UMA MÁQUINA VIRTUAL

PARA SNOBOL4

Ronaldo Zwicker

DISSERTAÇÃO APRESENTADA AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA

UNIVERSIDADE DE SÃO PAULO
PARA OBTENÇÃO DO GRAU DE MESTRE
EM
MATEMÁTICA APLICADA

ORIENTADOR:

Prof. Dr. TOMASZ KOWALTOWSKI

-São Paulo, maio de 1979-

AGRADECIMENTOS

Ao Prof. Dr. Tomasz Kowaltowski, pela valiosa orientação e incentivo,

à minha esposa, pela paciência,

e à Fundação Instituto de Pesquisas Econômicas, pelo serviço gráfico.

ÍNDICE

PREFÁCIO

Capítulo 1. MÁQUINAS VIRTUAIS	1-1
1.1 Estrutura Fundamental	1-1
1.2 Elementos de Microprogramação	1-4
1.2.1 Recursos Básicos de Processadores	1-5
1.2.2 Controle Microprogramado	1-7
1.2.3 Características de Computadores Micropro gramados	1-9
1.3 Sistema Burroughs B1700	1-11
1.3.1 Características Gerais	1-12
1.3.2 Recursos e Funcionamento	1-14
1.3.3 Linguagem de Microprogramação	1-17
1.4 Outros Sistemas Microprogramados	1-18
Capítulo 2. A MÁQUINA SNOBOL	2-1
2.1 Considerações Gerais	2-1
2.2 A Linguagem SNOBOL	2-4
2.2.1 Sintaxe Abstrata	2-4
2.2.2 Avaliação de Expressões	2-6
2.2.3 Comandos	2-11
2.2.4 Desvios	2-13
2.2.5 Blocos	2-14
2.2.6 Programas	2-14
2.3 Estruturas de Dados	2-15
2.3.1 Descritores	2-15
2.3.2 Tabela de Variáveis Naturais	2-16
2.3.3 Blocos	2-18
2.3.4 Tabela de Tipos	2-18
2.3.5 Tabela de Funções	2-18
2.3.6 Tabela de Rótulos	2-21

2.3.7	Estruturas para Tipos Definidos pelo Usuário	2-24
2.3.8	Matrizes	2-31
2.3.9	Tabelas	2-34
2.3.10	Tabela de Palavras Chave	2-35
2.3.11	Estruturas para Tratamento de Atributos ..	2-35
2.4	Código Normal	2-38
2.4.1	Formato de Objetos Tipo CODE	2-39
2.4.2	Tratamento de Rótulos e Desvios	2-43
2.4.3	Operador Negação	2-49
2.4.4	Operador Expressão não Avaliada	2-51
2.4.5	Avaliação de Funções Primitivas Especiais	2-52
2.4.6	Otimização do Código	2-53
2.5	Código para Padrões	2-54
2.5.1	Formato de Objetos Tipo PATTERN	2-54
2.5.2	Construção de Padrões	2-58
2.5.3	Casamento de Padrões em Modo QUICKSCAN ...	2-61
2.5.4	Execução do Código de Casamento de Padrões	2-66
2.5.5	Padrões Compostos e Especiais	2-69
Capítulo 3.	ASPECTOS DA MICROIMPLEMENTAÇÃO	3-1
3.1	Características Básicas	3-1
3.2	Descritores	3-3
3.2.1	Descritores para Cadeias	3-4
3.2.2	Descritores para Valores Numéricos	3-9
3.2.3	Descritores para outros Tipos de Objetos .	3-11
3.2.4	Registrador de Cadeias	3-12
3.3	Processamento de Cadeias	3-15
3.3.1	Funções e Operadores Primitivos	3-15
3.3.2	Padrões Primitivos	3-17
3.4	Manipulação das Estruturas na Memória	3-26
3.4.1	Alocação de Espaço	3-26
3.4.2	Coleta de Áreas Livres	3-27
Capítulo 4.	COMENTÁRIOS FINAIS	4-1

BIBLIOGRAFIA	5-1
APÊNDICE A	A-1
APÊNDICE B	B-1

PREFÁCIO

Este trabalho apresenta o desenvolvimento de uma máquina virtual adequada para a execução de programas em linguagem SNOBOL (especificamente SNOBOL4). A máquina foi desenvolvida com o propósito de implementá-la, sob forma microprogramada, em computadores atualmente disponíveis.

O capítulo um introduz o conceito de nível ou máquina virtual, passando a seguir para a descrição do nível de microprogramação. Também é apresentada, com certo detalhe, a estrutura de um computador microprogramado atual.

O capítulo dois apresenta inicialmente um resumo da sintaxe abstrata da linguagem SNOBOL, e paralelamente assinala suas peculiaridades e pontos importantes para o desenvolvimento da máquina virtual. A seguir são apresentadas as estruturas de dados que serão manipuladas, sendo dada particular atenção às estruturas de código normal e código de padrões. Procurou-se neste capítulo, manter sem modificações a sintaxe e a semântica das construções da linguagem, o que torna complexas algumas estruturas empregadas.

O capítulo três focaliza a implementação da máquina em microcódigo. O computador descrito no capítulo um é escolhido como suporte e, em função de suas características são obtidos os formatos das estruturas elementares de dados. Também é visto o microcódigo de algumas funções primitivas da linguagem.

O capítulo quatro comenta a tarefa efetuada.

CAPÍTULO 1

MÁQUINAS VIRTUAIS

A organização de um computador pode ser vista como uma hierarquia de níveis, onde cada nível representa um novo computador. Esta abordagem conduz a definição de máquinas virtuais, preocupação inicial deste capítulo. Em seguida estudaremos o nível mais elementar da estrutura, evoluindo até a apresentação da organização do sistema Burroughs B1700, como exemplo marcante dos tópicos abordados.

A matéria deste capítulo é composta de assuntos bem estabelecidos e certo grau de detalhe foi omitido. Informações adicionais podem ser obtidas na bibliografia correspondente, particularmente em manuais de sistema dos computadores citados.

1.1 Estrutura Fundamental

Um computador digital é um recurso para resolver problemas através da execução de conjuntos de instruções (programas) que lhe foram fornecidos. A comunicação homem/máquina é feita através de uma linguagem própria, dita linguagem de máquina (L_1), na qual são especificadas as instruções para o computador.

O desenvolvimento de um computador exige, em geral, a definição das instruções básicas que ele será capaz de executar e a implementação destas através dos recursos atualmente disponíveis. Tais instruções são normalmente muito simples em função do barateamento dos custos de engenharia e da diminuição da complexidade dos circuitos. Como consequência, torna-se inconveniente ao usuário utilizar a linguagem L_1 como forma de expressar os seus algoritmos.

Como alternativa, podemos desenvolver uma nova linguagem L_2 conveniente ao usuário. Cada programa escrito nesta linguagem poderá ser executado segundo duas sistemáticas:

- a. Conversão do conjunto de instruções em L_2 para um conjunto equivalente de instruções em L_1 através de um programa disponível na linguagem L_1 dito Compilador. Em seguida o novo conjunto é executado. O processo é chamado de Compilação.
- b. Executar um programa disponível em L_1 , dito Interpretador, que examina cada instrução em L_2 e executa diretamente as instruções de L_1 que lhe são equivalentes. Este processo é chamado de Interpretação.

O processo explícito de compilação ou interpretação pode ser muitas vezes ignorado, e em seu lugar imaginada uma máquina virtual cuja linguagem de máquina é a própria L_2 . O usuário deve imaginar que tudo se passa como se realmente existisse a nova máquina. Além disso, o conceito envolvido pode ser estendido, uma linguagem L_3 pode ser definida, tendo como suporte L_2 ou L_1 , e assim por diante. Desta forma obtemos uma série de níveis ou máquinas virtuais, o nível mais baixo o mais simples e o mais alto, o mais sofisticado. O nível mais baixo será executado pelos circuitos do computador.

Cada máquina, virtual ou não, define uma linguagem constituída de todas as instruções que a máquina pode executar. Por outro lado, cada linguagem também define uma máquina, virtual ou não, capaz de executar todos os programas escritos na linguagem. A estrutura de uma máquina de múltiplos níveis é vista na figura 1.1.

Computadores atuais são normalmente constituídos de dois ou mais níveis. Séries tradicionais são compostas de cinco níveis:

Figura 1.1 - Hierarquia de máquinas virtuais

- nível 1 : É o nível de microprogramação. Os microprogramas são executados diretamente pelos circuitos do computador.
- nível 2 : É o nível da linguagem de máquina tradicional que é executada através de interpretação em nível 1 por microprograma.
- nível 3 : É o nível do sistema operacional, constituído em grande parte de instruções que também comparecem no nível 2. As instruções de nível 3 são executadas através de um interpretador em nível 2 dito sistema operacional. As instruções em nível 2 também são executadas por microprograma.
- nível 4 : É o nível da linguagem de montagem(*). Programas nesta linguagem são traduzidos para linguagem de nível 1, 2 ou 3 e executados em seguida. Permite acesso a recursos dos níveis inferiores.

(*) Em inglês: "assembly language"

nível 5 : É o nível das linguagens de alto nível, tais como ALGOL, LISP, FORTRAN, SNOBOL, etc.. Programas nestas linguagens são em geral traduzidos para os níveis 3 ou 4; eventualmente são interpretadas.

Em alguns computadores não existe o nível de microprogramação, as instruções em linguagem de máquina tradicional são executadas diretamente pelos circuitos. Programas de aplicação podem constituir um sexto nível.

Consideremos um esquema de três níveis:

nível 1 : Nível de microprogramação (execução por circuitos).

nível 2 : Nível de linguagem de máquina adequada a linguagem de alto nível (interpretação por microprograma).

nível 3 : Nível da linguagem de alto nível (tradução para linguagem de máquina).

Neste esquema, o aspecto essencial é a interpretação de instruções de máquina desenvolvidas em função da linguagem de programação e não em função do computador. Ignoramos o nível do sistema operacional a fim de simplificar o raciocínio, mas nada impede sua presença, inclusive podendo estar implícito no nível 1. Transparece uma relativa simplicidade e provável eficiência do esquema, que é viável para diversas linguagens de programação num mesmo computador. Nosso objetivo é estudar e viabilizar um esquema deste tipo. O passo inicial será obter mais subsídios sobre o nível 1.

1.2 Elementos de Microprogramação

As idéias básicas da microprogramação foram lançadas em 1951 por M. V. Wilkes (Wilkes 1951) com o objetivo de fornecer

uma alternativa sistemática para o desenvolvimento das unidades aritméticas e de controle de computadores, que na época eram construídas completamente sob forma de circuitos eletrônicos.

Somente na década de 60, a microprogramação passou a ser efetivamente utilizada em computadores comercializados (a série IBM/360 é marcante). Nestes equipamentos o usuário não tinha acesso às memórias microprogramadas pelo fabricante. A microprogramação era utilizada principalmente como alternativa para implementação de instruções de máquina. No final da década de 60 passaram a ser comercializados computadores microprogramáveis pelo usuário, que dessa forma podia ajustar melhor o computador a seus objetivos.

Infelizmente a microprogramação é uma tarefa que utiliza recursos muito primitivos e por vezes escassos, tornando-se uma atividade incômoda. R.F.Rosin (Rosin 1974) define microprogramação como "a implementação de sistemas esperançosamente racionais através de interpretação em máquinas não racionais". A definição fica aparente nos parágrafos que seguem.

1.2.1 Recursos Básicos de Processadores

Um processador pode efetuar uma série de operações com os elementos que o compõem. Podemos caracterizar três componentes básicos num computador:

- a. **Unidades de Transformação:** São unidades capazes de promover transformações específicas sobre dados através de circuitos adequados. Por exemplo, o somador que é capaz de receber como entrada duas informações, efetuar sua soma e fornecer como saída o respectivo resultado.
- b. **Unidades de Armazenamento:** São unidades capazes de armazenar dados durante um certo período, e têm por característica um comprimento fixo em bits. Por exemplo, os registradores e palavras da memória.

- c. **Vias de Interligação:** São rotas por onde fluem informações entre unidades de transformação e/ou de armazenamento. As informações podem ser dados ou controles, em geral compostos por conjuntos de bits. De preferência, os bits de um conjunto fluem simultaneamente na rota. Cada via é caracterizada pelo número de bits que simultaneamente nela podem caminhar; este número é a largura da via. Normalmente diversas unidades de armazenamento e transformação estão conectadas numa única via de interligação.

Através da conexão destes três componentes, o processador realiza operações primitivas de três tipos:

- a. **Operações de Movimento:** Simplesmente transferem dados de uma unidade de armazenamento para outra.
- b. **Operações n-árias:** Nestas operações uma unidade de transformação recebe como entrada o conteúdo de n unidades de armazenamento através de uma ou mais vias de interligação, efetua a transformação e envia o resultado através de uma via de ligação para uma unidade de armazenamento.
- c. **Operações de Conversão:** São operações n-árias cujo resultado são informações de controle.

A execução de uma operação primitiva exige uma série de controles. Por exemplo, uma operação unária pode exigir a seleção de uma unidade de armazenamento, da via de ligação, da unidade de transformação, de outra via de ligação e da unidade de armazenamento para o resultado. Eventualmente a operação primitiva exige que os controles sejam realizados em uma sequência de instantes de tempo. Por exemplo, numa operação unária, pode ser que a unidade de transformação consuma um tempo relevante na operação e a ativação da via de ligação e da unidade de armazenamento do resultado deva ser feita após, e não simultaneamente.

te, com a ativação da unidade de armazenamento e via de ligação da entrada. De qualquer forma, as operações primitivas em geral são completadas em uma ou duas fases, o que significa um mínimo controle seqüencial.

As operações primitivas encontradas nos computadores atuais são somas, operações booleanas, deslocamentos de bits, etc. Contudo, um computador necessita efetuar operações mais complexas tais como multiplicações e divisões. Um método de implementar estas operações é construir um circuito lógico seqüencial que gera na seqüência adequada e nos instantes corretos as informações de controle para ativar as operações primitivas que armam a operação complexa. Num computador de porte é de se prever uma proliferação destes circuitos o que conduz a uma unidade de controle complexa.

Muitos computadores têm esta estrutura, e as operações complexas constituem as instruções de máquina. Cada instrução será lida da memória e decodificada em sinais de controle que ativam o circuito lógico correspondente; este por sua vez ativa as operações primitivas na seqüência adequada. Desta forma o computador é capaz de executar somente as instruções de máquina para as quais ele foi desenvolvido.

Outro método para implementar operações complexas é através de microprogramação, evitando uma complicada unidade de controle, e adicionando maior flexibilidade no repertório de operações complexas que ela pode executar. Este método é discutido no parágrafo seguinte.

1.2.2 Controle Microprogramado

A série de operações primitivas que constitui uma operação complexa é codificada e armazenada numa memória, em geral distinta da memória do sistema, conhecida por memória de controle ou memória de microprogramas. A série é executada seqüencialmente, e envolve a leitura de cada operação primitiva da memória de controle, sua decodificação e a ativação dos controles correspondentes.

As operações primitivas passam a ser referenciadas como microoperações e são identificadas univocamente pela unidade de controle. Em geral, seqüências de certas microoperações podem ser agrupadas e executadas em paralelo; a estas seqüências dá-se o nome de microinstruções. Assim, cada microinstrução é composta de uma ou mais microoperações. Por simplicidade consideraremos inicialmente cada microinstrução composta por uma única microoperação.

Figura 1.2 - Esquema de um computador microprogramado

Na figura 1.2 apresentamos o esquema de um computador microprogramado. A execução das microinstruções é efetuada em quatro passos:

passo 1: Sob ação da unidade de controle a próxima microinstrução é lida e armazenada num registrador de microinstruções.

passo 2: A microinstrução é decodificada e convertida em sinais de controle pela unidade de controle.

passo 3: Os sinais de controle ativam as operações primitivas

correspondentes.

passo 4: O endereço na memória de controle da próxima microinstrução a executar é obtido, por exemplo, incrementando um registrador de endereço de microinstruções.

A execução de cada microinstrução exige uma seqüência de ações composta de leitura, decodificação e ativação de sinais. Esta seqüência de ações é provocada por circuitos eletrônicos de estrutura muito menos complexa do que os circuitos necessários a execução direta de instruções de máquina.

1.2.3 Características de Computadores Microprogramados

A característica marcante é a possibilidade de serem ou não microprogramáveis pelo usuário. Encontramos no mercado uma grande variedade de sistemas microprogramados, porém, não microprogramáveis pelo usuário. A maioria dos modelos das séries IBM/360, IBM/370, PDP/11 são um exemplo. Relativamente poucos, e em geral de pequeno porte, são os sistemas microprogramáveis pelo usuário e as séries Burroughs/1700, VARIAN/73, HP/21MX são um exemplo.

Sistemas não microprogramáveis pelo usuário normalmente utilizam memórias de controle do tipo que permite somente leitura, relativamente baratas e rápidas em comparação a outras memórias. O objetivo nestes sistemas é obter uma unidade de controle mais simples e flexível, sendo fixa a linguagem de máquina do computador.

Sistemas microprogramáveis pelo usuário, em geral, utilizam memórias de controle do tipo que permite leitura rápida e gravação lenta. São de custo relativamente mais alto e normalmente exige-se boa quantidade de memória de controle, a fim de que, o usuário possa desenvolver seus próprios microprogramas. Estes sistemas também são conhecidos como de microprogramação dinâmica.

Outras características de sistemas microprogramados dizem respeito a memória de controle, unidade lógica e aritmética, registradores locais e repertório de microinstruções.

A memória de controle normalmente é organizada de modo a cada palavra armazenar uma microinstrução. Outras alternativas são: armazenar mais de uma microinstrução por palavra, ou prover uma memória de controle pequena e rápida, alimentada automaticamente e continuamente com microinstruções residentes numa memória mais lenta. Em geral, associados à memória de controle temos dois registradores: o registrador de endereço da próxima microinstrução a executar e o registrador da microinstrução correntemente em execução. Em alguns sistemas também encontramos uma pilha capaz de armazenar endereços de microinstruções, o que permite simplicidade na programação e execução de subrotinas ao nível de microprograma. Ela também permite um mecanismo simples para tratamento de interrupções; na ocorrência de uma, é automaticamente empilhado o conteúdo do registrador de endereço de microinstrução e em seu lugar colocado o endereço de microprograma da rotina de tratamento da interrupção.

A unidade lógica e aritmética em geral admite duas entradas e uma saída e efetua operações aritméticas e booleanas elementares. Normalmente as entradas para a unidade lógica e aritmética podem passar previamente através de unidades de deslocamento e/ou máscara. Unidades lógicas e aritméticas eventualmente admitem três entradas uma das quais pode ser o "vai um" de uma operação anterior.

Registradores ou memória local constituem um conjunto de registradores para armazenamento temporário de informações, são de acesso rápido e residentes na própria unidade de processamento. Não tem função específica, sua aplicação é dada ao nível da máquina virtual, por exemplo, como registrador de instrução de máquina.

Cada microinstrução pode acionar simultaneamente diversas microoperações. Sob este aspecto as microinstruções são classificadas em verticais, diagonais e horizontais. Microinstruções verticais acionam apenas uma microoperação, sua progra

mação é relativamente simples. Microinstruções horizontais acionam uma ou mais microoperações que normalmente são executadas em paralelo. No caso extremo, cada microinstrução controla todos os recursos do processador. Microinstruções horizontais têm comprimento maior em relação às verticais. Microinstruções diagonais discriminam menos paralelismo entre microoperações.

Dependendo do sistema, a microinstrução correntemente no registrador de microinstruções pode ser executada em paralelo com as fases de leitura e decodificação da próxima microinstrução. Conseqüentemente maior velocidade de processamento é obtida.

Outra característica das microinstruções se refere ao número de fases necessárias para sua execução. Na maioria dos computadores existe um relógio interno que emite pulsos em frequência constante, o intervalo entre dois pulsos sucessivos é dito ciclo básico da máquina. Cada microinstrução consome um número inteiro de ciclos durante os quais são emitidos os controles correspondentes às microoperações. Estes controles devem ser emitidos em instantes determinados, responsabilidade que pode pertencer ao usuário e tornar intrincada a tarefa de microprogramação.

1.3 Sistema Burroughs B1700

A série Burroughs B1700 é uma família de computadores com nível de microprogramação genérico, não dirigido para nenhuma linguagem de máquina particular. Permite microprogramação dinâmica simples e apresenta estrutura de fácil entendimento e boa eficiência. Em comparação a sistemas de mesmo porte, a literatura a firma como um marco nos sistemas microprogramados atuais. Em vista de suas qualidades marcantes, descreveremos este sistema com certo detalhe, na intenção de reforçar os conceitos de microprogramação, apresentar uma máquina disponível no mercado atual e escolher posteriormente o sistema como suporte para a máquina virtual que será desenvolvida.

O sistema B1700 permite a interpretação do código de máquina de várias máquinas virtuais. Assim, seu compilador FORTRAN gera código de máquina especialmente desenvolvido para a máquina FORTRAN, o que analogamente ocorre com COBOL, BASIC, RPG e SDL (linguagem de suporte interna). O sistema oferece também um sistema operacional que suporta memória virtual, controle de entrada e saída, microprogramas em disco, multiprogramação, multi interpretação, etc.

1.3.1 Características Gerais

Estudaremos o modelo B1726 do topo da série B1700. É um sistema que apresenta memória principal, memória de controle de até 4096 palavras de 16 bits e capacidade para acoplamento de periféricos através de controladores de entrada e saída. A velocidade de acesso à memória principal é de 667 nanosegundos e à memória de controle é de 167 nanosegundos. O processador funciona a 6 MHz e tem a estrutura indicada na figura 1.3. A relação dos registradores e microinstruções com suas correspondentes funções é apresentada respectivamente nos apêndices A e B.

As microinstruções são do tipo vertical e o usuário não se preocupa com o controle de ciclos da microinstrução nem com a sequência de ações internas que desencadeia. Algumas microinstruções consomem mais tempo do que outras, e acessos à memória são um exemplo. Algumas microinstruções também podem ser executadas em paralelo com outras microinstruções, o que é decidido automaticamente pelo sistema no instante da execução.

As microinstruções podem ser executadas diretamente da memória principal (os sistemas menores da série inclusive não possuem memória de controle). O sistema, a partir do esquema de endereçamento de microinstruções, decide automaticamente se a próxima microinstrução deve ser pesquisada da memória de controle ou da memória principal; a memória principal é tratada como extensão da memória de controle. Também permite-se segmentar o microprograma, mantendo segmentos na memória de controle.

Figura 1.3 - Esquema geral do processador Bl700

O sistema possui 16 registradores rápidos de 48 bits que formam a memória local ou auxiliar, e uma pilha de até 32 entradas para endereços de microinstruções ou para uso como memória auxiliar.

1.3.2 Recursos e Funcionamento

O BI700 oferece uma gama de recursos cuja eficiente administração exige pleno conhecimento de uma série de detalhes. Apresentamos tais recursos, resumidamente, em seguida.

a. Movimento de informações

Quase todos os pares de registradores possuem vias de ligação. Os registradores podem ter comprimentos diferentes, ocorrendo de maneira automática truncamento ou preenchimento com bits nulos no instante do movimento de seus conteúdos. Os registradores L, T, X e Y funcionam como entrada e saída da memória principal, sendo de 24 bits a largura máxima da via de acesso.

O movimento de informação para o registrador TAS automaticamente armazena a informação no topo da pilha; o uso de TAS como origem retira a informação do topo da pilha. Alguns registradores não podem ser usados como destino de informações; são os registradores de saída da unidade lógica e aritmética e os registradores de condições.

O movimento de informações de e para a memória principal é efetuado com o auxílio do campo FA do registrador F e que indica o bit inicial do campo que será manipulado. O número de bits referenciados é especificado na própria microinstrução ou através do campo CPL do registrador C. A leitura de microinstruções da memória de controle ou principal é efetuada com auxílio dos registradores A, TOPM e MBR que indicam o endereço da microinstrução. A microinstrução é lida para o registrador M. O registrador M também funciona como receptor em microinstruções de movimento; neste caso o sistema efetua um ou lógico dos bits do dado movido com os bits da próxima microinstrução. Este efeito

auxilia na decodificação de instruções de máquina.

b. Unidades de lógica e aritmética

O sistema possui uma unidade de lógica e aritmética de 24 bits e outra de 4 bits. A primeira é a unidade principal e tem por entrada os registradores X e Y como operandos e CP (campo do registrador C) como parâmetro. A alteração do conteúdo de X, Y ou CP automaticamente recomputa os valores dos registradores de saída que fornecem a soma e diferença entre X e Y, complementos, máscaras e resultados de operações lógicas de X e Y.

O registrador CP tem importância fundamental porque controla a unidade. Seu campo CPU indica a aritmética a ser empregada na soma e subtração; a aritmética pode ser binária, decimal de 4 bits ou decimal de 8 bits, sendo o resultado gerado de acordo. O campo CPL indica o número de bits de X e Y que devem ser utilizados pela unidade aritmética nas computações, podendo variar de 1 a 24 bits consistentemente com o campo CPU. Finalmente, o campo CYF indica o bit de "vai um" resultante de operações anteriores e que será usado como terceiro operando.

A capacidade de controlar a aritmética empregada é particularmente interessante e torna o sistema eficiente na computação de expressões em FORTRAN (aritmética binária) e em COBOL (aritmética decimal) sem necessidade de conversões. O controle do comprimento de X e Y através de CPL facilita computações com menos de 24 bits ou iterações com mais de 24 bits.

A unidade lógica e aritmética de 4 bits opera de modo completamente distinto, e é acionada através da microinstrução MANIPULAR 4 BITS que atua sobre campos de quatro bits de vários registradores. A finalidade desta unidade é permitir pequenas computações sem interferir com os processos em desenvolvimento na unidade de 24 bits.

c. Registro de condições e estado do sistema

Diversas condições do sistema são assinaladas em registradores específicos. Assim por exemplo, o registrador XYCN re-

gistra a relação entre os valores de X e Y indicando se $X = Y$, $X > Y$ ou $X < Y$. Registra também o bit apontado por CPL em X. A alteração no conteúdo de X, Y ou CPL reposiciona os bits de condições em XYCN. O teste dos bits é através de microinstruções.

Da mesma forma, as condições entre FL e SFL são registradas em FLCN, os estados de X e Y são registrados em XYST, e as situações do bit de "vai um" e do bit de resultado negativo são registradas em BICN.

Um bit em XYST assinala se alguma interrupção ocorreu no sistema e pede tratamento. O registrador INCN e os campos CC e CD do registrador C especificam o tipo de interrupção. A detecção de interrupções no sistema é de competência completa do microprograma. Assim, o microprograma deve de instante a instante testar se alguma interrupção ocorreu, e neste caso proceder o seu tratamento, desviando explicitamente para a rotina adequada.

