JOÃO JOSÉ NETO

"ASPECTOS DO PROJETO DE SOFTWARE DE UM MINICOMPUTADOR"

"Dissertação de Mestrado" apresentada à Escola Politécnica da Universidade de São Paulo, para obtenção do título de Mestre em Engenharia.

Área de Concentração - Engenharia de Eletricidade.

Orientador: Prof. Dr. Antonio Marcos de Aguirra Massola

São Paulo
-1975-

Ao Prof. Dirceu
oferece um exemplar
de meu trabalho de
mestrado, com meus agrade-
cimentos. 10/11/75

JOÃO JOSÉ NETO

"ASPECTOS DO PROJETO DE SOFTWARE DE UM MINICOMPUTADOR"

"Dissertação de Mestrado" apresentada à Escola Politécnica da Universidade de São Paulo, para obtenção do título de Mestre em Engenharia.

Área de Concentração - Engenharia de Eletricidade.

Orientador: Prof. Dr. Antonio Marcos de Aguirra Massola

São Paulo
-1975-

A meus pais e irmã.

A G R A D E C I M E N T O S

Ao Prof. Dr. Antonio Marcos de Aguirra Massola, pela dedicação com que acompanhou, orientou, apoiou e incentivou este trabalho.

Aos Professores Doutores James Gregory Rudolph, Antonio Hélio Guerra Vieira e Tamio Shimizu pelas idéias apresentadas.

Aos Engs. Benício José de Souza, Ting Kong Sen e Wanner Monteiro Pinheiro, e engenheirandos Luiz Sanches Filho, Mário Tachibana e Charlie Lin, pela participação efetiva no projeto e no aprimoramento dos programas desenvolvidos.

Aos Engs. Laércio Antonio Marzagão, Antonio Marcos de Aguirra Massola e Benício José de Souza, à Enga. Selma Shin Shimizu Melnikoff, à Profa. Selenê Cavalcanti Ferrari e aos meus familiares, pelos inúmeros diálogos e constante apoio e estímulo, decisivos para a concretização deste trabalho.

À srta. Sonia Regina Izarelli pelos serviços de datilografia e desenho.

A Valdecir Finco pelo serviços de impressão.

A todos, enfim, que de uma ou outra forma contribuíram para a concepção ou desenvolvimento deste trabalho, particularmente à equipe de estagiários do Laboratório de Sistemas Digitais, sem cuja valiosa participação não teria sido possível a realização deste trabalho.

R E S U M O

O presente trabalho descreve os métodos utilizados no desenvolvimento de alguns dos módulos do "software" básico do Patinho Feio, o primeiro minicomputador desenvolvido no Laboratório de Sistemas Digitais do Departamento de Engenharia de Eletricidade da Escola Politécnica da Universidade de São Paulo.

Para cada módulo apresentado, são discutidos os seus objetivos, sendo, quando conveniente, apresentados alguns dos problemas enfrentados durante seu desenvolvimento específico ou então problemas mais gerais, referentes ao projeto de programas semelhantes ao mesmo. Sempre que possível, são apresentadas alternativas de solução dos referidos problemas, descrevendo-se finalmente alguns detalhes de um exemplo de implementação.

No Capítulo 2 são discutidos mais profundamente os problemas enfrentados durante o projeto e a implementação de um montador, desde a sua concepção até a sua implantação definitiva, enfatizando-se a metodologia empregada na implementação e os critérios adotados nas decisões mais importantes.

No Capítulo 3 é descrito um simulador-interpretador, implementado em outro computador com a intenção de auxiliar o desenvolvimento do "software" básico do Patinho Feio. Algumas técnicas de simulação são discutidas neste Capítulo, ao lado dos algoritmos utilizados pelo interpretador.

Nos demais Capítulos, são descritos mais alguns programas do "software" básico desenvolvidos para o Patinho Feio, tais como um desmontador, um programa para auxiliar a depuração de outros programas e um editor simbólico.

Nos apêndices, são apresentados tópicos julgados convenientes para a complementação de algumas idéias, bem como alguns exemplos de utilização de programas apresentados neste trabalho.

