunto de

funções, na terminologia FORTH chamados de "palavras", específicas para a criação de programas de controle difuso.

III.4.1 O Algoritmo de Controle

O algoritmo de controle difuso segue a estrutura de blocos do CLD da figura II-E, e descrito no quadro iii-a.

```
- leitura dos dados de entrada (variáveis de entrada)
     - fuzzyficação dos dados de entrada
     - avaliação de regras
     - defuzzyficação
- escrita dos dados de saídas (variáveis de saída)
```

Quadro III-A Algoritmo de Controle.

Quando o controle estiver atuando, e o *hardware* conectado ao sistema controlado, o algoritmo será executado continuamente em forma de *loop*. O programa de controle deve ter a forma mostada no quadro iii-b.

```
- início
- leitura dos dados de entrada (variáveis de entrada)
     - fuzzyficação dos dados
     - avaliação de regras
     - defuzzyficação dos dados
- escrita dos dados de saídas (variáveis de saída)
- volta ao início
```

Quadro III-B Etapas do Programa de Controle.

As etapas entre o início e a volta ao início são executadas continuamente, mantendo o comportamento do sistema controlado conforme a manipulação dos dados no programa. Estas etapas, quando implementadas em definições (palavras) FORTH, executando em um *loop* num programa em FORTH, são mostradas no quadro iii-c.

```
1    : CONTROLAR
2     BEGIN
3          LerDados
4          Fuzzyficar
5          AvaliarRegras
6          Defuzzyficar
7          EscreverDados
8     AGAIN
9    ;
```

Quadro III-C Loop de Controle em FORTH.

Na linha 1 é definida a palavra CONTROLAR. Ela realiza a função de controlar o sistema. Os símbolos ':' na linha 1 e ';' na linha 9, indicam início e fim, respectivamente, da nova palavra definida. 'BEGIN' na linha 2 e AGAIN na linha 8 demarcam início e fim de *loop*, repetindo tudo o que estiver entre eles (as palavras que executam o algoritmo) continuamente. As palavras nas linhas 3 a 7 executam o controle, e sua abordagem será feita na próxima seção.

III.4.2 Definição do Algoritmo

Conforme definido na seção anterior, o algoritmo de controle pode ser divido em 5 etapas:

```
LerDados
Fuzzyficar
AvaliarRegras
Defuzzyficar
EscreverDados
```

Em "LerDados" são recebidos dados dos sensores na forma binária, via dispositivos de entrada do microcontrolador. A seguir os dados passam pelo processo de fuzzyficação em "Fuzzyficar", e os valores resultantes são avaliados pelas regras em "AvaliarRegras". Após são defuzzyficados em "Defuzzyficar", e jogados aos dispositivos de saída do microcontrolador em "EscreverDados", atuando sobre o sistema controlado.

Antes que o algoritmo possa ser executado, é necessário que seja fornecido ao sistema os dados relativos ao processo a ser controlado. Estes dados irão formar, conforme a figura II-C, a base de dados que será utilizada na fuzzyficação dos dados de entrada, na avaliação das regras, e na defuzzyficação, para obtenção dos dados de saída.

Para determinar estes dados, devem ser levantadas as variáveis linguísticas de entrada e saída do sistema, obtidas do conhecimento especialista que se tem do processo.

A determinação destes dados inicia com o levantamento das variáveis linguísticas de entrada e saída do sistema. Este levantamento pode ter o auxílio de alguem com bom conhecimento do processo a ser controlado. A seguir determina-se o universo de discurso de cada variável (entrada e saída), seus conjuntos difusos com seus nomes e suas funções de pertinência. Um procedimento comum para determinar conjuntos difusos é atribuir "nomes" aos estados que a variável linguística pode assumir, no processo controlado. Para a determinação das funções de pertinência não há, até o momento, métodos que forneçam tipos e formas de funções de pertinências para variáveis linguísticas, a partir do tipo de processo que se quer controlar. O procedimento, na prática, é adotar tipos genéricos de funções de pertinência, e modificá-las a medida que forem sendo experimentadas na prática, ou em simulações. Nesta fase, um sistema que possibilite verificar o comportamento do sistema controlado para determinada função de pertinência escolhida, e oferecer facilidade e rapidez na alteração e teste da função, traz retorno rápido ao projetista para entender que efeitos produzem as alteração realizadas, no comportamento do sistema. Tipos genéricos de funções de pertinência são mostrados na figura II-H.

Na representação de funções de pertinência com *singletons*, o procedimento de determinação dos conjuntos é o mesmo, diferindo apenas na função de pertinência, representada por um valor único no universo de discurso.

A seguir cria-se a base de regras a partir da estratégia de controle. Deve ser definida baseando-se no comportamento que se espera do sistema controlado, e também no conhecimento especialista do processo.

O levantamento destes dados, resumido no quadro iii-d, deverá assumir uma forma específica dentro do programa de controle. Nas seções seguintes será estabelecida esta forma, como também a forma como as demais etapas do algoritmo devem ser declaradas.

- determinação das variáveis linguísticas de entrada e saída;
- determinação do universo de discurso;
- determinação dos conjuntos difusos e funções de pertinência para entradas e saídas;
- criação da base de regras;

Quadro III-D Etapas do levantamento de dados.

III.4.2.1 Entrada e Saída de Dados

Em sistemas com microcontroladores os dados são lidos e escritos em portas de entrada e saída paralelas ou seriais, em dispositivos internos ao *chip,* ou de dispositivos externos mapeados em memória. A implementação do procedimento de leitura e escrita depende da linguagem utilizada. A fonte de dados (sensores) que variam conforme o processo controlado, deve ter seus sinais convertidos para sinais binários antes de serem entregues ao microcontrolador. A leitura de dados atribui valores às variáveis de entrada. A escrita de dados atribui o valor resultante do processamento, contido na variável de saída, ao dispositivo de saída, ou atuador.

III.4.2.2 Declaração das Variáveis de Entrada e Saída, e Universo de Discurso

Declaram-se as variáveis de entrada da seguinte forma:

```
FVARIN VAR n1 n2 n3 UNIVERS
```

As palavras destacadas em negrito são palavras criadas especialmente para utilização em programas de controle difuso, constarão do dicionário FORTH,. A palavra 'FVARIN' (*Fuzzy VARiable INput*) cria no dicionário FORTH a variável linguística de entrada de nome 'VAR'. Os valores n1, n2 e n3 definem os parâmetros para o universo de discurso da variável criada, onde:

```
n1 = limite inferior do universo de discurso;
n2 = limite superior do universo de discurso;
n3 = número de partições entre n1 e n2.
```

O número de partições determina a granularidade do universo de discurso. Estes parâmetros serão armazenados no cabeçalho da variável criada, para uso posterior na criação dos conjuntos difusos e funções de pertinência. Também é alocado no cabeçalho, espaço para o valor lido das entradas, e atribuído à variável.

As variáveis de saída são declaradas da seguinte forma:

```
FVAROUT VAR
```

'FVAROUT' (*Fuzzy VARiable OUTput*) cria no dicionário FORTH a variável de saída de nome genérico 'VAR'. Não são necessários parâmetros para universo de discurso, devido a utilização de *singletons* como funções de pertinência de saída. É alocado no cabeçalho, espaço para o valor de saída da variável.

III.4.2.3 Conjuntos Difusos e Funções de Pertinência

Um conjunto difuso de uma variável linguística é definido da seguinte forma:

```
VAR MEMBER NOME n1 n2 n3 FUNÇÃO
```

'VAR' é a variável previamente criada. 'MEMBER' cria no dicionário FORTH o conjunto difuso com o nome genérico 'NOME'. Os valores n1, n2, e n3 são parâmetros de entrada para a palavra 'FUNÇÃO', que definirá a funcão de pertinência do conjunto difuso. Estes parâmetro dependem da função de pertinência que será atribuída ao conjunto difuso. Três funções de pertinência básicas, são mostradas na figura iii-b.

Em '-SLOPE' os parâmetros necessários são os valores no universo de discurso n1 onde inicia a rampa de descida, e o valor n2 onde termina a rampa. Em '+SLOPE' n1 é o início da

rampa de subida, e n2 o final. Em 'TRIANGLE' é fornecido o valor n1 onde inicia a subida, o valor n2 onde a função atinge o máximo, e o valor n3 onde termina a descida.

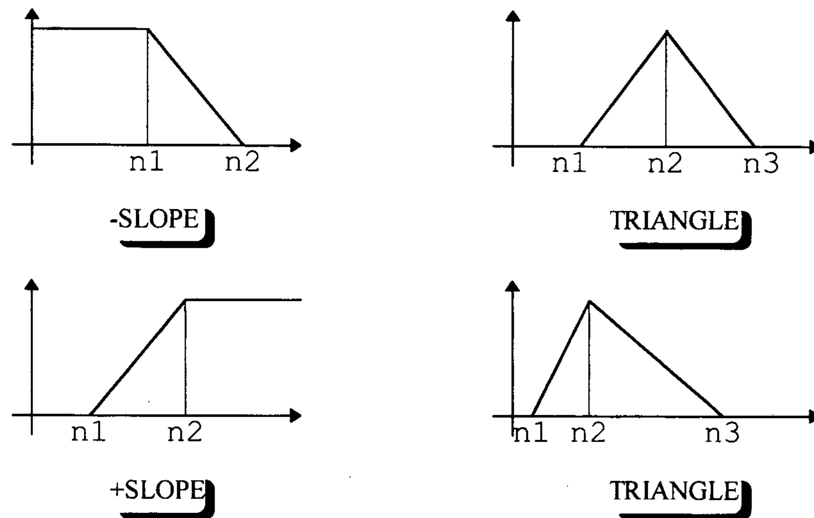


Figura III-B Funções de Pertinência mais Comuns.

O quadro iii-e mostra a forma de declarar os três tipos de funções de pertinência em um programa.

```
VAR MEMBER NOME1 n1 n2 -SLOPE
VAR MEMBER NOME2 n1 n2 n3 TRIANGLE
VAR MEMBER NOME3 n1 n2 +SLOPE
```

Quadro III-E Declaração das Funções de Pertinência.

'VAR' é o nome da variável a qual os conjuntos difusos criados pertencem. 'MEMBER' cria no dicionário FORTH um conjunto difuso de nome genérico 'NOME', com dados extraídos da variável VAR, para relacionar o universo de discurso a estes conjuntos. São criados três conjuntos: 'NOME1, NOME2, e NOME3'. Não há restrições quanto ao uso de letras maiúsculas e minúsculas, tanto para conjuntos difusos como para variáveis linguísticas, mas não deve haver nomes iguais.

Os dados n1, n2 e n3 são os parâmetros da função de pertinência, conforme a figura iii-b. Estes valores se referem ao universo de discurso real da variável. As funções de pertinência produzem tabelas de valores de grau de pertinência, com a extensão em bytes definidas pelo

tamanho do universo de discurso estabelecido quando da criação da variável linguística a qual o conjunto difuso pertence. Declarar o nome da variável linguística antes de criar o conjunto difuso, é a forma de associar o conjunto difuso à variável linguística que fornecerá o valor de entrada para etapa de *fuzzyficação*.

A definição de um *singleton* tem a forma definida no quadro iii-f.

```
n SINGLETON NOME
```

Quadro III-F Definição de Singleton.

Aqui definiu-se um *singleton* de nome genéricos 'NOME', com valor 'n' no universo de discurso da variável linguística de saída. Este valor é armazenado no cabeçalho da definição do *singleton*, e utilizado na defuzzyficação.

III.4.2.4 Fuzzyficação

Na fuzzyficação obtém-se graus de pertinência para os valores lidos dos sensores e atribuídos às variáveis linguísticas de entrada. Esta etapa é definida como uma palavra FORTH no quadro iii-g.

```
: FUZZYFICA
    MEMBERS[ NOME1 NOME2 NOME3 ]> VAR FUZZYFI ;
```

Quadro III-G Declaração da Fuzzificação.

Uma nova palavra é criada no dicionário FORTH com o nome 'FUZZYFICA'. Os símbolos ':' e ';' marcam início e fim da definição da palavra. A palavra 'MEMBERS[' delimita o início e ']>' delimita o fim, da declaração dos conjuntos difusos que entrarão na fuzzyficação da variável linguística de entrada de nome genérico VAR.

"FUZZYFI" é o comando de execução do procedimento de fuzzyficação. Os valores resultantes da fuzzyficação (grau de pertinência) serão armazenados nos cabeçalhos dos conjuntos difusos 'NOME1, NOME2 e NOME3', para utilização na avaliação das regras.

III.4.2.5 Base de Regras

A base de regras é composta pelo conjunto de regras definidas para o programa de controle. As regras são declaradas como palavras FORTH, sendo cada regra uma palavra no dicionário FORTH. Diferente da FPL, onde é citado na regra a variável linguística e o seu conjunto difuso, tanto no antecedente como no consequente, aqui citam-se apenas aos conjuntos difusos sem citar a variável linguística a qual pertencem.

O critério de decisão adotado com mais frequência em aplicações simples, é o "máximo dos mínimos". Neste critério atribui-se ao conjunto difuso no consequente, o maior valor de grau de pertinência entre o seu valor atual, e o menor valor de grau de pertinência resultante dos conjuntos difusos no antecedente. O algoritmo de avaliação de uma regra é mostrado no quadro iii-h.

```
- colocar na pilha o grau de pertinência dos conjuntos difusos
  no antecedente;

- verificar se algum dos valores na pilha é zero:

     se for: aborta a avaliação da regra;

- deixar na pilha apenas o menor valor;

- colocar na pilha o valor do grau de pertinência do conjunto
  difuso no consequente;

- deixar na pilha apenas o maior valor;

- atribuir o valor na pilha ao conjunto difuso no consequente.
```

Quadro III-H Algoritmo de Avaliação de Regras.

A declaração de uma regra pode ter duas formas, como mostra as duas declarações no quadro iii-i.

```
: REGRA1 IFF NOME1 THENF NOME2 END-RULE ;
: REGRA2 IFF NOME3 ANDF NOME4 THENF NOME5 END-RULE ;
```

Quadro III-I Declaração de uma Regra.

Na primeira regra declarada, o grau de pertinência do antecedente é atribuído ao consequente sem ter que escolher o menor valor de grau de pertinência dos conjuntos difusos no antecedente, como ocorre na segunda regra. As palavras 'NOME1 a NOME5', são nomes genéricos de conjuntos difusos. Em qualquer das declarações, se ocorrer valor 0 na fuzzyficação, a regra é abortada sem realizar sua completa avaliação. Este procedimento traz velocidade na interpretação das regras, evitando que uma regra sem efeito, seja avaliada até o fim, fazendo o sistema perder tempo.

No exemplo foram declaradas duas novas palavras no dicionário FORTH: 'REGRA1' e 'REGRA2'. Os resultados da avaliação das regras será utilizado na etapa de defuzzyficação.

III.4.2.6 Defuzzyficação

Na defuzzyficação é obtido um valor *crisp* a partir dos valores de grau de pertinência dos conjuntos difusos da variável linguística de saída. Esta etapa é definidas como uma palavra FORTH no quadro iii-j.

```
: DEFUZZYFICA
      VAR MEMBERS[ NOME1 NOME2 NOME3 ]> DEFUZZYFI ;
```

Quadro III-J Declaração da Defuzzyficação.

É criada uma nova palavra no dicionário FORTH, com o nome 'DEFUZZYFICA'. As palavras 'MEMBERS[' e ']>' delimitam início e fim da declaração dos conjuntos difusos 'NOME1, NOME2, e NOME3' que entrarão na defuzzyficação da variável linguística de saída de nome genérico VAR.

"DEFUZZYFI" é o comando de execução do procedimento de defuzzyficação. O valor resultante da defuzzyficação, o valor *crisp,* será armazenado no cabeçalho da variável de saída, para posteriormente ser atribuído ao dispositivo de saída.

III.4.2.7 Loop de Controle

loop de controle é a rotina que contém as etapas do algoritmo de controle, executada continuamente. A definição desta rotina em FORTH, contendo as etapas do controle, é mostrada no quadro iii-k.

```
: CONTROLAR
  BEGIN
      LerDados
      Fuzzyficar
      AvaliarRegras
      Defuzzyficar
      EscreverDados
  AGAIN
;
```

Quadro III-K Palavras do Loop de Controle.

A definição da palavra FORTH de nome 'CONTROLAR', executa continuamente as palavras entre 'BEGIN' e 'AGAIN'.

III.5 Estrutura do Programa de Controle

Partindo das concepções estudadas, estruturou-se o projeto em dois blocos: o bloco das declarações, e o bloco do programa de controle em si. O conteúdo do bloco das declarações é mostrado no quadro iii-l.

As declarações deste bloco utilizam palavras do dicionário FORTH escritas especificamente para a implementação de programas para controle difuso.

```
- declaração das variáveis linguísticas de entrada;

- declaração dos conjuntos difusos das variáveis linguísticas de
  entrada;

- declaração das variáveis linguísticas de saída;

- declaração dos conjuntos difusos das variáveis linguísticas de
  saída;

- declaração da fuzzyficação;

- declaração das regras;

- declaração da defuzzyficação;
```

Quadro III-L Conteúdo do bloco de declarações.

No bloco do programa de controle constam o programa de controle, e as definições de palavras que acessam o *hardware* para leitura e escrita de dados.

```
- definições de entrada e saída;

- programa de controle;
```

Quadro III-M Conteúdo do bloco de programa.

Definida esta estrutura, no capítulo seguinte será abordada a implementação das palavras FORTH que serão utilizadas na escrita dos programas de controle difuso.

IV. Implementação e Simulação

IV.1 Introdução

Neste capítulo é abordada a implementação de um sistema de controle difuso visando uma aplicação específica. A aplicação é analisada, o modelo difuso é criado, é proposta a solução em termos de controle difuso, e implementado o programa de controle. Para testar o funcionamento do sistema, são simuladas as entradas e saídas fornecendo e recebendo dados através de um emulador de terminal conectado via serial; ao *hardware* dedicado.

IV.2 O Sistema Simulado

Supondo que um sistema dedicado exerça o controle de um guindaste movimentador de cargas, que se desloca horizontalmente, conforme mostra a figura iv-a:

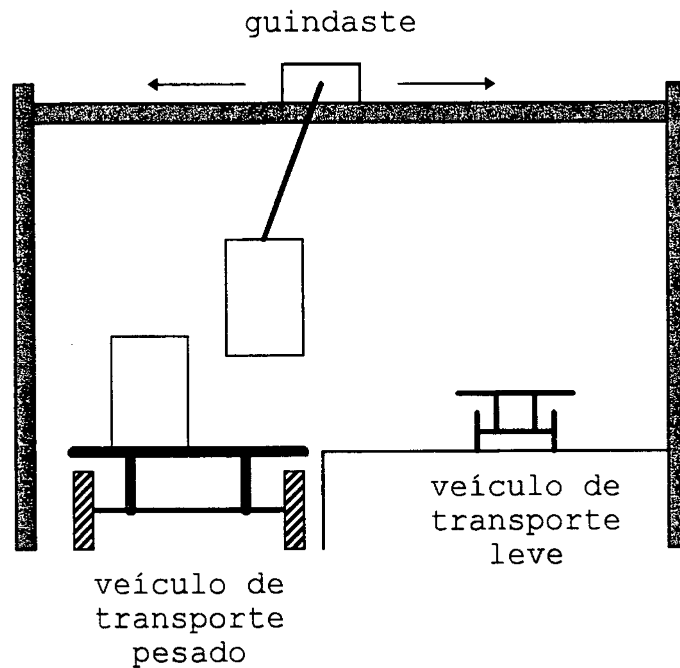


Figura IV-A Guindaste Movimentador de Cargas.

O guindaste leva *containers* do veículo de transporte pesado para o veículo de transporte leve. A movimentação da carga deve ser rápida para desocupar o veículo pesado o mais rápido possível. O *container* não pode sofrer impacto para não danificar o conteúdo. O movimento de retirada do *container* do veículo pesado, como o movimento de largar o *container* no veículo leve, deve então ser controlado. Por outro lado, a rapidez do descarregamento depende da velocidade com que o guindaste retira os *containers*. Isto implica em mover rapidamente a carga de um para outro veículo. Então deve-se controlar a velocidade com que o guindaste inicia o movimento de transporte da carga, procurar atingir a velocidade máxima possível, e controlar a velocidade com que se aproxima do ponto de descarga, até alcançar a posição desejada. Esta situação pode ser controlada monitorando-se o ângulo formado entre o cabo que suspende a carga e o eixo vertical, atuando sobre a velocidade da parte móvel. O movimento deve ser iniciado lentamente, sem deixar formar um ângulo muito grande, e ir aumentando a velocidade gradativamente conforme a inércia vai sendo vencida, até que a distância comece a se tornar menor, quando então a velocidade de aproximação deve ser diminuida, e atingir o ponto de descarga suavemente. Desta descrição do funcionamento, percebe-se que as variáveis a serem monitoras serão ângulo e distância, e a variável controlada será a velocidade.

IV.3 Modelagem Difusa do Sistema

As variáveis linguísticas de entrada serão ANGULO e DISTANCIA. A variável linguística de saída será VELOCIDADE. Como ponto inicial, baseado em ALTROCK (1993), devem ser estipulados valores para o universo de discurso e conjuntos difusos destas variáveis, como mostrado na figura iv-b.

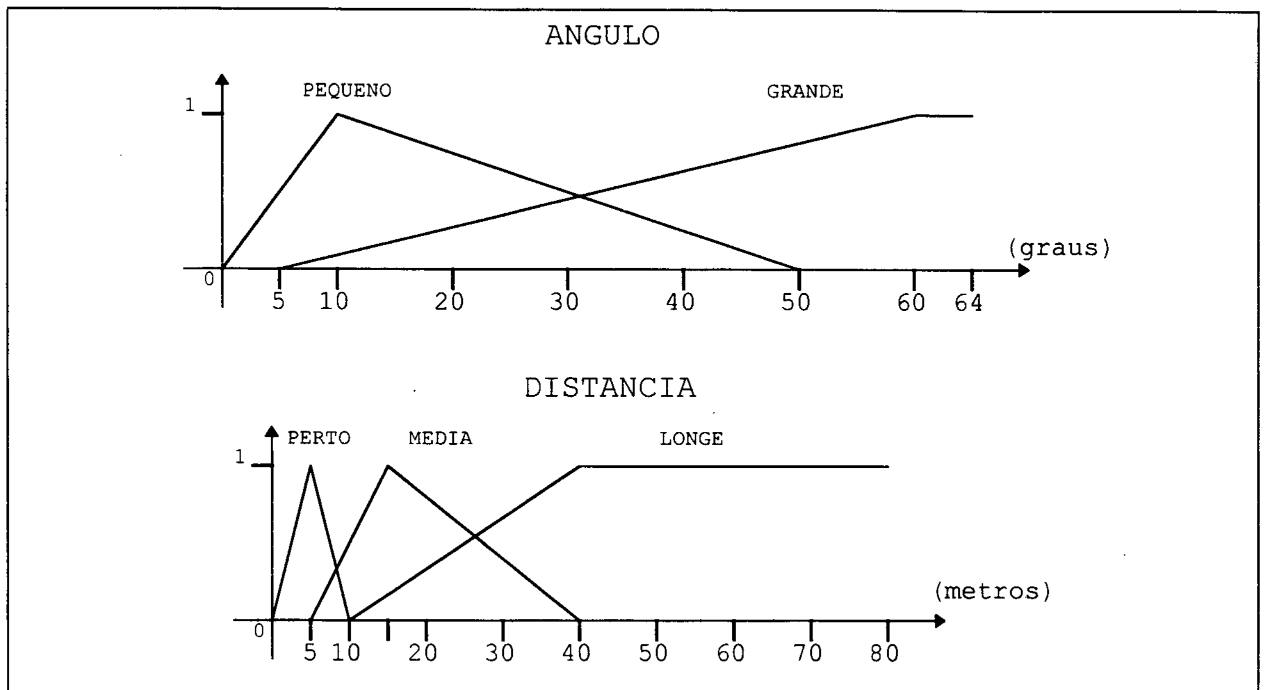


Figura IV-B Variáveis Linguísticas e Conjuntos Difusos do Sistema (Altrock, 1993).

Determinadas as variáveis linguísticas e os conjuntos difusos, define-se a estratégia de controle para o sistema, representada na forma de "matriz linguística":

(ANGULO) \ (DISTANCIA)	PERTO	MEDIA	LONGE
PEQUENO	VEL-MEDIA	VEL-MEDIA	VEL-ALTA
GRANDE	VEL-BAIXA	VEL-BAIXA	VEL-MEDIA

Figura IV-C Matriz Linguística.

Modelada a aplicação, abordar-se-à a estruturação do sistema de controle e desenvolvimento.

IV.4 Estrutura do Sistema

O sistema subdivide-se em duas partes principais: o sistema de controle, e o sistema de desenvolvimento. O sistema de controle atua sobre sistema controlado, enquanto o sistema de desenvolvimento apenas serve como meio de desenvolvimento da solução de controle. A estrutura genérica do sistema é mostrada na figura iv-d.