Várias interrupções podem ser assinaladas. Por exemplo, o sistema oferece proteção de memória com registradores base e limite no endereçamento da memória principal (registradores BR e LR). O endereço dado por F deve se situar entre estes limites em toda operação de acesso a memória. Se tal não ocorrer, é assinalada uma interrupção e o usuário deve prever tratamento da situação.

d. Capacidade para extração e deslocamento de bits

A microinstrução EXTRAIR BITS DE T permite o movimento de campos arbitrários do registrador T para um registrador receptor. Os registradores X, Y e T permitem deslocamento e rotação de seus bits através de microinstruções adequadas. Particularmente, X pode ser concatenado a Y e o conjunto sofrer deslocamento ou rotação como se fosse um único registrador de 48 bits.

Os registradores T, L, C e FL estão subdivididos em campos de quatro bits. Cada campo pode ser referenciado como se fosse um registrador, e seus bits manipulados pela unidade de 4 bits ou testados pelas microinstruções de TESTAR BITS.

c. As microinstruções

O endereçamento de um registrador nas microinstruções é feito através da especificação da linha e coluna que ele ocupa numa matriz de endereçamento. O código da microinstrução é compactado e ocupa 4, 8, 12 ou 16 bits. Algumas microinstruções me recem ser mencionadas.

A microinstrução COMPUTAR FA/FL permite fazer incrementos e/ou decrementos nos registradores FA e FL numa única operação. O objetivo é facilitar a iteração sobre campos da memória. A mesma facilidade acompanha as microinstruções que fazem acessos a memória.

A microinstrução AGREGAR TEMPORÁRIO A FA permite incrementar ou decrementar diretamente o registrador FA pelo valor contido num dos registradores da memória local. Neste caso o objetivo é facilitar a utilização de registradores base e endereços relativos na linguagem de máquina. O registrador F também pode ser carregado ou descarregado diretamente de e para a memória local na forma de 48 bits.

A microinstrução POSICIONAR CP permite simplificação na manipulação de campos da memória com comprimento não múltiplo de 24 bits.

A microinstrução MONITORAR é utilizada para controle e estatística de utilização de determinados trechos do microprogr^{ma}; é necessário auxílio de equipamento externo de monitoração.

A microinstrução NORMALIZAR X normaliza o conteúdo de X numa única operação.

1.3.3 Linguagem de Microprogramação

O sistema possui uma linguagem para desenvolvimento de microprogramas (MIL: Micro Implementation Language). É uma linguagem de montagem que oferece uma série de facilidades encontradas em linguagens de alto nível. Seu compilador gera microcô

digo diretamente executável. Apresentaremos a seguir e ao longo deste trabalho alguns trechos de microprogramas em MIL, e acreditamos serem auto-explicativos os exemplos apresentados.

O exemplo da figura 1.4 define a execução de uma instrução de máquina do tipo somar o conteúdo do campo A ao campo B:

somar (6 bits)	endereço A (16 bits)	endereço B (16 bits)
-------------------	--------------------------	--------------------------

sendo que:

- a. Os endereços dos campos A e B são relativos a uma base dada pelo registrador S4A da memória local e referenciam o bit mais a direita de cada campo.
- b. Os campos A e B na memória representam campos de dígitos de 4 bits sem sinal, o número de dígitos de cada campo é dado por S3A em forma de número de bits.
- c. FA aponta para o começo da instrução de máquina.
- d. Deve ser ignorado "estouro" (*).

1.4 Outros Sistemas Microprogramados

Foram considerados diversos computadores microprogramáveis como alternativa para a máquina virtual proposta. Um exemplo, o sistema Hewlett-Packard HP21MX. Trata-se de um minicomputador de microinstruções diagonais, memória de controle de 4096 palavras de 24 bits divididas em 16 módulos de 256 palavras cada. Quatro dos módulos são implementados como memória que per-

(*) Em inglês: "overflow"

LERINSTRUCAO	READ 6 BITS TO X INC FA	-ler código da instru-
	MOVE X TO M	ção e decodificar
	JUMP FORWARD	
	GO TO SOMA	
	GO TO SUBTRACAO	
	:	
	:	
SOMA		
	MOVE 24 TO CP	-posicionar CP para cál-
	MOVE S4A TO Y	culo dos endereços ab-
	READ 16 BITS TO X INC FA	solutos
	MOVE SUM TO S1A	
	READ 16 BITS TO X	
	MOVE SUM TO S2A	
	MOVE S3A TO FL	-especifica comprimento
	MOVE 4 TO FU	-aritmética decimal
.XXX	BIAS BY F TEST	-controla iterações so-
	GO TO LERINSTRUCAO	bre os campos
	MOVE S1A TO FA	-primeiro operando
	READ 0 BITS REVERSE TO X DEC FA DEC FL	
	MOVE FA TO S1A	
	MOVE S2A TO FA	-segundo operando
	READ 0 BITS REVERSE TO Y	
	MOVE SUM TO T	-armazenar resultado
	CARRY SUM	
	WRITE 0 BITS REVERSE FROM T DEC FA	
	MOVE FA TO S2A	
	GO TO -XXX	

Figura 1.4 - Trecho de microprograma em MIL

mite apenas leitura e contém as microinstruções que interpretam as instruções de máquina do sistema. Dez módulos são reservados para outros microprogramas (sistema operacional, funções intrínsecas, etc.) e dois módulos reservados para o usuário. Sua arquitetura não inclui pilha de endereços. Uma característica interessante é a possibilidade de carregar microprogramas na memória de controle através da execução de funções intrínsecas em FORTRAN ou ALGOL.

Outro sistema considerado foi o Varian 73, minicomputador de microinstruções horizontais com capacidade para 4 módulos de memória de controle com 512 microinstruções por módulo. O módulo número um é do tipo que permite somente leitura e contém o

microcódigo para interpretar a linguagem de máquina do sistema. Como característica, permite leitura de instruções de máquina concomitantemente com a interpretação de outras instruções de máquina. Tem microprogramação bastante complexa devido à horizontalidade e controle de fases das microoperações envolvidas.

Finalmente vale referenciar o computador NANODATA-QM1 de uso geral e arquitetura não usual. Sua característica marcante é a memória de controle de dois níveis. A primeira, rápida, de baixa capacidade e que armazena microinstruções horizontais com microoperações elementares, que interpretam por sua vez as microinstruções verticais armazenadas na segunda memória de controle, mais lenta e de grande capacidade (16 K palavras). As microinstruções verticais são definidas pelo usuário e interpretam por sua vez a linguagem de máquina. As duas memórias de controle são microprogramáveis dinamicamente.

De um modo geral, os sistemas atuais que admitem microprogramação dinâmica oferecem recursos para pequenas aplicações. A capacidade de microprogramação é encarada geralmente como recurso secundário. Neste contexto o sistema B1700 é uma rara exceção.

Mencione-se que a microprogramação tem outras aplicações além da interpretação de linguagem de máquina. Podemos citar aplicações em emulação, sistemas operacionais, otimização de programas, processamento de sinais, sistemas gráficos, diagnóstico de falhas, sistemas que toleram falhas, controle em geral, sistemas de tempo real, etc..

CAPÍTULO 2

A MÁQUINA SNOBOL

Passamos a partir deste ponto a desenvolver o tema central deste trabalho: a definição e viabilização de uma máquina virtual num sistema microprogramado.

Em vista dos recursos oferecidos pelo sistema e pelo fato de ser um sistema de produção de arquitetura atual, optou-se pela implementação em um computador da série Burroughs B1700 , mais especificamente o modelo B1726.

Dentre as muitas linguagens de programação que seriam candidatas a servir de base para a máquina virtual, a linguagem SNOBOL, mais precisamente SNOBOL4 (Griswold, Poage e Polonsky 1971) mostrou ser interessante, seja por sua própria complexidade, seja pela pouca atenção que mereceu até agora, ou seja pela perspectiva de estudar uma linguagem de características extremamente peculiares.

Em síntese, o desenvolvimento da máquina SNOBOL evolui da sintaxe abstrata da linguagem para a definição das estruturas de dados e estabelecimento dos algoritmos necessários à sua manipulação. Neste capítulo pouca referência se faz à microimplementação, sendo este assunto abordado no capítulo seguinte.

2.1 Considerações Gerais

SNOBOL é uma linguagem de programação dirigida à manipulação de caracteres, desenvolvida em 1962 pela Bell Telephone Laboratories, Incorporated. A característica marcante do sistema SNOBOL foi seu desenvolvimento visando plena portabilidade. Deste objetivo resultou uma linguagem de implementação para o sistema, denominada abreviadamente SIL (SNOBOL Implementation Language), do tipo linguagem de montagem independente de máqui-

na. As instruções desta linguagem são supostas de fácil implementação, na forma de macros, na maioria dos computadores atuais.

Deste modo, todo o sistema SNOBOL, compilador e interpretador, foi programado em SIL. Sua implementação exige basicamente a definição das macro-rotinas seguida da compilação (montagem) do sistema SIL para a linguagem de máquina do computador em questão. Nestes sistemas, os programas SNOBOL são traduzidos para um tipo de código que a seguir é interpretado.

Algumas tentativas de obter código diretamente executável em computadores atuais foram feitas: (Dewar e Mccann 1977) e (Santos 1971). Nestes sistemas, em vista da complexidade das estruturas manipuladas, o objetivo é atingido em detrimento da linguagem e às custas de artifícios e particularidades do computador em questão.

Uma alternativa é a definição de uma máquina adequada à execução de programas SNOBOL e sua implementação ao nível de circuitos ou ao nível de microprograma. Diversas tentativas neste sentido já foram efetuadas para vários sistemas de programação, e no nosso caso optamos pelo nível de microprograma.

O detalhamento desta máquina SNOBOL segue as linhas gerais propostas por M.D.Shapiro (Shapiro 1972). É suposto que o leitor esteja familiarizado com a linguagem SNOBOL e com sua implementação em SIL. Além disso, adotamos como objetivo fundamental a definição de uma máquina que mantivesse todas as características da linguagem, tanto no nível sintático quanto no nível semântico.

Certamente a máquina SNOBOL poderia ser implementada de diversas maneiras ao nível de microcódigo, uma das quais seria simplesmente a transcrição para microcódigo das rotinas em SIL. Contudo, pretende-se neste trabalho obter uma estrutura mais adequada para a máquina de modo a torná-la mais eficiente. Assim, a estrutura do sistema hospedeiro também terá que ser considerada, e a estrutura final obtida será função não somente da linguagem mas também do sistema utilizado na implementação. Rela-

cionamos a seguir alguns pontos básicos.

A máquina proposta é capaz de executar instruções de máquina muito poderosas no sentido de que cada instrução pode ter vários efeitos secundários associados, exigir a execução de rotinas especiais de monitoramento, etc.. Parte do trabalho consistiu em identificar estas "super-instruções", que, como veremos, tornam o processo de compilação muito simples.

Em princípio, todos os recursos necessários para a interpretação da linguagem de máquina gerada estão implantados em nível de microcódigo, como por exemplo funções primitivas, operadores, o sistema de monitoramento, etc.. Evita-se assim a necessidade de uma linguagem intermediária, que provavelmente conduziria a uma estrutura do tipo SIL e tornaria menos eficiente o processo de interpretação, além de adicionar um nível a mais de complexidade na estrutura. Por outro lado, uma implementação total em microcódigo tornará extenso o interpretador.

As instruções de máquina geradas provavelmente não serão convenientes para outros sistemas de programação. Contudo, isto não conduz a maiores problemas, visto que a filosofia do sistema B1700 é prover um interpretador em microcódigo para cada máquina virtual.

Em linhas gerais os elementos da estrutura proposta são:

- a. a linguagem de máquina constituída de instruções "poderosas"
- b. um interpretador completamente em microcódigo para estas instruções
- c. um sistema monitor também em microcódigo capaz de controlar e tratar condições de exceção detectadas pelo interpretador (o monitor é considerado uma extensão do interpretador)
- d. as estruturas de dados

Focalizaremos a seguir um dos pontos fundamentais para definição da máquina SNOBOL: as estruturas de dados que deverão

ser manipuladas pelo interpretador. Conseqüentemente o interpretador e o monitor passam a ser função das estruturas escolhidas. O objetivo deste capítulo é detalhar tais estruturas de modo a tornar o interpretador e o monitor elementos viáveis e eficientes.

2.2 A Linguagem SNOBOL

Inicialmente é feito um breve resumo da sintaxe e semântica da linguagem SNOBOL. A apresentação está baseada na sintaxe abstrata utilizada em (Berndl 1975). Pretendemos mostrar, sem formalização exagerada, características peculiares e fundamentais que influenciarão decididamente a arquitetura da máquina virtual.

2.2.1 Sintaxe Abstrata

A sintaxe abstrata serve para indicar de que modo os elementos (ou meta variáveis):

- cad : cadeias de caracteres
- num : números
- id : identificadores
- op : operadores
- exp : expressões
- cmd : comandos
- des : desvios
- bloco: blocos
- prog: programas

podem ser combinados. A formação de cadeias de caracteres, números, identificadores e operadores é análoga a linguagens correntes e pode ser encontrada em (Griswold 1972). A notação {x} indica que o termo x pode ser omitido.

exp ::=	"cad"	literal não numérico
	num	literal numérico
	id	identificador
	op exp	operação unária
	exp ₁ op exp ₂	operação binária
	id({exp ₁ },..., {exp _n })	chamada de função
	id<{exp ₁ },..., {exp _n }>	indexação de matriz

cmd ::=	{exp}	avaliação de expressão
	exp =	atribuição da cadeia nula
	exp ₁ = exp ₂	atribuição
	exp ₁ exp ₂	busca de padrão
	exp ₁ exp ₂ =	substituição pela cadeia nula
	exp ₁ exp ₂ = exp ₃	substituição por cadeia

des ::=	(exp)	desvio normal
	<exp>	desvio direto

bloco ::=	cmd	comando sem desvio
	cmd : desv	comando com desvio incondicional
	cmd : S desv	comando com desvio de sucesso
	cmd : S desv ₁ F desv ₂	comando com desvios de sucesso e falha
	cmd : F desv ₁ S desv ₂	comando com desvios de falha e sucesso
	cmd : F desv	comando com desvio de falha

```

prog ::= {cad1} bloco1
        {cad2} bloco2
        :
        {cadn} blocon
        END {cad}

```

2.2.2 Avaliação de Expressões

A avaliação de expressões em SNOBOL é feita segundo as regras usuais de prioridade, da esquerda para a direita, podendo ter efeitos laterais. O que caracteriza a avaliação é o término da mesma, que pode ser de três tipos:

- a. A avaliação termina com sucesso e o resultado da avaliação é um valor armazenável. É o retorno por valor.
- b. A avaliação termina com sucesso e o resultado da avaliação é um endereço. É o retorno por nome.
- c. A avaliação falha. É o retorno por falha, nenhum objeto é devolvido, e apenas a condição é assinalada.

Em qualquer lugar onde ocorra uma expressão pode haver qualquer um dos três tipos de retornos assinalados. Em geral o tipo de retorno não pode ser determinado durante a compilação. Durante a avaliação da expressão também pode ocorrer um erro fatal, que será interpretado como erro de processamento e normalmente encerra a execução do programa.

Uma expressão que retorna um valor, produz um objeto armazenável de tipo definido na linguagem; estes tipos são:

STRING : cadeia de caracteres

INTEGER : número inteiro

REAL : número real

PATTERN : padrão

ARRAY : matriz

TABLE : tabela

NAME : nome

EXPRESSION: expressão não avaliada

CODE : código

nome definido: tipo definido pelo usuário

Uma expressão que retorna por nome, retorna na realidade um endereço, que não é um objeto definido na linguagem. Internamente ao sistema SNOBOL há endereços associados com:

- cada cadeia de caracteres ou variável natural
- cada nome ou variável criada
- cada palavra chave

Podemos relacionar agora os objetos retornados pelos vários tipos de expressões:

- "cad" : Literais não numéricos retornam por valor o descritor da cadeia ou uma cópia do descritor.
- num : Literais numéricos retornam por valor o descritor do valor ou uma cópia do descritor.
- id : Identificadores retornam um endereço, portanto, é um retorno por nome. O endereço retornado normalmente é a posição do identificador na tabela de variáveis naturais.
- op exp : Operadores unários são avaliados aplicando a função atualmente associada ao operador a seu argumento. Os três tipos de retorno são possíveis.
- exp₁ op exp₂ : Operadores binários são avaliados, aplicando a função atualmente associada ao operador a seus argumentos. Os três tipos de retorno são possíveis.
- id({exp₁}, ..., {exp_n}) : Chamadas de funções são avaliadas pela aplicação da função correntemente associada ao atributo função do identificador id a lista de argumentos. Os três tipos de retorno são possíveis.
- id<{exp₁}, ..., {exp_n}> : Indexação de matrizes retorna por nome o endereço do elemento indexado pelos

expressões. Caso algum dos índices esteja fora dos limites, a indexação retorna falha.

O retorno por falha acarreta falha do comando no qual comparece a expressão, a menos que ela esteja no contexto de um operador negação. A execução do comando é encerrada prematuramente.

Operadores e funções são equivalentes no que concerne à sua estrutura interna; distinguem-se somente ao nível da linguagem fonte. Iremos nos referir aos dois, indistintamente, como funções (primitivas ou não conforme o caso).

Em SNOBOL a avaliação dos argumentos de uma função é efetuada pela própria função se ela for primitiva ou pela função encarregada de chamar funções do usuário. A avaliação dos argumentos na própria função é possível porque o endereço no código onde são calculados os argumentos é transferido como parâmetro. Desta sistemática resulta, que funções do sistema avaliam seus argumentos depois que o controle reside nelas, enquanto que funções do usuário tem os argumentos avaliados antes do controle passar a elas.

Funções podem ser redefinidas através das primitivas DATA, DEFINE e OPSYN. Isto permite, que durante a avaliação dos argumentos de uma chamada de função do usuário, a mesma seja redefinida. A nova função é que deverá ser executada. Contudo, se a nova função for uma função primitiva defrontamo-nos com uma situação ambígua, uma vez que, a mesma deveria ter avaliado seus argumentos. A ambigüidade ocorre na implementação em SIL e provoca término anormal nos sistemas de nosso conhecimento.

A avaliação de uma função pode retornar um objeto do tipo NAME, que é um resultado distinto do retorno por nome de um endereço.

Caso o argumento na chamada de uma função seja um endereço (mas é esperado um valor) então a função se encarrega de

efetuar a coerção do argumento obtendo o seu valor. Novamente em tempo de compilação não é possível prever a necessidade ou não de uma coerção.

O tratamento de funções em SNOBOL é bastante complexo; podemos classificar as funções em 5 tipos:

a - Funções primitivas

São funções intrínsecas do sistema. Devem ser chamadas com número correto de argumentos. Admitem redefinição, porém não podem ser rastreadas (*) diretamente. Sua avaliação envolve:

- desvio para a função
- avaliação e coerção dos argumentos se necessário
- execução
- retorno por meio de um dos três tipos.

b - Funções definidas pelo usuário

São criadas através da chamada da primitiva DEFINE. Sua avaliação envolve:

- avaliação e coerção dos argumentos, uma vez que somente valores podem passar como parâmetros
- armazenamento dos parâmetros formais, nome da função e variáveis locais
- atribuição aos parâmetros formais do valor dos argumentos
- inicialização de variáveis locais com cadeia nula
- teste para rastreamento
- desvio para o ponto de entrada na função
- retorno por qualquer um dos três tipos.

(*) Em inglês: "trace"

c - Funções que criam dados

São criadas através da chamada da primitiva DATA. Sua avaliação envolve:

- avaliação e coerção dos argumentos
- construção de um novo objeto composto dos valores dos argumentos convenientemente dispostos na estrutura do novo objeto
- o objeto criado é retornado por valor (internamente é um apontador para o valor).

d - Funções de seleção de campos

São também criadas através da chamada da primitiva DATA. Sua avaliação envolve:

- avaliação e coerção dos argumentos; só o primeiro será usado
- teste para verificar se o tipo do valor do primeiro argumento é igual ao tipo associado com a função de seleção ; se o teste falha, um erro de execução ocorre
- caso contrário a função retorna (por nome) o endereço do campo selecionado dentro do objeto do usuário.

e - Funções externas

São criadas pela chamada da primitiva LOAD. Sua avaliação envolve:

- avaliação e coerção dos argumentos
- conversão dos valores coercidos para um formato dependente da implementação e adequado à função
- o controle passa à função externa
- após a execução um valor deverá ser retornado.

O atributo função de qualquer variável natural pode

ficar indefinido através de chamadas da primitiva UNLOAD. O atributo é trocado para função indefinida. Apesar de cada variável natural possuir o atributo função, nomes de funções só podem ser computados em tempo de execução se forem argumento da primitiva APPLY.

Dois operadores unários primitivos também merecem cuidado especial. O operador "expressão não avaliada" (*) em lugar de avaliar seu argumento retorna um apontador para o código da expressão argumento. Expressões não avaliadas merecem especial atenção no casamento de padrões. O operador "negação" (¬) retorna por falha se seu argumento ao ser avaliado retornar por valor ou nome, e retorna a cadeia nula por valor se seu argumento retornar por falha.

Finalmente, uma variável natural ou nome pode estar associado com arquivo de entrada. Neste caso antes que um endereço seja coercido para valor, um novo registro deve ser lido. Se o final do arquivo é detectado, a avaliação retorna por falha.

2.2.3 Comandos

A avaliação de expressões num comando pode resultar em um retorno por falha. Neste caso todas as chamadas de funções empilhadas devem retornar por falha, em cascata, até o nível do comando. Neste ponto todo o comando falha, e o restante dele é ignorado. O retorno por falha em cascata pode ser interrompido pelo operador negação.

Cada tipo de comando tem um modo de avaliação:

{exp} O comando de avaliação de expressões pode retornar por falha ou não. Se for o comando vazio então sempre retorna com sucesso.

exp = : No comando de atribuição, a expressão à esquerda é avaliada primeiro; se ela re-

torna por valor ocorre um erro de execução. Caso contrário o lado direito é avaliado e coercido, o valor retornado é atribuído ao endereço retornado por exp_1 . Se o endereço retornado é uma palavra chave, mais testes são necessários.

$\text{exp}_1 \text{ exp}_2$: A busca de padrão avalia e coerce exp_1 e em seguida exp_2 ; conversões pertinentes ocorrem logo após. A seguir, a rotina de casamento de padrões é chamada, podendo ocorrer efeitos laterais tais como novas avaliações e atribuições durante o processo. O retorno da rotina pode ser com sucesso ou falha.

$\text{exp}_1 \text{ exp}_2 =$: Comandos de substituição em casamento
 $\text{exp}_1 \text{ exp}_2 = \text{exp}_3$ de padrões avaliam primeiro exp_1 ; se o retorno for por valor ocorre um erro de execução. Em seguida exp_2 é avaliada e coercida bem como exp_1 é coercida para um valor x. Novamente a rotina de casamento de padrões é chamada. Se o casamento tem sucesso são retornadas duas cadeias: a parte inicial e final de x que não casaram. Agora exp_3 é avaliada e coercida. Finalmente a parte inicial de x, o valor de exp_3 e a parte final de x são concatenados e atribuídos ao endereço inicialmente retornado por exp_1 . Se a rotina de casamento de padrões retorna falha ou a avaliação de exp_3 falha, então o comando todo falha.

Nas diversas situações em que pode haver uma atribuição a va-

riável natural, a variável criada ou a palavra chave, deve-se testar se ela tem associação com arquivo de saída. Neste caso, a transferência do valor para o arquivo é efetuada após a atribuição. Da mesma forma, associação com rotina de rastreamento invoca a rotina após a atribuição.

2.2.4 Desvios

Desvios determinam o próximo comando a ser executado. A avaliação é distinta para as duas formas:

(exp) : A expressão, quando avaliada, retorna o endereço da variável natural cujo atributo rótulo fornece a posição no código para onde deve ser o desvio. O nível indireto é necessário, porque código pode ser gerado em tempo de execução permitindo definição e redefinição de rótulos. Caso a avaliação não retorne um endereço de variável natural, um erro de execução ocorrerá.

<exp> : A expressão é avaliada e coercida; se o valor retornado é do tipo CODE, o desvio é feito para o primeiro comando deste trecho de código, caso contrário ocorre um erro de execução.

Desvios para os rótulos RETURN, FRETURN e NRETURN produzem respectivamente retorno por valor, falha e nome, de função do usuário. Desvios para estes rótulos não são reconhecidos em tempo de compilação. Após o desvio o valor da função é temporariamente salvo, e as variáveis locais e parâmetros formais tem seus conteúdos revertidos para os valores anteriores a chamada da função.

O desvio para FRETURN fará com que a "condição de fa-

lha" seja retornada ao ponto de chamada.

O desvio para RETURN fará com que o valor temporariamente salvo seja retornado ao ponto de chamada; o valor não é alterado.

O desvio para NRETURN fará com que o valor temporariamente salvo seja retornado por nome. Portanto, será retornado um endereço. Se o valor temporário for do tipo STRING, INTEGER ou REAL, é retornado o endereço da variável natural com este nome. Se o valor temporário é do tipo NAME é retornado o endereço da variável (criada) com este nome. Se o valor temporário é de outro tipo, é retornado o endereço nulo e qualquer tentativa de utilizar este endereço causará erro de execução posterior.

Ao tempo do retorno, a palavra chave &RTNTYPE indica o tipo de retorno que ocorreu. Além disso, &FNCLEVEL que houvera sido incrementada no instante da chamada da função, é decrementada de um.

2.2.5 Blocos

Um bloco é composto de um comando mais as opções de desvio em caso de falha ou sucesso. Sua avaliação envolve:

- &STCOUNT é incrementado de 1, a menos que o comando seja vazio
- se &STCOUNT excede &STLIMIT um erro de execução ocorre
- o comando é executado
- o desvio necessário é avaliado e executado.

2.2.6 Programa

Um programa é uma sequência de blocos precedidos ou não

de rótulos para os quais podem ser efetuados desvios. Sua avaliação é obtida mediante a execução seqüencial dos comandos. A seqüência é desobedecida apenas se um desvio (incluindo chamadas e retornos de função) for executado. A seqüência é retomada a partir do ponto atingido pelo desvio.

Um programa é um objeto dinâmico, isto é, código pode ser gerado e executado em tempo de execução do programa.

2.3 Estruturas de Dados

As estruturas de dados necessárias à execução de um programa SNOBOL são pouco tradicionais se comparadas com as estruturas necessárias a linguagens de uso comum. Em vista disto, elas serão tratadas apenas no contexto da linguagem SNOBOL. As estruturas de dados caracterizam a máquina SNOBOL e são análogas, porém distintas ao nível de implementação, às propostas em (Griswold 1972).

