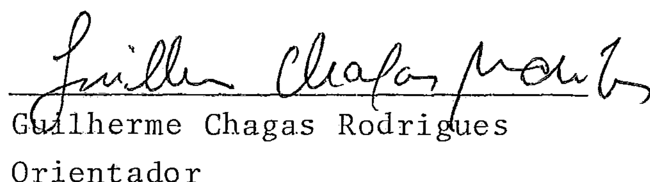
UM COMPILADOR BASIC INCREMENTAL
PARA O TERMINAL INTELIGENTE

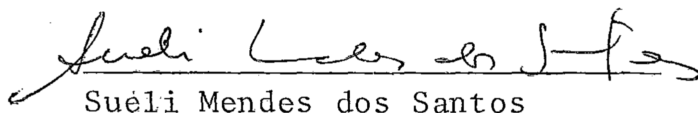
Eliane Martins

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.Sc.).

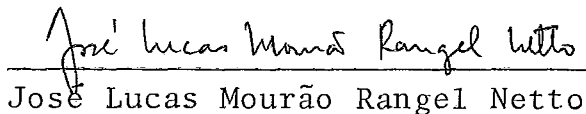
Aprovada por:



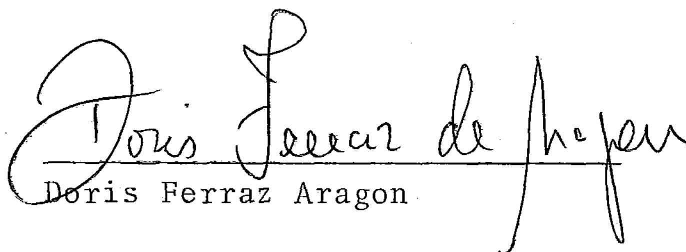
Guilherme Chagas Rodrigues
Orientador



Sueli Mendes dos Santos



José Lucas Mourão Rangel Netto



Doris Ferraz Aragon

RIO DE JANEIRO, RJ - BRASIL
MARÇO DE 1982

MARTINS, ELIANE

Um Compilador Basic Incremental para o Terminal Inteligente - (Rio de Janeiro) 1982

VIII, 155 p. 29.7 cm (COPPE-UFRJ, M. Sc., Engenharia de Sistemas e Computação.

Tese - Univ. Fed. do Rio de Janeiro. Fac. Engenharia

1. Compilador

I. COPPE/UFRJ

II. Título (Série)

À minha mãe

RESUMO

Em um compilador incremental, a análise sintática é feita de modo que o programa não precise ser totalmente recompilado após cada modificação.

Neste trabalho é apresentado um compilador incremental para a linguagem BASIC, a ser implementado no Terminal Inteligente desenvolvido no NCE da UFRJ. O objetivo principal é permitir o uso do referido equipamento por usuários que estejam aprendendo computação.

ABSTRACT

An incremental compiler is a device which is able to perform syntax analysis in an incremental way, avoiding complete reparsing of a program after each modification.

This work presents an incremental compiler for the BASIC language, to be implemented in the intelligent terminal developped by the Nucleus of Electronic Computing of the UFRJ. The project is oriented for users that are learning about computing.

ÍNDICE

Pág.

PARTE I - DESCRIÇÃO DO PROJETO........................ 1

I. INTRODUÇÃO.................................... 1

I.1 - O Compilador Incremental e Suas Vantagens........ 1
I.2 - Recursos Disponíveis.............................. 3
I.3 - Porque o BASIC Incremental........................ 4
I.4 - Características da Linguagem...................... 4
I.5 - O Sistema, Visto pelo Usuário..................... 6

PARTE II - GERAÇÃO DO SISTEMA.......................... 12

I. ESTRUTURA DO COMPILADOR INCREMENTAL................. 13

I.1 - Características de um Compilador Incremental..... 13
I.2 - Componentes....................................... 16
I.3 - Estruturas de Dados............................... 22
I.4 - Organização da memória............................ 42
I.5 - Organização de Arquivos........................... 45
I.6 - Resumo do Funcionamento........................... 48

II. O PROCESSADOR DO CÓDIGO GERADO.................... 52

Pág.

III. O COMPILADOR BASIC.............................. 55

III.1 - Descrição Geral.............................. 55
III.2 - Análise Léxica.............................. 56
III.3 - Análise Sintática.............................. 61
III.4 - Análise Semântica e Geração de Código.......... 91

IV. A EXECUÇÃO.............................. 111

IV.1 - Erros.............................. 112
IV.2 - Alocação de Variáveis.............................. 114
IV.3 - Pilha de Blocos.............................. 123
IV.4 - Passagem de Parâmetros.............................. 126
IV.5 - Chamada de Segmento.............................. 128
IV.6 - Saída de Segmento.............................. 130
IV.7 - Tratamento de Interrupção.............................. 130
IV.8 - Temporárias.............................. 132

V. PASSO 2.............................. 135

VI. PROCESSAMENTO DE COMANDOS IMEDIATOS................ 138

VII. CARGA DE PROGRAMAS.............................. 139

VIII. O EDITOR.............................. 141

Pág.

IX. DISCUSSÃO E CONCLUSÕES 144

BIBLIOGRAFIA 147

APÊNDICE A - AUTÔMATO LÉXICO 149

APÊNDICE B - COMANDOS DO SISTEMA 152

APÊNDICE C - PALAVRAS RESERVADAS 155

APÊNDICE D - MICRO-ROTINAS DO PLTI 156

I. INTRODUÇÃO

I.1 - O Compilador Incremental e suas Vantagens

Um compilador incremental é interativo. A comunicação com o usuário é feita tanto na fase de compilação quanto na de execução.

O principal aspecto do trabalho é o uso de incrementalismo. Com esta técnica, o programa fonte é considerado como um conjunto de linhas independentes entre si. Desse modo, se o usuário fizer qualquer correção ou alteração em uma linha, o programa não precisará ser recompilado: somente a linha alterada é re-analisada.

As vantagens na utilização desse método estão resumidas a seguir:

1. Edição interativa

O processo de desenvolvimento de um programa é constituído por várias alternâncias entre execuções e edições para corrigir os erros encontrados. No presente caso, o usuário poderá interromper a execução se o programa não estiver funcionando direito, corrigi-lo e em seguida continuar a execução.

2. Erros são detetados imediatamente

Quando uma linha errada é fornecida, o usuário é notificado, podendo corrigir o erro imediatamente.

3. Comandos imediatos

Os comandos imediatos são executados no momento em que são fornecidos. São um subconjunto dos comandos do BASIC, e a diferença entre uma linha de um programa e uma linha contendo comando imediato é que a primeira é numerada e a segunda não.

A grande vantagem desses comandos é que eles permitem depuração interativa. Após (ou durante) uma execução do programa em que haja erros, o usuário poderá obter ou alterar o conteúdo de variáveis entre outras.

4. Facilidade de uso

Devido às vantagens já citadas, o uso do computador é facilitado para usuário sem conhecimentos de computação.

Este projeto está baseado no trabalho de M. Berthaud e M. Griffiths, cuja descrição pode ser encontrada nos itens (1) e (2) da bibliografia.

I.2 - Recursos Disponíveis

A implementação se dará no Terminal Inteligente (T.I) desenvolvido pelo Núcleo de Computação Eletrônica da UFRJ, cuja configuração atual é a seguinte: teclado, vídeo, UCP com até 64 K bytes de memória, uma unidade de disco, leitora de cartões e impressora.