A B S T R A C T

The present work describes the methods that had been employed during the development of some of the basic software modules of the "Patinho Feio", the first minicomputer of the "Laboratório de Sistemas Digitais da Escola Politécnica da Universidade de São Paulo".

For each module, after a brief discussion of their objectives, some of the main problems which had to be solved during their development are presented, as well as more general ones, which arrive when designing programs of the same class. Whenever possible, alternative solutions of these problems are presented, and, at last, an example of implementation is described.

Chapter 2 discusses in detail the problems arrived during the design and implementation of an assembler, from the phase of its conception until that of its final installation, emphasizing the methods employed in the implementation and the criteria which were used when the most important decisions had to be taken.

Chapter 3 describes a simulator-interpreter, which had been implemented in an auxiliary computer, and was intended to help the development of the basic software of the "Patinho Feio".This chapter discusses also some simulation techniques and the algorithms employed in the interpreter.

The remaining chapters describe some additional programs of the basic software developed for the "Patinho Feio", like a disassembler, a debugging routine and a symbolic editor.

The appendices present some subjects considered helpful for completing some ideas, and some execution examples of the programs presented in this work.

ÍNDICE

CAPÍTULO 1. INTRODUÇÃO

1. INTRODUÇÃO

1.1 - Objetivos

No presente trabalho expõe-se os métodos utilizados no projeto e construção de algumas das peças do "software" básico do Patinho Feio, o primeiro minicomputador desenvolvido no Laboratório de Sistemas Digitais (LSD) do Departamento de Engenharia de Eletricidade da Escola Politécnica da Universidade de São Paulo. Descreve-se as diversas fases do projeto, analisando detalhadamente os principais problemas que cada uma delas apresenta. Esta análise é feita em nível geral, quando se tratar de problemas que devam ser enfrentados independentemente da máquina, ou então em nível particular,em caso contrário. Esta descrição destina-se principalmente à orientação de futuros projetos semelhantes.

A análise dos problemas dependentes da máquina, bem como a dos detalhes de implementação dos programas descritos, restringe-se ao caso particular dos exemplos implementados, devendo-se levar em conta a configuração do sistema para o qual foram desenvolvidos. Assim, algumas das soluções apresentadas em tais exemplos podem não ser vantajosas em sistemas que não tenham as mesmas características.

São apresentadas alternativas para as soluções de alguns dos problemas enfrentados, fazendo-se uma análise comparativa das mesmas. Procurou-se, para isto, ressaltar vantagens e desvantagens de cada uma das alternativas mencionadas, terminado-se por optar por uma delas e, sempre que possível, justificando-se a opção adotada na implementação do exemplo, com uma argumentaçãobaseada nas características e limitações do computador utilizado.

Além disto, é descrito aqui o funcionamento dos programas implementados, especialmente com a finalidade de documentar as linhas de raciocínio e a filosofia de projeto e de implementação adotadas para facilitar futuras alterações de tais programas.

1.2 - Generalidades (ref. 11, 14)

Nos próximos Capítulos estão desenvolvidos, dentro das linhas apresentadas em 1.1, os seguintes módulos do "software" :

- um montador ("assembler"), cuja finalidade é a de dispensar o programador da tarefa de programação em linguagem de máquina, permitindo que os programas sejam escritos em linguagem simbólica mnemônica. Uma das versões implementadas dá, além disto, a possibilidade de modularização dos programas, introduzindo a facilidade de se utilizar programas escritos e traduzidos independentemente sem a necessidade de nova tradução a cada utilização dos programas parciais;

- uma rotina de depuração, cuja finalidade é a de facilitar a pesquisa e correção de erros existentes em programas ainda não depurados;

- um desmontador ("disassembler"), cuja finalidade é a de gerar programas escritos na linguagem do montador, a partir de uma fita binária contendo o programa correspondente em linguagem de máquina;

- uma rotina de edição de arquivos ASCII, cuja finalidade é a de mecanizar a correção, a reprodução, a modificação e a listagem de arquivos ASCII, como por exemplo, programas perfurados, em linguagem simbólica, em fita de papel ("programas fonte").

