

E

A

DE

MARIA ALICE GRIGAS VARELLA FERREIRA

SISTEMA DE PROGRAMAÇÃO "CROSS-MONTADOR" PARA UM MINICOMPUTADOR

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

Área de Concentração - Engenharia
de Eletricidade.

Orientador: Prof. Dr. Tamio Shimizu

São Paulo

-1974-

FD-85

BIBLIOTECA
Dept. Eng. ELETRICIDADE
E. P. U. S. P.

DEDALUS - Acervo - EPEL



31500011832

A meus pais e a meu marido,
cujo apoio e incentivo são
fundamentais em minha car-
reira, dedico este trabalho.

BIBLIOTECA
Dept. Eng. ELETRICIDADE
E. P. U. S. P.

ABSTRACT

In this work we present some views of a parallel development of the software and logical project of a minicomputer. A brief explanation of the two projects is presented.

In the aim to provide the LSD (Laboratório de Sistemas Digitais) minicomputer software designers with a simple and versatile programming system an Assembler language was defined.

The main portion of this work has consisted in the definition and implementation using the "Cross-Translator techniques" of the followings: a relocatable Assembler Translator, a Loader and a Library.

A complete description of the language and machine characteristics are provided in appendices, together with an user's manual.

SINOPSE

Neste trabalho são apresentados alguns aspectos do desenvolvimento em paralelo de projetos de "software" e de projetos lógicos de minicomputadores.

Visando fornecer ao projetista de "software" do minicomputador do LSD um meio de programação simples e versátil, definiu-se uma linguagem de Montador.

A parte principal do trabalho consiste na definição e implementação de um Tradutor "Assembler" relocável para a linguagem definida, de um Carregador e de uma Biblioteca de programas, usando-se a técnica de "Cross-Tradutores".

Uma descrição completa da linguagem definida e das características da máquina são fornecidas nos apêndices, bem como o manual de usuário do sistema.

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

A autora deseja deixar aqui expressos seus agradecimentos a:

- Prof.Dr. Tamio Shimizu, pela orientação deste trabalho.
- Prof.Dr. Antonio Hélio Guerra Vieirae Prof.Dr.Oswaldo Fadigas Torres pelo apoio e orientação.
- Prof. Glen George Langdon,Jr, Dr. Antonio Marcos de Aguirra Massola, Engº Geraldo Lino de Campos,Engº Edson Fregni, e Engº Vitor Mammana de Barros por valiosas sugestões em inúmeros pontos do projeto.
- Engº João José Neto, Engº Benicio José de Souza e Engº Fernando Albuquerque Lins pela sua colaboração neste projeto.
- Engº José Sidnei Colombo Martini e a todo o pessoal do Centro de Processamento de Dados do Laboratório de Sistemas Digitais pelos trabalhos de processamento de dados.
- Srta. Marylúcia Rosa da Silva pelo excelente trabalho de datilografia e impressão.
- a FDTE - FUNDAÇÃO PARA O DESENVOLVIMENTO TECNOLÓGICO DA ENGENHARIA, que através do desenvolvimento de seus projetos nos campos de "hardware" e "software" me proporcionou a oportunidade de enriquecer o presente trabalho em vários pontos, pela observação do problema através de diferentes ângulos.

I N D I C E

<u>INTRODUÇÃO</u>	1
<u>PARTE I</u>	
- O desenvolvimento de "Software" dentro de um projeto de desenvolvimento de um minicomputador	3
<u>PARTE II</u>	
- Linguagem para o "Cross-Montador"	30
<u>PARTE III</u>	
- O "Cross-Montador"	39
<u>PARTE IV</u>	
- Programa Carregador e Sistema de Bibliotecas	106
<u>PARTE V</u>	
- Análise e conclusões sobre o "Cross-Montador" desenvolvido	121
<u>APÊNDICE A</u>	
- Características físicas do minicomputador - Linguagem de máquina	126
<u>APÊNDICE B</u>	
- Descrição da Linguagem de Montagem	152
<u>APÊNDICE C</u>	
- Utilização do Sistema	212
<u>APÊNDICE D</u>	
- Tabelas de erros do "Cross-Montador"	249
<u>APÊNDICE E</u>	
- Tabelas de erros do "Cross-Carregador"	249
<u>APÊNDICE F</u>	
- Tabela de erros da Biblioteca	251
<u>BIBLIOGRAFIA</u>	253

INTRODUÇÃO

O primeiro objetivo desta dissertação é descrever o desenvolvimento de um projeto de "software", para um minicomputador, em paralelo ao seu desenvolvimento de "hardware".

Uma vez definido o conjunto de instruções da máquina, teve início o desenvolvimento do "software", visando implementar programas básicos para ela. A finalidade destes programas é permitir ao usuário utilizar os recursos computacionais de que dispõe.

Quando se deseja executar um programa, este deve residir no computador, na memória principal, em uma forma interna, própria para execução. A correspondência numérica deste código é chamada "Linguagem Interna". O programador, entretanto, usa na codificação de seus trabalhos, sequências de caracteres as quais devem ser convertidas ao formato interno. Para a realização de tal tarefa são elaborados programas especiais que recebem, de maneira geral, o nome de "Tradutores". Dependendo do tipo de tradutor utilizado, outros programas, que fazem a colocação do código produzido na memória do computador, se fazem necessários: são os chamados "Carregadores".

Uma breve explanação sobre os processos de Tradução é apresentada na parte I, visando situar o leitor não especializado dentro do processo e também definir alguns termos que serão adotados no presente trabalho.

Os dois programas acima mencionados constituem o ponto de partida para a elaboração de qualquer outro, para a máquina em questão.

Entretanto, quando um projeto de "software" é desenvolvido em paralelo ao projeto da máquina a qual se destina, uma série de problemas vem dificultar a tarefa: a necessidade de se programar em linguagem de máquina apresenta por si só um grande inconveniente. Por outro lado, a escrita de programas em paralelo com o projeto lógico do sistema, impede que os mesmos sejam submetidos a testes e implantados.

Visando contornar-se tais problemas, algumas técnicas foram estudadas e são discutidas na parte I; dentre elas, tendo-se em conta os recursos disponíveis e os propósitos iniciais do projeto, uma foi escolhida para implantação, e desenvolvida. Vantagens e desvantagens são discutidas.

O processo adotado consiste na elaboração de um Sistema constituído por um "Cross-Montador"/"Cross-Carregador" e por uma Biblioteca de programas. O "Cross-Montador" e o "Cross-Carregador" são, respectivamente, um Montador e um Carregador, que geram, em uma máquina, código para uma outra, já operacional ou em fase de desenvolvimento.

Uma vez escolhida tal diretiva, passou-se à definição de uma "linguagem de montador" para o minicomputador, segundo objetivo deste trabalho. Na parte II, é apresentado um resumo da linguagem adotada, abordando principalmente as razões das escolhas feitas. Uma apresentação completa da linguagem, visando mais a utilização por parte do usuário, compõe o Apêndice B, não sendo incluída na parte II a fim de não interromper a sequência de exposição.

O terceiro objetivo desta dissertação consiste na implantação do sistema escolhido; minúcias desta implementação são apresentadas nas partes III e IV.

Na parte III é apresentado o esquema detalhado do "Cross-Montador" juntamente com fluxogramas do processo.

Na parte IV são apresentados esquemas do "Cross-Carregador" e do Sistema de Biblioteca. O detalhamento deste último é fornecido ao mínimo uma vez que este programa não apresenta técnicas especiais.

O nível de detalhes apresentado no presente trabalho, teve em vista prover uma documentação eficiente para o sistema. Uma documentação mais rica em pormenores acompanha o programa, constituindo o seu manual de programação.

No Apêndice A, são fornecidas as características de arquitetura do mini-computador, bem como suas instruções de linguagem de máquina, fundamentais à implementação dos programas desejados.

No Apêndice C, são descritos processos para a utilização do sistema elaborado, bem como ilustrações destes.

P A R T E I

O DESENVOLVIMENTO DE "SOFTWARE" DENTRO DE UM PROJETO DE DESENVOLVIMENTO DE UM MINICOMPUTADOR

I.1. - Projeto de "hardware" do minicomputador	I.1
I.1.a. - Projeto da memória	I.2
I.1.b. - Projeto da Entrada e Saída	I.3
I.1.c. - Projeto da Unidade Central de Processamento	I.3
I.2. - Escolha de um conjunto de instruções	I.4
I.3. - Desenvolvimento do projeto de "hardware"	I.8
I.4. - Finalidades do projeto de "software"	I.8
I.4.a. - Processos de tradução	I.10
I.4.b. - Objetivos do projeto de "software" a ser desenvolvido	I.11
I.4.b.1. - Tipos de programas montadores e Programa Carregador ou Editor Relocável	I.13
I.4.b.2. - Carregador Absoluto ou "Boots - trap"	I.14
I.5. - Dificuldade na implantação de programas: o programa Simulador	I.14
I.6. - Recursos de implementação de Sistemas de Programação	I.17
I.6.a. - A técnica de "bootstrapping"	I.17
I.6.b. - Linguagens para o desenvolvimento de sistemas	I.18
I.6.c. - Uso de "Cross-Tradutores"	I.20
I.7. - O "Cross-Montador"	I.21
I.7.a. - Vantagens e desvantagens do uso de "Cross-Montador"	I.22
I.7.b. - Escolha da linguagem para a escrita do "Cross-Montador"	I.23
I.8. - "Cross-Montador" para Minicomputador	I.25

P A R T E I

A finalidade desta parte é situar o presente trabalho dentro do esquema do projeto de desenvolvimento de um minicomputador.

Este projeto, iniciado em julho de 1971 pelo Laboratório de Sistemas Digitais do Departamento de Engenharia de Eletricidade da Escola Politécnica da Universidade de São Paulo, teve como objetivo a construção de um minicomputador, provido de uma memória de 4K palavras, de 8 "bits" cada uma.

Ao lado do desenvolvimento de "hardware", desenvolveu-se um projeto de "software", destinado a gerar os programas básicos para esta máquina, visando a sua futura utilização por parte dos programadores usuais. Uma fase intermediária do projeto compreendeu a escolha do conjunto de instruções.

Será aqui descrita a linha cronológica dentro da qual se desenvolveram tais trabalhos, situando-se nela o projeto do Sistema "Cross-Montador"/"Cross-Carregador"/Biblioteca, objeto desta exposição.

Além disso, algumas definições iniciais são colocadas, as quais deverão ser utilizadas durante o correr da explanação.

I.1. - Projeto de "hardware" do minicomputador

Traçadas as linhas gerais do projeto, passou-se a um estudo pormenorizado da organização interna do minicomputador.

A primeira etapa é o estudo dos seus "subsistemas". Chama-se "subsistema" a toda parte capaz de realizar uma tarefa que lhe é destinada, sem interferência das demais. As partes constituintes dos "subsistemas" são chamadas "unidades", que podem ser funcionais, operacionais e lógicas.

Os sub-sistemas são: a memória, a "Entrada e Saída", o Processador Central e o Controle.

O projeto de cada uma destas partes pode ser feito independentemente desde que os vínculos gerais sejam respeitados.

I.1.a.- Projeto da memória

A memória é o meio onde fica armazenada toda a informação de que depende o funcionamento do computador: ela recebe/envia informação do/para o meio externo, armazenando ainda todas as informações do processamento. A quantidade de memória disponível para um sistema pode variar segundo a versão em apreço e o modelo do computador. Um máximo porém é estipulado para cada caso.

A memória é constituída por registros de memória, que são chamados "palavras". As palavras do minicomputador em questão são formadas cada uma por oito "bits". O tamanho da memória é muito importante para o esquema de endereçamento escolhido.

A memória apoia o seu funcionamento em dois registradores de memória: o Registrador de Endereço de Memória (REM) e o Registrador de Dados da Memória (RDM), os quais contém o endereço da palavra de memória cuja informação deve ser manipulada e o dado que deverá ser escrito ou lido na/da memória.

A escolha da capacidade máxima da memória e do tamanho da palavra é o primeiro passo para o projeto de todo o sistema. Existe uma relação ótima entre estes dois parâmetros: se a largura da palavra for muito grande, o número destas que podem ser guardadas na memória diminui muito (uma vez fixada a capacidade desta); por outro lado, se as palavras forem muito pequenas, pode ocorrer que as instruções por elas constituídas sejam pouco poderosas, e que o seu tamanho permita o armazenamento de dados muito pequenos, cuja precisão é insatisfatória.

Ainda, o tamanho da palavra influí diretamente no fluxo dos dados. Se este tiver sua largura já fixada, e for es colhida uma palavra de tamanho grande, a velocidade do sistema será comprometida.

I.1.b. - Projeto_da_Estrada_e_Saída

A Entrada e Saída manipula informação entre o computador e o meio externo. Entrada é o termo dado à retirada de da dos de uma fonte externa e sua transferência para a memória principal. Saída é o termo utilizado ao envio de dados da memória para uma fonte externa. São chamados "Dispositivos de Entrada e Saída" às máquinas que transformam as informações externas fornecidas pelo homem em sinais elétricos que podem ser compreendidos pelo computador e por ele manipulados, e vice-versa.

Chama-se "Registro de Entrada e Saída" a unidade de armazenamento de informação no meio de Entrada e Saída. Os registros de entrada, que foram lidos e processados, não podem alterar o resultado de uma computação já efetuada; este consiste, em geral, de uma sequência de registros de saída.

O projeto de uma "Entrada e Saída", é determinado pelas características estipuladas para o sistema e também por uma série de características dos dispositivos escolhidos. Uma vez determinadas tais características o projeto pode ser desenvolvido paralelamente ao do resto do sistema, constando principalmente da implementação de uma série de circuitos denominados "interfaces".

I.1.c. - Projeto_da_Unidade_Central_de_Processamento

A Unidade Central de Processamento (UCP) comprehende: a Unidade de Aritmética e Lógica (ULA) e a Unidade de Controle (UC). A ULA executa operações de deslocamento, comparação, testes de "bits" e operações aritméticas lógicas. A UC é que regula todo o trabalho que se processa no computador. Ela é quem realiza a sequência de operações através do fluxo dos dados entre as diferentes partes do computador.

O projeto da Unidade Central de Processamento é baseado em uma série de escolhas anteriores: do tipo de controle, de um conjunto de registradores centrais e, principalmente, a escolha de um conjunto de instruções.

O projeto da Unidade Central de Processamento para o Minicomputador encontra-se detalhado no ítem F4 da Bibliografia.

I.2. - Escolha de um conjunto de instruções

Este é um dos passos mais importantes no projeto do computador, uma vez que dele dependem a maioria dos projetos a serem desenvolvidos em paralelo.

A instrução é a unidade básica de trabalho do computador. Os conjuntos de "bits", que a compõe, tem a propriedade de conectar e desconectar determinados circuitos dentro do fluxo dos dados, permitindo a realização de uma sequência bem determinada de operações e transferências de informação dentro do computador.

Vamos chamar de "programa" a um conjunto de instruções dispostas em uma certa ordem e cujo objetivo é realizar uma tarefa escolhida. Um programa que se deseje executar deve ocupar uma determinada porção da memória principal, isto é, deve estar ali residente.

A escolha do tamanho da instrução baseia-se numa série de fatores, tais como os modos de endereçamento possíveis e a potencialidade desejada.

A instrução que vai ser executada é decodificada em um registrador denominado Registrador de Instruções (RI), o qual pertence ao fluxo dos dados. A instrução a ser colocada no RI é aquela cujo endereço se encontra em um registrador denominado "Contador de Instruções" (CI), e cuja função é a de guardar, durante toda a computação, o endereço da instrução que deve ser executada a seguir.

Quando a instrução é executada, o Contador de Instruções, sendo incrementado, aponta o registro de memória consecutivo, de maneira a estar pronto para executar a instrução no registro adjacente, quando a mesma tiver sido completada.

Em princípio, pode-se considerar uma instrução como sendo um arranjo de "bits" do seguinte formato:

código de operação	indicadores	operando
--------------------	-------------	----------

O código de operação determina o conjunto de operações que deverá ser realizado. Os "bits" referidos como Indicadores, servem para especificar opções desta instrução: em geral são usados para indicar tipos de operandos, comprimentos de instruções ou outra especificação qualquer. Os "bits" que constituem o Operando podem designar toda uma classe deles. O "campo de operando", designará, em geral, um endereço de memória no qual o dado que será usado pela instrução pode ser encontrado.

Na maior parte dos computadores as instruções fazem referências a pelo menos um operando. Dependendo do tamanho da memória utilizada, os endereços em jogo podem ser maiores ou menores, necessitando-se, consequentemente, de maior ou menor quantidade de "bits" na instrução. O uso de instruções curtas tem dois efeitos positivos: redução do espaço de armazenamento e um maior número de instruções alcançadas na unidade de tempo. Assim sendo, no caso de memórias grandes recorre-se a técnicas especiais, a fim de diminuir, na instrução, o tamanho dos operandos.

As instruções podem ser classificadas quanto ao endereço segundo dois critérios distintos: quanto ao tipo de endereçamento e quanto ao número de operandos.

Quanto ao tipo de endereçamento podem ser: de endereçamento imediato, direto, relativo, indireto, indexado e implícito.

A escolha do tipo de endereçamento é muito importante no projeto de um computador, uma vez que influi tanto na escolha da instrução quanto no número e tipo de circuitos necessários. Em geral, para o tratamento de endereços, utiliza-se o mesmo conjunto de circuitos empregados nas operações do computador. Assim, é necessário suprir caminhos entre os circuitos que executam operações de transformação de dados e os registradores do sistema de endereçamento.

Quanto ao número de operandos, as instruções podem ser: sem operandos, de um, dois ou três operandos.

As operações de um operando são as mais interessantes no caso de um minicomputador, pois evitam o desperdício dos "bits" que indicariam o segundo operando e que, neste caso, fica implícito (Acumulador). O uso do Acumulador torna superior o desempenho deste tipo de endereçamento, no caso em questão, em relação a um sistema de dois operandos, pois evita o tempo de busca do segundo operando. O uso do segundo tipo de estrutura (dois operandos) é comum em máquinas que manipulam dados de tamanhos variáveis.

As instruções de acordo com suas funções, podem ser divididas em alguns grupos principais: de entrada e saída, de desvios, de movimentação de dados e instruções transformatórias.

As instruções de desvios fornecem ao computador um mecanismo de desvio do caminho natural; estas instruções alteram o valor do Contador de Instruções de maneira que a próxima instrução a ser executada não se encontre no registro de memória a ela adjacente, mas em qualquer outro dentro da memória, aquele apontado pelo valor modificado do Contador de Instruções. Os desvios podem ser de vários tipos: condicionais, incondicionais, retornos de interrupção ou desvios para sub-rotinas.

As instruções de entrada e saída realizam o intercâmbio de informação entre o computador e o meio externo; essas instruções se destinam à utilização dos dispositivos de entrada e saída. De uma maneira geral, podem ser divididas em tres grupos: as de transferência de dados, as funções auxiliares de entrada e saída (que não fazem a transferência de dados, porém se relacionam à utilização dos dispositivos) e as de testes do estado destes dispositivos.

As instruções de movimentação de dados fazem a transferência de dados dentro do computador. As movimentações podem se dar entre duas posições de memória, entre dois registradores ou entre uma posição de memória e um registrador.

As instruções transformatórias são as que implicam na transformação dos dados e sempre dizem respeito ao uso da Unidade Central de Processamento. Podem ser classificadas em dois grupos: as aritméticas e lógicas e as instruções de deslocamento. As primeiras executam as instruções de soma, subtração, multiplicação e divisão, e as operações lógicas realizam as operações "booleanas", de grande importância nas manipulações de "bits". As instruções de tipo deslocamento podem ser simples ou giros. As primeiras fazem com que o valor colocado em uma posição de memória, ou registrador, seja deslocado de um certo número de posições, para a direita ou para a esquerda, dentro desta posição de memória, ou registrador. Os "bits" das extremidades, que durante o deslocamento tem suas posições ocupadas por "bits" deslocados, são perdidos ou recolhidos em outros registradores. As posições esvaziadas durante o deslocamento são preenchidas com zeros, com "bits" previamente escolhidos, ou então permanecem com seu antigo valor. As instruções tipo "giro", não perdem os "bits" das extremidades deslocadas, mas estes são realimentados nas extremidades livres. Estas instruções podem envolver um ou mais registradores ou posições de memória, constituindo um único conjunto.

Além destas instruções mencionadas, que constituem os grandes grupos, existem ainda algumas instruções que podem suspender temporariamente o processamento, ou mesmo encerrá-lo.

A fim de se fazer a escolha do conjunto mínimo de instruções de um dado computador, é necessário ter-se bem determinados os fins aos quais se propõe o projeto; desta maneira podendo incluir, ou excluir, determinadas instruções mais sofisticadas. Um conjunto mínimo é aquele sem o qual o computador não pode executar todas as operações as quais se destina. Isto é, o conjunto mínimo de instruções é aquele constituído pelas instruções obrigatórias. As instruções opcionais são incluídas para melhorar o desempenho da máquina, mas a sua retirada não impediria que funções especificadas fossem realizadas.

Assim, na maior parte das máquinas, um conjunto mínimo é escolhido em primeira fase, dentre os tipos que foram apresentados e, a partir desta escolha, o conjunto é alterado de acordo com as características próprias de cada caso.

O conjunto de instruções do minicomputador foi escolhido desta maneira, por um grupo de engenheiros e estagiários do Laboratório de Sistemas Digitais.

Detalhes deste conjunto, bem como da arquitetura do minicomputador, constituem o Apêndice A.

I.3. - Desenvolvimento do projeto de "hardware"

Definida a arquitetura do sistema e o conjunto mínimo de instruções, pode-se passar ao projeto das instruções de máquina do computador. Uma vez estas definidas, e estabelecidas as características principais do sistema, desenvolvem-se em paralelo os projetos da Unidade Central de Processamento, da Entrada e Saída e da Memória.

Do mesmo modo, pode-se iniciar um projeto de "software", que se desenvolverá em paralelo.

Assim que o projeto lógico dos subsistemas é encerrado, estes devem ser implementados em laboratório, o que constituirá a sua fase final.

I.4. - Finalidade do projeto de "software"

A finalidade de um projeto de "software" é o desenvolvimento de programas, com dois objetivos: tornar ao usuário fácil a escrita de um programa, e tornar fácil o seu processamento, uma vez colocado no computador.

A fim de atingir o primeiro objetivo, foram desenvolvidos os "Sistemas de Programação", constituídos por uma Linguagem de Programação e por um Tradutor específico para esta Linguagem.

Com o seu aparecimento, o processo de Tradução, que até então era efetuado através de métodos mecânicos de conversão, passou a ser efetuado pelo próprio computador, e é normalmente conhecido como "Compilação"; o Tradutor é denominado "Compilador", a linguagem de entrada é dita "Linguagem Fonte" e o código de saída do Compilador é chamado "Linguagem Objeto".

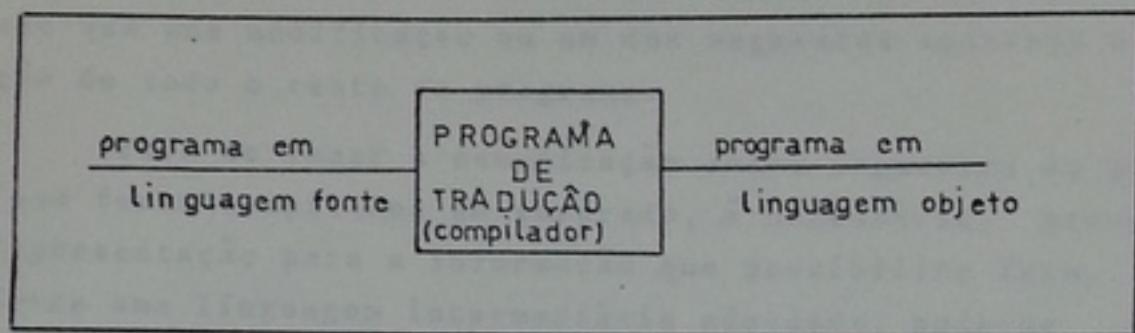
Existem inúmeras Linguagens fonte, em diferentes níveis de complexidade para o programador, destinadas a múltiplas aplicações. O nível de complexidade de uma linguagem é inversamente proporcional à dificuldade de tradução desta linguagem. Assim, as linguagens que oferecem grande facilidade de uso exigem tradutores bastante elaborados.

Um minicomputador de reduzidas proporções, como é o caso exposto, destina-se a aceitar uma linguagem bastante simples, ou mesmo uma linguagem um pouco mais sofisticada, quando dispor de alguns recursos de armazenamento secundário, mas certamente nunca se destinará a fins muito complexos.

Para se atingir o segundo objetivo proposto, isto é, tornar fácil o processamento de um programa, existem algumas diretivas a serem tomadas: a elaboração de um programa de controle, denominado Sistema Operacional, o qual deve fazer o escalonamento das tarefas a serem realizadas pelo computador, melhorando a eficiência de processamento do mesmo; a esquematização de programas que não necessitem ser traduzidos todas as vezes em que se faz necessária a sua execução (geração de código objeto relocável), a providência para que todos os Compiladores gerem códigos objetos do mesmo tipo, o que permite que "segmentos" escritos em diferentes linguagens possam ser ligados, e a criação de um sistema de Biblioteca eficiente.

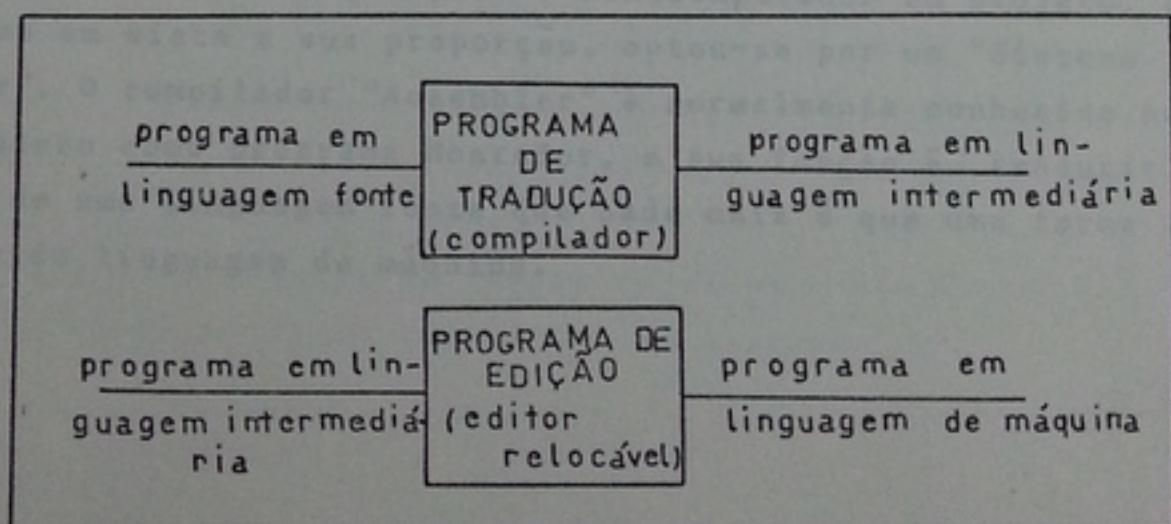
I.4.a. - Processos de tradução

Tendo-se em vista que a função de um Tradutor é a transformação de Linguagem Fonte em Linguagem Interna para o computador, um esquema muito simples seria o seguinte:



Este esquema é aplicado quando os programas traduzidos em linguagem de máquina são executados em seguida. Em muitos casos, porém, os programas traduzidos devem ser armazenados para uso posterior. Convém então que tais programas sejam armazenados na forma de código objeto. Em outros casos, certos parâmetros do programa devem ser supridos apenas no instante da execução. Quando isso ocorre, convém que a tradução não seja feita diretamente para código interno, mas sim para uma "linguagem intermediária". Desta forma o processo consiste de dois estágios: o primeiro, no qual a linguagem fonte é convertida em linguagem intermediária (compilação), e o segundo, em que a linguagem intermediária é transformada em linguagem objeto (carregamento).

O esquema do processo se torna:



A grande vantagem deste processo é que programas podem ser compilados, guardados em forma intermediária e montados posteriormente. Um programa grande normalmente consiste de certo número de segmentos, ou trechos de programa, que interagem entre si e que podem ser programados e modificados independentemente. Com este método, os programas compilados separadamente podem ser ligados entre si apenas no instante do carregamento, evitando que uma modificação em um dos segmentos acarrete a compilação de todo o resto do programa.

Para se fazer a comunicação entre segmentos de programas, que foram compilados em separado, é necessário prover-se uma representação para a informação que possibilite isso. Escolhendo-se uma linguagem intermediária adequada, pode-se escrever linguagens - fonte diferentes que gerem, todas elas, a mesma linguagem intermediária, e os segmentos escritos nas diferentes linguagens, podem ser combinados como um único programa.

Geralmente, quando este esquema é adotado, os segmentos de programa, bem como programas de biblioteca, são disponíveis ao programador na forma intermediária, sendo incorporados ao programa no instante de execução, usando-se as facilidades de comunicação entre módulos expostas acima.

I.4.b. - Objetivos do projeto de "software" a ser desenvolvido

Uma vez constatada a necessidade de elaboração de um Sistema de Programação para o minicomputador em projeto, e tendo-se em vista a sua proporção, optou-se por um "Sistema Assembler". O compilador "Assembler" é normalmente conhecido na literatura como programa Montador, e sua função é traduzir comandos de uma linguagem fonte que nada mais é que uma forma mnemônica da linguagem de máquina.

A cada comando de linguagem fonte, o Montador produz um comando de linguagem objeto. A sua vantagem em relação a linguagem de máquina reside nos seguintes pontos:

- uso de códigos, que lembrem a operação a ser realizada ao invés de códigos numéricos, ao se codificar a instrução;
- o programador está livre do manuseio dos endereços de seu programa, podendo referir-se a eles através de nomes;
- provê ao usuário um meio de documentação eficiente;
- utilizam-se as vantagens da indexação e facilidades de locação de dados e reservas de memória.

A grande importância da linguagem "Assembler" se deve, entretanto, à sua simplicidade, aqui encarada não no sentido de facilidade de uso pelo programador, mas sim na escrita de seu tradutor. As máquinas de pequeno porte possuem, em geral, memórias de reduzido tamanho, as quais não conseguem conter um programa compilador complicado, mas podem muito bem comportar um tradutor simples.

Além disso, grande parte dos compiladores usam montadores para traduzirem o código gerado em uma fase intermediária sendo poucos os que geram diretamente o código final.

Ainda no caso de sistemas pequenos, linguagens de programação sofisticadas se tornam ineficientes, e a programação direta, além de ocupar menos memória, permite uma série de recursos, que apesar de não serem recomendados na maior parte dos casos, é de grande valia nestes: a propriedade dos programas escritos em "Assembler" de alterarem a si próprios.

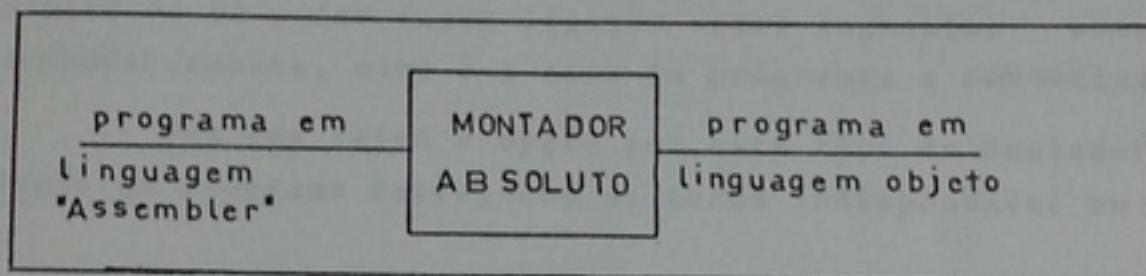
A linguagem "Assembler" tem, entretanto, a sua maior utilização na construção de "Software", caso em que se espera grande eficiência por parte do sistema de programação, uma vez que são as operações de "software" básico que tomam a maior parte do tempo de um computador em operação.

I.4.b.1. - Tipos de programas Montadores e Programa Carregador ou Editor Relocável

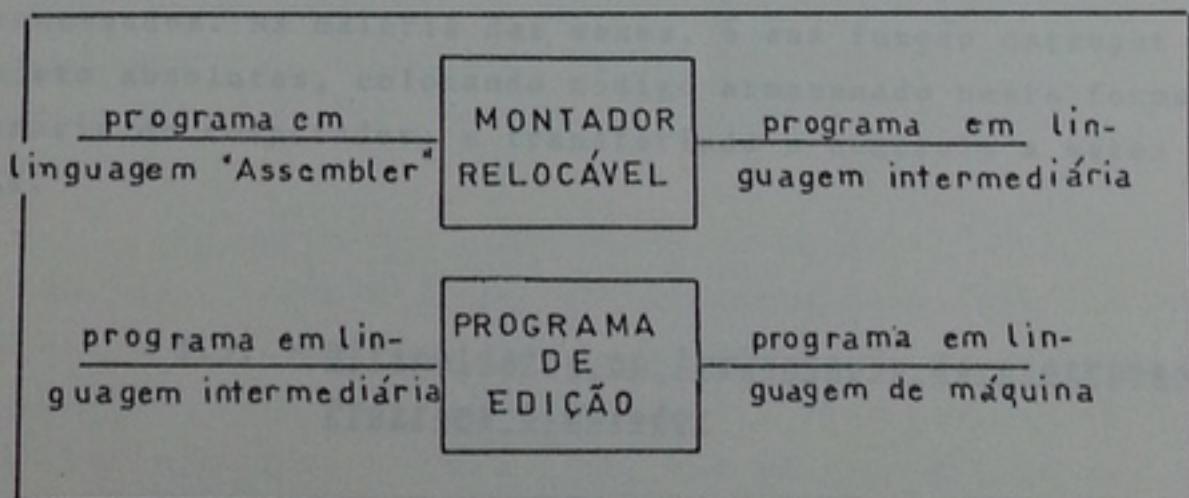
Os programas Montadores podem ser de dois tipos: absolutos e relocáveis.

Chamam-se montadores absolutos aqueles que gerem diretamente linguagem objeto, num processo semelhante ao visto para tradutores em geral.

No caso de montadores absolutos o processo seria o seguinte:



Os montadores relocáveis, por sua vez, geram uma linguagem intermediária, a qual necessita de um carregador para traduzi-la em linguagem objeto. De maneira semelhante ao visto para o sistema tradutor, teríamos neste caso:



Ao programa codificado em linguagem intermediária, costuma-se dar o nome "programa objeto relocável", enquanto que ao programa em linguagem objeto costuma-se chamar "programa objeto absoluto".

No caso em que se dispõe de um programa objeto relocável, para a execução do programa é necessário que este seja convertido em um programa objeto absoluto, requerendo-se então o uso de um outro programa, denominado "Carregador" ou "Programa Editor Relocável". A vantagem deste esquema é que uma vez o programa sendo colocado na forma intermediária e nesta forma armazenado, a fase de tradução se torna dispensável. Outra vantagem é o fato de se poder fazer ligação entre segmentos compilados independentemente, como é o caso de programas e sub-rotinas.

Uma vez feita a opção por este tipo de Montador, a elaboração do Programa Carregador se torna indispensável ao projeto.

I.4.b.2. - Carregador_Absoluto_ou_"Bootstrap"

O Carregador Absoluto, ou "Bootstrap", é fundamental ao sistema de computação. Sua função é iniciar o sistema de maneira que, uma vez executado o computador fique em um estado tal que programas codificados em formato interno podem ser lidos e processados. Na maioria das vezes, é sua função carregar fitas objeto absolutas, colocando código armazenado nesta forma, na memória do computador, e transferindo o controle a estes programas.

I.5. - Dificuldades_na_implementação_de_programas:_____o Programa_Simulador

Neste ponto de definições, em que se determina os objetivos a serem alcançados, deve ser levado em conta um fato: nenhum dos programas desenvolvidos podem ser testados e executados, ainda que escritos em linguagem de máquina.

Isto é devido ao fato da linha de projeto do "Software" estar sendo desenvolvida em paralelo a linha de projeto de "hardware", e portanto, o minicomputador ainda não se encontra operante. Tal fato é um grande obstáculo à elaboração da programação básica e deve ser contornado. A maneira de fazê-lo, é elaborar-se um programa denominado "Simulador".

Em senso geral, Simulador é todo programa desenvolvido com a finalidade de simular o desempenho de um sistema qualquer. No caso em questão, "Simulador" é um programa que simula o funcionamento lógico de um computador, e que é executado em um outro computador qualquer.

Existem programas Simuladores em diferentes graus de complexibilidade, desde os que simulam as operações lógicas mais simples, como os Simuladores de Gates, até os Simuladores em nível de Sistema Computacional, que simulam o funcionamento de um sistema composto de unidades periféricas e processadores.

O Simulador a ser elaborado pertence a categoria dos Simuladores de Instruções, isto é, o comportamento do minicomputador na execução de instruções é a característica desejada neste caso. A entrada para o Simulador consta de uma instrução em Linguagem de Máquina, e ele produz, como saída, os resultados esperados desta execução, isto é, o estado em que o minicomputador se encontra após esta execução.

Para o Simulador, a memória e os registradores do computador sendo simulado são representados por posições da memória principal do computador hospedeiro. Uma série de rotinas simulam a execução das instruções, operando com estas posições de memória.

Consideremos como exemplo a execução de uma instrução de "soma entre dois dados", um deles localizado na memória e o outro localizado em um registrador. Seja o caso em que para a execução desta instrução, o primeiro dado deve ser trazido da memória, somado ao segundo dado, que se encontra no registrador ficando o resultado da operação no registrador. A execução dessa instrução pelo computador consistiria na soma de duas posições de memória do computador hospedeiro:

aquela correspondente à posição de memória do computador simulado e a correspondente ao registrador, ficando o resultado na posição correspondente ao registrador.

Como ilustração a figura I.5.1. apresenta o esquema geral de um Simulador em nível de Instruções.

Um programa Simulador foi desenvolvido para o Computador IBM - 1130 do Laboratório de Sistemas Digitais.

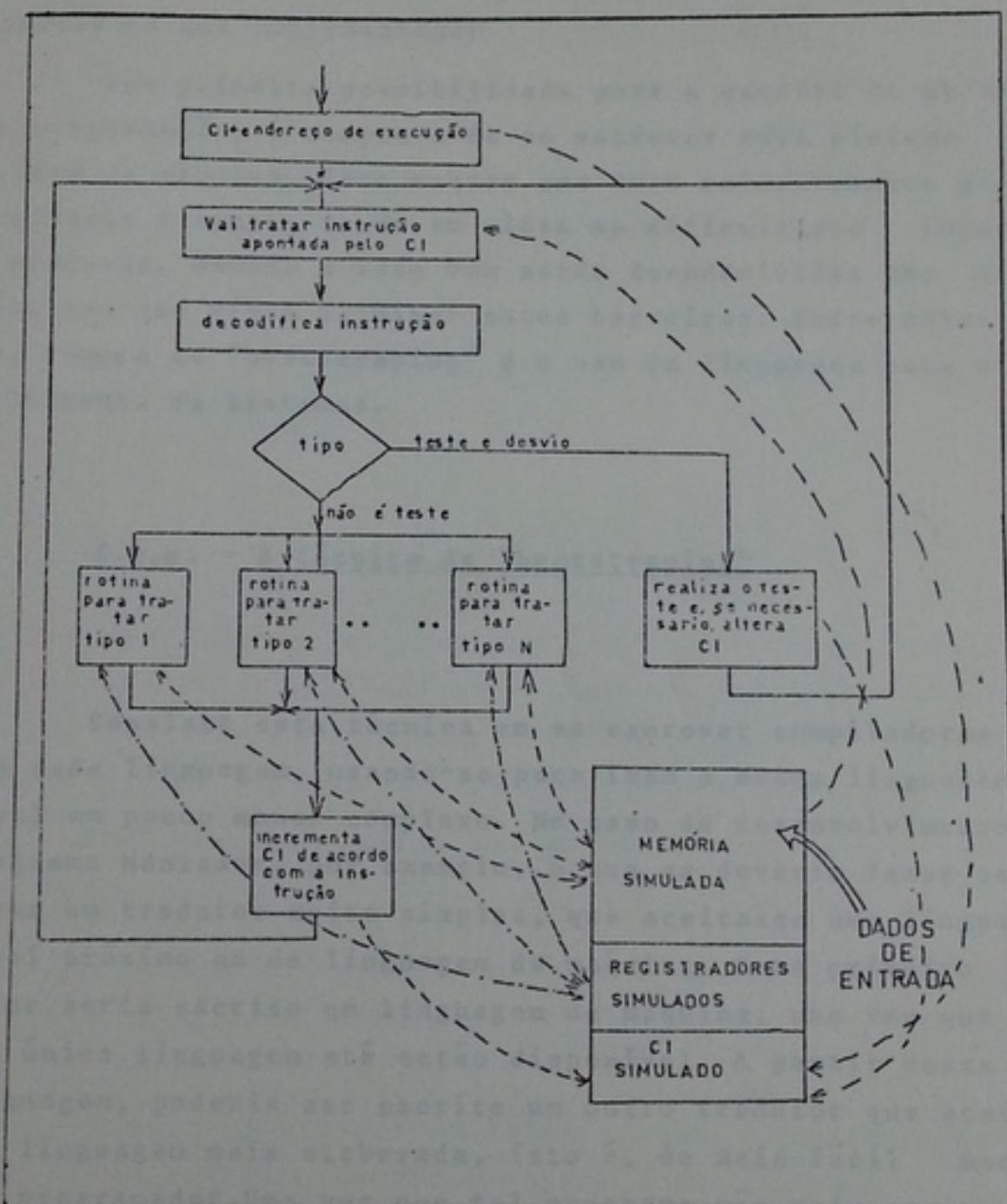


Figura I.5.1. - Esquema de um Simulador com esquema de parte da memória mostrando memória simulada, registradores e contador de instruções.

I.6. - Recursos de Implementação de Sistemas de Programação

Quando se planeja a realização de um novo sistema de programação, é necessário verificar as técnicas que podem ser utilizadas na sua implementação.

Uma primeira possibilidade para a escrita de um sistema de programação, é sempre a de se escrever este sistema em linguagem de máquina. Isto requer uma dose relativamente grande de paciência e tempo, tendo em vista as dificuldades impostas pelo processo. Devido a isso vem sendo desenvolvidas uma série de técnicas que visam eliminar estas barreiras. Entre estas técnicas, temos a de "bootstrapping" e o uso de linguagem para o desenvolvimento de sistemas.

I.6.a. - A técnica de "bootstrapping"

Consiste esta técnica em se escrever compiladores para uma dada linguagem, usando-se para isso a mesma linguagem em um nível um pouco menos complexo. No caso de desenvolvimento de um Programa Montador, por exemplo, o que se deveria fazer seria escrever um tradutor muito simples, que aceitasse uma linguagem de nível próximo ao da linguagem de máquina. Este primeiro processador seria escrito em linguagem de máquina, uma vez que esta é a única linguagem até então disponível. A partir desta nova linguagem, poderia ser escrito um outro tradutor que aceitasse uma linguagem mais elaborada, isto é, de mais fácil manejo para o programador. Uma vez que tal programa não mais seria escrito em linguagem de máquina, a sua elaboração seria feita mais facilmente e em tempo bem menor. Com este processo pode-se alcançar o nível de complexibilidade desejado para uma dada linguagem, ou aquele permitido pelas disponibilidades do sistema em uso.

Este esquema é o proposto para a escrita de um Sistema de Programação usando-se o mesmo computador para o qual este sistema se destina.

Num esquema de projeto como o até aqui desenvolvido, isto se aplicaria do seguinte modo: escrever-se-ia um primeiro programa em linguagem de máquina, destinado a aceitar comandos bem simples (subconjuntos da linguagem "Assembler"). Este programa poderia ser testado e executado através do Simulador. Uma vez correto, usando-se este primeiro subconjunto da linguagem, um novo programa poderia ser escrito e novamente testado até estar perfeitamente correto. O número de vezes em que o processo se deve repetir, depende do nível desejado para a linguagem e os esquemas de implementação empregados.

I.6.b. - Uso de linguagens de desenvolvimento de sistemas

O uso de linguagens para desenvolvimento de sistemas de programação, vem se tornando dia a dia mais comum. Este processo consiste em se utilizar na escrita de tais sistemas, linguagens de alto nível em um modo geral, e linguagens próprias para este tipo de tarefa, em sentido mais restrito.

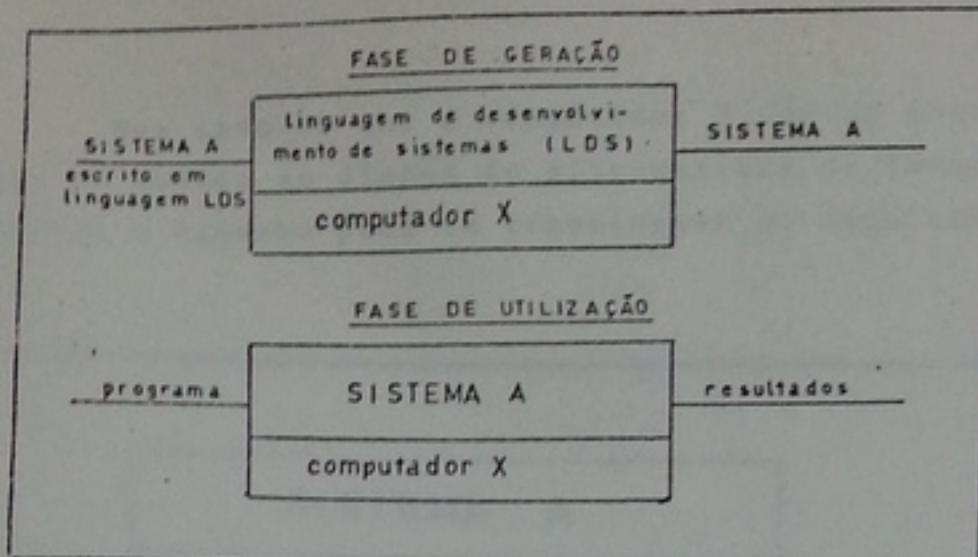
Esta técnica oferece uma série de vantagens:

- facilidades de conversão para outras máquinas, uma vez que as linguagens de alto nível são independentes de máquina;
- ganhos razoáveis em tempo de escrita do sistema e em trabalho de programação, o que permite que o sistema fique operacional em tempo relativamente curto, utilizando-se menos pessoas na sua elaboração, o que apresenta outro fator positivo, que é a facilidade de comunicação entre os membros do projeto;

- facilidades de manutenção do sistema e de documentação;
- facilidades de entendimentos do sistema, na eventual necessidade de ser reprojetado;

As linguagens de desenvolvimento de sistemas são projetadas para satisfazer os mais diversos tipos de necessidades, desde sistemas que necessitam ser totalmente independentes de máquina até aqueles que devem ser particulares para uma dada aplicação, ou computador. O tipo de aplicação é que deve determinar o tipo de linguagem a ser utilizada. Um fator importante nesta escolha é a eficiência final do sistema gerado. Os critérios para a colocação desta eficiência estão relacionados com o fim ao qual eles se destinam. Dependendo dos fins, a eficiência pode até mesmo ser deixada de lado; em outros casos, entretanto, ela é fundamental, tornando o emprego de certas linguagens proibitivos. É importante ter-se em mente que ao se decidir pela escolha de uma linguagem para o desenvolvimento de um sistema, deve-se levar em consideração não apenas a linguagem mas também o compilador existente para esta linguagem, bem como o conhecimento a seu respeito por parte dos programadores encarregados da escrita do sistema. Nos casos em que a linguagem ou o compilador se mostrem ineficientes, então a linguagem "Assembler" deve ser usada como recurso final.