O compilador rodará sob o SOCO, o atual sistema operacional em disco do T.I. que contem, entre outras facilidades:

- o método de acesso sequencial indexado;

- um compilador PLTI, que é uma linguagem de alto nível baseada no PL/I, desenhada, entre outros motivos, para a implantação do "software" do T.I.;

- um interpretador, que analisa as instruções de uma linguagem de u'a máquina virtual, desenvolvida para que haja a maior independência possível entre os sistemas desenvolvidos e a máquina. Desse modo, caso haja mudanças no processador, basta alterar esse interpretador, permanecendo o restante dos sistemas inalterados;

- o uso de certas chaves no teclado do terminal para a interrupção de programas para a depuração interativa.

I.3 - Porque o BASIC Incremental

Foi escolhida essa linguagem pois pretende-se que o T.I., seja utilizado para fins educacionais, ou seja, serão atendidos usuários com pouca ou nenhuma experiência em computação.

O BASIC é uma linguagem simples, fácil de aprender e voltada para sistemas interativos.

O método incremental, além das vantagens citadas em (I.1), vai evitar que haja dois níveis de interpretação. A saída do compilador incremental será constituída pelas micro-rotinas do PLTI, acrescidas de algumas rotinas que terão, entre outras finalidades, a tarefa de tornar conversacional a fase de interpretação.

I.4 - Características Gerais da Linguagem

O BASIC a ser implementado tem como principais características:

- permitir a utilização de 3 tipos de dados: inteiros, reais e alfanuméricos;

- contem comandos e funções que manipulem os 3 tipos de dados descritos acima;

- os identificadores poderão ter até 6 caracteres;

- permite a utilização de variáveis subscritas de 1 ou 2 dimensões, sendo que se os limites forem 10 ou 10 x 10, a variável não precisará ser declarada;

- as variáveis e constantes inteiras podem ser usadas como operandos lógicos;

- disponibilidade de comandos de E/S, formatada ou não, permitindo acesso ao vídeo, teclado e disco do sistema;

- permite que o usuário defina funções com mais de uma linha;

- permite ao usuário definir rotinas especiais que admitam a passagem de parâmetros e sejam chamadas por CALL. As subrotinas comuns do BASIC também estarão disponíveis;

- dispõe de comandos IF-THEN-ELSE e comandos iterativos FOR-TO, FOR-WHILE e FOR-UNTIL;

- os espaços não serão transparentes;

- os números de linha são obrigatórios, pois darão a ordem de execução do programa;

- pode ser fornecido mais de um comando por linha, admitindo até 3 linhas de continuação;

- permite a inserção de comentários no programa, o que ajuda ao usuário na documentação do programa.

A descrição completa da linguagem está no diagrama sintático, no capítulo III.3 da Parte II.

I.5 - O Sistema, Visto pelo Usuário

Um programa será constituído por segmentos. Um segmento poderá conter uma função, subrotina ou o programa principal.

O formato de um segmento é mostrado a seguir:

<u>segmento</u> <nome do segmento>

⋮ conteúdo

<u>fim</u>

O <nome do segmento> é dado por um identificador. O seu conteúdo é um conjunto de linhas numeradas contendo um ou mais incrementos. Um incremento, do mesmo modo que no trabalho desenvolvido por Berthaud e Griffiths, pode ser um comando ou declaração do BASIC ou parte de um comando. Como um incremento pode ser fornecido em qualquer ordem, o número da linha vai indicar a sequência de execução dentro do segmento.

A análise de cada linha é feita no momento em que a mesma é fornecida ao sistema. Caso haja algum erro, a mensagem correspondente é emitida e aguardar-se-á a correção do usuário.

O código gerado para cada segmento vai sendo guardado em disco. No momento da execução, todo o programa é carregado na memória. Quando ocorrer uma interrupção, os segmentos serão copiados de volta para o disco.

Não será permitida a definição de funções ou subrotinas embutidas, e nem haverá recursividade. Todos os identificadores serão considerados locais ao segmento que os contém; as ligações entre as variáveis dos diversos segmentos se dará através da passagem de parâmetros.

A interação entre o usuário e o sistema se dará através dos seguintes tipos de comandos:

Comandos de controle

São comandos que indicam ao sistema que devem ser executadas tarefas do tipo: salvar um programa, apagar um programa que havia sido salvo anteriormente, condensar a área objeto, de modo a obter melhor aproveitamento do espaço, etc.

Comandos de edição

Permitem ao usuário manipular com os segmentos que foram fornecidos, ou seja, apagar uma ou várias linhas, listar trechos ou todo um segmento, renumerar as linhas do segmento.

Comandos imediatos

São um subconjunto dos comandos do BASIC que facilitarão ao usuário na depuração de seus programas, podendo também ser chamados de comandos de depuração. Permitem listar ou alterar o conteúdo da área de dados, efetuar testes com esses valores, etc.

A execução de um segmento pode ser suspensa nos seguintes casos:

a) foi encontrado um comando para leitura conversacional;

b) a tela está cheia;

c) foi executado um STOP;

d) foi pressionada a chave de interrupção no teclado;

e) foi encontrado um erro.

Em caso de leitura conversacional, a execução é suspensa até que o usuário forneça os dados necessários para continuar a execução.

No caso de tela cheia, a execução é interrompida pois nenhuma linha mais poderá ser visualizada. Para continuar, basta que o usuário aperte a tecla VALIDADE.

Nos casos c), d) e e) o sistema fica aguardando o próximo passo do usuário. Este pode fornecer novos comandos da linguagem, ou comandos de edição ou ainda comandos imediatos. Após executar cada comando de um dos tipos descritos acima, o sistema continua aguardando até que o usuário resolva continuar a execução do ponto em que ela foi interrompida ou de outro ponto qualquer dentro do segmento. Em caso de erro grave, como por exemplo, desvio para uma instrução inexistente, o programa é cancelado.

Modos de Operação

O sistema poderá estar sempre em um dos seguintes modos de operação:

Modo de Controle

É o modo básico de operação; a partir dele o usuário poderá se utilizar dos comandos de controle, podendo ainda passar para o modo de entrada ou de execução.

Modo de entrada

O sistema estará nesse modo quando for dado o comando SEGMENTO. O usuário poderá então fornecer um novo segmento, inserir novos incrementos, editar um segmento ou ainda fornecer comandos imediatos. Seu término se dará com o comando FIMSEG, fazendo com que o sistema volte ao modo de controle.