Os quatro programas citados acima foram desenvolvidos para o Patinho Feio estando atualmente à disposição para execução no mesmo.

Alguns programas de utilidade foram implementados no computador Hewlett-Packard 2116-B, com a finalidade de facilitar e de tornar mais versátil a utilização do "software" do Patinho Feio.

Escolheu-se o computador HP-2116-B principalmente pela sua compatibilidade de entrada e saída com a do Patinho Feio (fita de papel). Os programas desenvolvidos para o HP-2116-B não foram implementados diretamente no Patinho Feio devido a algumas restrições atualmente existentes no mesmo, como por exemplo sua pequena capacidade de memória, a inexistência de memória de massa e a baixa velocidade dos periféricos disponíveis. Entretanto, poderão alguns desses programas vir a ser implantados no Patinho Feio assim que estiverem disponíveis no mesmo os periféricos necessários. Os programas aqui apresentados são os seguintes:

- um simulador em nível de registradores, que funciona como um interpretador da linguagem de máquina do Patinho Feio e que também permite a utilização dos periféricos rápidos do HP-2116-B em substituição aos disponíveis no Patinho Feio, bem como o acompanhamento do programa em execução por meio de rastreamento ("traces"), descarregamentos ("dumps") de memória, monitoramento dos processos de entrada e saída, etc. Naturalmente a utilização deste simulador-interpretador é às vezes mais conveniente que a do próprio Patinho Feio, como por exemplo, quando o programa for limitado em velocidade pela entrada e saída (caso de rastreamento e de listagens extensas com pouco processamento);

- um montador ("cross-assembler") equivalente ao desenvolvido para o Patinho Feio, que incorpora uma opção de geração da tabela de referências cruzadas ("cross-reference symbol table") e ordenação alfabética da tabela de símbolos, permitindo utilização total dos recursos do sistema operacional DOS (" disc operating system") do computador HP-2116-B. A tabela de referências cruzadas tem como finalidade facilitar a documentação e mesmo a depuração de programas extensos, gerando, a partir da linguagem simbólica, uma tabela em ordem alfabética, de todos os rótulos ("labels", isto é, nomes simbólicos dados aos endereços das instruções ou dos dados do programa), ao lado de cada qual são listados o número da linha onde o mesmo foi definido e o conjunto de todas as linhas onde foi referenciado. Indica também os rótulos indefinidos, os não utilizados, e o endereço de memória associado a cada rótulo.

1.3 - Observações

- Os dois computadores do LSD em que se efetuou a elaboração dos programas descritos neste trabalho, foram utilizados - com a seguinte configuração:

PATINHO FEIO:

1 Unidade Central de Processamento, com memória de núcleos de ferrite com 4K palavras de 8 bits (1K = 1ø24).

1 Leitora Ótica de Fita de Papel HP-2737-A, 3øø caracteres por

segundo (máximo).

2 Terminais Teletype ASR33 com leitora e perfuradora de fita de papel, entrada por teclado e saída impressa, 1Ø caracteres por segundo (perfuração opcional).

HEWLETT-PACKARD 2116-B:

1 Unidade Central de Processamento, com memória de núcleos de ferrite com 16K palavras de 16 bits.

1 Unidade de Disco Magnético HP-2770-A, com capacidade de 3 Megabits, organizados em 64 trilhas, de 92 setores cada, tendo cada setor 64 palavras de 17 bits.

1 Impressora HP-2767-A de 8Ø colunas por linha, 111Ø linhas por minuto (máximo).

3 Unidades de Fita Magnética HP-7970-A, velocidade máxima 37.5 ips, densidade 8ØØ bpi.

1 Console Teletype ASR35, de 1ØØ caracteres por minuto.

1 Leitora Ótica de Fita de Papel HP-2748-B, de 5ØØ caracteres por segundo (máximo).

1 Perfuradora de Fita de Papel HP-2895-B, de 75 caracteres por segundo.

1 Leitora de Cartões de marcas óticas HP-2761-A, de 2ØØ cartões por minuto.

- Os programas desenvolvidos para o Patinho Feio foram inicialmente projetados para a configuração mínima acima descrita. Entretanto, com a inclusão de novos periféricos, é conveniente que se configure os programas existentes, adaptando-os aos novos equipamentos disponíveis, o que pode tornar mais rápida e eficiente a execução de tais programas. Assim, todas as vezes que um

novo periférico é incorporado ao sistema, faz-se a adaptação dos programas às novas condições, permitindo que sejam utilizados os recursos implantados.