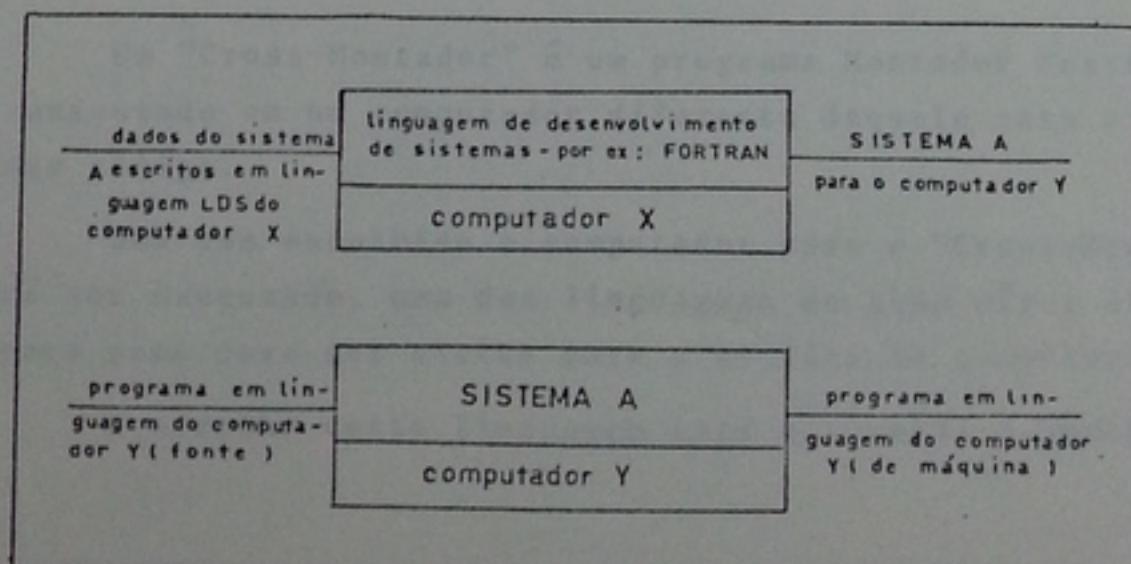
De uma maneira geral, as linguagens de alto nível mais comuns se prestam ao desenvolvimento de sistemas; entretanto, nem sempre elas oferecem os melhores recursos para este tipo de trabalho. Assim sendo, linguagens especiais, baseadas nas linguagens mais conhecidas (PL/I, FORTRAN, ALGOL) vêm sendo desenvolvidas, muitas vezes constituindo extensões destas. Os compiladores para tais linguagens estão sendo elaborados em diversos centros de pesquisa, escritos geralmente para computadores de grande porte e que possuam sistemas operacionais poderosos. Estes desenvolvimentos pretendem fornecer ao programador de sistemas uma série de facilidades de testes, constituindo ferramentas indispensáveis aos trabalhos futuros neste campo.



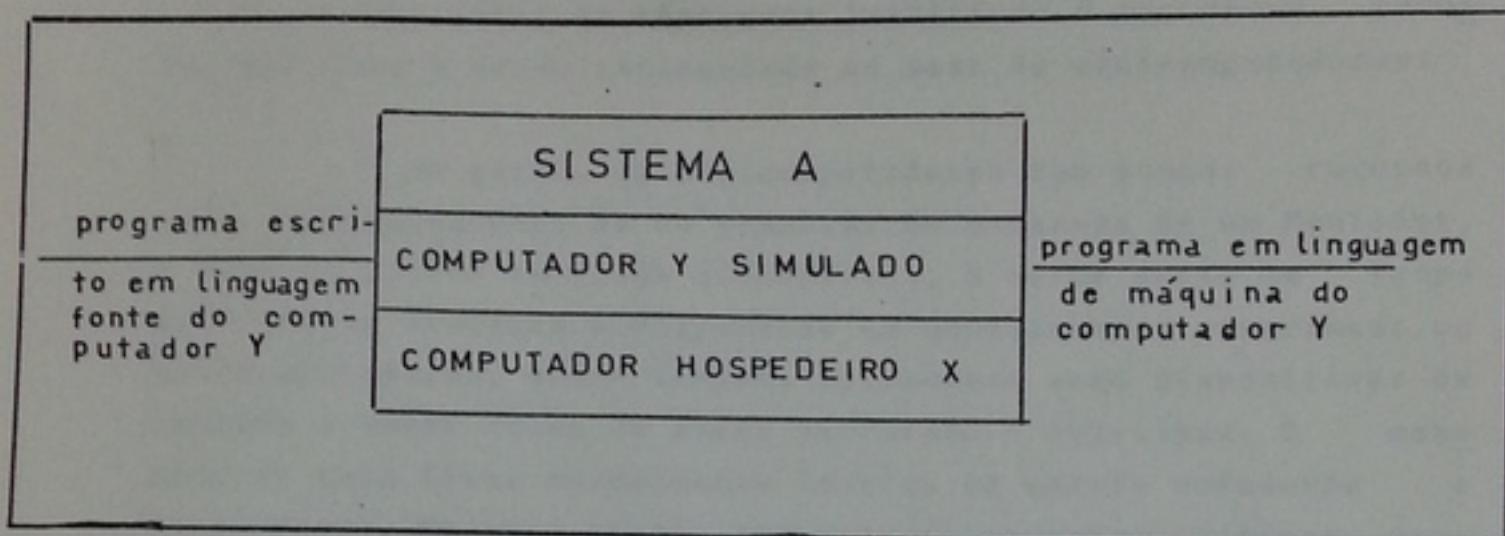
I.6.c. - Uso de "Cross-Tradutores"

Uma terceira técnica, derivada da anterior, consiste em se escrever um compilador final para uma dada linguagem, utilizando para isso um computador diferente daquele em que o modelo é implantado. Isto significa que o Tradutor desenvolvido gera código para uma máquina diferente daquela na qual ele é processado. A diferença entre este tipo de Tradutor e os desenvolvidos através de LDS, é que ele gera código para apenas uma máquina, aquela para a qual ele foi desenvolvido, e que a compilação de um programa se dá em um computador, enquanto a sua execução só pode ser efetuada no outro.

O esquema deste tipo de técnica, é o seguinte:



Nos caso em que o computador Y não se encontra operacional, ou quando se dispõe de alternativas de testes do tipo Simulador, o esquema pode se transformar no seguinte:



No caso particular em que a linguagem a ser processada é uma linguagem tipo "Assembler" ou de Montagem, o "Cross-Trator" é conhecido como "Cross-Montador". Nos caso em que a linguagem a ser processada é de alto nível, ele recebe o nome de "Cross-Compilador".

I.7. - O "Cross-Montador"

Um "Cross-Montador" é um programa Montador destinado a ser executado em um computador diferente daquele para o qual ele gera código.

Uma vez escolhido o computador onde o "Cross-Montador" deverá ser executado, uma das linguagens de alto nível disponíveis para este deve ser eleita para a escrita do programa.

A escolha desta linguagem será discutida a seguir.

I.7.a. - Vantagens e desvantagens do uso do "Cross-Montador"

Uma série de vantagens justificam o uso desta técnica, bem como a fazem recomendada no caso de minicomputadores:

- em geral, os minicomputadores tem poucos recursos para o processamento de um programa da natureza de um Montador, e como os parcos recursos disponíveis, a maior parte do tempo útil destes sistemas é dispensado em montagens de programas; os minicomputadores, quase sempre, apresentam como dispositivos de entrada e saída fitas de papel perfurado e teletipos. O manuseio de tais fitas normalmente implica em tarefa enfadonha e dispendiosa. Em tais casos, torna-se compensador utilizar, para a tradução de programas, computadores de maior porte, providos, de maiores recursos e capazes de executarem essas traduções em poucos minutos, utilizando outros recursos de entrada e saída, como fitas e discos magnéticos. Em tal esquema, o computador fica incumbido de executar apenas programas já prontos, utilizando seu tempo em trabalhos realmente produtivos;

- no desenvolvimento de novos sistemas de computação, os "Cross-Montadores" são recursos poderosos na elaboração de "software" básico, enquanto o computador não se encontra em estado de entrar em operação. Nestes casos, com a utilização associada de um Programa Simulador, os programas elaborados para o novo computador podem ser desenvolvidas sem problemas maiores;

- no caso de minicomputadores com memórias muito reduzidas, e destinados a aplicações bem específicas, a utilização de um "Cross-Montador" é muitas vezes o único recurso de que se dispõe para programação, além da linguagem de máquina. Outras vezes, o conjunto de instruções da máquina é que não permite a escrita de um tradutor eficiente, devendo o programador utilizar a linguagem de máquina diretamente.

As principais desvantagens dos "Cross-Montadores" dizem respeito principalmente à escolha das linguagens utilizadas, e à implementações destas linguagens. Deve-se levar em conta, que os "Cross-Montadores" disponíveis para um dado computador devem ser utilizados por toda uma série de usuários deste, e que muitas vezes, nem todos eles possuem um outro computador maior disponível. Em tais casos estes usuários devem se contentar com os demais recursos apresentados pelo minicomputador e esquecer o "Cross-Montador".

Nos casos entretanto em que se pretende utilizar este recurso, mas a máquina disponível não é a mesma para a qual o "Cross-Montador" foi escrito, possuindo entretanto a mesma linguagem, a implementação das duas linguagens deve ser verificada bem como algumas características das duas máquinas: devem ser verificadas a reserva de área para dados, a maneira de se armazenar matrizes, o manuseio de arquivos e o tamanho da palavra de memória.

Problemas maiores, ainda, podem ser causados devido a limitação de certas capacidades, como o tamanho da memória e os dispositivos de entrada e saída empregados. Quando alguma destas capacidades é excedida, o "Cross-Montador" deve ser reconfigurado, reduzindo-se o tamanho das tabelas, alterando-se os procedimentos de entrada e saída ou mesmo eliminando-se algumas características da linguagem.

I.7.b. - Escolha da linguagem para a escrita do "Cross-Montador"

A maior parte dos "Cross-Montadores" existentes na prática estão codificados em linguagens de alto nível, tendo-se em vista a sua rápida implantação, bem como sua utilização por uma gama bastante ampla de computadores diferentes. Uma das linguagens mais utilizadas é o FORTRAN, uma vez tratar-se da linguagem mais difundida (estatisticamente, segundo Jean Sammet, existem maior número de máquinas FORTRAN do que de outro tipo).

Um "Cross - Montador", codificado em uma linguagem de alto nível, é maior em tamanho do que um codificado em linguagem de máquina, ou uma tipo "Assembler" e como consequência lógica de tal fato, deve ser um pouco mais lento. Por outro lado, o esforço dispendido em se produzir o segundo, é aproximadamente o dobro do esforço em produzir o primeiro, acontecendo ainda que este segundo será restrito a um tipo específico de máquina.

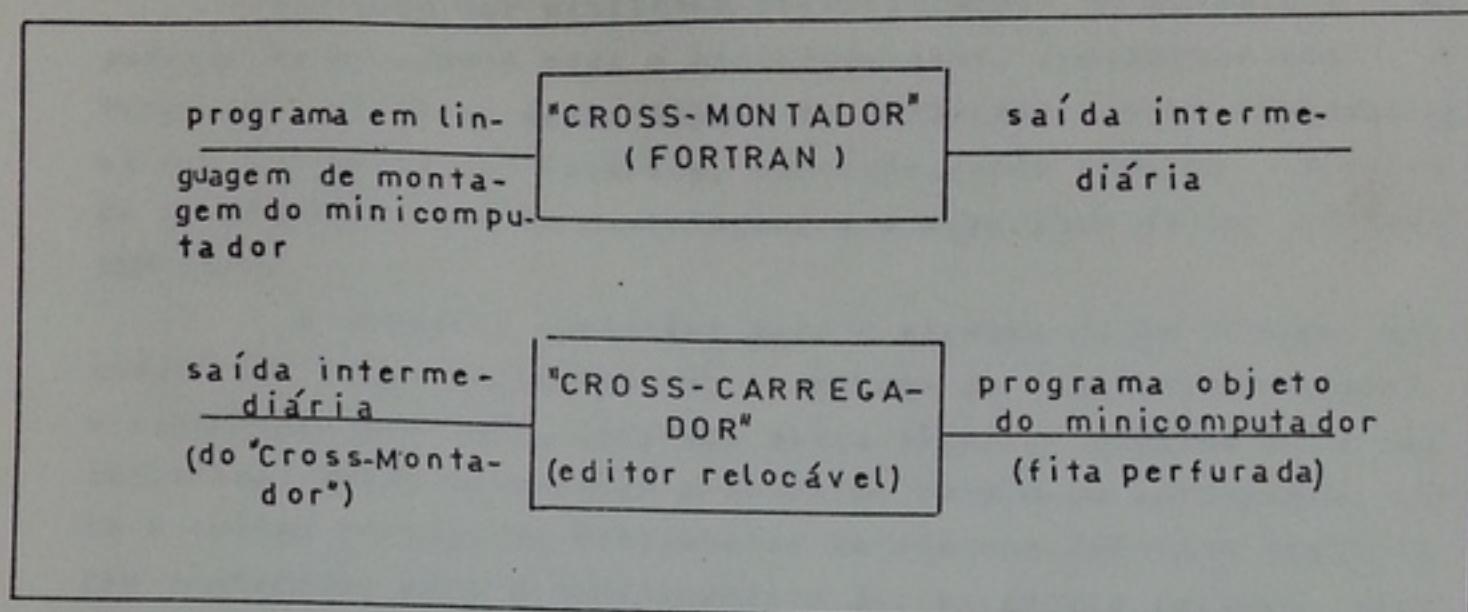
Uma medida bastante eficiente para a produção de "Cross-Montadores" diz respeito ao uso de "Macro-Assembler" para este fim. A técnica é bastante eficiente, principalmente em casos de sistemas que não possuem linguagens de alto nível disponível. Consiste agora o processo não em se produzir um programa final "Cross-Montador", mas em se utilizar o próprio "Macro-Assembler" do computador, e escrever uma série de macro-rotinas, que gerem as instruções do minicomputador através do Assembler do computador hospedeiro. As "macros" são "chamadas" através dos mnemônicos da linguagem de "Cross-Assembler". A técnica é bastante econômica, tendo ainda a vantagem de que todas as facilidades do sistema hospedeiro estão disponíveis, como por exemplo a utilização das próprias bibliotecas do "Macro-Assembler".

Uma outra consideração a ser feita é a questão de compatibilidade quanto às entradas e saídas do "Cross-Montador" e às dos demais Montadores disponíveis para a máquina.

A compatibilidade deve ser total entre os programas. O "Cross-Montador" deve aceitar as mesmas entradas e produzir as mesmas saídas que os demais Montadores. As incompatibilidades mais frequentes dizem respeito as saídas dos sistemas, principalmente nos caso em que se utiliza a linguagem FORTRAN, que não produz, em geral, fitas de papel do tipo requerido como entradas de minicomputadores.

Quando isso ocorre, um Carregador especial necessita ser desenvolvido para a saída do "Cross-Montador", ou então um programa "Reformulador", que traduza as fitas em formato conveniente para execução. A escolha de um programa Carregador parece ser a solução mais interessante quando possível, possibilitando o uso de uma biblioteca para programas e sub-rotinas e a utilização de programas relocáveis.

Um processo deste tipo seria representado pelo seguinte esquema:



À medida que os computadores se tornam menores e cada vez mais especializados, é maior o interesse concentrado em torno de tecnologias deste tipo. O emprego de computadores de grande porte na preparação de "software" para minicomputadores, está se tornando dia a dia mais comum. A tendência atual é fornecer, juntamente com o "Cross-Montador", um programa Simulador do Minicomputador, afim de que não apenas a montagem dos programas seja efetuada neste sistema maior, mas também o seu "debug", de maneira a que somente após o programa estar funcionando perfeitamente ele é transplantado para o seu sistema original.

I.8. - "Cross-Montador" para o Minicomputador

Para a escolha da técnica de "Cross-Montador", como processo intermediário de geração de "software" para o minicomputador, foram considerados os seguintes pontos:

- permite o desenvolvimento de "software" em paralelo ao desenvolvimento de "hardware";

- pode ser utilizado posteriormente no preparo e de puração de programas para o minicomputador, juntamente com o Programa Simulador, mesmo depois do computador estar liberado para uso por parte dos usuários, servindo então como um suporte de programação para as instalações que disponham de um sistema IBM-1130.

A primeira tentativa para a escrita de um "Cross-Montador" foi feita utilizando-se o emprego de "macro-instruções". O computador IBM-1130, dispondendo desta técnica, prestou-se a tal tentativa, sendo na ocasião construído também um carregador para o código gerado. As bibliotecas do sistema IBM-1130 (LET) foram empregadas para o armazenamento dos programas gerados através do "Macro-Assembler". Entretanto esta técnica mostrou-se inefficiente, devido ao tempo dispendido na compilação de programas (o "Macro-Assembler" oferecido pela IBM para o sistema 1130 compila programas muito lentamente, e a compilação dos programas do minicomputador, sendo constituídas unicamente por "macros" que devem ser expandidas, se tornou proibitiva).

Assim sendo, decidiu-se pela escrita de um novo sistema de "Cross-Montador", desenvolvido agora em uma linguagem de alto nível, a fim de tornar mais rápida a sua implantação. Visando a utilização do "Cross-Carregador" já elaborado e do Simulador, este novo "Cross-Montador" foi também desenvolvido para o computador IBM-1130. Posteriormente foi desenvolvido um sistema de Bibliotecas, a fim de armazenar apenas os programas para o minicomputador, e um novo "Cross-Carregador", o qual utiliza esta Biblioteca.

Entre as linguagens de alto nível oferecidas pela IBM para o computador IBM-1130, estão o FORTRAN e o ALGOL. Destas linguagens o Algol é bastante deficitário, utilizando uma entrada de difícil manuseio; desta maneira a escolha recaiu sobre o FORTRAN, que apesar de apresentar algumas deficiências, como a impossibilidade de manuseio de "bits", apresenta a vantagem de ser facilmente convertido para outras máquinas e bastante difundido. Os problemas apresentados foram contornados através do emprego de sub-rotinas escritas em "Assembler".

PARTE II

LINGUAGEM PARA O "CROSS-MONTADOR"

II.1. - Estrutura da linguagem	II.1
II.2. - Escolhas dos tipos de operandos	II.4
II.3. - Escolha de um conjunto de "pseudos"	II.5
II.3.a. - Definição de dados através de "pseudos"	II.5
II.3.b. - "Pseudos" de tratamento de subrotinas	II.6
II.3.c. - "Pseudo" de definição de origem de programa	II.7
II.3.d. - "Pseudo" de definição de fim de programa	II.7
II.3.e. - "Pseudo" para equivalência de nomes	II.7
II.3.f. - "Pseudos" de controle de listagem	II.8
II.4. - Escolha dos "mnemônicos"	II.8

coluna do cartão	conteúdo
1 a 20	brancos
21 a 23	campo de rotulo
24	branco
25 a 30	campo de operando
31	branco
32	campo de identificadores
33 a 34	brancos
35 a 71	campo de operando e rotulos
72 a 80	campo de identificadores

P A R T E I I

Nesta parte será feito um breve comentário sobre a linguagem "Assembler" ou de Montador definida para o "Cross-Montador". Maiores detalhes, bem como exemplos de uso das instruções, encontram-se no Apêndice B, onde procurou-se organizar os mnemônicos segundo suas funções, tendo em vista a facilidade de uso para programação.

II.1. - Estrutura da Linguagem

A estrutura da linguagem é baseada em uma série de campos; cada campo é constituído por um conjunto de caracteres, que o Programa "Cross-Montador" deve reconhecer, a fim de montar de maneira correta a instrução.

A linguagem é de formato fixo, isto é, cada campo ocupa uma posição bem definida em relação ao início do registro de entrada.

O formato deste registro de entrada é o seguinte:

coluna do cartão	conteúdo
1 a 20	brancos
21 a 25	campo de rótulo
26	branco
27 a 30	campo de operação
31	branco
32	campo de indireto
33 e 34	brancos
35 a 71	campo de operandos e comentários
72 a 80	campo de identificação

Campo de rótulo

O campo de rótulo, se usado, necessita iniciar na coluna 21, e pode ter de um a cinco caracteres. O rótulo é um símbolo, e deve ser iniciado por um caracter alfabético, \$ ou @ ou = seguido de uma combinação de um a quatro caracteres alfabéticos ou numéricos. Em geral, um rótulo é atribuído a uma instrução a fim de ela possa mais tarde ser referenciada em outra instrução. Outro uso de um campo rotulado, é demarcar o início de uma determinada porção do programa.

Campo de operando e comentários

Consiste este campo, de dois sub-campos separados um do outro através de pelo menos um branco. O campo de operando se inicia na coluna 35.

Um operando é uma expressão constituída de elementos e de operadores. Uma instrução pode ter um ou mais operandos, quando então estes são separados por vírgulas. Não são permitidos brancos entre operandos.

Um elemento é um símbolo ou um valor definido por si mesmo. Um símbolo, para ser um operando válido, deve ser definido num campo de rótulo. Um valor auto-definido pode ser um decimal, um hexadecimal ou uma cadeia de caracteres. Considera-se valor decimal, um inteiro na faixa 0 a 4095. Um valor hexadecimal é um número inteiro na faixa 000 a FFF em base 16. Identifica-se um valor hexadecimal pela colocação de uma barra antes do número. Por exemplo, teríamos: /100, para a definição do hexadecimal 100. Uma cadeia de caracteres é identificada por uma cadeia de caracteres alfabéticos, numéricos ou especiais, compreendidos entre pontos (.). Por exemplo, poderíamos ter .ABCD. como definição da cadeia composta pelos símbolos alfabéticos A, B, C e D.

Se na coluna 35, do campo de operando aparecer um as terístico (*) em lugar de um elemento ou de um valor, a opera ção é relativa ao Contador de Instruções.

Um operador é um caracter que representa a operação aritmética a ser efetuada, na avaliação do operando. Os operado res permitidos são: + (mais) e - (menos), que indicam respecti vamente: adição e subtração.

Um comentário pode ser colocado em cada instrução; ele é uma breve descrição da instrução. Os comentários são colo cados logo em seguida ao operando da instrução, separados deste por brancos, sendo sua finalidade simplesmente melhorar a docu mentação do programa . Eles não são analisados pelo programa "Cross-Montador".

Se o comentário for muito longo e não couber em uma linha, cartões especiais podem ser usados para codificá-lo. Co mentários deste tipo são identificados por um "*" na coluna 21 do cartão. Após o asterístico, todas as colunas até a de número 71 podem ser utilizadas.

Campo de identificação

As colunas 72 a 80 do cartão são usadas para se fazer a identificação do programa e numerar os cartões.

Campo de indireto

A coluna 32 do cartão constitui o campo de indireto. A codificação da letra "I" neste campo indica modo indireto de endereçamento. Este modo é válido somente para as instruções de referências à memória.

A escolha de um formato de instrução fixo é bastante interessante no que se refere ao manuseio de cartões, pois além de manter uma certa uniformidade da escrita, permite mais flexibilidade por parte das rotinas tratadoras dos campos. Por outro lado, formato de campos fixos, não se aplicam convenientemente no que se refere ao emprego de outros dispositivos de entrada como leitoras de fita perfurada ou "teletypes". Entretanto, des de que o Programa Montador, escrito para o próprio minicomputador, aceita linguagem livre, (o que normalmente deve ocorrer ao se lidar com aqueles dispositivos), os programas gerados nesta fase seriam compatíveis com este "Montador" final.

Um motivo bastante forte que levou a se manter um formato fixo, e do tipo escolhido, é que, inicialmente, foi escrito um primeiro Montador usando-se os recursos de "Macro - Assembler" do IBM-1130. Este Montador exigia formato fixo para os programas escritos para o minicomputador, visto este formato ser exigido pelo referido "Macro-Assembler". Assim sendo, decidiu -se manter a forma desta primeira versão da linguagem, com a finalidade de se fazer a compatibilidade com os programas já escritos.

II.2. - Escolha dos tipos de operandos

Tendo por base os modos de endereçamento possíveis para o minicomputador, os seguintes tipos de endereçamento são permitidos para a linguagem "Assembler": absoluto, simbólico, imediato, relativo e indireto. Tais endereçamentos são permitidos para as instruções de endereçamento da memória e de endereçamento imediato.

A escolha dos tipos de operandos descritos baseou-se principalmente nas opções permitidas pelas instruções do Minicomputador. Elas estão intimamente relacionadas com os tipos de endereçamentos de "hardware". Sendo o "Cross-Montador" um gerador de código relocável, em muitos casos é necessário referir -se a endereços absolutos de memória: a liberdade de escolha entre endereçamentos absolutos hexadecimais e decimais é uma grande vantagem para o usuário, sendo a ele mais fácil usar a primeira representação em determinados casos, e a segunda em outros.

Por outro lado, o uso de endereçamentos simbólicos relativos é de grande importância para a linguagem; já o endereçamento relativo a si próprio tem importância secundária, sendo interessante porém em alguns casos. O uso de expressões envolvendo endereços de símbolos é bastante restrito, sendo incorporado devido ao pouco esforço necessário a sua implementação.

II.3. - Escolha de um conjunto de Pseudo-Instruções ou "pseudos"

As pseudos-instruções implementadas constituem o conjunto clássico de comandos deste gênero numa linguagem tipo "Assembler". As "pseudos" condicionais não foram incluídas, pois não era idéia gerar-se um "Cross-Montador" para "Macros"; por outro lado, "pseudos" demasiado complicadas não foram acrescentadas, pois certamente não constariam da versão final do Montador, devido às reduzidas dimensões da máquina a que se destina.

II.3.a. - Definição de dados através de "pseudos"

As instruções de referência à memória lidam com dados que são reconhecidos pelo "Cross-Montador", através de nomes, no maior número dos casos.

O "Cross-Montador" exige que todo símbolo que apareça no campo de operando apareça também em campo de rótulo. Isto implica em que toda variável ou constante deve ser definida pelo programador, em algum ponto do programa.

Os dados foram classificados em quatro grupos: variáveis simples inteiras, vetores inteiros, cadeia de caracteres e endereços.

Esta diversificação foi feita no intuito de simplificar as rotinas tratadoras, tratando dados de tipos diferentes através de rotinas diferentes.

As variáveis simples inteiras podem ser especificadas na forma decimal ou hexadecimal. Elas se encontram em um grupo separado daquele dos endereços, devido a utilizarem apenas uma palavra de memória, enquanto os endereços utilizam duas.

As "pseudos" de definição de endereço aceitam a definição feita em qualquer dos modos próprios de endereçamento, exceção feita, naturalmente, aos endereçamentos indiretos.

A definição de cadeias de caracteres permite a definição de uma cadeia com comprimento máximo de 35 caracteres, o máximo possível de ser contido em um cartão. Não são permitidas cadeias binárias.

"Pseudos" para definição de dados na forma dupla ou flutuante não foram implementadas devido ao fato de que a primeira pode ser facilmente definida através de representação hexadecimal, e a segunda não estar definida na sua forma final.

A definição de vetores permite a reserva de um conjunto de registros consecutivos de memória; a primeira posição do bloco de dados pode ou não ter um nome especificado.

II.3.b. - "Pseudos" de tratamento de sub-rotinas

Têm este tipo de estrutura de dados a função de auxiliar a definição e operação de sub-rotinas. O ponto de execução de uma rotina deve ser definido através de um "pseudo".

Neste grupo estão as "pseudos" de declaração de pontos de entrada de sub-rotinas e as de chamada de sub-rotinas. A escolha destas "pseudos" baseou-se principalmente na tentativa de uso de "Macro-Assembler" para a geração de código para o Minicomputador. Assim, as "pseudos" escolhidas são baseadas naquelas utilizadas pelo "Macro-Assembler" do Sistema IBM-1130.

As "pseudos" de definição de pontos de entrada das sub-rotinas são importantes devido a caracterização tipo de sub-rotina utilizada. Dois tipos de subrotinas podem ser caracterizados: as de entrada e saída e as outras, diferenciadas entre si através de duas "pseudos" diferentes.

As chamadas de sub-rotinas são feitas através de uma única "pseudo". Neste caso, o nome é limitado a quatro caracteres, e a sub-rotina deve ter sido definida como nome externo.

II.3.c. - "Pseudos" de definição de origem de programa
ma

Um programa pode ser carregado a partir de uma posição fixa de memória. Para isso, uma "pseudo" é usada para indicar ao "Cross-Montador" que a próxima seção de código deve ser carregada em posições de memória a partir daquela indicada em seu operando. Estas posições são absolutas, não se permitindo, neste campo, símbolos, como ocorre em alguns Montadores.

II.3.d. - "Pseudo" de definição de fim de programa

Uma "pseudo" marca o fim de um segmento de código. Se este for um programa principal, a "pseudo" serve também para definir o ponto de execução deste programa. O ponto de execução deve ser definido através de um nome, rótulo de uma instrução executável do programa. Se o programa for uma sub-rotina, o nome se aparecer, é ignorado.

II.3.e. - "Pseudo" para equivalência de nomes

Esta "pseudo" indica ao "Cross-Montador" que dois símbolos são definidos como sendo a mesma posição de memória. Uma restrição é imposta quanto ao uso desta "pseudo": no caso em que se faz a equivalência entre um símbolo e uma expressão os elementos que constituem a expressão já devem ter sido definidos.

II.3.f. - "Pseudo" para a definição de símbolos em
área comum

Quando módulos de programas são compilados e carregados separadamente, surge o problema de se utilizar variáveis comuns entre eles. Um meio de se fazer isso é transferir estas variáveis entre os módulos, como parâmetros, nas chamadas de sub-rotinas. Um outro processo bem mais simples é definir estes símbolos comuns em uma área especial: a área de "common". O endereçamento de variáveis nesta área é sequencial, isto é, as variáveis vão sendo alocadas à medida que as "pseudos" aparecem. A equivalência é portanto posicional. As variáveis são dispostas no fim da memória.

II.3.g. - "Pseudos" de controle da listagem

Um conjunto de "pseudos" foi escolhido a fim de controlar a listagem de saída do "Cross-Montador". Estas "pseudos" fazem a mudança de página de listagem, colocam cabeçalhos, suprimem parte da listagem ou pulam linhas sob o controle do programador.

II.4. - Escolha dos mnemônicos

Os mnemônicos foram escolhidos de maneira a lembrarem sempre a função da instrução. Assim, o mnemônico SOM visa lembrar a operação de SOMA. Algumas vezes algo é mais acrescentado a fim de diferenciar opções da instrução: assim SOMI indica SOMA imediata, uma das opções, e SOMX, indica SOMA INDEXADA, a outra opção da instrução de SOMA.

PARTE III

O --- "CROSS" ----- MONTADOR"

III.1. - Tradução da linguagem Assembler em linguagem de máquina.	III.1
III.2. - Organização do Processador	III.2
III.2.a. - Fase de locação ou primeiro passo	III.5
III.2.b. - Fase de geração ou segundo passo	III.10
III.3. - Processamento dos Mnemônicos	III.13
III.3.a. - Processamento do grupo de endereçamento da memória	III.20
III.3.b. - Processamento das instruções imediatas	III.22
III.3.c. - Processamento das instruções de deslocamento	III.23
III.3.d. - Processamento das instruções de entrada e saída	III.23
III.3.e. - Processamento das instruções de saltos	III.25
III.3.f. - Processamento das instruções de painel	III.26
III.3.g. - Processamento da instrução NOP	III.26
III.3.h. - Processamento da instrução LIMP	III.27
III.3.i. - Processamento das demais instruções	III.27
III.4. - Tratamento das pseudo-instruções	III.27
III.4.a. - Processamento de "bloc"	III.29
III.4.b. - Processamento de "EQU"	III.31
III.4.c. - Processamento de "DEFC"	III.34
III.4.d. - Processamento de "DEFE"	III.36
III.4.e. - Processamento de "COM"	III.38
III.4.f. - Processamento de "ORG"	III.38
III.4.g. - Processamento de "ENT"	III.40
III.4.h. - Processamento de "EBC" e "ASC"	III.43
III.4.i. - Processamento de "ISS"	III.44
III.4.j. - Processamento de "END"	III.47
III.4.k. - Processamento de "CAB"	III.47
III.4.l. - Processamento das instruções de controle de listagem	III.49
III.4.m. - Processamento de "CALL"	III.50

Continuação

III.5. - Tratamento dos comentários	III.51
III.6. - Ocorrência de erros	III.51
III.7. - Ocorrência de "pseudo-macros"	III.52
III.8. - Arquivo de Informação Suplementar	III.52
III.9. - Tabelas Utilizadas	III.54
III.9.a. - Tabela de comandos	III.55
III.9.b. - Tabela de símbolos	III.58
III.9.c. - Tabela de equivalências	III.61
III.10.- Saídas do "Cross-Montador"	III.61
III.10.a. - Saída Impressa	III.61
III.10.b. - Saída de disco	III.62

P A R T E I I I

Nesta parte estão descritas as técnicas utilizadas na construção do "Cross-Montador". Uma vez escolhida a linguagem a ser utilizada neste sistema, passou-se a implementação do seu Processador.

A linguagem foi descrita na parte II e no Apêndice B. A linguagem de máquina aceita pelo minicomputador se encontra descrita no Apêndice A. O "Cross-Montador" deverá fazer a ligação entre estas duas linguagens, traduzindo a primeira na segunda.

III.1. - Tradução da linguagem "Assembler" ou de Montagem em linguagem de máquina

Como foi visto na parte anterior, a instrução de linguagem fonte é composta pelos campos de rótulo, mnemônico, operando e comentários.

O campo de rótulo, como já foi dito tem a finalidade de permitir à instrução que o contém ser referenciada por outra qualquer. Os rótulos perdem o seu significado após a primeira análise do comando fonte ter sido efetuada. São então atribuídos endereços aos símbolos que constituem os rótulos. Quando um símbolo, identificado como um rótulo aparece como operando de uma instrução, ele é substituído pelo seu endereço, quando a instrução é gerada.

Os operandos podem ser símbolos, expressões, números ou qualquer outra informação importante para a definição da instrução codificada. O tratamento do operando é feito pois de acordo com a instrução.

Os dois tipos principais de comandos são: comandos que são traduzidos em linguagem de máquina, que constituem as "instruções propriamente ditas", e comandos que são inseridos apenas para fornecer informações ao programa Montador, as chamadas "pseudo-instruções", ou simplesmente "pseudos". As primeiras sempre são traduzidas em código de máquina e ocupam regis-tros de memória, enquanto que as segundas podem ou não ocupar tais registros.

À medida que as instruções são endereçadas, elas vão se dispondo sequencialmente na memória. O "Cross-Montador" não pode impor limites para a área ocupada pelo programa, os quais serão impostos pela memória física na hora do carregamento. Isto se deve ao fato, de que na fase de montagem de um programa não se tem idéia de quanta memória ficará disponível para ele, uma vez que esta deverá ser compartilhada com outros programas, tais como o Sistema Operacional, as rotinas residentes e as próprias rotinas do programa em apreço. Se a quantidade de memória dispo-nível ao programa for insuficiente, endereçamentos serão feitos além dos limites desta, ocasionando erros que impedem o pro-seguimento do mesmo.

Ao lado dos endereçamentos feitos para as instruções do programa, deve ser feito o endereçamento dos dados do progra-ma. No caso de linguagem "Assembler", as posições onde os dados deverão residir é fornecida pelo programador, isto é, a posição destes dentro do programa é dada pela posição do registro de entrada em que se encontra a ordem para a locação do referido dado, em relação ao registro inicial. Desta maneira, dados e instruções podem vir intercalados, dependendo da vontade do pro-gramador, e da lógica do programa.

III.2. - Organização do processador

O principal problema com que se depara o "Cross-Monta-dor" ao analisar um programa, reside no fato de que as instruções de linguagem fonte são fornecidas sequencialmente, e no en-tanto, podem ser feitas referências em uma instrução, ao prefixo de qualquer outra, dentro do programa.

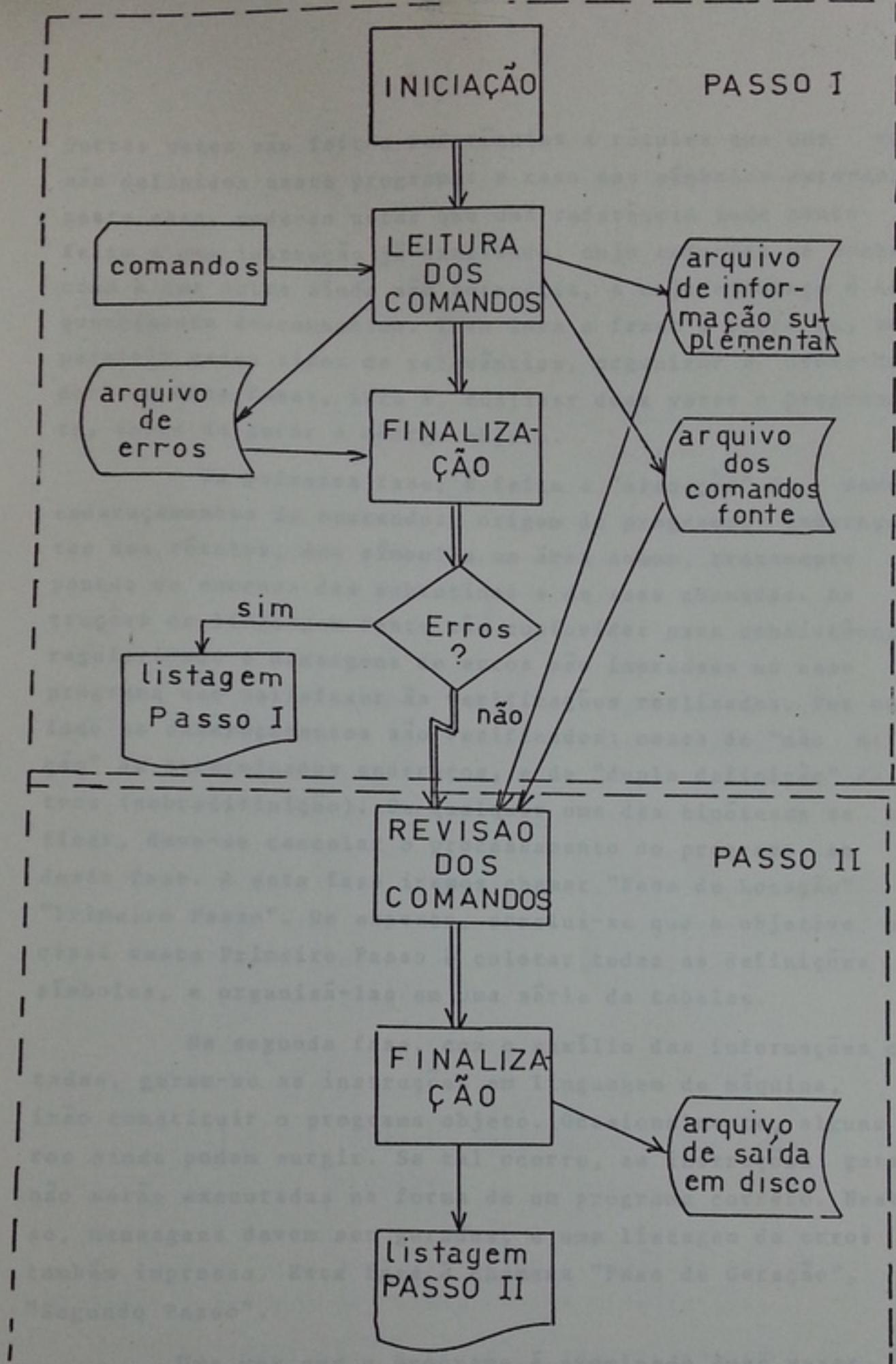


Figura III.0- O Esquema do Sistema

Outras vezes são feitas referências a rótulos que nem mesmo são definidos neste programa: o caso dos símbolos externos: neste caso, pode-se notar que uma referência pode tanto ser feita a uma instrução já analisada, cujo endereço se conhece, como a uma outra ainda não fornecida, e cujo endereço é consequentemente desconhecido. Isto leva a fazer a opção de, ao se permitir esses tipos de referências, organizar o "Cross-Montador" em duas fases, isto é, analisar duas vezes o programa fonte, antes de gerar o código objeto.

Na primeira fase, é feita a "alocação" de memória: endereçamentos de operandos, origem de programa, endereçamentos dos rótulos, dos símbolos em área comum, tratamento dos pontos de entrada das subrotinas e de suas chamadas. As instruções de linguagem fonte são conferidas para consistência e regularidade e mensagens de erros são impressas no caso do programa não satisfazer às verificações realizadas. Por outro lado os endereçamentos são verificados: casos de "não definição" de determinados endereços, e de "dupla definição" de outros (sobredefinição). Se qualquer uma das hipóteses se verificar, deve-se cancelar o processamento do programa, ao fim desta fase. A esta fase iremos chamar "Fase de Locação" ou "Primeiro Passo". Do exposto, conclui-se que o objetivo principal deste Primeiro Passo é coletar todas as definições de símbolos, e organizá-las em uma série de tabelas.

Na segunda fase, com o auxílio das informações coleadas, geram-se as instruções em linguagem de máquina, que irão constituir o programa objeto. Ocasionalmente, alguns erros ainda podem surgir. Se tal ocorre, as instruções geradas não serão executadas na forma de um programa correto. Neste caso, mensagens devem ser geradas, e uma listagem de erros é também impressa. Esta fase é chamada "Fase de Geração", ou "Segundo Passo".

Uma vez que o programa é examinado duas vezes, torna-se necessário um meio de armazenamento intermediário para a informação de entrada. Devido ao grande volume destas informações (cartões de entrada), mais o acréscimo das informações adicionais, elas não podem residir na memória principal. Assim, um disco magnético é utilizado para o seu armazenamento.

A figura III.0. mostra o esquema do "Cross- Montador", detalhando as duas fases e apresentando os principais arquivos utilizados.

A escolha de um esquema, que trabalhasse em duas fases distintas, e totalmente separáveis, foi feita com dois propósitos:

- fazer na primeira fase uma análise do programa, que pudesse detectar falhas grosseiras de programação, cancelando o programa ao fim desta, ao serem detectados erros deste tipo; desta maneira evita-se a tarefa de se gerar código simultaneamente, e resolve-se também o problema das referências futuras;
- desmembrando-se em dois programas independentes, diminuir a área de memória gasta para o sistema, e desta maneira reservar a maior parte dela para tabelas, evitando a flutuação destas entre memória principal e secundária. Por outro lado, as duas partes do programa se interligam através de uma área de "Common" e de uma série de arquivos, que conterão as informações coletadas na primeira fase do programa.

III.2.a. - Fase de Locação ou Primeiro Passo

Esta fase pode ser dividida em três sub-fases: a iniciação a leitura dos comandos e a finalização.

Na iniciação, o programa tem posicionadas no início as suas tabelas; o contador de instruções é feito igual a zero, bem como os contadores de páginas, de cartões e de erros; o cabeçalho é preenchido com brancos e uma série de indicadores é convenientemente iniciada .

A figura III.2.a.1., mostra este processo.

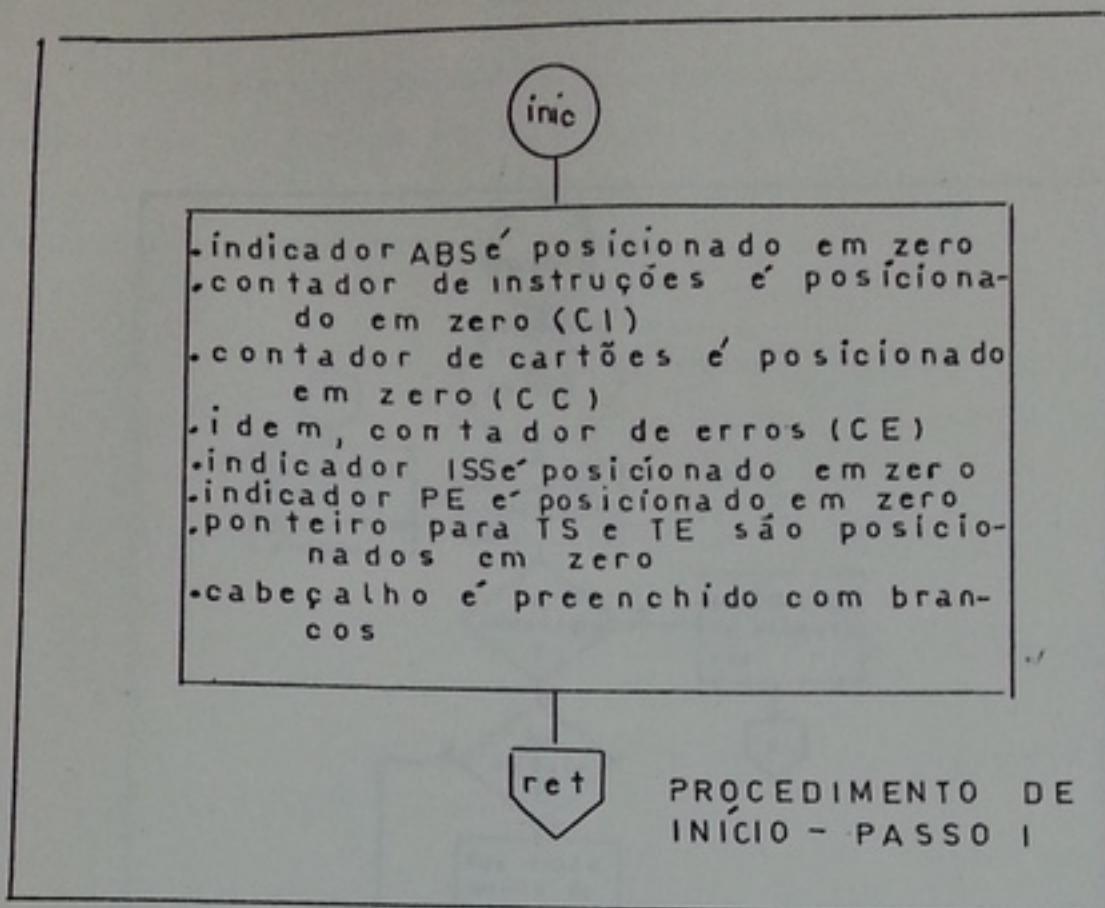


Figura III.2.a.1. - Procedimento de início - Passo I

O programa inicia então a leitura dos comandos. Os cartões contendo informação são lidos até ser encontrado um cartão de fim de programa fonte ("pseudo" END), quando então é encerrada esta sub-fase, passando-se a finalização. O cartão começa a ser analisado a partir da coluna 21. As colunas 72 a 80 não são analisadas, qualquer tipo de perfuração podendo aparecer nestas colunas.

Se a coluna 21 apresentar um "*", este cartão é de comentário. Qualquer outro caracter diferente de "branco" nesta coluna é o caracter inicial de um rótulo. Cartões de comentários não sofrem qualquer tipo de análise. O cartão é guardado para posterior impressão, e um novo cartão é lido.

Quando um comando é analisado, este exame pode provar entrada de informação em uma ou mais tabelas, com consequente avanço de seus ponteiros. É o caso de um comando que apresente rótulo; este rótulo, juntamente com o endereço da instrução (valor do Contador de Instruções), é inserido em uma tabela conveniente. Da mesma maneira, se a instrução possui operандos, estes são examinados e podem provocar a entrada em diversas tabelas, dos símbolos que aparecem nestes comandos. A análise do comando nesta fase é feita de acordo com o Grupo de Tratamento da instrução (ver tabela de mnemônicos).