As estruturas propostas por Griswold são adequadas para a implementação em SIL. Mas, tornam-se inadequadas quando os objetivos passam a ser execução direta do código produzido pelo compilador, minimização de espaço necessário e minimização do tempo de execução. As características do computador que suportará a máquina virtual também precisam ser consideradas. As estruturas que pretendem atender estes objetivos estão descritas a seguir.

2.3.1 Descritores

Como em SNOBOL não existem declarações, e em tempo de execução uma variável pode armazenar qualquer tipo de objeto, é necessário representar cada valor juntamente com seu tipo. Uma representação destas chamamos de descritor. Esquemáticamente temos:

onde T é o tipo do descritor e V é o valor propriamente dito. Se a capacidade do descritor for insuficiente para armazenar V então em seu lugar constará um apontador para o local onde o valor está armazenado.

O conceito de descritor é estendido para representar também objetos internos ao sistema.

2.3.2 Tabela de Variáveis Naturais

Uma variável natural é caracterizada através de seu nome, atributos a ela associados e seu valor atual. Estas características são especificadas por três descritores, dando origem a representação esquemática:

Em SNOBOL, qualquer cadeia de caracteres constitui uma variável natural, com exceção da cadeia nula. Qualquer variável natural é considerada alocada ao início do processamento, com valor inicial igual a cadeia nula. Há algumas exceções que são as variáveis naturais com valor pré-definido diferente de cadeia nula.

A estrutura de dados conveniente para tratar esta sistemática é uma tabela com todas as variáveis naturais realmente referenciadas no programa além das pré-definidas. Como, durante a compilação, não é possível detectar todas as variáveis naturais que ocorrerão durante a execução, a tabela é parcialmente construída pelo compilador com possibilidade de expansão durante a execução.

A cada variável natural fica assim associado um número ou endereço, que é a entrada na tabela na qual ela ocorre. Referências a variáveis naturais no código objeto serão, em geral, através de seus endereços na tabela. Entretanto, algumas situações exigem referência através do nome; nestes casos uma busca na tabela é necessária. A técnica de espalhamento (*) será utilizada para proporcionar maior velocidade nas buscas.

O esquema final da tabela está representado na figura 2.1 e inclui os descritores de ligação da cadeia de espalhamento.

Figura 2.1 - Tabela de variáveis naturais

O descritor de atributos de uma variável natural relaciona os atributos presentes e ausentes da variável, e o bloco com informações adicionais que permitam o tratamento destes atributos. Os atributos de variáveis naturais são:

- associação com arquivo de entrada
- associação com arquivo de saída
- rastreamento por modificação de valor
- uso como rótulo

(*) Em inglês : "hashing"

- uso como palavra chave
- uso como função (não tratado pelo descritor de atributos).

2.3.3 Blocos

Blocos são estruturas de aplicação geral constituídos de seqüências de descritores precedidos por um descritor que especifica o número de descritores que compõem o bloco, o tipo de bloco, além de outras informações que forem necessárias.

2.3.4 Tabela de Tipos

A tabela de tipos é uma tabela de descritores dos nomes de cada tipo de dado do sistema ou criado pelo usuário. A cada nome fica associado um número x que é a posição do nome do tipo na tabela. Este índice comparece nos descritores de valores caracterizando o tipo de valor descrito, e permite acesso direto à tabela toda vez que o nome explícito do tipo for necessário.

2.3.5 Tabela de Funções

A tabela de funções fornece o nível indireto necessário a chamadas de funções. Ela engloba não somente funções definidas pelo usuário mas também funções primitivas do sistema.

Cada entrada nesta tabela se compõe de três descritores:

Em SNOBOL, o nome de uma função só pode ser computado em tempo de execução se for argumento da primitiva APPLY. Assim, não há necessidade de ligação da tabela de variáveis naturais para a tabela de funções.

A criação de novas funções e a redefinição de antigas é possível através de diversos meios, contudo, o volume destes comandos num programa espera-se ser pequeno. Conseqüentemente, nenhuma técnica especial de busca na tabela de funções é proposta.

Os atributos de uma função estão relacionados no descritor de atributos, que também relaciona um bloco com informações adicionais necessárias para computá-los. Os atributos que podem ser relacionados são:

- rastreamento de chamada de função
- rastreamento de retorno de função.

O descritor do bloco de definição da função relaciona o tipo de função (primitiva ou não), o número de argumentos, e o bloco de definição da função. O bloco de definição relaciona por sua vez:

- o número de entradas no bloco
- o nome da função
- o nome do ponto de entrada da função
- os nomes dos argumentos formais
- os nomes das variáveis locais.

Todos os nomes relacionados são na realidade endereços da tabela de variáveis naturais, a menos do nome do ponto de en

trada que é um endereço da tabela de rótulos. Todas as entradas do bloco de definição são construídas quando da execução da função DEFINE para a função que está sendo criada. Funções primitivas não possuem bloco de definição, em seu lugar consta um endereço de entrada já fixo no microcódigo do sistema.

A estrutura resultante da execução do comando de definição de função DEFINE("F(X1,X2)L1","E") e uma chamada de F podem ser vistas na figura 2.2. (Observação: nas figuras que seguem, a notação "x" representa o descritor do nome x).

Figura 2.2 - Estrutura resultante da execução de DEFINE("F(X1,X2)L1","E")

Observe-se que o descritor do nome da função e o endereço da variável que conterá o resultado da chamada da função têm tarefas distintas. Elas podem ser observadas em funções que foram redefinidas. Por exemplo, OPSYN("F", "G") implicará numa fun

ção de nome F, porém a variável resultante terá em princípio o nome G.

O esquema proposto fornece um acesso rápido na chamada de funções, pois cada referência no código aponta diretamente para uma entrada na tabela de funções. Este esquema é uma variação do empregado em SIL. Outra alternativa seria tratar funções como atributos de variáveis naturais, contudo, um nível in direto adicional e mais entradas na tabela de variáveis naturais seriam necessários.

Chamadas e avaliações de funções serão tratadas mais adiante.

2.3.6 Tabela de Rótulos

A tabela de rótulos relaciona os rótulos dos blocos no programa fonte com endereços no código objeto correspondente. A tabela torna mais rápidos os comandos de desvio do programa fonte, com rótulos explícitos não computados, do tipo : (x), onde x é um identificador. Observando que a linguagem SNOBOL não oferece nenhum recurso para iteração de trechos de código fonte além de desvios, e a grande maioria dos desvios tem rótulos explicitados, fica evidente a vantagem da tabela. Cada entrada na tabela é composta de 3 descritores:

--	--	--

descritor
do nome
do rótulo

descritor
de
atributos

descritor do
endereço de
desvio

O descritor do nome do rótulo poderia ser omitido, já que, referências através do nome são realizadas sempre com auxílio da tabela de variáveis naturais. A presença do nome é conveniente para tratar rastreamento de rótulos.

O descritor de atributos relaciona o atributo "rastrea

mento de rótulo" e o bloco com informações adicionais necessárias ao rastreamento. O descritor do endereço de desvio especifica o ponto do código objeto correspondente ao rótulo no programa fonte.

Com a tabela de rótulos, o processo de compilação passa a ser de um passo. Basicamente, os passos seguidos pelo compilador ao encontrar um rótulo x de um bloco são:

- a. Percorrer a tabela de variáveis naturais até encontrar a variável x ou criar uma entrada para ela.
- b. Se a variável não possui o atributo "uso como rótulo" significa que x ainda não ocorreu nem como rótulo nem como desvio explícito. Cria-se uma nova entrada na tabela de rótulos com o endereço atual do código que está sendo gerado. Posiciona-se o atributo "uso como rótulo" e liga-se o bloco de atributos da variável à entrada criada na tabela de rótulos.
- c. Se a variável possui o atributo "uso como rótulo", a partir dele obtém-se a entrada na tabela de rótulos. Se o correspondente descritor de endereço estiver vazio, completa-se com o endereço atual do código gerado. Caso contrário, o rótulo já existe e um erro de compilação ocorre.

Se o compilador encontrar um identificador x num desvio explícito, os passos efetuados são:

1. Percorrer a tabela de variáveis naturais até encontrar a variável x ou criar uma entrada para ela.
2. Se a variável não possui o atributo "uso como rótulo" significa, que x não ocorreu ainda nem como rótulo nem como desvio explícito. Cria-se uma entra-

da na tabela de rótulos contendo por ora somente o nome do rótulo. Posiciona-se o atributo "uso como rótulo" e liga-se o bloco de atributos da variável à entrada criada na tabela de rótulos. Um desvio de tipo imediato, com esta entrada, é gerado no código.

3. Se a variável já possui o atributo "uso como rótulo", um desvio imediato com a entrada na tabela de rótulos é gerado no código objeto.

Além de desvios imediatos devem ser considerados os desvios referidos nesta implementação como não imediatos. São gerados quando no programa fonte o desvio for do tipo $:(y)$, sendo y uma expressão que avaliada retorna o endereço de uma variável natural; exigindo que o desvio se efetue através da tabela de variáveis naturais. A estrutura resultante para desvios imediatos e não imediatos está assinalada na figura 2.3.

Figura 2.3 - Desvios imediatos e não imediatos

Fica a observação que desvios não imediatos também pode-

riam ser implementados pesquisando o nome da variável natural retornada, na tabela de rótulos.

Durante a execução de um programa pode haver chamadas da função primitiva CODE, podendo ocorrer a redefinição ou inclusão de novos rótulos na tabela. O processo de compilação é idêntico ao apontado anteriormente. O único detalhe é a não detecção da duplicação de rótulos, mas a conseqüente redefinição dos mesmos.

Desvios do tipo :<z>; onde z é uma expressão ou variável com valor tipo CODE, são executados sem a participação do esquema proposto. O código correspondente é atingido através do descritor de código retornado na avaliação de z. Novamente, este tipo de desvio é reconhecido em tempo de compilação.

Ao nível de máquina permanecemos com três instruções de desvio: imediato, não imediato e para objeto tipo CODE. Estas instruções serão comentadas mais adiante.

Finalmente, assinale-se que em SIL todos os tipos de desvios são realizados através da tabela de variáveis naturais.

2.3.7 Estruturas para Tipos Definidos pelo Usuário

Objetos de tipo definido pelo usuário são seqüências de variáveis chamadas campos. A função primitiva DATA define objetos deste tipo. Por exemplo, DATA("COMPLEX(R,I)") define o tipo COMPLEX com campos denominados R e I. Além disso, COMPLEX passa a ser função de criação de objetos, e R e I passam a ser as funções de seleção de campos de objetos do tipo COMPLEX.

Internamente, um objeto de tipo definido pelo usuário é um descritor cujo campo de valor descreve dois blocos: um bloco de descritores dos valores de cada campo e um bloco opcional de descritores de atributos de cada campo do objeto. O bloco de descritores de atributos só estará presente se pelo menos um dos campos possuir algum atributo ativo.

Estruturas definidas pelo usuário serão tratadas sob cinco aspectos:

- a. definição da função de criação do objeto
- b. definição das funções de seleção de campos do objeto
- c. criação do objeto
- d. referência a campos do objeto
- e. atributos do objeto

a. Definição da função de criação do objeto

A execução da primitiva DATA com protótipo $D(F_1, \dots, F_N)$ fornece os seguintes resultados:

- i. É alocado um bloco de definição para a função D que cria objetos, composto de descritores para:
 - número de entradas e tipo de bloco
 - código interno do tipo D
 - nome do tipo D
 - nomes das funções de seleção $F_1, \dots, F_N$

Os nomes das funções de seleção são relacionados no bloco para permitir a obtenção de nomes de campos de tipos do usuário através da primitiva FIELD.

- ii. São criadas as funções de seleção de campos.
- iii. O tipo D é acrescentado a tabela de tipos.
- iv. É alocado um descritor para o bloco de definição da função D relacionando: o tipo da função (função de criação), o número de campos do tipo e o bloco de definição propriamente dito.
- v. O descritor do bloco de definição mais o nome da função são acrescentados a tabela de funções.

A função de criação propriamente dita é uma função primitiva interna ao sistema. Em SIL tal função (DEFDAT) é invocada quando uma referência a função de criação de objetos é feita; ela avalia os argumentos e em seguida cria o objeto. Entretanto, se durante a avaliação dos argumentos a função de criação for redefinida, seja através de OPSYN, DEFINE ou DATA, pode ocorrer término anormal da execução (como foi constatado no sistema SNOBOL implantado em SIL utilizado nos testes).

Em SIL, o problema decorre do fato de os argumentos serem avaliados pela própria função DEFDAT, como se fosse uma chamada de função do usuário. Isto permite que o bloco de definição seja alterado, contudo, esta eventual alteração não é considerada por DEFDAT.

No sistema proposto somente duas alternativas são possíveis: ou considerar funções de criação como primitivas ou como do usuário. As duas alternativas conduzem a soluções corretas e uma delas deve ser adotada, já que a bibliografia não é clara a respeito. A solução propõe usar funções de criação como sendo funções do usuário, o que parece mais coerente com o próprio fato destas funções serem definidas pelo usuário. Com isto, uma função deste tipo pode ser redefinida indistintamente durante a avaliação dos argumentos de chamada.

b. Definição das funções de seleção de campos do objeto

A criação das funções referidas no item ii do tópico anterior requer a inclusão de cada função de seleção na tabela de funções. Assinale-se que chamadas de funções que criam objetos ou selecionam campos não diferem de chamadas comuns de funções, não sendo distinguíveis em tempo de compilação.

A obtenção de cada função F_i se inicia tentando localizá-la na tabela de funções. Pode ocorrer:

- a. Se F_i ainda não existe como função de seleção, é alocado um bloco de definição de função que relacionará descritores para:

- número de entradas e tipo de bloco
- código do tipo a que se refere a função
- ordem do campo dentro do tipo considerado

Particularmente, se F_1 existir como função (não de seleção), então ela será redefinida.

- b. Se F_1 já existe como função de seleção definida sobre outro tipo de dados, então dois descritores a mais são acrescentados ao bloco de definição de F_1 , sendo preenchidos com as duas últimas informações indicadas em a.

A função de seleção propriamente dita é uma função primitiva interna ao sistema. Em SIL tal função (FIELD) é utilizada nos mesmos moldes da função DEFDAT, apresentando os mesmos problemas já assinalados para esta última. Em consequência disso, no sistema proposto, novamente adotou-se tratar funções de seleção como funções do usuário, permitindo redefinição durante a avaliação dos argumentos de chamada.

A estrutura resultante da chamada da função primitiva $DATA("D(FA,FB)")$, supondo que o tipo D terá código k e FB já é de seleção de campo de outro tipo (com código j), está indicada na figura 2.4.

c. Criação do objeto

A criação de um objeto é feita ao nível da linguagem através da chamada de função de criação. Internamente, o sistema terá empilhado a entrada na tabela de funções correspondente a função chamada, e terá empilhado todos os argumentos já avaliados; argumentos faltantes são completados com cadeias nulas e argumentos em excesso são desprezados. O descritor do bloco de definição da função também é empilhado e a função interna de criação de objetos é chamada. Nesta função:

Figura 2.4 - Estrutura resultante da execução de `DATA("D(FA,FB)")`

- a. Um bloco é alocado e os argumentos empilhados são eventualmente coercidos e nele copiados. O descritor de identificação do bloco armazenará o total de argumentos copiados, o tipo de bloco e o valor nulo representando o endereço do bloco de descritores de atributos, ausente até este instante.
 - b. É criado um descritor que relaciona o tipo deste objeto e o bloco criado em a. O descritor é retornado por valor.
- d. Referência a campos do objeto

A referência ao campo de um objeto é feita ao nível da linguagem através da chamada de função de seleção de campos. Internamente, o sistema terá empilhado a entrada na tabela de fun

ções correspondente a função chamada, e todos os argumentos já avaliados. Somente o primeiro argumento será usado, se este fal_{tar}, uma mensagem de erro será fornecida. O descritor do bloco de definição da função também é empilhado e a função interna de seleção de campos de objetos é chamada; seus passos são:

- a. O argumento empilhado é eventualmente coercido obtendo-se seu valor. A partir do valor, que deve ser de algum tipo definido pelo usuário, é obtido o tipo do objeto e a localização do bloco de descritores dos valores dos campos do objeto.
- b. O bloco de definição da função de seleção é pesquisado afim de localizar uma entrada que tenha o tipo do objeto.
- c. Se a pesquisa tem sucesso, a ordem do campo no objeto é obtida, se não tiver sucesso uma mensagem de erro é enviada.
- d. A partir da localização do bloco de descritores dos valores e da ordem do campo é montado um par contendo o endereço do bloco de descritores e a ordem ou endereço relativo do campo no objeto. O par de endereços é retornado por nome.
- e. Atributos do objeto

Objetos de tipo do usuário possuem valor especificado através de dois blocos: um com descritores de valores, e um com descritores de atributos. O bloco de descritores de atributos é opcional estando presente por completo somente se algum campo possuir algum atributo ativo.

Com esta implementação pretende-se evitar a alocação

sistemática de espaço para atributos, uma vez que em geral eles não estão presentes para a grande maioria das estruturas processadas num programa. Em princípio, cada campo de um objeto pode ter qualquer dos atributos relacionados:

- associação com arquivo de entrada
- associação com arquivo de saída
- rastreamento por modificação de valor.

Endereços de campos de objetos de tipo do usuário passam a ser sempre pares de endereços:

- o endereço do bloco de descritores de valores
- o endereço relativo do campo no objeto.

O endereço do bloco de descritores de valores dá acesso ao bloco de descritores de atributos, já que o endereço deste último consta do descritor identificação do primeiro. Permite a criação eventual de um bloco de descritores de atributos e a ligação deste ao bloco de descritores de valores. A figura 2.5 mostra a estrutura resultante da execução de três comandos indicados.

Figura 2.5 - Exemplo para atributos em objetos de tipo do usuário

Convém assinalar, que o par de endereços também estará presente em objetos do tipo NAME quando referidos a nomes de campos de objetos de tipo do usuário. Esta situação ocorre no exemplo da figura 2.5 quando o operador $\cdot$ é aplicado a $R(P)$, retornando um NAME.

2.3.8 Matrizes

Matrizes são estruturas criadas através da execução da primitiva ARRAY. Por exemplo:

```
X = ARRAY("2:3,11:14,1:2")
```

cria uma matriz e a atribui como valor à variável natural X. A matriz criada é de três índices e o protótipo especifica também os limites entre os quais devem estar os índices numa referência a um elemento da matriz. Referências a matriz são da forma $X\langle I,J,K \rangle$ e são reconhecidas em tempo de compilação. Referências também podem ser feitas através da primitiva ITEM.

A implementação de matrizes em SNOBOL deve considerar alguns aspectos:

- possibilidade de tratar matrizes como valores
- teste de limites em indexações
- permitir que elementos de matrizes possuam atributos
- ocupar o menor espaço possível
- permitir rapidez nas indexações.

A maneira tradicional de implementar matrizes como estruturas lineares (tipo FORTRAN) adotada na implementação SIL, oferece algumas desvantagens, tais como: toda estrutura precisa ser alocada, e a indexação exige multiplicações. A alternativa sugerida em (Shapiro 1972), com algumas alterações, será descrita e adotada como solução.

A solução proposta utiliza um esquema de vetores de alocação (*) que apontam ou para novos vetores de alocação ou

(*) Em inglês: "dope vectors"

para vetores de descritores de valores ou atributos. Este esquema usa mais memória que um esquema linear para matrizes não esparsas, contudo, é adequado para sistemas com administração dinâmica de memória.

A estrutura resultante da execução de um exemplo pode ser vista na figura 2.6 e descreve a implementação proposta.

Figura 2.6 - Exemplo da estrutura para matrizes

A ocorrência do protótipo da matriz no bloco de definição da matriz é usada pelas primitivas PROTOTYPE e DUMP.

Como se depreende da figura, os limites dos índices ocorrem no bloco de definição da matriz, que também contém o a-

pontador para o primeiro vetor de alocação da matriz. Supondo que este apontador tenha o valor a, então o acesso ao elemento $X\langle I, J, K \rangle$ é obtido computando a expressão:

$$\text{conteúdo}(\text{conteúdo}(a+I-2)+J-1)+K-1$$

que é o endereço do elemento considerado. Como a computação po de tanto se referir ao descritor de valor do elemento quanto ao descritor dos atributos do elemento, novamente o endereço final deverá ser um par:

$$(\text{conteúdo}(a+I-2)+J-1, K-1)$$

que permite acesso a variável $X\langle I, J, K \rangle$. Portanto, a função interna que computa a variável resultante de indexações, deverá retornar o par de endereços por nome.

Durante o processo de indexação, se algum vetor de descritores de alocação não estiver alocado, ele o deverá ser, e em seguida a indexação deve prosseguir. Um vetor não estará alocado se seu endereço for nulo (indicado por 0 na figura 2.6). O vetor de descritores de valores, pode ou não estar alocado ao final da indexação. Caso seja feita uma atribuição ao par de endereços e o elemento não está alocado, então o vetor é criado e o valor atribuído. Caso seja feita uma coerção do par de endereços e o elemento não está alocado, é devolvida a cadeia nula. O mesmo esquema vale para o vetor de descritores de atributos.

A indexação está sujeita a diversas situações de exceção. Caso durante a indexação um índice não puder ser convertido para um inteiro, ou cair fora dos limites, ou se houver índices em excesso, um erro de execução é assinalado. Se faltarem índices eles serão considerados como tendo valor nulo.

Caso a matriz seja inicializada com certo valor, não nulo, toda a estrutura deverá ser alocada, os vetores de descritores de valores inclusive.

2.3.9 Tabelas

Tabelas são criadas através de chamadas da primitiva `TABLE`. `TABLE(M,N)` cria uma tabela com inicialmente M entradas. Se estas se esgotarem durante o processamento, mais entradas serão acrescentadas à tabela, na forma de conjuntos de N elementos, à medida do necessário.

Uma estrutura deste tipo é constituída por uma relação de pares índice-valor. Qualquer indexação na tabela exige uma busca do índice de referência na seqüência de pares. Se ele for localizado, o resultado da busca é o valor associado; se não for localizado o par formado pelo índice de referência e a cadeia nula é inserido ao final da tabela; o resultado será a cadeia nula.

Tabelas são implementadas como blocos, o descritor de identificação do bloco relaciona o número de entradas neste trecho da tabela, um apontador para a eventual continuação da tabela e um apontador eventual para o bloco de descritores de atributos dos elementos deste trecho da tabela.

A figura 2.7 mostra uma tabela com continuação e trecho com bloco de descritores de atributos.

Figura 2.7 - Estrutura para tabelas

Novamente, qualquer indexação retorna um par de endereços composto do endereço do começo do bloco e do índice do ele-

mento dentro deste bloco. Esta sistemática permite acesso ao elemento como variável, e minimiza o espaço necessário para tratamento de eventuais atributos.

2.3.10 Tabela de Palavras Chave

O operador primitivo & retorna por nome palavras chave não protegidas e retorna por valor palavras chave protegidas. Em princípio, endereços de palavras chave devem ser reconhecidos em tempo de execução, pois atribuições a elas alteram o estado do sistema.

As palavras chave constituem duas tabelas, alocadas no começo da memória. Acessos às tabelas são efetuados pelo operador primitivo & através do bloco de atributos da variável natural argumento. Em função de seu uso constante e do espaço disponível, algumas palavras chave poderão ser alocadas diretamente em registradores do processador.

Uma das tabelas é constituída das palavras chave protegidas e a outra das palavras chave não protegidas. Como a linguagem permite rastreamento de algumas palavras (ERRLIMIT , FNCLEVEL, STCOUNT e STFCOUNT) por mudança de valor, bem como associação com arquivos de entrada e saída, resulta um descritor de atributos adicional para cada entrada nas tabelas. A estrutura de acesso está descrita na figura 2.8.

2.3.11 Estruturas para Tratamento de Atributos

Atributos são propriedades adicionais, em geral ausentes, que podem ser associadas (ativadas) com determinados objetos da linguagem SNOBOL. As propriedades que devem ser consideradas nesta implementação são:

- a. Associação de variáveis com arquivos de entrada ou saída

Figura 2.8 - Acesso a palavras chave

- b. Rastreamento de variáveis, palavras chave, rótulos e funções
- c. Uso de variáveis naturais como rotúlos e palavras chave.

Atributos presentes são assinalados por bits ligados e ausentes por bits desligados constantes no descritor de atributos associado com o objeto. O descritor também relaciona um bloco, isto é, tem um apontador para um bloco, que contém informações adicionais necessárias ao tratamento dos atributos.

Os bits são distribuídos adequadamente de modo a permitir a melhor manipulação interna. Caso um objeto que possa ter atributos for manipulado, os bits específicos associados com a operação em desenvolvimento serão testados no momento oportuno, e se nenhum estiver ligado o processamento prossegue. No caso de algum atributo estar presente, normalmente deve-se ainda testar o estado de palavras chave que indicam se a propriedade do atributo está ou não autorizada (*) no sistema. Caso negativo o

(*) Em inglês: "enabled"

processamento prossegue, senão as rotinas adequadas serão chamadas para produzir o efeito do atributo.

Descritores de atributos aparecem em quatro situações distintas nas estruturas do sistema:

1. Na tabela de funções, relacionando rastreamento de funções por chamada e retorno.
2. Nas estruturas de variáveis naturais, relacionando associação com arquivos de entrada e saída, rastreamento por mudança de valor, uso de variáveis naturais como rótulos e palavras chave.
3. Na tabela de rótulos, relacionando rastreamento de rotúlos.
4. Nas estruturas de variáveis não naturais, relacionando associação com arquivos de entrada e saída e rastreamento por mudança de valor.

Em todas as situações, o tratamento das propriedades é análogo; sempre é alocado um bloco com tantas entradas quantos forem os atributos presentes. A obtenção da entrada no bloco exige contagem de bits ligados no descritor de atributos, processo pouco eficiente, porém deve-se notar que atributos em geral não estarão presentes e o número de bits a percorrer não ultrapassa cinco.

É necessário também um breve comentário a respeito das informações adicionais presentes no bloco de atributos para cada tipo básico de propriedade que pode ser associada:

- a. A associação com entrada e saída é feita pelas funções primitivas INPUT e OUTPUT. A associação com entrada exige dois descritores: o número do arquivo e o comprimento do registro. A associação com saída exige também dois descritores: o número do arquivo e a cadeia que representa o formato.
- b. A associação com rastreamento é feita com o uso da

primitiva TRACE. Cada tipo de rastreamento exige um descritor da cadeia que é a marca(*) associada com o objeto a ser rastreado; será a cadeia nula se não houver marca associada. Exige ainda outro descritor, que especifique o endereço na tabela de funções, da função do usuário que fará o rastreamento ou o endereço da função padrão para tal.

- c. A utilização de variáveis como rótulo exige um descritor que especifica a entrada na tabela de rótulos. A utilização como palavra chave exige um descritor que especifica a tabela e a entrada na tabela para a palavra chave.