Atualmente, dispõe-se dos seguintes periféricos adicionais, já incorporados ao sistema pela equipe do LSD:

1 Impressora de linha de 80 colunas HP-2767-A (1110 linhas/minuto).

1 Terminal DECWRITER DEC LA 300, de 80 colunas, 30 caracteres por segundo (máximo).

1 Perfuradora Rápida de Fita de Papel HP-2735-A, de 115 caracteres por segundo.

Os programas menos extensos permitem a inclusão de uma "seção de configuração", parte do programa que o adapta para a utilização dos periféricos desejados. Os mais extensos, no entanto, não dispõem de memória livre suficiente que possa alojar a seção de configuração. Para estes programas, a configuração pode ser feita manualmente modificando-se certas posições de memória, ou, então montando-se novamente o programa com as devidas alterações.

- Em todos os programas aqui descritos, procurou-se orientar o projeto de tal modo que o programa resultante apresentasse o maior número possível de recursos por unidade de área de memória ocupada. É evidente que, na maioria das vêzes, uma tentativa de adicionar mais recursos ao programa sem aumentar a área ocupada pelo mesmo pode trazer problemas no desempenho do programa, tais como queda de velocidade, quebra de estrutura, etc. Em alguns destes casos, como será visto, optou-se por soluções de compromisso, e, em outros, por soluções que dessem ao programa maiores recursos de utilização.

No presente caso, tem-se os seguintes fatores a considerar :

a) baixa capacidade de memória (4K palavras de 8 bits)

b) baixa velocidade de saída dos dados (teletype: 1ø palavras por segundo)

c) deficiência do conjunto de instruções no que se refere a:

- manipulação de dados com mais de uma palavra (p/ex, dados em precisão dupla).
- comparação de dados
- extração e inserção de campos dentro de uma palavra (Inclusive mascaramentos)
- operações aritméticas

Analisando-se os fatores a) e b), levando-se em conta observações realizadas nos programas aqui descritos, chegou-se às seguintes conclusões:

- O tempo de processamento é desprezível em relação ao de entrada e saída. Isto permite concluir que mesmo se o programa for algumas vezes mais lento, o tempo total de execução será praticamente o mesmo.

- Rotinas otimizadas em relação ao tempo de processamento ocupariam, em certos casos, até cerca de 40% a mais de memória em relação a rotinas equivalentes, otimizadas em relação ao número de palavras ocupadas pela mesma na memória. Levando-se em conta o fator a), percebe-se que, quando a dimensão do programa for comparável à dimensão da memória disponível, pode-se chegar até a uma condição de inviabilidade de implementação do mesmo se se insistir em utilizar rotinas otimizadas em velocidade. Deve-se observar que tais rotinas sejam, em média, poucas vezes mais rápidas, o que já foi dito ser praticamente irrelevante no caso.

- Muitas vezes, na tentativa de se reduzir o espaço

ocupado na memória para executar uma certa tarefa, pode ser encontrada uma organização eficiente dos dados que permita a utilização de rotinas rápidas com baixo gasto de memória. Para isso, deve ser tirado o máximo proveito de características particulares da arquitetura da máquina.

- Deve-se levar sempre em conta que este trabalho é dirigido para um computador pequeno e restrito. Assim sendo, algumas das argumentações aqui apresentadas podem não ser válidas para máquinas maiores, e algumas considerações, decisivas neste contexto, podem tornar-se totalmente irrelevantes se no lugar do Patinho Feio for colocado um computador de parte maior.