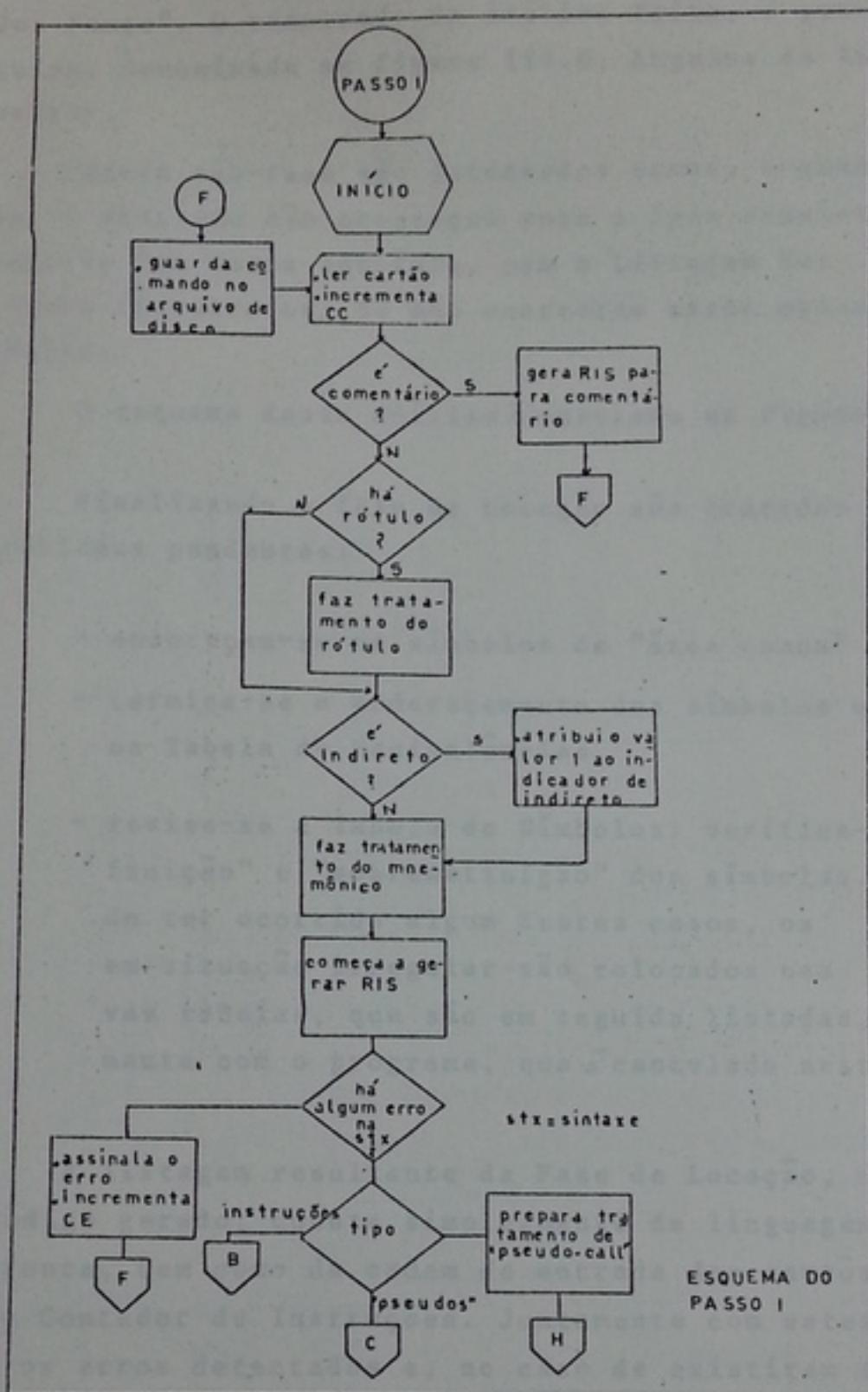


Fig.III.2.a.2. - Esquema do Passo I

Os comandos lidos são guardados em um arquivo de disco, que na figura III.Ø é apresentado como o "Arquivo dos Comandos Fonte". O resultado da análise feita, é guardado em um arquivo, denominado na figura III.Ø, Arquivo de Informação Suplementar.

Nesta sub-fase são detectados erros, e quando estes ocorrem, o programa não prossegue para a fase seguinte, sendo encerrado ao fim desta sub-fase, com a listagem dos comandos e dos erros encontrados. Se não ocorreram erros passa-se a finalização.

O esquema desta análise é mostrado na figura III.2.a.
2.

Finalizando a fase de Locação são tratados os últimos, problemas pendentes:

- endereçam-se os símbolos de "área comum"
- termina-se o endereçamento dos símbolos que estão na Tabela de Equivalências.
- revisa-se a Tabela de Símbolos: verifica-se a "definição" e "sobredefinição" dos símbolos. No caso de ter ocorrido algum destes casos, os símbolos em situação irregular são colocados nas respectivas tabelas, que são em seguida listadas, juntamente com o programa, que é cancelado nesta fase.

A listagem resultante da Fase de Locação, não contém o código gerado; consta simplesmente da linguagem dos comandos fonte, bem como da ordem de entrada dos cartões e do valor do Contador de Instruções. Juntamente com estes são impressos os erros detectados e, no caso de existirem símbolos mal definidos, a listagem destes.

O esquema deste tratamento é mostrado na figura
III.2. a3.

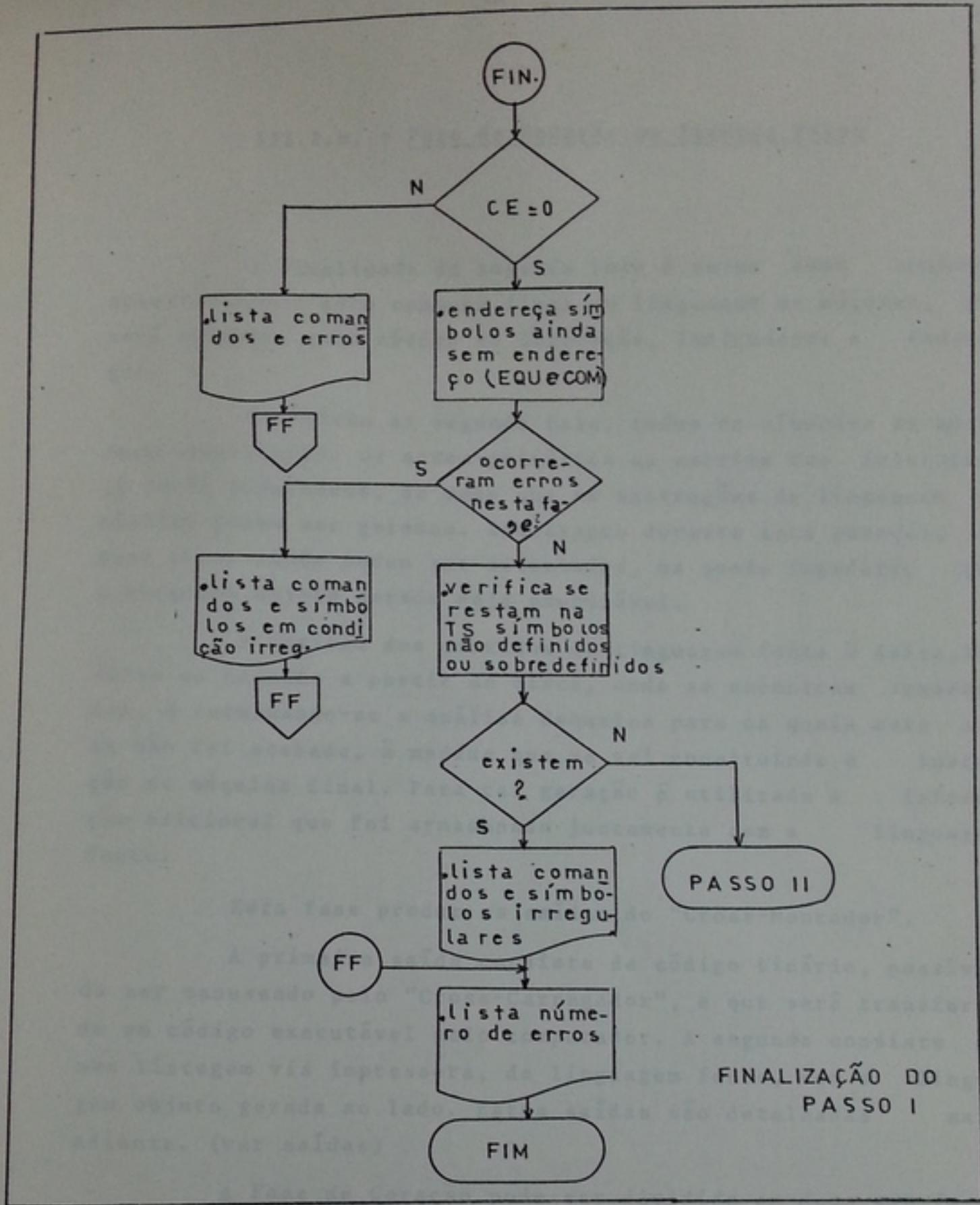


Fig. III.2.a.3. - Finalização do Passo I

III.2.b. - Fase de Geração ou Segundo Passo

A finalidade da segunda fase é rever cada comando, substituindo-o pelo comando final em linguagem de máquina, que será composto pelo código de instrução, indicadores e endereços.

No início da segunda fase, todos os símbolos se encontram endereçados; os erros primários na escrita das instruções já foram eliminados, de modo que as instruções de linguagem de máquina podem ser geradas. Entretanto durante esta geração, alguns erros ainda podem ser detectados, os quais impedirão que o programa objeto gerado seja executável.

A revisão dos comandos de linguagem fonte é feita, lendo-se os comando a partir do disco, onde se encontram armazenados, e terminando-se a análise daqueles para os quais esta ainda não foi acabada, à medida que se vai construindo a instrução de máquina final. Para tal geração é utilizada a informação adicional que foi armazenada juntamente com a linguagem fonte.

Esta fase produz as saídas do "Cross-Montador".

A primeira saída consiste de código binário, possível de ser manuseado pelo "Cross-Carregador", e que será transformado em código executável pelo computador. A segunda consiste de uma listagem via impressora, da linguagem fonte, com a linguagem objeto gerada ao lado. Estas saídas são detalhadas mais adiante. (ver saídas)

A Fase de Geração pode ser dividida em duas sub-fases: a revisão dos comandos e a finalização. (Figura III.2.b.)

Durante a primeira sub-fase, os comandos são revistos a fim de ser feita a codificação em linguagem objeto. A primeira etapa consiste na iniciação de ponteiros e indicadores: posicionam-se em zeros os contadores de instruções, de cartões, de páginas e o indicador do número de erros; o indicador de listagem é posicionado a fim de produzir a listagem, e o indicador de listagem para os cartões de controle é posicionado para suprimir estes comandos.

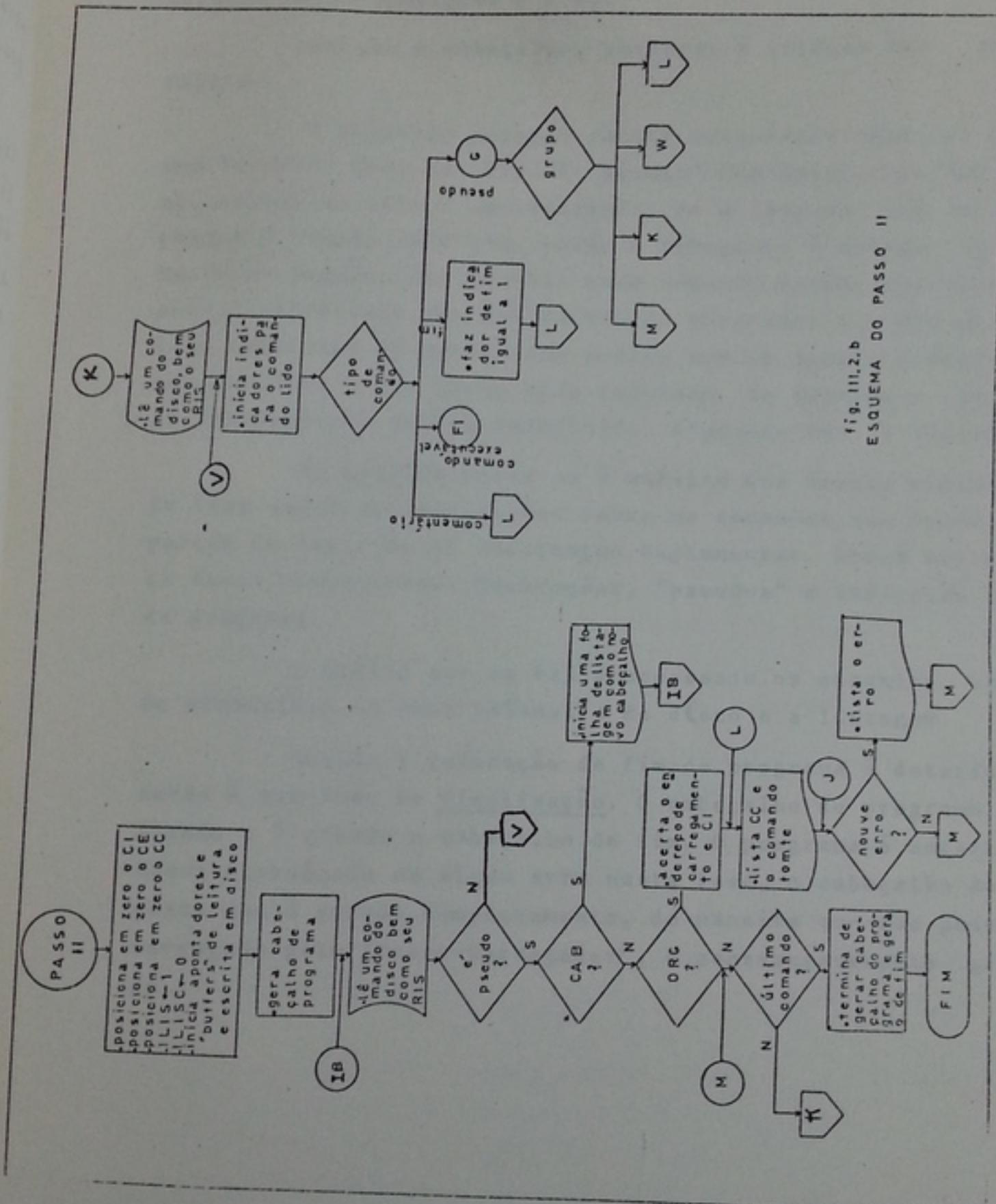


fig. III.2.b
ESQUEMA DO PASSO II

Em seguida passa-se à geração do cabeçalho do programa para a saída binária, que será produzida em disco. Alguns ítems deste cabeçalho, porém, só serão produzidos no final do Segundo Passo (posições 4 e 8).

Gerado o cabeçalho, passa-se à geração dos comandos objeto.

O primeiro comando recebe tratamento especial se for uma "pseudo" CAB, ou ORG. A "pseudo" CAB inicializa uma página, colocando nela o seu operando. Se a "pseudo" não for CAB, a página é também iniciada, porém o cabeçalho é branco. Se o comando em seguida for um ORG, este comando recebe tratamento especial, diferente dos demais ORG do programa. Ele não gera um novo cabeçalho de dados como ocorre com os demais comandos ORG, mas redefine o que havia sido iniciado. Em seguida o Contador de Instruções é também redefinido. A "pseudo" ORG é listada.

Em seguida passa-se à análise dos demais comandos. Nesta fase todas as informações sobre os comandos são tomadas a partir do Registro de Informação Suplementar. Podem ocorrer nesta fase: comentários, instruções, "pseudos" e indicação de fim de programa.

A medida que se vai processando os comandos, vão sendo produzidas as duas saídas: a de disco e a listagem.

Quando a indicação de fim de programa é detectada passa-se à sub-fase de finalização. O cabeçalho de programa é finalizado e é gerado o cabeçalho de fim. O programa é encerrado. Se houve ocorrência de algum erro nesta fase, o cabeçalho de programa não é gerado completamente, de maneira que não poderá ser carregado pelo "Cross-Carregador", e consequentemente processado.

III.3. - Processamento das Instruções Executáveis

Comandos com instruções executáveis, podem sempre conter rótulos. O "Cross-Montador", na análise de tais comandos, testa a existência de um rótulo. Se este for encontrado, inicia-se uma busca na Tabela de Símbolos e, se este rótulo já constar dela, testa-se a sua definição. Se não estiver fazendo parte desta tabela, ele será nela inserido. Em ambos os casos, o símbolo deve ser marcado como definido, e o conteúdo do Contador de Instruções é inserido na parte correspondente ao seu endereço. Se o símbolo já tiver sido anteriormente definido, ele é marcado como "sobredefinido", que constitui um erro.

O tratamento do mnemônico, a seguir, é feito de acordo com o tipo a que pertence:

- a) mnemônicos que devem conter operandos
- b) mnemônicos que não podem conter operandos

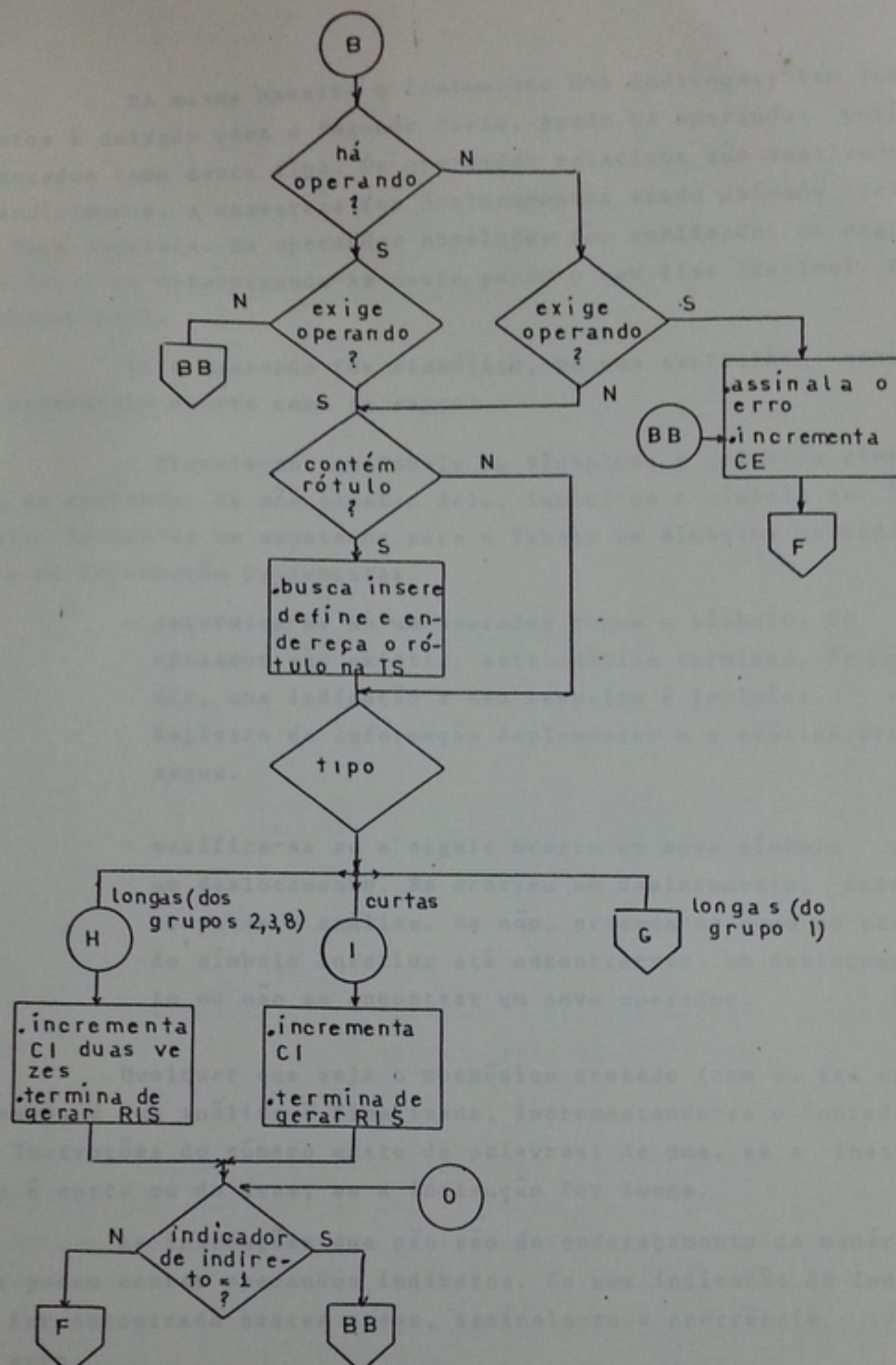
O programa Montador verifica se a instrução contém operando, isto é, verifica se a coluna 35 do cartão se encontra em branco. Se estiver, o conteúdo das colunas 36 a 71 constitui um comentário, e a instrução não contém operando.

O programa verifica a sua obrigatoriedade, e qualquer dos casos "a" ou "b", sendo contrariado, um erro é assinalado.

Uma vez tendo sido isto verificado, passa-se a análise da instrução. Para esta análise, determina-se a que grupo de tratamento pertence a instrução. Nesta fase, porém, somente as instruções de endereçamento da memória tem seus operandos analisados. São então diferenciados os seguintes tipos de operandos:

- absolutos (que podem ser decimais ou hexadecimais);
- relativos (que podem ser a "sí próprios" ou a uma outra instrução);
- simbólicos;
- indiretos, que podem ser de qualquer dos tipos acima.

As instruções dos demais grupos tem seus operandos analisados na Fase de Geração.



**TRATAMENTO DAS INSTRUÇÕES EXECUTÁVEIS
PASSO I**

Fig. III.3.2 - Tratamento das instruções executáveis
Passo I

Da mesma maneira o tratamento dos endereçamentos indiretos é deixado para o Segundo Passo, sendo os operandos porém marcados como deste tipo. Os operandos relativos são analisados parcialmente, a conversão dos deslocamentos sendo deixada para a fase seguinte. Os operandos absolutos são analisados na segunda fase, se determinando-se neste passo o seu tipo (decimal ou hexadecimal).

Se o operando for simbólico, ou uma expressão, então o tratamento ocorre como se segue:

- procura-se, na Tabela de Símbolos, o primeiro símbolo do operando. Se não constar dela, inclui-se o símbolo na Tabela. Inclui-se um apontador para a Tabela de Símbolos no Registro de Informação Suplementar.
- determina-se se um operador segue o símbolo. Se o operador não existir, esta análise terminou. Se existir, uma indicação a seu respeito é incluída no Registro de Informação Suplementar e a análise prossegue.
- verifica-se se a seguir ocorre um novo símbolo ou um deslocamento. Se ocorreu um deslocamento, interrompe-se a análise. Se não, procede-se como no caso do símbolo anterior até encontrar-se um deslocamento ou não se encontrar um novo operador.

Qualquer que seja o mnemônico tratado (com ou sem operando), a sua análise é finalizada, incrementando-se o Contador de Instruções do número exato de palavras: de uma, se a instrução é curta ou de duas, se a instrução for longa.

As instruções que não são de endereçamento da memória não podem conter operandos indiretos. Se uma indicação de indireto for encontrada nestes casos, assinala-se a ocorrência de um erro.

O procedimento descrito é mostrado na figura III.3.1.

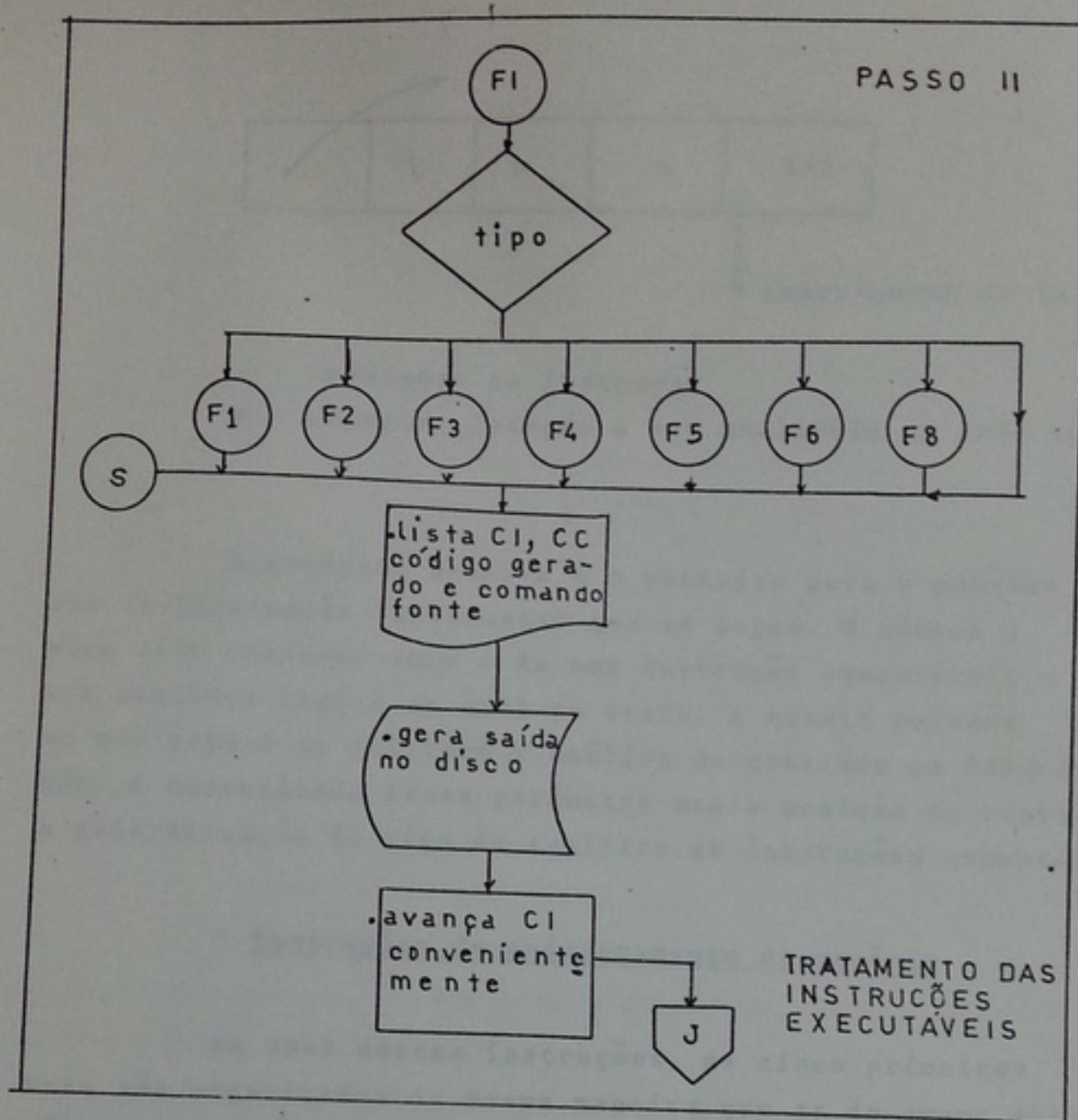
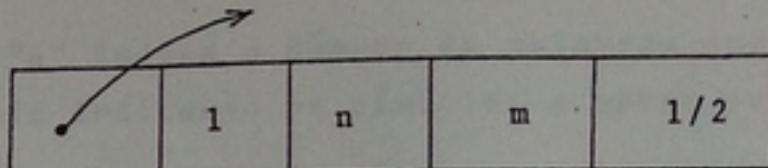


Figura III.3.1. - Tratamento das instruções executáveis - Passo II

O Registro de Informação Suplementar tem no caso dos mnemônicos uma configuração que varia de acordo com o tipo de mnemônicos e pode de maneira geral ser dividida em dois grupos:

- instruções curtas ou longas cujos operandos não são analisados nesta fase

Estas instruções tem um Registro de Informação Suplementar com a seguinte organização:



comprimento da instrução

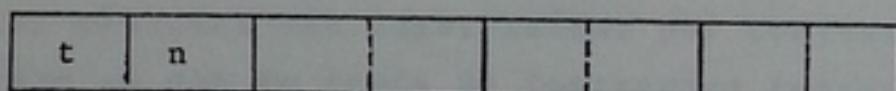
n - indicador de instrução

m - primeira posição a ser analisada na fase seguinte

A primeira palavra é o ponteiro para o próximo Registro de Informação Suplementar que se segue. O número 1 identifica este registro como o de uma instrução executável; a palavra seguinte indica de qual se trata. A quarta palavra indica em que coluna se iniciará a análise do operando na Fase de Geração. A necessidade desta parâmetro nesta posição do registro é a generalização do tipo de registro de instruções executáveis.

- instruções de endereçamento da memória

No caso destas instruções, as cinco primeiras palavras são organizadas da mesma maneira que as do grupo acima, porém o Registro de Informação Suplementar continua como se segue:



operandos em expressão

onde "t" designa o tipo de operando que se tem:

- 1 - endereçamento relativo a si próprio (direto)
- 2 - endereçamento absoluto com operando decimal (direto)
- 3 - endereçamento simbólico, ou expressão (direto)
- 4 - endereçamento absoluto com operando hexadecimal (direto)
- 5 - endereçamento relativo a si próprio (indireto)
- 6 - endereçamento absoluto com operando decimal (indireto)
- 7 - endereçamento simbólico, ou expressão (indireto)
- 8 - endereçamento absoluto com operando hexadecimal (indireto)

"n" indica o número de palavras que completam o registro as quais indicarão os símbolos e operadores envolvidos.

No caso de tais símbolos e operadores aparecerem, a constituição do registro continuaria do seguinte modo:

Palavra de ordem (7+j): ponteiro para a Tabela de Símbolos onde se encontra o ítem em consideração

Palavra de ordem (7+j+1): indica o delimitador. Se o delimitador for branco, tal ponteiro não existe.

$$1 \leq j \leq n$$

Da mesma maneira que no Primeiro Passo, na Fase de Geração o tratamento dos mnemônicos se faz de acordo com o grupo de tratamento ao qual pertence este mnemônico. Estes grupos são apresentados na figura III.9.1.

Antes porém de se indicar como são geradas as instruções de máquina para cada caso específico, será descrito o tratamento geral dado a elas.

O primeiro passo é, ao se detectar a instrução que se tem, mascarar uma palavra especial, com a máscara correspondente a instrução sendo tratada. Em seguida, passa-se para a rotina específica de cada caso, a qual finaliza a montagem iniciada.

As máscaras são constituídas por conjuntos de 16 "bits", mesmo no caso em que se trata de instruções curtas. Neste caso apenas os 8 "bits" mais significativos são utilizados. A tabela da figura III.9.1., mostra as máscaras utilizadas.

Os "bits" que são variáveis nas instruções, e que portanto dependem de opções do programador, são feitos iguais a zero nas máscaras. A palavra especial, onde é montada a instrução, será denominada INST. As partes variáveis das instruções será dado o nome de "componentes". A função das rotinas específicas é preencher adequadamente estes componentes. A lista dos componentes é apresentada a seguir:

INCT(end): corresponde aos 12 "bits" de endereço das instruções de referência a memória, e ocupam INST(0-11).

INST(imed): corresponde aos "bits" da segunda palavra das instruções do grupo das imediatas e ocupam INST(0-7).

INST(desl): corresponde aos 4 "bits" menos significativos da segunda palavra das instruções de deslocamento e ocupam INST (0-3).

INST(disp): corresponde aos 4 "bits" menos significativos da primeira palavra das instruções de entrada-saída e ocupam INST(8-11).

INST(cmd): corresponde aos 4 "bits" menos significativos da segunda palavra das instruções de Entrada e Saída e ocupam INST(0-3).

INST(salto): corresponde ao "bit" 1 das instruções curtas de testes e ocupa INST (9).

INST(pnl): corresponde aos 3 "bits" menos significativos das instruções de painel, e ocupam INST(8-10).

Uma vez descoberto o grupo de tratamento da instrução e a palavra INST mascarada, passa-se ao processamento específico de cada caso. Uma vez montada a instrução pela rotina tratadora, o procedimento volta a ser padrão para todas as instruções. Agora, providencia-se a listagem da instrução; são listados: código gerado, valor do Contador de Instruções, valor de ordem de entrada e comando fonte. A seguir é providenciada a saída de disco, isto é, a instrução gerada é colocada no "buffer" de saída juntamente com um código de relocação. O código é 0 para as instruções absolutas e 1 para as relocáveis.

Em seguida o Contador de Instruções é incrementado do número de palavras ocupadas pela instrução e retorna-se para tratar um novo comando.

O diagrama da segunda fase ilustra esta parte descrita (figura III.3.2.).

A seguir, será descrito o procedimento particular para cada instrução, no segundo passo.

III.3.a. - Processamento do grupo de endereçamento da memória

O procedimento adotado para estas instruções na segunda fase, depende do tipo de operando que se tem, isto é do modo de endereçamento adotado.

O tipo de operando foi determinado na FASE DE LOCAÇÃO, quando foi parcialmente tratado; o resultado deste tratamento se encontra no Registro de Informação Suplementar; para esta fase, é deixada na totalidade a análise dos operandos absolutos e dos relativos à própria instrução.

Os operandos absolutos podem estar na forma decimal ou hexadecimal; estes números são agora convertidos em binário e inseridos em INST (end).

Endereçamentos de tipo relativo à si próprio, são tratados nesta fase, sendo o deslocamento, se existir, convertido também em binário. A expressão relativa é avaliada modificando-se o conteúdo do Contador de Instrução pelo deslocamento, e colocando-se o resultado em INST(end).

Os operandos que envolvem símbolos tem no Registro de Informação Suplementar, ponteiros para suas posições na Tabela de Símbolos. Se o operando for do tipo expressão, ela é computada a partir dos endereços: pode restar ainda a parte correspondente a um deslocamento, ainda não tratado. Neste caso, o deslocamento é convertido a binário, afim de se terminar a avaliação da expressão. No caso de um endereçamento simbólico, a expressão se reduz a um simples endereço. A expressão avaliada, é colocada em INST (end).

O diagrama das figuras III.3.3. e III.3.4. ilustra o processo.

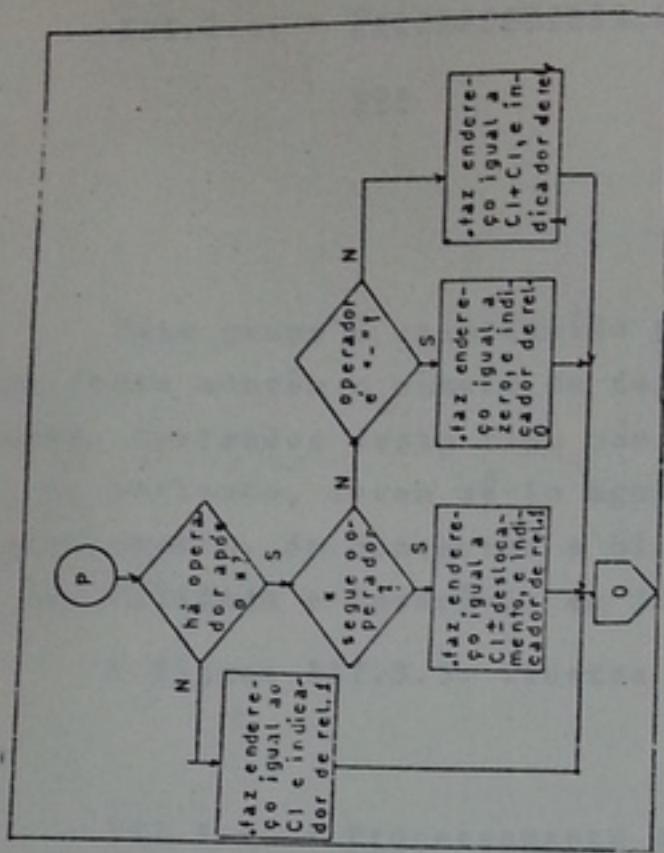
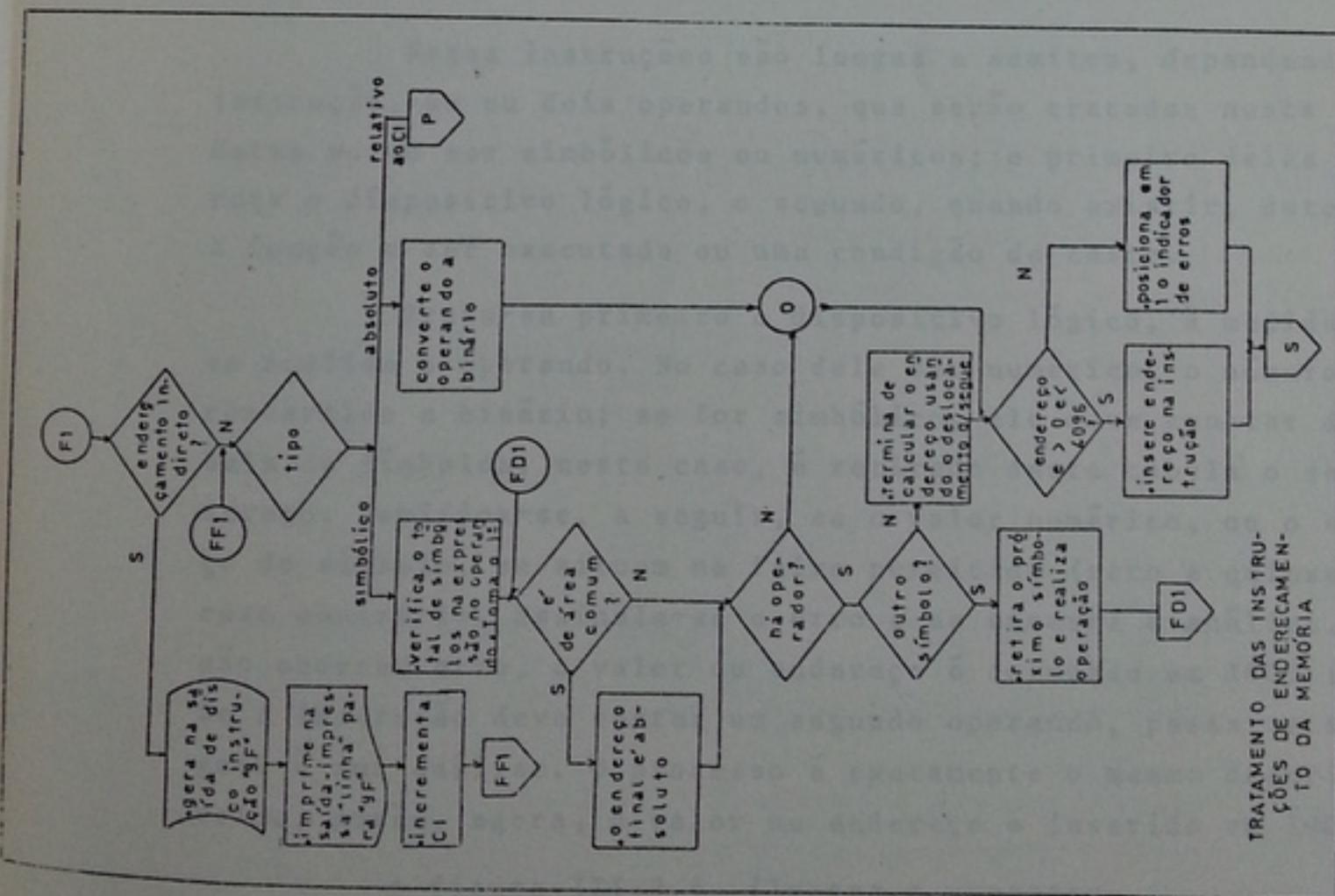


Figura III.3.3 e Figura III.3.4. - Tratamento das instruções de endereçamento da memória - Passo II



III.3.c. - Processamento das instruções de deslocamentos

Este grupo é constituído por instruções longas; o comando fonte contém o número de deslocamentos que devem ser efetuados. Operandos deste tipo não são tratados na Fase de Locação, e, portanto, devem sê-lo agora. A rotina se encarrega, consequentemente, de converter a binário este operando, verificar a sua validade e inserí-lo em INST (des1).

A figura III.3.5. ilustra o processo.

III.3.d. - Processamento das instruções de entrada e saída

Processamento do Grupo das instruções de entrada e saída.

Estas instruções são longas e admitem, dependendo da instrução, um ou dois operandos, que serão tratados nesta fase. Estes podem ser simbólicos ou numéricos; o primeiro deles endereça o dispositivo lógico, o segundo, quando existir, determina a função a ser executada ou uma condição de teste.

Trata-se primeiro o dispositivo lógico, à medida que se analisa o operando. No caso dele ser numérico, o número é convertido a binário; se for simbólico, ele deve constar da Tabela de Símbolos; neste caso, é retirado desta tabela o seu endereço. Verifica-se, a seguir, se o valor numérico, ou o endereço do símbolo, se situam na faixa permitida (zero a quinze). Em caso contrário, assinala-se o erro e se encerra a análise. Se não ocorreu erro, o valor ou endereço é colocado em IOCT (disp). Se a instrução deve conter um segundo operando, passa-se a seguir à sua análise. O processo é exatamente o mesmo descrito acima, porém, agora, o valor ou endereço é inserido em IOCT(fnc).

A figura III.3.6. ilustra o processo.

III.3.b. - Processamento das instruções imediatas

No caso destas instruções, o operando pode ou não estar explícito. Se isto ocorrer, a análise é finalizada neste ponto. Se porém este estiver explícito, deverá ser agora analisado. O tratamento feito consiste na transformação dos operandos que se encontram na forma decimal ou hexadecimal em números binários, na verificação de sua ordem de grandeza e inserção em INST(imed).

O processo é ilustrado na figura III.3.5.

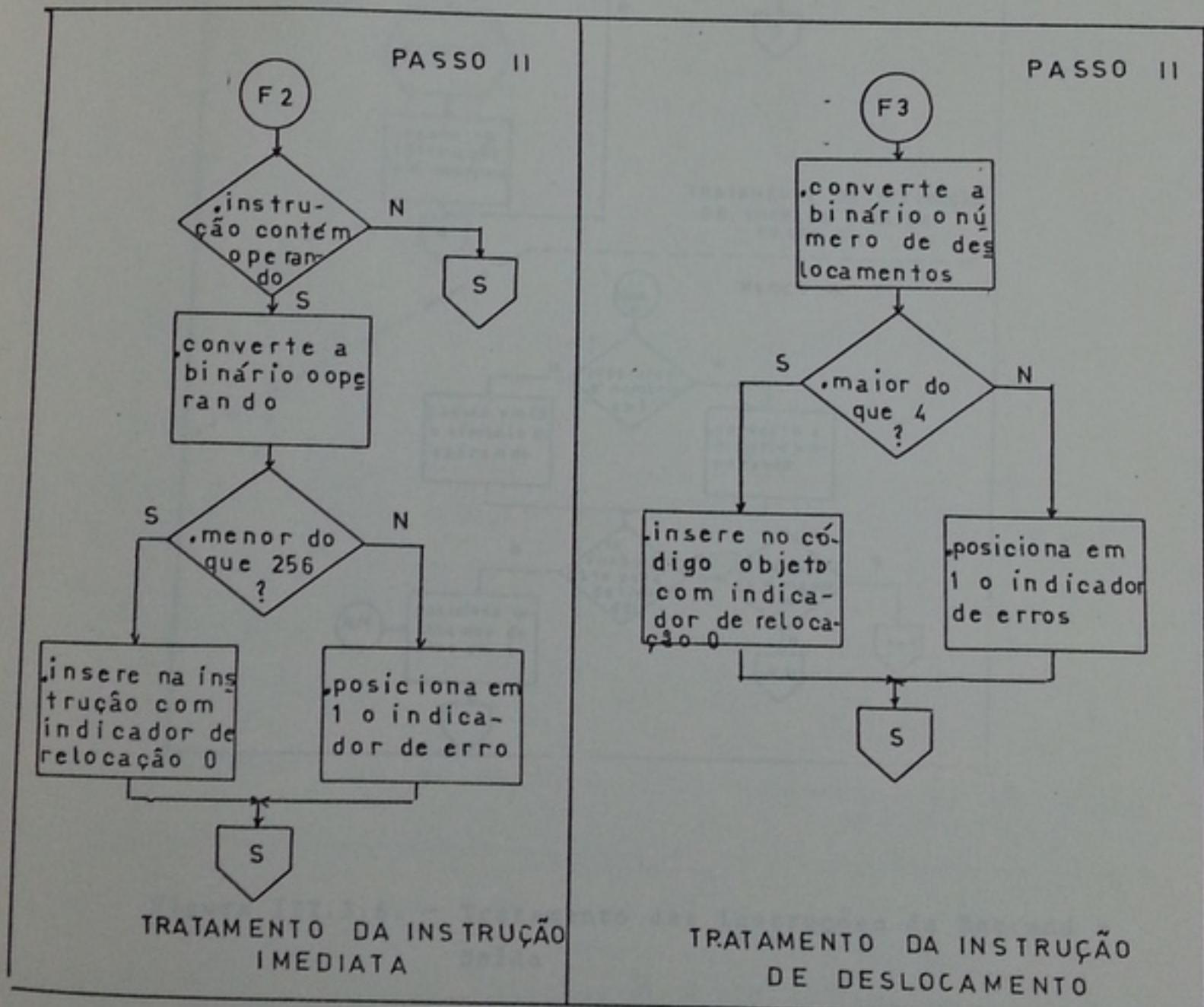


Fig.III.3.5. - Tratamento das instruções imediatas e de deslocamento - Passo II

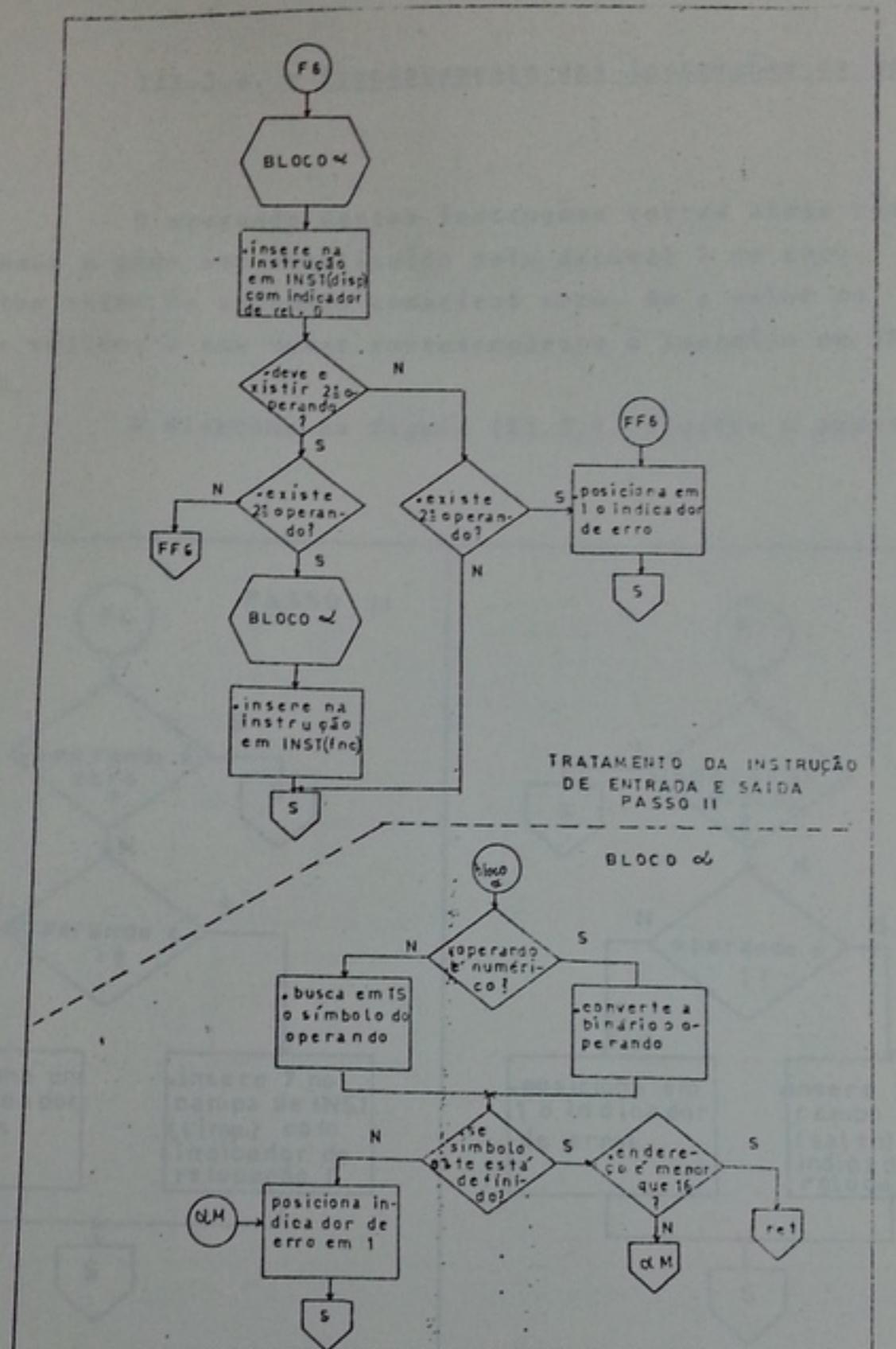


Figura III.3.6. - Tratamento das instruções de Entrada e Saída

III.3.e. - Processamento das instruções de saltos

O operando destas instruções curtas ainda não foi analisado e pode ser constituído pelo decimal 1 ou zero. Qualquer outro valor de operando constitui erro. Se o valor do operando for válido, o seu valor correspondente é incluído em INST (salto).