O sistema se encarrega de detectar a presença das propriedades nos instantes oportunos e efetuar todo o processamento necessário. O caso do rastreamento é o menos óbvio e exige, especificamente no caso de funções de rastreamento do usuário, que o empilhamento dos argumentos e desvio para a função seja feito exatamente de acordo com a sistemática para funções do usuário normais. Além disso, ao retornar de uma função, o sistema será capaz de retornar para pontos diversos do interpretador conforme seja execução normal de chamadas ou chamadas de rastreamento.

2.4 Código Normal

O código de máquina gerado é armazenado em objetos tipo CODE manipuláveis ao nível de linguagem fonte. O código assim gerado é executado em estado dito normal; há outro tipo de código de máquina utilizado no casamento de padrões e que será visto em 2.5.

O código é um articular igo prefixo. Distin

(*) Em inglês: "tag"

que-se do código prefixo gerado nas versões SIL pelo fato da avaliação dos argumentos de funções se processar na interpretação sequencial do código e não nas próprias funções. Em SIL, a execução de uma instrução de chamada de função desvia para a função especificada, seja ela primitiva ou a intrínseca encarregada de chamadas de funções do usuário. Esta recebe como parâmetro o endereço do código prefixo correspondente a avaliação dos argumentos, avalia recursivamente os argumentos, e executa a função indicada. No presente sistema, o mesmo efeito final é obtido através do empilhamento dos descritores da função e correspondentes argumentos já avaliados, e execução de uma instrução final que aplica a função aos argumentos (aval).

A execução do código normal exige uma pilha de recursão denotada pilha, e cujo apontador de topo chamaremos tp.

2.4.1 Formato de Objetos Tipo CODE

Objetos tipo CODE são blocos de descritores de código. O bloco é formado por um descritor cabeçalho que especifica:

- o número de descritores que compõem o bloco
- o código que identifica o tipo de bloco.

Cada descritor de código por sua vez é composto de dois campos:

op	arg
----	-----

código da instrução argumento

Diversas instruções tem por argumento um descritor de endereço que indica a posição relativa de um objeto na memória. Há endereços associados com variáveis naturais, palavras chave, funções e rótulos, sempre especificados em relação às bases das respectivas tabelas. Há também endereços associados com variáveis criadas.

Algumas instruções não necessitam argumento e a execução do código é sequencial, menos dos desvios. Um exemplo po

	15
inic	1, * + 15
cfun	F
carg	X
aval	1
tend	
cfun	*
carg	Y
cfun	-
cfun	SIZE
carg	W
aval	1
aval	1
aval	2
atrib	
term	1

Figura 2.9

Código gerado para o comando fonte: $F(X) = Y * -SIZE(W)$

de ser visto na figura 2.9; neste exemplo e em outros que se-
guem, a bem da simplicidade e clareza, as instruções e argumen-
tos estão representados de forma mnemônica.

O repertório de instruções está relacionado em seguida.
O tratamento de rótulos e desvios, os operadores expressão não
avaliada e negação, completam o funcionamento do interpretador
de código objeto.

carg k

(carregue argumento) : Carrega no topo da pilha do sistema o a-
pontador k para a tabela de variáveis na-
turais. Em geral não podemos decidir em
tempo de compilação se é o valor ou o endereço da variável natu-
ral que será necessário em tempo de execução. Assim, sempre é
carregado o endereço e as funções que utilizarem o argumento se
encarregam de sua coerção para valor, se necessário. Nesta coer-
ção é testada a associação com arquivo de entrada, e se necessá-
rio, um novo registro será lido; em caso de fim de arquivo esta
instrução falha.

ccons d

(carregue constante) : Carrega no topo da pilha do sistema o descritor d de uma constante que comparece no comando fonte.

cfun k

(carregue função) : Esta instrução verifica o tipo de função relacionada na entrada k da tabela de funções. Se a função especificada for do usuário ou não definida neste instante, então o próprio k é carregado no topo da pilha do sistema. Se a função especificada for primitiva então o descritor de seu bloco de definição será carregado.

Esta instrução exige um acesso a tabela de funções durante sua execução. Isto decorre da interpretação dada pela linguagem à chamada de funções (e da conseqüente ambigüidade apontada em 2.2.2). Funções primitivas avaliam seus próprios argumentos não permitindo a redefinição da própria função durante a computação dos argumentos. Funções do usuário, por outro lado, podem ser definidas ou redefinidas durante a avaliação dos argumentos.

aval n

(avale função) : Avalia funções do usuário ou primitivas. A entrada na tabela de funções ou o descritor de definição estará em pilha(tp-n) e os argumentos em pilha(tp-n+1) até pilha(tp).

No caso de função do usuário, a partir da entrada na tabela de funções é obtido o descritor de seu bloco de definição, que indica também o número de argumentos necessários. A avaliação se processa nos seguintes passos:

- a. Se o número de argumentos necessários for maior que n, argumentos extras com valor de cadeia nula serão empilhados; se for menor que n o topo da pilha será ajustado de modo a ignorar os argumentos extras.

- b. Aval obtém o bloco de definição da função.
- c. Através do bloco de definição são obtidos os endereços dos parâmetros formais na tabela de variáveis naturais. Os valores atuais destas variáveis são trocados com os valores empilhados. Deste modo os valores atuais serão preservados durante a execução da função.
- d. Através do bloco de definição é obtido o endereço da variável com o nome da função. O valor da variável é empilhado, e a variável atribuído a cadeia nula.
- e. Através do bloco de definição são obtidos os endereços das variáveis locais. Seus valores relacionados na tabela de variáveis naturais serão empilhados e cadeias nulas tomarão seus lugares.
- f. O estado do sistema é empilhado, basicamente constando de :
 - apontador da instrução de máquina corrente (i)
 - registrador de falha
 - descritor do bloco de definição
- g. Através do bloco de definição é obtido o endereço na tabela de rótulos do ponto de entrada da função. A entrada nesta tabela relaciona um descritor de endereço de desvio que fornece um novo valor para o apontador de instrução de máquina corrente (i). Se o rótulo de desvio não tiver sido definido ocorre um erro de execução.
- h. O sistema prossegue a execução a partir da nova instrução apontada por i até ocorrer um desvio para um rótulo reservado: RETURN, FRETURN ou NRETURN. Isto assinala o fim da execução da função e corresponde a um retorno para a rotina aval, que retomará o processamento.
- i. O apontador de instrução da máquina corrente é restaurado, o

bloco de definição é recuperado e o valor da variável com nome da função é temporariamente salvo.

- j. Os valores das variáveis: argumentos formais, argumentos locais e nome da função são restaurados a partir de seus valores previamente empilhados.
- k. O topo da pilha é ajustado.

Se o valor salvo no passo j for de tipo STRING, INTEGER ou REAL e o retorno foi via RETURN então este valor é empilhado; se foi via NRETURN o endereço da variável natural com este nome é empilhado.

Se o valor salvo no passo j for de tipo NAME e o retorno foi via RETURN então este valor é empilhado; se foi via NRETURN o endereço da variável (criada ou natural) referenciada pelo descritor (NAME) é empilhada.

Se o valor salvo no passo j é de tipo PATTERN, ARRAY, TABLE, EXPRESSION, CODE ou tipo do usuário e o retorno foi via RETURN então o valor é empilhado; se foi via NRETURN ocorre um erro de execução, porque objetos deste tipo não podem ser manipulados como variáveis em SNOBOL.

- l. Em caso de RETURN ou NRETURN o processamento prossegue com a instrução seguinte. Em caso de FRETURN o registrador de falha indica a atitude a tomar. Isto será visto em 2.4.2 e 2.4.3.

No caso de função primitiva, a partir da entrada na tabela de funções é obtido o descritor de seu bloco de definição, que indica o número de argumentos necessários. A avaliação se processa nos seguintes passos:

- a - Se o número de argumentos necessários não for igual ao número de argumentos empilhados ocorre um erro de execução.

- b - O endereço no microcódigo da rotina que avalia a função é obtido, e o controle passa para a rotina. O estado da computação não é empilhado, uma vez que é suposto que as rotinas estão completamente em microcódigo, e não alteram o estado da computação.
- c - A rotina se encarrega das coerções necessárias dos argumentos, computa o resultado, e retorna o controle e resultado para aval. Esta se encarrega dos ajustes da pilha e empilhamento do resultado.
- d - Aval prossegue o processamento em função do tipo de retorno feito em c.

A rotina aval atualiza a palavra chave &FNCLEVEL. Também se encarrega de efetuar o processamento necessário para eventuais rastreamentos de funções (por chamada e/ou retorno). Os dados necessários ao rastreamento constam no bloco de definição da função, no bloco de atributos da função e em palavras chave.

atrib

(atribua) : Atribui o valor em pilha(tp), após coerção, ao endereço em pilha(tp-1). Se o endereço referir uma palavra chave, um teste adicional para verificar se é protegida ou não, será necessário. Esta instrução também testa se a variável (criada ou natural), especificada do lado esquerdo da atribuição, tem ou não algum atributo ativo. Em caso de associação com arquivo de saída ou rastreamento por mudança de valor, as rotinas adequadas são chamadas.

tend

(teste se endereço) : Testa se no topo da pilha há um endereço de variável (natural ou criada). Se

não houver, um erro de execução ocorre. Esta instrução de teste é necessária para comandos SNOBOL do tipo $\text{exp}_1 = \text{exp}_2$, com o objetivo de fornecer um erro de execução imediatamente após o cálculo de exp_1 , se o objeto retornado não for um endereço.

A inserção desta instrução no código objeto fica a critério do compilador, já que é desnecessária se exp_1 for uma variável natural.

busca

(busque padrão) : Efetua a busca de padrões em comandos SNOBOL do tipo $\text{exp}_1 \text{ exp}_2$. No topo da pilha estarão o descritor da cadeia fonte e o descritor do padrão a procurar. A rotina retorna falha ou sucesso do casamento do padrão. A busca de padrões está detalhada em 2.5.

buscb

(busque padrão e retorne subcadeias não casadas) : Instrução análoga a busca, mas também retorna as subcadeias inicial e final da cadeia fonte, que não casaram. As subcadeias são retornadas no topo da pilha, no lugar do descritor do padrão. A rotina pode retornar falha.

subs

(substitua subcadeia) : É a instrução de máquina que efetua o processamento final de substituição de subcadeia em comandos do tipo:

$\text{exp}_1 \text{ exp}_2 = \text{exp}_3$

Comandos de substituição em busca de padrões geram código de máquina correspondente a seqüência simplificada:

avaliação de exp_1

tend

avaliação de exp_2

buscb

avaliação de exp_3

subs

Subs recebe como argumentos o endereço retornado por exp_1 , o descritor da subcadeia inicial que não casou, o descritor da subcadeia final e o resultado da computação de exp_3 . Subs coerce o resultado de exp_3 e concatena a subcadeia inicial, o valor de exp_3 e a subcadeia final, atribuindo o descritor resultante da concatenação ao endereço computado por exp_1 . Subs também se encarrega de testar associações da variável retornada por exp_1 com arquivo de saída ou rastreamento por mudança de valor, e efetuar o processamento necessário.

O comando de substituição pode falhar durante a avaliação de exp_1 , exp_2 ou exp_3 , bem como em decorrência de falha de casamento em buscb.

inic (n,e)

(inicie comando) : Efetua as computações necessárias antes de iniciar a execução de um comando do programa fonte. Incrementa &STCOUNT de um e atribui a &STNO o valor n que especifica o número do comando fonte. Inicializa também o registrador de falha com o endereço e, que indicará o endereço de desvio em caso de falha, e com o valor corrente do apontador tp que servirá para restabelecer o estado original da pilha.

term n

(termine comando) : Faz as computações necessárias para encerrar um comando ao nível do programa fonte; atribui n a &LASTNO e torna indefinido o registrador de falha. Sua função é evitar uso incorreto do registrador de falha se durante a avaliação de um rótulo de desvio ocorrer falha.

desv e

(desvie) : Transfere a execução para o endereço e especificado no argumento. O endereço é relativo a posição da própria instrução desv.

desvr e

(desvie através da : O argumento desta instrução especifica uma entrada na tabela de rótulos .
tabela de rótulos)

Esta entrada relaciona um endereço no código objeto para onde deve ser feito o desvio neste instante. O endereço pode estar ausente; neste caso, ocorre um erro de execução.

Esta instrução é resultado de otimização introduzida em tempo de compilação e constitui o desvio imediato que não necessita a tabela de variáveis naturais.

desvt

(desvie através da : Esta instrução não tem argumento. Ela
tabela de variáveis efetua um desvio utilizando o endereço
naturais) do da variável natural computado no
topo da pilha. A variável referenciada deverá possuir o atributo "uso como rótulo". Através de seu descritor de atributos obtemos o bloco de atributos, que finalmente especificará uma entrada na tabela de rótulos. Nesta entrada o endereço especificado indicará o ponto do código objeto para onde deve ser o desvio.

Caso o valor computado no topo não for um endereço de variável natural, ou se a variável natural não possuir o atributo "uso como rótulo", ou o endereço na tabela de rótulos ainda não estiver presente, ocorrerá um erro de execução.

Esta instrução constitui o desvio não imediato.

desvc

(desvie para objeto : A instrução não tem argumento. Ela faz
tipo CODE) um desvio para o início de um objeto
tipo CODE cujo descritor deverá estar computado no topo da pilha do sistema. Se no topo estiver um endereço ele será coercido e o valor retornado deve ser um descritor de um objeto tipo CODE; caso contrário ocorre um erro de execução.

2.4.2 Tratamento de Rótulos e Desvios

Desvios em geral não podem ser completamente compilados uma vez que os rótulos de desvio podem ser alterados via compilação de código fonte durante a execução de um programa. Além disso, permite-se a computação de rótulos de desvio bem como o desvio para objetos de tipo CODE. A estrutura que suporta estas propriedades está descrita em 2.3.6.

O código gerado para um bloco do tipo

comando : S(expressão₁)F(expressão₂)

será esquematicamente:

O comando inic basicamente carrega um registrador chamado Registrador de Falha com seu argumento. Este é o endereço no código objeto para onde deve ser desviada a computação caso durante a avaliação de alguma função chamada no comando retorne falha. Como o estado original da pilha deve ser restabelecido após o desvio, o valor do apontador de topo também é armazenado no registrador.

O retorno por falha é detectado na rotina aval, que tratará de atualizar o apontador da instrução corrente com o endereço contido no registrador de falha. Em seguida aval restabelece o controle para o interpretador.

Caso expressão₁ ou expressão₂ sejam variáveis naturais então o compilador pode otimizar o código objeto gerando instruções tipo desvr no lugar da sequência de instruções que avaliam

a expressão e desvt.

Caso o desvio de sucesso esteja omitido, o compilador insere uma instrução desv para pular o trecho de código correspondente a avaliação do desvio de falha. Caso o desvio de falha esteja omitido, o compilador não gera as instruções da expressão₂.

Desvios para objetos de tipo CODE são reconhecidos em tempo de compilação e geram uma estrutura de código análoga a a apresentada utilizando instruções desvc.

A avaliação de expressão₁ e/ou expressão₂ pode resultar em retorno por falha de alguma função que participa da expressão. Neste caso a rotina aval, que detecta o fato, deverá estar apta a reconhecer que está em estado de avaliação de um desvio, e enviar uma mensagem de erro de execução. A detecção da situação é feita pelo fato do registrador de falha durante a avaliação de desvios estar indefinido. Tornar indefinido o registrador de falha é a razão das instruções term.

2.4.3 Operador Negação

O operador negação retorna a cadeia nula se seu argumento retorna por falha e retorna falha se seu argumento retorna por valor ou nome. Importante, também, é lembrar que em SNOBOL se o argumento de uma função, distinta do operador negação, falha, então ela também retorna falha imediatamente.

Isto mostra que o sistema deve reconhecer se há operadores negação empilhados esperando avaliação, antes que a rotina aval dê prosseguimento a qualquer retorno por falha. Uma solução é aval verificar a existência de operadores negação na pilha através de uma busca. Contudo é mais adequado manter uma pilha de negações em andamento utilizando o registrador de falha como seu topo. Porém, agora também exigimos que a instrução cfun reconheça se o operador que está sendo carregado é ou não o operador de negação. Uma vez que cfun necessariamente precisa fazer um acesso a tabela de funções para decidir se deve carregar uma

função primitiva ou do usuário, bastará mais um teste para verificar se se trata do operador negação.

Tomemos por exemplo a expressão:

$$\neg(\neg F(G(X), H(Y)) + B)$$

que corresponde a seguinte representação na nossa notação prefixa:

$\neg \neg + \neg F G X \text{ aval}_1 H Y \text{ aval}_1 \text{ aval}_2 \text{ aval}_1 B \text{ aval}_2 \text{ aval}_1 \text{ aval}_1$.

Supondo que G ao ser avaliada falhe, então neste instante o apontador de instrução corrente (i) estará apontando para aval_1 que avalia G. O conteúdo simplificado da pilha do sistema após a falha e retorno para aval_1 é visto na figura 2.10.

Figura 2.10 - Operador negação no comando

$$\neg(\neg F(G(X), H(Y)) + B)$$

Desse modo, o registrador de falha tem função dupla de apontador para o último operador negação empilhado, e endereço de desvio em caso de falha. Um bit adicional caracteriza sua função.

Agora aval ao receber um retorno por falha verifica o tipo de conteúdo no registrador de falha. Se for endereço de falha então efetua o desvio. Se for apontador para a pilha significa que o código objeto deve ser pulado até o aval correspondente ao operador negação que foi detectado. A operação de pular código é feita casando descritores de funções empilhados e instruções cfun no código objeto com seus respectivos aval's até atingir o novo topo da pilha determinado pelo registrador de falha.

A execução da rotina correspondente ao operador negação deve restaurar o registrador de falha, ajustar o topo da pilha e retornar a cadeia nula por valor.

Observe-se que o esquema permite operadores de negação em expressões que computem rótulos de desvios. Ele complementa o esquema proposto em 2.4.2 para detectar retorno por falha em computação de rótulos.

2.4.4 Operador Expressão não Avaliada

Este operador ao ser executado retorna um descritor do tipo EXPRESSION que aponta para o endereço da instrução corrente no código objeto. Tomemos por exemplo a expressão:

$F(*G(X,Y),W)$

que corresponde em notação prefixa a

$F * G X Y \text{ aval}_2 \text{ aval}_1 W \text{ aval}_2$.

Quando o código de máquina correspondente for executado, e a instrução de empilhamento do operador $*$ for atingida, o operador expressão não avaliada deve ser reconhecido.

Novamente o reconhecimento compete a instrução cfun que nesta hora deverá montar no topo da pilha um descritor tipo EXPRESSION que aponte para a própria instrução cfun $*$ no código objeto. Em seguida cfun pula o código de máquina até detectar a instrução aval correspondente ao cfun. Neste instante o operador $*$

pode ser avaliado, o resultado desta avaliação é o próprio descritor que já estava montado na pilha.

2.4.5 Avaliação de Funções Primitivas Especiais

Algumas funções primitivas podem executar código objeto durante sua avaliação. É o caso de EVAL e APPLY.

Quando EVAL avalia uma expressão, ela recebe como argumento um descritor que aponta para o trecho de código que calcula a expressão. O apontador sempre indicará uma instrução cfun * e digamos que seu valor seja k. Os passos que EVAL deve seguir serão:

- a. armazenar o estado da computação na pilha
- b. carregar na pilha um descritor de bloco de definição de uma função interna denotada **
- c. atribuir o valor k+1 ao apontador de instrução corrente e passar o controle ao interpretador
- d. a computação prossegue até atingir a instrução avalcorrespondente a cfun * (na pilha seria **). A função ** retorna o controle para EVAL.

Enquanto a expressão argumento de EVAL está sendo avaliada, pode ocorrer algum retorno por falha. Neste caso, se não houver negações empilhadas aval se encarrega de desviar para o fim do comando. Se houver negações empilhadas, o casamento referido em 2.4.3 deve testar se o operador na pilha é ou não a função interna **. Caso afirmativo, o controle deve passar para EVAL que retornará falha explícita afim de restaurar o apontador de instrução corrente.

EVAL também avalia expressões na forma de cadeias. Neste caso ela chama o compilador que converte a expressão para código de máquina e em seguida executa o código da maneira descrita anteriormente.

APPLY também tem tratamento especial. Ela efetua a coerção de seu primeiro argumento para uma função primitiva ou do usuário. Em seguida transporta todos os seus argumentos uma posição para baixo na pilha do sistema (o que constrói uma chamada da função argumento de APPLY), ajusta o topo e desvia para a rotina aval. Esta aplicará a função indicada aos argumentos empilhados.

2.4.6 Otimização do Código

De um modo geral, vários pontos do código objeto podem ser otimizados durante a compilação. Vale destacar entretanto, que o processo básico de interpretação já visa uma alta efetividade na computação; assim assinalamos:

- a. Chamadas de funções primitivas não empilham o estado da computação já que normalmente não o alteram.
- b. Desvios imediatos são detectados e gerados de acordo.
- c. Processos de busca só ocorrem em situações especiais e em geral incomuns nos programas SNOBOL.

Contudo, há alguns pontos que poderiam ainda ser considerados durante a geração de código:

- a. Reconhecer sempre que possível desvios para RETURN, NRETURN e FRETURN gerando desvios internos, ao nível de microcódigo.
- b. Programas que não redefinem operadores e funções primitivas poderiam evitar os acessos a tabela de funções em instruções cfun. Bastaria ser uma opção de execução; mas escaparia à filosofia da linguagem SNOBOL.

Observe-se que na versão SIL é necessário efetuar bus-

cas na tabela de variáveis naturais todas as vezes que uma cadeia for gerada em tempo de execução. Na implementação proposta isto não ocorre, contudo, não se evitam cadeias iguais armazenadas na memória.

2.5 Código para Padrões

Quando uma instrução de busca de padrões é executada, o controle passa para uma rotina que recebe como parâmetros a cadeia fonte e o padrão que nela deve ser pesquisado. Como resultado, a rotina de busca de padrões pode retornar uma condição de falha indicando a inexistência do padrão na cadeia fonte, nas condições da pesquisa. Pode retornar sucesso indicando que o padrão foi encontrado, neste caso é retornado também o descritor da subcadeia que conferiu com o padrão.

O casamento de padrões pode ser imaginado como um estado especial do sistema no qual código de máquina de características distintas do normal é executado. Por outro lado, durante o estado de casamento de padrões pode haver necessidade de executar código de máquina normal. Nesta situação o estado da computação deve ser guardado e o estado normal reativado. Nota-se que os dois estados são mutuamente recursivos.

Um elemento fundamental no casamento de padrões é o cursor. Ele é um apontador para algum caráter da cadeia fonte e pode assumir valores de zero até n sendo n o comprimento total da cadeia. Supondo que seu valor seja k , o caráter referenciado será o $(k+1)$ -ésimo; $k=n$ indica fim de cadeia fonte. As posições do cursor antes e após um casamento de padrão com sucesso delimitam a subcadeia que conferiu com o padrão, e são ditas posições pré-cursor e pós-cursor.

2.5.1 Formato de Objetos Tipo PATTERN

Objetos tipo PATTERN são blocos de descritores de código. O bloco é formado por um descritor cabeçalho que especifica:

- o código que identifica o tipo de bloco
- o número de descritores que compõem o bloco (n)
- o comprimento mínimo do padrão (c).

Cada bloco deste tipo é finalizado com um descritor de instrução retn cujo significado é visto adiante; desta forma re presentamos um bloco deste tipo esquematicamente como:

Cada descritor de código por sua vez é composto de três campos:

Algumas instruções não necessitam a especificação de ar gumento e/ou resíduo. O repertório de instruções para casamento de padrões compõe-se basicamente de:

- Padrões primitivos
- Instrução alt
- Instrução noalt
- Instrução desv
- Instrução retn

a. Padrão primitivo

Padrões primitivos formam um conjunto de instruções com características homogêneas, e correspondem aos padrões e funções primitivas da linguagem SNOBOL: literal, LEN, TAB, RTAB, POS, RPOS, REM, ANY, NOTANY, SPAN, BREAK, @, ABORT e FAIL. Os

padrões e funções primitivas restantes: * (expressão não avaliada), \$(atribuição imediata), . (atribuição postergada), ARB, ARBNO, BAL, FENCE e SUCCEED são tratados como padrões compostos. Padrões compostos geram seqüências de instruções de máquina.

A execução de um padrão primitivo é efetuada por rotinas internas descritas com detalhe no capítulo três. Por exemplo, lit é um padrão primitivo que tem por argumento o descritor de uma cadeia que representa o literal a casar; nos exemplos que seguem este descritor está substituído pela própria cadeia a ca sar.

Um padrão primitivo pode ter três tipos de retorno da rotina que o executa:

- i. sucesso
 - ii. falha de casamento
 - iii. falha de comprimento
- i. Retorno por sucesso indica sucesso no casamento do padrão e a execução do código prossegue normalmente.
 - ii. Retorno por falha de casamento indica que o último endereço alternativo especificado deve ser usado para prosseguir a execução. Estes endereços são armazenados numa pilha através da instrução alt junto com a posição corrente do cursor. A execução prossegue a partir do endereço alternativo com reposicionamento do cursor.
 - iii. Retorno por falha de comprimento é uma situação especial ativada somente em modo QUICKSCAN que indica que não há caracteres suficientes na cadeia fonte para casar o padrão e seu subsequente. Neste caso também a alternativa é tentada.

Se eventualmente a pilha de alternativas tiver sido esvaziada, significa que não há alternativa a tentar, e indica que o casamento falhou. Dependendo do modo (QUICKSCAN ou FULLSCAN) e do contexto no qual as alternativas se esgotaram, pode signifi-

car falha da própria instrução de busca de padrões.

b. alt (carregue endereço de alternativa)

Carrega na pilha de alternativas (denominada hist) o endereço especificado no argumento e o valor do cursor neste instante. Os pares (endereço, cursor) em hist indicam as alternativas que devem ser tentadas se necessário.

O argumento da instrução alt é um endereço relativo a própria instrução alt no bloco. O argumento de alt é indicado nos exemplos por * $\pm$ i sendo i o endereço relativo. Na pilha hist são armazenados os endereços convertidos absolutos.

c. noalt (ignore esta instrução)

Instruções noalt tem o objetivo de simplificar a construção de padrões. Elas indicam fim de uma cadeia de alternativas. A execução de uma instrução noalt não causa nenhum efeito além do de avançar o apontador de instrução corrente.

d. desv (desviar para endereço)

Transfere a execução para o endereço especificado no argumento. O endereço especificado é relativo a própria instrução desv. Ela difere apenas no formato em relação a correspondente para código normal.

e. retn (fim de execução de padrão)

Assinala sucesso no casamento de um padrão. A execução de uma instrução retn ao nível do sistema pode significar sucesso da própria instrução de busca de padrão. Neste caso o sistema deve ainda:

- efetuar atribuições postergadas
- montar descritores das sucadeias que não casaram

- restaurar o estado da computação

A instrução não possui argumento.

