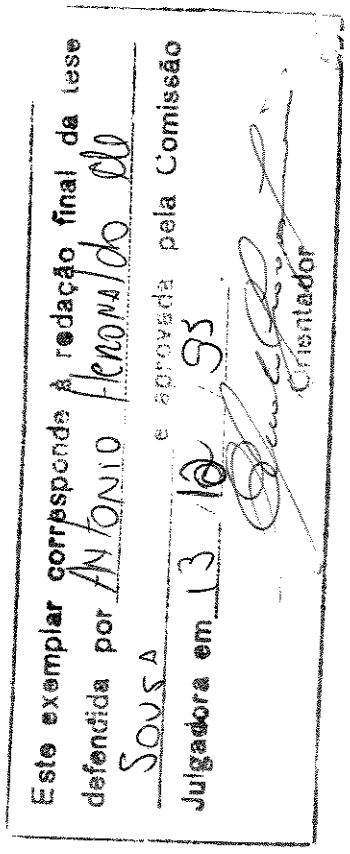
UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ELETRÔNICA E MICROELETRÔNICA

ONAGRO - Um Ambiente Gráfico para Desenvolvimento de Software para Microcontroladores

Antonio Heronaldo de Sousa



ORIENTADOR: Prof. Dr. Elnatan Chagas Ferreira

Tese apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Elétrica.

Dezembro - 1995

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

So85o

Sousa, Antonio Heronaldo de
ONAGRO - um ambiente gráfico para desenvolvimento de software para microcontroladores / Antonio Heronaldo de Sousa.--Campinas, SP: [s.n.], 1995.

Orientador: Elnatan Chagas Ferreira.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica.

1. Linguagem de programação (Computadores). 2. Compiladores (Computadores). 3. Compiladores (Programas de computador). 4. Microcomputadores - Programação. I. Ferreira, Elnatan Chagas. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica. III. Título.

À minha esposa Cláudia, que está esperando nosso primeiro filho, pelo constante incentivo e pela compreensão da minha ausência.

Agradecimentos:

Ao Prof. Elnatan, pela orientação e incentivo.

Ao Prof. Oséas, que me proporcionou a entrada na FEE-UNICAMP.

Ao Gustavo, pela grandiosa colaboração na parte gráfica deste trabalho.

À Ademilde, pelas dicas de redação.

Ao Marcos, pelo apoio e sugestões ao trabalho.

Ao Dep. de Eng. Elétrica da FEJ-UDESC, pela confiança em mim depositada.

À CAPES, pelo apoio financeiro.

Às demais pessoas que contribuíram para a realização deste trabalho.

RESUMO

O ONAGRO é um sistema tradutor que reconhece uma linguagem gráfica de descrição de algoritmos e possibilita a geração de código em Assembly para microcontroladores. Além do tradutor propriamente dito, ele incorpora um editor gráfico para a entrada do programa-fonte, que se assemelha a um algoritmo descrito em linguagem de fluxogramas. Ele também possui um editor de identificadores que permite a descrição dos símbolos identificadores usados no programa.

O sistema ONAGRO opera em ambiente Microsoft Windows, oferecendo uma interface amigável com o usuário. Esta interface baseia-se em estruturas gráficas: ícones, janelas, *menus* e diálogos que o usuário pode ativar através da utilização do *mouse* ou, se preferir, do próprio teclado. O sistema foi desenvolvido para trabalhar em computadores compatíveis com o IBM-PC AT e foi implementado em linguagem Visual C++, usando metodologia orientada ao objeto.

Diferentemente dos compiladores tradicionais, ele interage com o usuário através de diálogos logo na entrada das instruções, a fim de diminuir erros posteriores de compilação. Além disso, o ONAGRO permite uma maior rapidez na entrada do programa, pois ele é orientado a ícones e não a textos, como nas linguagens convencionais.

A programação em ONAGRO é feita com um elevado nível de abstração dos detalhes de *hardware*. Entretanto, há mecanismos disponíveis para total controle das características físicas das aplicações.

Os testes realizados mostraram que o ambiente proposto é bastante intuitivo e amigável. A documentação dos programas é feita em tempo-real, visto que o próprio programa-fonte se constitui em uma ótima ferramenta de inspeção. Outro aspecto importante, observado nos testes, foi que o código gerado se mostrou relativamente compacto.

ABSTRACT

ONAGRO is a translation system that recognizes an algorithm description graphical language and allows the code generation in Assembly for microcontrollers. Besides, it incorporates a graphic editor for the source-program input that is similar to an algorithm described in flowchart language. ONAGRO has also an identifier editor that allows the identifier symbol description used on the program.

ONAGRO is a fully Microsoft Windows compatible software offering a friendly interface with user. This interface is based on graphical structures: icons, windows, menus and dialogues that the user might activate through the mouse or, if he wishes, through the keyboard. ONAGRO was developed to work on IBM-PC AT compatible computers. It was implemented in Visual C++ language by using object oriented methodology.

Differently from the traditional compilers, it interacts with the user through the dialogues immediately in the instructions input to reduce later compiler errors. Besides, ONAGRO allows a major quickness in the program input since it is icon oriented and not text oriented like in the conventional languages.

The ONAGRO programming is made with high level abstraction of the hardware details. However, there are available mechanisms for full control of physical characteristics applications.

The accomplished tests showed that the proposed environment is very intuitive and friendly. The program documentation is made in real-time since the proper source-program is a good inspection tool. Another important aspect observed in the tests was that the generated code proved to be relatively compact.

ÍNDICE

CAPÍTULO 1

INTRODUÇÃO

Este trabalho tem como objetivo o desenvolvimento de um ambiente alternativo para programação de microcontroladores, cuja característica principal é a utilização de mecanismos gráficos para expressar algoritmos. O ambiente possui interface amigável com o usuário e permite a geração, sintaticamente correta, de código em linguagem Assembly do programa-fonte introduzido. Este programa-fonte é constituído por ícones que se interligam e interagem com elementos de dados para determinar o fluxo de execução.

O trabalho é fundamentado na teoria que afirma que a melhor forma para desenvolver e compreender algoritmos é unir informações textuais com gráficas.[1][2] Portanto o ambiente proposto cria uma linguagem icônica de descrição de algoritmos, cujas operações são, em complemento, descritas de forma textual.

O ambiente desenvolvido foi denominado ONAGRO (máquina de guerra usada pelos antigos romanos para arremessar projéteis) e em algumas partes deste texto, será referenciado por *sistema* ou simplesmente *programa*, em menção ao conjunto de programas que o compõem; por *linguagem*, já que o mesmo possui uma linguagem própria de descrição de algoritmo; ou por *compilador* ou ainda *tradutor*, quando for conveniente ressaltar o módulo de tradução.

O ONAGRO foi desenvolvido para trabalhar em computadores compatíveis com o IBM-PC AT e utiliza a plataforma Windows da Microsoft. Ele foi desenvolvido em linguagem Visual C++, usando metodologia orientada ao objeto.

Neste capítulo, é feito um levantamento das linguagens de programação disponíveis comercialmente para microcontroladores e uma discussão a respeito do desenvolvimento de *software* em linguagens textuais. No final, é apresentado um estudo sobre linguagens gráficas, enfocando-se alguns trabalhos realizados na área.

1.1. LINGUAGENS DE PROGRAMAÇÃO PARA MICROCONTROLADORES

O desenvolvimento de *software* para sistemas microcontrolados não é realizado usando-se apenas os recursos do microcontrolador, que não são muito apropriados. É comum utilizar-se de outras plataformas de desenvolvimento, como por exemplo, os microcomputadores pessoais que oferecem maior capacidade de recursos (mais memória, *display* gráfico, teclado e *mouse*, memória de massa, dentre outros). Desta forma pode-se utilizar esses recursos para criar um ambiente mais amigável aos programadores desse tipo de sistema.

Mesmo assim, por ser um tipo de dispositivo com baixa quantidade de memória e estar sempre empregado em aplicações com forte interação com o *hardware*, os microcontroladores sempre tiveram sua programação associada com a linguagem Assembly.

A linguagem Assembly dos microcontroladores é semelhante as outras linguagens Assembly de microprocessadores comuns. O conjunto de instruções dispõe de um maior número de operações para manipular *bits*, em comparação aos microprocessadores usuais, entretanto o fato dos microcontroladores terem normalmente diferentes regiões de memória torna as coisas significativamente complicadas. As instruções de movimentação de dados, lógicas e desvio de execução são geralmente similares as da maioria dos outros microprocessadores. Desta forma, para quem já trabalhou com linguagem Assembly de qualquer microprocessador, o processo é o mesmo, com suas vantagens e desvantagens. [3]

Além da linguagem Assembly, são comumente utilizadas na programação de microcontroladores, três outras linguagens: PL/M, BASIC e C.

O PL/M é uma linguagem elaborada pela INTEL e tem sido disponível para seus microprocessadores, começando com o 8080. Ela se assemelha ao PASCAL, mas se originou do PL-1. Como C, ela é uma linguagem estruturada, mas usa muitos arranjos de palavras chaves para definir suas estruturas. O compilador PL/M produz um código fortemente compacto, tão bom quanto um programa escrito em Assembly. PL/M é muito mais fácil de se usar que a linguagem Assembly, pois os compiladores e "linkers" gerenciam detalhes de alocação de variáveis e movimentação de dados entre as áreas de memória. Pode-se dizer que PL/M é uma "linguagem Assembly de alto-nível", tanto no sentido negativo como no positivo, apresentando as mesmas vantagens e desvantagens. Ela permite

controlar vários detalhes de geração de código, mas para microcontroladores, PL/M não comporta números complexos, variáveis tipo ponto flutuante, ou funções trigonométricas. [3]

Outra linguagem utilizada na programação de sistemas dedicados é o BASIC, que é facilmente encontrada em computadores IBM-PC e é comumente a primeira linguagem de programação que se aprende. Ela atende bem ao seu propósito: o BASIC é uma linguagem de introdução à programação.

O BASIC é muito fácil de se usar. Na maioria das implementações, ela é interpretada, o que possibilita detectar erros ao final de cada linha do programa, ao invés de conhecê-los somente quando o programa termina de ser traduzido. Porém, existem duas razões pelas quais o BASIC não é conveniente em sistemas dedicados.

Em primeiro lugar, como ele é interpretado, ele é naturalmente lento. Cada linha deve ser convertida para o código de máquina toda vez que for executada. O processo de interpretação faz com que seja perdido muito tempo de processamento, que deveria ser usado para a aplicação propriamente dita. Existem versões do BASIC compilado (QuickBASIC, por exemplo), que evitam esse problema. Entretanto, não há até o momento, nenhuma versão comercialmente difundida do BASIC compilado para microcontroladores. [3]

Em segundo lugar, pode-se destacar a inconveniente simplificação no uso de variáveis. Todas as variáveis são, usualmente, implementadas como ponto-flutuante, o que resulta na necessidade de se executar rotinas complexas, mesmo para valores tipo inteiro. Isto torna os programas lentos e grandes.

Pode-se dizer que o BASIC, no contexto de sistemas dedicados, deve ser indicado para aplicações onde a facilidade de programação seja mais importante que a eficiência ou que a velocidade.

Finalmente, pode-se destacar a linguagem C. C é uma linguagem que surgiu com o sistema operacional UNIX. Ela é estruturada e produz um código compacto. A estrutura da linguagem é marcada pelas chaves { } delimitadoras de blocos, aos invés de palavras reservadas (begin...end, por exemplo). A linguagem C faz uso de símbolos especiais que raramente são usados na escrita cotidiana. Ela permite atingir detalhes de controle da máquina sem recorrer ao Assembly. Entretanto, os programas em C podem ser tão condensados que sua manutenção fica bastante dificultada. [3]

Desde 1985, o compilador C está disponível para microcontroladores, e há seis fabricantes diferentes que oferecem o compilador C. Cada um com suas vantagens e desvantagens. O Archimedes e o Franklin estão no topo da lista. O Franklin pelo seu código compacto e facilidade de uso; o Archimedes pelas suas facilidades complementares e boa documentação. Após estes, vêm os da BSO/Tasking e Avocet. O produto da BSO é razoavelmente rápido sem geração excessiva de código. Já o compilador Avocet dispõe de uma excelente documentação. Isso para mencionar apenas os quatro mais conhecidos.[3]

Apesar das diferentes vantagens e desvantagens oferecidas pelas quatro linguagens mencionadas, elas têm em comum o fato de o programa-fonte ser expresso por um texto (por razões históricas, um texto em inglês). Desta forma, essas linguagens aproximam o programa escrito para o computador à linguagem humana, umas em maior grau que outras. Entretanto, o nível de detalhamento das instruções do programa sintetizadas em texto, torna sua elaboração e manutenção, em muitos casos, bastante complicadas.

1.2. DESENVOLVIMENTO DE SOFTWARE EM LINGUAGENS TEXTUAIS

Segundo Willian S. Davis [4], em seu livro "Systems Analysis and Design", que se tornou um clássico da literatura na área de projeto estruturado de sistemas, o ciclo de vida de um sistema pode ser sintetizado nas seguintes fases:

❶ Definição do problema e estudo de viabilidade
❷ Análise e projeto do sistema
❸ Implementação e manutenção

Na primeira fase, procura-se definir qual o problema que se quer solucionar e analisar a viabilidade técnica e econômica da solução encontrada, bem como delinear o alcance desta. Após esta avaliação pode-se partir para as fases seguintes ou buscar novos caminhos para resolver o problema.

O passo seguinte consiste em descrever o que deve ser feito para resolver o problema e como este deve ser resolvido. Nesta fase faz-se uso de ferramentas de análise e projeto, como *diagrama de fluxo de dados*, *dicionário de dados*, *fluxogramas* e *diagrama de blocos*, dentre outras.

Por último, tem-se a fase de implementação e manutenção, em que a solução do problema é finalmente escrita em linguagem de computador.

Apesar da abordagem simplista sobre o ciclo de vida de um sistema, que não é o objeto principal deste trabalho, deve-se observar que ao final da segunda fase a solução do problema é descrita, na maioria dos casos, por meio de mecanismos gráficos e ao final da terceira fase, a mesma solução é obtida através de linguagem de formato textual, caso se utilize uma linguagem de programação convencional.

Freqüentemente, em pequenas aplicações ou em aplicações que necessitem de soluções rápidas, é comum se partir diretamente para a fase de implementação, o que torna a documentação do sistema escassa ou quase inexistente. E mesmo nas aplicações que se respeite todo o ciclo de desenvolvimento, as especificações geradas na análise do sistema podem ser mal interpretadas na fase de implementação, pois normalmente estas fases são executadas por pessoas diferentes: o analista e o programador, respectivamente.

Isto ocorre porque o processo de codificação de uma algoritmo para uma linguagem textual de computadores (como por exemplo BASIC, PASCAL e C), envolve algumas restrições: em primeiro lugar, deve-se salientar que existe um grande distanciamento entre a sintaxe das linguagens textuais e os mecanismos clássicos de representação de algoritmos, que usualmente utilizam uma linguagem gráfica, como por exemplo os fluxogramas; e em segundo lugar, se por um lado, as linguagens procedimentais e as orientadas a objetos possuem uma sintaxe que se assemelha à linguagem natural (o inglês) por outro, o grau de detalhamento envolvido em suas operações torna os programas muitas vezes bastante extensos, dificultando a manutenção deste por parte de um programador não envolvido em sua elaboração.

1.3. LINGUAGENS GRÁFICAS: UMA ALTERNATIVA ÀS LING. TEXTUAIS

Com o advento da microeletrônica, houve uma profunda alteração nos rumos da utilização dos computadores. Tanto a capacidade de processamento como a quantidade de memória aumentou em grande escala, ao mesmo tempo que as dimensões físicas e o custo caíram quase em igual proporção. Isto possibilitou uma maior diversificação de aplicações e um grande aumento no número de pessoas diretamente envolvidas no desenvolvimento dessas aplicações.

Desta forma, o número de equipamentos disponíveis excedeu ao de especialistas em desenvolvimento de sistemas. Assim um número considerável de aplicações deve ser implementado por pessoas não especializadas em programação. Surge então a necessidade de tornar essa tarefa mais simples e mais próxima das aplicações.

Uma alternativa encontrada, em face aos problemas inerentes às linguagens textuais, foi a criação dos compiladores gráficos, que permitem uma geração automática do programa para uma linguagem textual ou para o próprio código de máquina. Com isto se elimina o problema de comunicação entre analistas e programadores e o duro trabalho da fase de codificação. Além disso, o uso de linguagens gráficas torna a documentação sempre atualizada, pois os compiladores gráficos utilizam as próprias ferramentas de documentação como fonte de entrada.

Entretanto, o aspecto mais importante das linguagens gráficas pode ser resumido pelas palavras do gênio italiano Leonardo da Vinci: ***"uma figura vale por mil palavras"***.

Isto fica evidente se observarmos os *softwares* para microcomputadores, que tendem a utilizar um ambiente interativo com o usuário, explorando fundamentalmente mecanismos gráficos para melhor implementar a interface homem-máquina.

No caso dos microcomputadores esta tendência tornou-se nítida em meados da década de 80 com a mudança dos sistemas operacionais orientados a comandos de linha para os sistemas operacionais orientados a ícones. Com isso, foi possível uma maior abstração do *hardware* e do próprio *software* (o sistema operacional) por parte do usuário, que para realizar suas tarefas não necessita se deter em tantos detalhes de sintaxe ou conhecer com maior profundidade os mecanismos de acesso aos recursos da máquina.

Além disso, sabe-se que o ser humano tem maior habilidade em interpretar um algoritmo através da linguagem gráfica, ao invés da linguagem textual.

Pesquisas na área de "neurocognição" (Springer e Deutsch [5]) mostraram que o hemisfério esquerdo do cérebro processa informações seqüencialmente, verbalmente e logicamente. Já o hemisfério direito processa informações simultaneamente, visualmente e espacialmente. Acredita-se que os atributos visual e espacial formam uma única habilidade cognitiva. Veja ilustração na figura 1.1.

As técnicas textuais ou verbais de descrição de algoritmos, como o pseudocódigo e as linguagens de programação tradicionais, ativam em maior grau os recursos neurocognitivos do hemisfério esquerdo do cérebro. Isto ocorre porque as técnicas textuais

de construção de algoritmo contêm mais estímulos seqüenciais, verbais e lógicos, que são os estímulos que mais sensibilizam este hemisfério. [5]

Os mecanismos textuais contêm muito pouca informação espacial, desta forma o hemisfério direito não contribui significativamente com eles. Conseqüentemente, somente metade do cérebro pode estar influenciando no processo de compreensão quando se usa estas técnicas. [5]

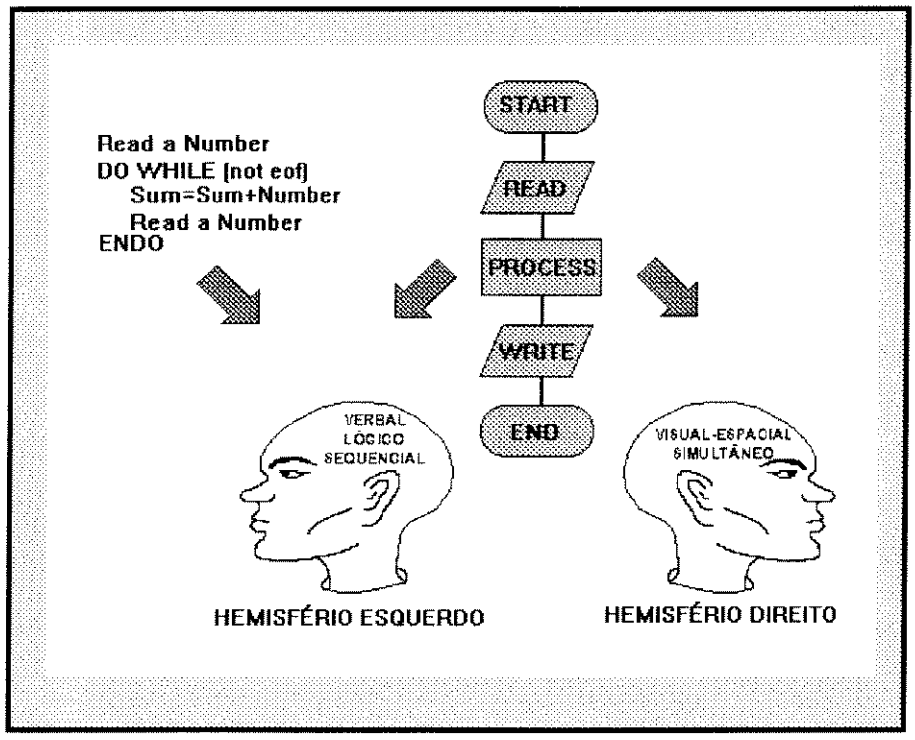


Figura 1.1: O processamento de informações no cérebro

Ao contrário das técnicas textuais, os mecanismos gráficos de compreensão de algoritmo, como os fluxogramas estruturados, tendem a estimular todo o cérebro. As técnicas gráficas possuem informações seqüenciais, lógicas e, em menor proporção, informações verbais. Assim possibilita o estímulo do hemisfério esquerdo do cérebro. Além disso, as técnicas gráficas também estimulam o hemisfério direito, pois contêm muita informação visual-espacial. [5]

Pesquisas realizadas por B. A. Calloni e Donald J. Bagert [6] mostraram que as linguagens gráficas são mais intuitivas e facilitam a aprendizagem e compreensão de algoritmos.

Eles desenvolveram uma linguagem de programação procedimental baseada em ícones para o ensino de programação, chamada BACII e realizaram comparação com a linguagem textual PASCAL, muito usada nas universidades para o ensino de iniciação à programação.

A pesquisa foi motivada pelos resultados obtidos no trabalho, também experimental, do psicólogo David A. Scanlan na área de compreensão de algoritmos, que após testes realizados em 1988 [1] e 1989 [2] concluiu que os métodos gráficos possibilitam a abstração necessária dos detalhes de sintaxe e apresentam-se melhores que os métodos textuais nos processos mentais intuitivos para o ensino (e compreensão) de desenvolvimento de algoritmos.

No trabalho de B. A. Calloni e Donald J. Bagert, os alunos submetidos à pesquisa foram alocados, aleatoriamente, para turmas que usaram o BACII e outras que usaram o PASCAL. Após uma análise estatística do desempenho dos alunos, os autores concluíram que a linguagem gráfica foi mais eficaz que a linguagem textual, melhorando a aprendizagem e a compreensão de algoritmos.

Atualmente os compiladores gráficos ou os ambientes visuais estão consolidando um papel importante na programação de computadores. Essa tendência vêm aumentando graças ao grande desenvolvimento tecnológico que possibilitou o barateamento de monitores de vídeo de alta resolução.

São disponíveis, hoje em dia, diversas ferramentas gráficas para programação de sistemas, desde compiladores para aplicações de *multimídia* como AUTORWARE® e o DIRECTOR® até gerador automático de código para sistemas de aquisição de dados e controle, como é o caso do *software* LABTECH-CONTROL®. Nestas ferramentas o usuário introduz o algoritmo da aplicação, fundamentalmente, através de mecanismos gráficos.

CAPÍTULO 2

DESCRIÇÃO FUNCIONAL DO ONAGRO

INTRODUÇÃO

Abordaremos neste capítulo a justificativa para escolha do ambiente de desenvolvimento e da metodologia usada no trabalho, ressaltando as principais características do plataforma operacional (Microsoft Windows) e da linguagem de programação (Visual C++), bem como da Programação Orientada ao Objeto. Também será feita uma descrição do funcionamento do compilador ONAGRO e da forma pelo qual ele foi projetado. Os aspectos a serem levantados procuram oferecer uma visão do ponto de vista interno do sistema, enfocando mais os detalhes de implementação. Fica para o próximo capítulo uma abordagem mais operacional.

2.1. O AMBIENTE DE DESENVOLVIMENTO

Como o objetivo principal do trabalho é criar uma linguagem gráfica para programação de microcontroladores, optou-se por usar um ambiente de desenvolvimento que explorasse os recursos gráficos na implementação da interface homem-máquina. Assim, o ambiente escolhido foi o Windows da Microsoft, que é atualmente um ambiente operacional para microcomputadores mundialmente difundido.

A escolha da linguagem de programação se deu pelas facilidades oferecidas pela metodologia orientada ao objeto e pelos recursos de manipulação de estruturas gráficas. Desta forma, optou-se pelo Visual C++ por apresentar essas características e ser um pacote totalmente compatível com o Windows, pois foi desenvolvido pela própria Microsoft.

2.1.1. O Microsoft Windows

O Microsoft Windows surgido em meados da década de 80, apresenta uma interface com o usuário baseada no "ver e sentir" das normas CUA (Common User Access) da IBM. Faz forte valorização ao visual (gráfico, mouse e janelas) na comunicação homem-máquina

e recebeu grande influência dos ambientes orientados ao objeto. Ele é dirigido a eventos e permite o reaproveitamento de objetos de classes pré-definidas, que são usadas em programação baseada em hereditariedade. [7]

2.1.2. O Visual C++

O Visual C++ é um programa de desenvolvimento de aplicações orientadas ao objeto. Além de um Compilador, de um Linkeditor e de um Depurador, ele integra num mesmo pacote outras ferramentas de desenvolvimento, desde editor de programa até um gerador automático de código para aplicações básicas. Estas ferramentas formam um ambiente interativo de desenvolvimento baseado no Windows, chamado *Visual Workbench*. [8]

Uma das ferramentas mais interessantes do Visual C++ é, sem dúvidas, o editor de recursos, o *App Studio*. Este editor é responsável pela criação de *menus*, diálogos, *bitmaps*, ícones, tabelas de *string* e cursores de forma bastante amigável com o programador. Estes componentes são compilados para um arquivo especial em disco (o arquivo de recursos) ou vinculados ao programa executável. [8]

Para facilitar o processo de codificação o Visual C++ dispõe de um gerador automático de código, o *App Wizard*, que cria a estrutura básica de uma aplicação para Windows com recursos, classes e arquivos definidos através de caixas de diálogos, introduzindo de forma rápida o programador na nova aplicação. Com o App Wizard pode-se elaborar o esqueleto de uma aplicação genérica e depois acrescentar as implementações específicas. [9]

Com a finalidade de tornar o processo de manutenção menos tedioso, o Visual C++ oferece uma ferramenta de inspeção chamada *Source Brouser* que permite examinar uma aplicação a partir de uma classe ou de uma função. [8]

O Visual C++ possui também um programa gerenciador de classes, o *Class Wizard*, que proporciona a criação de protótipos, funções e o código para ligar as mensagens à estrutura da aplicação. [8]

Finalmente pode-se destacar no Visual C++ a *Microsoft Fundation Class* ou apenas MFC. A MFC é uma biblioteca de classes criada para facilitar o desenvolvimento de aplicações orientadas ao objeto. Ela possui cerca de cem classes agrupadas em três

categorias: a) classes destinadas à construção de aplicações Windows, que oferecem fácil acesso aos dispositivos de interface gráfica e permitem a incorporação e "linkagem" de objetos; b) classes de propósito geral, usadas para manipulação de listas encadeadas, arquivos, *strings*, dentre outras; c) Macros e Globais, que consistem em várias macros e objetos de propósito geral. [9]

2.2. METODOLOGIA DE DESENVOLVIMENTO

O Sistema ONAGRO foi projetado usando a metodologia orientada ao objeto (ou Programação Orientada ao Objeto - POO), pelas vantagens apresentadas em relação às metodologias tradicionais. Dentre essas vantagens pode-se destacar:

- ☑ acentuado aumento de produtividade, pois o projeto orientado ao objeto tende a aproximar a fase de desenvolvimento à fase de implementação, através de um modelamento mais próximo da realidade da aplicação.

- ☑ o projeto orientado ao objeto permite uma maior reutilização do código (rotinas e bibliotecas) já existente, incentivando a modularização tanto de operações como de dados.

2.2.1. Conceitos básicos da POO

Simplificadamente, pode-se dizer que a *programação orientada ao objeto* é uma ferramenta de construção de *software* básico, onde a reutilização é o ponto central do desenvolvimento de *software*, pois não se elabora completamente um programa a partir de definições do mesmo, e sim elabora-se rotinas reutilizáveis. [10]

A idéia é colocar a fase de desenvolvimento mais próxima da fase de implementação, aumentando a produtividade de analistas e programadores através de uma maior expansibilidade e reutilização de código; e controlando a complexidade e o custo de manutenção do *software* desenvolvido. [11] Desta forma, a modificação de um programa orientado ao objeto não afeta sua estrutura. O objetivo é que cada novo módulo (objeto)

introduzido no programa não afete os outros módulos, mas sim reutilize as operações e dados já definidos, conforme mostra a figura 2.1. [7]

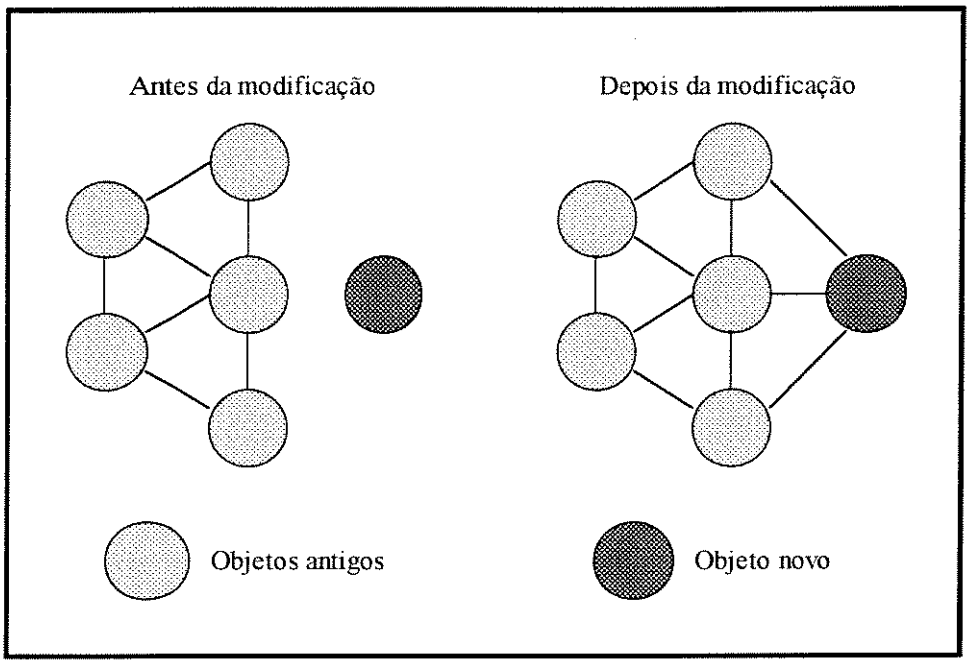


Figura 2.1: A modificação de um programa OO não afeta a estrutura do programa.