Modo de execução

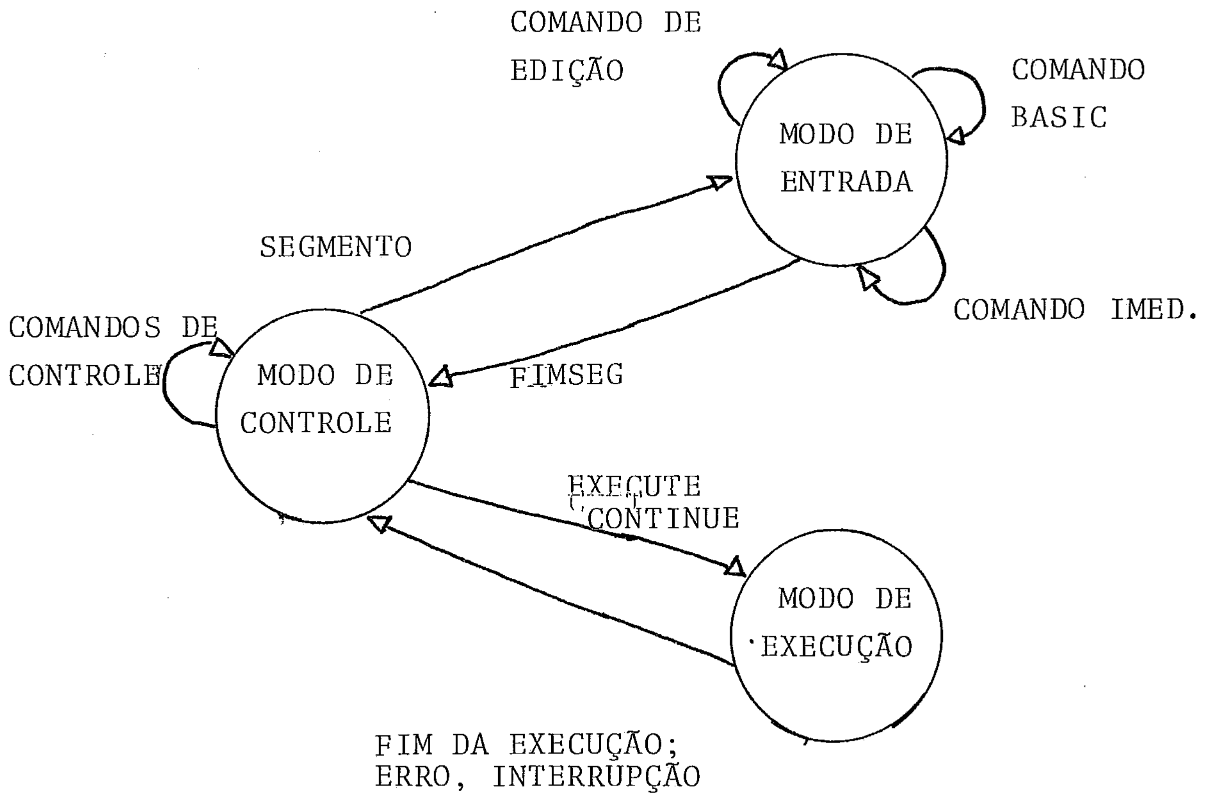
Iniciado através do comando EXECUTE ou CONT. Este último indica que havia algum segmento sendo executado. Uma vez em execução, o sistema ficará nesse modo até que:

1. haja alguma interrupção por chave do teclado;

2. seja executado um STOP;

3. seja encontrado erro ou acabe o programa.

O diagrama seguinte mostra como é feita a ligação entre os 3 modos.


COMANDO DE
EDIÇÃO
COMANDO
BASIC
MODO DE
ENTRADA
SEGMENTO
COMANDO IMED.
COMANDOS DE
CONTROLE
MODO DE
CONTROLE
FIMSEG
EXECUTE
CONTINUE
MODO DE
EXECUÇÃO
FIM DA EXECUÇÃO;
ERRO, INTERRUPÇÃO

PARTE II: GERAÇÃO DO SISTEMA

I. ESTRUTURA DO COMPILADOR INCREMENTAL

I.1 - Características de um Compilador Incremental

- Análise linha por linha

Dentre as principais vantagens da compilação interativa temos a deteção imediata de erros e a edição interativa. Em outras palavras, se o usuário fornecer uma linha com erro de sintaxe, o compilador lhe comunicará imediatamente o erro, podendo este ser logo corrigido.

Devido ao fato acima, a análise de um programa em um compilador incremental será feita linha por linha. Uma linha é analisada, os erros são detetados e para as linhas corretas é feita a geração do código intermediário. Tanto a linha fonte quanto o código gerado são acrescentados ao restante do programa.

A unidade mínima de comunicação entre o usuário e o compilador será portanto uma linha. Uma linha fonte pode conter vários comandos BASIC, bem como um comando pode ocupar várias linhas. Neste caso, o usuário deve teclar uma chave especial, indicando que há continuação. Será considerado o fim quando for teclada a chave indicando o fim da linha.

- Alterações não obrigam a recompilação do programa

Em um compilador incremental as linhas do programa fonte são consideradas independentes umas das outras. Neste

caso, quando o usuário fizer qualquer alteração no programa, este não precisará ser inteiramente recompilado. Somente a(s) linha(s) alterada(s) é(são) re-analisada(s).

- Gera micro-rotinas do PLTI

Não haverá nenhuma espécie de código intermediário (ou linguagem interna). Para cada linha correta serão geradas as micro-rotinas do PLTI que corresponderão ao(s) comando(s) do BASIC que a linha contem.

- Depuração interativa

O usuário contará com comandos imediatos para ajudar a depurar seus programas. Quase todos os comandos do BASIC poderão ser usados em modo imediato (alguns não têm sentido nesse modo, e portanto não serão permitidos, como por exemplo, as declarações).

Os comandos imediatos podem ser considerados como um programa de uma linha, seguido de um STOP. São executados assim que sua análise é completada.

- Edição interativa

A facilidade de edição vai permitir ao usuário: apagar uma ou mais linhas de um segmento; listar trechos de segmentos; re-sequenciar as linhas de um segmento.

A linha será também a unidade mínima de programa fonte a ser manuseada pelo editor. Quando uma linha for apagada, ela será re-analisada, de modo a atualizar certas informações do compilador.

- <u>Tipos de arquivos: programas e dados do usuário</u>

Os dispositivos de entrada e saída considerados no presente trabalho serão o teclado e vídeo do terminal e uma unidade de disco.

Através do teclado o usuário poderá fornecer as linhas de um programa BASIC, comandos do compilador ou comandos imediatos, bem como os dados a serem lidos conversacionalmente pelo seu programa.

No vídeo serão mostradas: as linhas correntes, as mensagens de erro, ou linhas criadas pelo programa do usuário.

Em disco estarão armazenados:

- os programas do usuário (o fonte e o código correspondente);

- informações sobre o programa do usuário, obtidas e utilizadas pelo compilador, em forma de tabelas;

- rotinas que não estejam em uso: o compilador será desenvolvido utilizando-se técnicas de "overlay", devido à restrição de me-

mória: as rotinas menos utilizadas ficarão em disco, sendo trazidas para a memória quando forem necessárias;

- áreas temporárias e de trabalho: sua utilização será descrita no decorrer do texto;

- arquivos criados pelos programas do usuário;

Interrupção

A fase de execução deverá permitir interrupções. O compilador deverá testar continuamente (antes de iniciar a execução de qualquer novo comando) se ocorreu alguma das interrupções citadas em (I.5), na Parte I.

I.2 - Componentes do Sistema

O compilador será programado de forma modular. Cada módulo será constituído por rotinas responsáveis por tarefas específicas. A comunicação entre os diversos módulos se dará através de chamadas de rotinas.

Além de facilitar a implementação e futuras alterações, a divisão em módulos vai permitir o uso da técnica de "overlay", mencionado no item anterior.

Os componentes que constituem o compilador são os seguintes:

- analisador de comandos
- editor
- depurador
- compilador BASIC
- módulo de controle da execução
- módulo de controle principal
- carga de programas
- tratamento de erros

Em seguida será vista uma descrição resumida dos diversos componentes.

I.2.1 - Analisador de Comandos

É o componente responsável pela análise e interpretação dos comandos de controle. É constituído por uma rotina principal que aciona as rotinas correspondentes a cada comando. Será ativada sempre que o sistema não se encontrar em modo de entrada ou execução.

I.2.2 - Editor

Formado por um conjunto de rotinas que analisam e interpretam os comandos de edição. Haverá uma rotina para cada comando, que é acionada quando, em modo de entrada, é reconhecido um comando de edição.