Devido a características das instruções alt e noalt, elas são tratadas pelo sistema como padrões primitivos. Nos exemplos que seguem sempre serão empregados padrões primitivos de significado óbvio. Os exemplos também esclarecerão a notação empregada.

A figura 2.11 mostra uma seqüência de código para um padrão, o campo de resíduo de cada descritor também está indicado mas seu significado só será visto adiante.

Figura 2.11
Código gerado para o padrão
LEN(3) ("X" | "YYYY") LEN(2)

8		6
len	3	3
alt	*+3	3
lit	"X"	2
desv	*+3	
noalt		6
lit	"YYYY"	2
len	2	0
refn		

2.5.2 Construção de Padrões

Os processos de construção e casamento de padrões são distintos; sendo a construção de padrões realizada executando código normal do tipo visto em 2.4. Ao nível da linguagem podem participar da construção de objetos tipo PATTERN:

- padrões primitivos associados com variáveis: FAIL, FENCE, ABORT, ARB, BAL, SUCCEED, REM
- funções primitivas que retornam padrões: LEN, SPAN, BREAK, ANY, NOTANY, TAB, RTAB, POS, RPOS, ARBNO

- c. operadores de atribuição durante a fase de casamento: @, . e \$
- d. expressões não avaliadas indicadas pelo operador unário *
- e. funções do usuário que retornem padrões
- f. variáveis cujo valor são padrões construídos pelo usuário ou cujo valor seja coercível para padrão
- g. operadores de construção de padrões: concatenação e alternância.

No sistema proposto:

- a. Padrões primitivos associados com variáveis, são blocos pré-construídos com o código necessário ao padrão e constituem o valor das variáveis.
- b. Funções primitivas que retornam padrões, operadores de atribuição e o operador expressão não avaliada, constroem a partir de seus argumentos um bloco com o código necessário; este bloco é o resultado da chamada da função ou operador.
- c. Funções do usuário que retornam padrões ou variáveis com valor tipo padrão retornam blocos de código tipo PATTERN.
- d. Os operadores de construção de padrões produzem blocos de código para padrões a partir de seus operandos. Os operandos, após coerção, devem ser também blocos de código para padrões. O código resultante das duas operações de construção é explicado em seguida.

Por conveniência, todo bloco de código para padrão tem uma única instrução `retn` que finaliza a sequência de instruções. Isto em geral exige comandos `desv` adicionais, mas facilita a concatenação de padrões.

Basicamente, a concatenação aloca um bloco suficiente para conter a concatenação dos blocos operandos. A instrução `retn` do primeiro operando não é copiada passando os desvios para `retn` a se referir a primeira instrução do bloco seguinte. A concatenação é vista na figura 2.12.

Figura 2.12 - Concatenação de padrões

A alternância de dois padrões é vista na figura 2.13, e consta da alocação de um bloco maior e cópia dos operandos com os ajustes necessários. É feita uma busca na cadeia de alternativas iniciada pela primeira instrução do primeiro operando; se for um `alt` a cadeia é seguida até encontrar um `noalt`, e este é substituído por um `alt` com argumento apontando para o início do segundo operando.

O início do segundo operando pode ou não, ser um `alt`. Caso não seja, um `noalt` é inserido neste ponto. Também, se o início do primeiro operando não for um `alt`, então é inserido um que aponta para o início do segundo operando. Finalmente, a instrução `retn` do primeiro operando é substituída por um `desv` para a instrução `retn` do segundo operando.

As instruções `noalt` indicam lugares prováveis no código para extensão de uma cadeia de alternativas. A busca na cadeia está de acordo com a definição de alternância utilizada em SIL e preserva os mesmos resultados. Uma definição mais simples é usada em (Shapiro 1972), mas não fornece os mesmos resultados para algumas construções em modo QUICKSCAN.

Figura 2.13 - Alternação de padrões

2.5.3 Casamento de Padrões em Modo QUICKSCAN

O casamento do padrão LIT é feito caráter a caráter entre a cadeia fonte e o argumento da instrução. As comparações entre caracteres são efetuadas até que:

- Os caracteres do argumento são esgotados, indicando sucesso.
- A comparação acusa caracteres diferentes, indicando falha.
- A cadeia fonte é esgotada, mas ainda permanecem caracteres no argumento; esta situação também indica falha.

A situação c pode ser percebida antes da comparação dos caracteres, comparando o comprimento restante na cadeia fonte com o comprimento total do argumento. Esta sistemática é trivial quando o argumento é uma cadeia, e também pode ser tentada com outros padrões primitivos. O teste é incorporado, se adequado,

nas rotinas para instruções tipo padrão primitivo.

Outra melhoria do processo de casamento pode ser obtida comparando mais de um caráter, entre a cadeia fonte e o argumento da instrução, a cada passo; até ocorrer a, b ou c.

As duas melhorias podem ser incorporadas ao microcódigo das instruções sem ônus para a semântica da linguagem. Uma otimização mais geral é obtida no modo QUICKSCAN no qual se obtém efeitos por vezes distintos do modo FULLSCAN. Uma discussão completa destes efeitos pode ser obtida em (Griswold, Poage e Polonsky 1971).

O modo QUICKSCAN estende a primeira melhoria para padrões mais complexos, adicionando em cada instrução tipo padrão primitivo o resíduo deste padrão. Definimos resíduo de um padrão primitivo, como o número mínimo de caracteres da cadeia fonte necessários para casar o subsequente deste padrão. O resíduo do padrão primitivo, mais o número de caracteres necessários para casar o próprio padrão, determinam um comprimento total mínimo da cadeia fonte necessário para prosseguir o casamento com pretensões de sucesso.

No modo QUICKSCAN, a rotina do padrão primitivo se encarrega de testar se há um número suficiente de caracteres, a partir da posição corrente do cursor, para casar o padrão e seu subsequente. Em caso afirmativo o casamento do padrão primitivo é tentado podendo resultar em sucesso ou falha de casamento; em caso negativo ocorre falha de comprimento.

Assim, no padrão:

LEN(5) (LEN(4) "ABC" | "XY" LEN(1)) ("R" | "NM") o resíduo de LEN(5) vale 4, o resíduo de LEN(4) vale 4, o resíduo de "ABC" vale 1, o resíduo de "XY" vale 2 e o resíduo de LEN(1) vale 1. É desnecessário tentar casar o padrão LEN(5) se na cadeia fonte restarem menos de 9 caracteres, ou tentar casar "ABC" se na cadeia fonte restarem menos de 4 caracteres. No mesmo padrão, a primeira alternativa terá um resíduo igual a 4, indicando que nenhuma das alternativas precisa ser tentada se houver menos de 4 caracteres restando na cadeia fonte. O código correspondente

Figura 2.14
Código gerado para o padrão
LEN(5) (LEN(4) "ABC" | "XY"
LEN(1)) ("R" | "NM")

.14		9
len	5	4
alt	* + 4	4
len	4	4
lit	"ABC"	1
desv	* + 4	
noalt		4
lit	"XY"	2
len	1	1
alt	* + 3	1
lit	"R"	0
desv	* + 3	
noalt		2
lit	"NM"	0
retn		

ao padrão é visto na figura 2.14.

No exemplo da figura 2.14, o comprimento mínimo do padrão vale 9. Isto significa, que para casar com sucesso o padrão, serão necessários no mínimo 9 caracteres. Este valor é utilizado na construção de padrões para ajustar o campo de resíduo de padrões primitivos.

O conceito de falha por comprimento conduz a definição de "futilidade de padrões". Um padrão é dito fútil, para determinada cadeia fonte e posição de cursor, se ele falha para todos os avanços do cursor.

Assim, se o sistema reconhecer que um padrão é fútil numa certa posição do cursor então ele não precisa avançar o cursor. Note-se que esta sistemática considera que o programador não está interessado nos efeitos laterais do casamento do padrão decorrentes do avanço do cursor. De qualquer forma, o teste de futilidade só é efetuado em modo QUICKSCAN.

A maioria dos padrões primitivos transmite futilidade detectada por um subsequente. Dizemos que o padrão primitivo P é transmissor se todas as vezes que P casa com sucesso e seu subsequente Q é fútil, PQ também for fútil.

Em SNOBOL, todos os padrões primitivos da linguagem são transmissores a menos de BAL. Será suposto que todos os primitivos são transmissores e as exceções devem ser tratadas especialmente. Assim, se um padrão falha numa posição qualquer do cursor e todos seus padrões primitivos que falharam, falharam por comprimento, então o padrão é fútil e não há necessidade de avançar o cursor para fazer nova tentativa. Contudo, se algum primitivo forneceu falha de casamento, então não se pode afirmar que o avanço do cursor só resultará em novas falhas.

Ao nível de rotina de casamento, a futilidade é detectada através de uma variável denominada futil, inicialmente valendo "sim", indicando que o padrão em princípio é fútil. Se durante o processo ocorre alguma falha de casamento, automaticamente o padrão se torna não fútil, e o fato é registrado atribuindo "não" a futil. Antes de avançar o cursor e tentar novo casamento do padrão a variável é testada.

Com o tratamento de futilidade, a construção de padrões passa a exigir cuidado especial:

- a. Na concatenação do padrão A com o padrão B, o comprimento mínimo do padrão B é adicionado ao comprimento mínimo e aos resíduos de todas as instruções do padrão A. Porque, A terá um subsequente e em princípio não há instrução `retn` no meio do código de A, a menos da instrução final. O único problema é o padrão `ABORT` que termina prematuramente o casamento; neste caso um processamento especial após a montagem é suficiente para efetuar os ajustes nos resíduos afetados. Veja figura 2.15.
- b. Na alternância do padrão A com o padrão B toda a cadeia de alternativas terá seu resíduo alterado para o mínimo entre o resíduo corrente e o comprimento mínimo do padrão B. O mes

A =

	6	5
alt	* + 3	5
len	5	0
desv	* + 3	
noalt		6
len	6	0
retn		

A B =

	7	8
alt	* + 3	8
len	5	3
desv	* + 3	
noalt		9
len	6	3
len	3	0
retn		

B =

	2	3
len	3	0
retn		

A = len (5) | len (6)
B = len (3)

Figura 2.15 - Padrão resultante da concatenação de A e B

A =

	7	4
alt	* + 3	4
len	5	0
desv	* + 4	
noalt		4
len	3	1
len	1	0
retn		

B =

	2	2
len	2	0
retn		

A | B =

	10	2
alt	* + 3	2
len	5	0
desv	* + 4	
alt	* + 4	2
len	3	1
len	1	0
desv	* + 3	
noalt		2
len	2	0
retn		

A = len (5) | len (3) | len (1)
B = len (2)

Figura 2.16 - Padrão resultante da alternância de A e B

mo ocorre com o comprimento mínimo do padrão A. Veja a figura 2.16 para maiores detalhes.

2.5.4 Execução do Código de Casamento de Padrões

A rotina que executa a busca de um padrão numa cadeia fonte recebe como argumentos estes dois objetos, e deve fornecer como resultado a condição de sucesso ou falha do casamento. Em caso de sucesso, deverão ser efetuadas as atribuições postergadas bem como devem ser montados os descritores das subcadeias não casadas. Basicamente a rotina segue os passos:

- a. Armazenar estado do sistema e base do bloco de alternativas.
- b. Entrar em estado de casamento de padrões.
- c. Posicionar cursor e marcar pilha de alternativas.
- d. Armazenar estado do casamento de padrões.
- e. Chamar rotina de casamento que executa seqüencialmente o código do padrão e retorna com falha ou sucesso.

Em caso de sucesso, a volta ao estado normal pode acontecer. Em caso de falha, testes de &ANCHOR, &FULLSCAN e da variável futil determinam se um avanço do cursor e nova tentativa de casamento devem ser efetuados. Conforme o caso, novo casamento é tentado ou é assinalada falha e volta ao estado normal.

As rotinas de casamento são descritas a seguir em um pseudo-Algol e referenciam as variáveis:

cursor : posição corrente do cursor na cadeia fonte (o caráter referenciado será o cursor+1)

pilha : pilha do sistema

hist : pilha de alternativas

tp : topo da pilha do sistema

th : topo da pilha de alternativas

i : apontador de instrução corrente em estado normal ou de casamento de padrões

l : comprimento da cadeia fonte disponível para casamen

to, incluindo o trecho já casado; a variável l tem a função de quebrar malhas à esquerda em padrões com expressões não avaliadas

```

início
  cursor← 0;
  pilha(tp+1)← #;
  pilha(tp+2)← th+1;
  pilha(tp+3)← i;
  pilha(tp+4)← endereço da primeira instrução no padrão;
  pilha(tp+5)← comprimento da cadeia fonte;
  pilha(tp+6)← e;
  tp← tp+6;
  faça início
    futil← "SIM";
    faça início
      i← pilha(tp-2);
      l← pilha(tp-1);
      hist(th+1)← NULL;
      hist(th+2)← cursor;
      th← th+2;
      executar rotina de casamento;
      se retorno por falha
        então se &ANCHOR ≠ 0
          então cont← "FALHA"
          senão início
            cursor← cursor+1;
            se &FULLSCAN = 0
              então se cursor > compr. da cadeia fonte
                então cont← "FALHA"
                senão se futil = "SIM"
                  então cont← "FALHA"
                  senão cont← "*"
              senão se cursor > compr. da cadeia fonte
                então cont← "FALHA"
                senão cont← "***"
            fim
          senão início
            efetuar atribuições postergadas;
            montar descritores de cadeias não casadas;
            cont← "SUCESSO";
            fim;
          fim enquanto cont = "***";
        fim enquanto cont = "*";
      i ← pilha(tp-3);
      th← pilha(tp-4);
      tp← tp-6;
    fim
  fim

```

Figura 2.17 - Rotina da instrução de busca de padrões

e : endereço de retorno no microcódigo
 # : marca para pilha do sistema
 NULL : marca para pilha de alternativas

Na figura 2.17 está representado o detalhamento geral para a rotina de busca do padrão na cadeia fonte. O estado das pilhas durante o casamento é visto na figura 2.18.

Figura 2.18 - Pilhas durante o casamento de padrões

Na rotina de casamento, o código de máquina do padrão, a partir da posição i, é executado. Desta execução resulta o casamento ou não casamento dos padrões primitivos que compõem o código. Os argumentos da rotina são i e l, sendo l utilizado somente com padrões tipo expressão não avaliada. Quem chama a ro

tina deve empilhar i, l e e em pilha, desviar para a rotina e após o retorno desta, desempilhar os valores se necessário. O retorno decorre ou da execução de uma instrução retn (sucesso) ou do esgotamento das alternativas (falha). A rotina está detalhada na figura 2.19.

```

início
  retorno← "A";
  faça se op(i) = "DESV"
    então i← arg(i)
    senão se op(i) é primitiva
      então início
        executar primitiva;
        se retorno por sucesso
          então i← i+1
          senão início
            se retorno por falha de
              casamento
            então futil← "NÃO";
            cursor← hist(th);
            i      ← hist(th-1);
            th     ← th-2;
            se i = NULL
              então retorno← "FALHA";
            fim;
          fim
        senão retorno← "SUCESSO"
      enquanto retorno = "A";
    desviar para pilha(tp);
  fim

```

Figura 2.19 - Rotina de casamento de padrões primitivos

2.5.5 Padrões Compostos e Especiais

Vários padrões não são implementados como padrões primitivos mas como seqüências de instruções envolvendo padrões primitivos. Estes padrões serão descritos através de dois tipos de esquemas: o esquema correspondente a sua implementação em SIL e o esquema correspondente a sua implementação em linguagem da máquina SNOBOL.

Nos esquemas para SIL os padrões primitivos envolvidos estão ligados através de linhas contínuas que indicam concatenação e linhas tracejadas que indicam alternância dos padrões conec

tados. É suposto que o casamento se inicie através do primitivo no alto a esquerda do esquema.

Nos esquemas em código de máquina omitiu-se a especificação do resíduo de cada instrução e o descritor de cabeçalho . Se a especificação do argumento aparece sombreada, indica endereço de subsequente ou de alternativa do padrão.

NULL representa o padrão primitivo que casa zero caracteres e sempre produz sucesso.

a. ARBNO (P)

O esquema do padrão construído pela função primitiva ARBNO é visto na figura 2.20; ele casa zero ou mais ocorrências consecutivas do padrão P. A eficiência do casamento deste pa-

alt	
cfut	
alt	* + 2
desv	
dfut	
P	
desv	* - 5

ARBNO (P) em código

ARBNO (P) em sil

Figura 2.20 - ARBNO(P)

drão é aumentada testando a chave futil afim de detectar a necessidade de continuar a expansão em QUICKSCAN.

Se o subsequente do padrão construído por ARBNO(P) falha

por comprimento, o casamento de ARBNO pode terminar visto que P casa um número não negativo de caracteres.

Assim, devemos obter um padrão composto tal que se o subsequente dele falha por comprimento, ele deve se comportar como se nenhum subsequente estivesse presente e também retornar falha de comprimento. A detecção da falha por comprimento exige empilhamento da chave futil em hist. O empilhamento é necessário visto que a chave pode valer "NAO" quando o padrão composto é atingido, e no entanto estamos interessados no valor "SIM" retornado do casamento apenas do subsequente. Assim, a chave é empilhada e ligada em "SIM". Caso o subsequente falha por casamento, então ela vale "NAO" e o padrão deve ativar a alternativa P. Contudo, se o subsequente falha por comprimento, o padrão composto deve retornar a futil seu valor antigo, pois tudo se passa como se não existisse o subsequente, e em seguida falhar por comprimento. Os padrões primitivos envolvidos estão detalhados na figura 2.21.

<u>cfut (carregue futilidade)</u>	<u>dfut (descarregue futilidade)</u>
<u>início</u>	<u>se</u> futil = "SIM"
hist(th+1)← futil;	<u>então</u> <u>início</u>
th← th+1;	futil← hist(th);
futil← "SIM";	th← th-1;
retornar sucesso;	retornar falha de comprimento;
<u>fim</u>	<u>fim</u>
	<u>senão</u> <u>início</u>
	th← th-1;
	retornar sucesso;
	<u>fim</u>

Figura 2.21 - Padrões primitivos de ARBNO

Na figura, o primitivo cfut trata de empilhar o valor de futil. A instrução alt que vem em seguida empilha a alternativa dfut e a instrução desv finalmente desvia para o subsequente. Se o subsequente falha de alguma forma, a alternativa dfut é ativada e desempilha o valor de futil; em seguida é executado o padrão P. Se P falha, a alternativa do padrão composto (primeiro alt) é ativada; se P tem sucesso desviamos para cfut e o processo recomeça.

O primitivo cfut também se encarrega de fazer o teste entre o comprimento disponível e o resíduo indicado na instrução, retornando eventualmente falha de comprimento.

b. ARB

O padrão para ARB é análogo a ARBNO(LEN(1)) com a mesma sistemática para a chave de futilidade. Eventualmente o padrão pode ser otimizado reunindo dfut e len(1) num único padrão primitivo que opere diretamente sobre as pilhas, evitando reempilhamento de alternativas e da chave futil.

c. BAL

O padrão primitivo BAL, visto na figura 2.22, casa a primeira expressão balanceada com respeito a parênteses a partir da posição atual do cursor. Como é padrão com alternativas implícitas, falhas de subseqüentes farão com que sejam tentadas concatenações de expressões balanceadas. BAL pode falhar se numa expansão não restarem caracteres na cadeia fonte (falha por comprimento) ou porque da posição atual do cursor só foi possível casar uma cadeia vazia (falha de casamento).

BAL em SIL

alt	
cfut	
desv	* + 2
nfut	
gbal	
alt	* - 2

BAL em código

Figura 2.22 - BAL

BAL é um padrão que não transmite futilidade, isto é, se seu subsequente é fútil nada se pode afirmar sobre BAL concatenado com o subsequente. Isto porque o avanço do cursor pode fazer com que BAL case uma cadeia menor do que a anterior, dando chance a seu subsequente casar. Os padrões primitivos envolvidos estão detalhados na figura 2.23.

nfut (não transmite futilidade)

```
se futil = "SIM"
  então retornar sucesso
senão início
  futil ← hist(th);
  th ← th-1;
  retornar falha de comprimento;
fim;
```

gbal (case cadeia balanceada)

início

casar a menor cadeia de caracteres balanceada com respeito a parênteses, a partir da posição atual do cursor;

se esgotou cadeia fonte

então início

futil ← "NÃO";

th ← th-1;

retornar falha de comprimento;

fim

senão se casou zero caracteres

então início

futil ← hist(th);

th ← th-1;

retornar falha de casamento;

fim

senão se casou somente um caráter

então retornar sucesso

senão início

hist(th) ← "NÃO";

th ← th+1;

retornar sucesso;

fim;

fim

Figura 2.23 - Padrões primitivos de GBAL

d. Atribuição postergada

A associação de variáveis com padrões, tipo $P \cdot v$, constrói um padrão com a estrutura da figura 2.24. Neste padrão é necessário guardar a posição do cursor antes do casamento de P e, se o casamento tiver sucesso, guardar as posições antiga e a atual numa lista de atribuições postergadas. A lista de atribuições postergadas é uma nova pilha, com nome `atrib` e `topo ta`, tratada ao final do casamento na forma de lista.

Figura 2.24 - $P \cdot v$

O armazenamento do cursor pelo padrão `ccur` é feito na pilha do sistema e não em `hist`, uma vez que P também pode empilhar alternativas em `hist` e que exigiria uma busca pelo padrão `C.`. O detalhamento dos padrões primitivos envolvidos está na figura 2.25.

O primitivo `ccur` armazena a posição corrente do cursor; mas como P pode falhar, é dada antes uma alternativa (`dcur`) que neste evento simplesmente descarrega o cursor armazenado. Supondo que P tenha sucesso, estamos aptos a guardar na lista de atribuições postergadas o trecho de cadeia que casou e a variável à

ccur (carregue cursor)

início
pilha(tp+1)← cursor;
tp← tp+1;
retornar sucesso;
fim

dcur (descarregue cursor)

início
tp← tp-1;
retornar falha de casamento;
fim

c. (carregue atribuição postergada)

início
atrib(ta+1)← pilha(tp);
atrib(ta+2)← cursor;
atrib(ta+3)← arg(1);
ta← ta+3;
tp← tp-1;
retornar sucesso;
fim

d. (descarregue atribuição postergada)

início
pilha(tp+1)← atrib(ta-2);
tp← tp+1;
ta← ta-3;
retornar falha de comprimento;
fim

Figura 2.25 - Padrões primitivos de P . v

qual este trecho deve ser atribuído; devemos também remover da pilha a posição inicial do cursor. C. efetua estas operações.

Contudo, o subsequente de P pode falhar e nova alternativa, possivelmente do código de P, deve ser tentada. Neste caso, d. entra em ação, descarregando as entradas na pilha de atribuições e restituindo o valor inicial do cursor à pilha do sistema. A restituição do cursor à pilha é necessária, porque se a alternativa de P tiver sucesso, C. será novamente ativado; ou, se P falhar, dcur deverá ter o que descarregar da pilha do sistema.

Quando a instrução de busca de padrões terminar com sucesso, a lista de atribuições postergadas deve ser percorrida num esquema FIFO (da esquerda para a direita em relação a cadeia fonte), efetuando as atribuições postergadas. Caso alguma das variáveis seja uma expressão não avaliada, ela será avaliada neste ponto do processamento. Durante as atribuições deverão ser consideradas associações com arquivo de saída e rastreamento de variáveis.

e. Atribuição imediata

A atribuição imediata, tipo $P \$ v$ é análoga à atribuição postergada (veja figura 2.26). As atribuições são efetuadas imediatamente por $c\$$ cujo argumento deve ser uma variável ou uma expressão não avaliada. Se for uma expressão ela será avaliada neste instante devendo retornar uma variável, denotada por v na figura 2.27 que detalha os novos padrões primitivos envolvidos.

Figura 2.26 - $P \$ v$

c\$ (carregue atribuição imediata)

início

```
atrib(ta+1)← pilha(tp);
ta← ta+1;
tp← tp-1;
v ← descritor da subcadeia;
retornar sucesso;
```

fim

d\$ (descarregue atribuição imediata)

início

```
pilha(tp+1)← atrib(ta);
tp← tp+1;
ta← ta-1;
retornar falha de comprimento;
```

fim

Figura 2.27 - Padrões primitivos de $P \$ v$

f. Posição do cursor

O padrão @v corresponde a uma instrução tipo padrão primitivo com um argumento: @ v (atribua cursor). A instrução monta um descritor inteiro com o valor do cursor e o atribui à variável v. Em seguida retorna sucesso.

g. ABORT, FAIL, FENCE e SUCCEED

ABORT deve provocar a falha total da instrução de busca em andamento. A instrução deve procurar a primeira marca (#) em pilha, obter a entrada em hist, ajustar os topos de hist e pilha, esgotar os caracteres da cadeia fonte e retornar falha de casamento. A rotina de casamento imediatamente retornará por falha à rotina de busca que por sua vez também falhará. Veja a figura 2.28.

FAIL é uma instrução tipo padrão primitivo que simplesmente retorna falha. Veja a figura 2.28.

abort (aborte casamento)

início

k ← endereço da marca #;
th ← pilha(k+1)+1;
tp ← k+5;
cursor ← pilha(k+4);
retornar falha de casamento;

fim

fail (falha)

início

retornar falha de com
primento;

fim

Figura 2.28 - ABORT e FAIL

FENCE casa a cadeia nula durante a computação da esquerda para a direita. Se no entanto alguma alternativa anterior a FENCE for tentada, então o comando todo de busca de padrões falha. Veja a figura 2.29.

SUCCEED corresponde a um padrão com infinitas alternativas impedindo o retorno à sua esquerda. Veja a figura 2.29.

alt	* + 2
desv	
abort	

FENCE em código

alt	* + 0
-----	-------

SUCCEED em código

Figura 2.29 - FENCE e SUCCEED

n. Expressão não avaliada

Há duas situações onde podem ocorrer expressões não avaliadas em padrões:

1. O padrão todo é uma expressão não avaliada ou algum operando de uma concatenação ou alternância é uma expressão não avaliada. Exemplo:

P = *F(X) "ABC"

2. A expressão não avaliada ocorre como argumento de uma função primitiva ou operador primitivo de construção de padrões. Exemplo:

P = ("1" | "2") \$ N LEN(*N)

P = "ABC" \$ *X(I)

Antes de apresentar o esquema de processamento para expressões não avaliadas, deve ficar clara a distinção existente entre a construção e a execução de um padrão com expressões não avaliadas. Seja por exemplo o comando:

P = "A" *("X" | "Y")

que terá esta sequência simplificada de código de máquina:

```
carg P
cfun concat
```

```

carg  "A"
cfun  *
cfun  alter
carg  "X"
carg  "Y"
aval  2
aval  1
aval  2
atrib

```

Supondo que esta seqüência seja executada, e que * se refira ao operador expressão não avaliada, então como resultado será atribuído a P um objeto tipo PATTERN com código:

```

lit  "A"
*comp k
retn

```

onde *comp representa um padrão composto detalhado adiante. Seu argumento k é um apontador para a instrução cfun * no código de construção do padrão.