Os requisitos básicos para o desenvolvimento de um *software* orientado ao objeto são: *modularização*, *abstração* e *ocultação*. Estes requisitos são a base da metodologia orientada ao objeto. [12]

❶ Modularização

Sabe-se que os métodos convencionais estimulam apenas a modularização do processamento, entretanto o projeto orientado ao objeto busca conectar dados e operações, modularizando tanto o processamento como também os próprios dados.

R. S. Pressman [12] sugere que o *software* deva ser dividido em elementos com nomes e objetivos separados (os módulos) que devem se unir para satisfazer as definições do problema. Com isso poder-se-ia manejar intelectualmente o programa.

Ele também faz uma observação importante quanto a complexidade e esforço no desenvolvimento de *software*, conforme mostram as inequações:

a)	$C(p_1) > C(p_2)$	$\Rightarrow E(p_1) > E(p_2)$

b) $C(p_1 + p_2) > C(p_1) + C(p_2) \Rightarrow E(p_1 + p_2) > E(p_1) + E(p_2)$

C ⇨ Complexidade E ⇨Esforço p_1 e p_2 ⇨Problemas

Pode-se dizer que a modularização procura diminuir os efeitos das alterações em um sistema, desta forma deve-se refinar sucessivamente os módulos, até o ponto em que o esforço para interfaceá-los não comprometa o desenvolvimento do *software*.

❷ Abstração

No modelamento do sistema orientado ao objeto tanto os dados como os procedimentos devem ser abstraídos, ressaltando-se a composição e o comportamento dos dados e dos módulos. Deve-se deixar os detalhes de implementação para outras fases, de tal forma que os níveis superiores do sistema estabeleçam uma solução ampla e os níveis inferiores uma solução mais detalhada.

❸ Ocultação

Na programação orientada ao objeto deve-se ocultar ou encapsular dados e operações para impedir que as informações contidas em um módulo não sejam acessíveis a outros módulos que não necessitem daquela informação. Assim torna-se mais fácil a depuração e manutenção dos módulos, pois dificulta a propagação de erros.

2.2.2. Estruturas básicas da POO

Outro aspecto importante da programação orientada ao objeto se refere às suas estruturas básicas: os objetos, as mensagens, as classes e a hereditariedade.

Um **objeto** pode ser definido como um elemento do mundo real modelado como um componente de *software*. Todo programa orientado ao objeto consiste basicamente em objetos, estes por sua vez são formados por *membros de dados* e por *métodos* que manipulam esses dados. Desta forma pode-se caracterizar os objetos pelos seus atributos particulares (dados) e pelo seu comportamento (métodos).

Outra estrutura importante na programação orientada ao objeto é a geração de **mensagens** entre objetos. Pode-se dizer que uma mensagem é uma solicitação para que um objeto assuma um determinado comportamento, ou seja, que um método do objeto que

recebeu a mensagem seja executado com base na passagem de um parâmetro ou em um dado do próprio objeto. A figura 2.2 mostra a anatomia de um objeto e ressalta os mecanismos de mensagens como única forma de acesso aos dados do objeto.

Quando um conjunto de objetos apresenta as mesmas características, diz-se que estes objetos pertencem a uma mesma **classe**. Portanto, uma classe representa um modelo pelo qual podem ser criados os objetos. É preciso ressaltar que as classes são entidades estáticas, só podem ser modificadas em tempo de compilação, ao passo que os objetos são entidades dinâmicas presentes na memória, podendo ser alteradas em tempo de execução.

A programação orientada ao objeto permite a reutilização do código através da **hereditariedade**. Uma classe-filha pode ser criada herdando as estruturas de dados e os métodos de uma classe-pai. Assim novas classes podem ser criadas pela especialização da classe-pai, formando uma hierarquia de classes.

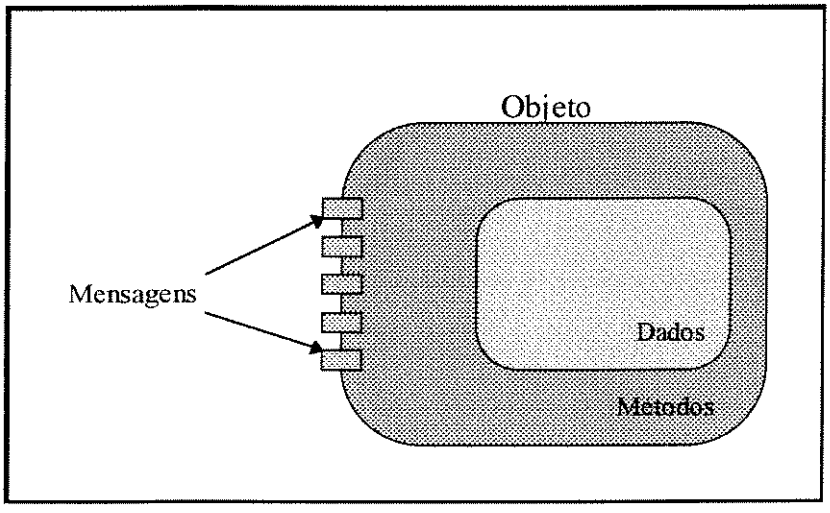


Figura 2.2: Anatomia de um objeto.

2.3. O PROJETO DO SISTEMA

Uma das principais dificuldades encontradas no projeto do ONAGRO, como na maioria dos projetos orientados ao objeto, foi a identificação dos elementos que seriam modelados como objetos e quais as características que eles iriam assumir.

Para resolver este problema foi adotada uma técnica que se baseia na *verificação gramatical* proposta por Grandy Booch.

"Grandy Booch (1983) criou a técnica gramatical e propôs que o projetista começasse com uma descrição formal do sistema desejado e observasse os substantivos como identificadores em potencial das classes dos objetos. Os verbos, por outro lado, identificariam métodos. A lista resultante de classes (substantivos) e métodos (verbos) seria usada para começar o processo do projeto". [7]

Conforme a metodologia proposta por Booch, elaborou-se inicialmente a definição textual do problema e a descrição da solução:

Definição do problema: "Desenvolver um ambiente gráfico para programação de microcontroladores"

Descrição da solução: "O ambiente deve permitir a entrada do dicionário de dados e dos fluxogramas de uma dada aplicação e a geração do código em linguagem Assembly a partir desses elementos. Os fluxogramas serão introduzidos através de um editor gráfico e serão compostos por ícones de vários tipos de operações pré-definidas ou de operações definidas pelo usuário, onde os ícones podem ser personalizados por um editor de ícones, incorporado ao sistema. Os ícones se interligarão, determinando o fluxo de execução.

O dicionário de dados será composto por uma descrição dos símbolos identificadores (constantes, variáveis, portas de E/S e nomes de sub-rotinas) usados no fluxograma. Os identificadores serão manipulados por um editor especial, o editor de identificadores.

Após a entrada dos fluxogramas e do dicionário de dados, o usuário pode ativar o compilador para realizar a alocação dos dados e a conversão dos ícones de operação para a linguagem Assembly.

Também será permitido o acionamento de um montador/linkeditor padrão para a geração do código em linguagem de máquina, bem como a comunicação do ambiente de programação com um sistema alvo, através de um programa de comunicação serial, como mostra a figura 2.3.

A programação das interrupções do microcontrolador, bem como dos dispositivos temporizadores e da interface de comunicação serial será feita por meio de diálogos amigáveis permitindo uma maior abstração dos detalhes de operação dos recursos do microcontrolador.

O sistema deve permitir a gravação e leitura em disco das informações de entrada, como também das geradas pelo próprio ambiente.

Os parâmetros usados pelo sistema deverão estar abertos e poderão ser configurados, de forma interativa, pelo usuário.

Por último, o sistema deverá ter um gerenciador de atividades, com forte interação com o usuário, para facilitar o acesso aos recursos do ambiente".

O passo seguinte é marcar os substantivos ou frases substantivas, a fim de identificar os objetos em potencial:

OBJETOS:
- ✓ lista de ícones (fluxogramas)
- ✓ ícone de operação
- ✓ lista de identificadores (dicionário de dados)
- ✓ símbolo identificador
- ✓ editor gráfico de fluxogramas
- ✓ editor de símbolos identificadores
- ✓ compilador para linguagem Assembly
- ✓ manipulador de disco (Serializador)
- ✓ configurador de parâmetros do sistema
- ✓ configurador de recursos (interrupção/timer/serial) do microcontrolador
- ✓ gerenciador de atividades
- ✓ montador/linkeditor padrão
- ✓ programa de comunicação (TERMINAL)
- ✓ editor de ícones

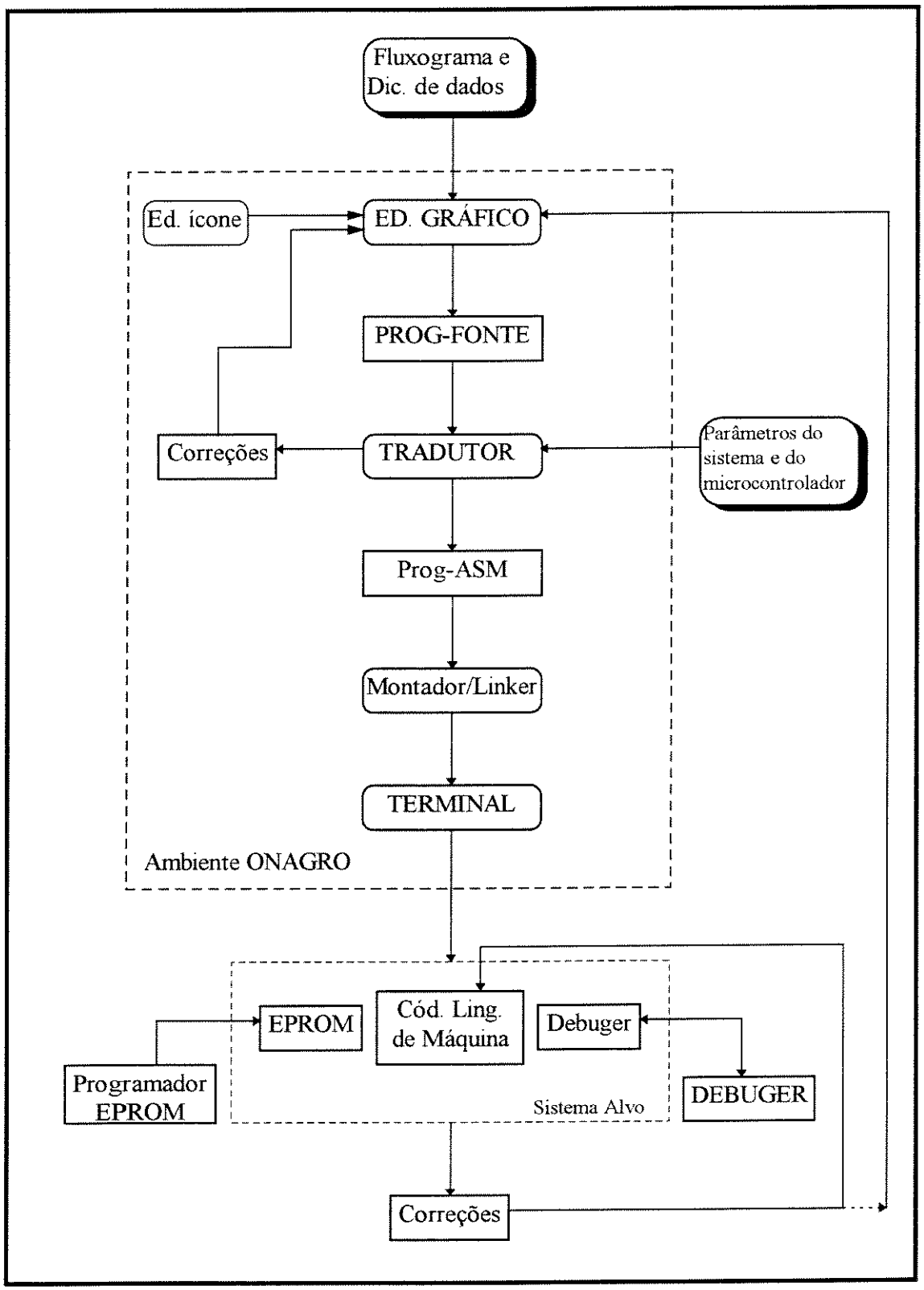


Figura 2.3: Diagrama em blocos do ambiente ONAGRO

Esses objetos se interligarão através de mensagens e irão compor o sistema ONAGRO, como visto na figura 2.4.

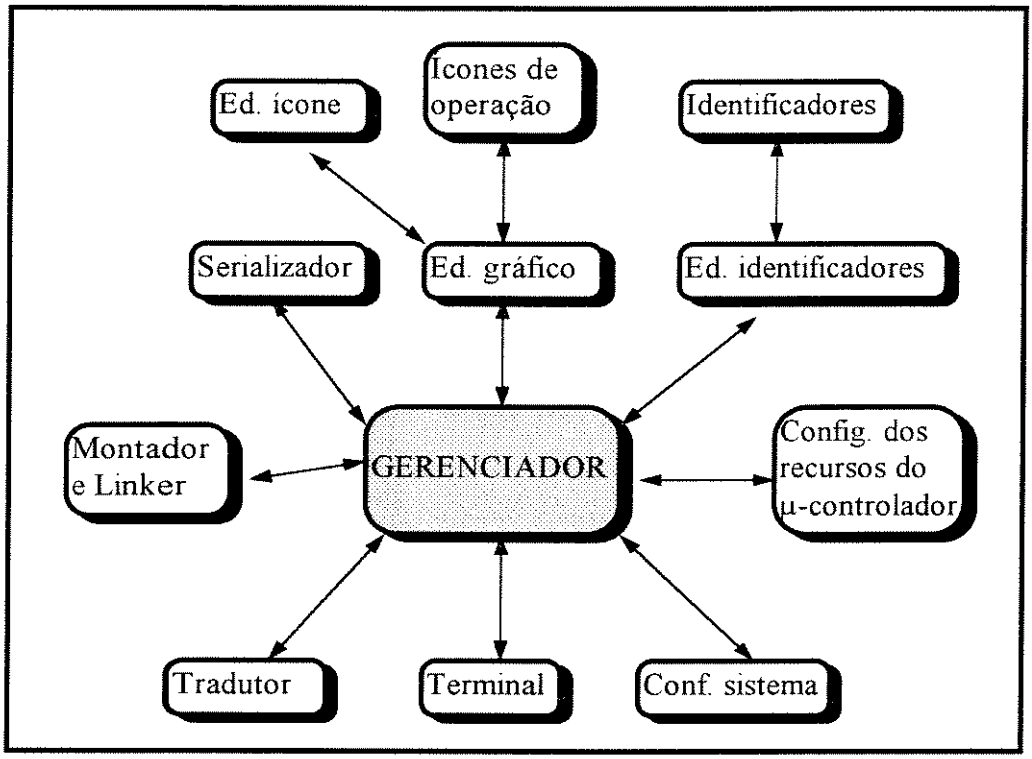


Figura 2.4: Os objetos do sistema ONAGRO

Após a definição dos objetos vem a fase de localização dos possíveis métodos, através da identificação dos verbos:

MÉTODOS:
- ✓ inserir/mover/remover ícones
- ✓ interligar e personalizar ícones
- ✓ editar símbolos identificadores
- ✓ gerar código para alocação dos dados
- ✓ traduzir fluxogramas para linguagem Assembly
- ✓ acionar montador/linkeditor padrão
- ✓ acionar programa de comunicação
- ✓ acionar programa editor de ícones
- ✓ armazenar/recuperar informações em disco
- ✓ programar os recursos do microcontrolador
- ✓ configurar parâmetros do sistema

Tendo sido determinado os prováveis objetos e métodos que serão usados pelo sistema, o próximo passo é fazer uma associação dos objetos com os métodos e procurar estabelecer os principais atributos desses objetos.

Temos a seguir uma descrição dos principais objetos modelados, ressaltando-se suas características e forma de implementação.

2.3.1. Os ícones de operação

Os *ícones de operação* são objetos que representam as operações básicas que podem ser executadas na linguagem. Eles são análogos aos códigos de operação de uma instrução de máquina, ou seja, eles essencialmente indicam a natureza do processamento que se deseja realizar. É preciso, em complemento, que se indique os parâmetros que serão usados na operação. Esses parâmetros normalmente são identificadores de variáveis, portas de e/s, dentre outras.

Todo ícone de operação tem uma estrutura mínima de dados e métodos que fornecem as características básicas (dados) e possibilitam a interação com o resto do sistema (métodos). Eles são agrupados em classes distintas, porém hierarquizadas de forma a permitir o reaproveitamento das características comuns.

Como mostra a figura 2.5, pode-se observar que os ícones de operação apresentam diversas variações. A linguagem oferece já pré-definidos os ícones para realizar atribuição de dados; deslocamento de *bits*; comparação e iteração; operações lógicas e aritméticas; e manipulação de sub-rotinas, que são as operações mais comuns. Entretanto, também são fornecidos ícones mais específicos para aplicações dedicadas, como os ícones de comunicação serial, os de geração de bordas e o de geração de atrasos. Além disso, o usuário pode criar seus próprios ícones de operação e designar quais tarefas eles devem executar, pois a linguagem dispõe de um tipo de ícone cujo código é definido pelo usuário.

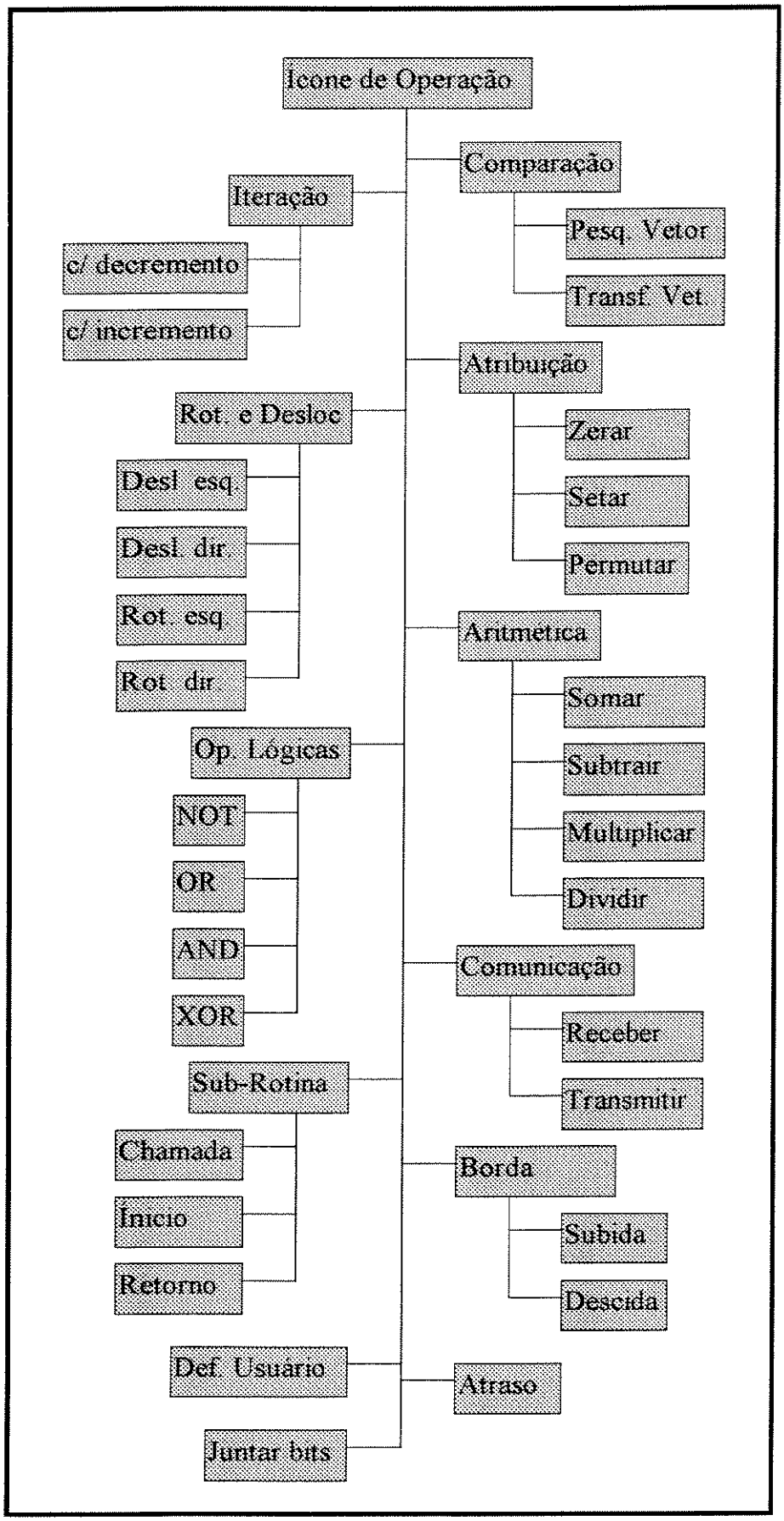


Figura 2.5: Hierarquia dos ícones de operação

2.3.2. O objeto lista de ícones

O programa-fonte submetido ao compilador é composto por um arranjos de ícones que simbolizam as operações válidas da linguagem e cujas ligações determinam o fluxo de execução do programa. O compilador ONAGRO reconhece diversos tipos de operações, individualizadas por ícones diferentes. A quantidade desses ícones dentro de um programa-fonte pode ser variável, conforme se inclua ou se exclua alguma operação. Por isso, o objeto que representa o programa-fonte para o compilador, a *lista de ícones*, utiliza uma estrutura heterogênea de componentes (figura 2.6), permitindo uma alocação dinâmica para os diferentes tipos de ícones.

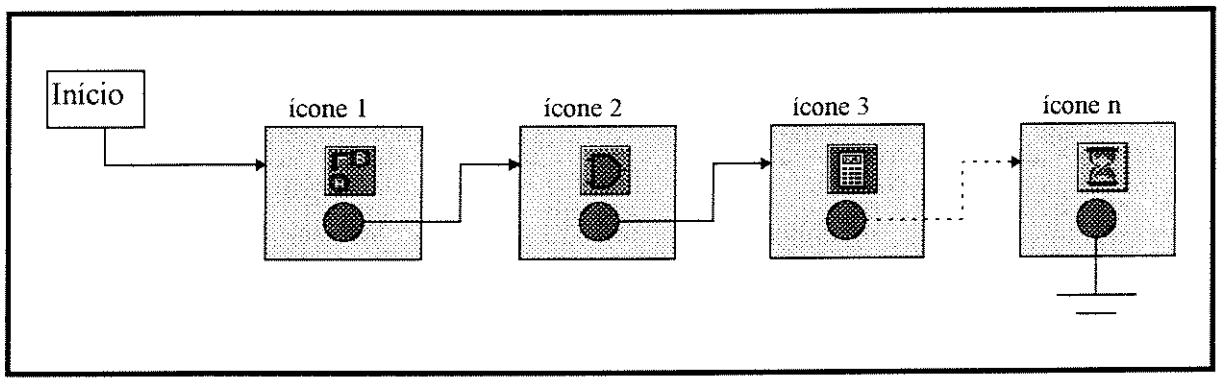


Figura 2.6: Lista heterogênea de ícones

Essa estrutura é montada pelo usuário através do editor gráfico, que envia mensagens (figura 2.7) para inserir ou remover um ícone de operação, obter o próximo elemento que vai ser exibido, verificar quais ícones estão ligados a um outro ícone em particular e determinar o número de componentes da lista, dentre outras.

A lista de ícones recebe também uma mensagem do objeto tradutor ou compilador para que comece o processo de geração de código. Ela faz as iniciações necessárias e envia uma mensagem para cada um de seus integrantes (ícones de operação). Essas mensagens são enviadas obedecendo o fluxo de execução. Quando um ícone de operação recebe essa mensagem, ele procede a geração de seu código, remetendo-o para a lista de ícones que por sua vez, envia ao tradutor o resultado de toda a compilação.

Além disso, a lista de ícones é responsável pelo seqüenciamento das informações relativas aos ícones de operação, que serão lidas ou escritas em disco. O processo é

semelhante ao de tradução, com envio de mensagens da lista aos seus componentes e desta ao objeto serializador que lida com as operações em disco.

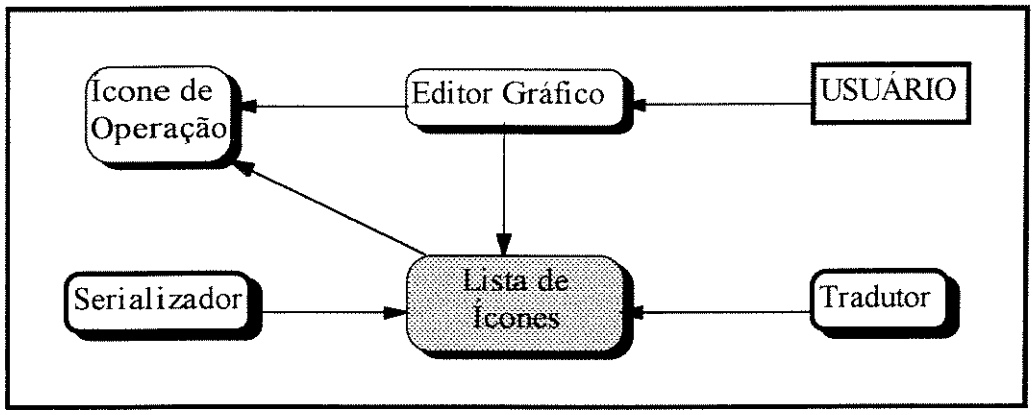


Figura 2.7: Relacionamento da lista de ícones com demais objetos

2.3.3. O objeto símbolo identificador

Os *identificadores* são entidades usadas no programa, tais como constantes, variáveis, portas de e/s, nomes de sub-rotinas e parâmetros locais. Eles servem para abstrair o usuário de detalhes da arquitetura do microcontrolador, facilitando o acesso à memória e às portas de e/s.

Antes de descrever os tipos de identificadores usados na linguagem, cabe aqui uma crítica às linguagens tradicionais de programação de microcontroladores no que diz respeito a manipulação das portas de entrada e saída.

Como se sabe, as linguagens de alto-nível foram criadas para abstrair o programador dos detalhes do *hardware*. Isso é importante quando este *hardware* é padrão, e sobre ele deseja-se realizar aplicações distintas, que não envolvem controle sobre qualquer dispositivo fisico. Entretanto, quando se programa microcontroladores, é necessário se envolver um pouco com os detalhes da máquina, a fim de melhor aproveitar seus parcos recursos e poder intervir diretamente na aplicação, que normalmente se constitui no controle de um dispositivo fisico pelo programa.

As linguagens tradicionais de programação de microcontroladores não possuem um mecanismo eficiente para manipulação das portas de e/s. Normalmente elas dispõem apenas de pseudo-instruções para criar identificadores associados a endereços de portas de e/s, ou de funções específicas para leitura ou escrita nas portas. Estes mecanismos nem sempre são

adequados aos uso das portas de e/s enquanto entidades que interagem fortemente com o processamento.

Em muitos casos as portas de e/s são constituídas por um conjunto de *bits*, cujo tamanho difere do número de *bits* da palavra da CPU. Para contornar este problema, o programador de linguagens como C e BASIC, por exemplo, tem que usar instruções adicionais para criar máscaras e espelhos, permitindo o acesso apenas aos *bits* que serão usados da palavra de dados transferida entre a CPU e os dispositivos externos.

Como as aplicações para microcontroladores fazem forte interação com dispositivos de e/s, criou-se na linguagem proposta, uma diferenciação entre portas de e/s e variáveis. Assim fica mais claro para o usuário o caráter das operações que envolvem transferência de dados da CPU com os dispositivos de e/s (estas operações são muito comuns e importantes em aplicações dedicadas). Esta característica do sistema ONAGRO facilita o entendimento e a manutenção dos programas.

Além disso, optou-se em fazer uma hierarquização dos identificadores devido às diferenças entre eles (ver figura 2.8). Desta forma a implementação dos dados e métodos comuns é aproveitada por todos os objetos identificadores e as peculiaridades inerentes a cada um são encapsuladas em instâncias separadas.

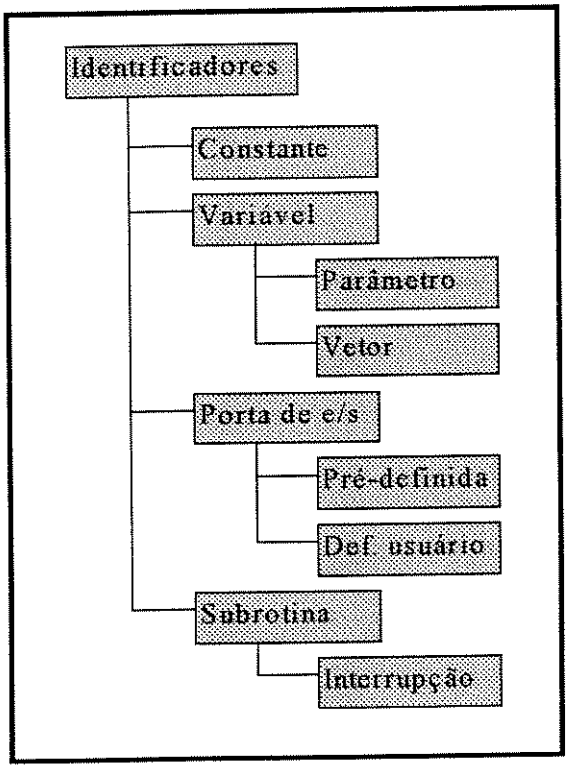


Figura 2.8: Hierarquia dos identificadores

A classe *Variável* possui duas especializações: parâmetro de sub-rotina e vetor. Os *parâmetros* representam variáveis locais alocadas na pilha do microcontrolador e são usadas na passagem de valores entre módulos distintos do programa. Já os *vetores* são variáveis que armazenam um conjunto de valores indicados por um índice.

As *portas de e/s* também possuem mais duas variações: as *portas pré-definidas* pela própria arquitetura do microcontrolador e as portas especiais *definidas pelo usuário*.

As *sub-rotinas* por sua vez têm uma espécie particular que são as *sub-rotinas de interrupção*, cuja codificação difere em alguns detalhes das subrotinas convencionais.

Cada objeto identificador é capaz de realizar a sua geração de código. As variáveis, por exemplo, são responsáveis pela geração do código para alocação de memória e para as operações de leitura e escrita.

Ocorre o mesmo com as portas de e/s e com as constantes que criam o código necessário para serem manipuladas. Já as subrotinas criam os rótulos dentro do programa e especificamente as subrotinas de interrupção reservam áreas particulares dentro da memória para alocação de suas operações.

2.3.4. O objeto lista de identificadores

Este objeto é semelhante ao objeto lista de ícones, sendo que as informações manipuladas pela *lista de identificadores* (figura 2.9) se referem, como o próprio nome sugere, aos símbolos identificadores de constantes, variáveis, portas de e/s e cabeçalhos de subrotinas. Assim, ele pode ser visto como o dicionário de dados armazenado numa entidade do *software* que permite tanto ao usuário como ao próprio sistema realizar operações de inserção, retirada ou modificação de algum identificador usado no programa.