I.2.3 - Compilador BASIC

Conjunto de rotinas responsáveis pela análise dos comandos do BASIC, bem como pela geração do pseudo-código.

I.2.4 - Carga de Programas

É chamado quando o usuário teclou o comando EXEC. Verifica os erros que não puderam ser detetados pela compilação e carrega na memória o programa a ser executado.

I.2.5 - Módulo de Controle da Execução

Esse módulo é constituído de várias rotinas que serão utilizadas em tempo de execução, quais sejam:

- Rotina de Atualização do Contador de Programas

O Contador de Programas (ou PC, da abreviatura do termo em inglês "programm counter") é um registro que deve conter sempre o endereço da próxima instrução a ser executada.

Essa rotina, no nosso caso, atualiza uma variável PC com o endereço do próximo incremento.

- Rotina de Entrada e Saída

Aciona as rotinas necessárias para a execução das operações de entrada e saída. Verifica a cadeia de formato, lê ou grava os dados de acordo com esse formato, verifica se a operação é válida para o arquivo, entre outras.

Testa se a operação foi terminada com sucesso.

- Rotina de Tratamento de Interrupção

Responsável pelas decisões a serem tomadas conforme o tipo de interrupção que ocorra durante a execução de um programa, como por exemplo:

i) se houve algum erro grave, a execução deve ser abortada;

ii) caso seja necessário uma leitura de dados, o controle é passado à Rotina de Entrada e Saída;

iii) caso o controle deva passar ao usuário devido a um STOP ou porque foi acionada alguma chave do teclado, o conteúdo atual da memória deve ser salvo para permitir que a execução possa continuar do ponto em que foi interrompida. O sistema passa então ao modo de controle.

- Rotina de Teste de Chave

Acionada após a execução de cada incremento.

Verifica se foi pressionada a chave do teclado que interrompe a execução. Em caso afirmativo, passa o controle à Rotina de Tratamento de Interrupção.

- Rotinas do FOR

Rotinas de controle de comando iterativo for. São responsáveis dentre outras tarefas, pelo incremento e teste da variável de controle.

- Rotina de Entrada de Blocos

Sempre que for chamada uma subrotina ou função de múltiplas linhas, bem como a entrada em um comando interativo, deve-se verificar se o segmento ou variável de controle correspondente ainda se encontra em uso, para evitar recursividade ou a atualização erronea da variável de controle de um for.

- Rotina de Inicialização de Variáveis

Antes de começar a execução de um segmento, as variáveis locais que são parâmetros formais devem ser inicializadas com os valores dos parâmetros reais correspondentes.

I.2.6 - Módulo de Controle Principal

Esse módulo tem como tarefas principais:

- Gerenciamento da memória

Devido às restrições de memória, ficarão em disco as rotinas referentes a comandos não muito utilizados, como por exemplo o REORGANIZE, que só serão carregadas quando forem ser executadas.

Quando for iniciar a execução de um programa, ficará na memória: o Módulo de Controle da Execução, o pseudo-código que será executado bem como as estruturas necessárias nesta fase.

- Gerenciamento de arquivos

Criação e manutenção dos arquivos fonte e de pseudo-código correspondente, permitindo o acesso aos diversos programas armazenados. Para isso deve ser mantida informações sobre cada programa, quais sejam: localização, número de segmentos,

etc. Quando o usuário desejar utilizar um programa, o sistema verifica se o mesmo existe e o trāz para a área intermediária ou emite mensagem informando se o programa não existe.

- Passar o controle ao módulo adequado conforme o modo de operação em que o sistema esteja e a entrada fornecida pelo usuário.

Será ativado sempre que o usuário for utilizar o sistema, e só devolve o controle ao SOCO (Sistema Operacional do TI) quando o usuário fornecer o comando ADEUS.

I.2.7 - Tratamento de Erros

Essa rotina é acionada por todos os módulos (à exceção do Módulo de Controle de Execução) sempre que for encontrado um erro em uma entrada fornecida pelo usuário. Sua principal tarefa é a emissão da mensagem correspondente ao erro ocorrido, e informar ao Módulo de Controle Principal que deve-se aguardar a correção do usuário.

I.3 - Estruturas de Dados

No decorrer da análise de um programa, o compilador vai obtendo informações tais como: nome e tipo dos segmentos que compõem o programa, número das linhas, incrementos que constituem cada linha, variáveis e constantes utilizadas, seu tipo e precisão, entre outras.

Estas informações devem ser armazenadas pois serão amplamente utilizadas, tanto na criação quanto nas futuras alterações do programa. Para esse armazenamento são usadas várias estruturas de dados.

Neste capítulo serão descritas as estruturas que serão criadas e utilizadas pela fase de compilação.

I.3.1 - Dicionário de Incrementos

Como as linhas que compõem um programa não serão fornecidas necessariamente na ordem em que serão executadas, essa estrutura, em forma de lista, conterá todos os incrementos de cada segmento, na ordem em que deverão ser executados. Cada nó dessa lista terá o formato mostrado na figura I.3.1.1.

BYTE	CONTEÚDO
Ø-1	número da linha em que se encontra o incremento. Esse campo dará a ordem em que estão as entradas nesse dicionário
2	tipo da instrução: indica se o incremento se refere a comando FOR, GOTO, etc.
3-4	endereço do código: endereço onde foi armazenado o código gerado para o incremento
5-6	apontador para referências: usado para referências futuras, como será mostrado em (III.4)
7-8	endereço do próximo incremento: apontador para a entrada referente ao incremento seguinte, no dicionário de incrementos

Fig. I.3.1.1 - Formato do Dicionário de Incrementos

É criada uma entrada nessa estrutura sempre que:

i) o usuário fornecer uma nova linha; ou

ii) for feita uma referência a uma linha ainda não existente. Nesse caso a entrada conterá somente o número da linha e o endereço da referência.

As informações restantes sobre o incremento só são preenchidas quando a linha fornecida não tiver erros.

I.3.2 - Tabela de Símbolos

Essa estrutura será muito utilizada durante a fase de compilação. Ela conterá os identificadores e as constantes reais ou alfanuméricas.

A busca e inserção de elementos nessa tabela é feita pela análise léxica, mas o restante dos campos é completado ou testado pelas rotinas semânticas.

I.3.2.1 - Organização

O acesso a essa tabela será direto, utilizando uma função de espalhamento aplicada a uma chave. Esse método é chamado de método de "hashing". A função de espalhamento é aplicada à chave e o seu valor dá o endereço onde deve ser iniciada a busca. Na aplicação dessa função pode ocorrer que duas chaves

diferentes sejam associadas ao mesmo endereço. Quando tal ocorre, diz-se que houve uma colisão. Para utilizar esse método deve-se escolher a função adequada e também um esquema para o tratamento de colisão.

A chave será formada pelos caracteres que constituem o identificador ou constante. A função a ser aplicada à chave obedecerá ao esquema multiplicativo, mostrado em Knuth[6]. Resumidamente, o método consiste em multiplicar a chave por um valor inteiro A que seja primo com w (o tamanho da palavra do equipamento, no caso, 2 bytes) e deslocar o resultado m bits para a esquerda, onde M (o tamanho da tabela) é um valor da ordem de 2^m. A função, a qual chamaremos aqui de h(K), onde K é a chave, tem seu valor então nos bits mais alta ordem da metade à direita do produto de A por K.