Supondo que P seja utilizado num comando de casamento de padrões, por exemplo:

```
"ABCD" P : S(XXX) F(YYY)
```

então a instrução de máquina correspondente recebe como argumento a cadeia fonte "ABCD" e o padrão P. A execução do casamento evolui até ser atingida a instrução *comp. Neste instante a execução do padrão composto se estende em duas fases distintas:

- 1ª fase : A expressão apontada por k deve ser avaliada.
- 2ª fase : O objeto retornado da fase anterior é coercido e aplicado a cadeia fonte a partir da posição corrente do cursor.

No caso do exemplo, na 1ª fase retorna um objeto que é

um padrão P' com código:

```
alt    ++3
lit    "X"
desv   ++2
lit    "Y"
retn
```

A 2ª fase constará da aplicação deste padrão à cadeia fonte na posição corrente do cursor. Isto é feito mediante uma chamada recursiva da rotina que casa padrões; após o retorno desta rotina o processamento prossegue com a execução das instruções do padrão P.

Nenhuma restrição existe quanto a expressões não avaliadas referenciarem novas expressões não avaliadas e/ou produzirem execução de comandos de busca de padrões. Expressões não avaliadas como argumento de padrões primitivos são avaliadas pelas próprias rotinas do padrão envolvido.

O padrão composto para *comp pode ser visto na figura 2.30 na qual k passa a ser argumento do padrão primitivo *. O padrão primitivo * se encarrega de avaliar seu argumento e aplicar (no caso geral, recursivamente) o resultado à cadeia fonte na posição corrente do cursor. Esta aplicação exige a marcação da pilha hist, daí a necessidade do outro padrão primitivo re* ativado em caso de falha do subsequente de *comp. O padrão re* se encarrega de verificar se em hist existem ou não alternativas da aplicação recursiva, evitando a utilização da marca feita por *, como alternativa.

Além disso, em modo QUICKSCAN, * e re* devem tentar quebrar malhas recursivas à esquerda mediante comparação dos resíduos dos padrões com o comprimento l disponível na cadeia fonte.

A rotina do padrão primitivo * efetua o casamento de uma expressão tipo PATTERN com a cadeia fonte na posição corrente do cursor; seu detalhamento aparece na figura 2.31. O primitivo * é computado em fases distintas:

Figura 2.30 - *comp k

1. Verificar se há chance do padrão casar em vista do resíduo necessário e do número de caracteres restantes na cadeia fonte.
2. Avaliar o padrão não avaliado retornando um padrão P'.
3. Marcar a pilha de histórico com marca que a rotina de casamento seja capaz de detectar como fim de pilha. Esta marcação é análoga a efetuada na rotina de busca.
4. Executar a rotina de casamento para casar P' a partir da posição corrente do cursor, até um comprimento máximo igual a 1-res(1) (resíduo constante no padrão primitivo *).
5. Retornar o resultado da rotina de casamento.

A fase 4 merece uma discussão adicional:

1. Se a rotina de casamento falhar, é porque ela esvaziou a pilha de alternativas até atingir a marca da fase três. Isto significa que P' não casou.

```

* (case expressão não avaliada)

se &FULLSCAN = 0 e cursor+res(i) > 1
  então retornar falha de comprimento
  senão início
    p' ← eval(arg(i)) com coerções necessárias;
    hist(th+1) ← NULL;
    hist(th+2) ← cursor;
    th ← th+2;
    pilha(tp+1) ← 1;
    pilha(tp+2) ← 1;
    pilha(tp+3) ← e;
    tp ← tp+3;
    l ← l-res(i);
    i ← primeira instrução em p';
    executar rotina de casamento;
    i ← pilha(tp-2);
    l ← pilha(tp-1);
    tp ← tp-3;
    se retorno por falha
      então retornar falha de casamento
      senão retornar sucesso;
  fim

```

Figura 2.31 - Primitiva * de *comp k

- ii. Se a rotina de casamento retornar sucesso significa que * deve retornar sucesso. Contudo, o processamento de P' pode ter empilhado uma série (eventualmente nula) de alternativas em hist. Falhas do subsequente automaticamente ativam estas alternativas correndo-se o risco de usar a marca da fase três como alternativa.

O problema é resolvido inserindo nova alternativa no histórico antes de prosseguir com o subsequente. Esta alternativa aponta para o padrão re* no padrão composto. Se a alternativa vier a ser ativada, ela desempilha de hist a alternativa que estava imediatamente abaixo dela, podendo ser:

- a marca da fase três
- uma alternativa verdadeira, isto é, um par com endereço de padrão primitivo (inclusive pode ser de outro *re) e valor do cursor.

Se for a marca, então a alternativa abaixo da marca deve ser tentada, neste caso `re*` simplesmente retorna falha e o controle da rotina de casamento se encarregará de tentar nova alternativa de `hist`.

Se for alternativa verdadeira, `re*` aplicará o padrão, indicado pelo endereço constante na alternativa, à cadeia fonte na posição corrente do cursor. A aplicação se processa através de chamada recursiva da rotina de casamento. Novamente o comprimento irá até `1-res(1)` (resíduo constante no padrão primitivo `re*`).

Interessante é observar, que na chamada da rotina de casamento por `re*`, não é marcada a pilha de alternativas. Isto é consistente, porque:

- Se a rotina retornar falha, todas as alternativas abaixo da alternativa corrente (referida neste contexto como verdadeira) terão sido tentadas até atingir a marca da fase três. A marca força retorno por falha da rotina de casamento, tal retorno será para `re*`, que agora tem certeza de ter esgotado as alternativas deixadas por `*`.
- Se a rotina retornar sucesso, zero ou mais alternativas verdadeiras terão sido desempilhadas e obteve-se sucesso sem atingir a marca deixada por `*`. Neste caso, restam alternativas em `hist` bem como a própria marca. A primitiva `re*` retorna sucesso, contudo pela figura 2.33 observa-se que seu subsequente volta a empilhar ela como alternativa repetindo o processo.

O aspecto da pilha de alternativas pode ser visto na figura 2.32 numa situação que pode corresponder a um casamento com um padrão `Q` não avaliado que referencia outro padrão não avaliado `P`.

O detalhamento do padrão primitivo `re*` está apresentado na figura 2.33.

Figura 2.32 - Pilha de alternativas com padrões não avaliados

```

re* (recupere-se de expressão não avaliada)
início
  cursor ← hist(th);
  p' ← hist(th-1);
  th ← th-2;
  se p' = NULL
    então retornar falha de casamento
    senão início
      pilha(tp+1) ← i;
      pilha(tp+2) ← l;
      pilha(tp+3) ← e;
      tp ← tp+3;
      l ← l-res(i);
      i ← primeira instrução em p';
      executar rotina de casamento;
      i ← pilha(tp-2);
      l ← pilha(tp-1);
      tp ← tp-3;
      se retorno por falha
        então retornar falha de casamento
        senão retornar sucesso;
  fim
fim

```

Figura 2.33 - Primitiva re* de *comp k

ASPECTOS DA MICROIMPLEMENTAÇÃO

Neste capítulo tentaremos propor uma linha de trabalho para a implementação ao nível de microcódigo da máquina SNOBOL. Não se pretende esgotar o assunto, apenas assinalar diversos aspectos que devem ser considerados numa implementação.

A definição e implantação completa da máquina SNOBOL em sistemas B1700 se constitui, sem dúvida, numa extensa e meticulosa tarefa que envolve problemas técnicos e principalmente operacionais, cuja solução está além dos objetivos deste trabalho.

Deve-se mencionar a escassez de referências bibliográficas e material informativo acessível, relativo ao sistema B1700, do que resultou uma reduzida abordagem das interfaces sistema operacional/máquina SNOBOL e sistema de entrada e saída/máquina SNOBOL.

3.1 Características Básicas

Inicialmente assinalamos vários pontos que servirão de referência para o desenvolvimento posterior do trabalho. Desde já fica estabelecido que o sistema B1700 será suporte para a máquina SNOBOL e serão referenciadas a seguir suas características, nomenclatura e estrutura.

a. Miscelânea

A memória principal permite acessos a até 24 bits contíguos e registradores têm em geral este comprimento. Assim, as estruturas elementares da máquina SNOBOL devem ter, de preferência, comprimento total igual ou próximo a um múltiplo de 24 bits.

Será adotado um tamanho de oito bits para caracteres, em codificação usual EBCDIC. Tal escolha envolve algum desperdício

cio de memória, porém permite uso de caracteres minúsculos. Além disso, o processador B1700 está apto a manipular de maneira eficiente somente caracteres de um, quatro e oito bits.

É previsto um sistema supervisor, em princípio em micro código, para o tratamento de condições de exceção, operações de entrada e saída, alocação de espaço, tratamento de interrupções, etc. O sistema supervisor funciona como interface entre a máquina SNOBOL e o sistema operacional residente.

A linguagem SNOBOL não declara o tipo das variáveis que manipula, desta forma, nenhuma verificação é possível em tempo de compilação. A descrição do tipo deve acompanhar o objeto, e as instruções que usam o objeto fazem as verificações e coerções necessárias. O dado passa a ser um par, dito descritor, composto do código do tipo e do valor. A escolha adequada do formato dos descritores é essencial e será vista mais adiante.

b. Estrutura global da memória

A estrutura da memória de um programa está esquematizada na figura 3.1; cada área tem a função relacionada.

Figura 3.1 - Áreas da memória

as setas indicam o sentido de expansão

A: Área para constantes do sistema: armazena entre outros, os apontadores para as bases das tabelas e pilhas, constantes utilizadas durante a interpretação, tabelas de compilação,

áreas auxiliares etc. Esta área é estática durante a execução.

B: Área para tabelas de objetos da linguagem: é composta basicamente pela tabela de variáveis naturais, tabela para espalhamento, tabela de tipos, tabela de funções, tabela de rótulos e tabela de palavras reservadas. Esta área não é estática. Para cada tabela é prevista uma área para expansão, e todas são contínuas. O endereçamento nas tabelas é relativo às respectivas bases e não traz inconvenientes na eventualidade de uma realocação das tabelas.

C: Área para pilha do sistema

D: Área para pilha de alternativas

E: Área para pilha de atribuições postergadas

Nas três pilhas o endereçamento também é relativo às respectivas bases, permitindo fáceis realocações.

F: Área dinâmica para blocos e cadeias: pode ter seu conteúdo realocado em situações de falta de espaço disponível. A administração desta área está detalhada em 3.4.

Todas as áreas indicadas não precisam ser necessariamente contíguas, cada área pode eventualmente ser segmentada e a utilização de sistemática de memória virtual deve ser considerada. O detalhamento final da estrutura da memória será função do sistema supervisor.

3.2 Descritores

Na máquina SNOBOL reconhecemos dois tipos básicos de

descritores:

- a. Descritores que armazenam o tipo e o dado de maneira concatenada na memória, são os descritores de valores numéricos.
- b. Descritores que armazenam o tipo e um apontador para o valor, sendo este em geral uma estrutura complexa. Do ponto de vista conceitual, o descritor é composto do tipo e do valor (e não do apontador).

Podemos representar os dois tipos de descritores na forma:

onde a ordem dos campos é irrelevante ao nível da representação gráfica. O campo T representa um conjunto de bits que caracterizam o tipo do descritor e V é outro conjunto de bits que determina o valor do dado.

Nosso objetivo é obter um formato adequado aos campos básicos T e V facilmente manipulável pelo processador. Devem ser levados em consideração o formato dos registradores, o repertório de microinstruções e o espaço total ocupado. Inicialmente analisamos o descritor de cadeias, que é o mais comum; em função dele serão obtidos os formatos dos outros descritores. Todos os formatos obtidos devem, se possível, ter os mesmos comprimentos afim de simplificar a implementação de matrizes, tabelas e estruturas do usuário.

3.2.1 Descritores para Cadeias

No descritor de cadeias, a cadeia propriamente dita reside na área dinâmica do sistema. Cada cadeia é caracterizada pelo seu comprimento, seu endereço na memória e caracteres que a compõem. O endereço pode indicar a posição da cadeia dentro de

uma cadeia maior, permitindo representar cadeias e subcadeias. O descritor de cadeias tem quatro campos esquematizados como:

TC	C	D	E
----	---	---	---

TC : Será o tipo do descritor, indica tratar-se de um descritor de cadeias.

C : Será o comprimento da subcadeia na memória.

D : Será o deslocamento em relação ao começo da cadeia; opcionalmente pode ser o endereço da subcadeia sendo o deslocamento dado por (D-E).

E : Será o endereço do começo da cadeia na memória.

O campo TC deve ter tamanho reduzido, a fim de fornecer o maior espaço possível aos outros campos, e não precisa ter o mesmo comprimento em todos os tipos de descritores. O campo E deve permitir endereçar toda a memória disponível no sistema. Se o campo D representar um comprimento, então D e C devem ter o mesmo comprimento, se possível. O mesmo vale para D e E se D representar um endereço.

Vamos propor dois esquemas para implementar o descritor de cadeias, levando em consideração a eficiência do microcódigo de endereçamento e os limites impostos pelo sistema.

Esquema A

TC : Campo de 4 bits, o primeiro dos quais deve estar ligado, indicando tratar-se de descritor de cadeias. Os três bits restantes em princípio não são utilizados, uma aplicação para os mesmos seria em esquemas de paginação de memória, memória virtual ou extensão de comprimentos de cadeias. O campo TC corresponde ao registrador FT.

C : Campo de 16 bits que exprime o número de bits da subcadeia;

apenas 8191 caracteres podem ser utilizados. Contudo, podemos utilizar diretamente as microinstruções: LER/GRAVAR MEMÓRIA e POSICIONAR CP nos ciclos de processamento de caracteres. Uma opção, que ainda se ajusta ao esquema, seria utilizar um dos bits de TC para acusar se o comprimento está medido um número de bits ou caracteres. O campo C corresponde ao registrador FL.

D : Campo de 24 bits que fornece não um deslocamento em relação a E, mas, um novo endereço de memória da subcadeia considerada. Assim, evitam-se contínuas adições entre E e D ao re-referenciar caracteres na memória; por outro lado D terá o tamanho de E. O campo D corresponde ao registrador FA.

E : Campo de 24 bits que fornece o endereço do bit onde começa a cadeia na memória; 2097152 caracteres podem ser endereçados. É um valor exagerado, talvez 22 bits bastassem, no entanto, este comprimento é inadequado para o descritor de valores numéricos. O campo E corresponde ao registrador FA.

Esquema B

TC: Campo de 4 bits análogo ao esquema A.

C : Campo de 16 bits que exprime o comprimento da subcadeia em caracteres (máximo de 65535). A medida em caracteres aumenta o tamanho máximo, mas não permite utilizar as microinstruções: LER/GRAVAR MEMÓRIA com computação de FA e FL e POSICIONAR CP diretamente sobre o registrador FL.

D : Campo de 20 bits que especifica o deslocamento em bits da subcadeia em relação ao começo da cadeia (máximo de 131072 caracteres, o que ultrapassa C). Qualquer endereçamento exige a soma prévia de E com D.

E : Campo de 24 bits que armazena o endereço do bit onde começa

a cadeia na memória (no máximo 2097152 caracteres podem ser endereçados diretamente).

É aparente a limitação do campo C. Como ele pode ser utilizado diretamente no registrador FL, deve fornecer o comprimento em número de bits o que diminui o comprimento máximo representável. Porém, há um fator adicional a favor do comprimento em número de bits, é o fato da microinstrução POSICIONAR CP poder ser aplicada com todas as possibilidades que oferece.

O campo C medido em caracteres aumenta o comprimento máximo representável, porém, impossibilita o uso direto de FL. Agora é necessário efetuar deslocamento de bits (inclusive com perspectiva de truncamento de bits significativos) a fim de obter comprimentos medidos em bits. A microinstrução POSICIONAR CP fica inadequada.

No esquema A o tamanho do descritor é de 68 bits, no esquema B o tamanho do descritor é de 64 bits. Uma redução destes números ainda é possível, porém, não indicada para descritores numéricos. O tamanho do descritor, apesar de grande, e de exigir pelo menos três acessos a memória, parece compatível com os descritores utilizados em outros sistemas.

Com a finalidade de evidenciar as vantagens e desvantagens dos dois esquemas, vamos observar o microcódigo necessário para duas operações elementares. Para tanto, suponhamos que os campos do descritor estão armazenados nos registradores temporários:

E em S14A

D em S15A

TC e C em S15B

Além disso, suponhamos a base da área dinâmica armazenada em S10A, e que as duas operações não ocasionam condições de exceção.

Primeira operação: Ler o caráter referenciado pelo descritor

para o registrador X e posicionar o descritor no próximo caráter. Observação: se o comprimento dado pelo descritor for nulo, não efetuar as operações e desviar para um ponto referenciado pelo rótulo XXX. O microcódigo em MIL correspondente a cada esquema está na figura 3.2.

esquema A:	LOAD F FROM S15	-carregar D, TC e C
	IF FL = 0 THEN GO TO +XXX	
	ADD S10A TO FA	-obter end. absoluto
	READ 8 BITS TO X INC FA AND DEC FL	
	SUBTRACT S10A FROM FA	-posicionar descritor
	STORE F INTO S15	
esquema B:	LOAD F FROM S15	-carregar D, TC e C
	IF FL = 0 THEN GO TO +XXX	
	ADD S14A TO FA	
	ADD S10A TO FA	-obter end. absoluto
	READ 8 BITS TO X INC FA	
	SUBTRACT S10A FROM FA	-posicionar descritor
	SUBTRACT S14A FROM FA	
	COUNT FL UP BY 1	
	STORE F INTO S15	

Figura 3.2 - Microcódigo da primeira operação

segunda operação: Ler os $\min(3, C)$ caracteres referenciados pelo descritor para o registrador X e posicionar o descritor para os próximos caracteres. O número de bits lidos deve estar em CPL ao final da operação. Se o comprimento C dado pelo descritor for nulo, não efetuar as operações e desviar para um ponto referido pelo rótulo XXX. O microcódigo em MIL está na figura 3.3.

Na primeira operação, a proporção de passos necessários entre o esquema A e B é de 6 para 9. Na segunda operação a proporção é de 6 para 13 quando houver mais de 3 caracteres na cadeia e de 6 para 15 se houver 3 ou menos caracteres para ler.

Aceitando os limites de capacidade, e na expectativa da simplicidade do microcódigo se manter para outras operações, o esquema A será adotado a partir deste ponto; o descritor de cadeias terá 68 bits.

É aparente a economia de espaço ao utilizar o próprio

esquema A:	LOAD F FROM S15	-carregar descritor
	BIAS BY FL TEST	-faz $CPL \leftarrow \min(24, FL)$
	GO TO +XXX	se $CPL \neq 0$ então salta
	ADD S10A TO FA	próxima micro.
	READ 0 BITS TO X INC FA AND DEC FL	
	SUBTRACT S10A FROM FA	-posicionar descritor
	STORE F INTO S15	
esquema B:	LOAD F FROM S15	-carregar descritor
	MOVE FL TO X	
	SHIFT X LEFT BY 3 BITS	-transformar comprimento e comparar
	MOVE 24 TO Y	
	IF X > Y THEN GO TO +AAA	
	IF X = 0 THEN GO TO +XXX	
	MOVE X TO CPL	
	MOVE NULL TO FL	-ler X caracteres
	GO TO +BBB	
.AAA	MOVE 24 TO CPL	-ler 3 caracteres
	COUNT FL DOWN BY 3	
.BBB	ADD S14A TO FA	-obter end. absoluto
	ADD S10A TO FA	
	READ 0 BITS TO X INC FA	
	SUBTRACT S10A FROM FA	-posicionar descritor
	SUBTRACT S14A FROM FA	
	STORE F INTO S15	

Figura 3.3 - Microcódigo da segunda operação

descritor para armazenar cadeias curtas de até 6 caracteres nos campos D e E. Um bit de TC indica este novo tipo, interno ao sistema e transparente ao usuário. A cadeia curta pode ser previamente convertida em cadeia normal, todas as vezes que isto simplifique alguma operação.

Nos exemplos de microcódigo que seguem, não foi considerado, por simplicidade, a possibilidade de haver cadeias curtas nos descritores.

3.2.2 Descritores para Valores Numéricos

Em função do tamanho de descritores de cadeias, será determinado o formato de descritores de tipo numérico.

Em princípio, dados do tipo REAL e INTEGER poderiam ser implementados como cadeias de algarismos e que obedeceriam algu

mas regras de formação. Tal sistemática torna-se interessante para valores inteiros, já que não teriam limitação no comprimento e nem haveria necessidade de conversões tipo cadeia/inteiro ou inteiro/cadeia. Valores reais incluiriam um ponto decimal e, se certa precisão limite fosse ultrapassada, seriam armazenados em formato padrão com mantissa e expoente (ainda em forma de cadeia).

Esta forma, pouco tradicional, para reais e inteiros é adequada ao processador B1700, que inclui aritmética de caracteres e portanto evita conversões para forma binária nas operações aritméticas.

Outra solução, diametralmente oposta, é implementar reais e inteiros como valores binários, e que é empregada na maioria dos sistemas atuais em função da aritmética disponível. Contudo, as conversões de formato binário/cadeia e cadeia/binário, efetuadas por processos iterativos a nível de "software", são extremamente lentas em comparação às operações aritméticas.

Uma solução intermediária, que evita a utilização de espaço excessivo e minora os problemas de conversão será empregada. Nesta solução o descritor armazena o valor na forma de dígitos e as computações serão efetuadas utilizando a aritmética decimal do sistema. A representação do descritor será:

S : Campo de quatro bits denotados S_1, S_2, S_3, S_4

S_1 é nulo indicando que não se trata de um descritor de cadeias

S_2 vale um indicando ser descritor de um valor inteiro ou real

S_3 é o sinal do expoente

S_4 é o sinal da mantissa.

E : Campo de 8 bits que armazena o valor do expoente em formato binário, permitindo variação de -255 a +255. A distinção

entre reais e inteiros é feita, adotando expoente nulo e sinal de expoente negativo para valores inteiros. Assim 723 tem expoente -0 e 723.0 tem expoente +0 já que o ponto decimal deve estar à direita da mantissa.

M : Campo de 56 bits que armazena a mantissa como 14 dígitos, formato decimal. Permite conversão rápida de e para cadeia. Utiliza aritmética decimal do processador.

A máquina SNOBOL prevê as operações aritméticas de soma, subtração, multiplicação, divisão e exponenciação implementadas como funções primitivas. A detecção de condições de exceção durante as operações ocasiona retorno por falha. Funções matemáticas podem ser implementadas através de funções externas e eventualmente intrínsecas do sistema.

A representação de valores numéricos em forma de dígitos não evita completamente a conversão de/para forma binária. Isto decorre da aritmética utilizada ao nível de microinstruções (endereçamento de memória, controle da unidade aritmética e operações de 4 bits) ser intrinsecamente binária. Na parte referente a padrões primitivos este ponto será assinalado novamente.

3.2.3 Descritores para outros Tipos de Objetos

Objetos de tipo NAME, EXPRESSION, PATTERN, ARRAY, TABLE, CODE e do usuário, possuem sempre descritor de 68 bits. O tamanho todo não é empregado, porém, a uniformização facilita a administração de estruturas tipo matriz, tabela e do usuário. O tipo do descritor é indicado nos campos TC e C e testado no registrador FB através da microinstrução SALTAR MICROINSTRUÇÃO SE.

Descritores de endereços internos, descritores de identificação de blocos e descritores de código normal ou de padrões, não precisam ter necessariamente 68 bits. Particularmente, descritores de código podem ter comprimento função da instrução que

armazenam, permitindo compactação do código objeto com economia de espaço e tempo de acesso. O formato final dos descritores de código será determinado pela simplicidade de implementação.

Endereços internos e objetos tipo NAME podem relacionar endereços da tabela de palavras-chave, tabela de variáveis naturais ou endereços internos a blocos residentes na área dinâmica (matrizes, tabelas, etc.). Referências a elementos das tabelas necessitam somente um endereço relativo ao início da tabela. Referências a blocos alocados na área dinâmica exigem um par: o endereço do descritor identificação do bloco e o endereço relativo do elemento no bloco.

Objetos tipo EXPRESSION referenciam pontos do código objeto e especificam dois endereços: do descritor identificação do bloco de código e o endereço relativo da instrução cfun * que inicia a expressão não avaliada no bloco.

Objetos tipo PATTERN, ARRAY, TABLE, CODE e tipo do usuário relacionam no descritor o endereço do bloco correspondente na área dinâmica. Descritores de identificação ocorrem no início destes blocos relacionando outras informações.

Entre as operações previstas para os descritores mencionados, temos: busca linear em tabelas, indexação em matrizes, conversão para formato binário, busca na tabela de variáveis naturais, compilação de código fonte, etc..

3.2.4 Registrador de Cadeias

O registrador de cadeias é um registrador residente no processador central que permite o armazenamento e referência às cadeias em processamento. Como as cadeias não têm um comprimento fixo, as mesmas são processadas em acessos repetidos à memória, mantendo no processador somente o trecho necessário ao processo em desenvolvimento.

Qualquer cadeia normalmente é processada da esquerda para a direita, porém, há exceções (por exemplo, processamento de alternativas em casamento de padrões). Em geral, as operações e

lementares atuam sobre apenas um caráter independentemente do contexto que o cerca. Frequentemente também é necessário o comprimento da cadeia que resta processar, sendo conveniente manter este valor continuamente disponível no processador.

Em (Shapiro 1972) sugere-se um registrador de cadeias formado por duas partes distintas: o descritor da cadeia e o trecho da cadeia em processamento. Esquemáticamente:

que armazena os k caracteres consecutivos da cadeia dada por: $c_1 c_2 \dots c_i c_{i+1} \dots c_{i+k-2} c_{i+k-1} \dots c_{n-1} c_n$ sendo que:

- TC : armazena o tipo da cadeia e outras informações
- C : é o comprimento restante da cadeia, de c_i até c_n
- D : é o endereço de memória do caráter c_i
- E : é o endereço de memória do início da cadeia.

A carga do descritor de cadeias, na primeira parte do registrador, automaticamente alimenta a segunda parte com os caracteres correspondentes. Ao ter sido processado c_i , D é incrementado, C é decrementado e a cadeia armazenada é deslocada para a esquerda liberando uma posição à direita. O processo prossegue até que um número fixo de posições à direita sejam liberadas e uma realimentação, também automática, se processe. Desta forma, o comprimento total da cadeia será dado por $E-D+C$ e o comprimento da subcadeia a processar será dado por C . Quando C atingir zero, a cadeia terá sido processada.