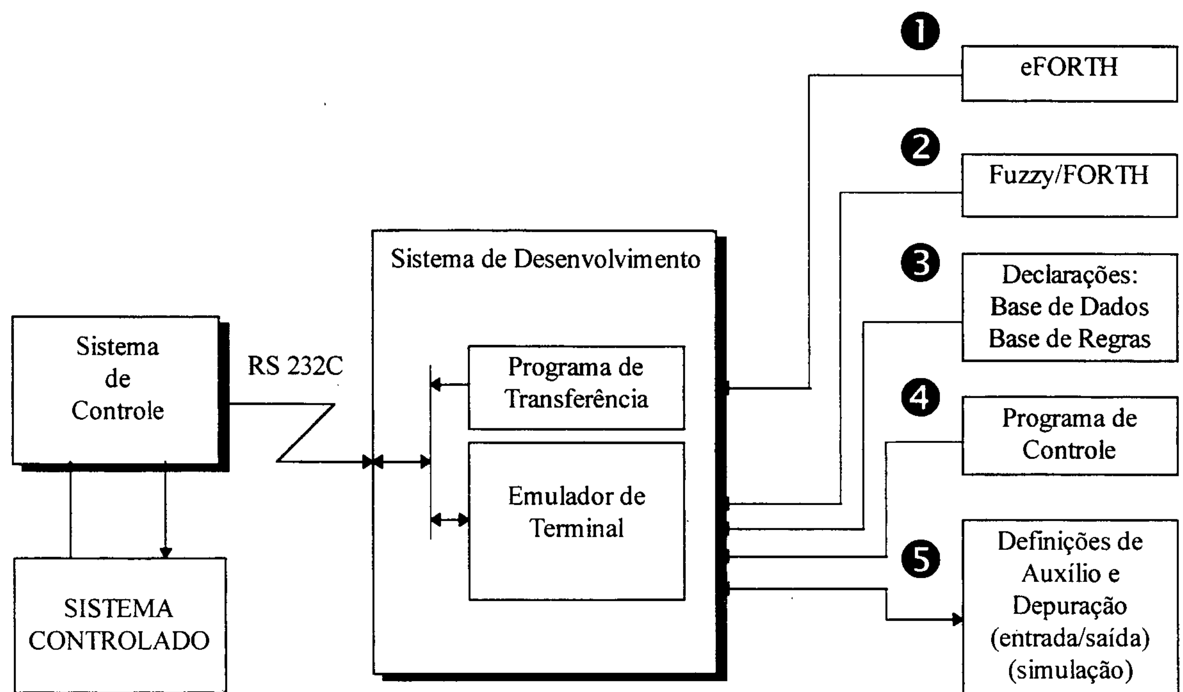


Figura IV-D Estrutura do Sistema.

O sistema de controle comunica-se com o sistema de desenvolvimento via linha de comunicação serial, para transferência de programas e dados. Portas de entrada e saída são conectadas ao sistema a ser controlado.

O sistema de desenvolvimento envia comandos para o sistema de controle através de um emulador de terminal, e envia arquivos através de um programa de transferência de arquivos via serial. Cinco blocos de programas são armazenados no sistema de desenvolvimento. O conteúdo dos blocos é descrito no quadro iv-a.

1 - *versão da linguagem FORTH utilizada no sistema de controle;*

2 - *definições FORTH para programas de controle difuso;*
3 - *definições contendo declarações de variáveis e regras;*
4 - *programa de controle;*
5 - *definições auxiliares para testes, depuração, e entrada e saída de dados na simulação.*

Quadro IV-A Conteúdo dos blocos da estrutura do sistema.

IV.5 Utilização do Sistema

A estrutura implementada é mostrada é figura iv-e.

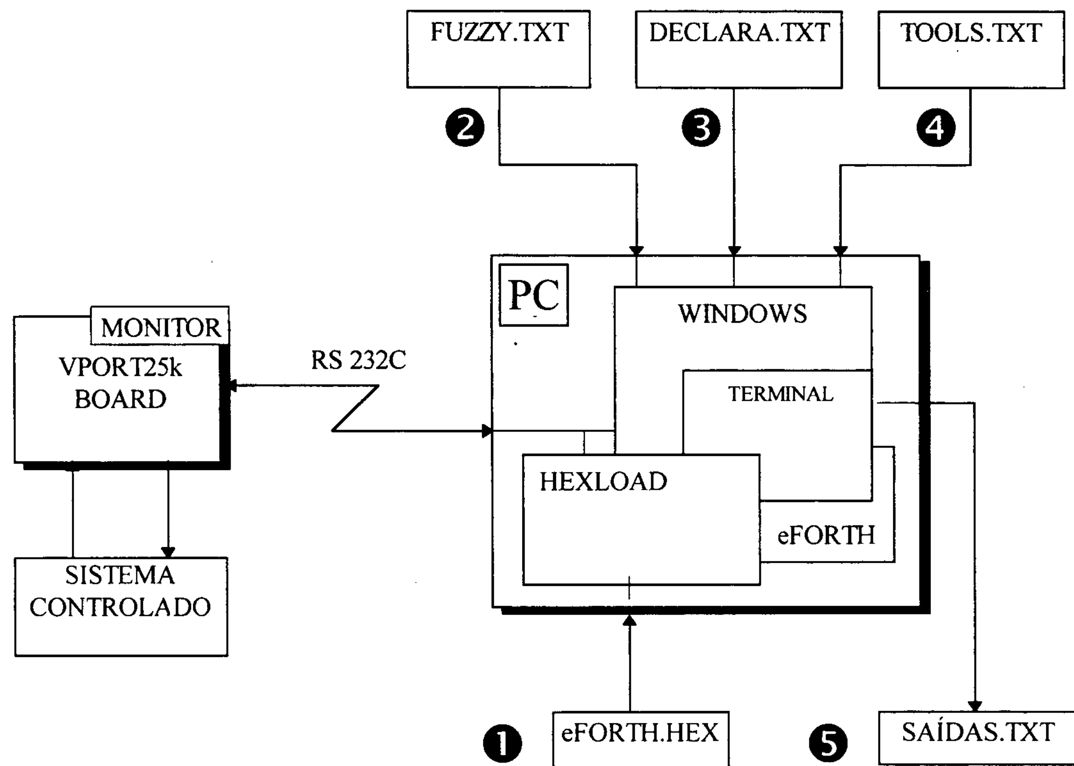


Figura IV-E Sistema Implementado.

O sistema de controle consiste de um *kit* de desenvolvimento VPORT-25k. A placa do sistema, mostrada na figura IV-F, possui um microcontrolador ©NEC V25®, 128 Kbytes de EPROM com programa monitor residente e protocolo para comunicação e transferência de programas via serial, 64 Kbytes de RAM. As características do processador constam no anexo D.

A vantagem da utilização deste microcontrolador, é sua compatibilidade de *software* com os microprocessador Intel® 8086. Isto significa que as ferramentas de *software* utilizadas em PCs IBM® compatíveis, amplamente difundidas, conhecidas e utilizadas, e de baixo custo, podem ser utilizadas para o desenvolvimento de *software* para este microcontrolador. Ele possui duas linhas de comunicação serial, sendo que a linha 1 será utilizada para a comunicação serial com o sistema de desenvolvimento. A comunicação é estabelecida a 19200 *bps*, 8 *bits*, sem paridade e 1 *stop bit*. O programa monitor residente oferece opções de *debug* similares as do *Microsoft®* *debug* que acompanha o sistema operacional para *PCs* IBM® compatíveis, *MSDOS®*. Algumas ferramentas de desenvolvimento de *software* acompanham o *Kit*, como o conversor/relocador de arquivos binários para formato Intel® (*Intel HEX86 Format*), e um programa para transferência serial de arquivos neste formato.

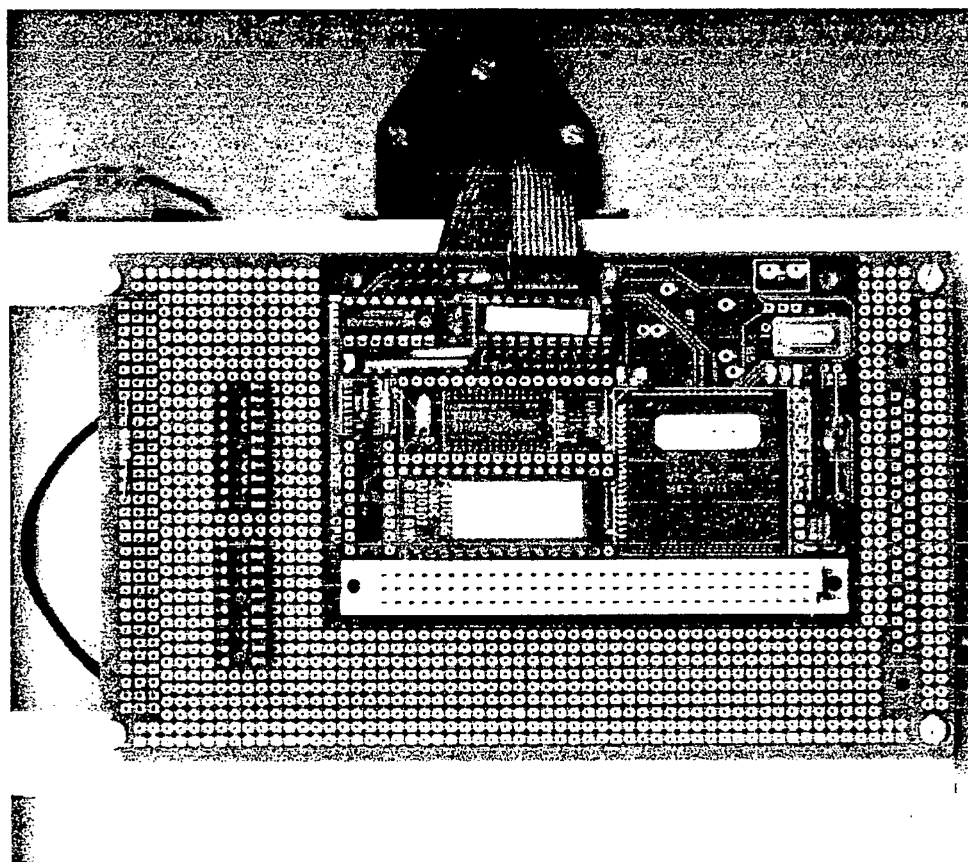


Figura IV-F Hardware do Sistema de Controle - placa VPORT-25k.

Como sistema de desenvolvimento é utilizado um PC IBM® compatível. Nele são armazenados os arquivos de programa representados pelos blocos numerados de 1 a 5 na figura iv-d, e utilizadas as ferramentas de *software*.

O bloco 1 corresponde à versão eFORTH da linguagem FORTH. O arquivo eFORTH.HEX é obtido da seguinte sequência de montagem, partindo-se do fonte em assembler (eFORTH.ASM):

1- montagem com o *Microsoft Macro Assembler* (MASM), gerando o arquivo eFORTH.OBJ;

2- *link*-edição com o *Microsoft Linker* (LINK), gerando o arquivo eFORTH.EXE;

3- relocação e conversão para formato Intel com o *Locate exe-file converter*, gerando o arquivo eFORTH.HEX;

A relocação prepara o programa para ser executado no endereço físico 0100:0100 da placa VPORT-25k. Nesta área está mapeada a memória RAM.

Obtido o arquivo, sua transferência para a placa VPORT-25k é feita com o programa 'HEXLOAD'. Após a transferência o eFORTH está instalado na placa mas fora de execução. Estas operações tanto podem ser realizadas no ambiente DOS, como no ambiente *Windows*, desde que seja utilizado um executor de comandos DOS. As etapas seguintes são realizadas sobre o ambiente *Windows*, utilizando o programa *Windows Terminal* para comunicar com a placa VPORT-25k. Através deste programa com os parâmetros de comunicação devidamente configurados para a porta serial conectada, acessasse o programa monitor da placa onde é executado o comando '*g*' (*go*), disparando a execução do eFORTH, que assume as funções de sistema operacional da placa VPORT-25k. A partir deste momento todas as operações realizadas na placa através do emulador de terminal do PC, são feitas sobre o eFORTH.

O bloco 2, contendo as definições FORTH para programas de controle difuso, é enviado à placa através da opção de transferência de arquivos texto do *Windows Terminal*. Ao final da transferência estas definições terão sido acrescentadas ao dicionário do eFORTH já instalado na

placa, podendo ser iniciada a etapa de escrita das declarações e do programa de controle, ou então enviar os arquivos previamente editados, contendo as declarações e o programa.

O bloco 3, formado pelo arquivo 'DECLARA.TXT', contém a declaração das variáveis linguísticas e dos conjuntos difusos, e o bloco 4, o arquivo 'TOOLS.TXT', contém o programa de controle. Estes blocos também são transferidos à placa VPORT-25k pela opção de transferência de arquivos texto do *Windows Terminal.* O arquivo 'TOOLS.TXT' também contém definições FORTH para verificação das estruturas de dados das variáveis linguísticas e conjuntos difusos. A partir deste momento o programa de controle é disparado, solicitando que os valores de entrada sejam digitados pelo teclado. Após digitados, os resultados intermediários assim como o valor final, são mostrados na tela do terminal. Este procedimento é repetido indefinidamente. O dados mostrados na tela do terminal são então armazenados em arquivo, formando o bloco 5 da figura iv-d.

IV.6 O Programa de Controle

O programa de controle faz uso das definições FORTH criadas para a utilização em programas de controle difuso. O anexo B contém a listagem destas definições.

Para a aplicação exemplo, DISTÂNCIA e ÂNGULO são declaradas como variáveis de entrada, e VELOCIDADE como variável de saída. Seguindo a declaração de cada variável, declaram-se os seus conjuntos difusos. Um procedimento de inicialização dos conjuntos difusos da variável de saída é exigido neste algoritmo, de modo a evitar que valores resultantes de uma avaliação das regras, interfiram na avaliação seguinte, ou seja, para evitar que os valores intermediários de uma execução do *loop* de controle, interfira na execução seguinte. Prosseguindo, são declarados os conjuntos e as variáveis de entrada que serão fuzzificadas, e os conjuntos e as variáveis de saída que serão defuzzyficados. Finalizando, são declaradas as regras. Algumas destas declarações são feitas na forma de definições de palavras FORTH, ou seja, a declaração é uma nova palavra FORTH que será incluída no dicionário.

IV.6.1 Entrada de Dados

Em sistemas com microcontroladores, dados são lidos e escritos em portas de entrada e saída, em dispositivos internos ou externos ao *chip* microcontrolador, ou de/para memória. Os sensores e atuadores do sistema controlado são conectados a estas portas. Como o sistema será simulado, sem a conexão dos sensores e atuadores nas portas, a entrada e saída de dados é feita através do emulador de terminal do *Windows*. O programa de controle solicita ao terminal que sejam enviados os valores para as variáveis DISTANCIA e ANGULO. Digitados estes valores no teclado, o programa segue a execução, processando e apresentando resultados no terminal.

As palavras do quadro iv-b, definidas no Anexo C, foram incluídas no *loop* de controle.

```
INICIALIZA
>INPUT
```

Quadro IV-B Declaração das palavras de inicialização e entrada de dados.

'INICIALIZA' faz a inicialização dos conjuntos difusos da variável de saída, zerando os valores de grau de pertinência. '>INPUT' (pronuncia-se "*to input*") solicita ao terminal que seja digitado os valores para DISTANCIA e ANGULO.

IV.6.2 Variáveis de Entrada: Universo de Discurso e Conjuntos Difusos

A declaração das variáveis de entrada DISTANCIA e ANGULO é mostrada no quadro iv-c.

```
FVARIN DISTANCIA 0 80 256 UNIVERS             \ 0 a 80 m
DISTANCIA MEMBER PERTO 0 5 10 TRIANGLE
DISTANCIA MEMBER MEDIA 5 15 40 TRIANGLE
DISTANCIA MEMBER LONGE 10 40 +SLOPE

FVARIN ANGULO 0 64 256 UNIVERS                \ 0 a 64 graus
ANGULO MEMBER PEQUENO 0 10 50 TRIANGLE
ANGULO MEMBER GRANDE  5 60 +SLOPE
```

Quadro IV-C Declaração das variáveis linguísticas de entrada.

Para ambas é declarado um universo de discurso com extensão de 256 bytes (ou valores). DISTANCIA tem seu universo de discurso de 0 a 80 (metros), e 3 conjuntos difusos com suas respectivas funções de pertinência conforme mostrado na figura iv-b. O universo de discurso de ANGULO vai de 0 a 64 (graus), com dois conjuntos difusos.

Estas declarações não estão na forma de definições de novas palavras FORTH. As palavras FVARIN e MEMBER são "palavras de definição", criando, respectivamente, as variáveis linguísticas de entrada e os conjuntos difusos, no dicionário FORTH. Em FORTH, palavras de definição são "palavras que criam palavras".

IV.6.3 Variável de Saída: Universo de Discurso e Conjuntos Difusos

VELOCIDADE é declarada como variável de saída no quadro iv-d.

```
FVAROUT VELOCIDADE
5  SINGLETON VEL-BAIXA
25 SINGLETON VEL-MEDIA
50 SINGLETON VEL-ALTA
```

Quadro IV-D Declaração da variável linguística de saída.

Devido a utilização de *singletons*, não é definido um universo de discurso para 'VELOCIDADE'. Três *singletons* são declarados com valores 5, 25, e 50. 'FVAROUT' e 'SINGLETON' são palavras de definição.

IV.6.4 Fuzzyficação

A declaração da fuzzyficação é feita na forma de definição de palavra FORTH, como mostra o quadro iv-e.

```
: FUZZYFICA
     MEMBERS[ PERTO MEDIA LONGE ]> DISTANCIA FUZZYFI
     MEMBERS[ PEQUENO GRANDE ]> ANGULO FUZZYFI ;
```

Quadro IV-E Declaração da fuzzyficação.

Na palavra 'FUZZIFICA' declaram-se os nomes dos conjuntos difusos e das variáveis que serão fuzzyficadas, seguindos da palavra FUZZYFI que dispara o processo de fuzzyficação.

IV.6.5 Base de Regras

Baseando-se na matriz linguística da figura iv-c, é declarada a base de regras mostrada no quadro iv-f, onde cada regra é definida como uma palavra FORTH.

```
: REGRA1 IFF PERTO ANDF PEQUENO THENF VEL-MEDIA END-RULE ;
: REGRA2 IFF MEDIA ANDF PEQUENO THENF VEL-MEDIA END-RULE ;
: REGRA3 IFF LONGE ANDF PEQUENO THENF VEL-ALTA END-RULE ;
: REGRA4 IFF PERTO ANDF GRANDE THENF VEL-BAIXA END-RULE ;
: REGRA5 IFF MEDIA ANDF GRANDE THENF VEL-BAIXA END-RULE ;
: REGRA6 IFF LONGE ANDF GRANDE THENF VEL-MEDIA END-RULE ;
```

Quadro IV-F Base de regras.

IV.6.6 Defuzzyficação

A declaração da defuzzyficação é feita na forma de definição de palavra FORTH, como mostrado no quadro iv-g.

```
: DEFUZZYFICA
     VELOCIDADE MEMBERS[ VEL-BAIXA VEL-MEDIA VEL-ALTA ]> DEFUZZYFI ;
```

Quadro IV-G Declaração da defuzzyficação.

Na palavra 'DEFUZZYFICA' declara-se a variável linguística e os conjuntos difusos que entram na defuzzyficação. A palavra DEFUZZYFI dispara o processo de defuzzyficação.

Após a defuzzyficação, resta mostrar o valor *crisp* resultante do processo.

IV.6.7 Saída de Dados

A saída de dados, tanto dados intermediários como o valor final, é enviada ao emulador de terminal no PC. Três palavras, definidas no anexo C, são utilizadas para a saída de dados. No quadro iv-h estão seus nomes.

```
OUT
DIST
ANG
```

Quadro IV-H Palavras de saída de dados.

A palavra 'OUT' mostra no terminal os valores de grau de pertinência atribuídos aos conjuntos difusos da variável de saída. As palavras 'DIST' e 'ANG' mostram valores intermediários do programa, respectivamente os valores de entrada fuzzyficados para DISTANCIA e ANGULO.

A união destas etapas constituem o *loop* de controle.

IV.6.8 *Loop* de Controle

O *loop* de controle, no quadro iv-i, engloba as etapas relacionadas, mais uma estrutura de controle para a repetição contínua.

```
1   : CONTROLE
2   BEGIN
3     INICIALIZA
4     >INPUT
5     FUZZYFICA
6     DIST ANG
7     REGRA1 REGRA2 REGRA3 REGRA4 REGRA5 REGRA6
8     DEFUZZYFICA
9     OUT
10  AGAIN ;
```

Quadro IV-I Declaração do Loop de Controle.

Na linha 1 do quadro iv-i inicia a definição do *loop*. Definido como uma palavra FORTH chamada CONTROLE, BEGIN na linha 2, e AGAIN na linha 10 formam a estrutura de controle para a repetição em *loop*.

Após transferir o eFORTH para placa VPORT-25k, executá-lo, e transferir também as definições FORTH para programas de controle difuso no dicionário, as declarações e o programa de controle, digita-se a palavra 'CONTROLE' no emulador de terminal para o *loop* ser disparado. Como a entrada de dados é simulada (que deveria ser via portas de entrada e saída de 8 bits) através do emulador de terminal, os valores digitados deverão estar entre 0 e 255.

Disparado o *loop* de controle, o programa espera que sejam digitados no terminal, os valores para as variáveis de entrada ANGULO e DISTANCIA. A seguir, apresenta o nome das variáveis e os valores digitados, e também os graus de pertinência resultantes da fuzzyficação, para cada um dos seus conjuntos difusos. Antes de repetir este processo novamente, apresenta os valores de grau de pertinência atribuídos nas regras aos conjuntos difusos da variável linguística de saída, e o valor *crisp* resultante da defuzzyficação. Após isto, volta a solicitar os valores para as variáveis linguísticas de entrada, repetindo estes passos continuamente.

Supondo uma situação inicial onde o guindaste está parado, e é desejado movê-lo 40 metros (ou outra unidade qualquer). A cada repetição do *loop*, quando forem solicitados os valores de entrada, é digitado um valor para o ângulo e um valor para a distância do ponto atual ao ponto de parada, considerando que o guindaste esteja se movendo. Os valores escolhidos foram baseados apenas na intuição, e são apresentados no quadro iv-j.

ANGULO	0	20	30	25	20	15	15	10	8	5	5	2
DISTANCIA	40	35	30	25	20	15	10	5	4	3	1	0

Quadro IV-J Valores de entrada para simulação.

IV.6.9 Resultados Obtidos

Os resultado apresentados foram manualmente verificados, confirmando, com precisão, a correta execução do algoritmo implementado. Os resultados intermediários da fuzzyficação, avaliação de regras, e defuzzyficação produzidos pelo programa, foram checados, conferindo com os valores esperados.

A seguir apresentam-se os resultado de uma simulação obtidos de um arquivo de saída produzido no terminal. As linhas pontilhadas separam as etapas de repetição do *loop*.

```
------------------------------------------------

ANGULO:      0

DISTANCIA:   40


in: DISTANCIA....  40

PERTO...  0     MEDIA...  150    LONGE...  16

in: ANGULO.......  0

PEQUENO. :  0   GRANDE.. :  0

VEL-BAIXA...  0 VEL-MEDIA...  0 VEL-ALTA....  16

out: VELOCIDADE..  50

------------------------------------------------

ANGULO:      20
```

```
DISTANCIA:  35


in: DISTANCIA....  35

PERTO...  0      MEDIA...  118    LONGE...  6

in: ANGULO.......  20

PEQUENO. :  100 GRANDE.. :  0

VEL-BAIXA...  0 VEL-MEDIA...  0 VEL-ALTA....  100

out: VELOCIDADE..  50

------------------------------------------------

ANGULO:     30

DISTANCIA:  30


in: DISTANCIA....  30

PERTO...  25    MEDIA...  87    LONGE...  0

in: ANGULO.......  30

PEQUENO. :  150 GRANDE.. :  9

VEL-BAIXA...  25        VEL-MEDIA...  9 VEL-ALTA....  87

out: VELOCIDADE..  39

------------------------------------------------

ANGULO:     25

DISTANCIA:  25


in: DISTANCIA....  25

PERTO...  87    MEDIA...  56    LONGE...  0

in: ANGULO.......  25

PEQUENO. :  125 GRANDE.. :  4

VEL-BAIXA...  87        VEL-MEDIA...  4 VEL-ALTA....  56
```

```
out: VELOCIDADE..  23
------------------------------------------------
ANGULO:      20
DISTANCIA:  20

in: DISTANCIA....  20
PERTO...  150   MEDIA...  25    LONGE...  0
in: ANGULO.......  20
PEQUENO. :  100 GRANDE.. :  0
VEL-BAIXA...  150       VEL-MEDIA...  0 VEL-ALTA....  25
out: VELOCIDADE..  12
------------------------------------------------
ANGULO:      15
DISTANCIA:  15

in: DISTANCIA....  15
PERTO...  187   MEDIA...  0     LONGE...  0
in: ANGULO.......  15
PEQUENO. :  75  GRANDE.. :  0
VEL-BAIXA...  187       VEL-MEDIA...  0 VEL-ALTA....  0
out: VELOCIDADE..  5

ANGULO:      15
DISTANCIA:  10

in: DISTANCIA....  10
PERTO...  125   MEDIA...  0     LONGE...  0
```

```
in: ANGULO.......  15
PEQUENO. :  75  GRANDE.. :  0
VEL-BAIXA...  125       VEL-MEDIA...  0 VEL-ALTA....  0
out: VELOCIDADE..  5
------------------------------------------------
ANGULO:      10
DISTANCIA:  5


in: DISTANCIA....  5
PERTO...  62    MEDIA...  0     LONGE...  0
in: ANGULO.......  10
PEQUENO. :  50  GRANDE.. :  0
VEL-BAIXA...  62        VEL-MEDIA...  0 VEL-ALTA....  0
out: VELOCIDADE..  5
------------------------------------------------
ANGULO:      8
DISTANCIA:  4


in: DISTANCIA....  4
PERTO...  50    MEDIA...  0     LONGE...  0
in: ANGULO.......  8
PEQUENO. :  40  GRANDE.. :  0
VEL-BAIXA...  50        VEL-MEDIA...  0 VEL-ALTA....  0
out: VELOCIDADE..  5
------------------------------------------------
ANGULO:      5
DISTANCIA:  3
```

```
in: DISTANCIA....  3
PERTO...  37    MEDIA...  0     LONGE...  0
in: ANGULO.......  5
PEQUENO. :  25  GRANDE.. :  0
VEL-BAIXA...  37        VEL-MEDIA...  0 VEL-ALTA....  0
out: VELOCIDADE..  5
------------------------------------------------
ANGULO:      5
DISTANCIA:  1


in: DISTANCIA....  1
PERTO...  12    MEDIA...  0     LONGE...  0
in: ANGULO.......  5
PEQUENO. :  25  GRANDE.. :  0
VEL-BAIXA...  12        VEL-MEDIA...  0 VEL-ALTA....  0
out: VELOCIDADE..  5
------------------------------------------------
ANGULO:      2
DISTANCIA:  0
in: DISTANCIA....  0
PERTO...  0     MEDIA...  0     LONGE...  0
in: ANGULO.......  2
PEQUENO. :  10  GRANDE.. :  0
VEL-BAIXA...  0 VEL-MEDIA...  0 VEL-ALTA....  0
out: VELOCIDADE..  0
```

Quadro IV-K Valores de saída do sistema.