CAPÍTULO 2. O MONTADOR

2. O MONTADOR

Neste capítulo é descrito o programa montador desenvolvido para o Patinho Feio, e cuja finalidade é a de permitir que se programe em uma linguagem simbólica próxima de linguagem de máquina deste minicomputador. Parte-se de uma discussão sobre problemas gerais de programação em linguagem de baixo nível, estudando-se a seguir as características desejáveis para o programa montador, ao lado dos problemas enfrentados na implementação das mesmas num computador do porte do Patinho Feio. A seguir , descreve-se alguns detalhes importantes de projeto e de implementação, finalizando com a descrição de alguns recursos de controle do programa e da lógica do montador implementado.

2.1. O conjunto de instruções e a linguagem do montador

2.1.1. O conjunto de instruções: O Patinho Feio possui um conjunto de 56 instruções, descritas quanto ao funcionamento nas referências (1) e (4). Estas instruções podem ser divididas , para efeito dos algoritmos de montagem que são utilizados, em seis grupos, detalhados no apêndice 1:

- instruções de referência à memória;
- instruções de endereçamento imediato;
- instruções de deslocamentos e giros;
- instruções de entrada e saída;
- instruções curtas com operando;
- instruções curtas sem operando.

Todas as instruções pertencentes a um dado grupo, utilizam a mesma rotina de montagem, como será visto em 2.5.3.

2.1.2. Programação em linguagem de máquina: Dado um computador com seu conjunto de instruções, a única maneira de fazê-lo funcionar, se não se dispuser de nenhum outro recurso auxiliar como,por exemplo, um montador em outro computador ("Cross-Assembler"), será programá-lo em linguagem de máquina, isto é, construir uma sequência lógica de zeros e uns, tais que, carregados convenientemente na memória do computador, e a seguir executados, perfaçam a tarefa prevista.

Quando se procede desta maneira, o trabalho de escrever um programa, carregá-lo na memória e em seguida executá-lo é árdua, pois requer muitos cuidados para que o problema que se está atacando seja resolvido corretamente. Assim, dado um problema para ser resolvido em um computador no qual só se dispõe de linguagem de máquina e de um painel, deve-se primeiramente estabelecer um diagrama de blocos da solução do problema, e, em seguida, traduzí-lo para a linguagem de máquina do computador. Como esta tarefa é muito desagradável e trabalhosa para ser efetuada diretamente, convém que o programa não seja imediatamente codificado em linguagem de máquina a partir do diagrama de blocos, mas que se escreva tal programa em uma linguagem intermediária mais acessível, onde cada instrução, que na linguagem de máquina é representada por uma sequência de zeros e uns (que em nada lembram a função que ela deverá executar no programa), será representada por um símbolo, formado de um ou mais caracteres, que de alguma maneira informe ao programador o papel de tal instrução no programa.

Na linguagem simbólica do Patinho Feio, uma instrução qualquer terá no caso mais geral o seguinte atributos (fig.2.1.2.1):

A	B	C	D
INIC	CARI ARM ARM	2Ø CONTA CONTB	(INICIO DO PROGRAMA)

Fig. 2.1.2.1

Exemplo de instruções em linguagem simbólica do montador

A - um endereço, que poderá ser simbólico (optativo);

B - um mnemônico, que indica qual instrução a ser executada;

C - um operando, que completará a função representada pelo mnemônico;

D - um comentário, para efeito de documentação do programa;

Por exemplo, um pequeno programa, escrito numa destas linguagens, poderia ter o seguinte aspecto:

1. carregue a variável X no acumulador (inicialmente X = 16);
2. some 8Ø ao acumulador
3. guarde o resultado em X;
4. pare;
5. desvie incondicionalmente para 1.