Uma vantagem desse método é que não utiliza divisão, que é uma operação bastante lenta no T.I. Outra vantagem, conforme cita Knuth[6], é que ele possibilita uma boa randomização quando as chaves não são tão aleatórias (por exemplo, K tem valores do tipo TIPO1, TIPO2, TIPO3), o que diminui bastante o número de colisões. Tal vai depender da escolha de um bom valor para A.

Como o método descrito se aplica a uma palavra (isto é, 2 bytes) e a chave será composta de vários bytes (até 6, no caso de identificadores, até 256 no caso de constantes alfanuméricas ou 6, para as reais), antes de se aplicar a

função, os caracteres que compõem a chave serão combinados através de um "ou exclusivo" até obter um resultado em uma palavra. Para obter valores o mais randômico possível, será dado um deslocamento circular de cada palavra que compõe a chave antes de ser dado esse "ou exclusivo". Tal artifício fará com que diminua a possibilidade de que chaves tais como XY e YX sejam reduzidas ao mesmo valor, pois essa operação, é comutativa.

As colisões serão tratadas também através de espalhamento, que consistirá em descobrir um valor que seja primo como M, no caso m número ímpar. A sugestão, contida na mesma fonte de referência citada acima, é deslocar h(K) obtida mais m bits para a esquerda e somar 1. O valor assim obtido será acrescentado ao valor de H(K), dando o novo endereço. Esse processo deve se repetir até que não haja mais colisão ou caso a tabela esteja cheia.

Com esse método, o tempo de acesso à tabela será relativamente curto, principalmente se comparado com outros métodos, o que é de alguma vantagem de vez que essa estrutura será muito acessada na fase de compilação. No entanto, a deleção de uma entrada não é tão simples, pois pode-se reparar, pelo método de tratamento de colisão, que se uma entrada for retirada da Tabela de Símbolos, perde-se o acesso a todos os elementos aos quais a aplicação da função de espalhamento gerou o mesmo endereço. No presente trabalho pode ocorrer que vez por outra seja necessário retirar alguma entrada, pois sucessivas deleções podem fazer com que algum identificador não seja mais usado no progra-

ma, e o usuário deseje criar um identificador com o mesmo nome, mas com características diferentes, conforme pode ser visto pelo exemplo abaixo:

```
(1) segmento X
      :
    10 P% = P%+2
      :
    30 print P%
      :
    fim
```

supor que o usuário forneça o seguinte comando de edição:

```
apaga 10,30
```

e em seguida faça as seguintes inserções

```
(2) 10 dim P(5,3)
         :
    30 mat read P
         :
```

Se as linhas 10 e 30 em (1) forem as únicas referências a P%, as inserções em (2) serão perfeitamente válidas. Caso a entrada referente a P% não seja retirada da lista de

atributos na Tabela de Símbolos, ocorreria erro em (2), pois P% e P( ) não são consideradas aqui como variáveis diferentes.

Para que a deleção de uma entrada não obrigue a recriar toda a tabela (tal ocorrerá quando a lista de atributos referentes a um determinado identificador ficar vazio), o que seria necessário para que algumas entradas não se tornem inatingíveis, haverá um campo indicando se a entrada foi desativada. Desse modo, a busca a um elemento na Tabela de Símbolos seguirá os seguintes passos:

i) aplicar a função de espalhamento ao símbolo;

ii) se a posição estiver vazia, indicar que não houve sucesso na busca;

iii) se a posição estiver ocupada com um símbolo diferente do que é esperado, aplicar os procedimentos para o tratamento de colisão; senão, pare a busca e acuse sucesso;

iv) se a posição foi desativada, presseguir na busca.

Caso a busca não tenha tido sucesso e seja necessário inserir o novo símbolo, essa inserção se dará na primeira posição vazia ou desativada que for encontrada.

I.3.2.2 - Conteúdo

A tabela será acessada pelo método descrito em (I.3.2.1), onde a chave poderá ser formada por:

- nome e identificação do segmento, no caso de variáveis
- nome da função ou subrotina
- valor da constante real ou alfanumérica.

Cada entrada será dividida em 3 partes:

i) espécie: indica o tipo de símbolo (identificador, constante, etc.) ou se está vazia ou desativada

ii) nome ou valor

iii) atributos.

O campo de nome terá 6 bytes, que é o número máximo de caracteres para um identificador. Para as constantes alfanuméricas a busca será um tanto mais demorada, pois como podem conter até 256 caracteres, não caberá todo nesse campo, devendo a busca se dar na área de constantes alfanuméricas.

A terceira parte contem as informações que o compilador irá necessitar sobre o símbolo, o que cons

tituem os atributos. Estes variam de acordo com a espécie de sím bolo, como pode ser notado a seguir:

a) Variáveis

- tipo:inteira, real ou alfanumérica
- precisão: simples ou dupla
- endereço de alocação
- se foi definida ou não
- número de dimensões
- contador de referências
- segmento em que está sendo usada
- se é parâmetro formal ou não

b) Funções e subrotinas

- tipo da função: inteira, real ou alfanumérica
- espécie: se é declaração ou função de múltiplas linhas
- se já foi definida ou não
- se foi atribuído valor à função dentro do segmento ou não
- segmento de definição

c) Arquivos

- periférico associado
- tamanho do registro

d) Constantes

- tipo
- precisão
- endereço de alocação
- segmento

O formato da entrada da tabela é mostrado na figura (I.3.2.2). Por questão de economia de espaço as informações mutuamente excludentes ocuparão as mesmas posições de memória.

BYTE	BYTE	CONTEÚDO
0	0-7	espécie:=0: entrada vazia; =1: entrada deletada; =2: identificador; =3: constante real; =4: cte. alfanumérica
1-6	0-47	símbolo: nome do identificador|valor da constante|6 primeiros caracteres da cadeia alfanumérica
7	0-7	identificador do segmento de definição
8	0-3	uso:=0000: var. simples; =0001: var. subscrita; = 0010: função; =0011: função declaração; =0100: subrotina; = 0101: arquivo; =0110: constante
	4-7	tipo: =0000: inteiro; =0001: real; =0010: alfan.

continua...

Continuação...

BYTE	BITS	CONTEÚDO
9	0	indicador de parâmetro formal
	1	indica se já houve definição
	2	se foi atribuído valor no segmento
	3	precisão (=0: simples; =1: dupla)
	4-7	número de dimensões|número de parâmetros|periférico associado ao arquivo
10	0-7	contador de referências (*)
11-12	0-15	- para arquivos: tamanho do registro - para fç. e subr.: ender. da lista de parâmetro

Fig. I.3.2.2

Observações:

(*) é incrementado de 1 a cada aplicação do identificador e decrementado do mesmo modo quando uma linha contendo esse identificador é deletada. Quando chegar a zero, significa que o símbolo não está mais sendo utilizado no segmento, e a entrada correspondente na tabela é marcada como deletada. Uma entrada na Tabela de Símbolos referente a uma função ou subrotina só é retirada da tabela (isto é, marcada como deletada), quando o segmento de definição for deletado. Esse controle será feito pelo Analisador Léxico.

I.3.3 - Tabela de Designadores de Arquivos

Esta tabela será utilizada em fase de compilação para conter os designadores de arquivos que aparecem em cada declaração FILE que é fornecida pelo usuário. Haverá sempre uma entrada correspondente ao terminal, que é dispositivo básico de entrada e saída e está associado ao designador Ø.

Cada entrada dessa tabela terá o formato:

valor	descrição

- descrição: aponta para a descrição do arquivo, que se encontra na Tabela de Símbolos
- valor: valor entre Ø e 12, indicando o número do designador

A utilização dessa tabela é para fins semânticos; pode-se assim determinar se o usuário definiu arquivos diferentes associado ao mesmo designador.