V. Conclusões e Recomendações

Embora os resultados obtidos não possibilitem uma avaliação da modelagem e o desempenho do sistema, eles validam o algoritmo implementado em termos de execução lógica e exatidão dos resultados.

A utilização do ambiente de desenvolvimento e das palavras FORTH criadas para escrever o programa de controle, permitiram que testes fossem realizados diretamente no sistema dedicado de modo rápido e prático, e em poucas linhas de código.

As palavras criadas formam uma "linguagem" para que programas de controle difuso sejam escritos. Não tendo que preocupar-se com a implementação do algoritmo em alguma linguagem, o projetista pode dedicar mais tempo na busca da melhor solução.

Outros aspectos a destacar são a portabilidade do algoritmo implementado por estar escrito em FORTH, facilidade e rapidez na edição e teste de alterações no programa de controle (base de dados e regras), como de palavras auxiliares para verificação do funcionamento do sistema (*software* e *hardware*).

Utilizando as linhas de comunicação serial da placa, pode-se conectar o sistema de controle em rede de microcontroladores, ou a computadores com maior poder computacional, com algoritmos de redes neurais ou genéticos para tornar o controle adaptativo, expandindo a capacidade do sistema.

V.1 Proposta para futuros trabalhos

Como proposta para futuros trabalhos, o eFORTH pode ser modificado para se tornar residente em EPROM na placa VPORT-25k, com as palavras especiais para programas de controle difuso em seu dicionário, em substituição ao programa monitor hoje utilizado. Com isto

obteremos um sistema dedicado completo para o desenvolvimento de sistemas de controle de pequeno porte, empregando controle difuso. E de forma a ilustrar o funcionamento do sistema, aplicar a um protótipo mecânico de um guindaste, para o controle da velocidade.