O usuário tem acesso a esse objeto por intermédio do editor de identificadores (figura 2.10), que permite introduzir a maioria dos identificadores usados no programa. Em alguns casos o tradutor insere identificadores pré-definidos pela arquitetura do microcontrolador e cria automaticamente variáveis ou constantes usadas por alguns ícones de operação, a fim de diminuir o trabalho do usuário.

O tradutor interage com a lista de identificadores (figura 2.10) da mesma forma que faz com a lista de ícones, solicitando que o processo de geração de código, relativo aos símbolos identificadores, seja iniciado. A lista de identificadores envia uma mensagem para

que cada elemento gere seu código, obedecendo a seguinte seqüência: primeiro é enviado uma mensagem para os identificadores de constantes, depois para as portas de e/s, em seguida para as variáveis e por último para as subrotinas. No final do processo, a lista de identificadores envia ao tradutor o código obtido para que o mesmo faça os últimos acabamentos necessários.

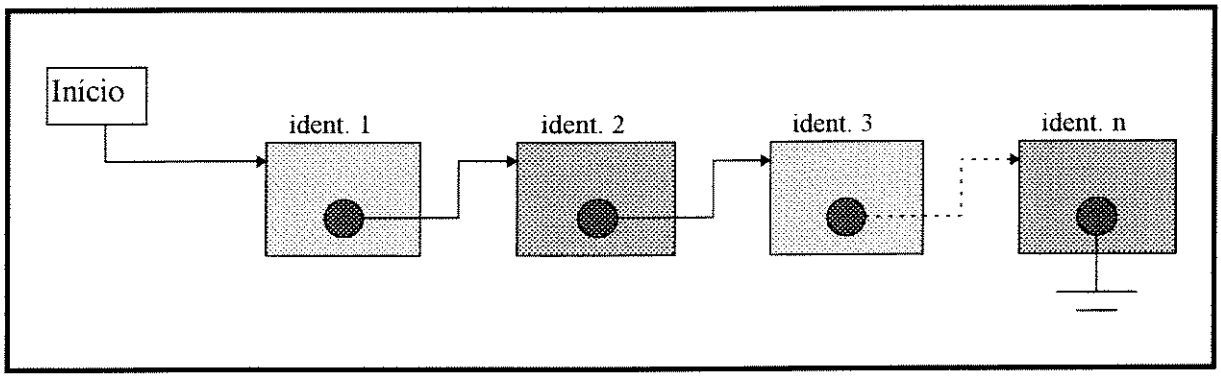


Figura 2.9: Lista de identificadores (heterogênea)

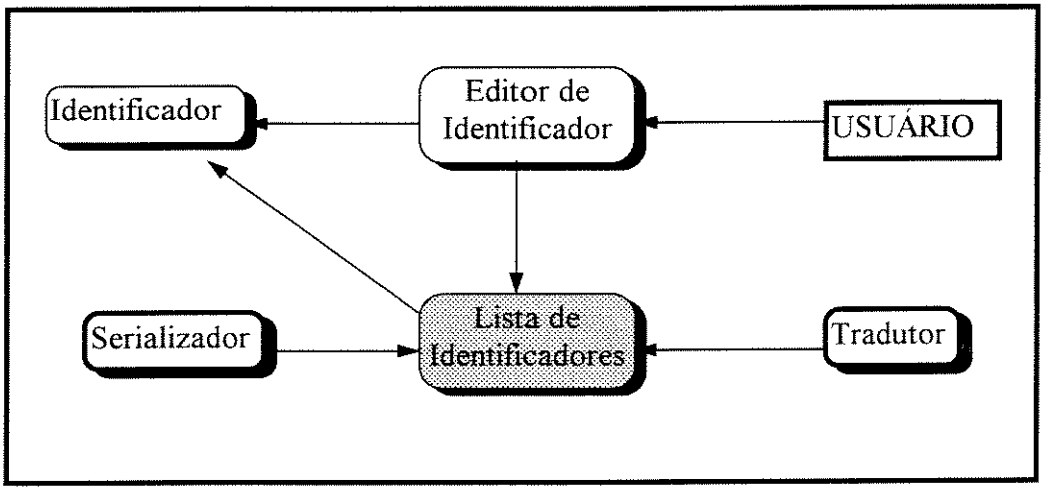


Figura 2.10: Relacionamento da lista de identificadores com demais objetos

2.3.5. A estrutura do código e os mecanismos de tradução

O ONAGRO possui um código orientado a ícones, ou seja, as operações são descritas por arranjos de símbolos padronizados (veja exemplo na figura 2.11), os ícones de operação. Estes arranjos possuem uma sintaxe dirigida pelo editor gráfico, de tal forma que só são permitidas combinações corretas de declaração de operações. Isto além de diminuir excessivos testes de sintaxe, como ocorre nas linguagens tradicionais , torna a estrutura do

código mais previsível para o compilador sem, no entanto, obrigar o programador a seguir rigidamente uma sintaxe como no caso das linguagens de formato fixo.

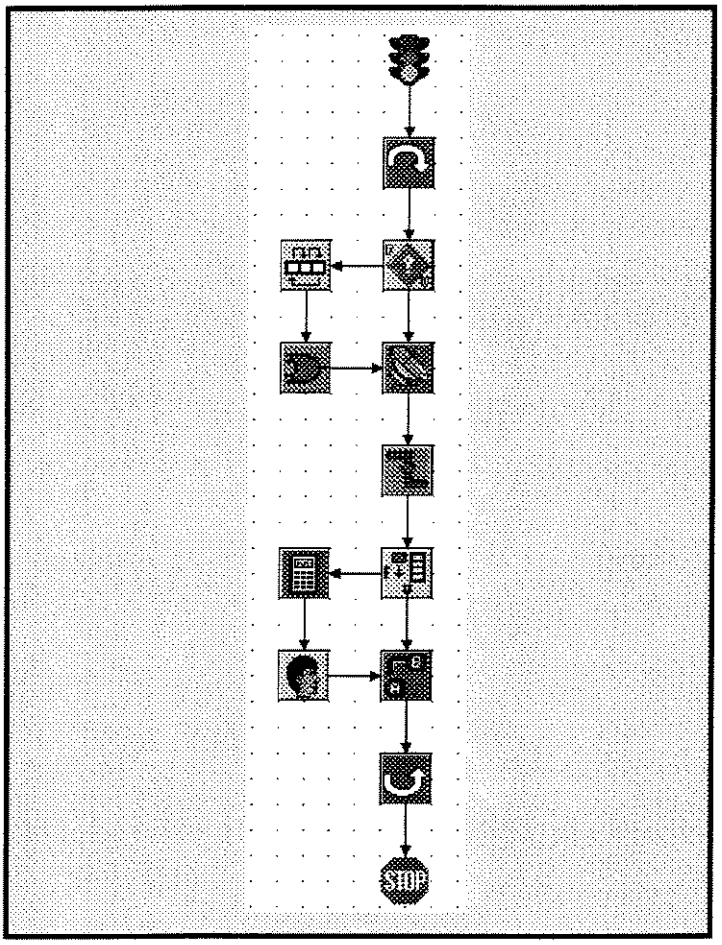


Figura 2.11: Exemplo de um programa em ONAGRO

Cada ícone de operação é responsável pela geração de seu próprio código ou da maior parte dele. Isto facilita o processo de tradução e diminui a necessidade de se usar otimizadores de código, pois o ícone se comunica com os outros objetos envolvidos na operação (símbolos identificadores e/ou outros ícones) para que eles gerem a parte do código a eles pertinente ou indiquem a natureza dos dados a serem manipulados. Assim, o ícone de operação pode determinar o melhor código possível, criando-o semanticamente correto. Na maioria dos casos, por exemplo, o ícone de operação não compõe o código de acesso à memória ou às portas de e/s, isso fica a cargo dos objetos identificadores de variáveis e portas de e/s. O ícone de operação apenas envia mensagens a esses objetos para que eles gerem o código de leitura/escrita em memória ou em porta de e/s.

Evidentemente, nem todas as redundâncias são eliminadas. Em operações que envolvem mais de uma vez um determinado elemento de dado, pode ocorrer geração de código redundante, pois para cada novo acesso ao dado é enviado uma mensagem ao objeto que o representa, a fim de que este gere o código de máquina de acesso à memória ou às portas de e/s.

Na geração do código, procurou-se adotar um mecanismo que fosse o mais genérico possível no que diz respeito ao tipo de microcontrolador. Por isto a maior parte do código gerado utiliza, fundamentalmente, as instruções essenciais e os registradores básicos da arquitetura dos microcontroladores. Obviamente, nas operações associadas às características do microcontrolador isto não é possível, pois cada microcontrolador possui suas particularidades. Entretanto, adotou-se no processo de tradução uma **tabela de mnemônicos** que armazena o conjunto de instruções em Assembly necessárias para implementar cada ícone de operação. Esta tabela é guardada em disco num arquivo de formato ASCII.

Para cada família de microcontrolador utilizado é necessário um arquivo que contenha a tabela de mnemônicos correspondente. O ONAGRO não dispõe de nenhuma ferramenta de apoio para edição desta tabela entretanto, ela pode ser modificada através de um editor de textos convencional.

CAPÍTULO 3

DESCRIÇÃO OPERACIONAL DO ONAGRO

INTRODUÇÃO

Neste capítulo faremos uma descrição do sistema proposto do ponto de vista operacional, enfocando os aspectos de utilização do programa e as características visuais dos elementos envolvidos. Será dada uma descrição dos ícones implementados, bem como dos mecanismos de criação de identificadores e outros detalhes de utilização da interface do usuário com o ambiente de desenvolvimento. Desta forma, pretende-se que este capítulo sirva como texto de apoio para os possíveis usuários deste programa.

3.1. A VISTA PRINCIPAL DO PROGRAMA

Por ser uma aplicação desenvolvida para trabalhar no ambiente Windows, o ONAGRO, do ponto de vista operacional, se comporta como as demais aplicações desenvolvidas para este ambiente.

Quando acionado, ele ativa inicialmente uma janela, chamada de *vista principal* do programa. Esta vista, como mostra a figura 3.1, é composta por:

- ☑ um *menu de linha* para seleção das tarefas que o usuário deseja realizar;
- ☑ uma *barra de ícones de operação* formada por botões pré-definidos, usados para simbolizar as instruções da linguagem;
- ☑ uma *barra de ferramentas*, também formada por botões pré-definidos, que permite a seleção de forma mais rápida, das principais tarefas disponíveis no menu de linha;
- ☑ *área de desenho* do programa-fonte;
- ☑ *linha de status* e *botões para rolagem* da área de desenho.

Através do **menu de linha** o usuário pode realizar as seguintes tarefas, selecionado a opção correspondente:

- ☑ operações usuais de acesso à memória de massa e de impressão de arquivos (opção Arquivos);
- ☑ operações de edição (opção Editar) como ligar, remover ou inspecionar ícones, alterar a grade da área de desenho, redesenhá-la e voltar à rotina principal do programa;
- ☑ ativar o editor de identificadores para manipular os símbolos usados no programa (opção Identificadores);
- ☑ inserir ícones de operação (opção Inserir);
- ☑ gerar o programa-objeto, transmiti-lo para um sistema alvo e modificar parâmetros de compilação (opção Compilar);
- ☑ configurar os recursos do microcontrolador como o sistema de interrupção, os contadores/temporizadores e o canal de comunicação serial (opção Recursos); e
- ☑ obter informações de ajuda do sistema (opção Ajuda).

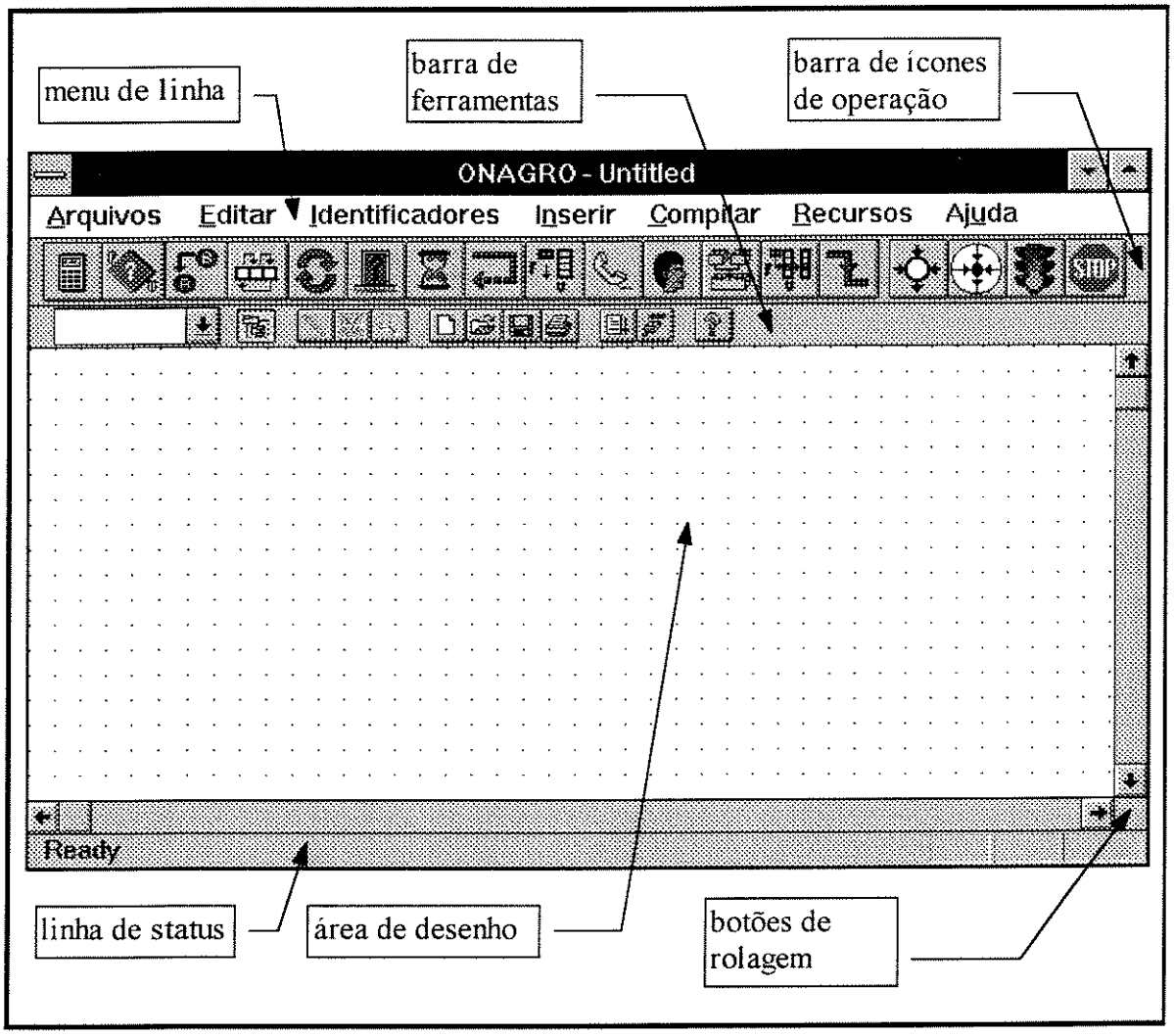


Figura 3.1: Vista principal do programa ONAGRO

A **barra de ícones de operação** agrupa os ícones que simbolizam as operações que a linguagem pode executar. Os ícones que a compõem são chamados de ícones de classe e serão relatados na seção 3.3. Esta barra de ícones serve para aumentar a rapidez na introdução dos ícones de operação no programa-fonte com o auxílio do mouse.

Como já foi dito, por meio do menu de linha podem ser realizadas todas as tarefas disponíveis no programa. Entretanto, podemos acelerar a escolha de determinadas tarefas (as mais utilizadas) por meio da **barra de ferramentas**. Ela permite ao usuário:

- ☑ selecionar qual subrotina ele deseja editar, através de uma caixa de listagem, que é iniciada com os identificadores de subrotinas;
- ☑ voltar ao programa principal, caso ele esteja editando uma subrotina;
- ☑ ligar, remover e inspecionar ícones;
- ☑ realizar operações com o disco e com a impressora;
- ☑ compilar e transmitir o programa-objeto gerado; e
- ☑ obter informações do sistema de ajuda.

A **área de desenho**, como o próprio nome sugere, é o espaço destinado para a introdução dos ícones de operação que quando ligados formarão o programa-fonte na forma de um fluxograma. Esta área da tela é marcada por pontos eqüidistantes, formando uma grade de desenho que permite melhor alinhamento dos ícones. Esta grade pode ser configurada através da opção Editar-Grade do menu de linha. Esta opção abre o diálogo visto na figura 3.2, onde pode-se modificar o tamanho da grade (2, 1 ou ½ do tamanho do ícone), indicar se a grade deve ficar visível (caixa de verificação Visualizar) e se os ícones devem ser arrastados para a grade corrente.

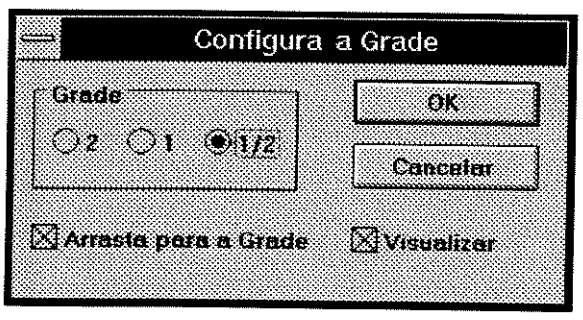


Figura 3.2: Caixa de diálogo de configuração da grade da área de desenho

Por último, temos a **linha de status** e os **botões para rolagem** que servem, respectivamente, para reportar mensagens do sistema e realizar "scroll" horizontal e vertical da área de desenho.

3.2. OS TIPOS DE IDENTIFICADORES

O ONAGRO referencia variáveis, constantes, portas de e/s e subrotinas através de *identificadores* definidos pelo usuário. Estes identificadores devem conter pelo menos um e no máximo 20 caracteres. Os caracteres permitidos para a construção dos identificadores são letras, dígitos e o caracter separador '_'. Todo identificador deve começar com uma letra, terminar com uma letra ou um dígito e não conter dois caracteres separadores seguidos. Estas regras de construção dos identificadores foram implementadas pelo compilador através do autômato finito ilustrado pela figura 3.3. Este é um conjunto básico de regras, encontrado em diversas linguagens de programação. [13]

Os identificadores são introduzidos nos ícones de operação de acordo com a operação que se deseja realizar. Entretanto, antes de usar os identificadores o usuário deve defini-los, através do editor de identificadores. Este editor divide os identificadores em seis categorias: constantes, variáveis, portas de e/s, vetores, subrotinas e parâmetros locais.

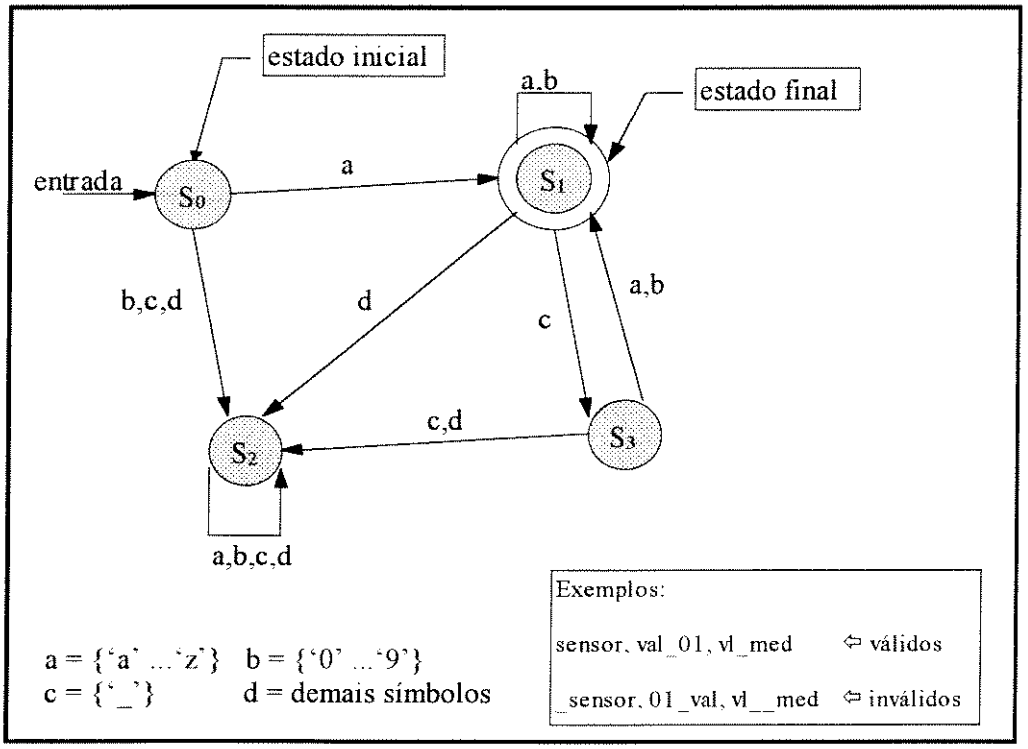


Figura 3.3: Autômato finito que ilustra as regras de construção dos identificadores

Os identificadores de constantes

As *constantes* são associadas a valores fixos que não podem ser modificados em tempo de execução. Estes valores podem ser expressos usando-se o sistema decimal ou hexadecimal. Por "default" toda constante será decimal, porém quando se desejar expressar uma constante em hexadecimal deve-se finalizá-la com um "h" e necessariamente ela deve começar com um dígito decimal.

O processo de criação de uma constante, como os demais tipos de identificadores, é bastante simples e interativo. Ativa-se o editor de identificadores, por meio do menu de linha, e seleciona-se um dos seis tipos de identificadores, no caso, a constante. Neste momento o editor de identificadores abre uma caixa de diálogo (ver figura 3.4) onde o usuário preenche o campo LABEL, que recebe o identificador propriamente dito, seleciona o tipo de dado da constante, indica seu valor e se achar conveniente faz uma descrição textual sobre a utilização da constante.

Quando o diálogo é fechado (através de um *clique* esquerdo do mouse no botão **Fechar** do diálogo ou teclando-se ENTER), o sistema realiza uma verificação das informações introduzidas e reporta ao usuário qualquer erro encontrado, como por exemplo, um identificador já existente ou fora das regras de construção ou até mesmo se o usuário esqueceu de indicar o valor da constante ou ocorreu alguma incompatibilidade entre o valor e o tipo de dado associados ao identificador. Caso as informações estejam corretas, o editor envia uma mensagem ao objeto lista de identificadores para que o novo elemento seja armazenado.

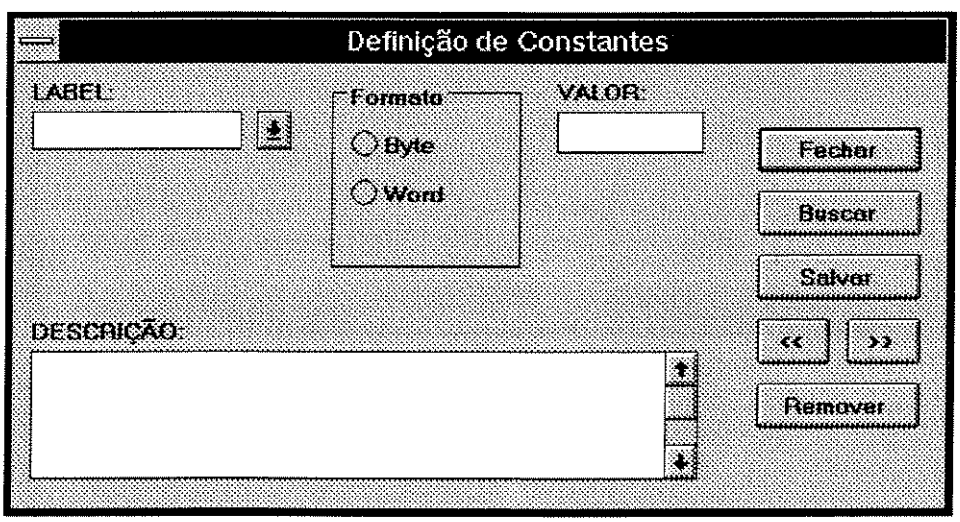


Figura 3.4: Caixa de diálogo de definição de constantes

Além de criar os identificadores, o objeto editor de identificadores permite realizar operações (através de botões incluídos no próprio diálogo) de consultas e alterações na lista de identificadores. Pode-se percorrer seqüencialmente a lista ou obter um determinado elemento a fim de realizar alguma modificação ou até mesmo removê-lo da lista.

Os identificadores de variáveis

As *variáveis*, ao contrário das constantes, são criadas para armazenar valores que podem ser modificados em tempo de execução. Seu processo de criação é semelhante ao das constantes. Após o usuário ter acionado a opção Identificadores-Variável do menu de linha, o sistema abre uma caixa de diálogo para detalhamento das características da variável que se pretende introduzir no programa. As diferenças entre este diálogo e o diálogo de descrição das constantes são: a) o campo VALOR na descrição das variáveis, é associado ao valor inicial da variável e o mesmo é opcional, pois o não preenchimento deste campo indica que a referida variável será iniciada com zero; b) existe um campo a mais, o campo ENDEREÇO. Este campo, que também é opcional, pode ser usado para possibilitar a indicação de um endereço fixo na memória, onde será alocada a variável. Todavia, na maioria dos casos, o próprio sistema gera o endereço das variáveis, sendo este campo usado pelo usuário apenas em ocasiões especiais, onde em virtude de um detalhe de *hardware*, se deseje que uma variável ocupe um posicionamento específico.

As demais informações do diálogo de descrição de variáveis (figura 3.5) são idênticas às das constantes. O mesmo ocorre em relação as críticas sobre os valores introduzidos e ações tomadas. No final do processo a nova variável é introduzida na lista de identificadores.

Os identificadores de portas de e/s

Para melhorar o discernimento dos identificadores diretamente envolvidos nas operações de entrada e saída, foi criado um tipo especial de identificador: as *portas de e/s*. Isto se justifica pelo fato da maioria das aplicações que usam microcontroladores terem uma forte interação com dispositivos de e/s. Assim, nada melhor que discriminar esses

identificadores dos demais usados no programa. Com isto aumenta-se a clareza do programa e facilita a sua elaboração.

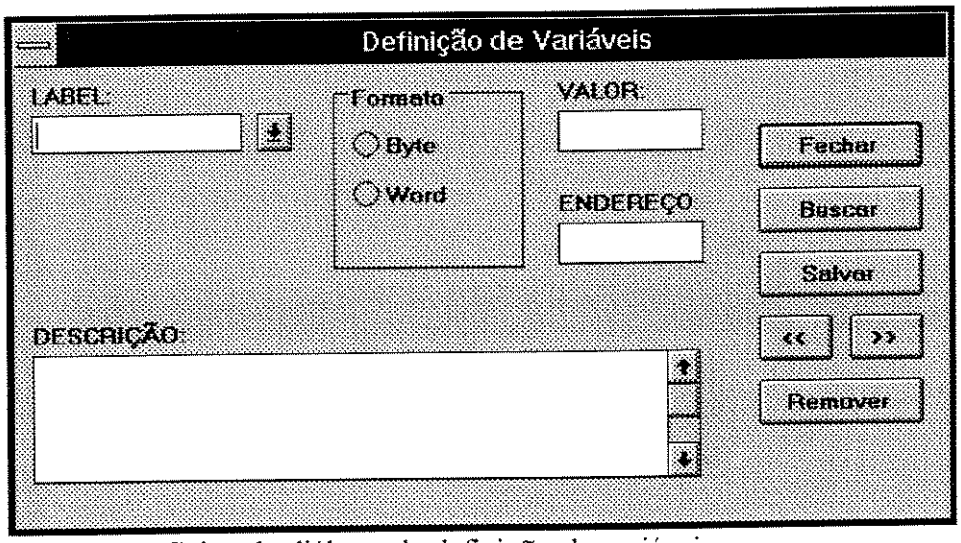


Figura 3.5: Caixa de diálogo de definição de variáveis

O diálogo relacionado às portas de e/s, como mostra a figura 3.6, possui campos especiais, onde se especifica o tipo da porta (só entrada, só saída, ou bidirecional); se a mesma será mapeada em endereço de memória externa; e quais os *bits* que serão utilizados pela porta. Os campos restantes e os botões desse diálogo são análogos aos dos diálogos das constantes e variáveis.

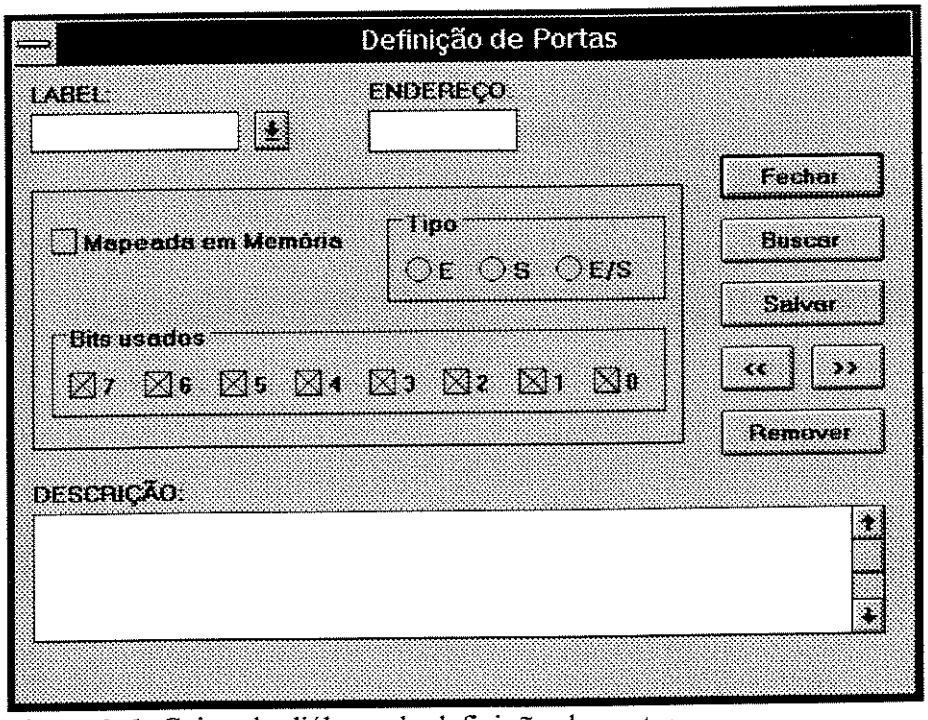


Figura 3.6: Caixa de diálogo de definição de portas

Os identificadores de vetores

Os *vetores* são um tipo particular de variável. Ao contrário das variáveis comuns, que armazenam apenas um valor, eles são constituídos por um conjunto de até 256 elementos, todos do mesmo tipo. Estes elementos são referenciados pelo mesmo nome, o identificador do vetor. Porém, cada elemento do vetor possui um índice que o identifica dos demais. O índice usado para indicar um determinado elemento do vetor varia de zero até o limite determinado ao vetor, sendo geradas mensagens de erros caso se queira obter um elemento cujo índice seja igual ou superior ao tamanho do vetor.