A parte do descritor define completamente o conteúdo atual do registrador, pode ser guardada e utilizada a qualquer instante. Ao carregá-la novamente no registrador, segue-se a alimentação automática de caracteres que reconstituirá o registrador.

Na nossa implementação, o registrador de cadeias será um caso particular do esquema sugerido. Isto porque o sistema B1700 não oferece capacidade de armazenamento contíguo de caracteres

no processador, e todos os acessos à memória são de no máximo três caracteres.

O registrador armazenará apenas o descritor da cadeia, e a rotina em andamento se encarregará de efetuar os acessos à memória da forma que lhe é mais adequada. Inclusive pode tirar proveito da capacidade do sistema de referenciar qualquer bit na memória, em operações que envolvem campos de 1 a 24 bits. O descritor da cadeia pode ser imaginado como um cursor que referenciava o caráter em processamento na memória.

Os registradores de cadeias são manipulados independentemente das pilhas de recursão e não constituem seus topos. Os topos das pilhas residirão no processador desde que haja capacidade disponível. As pilhas podem ser utilizadas durante o processamento do conteúdo dos registradores de cadeias.

A partir deste ponto vamos supor que os pares de registradores temporários: (S00, S01), (S02, S03) e (S04, S05) tenham funções de registradores de cadeias. Assim, para um descritor com campos TC, C, D e E a correspondência no primeiro registrador será:

S00B	armazena	TC e C
S00A	"	D
S01A	"	E
S01B	"	resíduos, temporários, etc..

Esta correspondência é ditada pelas microinstruções CARREGAR/ARMazenar F DE/EM DUPLO TEMPORÁRIO, AGREGAR TEMPORÁRIO A FA e PERMUTAR F E DUPLO TEMPORÁRIO. A carga dos registradores é efetuada pelas funções intrínsecas do sistema a partir do topo das pilhas, pela rotina de casamento de padrões a partir do código para padrões, pela rotina de busca de padrões ao carregar a cadeia fonte, etc.. Em particular, no estado de casamento de padrões, o cursor deve ser armazenado, preferivelmente em (S00, S01), a fim de tirar proveito do registrador FLCN que registra as comparações entre FL e SFL. Nenhuma restrição existe quanto a utilizar os registradores temporários citados para outros fins.

3.3 Processamento de Cadeias

O processamento de cadeias ao nível do microcódigo é apresentado em seguida. O objetivo é mostrar de modo um pouco mais concreto o microcódigo para algumas funções primitivas bem como assinalar um modo de implementar cadeias de caracteres não ordenados. Serão discutidos aspectos que facilitam e dificultam determinadas operações no sistema B1700.

Nos exemplos que seguem não se teve por objetivo especificar detalhes, nem tampouco apresentar o melhor microcódigo para os mesmos. Os pontos importantes são abordados de maneira genérica e prefere-se omitir aspectos corriqueiros, a fim de não aumentar excessivamente o microcódigo apresentado.

3.3.1 Funções e Operadores Primitivos

São chamados pela rotina aval através da tabela de funções. A rotina aval desvia para o microcódigo da primitiva correspondente que trata de obter seus argumentos da pilha do sistema e armazená-los em registradores adequados. As operações elementares ocorrem sobre os registradores até que um resultado seja computado. A primitiva retorna para a rotina aval através de um desvio para um ponto de seu microcódigo.

Como exemplo, é apresentado na figura 3.4 o código em MIL da primitiva TRIM, supondo que seu argumento seja uma cadeia cujo descritor já está armazenado no registrador temporário S00. A nova cadeia é retornada também no registrador temporário S00.

A primitiva TRIM é implementada utilizando o recurso da leitura reversa. O endereço do fim da cadeia é obtido e os caracteres processados em grupos de três, do fim para o começo da cadeia, até obter um caráter não branco. O microcódigo torna-se extenso pois prefere-se processar grupos de três caracteres o que fornece o menor número de acessos à memória principal, diminuindo o tempo total de processamento da primitiva. Quando u-

LOAD F FROM S0	
MOVE FL TO S4A	
ADD S4A TO FA	
MOVE " " TO Y	-endereço do fim da cadeia em FA
.AAA BIAS BY FL TEST	-CPL+ min(24,FL) se CPL
GO TO +CCC	resultar nulo pula próxima microinstrução
READ 0 BITS REVERSE TO X	-lê caracteres do fim e
MOVE MSKY TO Y	compara CPL caracteres
IF X NEQ Y THEN GO TO +BBB	
COUNT FA DOWN AND FL DOWN BY CPL	
GO TO -AAA	
.BBB MOVE " " TO Y	-caracteres finais
MOVE X TO T	
EXTRACT 8 BITS FROM T(16) TO X	-caráter a direita de T
IF X NEQ Y THEN GO TO +CCC	-primeiro não branco
COUNT FA DOWN AND FL DOWN BY 8	
EXTRACT 8 BITS FROM T(8) TO X	-penúltimo caráter de T
IF X NEQ Y THEN GO TO +CCC	-primeiro não branco
COUNT FA DOWN AND FL DOWN BY 8	
.CCC MOVE SOA TO FA	
STORE F INTO S0	

Figura 3.4 - Microcódigo da primitiva TRIM

ma condição de desigualdade é detectada, os possíveis dois últimos caracteres brancos são processados um a um a partir do registrador T.

O código de TRIM foi obtido supondo que não ocorram condições de exceção durante o processamento da cadeia. Algumas das condições anormais que poderiam acontecer são: erro de paridade da memória, interrupções devidas a processos de entrada e saída de dados em andamento, erros de limite de endereçamento na memória principal, etc.

Algumas condições devem ser tratadas imediatamente após o instante em que ocorreram, como é o caso do erro de paridade ou de limite de endereçamento. Outras podem ter a sua detecção e tratamento adiados até um instante mais oportuno. O tratamento das condições de exceção é delegado a rotinas padrões do sistema supervisor e/ou sistema operacional.

3.3.2 Padrões Primitivos

São rotinas chamadas pela rotina de casamento de padrões e correspondem a instruções de máquina do sistema. Padrões primitivos retornam sucesso de casamento, falha de casamento ou falha de comprimento à rotina de casamento. Os três tipos de retorno podem corresponder a três pontos distintos do microcódigo da rotina de casamento, para onde é feito o desvio da primitiva.

De um modo geral, todo padrão primitivo testa se há caracteres suficientes na cadeia fonte para tentar o casamento. Se não houver, a condição de falha de casamento é imediatamente assinalada. Além disso, em modo QUICKSCAN, a rotina primitiva verifica, se o resíduo mais o número de caracteres necessários ao padrão estão ainda disponíveis na cadeia fonte. Se tal não ocorrer, a condição de falha de comprimento é imediatamente assinalada.

Cada instrução de máquina do código para casamento de padrões, possui três campos: o código da instrução primitiva, o argumento da primitiva, e o seu resíduo dentro do padrão. Um formato indicado para as instruções seria:

código (8 bits)	argumento (68 bits)	resíduo (16 bits)
--------------------	------------------------	----------------------

onde o argumento deve ter tamanho suficiente para armazenar um descritor. Um objeto de tipo "CODE" passa a ser um bloco, contendo uma seqüência de instruções com o formato dado.

Adiante detalhamos alguns padrões primitivos em MIL. Será suposto sempre que o argumento e o resíduo do padrão estão armazenados em:

S2A, S2B, S3A : argumento do primitivo
S3B : resíduo.

O argumento do primitivo é em geral um descritor, e portanto ocupa um registrador de cadeias. O argumento pode, eventualmente,

ser uma expressão não avaliada, e esta situação deve ser prevista no microcódigo do padrão primitivo. Este avaliará o argumento através de chamada da primitiva EVAL e em seguida utiliza o resultado como argumento.

Nos exemplos que seguem, a bem da simplicidade, não foi considerada a possibilidade do argumento ser uma expressão não avaliada. Também, da mesma forma que em 3.3.1, não foram levadas em consideração as condições de exceção que podem ocorrer durante o processamento do padrão primitivo.

Nos exemplos também é suposto que o bit mais à esquerda no registrador CA indica o estado da palavra chave &FULLSCAN (ligado se modo QUICKSCAN) e que o cursor está armazenado no registrador de cadeias (S00, S01). Serão caracterizados por meio de exemplos, três tipos de padrões primitivos.

a. Padrão tipo cadeia ordenada

Padrões deste tipo são cadeias comuns de caracteres que devem casar integralmente na cadeia fonte. Seu código de máquina é LIT e o argumento especifica a cadeia a casar. Na figura 3.5 aparece seu microcódigo.

O argumento do padrão primitivo LIT é comparado integralmente com a cadeia fonte, a partir da posição atual do cursor. Como o cursor é atualizado ao longo do processo, e pode posteriormente ocorrer falha de casamento, ele é previamente salvo em S4. A rotina processa três caracteres por vez, a menos dos trechos finais das cadeias, utilizando a capacidade máxima de acesso a memória.

Observe-se, que as microinstruções STORE, LOAD e BIAS, que seguem imediatamente as microinstruções READ, são executadas em paralelo com o ciclo da instrução de leitura da memória, uma característica já apontada para o sistema B1700. Observe-se também, que incrementos arbitrários em FL precisam ser efetuados via registrador SUM, nenhuma alternativa existe se os incrementos excederem 5 bits de comprimento.

```

IF CA(0) TRUE THEN
BEGIN
    MOVE S3B TO X
    MOVE S2B TO Y
    MOVE 16 TO CP
    MOVE SUM TO FL
    IF FL > SFL THEN GO TO FALHA-DE-COMPRIMENTO
END ELSE
BEGIN
    MOVE S2B TO FB
    IF FL > SFL THEN GO TO FALHA-DE-CASAMENTO
END
LOAD F FROM S0
STORE F INTO S4
.AAA LOAD F FROM S2
BIAS BY FL TEST
GO TO SUCESSO
READ 0 BITS TO X INC FA AND DEC FL
STORE F INTO S2
LOAD F FROM S0
BIAS BY FL
READ 0 BITS TO Y INC FA AND DEC FL
STORE F INTO S0
IF X = Y THEN GO TO -AAA
LOAD F FROM S4
STORE F INTO S0
GO TO FALHA-DE-CASAMENTO

```

-modo QUICKSCAN

-modo FULLSCAN

-guarda cursor

-carrega cadeia argumen
to

-carrega cadeia fonte

-restitui cursor

Figura 3.5 - Microcódigo do primitivo LIT

b. Padrão do tipo posição/comprimento

Estes padrões primitivos têm por argumento um valor inteiro em representação binária. O argumento é definido no instante da construção do padrão pela função construtora de padrões correspondente. O argumento da função construtora é um objeto que coercido deve resultar num valor de tipo INTEGER, portanto, um valor inteiro em representação decimal.

A função construtora se encarrega de efetuar a conversão para binário e transformação do valor para posição/comprimento em bits. A conversão é necessária já que a aritmética de endereçamento é binária. A fim de evitar conversões contínuas,

em tempo de casamento de padrões, a conversão ocorre em tempo de construção de padrões, sempre que possível. Se a construção do padrão puder ser isolada de seu casamento, ao nível do programa fonte, melhor velocidade será obtida (aliás, isto vale para qualquer padrão).

Como exemplo, é apresentado na figura 3.6 o padrão primitivo LEN. O argumento do padrão é suposto ocupar os 24 bits do campo D do argumento da instrução. O argumento também pode ser uma expressão não avaliada, cujo tratamento é feito através de EVAL, com conversão do resultado para formato binário. O padrão LEN simplesmente ajusta o valor do cursor adequadamente, e não necessita efetuar acessos a memória.

```

IF CA(0) TRUE THEN
BEGIN
    MOVE S3B TO X
    MOVE S2A TO Y
    MOVE 16 TO CP
    MOVE SUM TO FL
    IF FL > SFL THEN GO TO FALHA-DE-COMPRIMENTO
END ELSE
BEGIN
    MOVE S2A TO FB
    IF FL > SFL THEN GO TO FALHA-DE-CASAMENTO
END
LOAD F FROM S0
ADD S2A TO FA
MOVE 16 TO CP
MOVE FL TO X
MOVE S2A TO Y
MOVE DIFF TO FL
STORE F INTO S0
GO TO SUCESSO

```

-modo QUICKSCAN

-modo FULLSCAN

-ajuste do cursor

Figura 3.6 - Microcódigo do primitivo LEN

Além do padrão primitivo LEN, há mais cinco padrões primitivos que apenas operam sobre o valor do cursor. Resumidamente relacionamos cada um:

POS(n) : Casa a cadeia nula e retorna sucesso se $n=D-E+1$, sendo D e E os campos de deslocamento e endereço do cur

sor. Retorna falha de casamento caso contrário.

RPOS(n) : Casa a cadeia nula e retorna sucesso se $n=C$ sendo C o campo de comprimento do cursor. Retorna falha de casamento caso contrário.

TAB(n) : Ajusta o campo D do cursor para $E+n-1$ e casa a cadeia entre as duas posições do cursor. Se n for menor que $D-E+1$ então retorna falha de casamento.

RTAB(n) : Ajusta o campo C do cursor igual a n. O deslocamento D também é ajustado. Se n for maior do que C então retorna falha de casamento.

REM : Tem o mesmo efeito que RTAB(0).

c. Padrão do tipo cadeia não ordenada

Num padrão deste tipo não importa o contexto no qual se encontra cada caráter do padrão. A cadeia não ordenada determina somente um conjunto de caracteres válidos ou não válidos durante a fase de casamento do padrão. O padrão primitivo precisa somente reconhecer os caracteres na cadeia fonte. Uma maneira adequada de implementar padrões deste tipo é através de um vetor de bits de presença, sendo que a configuração em bits do caráter a casar endereça um dos bits do vetor.

A aplicação típica destes padrões é em casamento de cadeias através dos primitivos SPAN, BREAK, ANY e NOTANY. O problema principal que se pretende resolver é implementar cadeias não ordenadas de modo a obter um processamento rápido e evitar desperdício de espaço para armazenamento. Em função das características do sistema B1700 e da estrutura de descritores proposta, optamos por um esquema de representação de cadeias não ordenadas em três tipos:

1. Cadeias curtas de somente um caráter, muito comuns em pa-

drões do tipo SPAN("caráter"), por exemplo.

2. Cadeias que representam classes de caracteres e cuja configuração em bits no código EBCDIC apresente características análogas; usadas, por exemplo, em SPAN("0123456789"). A característica comum empregada são os dois bits de maior ordem na representação EBCDIC. Os 256 caracteres possíveis são particionados em quatro conjuntos de 64 caracteres cada.
3. Cadeias que reúnem caracteres, que não pertencem todos ao mesmo tempo, a um dos quatro conjuntos citados em 2. Estas cadeias sempre ocupam 256 bits.

O campo TC do descritor de cadeias indica, através de seus quatro bits, qual dos três tipos de cadeias não ordenadas ele representa. Supondo que os bits de TC sejam $b_1b_2b_3b_4$ então:

- $b_1 = 0$ Indica tratar-se de um descritor de cadeias.
- $b_2 = 0$ Indica tratar-se de um descritor de cadeia não ordenada com 64 bits armazenados no próprio descritor. Os bits estão armazenados ordenadamente nos campos C, D e E. Os bits b_3 e b_4 dão a configuração dos dois bits de maior ordem do conjunto de 64 bits.
- $b_2 = 1$ Indica tratar-se de uma cadeia não ordenada de um caráter ou de 256 bits.
- $b_3 = 1$ Indica cadeia de um único caráter armazenado a direita no campo D.
- $b_3 = 0$ Indica cadeia de 256 bits armazenados na memória, o enfileiramento é dado pelos campos C, D e E do descritor.

A obtenção das cadeias de caracteres não ordenados é efetuada pelas funções construtoras de padrões. Mostramos em se-

guida o código em MIL para o padrão primitivo SPAN para cada um dos três tipos de cadeias. Em qualquer caso, o argumento está em (S02, S03) e o cursor em (S00, S01).

1. SPAN tendo como argumento uma cadeia não ordenada de um único caráter, está detalhado na figura 3.7. A rotina lê apenas um caráter por vez da cadeia fonte, porque assim, obtêm-se uma velocidade maior. Deve-se o fato à pequena capacidade de X, Y, T ou L (3 caracteres somente). A rotina fornece uma economia de 40 a 50% de tempo de processamento em relação ao esquema de 64 bits.

```
MOVE S2A TO Y                -carrega caráter em Y
LOAD F FROM S0               -carrega cursor em F
IF FL = 0 THEN GO TO FALHA-DE-CASAMENTO
READ 8 BITS TO X
IF X ≠ Y THEN GO TO FALHA-DE-CASAMENTO
.AAA COUNT FA UP AND FL DOWN BY 8
IF FL = 0 THEN GO TO +BBB    -esgotou cadeia fonte
READ 8 BITS TO X
IF X = Y THEN GO TO -AAA
.BBB STORE F INTO S0        -atualiza cursor
GO TO SUCESSO
```

Figura 3.7 - Microcódigo do primitivo SPAN para um caráter de argumento

2. SPAN tendo como argumento uma cadeia não ordenada de 64 bits, é detalhado na figura 3.8. A rotina em MIL determina a localização do bit em relação aos campos C, D e E. Em seguida carrega o campo em questão em X e testa o bit indicado através de MSBX. O esquema de 64 bits fornece economia de espaço apreciável; a velocidade de processamento é comparável com o esquema de 256 bits.

3. SPAN tendo por argumento uma cadeia não ordenada de 256 bits,

MOVE 1 TO LB	
LOAD F FROM S2	
MOVE FT TO X	
SHIFT X RIGHT BY 2 BITS	-guarda os 2 bits em LA
MOVE X TO LA	
MOVE FL TO S3B	-separa os 16 bits iniciais e carrega cursor
LOAD F FROM S0	
.AAA IF FL = 0 THEN GO TO +BBB	
READ 8 BITS TO T	-ler caráter
MOVE LA TO X	
EXTRACT 2 BITS FROM T(16) TO Y	
IF X ≠ Y THEN GO TO +BBB	-compara 2 bits de maior ordem
EXTRACT 6 BITS FROM T(18) TO X	
MOVE 40 TO Y	
IF X ≥ Y THEN	
BEGIN	-bit nos últimos 24 do descritor
MOVE DIFF TO CPL	
MOVE S3A TO X	
IF MSBX FALSE THEN GO TO +BBB	
END ELSE	
BEGIN	
MOVE 16 TO Y	
IF X ≥ Y THEN	
BEGIN	-bit nos 24 intermediários do descritor
MOVE DIFF TO CPL	
MOVE S2A TO X	
IF MSBX FALSE THEN GO TO +BBB	
END ELSE	
BEGIN	-bit nos 16 iniciais do descritor
MOVE X TO CPL	
MOVE S3B TO X	
IF MSBX FALSE THEN GO TO +BBB	
END	
END	
MOVE 0 TO LB	
COUNT FA UP AND FL DOWN BY 8	
GO TO -AAA	
.BBB SKIP WHEN LB EQL 0	-testa se pelo menos um caráter casou
GO TO FALHA-DE-CASAMENTO	-restaura cursor
STORE F INTO S0	
GO TO SUCESSO	

Figura 3.8 - Microcódigo do primitivo SPAN até 64 caracteres

está detalhado na figura 3.9. É utilizada a capacidade de ler campos de um único bit da memória. O método é extremamente simples, porém, exige acessos à memória principal e ocupa muito es

MOVE 1 TO LB	
LOAD F FROM S0	
GO TO +BBB	
.AAA MOVE 0 TO LB	-SPAN casa pelo menos
LOAD F FROM S0	um caráter, LB regis-
COUNT FA UP AND FL DOWN BY 8	tra esta condição
STORE F INTO S0	
.BBB IF FL = 0 THEN GO TO +CCC	
READ 8 BITS TO X	-ler caráter da cadeia
LOAD F FROM S2	fonte
MOVE 24 TO CP	
MOVE FA TO Y	
MOVE SUM TO FA	
READ 1 BITS TO T	-lê o bit especificado
SKIP WHEN TF EQL 0	para o registrador T
GO TO -AAA	
.CCC SKIP WHEN LB EQL 0	
GO TO FALHA-DE-CASAMENTO	
GO TO SUCESSO	

Figura 3.9 - Microcódigo do primitivo SPAN até 256 caracteres

paço, já que a maioria das cadeias não ordenadas é composta de poucos caracteres.

Outros padrões primitivos, do tipo cadeia não ordenada, admitem código microprogramado análogo ao apresentado para SPAN, e são relacionados resumidamente a seguir. Todos eles admitem os três tipos de cadeias não ordenadas como argumento.

ANY(cadeia) : Efetua o teste somente uma vez para o próximo caráter da cadeia fonte. Se o caráter não estiver presente na cadeia não ordenada, retorna falha de casamento.

NOTANY(cadeia): Idem a ANY. Falha se houver casamento.

BREAK(cadeia): Avança o cursor até atingir um caráter que conste da cadeia não ordenada. O cursor permanece a-

pontando para o caráter imediatamente anterior e retorna sucesso. Se a cadeia fonte for esgotada antes de ocorrer um casamento, então é retornada falha de casamento.

Mencione-se que cadeias não ordenadas podem ser representadas através de cadeias comuns. Isto exige, durante seu casamento, buscas dos caracteres da cadeia fonte na cadeia não ordenada. O processo é tanto mais lento quanto maior o comprimento das cadeias envolvidas, exige muitos acessos a memória e não fornece economia de espaço apreciável.

3.4 Manipulação das Estruturas na Memória

Basicamente, a estrutura da memória se compõe de quatro áreas distintas:

1. área para constantes do sistema
2. área para tabelas de objetos da linguagem
3. área para as pilhas de recursão
4. área dinâmica.

As três últimas áreas prevêm um espaço para expansão utilizado ao ser atingida uma situação na qual não haja mais espaço disponível no sistema. Porém, o sistema SNOBOL tem por característica a capacidade de liberar objetos ao longo do processamento, objetos estes que não podem mais ser referenciados. O objetivo é conseguir reaproveitar o espaço não mais necessário, tornando-o disponível ao sistema.

3.4.1 Alocação de Espaço

A alocação de espaço é analisada separadamente para cada uma das quatro áreas que constituem a memória do sistema.

A área para constantes é estática e nunca exige alocação adicional, nem tampouco libera espaço.

A área para tabelas normalmente prevê uma área de expansão para cada tabela; somente a tabela de palavras reservadas não admite expansão. Pode ocorrer que a área de expansão também se esgote, e neste caso as tabelas são copiadas, como um todo, para uma área maior. Não há necessidade de realocação dos apontadores, visto que todo endereçamento é relativo às bases das respectivas tabelas. Cada base é uma constante do sistema.

A área para pilhas de recursão prevê uma área de expansão para cada pilha. Novamente, todos os endereços são relativos às bases das pilhas não provocando problemas durante uma eventual realocação. Observe-se que processos recursivos também podem esgotar facilmente as áreas de expansão.

A área dinâmica, na qual são alocados os blocos e cadeias, é normalmente uma área razoavelmente extensa e inicialmente livre, da memória do sistema. Ela passa a ser utilizada na medida em que o sistema requisita espaço. O esgotamento desta área exige a pesquisa de áreas não mais referenciáveis, sua coleta e reaproveitamento pelo sistema. Eventualmente o processo de coleta pode ser estendido para a área de tabelas, caso o espaço ocupado pelas mesmas ultrapasse um limite interno.

3.4.2 Coleta de Áreas Livres

A coleta de áreas livres nas tabelas de variáveis naturais, funções, rótulos e tipos, normalmente deve ser evitada, uma vez que os programas não liberam ou liberam pouco espaço nesta área. É recomendável que a coleta seja tentada somente em situações de total esgotamento de memória livre. A coleta de áreas livres pesquisa, de preferência, a área dinâmica da memória.

Vários métodos de coleta podem ser propostos. Em (Harrison 1973) encontramos os métodos descritos informalmente a seguir

em função da máquina SNOBOL.

Método 1

Consta basicamente em percorrer todas as tabelas e pilhas do sistema pesquisando referências à área de tabelas e área dinâmica. Todos os objetos apontados serão marcados de alguma forma, indicando tratar-se de objetos referenciáveis após a coleta das áreas não marcadas. Objetos apontados, por outro lado, podem conter novos apontadores, tornando recursivo o processo de marcação. A recursividade exige espaço adicional nas pilhas de recursão; uma solução seria a utilização de memória secundária durante o processo de marcação.

Toda referência a blocos contém dois endereços, o endereço do descritor de identificação e o endereço do objeto no bloco. O endereço do descritor de identificação fornece diretamente o descritor que sofrerá marcação.

Após o processo de marcação, a área dinâmica será realocada. Isto consiste em percorrer todos os objetos desta área, descartando áreas livres não marcadas, e inserindo no descritor de identificação de cada bloco o endereço da nova localização do objeto na área dinâmica. Em seguida, novamente, todas as tabelas e objetos serão percorridos e cada apontador substituído pelo apontador para a nova localização do objeto apontado. Finalmente, pode ter lugar a transferência de cada objeto para seu novo endereço, finalizando o processo de realocação da memória.

Todos os blocos da área dinâmica precisam ter especificado, no seu descritor de identificação, o comprimento do bloco bem como, devem ter espaço suficiente para armazenar o apontador para sua nova posição na memória, durante o processo de coleta das áreas livres. Blocos de descritores possuem espaço para tanto, mas cadeias de caracteres passam a necessitar também um descritor de identificação. Também a realocação de subcadeias se processa pela realocação da cadeia toda; a manipulação da subcadeia isoladamente exige tratamento especial.

Método 2

Neste processo a fase de marcação consta em obter uma lista de todos os apontadores para determinado bloco. Na fase de marcação percorre-se todas as tabelas e objetos da área dinâmica. Quando um apontador para um bloco é encontrado, atribui-se ao apontador o conteúdo do campo de comprimento do bloco, e ao campo de comprimento o endereço do apontador. Desta forma obtemos para cada bloco uma lista ligada dos descritores que para ele apontam.

Em seguida principia a fase de realocação dos apontadores. Consiste em percorrer a área dinâmica identificando espaços liberados e para cada bloco que deva ser realocado, percorrer sua lista de apontadores trocando os mesmos pelo novo endereço que o bloco ocupará. Ao final da lista o campo de comprimento é reconstituído.

O passo final consiste na cópia dos blocos para suas novas posições. Uma nova pesquisa na área dinâmica seria necessária; pode-se evitar a pesquisa se construirmos durante o passo dois uma nova lista ligada com os blocos que precisam ser copiados. Esta lista é percorrida e os blocos copiados.

Também neste método há a necessidade de armazenar cadeias em blocos; da mesma forma é problemática a realocação de subcadeias.