O diagrama na figura III.3.7. ilustra o exposto.

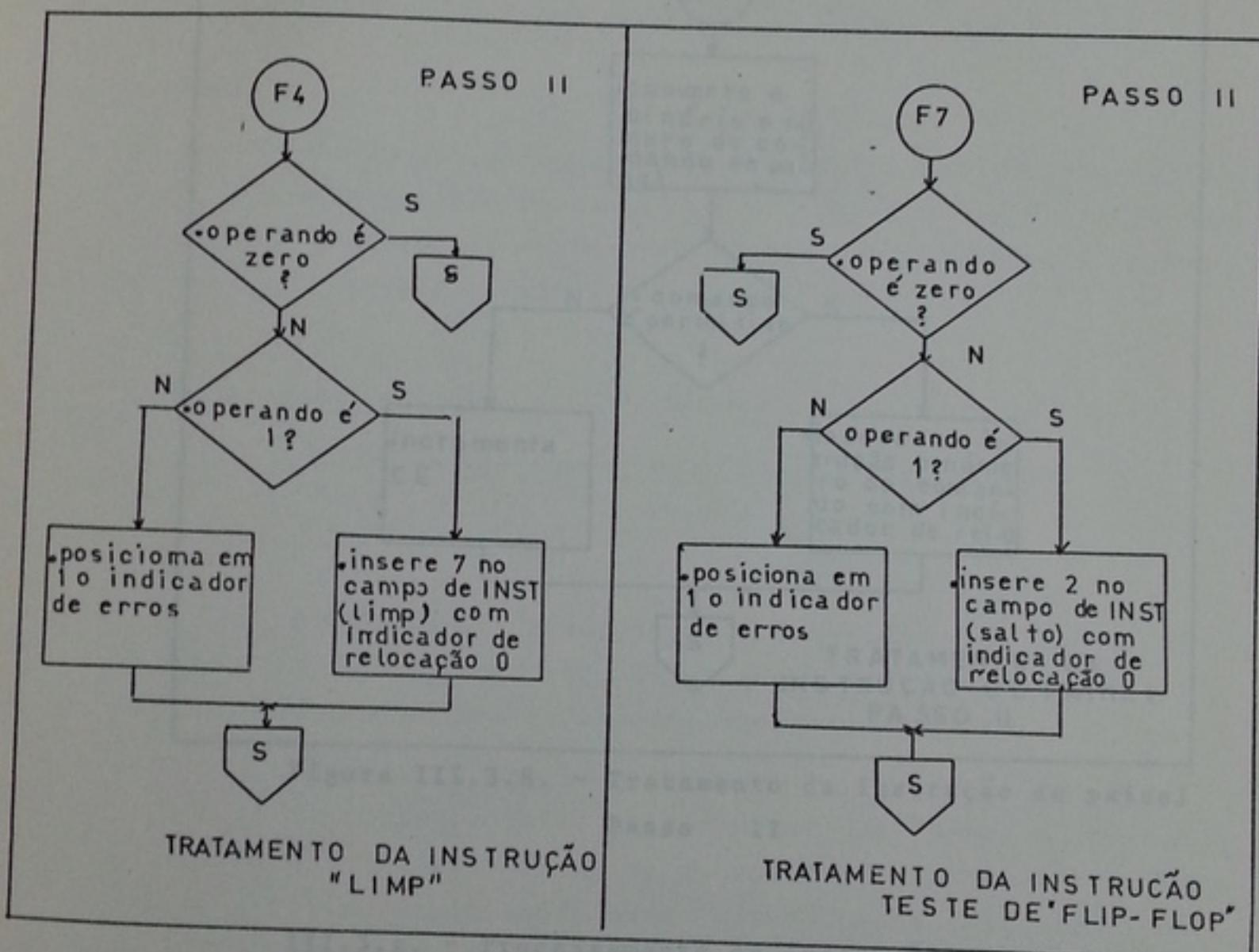


Figura III.3.7. - Tratamento da instrução "LIMP" e das instruções de teste "flip-flop" - Passo II

III.3.f. - Processamento das instruções de painel

O operando destas instruções é numérico e deve ser tratado nesta fase. O tratamento consiste em se converter a binário este número e verificar se ele se encontra na gama permitida para funções de painel e inserí-lo em INST (pn1).

O diagrama na figura III.3.8. ilustra o processo

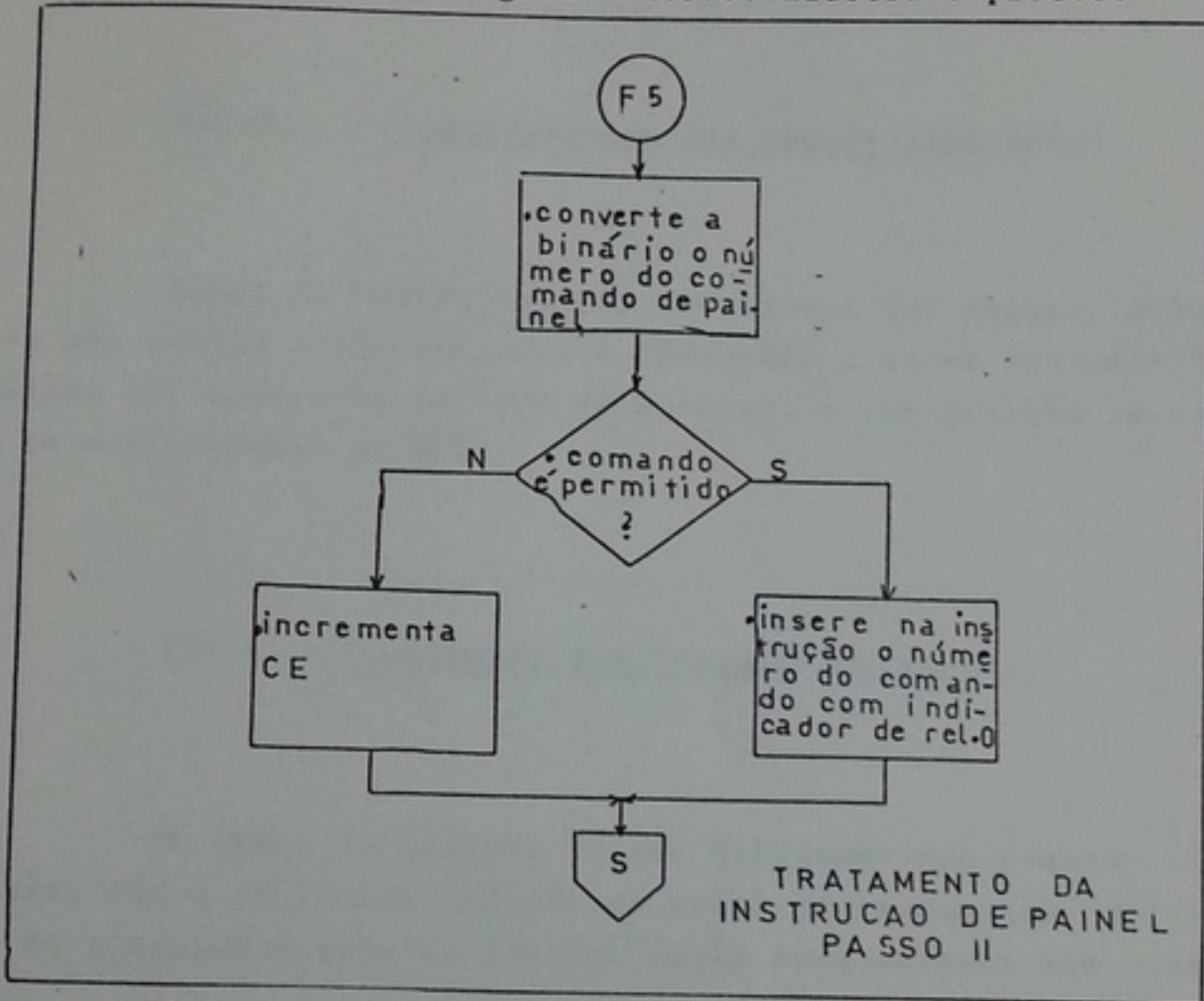


Figura III.3.8. - Tratamento da instrução de painel
Passo II

III.3.g. - Processamento da instrução NOP

Esta instrução é codificada como um deslocamento, com número de deslocamentos igual a zero. Seu tratamento comprehende apenas o "mascaramento" da instrução.

III.3.h. - Processamento da instrução LIMP

O operando desta instrução pode ser 1 ou zero, e deve ser tratado nesta fase. O tratamento consiste em se verificar se o valor no comando fonte é um destes dois, e na hipótese afirmativa, inserí-lo em INST(limp).

O diagrama na figura III.3.7. ilustra o processo.

III.3.i. - Processamento das demais instruções

Todas as instruções não constantes dos grupos descritos, são curtas e não apresentam operandos a serem tratados. Sua análise foi encerrada na Fase de Locação, e sua geração se resume ao mascaramento de INST.

Figura III.4.1. - Tratamento das "pseudos". Passo 1

III.4. - Tratamento das "pseudo-instruções"

Em ambos os passos, ao ser detectada uma pseudo-instrução, ela é analisada segundo um grupo de tratamento. Os grupos de tratamento para as "pseudos" estão apresentados nas tabelas.

Na primeira fase, o tratamento das "pseudos" começa pela verificação da existência de um rótulo no comando. Se este rótulo existir, verifica-se se a "pseudo" pode contê-lo ou não, ou se seu aparecimento é obrigatório. Qualquer das condições impostas, sendo contrariada, indica ocorrência de erro. Em seguida, não se verificando o erro, a "pseudo" é tratada de acordo com seu grupo.

O esquema da figura III.4.1. ilustra o processo descrito.

O tratamento das "pseudos" na segunda fase também é feito de acordo com seu grupo. O processo particular de cada grupo, é feito de acordo com o esquema geral mostrado no diagrama da segunda fase. (figura III.4.2.)

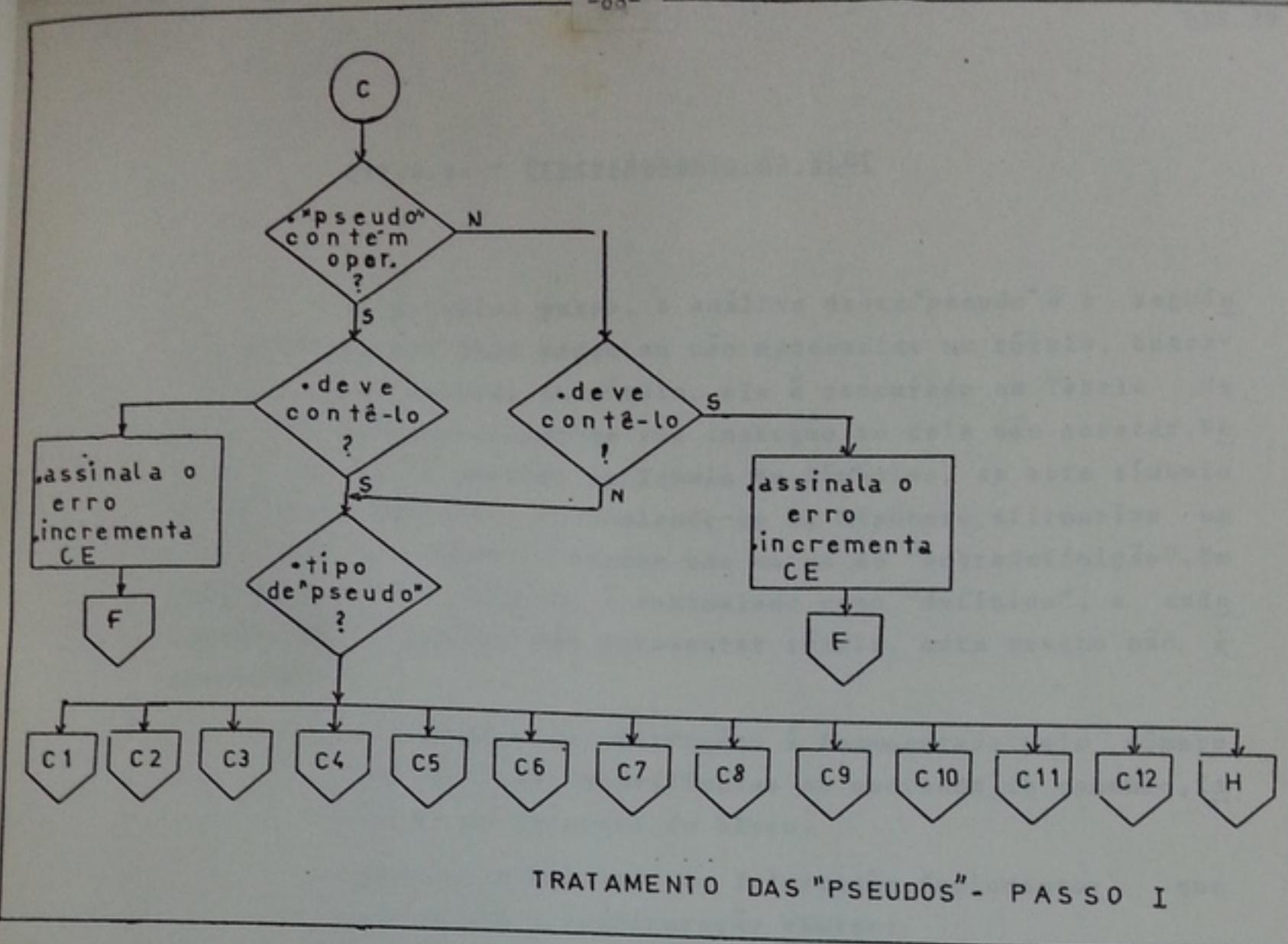


Figura III.4.1. - Tratamento das "pseudos"- Passo I

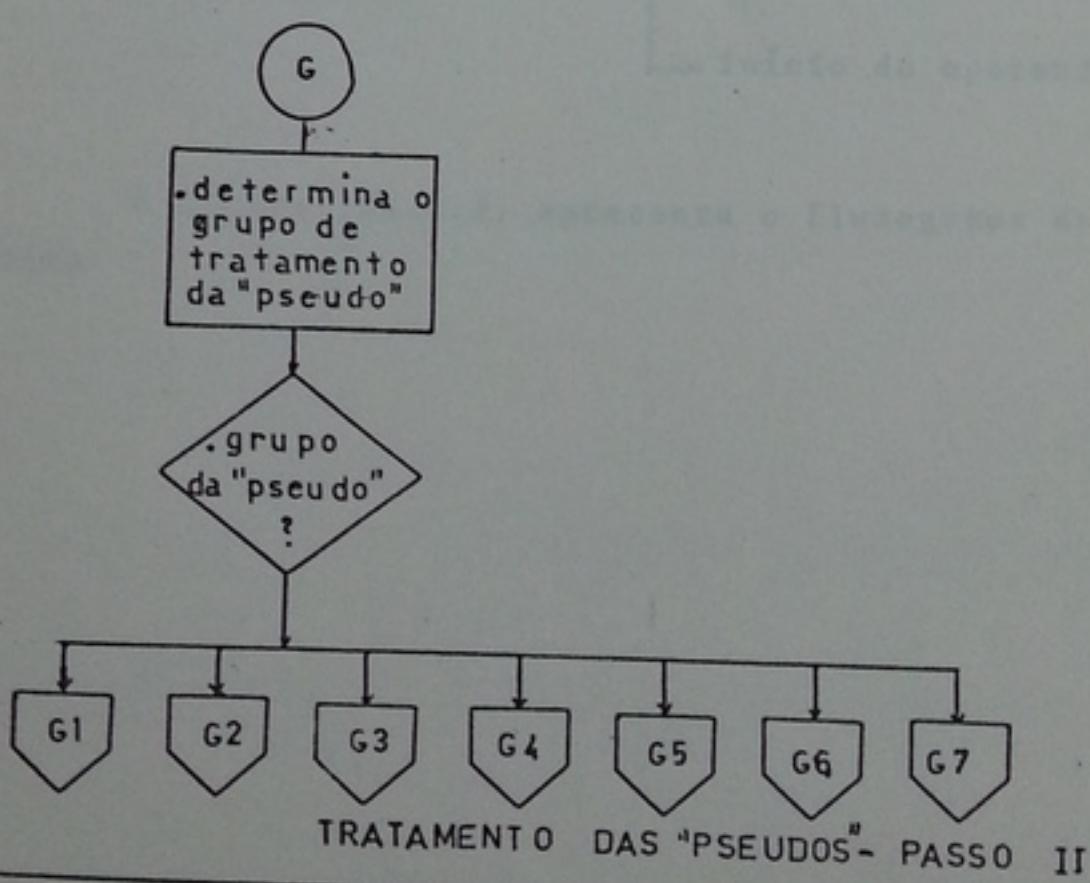
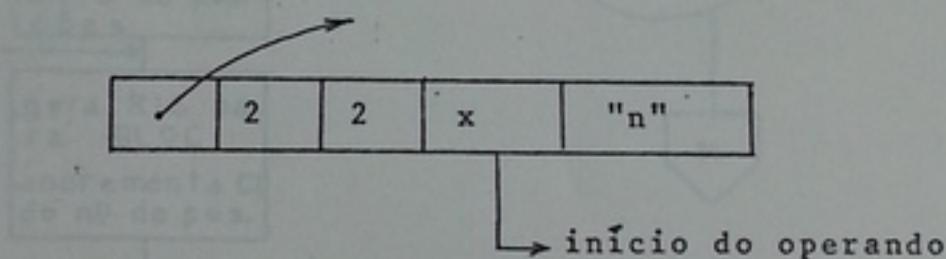


Figura III.4.2. - Tratamento das "Pseudos"- Passo II

III.4.a. - Processamento de BLOC

No primeiro passo, a análise desta "pseudo" é a seguinte: como "pseudos" BLOC podem ou não apresentar um rótulo, busca-se por este. Havendo um rótulo, ele é procurado na Tabela de Símbolos, providenciando-se sua inserção se dela não constar. Verifica-se, se já estiver na Tabela de Símbolos, se este símbolo se encontra definido, assinalando-se na hipótese afirmativa um erro, isto é, o símbolo recebe uma marca de "sobredefinição". Em caso contrário, o símbolo é assinalado como "definido", e endereçado. Se a "pseudo" não apresentar rótulo, este trecho não é executado.

- o Contador de Instruções é incrementado pelo número de posições especificadas no operando do comando, isto é, da extensão do bloco.
- gera-se o Registro de Informação Suplementar que apresenta a configuração abaixo:



A figura III.4.3. apresenta o fluxograma do processo descrito.

Figura III.4.3. - Tratamento da "pseudo".

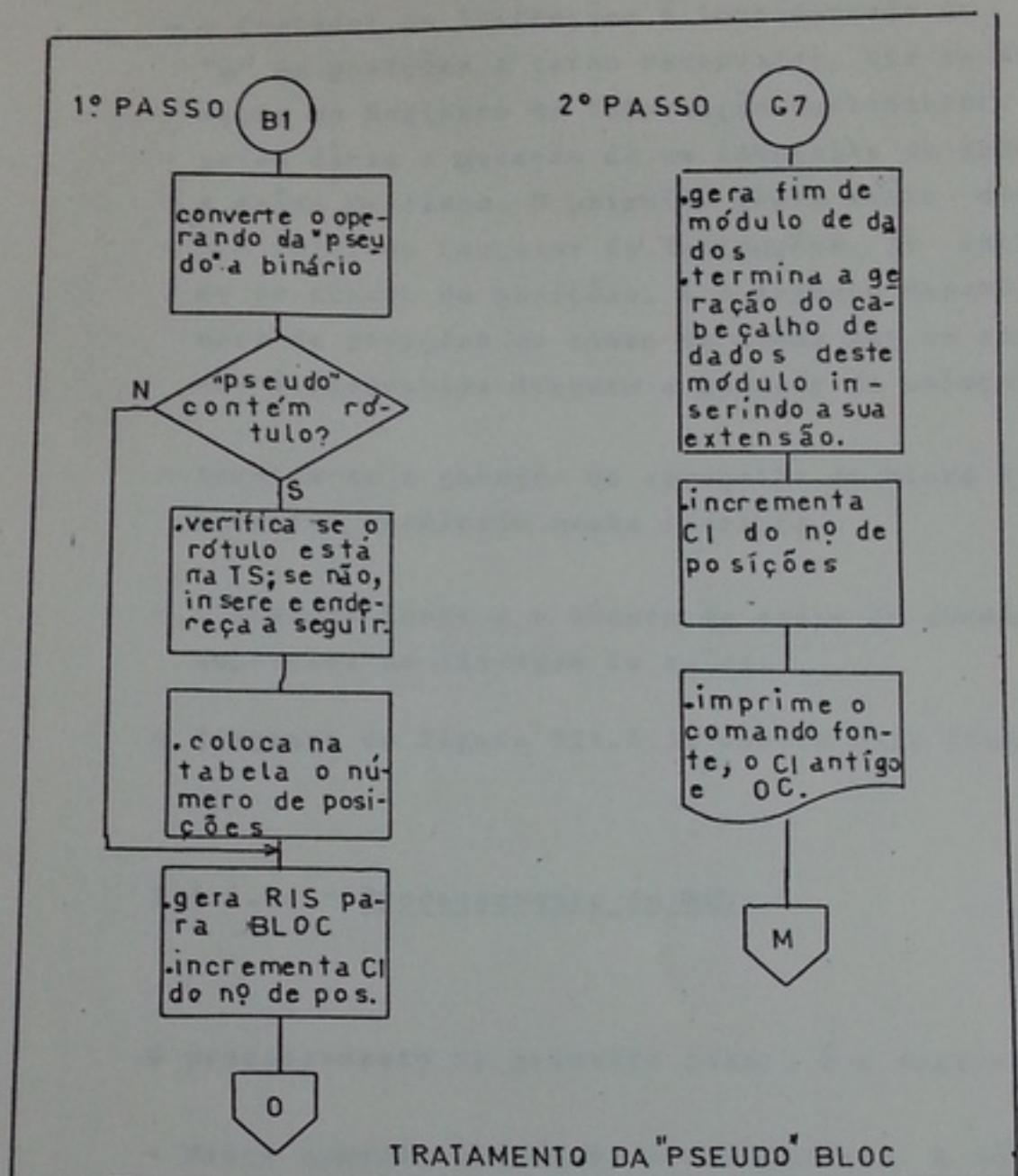


Figura III.4.3. - Tratamento da "pseudo" BLOC

Na segunda fase o processamento é o seguinte:

- o Contador de Instruções é incrementado do número "n" de posições a serem reservadas, que se encontra agora no Registro de Informação Suplementar. Em seguida dá-se a geração de um cabeçalho de dados, para a saída de disco. O primeiro valor deste cabeçalho é o valor do Contador de Instruções, já incrementado do número de posições, e o segundo depende do número de posições no bloco de dados que se segue, e não é preenchida durante a geração do cabeçalho.
- termina-se a geração do cabeçalho do bloco de dados anterior, terminado neste instante.
- o comando fonte e o número de ordem do comando, são impressos na listagem de saída.

O diagrama da figura III.4.3. mostra ente fluxo.

III.4.b. - Processamento de EQU

O processamento no primeiro passo, é o seguinte:

- Neste comando um rótulo é obrigatório. A não existência dele implica em erro de programação.
- Se o comando contém rótulo, verifica-se se este faz parte da Tabela de Símbolos. Se não faz parte, o símbolo é inserido na Tabela. A seguir verifica-se se o operando é um símbolo, uma expressão ou um número.

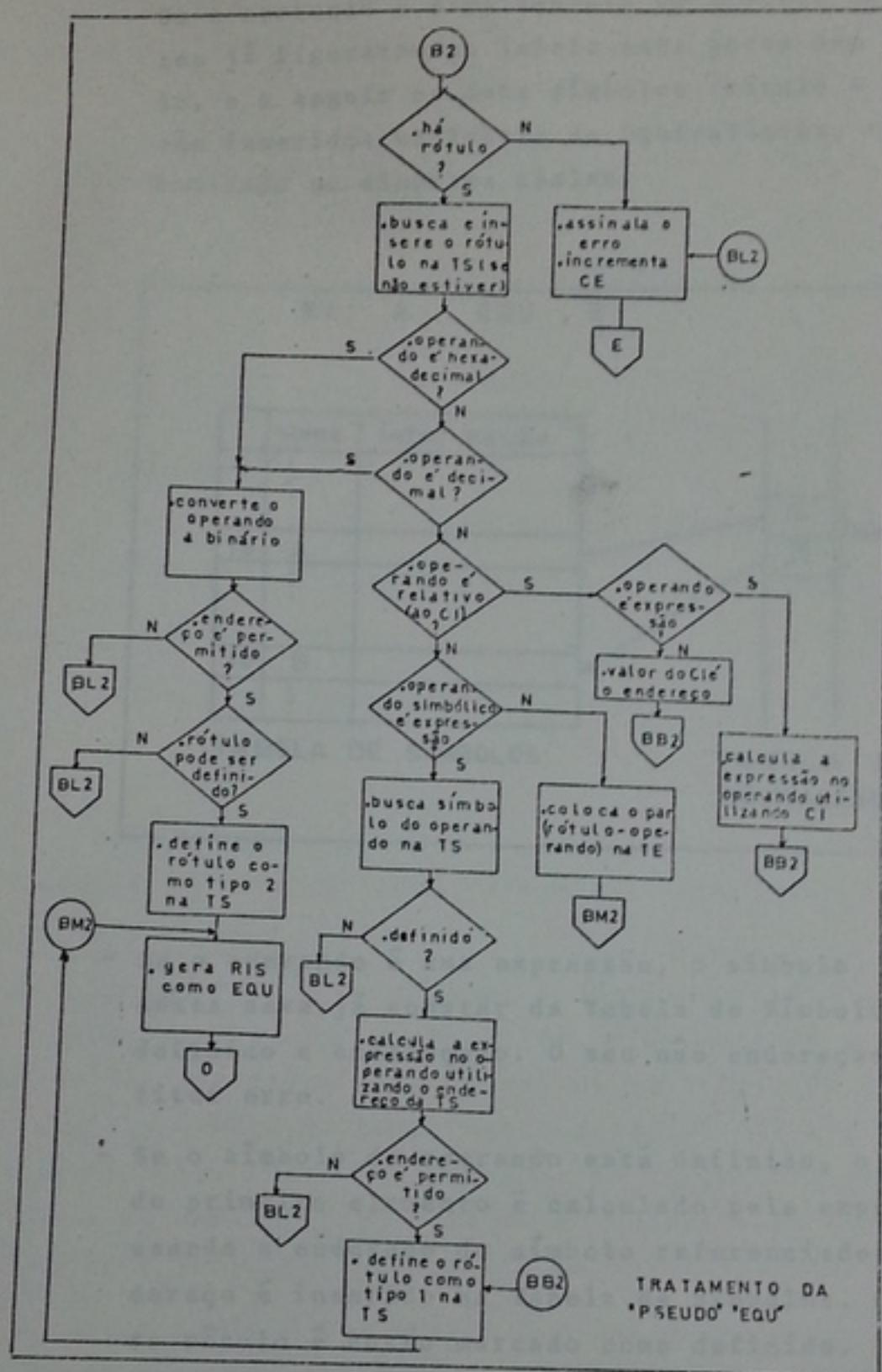
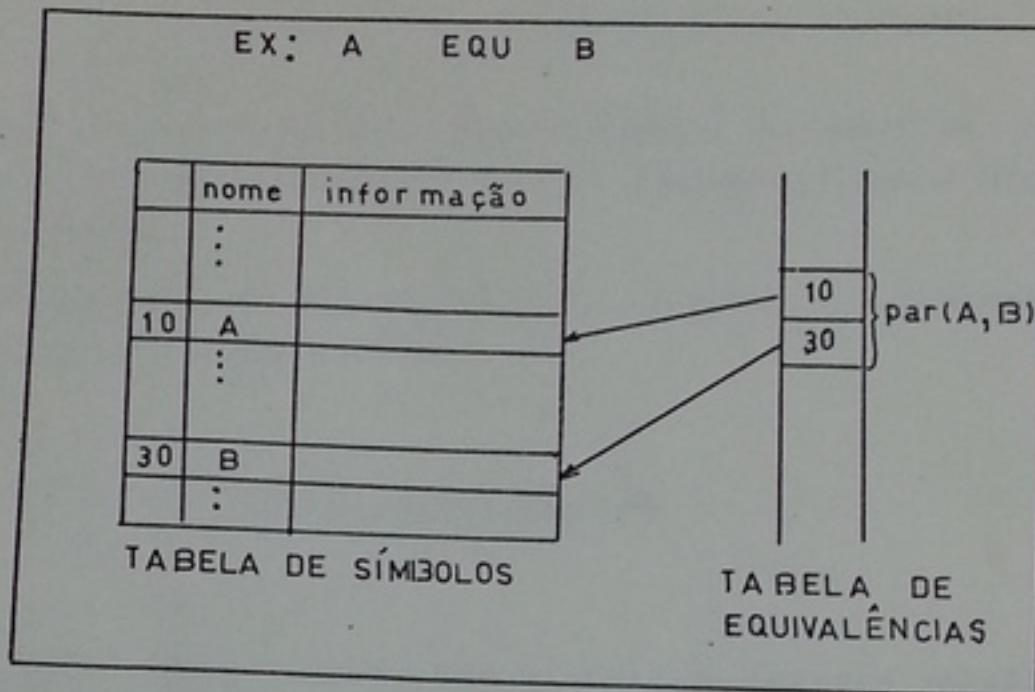


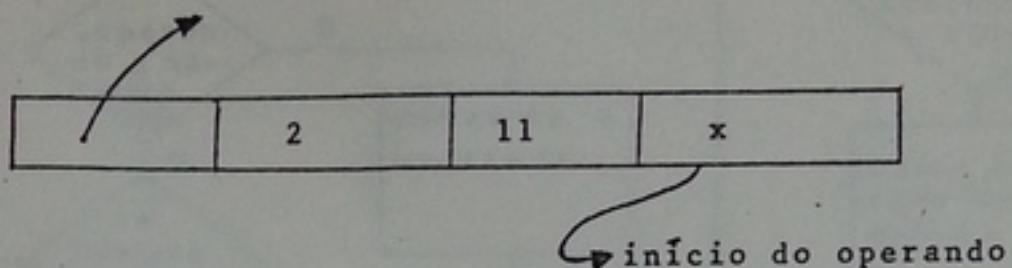
Figura III.4.4. - Tratamento da "pseudo" EQU - Passo I

- Se o operando é simplesmente um símbolo ou os símbolos já figurarem na Tabela este passo não é executado, e a seguir os dois símbolos (rótulo e operando) são inseridos na Tabela de Equivalência, conforme é mostrado no diagrama abaixo.



- Se o operando é uma expressão, o símbolo constante desta deve já constar da Tabela de Símbolos, estar definido e endereçado. O seu não endereçamento constitui erro.
- Se o símbolo do operando está definido, o endereço do primeiro elemento é calculado pela expressão, usando o endereço do símbolo referenciado; este endereço é inserido na Tabela de Símbolos. O símbolo de rótulo é então marcado como definido. Neste caso nem rótulo nem operando entram para a Tabela de Equivalência.
- Se o operando é numérico, o número é convertido a binário e constitui na Tabela de Símbolos o endereço do rótulo. Se o rótulo for um símbolo definido, ele estará agora "sobredefinido". Se não definido, será marcado como tal.

A configuração do Registro de Informação Suplementar em qualquer dos casos vistos será:



Na segunda fase, o tratamento desta "pseudo" se resume a simples listagem do comando fonte, juntamente com o número de ordem do cartão.

O diagrama na figura III.4.4., ilustra o tratamento dado a "pseudo" na primeira fase.

III.4.c. - Processamento de DEFC

Quando DEFC é encontrada na fase de Locação verifica-se a existência de um rótulo, pois esta é uma "pseudo" que pode ou não conter um.

Se ela contém um rótulo, o programa verifica se este já se encontra na Tabela de Símbolos, e se está ou não definido. Se estiver definido, isto constitui um erro, que é assinalado e o símbolo é marcado como "sobredefinido". Se não está definido, é definido, o valor do Contador de Instruções sendo colocado no campo de endereço; o operando é traduzido em binário e esta definição é incluída no Registro de Informação Suplementar.

Se o comando não contém rótulo, o operando é apenas traduzido em binário, entrando em seguida para o Registro de Informação Suplementar. O formato deste registro é o seguinte:

III.35

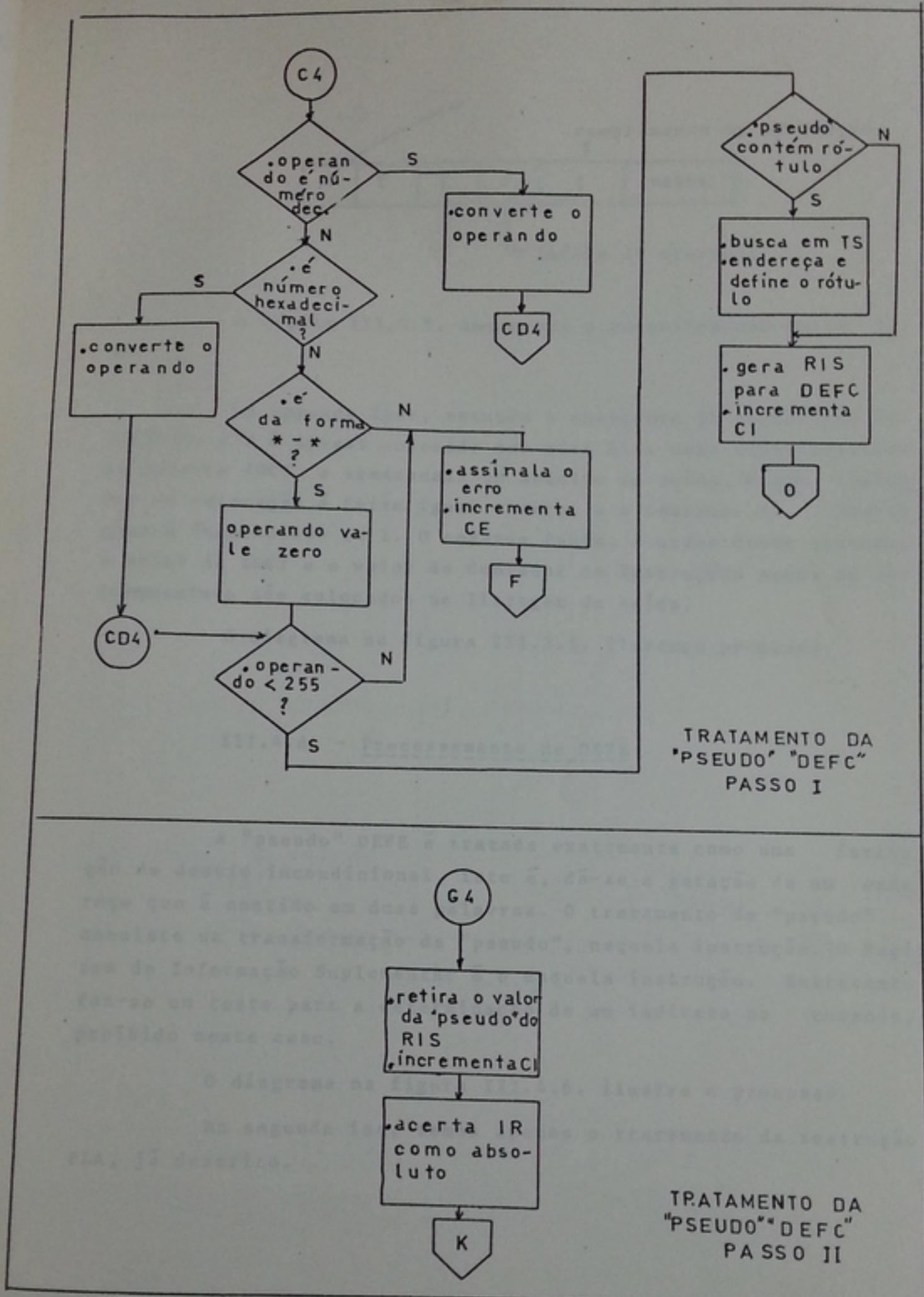
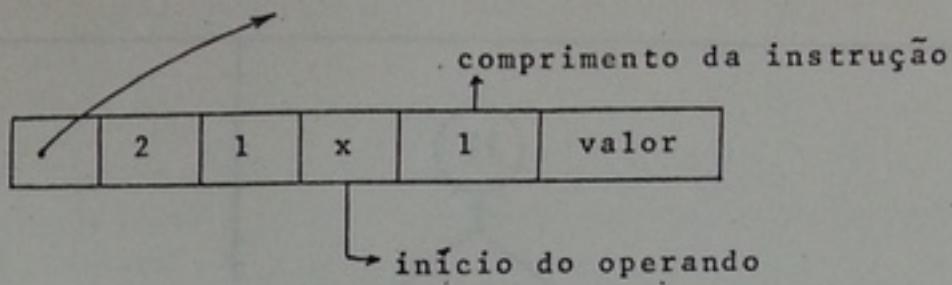


Figura III.4.5. - Tratamento da "pseudo" DEFC-Passo I
e Passo II



A figura III.4.5. documenta o processamento desta fse.

Na segunda fase, estando a constante já convertida em binário, ela é apenas colocada nos oito bits mais significativos da palavra IOCT, e armazenada no arquivo de saída. O seu indicador de relocação é feito igual a zero, e o Contador de Instruções é incrementado de 1. O comando fonte, a ordem deste comando, o valor de IOCT e o valor do Contador de Instruções antes de ser incrementado são colocados na listagem de saída.

O diagrama na figura III.3.5. ilustra o processo.

III.4.d. - Processamento de DEFE

A "pseudo" DEFE é tratada exatamente como uma instrução de desvio incondicional, isto é, dá-se a geração de um endereço que é contido em duas palavras. O tratamento da "pseudo" consiste na transformação da "pseudo", naquela instrução. O Registro de Informação Suplementar é o daquela instrução. Entretanto faz-se um teste para a determinação de um indireto no comando, proibido neste caso.

O diagrama na figura III.4.6. ilustra o processo.

Na segunda fse, temos apenas o tratamento da instrução PLA, já descrito.

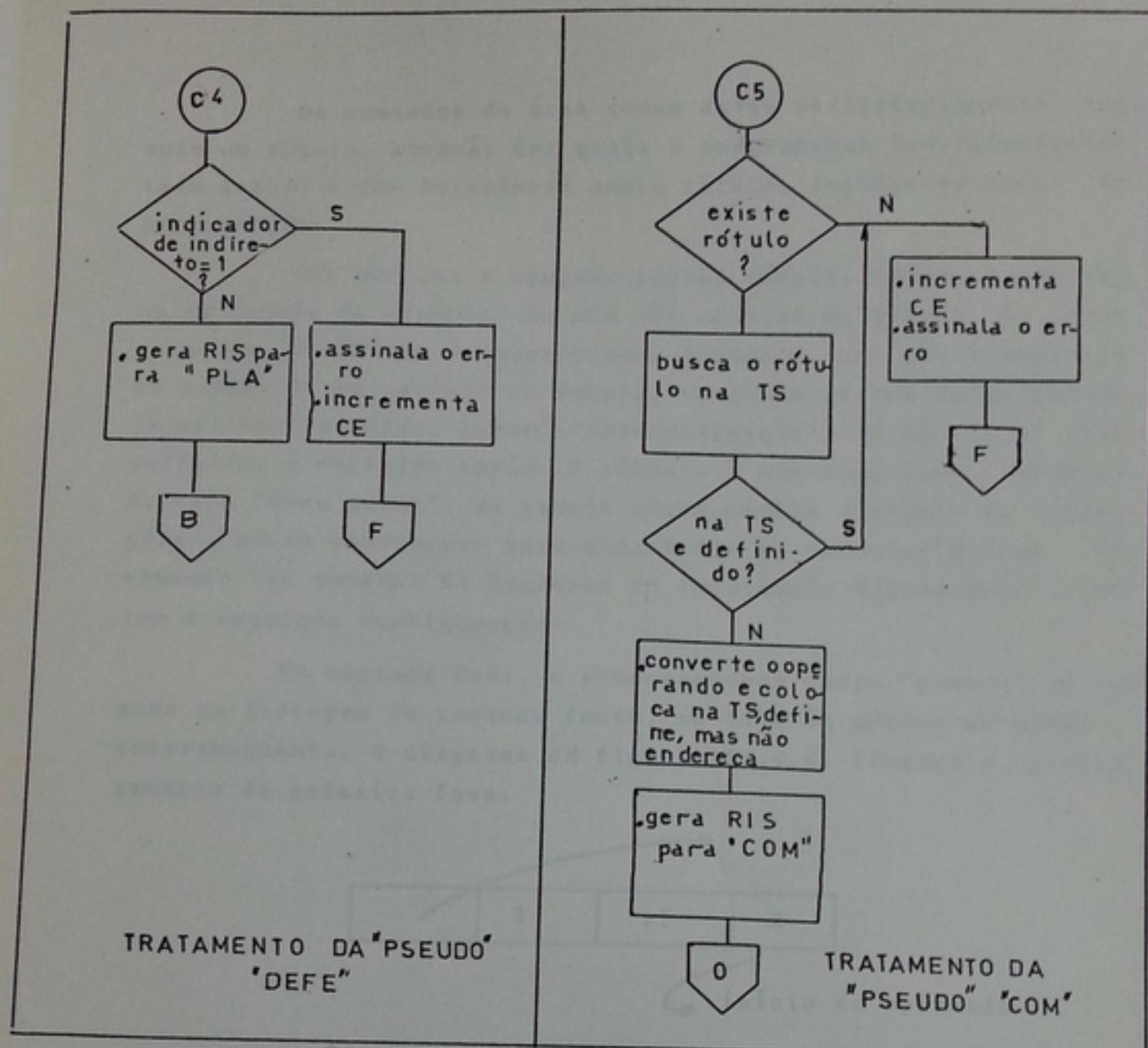


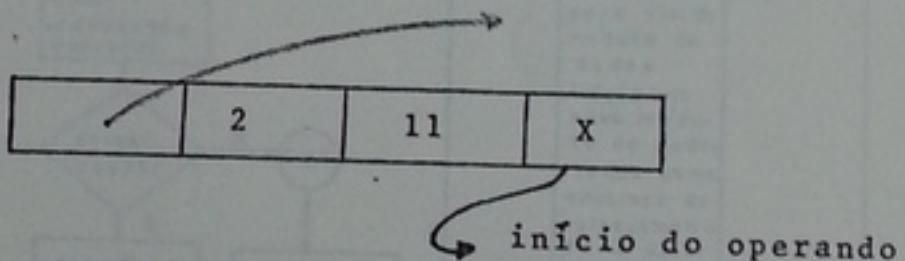
Figura III.4.6. - Tratamento da "pseudo" DEFE e da "pseudo" COM - Passo I

III.4.e. - Processamento de COM

Os comandos de área comum devem obrigatoriamente posuir um rótulo, através dos quais o programador irá referenciar tais dados. A não existência deste rótulo, implica em erro de programação.

Uma vez que o comando possui rótulo, faz-se a sua busca na Tabela de Símbolos. Se ele não constar da Tabela, é nela colocado, o símbolo é marcado como definido, mas não é endereçado ainda. Se já estiver na Tabela, verifica-se sua definição. Se já estiver definido, haverá "sobredefinição". Se não tiver sido definido, é definido então. O símbolo é marcado, como símbolo de tipo "área comum". Na tabela entra também o número de posições a serem reservadas para ele. Todas as características do comando vão constar do Registro de Informação Suplementar que tem a seguinte configuração:

Na segunda fase, o processamento desta "pseudo" se resume na listagem do comando fonte, ao lado do número de ordem correspondente. O diagrama da figura III.4.6. ilustra o processamento da primeira fase.

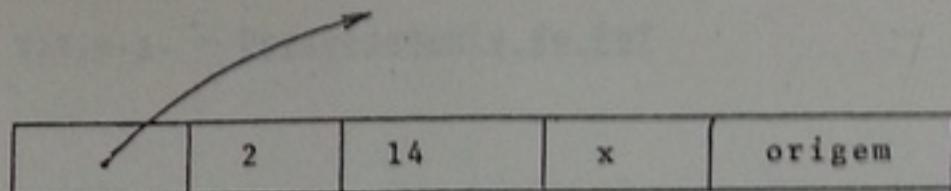


III.4.f. - Processamento de ORG

Um comando ORG, ao ser encontrado recebe o seguinte tratamento: verifica-se se o programa é do tipo sub-rotina quando então ORG não é permitido. Sendo permitido o comando, o Contador de Instruções é reinicializado com a constante que aparece no operando, devidamente convertida a binário.

Não é permitida a existência de um rótulo neste tipo de comando.

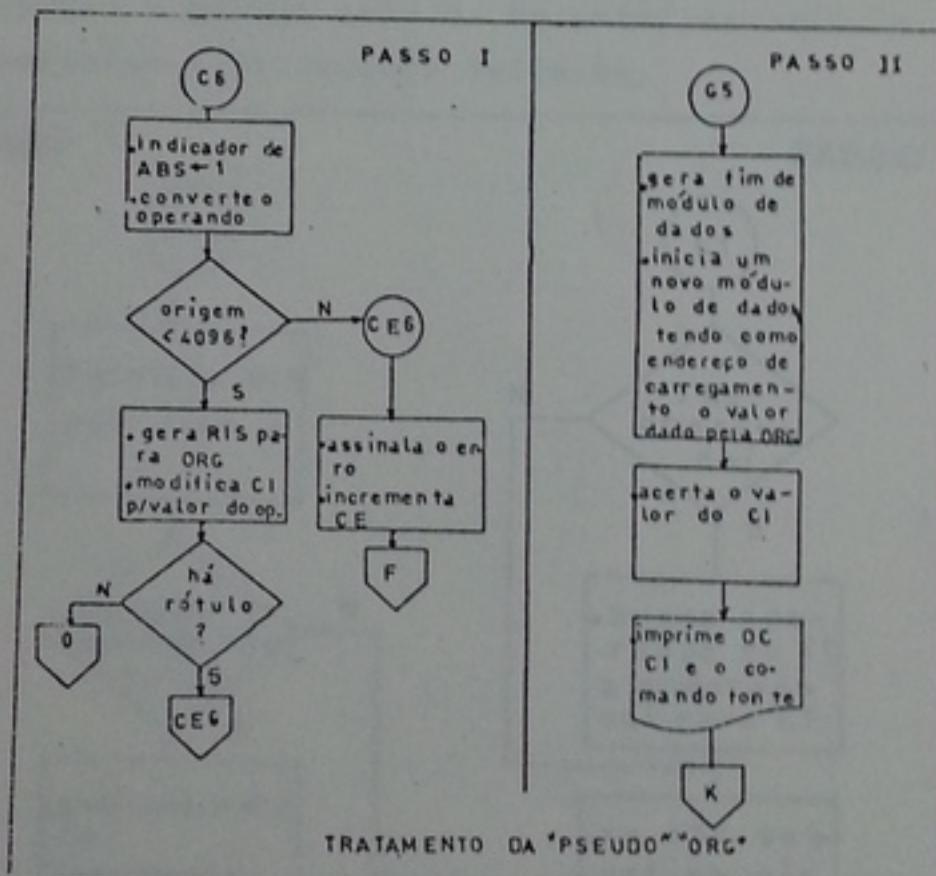
Tratado o Contador de Instruções, organiza-se o Registro de Informação Suplementar para este comando, que tem a forma:



início do operando

A "pseudo" ORG, ao ser encontrada na Segunda Fase, inicia o Contador de Instruções no valor da origem que aparece no Registro de Informação Suplementar. Um novo cabeçalho de dados é gerado, e o bloco de dados anterior é finalizado. O primeiro valor neste cabeçalho, é o novo Contador de Instruções. A segunda palavra do cabeçalho como no caso de BLOC é preenchida depois. O comando fonte, juntamente com o valor do Contador de Instruções, e o número de ordem é impresso na listagem de saída.