Devido a pequena quantidade de memória interna disponível nos microcontroladores e ao grande número de instruções para obter um elemento de vetores polidimensionais, optou-se em fazer uma restrição quanto à criação dos vetores, sendo apenas permitido a criação de vetores unidimensionais e que, conforme já mencionado, não ultrapassem a 256 elementos. Em contra partida, foram criados ícones especiais de operação para manipulação de vetores unidimensionais, permitindo de forma prática (e sem geração excessiva de código) o relacionamento entre vetores, o que possibilita a implementação de aplicações que necessitariam de vetores polidimensionais.

Além dos campos existentes no diálogo de descrição de variáveis, o diálogo de descrição de vetores (ver figura 3.7) possui um campo extra para que o usuário indique o tamanho (número de elementos) do vetor. Outra diferença deste diálogo em relação aos primeiros é que o preenchimento do valor inicial deve fornecer valores (separados por vírgula) para todos os elementos do vetor. Como este campo é opcional, o programador pode deixá-lo vazio, neste caso o editor de identificadores iniciará todos os elementos com zero. Ainda com relação ao valor inicial, a linguagem permite que se inicie um vetor com uma seqüência de caracteres delimitados por aspas duplas. Isto é útil quando se deseja criar os tradicionais *strings* de caracteres.

Os identificadores de subrotinas

Finalmente, temos os identificadores de *subrotina* que servem para identificar cada subrotina criada no programa. É por meio deste identificador que o programador referencia

as subrotinas, seja para realizar uma simples chamada ou para modificar qualquer característica da mesma.

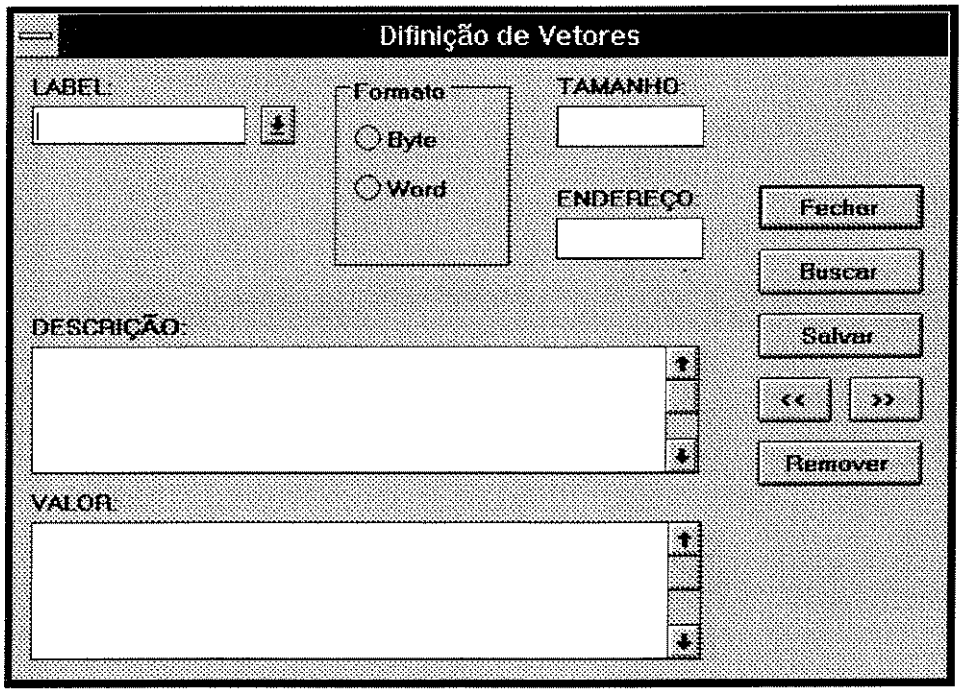


Figura 3.7: Caixa de diálogo de definição de vetores

Quando se seleciona, por meio do menu de linha, a opção Identificadores-Subrotina, o editor de identificadores abre uma caixa de diálogo para descrição das características da subrotina que se quer manipular. Neste diálogo podem ser introduzidas ou modificadas informações quanto ao tipo da subrotina, quanto às características do valor por ela retornado e quanto aos parâmetros envolvidos na sua chamada. O diálogo permite ainda associar a esta subrotina um ícone especial a fim de melhor evidenciar o propósito de suas operações.

A figura 3.8 mostra o diálogo usado para definição de subrotinas. Este diálogo possui campos semelhantes aos identificadores já mencionados, o que torna suas descrições dispensáveis. Entretanto, há outros campos específicos. Um deles é o de detalhamento do valor de retorno, onde se descreve uma variável, usada localmente na subrotina, para armazenar o valor de retorno. Outro é uma simples caixa de verificação que indica se a referida subrotina é ou não de tratamento de interrupção. Esta informação é útil ao compilador, pois a codificação das subrotinas de tratamento de interrupção envolve detalhes não existentes nas demais subrotinas.

Há também neste diálogo, um campo para que se possa personalizar o ícone de chamada da subrotina. Por "default" existe um ícone padrão de chamada de subrotina. Entretanto, ele pode ser modificado para cada nova subrotina criada. Para isto, basta que o usuário indique o nome do arquivo que contém ou conterá o ícone personalizado (campo **Arq. ICO**) e marque a caixa de verificação **Personalizar**. Após isto, o editor de identificadores ativa o editor de ícones, onde o usuário pode desenhar seu próprio ícone ou gerá-lo a partir de bibliotecas pré-gravadas. Esta possibilidade de personalização do ícone de chamada de subrotina é muito importante, pois ninguém melhor que o próprio programador para indicar um símbolo que evidencia de forma clara e objetiva a natureza da função realizada por aquela subrotina. É claro que isto envolve um pouco mais de trabalho. Todavia, o esforço compensa em virtude do aumento de clareza e facilidades de manutenção do programa.

Até o momento, tratamos as subrotinas apenas do ponto de vista de identificação, enfocando as informações necessárias para definir seus cabeçalhos. Todavia, no item "operação com subrotina" da seção seguinte, mostraremos os ícones de operação associados às subrotinas e retomaremos a questão da personalização do ícone de chamada.

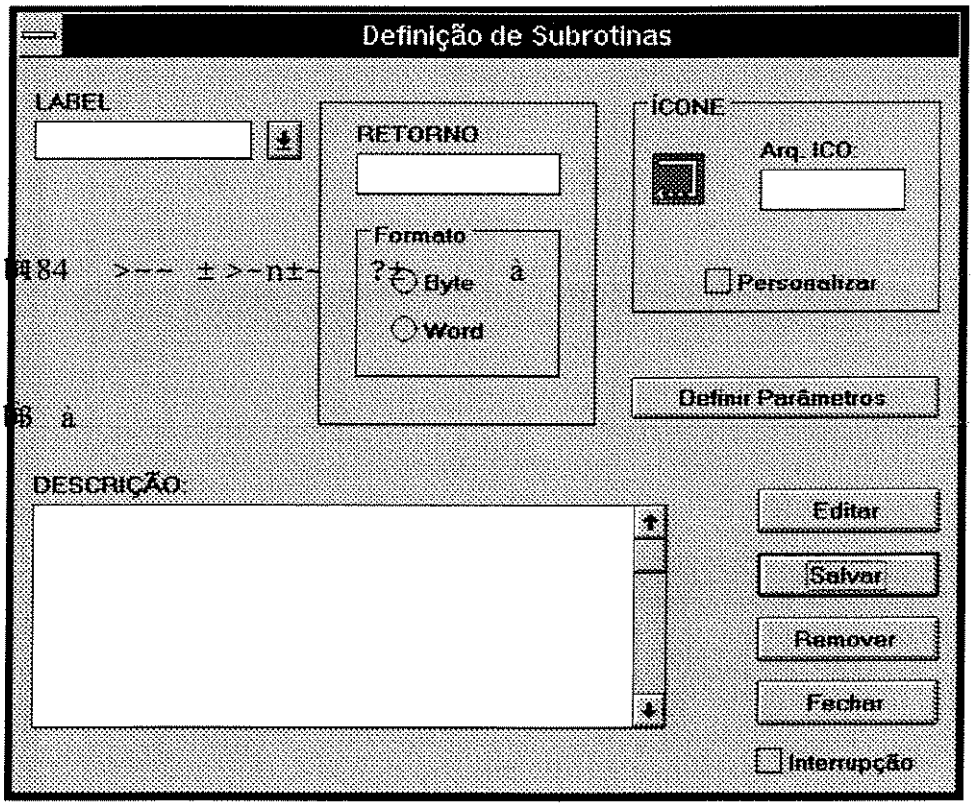


Figura 3.8: Caixa de diálogo de definição de subrotinas

Caso a subrotina que está sendo criada ou modificada necessite de parâmetros de entrada, pode-se acionar um outro diálogo específico para descrição dos argumentos da subrotina. Este diálogo, que pode ser visto na figura 3.9, é ativado através do botão **Definir Parâmetros** e será descrito no próximo item.

Por último, o diálogo de descrição de subrotinas pode acionar o editor de fluxogramas para que sejam introduzidos ou modificados os ícones de operação que implementarão a subrotina.

O diálogo de definição de identificadores de subrotina, como visto, é um pouco maior que os dos demais identificadores. Porém, deve ser observado que nele são introduzidas todas as informações relativas aos dados que a subrotina comporta. Isto não representa uma desvantagem do ONAGRO em relação às outras linguagens para microcontroladores, pois nas mesmas o programador explicita de forma textual quase todas essas informações, não recebendo do ambiente de programação nenhum direcionamento que o ajude a evitar erros. Por outro lado, este diálogo organiza essas informações, se constituído num roteiro preliminar para criação dos cabeçalhos de subrotina.

Os identificadores de parâmetros locais

Muitas vezes quando usamos subrotinas é necessário passar-lhes parâmetros a fim de aumentar seu grau de generalidade. Esse parâmetros representam o meio pelo qual a parte do programa que ativou uma determinada subrotina possa indicar a esta, quais os dados que devem ser envolvidos nas operações executadas por ela. A passagem dos argumentos indicados na chamada da subrotina para seus parâmetros formais, ocorre pela simples transferência de valor. Ou seja, o valor de cada argumento usado na chamada da subrotina é passado ao seu respectivo parâmetro formal, que tem escopo local e é armazenado na pilha do microcontrolador. Cada parâmetro da subrotina deve ser definido através do diálogo visto na figura 3.9.

Neste diálogo o programador apenas indica o LABEL e o FORMATO de cada parâmetro usado na subrotina e seleciona, por meio dos botões **Inserir** e **Remover**, a operação de inserção ou remoção do parâmetro, respectivamente. Ele é ativado, como já dito, pelo próprio diálogo de definição de subrotinas.

Vale ressaltar que estes parâmetros são um tipo especial de variável. Eles são variáveis locais à subrotina, para a qual eles foram definidos e desta forma só podem ser manipulados pelos ícones de operação da mesma. Porém, isto fica transparente ao usuário, pois o sistema inicia automaticamente as listas dos valores usados nos diversos diálogos de descrição de operação, respeitando o escopo destes valores. Assim, se facilita a descrição dos ícones e evita erros comuns de compilação, como ocorre nas linguagens tradicionais de programação de microcontroladores.

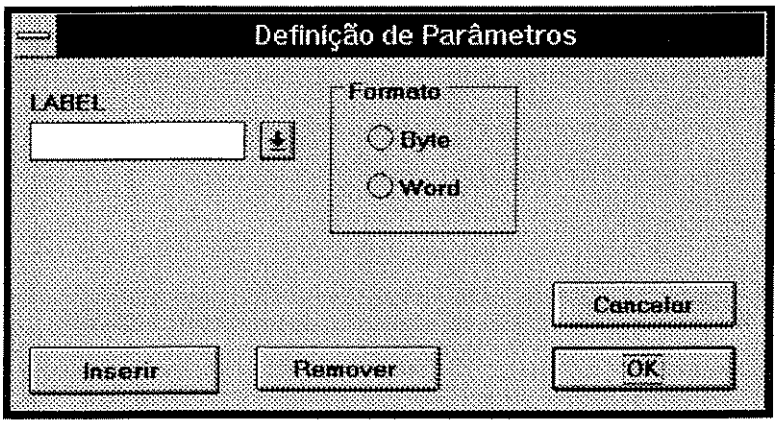


Figura 3.9: Caixa de diálogo de definição de parâmetros

3.3. OS ÍCONES DE OPERAÇÃO

Os *ícones de operação* são símbolos gráficos usados para representar as operações que podem ser executadas pela linguagem. Durante sua elaboração houve a preocupação em torná-los o mais intuitivo possível. A maioria deles foi derivada da própria simbologia usada nos fluxogramas. Entretanto, alguns ícones mais específicos foram criados para possibilitar uma gama maior de operações, como por exemplo: comunicação serial, pesquisa em vetores e geração de atrasos, dentre outros.

Eles estão organizados em classes de operações, onde cada classe é representada por uma figura diferente e indica um conjunto de operações semelhantes. São disponíveis, por exemplo, quatro operações de atribuição (atribuição propriamente dita, atribuir zero, atribuir um e permutar dados) agrupadas numa mesma classe. Na maioria dos casos o ícone que representa a classe é o mesmo que indica a operação mais usada daquela classe (veja figura 3.10). No entanto, existem classes que possuem um ícone específico para designá-la, diferente dos demais ícones de operação da classe. É o que ilustra a figura 3.11.

É importante ressaltar que mesmo havendo diferentes ícones numa classe, eles têm características visuais semelhantes. A cor de preenchimento é a mesma e a simbologia entre eles só muda no que eles têm de peculiar. Isto pode ser observado na figura 3.10. Os últimos quatro quadros representam ícones diferentes relacionados com a operação de atribuição. Existem pequenas, mas importantes, mudanças no desenho a fim de diferenciá-los. Em contra partida, existe ao mesmo tempo entre eles uma semelhança que nos leva a perceber que eles pertencem a mesma classe.

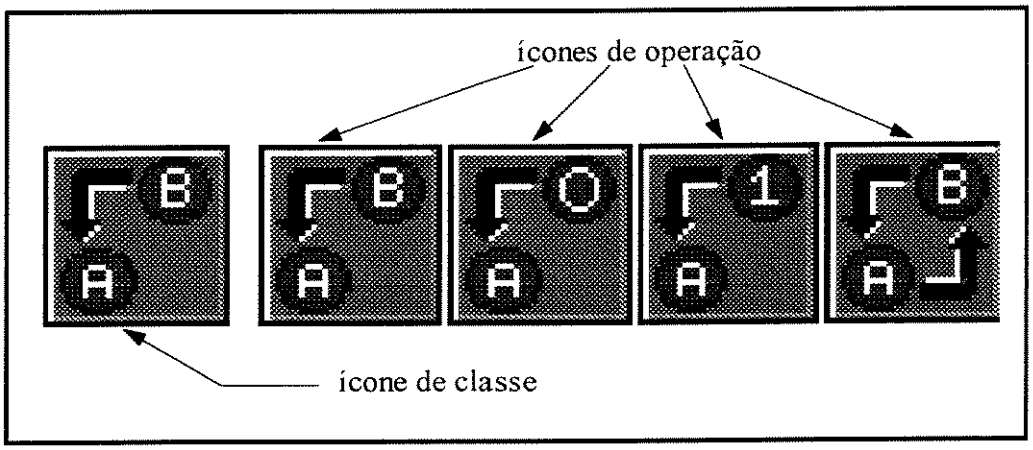


Figura 3.10: Ícones de atribuição

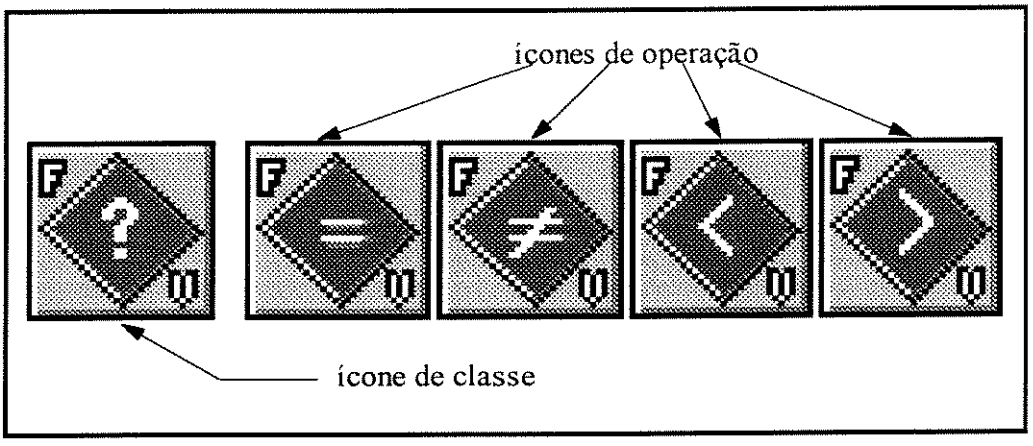


Figura 3.11: Ícones de comparação

Alguns ícones, de classes distintas, possuem a característica de poder mudar o fluxo de execução do programa. Muitas vezes estas mudanças (paradas, instrução de repetição, chamadas e retorno de subrotinas, principalmente) dificultam o entendimento e a depuração do programa. Para diminuir os inconvenientes pertinentes às mudanças no fluxo de

execução, adotou-se a mesma cor de fundo para a esses ícones, o que melhora a clareza do programa.

Para se inserir um ícone de operação na área de desenho, basta que o usuário entre no modo de inserção (opção Inserir do menu de linha) e escolha a classe do ícone desejado. O usuário pode poupar tempo indicando o ícone a ser inserido, por meio de um *clique* do botão esquerdo do mouse sobre o ícone correspondente na barra de ícones de operação.

Após inserir um ícone de classe na área de desenho, o usuário ativa uma caixa de diálogo para especificar os detalhes da operação. Isto é feito pressionando-se duas vezes seguidas o botão esquerdo do mouse com o cursor posicionado dentro do ícone. Neste diálogo, o usuário é conduzido a indicar a operação, dentre as disponíveis na classe, que ele deseja realizar e preencher os campos de dados usados na operação selecionada. Além disto, todos os diálogos possuem um campo destinado à documentação, onde o usuário pode descrever textualmente o que a operação vai realizar. Esta informação é útil quando o usuário desejar depurar o programa, através do modo de inspeção. Este modo é ativado, conforme dito no início deste capítulo, pela opção Editar-Inspecionar do *menu* de linha ou através da barra de ferramentas.

O preenchimento dos campos é dirigido pelo diálogo, de tal forma, que o usuário possa conhecer os valores permitidos para aquele campo, dentro do contexto da operação. Para isso o diálogo dispõe, para cada campo de dado, de uma lista com os valores permitidos.

Alternativamente, o usuário pode preencher os campos do diálogo de forma direta, digitando os identificadores de símbolos envolvidos na operação. Mesmo assim, o diálogo só aceita os valores permitidos pela operação, sendo geradas mensagens de erro quando o usuário tentar fechar o diálogo, caso tenha sido cometido algum erro.

Assim, logo no detalhamento das operações, procura-se evitar erros comuns que ocorrem nas linguagens tradicionais, onde o programador tem que ficar atento a "contextualidade" da operação, no que se refere aos seus parâmetros. Normalmente isto é fonte de erros posteriores de compilação.

Para fechar o diálogo o usuário pode confirmar as informações inseridas, usando o botão OK, ou cancelá-las através do botão CANCELAR. No encerramento da caixa de diálogo o sistema faz uma crítica das informações introduzidas notificando qualquer

irregularidade sobre elas, como por exemplo, um aviso sobre um identificador que ainda não tenha sido definido.

Imediatamente após o fechamento do diálogo, o ícone de classe é modificado para indicar a operação selecionada. O usuário pode também mudar o ícone de operação pressionando o botão direito do mouse duas vezes seguidas, com o cursor posicionado dentro do ícone.

Nos ícones, existem quatro áreas para conexão, como mostra a figura 3.12. Na maioria dos casos, eles possuem uma área de saída de ligação e três de entrada. A ligação se faz quando o usuário entra no modo de ligação (opção Editar-Ligar do menu de linha ou através do segundo botão da barra de ferramenta) e pressiona o botão esquerdo do mouse numa das áreas livres do ícone inicial, arrastando-o até uma área livre do ícone final.

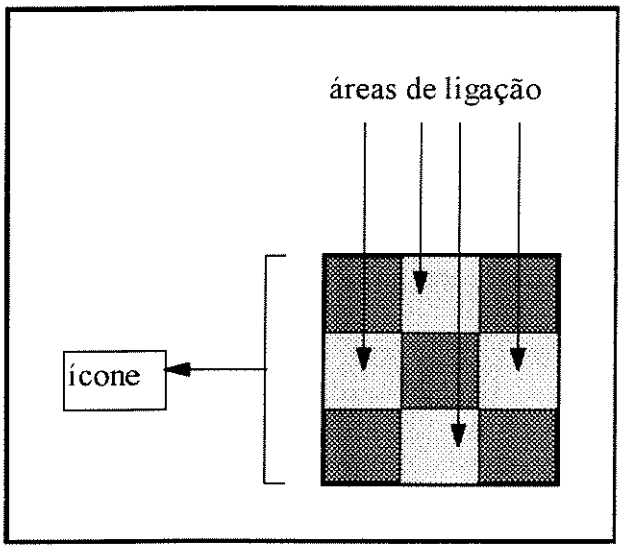


Figura 3.12: Áreas do ícone usadas para ligação

Após ter sido introduzido o programa-fonte na área de desenho, pode-se ativar o compilador para que o mesmo gere o correspondente código objeto. Isto é feito de duas formas: a primeira é através do menu de linha usando a opção Compilar-Começar e a segunda é por meio da barra de ferramentas, ativando seu terceiro botão da direita para a esquerda.

Feito isto, o compilador abre uma caixa de diálogo, mostrada na figura 3.13, e reporta os erros ocorridos ou, caso não seja detectado erros, mostra o código em Assembly do referido programa-fonte. Neste momento, o usuário pode pressionar o botão OK, fazendo com que o compilador ative um montador/linkeditor padrão para a geração do

código em linguagem de máquina e armazene, em disco, os arquivos gerados: o arquivo com o programa em linguagem Assembly (.ASM), o arquivo de listagem (.LST), o arquivo objeto (.OBJ), o arquivo em binário (.BIN) e o arquivo no formato hexadecimal (.HEX).

Caso tenha sido gerado algum erro de compilação, o compilador conduz o usuário ao diálogo de descrição do ícone de operação onde ocorreu o erro e permite que se faça as correções necessárias. Após isto, pode-se ativar o botão **Recompilar** do diálogo de compilação, solicitando ao compilador que recomece o processo de tradução.

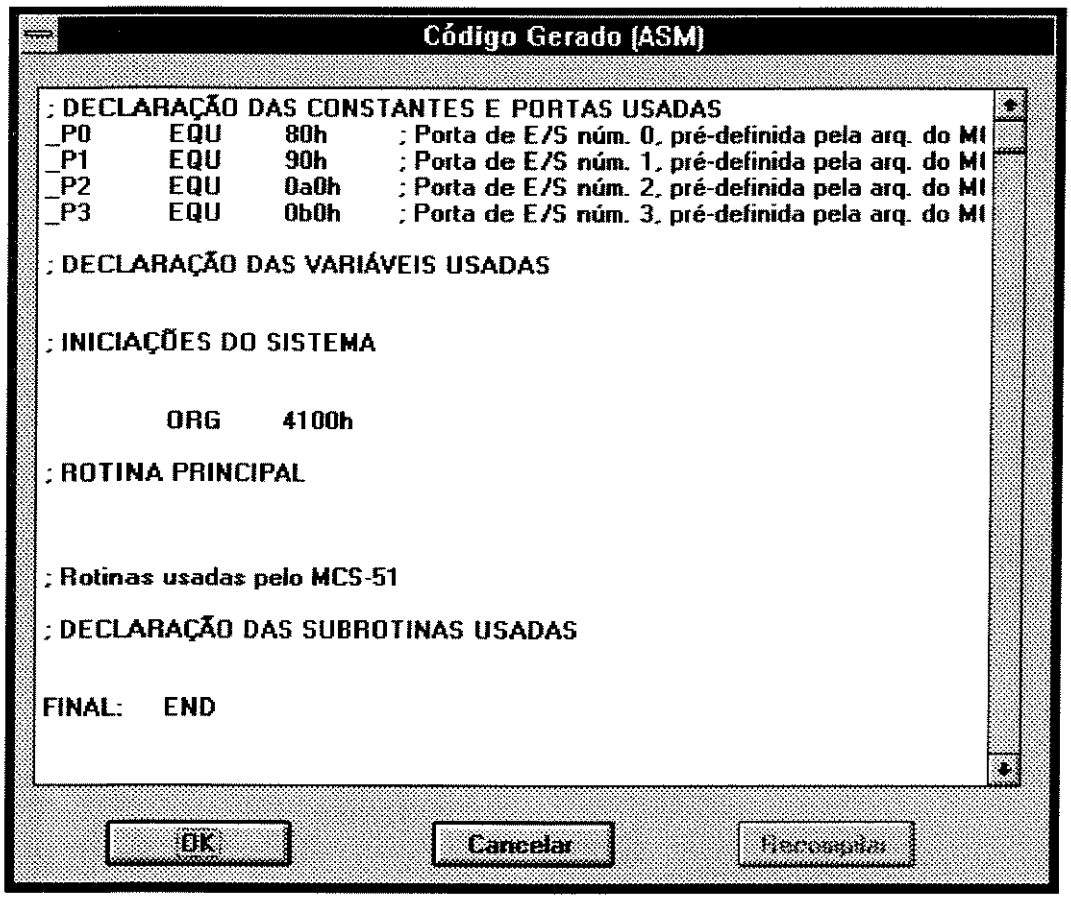


Figura 3.13: Caixa de diálogo do resultado da compilação

Apesar das verificações feitas logo na descrição dos ícones de operação, podem ser gerados erros posteriores de compilação, como por exemplo: tipos misturados na instrução de comparação, valor do campo destino inválido, incompatibilidade entre parâmetros, dentre outros. Tais erros podem ser conseqüência de alterações posteriores feitas em identificadores usados na operação onde foi detectado o erro.

Operação de atribuição

Como mencionado anteriormente, existem quatro operações de atribuição disponíveis na linguagem: atribuição propriamente dita, atribuir zero, atribuir um e permutar dados. No diálogo da operação de atribuição (figura 3.14), o campo DESTINO possui uma lista de identificadores de símbolos que podem receber valores (variáveis, portas de entrada e portas de entrada/saída). Já o campo ORIGEM do mesmo diálogo terá sua lista de valores modificados de acordo com a operação selecionada. Na operação de atribuição propriamente dita, esse campo tem sua lista composta por constantes, variáveis e portas de entrada e/ou saída. Entretanto, na operação de permuta de dados (SWAP) essa lista é também formada apenas por identificadores que possam receber valores. Por último, nas operações de ZERAR e SETAR o campo ORIGEM é "desabilitado", pois o valor de origem está implícito na própria operação.

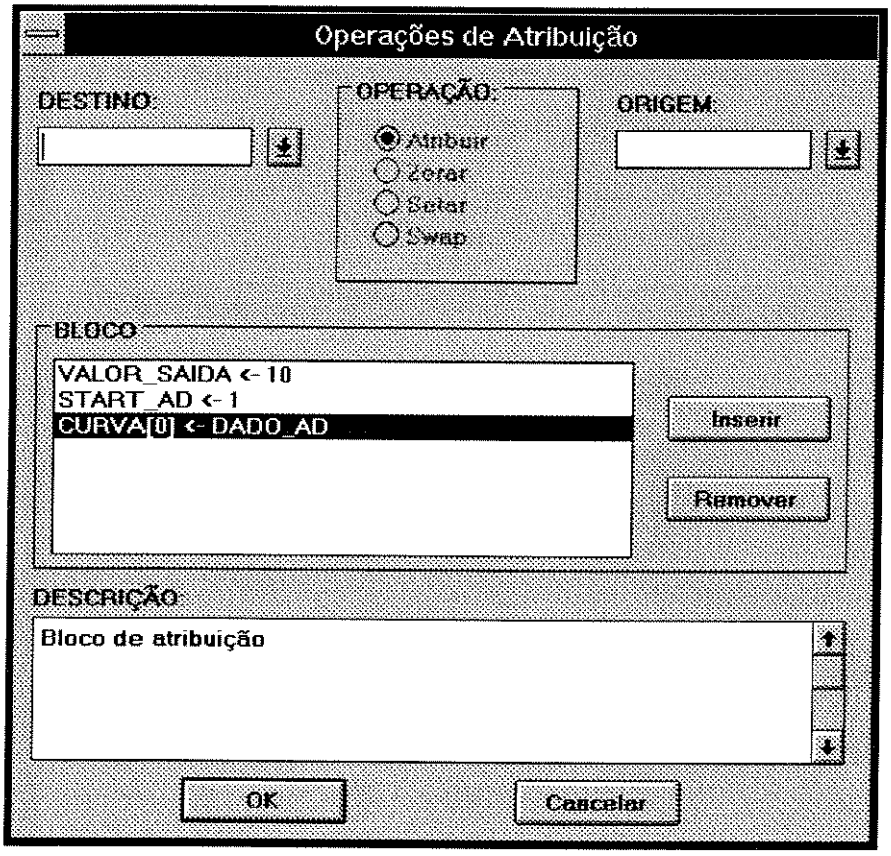


Figura 3.14: Caixa de diálogo das operações de atribuição

Caso seja envolvido algum identificador de porta de e/s na operação de atribuição, o ícone de operação é modificado para realçar que a operação manipula uma porta de e/s, conforme mostra a figura 3.15.

Como a operação de atribuição é uma das mais comuns dentro de um programa, seu diálogo dispõe de um campo opcional, onde o usuário pode criar um bloco de atribuições, que serão executadas uma após a outra. Isto evita repetição de ícones de atribuição, pois um único ícone pode representar um conjunto destas operações.