I.3.4 - Tabela de Segmentos

A Tabela de Segmentos constitui o primeiro nível de endereçamento para o arquivo objeto. Juntamente com o Dicionário de Incrementos, que seria assim o segundo nível de endereçamento, pode-se acessar o código gerado para qualquer incremento de qualquer segmento.

Esta estrutura será utilizada em fase de compilação e cada entrada terá as seguintes informações:

- nome do segmento: até 6 bytes, contendo o identificador que aparece a cada comando SEGMENTO

- tipo: indica se é o programa principal, subrotina ou função

- endereço da área de dados: indica onde começa a área de dados do segmento

- endereço de alocação: contem a posição de memória onde começará a alocação do segmento

- apontador para o Dicionário de Incrementos: indica onde começa a lista correspondente aos incrementos do segmento

- tamanho: número de bytes indicando a área necessária para alocar o código do segmento

- chave de erro: será ligada sempre que no Passo 2 da análise for detetado algum erro do tipo: número de linha referenciado não foi definido, etc. Esta chave só será desligada quando for dado o comando SEGMENTO, seguido do nome do segmento errado

- total de linha|incrementos: contem o número total de linhas e de incrementos que contem o segmento. É incrementado de 1 a ca

da nova linha que é fornecida sem erro, e diminuído de 1 quando o usuário apagar a linha.

I.3.5 - Tabela de Palavras Reservadas

Conterá todos os comandos que o usuário poderá utilizar no sistema, quais sejam, os comandos de controle, edição, entrada e da linguagem BASIC.

Além do verbo que indica o comando, essa tabela deverá conter o modo em que este pode ser utilizado. Assim, por exemplo, se o sistema se encontrar em modo de entrada e o usuário fornece o comando REORG, é acusado erro. Essa verificação é feita pelo analisador léxico ("scanner").

I.3.5.1 - Estrutura da Tabela

Para maior facilidade no fornecimento de comandos, será permitido o uso de abreviaturas, ou seja, com somente algumas das letras iniciais pode ser possível o reconhecimento de um comando. Assim, por exemplo, o comando APAGA pode ser fornecido de qualquer dos seguintes modos: AP|APA|APAG|APAGA.

Olhando-se a lista de palavras reservadas, no Apêndice E, vemos que somente a letra A não identifica o comando pois temos ainda ADEUS e AND.

Para representar esta tabela, foi utilizada uma variação da "trie structure", mostrada em Knuth[6].

Aqui, a busca de um determinado elemento é feita através de comparações entre caracteres, e não entre chaves, como é feito, por exemplo, no acesso à Tabela de Símbolos. É como se usássemos um dicionário, ou seja, a primeira letra da palavra indica aonde encontrar todas as palavras reservadas que comecem com essa letra.

A "trie structure" descrita na referência acima citada, consiste em um árvore "M-ária", na qual cada nó são vetores de M-elementos, cada qual contendo dígitos ou caracteres. Cada nó em um nível $\underline{\ell}$ dessa árvore representa um conjunto de todas as palavras que comecem com a sequência de $\underline{\ell}$ caracteres; o nó apresentaria então M possibilidades de desvios, dependendo do (ℓ+1)-ésimo caracter.

Uma das variações a esse método sugere que se use uma estrutura tipo floresta, ao invés de nós contendo M elementos. Tal economizaria memória, pois a maioria das entradas desse vetor estará vazia. A figura (I.3.5.1) dá um exemplo de como seria a representação da Tabela de Palavras Reservadas usando-se a "trie" (I.3.5.1.a) e usando a variação em forma de floresta (I.3.5.1.b). No caso do exemplo, é mostrado somente um subconjunto das palavras que comecem com R, S e W. O caracter ⊣ representa o fim da palavra.

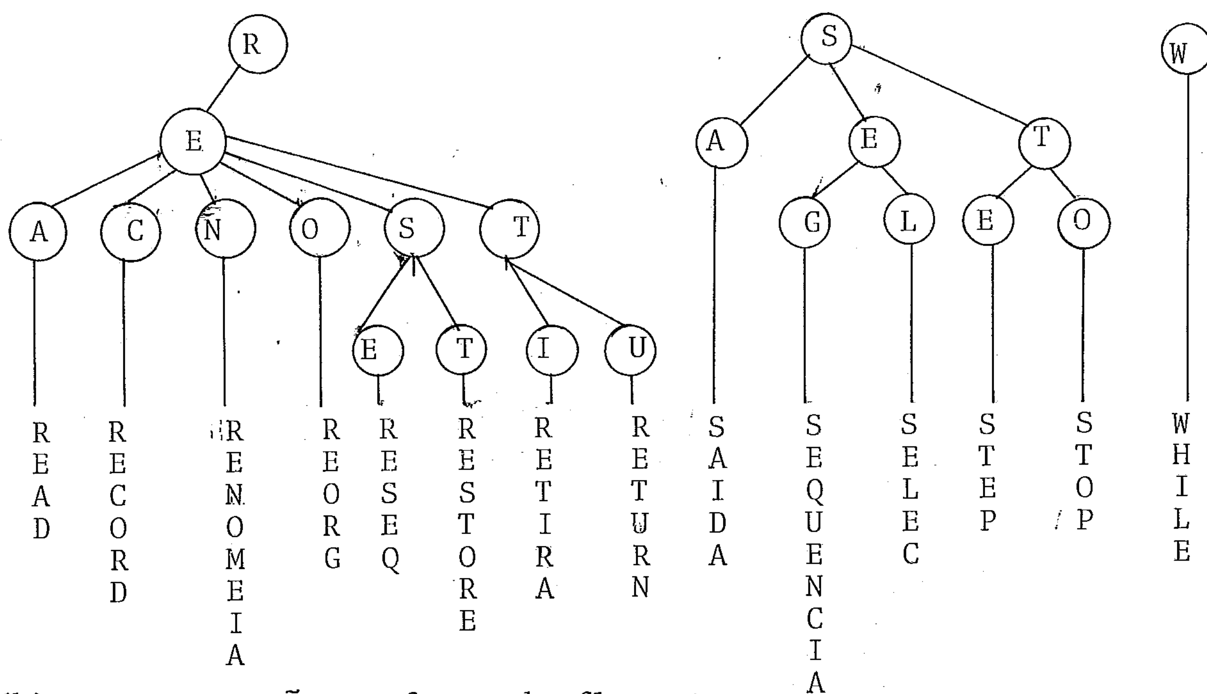
	1	2	3	4	5	6	7	8
⊣								
A				Read				
B								
C				Record				
D								
E		(4)	(5)			Step	Reseq	
F								
G					Segmento			
H								
I								Retira
J								
K								
L					Selec			
M								
N				Renomeia				
O				Reorg		Stop		
P								
Q								
R	(2)							
S	(3)			(7)				
T			(6)	(8)			Restore	
U								Return
V								
W	While							
X								

continua...

Continuação ...