Anexo - A

```
TITLE 8086 eFORTH.ASM
;===============================================================
;
;       eForth 1.0 by Bill Muench and C. H. Ting, 1990
;       Much of the code is derived from the following sources:
;               8086 figForth by Thomas Newman, 1981 and Joe smith, 1983
;               aFORTH by John Rible
;               bFORTH by Bill Muench
;
;       The goal of this implementation is to provide a simple eForth Model
;       which can be ported easily to many 8, 16, 24 and 32 bit CPU's.
;       The following attributes make it suitable for CPU's of the '90:
;
;               small machine dependent kernel and portable high level code
;               source code in the MASM format
;               direct threaded code
;               separated code and name dictionaries
;               simple vectored terminal and file interface to host computer
;               aligned with the proposed ANS Forth Standard
;               easy upgrade path to optimize for specific CPU
;
;       You are invited to implement this Model on your favorite CPU and
;       contribute it to the eForth Library for public use. You may use
;       a portable implementation to advertise more sophisticated and
;       optimized version for commercial purposes. However, you are
;       expected to implement the Model faithfully. The eForth Working
;       Group reserves the right to reject implementation which deviates
;       significantly from this Model.
;
;       As the ANS Forth Standard is still evolving, this Model will
;       change accordingly. Implementations must state clearly the
;       version number of the Model being tracked.
;
;       Representing the eForth Working Group in the Silicon Valley FIG Chapter.
;       Send contributions to:
;
;               Dr. C. H. Ting
;               156 14th Avenue
;               San Mateo, CA 94402
;               (415) 571-7639
;
;===============================================================

;; 0 = falso (F) , ffff = verdadeiro (T)
;
;; Tamanho atual: 48k

PAGE 62,132
```

```
;; Version control

VER             EQU    01H              ;major release version
EXT             EQU    01H              ;minor extension

;; Constants

COMPO                  EQU    040H      ;lexicon compile only bit
IMEDD           EQU    080H             ;lexicon immediate bit
MASKK                  EQU    07F1FH    ;lexicon bit mask

CELLL           EQU    2                ;size of a cell
BASEE           EQU 10                  ;default radix
VOCSS           EQU    8                ;depth of vocabulary stack

BKSPP           EQU    8                ;backspace
LF                     EQU    10        ;line feed
CRR                    EQU    13        ;carriage return
ERR                    EQU    27        ;error escape
TIC                    EQU    39        ;tick

CALLL           EQU    0E890H ;NOP CALL opcodes

;; Memory allocation      0//code>--//--<name//up>--<sp//tib>--rp//em

EM              EQU 0C000H              ;48k - topo de memória
COLDD           EQU    00100H           ;cold start vector

US              EQU    64*CELLL         ;user area size in cells
RTS             EQU    64*CELLL         ;return stack/TIB size

RPP             EQU    EM-8*CELLL       ;start of return stack (RP0)
TIBB            EQU    RPP-RTS          ;terminal input buffer (TIB)
SPP             EQU    TIBB-8*CELLL     ;start of data stack (SP0)
UPP             EQU    EM-256*CELLL     ;start of user area (UP0)
NAMEE           EQU    UPP-8*CELLL      ;name dictionary
CODEE EQU       COLDD+US                ;code dictionary

;; Initialize assembly variables

_LINK  = 0                              ;force a null link
_NAME= NAMEE                            ;initialize name pointer
_CODE = CODEE                           ;initialize code pointer
_USER  = 4*CELLL                        ;first user variable offset

;; Define assembly macros

;       Adjust an address to the next cell boundary.

$ALIGN          MACRO
        EVEN                            ;;for 16bit systems
        ENDM

;       Compile a code definition header.
```

```
$CODE MACRO        LEX,NAME,LABEL
      $ALIGN                              ;;force to cell boundary
LABEL:                                    ;;assembly label
      _CODE = $                           ;;save code pointer
      _LEN  = (LEX AND 01FH)/CELLL        ;;string cell count, round
                   ;;down
      _NAME= _NAME-((_LEN+3)*CELLL)       ;;new header on cell
                                          ;;boundary
ORG   _NAME                               ;;set name pointer
      DW    _CODE,_LINK                   ;;token pointer and link
      _LINK  = $                          ;;link points to a name string
      DB    LEX,NAME                      ;;name string
ORG   _CODE                               ;;restore code pointer
      ENDM

;     Compile a colon definition header.

$COLON      MACRO       LEX,NAME,LABEL
      $CODE LEX,NAME,LABEL
      NOP                                 ;;align to cell boundary
      CALL  DOLST                         ;;include CALL doLIST
      ENDM

;     Compile a user variable header.

$USER MACRO        LEX,NAME,LABEL
      $CODE LEX,NAME,LABEL
      NOP                                 ;;align to cell boundary
      CALL  DOLST                         ;;include CALL doLIST
      DW    DOUSE,_USER                   ;;followed by doUSER and offset
      _USER = _USER+CELLL                 ;;update user area offset
      ENDM

;     Compile an inline string.

D$    MACRO        FUNCT,STRNG
      DW    FUNCT                         ;;function
      _LEN  = $                           ;;save address of count byte
      DB    0,STRNG                       ;;count byte and string
      _CODE = $                           ;;save code pointer
ORG   _LEN                                ;;point to count byte
      DB     _CODE-_LEN-1                 ;;set count
ORG   _CODE                               ;;restore code pointer
      $ALIGN
      ENDM

;     Assemble inline direct threaded code ending.

$NEXT MACRO
      LODSW                               ;;next code address into AX
      JMP   AX                            ;;jump directly to code address
      ENDM
```

```
;; Main entry points and COLD start data

MAIN  SEGMENT
ASSUME      CS:MAIN,DS:MAIN,ES:MAIN,SS:MAIN

ORG   COLDD                             ;beginning of cold boot

ORIG:       MOV   AX,CS
            MOV   DS,AX                 ;DS is same as CS
            CLI                         ;disable interrupts, old 808x
                  ;CPU bug
            MOV   SS,AX                 ;SS is same as CS
            MOV   SP,SPP                ;initialize SP
            STI                         ;enable interrupts
            MOV   BP,RPP                ;initialize RP
            CLD                         ;direction flag, increment
            JMP   COLD                  ;to high level cold start

CTRLC:      IRET                        ;control C interrupt routine

; COLD start moves the following to USER variables.
; MUST BE IN SAME ORDER AS USER VARIABLES.

$ALIGN                                  ;align to cell boundary

UZERO:            DW    4 DUP (0)             ;reserved
            DW    SPP                   ;SP0
            DW    RPP                   ;RP0
            DW    QRX                   ;'?KEY
            DW    TXSTO                 ;'EMIT
            DW    ACCEP                 ;'EXPECT
            DW    KTAP                  ;'TAP
            DW    TXSTO                 ;'ECHO
            DW    DOTOK                 ;'PROMPT
            DW    BASEE                 ;BASE
            DW    0                     ;tmp
            DW    0                     ;SPAN
            DW    0                     ;>IN
            DW    0                     ;#TIB
            DW    TIBB                  ;TIB
            DW    0                     ;CSP
            DW    INTER                 ;'EVAL
            DW    NUMBQ                 ;'NUMBER
            DW    0                     ;HLD
            DW    0                     ;HANDLER
            DW    0                     ;CONTEXT pointer
            DW    VOCSS DUP (0)         ;vocabulary stack
            DW    0                     ;CURRENT pointer
            DW    0                     ;vocabulary link pointer
            DW    CTOP                  ;CP
            DW    NTOP                  ;NP
            DW    LASTN                 ;LAST
ULAST:
```

```
ORG     CODEE                                   ;start code dictionary

;; Device dependent I/O

;  BYE           ( -- )
;                Exit eForth.

                 $CODE 3,'BYE',BYE
   INT 044H

;  ?RX           ( -- c T | F )
;                Return input character and true, or a false if no input.

                 $CODE 3,'?RX',QRX
        mov   ah,01
        int   40h
        or    al,al
        jz    L0198

        mov   ah,00
        int   40h
        xor   ah,ah
        push  ax
        mov   bx,0ffffh          ; verdadeiro
        push  bx
        lodsw
        jmp   ax

L0198:  xor   bx,bx              ; 0 = false
        push  bx
        lodsw
        jmp   ax
                 $NEXT

;  TX!           ( c -- )
;                Send character c to the output device.

                 $CODE 3,'TX!',TXSTO
                 POP    AX
                 XOR    AH,AH
                 INT    41H
                 $NEXT

;  !IO           ( -- )
;                Initialize the serial I/O devices.

                 $CODE 3,'!IO',STOIO
                 $NEXT

;; Definicoes para depuracao ***************

;  @idb      ( -- u )
;          joga o valor do SFR IDB na pilha
```

```
        $CODE  4,'@idb',ATIDB
        PUSH   DS
        MOV    AX,0FF00h
        MOV    DS,AX
        XOR    AX,AX
        MOV    AL,BYTE PTR DS:[0FFFh]
        POP    DS
        PUSH   AX                           ; ( -- u )
        $NEXT

;    TX!!  ( c -- )
;       TX direto via TXB0

        $CODE  4,'TX!!',TXDIRETO
        MOV    AX,0FF00H                    ; area dos SFRs
        MOV    ES,AX
        MOV    BX,0F62h                     ; = TxB0
        POP    AX
        MOV    BYTE PTR ES:[BX],AL                  ; escreve o byte no TXB0
EsperaSair:                                 ; checa bit STF0
        TEST   BYTE PTR ES:[0F6Eh],80h
        JZ     EsperaSair                           ; se saiu = '1'
        MOV    BYTE PTR ES:[0F6Eh],47h
        $NEXT

;;*******************************************
;; The kernel

;  doLIT        ( -- w )
;               Push an inline literal.

                $CODE COMPO+5,'doLIT',DOLIT
                LODSW
                PUSH   AX
                $NEXT

;  doLIST       ( a -- )
;               Process colon list.

                $CODE COMPO+6,'doLIST',DOLST
                XCHG  BP,SP                 ;exchange pointers
                PUSH  SI                    ;push return stack
                XCHG  BP,SP                 ;restore the pointers
                POP   SI                    ;new list address
                $NEXT

;  EXIT ( -- )
;               Terminate a colon definition.

                $CODE 4,'EXIT',EXIT
                XCHG  BP,SP                 ;exchange pointers
                POP   SI                    ;pop return stack
                XCHG  BP,SP                 ;restore the pointers
                $NEXT
```

```
;  EXECUTE     ( ca -- )
;              Execute the word at ca.

               $CODE 7,'EXECUTE',EXECU
               POP    BX
               JMP    BX                        ;jump to the code address

;  next  ( -- )
;              Run time code for the single index loop.
;              : next ( -- ) \ hilevel model
;                r> r> dup if 1 - >r @ >r exit then drop cell+ >r ;

               $CODE COMPO+4,'next',DONXT
               SUB    WORD PTR [BP],1           ;decrement the index
               JC     NEXT1                     ;?decrement below 0
               MOV    SI,0[SI]                  ;no, continue loop
               $NEXT
NEXT1:         ADD    BP,CELLL                  ;yes, pop the index
               ADD    SI,CELLL                  ;exit loop
               $NEXT

;  ?branch     ( f -- )
;              Branch if flag is zero.

               $CODE COMPO+7,'?branch',QBRAN
               POP    BX                        ;pop flag
               OR     BX,BX                     ;?flag=0
               JZ     BRAN1                     ;yes, so branch
               ADD    SI,CELLL                  ;point IP to next cell
               $NEXT
BRAN1:                MOV    SI,0[SI]                  ;IP:=(IP)
               $NEXT

;  branch      ( -- )
;              Branch to an inline address.

               $CODE COMPO+6,'branch',BRAN
               MOV    SI,0[SI]                  ;IP:=(IP)
               $NEXT

;  !           ( w a -- )
;              Pop the data stack to memory.

               $CODE 1,'!',STORE
               POP    BX
               POP    0[BX]
               $NEXT

;  @           ( a -- w )
;              Push memory location to the data stack.

               $CODE 1,'@',AT
               POP    BX
```

```
                PUSH    0[BX]
                $NEXT

;  C!           ( c b -- )
;               Pop the data stack to byte memory.

                $CODE 2,'C!',CSTOR
                POP     BX
                POP     AX
                MOV     0[BX],AL
                $NEXT

;  C@           ( b -- c )
;               Push byte memory location to the data stack.

                $CODE 2,'C@',CAT
                POP     BX
                XOR     AX,AX                   ;AX=0 zero the hi byte
                MOV     AL,0[BX]
                PUSH    AX
                $NEXT

; Acesso as portas do V25

;  P0@      ( -- c )
;         Push byte from port 0 to the data stack

         $CODE   3,'P0@',P0AT
         MOV     AX,0FF00h
         MOV     ES,AX
         XOR     AX,AX                          ; AX=0 zero the hi byte
         MOV     AL,BYTE PTR ES:[0F44h]         ; P0 = 0F00h
         PUSH    AX
         $NEXT

;  P1@      ( -- c )
;         Push byte from port 1 to the data stack

         $CODE   3,'P1@',P1AT
         MOV     AX,0FF00h
         MOV     ES,AX
         XOR     AX,AX                          ; AX=0 zero the hi byte
         MOV     AL,BYTE PTR ES:[0F45h]         ; P1 = 0F08h
         PUSH    AX
         $NEXT

;  P2@      ( -- c )
;         Push byte from port 2 to the data stack

         $CODE   3,'P2@',P2AT
         MOV     AX,0FF00h
         MOV     ES,AX
         XOR     AX,AX                          ; AX=0 zero the hi byte
         MOV     AL,BYTE PTR ES:[0F46h]         ; P2 = 0F10h (EMS2 = 0F46h)
```

```
        PUSH   AX
        $NEXT

; !P2           ( c -- )
;               Push byte from then put it on port 2

        $CODE  3,'!P2',STOREP2
        MOV    AX,0FF00h
        MOV    ES,AX
        POP    AX
        XOR    AH,AH
        MOV    BYTE PTR ES:[0F46h],AL        ; P2 = 0F10h (EMS2 = 0F46h)
        $NEXT

; RP@           ( -- a )
;               Push the current RP to the data stack.

        $CODE 3,'RP@',RPAT
        PUSH   BP
        $NEXT

; RP!           ( a -- )
;               Set the return stack pointer.

                $CODE COMPO+3,'RP!',RPSTO
                POP     BP
                $NEXT

; R>            ( -- w )
;               Pop the return stack to the data stack.

                $CODE 2,'R>',RFROM
                PUSH    0[BP]
                ADD     BP,CELLL                ;adjust RP
                $NEXT

; R@            ( -- w )
;               Copy top of return stack to the data stack.

                $CODE 2,'R@',RAT
                PUSH    0[BP]
                $NEXT

; >R            ( w -- )
;               Push the data stack to the return stack.

                $CODE COMPO+2,'>R',TOR
                SUB     BP,CELLL                ;adjust RP
                POP     0[BP]                   ;push
                $NEXT

; SP@           ( -- a )
;               Push the current data stack pointer.
```

```
                $CODE 3,'SP@',SPAT
                MOV   BX,SP                 ;use BX to index the stack
                PUSH  BX
                $NEXT

;  SP!          ( a -- )
;               Set the data stack pointer.

                $CODE 3,'SP!',SPSTO
                POP   SP
                $NEXT

;  DROP         ( w -- )
;               Discard top stack item.

                $CODE 4,'DROP',DROP
                ADD   SP,CELLL              ;adjust SP
                $NEXT

;  DUP          ( w -- w w )
;               Duplicate the top stack item.

                $CODE 3,'DUP',DUPP
                MOV   BX,SP                 ;use BX to index the stack
                PUSH  0[BX]
                $NEXT

;  SWAP         ( w1 w2 -- w2 w1 )
;               Exchange top two stack items.

                $CODE 4,'SWAP',SWAP
                POP   BX
                POP   AX
                PUSH  BX
                PUSH  AX
                $NEXT

;  OVER         ( w1 w2 -- w1 w2 w1 )
;               Copy second stack item to top.

                $CODE 4,'OVER',OVER
                MOV   BX,SP                 ;use BX to index the stack
                PUSH  CELLL[BX]
                $NEXT

;  0<           ( n -- t )
;               Return true if n is negative.

                $CODE 2,'0<',ZLESS
                POP   AX
                CWD                         ;sign extend
                PUSH  DX
                $NEXT
```

```
;  AND        ( w w -- w )
;             Bitwise AND.

              $CODE 3,'AND',ANDD
              POP     BX
              POP     AX
              AND     BX,AX
              PUSH    BX
              $NEXT

;  OR         ( w w -- w )
;             Bitwise inclusive OR.

              $CODE 2,'OR',ORR
              POP     BX
              POP     AX
              OR      BX,AX
              PUSH    BX
              $NEXT

;  XOR        ( w w -- w )
;             Bitwise exclusive OR.

              $CODE 3,'XOR',XORR
              POP     BX
              POP     AX
              XOR     BX,AX
              PUSH    BX
              $NEXT

;  UM+        ( w w -- w cy )
;             Add two numbers, return the sum and carry flag.

              $CODE 3,'UM+',UPLUS
              XOR     CX,CX                  ;CX=0 initial carry flag
              POP     BX
              POP     AX
              ADD     AX,BX
              RCL     CX,1                   ;get carry
              PUSH    AX                     ;push sum
              PUSH    CX                     ;push carry
              $NEXT

;  ANDF       ( pfa -- GP | aborta )
;             Aborta palavra se topo da pilha = zero. Recupera pilha
;             para último valor salvo por !CSP.
;             Esta palavra tem que ser antecedida de um !CSP ou IFF (fuzzy)

              $CODE 4,'ANDF',ANDFUZZY
              pop     bx
              mov     ax,[bx]
              push    ax              ; emula @
              or      ah,al
              jz      Aborta
```

```
                $NEXT
Aborta:
                mov     bx,UPP          ; inicio user vars area; CSP offset=24
                add     bx,24h          ; ATENÇÃO: as user vars iniciais não
                                        ; podem ser deslocadas
                mov     ax,[bx]         ; lê stack salvo por !CSP
                mov     sp,ax           ; recupera pilha salva por !CSP
                jmp     EXIT            ; sai da palavra atual

;   THENF       (GP pfa -- MIN | aborta )
;               Esta palavra tem que ser antecedida de um !CSP ou IFF (fuzzy)

                $CODE 5,'THENF',THENFUZZY
                pop     bx
                mov     ax,[bx]
                push    ax              ; emula @
                or      ah,al
                jz      Abortar
; >>
                mov     bx,UPP          ; inicio user vars area; CSP offset=24
                add     bx,24h          ; ATENÇÃO: as user vars iniciais não
                                        ; podem ser deslocadas
                mov     ax,[bx]         ; lê stack salvo por !CSP
                mov     bx,sp           ; stack atual, com valor menor, a pilha desce.
                sub     ax,bx
                cmp     ax,02
                ja      CalcMIN         ; se pilha tem mais que uma word, faz o MIN
                $NEXT                   ; se tem apenas uma, ela é o MIN, e segue.
; >>
CalcMIN:
                jmp     MIN             ; usa a rot do MIN também para sair: $NEXT.
Abortar:                                ; reseta pilha e aborta palavra.
                mov     bx,UPP          ; inicio user vars area; CSP offset=24
                add     bx,24h          ; ATENÇÃO: as user vars iniciais não
                                        ; podem ser deslocadas
                mov     ax,[bx]         ; lê stack salvo por !CSP
                mov     sp,ax           ; recupera pilha salva por !CSP
                jmp     EXIT            ; sai da palavra atual

;   MIN  ( n1 n2 -- min ) Convertida de FORTH para o asm.
;
                $CODE 3,'MIN',MIN
                pop     ax
                pop     bx
                cmp     ax,bx
                jb      AXMenor
                push    bx
                jmp     FimMIN
AXMenor:
                push    ax
FimMIN:
                $NEXT

;   MAX( n1 n2 -- max ) Converti do forth para o asm.
```

```
;
                $CODE 3,'MAX',MAX
                pop     ax
                pop     bx
                cmp     ax,bx
                ja      AXMaior
                push    bx
                jmp     FimMAX
AXMaior:
                push    ax
FimMAX:
                $NEXT

;; System and user variables

;  doVAR        ( -- a )
;               Run time routine for VARIABLE and CREATE.

                $COLON          COMPO+5,'doVAR',DOVAR
                DW      RFROM,EXIT

;  UP           ( -- a )
;               Pointer to the user area.

                $COLON          2,'UP',UP
                DW      DOVAR
                DW      UPP

;  doUSER       ( -- a )
;               Run time routine for user variables.

                $COLON          COMPO+6,'doUSER',DOUSE
                DW      RFROM,AT,UP,AT,PLUS,EXIT

;  SP0          ( -- a )
;               Pointer to bottom of the data stack.

                $USER  3,'SP0',SZERO

;  RP0          ( -- a )
;               Pointer to bottom of the return stack.

                $USER  3,'RP0',RZERO

;  '?KEY        ( -- a )
;               Execution vector of ?KEY.

                $USER  5,"'?KEY",TQKEY

;  'EMIT        ( -- a )
;               Execution vector of EMIT.

                $USER  5,"'EMIT",TEMIT
```

```
;  'EXPECT     ( -- a )
;              Execution vector of EXPECT.

               $USER  7,"'EXPECT",TEXPE

;  'TAP ( -- a )
;              Execution vector of TAP.

               $USER  4,"'TAP",TTAP

;  'ECHO       ( -- a )
;              Execution vector of ECHO.

               $USER  5,"'ECHO",TECHO

;  'PROMPT     ( -- a )
;              Execution vector of PROMPT.

               $USER  7,"'PROMPT",TPROM

;  BASE        ( -- a )
;              Storage of the radix base for numeric I/O.

               $USER  4,'BASE',BASE

;  tmp         ( -- a )
;              A temporary storage location used in parse and find.

               $USER  COMPO+3,'tmp',TEMP

;  SPAN        ( -- a )
;              Hold character count received by EXPECT.

               $USER  4,'SPAN',SPAN

;  >IN         ( -- a )
;              Hold the character pointer while parsing input stream.

               $USER  3,'>IN',INN

;  #TIB ( -- a )
;              Hold the current count and address of the terminal input buffer.

               $USER  4,'#TIB',NTIB
               _USER = _USER+CELLL

;  CSP         ( -- a )
;              Hold the stack pointer for error checking.

               $USER  3,'CSP',CSP

;  'EVAL       ( -- a )
;              Execution vector of EVAL.
```

```
                $USER 5,"'EVAL'",TEVAL

; 'NUMBER       ( -- a )
;               Execution vector of NUMBER?.

                $USER 7,"'NUMBER'",TNUMB

; HLD           ( -- a )
;               Hold a pointer in building a numeric output string.

                $USER 3,'HLD',HLD

; HANDLER       ( -- a )
;               Hold the return stack pointer for error handling.

                $USER 7,'HANDLER',HANDL

; CONTEXT       ( -- a )
;               A area to specify vocabulary search order.

                $USER 7,'CONTEXT',CNTXT
                _USER = _USER+VOCSS*CELLL          ;vocabulary stack

; CURRENT       ( -- a )
;               Point to the vocabulary to be extended.

                $USER 7,'CURRENT',CRRNT
                _USER = _USER+CELLL                ;vocabulary link pointer

; CP            ( -- a )
;               Point to the top of the code dictionary.

                $USER 2,'CP',CP

; NP            ( -- a )
;               Point to the bottom of the name dictionary.

                $USER 2,'NP',NP

; LAST          ( -- a )
;               Point to the last name in the name dictionary.

                $USER 4,'LAST',LAST

; tmp1 ( -- a )
;               Variavel temporaria. Usado por DEFUZZYFI>.

        $USER  4,'tmp1',tmp1

; tmp2 ( -- a )
;               Variavel temporaria. Usado por DEFUZZYFI>.

        $USER  4,'tmp2',tmp2
```

```
;  PTO-A       ( -- a )
;              Ponto A nas definições de conjuntos difusos.

               $USER  5,'PTO-A',PTO_A

;  PTO-B       ( -- a )
;              Ponto B nas definições de conjuntos difusos.

               $USER  5,'PTO-B',PTO_B

;  PTO-C       ( -- a )
;              Ponto C nas definições de conjuntos difusos.

               $USER  5,'PTO-C',PTO_C

;  PTO-D       ( -- a )
;              Ponto D nas definições de conjuntos difusos.

               $USER  5,'PTO-D',PTO_D

;; Common functions

;  doVOC       ( -- )
;              Run time action of VOCABULARY's.

               $COLON        COMPO+5,'doVOC',DOVOC
               DW     RFROM,CNTXT,STORE,EXIT

;  FORTH       ( -- )
;              Make FORTH the context vocabulary.

               $COLON        5,'FORTH',FORTH
               DW     DOVOC
               DW     0                        ;vocabulary head pointer
               DW     0                        ;vocabulary link pointer

;  ?DUP( w -- w w | 0 )
;              Dup tos if its is not zero.

               $COLON        4,'?DUP',QDUP
               DW     DUPP
               DW     QBRAN,QDUP1
               DW     DUPP
QDUP1:         DW     EXIT

;  ROT         ( w1 w2 w3 -- w2 w3 w1 )
;              Rot 3rd item to top.

               $COLON        3,'ROT',ROT
               DW     TOR,SWAP,RFROM,SWAP,EXIT

;  2DROP       ( w w -- )
;              Discard two items on stack.
```

```
                $COLON      5,'2DROP',DDROP
                DW      DROP,DROP,EXIT

;   2DUP        ( w1 w2 -- w1 w2 w1 w2 )
;               Duplicate top two items.

                $COLON      4,'2DUP',DDUP
                DW      OVER,OVER,EXIT

;   +           ( w w -- sum )
;               Add top two items.

                $COLON      1,'+',PLUS
                DW      UPLUS,DROP,EXIT

;   D+          ( d d -- d )
;               Double addition, as an example using UM+.
;
;               $COLON      2,'D+',DPLUS
;               DW      TOR,SWAP,TOR,UPLUS
;               DW      RFROM,RFROM,PLUS,PLUS,EXIT

;   NOT         ( w -- w )
;               One's complement of tos.

                $COLON      3,'NOT',INVER
                DW      DOLIT,-1,XORR,EXIT

;   NEGATE      ( n -- -n )
;               Two's complement of tos.

                $COLON      6,'NEGATE',NEGAT
                DW      INVER,DOLIT,1,PLUS,EXIT

;   DNEGATE     ( d -- -d )
;               Two's complement of top double.

                $COLON      7,'DNEGATE',DNEGA
                DW      INVER,TOR,INVER
                DW      DOLIT,1,UPLUS
                DW      RFROM,PLUS,EXIT

;   -           ( n1 n2 -- n1-n2 )
;               Subtraction.

                $COLON      1,'-',SUBB
                DW      NEGAT,PLUS,EXIT

;   ABS         ( n -- n )
;               Return the absolute value of n.

                $COLON      3,'ABS',ABSS
                DW      DUPP,ZLESS
                DW      QBRAN,ABS1
```

```
                DW      NEGAT
ABS1:           DW      EXIT

;  =            ( w w -- t )
;               Return true if top two are equal.

                $COLON          1,'=',EQUAL
                DW      XORR
                DW      QBRAN,EQU1
                DW      DOLIT,0,EXIT            ;false flag
EQU1:           DW      DOLIT,-1,EXIT           ;true flag

;  U<           ( u u -- t )
;               Unsigned compare of top two items.

                $COLON          2,'U<',ULESS
                DW      DDUP,XORR,ZLESS
                DW      QBRAN,ULES1
                DW      SWAP,DROP,ZLESS,EXIT
ULES1:          DW      SUBB,ZLESS,EXIT

;  <            ( n1 n2 -- t )
;               Signed compare of top two items.

                $COLON          1,'<',LESS
                DW      DDUP,XORR,ZLESS
                DW      QBRAN,LESS1
                DW      DROP,ZLESS,EXIT
LESS1:          DW      SUBB,ZLESS,EXIT

;  WITHIN       ( u ul uh -- t )
;               Return true if u is within the range of ul and uh.

                $COLON          6,'WITHIN',WITHI
                DW      OVER,SUBB,TOR                   ;ul <= u < uh
                DW      SUBB,RFROM,ULESS,EXIT

;; Divide

;  UM/MOD       ( udl udh u -- ur uq )
;               Unsigned divide of a double by a single. Return mod and quotient.

                $COLON          6,'UM/MOD',UMMOD
                DW      DDUP,ULESS
                DW      QBRAN,UMM4
                DW      NEGAT,DOLIT,15,TOR
UMM1:           DW      TOR,DUPP,UPLUS
                DW      TOR,TOR,DUPP,UPLUS
                DW      RFROM,PLUS,DUPP
                DW      RFROM,RAT,SWAP,TOR
                DW      UPLUS,RFROM,ORR
                DW      QBRAN,UMM2
                DW      TOR,DROP,DOLIT,1,PLUS,RFROM
                DW      BRAN,UMM3
```

```
UMM2:       DW      DROP
UMM3:       DW      RFROM
            DW      DONXT,UMM1
            DW      DROP,SWAP,EXIT
UMM4:       DW      DROP,DDROP
            DW      DOLIT,-1,DUPP,EXIT    ;overflow, return max

;  M/MOD    ( d n -- r q )
;           Signed floored divide of double by single. Return mod and quotient.

            $COLON      5,'M/MOD',MSMOD
            DW      DUPP,ZLESS,DUPP,TOR
            DW      QBRAN,MMOD1
            DW      NEGAT,TOR,DNEGA,RFROM
MMOD1:              DW      TOR,DUPP,ZLESS
            DW      QBRAN,MMOD2
            DW      RAT,PLUS
MMOD2:              DW      RFROM,UMMOD,RFROM
            DW      QBRAN,MMOD3
            DW      SWAP,NEGAT,SWAP
MMOD3:              DW      EXIT

;  /MOD     ( n n -- r q )
;           Signed divide. Return mod and quotient.

            $COLON      4,'/MOD',SLMOD
            DW      OVER,ZLESS,SWAP,MSMOD,EXIT

;  MOD      ( n n -- r )
;           Signed divide. Return mod only.

            $COLON      3,'MOD',MODD
            DW      SLMOD,DROP,EXIT

;  /        ( n n -- q )
;           Signed divide. Return quotient only.

            $COLON      1,'/',SLASH
            DW      SLMOD,SWAP,DROP,EXIT

;; Multiply

;  UM*      ( u u -- ud )
;           Unsigned multiply. Return double product.

            $COLON      3,'UM*',UMSTA
            DW      DOLIT,0,SWAP,DOLIT,15,TOR
UMST1:              DW      DUPP,UPLUS,TOR,TOR
            DW      DUPP,UPLUS,RFROM,PLUS,RFROM
            DW      QBRAN,UMST2
            DW      TOR,OVER,UPLUS,RFROM,PLUS
UMST2:              DW      DONXT,UMST1
            DW      ROT,DROP,EXIT
```

```
;  *            ( n n -- n )
;               Signed multiply. Return single product.

                $COLON      1,'*',STAR
                DW      UMSTA,DROP,EXIT

;  M*           ( n n -- d )
;               Signed multiply. Return double product.

                $COLON      2,'M*',MSTAR
                DW      DDUP,XORR,ZLESS,TOR
                DW      ABSS,SWAP,ABSS,UMSTA
                DW      RFROM
                DW      QBRAN,MSTA1
                DW      DNEGA
MSTA1:                  DW      EXIT

;  */MOD        ( n1 n2 n3 -- r q )
;               Multiply n1 and n2, then divide by n3. Return mod and quotient.

                $COLON      5,'*/MOD',SSMOD
                DW      TOR,MSTAR,RFROM,MSMOD,EXIT

;  */           ( n1 n2 n3 -- q )
;               Multiply n1 by n2, then divide by n3. Return quotient only.

                $COLON      2,'*/',STASL
                DW      SSMOD,SWAP,DROP,EXIT

;; Miscellaneous

;  CELL+        ( a -- a )
;               Add cell size in byte to address.

                $COLON      5,'CELL+',CELLP
                DW      DOLIT,CELLL,PLUS,EXIT

;  CELL-        ( a -- a )
;               Subtract cell size in byte from address.

                $COLON      5,'CELL-',CELLM
                DW      DOLIT,0-CELLL,PLUS,EXIT

;  CELLS        ( n -- n )
;               Multiply tos by cell size in bytes.

                $COLON      5,'CELLS',CELLS
                DW      DOLIT,CELLL,STAR,EXIT

;  ALIGNED      ( b -- a )
;               Align address to the cell boundary.

                $COLON      7,'ALIGNED',ALGND
                DW      DUPP,DOLIT,0,DOLIT,CELLL
```

```
                DW      UMMOD,DROP,DUPP
                DW      QBRAN,ALGN1
                DW      DOLIT,CELLL,SWAP,SUBB
ALGN1:                  DW      PLUS,EXIT

;  BL           ( -- 32 )
;               Return 32, the blank character.

                $COLON          2,'BL',BLANK
                DW      DOLIT,' ',EXIT

;  >CHAR        ( c -- c )
;               Filter non-printing characters.

                $COLON          5,'>CHAR',TCHAR
                DW      DOLIT,07FH,ANDD,DUPP            ;mask msb
                DW      DOLIT,127,BLANK,WITHI           ;check for printable
                DW      QBRAN,TCHA1
                DW      DROP,DOLIT,'_'          ;replace non-printables
TCHA1:          DW      EXIT

;  DEPTH        ( -- n )
;               Return the depth of the data stack.

                $COLON          5,'DEPTH',DEPTH
                DW      SPAT,SZERO,AT,SWAP,SUBB
                DW      DOLIT,CELLL,SLASH,EXIT

;  PICK ( ... +n -- ... w )
;               Copy the nth stack item to tos.

                $COLON          4,'PICK',PICK
                DW      DOLIT,1,PLUS,CELLS
                DW      SPAT,PLUS,AT,EXIT

;; Memory access

;  +!           ( n a -- )
;               Add n to the contents at address a.

                $COLON          2,'+!',PSTOR
                DW      SWAP,OVER,AT,PLUS
                DW      SWAP,STORE,EXIT

;  2!           ( d a -- )
;               Store the double integer to address a.

                $COLON          2,'2!',DSTOR
                DW      SWAP,OVER,STORE
                DW      CELLP,STORE,EXIT

;  2@           ( a -- d )
;               Fetch double integer from address a.
```

```
                $COLON      2,'2@',DAT
                DW      DUPP,CELLP,AT
                DW      SWAP,AT,EXIT

;   COUNT       ( b -- b +n )
;               Return count byte of a string and add 1 to byte address.

                $COLON      5,'COUNT',COUNT
                DW      DUPP,DOLIT,1,PLUS
                DW      SWAP,CAT,EXIT

;   HERE        ( -- a )
;               Return the top of the code dictionary.

                $COLON      4,'HERE',HERE
                DW      CP,AT,EXIT

;   PAD         ( -- a )
;               Return the address of a temporary buffer.

                $COLON      3,'PAD',PAD
                DW      HERE,DOLIT,80,PLUS,EXIT

;   TIB         ( -- a )
;               Return the address of the terminal input buffer.

                $COLON      3,'TIB',TIB
                DW      NTIB,CELLP,AT,EXIT

;   @EXECUTE ( a -- )
;               Execute vector stored in address a.

                $COLON      8,'@EXECUTE',ATEXE
                DW      AT,QDUP                 ;?address or zero
                DW      QBRAN,EXE1
                DW      EXECU               ;execute if non-zero
EXE1:           DW      EXIT                ;do nothing if zero

;   CMOVE       ( b1 b2 u -- )
;               Copy u bytes from b1 to b2.

                $COLON      5,'CMOVE',CMOVE
                DW      TOR
                DW      BRAN,CMOV2
CMOV1:                  DW      TOR,DUPP,CAT
                DW      RAT,CSTOR
                DW      DOLIT,1,PLUS
                DW      RFROM,DOLIT,1,PLUS
CMOV2:                  DW      DONXT,CMOV1
                DW      DDROP,EXIT

;   FILL ( b u c -- )
;               Fill u bytes of character c to area beginning at b.
```

```
                $COLON      4,'FILL',FILL
                DW      SWAP,TOR,SWAP
                DW      BRAN,FILL2
FILL1:          DW      DDUP,CSTOR,DOLIT,1,PLUS
FILL2:          DW      DONXT,FILL1
                DW      DDROP,EXIT

;   -TRAILING   ( b u -- b u )
;               Adjust the count to eliminate trailing white space.

                $COLON      9,'-TRAILING',DTRAI
                DW      TOR
                DW      BRAN,DTRA2
DTRA1:          DW      BLANK,OVER,RAT,PLUS,CAT,LESS
                DW      QBRAN,DTRA2
                DW      RFROM,DOLIT,1,PLUS,EXIT     ;adjusted count
DTRA2:          DW      DONXT,DTRA1
                DW      DOLIT,0,EXIT            ;count=0

;   PACK$       ( b u a -- a )
;               Build a counted string with u characters from b. Null fill.

                $COLON      5,'PACK$',PACKS
                DW      ALGND,DUPP,TOR              ;strings only on cell boundary
                DW      OVER,DUPP,DOLIT,0
                DW      DOLIT,CELLL,UMMOD,DROP  ;count mod cell
                DW      SUBB,OVER,PLUS
                DW      DOLIT,0,SWAP,STORE  ;null fill cell
                DW      DDUP,CSTOR,DOLIT,1,PLUS     ;save count
                DW      SWAP,CMOVE,RFROM,EXIT       ;move string

;; Numeric output, single precision

;   DIGIT       ( u -- c )
;               Convert digit u to a character.

                $COLON      5,'DIGIT',DIGIT
                DW      DOLIT,9,OVER,LESS
                DW      DOLIT,7,ANDD,PLUS
                DW      DOLIT,'0',PLUS,EXIT

;   EXTRACT     ( n base -- n c )
;               Extract the least significant digit from n.

                $COLON      7,'EXTRACT',EXTRC
                DW      DOLIT,0,SWAP,UMMOD
                DW      SWAP,DIGIT,EXIT

;   <#          ( -- )
;               Initiate the numeric output process.

                $COLON      2,'<#',BDIGS
                DW      PAD,HLD,STORE,EXIT
```

```
;  HOLD         ( c -- )
;               Insert a character into the numeric output string.

                $COLON      4,'HOLD',HOLD
                DW    HLD,AT,DOLIT,1,SUBB
                DW    DUPP,HLD,STORE,CSTOR,EXIT

;  #            ( u -- u )
;               Extract one digit from u and append the digit to output string.

                $COLON      1,'#',DIG
                DW    BASE,AT,EXTRC,HOLD,EXIT

;  #S           ( u -- 0 )
;               Convert u until all digits are added to the output string.

                $COLON      2,'#S',DIGS
DIGS1:          DW    DIG,DUPP
                DW    QBRAN,DIGS2
                DW    BRAN,DIGS1
DIGS2:          DW    EXIT

;  SIGN( n -- )
;               Add a minus sign to the numeric output string.

                $COLON      4,'SIGN',SIGN
                DW    ZLESS
                DW    QBRAN,SIGN1
                DW    DOLIT,'-',HOLD
SIGN1:          DW    EXIT

;  #>           ( w -- b u )
;               Prepare the output string to be TYPE'd.

                $COLON      2,'#>',EDIGS
                DW    DROP,HLD,AT
                DW    PAD,OVER,SUBB,EXIT

;  str          ( n -- b u )
;               Convert a signed integer to a numeric string.

                $COLON      3,'str',STR
                DW    DUPP,TOR,ABSS
                DW    BDIGS,DIGS,RFROM
                DW    SIGN,EDIGS,EXIT

;  HEX          ( -- )
;               Use radix 16 as base for numeric conversions.

                $COLON      3,'HEX',HEX
                DW    DOLIT,16,BASE,STORE,EXIT

;  DECIMAL      ( -- )
;               Use radix 10 as base for numeric conversions.
```

```
                $COLON      7,'DECIMAL',DECIM
                DW      DOLIT,10,BASE,STORE,EXIT

;; Numeric input, single precision

;   DIGIT?      ( c base -- u t )
;               Convert a character to its numeric value. A flag indicates success.

                $COLON      6,'DIGIT?',DIGTQ
                DW      TOR,DOLIT,'0',SUBB
                DW      DOLIT,9,OVER,LESS
                DW      QBRAN,DGTQ1
                DW      DOLIT,7,SUBB
                DW      DUPP,DOLIT,10,LESS,ORR
DGTQ1:                  DW      DUPP,RFROM,ULESS,EXIT

;   NUMBER?     ( a -- n T | a F )
;               Convert a number string to integer. Push a flag on tos.

                $COLON      7,'NUMBER?',NUMBQ
                DW      BASE,AT,TOR,DOLIT,0,OVER,COUNT
                DW      OVER,CAT,DOLIT,'$',EQUAL
                DW      QBRAN,NUMQ1
                DW      HEX,SWAP,DOLIT,1,PLUS
                DW      SWAP,DOLIT,1,SUBB
NUMQ1:                  DW      OVER,CAT,DOLIT,'-',EQUAL,TOR
                DW      SWAP,RAT,SUBB,SWAP,RAT,PLUS,QDUP
                DW      QBRAN,NUMQ6
                DW      DOLIT,1,SUBB,TOR
NUMQ2:                  DW      DUPP,TOR,CAT,BASE,AT,DIGTQ
                DW      QBRAN,NUMQ4
                DW      SWAP,BASE,AT,STAR,PLUS,RFROM
                DW      DOLIT,1,PLUS
                DW      DONXT,NUMQ2
                DW      RAT,SWAP,DROP
                DW      QBRAN,NUMQ3
                DW      NEGAT
NUMQ3:                  DW      SWAP
                DW      BRAN,NUMQ5
NUMQ4:                  DW      RFROM,RFROM,DDROP,DDROP,DOLIT,0
NUMQ5:                  DW      DUPP
NUMQ6:                  DW      RFROM,DDROP
                DW      RFROM,BASE,STORE,EXIT

;; Basic I/O

;   ?KEY        ( -- c T | F )
;               Return input character and true, or a false if no input.

                $COLON      4,'?KEY',QKEY
                DW      TQKEY,ATEXE,EXIT

;   KEY         ( -- c )
```

```
;               Wait for and return an input character.

                $COLON      3,'KEY',KEY
KEY1:           DW      QKEY
                DW      QBRAN,KEY1
                DW      EXIT

;   EMIT        ( c -- )
;               Send a character to the output device.

                $COLON      4,'EMIT',EMIT
                DW      TEMIT,ATEXE,EXIT

;   NUF?( -- t )
;               Return false if no input, else pause and if CR return true.

                $COLON      4,'NUF?',NUFQ
                DW      QKEY,DUPP
                DW      QBRAN,NUFQ1
                DW      DDROP,KEY,DOLIT,CRR,EQUAL
NUFQ1:          DW      EXIT

;   PACE        ( -- )
;               Send a pace character for the file downloading process.

                $COLON      4,'PACE',PACE
                DW      DOLIT,11,EMIT,EXIT

;   SPACE       ( -- )
;               Send the blank character to the output device.

                $COLON      5,'SPACE',SPACE
                DW      BLANK,EMIT,EXIT

;   SPACES      ( +n -- )
;               Send n spaces to the output device.

                $COLON      6,'SPACES',SPACS
                DW      DOLIT,0,MAX,TOR
                DW      BRAN,CHAR2
CHAR1:                  DW      SPACE
CHAR2:                  DW      DONXT,CHAR1
                DW      EXIT

;   TYPE        ( b u -- )
;               Output u characters from b.

                $COLON      4,'TYPE',TYPEE
                DW      TOR
                DW      BRAN,TYPE2
TYPE1:          DW      DUPP,CAT,EMIT
                DW      DOLIT,1,PLUS
TYPE2:          DW      DONXT,TYPE1
                DW      DROP,EXIT
```

```
;  CR          ( -- )
;              Output a carriage return and a line feed.

               $COLON      2,'CR',CR
               DW     DOLIT,CRR,EMIT
               DW     DOLIT,LF,EMIT,EXIT

;  do$         ( -- a )
;              Return the address of a compiled string.

               $COLON      COMPO+3,'do$',DOSTR
               DW     RFROM,RAT,RFROM,COUNT,PLUS
               DW     ALGND,TOR,SWAP,TOR,EXIT

;  $"|         ( -- a )
;              Run time routine compiled by $". Return address of a compiled string.

               $COLON      COMPO+3,'$"|',STRQP
               DW     DOSTR,EXIT              ;force a call to do$

;  ."|         ( -- )
;              Run time routine of ." . Output a compiled string.

               $COLON      COMPO+3,'."|',DOTQP
               DW     DOSTR,COUNT,TYPEE,EXIT

;  .R          ( n +n -- )
;              Display an integer in a field of n columns, right justified.

               $COLON      2,'.R',DOTR
               DW     TOR,STR,RFROM,OVER,SUBB
               DW     SPACS,TYPEE,EXIT

;  U.R         ( u +n -- )
;              Display an unsigned integer in n column, right justified.

               $COLON      3,'U.R',UDOTR
               DW     TOR,BDIGS,DIGS,EDIGS
               DW     RFROM,OVER,SUBB
               DW     SPACS,TYPEE,EXIT

;  U.          ( u -- )
;              Display an unsigned integer in free format.

               $COLON      2,'U.',UDOT
               DW     BDIGS,DIGS,EDIGS
               DW     SPACE,TYPEE,EXIT

;  .           ( w -- )
;              Display an integer in free format, preceeded by a space.

               $COLON      1,'.',DOT
               DW     BASE,AT,DOLIT,10,XORR        ;?decimal
```

```
                DW      QBRAN,DOT1
                DW      UDOT,EXIT                ;no, display unsigned
DOT1:           DW      STR,SPACE,TYPEE,EXIT          ;yes, display signed

;  ?            ( a -- )
;               Display the contents in a memory cell.

                $COLON          1,'?',QUEST
                DW      AT,DOT,EXIT

;; Parsing

;  parse ( b u c -- b u delta ; <string> )
;               Scan string delimited by c. Return found string and its offset.

                $COLON          5,'parse',PARS
                DW      TEMP,STORE,OVER,TOR,DUPP
                DW      QBRAN,PARS8
                DW      DOLIT,1,SUBB,TEMP,AT,BLANK,EQUAL
                DW      QBRAN,PARS3
                DW      TOR
PARS1:          DW      BLANK,OVER,CAT                ;skip leading blanks ONLY
                DW      SUBB,ZLESS,INVER
                DW      QBRAN,PARS2
                DW      DOLIT,1,PLUS
                DW      DONXT,PARS1
                DW      RFROM,DROP,DOLIT,0,DUPP,EXIT
PARS2:          DW      RFROM
PARS3:          DW      OVER,SWAP
                DW      TOR
PARS4:          DW      TEMP,AT,OVER,CAT,SUBB        ;scan for delimiter
                DW      TEMP,AT,BLANK,EQUAL
                DW      QBRAN,PARS5
                DW      ZLESS
PARS5:          DW      QBRAN,PARS6
                DW      DOLIT,1,PLUS
                DW      DONXT,PARS4
                DW      DUPP,TOR
                DW      BRAN,PARS7
PARS6:          DW      RFROM,DROP,DUPP
                DW      DOLIT,1,PLUS,TOR
PARS7:          DW      OVER,SUBB
                DW      RFROM,RFROM,SUBB,EXIT
PARS8:          DW      OVER,RFROM,SUBB,EXIT

;  PARSE        ( c -- b u ; <string> )
;               Scan input stream and return counted string delimited by c.

                $COLON          5,'PARSE',PARSE
                DW      TOR,TIB,INN,AT,PLUS ;current input buffer pointer
                DW      NTIB,AT,INN,AT,SUBB ;remaining count
                DW      RFROM,PARS,INN,PSTOR,EXIT

;  .(           ( -- )
```

```
;               Output following string up to next ) .

                $COLON      IMEDD+2,'.(',DOTPR
                DW      DOLIT,')',PARSE,TYPEE,EXIT

;  (            ( -- )
;               Ignore following string up to next ) . A comment.

                $COLON      IMEDD+1,'(',PAREN
                DW      DOLIT,')',PARSE,DDROP,EXIT

;  \            ( -- )
;               Ignore following text till the end of line.

                $COLON      IMEDD+1,'\',BKSLA
                DW      NTIB,AT,INN,STORE,EXIT

;  CHAR         ( -- c )
;               Parse next word and return its first character.

                $COLON      4,'CHAR',CHAR
                DW      BLANK,PARSE,DROP,CAT,EXIT

;  TOKEN        ( -- a ; <string> )
;               Parse a word from input stream and copy it to name dictionary.

                $COLON      5,'TOKEN',TOKEN
                DW      BLANK,PARSE,DOLIT,31,MIN
                DW      NP,AT,OVER,SUBB,CELLM
                DW      PACKS,EXIT

;  WORD         ( c -- a ; <string> )
;               Parse a word from input stream and copy it to code dictionary.

                $COLON      4,'WORD',WORDD
                DW      PARSE,HERE,PACKS,EXIT

;; Dictionary search

;  NAME>        ( na -- ca )
;               Return a code address given a name address.

                $COLON      5,'NAME>',NAMET
                DW      CELLM,CELLM,AT,EXIT

;  SAME?        ( a a u -- a a f \ -0+ )
;               Compare u cells in two strings. Return 0 if identical.

                $COLON      5,'SAME?',SAMEQ
                DW      TOR
                DW      BRAN,SAME2
SAME1:                  DW      OVER,RAT,CELLS,PLUS,AT
                DW      OVER,RAT,CELLS,PLUS,AT
                DW      SUBB,QDUP
```

```
                DW      QBRAN,SAME2
                DW      RFROM,DROP,EXIT             ;strings not equal
SAME2:                  DW      DONXT,SAME1
                DW      DOLIT,0,EXIT        ;strings equal

;  find   ( a va -- ca na | a F )
;               Search a vocabulary for a string. Return ca and na if succeeded.

                $COLON      4,'find',FIND
                DW      SWAP,DUPP,CAT
                DW      DOLIT,CELLL,SLASH,TEMP,STORE
                DW      DUPP,AT,TOR,CELLP,SWAP
FIND1:          DW      AT,DUPP
                DW      QBRAN,FIND6
                DW      DUPP,AT,DOLIT,MASKK,ANDD,RAT,XORR
                DW      QBRAN,FIND2
                DW      CELLP,DOLIT,-1              ;true flag
                DW      BRAN,FIND3
FIND2:          DW      CELLP,TEMP,AT,SAMEQ
FIND3:          DW      BRAN,FIND4
FIND6:          DW      RFROM,DROP
                DW      SWAP,CELLM,SWAP,EXIT
FIND4:          DW      QBRAN,FIND5
                DW      CELLM,CELLM
                DW      BRAN,FIND1
FIND5:          DW      RFROM,DROP,SWAP,DROP
                DW      CELLM
                DW      DUPP,NAMET,SWAP,EXIT

;  NAME?        ( a -- ca na | a F )
;               Search all context vocabularies for a string.

                $COLON      5,'NAME?',NAMEQ
                DW      CNTXT,DUPP,DAT,XORR         ;?context=also
                DW      QBRAN,NAMQ1
                DW      CELLM                       ;no, start with context
NAMQ1:                  DW      TOR
NAMQ2:                  DW      RFROM,CELLP,DUPP,TOR        ;next in search order
                DW      AT,QDUP
                DW      QBRAN,NAMQ3
                DW      FIND,QDUP                   ;search vocabulary
                DW      QBRAN,NAMQ2
                DW      RFROM,DROP,EXIT             ;found name
NAMQ3:                  DW      RFROM,DROP          ;name not found
                DW      DOLIT,0,EXIT                ;false flag

;; Terminal response

;  ^H           ( bot eot cur -- bot eot cur )
;               Backup the cursor by one character.

                $COLON      2,'^H',BKSP
                DW      TOR,OVER,RFROM,SWAP,OVER,XORR
                DW      QBRAN,BACK1
```

```
                DW      DOLIT,BKSPP,TECHO,ATEXE,DOLIT,1,SUBB
                DW      BLANK,TECHO,ATEXE
                DW      DOLIT,BKSPP,TECHO,ATEXE
BACK1:                  DW      EXIT


;   TAP         ( bot eot cur c -- bot eot cur )
;               Accept and echo the key stroke and bump the cursor.

                $COLON          3,'TAP',TAP
                DW      DUPP,TECHO,ATEXE
                DW      OVER,CSTOR,DOLIT,1,PLUS,EXIT

;   kTAP( bot eot cur c -- bot eot cur )
;               Process a key stroke, CR or backspace.

                $COLON          4,'kTAP',KTAP
                DW      DUPP,DOLIT,CRR,XORR
                DW      QBRAN,KTAP2
                DW      DOLIT,BKSPP,XORR
                DW      QBRAN,KTAP1
                DW      BLANK,TAP,EXIT
KTAP1:          DW      BKSP,EXIT
KTAP2:          DW      DROP,SWAP,DROP,DUPP,EXIT

;   accept      ( b u -- b u )
;               Accept characters to input buffer. Return with actual count.

                $COLON          6,'accept',ACCEP
                DW      OVER,PLUS,OVER
ACCP1:          DW      DDUP,XORR
                DW      QBRAN,ACCP4
                DW      KEY,DUPP
;               DW      BLANK,SUBB,DOLIT,95,ULESS
                DW      BLANK,DOLIT,127,WITHI
                DW      QBRAN,ACCP2
                DW      TAP
                DW      BRAN,ACCP3
ACCP2:          DW      TTAP,ATEXE
ACCP3:          DW      BRAN,ACCP1
ACCP4:          DW      DROP,OVER,SUBB,EXIT

;   EXPECT      ( b u -- )
;               Accept input stream and store count in SPAN.

                $COLON          6,'EXPECT',EXPEC
                DW      TEXPE,ATEXE,SPAN,STORE,DROP,EXIT

;   QUERY       ( -- )
;               Accept input stream to terminal input buffer.

                $COLON          5,'QUERY',QUERY
                DW      TIB,DOLIT,80,TEXPE,ATEXE,NTIB,STORE
                DW      DROP,DOLIT,0,INN,STORE,EXIT
```

```
;; Error handling

;  CATCH        ( ca -- 0 | err# )
;               Execute word at ca and set up an error frame for it.

                $COLON       5,'CATCH',CATCH
                DW      SPAT,TOR,HANDL,AT,TOR       ;save error frame
                DW      RPAT,HANDL,STORE,EXECU      ;execute
                DW      RFROM,HANDL,STORE           ;restore error frame
                DW      RFROM,DROP,DOLIT,0,EXIT     ;no error

;  THROW        ( err# -- err# )
;               Reset system to current local error frame an update error flag.

                $COLON       5,'THROW',THROW
                DW      HANDL,AT,RPSTO              ;restore return stack
                DW      RFROM,HANDL,STORE           ;restore handler frame
                DW      RFROM,SWAP,TOR,SPSTO        ;restore data stack
                DW      DROP,RFROM,EXIT

;  NULL$        ( -- a )
;               Return address of a null string with zero count.

                $COLON       5,'NULL$',NULLS
                DW      DOVAR                       ;emulate CREATE
                DW      0
                DB      99,111,121,111,116,101
                $ALIGN

;  ABORT        ( -- )
;               Reset data stack and jump to QUIT.

                $COLON       5,'ABORT',ABORT
                DW      NULLS,THROW

;  abort"       ( f -- )
;               Run time routine of ABORT" . Abort with a message.

                $COLON       COMPO+6,'abort"',ABORQ
                DW      QBRAN,ABOR1                 ;text flag
                DW      DOSTR,THROW                 ;pass error string
ABOR1:                  DW      DOSTR,DROP,EXIT             ;drop error

;; The text interpreter

;  $INTERPRET( a -- )
;               Interpret a word. If failed, try to convert it to an integer.

                $COLON       10,'$INTERPRET',INTER
                DW      NAMEQ,QDUP                  ;?defined
                DW      QBRAN,INTE1
                DW      AT,DOLIT,COMPO,ANDD         ;?compile only lexicon bits
                D$      ABORQ,' compile only'
```