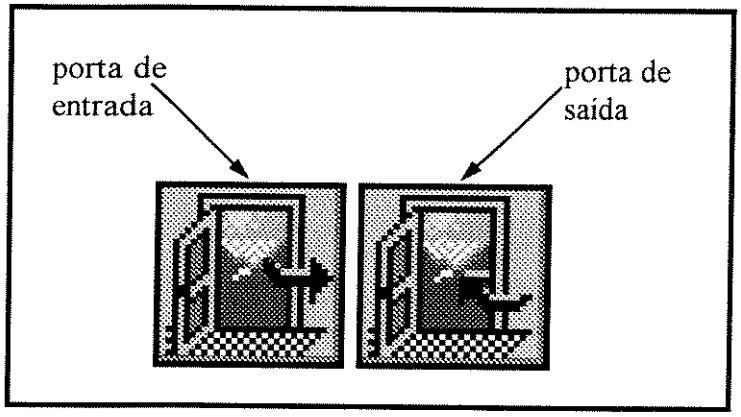


Figura 3.15: Ícone de operação com portas de e/s

Caso se deseje criar um bloco de atribuições, deve-se selecionar a operação e os identificadores desejados e ativar o botão INSERIR para cada uma das operações do bloco. Pode-se também eliminar uma operação do bloco através da seleção da operação que se pretende excluir e da ativação do botão REMOVER ou simplesmente, pressionado-se duas vezes seguidas o botão direito do mouse sobre a operação que se quer eliminar.

O diálogo mostrado na figura 3.14 possui um bloco com três operações de atribuição, todas elas representadas por um único ícone.

Operações aritméticas

As quatro operações aritméticas: soma, subtração, multiplicação e divisão possuem ícones próprios (ver figura 3.16) e executam cálculos entre dois operandos, levando o resultado para um identificador que possa receber valor (variáveis, portas de saída e portas bidirecionais). Assim, o diálogo de descrição destas operações (figura 3.17) inicia as caixas de listagens dos campos RESULTADO, VALOR1 e VALOR2 com identificadores

compatíveis. Porém, nos casos de tipos misturados de valores, o compilador assume o tipo base como sendo o do identificador que receberá o resultado. Desta forma, pode haver truncamento de valores, contribuindo para a geração de resultados não esperados.

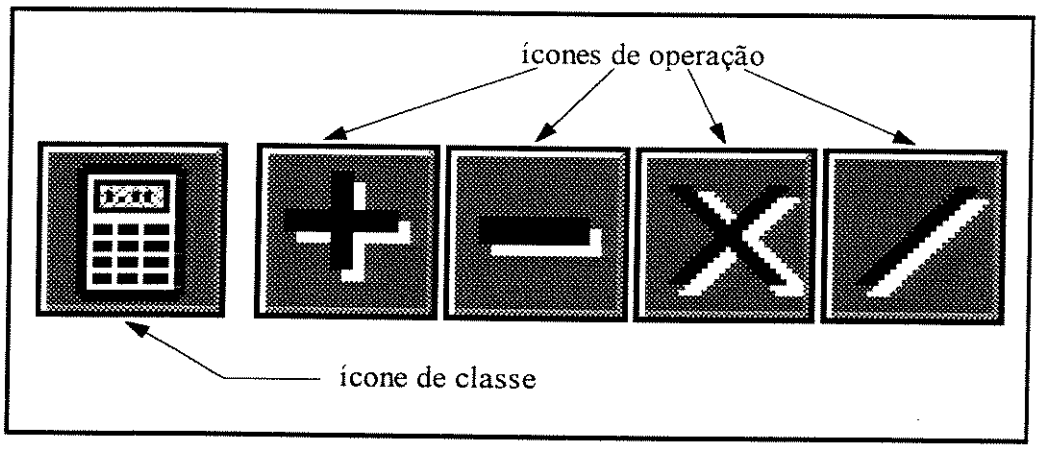


Figura 3.16: Ícones de operações aritméticas

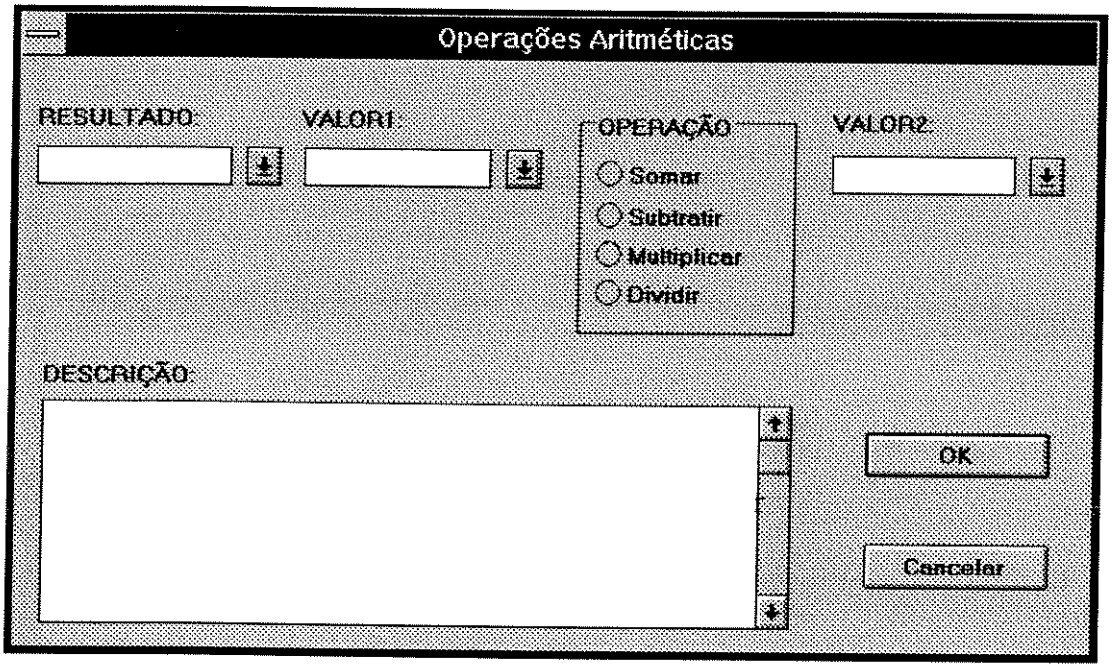


Figura 3.17: Caixa de diálogo das operações de aritméticas

Operações lógicas

Os ícones de operações lógicas são usados para realizar as operações lógicas *not*, *or*, *and* e *xor* bit a bit entre dois operandos. Da mesma forma como ocorre nas operações aritméticas, o resultado de uma operação lógica deve ser conduzido para um identificador

onde este resultado possa ser armazenado. Estas operações lógicas não geram fluxos condicionais de execução. Desta forma, os ícones (figura 3.18) só apresentam um ponto de saída. Este tipo de operação é muito útil para mascarar *bits* dentro de uma palavra onde nem todos os *bits* são efetivamente utilizados num processamento.

Na figura 3.19 temos a vista do diálogo relativo às operações lógicas. Ele é semelhante ao das operações aritméticas. A diferença está nos tipos de operadores envolvidos; porém, os dois possuem as mesmas restrições quanto ao procedimento de preenchimento dos campos.

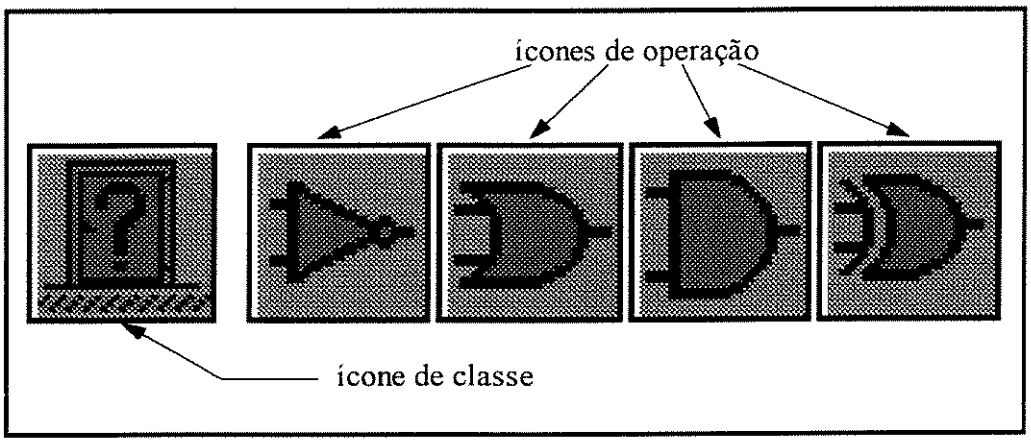


Figura 3.18: Ícones de operações lógicas

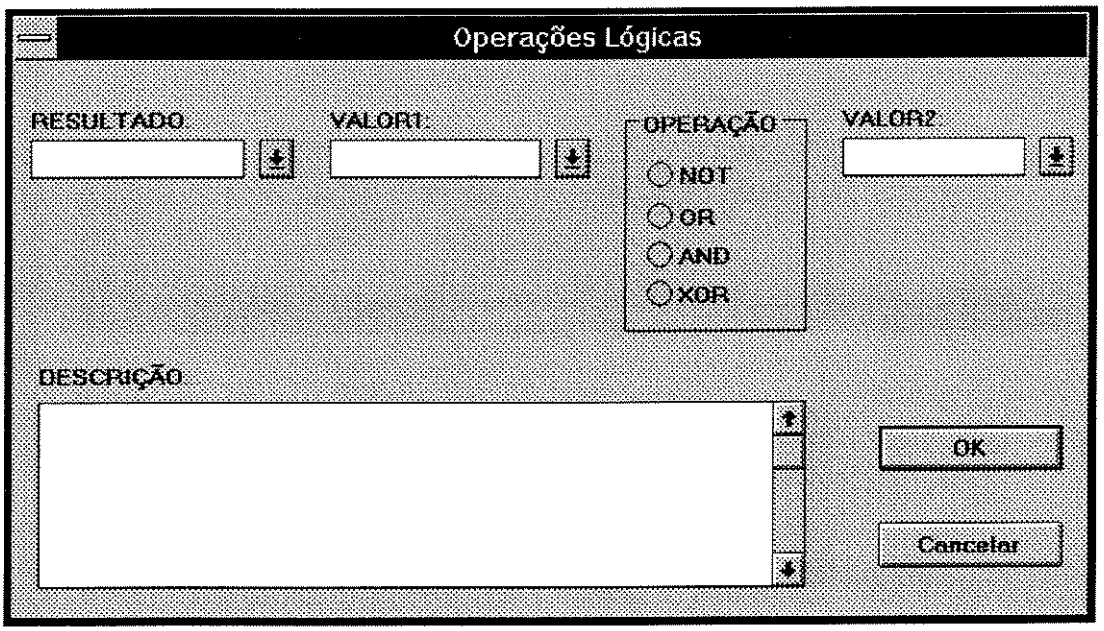


Figura 3.19: Caixa de diálogo das operações lógicas

Operações de rotação e deslocamento

Foram implementadas as operações tradicionais de rotação e deslocamento disponíveis na maioria de microcontroladores. Estas operações permitem realizar rotação e deslocamento, em ambos os sentidos, de identificadores do tipo variável ou portas de e/s.

Os ícones que representam estas operações podem ser vistos na figura 3.20. Podemos observar que nas operações de deslocamento (segundo e terceiro ícones) há a entrada de um zero no LSB (deslocamento a esquerda) e no MSB (deslocamento a direita). Os últimos dois ícones simbolizam as operações de rotação a esquerda e a direita, respectivamente.

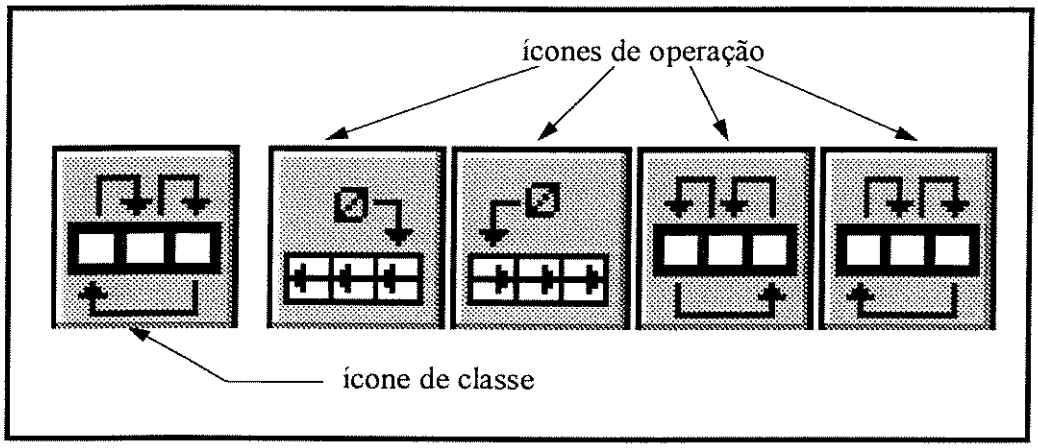


Figura 3.20.: Ícones de rotação e deslocamento

Para descrever a operação desejada, basta abrir o diálogo de preenchimento e indicar o identificador (campo LABEL) e o tipo de operação de que se quer executar, conforme figura 3.21.

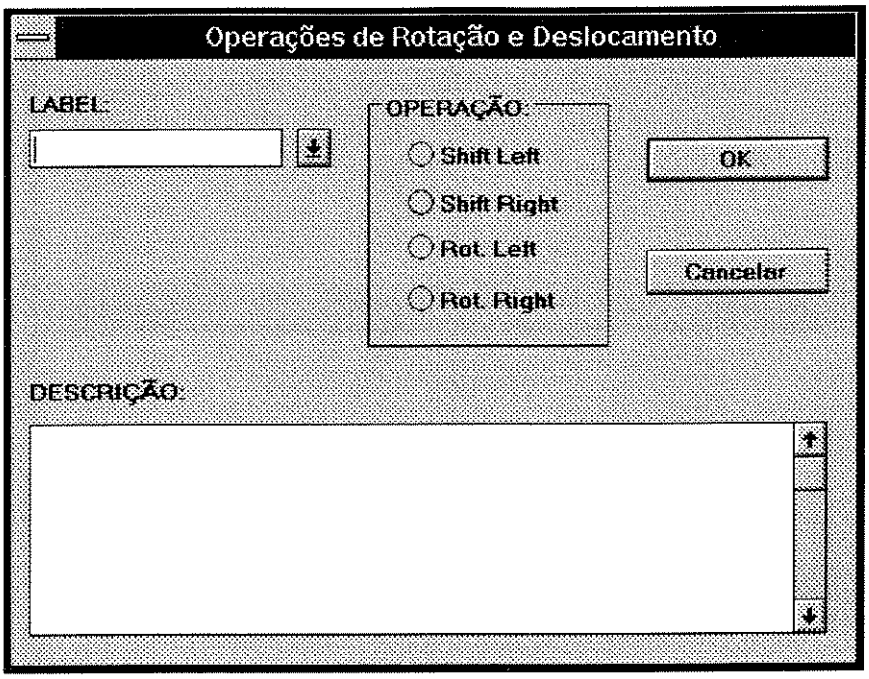


Figura 3.21.: Caixa de diálogo das operações de rotação e deslocamento

Operações de comparação

Conforme mostrado na figura 3.11, as tomadas de decisão usam os ícones de comparação: *igual a*, *diferente de*, *menor que* e *maior que*, que implementam os operadores relacionais =, **!**=, < e >, respectivamente. Desta forma, constroi-se as condições lógicas simples. Estes ícones podem ainda ser encadeados para formar condições compostas com AND, OR e NOT através de adequada interligação entre eles, como será mostrado posteriormente.

As condições lógicas são discriminadas por um diálogo próprio, visto na figura 3.22, e relaciona dois valores de mesmo formato através da indicação de uma das quatro operações possíveis. O preenchimento dos campos VALOR1 e VALOR2 pode ser feito diretamente usando o teclado ou através das caixas de listagens que contêm a lista de identificadores permitidos.

Os ícones de comparação, quando adequadamente combinados com outros ícones, podem ser usados para implementar as estruturas clássicas de decisão (*if-else*) e de repetição condicional (*while-do* e *do-while*) como mostra as figuras 3.23 e 3.24.

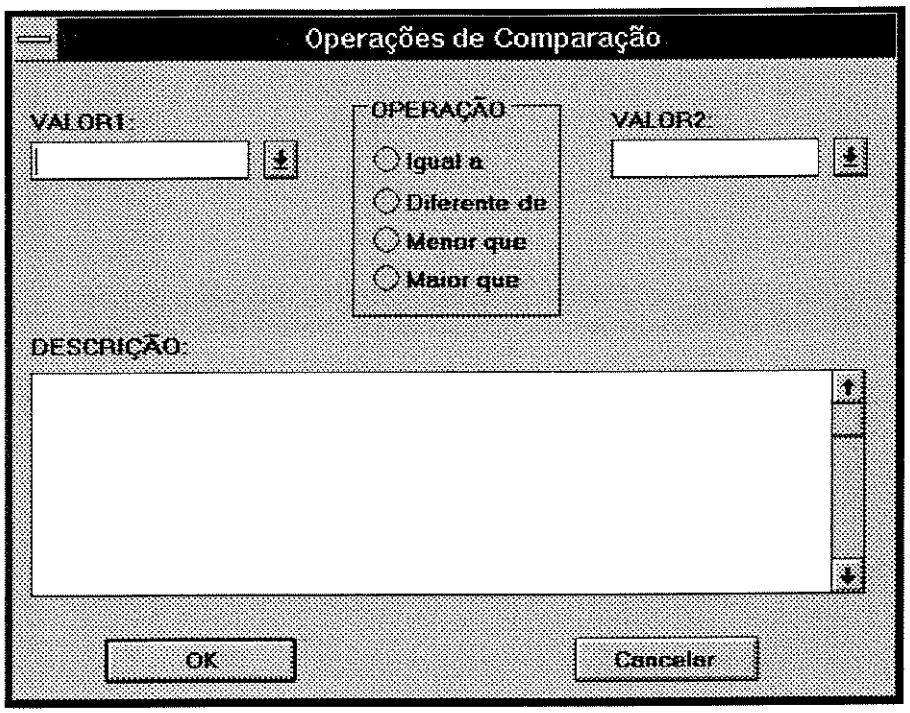


Figura 3.22: Caixa de diálogo das operações de comparação

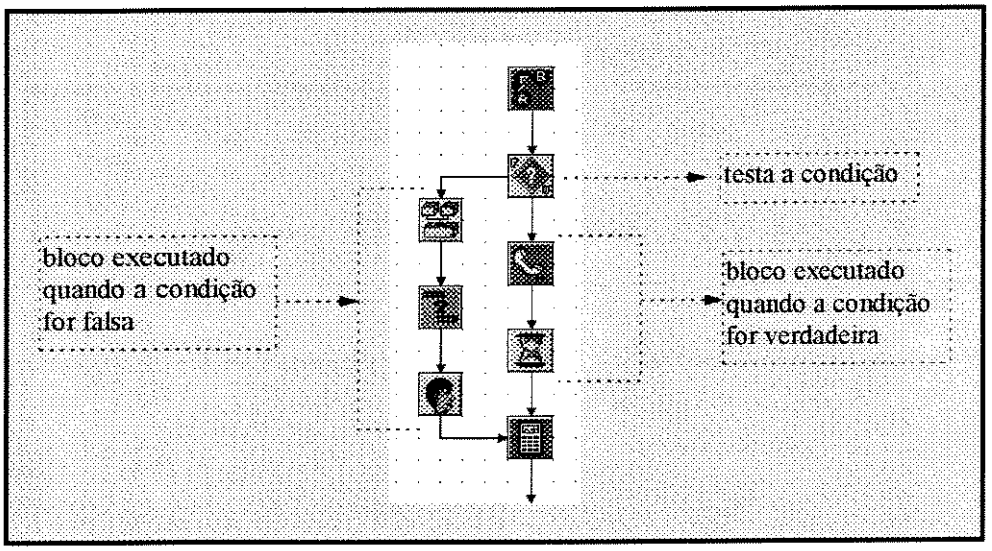


Figura 3.23: Estrutura ***if-else***

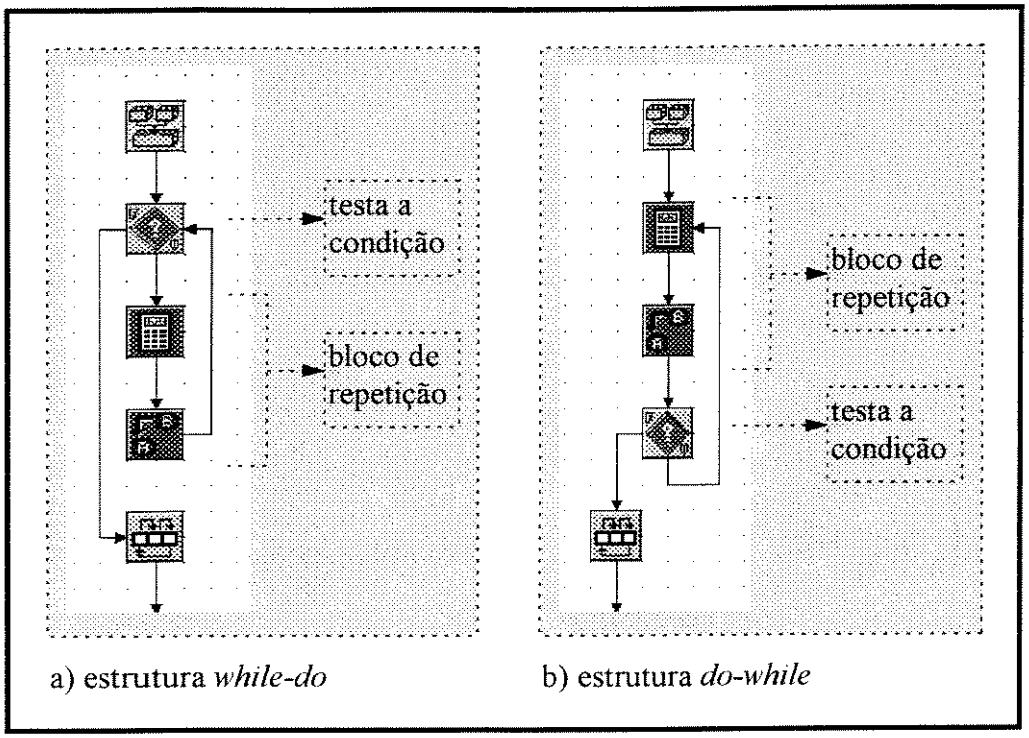


Figura 3.24: Estruturas ***while-do*** e ***do-while***

Para implementar condições compostas, que envolvem expressões relacionais com operadores lógicos, deve-se combinar ícones de comparação tomando por base o fluxo de execução. Assim, se tivermos que implementar uma condição composta, por exemplo **X and Y**, teremos dois ícones de operação. Um para cada condição simples (X e Y). A figura 3.25 mostra como essas duas condições podem ser combinadas para formar a condição completa. Note que o ponto C só será executado quando as duas condições forem verdadeiras.

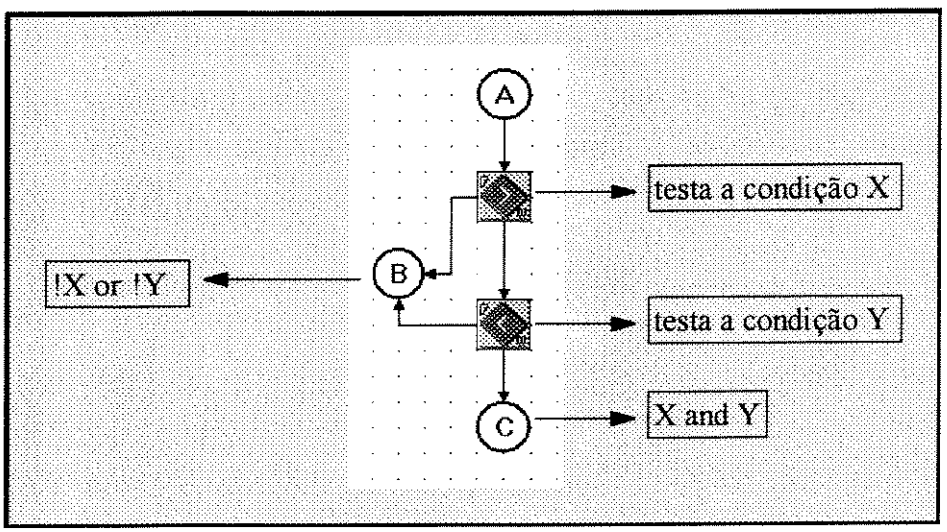


Figura 3.25: Exemplo de uma condição composta

Os ícones de comparação, diferentemente da maioria, possuem duas saídas em locais fixos indicados pelas letras V e F (ver figura 3.11). Uma para sinalizar o fluxo de execução no caso de a condição ser verdadeira e outra para o caso da condição ser falsa.

Operação de iteração

Para melhor visualização dos blocos que são executados repetidas vezes, através de "loops" de iterações, foram criados os ícones delimitadores de blocos de repetição, vistos na figura 3.26. O primeiro ícone é o ícone de classe, representando tanto o início como o final do bloco de repetição. Os outros dois são os ícones de delimitação de bloco propriamente dito. Eles sempre trabalham em conjunto, sendo permitido o aninhamento de blocos de repetição. Assim, para que não ocorra erro de compilação o número de ícones delimitadores de blocos deve ser sempre par, ou seja, o bloco deve começar e terminar por um delimitador de bloco. Estes ícones são semelhantes aos símbolos tradicionais BEGIN e END, usados nas linguagens textuais para marcação de blocos.

A repetição é controlada por uma variável de controle designada no diálogo da operação de iteração (figura 3.27). Esta variável pode ser automaticamente gerada pelo próprio ícone de operação. Para isto, basta deixar em branco o campo VAR. CTRL. Esta variável pode ser usada normalmente pelo programa para indicar a iteração corrente. O próprio programador, no entanto, deve tomar cuidado para não alterá-la, o que pode gerar possíveis erros de execução.

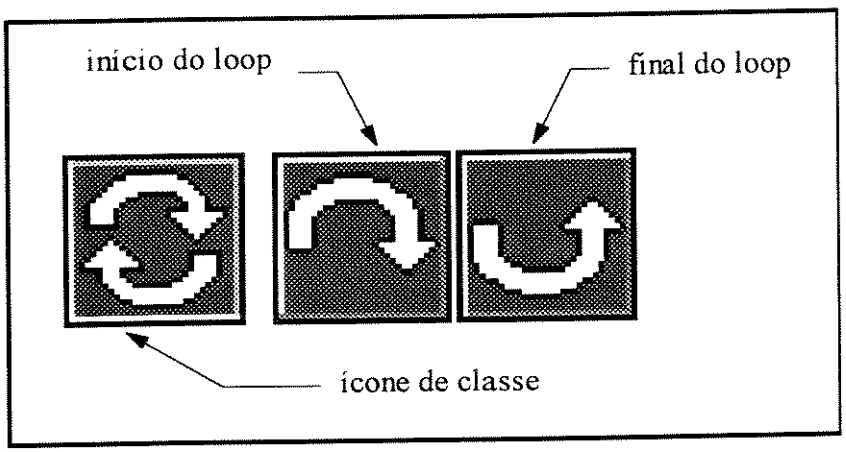


Figura 3.26: Ícones de operação de iteração

As opções presentes no diálogo referem se ao tipo de atualização que será feito na variável de controle, que pode ser incrementada ou decrementada, e a indicação do tipo de ícone (início ou final de bloco). Também é necessário que o usuário digite no campo NUM. REP. um valor inteiro ou um identificador que indicará o número de repetições do bloco. Vale salientar que para terminar um bloco só é preciso que se selecione o ícone de final de bloco, sendo os valores restantes automaticamente inseridos pelo compilador. Para evitar prováveis erros de compilação ou mesmo de lógica, os campos inseridos pelo compilador no ícone de final de bloco ficam desativados para o usuário.

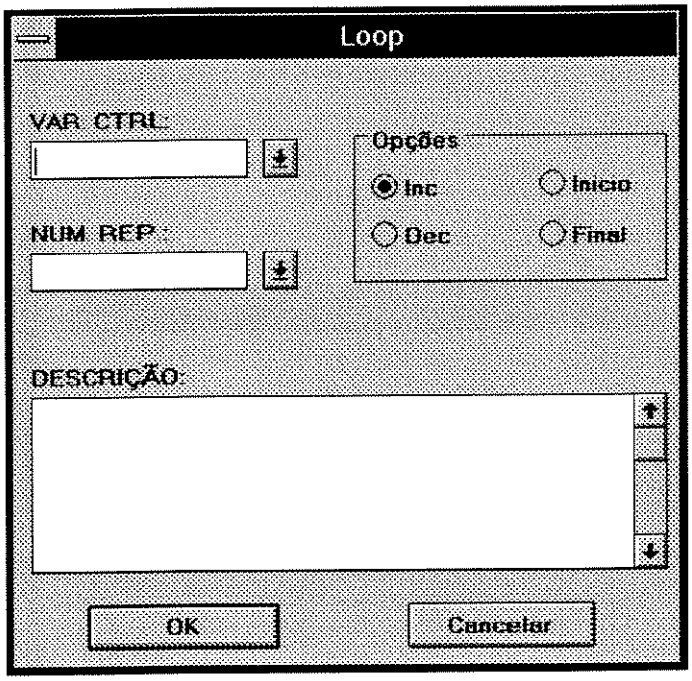


Figura 3.27: Caixa de diálogo da operação de iteração

Operação de geração de atraso

Freqüentemente, em aplicações dedicadas, ocorre a necessidade de se promover um atraso entre duas ações. Isto é comum quando a aplicação controla um sistema cujo tempo de resposta é bem maior que o do microcontrolador. Caso não seja preciso realizar nenhuma outra ação durante o tempo de espera, podemos usar um ícone de operação específico para a geração de atraso de tempos derivados do *clock* da CPU.

Temos na figura 3.28 o ícone que simboliza esta operação e o diálogo usado para especificá-la. Neste diálogo devemos indicar o tempo desejado (em segundos) para o

atraso, usando o campo **Tempo-Desejado**. Entretanto, o tempo efetivo de atraso nem sempre é igual ao tempo desejado, pois a implementação desse ícone utiliza laços aninhados para "gastar o tempo da CPU". Assim, o tempo efetivo está relacionado com a velocidade do microcontrolador e com o número de iterações dos laços envolvidos em sua implementação. Todavia, para que o usuário conheça este tempo o sistema reporta, através do campo **Tempo-Obtido** (em segundos), o seu valor.