	1	2	3	4	5	6	7	8
Y								
Z								

(a) representação em forma de trie das palavras reservadas começadas por R, T ou W



(b) representação em forma de floresta

Fig. I.3.5.1

O método utilizado neste trabalho englobará as duas estruturas citadas anteriormente, do seguinte modo:

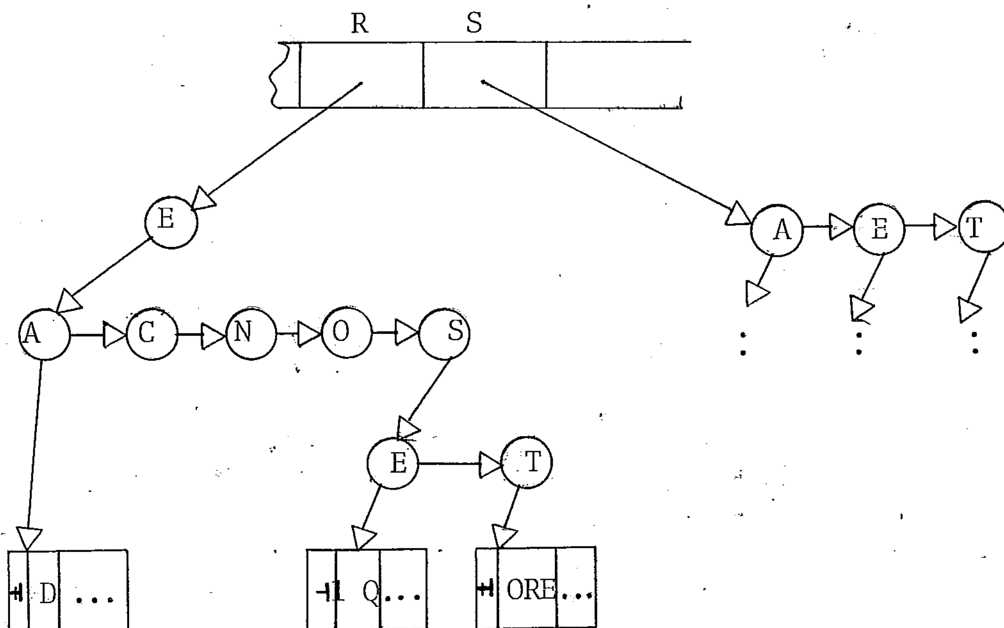
(1) a raiz de cada árvore da floresta em (I.3.5.1.b) estará em um vetor de 26 posições, correspondentes a cada letra do al-

fabeto. O acesso a um elemento se dará diretamente, portanto. O conteúdo de um elemento desse vetor será o endereço da árvore contendo todos os caracteres das palavras que comecem com uma determinada letra.

(2) os caracteres que compõem as palavras reservadas estarão em uma árvore binária. A busca será feita comparando-se o caracter da árvore, presseguindo pelos nós à direita até encontrar o caracter procurado. A partir de então toma-se o nó à esquerda e procede-se na busca dos caracteres seguintes do mesmo modo.

(3) as folhas das árvores conterão o restante dos caracteres das palavras, a partir de onde não haja mais dúvida possível sobre qual a palavra procurada. A partir daí, faz-se uma busca sequencial até atingir o fim da entrada.

A figura I.3.5.2 mostra como ficará a representação do exemplo anterior.



Apesar de ser mais lento que a estrutura "trie", esse método tem as seguintes vantagens:

- é mais econômico em termos de espaço;

- o acesso ao 1º caracter da palavra é direto, o que torna a busca um tanto mais rápida que na floresta;

- cada nó não falha tem tamanho fixo, o que os torna mais fáceis de manipular do que a forma de floresta.

I.3.5.2 - Conteúdo dos Componentes

Cada entrada do vetor conterá um endereço (2 bytes) de onde se encontra a árvore contendo as palavras que iniciem com determinada letra.

A árvore conterá dois tipos de nós: as folhas, que contêm o restante dos caracteres da palavra, e os outros, que são raízes de sub-árvores que iniciam com uma dada sequência de caracteres. Estes últimos nós serão formados pelos seguintes campos:

- tipo: =∅, indicando que é um nó não-falha

- apontador para a sub-árvore esquerda

- apontador para a sub-árvore direita

- caracter

O conteúdo de uma folha será:

- tipo: =1, indicando que é folha

- código interno associado à palavra reservada

- modo de operação onde a palavra pode ser usada

- número de caracteres restantes

- resto dos caracteres

A verificação sobre a validade do uso de uma determinada palavra será feita pelo analisador léxico.

I.4 - Organização da Memória

Em um compilador incremental, as tabelas utilizadas pela fase de compilação devem ser preservadas, mesmo que o programa já tenha sido executado. De modo análogo, a área de dados do programa deve ser preservada quando houver uma interrupção. O exemplo a seguir ilustra isto:

```
10 dim A(20)
20 let A(3) = 57
30 end
exec
print A(3)
```

Vê-se que a área contendo os valores de A deve ser preservada mesmo após o término da execução, e a Tabela de Símbolos não deve ser destruída após a compilação, senão o comando imediato: print A(3) jamais acharia o elemento A(3).

Como a memória disponível é pequena, não será possível manter todas as rotinas do compilador, suas estruturas, o programa do usuário com a respectiva área de dados o tempo todo na memória. A fase de compilação e a de execução serão portanto consideradas distintas, de modo que as rotinas e dados necessários em uma delas serão carregados depois que os da outra tenham sido retirados.

Na fase de compilação a memória deverá conter:

a) <u>rotinas do compilador</u>

- Módulo de Controle Principal: controlará a execução do compilador;

- Tratamento de Erros: recebe o controle sempre que for encontrada uma linha com erro de síntaxe;

- Análise de Comandos: analisa os comandos de controle que o usuário forneça;

- Editor: analisa os comandos de edição;

- Compilador BASIC: analisa uma linha de programa fonte e gera o código correspondente.

b) <u>estruturas</u>

As tabelas, dicionários e listas criadas e mantidas pelo compilador;

c) <u>"buffers"</u>

São áreas que conterão a linha fonte e o código gerado para essa linha.

d) área comum

Conterá informações que são compartilhadas tanto pela fase de compilação quanto pela de execução, como por exemplo: identificação do segmento corrente, ponto em que sua execução foi interrompida, entre outras.

Na fase de execução, o código do programa, que se encontra em disco, deve ser trazido para a memória, bem como sua área de dados. O conteúdo da memória será então:

a) Rotinas

- Módulo de Controle Principal: supervisionará a execução do programa;

- Tratamento de Erros: emitirá mensagens referentes aos erros que ocorram na fase de execução;

- Módulo de Controle da Execução: conterá rotinas necessárias a essa fase;

b) Programa do Usuário

Tem tamanho fixo nesta fase, pois não é permitida a recursividade e nem o aninhamento de funções ou subrotinas.

c) Área de Dados do Programa

Cresce no sentido inverso ao programa, e estará dividida em duas partes:

- área estática: conterá variáveis simples, constantes, áreas de E/S, a Tabela-data, variáveis subscritas e temporárias numéricas. Seu tamanho é fixo no decorrer da execução. Só serão referenciadas as variáveis que estejam na área de dados de cada segmento, e não será permitida a alocação dinâmica de variáveis subscritas.

- área dinâmica: conterá as constantes e variáveis temporárias alfanuméricas. Pode variar de tamanho na fase de execução, tomando e liberando continuamente espaço da área livre.

d) Área Comum.

I.5 - Organização dos Arquivos

Será tratado aqui somente a organização dos programas em disco. Os arquivos de dados do usuário serão manipulados através do SOCO.