A figura III.4.7. apresenta o processamento da "pseudo"



III.4.g. - Processamento de ENT

Quando na Fase de Locação esta "pseudo" é detectada, verifica-se a ocorrência anterior de uma "pseudo" ISS, ou ORG, o que implicaria na ocorrência de erro de programação, com consequente interrupção da análise.

Se entretanto a "pseudo" ENT é permitida, podem ocorrer duas situações:

- esta é a primeira "pseudo" ENT encontrada,
- ocorreram anteriormente outras "pseudos" ENT.

Uma vez encontrada a "pseudo", o seu operando, isto é, o nome do ponto de entrada é pesquisado na Tabela de Símbolos. Se ele não constar da tabela, é nela inserido, e recebe um indicador de tipo igual a 5. Se ela já se encontra nesta tabela, deve ter código 1 ou 2, ou então terá ocorrido um erro. Neste caso, este código é mudado para 8. No primeiro caso, o símbolo estará indefinido e no segundo definido.

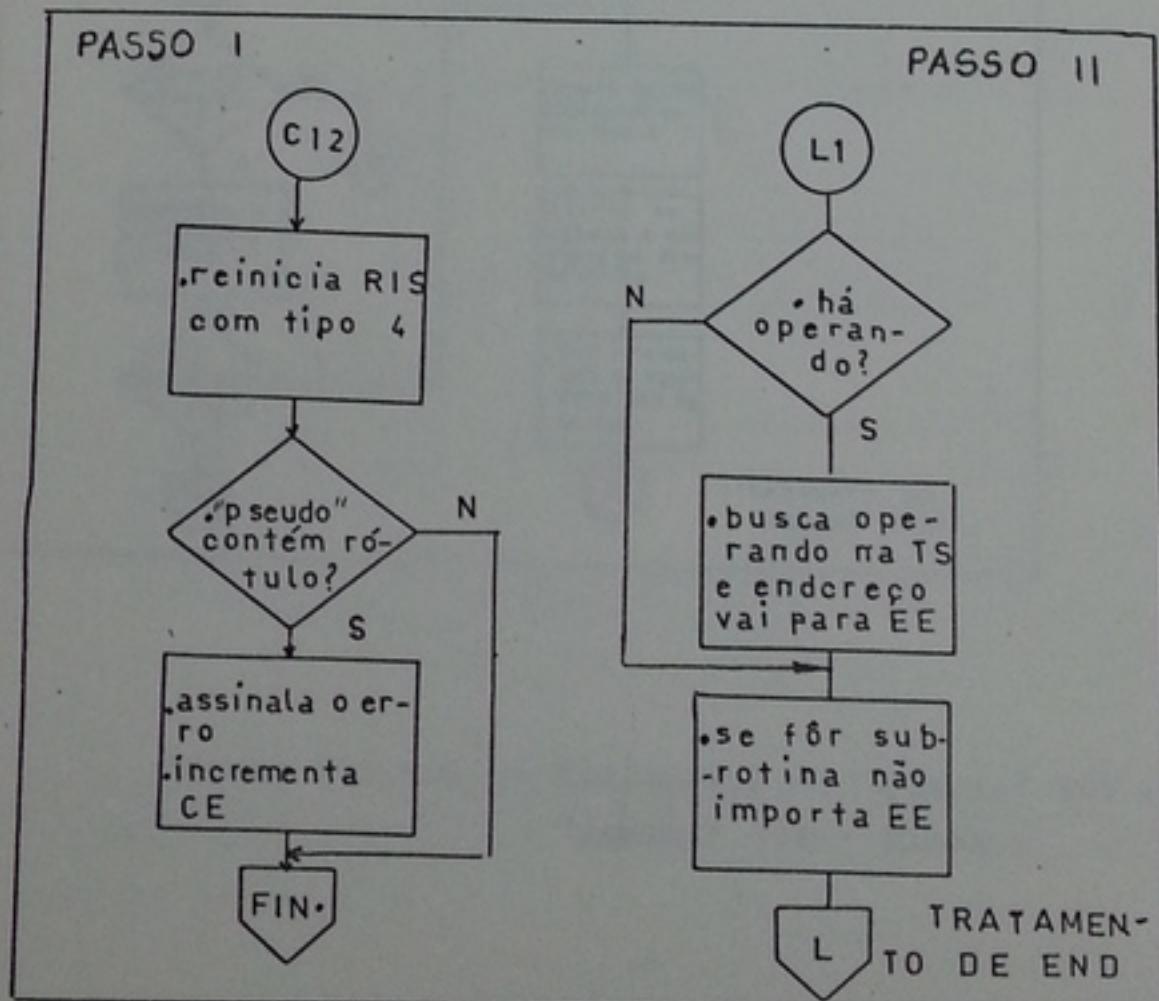


Figura III.4.11. - Tratamento da "pseudo" END

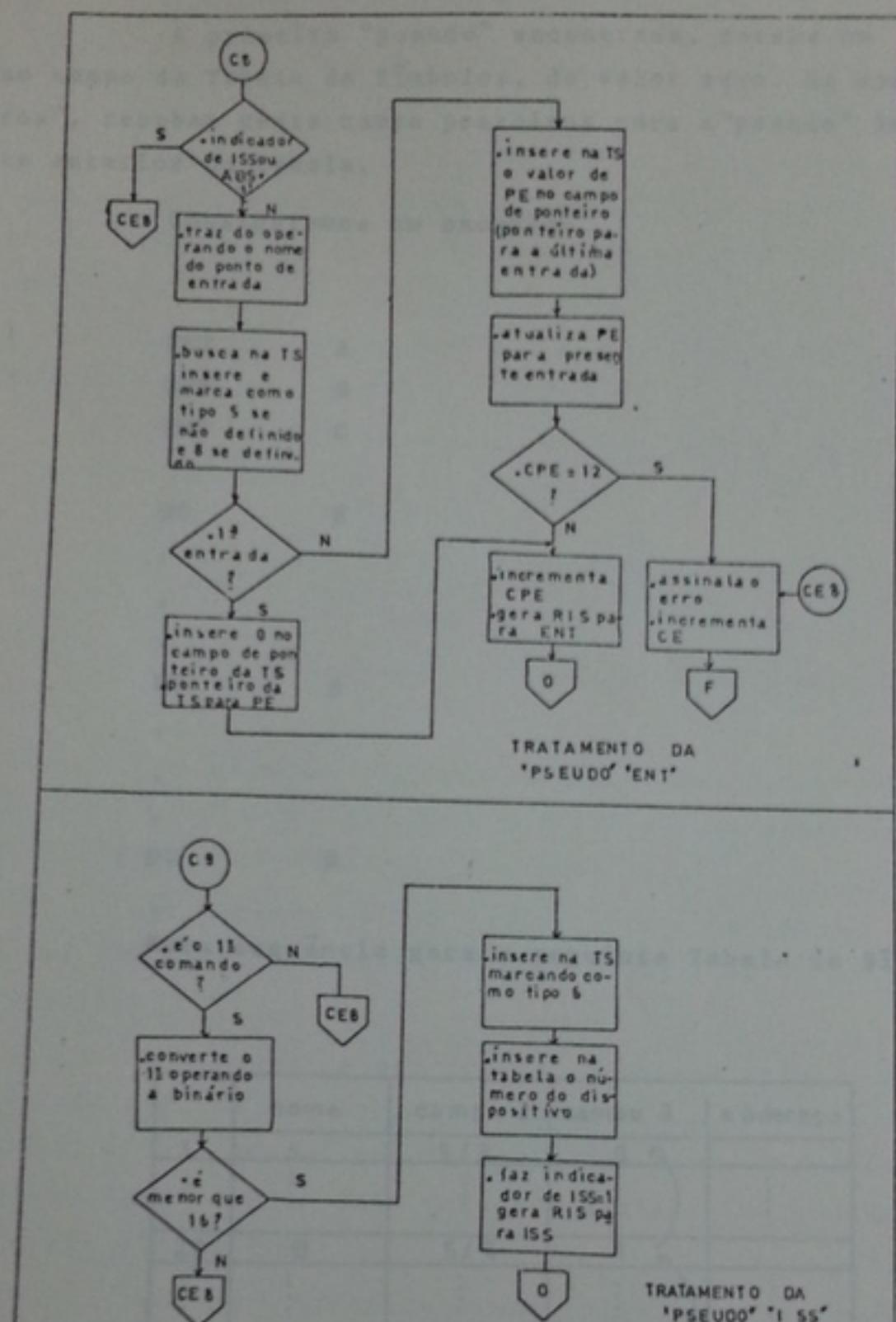


Figura III.4.8. - Tratamento da "pseudo" ENT e da "pseudo" ISS - Passo I

A primeira "pseudo" encontrada, recebe um indicador no campo da Tabela de Símbolos, de valor zero. As demais "pseudo"s, recebem neste campo ponteiros para a "pseudo" imediatamente anterior na tabela.

Consideremos um exemplo:

	ENT	A
	ENT	B
	ENT	C
	.	
A	DC	Ø
	.	
	.	
B	DC	Ø
	.	
	.	
C	DC	Ø

Esta sequência gera a seguinte Tabela de Símbolos:

	nome	campo 2	campo 3	endereço
1	A	5/8	0	
⋮	⋮	⋮	⋮	⋮
20	B	5/8	1	
⋮	⋮	⋮	⋮	⋮
30	C	5/8	20	

Uma tentativa de mais de 12 pontos de entrada, implica em um erro de programação.

III.4.h. - Processamento de EBC e ASC

Na primeira fase, verifica-se a extensão da cadeia, bem como a existência de um rótulo. Se este ocorrer, o programa procura por ele na Tabela de Símbolos, e procede-se como no caso da "pseudo" BLOC.

O Contador de Instruções é incrementado do número de caracteres na cadeia.

O Registro de Informação Suplementar toma a forma:

	2	20	início da cadeia	1	número de elementos na cadeia
		18			

Na segunda fase, o tratamento das "pseudos" EBC e ASC deve fazer a geração de uma série de caracteres EBCDIC ou ASCII, respectivamente.

O operando da instrução, isto é, a cadeia a ser gerada não foi ainda analisada, e esta análise se dá juntamente com a geração.

O caracter a ser gerado, é determinado a partir do operando através da tabela da figura III.4.10. Cada caracter gerado, é colocado no arquivo de saída, com um indicador de relocação igual a Ø. A cada caracter gerado, incrementa-se o Contador de Instruções.

Quando o primeiro elemento da cadeia é gerado, o seu valor é colocado na listagem de saída, juntamente com o valor do Contador de Instruções neste momento, o valor de ordem de entrada do cartão, e o comando fonte. Os demais caracteres gerados, são colocados, um em cada linha de listagem, juntamente com o valor do Contador de Instruções correspondente.

O Registro de Informação Suplementar tem neste caso o seguinte aspecto:

	12	15	X
--	----	----	---

início do operando

No Segundo Passo, o tratamento desta "pseudo" se restringe a listagem do comando juntamente com o número de ordem.

O diagrama da figura III.4.8., ilustra o processamento da "pseudo" na primeira fase.

III.4.i. - Processamento de ISS

Esta "pseudo" é analisada na primeira fase segundo o esquema apresentado na figura III.4.8. Na segunda fase o seu tratamento se limita a listagem do comando juntamente com o número de ordem.

O processamento na primeira fase é o seguinte:

- Verifica-se antes se não houve nenhum comando anterior o que consistiria num erro. Se isso não ocorreu, verifica-se se o número do dispositivo especificado é válido. Se não for válido, ocorreu erro.

Sendo válidos o nome do ponto de entrada da ISS, e a identificação lógica do dispositivo, o primeiro é colocado na Tabela de Símbolos, juntamente com o número de dispositivo lógico em questão, e uma identificação da sua natureza.

Indica-se para o programa, o aparecimento do ISS, a fim de prevenir "pseudos" de definição de pontos de entrada de sub-rotinas, ou comandos ORG. Para isso, posiciona-se um indicador de ISS.

Preenche-se o Registro de Informação Suplementar, que nesse caso tem a forma:

	12	17	X
--	----	----	---

início do operando

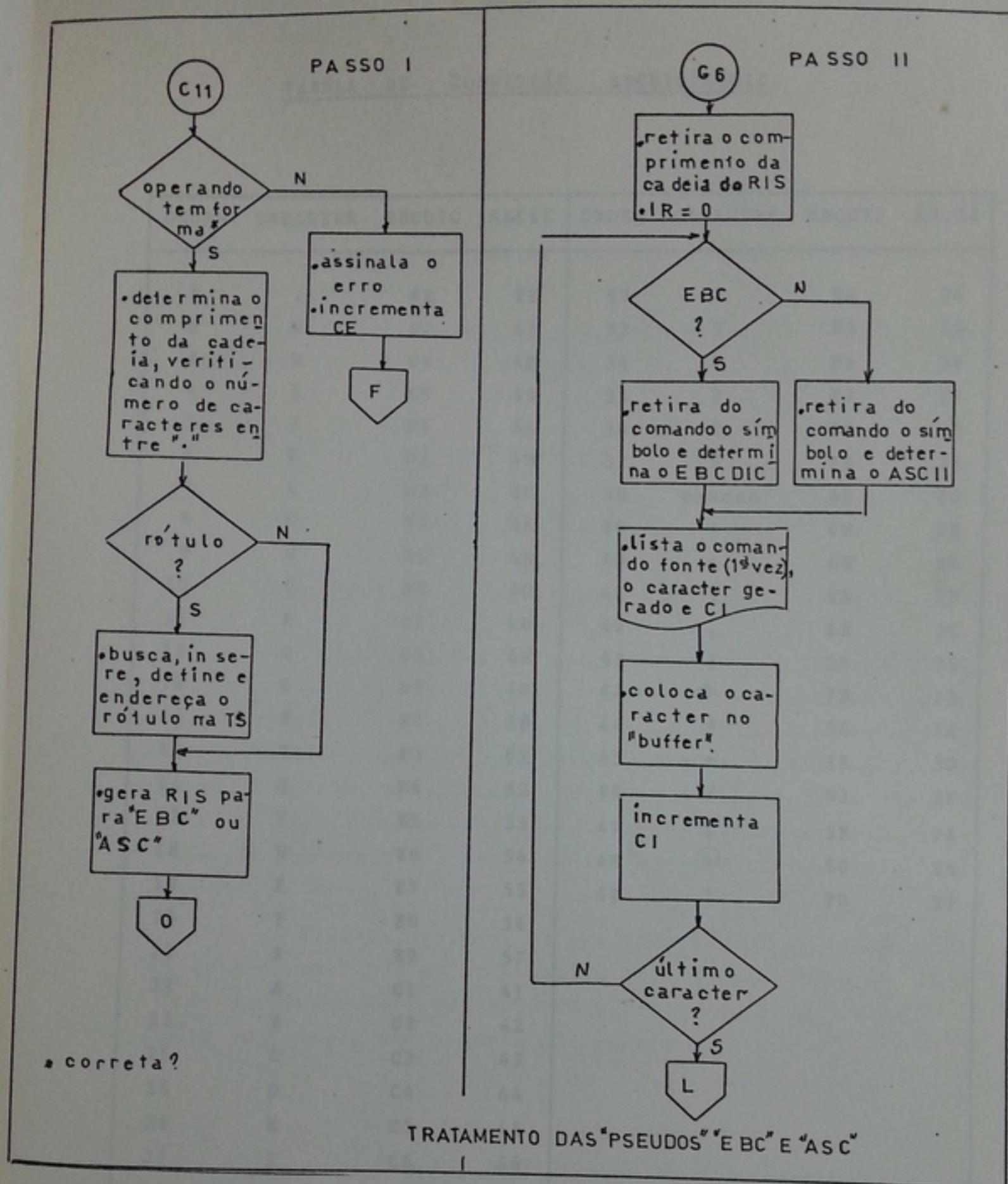


Figura III.4.9. - Tratamento das "pseudos" EBC e ASC - Passo I e Passo II

TABELA DE CONVERSÃO EBCDIC/ASCII

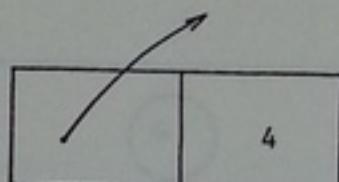
ORDEM	CARACTER	EBCDIC	ASCII	ORDEM	CARACTER	EBCDIC	ASCII
1	.	4B	2E	32	4	F4	34
2	G	C7	47	33	5	F5	35
3	H	C8	48	34	6	F6	36
4	I	C9	49	35	7	F7	37
5	J	D1	4A	36	8	F8	38
6	K	D2	4B	37	9	F9	39
7	L	D3	4C	38	branco	40	20
8	M	D4	4A	39	+	4E	2B
9	N	D5	4B	40	-	60	2D
10	O	D6	4C	41	(4D	28
11	P	D7	4D	42	,	6B	2C
12	Q	D8	4E	43)	5D	29
13	R	D9	4F	44	#	7B	23
14	S	E2	50	44	*	5C	2A
15	T	E3	51	45	=	7E	3D
16	U	E4	52	46	/	61	2F
17	V	E5	53	47	\$	5B	24
18	W	E6	54	48	@	50	26
19	X	E7	55	49	!	7D	27
20	Y	E8	56				
21	Z	E9	57				
22	A	C1	41				
23	B	C2	42				
24	C	C3	43				
25	D	C4	44				
26	E	C5	45				
27	F	C6	46				
28	Ø	FØ	3Ø				
29	1	F1	31				
30	2	F2	32				
31	3	F3	33				

Figura III.4.10

III.4.j. - Processamento de END

Esta "pseudo" ao ser encontrada na Fase de Locação é transformada em um novo tipo de comando, "fim de Montagem" ou "fim de Programa Fonte" (tipo 4).

Para este comando gera-se um Registro de Informação Suplementar com o seguinte aspecto:



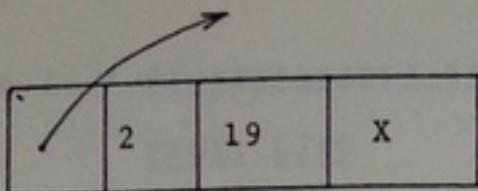
O tratamento desta "pseudo" é mostrado na figura III.4.11.

Se o programa for um programa principal, busca-se por um operando, o qual deve indicar o endereço de execução deste programa. O símbolo que aparece no operando é pesquisado na Tabela de Símbolos, onde já deve estar definido. O indicador de pontos de entrada, recebe um ponteiro para a posição do símbolo na Tabela. Se o símbolo não ocorrer no operando tem-se um erro. No caso de sub-rotinas o endereço de execução não necessita ser procurado.

Na Fase de Geração, ao ser detectada a indicação de "fim de programa" gerada no primeiro Passo, encerra-se a Fase de Geração de comandos passando-se a finalização. A "pseudo" é listada juntamente com a ordem do comando.

III.4.k. - Processamento de CAB

Esta "pseudo" define um cabeçalho com o qual se deve iniciar uma página nova de listagem. Seu tratamento consiste simplesmente na geração do Registro de Informação Suplementar que tem a forma:



Na segunda fase, ao ser encontrada, esta "pseudo" causa a iniciação de uma nova página, tendo por cabeçalho o seu operando.

O diagrama da figura III.4.12 ilustra o seu processamento.

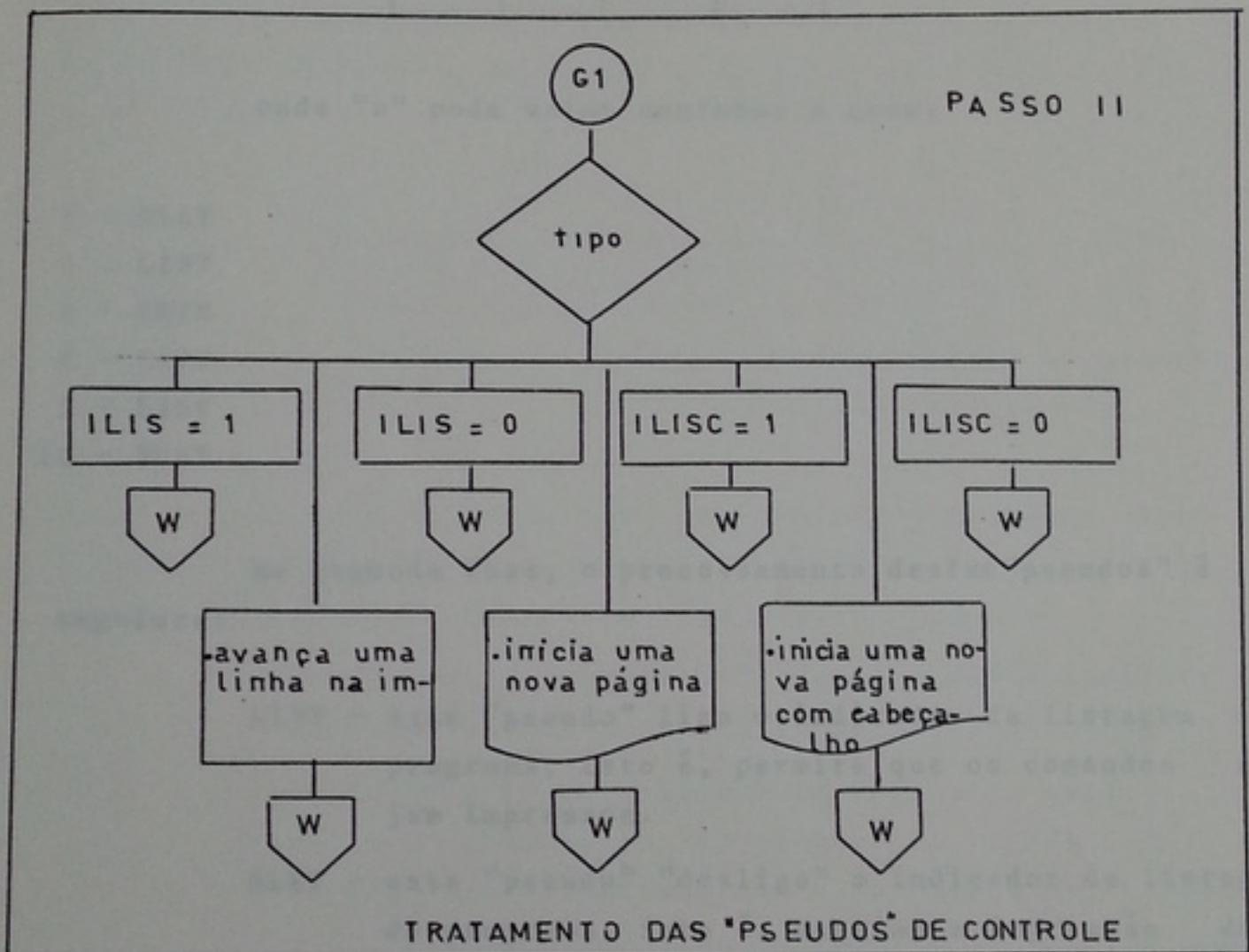
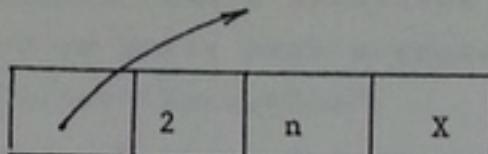


Figura III.4.12. - Tratamento das "pseudos" de controle

III.4.1. - Processamento de NLST, LIST, CNTR, NCNT, PAGE, LINE

Estas "pseudos" não são analisadas na primeira fase, gerando-se simplesmente o seu Registro de Informação Suplementar, que tem a forma:



onde "n" pode valer conforme o caso:

- 8 - NLST
- 7 - LIST
- 9 - CNTR
- 6 - PAGE
- 5 - LINE
- 10 - NCNT

Na segunda fase, o processamento destas "pseudos" é o seguinte:

LIST - esta "pseudo" liga o indicador de listagem do programa, isto é, permite que os comandos sejam impressos.

NLST - esta "pseudo" "desliga" o indicador de listagem do programa, isto é, suprime a impressão dos comandos.

CNTR - esta "pseudo" "liga" o indicador de listagem das pseudos de controle.

LINE - esta "pseudo" faz com que a impressora salte uma linha.

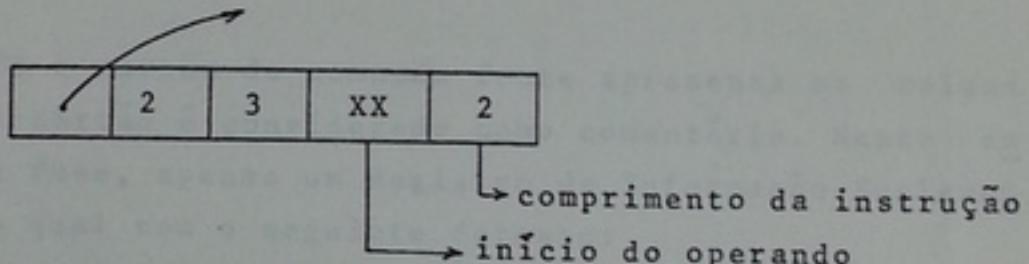
PAGE - esta "pseudo" faz com que o papel da impressora avance para uma nova página. Se houver um cabeçalho definido, este é impresso. Se não, é impresso um cabeçalho "branco".

A figura III.4.12, mostra o tratamento dado a estas "pseudos" em ambas as fases.

III.4.m. - Processamento de CALL

Esta "pseudo" não é analisada na primeira fase, deixando-se esta análise em parte para a segunda fase, e o resto do trabalho para o "Cross-Carregador".

O operando não é analisado, somente é testada a sua existência. O formato do Registro de Informação Suplementar assume o aspecto:



Na segunda fase, deve ser tratado o operando. Este é o nome de um símbolo externo, o qual não deve fazer parte da Tabla de Símbolos. O processo é o seguinte:

- o operando é convertido a "name code" e inserido na saída de disco, onde irá ocupar duas palavras.
- a primeira palavra recebe um indicador especial, indicando "chamada de sub-rotina": 3. O resto do tratamento dado à chamada não é da alçada do "Cross - Montador", mas sim do "Cross-Carregador". A segunda palavra recebe indicador de relocação Ø.
- a listagem de saída recebe as duas palavras em "name code", a ordem de entrada do comando, o valor do Contador de Instruções e o comando fonte.
- o contador de instruções é incrementado de dois.

A figura III.4.13 apresenta o diagrama deste processo.

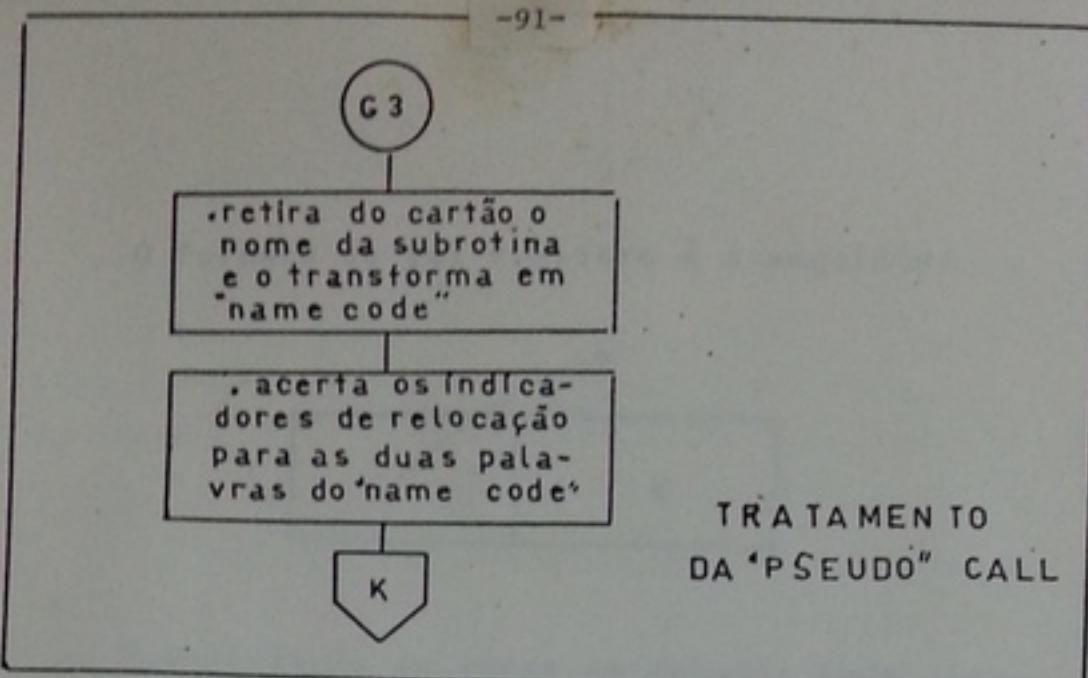
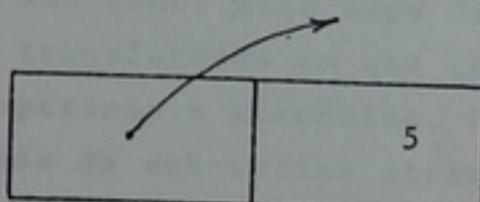


Figura III.4.13. - Tratamento da "pseudo" CALL

III.5. - Tratamento dos comentários

Quando o cartão de comando fonte apresenta na coluna 21 um "*", este cartão é considerado como comentário. Neste caso, na primeira fase, apenas um Registro de Informação Suplementar é gerado, o qual tem o seguinte formato:

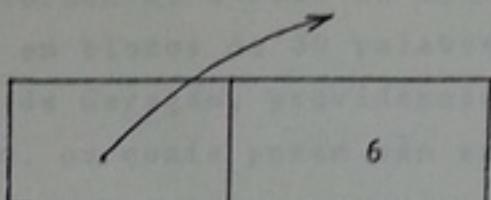


Os comentários na segunda fase, são impressos na listagem de saída, juntamente com a ordem de entrada do comando fonte, desde que naturalmente, esta listagem não esteja inibida através de algum comando NLST.

III.6. - Ocorrência de erros

Quando um comando contém erro, na primeira fase, o seu Registro de Informação Suplementar não é gerado completamente. Conforme a análise é interrompida, um registro especial é gerado, a fim de possibilitar a impressão posterior do programa fonte, juntamente com os erros ao final do Primeiro Passo.

O formato de tal registro é o seguinte:



A ocorrência de erros na segunda fase, deve impedir o programa de ser considerado executável. Neste caso, o erro é listado logo abaixo do comando e um indicador é posicionado indicando esta ocorrência.

III.7. - Ocorrência de "pseudo-macros" (chamadas de sub-rotinas, através do nome)

As "pseudo-macro" são chamadas de sub-rotinas, feitas apenas através de seu nome. Este tipo de comando, ao ser detetado, nesta fase, é transformado em uma chamada de sub-rotina simples, tendo como operando o mnemônico. O procedimento a seguir é o mesmo da chamada de sub-rotina através da "pseudo-CALL".

III.8. - Arquivo de Informação Suplementar

A finalidade de se gerar um Arquivo de Informação Suplementar, é não se analisar as partes de um comando, na Fase de Geração, que já forma analisadas na Fase de Locação. As informações coletadas na primeira fase, e que são fundamentais a geração das instruções na segunda, são armazenadas em registros, cuja configuração é variável e depende da instrução em análise.

Os registros são gravados em disco, constituindo um arquivo o qual é fornecido a Fase de Geração. A gravação destes registros é feita em blocos de 80 palavras cada um. As duas fases, de Locação e de Geração, providenciam mecanismos de manuseio deste arquivo, os quais porém não serão aqui abordados.

A grande vantagem destes registros é impedir a busca de Símbolos na Tabela de Símbolos, durante o Segundo Passo do processo, pois no caso das instruções de operandos simbólicos, eles trazem o ponteiro destes símbolos na Tabela de Símbolos. O acesso a tal tabela, se faz agora de maneira randômica.

III.9. - Tabelas utilizadas

A maior parte do trabalho do "Cross-Montador" consiste em conversão de representações simbólicas, em representações binárias, e para tal, empregam-se uma série de Tabelas. Dois problemas devem aí ser levados em conta: o mecanismo como é organizada a tabela, de maneira que novas entradas possam ser feitas, e as já existentes consultadas, e a semântica, pela qual, a cada símbolo é associado um valor.

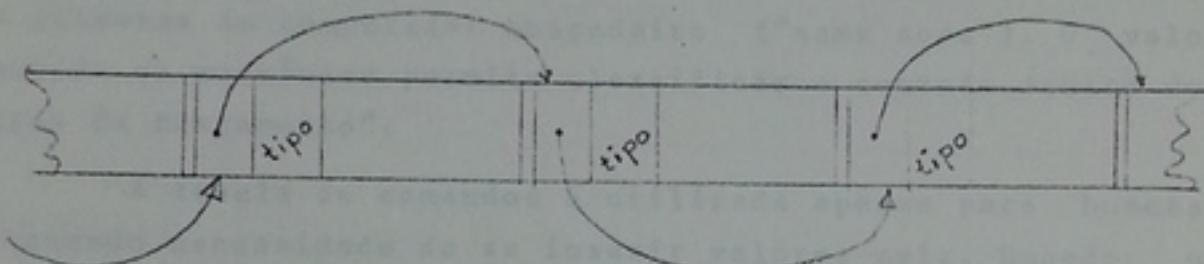
As principais tabelas utilizadas pelo "Cross-Montador" são:

- Tabela de Comandos (TC)
- Tabela de Símbolos (TS)
- Tabela de Equivalência (TE)

De uma maneira geral, estes registros são organizados, na forma de uma lista ligada e portanto compostos de duas partes:

- a primeira, composta de uma palavra, a qual contém um ponteiro para a primeira palavra do registro seguinte (campo de "link");
- a segunda, possui número variável de palavras, dependendo de dois fatores: o tipo de comando em análise, e o comando fonte do programa em questão.

O esquema que se segue, mostra esta configuração:



O registro pode ter no mínimo 2 palavras: o ponteiro, e a primeira palavra da segunda parte. Esta indica sempre, o tipo de comando, com que se está tratando. Dependendo deste valor, uma série de atitudes são tomadas na Segunda Fase, para a geração de uma instrução. Estas atitudes, não dependem unicamente do tipo, mas na maioria das vezes, dependem principalmente das outras palavras que se seguem.

Os comandos podem ser dos seguintes tipos:

- 1 - instrução propriamente dita
- 2 - pseudo - instrução
- 3 - "pseudo - macro"
- 4 - indicação de fim de programa
- 5 - comentários
- 6 - erro

III.9.a. - Tabela de comandos

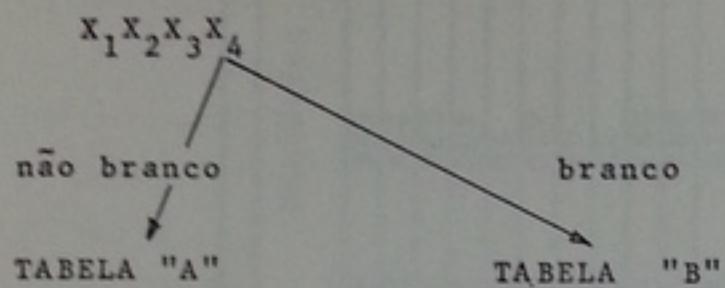
Esta tabela, apresenta duas características importantes: possui número constante de entradas, e os valores das entradas são conhecidos a priori. Como os mnemônicos podem ser de dois tipos principais: instruções de linguagem de máquina e "pseudos", esta tabela foi dividida em duas outras, respectivamente, de instruções de linguagem de máquina e de "pseudos". Estas tabelas fazem parte integrante do "Cross-Montador". As entradas, constituídas pelos mnemônicos, possuem três ou quatro letras, e podem ser assim representadas numa forma compactada, em duas palavras do computador hospedeiro ("name code"). O valor associado ao mnemônico permite classificar o comando dentro dos "grupos de tratamento".

A tabela de comandos é utilizada apenas para buscas, não havendo necessidade de se inserir valores nela. Quando um determinado mnemônico não se encontra nela, este mnemônico não faz parte da linguagem. Para a determinação do valor, pesquisa-se primeiro a tabela de mnemônicos e depois a de "pseudos". Pode-se ainda pesquisar a tabela de símbolos externos, considerada como uma extensão da tabela de comandos, gerada porém em tempo de execução, ao aparecerem comandos "EXT".

Dentre as organizações permitidas para a tabela de comandos, escolheu-se as apresentadas nas figuras III.9.1, e III.9.2.

Estas tabelas são organizadas em um misto de organização em árvore e organização estatística. Cada uma delas é dividida em duas sub-tabelas, de acordo com o valor do último caractere do mnemônico ou "pseudo". Se este valor for "branco", então o mnemônico é buscado na primeira sub-tabela, e se o caractere for diferente de "branco", ele é procurado na segunda sub-tabela.

ORGANIZAÇÃO DA TABELA DE "PSEUDOS"



ordem	mnemônico	GT1	GT2
1	DEF _C	3	4
2	BLOC	1	7
3	CALL	14	3
4	DEF _E	4	-
5	LINE	13	1
6	PAGE	13	1
7	LIST	13	1
8	NLST	13	1
9	CNTR	13	1
10	NCTR	13	1

ordem	mnemônico	GT1	GT2
1	COM	5	2
2	EQU	2	2
3	END	12	-
4	ORG	6	5
5	ENT	8	2
6	ISS	9	2
7	ASC	11	2
8	CAB	13	1
9	EBC	11	6
10	EXT	10	-

GT1 = Grupo de Tratamento no Passo 1

GT2 = Grupo de Tratamento no Passo 2

Figura III.9.1.

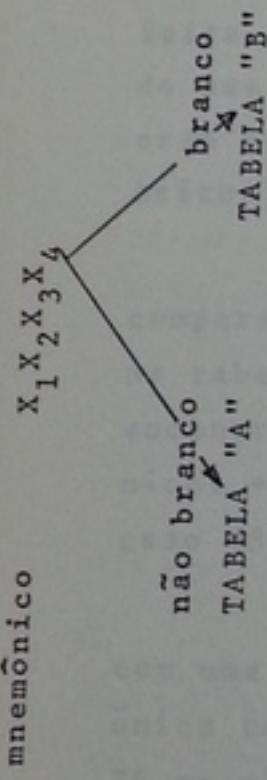


TABELA "A"

	Mnemônico	Grupo de tratamento	Máscara
1	CARI	3	2 DD00
2	CMP1	4	7 8200
3	SOMI	3	2 D800
4	NAND	3	2 D400
5	PLAZ	1	1 B000
6	PLAN	1	1 A000
7	LIMP	10	4 8000
8	CMP2	4	7 8300
9	PLAX	1	1 1000
10	PARE	4	7 9D00
11	ARMX	1	1 3000
12	CARRX	1	1 5000
13	SOMX	1	1 7000
14	ENTR	2	6 C040
15	PERM	4	7 9800
16	SLTM	6	8 9100
17	SLVM	6	8 9500
18	UNEG	4	7 8600
19	INIB	4	7 9A00

TABELA "B"

	Mnemônico	GT1	GT2	Máscara
1	PLA	1	1	00000
2	PUG	1	1	F0000
3	ARM	1	1	20000
4	TRE	4	7	99000
5	CAR	1	1	40000
6	SLV	6	8	94000
7	TRI	4	7	9E000
8	XOR	3	2	D2000
9	DD	8	3	D1000
10	INC	4	7	85000
11	UM	4	7	81000
12	SOM	1	1	60000
13	PNL	7	5	88000
14	SAI	2	6	C0800
15	SAL	2	6	C0200
				GEV

GT1 = Grupo de Tratamento no Passo 1
 GT2 = Grupo de Tratamento no Passo 2

Dentro de cada sub-tabela, procurou-se dar uma organização estatística no arranjo dos mnemônicos e "pseudos", de maneira que aqueles que aparecem mais frequentemente se situam nas primeiras posições desta.

Dentro de cada uma destas sub-tabelas a busca será do tipo linear; o arranjo dos elementos de acordo com as frequências de aparecimento permite que o tempo de busca seja minimizado.

Organizada conforme exposto, a pesquisa é bastante eficiente, uma vez que a sub-tabela com maior número de mnemônicos tem a extensão de 30 (em média tem-se 15 comparações até que seja encontrado). Por outro lado, sendo estas tabelas organizadas com base em uma estatística realizada utilizando-se programas já em fase de implementação, o número médio de comparações é ainda menor. Dois pontos, entretanto, devem ser observados em relação a esta estatística: em primeiro lugar ela foi feita com base em um número muito pequeno de programas, fornecendo assim uma amostra reduzida, e em segundo, estes programas eram orientados para "software" básico (os primeiros sendo escritos), o que pode ter contribuído para viciar a amostragem.

No caso da busca na tabela de "pseudos", o número de comparações é de 5. Considerando-se que a busca se dá primeiro na tabela de mnemônicos, o número médio de comparações para se encontrar uma "pseudo" será em média 35 (30 elementos de mnemônicos + 5 na tabela de "pseudos") e de 40 comparações no pior caso (30 + 11).

Estes resultados visam comparar o método utilizado com uma busca simples, isto é, a colocação das instruções em uma única tabela, para a qual teríamos um número de comparações de 25 em média para as instruções e 49 no pior caso, e de 59 em média para as pseudos contra 70 no pior caso.

III.9.b. - Tabela de Símbolos

Esta tabela é construída na fase de Locação, durante a sub-fase de leitura dos comandos. A tabela de símbolos apresenta todos os símbolos encontrados no programa fonte, tais como:

rótulos de instruções, operandos, elementos integrantes da área comum, constantes, cadeias de caracteres, pontos de entrada de sub-programas. O par constituinte da tabela: ("entrada" - "valor associado"), é formado pelo nome do símbolo e um conjunto de informações a seu respeito.

Esta tabela é utilizada tanto para busca, como para inserção destes. A organização da tabela é linear.

Um símbolo sendo incluído na tabela, ocupará sempre a última posição desta, ou seja, a primeira posição livre. Ao se incluir o símbolo, são incluídos juntamente uma série de indicadores. A organização da Tabela de Símbolos é bastante complexa, cada entrada sendo organizada de acordo com o elemento que a caracteriza.

A escolha do método linear foi feita em primeiro lugar devido a sua simplicidade.

Entretanto, o ponto que influiu mais fortemente nessa escolha foi o de que a tabela gerada não pode ser reorganizada. Isto se deve ao fato de que a geração de um registro de informação suplementar inclui o armazenamento de ponteiros para a Tabela de Símbolos, os quais serão utilizados no tratamento de instruções no Segundo Passo do procedimento. Por meio deste ponteiro o acesso as informações na Tabela de Símbolos se dá de maneira randômica nesta fase. A fim de não destruir estes ponteiros a tabela não pode ser reorganizada, eliminando os métodos de busca que incluem reorganização para a tabela.

A utilização de um método linear de busca não é entre tanto muito crítico. Isto pode ser deduzido do fato de que sendo o sistema de pequeno porte, a extensão dos programas tende a ser pequena; desta maneira a Tabela de Símbolos também deve ser de tamanho reduzido.

III.9.b.1. - Organização_da_tabela_de_símbolos

Cada entrada desta tabela é formada por 4 campos. O primeiro campo contém o nome do símbolo. O segundo campo contém informação sobre o tipo da variável, e o terceiro complementa esta informação. O quarto campo é o campo de endereço.

A tabela da figura III.9.3. apresenta o significado do terceiro e quarto campo, dependendo de como ocorram os símbolos:

T-A-B-L-A

Valor no campo	Significado	Conteúdo do campo 3	Conteúdo do campo 4 (endereço)
0	Símbolo apareceu como operando de um comando, mas ainda não foi definido		
1	Símbolo apareceu como rótulo de uma instrução, ou então da "pseudocomando" DEPC		Absoluto/relativo conforme o programa
2	Símbolo foi endereçado de modo absoluto, através de uma "pseudocomando" EQU		Absoluto
3	Símbolo foi definido através de uma "pseudocomando" COM	Número de posições a serem reservadas em área comum	Absoluto/relativo conforme o programa
4	Símbolo foi definido através de uma "pseudocomando" BLOC	Número de posições ocupadas pelo bloco	Absoluto
5	Símbolo é ponto de entrada de uma sub-rotina, porém ainda não ocorreu como rótulo	O terceiro campo contém um ponteiro para a próxima entrada da tabela, que por sua vez contém um ponto de entrada de sub-rotina simples. Se este valor for ZERO, significa, que se trata do único ponto de entrada desta rotina, ou que este ponto é o último da cadeia	
6	Símbolo é ponto de entrada de uma sub-rotina de entrada/saída, porém ainda não ocorreu como rótulo de instrução	Número do dispositivo lógico para o qual se destina a sub-rotina	
7	Símbolo define nome do programa		Relativo
8	Símbolo definido que é ponto de entrada de uma sub-rotina	Idem valor 5	
9	Símbolo definido, que é ponto de entrada de uma sub-rotina de entrada e saída	Idem valor 6	Relativo
10	Símbolo sobredefinido		

O quarto campo é o campo de endereço. Este pode ser absoluto ou relativo ao início do programa. A tabela apresenta o tipo de endereço de acordo com os valores do 2º campo.

III.9.c. - Tabela de equivalências

Esta tabela é gerada durante o Primeiro Passo, na sub-fase de leitura dos comandos. Uma entrada é feita a ela toda vez que aparece uma "pseudo" que faz equivalência entre dois símbolos, isto é, dois "nomes" diferentes que devem ter o mesmo endereço. Esta tabela é composta de pares de pontos tais que: o primeiro é um ponteiro para a tabela de símbolos, onde se encontra o primeiro símbolo, e o segundo é um ponteiro para o segundo símbolo da equivalência na mesma tabela.

O método de busca consiste em se percorrer a lista no fim do Primeiro Passo, endereçando os pares de pontos, até que estes se esgotem, ou não possam mais ser endereçados (condição de erro).

III.10. - Saídas do "Cross-Montador"

Foi mencionado anteriormente que o Processador produz duas saídas: uma através da impressora, constituída pelos comandos fonte juntamente com o código gerado; outra, uma saída em disco, a ser utilizada pelo "Cross-Carregador", ao produzir o programa que será processado no minicomputador.

Vejamos agora a composição de cada uma destas saídas.

III.10.a. - Saída Impressa

Esta saída, destina-se principalmente à documentação do programa. Assim sendo, ela deve apresentar toda a informação de interesse à documentação.