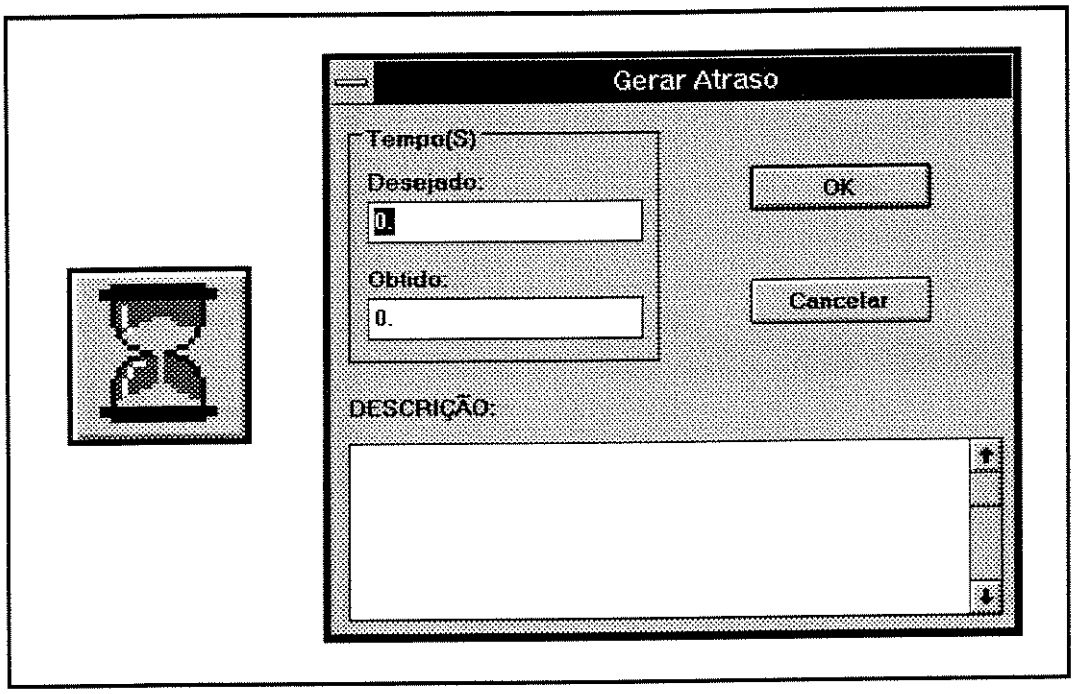


Figura 3.28: Ícone e caixa de diálogo da operação geração de atraso

Para se ter uma noção dos tempos possíveis para o ícone de geração de atraso, considere por exemplo, que o *clock* do microcontrolador seja de 6,144MHz, então o tempo máximo será de 65,66s com uma resolução de 33,20μs.

Operação de comunicação serial

Em virtude da maioria dos microcontroladores disporem, em sua arquitetura, de uma interface de comunicação serial, optou-se em implementar ícones de operação para manuseio desta interface. Assim, são disponíveis, na linguagem, dois ícones de operação para transferir dados do microcontrolador para dispositivos externos e destes para o primeiro. Estes ícones, mostrados na figura 3.29, são responsáveis pela transmissão de

qualquer identificador do programa para o meio exterior, bem como pela recepção dos dados de dispositivos externos, que são armazenados em identificadores do programa.

Para descrever a operação de comunicação serial, basta abrir o diálogo associado a esta operação (figura 3.30) e indicar o identificador que será usado, bem como o tipo de operação: *enviar* ou *receber*.

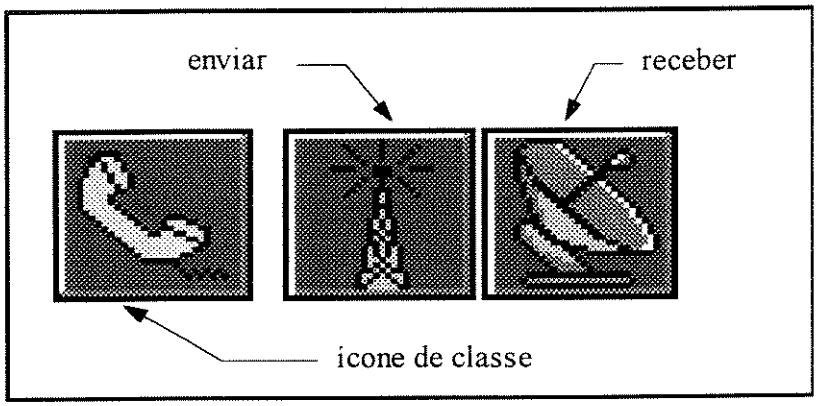


Figura 3.29: Ícones de operação de comunicação serial

Evidentemente, antes de usar qualquer um dos ícones de comunicação serial, deve-se programar os parâmetros da interface, como é indicado na seção 3.4.2.

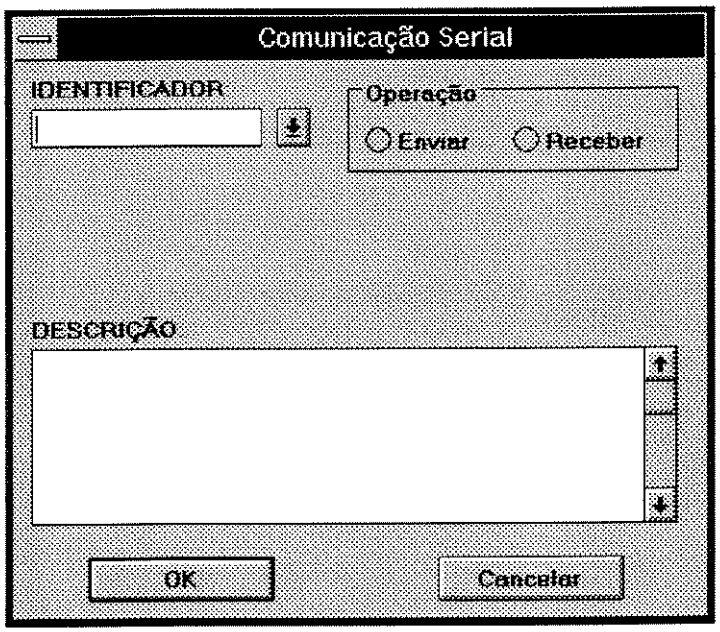


Figura 3.30: Caixa de diálogo da operação comunicação serial

Operação de pesquisa em vetor

Esta operação, cujo ícone e a caixa de diálogo são mostrados na figura 3.31, pode ser utilizada para verificar se um dado elemento pertence a um vetor. Tal operação é condicional, pois o elemento pode ou não pertencer ao vetor pesquisado. Portanto, existem dois fluxos de execução possíveis para o ícone dessa operação. Um associado à condição verdadeira, indicando que o elemento pertence ao vetor (saída V do ícone) e outro à condição falsa, indicando que o mesmo não foi encontrado (saída F do ícone).

Na caixa de diálogo o usuário deve indicar o nome do vetor utilizado na pesquisa (campo VETOR) e este deve ser previamente declarado, pois é feita uma verificação sobre a consistência das informações introduzidas. Além do nome do vetor, deve ser indicado o valor a ser pesquisado (campo **Valor Pesquisado**) e, opcionalmente, o usuário pode requerer que o índice do elemento encontrado no vetor seja armazenado em um identificador. Este índice se constitui no retorno da operação de pesquisa. Entretanto, nos casos em que o elemento pesquisado não pertença ao vetor, o valor de retorno não será usado pelo compilador.

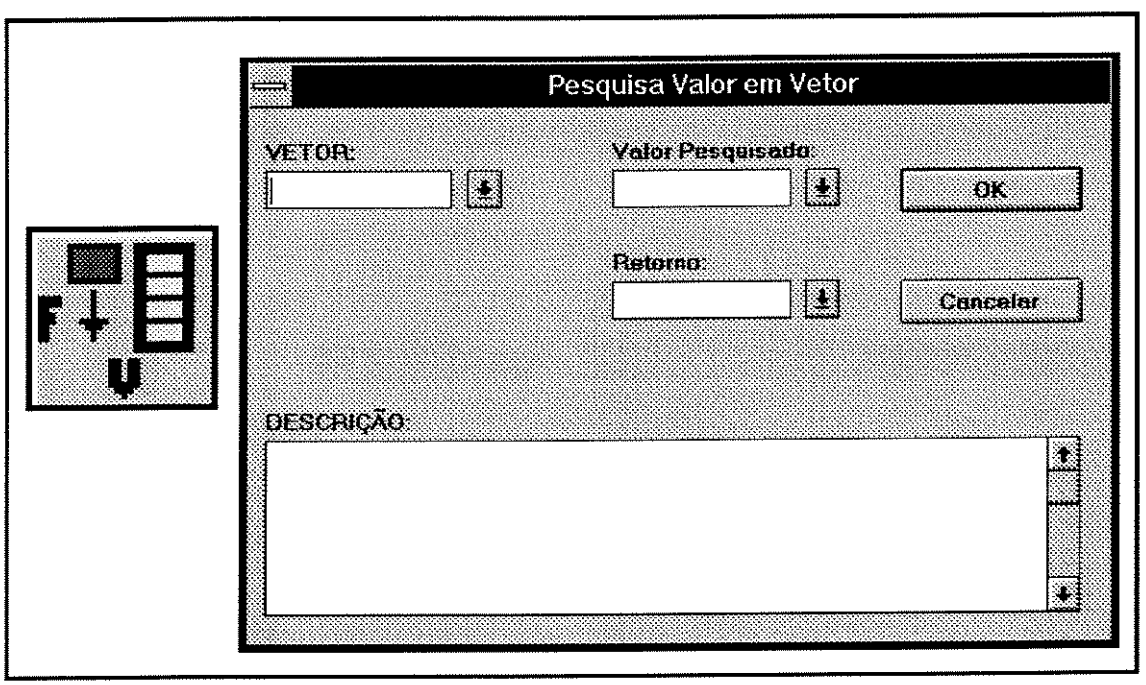


Figura 3.31: Ícone e caixa de diálogo da operação pesquisa em vetor

Operação de transformação vetorial

Este é outro ícone de operação destinado a manipulação de vetores. Sua função é relacionar elementos de dois vetores. Ele pesquisa um valor de entrada em um vetor e a partir do índice gerado, atribui a um identificador o elemento correspondente do outro vetor.

Um dos objetivos da implementação deste ícone de operação foi o de diminuir o esforço do programador em projetar rotinas de conversão de códigos, usuais em aplicações dedicadas.

Como pode ser visto na figura 3.32, o diálogo de especificação dessa operação solicita os nomes dos vetores utilizados (campos VETOR1 e VETOR2); o dado de entrada (campo FONTE); e o nome do identificador que receberá o dado de saída (campo DESTINO).

Da mesma forma que o ícone de operação de pesquisa em vetor, o ícone de operação de transformação vetorial (figura 3.32) possui duas saídas, pois o valor de entrada pode não ser encontrado no primeiro vetor. Neste caso, o identificador indicado pelo campo DESTINO não será modificado e o fluxo de execução assumirá a saída F. Caso contrário, o fluxo assumirá a saída complementar V.

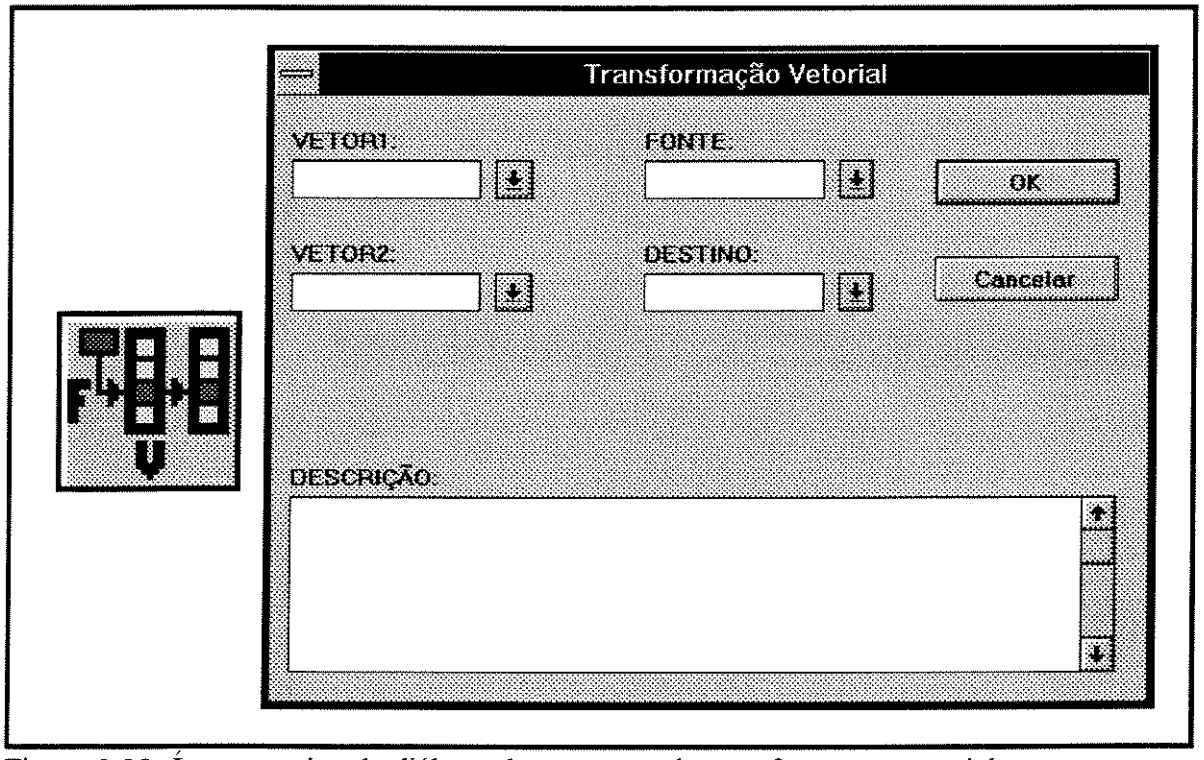


Figura 3.32: Ícone e caixa de diálogo da operação de transformação vetorial

Operação de geração de bordas

Freqüentemente em aplicações com microcontrolador é comum aplicar a *bits* externos, bordas de subida ou descida, a fim de disparar um determinado evento (gerar um sinal de START em um conversor analógico-digital, por exemplo). Este tipo de operação pode ser realizado com o uso dos ícones de geração de bordas, expostos na figura 3.33.

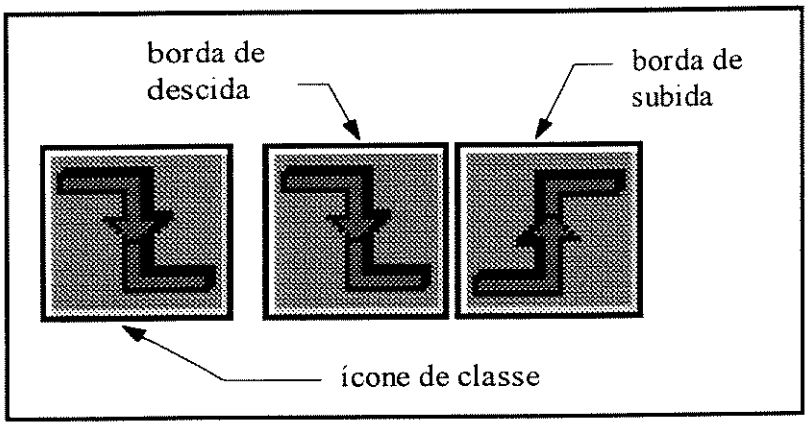


Figura 3.33: Ícones de geração de bordas

Deve-se tomar um certo cuidado com essa operação, pois tanto na borda de subida como na de descida, o tempo de transição depende da velocidade do microcontrolador. O código gerado simplesmente liga e desliga o *bit* do identificador (borda de descida) ou faz a operação inversa (borda de subida).

É importante observar que esta operação só faz sentido se o identificador (campo LABEL do diálogo mostrado na figura 3.34) for uma porta de e/s. Entretanto, o compilador permite que a operação seja executada também com variáveis, para fins de simulação.

Operação juntar bits

Em virtude de a linguagem dispor de mecanismos para criação de portas de e/s, com número de *bits* programáveis pelo usuário, e da necessidade de agrupar os *bits* de diferentes portas de e/s em um único identificador, foi implementado o ícone de operação juntar *bits*.

Na figura 3.35 temos o ícone e o diálogo dessa operação. No diálogo devem ser indicados os identificadores envolvidos na operação: o que vai receber os *bits* agrupados (campo RESULTADO), o que indica os *bits* mais significativos (campo HIGHBITS) e o que indica os *bits* menos significativos (campo LOWBITS). Nesta operação o número total de *bits* agrupados não pode exceder a oito.

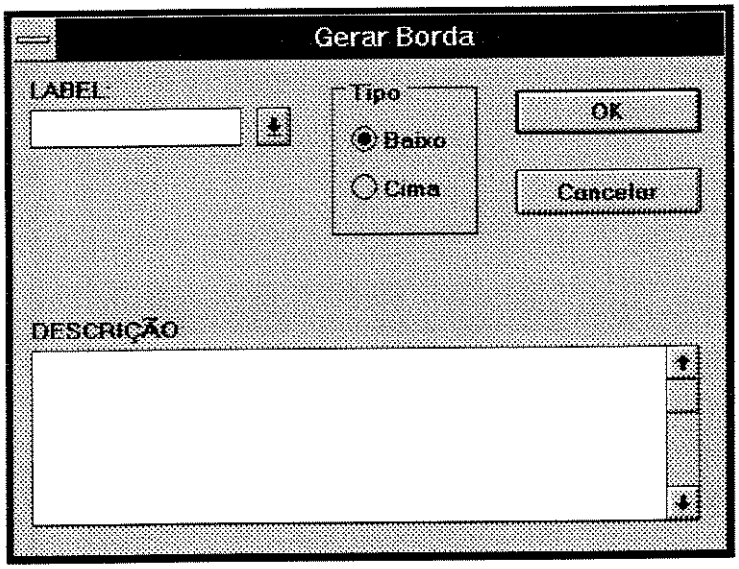


Figura 3.34: Caixa de diálogo da operação geração de bordas

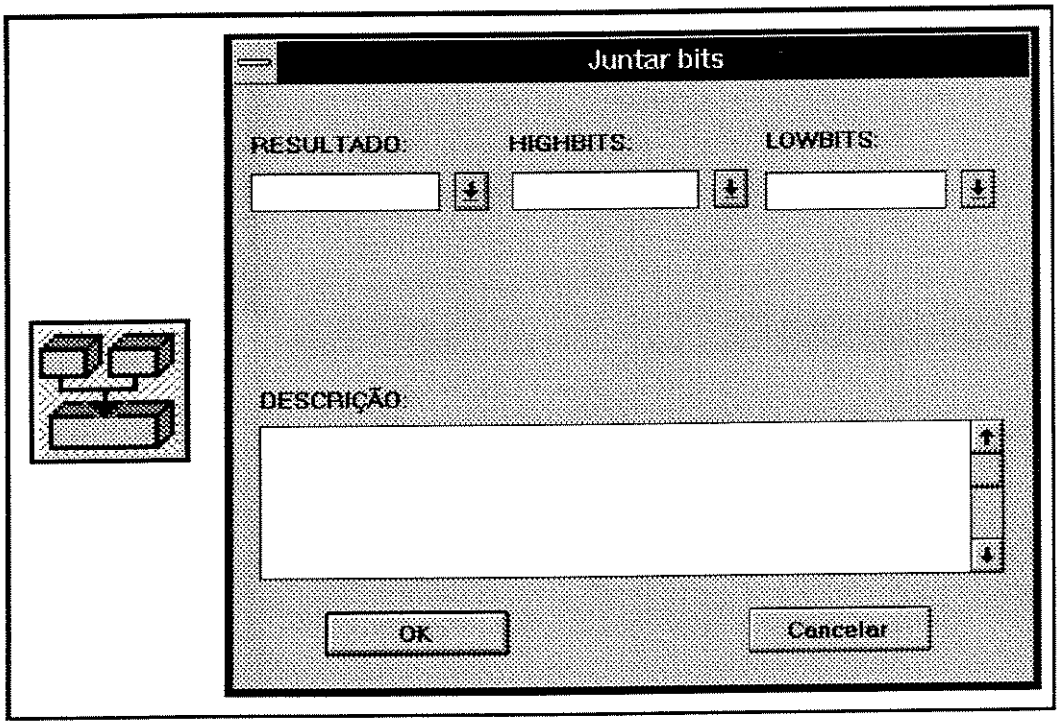


Figura 3.35: Ícone e caixa de diálogo da operação juntar *bits*

Operação com subrotinas

Os ícones utilizados para indicar o início de uma subrotina, ativá-la e promover o retorno dela para o módulo que a ativou, são chamados *ícones de operação com subrotina*, mostrados na figura 3.36. Estes ícones interagem com os identificadores de subrotinas permitindo a estruturação do programa em blocos funcionais, a fim de melhorar sua organização e também, promover um maior reaproveitamento das rotinas já implementas.

Conforme visto anteriormente, podemos entrar no modo de edição de uma subrotina, selecionando a opção Identificadores-Subrotinas do menu de linha e acionando o botão **Editar** do diálogo de subrotina. Podemos também realizar esta mesma operação por meio da caixa de listagem localizada na barra de ferramentas. Quando entramos pela primeira vez no modo de edição de subrotina, o sistema automaticamente insere dois ícones de operação. Um para indicar o início da subrotina e outro para indicar o final da mesma. Este último ícone de operação indica também o ponto de retorno ao módulo que a ativou. Portanto, estes dois ícones só podem aparecer dentro de uma subrotina. O editor de fluxogramas não permite que eles sejam parte integrante do programa principal.

Os ícones de subrotina geram o código de máquina necessário para criar a interface da subrotina com o bloco que a ativou. Eles trabalham com a pilha do microcontrolador para garantir a integridade dos dados dos módulos, salvando e recuperando o contexto destes sempre que houver chamada ou retorno de subrotina. Eles ainda passam, através da pilha, valores de argumentos e de retorno, permitindo a comunicação entre os módulos envolvidos.

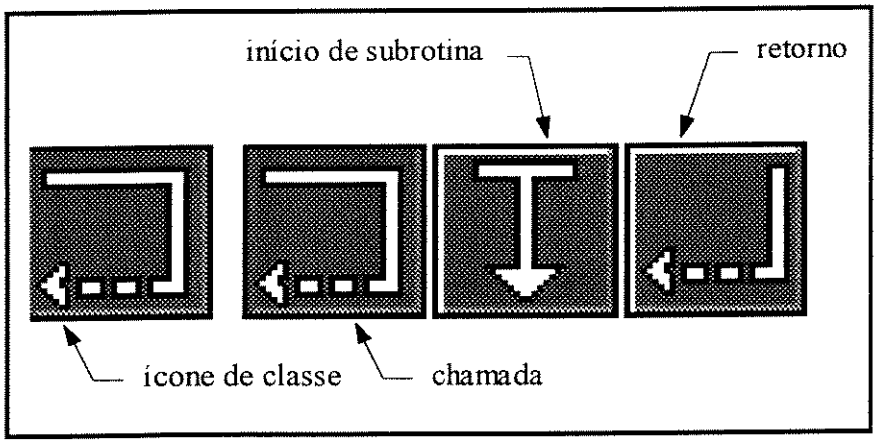


Figura 3.36: Ícones de subrotinas

Para utilizar um dos ícones associados às subrotinas, deve-se preencher o diálogo mostrado na figura 3.37. Para os ícones de início e retorno será preciso apenas indicar a operação (campo **Operação**) e o identificador da subrotina (campo LABEL). Entretanto, para o ícone de chamada, além destas informações, é necessário que o usuário indique a lista de argumentos (campo ARGUMENTOS) que serão passados e qual identificador receberá o valor de retorno (campo RETORNO), desde que a subrotina em questão tenha sido definida com estas especificações. A caixa de verificação **Interrupção** é usada pelo sistema para indicar ao usuário a natureza da subrotina, ou seja, se ela é de interrupção ou não.

Nos casos em que se seleciona o ícone de chamada de uma subrotina que tenha sido descrita com ícone próprio, o editor de fluxogramas troca o ícone padrão do sistema pelo ícone indicado na definição da subrotina (visto na seção 3.2). Isto é feito logo após o fechamento do diálogo, permitindo expressar numa única imagem a idéia fundamental do processamento efetuado por aquela subrotina, o que facilita o entendimento do programa.

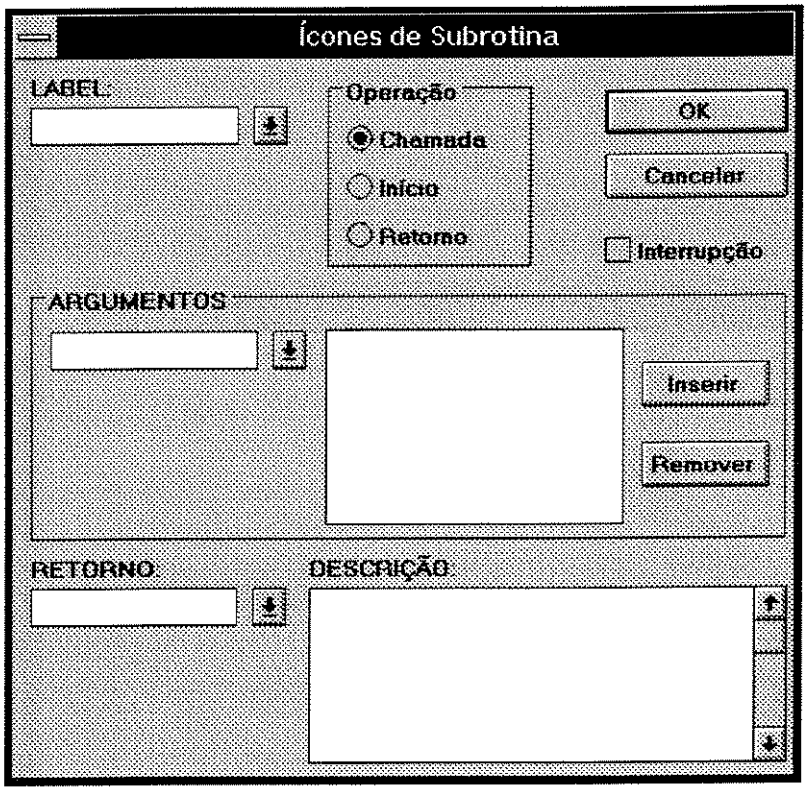


Figura 3.37: Caixa de diálogo dos ícones de subrotinas

Ícones complementares

A linguagem dispõe de quatro ícones que não são propriamente ditos ícones de operação. Eles são ícones auxiliares usados para facilitar a introdução do programa-fonte através do editor gráfico. Estes ícones, mostrados na figura 3.38, são opcionais porém, eles permitem, quando convenientemente utilizados, uma melhor organização do programa.

Dentre estes ícones, dois são usados para demarcar o início e o final do programa-fonte. Eles orientam o compilador sobre onde começa o fluxo de execução (ícone de início) e onde termina o programa (ícone de final). Caso não seja incluído o ícone de início de programa, o compilador assume que o primeiro ícone a ser executado é o que fica mais próximo do canto superior esquerdo da área de desenho.

O ícone de início de programa, que só deve aparecer uma vez dentro do programa-fonte, é responsável também, pela geração de parte do chamado "start-up code" ou código de partida, onde se inicia alguns registradores e posições de memória. Já o ícone de final de programa, pode aparecer em mais de uma posição do programa entretanto, para uma boa estruturação da aplicação, recomenda-se que ele seja usado em um único ponto do programa.

Os outros dois ícones são utilizados para criar junções e conexões entre ícones. O ícone de junção deve ser utilizado nos casos onde o número de pontos de entrada de um determinado ícone for menor que o número de ícones cujas saídas se direcionam a ele, pois como já mencionado, cada ícone possui no máximo três pontos de entrada.

O ícone de conexão tem a função de conectar um ou mais ícones a um outro que não esteja geograficamente próximo, onde a ligação usual entre eles se mostraria esteticamente inconveniente.

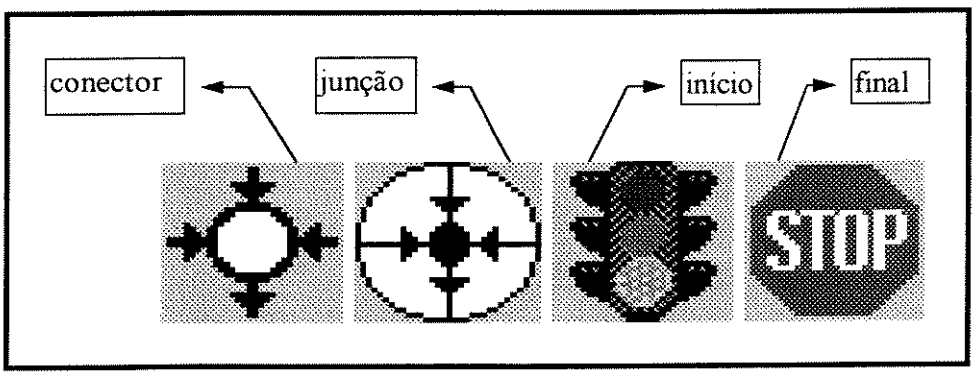


Figura 3.38.: Ícones complementares

3.4. CONFIGURAÇÃO DOS RECURSOS DO MICROCONTROLADOR

Como se sabe, a maioria dos microcontroladores possuem além da CPU, uma série de componentes integrados num mesmo "chip". Dentre estes componentes, podemos citar: o controlador de interrupção, o canal de comunicação serial e os temporizadores/contadores; que são os mais usuais. Estes componentes, chamados aqui de *recursos do microcontrolador*, possuem características peculiares, podendo mudar de uma família para outra de microcontrolador. Assim, torna-se difícil realizar a programação ou configuração de tais elementos de forma padronizada.

Para configurar estes recursos, o usuário da linguagem ONAGRO deve utilizar ícones personalizados, introduzindo o código necessário para utilizar adequadamente os recursos desejados.

Entretanto, a linguagem oferece a possibilidade de configuração destes recursos, através de caixas de diálogos, para uma única família de microcontroladores. No caso, a família escolhida foi a MCS-51 da INTEL, que é atualmente uma das mais difundidas no mercado.

O menu de linha do programa ONAGRO oferece a opção **Recursos**, onde o usuário dispõe de três caixas de diálogo para configurar os recursos do microcontrolador. O objetivo destes diálogos é facilitar a configuração dos referidos recursos, permitindo ao usuário uma programação em alto-nível dos recursos escolhidos. As informações solicitadas pelos diálogos geram uma maior abstração de detalhes de configuração, como os *bits* que indicam o modo de trabalho de um certo recurso; nomes de registros ou *bits* de registro usados na configuração; endereços de memória associados; "flags" e máscaras; dentre outros.

A partir dos dados introduzidos nesses diálogos, o compilador gera o código necessário para programar os recursos selecionados. Todavia, pode-se notar que esta programação é feita de forma estática, ou seja, ela é feita em tempo de compilação. Caso se deseje alterar estes parâmetros em tempo de execução é preciso que o programador utilize ícones personalizados.

3.4.1. Configuração do sistema de interrupção

No caso específico da família MCS-51, os serviços de interrupção consistem de cinco fontes de requisição de interrupção mascarada, com dois níveis de prioridade. Das cinco fontes de requisição, duas são externas, uma é para a interface serial e as últimas duas são para os Temporizadores/Contadores. Cada interrupção é atendida por uma rotina específica de tratamento de interrupção, cujo endereço inicial para cada fonte é mostrado na tabela 3.1. A configuração das interrupções é feita através de quatro registros da CPU: TCON, TMOD, IE e IP. [14]

As interrupções externas podem ser ativadas por nível ou por transição, dependendo da programação feita no registro TCON.