Recodificado na linguagem simbólica do Patinho Feio, o programa acima poderá tomar o seguinte aspecto:

```
UM    CAR   X    CARREGA X NO ACUMULADOR
      SOMI  8Ø   SOMA 8Ø AO ACUMULADOR
      ARM   X    ARMAZENA O RESULTADO EM X
      PARE       PARA
      DSVI  UM   DESVIA INCONDICIONALMENTE PARA UM
X     DEFC  16   VALOR INICIAL DE X = 16
```

Consultando uma tabela de correspondência entre as instruções simbólicas e os respectivos códigos de máquina, e atribuindo arbitrariamente o endereço $1ØØ_{16}$ à primeira instrução do programa, poder-se-á recodificar o programa, agora em linguagem de máquina (os códigos abaixo estão em notação hexadecimal):

ENDEREÇO	CÓDIGO	RÓTULO	MNEMÔNICO	OPERANDO	COMENTÁRIO
1ØØ	41Ø9	UM	CAR	X	AC: = X
1Ø2	D85Ø		SOMI	8Ø	AC: = AC + 8Ø
1Ø4	21Ø9		ARM	X	X: = AC
1Ø6	9Ø		PARE		PARE
1Ø7	ØlØ4		DSVI	UM	VOLTE P/UM
1Ø9	1Ø	X	DEFC	16	X (INICIAL) = 16

Este programa, no seu formato definitivo para utilização do montador, poderia ficar com o seguinte aspecto:

```
        ORG     /1ØØ    ESTABELECE ENDEREÇO DE ORIGEM
UM      CAR     X       AC: = X
        SOMI    8Ø      AC: = AC + 8Ø
        ARM     X        X: = AC
        PARE            PARE
        PIA     UM      VOLTE P/UM
X       DEFC    16      X (INICIAL) = 16
        FIM     UM      ESTABELECE FIM DO PROGRAMA
```

As colunas de endereço e de código não aparecem nesta versão.

Observe-se que o programa assim escrito é auto-explicativo, o que não ocorre com o mesmo programa escrito em linguagem de máquina, como se pode notar observando apenas as duas primeiras colunas de penúltima versão.

Como foi visto no exemplo acima, o programa escrito nesta linguagem intermediária pode ter um formato completamente livre, a critério do programador. Se forem estabelecidas algumas regras de sintaxe e padronizações para esta linguagem intermediária, de tal modo que todos os programadores representem com os mesmos símbolos as mesmas operações, terá sido estabelecida uma linguagem de programação para este minicomputador. A esta linguagem dá-se o nome de linguagem do montador ("assembly language").

Desde que a sintaxe da linguagem do montador obedeça fielmente a regras bem definidas, haverá uma relação unívoca entre um programa escrito na linguagem do montador e o programa obtido a partir dele, escrito em linguagem de máquina. Sendo esta montagem algorítmica, poder-se-á escrever um programa relativamente simples que permita a automação desta tarefa de tradução. A este programa dá-se o nome de montador ("assembler").

2.2. A conveniência de um montador

Como foi dito em 2.1., a programação em linguagem de máquina requer inúmeros cuidados para que o programa obtido seja

executado com sucesso. Uma das tarefas mais laboriosas é a da atribuição correta de endereços numéricos aos endereços simbólicos definidos no programa. Isto decorre principalmente pelo fato de que as instruções do computador não ocupam todas o mesmo número de posições de memória, fato que se verifica em todos os computadores cujas instruções não tenham o comprimento uniforme.

Calculados corretamente todos os endereços e montado o programa, este problema surge cada vez que se desejar corrigir o mesmo, nele inserindo, ou dele retirando algumas instruções (por exemplo, quando da sua depuração ou otimização). Nestes casos será sempre necessário recalcular todos os endereços e corrigir todas as referências à memória que houverem sido alteradas pela correção. Além disto, como o comprimento do programa terá mudado, bem como a posição de memória ocupada pelas instruções deverá o programa ser, mais uma vez, carregado na memória integralmente.

Levando em conta que se dispõe apenas de um painel de chaves para a carga de programas, pode-se facilmente verificar que esta tarefa deve ser substituída por outra mais suave e confiável. Se for viável gerar uma fita de papel que possa ser lida por uma leitora de fitas, e que contenha como informações o código de máquina correspondente ao programa que se deseja carregar e executar, o problema estará resolvido. Para isto, será necessário escrever um programa montador, ou em linguagem de máquina, ou então em outro computador.