Método 3

Tendo em vista a capacidade do sistema operacional fornecer páginas disponíveis de memória (inclusive memória secundária), um processo alternativo de realocação de memória consiste em copiar os objetos a serem salvos para uma nova área de memória. Após a cópia, a área antiga é liberada para o sistema operacional.

O processo de marcação, realocação de apontadores e cópia das estruturas é concomitante. Cada estrutura é marcada após a cópia e o endereço de sua nova localização colocado no lu

gar de sua posição antiga e do apontador para ela. Antes da cópia é verificado se a estrutura já foi copiada, em caso afirmativo a nova localização é obtida e colocada no apontador para ela. Cuidado especial merecem os blocos com apontadores para outras estruturas, neste caso o processo é recursivo, sendo o bloco copiado somente depois que todos os seus apontadores tiverem sido processados e realocados.

O processo é simples, porém exige mais memória que os métodos tradicionais. Cadeias de caracteres passam a não necessitar descritores de identificação apenas um bit adicional, antes de seu primeiro caráter, que informa se a cadeia já foi copiada (e agora armazena um descritor de endereço) ou se ela ainda não foi copiada. A implementação deste bit adicional não oferece nenhuma dificuldade no sistema proposto. Agora exige-se cadeias com comprimento mínimo igual ao comprimento do descritor de endereço; a sistemática de subcadeias e/ou a implementação de "cadeias curtas" completa a solução deste problema.

Subcadeias agora podem ser processadas de modo alternativo: se a subcadeia for subcadeia de uma cadeia já copiada então ela permanece subcadeia. Caso a cadeia ainda não foi copiada pode-se optar em copiar apenas a subcadeia, não marcando a cadeia. O processo também pode ser feito em duas partes, copiando as cadeias completas na primeira parte e relacionando as subcadeias para uma cópia posterior. Assim descartamos as "bordas" não referenciáveis em cadeias.

Dentre os métodos apresentados, este último parece o mais adequado ao contexto da máquina SNOBOL.

COMENTÁRIOS FINAIS

Mostramos ao longo deste trabalho uma maneira de viabilizar uma máquina SNOBOL num computador com capacidade de microprogramação dinâmica. Verificamos a importância da definição das estruturas de dados que o sistema manipula, sendo todo o capítulo 2 devotado a esta tarefa. Verificamos também o cuidado necessário, o grau de detalhe envolvido e a extensão da microimplementação discutida no capítulo 3. Como subproduto deste trabalho, as estruturas definidas no capítulo 2 permitem implementar SNOBOL como sistema compilador/interpretador tradicional em nível de software, fornecendo uma alternativa para as implementações em SIL.

A tarefa de consolidação deste trabalho envolveu, entre outros, o estudo detalhado do sistema B1700 a partir da bibliografia existente. Com o intuito de reforçar os conhecimentos, de efetivamente microimplementar uma pequena máquina (do tipo ALGOL) no sistema, e na falta de outras alternativas, foi desenvolvido um simulador para o sistema B1700. Tal trabalho revelou-se didático e útil no estabelecimento dos pontos básicos de um sistema interpretativo microprogramado, e serviu também para testar alguns trechos de rotinas da máquina virtual.

A linguagem SNOBOL também mereceu atenção. Ficou patente a dificuldade na interpretação da semântica de várias construções, em vista da própria complexidade da linguagem, e das breves informações disponíveis. Amiúde recorreu-se a testes em computador, com a versão SIL do SNOBOL, que esclarecessem pontos duvidosos ou omissos da bibliografia disponível. Particularmente árdua foi a tarefa de esclarecer detalhes da utilização de funções e do casamento de padrões em modo QUICKSCAN.

Vários comentários ainda poderiam ser feitos; relacionamos alguns:

- a. A extensão do microcódigo que deve resultar da implementação completa da máquina SNOBOL: em vista da relativamente pouca memória de controle disponível no sistema, a segmentação do microcódigo é certa. A presença alternada e contínua de segmentos distintos na memória de controle deve ser vista com cuidado, inclusive pode tornar o sistema menos eficiente do que um sistema interpretado tradicional.
- b. A necessidade de certificar a correção de todo o microcódigo é ponto absolutamente fundamental num sistema de produção.
- c. Apesar da potencialidade do sistema B1700, ele não corresponde plenamente às necessidades da máquina SNOBOL. A limitação de 24 bits nos registradores, a limitação do registrador FL, a dificuldade em processar caracteres armazenados no processador, a necessidade de conversões, são alguns aspectos que complicam a implementação.
- d. Diversos pontos da máquina SNOBOL e da microimplementação merecem um estudo mais profundo com pesquisa e avaliação de alternativas. Citamos:
 - A implementação de um subconjunto da linguagem.
 - A alteração da linguagem com exclusão de pontos nevrálgicos (tais como OPSYN e QUICKSCAN).
 - A codificação de parte da máquina SNOBOL em código de máquina disponível no sistema, parte em microcódigo.
 - A definição alternativa das estruturas de dados, tal como utilizada em (Dewar e Mccann 1977).
 - A otimização do microcódigo das rotinas básicas do sistema em função dos recursos disponíveis pelo sistema B1700.

A velocidade da máquina virtual foi estimada superficial

mente. Testes, para algumas rotinas de processamento de cadeias, entre a versão SIL do sistema B6700 e a máquina virtual para o sistema B1700, indicaram ser esta última uma a cinco vezes mais rápida. Considerando a diferença de porte dos dois sistemas, parece razoável supor que a máquina virtual será no mínimo dez vezes mais rápida que uma versão SIL para o sistema B1700.

Dependendo do contexto no qual a linguagem SNOBOL é empregada, pode-se justificar o esforço requerido para uma implementação completa da máquina virtual. Sem dúvida, o sistema todo é muito complexo e as dificuldades práticas de interação com o sistema operacional, em esquema de multiprogramação, certamente não serão poucas.

Acreditamos que uma implementação da máquina SNOBOL deve evoluir da implementação de um subconjunto da linguagem. As dificuldades desta tarefa e a eficiência do sistema obtido darão a palavra final para o sistema completo SNOBOL.

BIBLIOGRAFIA

- [Agrawala e Rauscher 1976] Agrawala, A.K. e Rauscher, T.G. ,
"Foundations of Microprogramming - Architecture, Software
and Applications", ACM Monograph Series, Academic Press,
New York, 1976
- [Bashkow, Sasson e Kornfeld 1967] Bashkow, T.R., Sasson, A. e
Kornfeld, A., "System Design of a FORTRAN Machine", IEEE
Transactions on Electronic Computers, vol. 16, n.º.4, pg.
485-499, Abril 1967
- [Bell 1973] Bell, J.R., "Threaded Code", Communications of the
ACM, vol. 16, n.º.6, pg. 370-372, Junho 1973.
- [Berndt 1975] Berndt, W., "An Analysis of the SPITBOL System",
M. Sc. thesis, Department of Computer Science, University
of Toronto, 1975
- [Burroughs 1972] Burroughs B1700 Systems, "Reference Manual",
Burroughs Corporation, form 1057155, 1972
- [Burroughs 1973] Burroughs B1700 Systems, "Micro Implementation
Language (MIL)", Burroughs Corporation, form 1072568 ,
1973
- [Burroughs 1975] Burroughs B1700 Systems, "Master Control Pro
gram (MCP)", Burroughs Corporation, form 1088010, 1975
- [Dewar 1975] Dewar, R.B.K., "Indirect Threaded Code", Commu
nications of the ACM, vol. 18, no.6, pg. 330-331, Junho
1975
- [Dewar e Mccann 1977] Dewar, R.B.K. e Mccann, A.P., "MACRO
SPITBOL - a SNOBOL Compiler", Software - Practice and Ex
perience, vol. 7, pg. 95-113, 1977.
- [Foster 1972] Foster, C.C., "The Next Three Generations", Com
puter, vol. 5, no. 3, pg. 39-42, Março 1972
- [Gimpel 1973] Gimpel, J.F., "A Theory of Discrete Patterns
and Their Implementation in SNOBOL4", Communications of
the ACM, vol. 16, no. 2, pg. 91-100, Fevereiro 1973
- [Goldberg 1974] Goldberg, R.P., "Survey of Virtual Machine Re
search" Computer, vol. 7, no. 6, pg. 34-43, Junho 1974
- [Griss e Swanson 1977] Griss, M.L. e Swanson, M.R., "MBALM/1700 :
A Microprogrammed LISP Machine for the Burroughs B1726",
MICRO10 Proceedings - Tenth Annual Workshop on Micropro
gramming, Niagara Falls, New York, Outubro 1977

- [Griswold 1972] Griswold, R.E., "The Macro Implementation of SNOBOL4 - A Case Study of Machine Independent Software Development", W.H. Freeman and Company, San Francisco , 1972
- [Griswold, Poage e Polonsky 1971] Griswold, R.E., Poage, J.F. e Polonsky, I.P., "The SNOBOL4 Programming Language", segunda edição, Prentice-Hall, Englewood Cliffs, 1971
- [Harrison 1973] Harrison, M.C., "Data Structures and Programming", Scott, Foresman and Company, Glenview, 1973
- [Pasko 1973] Pasko, H.J., "A Pseudo-machine for Code Generation", M. Sc. thesis, Department of Computer Science, University of Toronto, 1973
- [Rosin 1974] Rosin, R.F., "The Significance of Microprogramming", International Computing Symposium, Davos, Suíça , American Elsevier, New York, 1973
- [Santos 1971] Santos, P.J.Jr., "FASBOL, A SNOBOL Compiler", Ph. D. thesis, Electronics Research Laboratory, University of California, Berkeley, 1971
- [Shapiro 1972] Shapiro, M.D., "A SNOBOL Machine: Functional Architectural Concepts of a String Processor", Ph.D. thesis, Department of Computer Science, Purdue University , 1972
- [Tanenbaum 1973] Tanenbaum, A.S., "Design and Implementation of an ALGOL 68 Virtual Machine", Mathematisch Centrum , Amsterdam, Abril 1973
- [Tanenbaum 1976] Tanenbaum, A.S., "Structured Computer Organization", Prentice-Hall, Englewood Cliffs, 1976
- [Varian 1972] Varian 70 Series, "Varian 73 System Handbook", Varian Data Machines, form 98A9906010, 1972
- [Varian 1974] Varian 70 Series, "V70 Microprogramming User's Manual", Varian Data Machines form 96A9906073, 1974
- [Weber 1967] Weber, H., "A Microprogrammed Implementation of EULER on IBM System 360/30", Communications of the ACM , vol. 10, no. 9, pg. 549-558, Setembro 1967
- [Wilkes 1951] Wilkes, M.V., "The Best Way to Design an Automatic Calculating Machine", Manchester University Computer Inaugural Conference, pg. 16-21, London 1951
- [Wilner 1972] Wilner, W.T., "Burroughs B1700 Memory Utilization", Proceedings of the 1972 Fall Joint Computer Conference, Anaheim, California, 1972

APÊNDICE A

RESUMO DE REGISTRADORES DO PROCESSADOR B1700

Os registradores estão classificados em seis grupos de características específicas e relacionados em esquemas que definem os nomes de seus campos. Entre parênteses aparece o número de bits de cada campo.

1. REGISTRADORES ATIVOS

Normalmente têm função de controle no processador. Servem tanto como origem quanto destino em microinstruções que envolvem movimento de informações.

M
microinstrução corrente (16)

Se utilizado como destino, a informação movida sofrerá um "ou lógico" com a próxima microinstrução que for executada.

A
endereço da próxima microinstrução (18)

Pode endereçar até 16384 microinstruções localizadas na memória de controle ou na memória principal. A é incrementado antes da execução da microinstrução atual.

TOPM
topo da memória de controle (4)

Determina se a próxima microinstrução está na memória de controle ou memória principal. Caso $A > 512 * TOPM$ a próxima microinstrução está na memória principal.

MBR
base do microcódigo na memória principal (24)

gistrador A somado ao valor do registrador MBR.

Especifica o endereço acima do qual estão armazenadas as microinstruções na memória principal. O endereço efetivo na memória principal é dado pelo valor do re

BR, LR
base e limite da memória principal (24)

se o conteúdo do registrador utilizado em microinstruções na memória principal. Registradores que contêm os endereços base e limite da memória principal, para a tarefa em andamento. São utilizados pelos circuitos do sistema para verificar se o endereço válido quando LER/GRAVAR e PERMUTAR informações

TAS	
topo da pilha de endereços	(24)

A pilha de endereços tem 32 elementos. TAS é o registrador do topo. As microinstruções CHAMAR e MOVER empilham e desempilham endereços ou outras informações.

X,Y	
entradas da unidade lógica e aritmética	(24)

Registradores operandos da unidade lógica e aritmética. Podem sofrer rotação ou deslocamento de bits, bem como receber e enviar dados entre a memória principal.

L					
LA	LB	LC	LD	LE	LF
registrador local					
(24)					

Uso geral. Seus campos podem ser operandos da unidade lógica e aritmética de quatro bits. Pode receber e enviar dados entre a memória principal.

T					
TA	TB	TC	TD	TE	TF
registrador de transformação					
(24)					

Uso geral. Seus campos podem ser operandos da unidade de quatro bits. Pode receber e enviar dados entre a memória principal. Pode sofrer rotação, deslocamento e extração de bits.

C						
				CP		
CA	CB	CC	CD	CYF	CPU	CPL
uso geral (8)	interrupções (8)	vai um (1)	tipo (2)	comprimento (5)		

Controla o processador. CA, CB, CC e CD têm quatro bits cada um. CYF é a chave de "vai um" e é entrada da unidade de 24 bits ao lado de X e Y. CPU controla a aritmética (binária, dígitos ou caracteres). CPL controla o comprimento dos operandos da unidade de 24 bits.

F						
FA		FB				
		FU	FT	FL		
				FLC	FLD	FLF
endereço absoluto (24)		unidade de conteúdo (4)	tipo de conteúdo (4)	comprimento (16)		

Especifica endereços e comprimentos de campos na memória principal. FA indica um endereço de bit. FU e FT tem função descritiva, a unidade e o tipo do conteúdo do campo em geral é fornecido pelas microinstruções. FU pode ser utilizado na microinstrução POSICIONAR CP. FL indica o comprimento do campo. FLC, FLD, FLE e FLF têm quatro bits cada.

2. REGISTRADORES DE RESULTADOS

Armazenam resultados da unidade lógica e aritmética de 24 bits. Entradas da unidade são X, Y e CYF que são processados sob controle de CPU e CPL. Uma alteração em X, Y, CYF, CPU ou CPL produz todos os resultados novamente. Registradores de resultados só funcionam como origem.

SUM
soma de X, Y e CYF (24)

Soma binária ou decimal (em dígitos ou caracteres). Se a soma ultrapassa o comprimento dado através de CPL, então a chave CYL é ligada.

DIFF
diferença $X - (Y + CYF)$ (24)

Diferença binária ou decimal (em dígitos ou caracteres). Se o resultado é negativo então DIFF armazena o complemento de 2 ou 10 e a chave CYD será ligada.

XORY, XEOY, XANY
resultado de operação lógica (24)

"Ou inclusivo", "ou exclusivo" ou "e lógico" entre os conteúdos de X e Y. O número de bits usados na operação é controlado por CPL.

CMPX, CMPY
complemento de um de X ou Y (24)

CPL bits são usados na complementação. Bits mais significativos, não usados, serão zerados.

MSKX, MSKY
X ou Y mascarado por CPL (24)

CPL bits de mais baixa ordem de X ou Y. Bits à esquerda, não afetados, serão zerados.

3. REGISTRADORES TEMPORÁRIOS

Conjunto de 16 registradores endereçáveis como campos de 48 bits (S00 a S15) ou 32 campos de 24 bits (S00A a S15A e S00B a S15B). Podem ser origem ou destino.

S15A	(24)	S15B	(24)
:		:	
S01A	(24)	S01B	(24)
S00A	(24)	SFU	SFL
		(4)	(16)

FL e SFL são continuamente comparados entre si e a relação entre eles (maior, igual, menor ou zero) automaticamente registrada em FLCN. SFU e SFL podem ser utilizados para alterar CPU e CPL na microinstrução POSICIONAR CP.

4. REGISTRADORES CONSTANTES

MAXS
total de memória principal (24)

Número de bits instalados. Pode ser utilizado apenas como origem.

MAXM
total de memória de controle (24)

Número de palavras de 16 bits instaladas. Pode ser utilizado apenas como origem.

NULL
todos os bits nulos (24)

Usado para movimentar bits nulos para outros registradores. Origem apenas.

5. REGISTRADORES DE ENTRADA E SAÍDA

U
entrada de fita cassete (16)

Armazena microinstruções lidas de fita cassete. Origem somente.

CMND, DATA
comandos e dados de controladores de E/S (24)

Registradores de comunicação com os controladores de entrada e saída. CMND é destino somente.

6. REGISTRADORES DE CONDIÇÕES

Há cinco registradores de condições. Cada um tem quatro bits e pode ser usado somente como origem. São referenciadas em microinstruções de teste de bits. A seguir é dada uma descrição sumária da função de cada bit.

BICN			
LSUY		CYD	CYL
menor unidade de Y	CYF	$X - (Y + CYF) < 0$	$X + Y + CYF > SUM$

BICN é registrador de condições binárias. LSUY vale um se o bit de menor ordem de Y valer um e CPU for zero ou o

dígito de menor ordem de Y valer nove e CPU for igual a 1 ou igual a 3. Vale zero em caso contrário.

XYCN			
MSBX			
bit dado por CPL	$X = Y$	$X < Y$	$X > Y$

XYCN é registrador de condições entre X e Y. As condições independem de CPL, os 24 bits são sempre comparados.

XYST			
LSUX	INT		
menor unidade de $\bar{X}$	alguma interrupção	$Y \neq 0$	$X \neq 0$

XYST é registrador dos estados de X e Y. LSUX tem definição análoga a LSUY. As comparações independem de CPL.

FLCN			
$FL = SFL$	$FL > SFL$	$FL < SFL$	$FL \neq 0$

FLCN é registrador de condições entre FL e SFL.

INCN			
unidade ausente	prioridade	existe interrupção	interrup. em uso

INCN registra as interrupções que necessitam tratamento.

APÊNDICE B

MICROINSTRUÇÕES DO PROCESSADOR B1700

As microinstruções tem sempre 16 bits de comprimento. Na descrição que segue foram atribuídos nomes em português a cada microinstrução; entre parênteses à direita comparecem os mnemônicos correspondentes em MIL. A descrição é resumida, evitando-se detalhar por completo cada microinstrução; também foram omitidos os comprimentos dos campos e os códigos que compõem a instrução. A ordem de apresentação dos esquemas reúne em geral microinstruções com características parecidas precedidas de um breve comentário.

Microinstruções de movimento de informações normalmente permitem mover informações entre registradores de tamanhos diferentes com truncamento ou preenchimento de bits a esquerda. Registradores temporários serão referidos neste contexto como "temporários" e têm tratamento diferenciado dos demais. Registradores de condições, de resultados e constantes não podem ser destino.

MOVER REGISTRADOR	registrador origem	registrador destino	(MOVE)
MOVER LIT. DE 8 BITS	registrador destino	literal	(LIT, MOVE)
MOVER LIT. DE 24 BITS	registrador destino	literal	(LIT, MOVE)
MOVER TEMPORÁRIO	registrador orig./dest.	de/para temporário orig./dest.	(MOVE)
CARREGAR F DE DU- PLO TEMPORÁRIO	temporário		(LOAD F)
ARMAZENAR F EM DU- PLO TEMPORÁRIO	temporário		(STORE F)
PERMUTAR F E DUPLO TEMP.	temporário destino	temporário origem	(XCH)

AGREGAR TEMPORÁRIO A FA	somar/ subtrair	temporário esquerdo
-------------------------	--------------------	------------------------

(ADD/SUBTRACT
SCRATCHPAD)

Referências a campos na memória principal em geral exigem, ao nível de instruções de máquina da linguagem, iteração sobre o comprimento e endereço do campo. Esta situação é tratada diretamente pelas duas microinstruções seguintes.

COMPUTAR FA/FL	opção	literal
-------------------	-------	---------

(COUNT FA/FL)

Esta microinstrução incrementa e/ou decrementa FA e/ou FL pelo valor do literal. As opções são: FA↑, FL↑, FA↑ e FL↓, FA↓ e FL↑, FA↓, FL↓, FA↓ e FL↓, não incrementar. Caso o literal seja nulo, CPL será utilizado em seu lugar.

LER/GRAVAR MEMÓRIA	1 g	opção X,Y,T,L	direção do campo	comprimento do campo
-----------------------	--------	---------------	---------------------	-------------------------

(READ,
WRITE)

Neste caso o endereço do primeiro bit do campo a ser lido ou gravado deve estar em FA. Caso o comprimento do campo seja nulo então CPL é usado em seu lugar. Paralelamente pode ser feito o incremento/decremento de FA/FL de modo idêntico à microinstrução COMPUTAR FA/FL.

PERMUTAR MEMÓRIA	registrador X,Y,T,L	direção do campo	comprimento do campo
---------------------	------------------------	---------------------	-------------------------

(SWAP)

O endereço do campo também é dado por FA. Se o comprimento indicado é nulo então CPL é usado em seu lugar. A microinstrução permuta o conteúdo do registrador com o conteúdo do campo indicado.

As microinstruções que produzem deslocamento de bits, truncam bits à esquerda/direita e preenchem com bits nulos o campo livre à direita/esquerda.

NORMALIZAR X

(NORMALIZE)

Desloca o registrador X para a esquerda decrementando o registrador FL até ele valer zero ou o bit em X referenciado pelo registrador CPL valer um.

DESLOCAR/RO- DAR X E Y	deslôcar/ro- dar	esquerda/di- reita	número de bits
---------------------------	---------------------	-----------------------	-------------------

(ROTATE
SHIFT
XY)

Desloca ou roda X e Y como se fosse um registrador único de 48 bits. Se o número de bits a deslocar/rodar é nulo, CPU determina a quantidade.

DESLOCAR/RO- DAR X OU Y	deslocar/ro- dar	esquerda/di- reita	X/Y número de bits
----------------------------	---------------------	-----------------------	-----------------------

(ROTATE
SHIFT
X,Y)

DESLOCAR/RO- DAR T ESQ.	registrador destino	deslocar/ro- dar	número de bits
----------------------------	------------------------	---------------------	-------------------

(ROTATE
SHIFT
T)

Desloca ou roda os bits de T e em seguida os move para o registrador de destino. Se o número de bits é nulo, CPL é utilizado em seu lugar. T não é alterado a menos que seja também registrador de destino.

EXTRAIR BITS DE T	no. de bits a rodar	reg. destino X,Y,T ou L	no. de bits a extrair
----------------------	------------------------	----------------------------	--------------------------

(EXTRACT)

Roda os bits de T, extrai os bits à direita e os move para o registrador destino. Se o número de bits a extrair for nulo, CPL é utilizado em seu lugar. T não é alterado a menos que seja também o registrador de destino.

A manipulação dos registradores que controlam a unidade lógica e aritmética de 24 bits é efetuada principalmente por duas microinstruções:

POSICIONAR CYF	igualar a .0, 1, CYL ou CYD
-------------------	--------------------------------

(CARRY)

POSICIONAR CP	opção	chave para teste
------------------	-------	---------------------

(BIAS)

Esta última atribui o valor 1, 3 ou 0 a CPU se FU ou SFU valer 4, 8 ou outro valor qualquer, respectivamente. Atribui também valor a CPL. Ambos de acordo com a opção especificada na microinstrução. Se a chave de teste valer 1 e o valor de CPL ao final da execução da microinstrução for zero, então a próxima microinstrução será saltada. Esta instrução é indicada no caso de iterações sobre campos de dados. As opções são:

<u>opção</u>	<u>CPU</u>	<u>CPL</u>
0	FU	FU
1	FU	min(24,FL)
2	SFU	min(24,SFL)
3	FU	min(24,FL,SFL)
4	-	-
5	FU	min(24,CPL,FL)
6	-	-
7	FU	min(24,CPL,FL,SFL)

Microinstruções que permitem fazer testes e desvios no microcódigo sempre referenciam endereços relativos. As duas primeiras microinstruções seguintes são particulares para manipular registradores de quatro bits e têm opções para pular a microinstrução seguinte. Novamente, registradores para condições não podem ser alterados.

MANIPULAR 4 BITS	registrador de 4 bits	opção	literal
---------------------	--------------------------	-------	---------

(AND,OR,EOR,SET,RESET,COMPLEMENT,DEC,INC)

Em função da opção pode-se: igualar o registrador ao literal, fazer operações de AND, OR e EOR do literal sobre o registrador, incrementar ou decrementar o registrador detectando "overflow" ou "underflow", e em caso afirmativo pular a próxima microinstrução.

SALTAR MICRO INSTRUÇÃO SE	registrador de 4 bits	opção	máscara	(SKIP, IF)
------------------------------	--------------------------	-------	---------	---------------

Pula a próxima microinstrução em função da opção:

- se algum dos bits referenciados for 1
- se todos os bits referenciados forem 1 (e que, opcionalmente podem ser zerados)
- se o registrador é igual à máscara.

Não pula a próxima microinstrução em função da opção:

- se algum dos bits referenciados for 1
- se todos os bits referenciados forem 1 (e que, opcionalmente podem ser zerados)
- se o registrador é igual à máscara.

TESTAR BIT, DES- VIAR SE LIGADO	registrador de 4 bits	número do bit	sinal do desvio	desvio	(IF)
------------------------------------	--------------------------	------------------	--------------------	--------	------

TESTAR BIT, DES- VIAR SE DESLIG.	registrador de 4 bits	número do bit	sinal do desvio	desvio	(IF)
-------------------------------------	--------------------------	------------------	--------------------	--------	------

As microinstruções anteriores testam o bit indicado. O desvio é relativo ao endereço da microinstrução e é dado em número de palavras de 16 bits, até um máximo de 15 palavras.

DESVIAR RELATIVAMENTE	sinal do desvio	desvio	(JUMP, GO TO)
CHAMAR	sinal do desvio	desvio	(CALL)

O desvio é dado em número de palavras de 16 bits até um máximo de 4095. A microinstrução CHAMAR tem a particularidade adicional de empilhar o endereço da próxima microinstrução na pilha de endereços e em seguida desviar relativamente. O retorno é feito com a microinstrução MOVER REGISTRADOR TAS para A.

As microinstruções seguintes completam o repertório do sistema B1700.

ZERAR REGISTRADORES	chaves para X Y T L FA FL FU CP	(CLEAR)
OPERAÇÃO NULA		(NOP)
PARAR EXECUÇÃO		(HALT)
CARREGAR MEMÓ- RIA DE CONTROLE		(OVERLAY)
COMUNICAR OPE- RAÇÃO DE E/S		(DISPATCH)
CONTROLAR FITA CASSETTE	opção	(CASSETTE)
MONITORAR		(MONITOR)