Os programas do usuário poderão ser armazenados de forma temporária ou permanente. Neste último caso, o usuário deve fornecer o comando GUARDE. Será necessária portanto uma área temporária, para conter o programa que está sendo criado ou edi-

tado. O conteúdo dessa área é perdido sempre que:

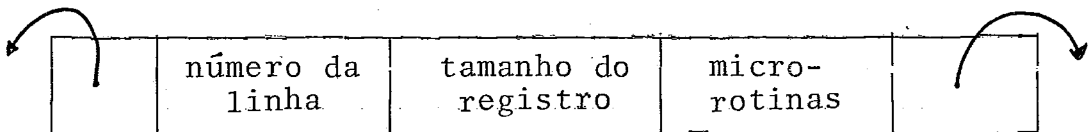
a) o usuário fornecer o comando de controle PEGUE;

b) o usuário fornecer o comando ADEUS, abandonando o sistema.

Os arquivos de programa, quer sejam temporários ou não, estão divididos em duas partes:

- área fonte: contém as linhas do programa fonte, acessadas através do método sequencial indexado do SOCO, organizado por ordem do número das linhas. Tal permitirá tanto o acesso direto, quando se desejar alguma linha específica, quanto o sequencial, por exemplo, quando o programa for listado.

- área de código: organizado em forma de lista de apontadores, por ordem do número da linha dentro de cada segmento. Tal evita que o arquivo tenha que ser re-arrumado sempre que for feita uma edição. Para obter a próxima instrução a ser executada, bastará seguir os apontadores. A cada linha corresponderá um nó nessa lista, com o seguinte formato:



onde:

- os campos das extremidades são apontadores para o registro anterior e para o próximo, necessários para a edição;

- o 2º campo é o número da linha fornecida pelo usuário;

- o tamanho determina onde acaba um determinado registro, de vez que seu formato é variável;

- o 4º campo conterás as micro-rotinas geradas para cada incremento da linha.

Além do programa, a área de código conterá ainda os dados e as tabelas criadas pelo compilador. Precedendo essa área haverá um registro de identificação contendo:

- o nome do programa

- o número total de segmentos

- endereço da Tabela de segementos

- endereço da área livre (área constituida por registros que ainda não tenham sido utilizados pelo programa).

Devido a facilidade de edição, essa área de código é dinâmica, ou seja, seu tamanho varia conforme registros sejam inseridos ou retirados do arquivo. Por questões de simplificação e pouco uso de memória, não serão reaproveitados os espaços referentes a registros que foram apagados. Caso não haja mais espaço disponível, o usuário deverá fornecer o comando REORG, para

compactação do arquivo de código. Apesar do custo envolvido neste tipo de operação, obtem-se maior economia de tempo do que se fosse feita uma rearrumação a cada inserção ou deleção.

Além dos programas, o compilador deve dispor também de uma área de trabalho onde será armazenada, por exemplo, a imagem da memória de um programa que estava sendo executado. Essa imagem é necessária pois, como já foi mencionado, a área de dados do programa que foi executada deve ser preservada na fase de compilação.

I.6 - Resumo do Funcionamento

Será mostrado resumidamente aqui a relação entre os diversos componentes e como eles são acionados, sob a forma de um algoritmo.

1. Início

Quando o usuário teclar BASIC, o SOCO passa o controle ao módulo de controle principal, que traz para a memória as rotinas do analisador de comandos, do editor, do compilador BASIC e de tratamento de erros.

2. Usuário Fornece um Novo Programa

O compilador BASIC analisa cada linha, reporta os erros e gera o código; são criadas as estruturas de dados refe-

rentes ao programa. A linha fonte e o código correspondente vão sendo guardadas nas respectivas áreas temporárias.

3. Em vez de criar um programa, o usuário utilizou o comando PEGUE

Nesse caso deve ser verificado se o programa desejado realmente existe. O fonte e o código são trazidos para a área temporária. As tabelas são carregadas na memória.

4. Usuário fornece o comando EXEC

O módulo de controle principal chama a rotina de carga de programas, que carrega o código a ser executado na memória. Em seguida são carregadas as rotinas do módulo de controle da execução (MCE).

5. Durante a execução

Caso haja algum erro, o módulo de controle de execução (MCE) aciona a rotina de tratamento de erros, e em seguida a execução é terminada.

6. Interrupção da execução

A rotina de tratamento de interrupção, que faz parte do MCE, deve salvar na área de trabalho a imagem da memória e o controle passa ao módulo de controle principal, e volta-se à

fase de compilação.

7. Usuário fornece comando de edição

O módulo de controle principal chama o editor. É acionada uma chave na tabela de segmentos indicando que houve edição.

8. Usuário fornece comando imediato

O compilador BASIC verifica se não há erro de sintaxe e gera o código. Em seguida são carregadas as rotinas de módulo de controle da execução, bem como a área de dados do segmento corrente.

9. Usuário fornece o comando CONT

O módulo de controle principal deve verificar:

a) se havia uma execução de programa do usuário em andamento;

b) se nenhum segmento foi editado. Neste caso, o usuário deverá fornecer o comando EXEC.

Em seguida, são carregadas as rotinas do módulo de controle da execução, e a imagem da memória é reconstituída a partir da área de trabalho.

10. Fim da execução

Procedimento análogo a (6).

11. Usuário fornece ADEUS

O controle é devolvido ao SOCO.

II.O PROCESSADOR DO CÓDIGO A SER GERADO

Para tornar o desenvolvimento de sistemas o mais independente possível da máquina real e do processador ora existente, foi criado um processador hipotético, especialmente voltado para as necessidades do PLTI, que é a linguagem na qual são desenvolvidos a maioria dos sistema do T.I., inclusive o presente projeto. Esse processador foi chamado de hipotético porque na verdade ele não existe fisicamente (i.e., não há hardware que lhe corresponda), constituindo então uma máquina virtual.

Essa máquina vai lidar com dois tipos de endereçamento: de instruções do programa e dos dados. As instruções do programa são carregadas na memória quando da sua execução e permanecem inalterados durante a fase de execução. Os dados são manuseados por uma pilha, e todas as operações aritméticas operam com os dados que estão no topo da pilha. O elemento no topo da pilha é apontado por S, e as posições seguintes são referenciadas como SØ, S1, ..., S|N|. O contador de programas (ou programm counter - PC) contem o endereço da instruções seguinte à que está sendo executada.

As intruções do processador PLTI, chamadas de micro-rotinas do PLTI são interpretadas, e manipulam com dois tipos de dados: byte e address, sendo que este último ocupa 2 bytes. Dispõe-se de instruções que:

- carreguem na pilha os dados a serem manuseados (instruções do tipo LOAD), por exemplo:

  LIB - carrega um dado do tipo byte

- armazenem o topo da pilha em uma posição de memória cujo endereço (byte ou address) se encontra na posição seguinte na pilha. Exemplos:

  SB - armazena dado byte
  SA - armazena dado address

- efetuem operações aritméticas, lógicas e comparações com operando(s) no topo da pilha; como por exemplo:

  ADD - adiciona dois valores no topo da pilha
  MINU - troca sinal do elemento no topo da pilha

- executem deslocamentos ('shift') de bits para a esquerda ou para a direita com o elemento no topo da pilha:

  SLR - deslocamento para a direita

- executem comandos do PLTI:

  CASE - desvio computado
  IF - desvio condicional
  DOTA - comando iterativo com variável de controle addr.

- executem operações de entrada e saída:

  READ, WRITE

- ativem subrotinas quando houver alguma chamada e retornem ao ponto de partida:

  ENT - entrada de rotina

  RET - retorno ao ponto de chamada

O formato de cada instrução vai depender do tipo de parâmetros necessários. Em geral, o tamanho de cada instrução varia de 1 a 5 bytes.