Optou-se pela alternativa de escrever o montador em linguagem de máquina.

2.2.1 Rotinas auxiliares para a elaboração do montador

Para que o processo da implementação do montador, em linguagem de máquina, se tornasse menos ineficiente, foram escritos, também em linguagem de máquina, alguns programas auxiliares, com a função de atenuar as dificuldades encontradas na operação do sistema:

- um carregador de fitas binárias absolutas, cuja função é a de carregar na memória um programa apresentado em fita

de papel, em formato binário;

- um descarregador de código de máquina para fita de papel, em formato aceito pelo carregador de fitas, cuja finalidade é a de salvar o conteúdo da memória em fita de papel, para posterior utilização;

- um descarregador de memória para o terminal,cuja finalidade é a de informar o estado da memória num certo instante,fornecendo no terminal um mapa com o conteúdo da memória em formato hexadecimal;

- um carregador de memória a partir do teclado do terminal, cuja finalidade é a de evitar o uso do painel de chaves,fornecendo um modo mais cômodo e confiável de carregar programas ou dados na memória, a partir de dados escritos em formato hexadecimal do teclado do terminal.

Com estes novos recursos à disposição (apêndice 2), passou-se a elaborar o montador.

2.3. Definição das Características Gerais do Montador

O primeiro passo para a elaboração do montador foi o de estabelecer algumas regras gramaticais para sua linguagem, bem como escolher os mnemônicos que representariam doravante as instruções de máquina. Além disso, deciciu-se também neste estágio sobre a existência ou não de determinados recursos na linguagem. O resultado desta fase foi o seguinte:

A) o formato de entrada é livre. Um comando genérico consta de campos, os quais são separados obrigatoriamente por brancos, e é terminado pela sequência de caracteres especiais"return", "linedeed";

B) um caráter especial "rubout" em qualquer posição da linha, anula inteiramente esta linha, até que seja encontrada a sequência "return", "linefeed";

C) um asterisco na 1a. coluna signfica que a linha é de comentário;

D) todos os comandos terão basicamente 4 campos:

- campo de rótulos - que começa na primeira coluna, e que é opcional. Isto quer dizer que, se a primeira coluna estiver em branco, o campo dos rótulos está vazio.

Se existir, deve constar de um ponto ou então de uma sequência de letras, de comprimento qualquer, das quais são consideradas, se existirem, apenas as duas primeiras e a última, para efeito de endereçamento simbólico (esta restrição não é uma limitação séria quanto ao número de rótulos, uma vez que se consegue formar, com as 26 letras do alfabeto, segundo a regra descrita , 18278 rótulos diferentes, embora o número máximo de símbolos permitidos em um programa seja de 256).

- campo dos mnemônicos - que se inicia no primeiro caráter não branco encontrado na linha após o primeiro branco e termina no caráter que precede o próximo branco ou "return" da linha. Este campo é obrigatório. São válidos neste campo os mnemônicos definidos no apêndice 1 e que foram inspirados nos da ref. 1. Também neste campo são utilizados para identificação apenas os dois primeiros e o último caráter do mnemônico. Os demais são ignorados.

É este campo que irá definir o comportamento do montador em relação à linha que o contém, estabelecendo se a linha deve gerar duas, uma ou nenhuma palavra de código, se é apenas controle do montador, etc, ou então, no caso de geração de código, definindo qual o algoritmo de montagem a ser utilizado.

- campo dos operandos - este campo está subordinado ao campo dos mnemônicos. Conforme a classe do mnemônico aí encontrado, o campo dos operandos deve ser composto conforme as regras seguintes:

o instruções de referência à memória: exigem operando, que poderá ser:

1) Simbólico puro. Ex:	CAR YA	(qualquer símbolo definido no programa)
2) Simbólico relativo. Ex.:	CAR YA+3	(idem, seguido de deslocamento absoluto)
3) Relativo puro. Ex.:	CAR *+5	(*seguido ou não de deslocamento absoluto)