Compõe-se, esta listagem, da ordem de entrada do cartão, da posição da instrução gerada, na memória (relocável ou absoluta, dependendo do tipo de programa que se tem), do código gerado e da listagem do cartão que contém a instrução. Na eventualidade de ocorrência de um erro durante o Segundo Passo, a mensagem de erro é impressa logo abaixo do comando. Nesta listagem, a posição de memória ocupada pela instrução, bem como o código gerado, são impressos em formato hexadecimal, enquanto a ordem de entrada do cartão é impressa em decimal.

As instruções do minicomputador podem ser de dois tipos: longas e curtas. As instruções longas são compostas de 16 "bits", enquanto as curtas são compostas por apenas 8 "bits". Desta maneira, a listagem de uma instrução longa deve produzir 4 dígitos hexadecimais, e a listagem de uma instrução curta dois dígitos hexadecimais.

A listagem das posições de memória, onde se encontram as instruções geradas, se compõe de 3 dígitos hexadecimais (000 a FFF), compreendendo, desta maneira, 4 K palavras do minicomputador.

III.10.b. - Saída de disco

Esta é a mais importante saída produzida pelo sistema. A saída de disco é composta de uma linguagem objeto relocável. Esta código necessita naturalmente de um programa "Cross - Carregador", o qual produz a saída executável do sistema.

Sendo a linguagem objeto gerada relocável, ela deve ser composta do código, e de uma série de indicadores, os quais permitam a relocação deste código, para gerar o código final.

Uma vez que este sistema foi preparado visando-se a sua utilização em conjunto com o sistema "Cross-Carregador" existente, a saída de disco foi adaptada à entrada daquele programa. A saída é a mesma produzida pelo sistema IBM-1130, e pode ser vista na Figura III.10.

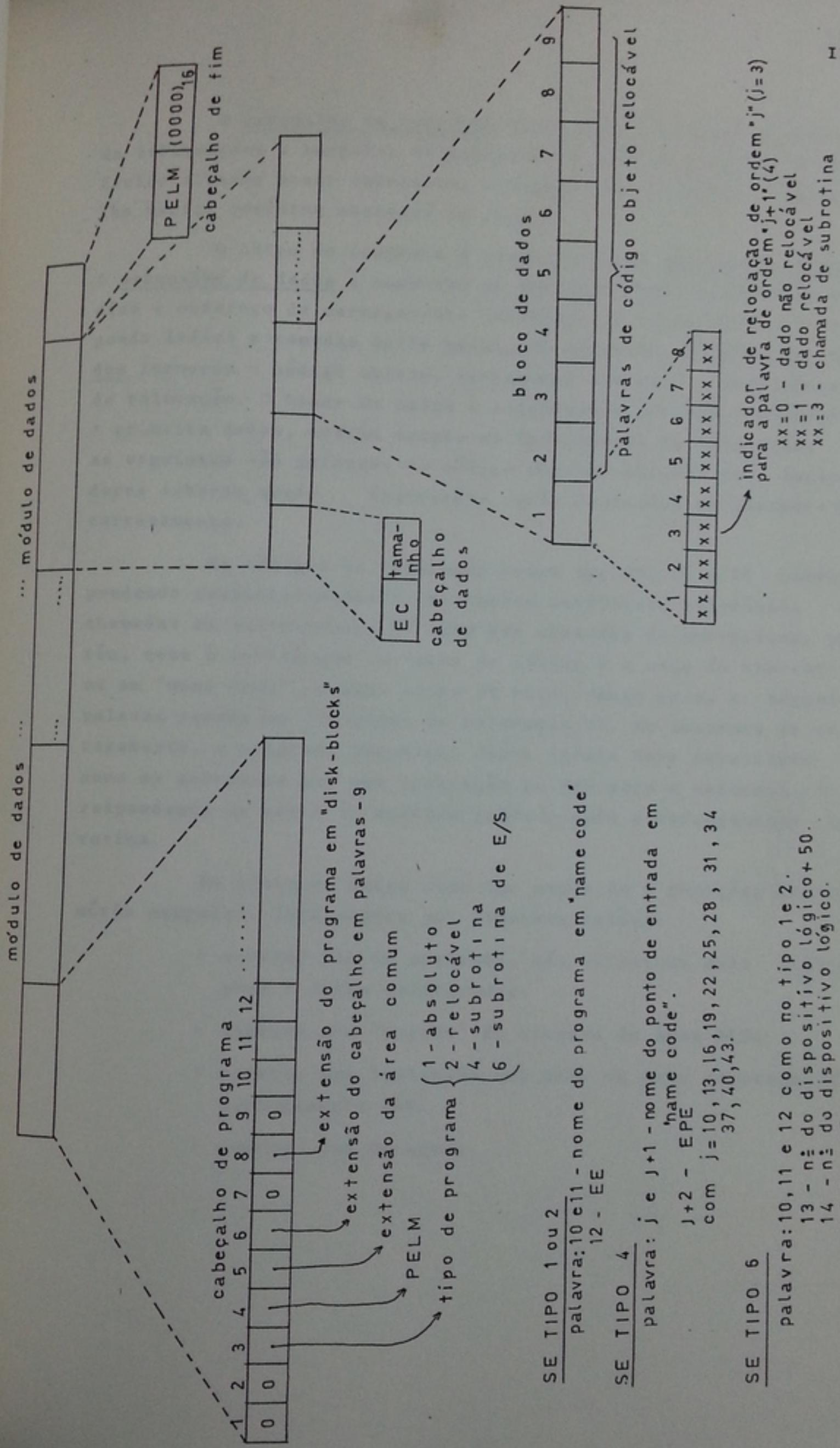


Figura 3.10 - Saída de Disco

O cabeçalho de programa serve para fornecer uma série de informações a respeito do programa. O tipo de programa é definido através deste cabeçalho, e dependendo do tipo o cabeçalho varia, conforme mostrado na figura.

O corpo do programa é composto pelos módulos de dados. O cabeçalho de dados é composto de duas palavras: a primeira indica o endereço de carregamento (absoluto ou relativo), e a segunda indica o tamanho deste módulo em palavras. O bloco de dados fornecem o código objeto, juntamente com os indicadores de relocação. O bloco de dados é constituído por até 9 palavras. A primeira delas, contém sempre os indicadores de relocação, e as seguintes são palavras do código objeto. Através dos indicadores sabe-se quais instruções serão relocadas no instante do carregamento.

Os códigos de relocação podem ser 00, 01 e 11 correspondendo respectivamente a instruções absolutas, relocáveis e chamadas de sub-rotinas. No caso das chamadas de sub-rotina, porém, quem é codificado no lugar do código é o nome da sub-rotina em "name code", o qual ocupa 30 bits. Neste caso, a segunda palavra recebe um indicador de relocação 00. No instante de carregamento, o programa incumbido desta tarefa deve substituir o nome da subrotina por uma instrução de PUG para o endereço correspondente ao ponto de entrada pedido, após o carregamento da rotina.

Um bloco de dados pode ter menos de 9 posições de memória ocupadas. Isto ocorre por diversas razões:

- ocorreu fim de programa, não existindo mais instruções a serem codificadas.
- ocorreu uma "pseudo" de reserva de área BLOC.
- ocorreu uma instrução que pede um novo endereço de carregamento ORG.
- ocorreu fim de setor.

O cabeçalho de fim, indica ao programa "Cross-Carregador" que terminaram as instruções do programa, devendo ele providenciar o fim do carregamento. Este cabeçalho é constituído por duas palavras:

- próximo endereço livre da memória (endereço par).
- indicador de fim: conteúdo zero.

Algumas características especiais desta saída são exigidas a fim de satisfazer as exigências do "Cross-Carregador". As instruções do minicomputador ocupam uma palavra de 16 "bits", na saída de disco, se forem curtas, ou duas se forem longas. Nelas palavras, os oito "bits" menos significativos são preenchidos com zero e não tem significado para o "Cross-Carregador", exceto em um caso: o das instruções de endereçamento da memória. Neste caso, a instrução longa ocupa os dezesseis "bits" da palavra. A segunda palavra, neste caso, não tem significado, isto é, ela é um "dummy" apenas para manter a uniformidade.

P A R T E IV

PROGRAMA CARREGADOR E SISTEMA DE BIBLIOTECAS

IV.1. - Estrutura do programa "Cross-Carregador"	IV.1
IV.1.a. - Estrutura da tabela do "Cross-Carregador"	IV.4
IV.1.b. - Esquema do "Cross-Carregador"	IV.5
IV.1.c. - Saídas do "Cross-Carregador"	IV.8
IV.2. - Estrutura do sistema de bibliotecas	IV.9
IV.2.a. - Organização da diretoria	IV.11
IV.2.b. - Organização do índice	IV.12
IV.2.c. - Esquema do programa de manutenção da biblioteca	IV.13
IV.3. - Organização da Memória	IV.13

As entradas ao sistema são feitas por meio de cartões perfurados, e saídas do "Cross-Carregador" são feitas através de cartões perfurados, os quais são processados pelo computador IBM 3116-B. O programa que é executado, escritos para este computador, podem ser alterados pelo usuário.

A figura IV.1, mostra a relação entre a biblioteca, o "Cross-Carregador" e o programa "Cross-Indexador".

IV.1. - RELAÇÃO ENTRE A BIBLIOTECA

O "Cross-Carregador", realiza a leitura das informações de "Cross-Indexador", apresentadas em forma de cartões perfurados, e as transmite ao "Cross-Carregador".

As funções exercidas pelo "Cross-Carregador" são:

• Execução de procedimentos de tratamento de dados de diferentes tipos;

P A R T E IV

PROGRAMA "CROSS-CARREGADOR" E SISTEMA DE BIBLIOTECA

Nesta parte serão descritos o programa "Cross-Carregador" e a Biblioteca para o Sistema "Cross-Montador"/"Cross-Carregador". Conforme foi mencionado anteriormente, estes programas se tornam necessários para produzir as saídas finais para o mini-computador.

A finalidade da Biblioteca é armazenar módulos de programas, compilados separadamente por intermédio do "Cross-Montador". É função desta Biblioteca armazenar também, programas já prontos, resultantes do processamento do Programa "Cross-Carregador".

Como o Computador IBM-1130 utilizado não possui dispositivos de fita perfurada acoplados, a saída do "Cross-Carregador" deve ser feita através de cartões perfurados, os quais são processados pelo computador HP-2116-B. Um programa que lê cartões, escrito para este computador, produz as saídas exigidas pelo mini-computador.

A figura IV.1. mostra a relação entre a Biblioteca, o "Cross-Carregador" e o programa "Cross-Montador".

IV.1. - Estrutura do Programa Carregador

O "Cross-Carregador", aceita, como entradas, as saídas do "Cross-Montador", apresentada no ítem III.10 da parte anterior.

As funções executadas pelo "Cross-Carregador" são:

- resolver os problemas de referência entre os diversos módulo ("ligação").

- ajustar os endereços de todas as instruções não absolutas dos programas ("relocação").
- gerar na memória do computador uma "imagem" do que seria a memória do minicomputador depois de montado o programa.

Esta "imagem" constitui a saída do "Cross-Carregador". A atitude tomada em seguida depende do programador. Este pode especificar a saída por cartões, ou pode pedir que o programa seja guardado em uma biblioteca.

O "Cross-Carregador" é do tipo "Carregador-Ligador", pois uma de suas funções é completar as chamadas de sub-rotinas, que foram deixadas incompletas pelo "Cross-Montador".

Estas chamadas de sub-rotinas são transformadas em "desvios de sub-rotina", diretos para o endereço especificado na instrução, isto PUG END (ver instrução PUG no apêndice A e B).

A razão para a escolha deste tipo de procedimento é devido a que, na ocasião em que o "Cross-Carregador" foi escrito, o modo de endereçamento indireto não havia ainda sido implementado no minicomputador. Esta implementação constitui etapa posterior no projeto .

Nesta filosofia, dois procedimentos podem ser escolhidos:

- gerar uma tabela de pedidos de sub-rotinas, onde se rião gerados pulos incondicionais para a sub-rotina; e gerar o PUG no programa para esta posição da tabela.
- gerar um PUG no programa, diretamente para a posição da sub-rotina correspondente ao ponto de entrada.

O primeiro procedimento tem a desvantagem de que a sub-rotina deve conhecer a posição da tabela onde se encontra o Pulo incondicional para o seu ponto de entrada, o que complica o retorno de sub-rotina e a transformação de parâmetros.

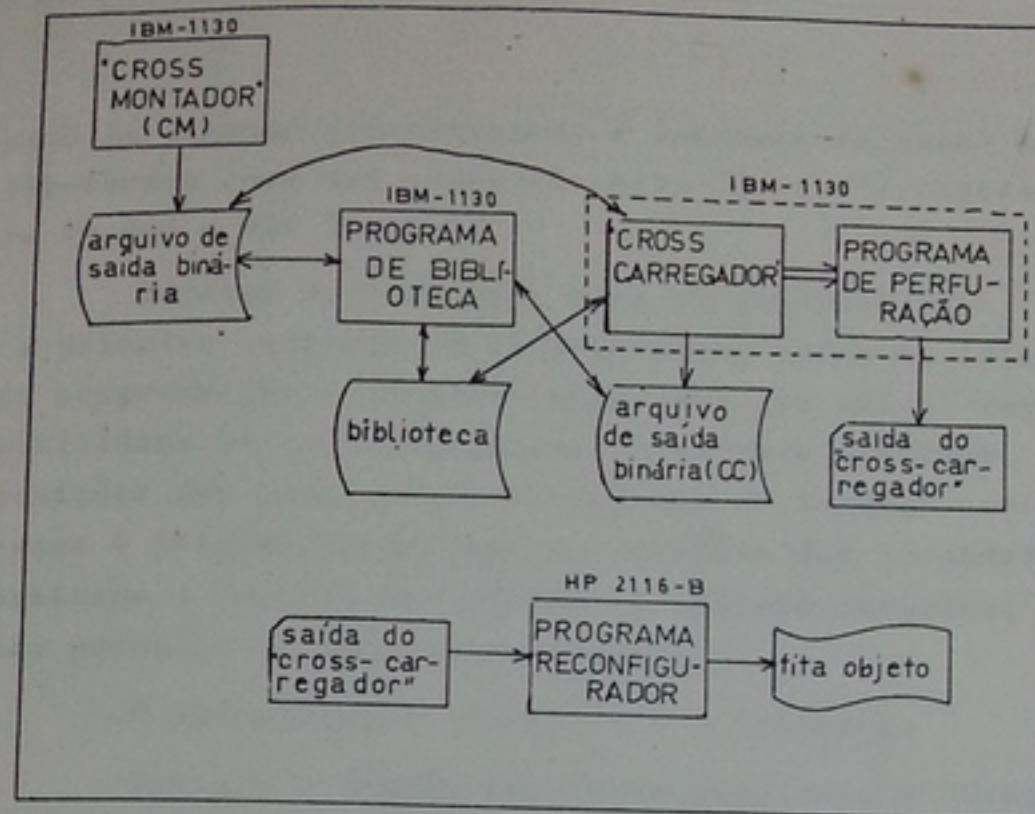


Figura IV.1. - Relação entre a Biblioteca o "Cross-Carregador" e o "Cross-Montador"

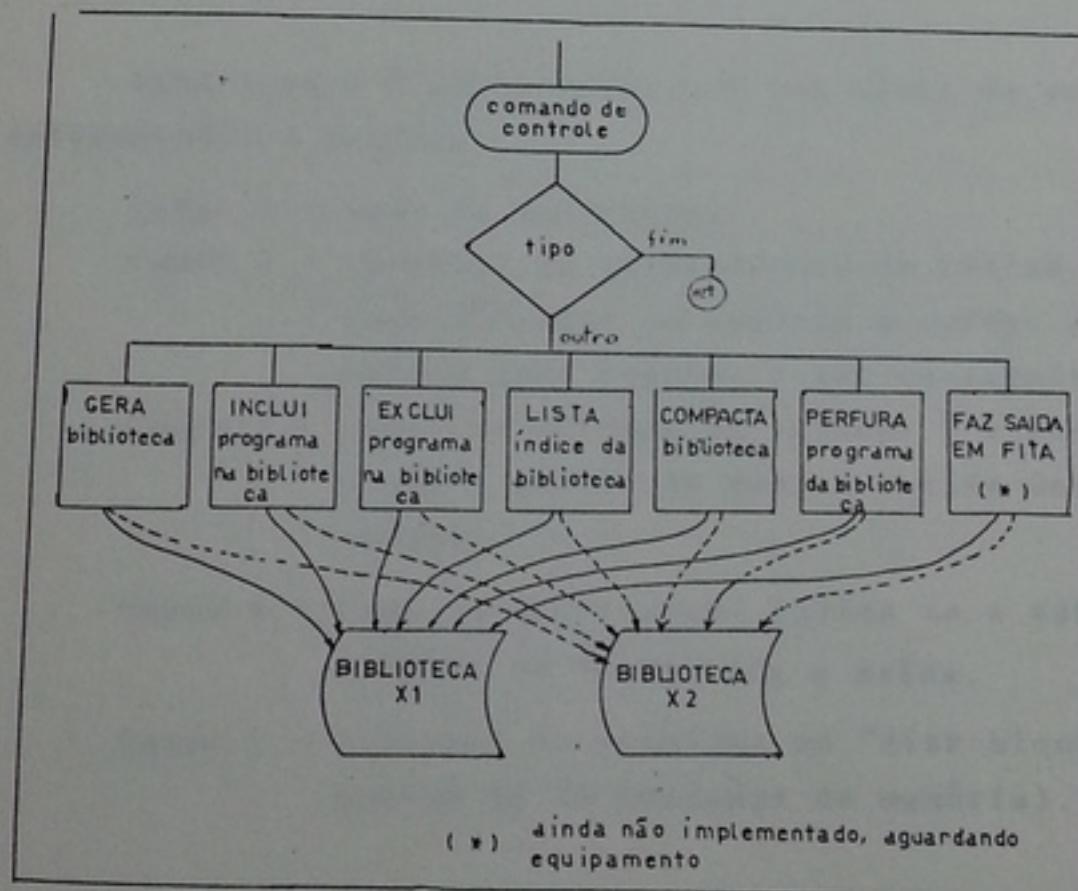


Figura IV.2.c. - Esquema do programa de manutenção da Biblioteca

No caso do segundo procedimento, o endereço do ponto de entrada da sub-rotina deve ser conhecido pelo "Cross-Carregador", no instante em que o PUG é gerado.

Existem dois métodos para se realizar tal procedimento: o primeiro consiste em se determinar antes de iniciar a carga do programa quais as sub-rotinas que ele exige, verificar a possibilidade de seu carregamento e em caso positivo, determinar as posições dos seus pontos de entrada. O segundo consiste em carregar o programa guardando as posições das chamadas. Uma vez determinada a posição dos pontos de entrada chamados, sobrepor nestas posições os endereços corretos.

O procedimento adotado foi o primeiro.

Durante a composição de um programa, o "Cross-Carregador" monta uma tabela, onde coloca os nomes das sub-rotinas pedidas, e suas características. Esta tabela é montada a partir da biblioteca, onde devem estar estas rotinas.

IV.1.a. - Estrutura_da_tabela_do_Carregador

Esta tabela é constituída por uma série de campos que são apresentados a seguir:

Campo 1 - nome da sub-rotina

Campo 2 - endereço de carregamento da rotina. É o endereço absoluto de memória a partir do qual a rotina deve começar a ser carregada.

Campo 3 - ponto de entrada da rotina. É o ponto de execução a partir do qual a rotina deve ser executada.

Campo 4 - tipo de sub-rotina. Indica se a sub-rotina é simples ou de entrada e saída.

Campo 5 - extensão do programa em "disk-blocks" (conjuntos de 20 posições de memória).

Campo 6 - próximo endereço livre na memória (primeiro endereço par da memória, após o programa).

Campo 7 - respectivamente número do arquivo em disco e 8 onde se encontra o programa, e endereço deste programa neste arquivo.

Campo 9 - indicador de carregamento -indica se este programa deve ser carregado ou não. A necessidade de inclusão deste indicador é devida a existência de rotinas com mais de um ponto de entrada. Quando estas rotinas são carregadas, todos os seus pontos de entrada vão para a tabela, mas a carga da rotina é feita uma única vez. Nesse caso, um dos pontos de entrada recebe indicador para carga, e as demais tem a carga suprimida.

IV.1.b. - Esguema do "Cross-Carregador"

A figura IV.2., apresenta o esquema do programa Carregador.

O programa principal, pode ser um programa absoluto ou relocável em formato relocável. As rotinas chamadas podem ser simples ou de entrada e saída também em formato relocável.

O programa principal, pode ser carregado a partir da Biblioteca ou a partir do arquivo de saída binária do "Cross - Montador". O programador deve fornecer esta informação ao programa "Cross-Carregador".

As rotinas são carregadas sempre a partir da Biblioteca.

No caso de serem utilizadas rotinas de entrada e saída, uma rotina tratadora de interrupção deve ser especificada pelo programador.

Uma vez lido o programa, determina-se o seu tipo, o seu endereço de execução e de carregamento. O nome do programa é inserido na tabela e definido.

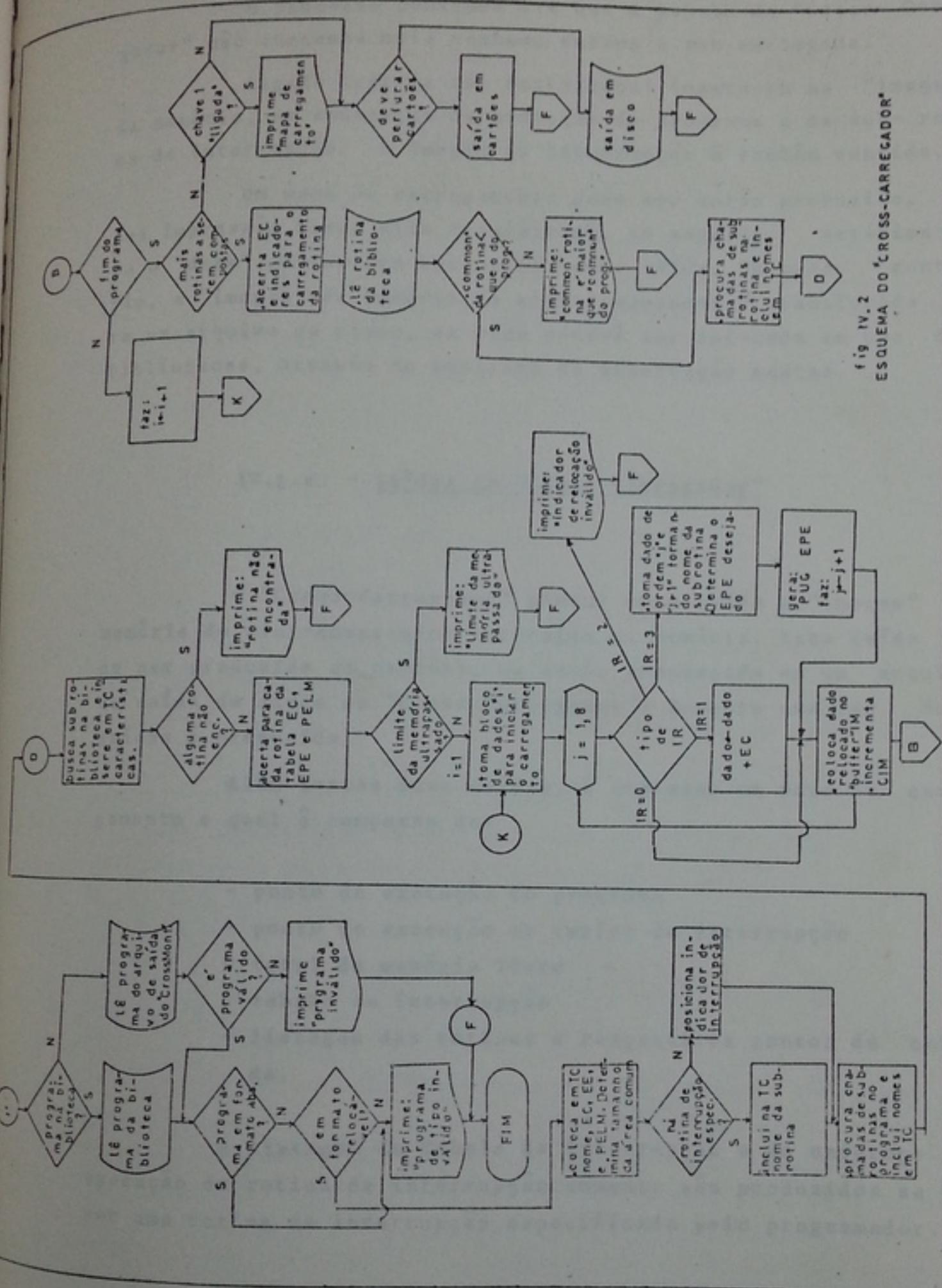
Verifica-se em seguida se o programador especificou o uso de alguma rotina de interrupção, e posiciona-se um indicador de acordo com esta informação. Se a rotina tiver sido especificada, o seu nome é inserido na tabela, e ela é procurada na Biblioteca.

Em seguida procuram-se chamadas de sub-rotinas no programa principal. No caso destas ocorrerem, os seus nomes são inseridos na Tabela do "Cross-Carregador". Terminada esta busca, procuram-se na Biblioteca as rotinas cujos nomes se encontram na Tabela sem indicação de definição. Se alguma delas não for encontrada o carregamento é encerrado e uma mensagem de erro é impressa. Se todas forem encontradas, elas são então definidas na Tabela do "Cross-Carregador". Em seguida, os endereços de carregamento e execução são devidamente acertados, bem como o próximo endereço livre da memória.

Algumas verificações são efetuadas para determinar a validade do carregamento: verifica-se se os nomes pedidos realmente são nomes de sub-rotinas e se o próximo endereço livre da memória não ultrapassa o limite desta. No caso de uma sub-rotina de entrada e saída, verifica-se se existe uma sub-rotina de interrupção especificada para o dispositivo pedido pela rotina. Para isto, consulta-se a Tabela de Interrupção.

Uma vez efetuadas estas verificações, inicia-se a composição do programa. Para isso utilizam-se as informações de relocação armazenadas juntamente com o código. Os endereços relokáveis são acertados, e são gerados os desvios para sub-rotinas quando é encontrado o indicador de relocação específico do caso. O código é disposto em posições de memória consecutivas correspondentes à "imagem" da memória do mini-computador.

Terminada a montagem do programa principal, inicia-se o carregamento das sub-rotinas. Este processo de carregamento é semelhante ao descrito para programa principal.



O processo continua até que a tabela do "Cross-Carregador" não contenha mais nenhuma rotina a ser carregada.

Alguns acertos são realizados: insere-se na "imagem" da memória, os endereços de execução do programa e da sub-rotina de interrupção. A Tabela de Interrupção é também suprida.

Um mapa de carregamento pode ser então produzido, se sua impressão tiver sido requisitada. Em seguida, verifica-se se o "deck" de cartões foi pedido como saída. Em caso contrário, a "imagem" da memória do mini-computador é transferida para um arquivo de disco, de onde poderá ser colocada em uma das bibliotecas, através do programa de manutenção destas.

IV.1.c. - Saídas do "Cross-Carregador"

O "Cross-Carregador" produz como saída a "imagem" de memória do mini-computador produzida na memória. Esta saída pode ser produzida em cartões, ou então armazenada em um arquivo de saída de disco do "Cross-Carregador". Somente uma das duas saídas é produzida.

Além destas duas saídas, é impresso um mapa de carregamento o qual é composto de:

- ponto de execução do programa
- ponto de execução da rotina de interrupção
- total de memória livre
- tabela de interrupção
- listagem das rotinas e respectivos pontos de entrada.

A listagem da Tabela de Interrupção e do endereço de execução da rotina de interrupção somente são produzidos se houver uma rotina de interrupção especificada pelo programador.

A saída em cartões consta de um cabeçalho e dos blocos de dados do programa.

O cabeçalho tem a seguinte configuração:

- número de palavras - 1
- desvio para rotina tratadora de interrupção
- desvio para endereço de execução do programa
- indicador de rotina de entrada e saída em andamento
- "checksum"

O corpo de cada bloco de dados tem a seguinte configuração:

- número de palavras - 1
- 128 "bytes" de instruções
- "checksum"

A figura C2(Ap. C) mostra um programa já composto pelo "Cross-Carregador" e a saída de cartões produzida.

A saída de disco é produzida no caso da saída em cartões não ser requisitada. Esta é composta de um cabeçalho, cuja configuração é mostrada na figura por um conjunto de "disk blocks" que contém a imagem do programa como teria sido armazenado na memória do minicomputador.

IV.2. - Estrutura do Sistema de Bibliotecas

O sistema de Biblioteca tem como função armazenar programas e sub-rotinas produzidos pelo "Cross-Montador" e pelo "Cross-Carregador", ou seja, programas em modo relocável e em modo absoluto, respectivamente. As principais funções executadas pelo Programa de Manutenção da Biblioteca são:

- incluir programas na biblioteca,
- excluir programas da biblioteca,
- compactar a biblioteca, quando ocorrem exclusões,
- listar os programas armazenados,
- fornecer informação sobre os programas armazenados na biblioteca
- fazer a geração de novas bibliotecas

Além de programas incumbidos de desempenharem estas funções, programas para realizarem funções auxiliares são utilizados: para buscas de programas na biblioteca, para testar os módulos que se encontram nos arquivos do "Cross-Carregador" e do "Cross-Montador", para manutenção do índice da biblioteca.

A biblioteca, nada mais é que um conjunto de programas, os quais são utilizados pelo "Cross-Carregador", na montagem de um programa relocável. A biblioteca exerce também a função de armazenar módulos já montados, para posterior "dump" em cartões, e produção da fita binária.

Ela é constituída logicamente por dois arquivos em disco. Fisicamente pode-se ter uma série de arquivos, os quais podem ser usados por uma série de usuários diferentes.

A razão para se permitir isso, é que muitos usuários desejam ter suas bibliotecas particulares, e usar estas bibliotecas na montagem de seus programas. Assim, a escolha da ordem de busca, bem como as bibliotecas utilizadas nestas buscas são escolhidas pelo usuário. Isto é permitido através de uma hierarquia entre os arquivos da biblioteca. A hierarquia é estabelecida através dos números que o programa atribui aos arquivos. A ligação entre estes números e os nomes das bibliotecas é estabelecida pelos cartões de controle FILES (ver Apêndice C).

O "Cross-Carregador", por opção do usuário pode restringir a busca a apenas uma das bibliotecas.

A pesquisa de um programa na biblioteca é feita, através do índice; quando o programa é encontrado, este é copiado pelo "Cross-Carregador" da mesma maneira, como se a cópia se fizesse do arquivo do "Cross-Montador", isto é, o formato do programa neste arquivo é o mesmo da saída daquele programa.

Quando o programa não se encontra em nenhum dos arquivos, e foi requisitado pelo "Cross-Carregador", então tem-se uma condição de erro.

A Biblioteca, isto é, cada um de seu arquivos tem a seguinte configuração: diretoria, Índice e área de armazenamento de programas.

A Biblioteca se acha dividida em blocos de 20 palavras cada um, denominados "disk blocks". Conjuntos de 16 "disk blocks" são denominados "setores". Esta organização é relacionada com a organização do disco do sistema IBM 1130.

IV.2.a. - Organização_da_diretoria

A finalidade da diretoria é armazenar uma série de informações sobre o estado da biblioteca, com as quais os programas de manutenção controlam a inclusão, exclusão e manutenção em geral dos programas. A organização da diretoria é a que se segue:

- 1a. palavra - número total de "disk - blocks"
- 2a. palavra - número de "disk - blocks" ocupados por programas e índices.
- 3a. palavra - número de "disk - blocks" livres
- 4a. palavra - primeiro "disk - blocks" disponível para o armazenamento
- 5a. palavra - número de programas armazenados na biblioteca
- 6a. 7a. e 8a. palavras - nome da biblioteca
- 9a. palavra - número de setores utilizados no índice
- 10a. palavra - primeira posição livre no índice

IV.2.b. - Organização do Índice

A finalidade do índice é guardar os nomes dos programas que se encontram na biblioteca, bem como uma série de informações a respeito destes programas, requisitados pelo "Cross - Carregador". O índice é formado por conjuntos de oito palavras cada grupo destes, guardando informação a respeito de um programa. A configuração destes registros é a seguinte:

1a. e 2a. palavras - nome do programa em "name - code"

3a. palavra - formato e tipo de programa. Programas absolutos tem o tipo armazenado na forma de número positivo e programas relocáveis tem o tipo armazenado na forma negativa. Os tipos podem ser: 1, 2, 4, ou 6. No caso de programas tipo 6, a configuração especial abaixo é adotada:

0	DISP	Ø	TIPO
---	------	---	------

formato relocável

4a. palavra - comprimento do programa (em "disk - blocks").

5a. palavra - endereço na biblioteca do programa (em "disk - blocks").

6a. palavra - ponto de entrada da rotina, ou endereço de execução do programa

7a. palavra - endereço de carregamento do programa

8a. palavra - próximo endereço livre após o programa.

IV.2.c. - Esquema do programa de manutenção da biblioteca

Um esquema do Programa de Manutenção da Biblioteca é mostrado na figura IV.2.c. Este é composto por uma série de pequenos programas auxiliares, e pelos programas que realizam as funções principais da biblioteca: inclusão, exclusão, compactação, listagem, pesquisa. Um programa para o "dump" de 'decks' de cartões de programas já montados é incluído também neste conjunto.

Existe um programa para fazer a geração da biblioteca. A função deste programa, é redefinir uma biblioteca, com um novo nome, ou gerar uma nova, em um arquivo recém gerado.

IV.3. - Organização da memória

A organização da memória escolhida é a mesma utilizada para o Programa Simulador. A organização escolhida é fundamental para a esquematização do "Cross-Carregador".

Hex	Dec	Descrição (sempre programado)
000	0	Indexador
001	1	Extensão do Acumulador
002	2	Reservado para interrupção
003	3	Reservado para interrupção
004	4	Desvio para rotina tratadora de interrupção
005	5	Desvio para rotina tratadora de interrupção
006	6	Desvio para endereço de execução do programa
007	7	Desvio para endereço de execução do programa
008	8	Zero de não há rotina de interrupção em andamento
009	9	Reservado

00A	10	Primeira palavra do programa
:		

XXX	X	Fim do programa
-----	---	-----------------

YYY	Y	Primeira palavra da sub-rotina de interrupção
:		

ZZZ	Z	Fim da sub-rotina de interrupção
-----	---	----------------------------------

WWW	W	
:		

		Sub-rotinas
--	--	-------------

KKK	K	
-----	---	--

VVV	V	
-----	---	--

:		Memória livre
---	--	---------------

LLL	L	
-----	---	--

CCC	C	
-----	---	--

		Primeira posição da área de Common
--	--	------------------------------------

FAF	4015	
-----	------	--

		Última posição da área de Common
--	--	----------------------------------

FBO	4016	
-----	------	--

:		
---	--	--

		Área de trabalho para o Bootstrap
--	--	-----------------------------------

FBF	4031	
-----	------	--

FCO	4032	
-----	------	--

:		
---	--	--

		Bootstrap (memória protegida)
--	--	-------------------------------

FFF	4095	
-----	------	--

P A R T E V

ANÁLISE E CONCLUSÕES SOBRE O "CROSS-MONTADOR" DESENVOLVIDO

Do trabalho efetuado, algumas conclusões puderam ser visualizadas, seja quanto à metodologia adotada, seja pelos recursos necessários.

A utilização de uma linguagem de alto nível tipo FORTRAN na elaboração do "Cross-Montador" e do "Cross-Carregador", trouxe, como era inicialmente esperado, alguns problemas; estes dizem respeito principalmente à manipulação de cadeias de "bits" e compactação de caracteres. Para se resolver tal problema foi necessário a utilização da linguagem ASSEMBLER, pois esta linguagem foi a única indicada ao processo frente ao sistema computacional utilizado; o segundo grande problema diz respeito à impossibilidade do computador escolhido produzir fitas de papel como saída: o problema foi resolvido gerando-se um conjunto de cartões intermediários, o qual é lido por um outro computador. Este, através de um programa reconfigurador, gera as fitas necessárias.

O uso de programas separáveis mostrou ser esta técnica a mais indicada para aproveitamento de memória; pois uma boa parte desta pode assim ser reservada ao dimensionamento de tabelas e "buffers" de leitura e escrita. A Tabela de Símbolos definidos no programa residem na memória, evitando flutuações para disco. As pesquisas na Tabela de Símbolos foram eliminadas no segundo passo graças à utilização do Arquivo de Informação Suplementar. Por outro lado, o uso deste arquivo proibiu a utilização de organizações não lineares para a Tabela de Símbolos, mais precisamente, de organizações que impliquem em re-estruturação desta tabela.

Ainda, quanto ao tipo de programas a serem utilizados, a experiência demonstrou que a técnica a se empregar é a de construir programas o mais modulares possíveis; a utilização de rotinas específicas em cada caso é particularmente útil no caso de ser necessário modificar-se o programa. Esta recomendação, comum em qualquer desenvolvimento de "software", é particularmente importante em "Cross-Montador". Alterações desta natureza são frequentes no caso do projeto de uma arquitetura nova, onde inclusões ou exclusões de instruções se mostram interessantes em determinados casos, complementando ou reduzindo o conjunto de instruções do computador. No caso de tais alterações, a simples troca de uma rotina é mais prático e menos danoso que a revisão de todo um programa.

A importância do "timing" no projeto de "software" é muito grande. Assim, mesmo que seja necessário comprometer um pouco a eficiência do sistema, deve-se ter em vista tal vínculo. Um pequeno atraso em uma das linhas pode ser bastante prejudicial a todo o projeto. Tais fatores devem ser levados em conta principalmente sob um ponto de vista industrial, onde a sincronização "software" - "hardware" é fundamental podendo se transigir quando o interesse acadêmico entra em jogo, ou quando estes atrasos não impliquem no atraso de outros projetos. Com isto, não se pretende dizer que a eficiência só é importante no trabalho de pesquisa, mas sim que o "timing" é um vínculo bastante restrito, sendo preferível em muitos casos deixar a fase de aperfeiçoamento como uma segunda etapa no trabalho.

Do mesmo ângulo, alterações no projeto de "hardware", são também fatores que comprometem o andamento paralelo do projeto de "software", devendo ser levado em conta quando do estabelecimento do cronograma deste último. A utilização de linguagens para o desenvolvimento de sistemas, que nos últimos tempos vem se tornando mais difundidas como meio de geração de "software" de base, procuram justamente minimizar os problemas ocasionados por tais alterações. Entretanto, tais sistemas apresentam também suas limitações e estão ainda pouco difundidos constituindo mais linhas de pesquisa, que propriamente linhas industriais.

A importância do "Cross-Montador" e do "Cross-Carregador", é bastante grande também após a fase do projeto, já com o minicomputador em sua fase operacional. Então, em uso conjunto com o Simulador, torna-se um recurso valioso na implantação de programas para o minicomputador, permitindo que este trabalho seja feito em um computador que dispõe de recursos mais poderosos que os daquele; utilizando saídas em disco tanto para fase de montagem como para a de carregamento, pode-se executar em seguida o programa através do simulador, realizando a sua depuração. Uma vez o programa corrigido, livre de erros de programação ou lógica, ele pode ser processado pelo programa Reconfigurador e gerada a fita final para processamento no minicomputador.

É sob este aspecto que a característica de transponibilidade do sistema "Cross-Montador" - "Cross-Carregador" e do Simulador é importante. A utilização do FORTRAN tem sua grande importância neste ponto, visto ser esta a linguagem mais difundida, encontrando-se disponível para quase todos os computadores. É neste ponto que o uso do ASSEMBLER se torna não recomendável, visto a vinculação entre esta linguagem e a máquina à qual se destina. Quando o projeto leva em conta a conversão do sistema para outras máquinas, o uso do ASSEMBLER deve ser reduzido ao mínimo, de maneira a não comprometer esta conversão.

No caso em que se dispõe ou se pretende usar sistema com mais recursos, tais como linguagens mais poderosas tipo ALGOL ou PL-1, o FORTRAN deve ser considerado como segunda opção, uma vez que estas linguagens quase sempre dispensam o uso de ASSEMBLER. Por outro lado, a utilização de linguagens mais poderosas diminui o tempo dispendido em testes e correções, bem como os recursos humanos necessários.

Sobre o sistema "Cross-Montador"/"Cross-Carregador" / Biblioteca desenvolvida podemos concluir:

- é o mais completo possível, incluindo a existência de grande número de pseudo-instruções, que facilitam os trabalhos de programação para o desenvolvimento de "software" básico ou de aplicação. Além disso, o trabalho de documentação é facilitado, através da sistematização do mesmo. Deve ainda ser notado, que a Biblioteca permite o armazenamento de grande número de programas em formato relocável ou absoluto, liberando o programador da tarefa de ter de compilar os segmentos de programas todas as vezes em que estes se fazem necessários, fornecendo uma maneira eficiente para o armazenamento de sub-rotinas. O "Cross-Carregador" permite ao usuário fazer a ligação desta sub-rotinas com o programa sem que para isso ele necessite fazer ligações "a mão". Isto permite o estabelecimento de uma Biblioteca de rotinas de aplicações para o sistema. Por outro lado, o programador tem a opção de utilizar as suas próprias Bibliotecas, por ele mesmo configuradas. A extensão da Biblioteca do usuário está limitada: apenas pela dimensão da área livre de seus discos.

- é eficiente, uma vez que seu processamento é rápido. Por outro lado, estando os programas armazenados na Biblioteca em formato relocável, durante a fase de testes, um erro ocorrido em um dos segmentos do programa, não implica na necessidade de recompilar todo o sistema, mas somente o módulo que foi corrigido.

- poderia ser melhorado, pela adição de programas que permitissem a operação com fita perfurada diretamente, no caso de equipamentos IBM-1130 que dispusessem de tal recurso, dispensando desta maneira a fase de passagem por cartão perfurado, a fim de produzir a fita de papel no outro computador. Um outro recurso que poderia ser de grande utilidade seria a inclusão de facilidade para "debug" através de um programa. o que permitisse a modificação do código fonte sem a necessidade de se ler novamente o "deck" de cartões, isto é, a reestruturação de uma Biblioteca de módulos fonte.

- para muitos projetos de computador é poderoso e completo demais, podendo em tais casos utilizar-se uma versão mais simplificada, e portanto mais eficiente em tempo de operação. Existe ainda a possibilidade de nestes casos, e para a compilação de programas pequenos ser mais vantajoso o uso de uma versão reconfigurada, a qual produza todo o programa na memória, evitando o esquema de dois passos, e tornando o sistema mais rápido, devido ao fato de não se necessitar armazenar o programa em meio de armazenamento secundário.

A P É N D I C E A

CARACTERÍSTICAS FÍSICAS DO MINICOMPUTADOR - LINGUAGEM DE MÁQUINA

I. - Introdução	A.1
II. - Fluxo dos Dados, Registradores da Unidade Central, Sistema de Interrupção, Modos de Operação e de Endereçamento	A.1
II.1. Fluxo dos Dados	
II.1.a. - Memória principal e registradores da Memória	A.3
II.1.b. - Registrador Acumulador	A.4
II.1.c. - Registrador Contador de Instruções	A.4
II.1.d. - Registrador de Instruções	A.4
II.1.e. - Registrador de Chaves do Painel	A.5
II.1.f. - Registradores Transbordo e Vem-Um	A.5
II.1.g. - Unidade Aritmética	A.5
II.2. Registrador Indexador	A.6
II.3. Extensão do Acumulador	A.6
II.4. Sistema de Interrupção	A.6
II.5. Estados do Sistema	A.7
II.6. Modos de Endereçamento	A.7
II.7. Classificação quanto ao número de operandos	A.8
III. - Conjunto de Instruções	A.8
III.1. Formato das Instruções	A.8
III.2. Instruções Longas	A.9
III.2.a. - Grupo de endereçamento da memória	A.10
III.2.b. - Grupo de endereçamento imeditao	A.12
III.2.c. - Grupo de deslocamento e giros	A.14
III.2.d. - Grupo de entrada e saída	A.17
III.3. Instruções Curtas	A.19
III.3.a. - Micro-instruções de controle do Acumulador	A.20
III.3.b. - Micro-instruções de teste para os "flip-flops" T e V	A.22
III.3.c. - Micro-instruções especiais	A.23
IV. - Endereçamento Indireto	A.24

A P E N D I C E AI - Introdução

Neste apêndice serão apresentadas as características dos Sistemas ao qual se destinou o "Cross-Montador" construído. Como já foi visto na Parte I, este minicomputador foi projetado e construído pelo Laboratório de Sistemas Digitais, do Departamento de Engenharia de Eletricidade da Escola Politécnica da Universidade de São Paulo.

Não é objetivo, apresentar aqui características de implementação deste sistema, uma vez que elas constam de trabalho anterior, (F.4) mas apenas dar uma rápida visão de alguns pontos, os quais são de grande importância para o presente trabalho.

II - Fluxo dos dados, registradores de memória, sistema de interrupção, modos de operação e de endereçamento

II.1. - Fluxo dos dados: A figura A0 mostra um esquema simplificado do fluxo dos dados deste minicomputador, o qual evidencia os principais caminhos e registradores do processador central.

Vamos observar alguns pontos fundamentais deste fluxo.

Figura A0 - Fluxo dos Dados

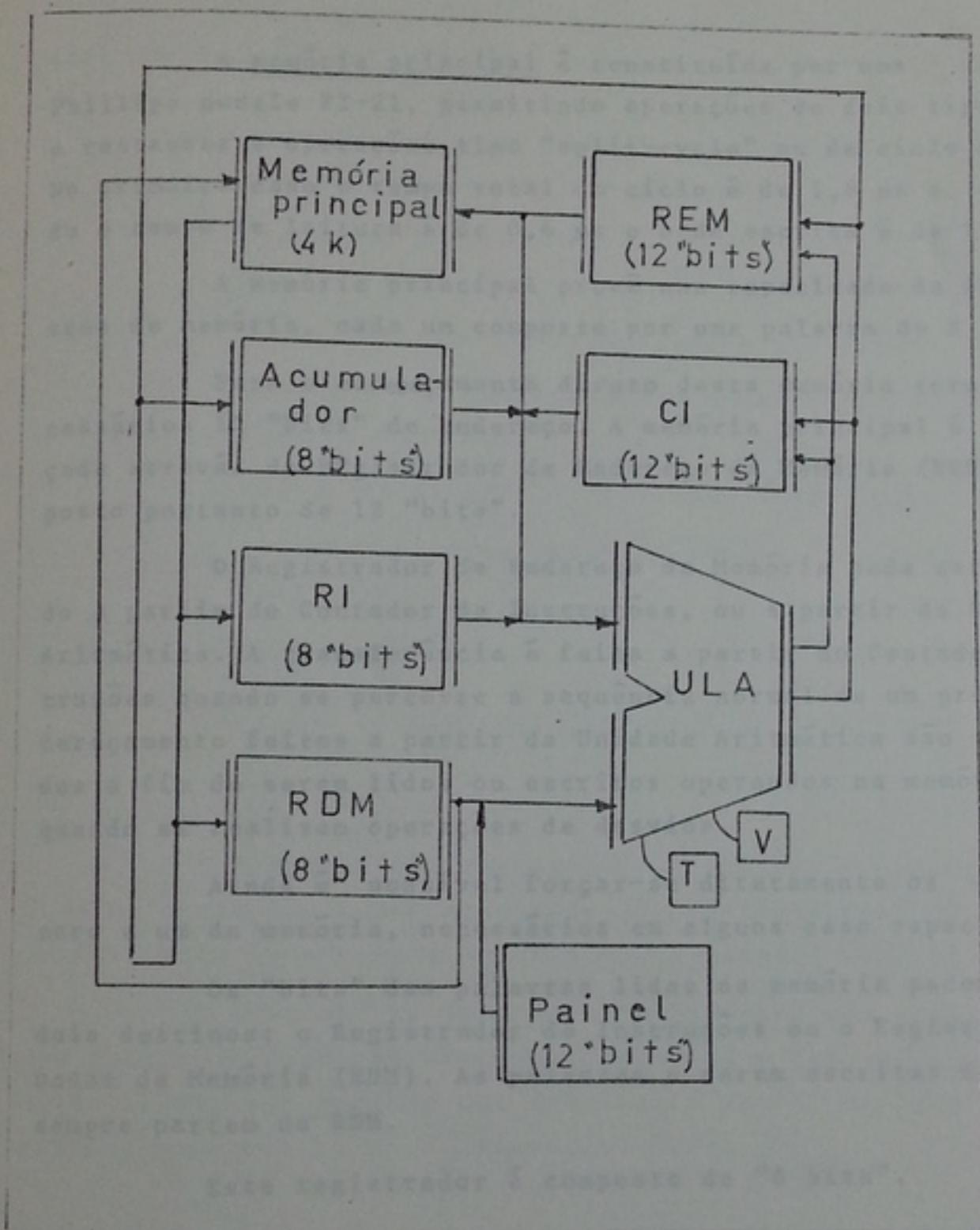


Figura A.0. - Fluxo dos Dados

II.1.a. - Memória Principal e registradores da memória

A memória principal é constituída por uma memória Phillips modelo FI-21, permitindo operações de dois tipos; ler e restaurar e operações tipo "split-cycle" ou de ciclo dividido. No primeiro caso o tempo total do ciclo é de 1,6 μ s e no segundo o tempo de leitura é de 0,6 μ s e o de escrita é de 1,0 μ s.