Biblioteca Universitária
UFSC

```
                DW      EXECU,EXIT                  ;execute defined word
INTE1:          DW      TNUMB,ATEXE                         ;convert a number
                DW      QBRAN,INTE2
                DW      EXIT
INTE2:          DW      THROW                       ;error

;  [            ( -- )
;               Start the text interpreter.

                $COLON          IMEDD+1,'[',LBRAC
                DW      DOLIT,INTER,TEVAL,STORE,EXIT

;  .OK          ( -- )
;               Display 'ok' only while interpreting.

                $COLON          3,'.OK',DOTOK
                DW      DOLIT,INTER,TEVAL,AT,EQUAL
                DW      QBRAN,DOTO1
                D$      DOTQP,' ok'
DOTO1:                  DW      CR,EXIT

;  ?STACK       ( -- )
;               Abort if the data stack underflows.

                $COLON          6,'?STACK',QSTAC
                DW      DEPTH,ZLESS                 ;check only for underflow
                D$      ABORQ,' underflow'
                DW      EXIT

;  EVAL         ( -- )
;               Interpret the input stream.

                $COLON          4,'EVAL',EVAL
EVAL1:          DW      TOKEN,DUPP,CAT              ;?input stream empty
                DW      QBRAN,EVAL2
                DW      TEVAL,ATEXE,QSTAC           ;evaluate input, check stack
                DW      BRAN,EVAL1
EVAL2:          DW      DROP,TPROM,ATEXE,EXIT       ;prompt

;; Shell

;  PRESET       ( -- )
;               Reset data stack pointer and the terminal input buffer.

                $COLON          6,'PRESET',PRESE
                DW      SZERO,AT,SPSTO
                DW      DOLIT,TIBB,NTIB,CELLP,STORE,EXIT

;  xio          ( a a a -- )
;               Reset the I/O vectors 'EXPECT, 'TAP, 'ECHO and 'PROMPT.

                $COLON          COMPO+3,'xio',XIO
                DW      DOLIT,ACCEP,TEXPE,DSTOR
                DW      TECHO,DSTOR,EXIT
```

```
;  FILE ( -- )
;               Select I/O vectors for file download.

                $COLON      4,'FILE',FILE
                DW      DOLIT,PACE,DOLIT,DROP
                DW      DOLIT,KTAP,XIO,EXIT

;  HAND         ( -- )
;               Select I/O vectors for terminal interface.

                $COLON      4,'HAND',HAND
                DW      DOLIT,DOTOK,DOLIT,EMIT
                DW      DOLIT,KTAP,XIO,EXIT

;  I/O          ( -- a )
;               Array to store default I/O vectors.

                $COLON      3,'I/O',ISLO
                DW      DOVAR                       ;emulate CREATE
                DW      QRX,TXSTO                   ;default I/O vectors

;  CONSOLE      ( -- )
;               Initiate terminal interface.

                $COLON      7,'CONSOLE',CONSO
                DW      ISLO,DAT,TQKEY,DSTOR        ;restore default I/O device
                DW      HAND,EXIT                   ;keyboard input

;  QUIT( -- )
;               Reset return stack pointer and start text interpreter.

                $COLON      4,'QUIT',QUIT
                DW      RZERO,AT,RPSTO              ;reset return stack pointer
QUIT1:          DW      LBRAC                       ;start interpretation
QUIT2:          DW      QUERY                       ;get input
                DW      DOLIT,EVAL,CATCH,QDUP       ;evaluate input
                DW      QBRAN,QUIT2                 ;continue till error
                DW      TPROM,AT,SWAP               ;save input device
                DW      CONSO,NULLS,OVER,XORR       ;?display error message
                DW      QBRAN,QUIT3
                DW      SPACE,COUNT,TYPEE           ;error message
                D$      DOTQP,' ? '                 ;error prompt
QUIT3:          DW      DOLIT,DOTOK,XORR            ;?file input
                DW      QBRAN,QUIT4
                DW      DOLIT,ERR,EMIT              ;file error, tell host
QUIT4:          DW      PRESE                       ;some cleanup
                DW      BRAN,QUIT1

;; The compiler

;  '            ( -- ca )
;               Search context vocabularies for the next word in input stream.
```

```
                $COLON      1,"'",TICK
                DW    TOKEN,NAMEQ                 ;?defined
                DW    QBRAN,TICK1
                DW    EXIT                        ;yes, push code address
TICK1:          DW    THROW                       ;no, error

;   ALLOT       ( n -- )
;               Allocate n bytes to the code dictionary.

                $COLON      5,'ALLOT',ALLOT
                DW    CP,PSTOR,EXIT                     ;adjust code pointer

;   ,           ( w -- )
;               Compile an integer into the code dictionary.

                $COLON      1,',',COMMA
                DW    HERE,DUPP,CELLP             ;cell boundary
                DW    CP,STORE,STORE,EXIT               ;adjust code pointer, compile

;   [COMPILE]   ( -- ; <string> )
;               Compile the next immediate word into code dictionary.

                $COLON      IMEDD+9,'[COMPILE]',BCOMP
                DW    TICK,COMMA,EXIT

;   COMPILE     ( -- )
;               Compile the next address in colon list to code dictionary.

                $COLON      COMPO+7,'COMPILE',COMPI
                DW    RFROM,DUPP,AT,COMMA               ;compile address
                DW    CELLP,TOR,EXIT                    ;adjust return address

;   LITERAL     ( w -- )
;               Compile tos to code dictionary as an integer literal.

                $COLON      IMEDD+7,'LITERAL',LITER
                DW    COMPI,DOLIT,COMMA,EXIT

;   $,"         ( -- )
;               Compile a literal string up to next " .

                $COLON      3,'$,"',STRCQ
                DW    DOLIT,'"',WORDD             ;move string to code dictionary
                DW    COUNT,PLUS,ALGND            ;calculate aligned end of string
                DW    CP,STORE,EXIT                     ;adjust the code pointer

;   RECURSE     ( -- )
;               Make the current word available for compilation.

                $COLON      IMEDD+7,'RECURSE',RECUR
                DW    LAST,AT,NAMET,COMMA,EXIT

;; Structures
```

```
;  FOR        ( -- a )
;             Start a FOR-NEXT loop structure in a colon definition.

              $COLON      IMEDD+3,'FOR',FOR
              DW     COMPI,TOR,HERE,EXIT

;  BEGIN      ( -- a )
;             Start an infinite or indefinite loop structure.

              $COLON      IMEDD+5,'BEGIN',BEGIN
              DW     HERE,EXIT

;  NEXT       ( a -- )
;             Terminate a FOR-NEXT loop structure.

              $COLON      IMEDD+4,'NEXT',NEXT
              DW     COMPI,DONXT,COMMA,EXIT

;  UNTIL      ( a -- )
;             Terminate a BEGIN-UNTIL indefinite loop structure.

              $COLON      IMEDD+5,'UNTIL',UNTIL
              DW     COMPI,QBRAN,COMMA,EXIT

;  AGAIN      ( a -- )
;             Terminate a BEGIN-AGAIN infinite loop structure.

              $COLON      IMEDD+5,'AGAIN',AGAIN
              DW     COMPI,BRAN,COMMA,EXIT

;  IF         ( -- A )
;             Begin a conditional branch structure.

              $COLON      IMEDD+2,'IF',IFFF
              DW     COMPI,QBRAN,HERE
              DW     DOLIT,0,COMMA,EXIT

;  AHEAD      ( -- A )
;             Compile a forward branch instruction.

              $COLON      IMEDD+5,'AHEAD',AHEAD
              DW     COMPI,BRAN,HERE,DOLIT,0,COMMA,EXIT

;  REPEAT     ( A a -- )
;             Terminate a BEGIN-WHILE-REPEAT indefinite loop.

              $COLON      IMEDD+6,'REPEAT',REPEA
              DW     AGAIN,HERE,SWAP,STORE,EXIT

;  THEN       ( A -- )
;             Terminate a conditional branch structure.

              $COLON      IMEDD+4,'THEN',THENN
              DW     HERE,SWAP,STORE,EXIT
```

```
;  AFT          ( a -- a A )
;               Jump to THEN in a FOR-AFT-THEN-NEXT loop the first time through.

                $COLON      IMEDD+3,'AFT',AFT
                DW     DROP,AHEAD,BEGIN,SWAP,EXIT

;  ELSE( A -- A )
;               Start the false clause in an IF-ELSE-THEN structure.

                $COLON      IMEDD+4,'ELSE',ELSEE
                DW     AHEAD,SWAP,THENN,EXIT

;  WHILE        ( a -- A a )
;               Conditional branch out of a BEGIN-WHILE-REPEAT loop.

                $COLON      IMEDD+5,'WHILE',WHILE
                DW     IFFF,SWAP,EXIT

;  ABORT"       ( -- ; <string> )
;               Conditional abort with an error message.

                $COLON      IMEDD+6,'ABORT"',ABRTQ
                DW     COMPI,ABORQ,STRCQ,EXIT

;  $"           ( -- ; <string> )
;               Compile an inline string literal.

                $COLON      IMEDD+2,'$"',STRQ
                DW     COMPI,STRQP,STRCQ,EXIT

;  ."           ( -- ; <string> )
;               Compile an inline string literal to be typed out at run time.

                $COLON      IMEDD+2,'."',DOTQ
                DW     COMPI,DOTQP,STRCQ,EXIT

;; Name compiler

;  ?UNIQUE      ( a -- a )
;               Display a warning message if the word already exists.

                $COLON      7,'?UNIQUE',UNIQU
                DW     DUPP,NAMEQ            ;?name exists
                DW     QBRAN,UNIQ1           ;redefinitions are OK
                D$     DOTQP,' reDef '       ;but warn the user
                DW     OVER,COUNT,TYPEE      ;just in case its not planned
UNIQ1:          DW     DROP,EXIT

;  $,n          ( na -- )
;               Build a new dictionary name using the string at na.

                $COLON      3,'$,n',SNAME
                DW     DUPP,CAT                     ;?null input
```

```
                DW      QBRAN,PNAM1
                DW      UNIQU                       ;?redefinition
                DW      DUPP,LAST,STORE             ;save na for vocabulary link
                DW      HERE,ALGND,SWAP             ;align code address
                DW      CELLM                       ;link address
                DW      CRRNT,AT,AT,OVER,STORE
                DW      CELLM,DUPP,NP,STORE         ;adjust name pointer
                DW      STORE,EXIT                  ;save code pointer
PNAM1:                  D$      STRQP,' name'       ;null input
                DW      THROW

;; FORTH compiler

;   $COMPILE    ( a -- )
;               Compile next word to code dictionary as a token or literal.

                $COLON          8,'$COMPILE',SCOMP
                DW      NAMEQ,QDUP                  ;?defined
                DW      QBRAN,SCOM2
                DW      AT,DOLIT,IMEDD,ANDD         ;?immediate
                DW      QBRAN,SCOM1
                DW      EXECU,EXIT                  ;its immediate, execute
SCOM1:                  DW      COMMA,EXIT          ;its not immediate, compile
SCOM2:                  DW      TNUMB,ATEXE                 ;try to convert to number
                DW      QBRAN,SCOM3
                DW      LITER,EXIT                  ;compile number as integer
SCOM3:                  DW      THROW               ;error

;   OVERT       ( -- )
;               Link a new word into the current vocabulary.

                $COLON          5,'OVERT',OVERT
                DW      LAST,AT,CRRNT,AT,STORE,EXIT

;   ;           ( -- )
;               Terminate a colon definition.

                $COLON          IMEDD+COMPO+1,';',SEMIS
                DW      COMPI,EXIT,LBRAC,OVERT,EXIT

;   ]           ( -- )
;               Start compiling the words in the input stream.

                $COLON          1,']',RBRAC
                DW      DOLIT,SCOMP,TEVAL,STORE,EXIT

;   call,   ( ca -- )
;               Assemble a call instruction to ca.

                $COLON          5,'call,',CALLC
                DW      DOLIT,CALLL,COMMA,HERE              ;Direct Threaded Code
                DW      CELLP,SUBB,COMMA,EXIT               ;DTC 8086 relative call

;   :           ( -- ; <string> )
```

```
;               Start a new colon definition using next word as its name.

                $COLON       1,':',COLON
                DW     TOKEN,SNAME,DOLIT,DOLST
                DW     CALLC,RBRAC,EXIT

; IMMEDIATE ( -- )
;               Make the last compiled word an immediate word.

                $COLON       9,'IMMEDIATE',IMMED
                DW     DOLIT,IMEDD,LAST,AT,AT,ORR
                DW     LAST,AT,STORE,EXIT

;;************************ Defining words

; USER          ( u -- ; <string> )
;               Compile a new user variable.

                $COLON       4,'USER',USER
                DW     TOKEN,SNAME,OVERT
                DW     DOLIT,DOLST,CALLC
                DW     COMPI,DOUSE,COMMA,EXIT

; CREATE        ( -- ; <string> )
;               Compile a new array entry without allocating code space.
;               Foi modificada devido a insercao do DOES>.

        $COLON  6,'CREATE',CREAT
                DW     TOKEN,SNAME,OVERT
                DW     DOLIT,DOLST,CALLC
        DW    COMPI,NNOPP,COMPI,DOVAR,EXIT  ; insercao de palavras

; VARIABLE  ( -- ; <string> )
;               Compile a new variable initialized to 0.

                $COLON       8,'VARIABLE',VARIA
                DW     CREAT,DOLIT,0,COMMA,EXIT

; NOP   ( nao faz nada, apenas um espaco para salvar endereco )
;       Definicao utilizada no DOES> e CREATE.

        $COLON  3,'NOP',NNOPP
        DW    EXIT

; does>     ( funcao auxiliar do DOES> )

        $COLON  5,'does>',DODOES
        DW    LAST,AT,NAMET,DUPP
        DW    DOLIT,BRAN,SWAP,DOLIT,0004,PLUS,STORE,DUPP
        DW    HERE,SWAP,DOLIT,0006,PLUS,STORE
        DW    COMPI,DOLIT
        DW    DOLIT,0008,PLUS,COMMA
        DW    COMPI,BRAN,EXIT
```

```
;  DOES>      ( DOES> propriamente. E' immediate. )

        $COLON  IMEDD+5,'DOES>',DOESTO
        DW      DOLIT,DODOES,COMMA
        DW      DOLIT,DOLIT,COMMA
        DW      HERE,DOLIT,0006,PLUS,COMMA
        DW      DOLIT,COMMA,COMMA
        DW      DOLIT,EXIT,COMMA
        DW      EXIT

;; ************************** Tools

;  _TYPE        ( b u -- )
;               Display a string. Filter non-printing characters.

                $COLON      5,'_TYPE',UTYPE
                DW      TOR                     ;start count down loop
                DW      BRAN,UTYP2              ;skip first pass
UTYP1:          DW      DUPP,CAT,TCHAR,EMIT           ;display only printable
                DW      DOLIT,1,PLUS            ;increment address
UTYP2:          DW      DONXT,UTYP1             ;loop till done
                DW      DROP,EXIT

;  dm+          ( a u -- a )
;               Dump u bytes from , leaving a+u on the stack.

                $COLON      3,'dm+',DMP
                DW      OVER,DOLIT,4,UDOTR ;display address
                DW      SPACE,TOR               ;start count down loop
                DW      BRAN,PDUM2              ;skip first pass
PDUM1:                  DW      DUPP,CAT,DOLIT,3,UDOTR      ;display numeric data
                DW      DOLIT,1,PLUS            ;increment address
PDUM2:                  DW      DONXT,PDUM1                 ;loop till done
                DW      EXIT

;  DUMP         ( a u -- )
;               Dump u bytes from a, in a formatted manner.

                $COLON      4,'DUMP',DUMP
                DW      BASE,AT,TOR,HEX                 ;save radix, set hex
                DW      DOLIT,16,SLASH                  ;change count to lines
                DW      TOR                     ;start count down loop
DUMP1:                  DW      CR,DOLIT,16,DDUP,DMP        ;display numeric
                DW      ROT,ROT
                DW      SPACE,SPACE,UTYPE  ;display printable characters
                DW      NUFQ,INVER              ;user control
                DW      QBRAN,DUMP2
                DW      DONXT,DUMP1                     ;loop till done
                DW      BRAN,DUMP3
DUMP2:                  DW      RFROM,DROP              ;cleanup loop stack, early exit
DUMP3:                  DW      DROP,RFROM,BASE,STORE       ;restore radix
                DW      EXIT

;  .S           ( ... -- ... )
```

```
;               Display the contents of the data stack.

                $COLON      2,'.S',DOTS
                DW      CR,DEPTH                ;stack depth
                DW      TOR                     ;start count down loop
                DW      BRAN,DOTS2              ;skip first pass
DOTS1:          DW      RAT,PICK,DOT            ;index stack, display contents
DOTS2:          DW      DONXT,DOTS1             ;loop till done
                D$      DOTQP,' <sp'
                DW      EXIT

;  !CSP ( -- )
;               Save stack pointer in CSP for error checking.

                $COLON      4,'!CSP',STCSP
                DW      SPAT,CSP,STORE,EXIT ;save pointer

;  ?CSP ( -- )
;               Abort if stack pointer differs from that saved in CSP.

                $COLON      4,'?CSP',QCSP
                DW      SPAT,CSP,AT,XORR        ;compare pointers
                D$      ABORQ,'stacks'          ;abort if different
                DW      EXIT

;  >NAME        ( ca -- na | F )
;               Convert code address to a name address.

                $COLON      5,'>NAME',TNAME
                DW      CRRNT                           ;vocabulary link
TNAM1:          DW      CELLP,AT,QDUP           ;check all vocabularies
                DW      QBRAN,TNAM4
                DW      DDUP
TNAM2:          DW      AT,DUPP                 ;?last word in a vocabulary
                DW      QBRAN,TNAM3
                DW      DDUP,NAMET,XORR                 ;compare
                DW      QBRAN,TNAM3
                DW      CELLM                           ;continue with next word
                DW      BRAN,TNAM2
TNAM3:          DW      SWAP,DROP,QDUP
                DW      QBRAN,TNAM1
                DW      SWAP,DROP,SWAP,DROP,EXIT
TNAM4:          DW      DROP,DOLIT,0,EXIT       ;false flag

;  .ID          ( na -- )
;               Display the name at address.

                $COLON      3,'.ID',DOTID
                DW      QDUP                            ;if zero no name
                DW      QBRAN,DOTI1
                DW      COUNT,DOLIT,01FH,ANDD           ;mask lexicon bits
                DW      UTYPE,EXIT                      ;display name string
DOTI1:          D$      DOTQP,' {noName}'
                DW      EXIT
```

```
;   SEE         ( -- ; <string> )
;               A simple decompiler.

                $COLON      3,'SEE',SEE
                DW      TICK                    ;starting address
                DW      CR,CELLP
SEE1:           DW      CELLP,DUPP,AT,DUPP ;?does it contain a zero
                DW      QBRAN,SEE2
                DW      TNAME                   ;?is it a name
SEE2:           DW      QDUP                    ;name address or zero
                DW      QBRAN,SEE3
                DW      SPACE,DOTID             ;display name
                DW      BRAN,SEE4
SEE3:           DW      DUPP,AT,UDOT            ;display number
SEE4:           DW      NUFQ                    ;user control
                DW      QBRAN,SEE1
                DW      DROP,EXIT

;   WORDS       ( -- )
;               Display the names in the context vocabulary.

                $COLON      5,'WORDS',WORDS
                DW      CR,CNTXT,AT             ;only in context
WORS1:          DW      AT,QDUP         ;?at end of list
                DW      QBRAN,WORS2
                DW      DUPP,SPACE,DOTID        ;display a name
                DW      CELLM,NUFQ              ;user control
                DW      QBRAN,WORS1
                DW      DROP
WORS2:          DW      EXIT

;; Hardware reset

;   VER         ( -- n )
;               Return the version number of this implementation.

                $COLON      3,'VER',VERSN
                DW      DOLIT,VER*256+EXT,EXIT

;   hi          ( -- )
;               Display the sign-on message of eForth.

                $COLON      2,'hi',HI
                DW      STOIO,CR                ;initialize I/O
                D$      DOTQP,'eForth v'        ;model
                DW      BASE,AT,HEX             ;save radix
                DW      VERSN,BDIGS,DIG,DIG
                DW      DOLIT,'.',HOLD
                DW      DIGS,EDIGS,TYPEE        ;format version number
                DW      BASE,STORE,CR,EXIT ;restore radix

;   'BOOT       ( -- a )
;               The application startup vector.
```

```
            $COLON      5,"'BOOT",TBOOT
            DW    DOVAR
            DW    HI                  ;application to boot

;  COLD     ( -- )
;           The hilevel cold start sequence.

            $COLON      4,'COLD',COLD
COLD1:      DW    DOLIT,UZERO,DOLIT,UPP
            DW    DOLIT,ULAST-UZERO,CMOVE ;initialize user area
            DW    PRESE               ;initialize stack and TIB
            DW    TBOOT,ATEXE         ;application boot
            DW    FORTH,CNTXT,AT,DUPP       ;initialize search order
            DW    CRRNT,DSTOR,OVERT
            DW    QUIT                ;start interpretation
            DW    BRAN,COLD1          ;just in case

;===========================================================

LASTN       EQU   _NAME+4                   ;last name address

NTOP        EQU   _NAME-0                   ;next available memory in name dictionary
CTOP        EQU   $+0                 ;next available memory in code dictionary

MAIN  ENDS
END   ORIG

;===========================================================
```