INTERRUPÇÃO FONTE	ENDEREÇO INICIAL
Requisição externa 0	0003H
Temporizador/Contador 0	000BH
Requisição externa 1	0013H
Temporizador/Contador 1	001BH
Interface serial	0023H

Tabela 3.1: Endereço inicial das rotinas de interrupção do MCS-51

Cada pedido de interrupção ativa seu correspondente "flag" nos registros TCON e SCON. O pedido só será reconhecido se sua máscara, presente no registro IE, estiver habilitada.

Além do registro IE, o MCS-51 dispõe de outro registro para controle das interrupções. É o registro IP, no qual se estabelece os níveis de prioridade das interrupções. Este registro é composto por 5 *bits*, cada um associado a uma fonte de interrupção diferente. Quando um destes *bits* vai a zero, indica que a interrupção correspondente tem prioridade baixa, caso contrário, a interrupção tem prioridade alta. É importante salientar que uma solicitação de interrupção não pode ser aceita durante o tratamento de uma interrupção de mesma ou maior prioridade. [15]

O diálogo de programação das interrupções, mostrado na figura 3.39, solicita para cada fonte de interrupção os parâmetros necessários à configuração: tipo de ativação (só para as externas), a prioridade e se é para habilitar ou não àquela interrupção. Além disto, o

diálogo permite a edição da subrotina de tratamento da interrupção, através dos botões **Editar** e **Fechar**. As subrotinas são "vetorizadas" obedecendo as especificações contidas na tabela 3.1, vista anteriormente.

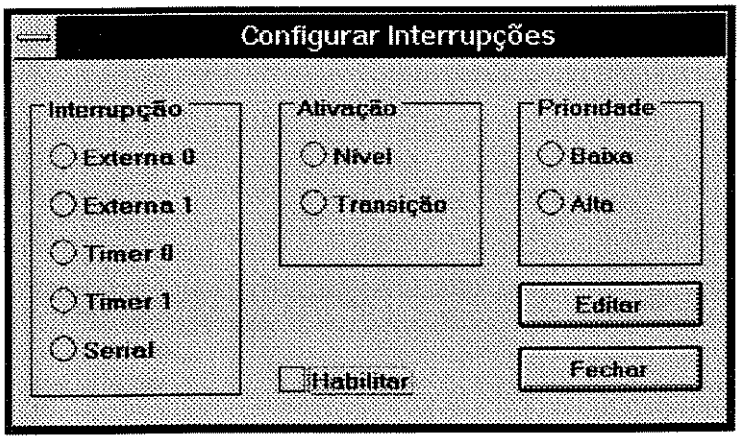


Figura 3.39: Caixa de diálogo de configuração das interrupções

3.4.2. Configuração da interface de comunicação serial

O MCS-51 dispõe de uma porta de comunicação serial "full-duplex" que pode trabalhar tanto no modo síncrono como no modo assíncrono. Esta porta serial pode ser usada para interligar o microcontrolador a diversos dispositivos, como por exemplo: impressoras e "plotters"; terminais de comunicação; e computadores de maior porte. Também é possível a conexão de vários microcontroladores, através da porta serial, compondo assim um sistema distribuído.

Na interface serial existem dois registros mapeados num mesmo endereço de memória (SBUF) usados para armazenar o dado recebido e o que será transmitido. Um de leitura, que guarda o dado recebido, e outro de escrita, que guarda o dado a ser transmitido. A recepção ainda dispõe de duplo "buffer", evitando assim, o problema de sobreposição dos dados recebidos. [14]

Existe outro registro associado à interface serial, é o SCON. Ele é usado para selecionar o modo de trabalho e os parâmetros de configuração da interface e monitorar o estado de operação da mesma. [14]

Como mencionado na descrição dos ícones de comunicação serial, antes de usar qualquer um deles, o usuário deve programar os parâmetros da interface. Isto é feito por

meio da opção **Recursos-Serial** do menu de linha. Esta opção abre um diálogo (figura 3.40) onde são especificados: o tamanho da palavra de dados (campo **Palavra**); a velocidade de comunicação (campo **Taxa de Transferência**); o tipo de paridade (campo **Paridade**); e por último, se a interface pode ser usada pelo programa (caixa de verificação **Habilitar**).

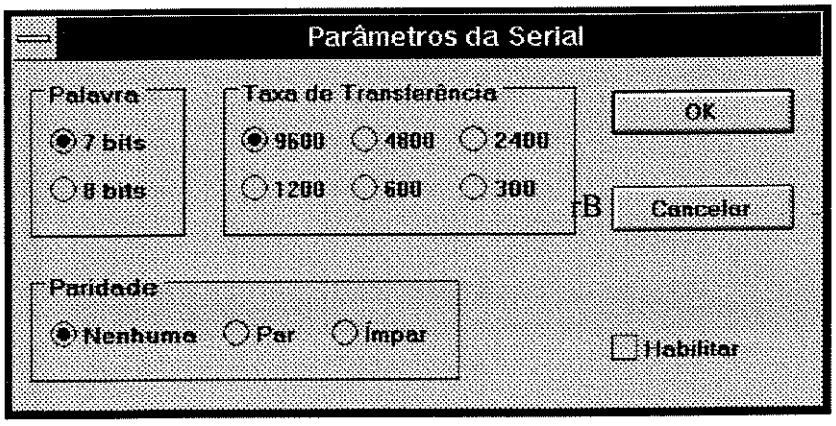


Figura 3.40: Caixa de diálogo de configuração da interface serial

3.4.3. Configuração dos temporizadores/contadores

A arquitetura do MCS-51 possui dois Temporizadores/Contador (T/C) de 16 *bits*, usados para medir intervalos de tempo, medir largura de pulsos, contar eventos e gerar base precisa de tempo. Os T/C são independentes e programados por *software*, através de alguns *bits* dos registros TCON e TMOD. [15]

Existem quatro modos de trabalho para o T/C 0 e três modos para o T/C 1, os quais podem ser, sucintamente, descritos como mostra a tabela 3.2.

MODO DE TRABALHO	DESCRIÇÃO RESUMIDA
MODO 0	Temporizador ou contador de 8 bits com fator de divisão de freqüência de 5 bits (total de 13 bits de contagem)
MODO 1	Temporizador ou contador de 16 bits
MODO 2	Temporizador ou contador de 8 bits com recarga automática
MODO 3:	Um temporizador de 8 bits e um contador de 8 bits. Somente para o T/C 0.

Tabela 3.2: Modos de trabalho dos T/C do MCS-51

Na configuração dos T/C (figura 3.41) são solicitados para cada um dos dois T/C os seguintes dados: o tipo de operação (temporização ou contagem), o modo de trabalho, a

Na configuração dos T/C (figura 3.41) são solicitados para cada um dos dois T/C os seguintes dados: o tipo de operação (temporização ou contagem), o modo de trabalho, a forma de ativação do T/C (campo GATE), o valor inicial de contagem, o valor de recarga (quando for o caso) e, finalmente, indicar se o T/C deve ser ligado ou não (caixa de verificação **Ligar**).

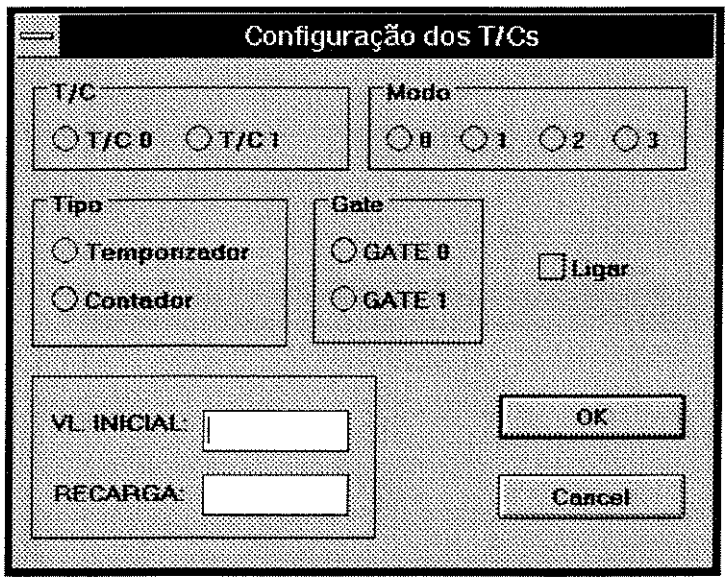


Figura 3.41: Caixa de diálogo de configuração dos temporizadores/contadores

3.5. CONFIGURAÇÃO DE PARÂMETROS DO COMPILADOR

Através do diálogo visto na figura 3.42, ativado pelo menu opção **Compilar-Opções**, é possível alterar parâmetros do compilador, permitindo uma maior flexibilidade do código gerado. Pode-se editar os seguintes dados do compilador: tipo de microcontrolador usado; o modelo de memória; os endereços iniciais de alocação das instruções, dos dados e da pilha; e o valor do *clock* da CPU. Este último dado é importante para o compilador poder gerar corretamente o código referente ao ícone de operação geração de atrasos.

O compilador ONAGRO pode gerar código seguindo dois modelos de memória, um denominado **Grande** e o outro **Pequeno**. Estes modelos diferem na forma como os dados são alocados. No primeiro, todos os dados são armazenados em memória externa ao microcontrolador. Ele deve ser utilizado quando se dispõe de uma quantidade de dados maior que a capacidade de armazenamento da memória interna (normalmente muito

pequena). O segundo modelo, por sua vez, aloca os dados na memória interna do microcontrolador, sendo usualmente restrito às aplicações com pequena quantidade de dados. Nos dois casos, o código das instruções são alocados em memória de programa, que tanto pode ser interna como externa.

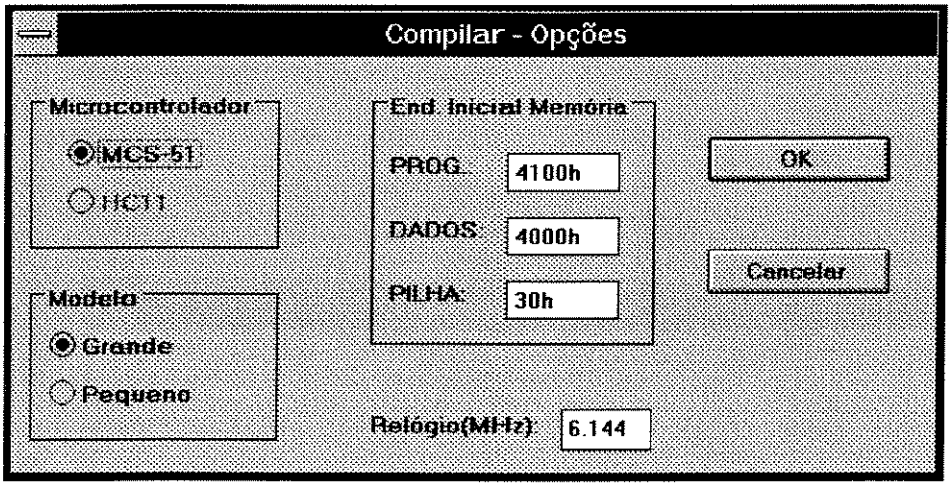


Figura 3.42: Caixa de diálogo das opções de compilação

CAPÍTULO 4

TESTES E RESULTADOS

INTRODUÇÃO

Neste capítulo é feito um relato dos testes realizados do ambiente ONAGRO e da análise dos resultados obtidos nestes testes. O objetivo foi coletar dados para uma avaliação do desempenho do compilador, no tocante ao tamanho do código gerado. Desta forma, nos concentramos numa análise mais quantitativa. É claro que este tipo exclusivo de análise não é suficiente para validar o trabalho proposto. Entretanto, partimos do pressuposto que as pesquisas comentadas no capítulo 1 sirvam de referendo, pois o desenvolvimento do ONAGRO se baseou nas propostas apresentadas por aqueles trabalhos.

Nos testes realizados, procurou-se implementar aplicações que utilizassem a maior parte dos ícones de operação disponíveis na linguagem. Não houve grande preocupação com a quantidade de ícones envolvidos, mas sim no repertório usado. As aplicações propostas foram idealizadas com base nas principais rotinas utilizadas em sistemas dedicados. Em todas elas, o *hardware* da aplicação foi hipoteticamente definido, haja visto que o nosso interesse era avaliar o código gerado para o *software* das aplicações.

No total foram realizados 18 testes com cinco aplicações compiladas em dois modelos diferentes de memória, utilizando-se portas de e/s pré-definidas pela arquitetura do microcontrolador como também portas de e/s mapeadas em memória externa.

Nossa intenção era conhecer o tamanho do código gerado pelo ONAGRO comparativamente a outros compiladores comercialmente disponíveis. É importante ressaltar que os objetivos iniciais do trabalho não propunham a geração de código mais otimizado que outros compiladores. Portanto, a proposta de comparar o desempenho do ONAGRO com outro compilador tem o objetivo de apenas criar um parâmetro de referência que qualifique o trabalho desenvolvido.

Optamos pela utilização da família de microcontroladores MCS-51 da INTEL, pelas razões anteriormente expostas. Todos os testes submetidos ao ONAGRO foram também submetidos ao compilador AVOCET C V1.217, que é um dos compiladores mais conhecidos para essa família.

4.1. TESTES REALIZADOS

Foram implementadas cinco aplicações, que serão referenciadas pelas letras A, B, C, D e E. A primeira delas consiste na linearização do sinal de um transdutor hipotético, cuja curva característica apresenta-se não-linear. Nesta aplicação o microcontrolador é conectado a um conversor analógico-digital de resolução de 8 *bits*, que serve de interface entre o transdutor e o microcontrolador. O programa aciona o conversor através do pino de START; espera o sinal de final da conversão EOC (End Of Coversion); ler o dado convertido; faz sua linearização através de uma tabela de pontos; e gera uma saída deste dado linearizado.

Temos no anexo 1 as implementações da aplicação A em linguagem C e em ONAGRO, respectivamente. No caso da versão para o ONAGRO, os identificadores não aparecem no fluxograma, pois os mesmos são introduzidos por meio de diálogos de descrição de identificadores, vistos no capítulo 3. As informações demarcadas pelos retângulos pontilhados são pequenos comentários introduzidos no campo DESCRIÇÃO dos ícones de operação e são facilmente acessíveis pelo usuário através do modo de inspeção, também relatado no capítulo 3. Na versão em linguagem C utilizou-se as estruturas mais conhecidas e padronizadas de programação para implementar a aplicação. Por uma questão de simplificação o vetor ***curva*** não foi iniciado com os 256 valores que caracterizam o transdutor.

A segunda aplicação (anexo 2) é uma rotina de leitura de teclado. Ela localiza uma tecla pressionada, através de varredura em cada uma das teclas, e realiza a transformação do código de varredura em seu correspondente código ASCII, enviando-o para um conjunto de 8 LEDs de saída. Esta transformação é conseguida por meio de dois vetores. Um que armazena os possíveis códigos de varredura das teclas e o outro que armazena os equivalentes códigos ASCII. No caso da versão em ONAGRO, foi utilizado um ícone apropriado, o ícone de transformação vetorial. Já na versão em linguagem C foi preciso codificar uma bloco de instruções para realizar esta operação.

A aplicação C, mostrada no anexo 3, controla um motor de passos de quatro fases, enviando pulsos para que o mesmo se movimente de meio em meio passo. A velocidade é constante e o sentido (horário ou anti-horário) é indicado por dois *bits* de entrada. Estes

bits também são utilizados solicitar a parada do motor. Nesta aplicação, a rotina principal apenas verifica o estado de parada do motor e ativa uma subrotina de envio dos passos, que é responsável pela geração dos pulsos necessários para movimentar o motor, conforme o estado dos *bits* de entrada. Como mostra o anexo 3, esta subrotina teve, na versão em ONAGRO, seu ícone personalizado.

No anexo 4 temos as implementações em linguagem C e em ONAGRO da aplicação D, que consiste em um controlador de temperatura do tipo ON-OFF. O programa que implementa esta aplicação lê o sinal, já digitalizado, de um transdutor de temperatura, através de uma porta de entrada, e compara este valor com um "set-point". Caso a temperatura lida esteja abaixo de um valor mínimo estipulado, é ativada uma subrotina para compensação da temperatura. Ela espera o sinal de zero da rede e envia um trem de pulsos a um TIRISTOR, fazendo a temperatura elevar-se. Esta subrotina também teve seu ícone personalizado para facilitar o entendimento do programa principal.

Por último, foi implementado a aplicação E, vista no anexo 5. Esta aplicação é responsável pela gravação de dados recebidos pelo microcontrolador, através da porta serial, numa memória EPROM. Inicialmente o microcontrolador recebe o número de páginas que serão gravadas e logo em seguida recebe, para cada página, 256 dados (*bytes*) para gravação. Para cada dado a ser gravado, o microcontrolador gera o endereço correspondente e ativa o pino de gravação da EPROM. Na implementação em linguagem C foi preciso criar rotinas específicas para programação e leitura da porta serial. Entretanto, na versão em ONAGRO isto foi conseguido através do diálogo de configuração da serial e da utilização de ícone apropriado à recepção serial.

4.2. RESULTADOS OBTIDOS

Após a geração do código para as cinco aplicações, tanto na linguagem C como no ONAGRO, montou-se as tabelas 4.1 e 4.2 que mostram, respectivamente, os resultados obtidos por estas linguagens. No cálculo do código gerado não foram incluídas as áreas de alocação dos dados das aplicações, cujos tamanhos se mostraram semelhantes nas duas linguagens, considerando-se a mesma aplicação. Isto ocorreu porque para a mesma aplicação, procurou-se compatibilizar os tipos de variáveis definidas nas duas linguagens. Os valores presentes nas tabelas representam apenas os códigos de máquina gerados a partir

das instruções dos programas-fontes. Em cada linha da tabela temos o tamanho (em *bytes*) do código gerado nas aplicações para uma determinada configuração. Tal configuração se refere ao modelo de memória utilizado (Grande ou Pequeno) e ao tipo de mapeamento escolhido para as portas de e/s (mapeadas em Memória ou Internas ao microcontrolador).

Configuração	A	B	C	D	E
Grande Memória	129	179	206	208	283
Grande Interna	114	173	199	191	271
Pequeno Memória	------	176	163	140	184
Pequeno Interno	------	158	156	122	172

Tabela 4.1: Tamanho do código gerado pelo Avocet C

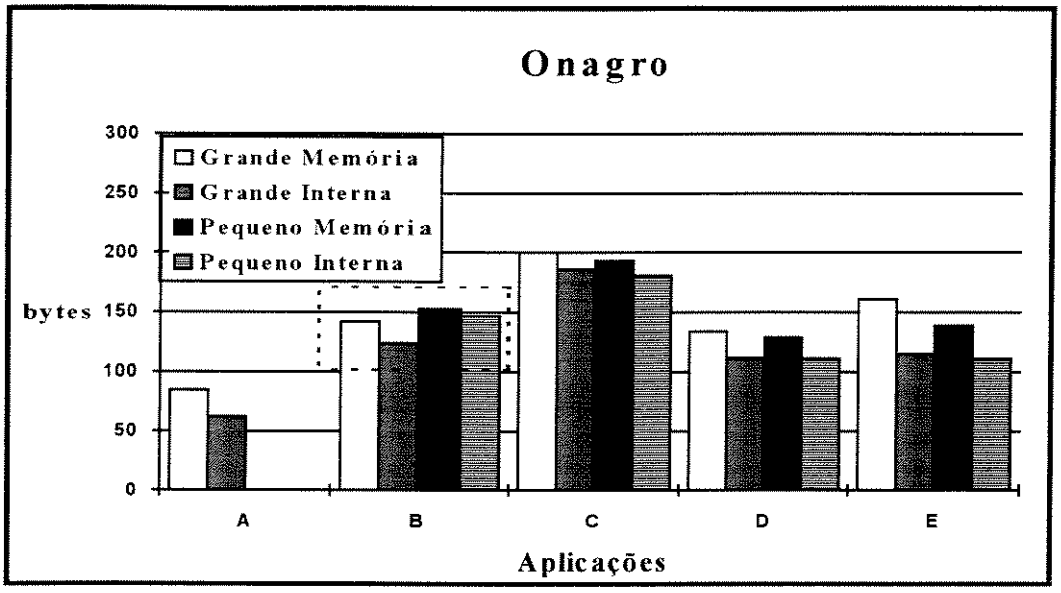
Configuração	A	B	C	D	E
Grande Memória	85	142	200	134	161
Grande Interna	62	124	186	112	115
Pequeno Memória	------	153	193	129	139
Pequeno Interno	------	147	181	111	111

Tabela 4.2: Tamanho do código gerado pelo Onagro

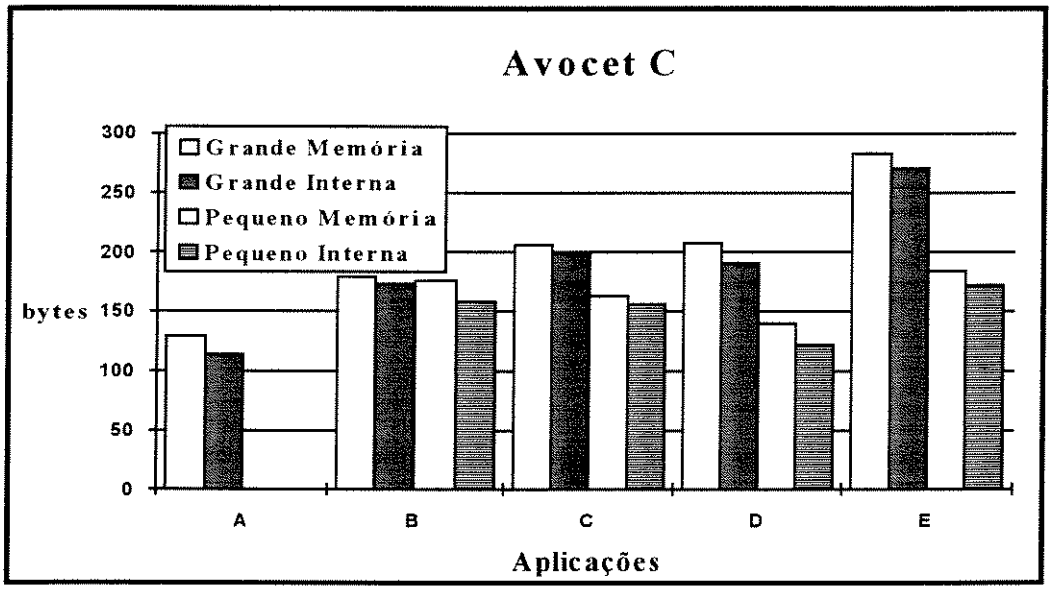
Para facilitar a comparação destes valores, foram montados gráficos de barras a partir dessas tabelas, como mostra a figura 4.1. Podemos observar nos gráficos que para o mesmo modelo de memória, a utilização de portas mapeadas em memória apresenta código maior em comparação ao gerado quando se utilizou portas internas definidas na própria arquitetura do microcontrolador. Tal constatação foi verificada em todas as aplicações feitas nas duas linguagens. Obviamente isso pode ser explicado pelo fato de que o acesso à uma porta mapeada em memória requer dois *bytes* de endereçamento, um a mais que as portas internas ao microcontrolador; e pela usual necessidade de se utilizar um registrador especial para endereçamento de memória externa.

Também foi verificado que, para o mesmo tipo de mapeamento de portas de e/s, a maioria das aplicações compiladas no modelo Grande teve geração de código maior que no modelo Pequeno, devido a alocação dos dados ser feita em memória externa. Neste tipo de memória os endereços são de 16 *bits* ao invés de apenas 8 quando se usa memória interna.

No caso específico do ONAGRO, esta situação não se registrou em todas as aplicações. Como podemos ver na figura 4.1a, a aplicação B no modelo Pequeno gerou código maior que no modelo Grande.



(a)



(b)

Figura 4.1: Desempenho dos compiladores Onagro e Avocet C

Tal aplicação possui um vetor iniciado em tempo de compilação, que é primeiro alocado em memória de programa e posteriormente, quando da iniciação do programa, transferido para a memória interna do microcontrolador. Isto ocasionou a geração de código adicional para realização desta transferência. Este adicional é mais perceptível quando o programa é pequeno. A medida que o número de instruções do programa cresce em proporção maior que os dados iniciado em tempo de compilação, o modelo Pequeno tende a gerar código menor que o modelo Grande. É o que ocorre na aplicação C, que também usa um vetor iniciado em tempo de compilação e utiliza, no entanto, um número de ícones maior que o da aplicação B. Neste último caso o código gerado no modelo Pequeno foi ligeiramente menor que no modelo Grande.

A exceção descrita acima não foi registrada pelos dados colhidos com a linguagem C, onde certamente o código de "start-up" é mais otimizado.

Faremos agora uma análise dos resultados coletados dando ênfase à comparação dos valores obtidos nas duas linguagens. Para isto foi montada a tabela 4.3, que nada mais é que uma reprodução, em conjunto, das duas tabelas anteriores. Nesta tabela temos os dezoito testes realizados com as cinco aplicações. Nesse mesmo sentido, dispomos os referidos dados em gráfico de barras como mostra a figura 4.2.

TESTE	APLICAÇÃO	MODELO	PORTAS	CÓDIGO Onagro	CÓDIGO Ling. C
01	A	Grande	Memória	85	129
02	A	Grande	Internas	62	114
03	B	Grande	Memória	142	179
04	B	Grande	Internas	124	173
05	B	Pequeno	Memória	153	176
06	B	Pequeno	Internas	147	158
07	C	Grande	Memória	200	206
08	C	Grande	Internas	186	199
09	C	Pequeno	Memória	193	163
10	C	Pequeno	Internas	181	156
11	D	Grande	Memória	134	208
12	D	Grande	Internas	112	191
13	D	Pequeno	Memória	129	140
14	D	Pequeno	Internas	111	122
15	E	Grande	Memória	161	283
16	E	Grande	Internas	115	271
17	E	Pequeno	Memória	139	184
18	E	Pequeno	Internas	111	172

Tabela 4.3: Resumo dos resultados obtidos

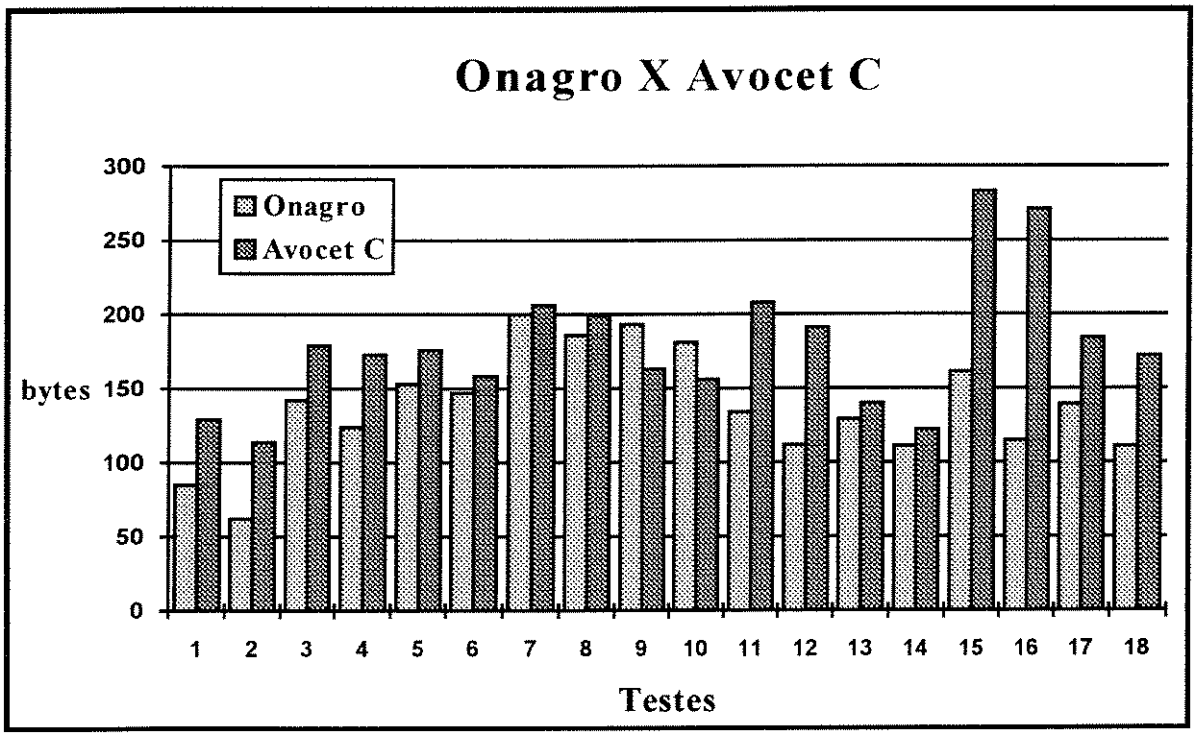


Figura 4.2: Comparação dos testes realizados com o Onagro e com o Avocet C

Analisando o gráfico e a tabela, nota-se que em todas os testes, com exceção do 9 e do 10, o código gerado pelo ONAGRO foi menor que o gerado pela linguagem C. Estes resultados foram obtidos, em parte, pelo fato do ONAGRO ter em relação à linguagem C, algumas restrições quanto aos dados manipulados. Não é disponível, por exemplo, variáveis tipo ponto-flutante ou vetores polidimensionais; só é permitida a criação de variáveis locais apenas para o intercâmbio entre as subrotinas; a passagem de parâmetros sempre é feita por troca de valor; e por último, o ONAGRO só comporta geração de código para dois modelos de memória e sempre só usa a pilha padrão do microcontrolador. No entanto, para as aplicações testadas, estas imposições não se constituíram em um grande empecilho para a implementação. A maior parte dos dados processados por elas estão ligados às portas de e/s, que têm no ONAGRO, mecanismos satisfatórios de manipulação.

Outro fator importante que explica os resultados obtidos é que no ONAGRO, existem ícones de operação específicos para aplicações dedicadas, projetados de forma otimizada e que numa outra linguagem (o C por exemplo), esta mesma operação tem que ser decomposta em instruções mais simples disponíveis na linguagem e que mesmo sendo utilizadas de forma adequada pode acarretar geração excessiva de código.

Também é importante observar que a maioria das linguagens usadas na programação de microcontroladores não foram inicialmente criadas levando-se em conta os parcos recursos do mesmo. Muito pelo contrário, por razões históricas elas foram projetadas para sistemas usualmente maiores e utilizadas em aplicações variadas. Essas linguagens garantem até mesmo uma certa portabilidade do programa-fonte para diferentes plataformas.

Já o compilador ONAGRO foi concebido no universo dos microcontroladores. Seus ícones de operação foram concebidos pensando-se nas aplicações dedicadas e implementados à luz dos recursos oferecidos pela arquitetura dos microcontroladores.