A memória principal provê uma capacidade de 4K regis tos de memória, cada um composto por uma palavra de 8 "bits".

Para o endereçamento direto desta memória tornam-se necessários 12 "bits" de endereço. A memória principal é endereçada através do Registrador de Endereço da Memória (REM), composto portanto de 12 "bits".

O Registrador de Endereço da Memória pode ser carregado a partir do Contador de Instruções, ou a partir da Unidade Aritmética. A transferência é feita a partir do Contador de Instruções quando se percorre a sequência normal de um programa. Endereçamento feitos a partir da Unidade Aritmética são realizados a fim de serem lidos ou escritos operandos na memória, ou quando se realizam operações de desvios.

Ainda é possível forçar-se diretamente os endereços zero e um da memória, necessários em alguns caso especiais.

Os "bits" das palavras lidas da memória podem ter dois destinos: o Registrador de Instruções ou o Registrador de Dados da Memória (RDM). As palavras a serem escritas na memória sempre partem do RDM.

Este registrador é composto de "8 bits".

II.1.b. - Registrador "Acumulador"

Este registrador é o núcleo da arquitetura do computador. Sua largura é de 12 "bits" e suas principais funções são:

- conter um dos operandos nas operações aritméticas e lógicas. O outro operando estará no RDM.
- armazenar o resultado de todas as operações realizadas.
- manusear dados entre a entrada e a saída.
- armazenar as quantidades a serem testadas nas operações de desvios.
- conter os dados e os resultados dos deslocamentos.
- carregar o registrador de índice como será visto mais adiante.

II.1.c. - Registrador Contador de Instruções (CI)

A função deste registrador é guardar o endereço da próxima instrução a ser executada. O seu conteúdo é transferido ao REM, para endereçar a memória e sua largura é de 12 "bits".

O Contador de Instruções é incrementado cada vez que uma instrução deve ser buscada na memória e tem seu conteúdo substituído, por ocasião dos desvios, a partir da Unidade Aritmética, conforme se vê no fluxo dos dados.

II.1.d. - Registrador de Instruções

O Registrador de Instruções (RI), é um registrador de 8 "bits", destinado a receber uma instrução. Estas podem ser longas ou curtas. No caso das instruções longas a primeira palavra é armazenada no Registrador de Instruções, ficando a segunda no RDM. A instrução é decodificada e analisada no RI, sendo então executada.

III.1.e. - Registrador_de_Chaves_do_Painel

Este registrador é mecânico, composto de chaves, e de tamanho 12 "bits". Através dele, o operador pode se comunicar com o computador durante o processamento, fornecendo dados para serem utilizados em operações, testes ou desvios.

III.1.f. - Registradores_Transbordo_(T)_e_"Vai-Um"(V)

São dois registradores de 1 "bit", ligados a Unidade Aritmética.

O "flip-flop" V, é o vai-um do somador e também funciona como o nono "bit" do Acumulador nas operações de deslocamento.

O "flip-flop" T serve para indicar quando as operações aritméticas produzem resultados grandes demais e que não cabem no Acumulador.

Estes dois registradores podem ser operados e testados por meio de instruções, como será visto na parte correspondente às instruções.

III.1.g. - Unidade_Aritmética

A Unidade Aritmética é constituída pelo somador - deslocador.

O somador é do tipo "desequilibrado", isto é, soma uma parcela de 8 "bits" a uma de 12 "bits". A necessidade de um somador deste tipo se deve ao fato de que ele é utilizado também no cálculo dos endereços efetivos, que são de 12 "bits", e não apenas para realizar operações sobre dados. As operações realizadas pela Unidade de Lógica e Aritmética são as operações de soma fundamental e as operações lógicas.

II.2. - Registrador Indexador

Algumas das instruções deste mini-computador são instruções indexadas. Para prover tal tipo de endereçamento o computador possui um registrador indexador, que se situa na posição 0 da memória principal. Circuitos especiais permitem, na fase de indexação que se force este endereço diretamente, tornando mais rápido o cálculo do endereço efetivo.

II.3. - Extensão do acumulador

A posição 1 da memória do minicomputador contém a extensão do registrador acumulador, situado no fluxo dos dados. A necessidade de tal extensão é para as instruções que usam dúpla precisão. Também nesse caso, ligações especiais permitem que se force diretamente este endereço, a fim de tornar mais rápido o seu acesso.

II.4. - Sistema de Interrupção

As interrupções possíveis no caso deste mini-computador, são do tipo entrada-saída, isto é, somente um nível de interrupção é permitido.

Quando chega um pedido de interrupção, o sistema coloca no Registrador de Instruções, uma instrução de desvio especial para a posição 2 da memória principal; nas posições 2 e 3 é guardado o valor do CI a fim de se poder retornar ao programa principal, uma vez atendida a interrupção.

Esta deve começar a ser tratada a partir da posição 4 da memória, isto é, este valor é carregado no Registrador CI.

A volta da rotina tratadora de interrupção se faz com um desvio para a posição 2 da memória, onde foi previamente armazenado o CI. Devido a características da instrução de desvio incondicional, a ser vista mais tarde, isto acarretará um novo desvio para a posição do programa a ser executada, no instante da ocorrência da interrupção.

O sistema de interrupção pode ser inibido por parte do programador por meio de instruções.

II.5. - Estados do sistema

Os estados possíveis do sistema, são três:

Parado ou de espera, quando o computador se encontra aguardando uma ordem para iniciar o seu funcionamento.

Normal, quando o computador se encontra executando normalmente suas instruções.

Interrompido, quando sofreu uma interrupção, a qual se encontra atendendo.

III.6. - Modos de endereçamento

Os seguintes modos de endereçamento são permitidos por este computador:

- a. Endereçamento imediato;
- b. Endereçamento direto;
- c. Endereçamento indireto;
- d. Endereçamento indexado;
- e. Endereçamento implícito;

II.7. - Classificação quanto ao número de operandos

É costume classificar-se os computadores quanto ao número de endereços que aparecem em suas instruções. De acordo com tal parâmetro estes podem ser máquinas de zero, um, dois ou três endereços.

Deste ponto de vista, o minicomputador pode ser classificado como uma máquina de um endereço.

Máquinas deste tipo apresentam na instrução o endereço explícito de um operando. Se como exemplo, tomarmos uma instrução de soma, o endereço efetivo calculado a partir deste endereço, indicaria a posição de memória onde se encontra a primeira parcela; a segunda parcela a se adicionar deve ser o conteúdo do acumulador. O resultado desta operação deve permanecer no acumulador.

III. - Conjunto de Instruções

Descritas as principais características físicas do minicomputador em apreço, cabe agora estudar-se o conjunto de instruções deste.

As instruções serão aqui agrupadas de acordo com sua classificação por grupos estruturais, isto é, segundo suas características de "hardware", e não sob o aspecto funcional.

III.1. - Formato das Instruções

O ponto de partida para a escolha do formato das instruções foi o tamanho máximo de um endereço efetivo. A capacidade máxima da memória de tal computador, conforme visto no item I, é de 4K palavras.

Para se endereçar de maneira direta todos esses registros de memória, são necessários 12 "bits".

Uma vez que o tamanho da palavra é de 8 "bits", é evidente que são necessárias pelo menos duas palavras para conter uma instrução de endereçamento direto.

Porém em muitos casos, duas palavras resultam em desperdício de "bits", pois as funções a serem implementadas são muito simples.

Para se contornar ambos os problemas, decidiu-se por dois tipos de instruções:

- longas, de duas palavras
- curtas, constituídas por uma única palavra

Estas instruções, por sua vez se subdividem em subgrupos como se verá a seguir.

III.2. - Instruções Longas

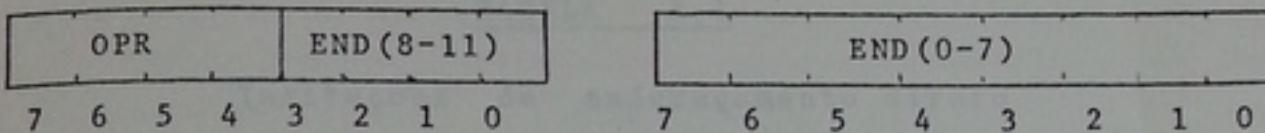
As instruções longas ocupam sempre duas posições sucessivas de memória, porém não tem todas a mesma configuração, podendo ser divididas, de acordo com seu formato, nos seguintes grupos:

- a) Grupo de endereçamento de memória
- b) Grupo de endereçamento imediato
- c) Grupo dos deslocamentos e giros
- d) Grupo de instruções de entrada e saída

Vejamos em detalhe cada um destes grupos.

III.2.a. - Grupo de endereçamento da memória

Compõe-se este grupo de 12 instruções de referência a memória, e cujo formato é o seguinte:



O significado de cada componente na figura é o seguinte:

OPR (Código de Operação): ocupa os "bits" 4 a 7 da primeira palavra, e especifica a operação a ser realizada. O OPR faz também a diferenciação entre as operações deste grupo, e os demais grupos.

END (Endereço explícito do operando): ocupa os "bits" 0 a 3 da primeira palavra, e os "bits" 0 a 7 da segunda. Temos dois tipos diferentes de endereçamento para as instruções deste grupo; direto e indexado. O tipo de endereçamento está implícito no código de operação da instrução. Nas instruções de endereçamento direto, o endereço explícito da instrução é o endereço efetivo, ou seja:

$$\text{Endereço efetivo} = \text{END}$$

Nas instruções indexadas, o endereço explícito é somado ao conteúdo do indexador, para se chegar ao endereço efetivo, ou seja:

$$\text{Endereço efetivo} = \text{END} + (\emptyset)$$

Estas instruções, além destes tipos apresentados de endereçamento, podem apresentar ainda endereçamento indireto e indireto indexado, conforme o tipo da instrução. Isto será visto adiante, num ítem reservado ao endereçamento indireto.

As duas tabelas A1 e A2 apresentam estas instruções, bem como uma breve descrição de cada uma delas.

TABELA A.1

Instruções de endereçamento direto

NOME	CÓDIGO	DESCRIÇÃO	MNEMÔNICO
Pulo incondicional	0000	CI \leftarrow End. Efetivo	PLA
Pula se Acumulador < zero	1010	se (ACC) $\geq 0 \Rightarrow$ CI \leftarrow (CI) + 1 se (ACC) < 0 \Rightarrow CI \leftarrow End. Efet.	PLAN
Pula se Acumulador = zero	1011	se (ACC) $\neq 0 \Rightarrow$ CI \leftarrow (CI) + 1 se (ACC) = 0 \Rightarrow CI \leftarrow End. Efet.	PLAZ
Carrega Acumulador	0100	ACC \leftarrow (End. Efetivo)	CAR
Armazena Acumulador	0010	End. Efetivo \leftarrow (ACC)	ARM
Soma com Acumulador	0110	ACC \leftarrow (End. Efetivo) + (ACC)	SOM
Subtrai um ou salta	1110	se (End. Efetivo) = 0 \Rightarrow CI \leftarrow CI + 2 se (End. Efetivo) $\neq 0 \Rightarrow$ (End. Efetivo) - 1	SUS
Pula e guarda CI	1111	End. Efet. \leftarrow 0000(CI[8-11]) End. Efet. \leftarrow (CI [0-7]) + 1 CI \leftarrow End. Efetivo + 2	PUG

TABELA A.2

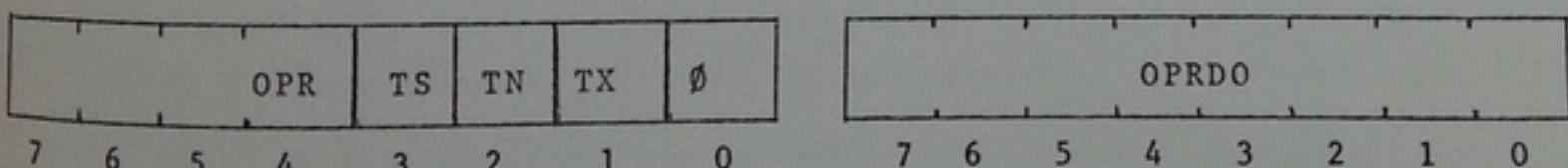
Instruções de endereçamento indexado

NOME	CÓDIGO	DESCRÍÇÃO	MNEMÔNICO
Pulo Incondicional	0001	CI ← End. Efetivo	PL AX
Soma com Acumulador	0111	ACC ← (End. Efetivo) + (ACC)	SOMX
Carrega Acumulador	0101	ACC ← (End. Efetivo)	CARX
Armazena Acumulador	0011	End. Efetivo ← (ACC)	ARMX

III.2.b. - Grupo de endereçamento imediato

Constituem este grupo as operações que tem operando de tipo imediato, isto é, o operando está contido na própria instrução, ocupando uma parte destinada, em outros casos, ao seu endereço.

O formato destas instruções é o seguinte:



O significado de cada ítem indicado na figura é o seguinte:

OPR (Código de operação): ocupa os "bits" 4 a 7 da primeira palavra, e indica que a instrução pertence ao grupo das imediatas, se o "bit" 0 da primeira palavra for zero. O valor do OPR deve ser "1101".

TS ("Tag" de soma): ocupa o "bit" da primeira palavra e indica, se for igual a 1, que a instrução é "Soma imediata". Isto obriga que os outros "tags", sejam iguais a zero.

TN ("Tag" de "Nand") : ocupa o "bit" 2 da primeira palavra, e indica, se igual a 1, que a instrução lógica do tipo "NAN" (não-e). Neste caso, todos os outros "tags" devem ser iguais a zero.

TX ("Tag de XOR"): ocupa o "bit" 1 da primeira palavra, e indica, quando igual a 1, que a instrução é uma operação lógica do tipo "XOR" (Ou exclusivo). Neste caso, todos os outros "tags" devem ser zero.

OPRDO (Operando Imediato): ocupa a segunda palavra da instrução, e constitui um dos operandos da operação a ser executada. O outro operando da instrução se encontra no Acumulador; neste também ficará armazenado o resultado da operação.

Quando TS e TX são feitos iguais a 1, a TN permanece zero, a instrução não pertence a nenhum dos tipos vistos, mas é um "Carrega Imediato no Acumulador". Esta instrução transfere o conteúdo da segunda palavra para o Acumulador.

A tabela A.3 apresenta as instruções com uma breve descrição de cada uma.

T A B E L A A.3

Instruções de endereçamento imediato

NOME	CÓDIGO					DESCRÍÇÃO	MNEMÔNICO
		TS	TX	TN			
Soma imediata	1101	1	0	0	0	ACC \leftarrow (ACC) + (End. Efet.)	SOMI
"NAND" imediato	1101	0	1	0	0	ACC \leftarrow ((ACC) AND (End. Efet.))'	NAND
"XOR" imediato	1101	0	0	1	0	ACC \leftarrow ((ACC) XOR (End. Efet.))	XOR
Carrega Acumulador imediato	1101	1	0	1	0	ACC \leftarrow (End. Efet.)	SOMI

Naturalmente nestas instruções o endereço efetivo é o conteúdo do Contador de Instruções.

$$\text{Endereço Efetivo} = (\text{CI})$$

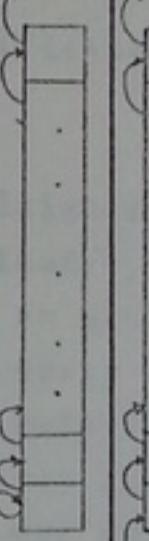
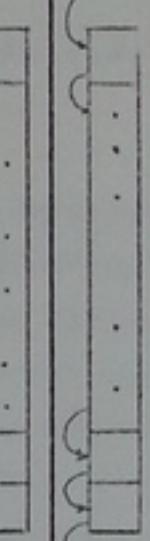
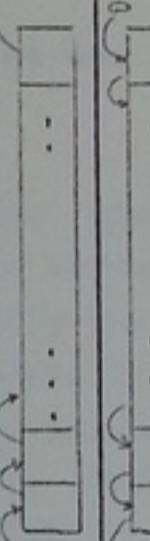
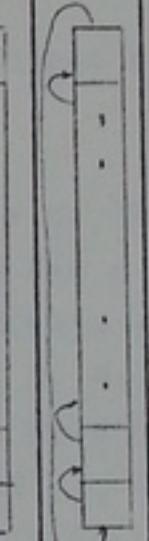
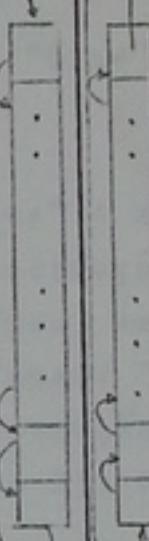
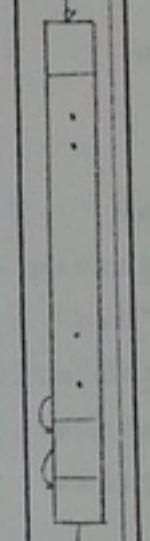
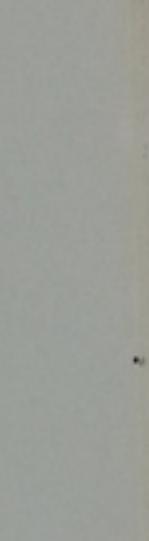
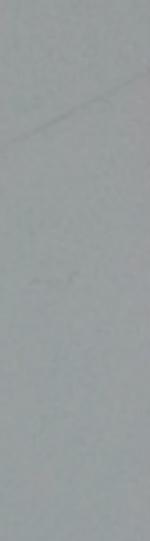
III.2.c. Grupo dos deslocamentos e giros

Constituem este grupo todas as instruções de deslocamentos e giros. Os deslocamentos e giros sempre se dão sobre o dado que se encontra no Acumulador. A manipulação do "flip-flop" V, o qual pode ser envolvido neste tipo de instrução, fica a cargo do programador, através de um subgrupo desta instruções.

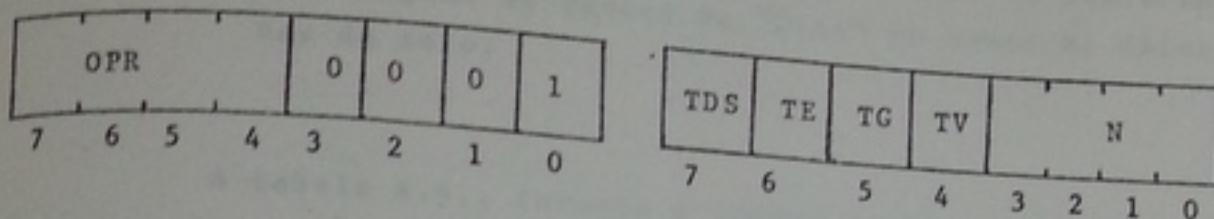
Nestes casos, ele recebe o "bit" que é deslocado para fora do Acumulador, podendo também o seu conteúdo ser intercalado no extremo oposto, no caso das instruções de giros.

DESLOCAMENTOS

E
GIROS

NOME	CÓDIGO	TDS	TE	TG	TV	DESCRIÇÃO	MNEMÔNICO
Deslocamento arit_mético a direita	1101 0001	1	0	0	0		DDS
Deslocamento sim_ples a direita	1101 0001	0	0	0	0		DD
Deslocamento sim_ples a esquerda	1101 0001	0	1	0	0		DE
Deslocamento a direita com uso V	1101 0001	0	0	1	0		DDV
Deslocamento a esquerda com uso V	1101 0001	0	1	0	1		DEV
Giro a direita	1101 0001	0	0	1	0		GD
Giro a esquerda	1101 0001	0	1	1	0		GE
Giro a direita com uso de V	1101 0001	0	0	1	1		GDV
Giro a esquerda com uso de V	1101 0001	0	1	1	1		GEV

O formato das instruções é o seguinte:



O significado de cada um dos grupos de "bits" é o seguinte:

OPR (Código de Operação): ocupa os "bits" 4 a 7 da primeira palavra, e indica que a instrução pertence ao grupo dos deslocamentos e giros, se o "bit" 0, desta mesma palavra for 1. Os demais "bits" desta palavra, não são usados pela instrução, sendo feitos iguais a zero ("bits" 1, 2 e 3). O valor do OPR deve ser: "1101".

TDS ("Tag" de deslocamento a direita com duplicação de sinal): ocupa o "bit" 7 da segunda palavra, e quando igual a 1, exige, que todos os demais "tags" sejam feitos zero, indicando neste caso, um deslocamento aritmético para a direita.

TE ("Tag" de sentido de deslocamento): ocupa o "bit" 6 da segunda palavra, e indica o sentido do deslocamento. Se este "bit" tiver o valor 1, o sentido do deslocamento é para a esquerda. Se tiver o valor 0, o sentido do deslocamento ou giro é para a direita.

TG ("Tag" dos tipos de deslocamentos): ocupa o "bit" 5 da segunda palavra, e especifica se o deslocamento é do tipo simples, ou em anel, (giro), isto é, o "bit" deslocado para fora é realimentado no outro lado do Acumulador. Quando o TG é igual a zero, tem-se deslocamentos simples. Quando TG é igual a um, o deslocamento é em anel.

TV ("Tag" de "flip-flop" V): ocupa o "bit" 4 da segunda palavra, e indica se for igual a um, que o conteúdo do "flip-flop" V, deve ser realimentado no extremo oposto do deslocamento.

N (Número de deslocamentos): ocupa os "bits" de 0 a 3 da segunda palavra. O número máximo de deslocamentos é igual ao número de "bits" no campo N, diferentes de zero.

A tabela A.4., fornece o código destas instruções, bem como uma descrição de cada uma.

III.2.d. - Grupo_de_entrada--saída

As instruções deste grupo são em número de quatro, porém podem ser desdobradas em uma série de outras.

O formato deste grupo é o seguinte:

OPR				N				TSD				TE				TS				TF				CMD							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

O significado de cada grupo de bits vai abaixo especificado:

OPR (Código de operação): ocupa os "bits" 4 a 7 da primeira palavra, indicando que no caso se trata de uma operação de Entrada ou Saída. Encontram-se neste grupo também as funções de Entrada e Saída e as operações de testes de dispositivos. O código do grupo é "1100".

N (Endereço do dispositivo): ocupa os "bits" 0 a 3 da primeira palavra. Estes "bits" identificam um dos dezesseis dispositivos possíveis, e que podem ser escolhidos pelo usuário.

TDS ("Tag" de operação de saída): ocupa o "bit" 7 da segunda palavra, e indica, quando seu valor for 1, que uma operação de saída de dados está sendo requisitada. Neste caso, os outros "tags" devem ter valor zero.

TE ("Tag" de operação de entrada de dados): ocupa o "bit" 6 da segunda palavra, e indica se for igual a 1, que uma operação de entrada de dados deve ser realizada.

Então os demais "tags" devem ser iguais a zero.

TS ("Tag" para instruções de testes): ocupa o "bit" 5 da segunda palavra, e indica que um determinado teste deve ser realizado, para o dispositivo. Se a condição testada for verdadeira, a instrução seguinte deverá ser pulada. Neste caso, todos os outros "tags" devem estar zerados.

TF ("Tag" para funções): ocupa o "bit" 4 da segunda palavra, e indica quando igual a 1, que uma determinada função de Entrada ou Saída deve ser realizada pelo dispositivo indicado. Quando este "tag" é feito igual a 1, os demais devem ser feitos iguais a zero.

CMD (Comando de Entrada ou Saída): ocupa os "bits" 0 a 3 da segunda palavra, e indica para as instruções de testes e funções, quais as condições, ou funções a serem testadas ou realizadas pelo dispositivo. São permitidas no máximo dezesseis funções ou condições para cada um deles. Tais funções são implementadas pelo "hardware".

A tabela A.5. indica respectivamente as instruções permitidas por este grupo.

T A B E L A A.5

Tabela de Instruções

NOME	CÓDIGO					DESCRIÇÃO	MNEMÔNICO
		TDS	TE	TS	TF		
Entrada de	1100	0	1	0	0	Entrada de dados	ENTR
Saída de da dos	1100	1	0	0	0	Saída de dados	SAI
Testes de dis positivo	1100	0	0	1	0	Testa condição CMD	SAL
Funções de dispositivo	1100	0	0	0	1	Executa função CMD	FNC

III.3. - Instruções Curtas

As instruções curtas ocupam uma única posição de memória. Tais instruções não manipulam endereços explícitos e são também chamadas "micro-instruções".

De acordo com seu formato podem ser classificadas em três grupos:

- a) - Micro-instruções de Controle do Acumulador
- b) - Micro-instruções de testes de estado dos "flip-flop" T e V
- c) - Micro-instruções especiais

III.3.a.- Micro-instruções de controle do Acumulador

Este grupo é constituído por oito instruções, as quais controlam o ciclo em que o conteúdo do Acumulador passa pelo Somador, sendo o resultado devolvido ao Acumulador.

O formato é o seguinte:

	OPR				TP	TA	TC	TV
7	6	5	4	3	2	1	0	

OPR (Código de Operação): ocupa os "bits" de 4 a 7 da primeira palavra, indicando a qual grupo pertence tal micro-instrução. O código de operação deste grupo é "1000".

TP ("Tag" de instrução de painel de chaves): ocupa o bit 3, e quando igual a 1, permite que os dados do painel de chaves, fluam através da entrada do somador. Neste caso, a combinação dos três outros "tags" da instrução especificam qual a função que deve ser realizada com esses dados.

TA ("Tag" de Acumulador): ocupa o "bit" 2 da instrução e quando o seu valor é 1, permite a entrada do valor contido no Acumulador para o Somador.

TC ("Tag" de Acumulador complementado): ocupa o "bit" 1, e se tiver valor 1, complementa-se o valor do Acumulador antes deste entrar para o Somador.

TV ("Tag" do "flip-flop" V): ocupa o "bit" 0 do Acumulador e quando posicionado em 1, faz com que o valor do "flip-flop" V, seja incluído nas operações do Somador.

Se todos os "tags" forem zero, o Acumulador é preenchido com zeros, bem como os "flip-flops" T e V. Se porém TV for igual a 1, este será o valor de V, após a execução desta instrução.

Todas as instruções limpam T, e com excessão da mencionada acima, elas fazem também V igual a zero.

A tabela A.6 fornece as instruções deste grupo:

T A B E L A A.6

NOME	CÓDIGO					DESCRIÇÃO	MNEMÔNICO
		TP	TA	TC	TV		
Limpa Acumula dor V = Ø	1000	0	0	0	0	ACC $\leftarrow \emptyset$ T $\leftarrow \emptyset$, V $\leftarrow \emptyset$	LIMP
Faz Acumula dor igual 1	1000	0	0	0	1	ACC $\leftarrow 1$ T $\leftarrow \emptyset$, V $\leftarrow \emptyset$	UM
Faz o comple mento de 1 do Acumulador	1000	0	0	1	0	ACC $\leftarrow (\text{ACC})'$ T $\leftarrow \emptyset$, V $\leftarrow \emptyset$	CMP1
Faz o comple mento de 2 do Acumulador	1000	0	0	1	1	ACC $\leftarrow (\text{ACC})' + 1$ T $\leftarrow \emptyset$, V $\leftarrow \emptyset$	CMP2
Faz V igual a zero	1000	0	1	0	0	V $\leftarrow \emptyset$	LIM
Incrementa Acumulador	1000	0	1	0	1	ACC $\leftarrow (\text{ACC}) + 1$	INC
Faz Acumula dor igual a -1	1000	0	1	1	0	ACC $\leftarrow -1$ T $\leftarrow 0$, V $\leftarrow \emptyset$	UNEG
Comando de painel	1000	1	X	X	X	executa comando de painel (XXX)	PNL
Limpa Acumula dor/V=1	1000	0	1	1	1	ACC $\leftarrow \emptyset$ T $\leftarrow 0$, V $\leftarrow 1$	LIMP

III.3.b. - Micro-instruções de teste para os "flip-flops"

Este grupo de instruções é constituído por aquelas que realizam testes sobre o estado deste "flip-flops", e dependendo da condição testada ser satisfeita, saltam a execução da instrução imediatamente seguinte.

O formato das instruções deste grupo está abaixo:

OPR				Ø	TF	TC	TM
7	6	5	4	3	2	1	0

OPR (Código de Operação): ocupa os "bits" 4 a 7 da instrução. As instruções deste grupo se diferenciam das do grupo "c", que será visto a seguir, pelo fato do "bit" 3 ser zero. O código de operação deste grupo é "1001".

TF ("Tag" de "flip-flop"): ocupa o "bit" 2 da palavra e indica que a instrução deverá realizar um teste do valor do "flip-flop" V se o seu valor for 1, ou do valor do "flip-flop" T, se seu valor for zero.

TM ("Tag" de mudança do valor do "flip-flop" testado): ocupa o "bit" Ø da instrução, e dependendo do seu valor, são disparados ou feitos zero os "flip-flops" após o teste. Estas mudanças no valor dos "flip-flops" só são verificadas, se o teste realizado for satisfeito. A mudança do valor é requerida para TM igual a 1.

TC ("Tag" de condição): ocupa o "bit" 2 da instrução, e dá a condição de comparação do "flip-flop", que vai ser testado.

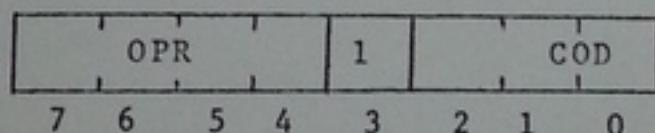
A tabela A.7., dá as instruções deste grupo:

T A B E L A A.7

NOME	CÓDIGO					OPERAÇÃO	MNEMÔNICO
		TF	TC	TM			
Testa T	1001	0	0	X	0	se (T)=X \Rightarrow CI \leftarrow (CI)+1	SLT
Testa V	1001	0	1	X	0	se (V)=X \Rightarrow CI \leftarrow (CI)+1	SLV
Testa T e complementa	1001	0	0	X	1	se (T)=X \Rightarrow CI \leftarrow (CI)+1 e T \leftarrow (T)	SLTM
Testa V e complementa	1001	0	1	X	1	se (V)=X \Rightarrow CI \leftarrow (CI)+1 e V \leftarrow (V)'	SLVM

III.3.c. Micro-instruções especiais

Este grupo é constituído por 7 instruções de uso geral, cujo formato é o seguinte:



O significado de cada um destes grupos de "bits" é o seguinte:

OPR (Código de Operação): ocupa os "bits" 4 a 7 da instrução. Este grupo se diferencia do grupo b, através do "bit" 3, que neste caso tem o valor 1.

COD (Código da micro-instrução): ocupa os "bits" 0 a 2 da instrução e caracteriza a instrução a ser executada.

A tabela A.8 fornece as instruções deste grupo:

T A B E L A A.8

NOME	CÓDIGO			DESCRÍÇÃO	MNMÔNICO
	COD				
Tratamento de interrupção	1001	1	000	(CI [8:11]) → 2 (CI [0: 7]) → 3 CI ← 3	PUL
Troca Acumulador e extensão	1001	1	001	ACC ↕ 1	TRE
Inibe estado de interrupção	1001	1	010	Inibe "flip-flop" de interrupção	INIB
Permite estado de interrupção	1001	1	011	Desinibe "flip-flop" de interrupção	PERM
Espera	1001	1	100	Coloca em estado de espera	ESP
Pare	1001	1	101	Coloca em estado de pare	PARE
Troca Acumulador e indexador	1001	1	110	ACC ↕ Ø	TRI

IV. - Endereçamento indireto

O endereçamento indireto, no minicomputador é conseguido através de uma instrução dupla. Esta instrução dupla, é montada da seguinte maneira:

- a primeira instrução é reconhecida pelo código $(9F)_{16}$
- a instrução seguinte é uma instrução de endereçamento da memória, e que pode ser qualquer uma deste grupo, e para a qual se deseja fazer o endereçamento indireto.

Quando a primeira das duas instruções é encontrada, busca-se a seguinte e teremos dois casos:

- a instrução seguinte é de endereçamento a memória, e então o endereçamento indireto é assumido. Este poderá ser de dois tipos, dependendo da instrução utilizada: indireto simples, e indexado indireto.
- a instrução seguinte não é de referência a memória e então assume-se a primeira (código 9F), como um simples NOP (instrução de "Não Operação").

A P É N D I C E B

DESCRIÇÃO DA LINGUAGEM DE MONTAGEM

B.1. - Codificação dos cartões de entrada	B.1
B.2. - Endereçamento dos Dados	B.3
B.3. - Caracteres permitidos	B.3
B.4. - Instruções executáveis da linguagem de Montagem	B.4
B.4.a. - Instruções utilizadas para carregar e armazenar	B.4
B.4.b. - Instruções de aritméticas e lógicas	B.9
B.4.c. - Instruções de entrada e saída	B.13
B.4.d. - Instruções de operação com o Acumulador	B.19
B.4.e. - Instruções de operação com o painel	B.24
B.4.f. - Instruções de testes dos "flip-flops" T e V	B.26
B.4.g. - Instruções de deslocamento de dados	B.29
B.4.h. - Instruções para mudança de curso do programa	
ma	B.34
B.4.i. - Instruções especiais	B.40
B.5. - Pseudo-instruções da linguagem de Montagem ou "pseudos"	B.45
B.5.a. - "Pseudos" de reserva de área de memória	B.45
B.5.a. - "Pseudos" de controle de paginação	B.50
B.5.c. - Definição de pontos de entrada de sub-rotinas e suas chamadas	B.53
B.5.d. - Comandos de origem e fim de programa	B.56
B.5.e. - Comandos de equivalência de endereços	B.57

APÊNDICE B

DESCRIÇÃO DA LINGUAGEM "ASSEMBLER"

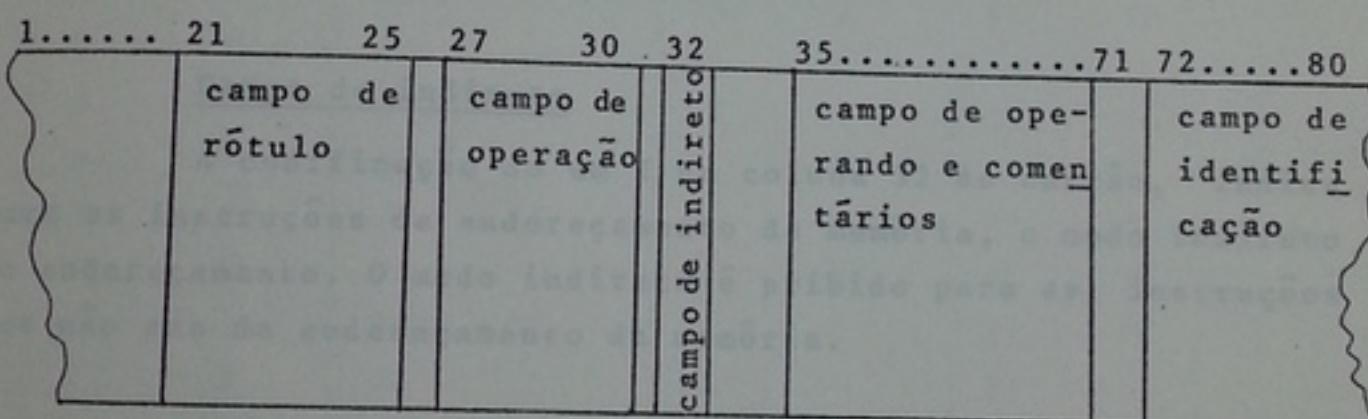
Esta parte visa fornecer ao usuário do minicomputador um manual de utilização da linguagem "Assembler" definida.

A organização desta parte foi feita colocando-se as instruções em grupos funcionais, de maneira a facilitar o trabalho de consulta.

B.1. - Codificação dos cartões de entrada

Os comandos são fornecidos ao computador através de cartões perfurados, codificados em formato fixo.

O cartão está subdividido em campos, conforme é mostrado na figura abaixo:



Uma explicação sobre estes campos já foi apresentada em parte anterior deste trabalho, porém um breve resumo é apresentado a seguir:

Campo de rótulo

Este campo é opcional, exceto para algumas poucas "pseudos": EQU e COM. Os rótulos quando existentes devem ser iniciados por um caracter alfabético, por \$, = , ou @, seguidos de letras ou números. O número máximo de caracteres permitidos em um rótulo é 5.

Campo de operação

Este campo é obrigatório, e especifica a instrução que está sendo codificada. Se nesse campo aparecer um código não reconhecido pela linguagem, assume-se em certos casos, como sendo o nome de uma rotina externa, e a instrução é encarada como CALL para o nome codificado.

Campo de operando e comentários

O operando pode ser constituído por uma expressão ou por indicadores, dependendo da instrução. Os comentários são separados dos operandos por pelo menos um branco. Os comentários podem conter qualquer dos caracteres permitidos pela linguagem.

Campo de indireto

A codificação de um I na coluna 32 do cartão, indica para as instruções de endereçamento da memória, o modo indireto de endereçamento. O modo indireto é proibido para as instruções que não são de endereçamento da memória.

Campo de identificação

São usados para se fazer a identificação do programa e numerar os cartões.

B.2. - Endereçamento dos dados

Para as instruções de endereçamento da memória são permitidos os seguintes tipos de endereçamento para a linguagem "Assembler" definida: absoluto, simbólico, imediato e relativo. Para os dois primeiros tipos e para o último permite-se ainda um endereçamento de tipo indireto e também indexado.

B.3. - Caracteres permitidos

Os caracteres permitidos pelo "Assembler" são os algarismos de 0 a 9, os caracteres alfabéticos de A a Z, e uma série de caracteres especiais. O conjunto de todos os caracteres permitidos está apresentado na tabela que se segue.

TABELA DE CARACTERES PERMITIDOS

1	.	17	V	33	5	49	>
2	G	18	W	34	6	50	\$
3	H	19	X	35	7	51	Ω
4	I	20	Y	36	8	52	-
5	J	21	Z	37	9	53	←
6	K	22	A	38	branco	54	
7	L	23	B	39	+	55	&
8	M	24	C	40	-	56	:
9	N	25	D	41	(57	;
10	O	26	E	42	,	58	¬
11	P	27	F	43)	59	'
12	Q	28	Ø	44	#	60	?
13	R	29	1	45	*	61	"
14	S	30	2	46	=	62	!
15	T	31	3	47	/		
16	U	32	4	48	<		

OBSERVAÇÃO: Deve-se considerar que alguns dos caracteres, tem internamente sua representação interna modificada em consequência da leitura feita em Fortran.

B.4. - Instruções executáveis da linguagem Assembler

Serão aqui apresentadas as "instruções propriamente ditas" da linguagem, isto é, aquelas, intimamente ligadas a operações de máquina do computador. Estas instruções serão apresentadas tendo em vista, a finalidade a que se destinam, isto é, sob o ponto de vista de programação.

b.4.a- Instruções utilizadas para carregar e armazenar informação.

Subdividem-se estas instruções em dois grandes grupos: as instruções de carregamento e as de armazenamento.

CAR - CARREGA DADO NO ACUMULADOR

CARX - CARREGA DADO NO ACUMULADOR (INDEXAÇÃO)

CARI - CARREGA DADO IMEDIATO NO ACUMULADOR

Uso: Estas instruções são instruções de carregamento. São utilizadas quando se deseja colocar um valor no Acumulador, a fim de se realizar alguma operação sobre este dado. A maior parte das instruções atuam sobre dados que se encontram no Acumulador, devendo pois este ser carregado antes de se realizar as operações desejadas.

A instrução CAR carrega um dado, que se encontra na posição indicada pelo endereço no operando, ou na posição indicada por esta posição (caso do endereçamento indireto), no Acumulador. O valor na posição de memória não é destruído.

Algumas vezes porém este tipo de operação não é suficiente aos propósitos que se deseja, como por exemplo, quando se deseja manusear uma tabela de dados. Em casos como este, um recurso de grande poder é a "indexação". A instrução CARX, faz endereçamento indexado através do registro de índice, somando o seu conteúdo ao endereço do operando, para determinar a posição de memória desejada. O endereçamento indireto, é permitido como no caso da instrução CAR.

A instrução CARI, carrega o conteúdo da segunda palavra da instrução no Acumulador, isto é, o operando da instrução. É muito valiosa principalmente nos casos em que se deseja carregar um valor bem determinado no Acumulador.

Formato:

...21	25	27	30	32	35
R O T U L	C A R			I	operando
	C A R				operando
	C A R X				operando
	C A R X			I	operando
	C A R I				operando
	C A R I				operando

No caso das instruções CAR e CARX os operandos podem ser endereços imbutidos, absolutos ou relativos, conforme será visto nos exemplos. O operando da instrução CARI, deve ser um número decimal ou hexadecimal entre 0 e 255, ou /00 e /FF.

Exemplos:

O exemplo de rótulo EX1, na figura B1, mostra uma instrução CAR com operando simbólico, onde o operando A, deve ser rótulo de outro comando do programa. A instrução na linha seguinte, referencia o mesmo rótulo, porém agora o endereço desejado não é mais o do símbolo A, mas sim o seu conteúdo. A instrução na linha 3, referencia um endereço absoluto, ou seja, esta instrução carrega o Acumulador com o conteúdo da posição 1000 da memória. A instrução na quarta linha também referencia uma posição fixa da memória, isto é a posição 256. A instrução na quinta linha referencia a posição situada duas além daquela endereçada pelo rótulo A. A instrução na sexta posição carrega no Acumulador o conteúdo da palavra situada duas posições além desta instrução.

CROSS-MONTADOR
Formulário para perfuração de cartões

FIGURA B1 - Instruções de carregamento

		Data										Seq. E.S.A.				
		10	11	12	21	30	31	40	41	50	51	60	61	70	71	80
EX1	CAR	CAR	I	A												
	CAR	CAR			1000											
	CAR	CAR			/100											
	CAR	CAR			A+2											
	CAR	CAR			*+2											
EX2	CARX	CARX	I	A												
	CARX	CARX			1000											
	CARX	CARX			/100											
	CARX	CARX			A+2											
	CARX	CARX			*+2											
EX3	CARI	CARI	I	A												
	CARI	CARI			10											
	CARI	CARI			/10											

O exemplo de rótulo EX2, da figura B.1, apresenta uma instrução CARX, com operando simbólico. Neste caso, o dado que se deseja carregar na memória, se encontra na posição cujo endereço está distante do endereço do rótulo A, do conteúdo do índice no instante em que a instrução é executada. Para as cinco outras instruções que se seguem vale a mesma observação quanto a soma do conteúdo do indexador aos endereços no operando, para os quais continuam válidas as observações do caso da instrução CAR.

O exemplo de rótulo EX3 na mesma figura, mostra o caso de uma instrução CARI, que carrega no Acumulador o valor 10. A instrução seguinte, carrega o Acumulador com o valor 16, e a terceira deve carregar o Acumulador com o valor que a segunda palavra desta instrução contiver no instante da execução. Neste caso, uma outra instrução deve ter armazenado aí, o valor do dado a carregar no Acumulador.

ARM - ARMAZENA DADO DO ACUMULADOR

ARMX - ARMAZENA DADO DO ACUMULADOR (INDEXADO)

Uso: Estas instruções são destinadas ao armazenamento do valor que se encontra no Acumulador, na posição de memória indicada pelo operando da instrução. O valor contido no Acumulador não é alterado.

A instrução ARM, armazena o conteúdo do Acumulador na posição de memória cujo endereço é dado pelo operando, ou pelo conteúdo do operando (caso do endereçamento indireto).

A instrução ARMX, faz o mesmo, porém agora o endereço é dado pelo operando, somado ao valor do registrador de índice. Também neste caso vale o endereçamento indireto.

formulário para perfuração de cartões

"CROSS-MONTADOR"

FIGURA B2 - Instruções de armazenamento

pág. B.G.A

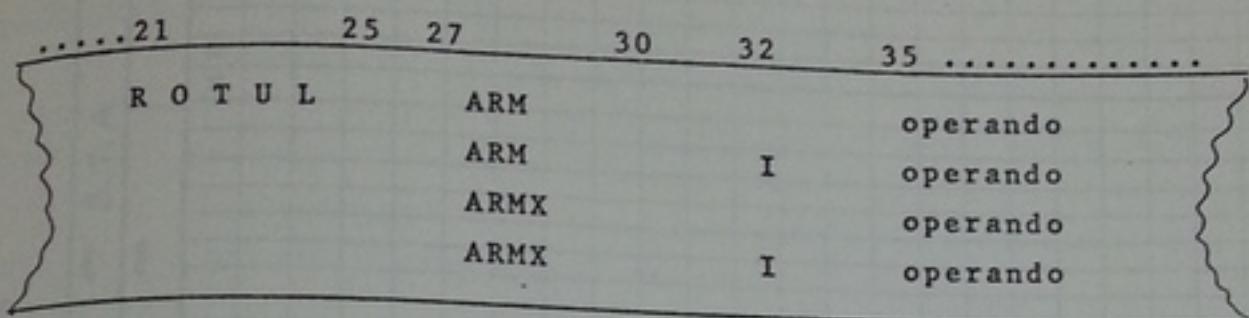
Nº e tipo do cartão

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

EX1 ARM ARM A A 1 0 0 0 / 1 0 0 A + 2 * + 2

EX2 ARMX ARMX A A 1 0 0 0 / 1 0 0 A + 2 * + 2

Formato:



Os operandos podem ser endereços simbólicos, absolutos, ou relativos.

Exemplos:

A figura B.2, apresenta exemplos de uso destas instruções. Valem para estes, as mesmas observações feitas para as instruções CAR e CARX.

B.4.b - Instruções de aritmética e lógica

Estas instruções se dividem em dois grandes grupos: as instruções aritméticas e as instruções lógicas.

São as únicas instruções que manipulam dois dados, dos quais o primeiro se encontra no Acumulador. O outro se encontra na posição de memória referenciada pelo operando da instrução, ou é do tipo imediato, isto é, se encontra na segunda palavra da instrução.