Anexo - B

```
\ FUZZY.TXT - definições para o programa de controle e declarações de dados
\ ***********************************************************************
\ Definições para declaração das variáveis linguísticas de entrada e saída

\ criar uma variável linguística de entrada
: FVARIN CREATE HERE ;   ( -- pfa ;<nome> )

\ criar o universo de discurso
: UNIVERS   ( pfa ud_ini ud_end tam -- )
     10 ALLOT           ( -- pfa ud_ini ud_end tam )
     0 3 PICK !                             \ zera off-set
     0 3 PICK 2 + !                    \ zera VL
     3 PICK             ( -- pfa ud_ini ud_end tam pfa )
     4 + !              ( -- pfa ud_ini ud_end )      \ salva tam
     2 PICK 6 + !       ( -- pfa ud_ini )             \ salva ud_end
     SWAP 8 + ! ;       ( -- )                        \ salva ud_ini

\ criar uma variável linguística de saída
: FVAROUT CREATE        ( -- ; <nome> )
               HERE          ( -- pfa )
               2 ALLOT       ( -- pfa )
               0 SWAP ! ;    ( -- )       \ zera posicao do valor de saída

\ declarar um singleton
: SINGLETON     ( VS --   / <string> )
     CREATE
     HERE                   ( -- VS pfa )
     4 ALLOT                ( -- VS pfa )
     DUP 0 SWAP !           ( -- VS pfa )     \ zera posicao para GP
     2 + ! ;                ( -- )      \ salva VS = valor do singleton

\ declarar um conjunto difuso normal
: MEMBER    ( pfa -- pfa pfa1 ; \<string> )
     DUP                ( -- pfa pfa )
       CREATE    HERE DUP    ( -- pfa pfa pfa1 pfa1 )
       ROT 4 + @             ( -- pfa pfa1 pfa1 tam )
     DUP 6 +                 ( -- pfa pfa1 pfa1 tam tam+6 )
     ALLOT              ( -- pfa pfa1 pfa1 tam )      \ aloca tam+6 bytes
     OVER               ( -- pfa pfa1 pfa1 tam pfa1 )
     0 SWAP !           ( -- pfa pfa1 pfa1 tam )      \ zera GP
     OVER DUP 6 +            ( -- pfa pfa1 pfa1 tam pfa1 pfa1+6 )
     SWAP 2 +           ( -- pfa pfa1 pfa1 tam pfa1+6 pfa1+2 )
     !                  ( -- pfa pfa1 pfa1 tam )   \ salva ptr_tab_gp
     SWAP 4 +           ( -- pfa pfa1 tam pfa1+4 )
     ! ;                ( -- pfa pfa1 )            \ salva tam

: MEMBERS[              ( -- )
     SP@ tmp1 ! ;                     \ salva SP atual em tmp1

\ O OFF-SET é um byte, não uma word.
: GP>MEMBER             ( pfa OFF-SET -- OFF-SET ) \ salva OFF-SET
     DUP                ( -- pfa OFF-SET OFF-SET ) \ na pfa da funcao membro
     ROT                ( -- OFF-SET OFF-SET pfa )
     DUP 6 +                 ( -- OFF-SET OFF-SET pfa pfa+6 ) \ ini. tabela
     ROT +              ( -- OFF-SET pfa pfa+6+OFF-SET)
     C@                 ( -- OFF-SET pfa GP )          \ lê o byte
     SWAP C! ;          ( -- OFF-SET )                 \ salva como byte

: ]>   ( pfa pfa.... pfa -- )
```

```
    tmp1 @ SP@ - 4 - 2 / ;  ( -- pfa pfa ... pfa n-1 )

: FUZZYFI               ( pfa pfa ... pfa n-1 pfa -- \<string> )
      C@                ( -- pfa pfa ... pfa n-1 OFF-SET )
    SWAP FOR            ( -- pfa pfa ... pfa OFF-SET )
         GP>MEMBER      ( pfa OFF-SET -- OFF-SET )
        NEXT            ( -- OFF-SET )
    DROP ;              ( -- )

\ ***********************************************************
\ definições para a declaração das regras

: IFF        ( -- )       \  Salva posição do stack.
             !CSP ;

: END-RULE         ( MIN pfa -- )
             DUP          ( -- MIN pfa pfa )
             @            ( -- MIN pfa GP )
             ROT          ( -- pfa GP MIN )
             MAX          ( -- pfa MAX )
             SWAP ! ;     ( -- )

\ *******************************************************
\ definições auxiliares para declaração da fuzzyficação e defuzzyficação

: CLEAR-MEMBERS         ( pfa pfa ... pfa -- )
     DEPTH 1 -          ( pfa pfa ... pfa n-1 -- )
     FOR                ( pfa pfa ... pfa -- )
        0 SWAP !
     NEXT ;             ( -- )


: PILE-OUT  ( pfa -- )  \ usa tmp1 e tmp2 para calculos intermediarios
     DUP                ( -- pfa pfa )
     @                  ( -- pfa GP )
     5 *                ( -- pfa GPn )          \ normaliza
     DUP                ( -- pfa GPn GPn )
     tmp1 @             ( -- pfa GPn GPn [tmp1] )
     +                  ( -- pfa GPn GPn+[tmp1] )
     tmp1 !             ( -- pfa GPn ) \ salva somatorio de GPs nao normal.
     SWAP 2 +           ( -- GPn pfa+2 )
     @                  ( -- GPn VS )
     *                  ( -- n )          \ n = GPn x VS / 100
     55 + 100 /         ( -- n1 )         \ n1=n arredondado e na escala
     tmp2 @ +           ( -- n1 [tmp2] )
     tmp2 ! ;               ( -- )        \ salva somatório de GPxVS
                                          \ ja na escala normalisada

\ n-1 é o número de pfas na pilha desde MEMBERS[, menos 1.
: DEFUZZYFI        ( pfa pfa pfa ... pfa n-1 -- )
     0 tmp1 ! 0 tmp2 ! ( -- pfa pfa pfa ... pfa n-1 )
     FOR                ( -- pfa pfa pfa ... pfa )
        PILE-OUT        ( pfa -- )
     NEXT               ( -- pfa )
     tmp2 @ 1000        ( -- pfa [tmp2] 1000 )   \ somat.GPsxVSs
     tmp1 @             ( -- pfa [tmp2] 1000 [tmp1] )
     */ 5 + 10 /        ( -- pfa quot )
     SWAP ! ;           ( -- )            \ salva val defuz. na var out


\ *****************************************************************
\ definições para a declaração das funções de pertinência
```

```
\ ajusta pto em decimal, na escala real, para escala do sistema
\ recebe pfa da FVARIN para extrair os parâmetros do ud, a cada chamada
: FIT ( pto pfa -- n )
     SWAP OVER          ( -- pfa pto pfa )
     8 + @ -            ( -- pfa {pto - ud-ini} )
     OVER 4 + @ 10 *    ( -- pfa {pto - ud-ini} tam*10 ) \ tam em nro de pos
     2 PICK             ( -- pfa {pto - ud-ini} tam*10 pfa )
     6 + @              ( -- pfa {pto - ud-ini} tam*10 ud-end )
     3 PICK             ( -- pfa {pto - ud-ini} tam*10 ud-end pfa )
     8 + @              ( -- pfa {pto - ud-ini} tam*10 ud-end ud-ini )
     -                  ( -- pfa {pto - ud-ini} tam*10 {ud-end - ud-ini} )
     */                 ( -- pfa n1 )     \ n1: na escala do sistema
     5 + 10 /     ( -- pfa n )            \ arredondado
     SWAP DROP ; ( -- n )                 \ tira pfa

\ calcula o GP para um pto n no ud do sistema para a rampa positiva

: n>GPr+    ( n -- n1 )
     PTO-A @ -          ( -- {n - PTO-A} )
     200 PTO-B @
     PTO-A @ -          ( -- {n - PTO-A} 200 {PTO-B - PTO-A} )
     */ ;               ( -- n1 )

\ calcula o GP para um pto n no ud do sistema para a rampa negativa
\ Ptos: C inicia descida, D termina descida.
: n>GPr-         ( n -- )    \ calcula o GP para um pto n no ud do sistema
para a rampa negativa
     PTO-D @ SWAP -     ( -- D-n )
     200 PTO-D @        ( -- D-n 200 D )
     PTO-C @ -          ( -- D-n 200 D-C )
     */ ;               ( -- n1 )

\ entrada: pfa1, da função membro
: POS-RAMP  ( pfa1 -- pfa1 )
     DUP 6 +            ( -- pfa1 ptr-tab )
     PTO-A @            ( -- pfa1 ptr-tab A )
     PTO-B @            ( -- pfa1 ptr-tab A B )
     OVER - 1 -         ( -- pfa1 ptr-tab A B-A-2 )
     FOR                ( -- pfa1 ptr-tab A )
        1 + DUP         ( -- pfa1 ptr-tab A+1 A+1 )
        n>GPr+          ( -- pfa1 ptr-tab A+1 n1 )
        1 PICK 3 PICK +      ( -- pfa1 ptr-tab A+1 n1 A+1+ptr-tab )
        C!              ( -- pfa1 ptr-tab A+1 )
     NEXT               ( -- pfa1 ptr-tab A+n ) \ checar se em B o GP = 200
     2DROP ;            ( -- pfa1 )

: NEG-RAMP  ( pfa1 -- pfa1 )
     DUP 6 +                 ( -- pfa1 ptr-tab )
     PTO-C @            ( -- pfa1 ptr-tab C )
     PTO-D @            ( -- pfa1 ptr-tab C D )
     OVER - 1 -         ( -- pfa1 ptr-tab C D-C-2 )
     FOR                ( -- pfa1 ptr-tab C )
        1 + DUP         ( -- pfa1 ptr-tab C+1 C+1 )
        n>GPr-          ( -- pfa1 ptr-tab C+1 n1 )
        1 PICK 3 PICK +      ( -- pfa1 ptr-tab C+1 n1 C+1+ptr-tab )
        C!              ( -- pfa1 ptr-tab C+1 )
     NEXT               ( -- pfa1 ptr-tab C+n )
     2DROP ;            ( -- pfa1 )

\ pfa: da FVARIN a quem o membro pertence
\ pfa1: pfa do membro criado
\ converte os ptos para escala do sistena
```

```
: TRIANGLE  ( pfa pfa1 ptoA ptoB ptoC -- )
      4 PICK                 ( -- pfa pfa1 ptoA ptoB ptoC pfa )
      FIT PTO-D !       ( -- pfa pfa1 ptoA ptoB )
      3 PICK FIT DUP PTO-B ! PTO-C ! ( -- pfa pfa1 ptoA )
      2 PICK FIT PTO-A !     ( -- pfa pfa1 )
      DUP 6 +                ( -- pfa pfa1 ptr-tab ) \ aponta tab direto
\ pode ser eliminada a pfa aqui abaixo
      2 PICK 4 + @ 0    ( -- pfa pfa1 ptr-tab tam 0 ) \ tam em bytes
      FILL              ( -- pfa pfa1 )   \ preenche com zeros (gp)
      POS-RAMP          ( -- pfa pfa1 )   \ calcula rampa positiva
      NEG-RAMP          ( -- pfa pfa1 )   \ e rampa negativa
      2DROP ;

\ Usa a pfa apenas para extrair dados para o FIT.
: +SLOPE    ( pfa pfa1 PTO-A PTO-B -- )
      3 PICK            ( -- pfa pfa1 PTO-A PTO-B pfa )
      FIT PTO-B !       ( -- pfa pfa1 PTO-A )
      2 PICK            ( -- pfa pfa1 PTO-A pfa )
      FIT PTO-A !       ( -- pfa pfa1 )
      DUP 6 +           ( -- pfa pfa1 ptr-tab ) \ aponta tab member
      ROT               ( -- pfa1 ptr-tab pfa )
      4 + @             ( -- pfa1 pfa1+6 tam )
      0 FILL            ( -- pfa1 )
      POS-RAMP          ( -- pfa1 )
      DUP 6 +           ( -- pfa1 ptr_tab )
      PTO-B @ +         ( -- pfa1 ptoB -- )
      SWAP 4 + @        ( -- ptoB tam -- )
      PTO-B @ -         ( -- ptoB quant -- )
      200 FILL ;        ( -- )

\ Usa a pfa apenas para extrair dados para o FIT.
: -SLOPE    ( pfa pfa1 PTO-C PTO-D -- )
      3 PICK            ( -- pfa pfa1 PTO-C PTO-D pfa )
      FIT PTO-D !       ( -- pfa pfa1 PTO-C )
      2 PICK            ( -- pfa pfa1 PTO-C pfa )
      FIT PTO-C !       ( -- pfa pfa1 )
      DUP 6 +           ( -- pfa pfa1 ptr-tab ) \ aponta tab member
      ROT               ( -- pfa1 ptr-tab pfa )
      4 + @             ( -- pfa1 pfa1+6 tam )
      200 FILL          ( -- pfa1 )
      NEG-RAMP          ( -- pfa1 )
      DUP 6 +           ( -- pfa1 ptr_tab )
      PTO-D @ +         ( -- pfa1 ptoD -- )
      SWAP 4 + @        ( -- ptoD tam -- )
      PTO-D @ -         ( -- ptoD quant -- )
      0 FILL ;          ( -- )

\***************************************************
```

Anexo - C

Anexo contendo definições de palavras auxiliares para depuração e simulação, e a definição do loop de controle.

```
\ TOOLS.TXT

: DIST CR CR
." in: DISTANCIA.... " DISTANCIA @ . CR
." PERTO... " PERTO @ . 9 EMIT
." MEDIA... " MEDIA @ . 9 EMIT
." LONGE... " LONGE @ . ;

: ANG CR CR
." in: ANGULO....... " ANGULO @ . CR
." PEQUENO. : " PEQUENO @ . 9 EMIT
." GRANDE.. : " GRANDE @ . ;

: OUT CR CR
." VEL-BAIXA... " VEL-BAIXA @ . 9 EMIT
." VEL-MEDIA... " VEL-MEDIA @ . 9 EMIT
." VEL-ALTA.... " VEL-ALTA @ . CR
." out: VELOCIDADE.. " VELOCIDADE @ . ;

: >FIN DISTANCIA ! ANGULO ! FVAR ;
\ ****************************************
\ Definições para entrada de dados
: DGT>VAR
      DUP
      @
      10 * ROT +
      SWAP ! ;
\ VARIABLE ANGULO
\ VARIABLE DISTANCIA

: !ANGULO
      10 DIGIT?
      IF
            ANGULO
            DGT>VAR
      ELSE
            DROP
      THEN ;
: !DISTANCIA
      10 DIGIT?
      IF
            DISTANCIA
            DGT>VAR
      ELSE
            DROP
      THEN ;
: ANGULO?
      2 FOR
            BEGIN
                  ?RX
            UNTIL
      !ANGULO
      NEXT ;
: DISTANCIA?
```

```
     2 FOR
          BEGIN
               ?RX
          UNTIL
     !DISTANCIA
     NEXT ;

: >INPUT
     0 ANGULO !
     0 DISTANCIA !
     CR ." ANGULO:    " ANGULO? ANGULO @ .
     CR ." DISTANCIA: " DISTANCIA? DISTANCIA @ . ;
\ **************************************
\ Loop de controle
: CTRL
     BEGIN
     INICIALIZA
     >INPUT
     FUZZYFICA
     DIST ANG
     REGRA1 REGRA2 REGRA3 REGRA4
     DEFUZZYFICA
     OUT
     AGAIN ;
```

Anexo - D

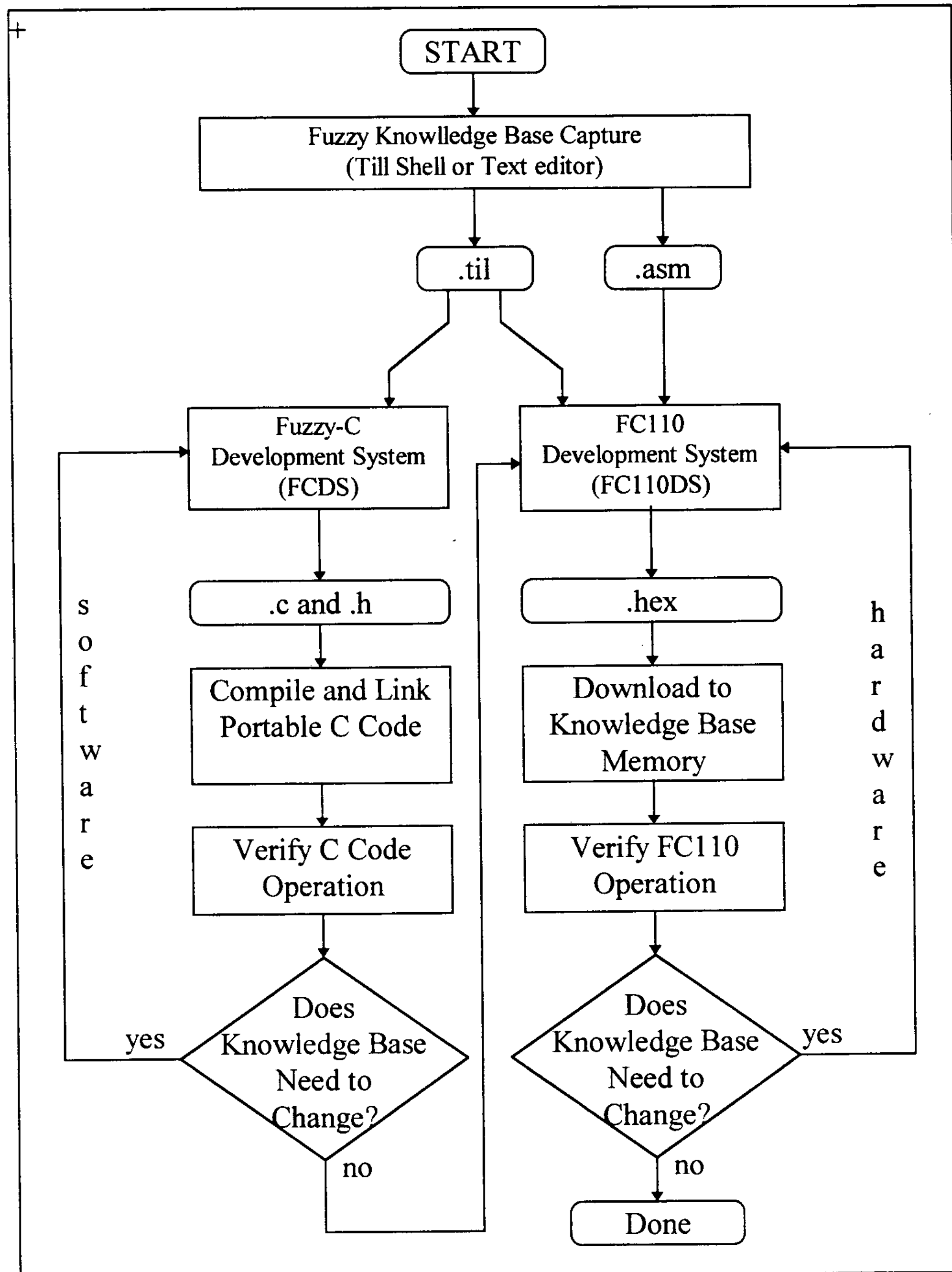
Anexo contendo as declarações da base de dados e base de regras.

```
\ DECLARA.TXT

FVARIN DISTANCIA 0 80 256 UNIVERS         \ 0 a 80 m
DISTANCIA MEMBER ZERO   0 5 -SLOPE
DISTANCIA MEMBER PERTO  0 5 10 TRIANGLE
DISTANCIA MEMBER MEDIA  5 15 40 TRIANGLE
DISTANCIA MEMBER LONGE  10 40 +SLOPE

FVARIN ANGULO 0 64 256 UNIVERS            \ 0 a 64 graus
ANGULO      MEMBER NULO 0 5 -SLOPE
ANGULO      MEMBER PEQUENO    0 10 50 TRIANGLE
ANGULO      MEMBER GRANDE     5 60 +SLOPE

FVAROUT VELOCIDADE
5  SINGLETON VEL-BAIXA
25 SINGLETON VEL-MEDIA
50 SINGLETON VEL-ALTA

: INICIALIZA
      0 VEL-BAIXA ! 0 VEL-MEDIA ! 0 VEL-ALTA ! ;

: FUZZYFICA
      MEMBERS[ ZERO PERTO MEDIA LONGE ]> DISTANCIA FUZZYFI
      MEMBERS[ NULO PEQUENO GRANDE ]> ANGULO FUZZYFI ;

: DEFUZZYFICA
      VELOCIDADE MEMBERS[ VEL-BAIXA VEL-MEDIA VEL-ALTA ]> DEFUZZYFI ;

: REGRA1 IFF ZERO THENF VEL-BAIXA END-RULE ;
: REGRA2 IFF PERTO ANDF NULO THENF VEL-BAIXA END-RULE ;
: REGRA3 IFF MEDIA ANDF NULO THENF VEL-MEDIA END-RULE ;
: REGRA4 IFF LONGE ANDF NULO THENF VEL-ALTA END-RULE ;
: REGRA5 IFF PERTO ANDF PEQUENO THENF VEL-MEDIA END-RULE ;
: REGRA6 IFF MEDIA ANDF PEQUENO THENF VEL-MEDIA END-RULE ;
: REGRA7 IFF LONGE ANDF PEQUENO THENF VEL-ALTA END-RULE ;
: REGRA8 IFF PERTO ANDF GRANDE THENF VEL-BAIXA END-RULE ;
: REGRA9 IFF MEDIA ANDF GRANDE THENF VEL-BAIXA END-RULE ;
: REGRA10 IFF LONGE ANDF GRANDE THENF VEL-MEDIA END-RULE ;
```

Anexo - E

O Sistema de desenvolvimento da Togai, e arquivos necessários para uma aplicação.