4) Absoluto.	Ex.:	CAR	/31Ø	(qualquer constante sem sinal ou *-*)
5) Local puro.	Ex.:	CAR	.-2	(.- ou .+ seguido ou não de um digito hexadecimal não nulo)
6) Local relativo.	Ex.:	CAR	.+ -5	(.- ou .+ seguido ou não de um digito hexadecimal não nulo, seguido de deslocamento - absoluto)

• instruções de endereçamento imediato: exigem operando que deverá ser uma constante, com ou sem sinal.

```
Ex.:    CARI    25
        SOMI    +@P
        NAND    -1Ø2
        XOR     -/Ø1
```

• deslocamento e giros: exigem operando constante, entre Ø e 4 inclusive.

```
Ex.:    DD      1
        DE      /Ø1
        GEV     +4
```

• entrada e saída: exigem operando constante. Depois de convertido para binário, o operando tem o significado seguinte: seus 4 primeiros bits indicam o canal utilizado e os 4 últimos o tipo de ação de entrada e saída a ser executada.

```
Ex.:    FNC     /D2
        SAL     /B2
        ENTR    48      (48₁₀ = /3Ø)
        SAI     -48     (-48₁₀ = /DØ)
```

• curtas com operando: exigem operando constante, a saber:

1 - Testes dos flipflops Transbordo (T) e Vai Um (V): o operando deve ser interpretado como booleano (ø ou ≠ ø) :

Ex.:	SVM	ø	≡	SVM ø
	ST	/3	≡	ST 1
	SV	-8	≡	SV 1
	STM	-@P	≡	STM 1

2 - Painel: operando deve ser uma constante entre ø e 7:

Ex.:	PNL	ø	
	PNL	/5	

• curtas sem operando: estas não exigem operando. Se algo for colocado no campo dos operandos, será ignorado.

Ex.:	TRI	*	≡ TRI
	INC	ABCDE	≡ INC
	PARE		≡ PARE

• pseudo-instruções ("pseudo"): cada uma das pseudos exige operando adequado:

NOME, SEGM, SUBR, EXT - Exigem operando simbólico puro

NOME	X
EXT	DRIVER

ENT - Exige operando simbólico puro ou relativo

ENT	ABC
ENT	A+5

ORG - No montador absoluto, (2.4.1), esta pseudo exige operando absoluto se for o 1º ORG do programa. Caso contrário, o operando poderá ser absoluto, simbólico puro, simbólico relativo, relativo puro, local puro ou local relativo.

Exceção. Se for local, não poderá ser .+ n.

No montador relocável: (2.4.2), valem as mesmas considera ções, exceto que o operando não pode ser absoluto. São permitidos apenas os tipos simbólicos puro ou relativo , relativo puro, e local puro ou relativo desde que não seja .+n .

DEFE, DEFI - Exigem operandos simbólicos, absolutos, relativos ou locais.

BLOC, DEFC, COM - Exigem operando constante decimal, ASCII, ou hexadecimal, com ou sem sinal.

EQU - Análogo ao ORG absoluto.

FIM - No montador absoluto (2.4.1), exige operando qualquer. No montador relocável, (2.4.2), o operando indica endere ço de execução se programa principal, ou deve ser zero se for uma subrotina.

- campo dos comentários - no campo dos comentários (opcional) pode figurar qualquer sequência de caracteres que não inclua os caracteres especiais "rubout" "return" ou "linefeed". O campo dos comentários é terminado com a sequência "return""linefeed", a qual também serve para encerrar a linha do comando.

E - qualquer programa deve ser iniciado por um comando de defini ção de origem. Há quatro destes comandos: ORG, NOME, SEGM e SUBR Como será visto em detalhes em 2.5.6, o primeiro é utilizado para definir a origem de programas absolutos, ao passo que os demais servem para associar o endereço relativo "zero" à primeira instrução de um programa principal, segmento ou subro tina, respectivamente;

F - qualquer programa deve ser finalizado por um comando de final de programa. Esta é a pseudo FIM, que, nos programas absolutos e subrotinas relocáveis indica apenas o final físico do pro grama, e, nos relocáveis, associa além disto no caso de progra