Por último, como comentado no final do capítulo 2, o algoritmo de tradução usado pelo ONAGRO baseou-se na troca de mensagens entre os objetos envolvidos nas operações. Isto além de facilitar a criação do algoritmo de tradução, ainda permitiu que, em grande parte, o código gerado fosse semanticamente correto, pois a maioria dos inconvenientes das generalizações de uma dada operação são eliminados pelo compilador, através da comunicação entre os elementos que compõem tal operação. Nos compiladores tradicionais, é comum encontrar situações em que o código que implementa uma operação é criado de forma genérica para poder manipular qualquer tipo de dado. Isto faz com que o

programa-objeto cresça e haja necessidade de rotinas adicionais para "limpeza" das redundâncias.

Nos testes, também pôde-se observar que quando a aplicação utiliza subrotinas com passagem de parâmetros e retorno de algum valor, o código gerado pelo ONAGRO se mostrou pior que o gerado pela linguagem C, ficando evidente nos resultados obtidos pelos testes 9 e 10, onde houve uma supremacia da linguagem C.

Através da análise dos códigos gerados pelos dois compiladores, observou-se que a manipulação dos parâmetros e do valor de retorno, feitos pela linguagem C, se mostrou mais otimizada que no ONAGRO. Esta característica da linguagem C fez com que o código gerado por ela fosse menor que o gerado pelo ONAGRO. Entretanto, também se observou que quando as subrotinas não usam parâmetros ou valor de retorno, o código gerado pelo ONAGRO volta a ser melhor que o da linguagem C. Esta conclusão é feita com base no código da aplicação D (testes 11, 12, 13 e 14), que foi favorável ao ONAGRO. Esta aplicação também usa subrotina, sem usar, no entanto, parâmetros ou valor de retorno.

Finalizando, podemos dizer que esta avaliação inicial evidenciou as pontecialidades do ambiente de programação proposto. Ele oferece várias facilidades para elaboração de algoritmos, sem no entanto, comprometer o código gerado, que para aplicações com microcontrolador deve sempre ser o menor possível.

CONCLUSÃO

A escolha da metodologia e do ambiente de desenvolvimento se mostrou pertinente. Por um lado, a metodologia ajudou de forma bastante significativa o modelamento do sistema e a própria implementação. A linguagem proposta foi estruturada por elementos independentes (objetos) e por forte comunicação entre eles (mensagens). Por outro lado, os recursos oferecidos pelo ambiente de desenvolvimento possibilitaram a criação de uma interface homem-máquina, cuja simbologia gráfica constitui o alicerce da comunicação entre o computador e o programador.

Na proposta da linguagem procurou-se implementar as estruturas essenciais para a programação de microcontroladores usando ícones intuitivos e auto-explicativos. Assim, a maioria dos ícones confeccionados representa nada mais que os próprios símbolos usados em fluxogramas. É claro que para produzir uma melhor organização destes símbolos, foi feita uma estilização dos mesmos, aproveitando os recursos de cores do monitor de vídeo e introduzindo detalhes que evidenciam as operações representadas por eles.

O algoritmo de tradução difere dos usados nas linguagens tradicionais, que supervalorizam as análises sintática e semântica dos programas, já que os mesmos são expressos por textos. Tais análises são realizadas normalmente de forma centralizada. Entretanto, o algoritmo usado na linguagem proposta por este trabalho, procura ressaltar a comunicação entre os ícones e os identificadores, gerando o código de forma não centralizada. Isto ocorreu pela dificuldade de se criar uma única rotina tradutora que reconhecesse uma malha de conexão de ícones de operação e os elementos de dados envolvidos. A tradução descentralizada permitiu uma grande diminuição da complexidade do algoritmo e possibilitou uma geração de código relativamente compacto (conforme testes descritos no capítulo 4), pois o reconhecimento do contexto das operações é conseguido pela troca de informações entre os objetos envolvidos (ícones de operação e identificadores).

Apesar das simplificações feitas na linguagem, o ambiente se mostrou bastante satisfatório tanto no tocante à interface com o usuário como nas instruções disponíveis. Os testes preliminares foram positivos e ressaltaram os pontos para futuros melhoramentos. Um deles é a criação de novos ícones de operação: ícones de manipulação de interrupção e

dos temporizadores/contadores, ícones para geração de pulsos com relação cíclica programável, ícones adicionais de aritmética (incrementar e decrementar) e ícones de depuração.

A configuração dos recursos do microcontrolador pode ser estendida a outras famílias de microcontroladores, através da implementação de novos diálogos de configuração. Modelos de memória adicionais podem ser criados para facilitar o controle do usuário sobre as áreas de alocação de dados. E também pode ser incluído no ambiente um simulador e um emulador de memória, que facilitarão ainda mais o processo de desenvolvimento das aplicações.

O principal propósito do ONAGRO é facilitar o desenvolvimento de aplicações dedicadas entretanto, ele pode ser utilizado como ferramenta de apoio ao ensino de programação. Com algumas modificações nos ícones de operação e com uma simplificação dos identificadores, pode-se compor um ambiente bastante didático para o ensino de programação. Um trabalho semelhante, usando uma outra linguagem icônica, foi realizado por [4], que conseguiu bons resultados.

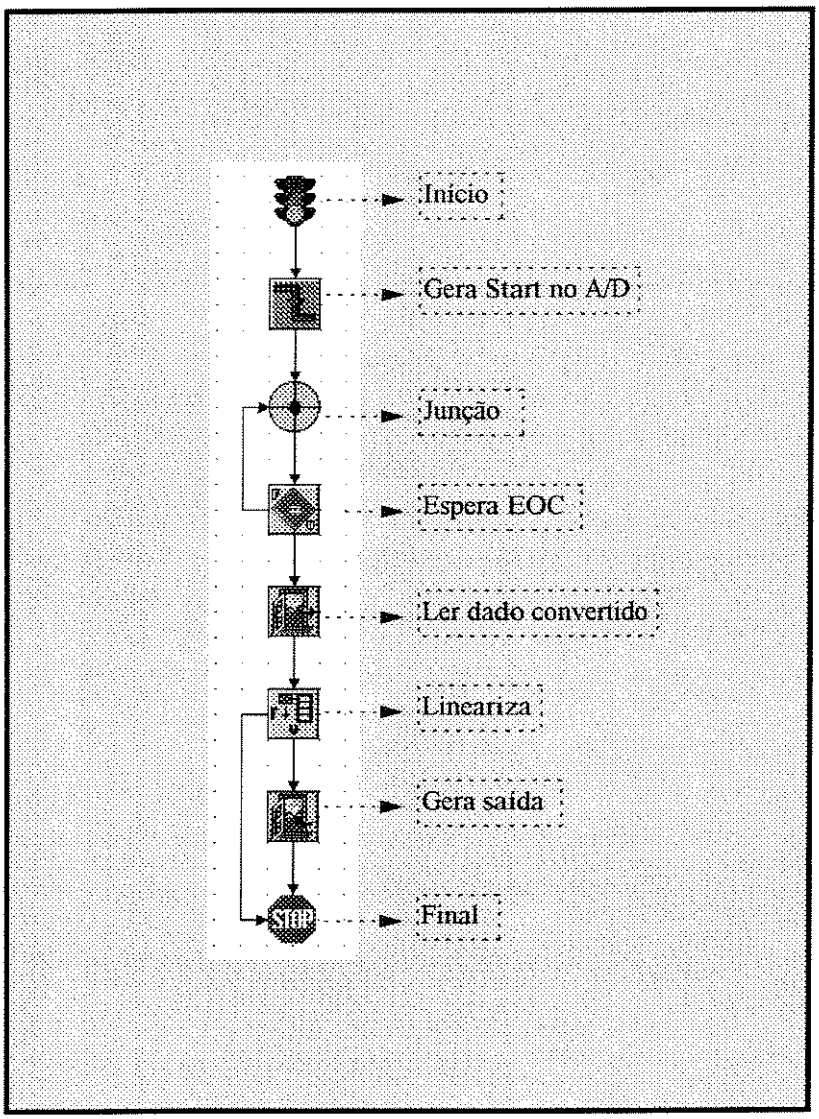
No momento, o ONAGRO está sendo usado como parte de um trabalho de doutorado, onde se pretende incluir ícones específicos para aquisição de dados e controle de processos. Além disso, o trabalho que está sendo desenvolvido vai incorporar ao ONAGRO ferramentas para simulação e depuração das aplicações, criando um ambiente gráfico completo para desenvolvimento de *softwares* para microcontroladores.

ANEXOS

ANEXO 1

```
/*  Aplicação A Usada nos testes 1 e 2  */

/* Define as portas usadas */
#define START_AD  (*(char*) 0x1000)
#define EOC_AD  (*(char*) 0x1001)
#define DADO_AD  (*(char*) 0x1002)
#define VALOR_SAIDA  (*(char*) 0x1003)

void main (void)
{
  /* Define as variaveis utilizadas */
  char curva[256];
  char dado_lido;
  int  ind;
  char achou;

  /* Gera start no A/D*/
  START_AD = 1;
  START_AD = 0;

  /* espera EOC */
  while (EOC_AD);

   /* ler o dado */
  dado_lido = DADO_AD;

  /* Faz a linearizacao */
  ind = 0;
  achou = 0;
  do
  {
    /* Verifica se encontrou o dado lido na tabela */
    if (curva[ind] == dado_lido)
    {
      achou = 1;
      VALOR_SAIDA = ind; /* Gera saida */
    }

    ind++; /* Atualiza indice */
  } while ((ind < 256) && (!achou));
}
```

Aplicação A (linearização de um transdutor) em linguagem C, usada nos teste 1 e 2.

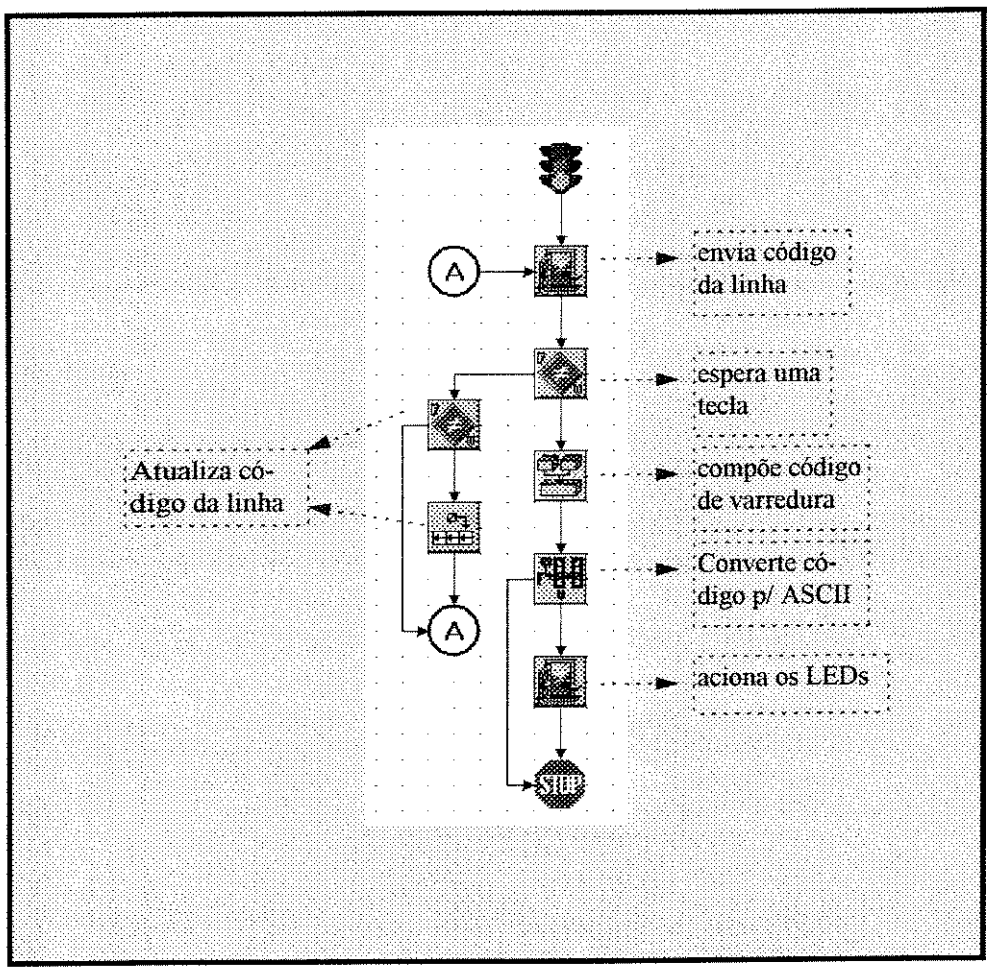


Aplicação A (linearização de um transdutor) em ONAGRO, usada nos teste 1 e 2.

ANEXO 2

```
/* Aplicação B. Usada nos testes 3, 4, 5 e 6 */

/* Define as portas utilizadas */
#define TEC_LINHA (*(char*) 0x1000)
#define TEC_COLUNA (*(char*) 0x1001)
#define LED_CODIGO (*(char*) 0x1002)

void main (void)
{
  char cod_var[] = { 0x18, 0x14, 0x12, 0x11, 0x28, 0x24, 0x22,  0x21,
                     0x48, 0x44, 0x42, 0x41, 0x88, 0x84, 0x82, 0x81 };
  char cod_ascii[16] = "0123456789ABCDEF";
  char linha = 0x01;
  char coluna = 0;
  char codigo;
  char achou = 0;
  int  ind=0;

  while (coluna == 0)  /* Espera uma tecla */
  {
    TEC_LINHA = linha;    /* Envia codigo da linha */
    coluna = TEC_COLUNA;  /* Ler estado das colunas */
    if (linha == 0x08) /* Atualiza codigo da linha */
        linha = 0x01;
    else
        linha = linha << 1;
  }
  codigo = (linha << 4) + coluna; /* Compoe o codigo da tecla */

  /* Procura o codigo ASCII equivalente */
  while ( (!achou) && (ind < 16) )
  {
    if (codigo == cod_var[ind])
    {
       /* Se achou, envia para os leds o codigo ascii da tecla */
       LED_CODIGO = cod_ascii[ind];
       achou = 1;
    }
    else
       ind++;
  }
}
```

Aplicação B (leitura de um teclado) em linguagem C, usada nos teste 3, 4, 5 e 6.

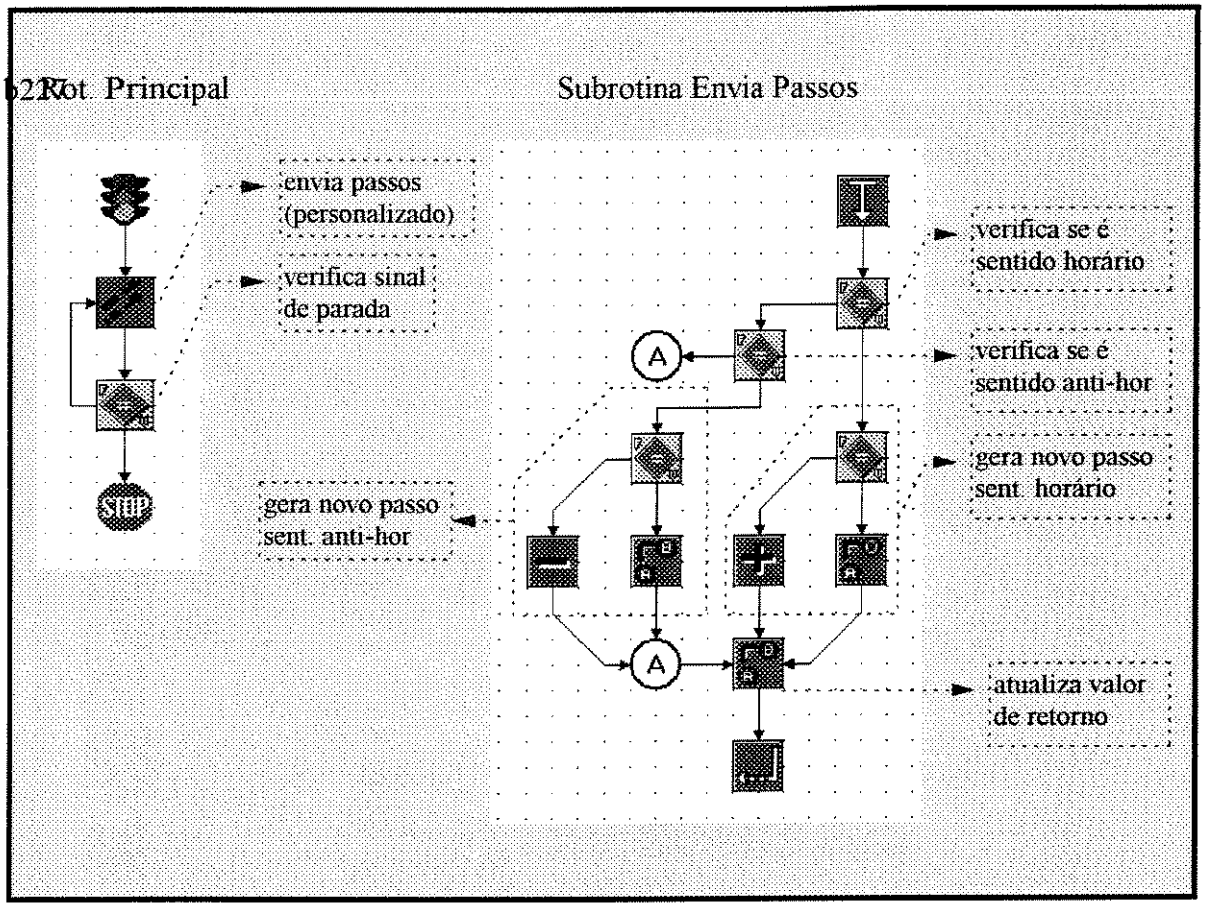


Aplicação B (leitura de um teclado) em ONAGRO, usada nos teste 3, 4, 5 e 6.

A N E X O 3

```
/* Aplicação C.Usada nos teste 7, 8, 9 e 10. */

#define MOTOR (*(char*) 0x1000)  /* porta de saida para acionamento do motor de passo */
#define SENTIDO (*(char*) 0x1001) /* porta de entrada para leitura das chaves       */
#define PARAR   0x00  /* sinal de parada              */
#define HORARIO 0x01  /* sinal de sentido horario     */
#define ANTI_HOR 0x02  /* sinal de sentido anti-horario */

char passos[8] = { 0x0a, 0x08,  0x09, 0x01,
                   0x05, 0x04,  0x06, 0x02 }; /* vetor com os valores para acionamento do motor */
int indice = 0;              /* indexador do vetor de passos              */

char gera_passo (int sentido)  /* subrotina de geracao dos passos           */
{                        /* recebe sentido e retorna valor de acionamento  */
   switch (sentido)
   {
     case HORARIO:  if (indice==7)
                      indice = 0;
                    else
                      indice++;
                    break;
     case ANTI_HOR: if (indice==0)
                      indice = 7;
                    else
                      indice--;
                    break;
   }
   return (passos[indice]);
}

void main (void)  /* rotina principal  */
{
  int sentido;

  do  /* fica em loop ate' que haja pedido de parada */
  {
    sentido = SENTIDO & 0x03;   /* verifica o sentido de rotacao   */
    MOTOR = gera_passo(sentido); /* envia proximo passo para o motor */
  } while (sentido != PARAR);    /* ate' pedido de parada          */
}
```

Aplicação C (controle de motor de passos) em linguagem C, usada nos teste 7, 8, 9 e 10.

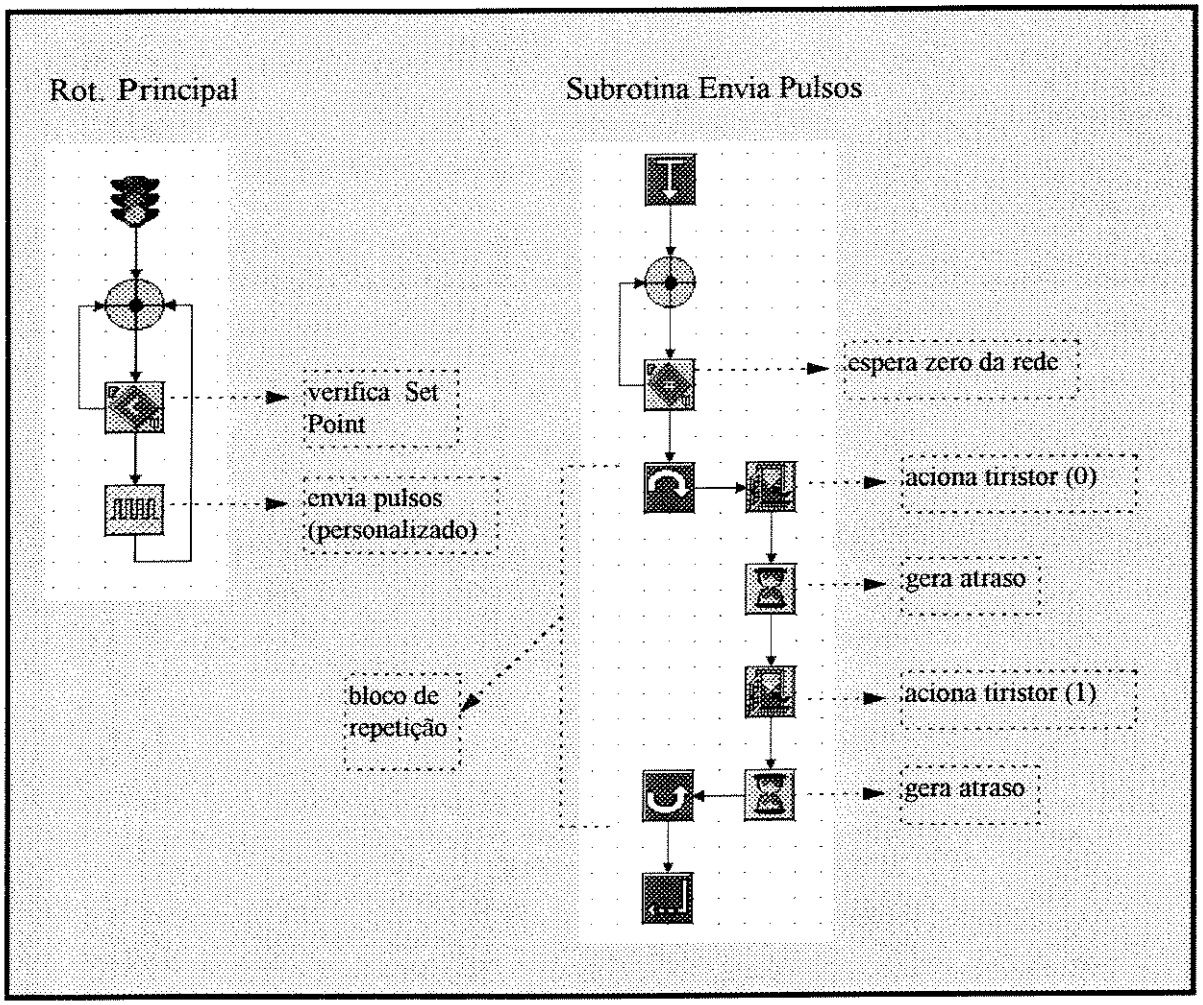


Aplicação C (controle de motor de passos) em ONAGRO, usada nos teste 7, 8, 9 e 10.

ANEXO 4

```
/* Aplicação D.Usada nos teste 11, 12, 13 e 14. */

#define TEMPERATURA (*(char*) 0x1000) /* porta de entrada que indica a temperatura */
#define REDE (*(char*) 0x1001)        /* porta de entrada que indica zero da rede  */
#define TIRISTOR (*(char*) 0x1002)    /* porta de saida de acionamento do tiristor */
#define TEMP_MIN 100  /* valor da temperatura minima (set-point) */

void envia_pulsos(void)  /* envia trem de pulsos ao tiristor */
{
   int i, j;  /* variaveis locais */

   while (REDE != 0); /* Espera zero da rede */

   for (i=0; i<50; i++)
   {
      TIRISTOR = 0;
      for (j=0;j<30000;j++); /* Gera delay */
      TIRISTOR = 1;
      for (j=0;j<30000;j++); /* Gera delay */
   }
}

void main (void)
{
  do
  {
    if (TEMPERATURA < TEMP_MIN)
      envia_pulsos();

  } while (1);
}
```

Aplicação D (controlador ON-OFF) em linguagem C, usada nos teste 11, 12, 13 e 14.

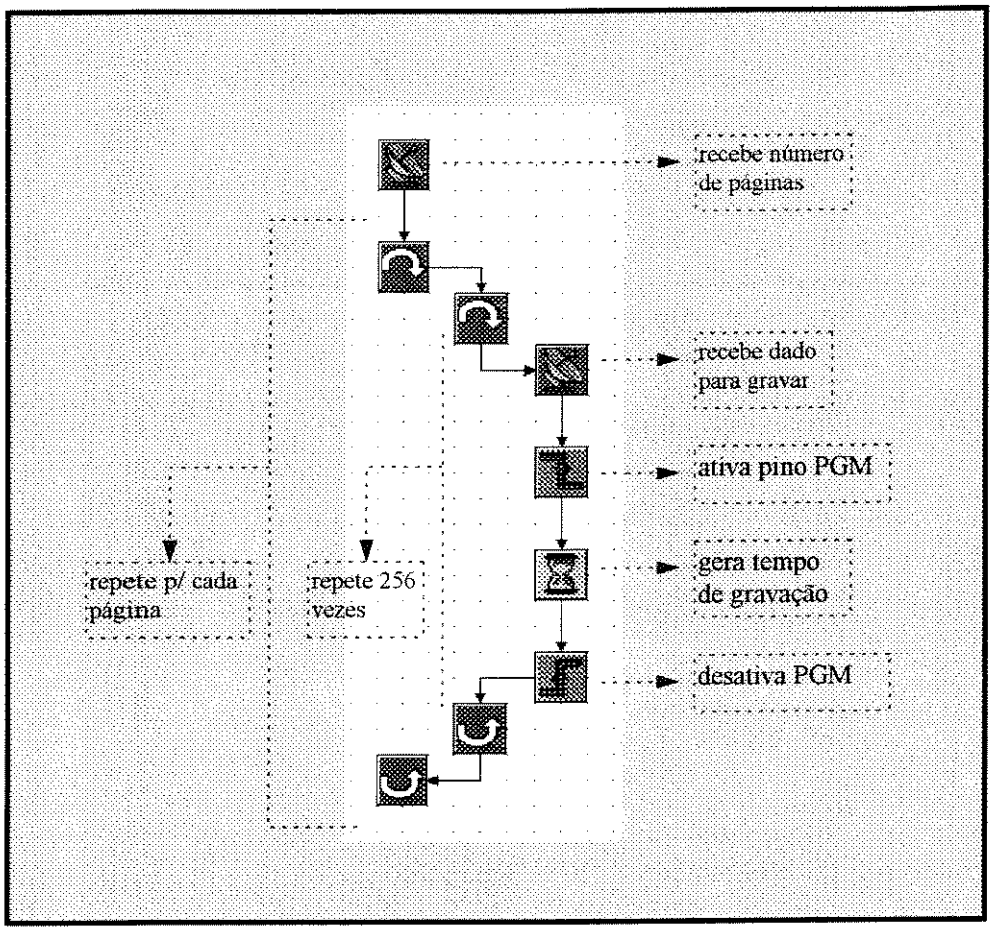


Aplicação D (controlador ON-OFF) em ONAGRO, usada nos teste 11, 12, 13 e 14.

A N E X O 5

```
/* Aplicação E.Usada nos teste 15, 16, 17 e 18.. */

#include      <conio.h>
#include      <8051.h>

#define P_ENDL (*(char*) 0x1000)
#define P_ENDH (*(char*) 0x1001)
#define P_DADO (*(char*) 0x1002)
#define P_CTRL (*(char*) 0x1003)

void init_uart()
{
        TMOD = 0x20;          /* timer 1 auto-reload mode */
        TL1 = -3;
        TH1 = -3;             /* about 9600 baud at 10.695MHz */
        TR1 = 1;              /* enable timer 1 */
        SCON = 0x52;          /* mode 1, receiver enable */
}

char getch()
{
        while(!RI) continue;
        RI = 0;
        if(SBUF == '\r' || SBUF == ('\r'|0x80))
                return '\n';
        return SBUF & 0x7f;
}

void main (void)
{
  int end_low, end_high, delay;
  char npg;
  init_uart();
  npg = getch(); /* ler numero de paginas */
  for (end_high=0; end_high<npg; end_high++)
  {
    P_ENDH = end_high;
    for (end_low=0; end_low<256; end_low++)
    {
                P_ENDL = end_low;
                P_DADO = getch();
                P_CTRL = 0x01;
                for (delay=0; delay<10000; delay++); /* tempo de gravacao */
                P_CTRL = 0x00;
    }
  }
}
```

Aplicação E (programador de EPROM) em linguagem C, usada nos teste 15, 16, 17 e 18.



Aplicação E (programador de EPROM) em ONAGRO, usada nos teste 15, 16, 17 e 18.

REFERÊNCIAS

[1] Scalan, David A., "Should Short Relatively Complex Algorithms be Taught Using Both Graphical and Verbal Methods ?", SIGCSE v. 20 n. 1, Feb. 1988, pp. 185-189

[2] Scalan, David A., "Structured Flowcharts Outperform Pseudocodes: An Experimental Comparison", IEEE Software v. 6 n. 5, Sep. 1989, pp. 28-36

[3] Schultz, Thomas W., "C and the 8051: Programmming and Multtasking", Prentice Hall, 1993

[4] Davis, Willian S., "Systems Analysis and Design", Addison-Wesley, 1983

[5] Springer, S. P., Devtsch, G., "Left Brain, Right Brain", W. H. Freeman and Company, New York, 1985

[6] Calloni, Ben A., Bargert, Donald J., "ICONIC Programming in BACII vs Textual Programming: which is a better Learning Environment ?", SIGCSE Bulletin v. 26 n. 1, Mar. 1994, pp. 188-192

[7] Winblad, A. L., Edwards, S. D., King, D. R., "Software Orientado ao Objeto"., Addison-Wesley/Makron Books, 1991

[8] Kruglinski, David J., "Explorando o Visual C++", Editora Campus Ltda, 1994

[9] Barkakati, Nabajyoti, Hipson, Peter D., "Visual V++ - Guia de Desenvolvimento Avançado", Berkeley Brasil Editora, 1994

[10] Cox, Brad J., "Programação Orientada a Objetos", Makron Books, São Paulo/SP, 1991

[11] Richard S. Wiener, Lewis J. Pinson, "Programação Orientada para Objeto e C++", Addison-Wesley/Makron Books, 1991

[12] Pressman, R. S., "Software Engineering: A Pratitioner's Approach", MacGraw-Hill, 1992

[13] Setzer W. W., Melo J. H., "A Construção de um Compilador", Ed. Campus, Rio de Janeiro/RJ, 1988

[14] MCS-51 Family of Single-Chip Microcomputers User's Manual SIEMENS, 1981

[15] 8051-Based 8 bit Microcontrollers, DATABOOK Philips Components, 1991