THE TOGAI INFRALOGIC PROGRAMMING ENVIROMENT



Arquivo demo.c:

```
/*
 * Togai InfraLogic (R) Fuzzy-C Compiler Version 2.4.0
 *
 * Source file: "demo.til"
 *
 * Compiled Sat Aug 01 01:58:05 1992
 *
 * Selected options:
 * Default map size:            256
 * Default map type:            FUBYTE
 * Output C format:             OFF
 * Emit unused MEMBERs:         ON
 * Force expr:                  OFF
 * Force map:                   ON
 * Generate debug code:         OFF
 */

/*
 * Portability definitions and internal structures.
 */

#include <stdio.h>
#include <string.h>
#include <tilcomp.h>
#include "demo.h"

#define min(a,b)  ((a)<(b)?(a):(b))
#define max(a,b)  ((a)<(b)?(b):(a))

struct _til_cbinfo { long moment, area; };
struct _til_cfinfo { double moment, area; };
struct _til_hbinfo { long hcentroid, height; };
struct _til_hfinfo { double hcentroid, height; };

FUBYTE _Theta_NM_map[] = {
/*00000*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00010*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00020*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00030*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00040*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00050*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00060*/ 255, 255, 255, 255, 255, 247, 239, 231, 223, 215,
/*00070*/ 207, 199, 191, 183, 175, 167, 159, 151, 143, 135,
/*00080*/ 127, 119, 111, 103,  95,  87,  79,  71,  63,  55,
/*00090*/  47,  39,  31,  23,  15,   7,   0,   0,   0,   0,
/*00100*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00110*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00120*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00130*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00140*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00150*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00160*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00170*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00180*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00190*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00200*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00210*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
/*00220*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00230*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00240*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00250*/   0,   0,   0,   0,   0,   0
};

FUBYTE _Theta_NS_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00050*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   0,   0,   0,   0,   7,  15,  23,  31,  39,
/*00070*/  47,  55,  63,  71,  79,  87,  95, 103, 111, 119,
/*00080*/ 127, 135, 143, 151, 159, 167, 175, 183, 191, 199,
/*00090*/ 207, 215, 223, 231, 239, 247, 255, 247, 239, 231,
/*00100*/ 223, 215, 207, 199, 191, 183, 175, 167, 159, 151,
/*00110*/ 143, 135, 127, 119, 111, 103,  95,  87,  79,  71,
/*00120*/  63,  55,  47,  39,  31,  23,  15,   7,   0,   0,
/*00130*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00140*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00150*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00160*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00170*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00180*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00190*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00200*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00210*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00220*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00230*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00240*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00250*/   0,   0,   0,   0,   0,   0
};

FUBYTE _Theta_Z_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00050*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00070*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00080*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00090*/   0,   0,   0,   0,   0,   0,   0,   7,  15,  23,
/*00100*/  31,  39,  47,  55,  63,  71,  79,  87,  95, 103,
/*00110*/ 111, 119, 127, 135, 143, 151, 159, 167, 175, 183,
/*00120*/ 191, 199, 207, 215, 223, 231, 239, 247, 255, 247,
/*00130*/ 239, 231, 223, 215, 207, 199, 191, 183, 175, 167,
/*00140*/ 159, 151, 143, 135, 127, 119, 111, 103,  95,  87,
/*00150*/  79,  71,  63,  55,  47,  39,  31,  23,  15,   7,
/*00160*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00170*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00180*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00190*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00200*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00210*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
/*00220*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00230*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00240*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00250*/   0,   0,   0,   0,   0,   0
};

FUBYTE _Theta_PS_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00050*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00070*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00080*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00090*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00100*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00110*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00120*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   7,
/*00130*/  15,  23,  31,  39,  47,  55,  63,  71,  79,  87,
/*00140*/  95, 103, 111, 119, 127, 135, 143, 151, 159, 167,
/*00150*/ 175, 183, 191, 199, 207, 215, 223, 231, 239, 247,
/*00160*/ 255, 247, 239, 231, 223, 215, 207, 199, 191, 183,
/*00170*/ 175, 167, 159, 151, 143, 135, 127, 119, 111, 103,
/*00180*/  95,  87,  79,  71,  63,  55,  47,  39,  31,  23,
/*00190*/  15,   7,   0,   0,   0,   0,   0,   0,   0,   0,
/*00200*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00210*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00220*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00230*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00240*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00250*/   0,   0,   0,   0,   0,   0
};

FUBYTE _Theta_PM_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00050*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00070*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00080*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00090*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00100*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00110*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00120*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00130*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00140*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00150*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00160*/   0,   7,  15,  23,  31,  39,  47,  55,  63,  71,
/*00170*/  79,  87,  95, 103, 111, 119, 127, 135, 143, 151,
/*00180*/ 159, 167, 175, 183, 191, 199, 207, 215, 223, 231,
/*00190*/ 239, 247, 255, 255, 255, 255, 255, 255, 255, 255,
/*00200*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00210*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
```

```
/*00220*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00230*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00240*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00250*/ 255, 255, 255, 255, 255, 255
};

FUBYTE _dTheta_NM_map[] = {
/*00000*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00010*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00020*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00030*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00040*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00050*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00060*/ 255, 255, 255, 255, 255, 247, 239, 231, 223, 215,
/*00070*/ 207, 199, 191, 183, 175, 167, 159, 151, 143, 135,
/*00080*/ 127, 119, 111, 103,  95,  87,  79,  71,  63,  55,
/*00090*/  47,  39,  31,  23,  15,   7,   0,   0,   0,   0,
/*00100*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00110*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00120*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00130*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00140*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00150*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00160*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00170*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00180*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00190*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00200*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00210*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00220*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00230*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00240*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00250*/   0,   0,   0,   0,   0,   0
};

FUBYTE _dTheta_NS_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00050*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   0,   0,   0,   0,   7,  15,  23,  31,  39,
/*00070*/  47,  55,  63,  71,  79,  87,  95, 103, 111, 119,
/*00080*/ 127, 135, 143, 151, 159, 167, 175, 183, 191, 199,
/*00090*/ 207, 215, 223, 231, 239, 247, 255, 247, 239, 231,
/*00100*/ 223, 215, 207, 199, 191, 183, 175, 167, 159, 151,
/*00110*/ 143, 135, 127, 119, 111, 103,  95,  87,  79,  71,
/*00120*/  63,  55,  47,  39,  31,  23,  15,   7,   0,   0,
/*00130*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00140*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00150*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00160*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00170*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00180*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00190*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00200*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00210*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
/*00220*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00230*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00240*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00250*/   0,   0,   0,   0,   0,   0
};

FUBYTE _dTheta_Z_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00050*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00070*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00080*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00090*/   0,   0,   0,   0,   0,   0,   0,   7,  15,  23,
/*00100*/  31,  39,  47,  55,  63,  71,  79,  87,  95, 103,
/*00110*/ 111, 119, 127, 135, 143, 151, 159, 167, 175, 183,
/*00120*/ 191, 199, 207, 215, 223, 231, 239, 247, 255, 247,
/*00130*/ 239, 231, 223, 215, 207, 199, 191, 183, 175, 167,
/*00140*/ 159, 151, 143, 135, 127, 119, 111, 103,  95,  87,
/*00150*/  79,  71,  63,  55,  47,  39,  31,  23,  15,   7,
/*00160*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00170*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00180*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00190*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00200*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00210*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00220*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00230*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00240*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00250*/   0,   0,   0,   0,   0,   0
};

FUBYTE _dTheta_PS_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00050*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00070*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00080*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00090*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00100*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00110*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00120*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   7,
/*00130*/  15,  23,  31,  39,  47,  55,  63,  71,  79,  87,
/*00140*/  95, 103, 111, 119, 127, 135, 143, 151, 159, 167,
/*00150*/ 175, 183, 191, 199, 207, 215, 223, 231, 239, 247,
/*00160*/ 255, 247, 239, 231, 223, 215, 207, 199, 191, 183,
/*00170*/ 175, 167, 159, 151, 143, 135, 127, 119, 111, 103,
/*00180*/  95,  87,  79,  71,  63,  55,  47,  39,  31,  23,
/*00190*/  15,   7,   0,   0,   0,   0,   0,   0,   0,   0,
/*00200*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00210*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
/*00220*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00230*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00240*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00250*/   0,   0,   0,   0,   0,   0
};

FUBYTE _dTheta_PM_map[] = {
/*00000*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00010*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00020*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00030*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00040*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00050*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00060*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00070*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00080*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00090*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00100*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00110*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00120*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00130*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00140*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00150*/   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
/*00160*/   0,   7,  15,  23,  31,  39,  47,  55,  63,  71,
/*00170*/  79,  87,  95, 103, 111, 119, 127, 135, 143, 151,
/*00180*/ 159, 167, 175, 183, 191, 199, 207, 215, 223, 231,
/*00190*/ 239, 247, 255, 255, 255, 255, 255, 255, 255, 255,
/*00200*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00210*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00220*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00230*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00240*/ 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
/*00250*/ 255, 255, 255, 255, 255, 255
};

/*
 * User defined objects.
 */

/*
 * VAR Theta
 */

/*
 * VAR dTheta
 */

/*
 * VAR Current
 */

/*
 * MEMBER NM
 */

/*
 *    moment: -6997.33
 *      area: 80
```

```
 *  centroid: -87.4667
 *    height: 1
 * hcentroid: -87.4667
 */

struct _til_cbinfo _Current_NM_cbinfo = { -3673209L, 41995L };

/*
 * MEMBER NS
 */

/*
 *    moment: -1024
 *      area: 32
 *  centroid: -32
 *    height: 1
 * hcentroid: -32
 */

struct _til_cbinfo _Current_NS_cbinfo = { -537542L, 16798L };

/*
 * MEMBER Z
 */

/*
 *    moment: 0
 *      area: 32
 *  centroid: 0
 *    height: 1
 * hcentroid: 0
 */

struct _til_cbinfo _Current_Z_cbinfo = { 0L, 16798L };

/*
 * MEMBER PS
 */

/*
 *    moment: 1024
 *      area: 32
 *  centroid: 32
 *    height: 1
 * hcentroid: 32
 */

struct _til_cbinfo _Current_PS_cbinfo = { 537542L, 16798L };

/*
 * MEMBER PM
 */

/*
 *    moment: 6869.83
 *      area: 79
 *  centroid: 86.9599
 *    height: 1
```

```
 * hcentroid: 86.9599
 */

struct _til_cbinfo _Current_PM_cbinfo = { 3606279L, 41470L };

/*
 * SOURCE dummy
 */

void dummy (void);

void dummy ()

{

}


/* Include some information from the graphical shell so that we */

/* can set some flags and communicate what's going on with the  */

/* Fuzzy-C interface.                                           */


#include "pendemo.h"

extern struct rulelistelem rulelist[10];

/*
 * FUZZY Pendulum
 */

void Pendulum (SBYTE Theta, SBYTE dTheta, SBYTE *Current)
{
    FUBYTE _alpha;
    FUBYTE _Theta_is_NS;
    FUBYTE _Theta_is_Z;
    FUBYTE _Theta_is_PS;
    FUBYTE _dTheta_is_NS;
    FUBYTE _dTheta_is_Z;
    FUBYTE _dTheta_is_PS;
    struct _til_cbinfo _Current_temp;

    memset (&_Current_temp, 0, sizeof (_Current_temp));

    _Theta_is_NS = _Theta_NS_map[ (int) (((SLONG) Theta) - -128L))];
    _Theta_is_Z = _Theta_Z_map[ (int) (((SLONG) Theta) - -128L))];
    _Theta_is_PS = _Theta_PS_map[ (int) (((SLONG) Theta) - -128L))];
    _dTheta_is_NS = _dTheta_NS_map[ (int) (((SLONG) dTheta) - -128L))];
    _dTheta_is_Z = _dTheta_Z_map[ (int) (((SLONG) dTheta) - -128L))];
    _dTheta_is_PS = _dTheta_PS_map[ (int) (((SLONG) dTheta) - -128L))];

/*
 * FRAGMENT Frag1a
 */
```

```
    if (!rulelist[PM_Z].disabled) {

/*
 * RULE Rule1
 */

    _alpha = min(_Theta_PM_map[ (int) ((((SLONG) Theta) - -128L))],
_dTheta_is_Z);

    if (_alpha != ((FUBYTE) 0)) {

        /* Current = NM */

        _Current_temp.moment += _alpha * _Current_NM_cbinfo.moment;
        _Current_temp.area += _alpha * _Current_NM_cbinfo.area;

    }

/*
 * FRAGMENT Frag1b
 */

      rulelist[PM_Z].DOM = _alpha;

    }

/*
 * FRAGMENT Frag2a
 */

    if (!rulelist[Z_NM].disabled) {

/*
 * RULE Rule2
 */

    _alpha = min(_Theta_is_Z, _dTheta_NM_map[ (int) ((((SLONG) dTheta) - -
128L))]);

    if (_alpha != ((FUBYTE) 0)) {

        /* Current = PM */

        _Current_temp.moment += _alpha * _Current_PM_cbinfo.moment;
        _Current_temp.area += _alpha * _Current_PM_cbinfo.area;

    }

/*
 * FRAGMENT Frag2b
 */

      rulelist[Z_NM].DOM = _alpha;

    }

/*
```

```
 * FRAGMENT Frag3a
 */

    if (!rulelist[Z_PM].disabled) {

/*
 * RULE Rule3
 */

    _alpha = min(_Theta_is_Z, _dTheta_PM_map[ (int) ((((SLONG) dTheta) - -
128L))]);

    if (_alpha != ((FUBYTE) 0)) {

        /* Current = NM */

        _Current_temp.moment += _alpha * _Current_NM_cbinfo.moment;
        _Current_temp.area += _alpha * _Current_NM_cbinfo.area;

    }

/*
 * FRAGMENT Frag3b
 */

      rulelist[Z_PM].DOM = _alpha;

      }

/*
 * FRAGMENT Frag4a
 */

    if (!rulelist[NM_Z].disabled) {


/*
 * RULE Rule4
 */

    _alpha = min(_Theta_NM_map[ (int) ((((SLONG) Theta) - -128L))],
_dTheta_is_Z);

    if (_alpha != ((FUBYTE) 0)) {

        /* Current = PM */

        _Current_temp.moment += _alpha * _Current_PM_cbinfo.moment;
        _Current_temp.area += _alpha * _Current_PM_cbinfo.area;

    }

/*
 * FRAGMENT Frag4b
 */

      rulelist[NM_Z].DOM = _alpha;
```

```
    }

/*
 * FRAGMENT Frag5a
 */

    if (!rulelist[PS_Z].disabled) {

/*
 * RULE Rule5
 */

    _alpha = min(_Theta_is_PS, _dTheta_is_Z);

    if (_alpha != ((FUBYTE) 0)) {

        /* Current = NS */

        _Current_temp.moment += _alpha * _Current_NS_cbinfo.moment;
        _Current_temp.area += _alpha * _Current_NS_cbinfo.area;

    }

/*
 * FRAGMENT Frag5b
 */

      rulelist[PS_Z].DOM = _alpha;

    }

/*
 * FRAGMENT Frag6a
 */

    if (!rulelist[Z_NS].disabled) {

/*
 * RULE Rule6
 */

    _alpha = min(_Theta_is_Z, _dTheta_is_NS);

    if (_alpha != ((FUBYTE) 0)) {

        /* Current = PS */

        _Current_temp.moment += _alpha * _Current_PS_cbinfo.moment;
        _Current_temp.area += _alpha * _Current_PS_cbinfo.area;

    }

/*
 * FRAGMENT Frag6b
 */

      rulelist[Z_NS].DOM = _alpha;
```

```
    }

/*
 * FRAGMENT Frag7a
 */

    if (!rulelist[NS_PS].disabled) {

/*
 * RULE Rule7
 */

    _alpha = min(_Theta_is_NS, _dTheta_is_PS);

    if (_alpha != ((FUBYTE) 0)) {

        /* Current = Z */

        _Current_temp.moment += _alpha * _Current_Z_cbinfo.moment;
        _Current_temp.area += _alpha * _Current_Z_cbinfo.area;

    }

/*
 * FRAGMENT Frag7b
 */

      rulelist[NS_PS].DOM = _alpha;

    }

/*
 * FRAGMENT Frag8a
 */

    if (!rulelist[Z_PS].disabled) {

/*
 * RULE Rule8
 */

    _alpha = min(_Theta_is_Z, _dTheta_is_PS);

    if (_alpha != ((FUBYTE) 0)) {

        /* Current = NS */

        _Current_temp.moment += _alpha * _Current_NS_cbinfo.moment;
        _Current_temp.area += _alpha * _Current_NS_cbinfo.area;

    }

/*
 * FRAGMENT Frag8b
 */

      rulelist[Z_PS].DOM = _alpha;
```

```
    }

/*
 * FRAGMENT Frag9a
 */

    if (!rulelist[NS_Z].disabled) {

/*
 * RULE Rule9
 */

    _alpha = min(_Theta_is_NS, _dTheta_is_Z);

    if (_alpha != ((FUBYTE) 0)) {

        /* Current = PS */

        _Current_temp.moment += _alpha * _Current_PS_cbinfo.moment;
        _Current_temp.area += _alpha * _Current_PS_cbinfo.area;

    }

/*
 * FRAGMENT Frag9b
 */

      rulelist[NS_Z].DOM = _alpha;

    }

/*
 * FRAGMENT Frag10a
 */

    if (!rulelist[PS_NS].disabled) {

/*
 * RULE Rule10
 */

    _alpha = min(_Theta_is_PS, _dTheta_is_NS);

    if (_alpha != ((FUBYTE) 0)) {

        /* Current = Z */

        _Current_temp.moment += _alpha * _Current_Z_cbinfo.moment;
        _Current_temp.area += _alpha * _Current_Z_cbinfo.area;

    }

/*
 * FRAGMENT Frag10b
 */

      rulelist[PS_NS].DOM = _alpha;
```

```
    }

/*
 * FRAGMENT Frag11a
 */

    if (!rulelist[Z_Z].disabled) {

/*
 * RULE Rule11
 */

    _alpha = min(_Theta_is_Z, _dTheta_is_Z);

    if (_alpha != ((FUBYTE) 0)) {

        /* Current = Z */

        _Current_temp.moment += _alpha * _Current_Z_cbinfo.moment;
        _Current_temp.area += _alpha * _Current_Z_cbinfo.area;

    }

/*
 * FRAGMENT Frag11b
 */

      rulelist[Z_Z].DOM =_alpha;

    }


    if (_Current_temp.area != 0L)
        *Current = ((SBYTE) (_Current_temp.moment / _Current_temp.area));
    else
        *Current = ((SBYTE) 0);

}

/*
 * PROJECT Pend
 */

void Pend (SBYTE Theta, SBYTE dTheta, SBYTE *Current)
{
/*
 * SOURCE dummy
 */

    dummy ();

/*
 * FUZZY Pendulum
 */

    Pendulum (Theta, dTheta, Current);

}
```

Arquivo demo.h

```
/*
 * Togai InfraLogic (R) Fuzzy-C Compiler Version 2.4.0
 *
 * Source file: "demo.til"
 *
 * Compiled Sat Aug 01 01:58:05 1992
 *
 * Selected options:
 * Default map size:            256
 * Default map type:            FUBYTE
 * Output C format:             OFF
 * Emit unused MEMBERs:         ON
 * Force expr:                  OFF
 * Force map:                   ON
 * Generate debug code:         OFF
 */

extern FUBYTE _Theta_NM_map[];
extern FUBYTE _Theta_NS_map[];
extern FUBYTE _Theta_Z_map[];
extern FUBYTE _Theta_PS_map[];
extern FUBYTE _Theta_PM_map[];
extern FUBYTE _dTheta_NM_map[];
extern FUBYTE _dTheta_NS_map[];
extern FUBYTE _dTheta_Z_map[];
extern FUBYTE _dTheta_PS_map[];
extern FUBYTE _dTheta_PM_map[];
extern struct _til_cbinfo _Current_NM_cbinfo;
extern struct _til_cbinfo _Current_NS_cbinfo;
extern struct _til_cbinfo _Current_Z_cbinfo;
extern struct _til_cbinfo _Current_PS_cbinfo;
extern struct _til_cbinfo _Current_PM_cbinfo;
void dummy (void);

void Pendulum (SBYTE Theta, SBYTE dTheta, SBYTE *Current);

void Pend (SBYTE Theta, SBYTE dTheta, SBYTE *Current);
```

Arquivo demo.til:

```
/*
 * Togai InfraLogic Fuzzy-C Development System Demonstration File: demo.til
 * $Header:    E:/vcs/fcds/demo.tiv   1.2   28 Jan 1990 22:03:28   jst  $
 *
 * This is the input file for the Fuzzy-C Compiler.  It describes the
 * fuzzy logic knowledge base needed to control the inverted pendulum
 * in an abstract manner.  The system takes two inputs (Theta and dTheta)
 * and produces one output (Current).  The fuzzy logic rules in this
 * description take the two inputs and determine what the crisp value for
 * Current should be.
 *
 * The Fuzzy-C Compiler generates the C source code necessary to implement
```

```
 * the inference engine.
 *
 * Copyright (c) Togai InfraLogic, Inc. 1989, 1990.
 * All rights reserved.
 */

PROJECT Pend

    /*
     * Input definition for Theta.  This variable has the associated
     * membership functions: NM, NS, Z, PS, and PM.  A membership function
     * is defined as a set of ordered pairs that form a piewise-liner
     * shape.  A membership function can also be defined non-linearly
     * by a user specified equation, but this is not shown.
     */

    VAR Theta
      TYPE signed byte

        MEMBER NM
          POINTS -128,1.0 -64,1.0 -32,0.0
        END

        MEMBER NS
          POINTS -64,0.0 -32,1.0 0,0.0
        END

        MEMBER Z
          POINTS -32,0.0 0,1.0 32,0.0
        END

        MEMBER PS
          POINTS 0,0.0 32,1.0 64,0.0
        END

        MEMBER PM
          POINTS 32,0.0 64,1.0 127,1.0
        END
    END

    /*
     * Input definition for dTheta.  This variable has the associated
     * membership functions: NM, NS, Z, PS, and PM.
     */

    VAR dTheta
      TYPE signed byte

        MEMBER NM
          POINTS -128,1.0 -64,1.0 -32,0.0
        END

        MEMBER NS
          POINTS -64,0.0 -32,1.0 0,0.0
        END

        MEMBER Z
          POINTS -32,0.0 0,1.0 32,0.0
```

```
        END

        MEMBER PS
          POINTS 0,0.0 32,1.0 64,0.0
        END

        MEMBER PM
          POINTS 32,0.0 64,1.0 127,1.0
        END
    END

    /*
     * Output definition for Current.  This variable has the associated
     * membership functions: NM, NS, Z, PS, and PM.
     */

    VAR Current

      TYPE signed byte
      DEFAULT 0

        MEMBER NM
          POINTS -128,1.0 -64,1.0 -32,0.0
        END

        MEMBER NS
          POINTS -64,0.0 -32,1.0 0,0.0
        END

        MEMBER Z
          POINTS -32,0.0 0,1.0 32,0.0
        END

        MEMBER PS
          POINTS 0,0.0 32,1.0 64,0.0
        END

        MEMBER PM
          POINTS 32,0.0 64,1.0 127,1.0
        END
    END

    SOURCE dummy
#CODE
void dummy (void);
void dummy ()
{
}

/* Include some information from the graphical shell so that we */
/* can set some flags and communicate what's going on with the  */
/* Fuzzy-C interface.                                           */

#include "pendemo.h"
extern struct rulelistelem rulelist[10];

#END_CODE
    END
```

```
/*
 * This is where the fuzzy logic rules are defined.  Each defiend rule
 * is surrounded with some C code.  The C code allows rules to be
 * individually disabled from the demonstration.
 */

    FUZZY Pendulum

      OPTIONS
          OUTPUTSCOPE="Public"
      END


/*
 * Rule1
 */

      FRAGMENT Frag1a
#CODE
    if (!rulelist[PM_Z].disabled) {
#END_CODE
        END

      RULE Rule1
            IF (Theta IS PM) AND (dTheta IS Z) THEN
            Current=NM
        END

      FRAGMENT Frag1b
#CODE
      rulelist[PM_Z].DOM = _alpha;
    }
#END_CODE
        END

/*
 * Rule2
 */

      FRAGMENT Frag2a
#CODE
    if (!rulelist[Z_NM].disabled) {
#END_CODE
        END

      RULE Rule2
            IF (Theta IS Z) AND (dTheta IS NM) THEN
                Current=PM
        END

      FRAGMENT Frag2b
#CODE
      rulelist[Z_NM].DOM = _alpha;
    }
#END_CODE
        END
```

```
/*
 * Rule3
 */

      FRAGMENT Frag3a
#CODE
    if (!rulelist[Z_PM].disabled) {
#END_CODE
        END

      RULE Rule3
            IF (Theta IS Z) AND (dTheta IS PM) THEN
                Current=NM
        END

      FRAGMENT Frag3b
#CODE
      rulelist[Z_PM].DOM = _alpha;
      }
#END_CODE
        END

/*
 * Rule4
 */

      FRAGMENT Frag4a
#CODE
    if (!rulelist[NM_Z].disabled) {
#END_CODE
        END

      RULE Rule4
            IF (Theta IS NM) AND (dTheta IS Z) THEN
            Current=PM
        END

      FRAGMENT Frag4b
#CODE
      rulelist[NM_Z].DOM = _alpha;
    }
#END_CODE
        END

/*
 * Rule5
 */

      FRAGMENT Frag5a
#CODE
    if (!rulelist[PS_Z].disabled) {
#END_CODE
        END

      RULE Rule5
            IF (Theta IS PS) AND (dTheta IS Z) THEN
            Current=NS
```

```
        END

      FRAGMENT Frag5b
#CODE
      rulelist[PS_Z].DOM = _alpha;
    }
#END_CODE
        END

/*
 * Rule6
 */

      FRAGMENT Frag6a
#CODE
    if (!rulelist[Z_NS].disabled) {
#END_CODE
        END

      RULE Rule6
            IF (Theta IS Z) AND (dTheta IS NS) THEN
                Current=PS
        END

      FRAGMENT Frag6b
#CODE
      rulelist[Z_NS].DOM = _alpha;
    }
#END_CODE
        END

/*
 * Rule7
 */

      FRAGMENT Frag7a
#CODE
    if (!rulelist[NS_PS].disabled) {
#END_CODE
        END

      RULE Rule7
          IF (Theta IS NS) AND (dTheta IS PS) THEN
            Current=Z
        END

      FRAGMENT Frag7b
#CODE
      rulelist[NS_PS].DOM = _alpha;
    }
#END_CODE
        END

/*
 * Rule8
 */

      FRAGMENT Frag8a
```

```
#CODE
    if (!rulelist[Z_PS].disabled) {
#END_CODE
        END

      RULE Rule8
            IF (Theta IS Z) AND (dTheta IS PS) THEN
                Current=NS
        END

      FRAGMENT Frag8b
#CODE
      rulelist[Z_PS].DOM = _alpha;
    }
#END_CODE
        END

/*
 * Rule9
 */

      FRAGMENT Frag9a
#CODE
    if (!rulelist[NS_Z].disabled) {
#END_CODE
        END

      RULE Rule9
            IF (Theta IS NS) AND (dTheta IS Z) THEN
            Current=PS
        END

      FRAGMENT Frag9b
#CODE
      rulelist[NS_Z].DOM = _alpha;
    }
#END_CODE
        END

/*
 * Rule10
 */

      FRAGMENT Frag10a
#CODE
    if (!rulelist[PS_NS].disabled) {
#END_CODE
        END

      RULE Rule10
          IF (Theta IS PS) AND (dTheta IS NS) THEN
            Current=Z
        END

      FRAGMENT Frag10b
#CODE
      rulelist[PS_NS].DOM = _alpha;
    }
```

```
#END_CODE
        END

/*
 * Rule11
 */

      FRAGMENT Frag11a
#CODE
    if (!rulelist[Z_Z].disabled) {
#END_CODE
        END


      RULE Rule11
            IF (Theta IS Z) AND (dTheta IS Z) THEN
                Current=Z
      END


      FRAGMENT Frag11b
#CODE
      rulelist[Z_Z].DOM =_alpha;
    }
#END_CODE
        END
    END

/*
 * The following CONNECT statements define that both Theta and dTheta
 * are inputs to the Pendulum rulebase and that Current is an output.
 */

    CONNECT
        FROM Theta
        TO Pendulum
    END

    CONNECT
        FROM dTheta
        TO Pendulum
    END

    CONNECT
        FROM Pendulum
        TO Current
    END
END
```

Arquivo demo110.lnk:

```
map demo110
entry 0 Pend_FC110
search demo110
cfile demo110
hfile demo110
write demo110
```

quit

Arquivo demo110.til:

```
/*
 * Togai InfraLogic Fuzzy-C Development System Demonstration File: demo.til
 * $Header:   E:/vcs/common/demo110.tiv   1.0   02 Mar 1990 16:52:06   eah  $
 *
 * This is the input file for the Fuzzy-C Compiler.  It describes the
 * fuzzy logic knowledge base needed to control the inverted pendulum
 * in an abstract manner.  The system takes two inputs (Theta and dTheta)
 * and produces one output (Current).  The fuzzy logic rules in this
 * description take the two inputs and determine what the crisp value for
 * Current should be.
 *
 * The Fuzzy-C Compiler generates the C source code necessary to implement
 * the inference engine.
 *
 * Copyright (c) Togai InfraLogic, Inc. 1989, 1990.
 * All rights reserved.
 */

PROJECT Pend_FC110

    /*
     * Input definition for Theta.  This variable has the associated
     * membership functions: NM, NS, Z, PS, and PM.  A membership function
     * is defined as a set of ordered pairs that form a piewise-liner
     * shape.  A membership function can also be defined non-linearly
     * by a user specified equation, but this is not shown.
     */

    VAR Theta
      TYPE signed byte

        MEMBER NM
          POINTS -128,1.0 -64,1.0 -32,0.0
        END

        MEMBER NS
          POINTS -64,0.0 -32,1.0 0,0.0
        END

        MEMBER Z
          POINTS -32,0.0 0,1.0 32,0.0
        END

        MEMBER PS
          POINTS 0,0.0 32,1.0 64,0.0
        END

        MEMBER PM
          POINTS 32,0.0 64,1.0 127,1.0
        END
    END

    /*
```

```
 * Input definition for dTheta.  This variable has the associated
 * membership functions: NM, NS, Z, PS, and PM.
 */

VAR dTheta
  TYPE signed byte

    MEMBER NM
      POINTS -128,1.0 -64,1.0 -32,0.0
    END

    MEMBER NS
      POINTS -64,0.0 -32,1.0 0,0.0
    END

    MEMBER Z
      POINTS -32,0.0 0,1.0 32,0.0
    END

    MEMBER PS
      POINTS 0,0.0 32,1.0 64,0.0
    END

    MEMBER PM
      POINTS 32,0.0 64,1.0 127,1.0
    END
END

/*
 * Output definition for Current.  This variable has the associated
 * membership functions: NM, NS, Z, PS, and PM.
 */

VAR Current

  TYPE signed byte
  DEFAULT 0

    MEMBER NM
      POINTS -128,1.0 -64,1.0 -32,0.0
    END

    MEMBER NS
      POINTS -64,0.0 -32,1.0 0,0.0
    END

    MEMBER Z
      POINTS -32,0.0 0,1.0 32,0.0
    END

    MEMBER PS
      POINTS 0,0.0 32,1.0 64,0.0
    END

    MEMBER PM
      POINTS 32,0.0 64,1.0 127,1.0
    END
END
```

```
/*
 * This is where the fuzzy logic rules are defined.  Each defiend rule
 * is surrounded with some C code.  The C code allows rules to be
 * individually disabled from the demonstration.
 *
 * DO NOT CHANGE THE ORDER OF THE RULES OR ADD OR REMOVE RULES WITH CHANGING
 * THE ROUTINE get_alphas IN PENDEMO.C!!!!
 */

    FUZZY Pendulum_FC110

      RULE Rule1
            IF (Theta IS PM) AND (dTheta IS Z) THEN
            Current=NM
        END

      RULE Rule2
            IF (Theta IS Z) AND (dTheta IS NM) THEN
                Current=PM
        END

      RULE Rule3
            IF (Theta IS Z) AND (dTheta IS PM) THEN
                Current=NM
        END

      RULE Rule4
            IF (Theta IS NM) AND (dTheta IS Z) THEN
            Current=PM
        END

      RULE Rule5
            IF (Theta IS PS) AND (dTheta IS Z) THEN
            Current=NS
        END

      RULE Rule6
            IF (Theta IS Z) AND (dTheta IS NS) THEN
                Current=PS
        END

      RULE Rule7
          IF (Theta IS NS) AND (dTheta IS PS) THEN
            Current=Z
        END

      RULE Rule8
            IF (Theta IS Z) AND (dTheta IS PS) THEN
                Current=NS
        END

      RULE Rule9
            IF (Theta IS NS) AND (dTheta IS Z) THEN
            Current=PS
        END

      RULE Rule10
```

```
          IF (Theta IS PS) AND (dTheta IS NS) THEN
            Current=Z
        END

      RULE Rule11
            IF (Theta IS Z) AND (dTheta IS Z) THEN
                Current=Z
      END
    END

/*
 * The following CONNECT statements define that both Theta and dTheta
 * are inputs to the Pendulum rulebase and that Current is an output.
 */

    CONNECT
        FROM Theta
      TO Pendulum_FC110
    END

    CONNECT
        FROM dTheta
      TO Pendulum_FC110
    END

    CONNECT
      FROM Pendulum_FC110
        TO Current
    END
END
```

Anexo - F

[1]O μPD70320L-8 (V25™)é um microcomputador single-chip, software compatível com o μPD70108/μPD70116 (V20™/V30™). Embora mantenha a compatibilidade de software, suas funçõs de interrupção, que são essencialmente para aplicações de controle, são significantemente superiores. Consequentemente, o processamento de interrupções é muito potente e de alta velocidade. Adicionalmente, um interval timer, interface serial, controlador de DMA, e outros dispositivos, são integrados em um single-chip.

O μPD70320L-8 é um dispositivo de tecnologia CMOS, portanto de baixa potencia de consumo. Tambem possui modos de economia de energia como o modo stanby, que possibilita a retenção de dados com grande redução no consumo de energia.

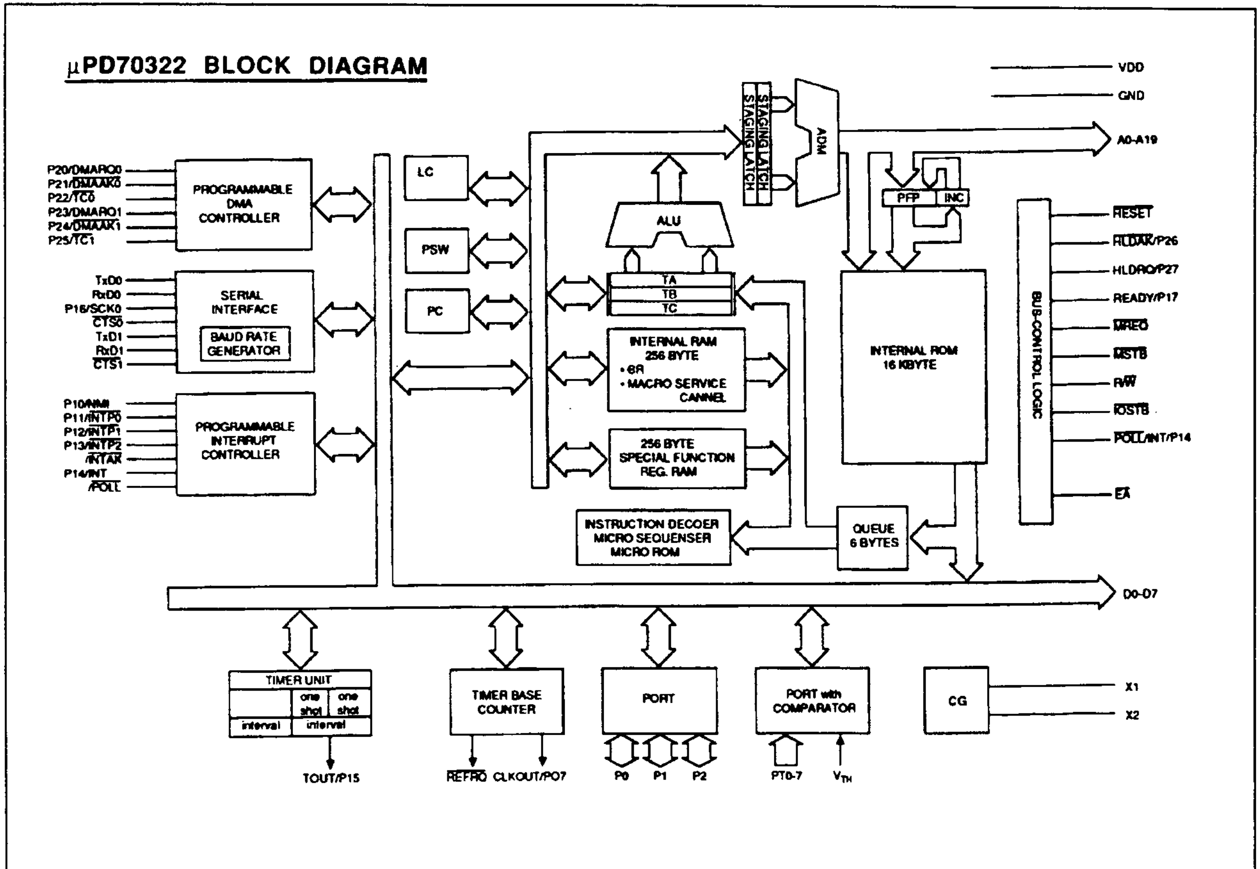
O μPD70320L-8 é ideal para aplicações onde grande volume de dados de controle de outros equipamentos devem ser processados, ou em processadores de texto portáteis, microcomputadores portáteis, terminais remotos, etc.

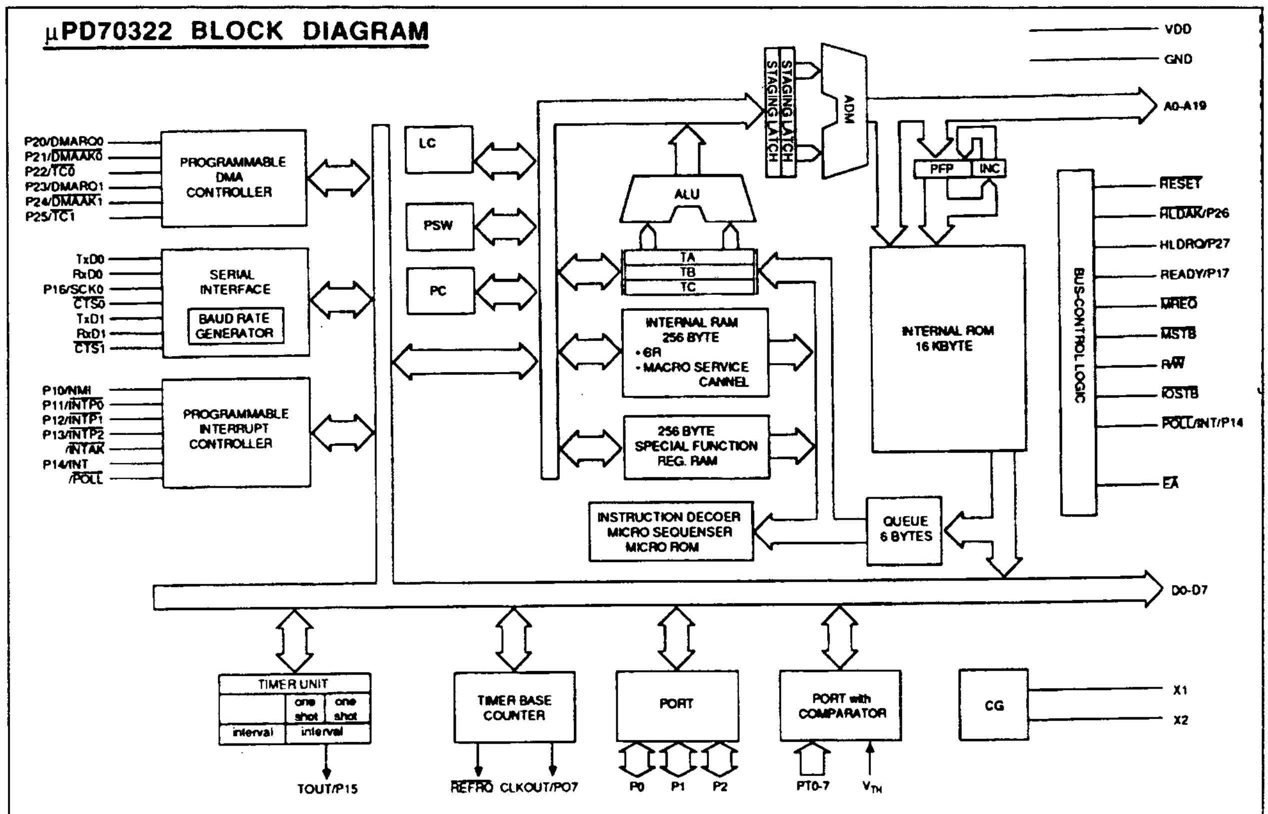
O μPD70322 é identico ao μPD70320L-8, possuindo ROM interna de 16kb para o armazenamento de programas em aplicações que requeiram alta velocidade de processamento como Kernels de sistemas operacionais.

As figuras a seguir mostram a estrutura interna do μPD70320L/ μPD70322, e a sua pinagem.

---

[1]Fonte: NEC User's Manual, V25/V35 Family V-Series 16-bit Microcomputers


µPD70322 BLOCK DIAGRAM
VDD
GND
A0-A19
STAGING LATCH
STAGING LATCH
ADM
PFP
INC
P20/DMARQ0
P21/DMAAK0
P22/TC0
P23/DMARQ1
P24/DMAAK1
P25/TC1
PROGRAMMABLE DMA CONTROLLER
TxD0
RxD0
P16/SCK0
CTS0
TxD1
RxD1
CTS1
SERIAL INTERFACE
BAUD RATE GENERATOR
P10/NMI
P11/INTP0
P12/INTP1
P13/INTP2
/INTAK
P14/INT
/POLL
PROGRAMMABLE INTERRUPT CONTROLLER
LC
PSW
PC
ALU
TA
TB
TC
INTERNAL RAM 256 BYTE
• GR
• MACRO SERVICE CANNEL
256 BYTE SPECIAL FUNCTION REG. RAM
INTERNAL ROM 16 KBYTE
BUS-CONTROL LOGIC
RESET
HLDAK/P26
HLDRQ/P27
READY/P17
MREQ
MSTB
R/W
IOSTB
POLL/INT/P14
EA
INSTRUCTION DECOER
MICRO SEQUENSER
MICRO ROM
QUEUE 6 BYTES
D0-D7
TIMER UNIT
one shot
one shot
interval
interval
TOUT/P15
TIMER BASE COUNTER
REFRQ
CLKOUT/PO7
PORT
P0
P1
P2
PORT with COMPARATOR
PT0-7
V_{TH}
CG
X1
X2


µPD70322 BLOCK DIAGRAM
VDD
GND
A0-A19
STAGING LATCH
STAGING LATCH
ADM
PFP
INC
P20/DMARQ0
P21/DMAAK0
P22/TC0
P23/DMARQ1
P24/DMAAK1
P25/TC1
PROGRAMMABLE DMA CONTROLLER
TxD0
RxD0
P16/SCK0
CTS0
TxD1
RxD1
CTS1
SERIAL INTERFACE
BAUD RATE GENERATOR
P10/NMI
P11/INTP0
P12/INTP1
P13/INTP2
/INTAK
P14/INT
/POLL
PROGRAMMABLE INTERRUPT CONTROLLER
LC
PSW
PC
ALU
TA
TB
TC
INTERNAL RAM 256 BYTE
• GR
• MACRO SERVICE CANNEL
256 BYTE SPECIAL FUNCTION REG. RAM
INTERNAL ROM 16 KBYTE
BUS-CONTROL LOGIC
RESET
HLDAK/P26
HLDRQ/P27
READY/P17
MREQ
MSTB
R/W
IOSTB
POLL/INT/P14
EA
INSTRUCTION DECOER
MICRO SEQUENSER
MICRO ROM
QUEUE 6 BYTES
D0-D7
TIMER UNIT
one shot
one shot
interval
interval
TOUT/P15
TIMER BASE COUNTER
REFRQ
CLKOUT/PO7
PORT
P0
P1
P2
PORT with COMPARATOR
PT0-7
V_{TH}
CG
X1
X2

Bibliografia

Allworth, S. T., e Zobel, R. N., *Introduction to Real-Time software design,* Macmillan,1987.

Loeliger, R. G., *Threaded Interpretive Languages,* Byte Books,1981.

Brodie, L., *Starting FORTH,* Prentice-Hall,1987.

Tzou, S.Y., Lim, J.J., Menon, J., e Palmer, D., *A Distributed Development Environment for Embedded Software.* Software-Practice and Experience, Vol. 23(11), 1235-1248, novembro, 1993.

Bennett, S., Linkens, D.A., *Real-Time Computer Control,* Peter Peregrinus Ltd, 1984

Fuzzy FAQ[1], *USENET News Groups,* InterNet, 1995.

Gomide, F., Gudwin, R., Andrade Netto, M.L., *Controle de Processos por Lógica Difusa,* Unicamp - FEE - DCA, 1993.

Lee, C. C., *Fuzzy Logic in Control Systems: Fuzzy Logic Controller,* Part I, IEEE Transactions on Control Systems, março/abril 1990.

Phillips, C.L., e Harbor, R.D., *Feedback Control Systems,* Prentice-Hall, 1991.

FuzzyFan , *USENET News Groups,* InterNet, 1994.[2]

Viot, G., *Fuzzy Logic in C,* Dr. Dobb's Journal, Fevereiro, 1993.

Togai InfraLogic FC110DS User's Manual, *Togai InfraLogic, Inc.,* 1991.

Altrock, C.V., *Fuzzy Logik,* Oldenbourg Verlag GmbH, 1993.

VPORT-25k Technisches Handbuch; *Taskit Rechnertechnik,* Berlin, 1989.

---

[1] *Frequently Asked Questions*

[2] trata-se de um programa exemplo que está na Internet. Não sei se devo colocá-lo!?

Wilson,R., Malinowski,C. W., Pawlowicz,H., *Real Time Executives*, Computer Design, fevereiro, 1989.

Payne, W. H.; *Embedded Controller Forth*, Academic Press Inc., 1990.

Hendtlass, T.; *Real Time Forth*, Swinburne University of Technology, Austrália, 1993.

Knagss, P. J.; *Pratical and Theoretical Aspects of Forth Software Development*, Tese de doutorado, Universidade de Tesside,UK, 1993.

Kogge, P. M., *an Architectural Trail to Threaded-COde Systems*, IEEE Computer, 1982.

Brown, J. F., *Embedded Systems Programming in C and Assembly*, Van Nostrand Reinhold, New York, 1994.

Tan, J. e Chang, Z., *Linearity and a Tunning Procedure for Fuzzy Logic Controllers*, Transactions of the ASAE, Vol. 37(3), pág. 973-979.

Wang, L.X., *A Supervisory Controller for Fuzzy Control Systems that Guarantees Stability*, IEEE Transactions on Automatic Control, vol. 39, número 9, setembro, 1994.

Sugeno, M., *An Introductory Survey of Fuzzy Control*, Information Sciences, vol. 36, pág. 59-83, Elsevier Science Publishing Co, Inc, New York, 1985.

Li, Y. F. e Lau, C. C., *Development of Fuzzy Algorithms for Servo Systems*, IEEE Control Systems Magazine, abril, 1989.

Sibigtroth, J. M., *Implementing Fuzzy Expert Rules in Hardware*, AI Expert, abril, 1992.

Rudi,, R., *Fuzzy in Control*, Sensor Review, vol. 14, número 3, MCB University Press, 1994.

Williams, T., Togai, M., Brubaker, D. L., *Fuzzy Logic Simplifies Complex Control Problems*, Computer Design, março, 1991.

Runkler, T., Halgamuge, S., Kothe, R., e Glesner, M., *Volle Pulle im Rückwärtsgang*, Elektronik, 15, 1994.

V25/V35 Family, V-Series, 16-bit Microcomputers, *User's Manual*, NEC Electronics, 1989.

Ting, C. H., *eFORTH Implementation Guide*, Offete Enterprise, Inc., 1990.

Coad, P., Yourdon, E., *Análise Baseada em Objetos*, Campus, 1992.

Coad, P., Yourdon, E., *Projeto Baseado em Objetos*, Campus, 1993.

von Staa, A., *Engenharia de Programas*, LTC, 1983.

Holzner, S., *Linguagem Assembly Avançada para IBM PC*, McGraw-Hill, 1990.

Stevens, W. P., *Projeto Estruturado de Sistemas*, Campus, 1988.

Guimarães, A. M., *Algoritmos e Estruturas de Dados*, LTC, 1985.

Duncan, R., *MS-DOS Avançado*, McGRaw-Hill, 1991.

System BIOS for IBM PC/XT/AT Computers and Compatibles, *Phoenix Technologies Ltd.*, 1989.

Robert, G.F., e Kemerer, C., *Object-Oriented and Conventional Analysis and Design Methodologies*, IEEE Computer, outobro, 1992.

Kosko, B., *Thinking Fuzzy*, ????.

Woehr, J. J., *Born to ROM*, Embedded Systems Programming, janeiro, 1992.

Woehr, J. J., *Forth Multitasker*, Dr. Dobb's Journal, junho, 1991.

Giarratano, J.C., *Expert Systems: Principles and Programming*, PWS-KENT Publishing Company, 1989.

comp.realtime, *USENET News Group*, InterNet, 1995.

Kaushik, G., Mukherjee, B., e Schwan, K., *A Survey of Real-Time Operating Systems*, Georgia Institute of Technology, College of Computing, InterNet, fevereiro, 1993.

Badr, S.M., Byrnes Jr, R.B., Brutzman, D.P., e Nelson, M.L., *Real-Time Systems*, Naval Postgraduate Scholl, Monterey, CA (USA), InterNet, 1992.

comp.lang.forth, *USENET News Group*, InterNet, 1995.

Guimarães, A. M., *Algoritmos e Estruturas de Dados*, LTC, 1985.

Gopal, M., *Modern Control Systems Theory*, Wiley Eastern Limited, 1984.

Zadeh, L. A., *Fuzzy Logic, Neural Networks, and Soft Computing*, Communications of the ACM, volume 37, número 3, março, 1994.

FORTH, Inc.'s Home Page, *World Wide Web*, Internet, 1995.

Cox, E., *The fuzzy systems handbook: a practitioner's guide to building, using, and maintaining fuzzy systems*, Academic Press, Inc., 1994.

Gauthier, F.A.O., Programação de Produção: uma abordagem utilizando algoritmos genéticos, Tese de doutorado, Universidade Federal de Santa Catarina, 1993.