SOM - SOMA DADO COM O ACUMULADOR

SOMX - SOMA DADO COM O ACUMULADOR (INDEXADO)

SOMI - SOMA DADO IMEDIATO COM O ACUMULADOR

Uso: São usadas para adicionar ao conteúdo do Acumulador o conteúdo da posição de memória, cujo endereço é dado pelo operando da instrução. O resultado fica no Acumulador, e o conteúdo da posição de memória não é alterado.

formulário para perfuração de cartões

"CROSS-MONTADOR"

FIGURA B 3 - Instruções de aritmética e lógica

Exercício	Nome do cartão	seq.	B. f. A.
EX1	SOM	30 31	42 41
	SOM	1	A
	SOM	1000	
	SOM	/100	
	SOM	A+2	
	SOM	*+2	
EX2	SOMX	30 31	42 41
	SOMX	1	A
	SOMX	1000	
	SOMX	/100	
	SOMX	A+2	
	SOMX	*+2	
EX3	SOMI	10	
	SOMI	/10	
	SOMI		
EX4	NAND	10	
	NAND	/10	
	NAND		
EX5	XOR	10	
	XOR	/10	
	XOR		

A instrução SOM, soma um dado que se encontra na posição de memória indicada pelo operando, ao conteúdo do Acumulador, ou então faz isso com a palavra cujo endereço é dado pelo endereço do operando (endereçamento indireto).

A instrução SOMX, soma ao endereço dado pelo operando, o conteúdo do registrador de índice, calculando desta forma o endereço do dado. No resto procede como a instrução SOM.

A instrução SOMI soma ao conteúdo do Acumulador o dado que se encontra na segunda palavra da instrução.

Formato:

.....	21	25	27	30	32	35
R O T U L	S O M					operando	
	S O M		I				
	S O M X					operando	
	S O M X		I			operando	
	S O M I					operando	
	S O M I					operando	

No caso das instruções SOM e SOMX, os operandos podem ser endereços simbólicos, absolutos ou relativos. O operando da instrução SOMI deve, se existir, ser um número decimal ou hexadecimal, na faixa: 0 a 255 ou / 00 a /FF.

Exemplos:

A figura B.3, apresenta exemplos destas instruções.

Para estes valem as mesmas observações feitas para as instruções CAR, CARX, e CARI.

Observações: Estes comandos causam a atualização dos "flip-flops" T e V. O primeiro é posicionado em 1, se o resultado da operação exceder a capacidade do Acumulador. O segundo recebe o valor do "bit" "CARRY-OUT" da última posição do Acumulador.

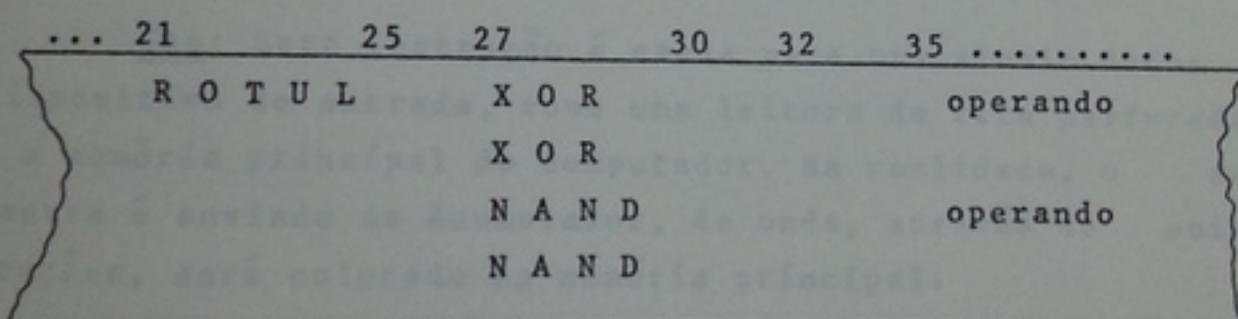
Não existe operação aritmética de subtração. Esta pode ser calculada somando-se ao minuendo o subtraendo, "em complemento para 2".

XOR - REALIZA OPERAÇÃO LÓGICA "XOR" COM ACUMULADOR
 NAND - REALIZA OPERAÇÃO LÓGICA "NAND" COM ACUMULADOR

Uso: Estas são instruções de lógica, as quais efetuam as operações na tabela abaixo, entre o conteúdo do Acumulador e o conteúdo da segunda palavra da instrução. Trata-se portanto de operações imediatas. O resultado da operação é colocado no Acumulador, e o conteúdo da posição de memória não se altera. As operações lógicas são realizadas "bit" a "bit" entre os dois operandos, e constituem recursos valiosos na manipulação de "bits".

BIT 1	BIT 2	RESULTADO(XOR)	RESULTADO(NAND)
0	0	0	1
0	1	1	1
1	0	1	1
1	1	0	0

Formato:



O operando deve ser um número decimal ou hexadecimal na faixa 0 a 255 ou / 00 a /FF.

Exemplos:

A figura B.3, mostra exemplos de uso destas operações (EX4 e EX5).

Valem para estas instruções as mesmas observações feitas para a instrução CARI.

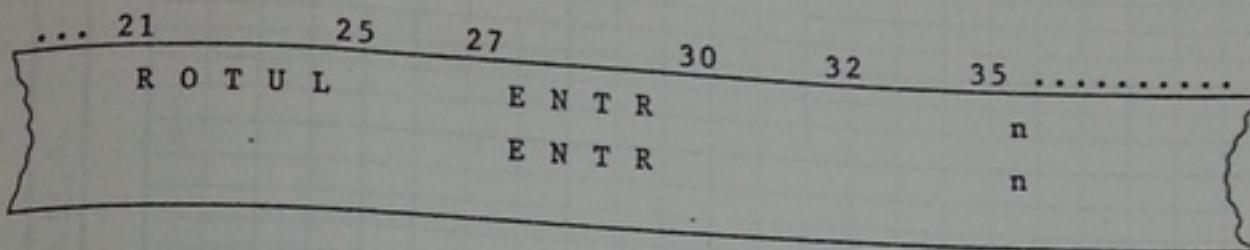
B.4.c - Instruções de Entrada e Saída

As instruções básicas de Entrada e Saída, servem para operar qualquer um dos dezesseis dispositivos de Entrada e Saída possíveis do sistema. Cada instrução realiza a entrada ou a saída de um dado na/da memória principal. A entrada ou saída de dados é sempre feita através do Acumulador. Assim, para se ter uma sequência de dados será necessária uma série de sub-rotinas de Entrada e Saída, tais como; entrada de dados, saída de dados, conversão de dados, e etc.

Além das Instruções de Entrada e Saída propriamente ditas, outras instruções são necessárias para determinar condições dos dispositivos como por exemplo, verificar se o dispositivo está ou não ocupado, e outras do tipo avançar para a linha seguinte da impressora.

ENTR - ENTRADA DE DADOS

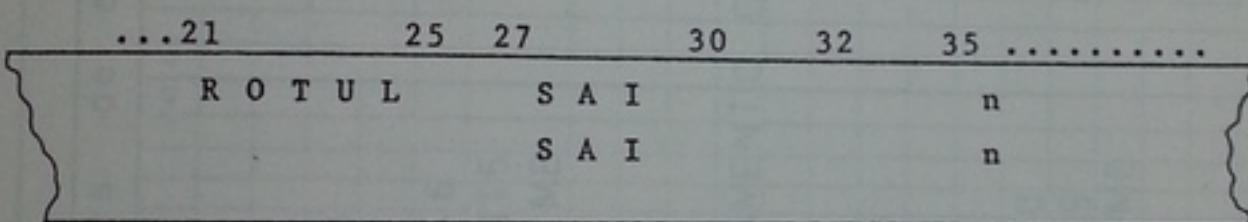
Uso: Esta instrução é usada para transferir dados de um dispositivo de entrada, como uma leitora de fita perfurada, para a memória principal do computador. Na realidade, o dado que entra é enviado ao Acumulador, de onde, através de outras instruções, será colocado na memória principal.

Formato:

Neste caso, "n" representa um número de um dispositivo lógico e pode estar na faixa de 0 a 15.

SAI - SAIDA DE DADO

Uso: Esta instrução transfere o dado que se encontra no Acumulador, para um dispositivo de saída. Este dispositivo pode ser qualquer um dos dezesseis possíveis no sistema. O conteúdo do Acumulador não é alterado.

Formato:

Neste caso, "n" representa um dispositivo lógico e pode estar na faixa de 0 a 15.

Exemplos:

O exemplo EX1 da figura B.4, mostra o uso da instrução ENTR e SAI: um dado, de um dispositivo lógico de número 5 é colocado no Acumulador. A instrução codificada na linha seguinte pede a saída deste mesmo valor através de um dispositivo lógico de número 7. O valor contido no Acumulador, isto é, o valor lido é então armazenado, através da instrução codificada na terceira linha, na posição de memória 56.

formulário para perfuração de cartões

"CROSS-MONTADOR"

FIGURA B4 -

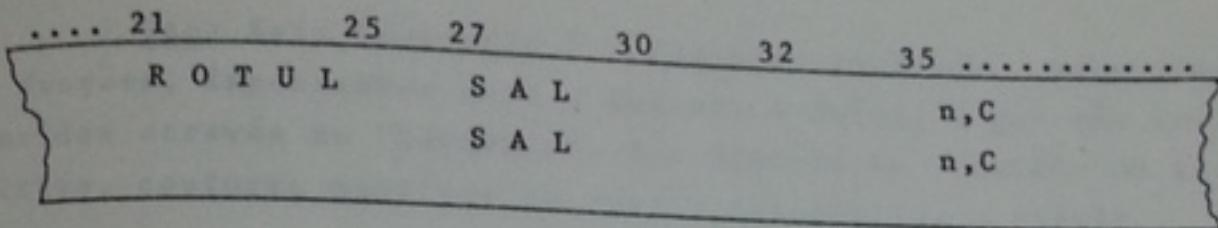
Sistema		Instruções de entrada e saída										pag. B.II.A					
nº de leitura		1	10	11	20	21	30	31	40	41	50	51	60	61	70	71	80
EX1		ENTR	5														
		SAI	7														
		ARM	56														
EX2		SAL	5,6														
		PLA	INT5														
		CAR	SIMB														
			:														
EX3		FNC	5,0														
		FNC	8,5														
		CAR	SIMB														

* ROTINA DE TRATAMENTO DE INTERRUPÇÃO PARA O
DISPOSITIVO 5
INT5

SAL - TESTE DE ESTADO DE DISPOSITIVO

Uso: Estas instruções são utilizadas para testar determinadas condições de um dispositivo, e se estas forem satisfeitas, saltar a instrução seguinte.

Formato:



"n" representa um dispositivo lógico entre os 16 permitidos pelo sistema. "n" deve ser um número decimal entre 0 e 15.

"C" representa a condição a ser testada. Estas condições são implementadas em "hardware", e as permitidas, são as que constituem a tabela abaixo:

Valor de C	Condição a testar
1	testar se o "flip-flop" de Estado do dispositivo "n", está ligado. Saltar a instrução seguinte se a condição é satisfeita.
6	testar se há pedido de interrupção para o dispositivo "n". Saltar a instrução seguinte se a condição não for satisfeita.

Exemplo:

A figura B.4, mostra em EX2 um exemplo de uso de SAL.

A instrução testa se há pedido de interrupção para o dispositivo 5, e se não existir, prossegue na sequência normal. Se houver, ocorre um desvio para a rotina de tratamento de interrupção em INT5.

FNC - FUNÇÃO DE ENTRADA E SAÍDA

Uso: Esta instrução é usada para realizar determinadas funções, importantes para a Entrada e Saída, e que são estabelecidas através de "hardware". São funções de controle do dispositivo, conforme mostrado na tabela apresentada a seguir.

Formato:

....	21	25	27	30	32	35
	R O T U L		F N C			n,C	
			F N C			n,C	

Neste caso, "n" representa um dispositivo lógico entre os 16 permitidos pelo sistema. "n" deve ser um número decimal entre 0 e 15.

"C" representa a função a ser realizada. As permitidas são apresentadas na tabela a seguir:

Valor de C	Função implementada
1	Faz igual a zero o "flip-flop" de Estado do dispositivo "n".
0	Impede interrupção para o dispositivo "n".
2	Dispara o "flip-flop" de Estado do dispositivo "n".
4	Faz igual a zero o "flip-flop" de interrupção do dispositivo "n".
5	Permite interrupção para o dispositivo "n".
6	Dispara o "flip-flop" de controle do dispositivo "n".
7	Faz igual a zero o "flip-flop" de controle do dispositivo "n".

Exemplo:

No exemplo EX3 apresentado na figura B.4, a primeira instrução impede que ocorram interrupções para o dispositivo lógico 5. A instrução seguinte vai permitir a ocorrência destas, para o dispositivo lógico 8. Em seguida, um dado que se encontra no endereço simbólico SIMB é carregado no Acumulador.

Observações: O formato das instruções de Entrada e Saída apresentado impunha que "n" e "C" fossem números decimais, porém a linguagem permite também que estes operandos sejam simbólicos. Neste caso, o valor tomado é o seu endereço. O seguinte código é permitido:

.....	21	25	27	30	32	35
				S A I		W	
				E N T R		J	
				S A L		MAT, 6	
				F N C		R0,5	
J				E Q U		2	
W				E Q U		5	
M A T				E Q U		4	
R O				E Q U		5	

A "pseudo" instrução EQU a ser vista mais a frente, está endereçando os dispositivos, cujo nome simbólico é J,W,MAT,RO, respectivamente nos endereços 2,5,4,5. Estes serão os valores substituídos nas instruções do grupo Entrada e Saída. A primeira instrução faz sair um dado pelo dispositivo lógico de número 5. A segunda instrução, faz a entrada de um dado pelo dispositivo lógico de número 2. A terceira instrução, testa se há interrupção para o dispositivo lógico de número 2. A quarta instrução, testa se há interrupção para o dispositivo de número 4, e a quinta permite interrupção para o dispositivo de número 5.

O uso de instruções neste formato é um recurso poderoso, quando se deseja fazer um programa flexível quanto ao uso de dispositivos lógicos. Referenciando-se estes através de "nomes" e não de números, facilita-se a troca de um dispositivo, por exemplo, devido a uma pane ocorrida. Para isso, é suficiente trocar o comando de EQU, que o endereça, substituindo o operando do comando, pelo número do novo dispositivo a ser usado.

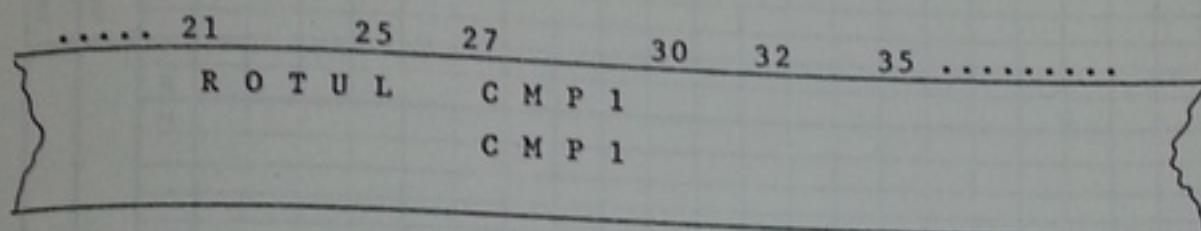
B.4.d-Operações com o Acumulador

É muito interessante ao programador dispor de algumas instruções, que lhe permitam operar diretamente com o Acumulador. Estas se tornam particularmente valiosas no caso de um minicomputador. Podem ser situadas neste grupo as instruções que se seguem:

CMP1 - COMPLEMENTO DE UM DO ACUMULADOR

Uso: Esta instrução calcula o "complemento de um" do dado armazenado no Acumulador. O resultado da operação permanece neste registrador. Além disso, ela faz igual a ZERO os "flip-flop" V e T.

Formato:



Exemplo:

No exemplo EX1 da figura B.5 a primeira instrução, carrega no Acumulador o valor 10. A configuração deste, após esta operação, será:

0000 1010

A segunda instrução "complementa para um" este valor, resultando:

1111 0101

CMP2 - COMPLEMENTO DE DOIS DO ACUMULADOR

Uso: Esta instrução calcula o "complemento de dois", de um dado armazenado no Acumulador, permanecendo neste o resultado. A operação faz igual a ZERO os "flip-flops" V e T. A vantagem desta instrução sobre a anterior está no fato de que ela dá a representação de um número negativo a partir do número positivo.

formulário para perfuração de cartões

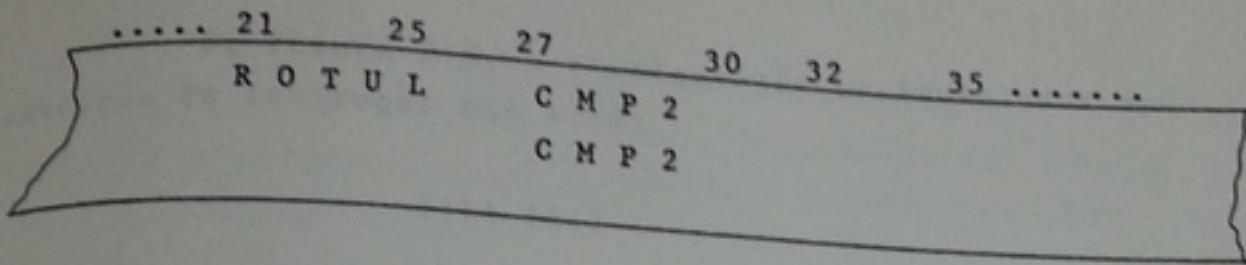
"CROSS - MONTADOR"

Sistema
para o cartão

FIGURA B5 - Instruções para o Acumulador

Ex	Car	Car	data												
			10	11	20	21	30	31	40	41	50	51	60	61	70
EX1	CARI	10													
	CMP1														
EX2	CARI	10													
	CMP2														
EX3	UM														
EX4	UNEG														
EX5	CARI	10													
	LIMP	1													

Formato:



Exemplo:

No exemplo EX2 da figura B.5, a primeira instrução carrega no Acumulador o valor 10. A configuração deste, após esta operação, será:

0000 1010

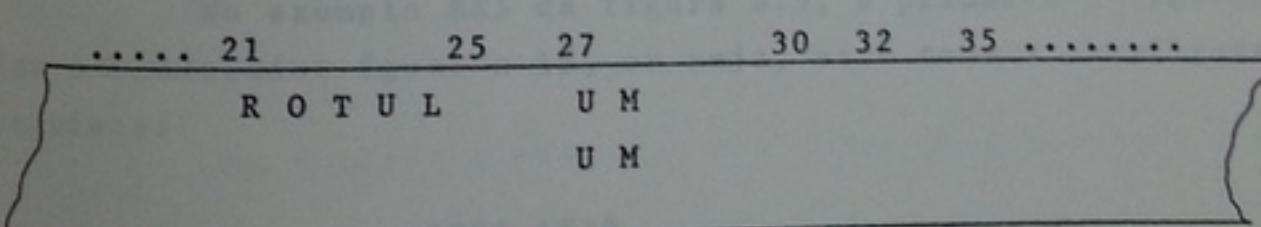
A instrução seguinte calcula a representação negativa deste número, ou seja:

1111 0110

UM - COLOCA UM NO ACUMULADOR

Uso: Esta instrução coloca o número 1 no Acumulador. Ao mesmo tempo, faz os "flip-flops" V e T igual a ZERO.

Formato:



Exemplo:

Na figura B.5, a configuração no Acumulador após a execução da instrução EX4 será:

1111 1111

LIMP - FAZ O ACUMULADOR ZERO

Uso: Esta instrução faz o Acumulador igual a ZERO, posicionando o "flip-flop" V no valor especificado pelo operando.

Formato:

.....	21	25	27	30	32	35
R	O	T	U	L	L	I	M
						P	n

O valor do operando "n" pode ser decimal: UM ou ZERO. Outros valores não são permitidos. O campo, entretanto, pode ser deixado em branco, assumindo-se então o valor ZERO. O "flip-flop" T é feito igual a ZERO.

Exemplo:

No exemplo EX5 da figura B.5, a primeira instrução faz o Acumulador igual a 10, ou seja, este toma a configuração seguinte:

0000 1010

A instrução de "LIMP" que se segue faz esta configuração, tornar-se

0000 0000

enquanto o "flip-flop" V recebe o valor 1.

B.4.e Operações com o Painel de Chaves

Constitui este grupo um conjunto de oito instruções que manipulam os dados de painel. O conteúdo das chaves do painel operados segundo uma função "n", especificada pelo programa dor, é colocado no Acumulador.

PNL - OPERAÇÃO COM PAINEL DE CHAVES

Uso: É a única instrução deste grupo, as 8 opções sendo dadas pelo valor do seu operando.

Formato:

.....	21	25	27	30	32	35
	R	O	T	U	L	P	N
							L

onde "n" representa uma das oito funções que podem ser realizadas por este comando. "n" é um número decimal, entre ZERO e SETE.

As funções permitidas são dadas pela tabela a seguir.

VALOR DE n	FUNÇÃO
Ø	O dado introduzido pela chave de painel é colocado no Acumulador. Os "flip-flops" V e T recebem o valor Ø.
1	O dado introduzido pelas chaves de painel é somado a 1 e colocado no Acumulador. Os "flip-flops" V e T recebem o valor Ø.
2	O conteúdo do Acumulador incrementado de 1 é subtraído do conteúdo do registro de chaves do painel, o resultado sendo colocado no Acumulador. Os "flip-flops" V e T recebem o valor Ø.
3	O conteúdo do Acumulador é subtraído do conteúdo do registrador de chaves, e o resultado é enviado ao Acumulador. Os "flip-flops" recebem o valor Ø.
4	O conteúdo do registrador de painel é somado ao conteúdo do Acumulador, sendo o resultado enviado ao Acumulador. Os "flip-flops" V e T recebem o valor Ø.
5	O conteúdo do registrador de chaves é somado ao conteúdo do Acumulador e incrementado de 1. O resultado da operação é colocado no Acumulador. Os "flip-flops" V e T recebem o valor Ø.
6	O conteúdo do registrador de chaves é dividido de 1 e enviado ao Acumulador. Os "flip-flops" V e T recebem o valor Ø.
7	O conteúdo do registrador de chaves é enviado ao Acumulador, e o "flip-flop" V e T iniciado com 1.

Exemplo:

A figura B.6, apresenta um exemplo desta instrução:
Suponhamos que o conteúdo do registrador de chaves seja:
ja:

0000 0000 0011

Pela primeira instrução do exemplo, o Acumulador receberá o valor 10. Após a instrução PNL, o valor do Acumulador será 13.

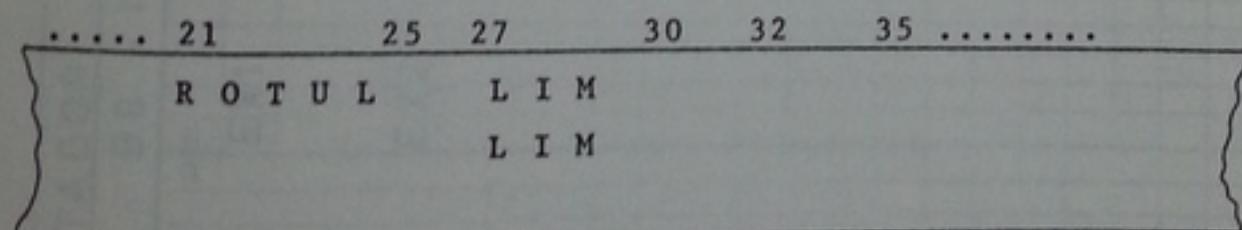
B.4.f Instruções para os "flip-flops" T e V

Estas instruções são usadas para testar o estado dos "flip-flops" T e V, da Unidade Aritmética. "V" é o bit de derrame nas instruções de deslocamento e giros e o "carry-out" das operações aritméticas, e T é o estouro das operações aritméticas.

LIM - FAZ ZERO OS "FLIP-FLOPS" "V" e "T"

Uso: Esta instrução é usada para tornar zero, os "flip-flops" V e T.

Formato :



formulário para perfuração de cartões

sistema

"CROSS-MONTADOR"

note do cartão

FIGURA B6 - Instruções para o painel de chaves/FF-T e V

peg.

B.21.A

10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

EX1 CARI 10
PNL 4

EX2 SLV 1
SLT 0
SLVM 0
SLTM 1

SLV - TESTA "FLIP-FLOP" "V"

SLVM - TESTA "FLIP-FLOP" "V" E MUDA O SEU VALOR

Uso: Estas instruções testam o estado do "flip-flop" "V". Se este "flip-flop" se encontrar no estado especificado pelo operando da instrução, então a instrução seguinte não é executada. Se o estado do "flip-flop" for diferente, então a instrução seguinte é executada. A instrução SLVM, no caso da condição testa ser verdadeira, faz a complementação do valor "flip-flop".

Formato:

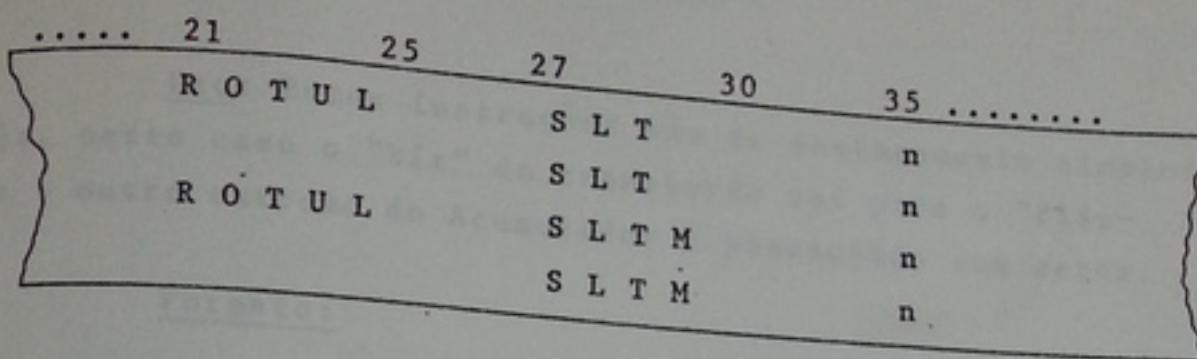
.....	21	25	27	30	32	35			
	R	O	T	U	L	S	L	V	n	
						S	L	V	n	
	R	O	T	U	L	S	L	V	M	n
						S	L	V	M	n

Onde "n", o operando, pode ter o valor 1 ou 0 e especifica a condição em teste. Valores diferentes são proibidos.

SLT - TESTA "FLIP-FLOP" "T"

SLTM - TESTA "FLIP-FLOP" "T" E MUDA

Uso: Essas instruções testam o estado do "flip-flop" T. Se este "flip-flop" se encontrar no estado especificado pelo operando da instrução, a instrução seguinte não é executada. Se o estado do "flip-flop" for diferente, então a instrução seguinte é executada. A instrução SLTM complementa o valor do "flip-flop" se a condição testada é satisfeita.

Formato:Exemplo:

No exemplo EX2 mostrado na figura B6 a primeira instrução testa o valor do "flip-flop" V e se este for 1 a instrução seguinte: SLT não é executada. Por sua vez, esta instrução verifica se o "flip-flop" T tem o valor Ø. Se este tiver tal valor, a instrução seguinte não é executada. A instrução de SLVM testa o valor do "flip-flop" V. Se este tiver o valor Ø, a instrução seguinte é saltada e o seu valor será feito 1. A instrução SLTM tem o mesmo procedimento da anterior, testando o valor do "flip-flop" T.

B.4. g - Instruções de deslocamentos de dados

Em alguns casos pode ser interessante deslocar-se os dados que se encontram no Acumulador, afim de se trabalhar com determinados grupos de "bits", ou testar estes grupos específicos. Para tanto são usadas as instruções de deslocamentos. Os deslocamentos podem ser de quatro tipos: giros, deslocamentos com propagação do sinal, deslocamento sem propagação de sinal e deslocamentos envolvendo o "flip-flop" V.

DE - DESLOCAMENTO A ESQUERDA
DD - DESLOCAMENTO A DIREITA

Uso: Estas instruções são de deslocamento simples, ou seja, neste caso o "bit" de transbordo vai para o "flip-flop" e o outro extremo do Acumulador é preenchido com zeros.

Formato:

.....	21	25	27	30	32	35
	R	O	T	U	L	D	D
						n	
	R	O	T	U	L	D	D
						n	
				D	E	n	
				D	E	n	

A instrução DD desloca o Acumulador para a direita, preenchendo com zeros à esquerda do Acumulador. A instrução DE faz exatamente o oposto.

O operando "n", que pode ser um número decimal entre 0 e 4, dá um número de posições a serem deslocadas.

Exemplo:

No exemplo EX1 da figura B.7, a primeira instrução desloca o Acumulador de duas posições para a direita. Suponhamos que o conteúdo deste fosse:

1000 1101

e ao fim da execução desta instrução o resultado seria:

0010 0011

A instrução seguinte, desloca o Acumulador de duas posições para a esquerda. Assumindo-se o mesmo conteúdo, o resultado seria:

0011 0100

formulário para perfuração de cartões

"CROSS-MONTADOR"

FIGURA B7 - Instruções de deslocamentos e giros

lote do cartão	data	reg. 8.24.A
	10 11	10 11
	22 23	22 23
EX1	DD DE	DD DE
	2	2
EX2	GE GO	GE GO
	2	2
EX3	DDS	DDS
	2	2
EX4	GDV GEV DDV DEV	GDV GEV DDV DEV
	2 2 2 1	2 2 2 1

GE - GIRO A ESQUERDA
 GD - GIRO A DIREITA

Uso: Estas instruções deslocam o conteúdo do Acumulador, respectivamente para a esquerda e para a direita, do número de posições especificadas pelo operando, sendo as posições vagas preenchidas com os "bits" deslocados para fora na outra extremidade.

Formato:

.....	21	25	27	30	32	35
	R	O	T	U	L	G	D
							n
						G	D
						E	n
						E	n
							n

O operando "n", deve ser um número decimal entre 0 e 4, e indica o número de posições que devem ser deslocadas em anel.

Exemplos:

No exemplo, vamos supor que o conteúdo do Acumulador seja:

1000 1101

A primeira instrução gira este conteúdo de duas posições para a esquerda. Resulta portanto:

0011 0110

A segunda instrução gira este conteúdo para a direita, assumindo que o valor do Acumulador fosse:

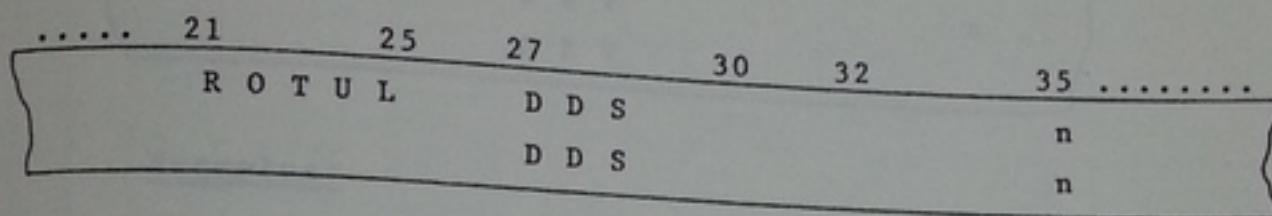
1000 1101

Após a sua execução, teríamos:

0110 0011

Uso: Nesta instrução o "bit" de sinal ("bit" 8 do Acumulador) é deslocado para a direita, propagando o seu valor para o interior do Acumulador, à medida que se processam os deslocamentos.

Formato:



O operando "n", deve ser um número entre 0 e 4, e indica o número de deslocamentos.

Exemplo:

No exemplo EX3 da figura B.7, vamos supor que o valor do Acumulador seja:

1101 0000

Após a execução da instrução, teremos:

1111 1010

DDV - DESLOCAMENTO A DIREITA COM "V"

DEV - DESLOCAMENTO A ESQUERDA COM "V"

GDV - GIRO A DIREITA COM "V"

GEV - GIRO A ESQUERDA COM "V"

Uso: As duas primeiras instruções são deslocamentos simples, correspondendo às instruções DD e DE. As duas últimas, correspondem às instruções GD e GE. As diferenças entre estas instruções e aquelas é que ao se dar o deslocamento ou o giro o conteúdo do "flip-flop" V é introduzido no Acumulador, no extremo oposto.

Formato:

...	21	25	27	30	32	35
ROTUL			D D V				
ROTUL			D D V			n	
ROTUL			D E V			n	
ROTUL			D E V			n	
ROTUL			G D V			n	
ROTUL			G D V			n	
ROTUL			G E V			n	
ROTUL			G E V			n	

Exemplos:

A figura B.7, exemplo EX4 mostra o uso destas instruções.

Suponhamos que o Acumulador, antes da execução da primeira instrução, contenha o valor:

1111 0010

e que o "flip-flop" V contenha um valor:

1

Após a execução da primeira instrução, assim como as seguintes, os resultados seriam, sucessivamente e respectivamente:

0111 1100

1111 0010

0111 1100

1111 1001

B.4 h - Instruções utilizadas na mudança de curso do programa

Durante a execução de um programa, existem situações em que se deseja desviar sob determinadas condições para outros pontos do programa, ou mesmo sem que qualquer teste seja feito. Para se conseguir tais objetivos, necessita-se de comandos que alterem a sequência normal do programa: os chamados comandos de desvios.

formulário para perfuração de cartões

"CROSS-MONTADOR"

FIGURA B8 - Instruções de desvios

nº do cartão		FIGURA B8 - Instruções de desvios										pág. B.24.A				
1	10	11	12	21	EX1	PLAN	A	43	44	50	51	52	53	70	71	80
						PLAN	1	A								
						PLAN		1000								
						PLAN		/100.								
						PLAN		A+2								
						PLAN		*+2								
EX2		PLAZ	A			PLAZ	1	A								
		PLAZ				PLAZ		1000								
		PLAZ				PLAZ		/100								
		PLAZ				PLAZ		A+2								
		PLAZ				PLAZ		*+2								
EX3		PUG	A			PUG	1	A								
		PUG				PUG		1000								
		PUG				PUG		/100								
		PUG				PUG		A+2								
		PUG				PUG		*+2								

PLAN - DESVIO SE ACUMULADOR NEGATIVO
PLAZ - DESVIO SE ACUMULADOR ZERO

Uso: Estes dois comando realizam desvios condicionais, isto é, desviam para uma dada posição do programa, indicada pelo operando quando determinadas condições são satisfeitas. O primeiro:PLAN, desvia se o conteúdo do Acumulador for negativo e o segundo:PLAZ, causa o desvio se o conteúdo do mesmo registrador for nulo.

Formato:

.....	21	25	27	30	32	35.....
	R O T U L	P L A N				operando
	R O T U L	P L A N				operando
	R O T U L	P L A Z				operando
		P L A Z				operando

Neste caso, os operandos podem ser endereços simbólicos, absolutos ou relativos.

Exemplos:

Nos rótulos EX1 e EX2 da figura B.8, mostram-se exemplos destas instruções. Para esses exemplos valem as mesmas observações feitas para as instruções CAR e CARX.

PLA - DESVIO INCONDICIONAL

PLAX - DESVIO INCONDICIONAL (INDEXADO)

Uso: Estes dois comandos desviam o curso normal do programa, sem efetuar quaisquer testes sobre o Acumulador, ou outros indicadores ou "flip-flops". PLA, tem modo de endereçamento direto em "hardware", enquanto que o segundo PLAX, tem endereçamento indexado implícito semelhante ao visto para CARX, SOMX e ARMX.

formulário para perfuração de cartões

*CROSS-MONTADOR"

FIGURA B9 - Instruções de desvios

sistema

nº do cartão

pag. 5.28.A

data

	10 11	12 21	30 31	40 41	50 51	60 61	70 71	80 81
EX1	PLA	I	A					
	PLA		A					
	PLA		1000					
	PLA		/100					
	PLA		A+2					
	PLA		*+2					
EX2	PLAX	I	A					
	PLAX		A					
	PLAX		1000					
	PLAX		/100					
	PLAX		A+2					
	PLAX		*+2					

Formato:

....	21	25	27	30	32	35.....
	R O T U L		P L A	I		operando
	R O T U L		P L A			operando
			P L A X			operando
			P L A X	I		operando

No caso os endereços podem ser endereços simbólicos, absolutos ou relativos.

Exemplos:

A figura B.9, mostra exemplos destas instruções. Para esses valem as mesmas observações feitas para as instruções CAR e CARX.

PUG - DESVIO PARA SUBROTINA

Uso: Esta instrução é usada para desviar o curso normal do programa, para um endereço que é dado pelo operando, e guardar aí o valor do Contador de Instruções. Isto é particularmente interessante quando se deseja desviar o curso do programa, e retornar depois ao ponto de onde se fez o desvio. O endereço de execução é o endereço efetivo da instrução acrescido de 2.

Formato:

....	21	25	27	30	32	35.....
	R O T U L		P U G			operando
			P U G	I		operando

No caso os endereços podem ser simbólicos , absolutos ou relativos.

Exemplos:

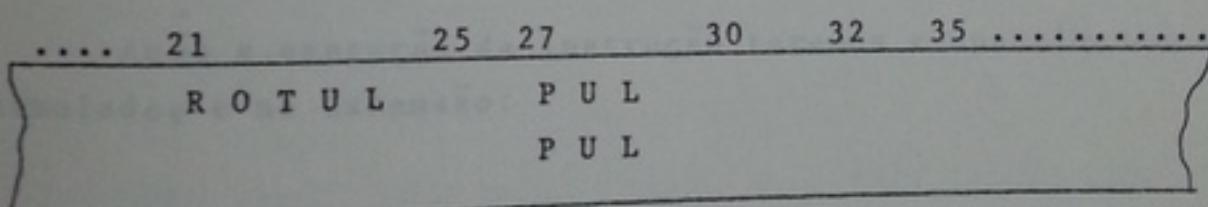
A figura B.8, mostra no EX3 exemplos de uso destas instruções.

Para esses, valem as mesmas observações feitas para as instruções CAR e CARX.

PUL - RETORNO DE INTERRUPÇÃO

Uso: Quando ocorre uma interrupção, o conteúdo do Contador de Instruções, é guardado nas posições 2 e 3 da memória principal. A instrução PUL, permite que se retorne de uma interrupção. O processo é o seguinte: a instrução de PUL carrega no Contador de Instruções o valor 2, isto é, o computador irá executar um desvio para esta posição e executar em seguida a instrução que aí se encontre. Como o Contador de Instruções, que foi aí depositado por ocasião da interrupção tem 12 bits de extensão, os 4 bits mais significativos, que restam para completar 16 bits, isto é, duas palavras são preenchidas com zeros. Quando se tenta executar esta instrução na posição 2, encontra-se o código de operação 0000, e que se pode ver, (Apêndice A) é uma instrução de Desvio incondicional "PLA". Desta maneira se retorna à posição anterior à interrupção.

Formato:



B.4. i - Comandos especiais

São alguns comandos de uso geral destinados a fins diversos e que poderíamos englobar numa parte chamada "miscelânea".

TRE - TROCA ACUMULADOR E EXTENSÃO

Uso: Esta instrução é utilizada para carregar a Extensão do Acumulador a partir dele mesmo. A extensão como foi visto se encontra na posição 1 da memória. Nesse caso o conteúdo da Extensão é colocado no Acumulador, e vice versa.

Formato:

.....	21	25	27	30	32	35.....
	R O T U L		T R E			
			T R E			

Exemplos:

O exemplo EX1 da figura B.10, mostra o seu uso. Vamos assumir que o conteúdo do Acumulador seja:

0001 1100

e que o conteúdo da Extensão seja:

1100 1111

Após a execução da instrução teremos respectivamente no Acumulador e na Extensão:

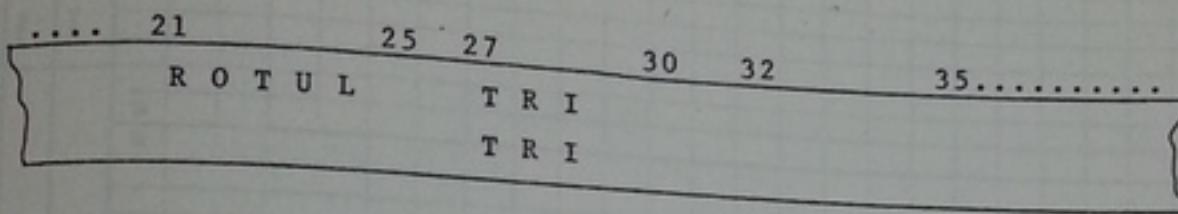
1100 1111

0001 1100

TRI - TROCA ACUMULADOR E INDICE

Uso: Esta instrução é utilizada para carregar o registro indexador a partir do Acumulador. Esta instrução faz a troca entre os conteúdos do Acumulador e do Indexador.

Formato:



Exemplos:

A figura B.10, mostra o uso desta instrução
Vamos assumir no Acumulador a seguinte configuração de
"bits":

0000 1101

e no registrador de Índice:

1101 1101

Após a execução da instrução teremos respectivamente,
no Acumulador e no Indexador as seguintes configurações:

1101 1101
0000 1101

PARE - PARADA

Uso: Esta instrução é utilizada quando se quer suspender a execução de um programa. O programa só pode ser reiniciado com intervenção do painel, não aceitando interrupção.

Formato:

..... 21 25 27 30 32 35.....
ROTUL P A R E P A R E
 { }

ESP - PAUSA

Uso: Esta instrução também serve para provocar a suspensão de um programa, que se encontra em execução e ficará, esperando por uma interrupção, ou por uma reiniciação via painel. Tais pretensões são justificadas no caso em se deseja tomar algumas decisões antes que determinadas partes do programa sejam executadas.

Formato:

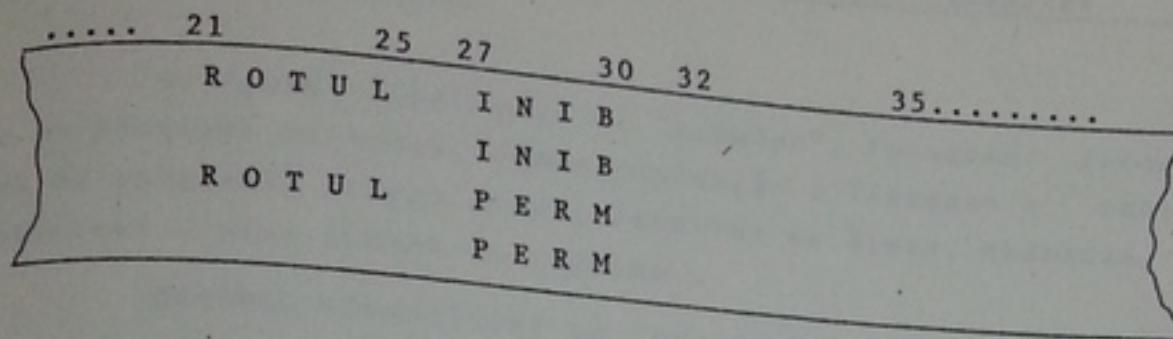
.... 21 25 27 30 32 35,
ROTUL E S P E S P
 { }

INIB - INIBE SISTEMA DE INTERRUPÇÃO

PERM - DESINIBE SISTEMA DE INTERRUPÇÃO

Uso: Quando alguns trechos de um programa não devem sofrer interrupções, a fim de não prejudicar o seu funcionamento, é conveniente prevenir tais fatos. Para isso existem duas instruções as quais controlam o sistema de interrupção. A primeira, INIB impede que interrupções ocorram, enquanto não for encontrada uma segunda instrução: PERM a qual desinibe o sistema. Durante o trecho que separa uma da outra, interrupções não suspendem o curso deste programa.

Formato:



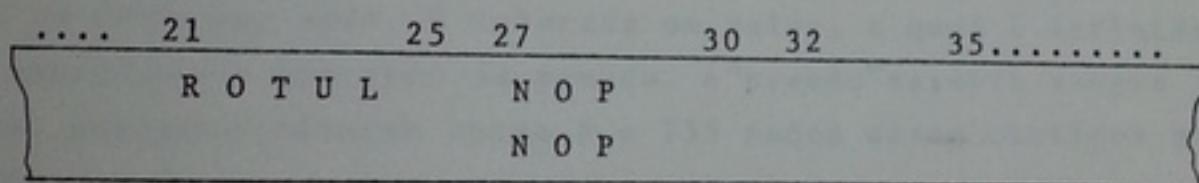
Exemplos:

O exemplo EX4 da figura B.10, é mostrado e exposto acima. Entre as instruções de rótulos A e B, as interrupções es tão inibidas.

NOP - NÃO OPERAÇÃO

Uso: O comando NOP não causa nenhuma modificação no estado do programa. Ela se torna interessante no caso de se desejar aceitar endereços ou na depuração de um programa. Um uso bastante importante é a carga de instruções sobre posições onde antes existia um NOP.

Formato:



Exemplo:

No exemplo EX5 apresentado na figura B.10 a posição chamada MUDA contém um NOP no instante da montagem. Um trecho do programa AAA altera o valor desta posição, montando aí uma instrução de desvio PLA. Quando o programa passar por esta posição, a instrução de PLA será executada alterando o curso do programa. No caso de um minicomputador alterações deste tipo são recurso de grande potência.

B.5. - Pseudo - instrução da linguagem Assembler ou "Pseudos"

As pseudo-instruções ou "pseudos", fornecem informações ao programa Montador, sobre paginação e listagem e cabeças de programa, origem deste, reservas de áreas, chamadas de subrotinas e seus pontos de entrada.

Podemos classificar as "pseudos" em alguns grupos:

- reservas de áreas
- controle de paginação
- definição de "pontos de entrada" e chamada de subrotina
- comandos de origem e fim de programa

B.5. a - "Pseudos" de reserva de área de memória

Pertencem a este grupo as "pseudos" por meio das quais se reserva área de memória preenchendo ou não esta área com um conteúdo bem definido.

DEFC - DEFINE CONSTANTE

Uso: Esta instrução é usada para se reservar uma posição na memória, onde é colocada um valor, o qual é definido pelo conteúdo do operando da pseudo. A "pseudo" reserva sempre 8 bits, portanto valores entre 0 e 255 podem estar contidos no operando.

Formato:

...	21	25	27	30	32	35.....
	R O T U L	D E F C			n	
		D E F C			*	*

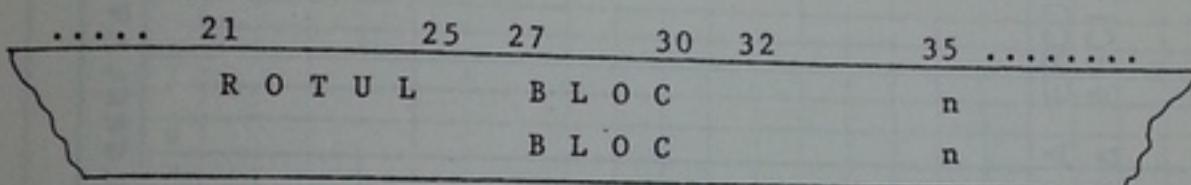
Neste caso "n" pode ser um número decimal, ou um número hexadecimal, neste caso representado por uma "/" seguida de uma combinação de dígitos hexadecimais.

Exemplos:

A primeira "pseudo" de rótulo EX1 na figura B.11, reser-
va uma posição de memória preenchendo-a com o valor 21. A segun-
da coloca na posição reservada o valor hexadecimal 10, isto é,
o decimal 16. A terceira preenche a posição reservada com o va-
lor 0.

BLOC - RESERVA BLOCO DE PALAVRAS

Uso: Esta "pseudo" é utilizada para reservar uma área de
tamanho especificado pelo operando, sem contudo colocar nenhum
valor nestas posições. Áreas reservadas desta maneira tem uma
infinidade de usos, como áreas de trabalho, de leitura ou para
armazenamento de resultados.

Formato:

Neste caso "n" é um número decimal.

Exemplo:

Na figura B.11, EX2 mostra o uso desta "pseudo".

Neste caso foi reservada após a chamada da rotina LE,
uma área de 60 posições de memória a qual se pretende usar
como área de trabalho para esta rotina.

formulário para perfuração de cartões

"CROSS-MONTADOR"

sistema

FIGURA B11 - Pseudos para reserva de área

Figura do cartão

Ex1 DEFC 21

DEFC /10

DEFC *-*

Ex2 CALL LE

BLOC 60

Ex3 DEFE A A+1

DEFE 1000

DEFE /100

DEFE *+1

A COM 10

B COM 1

C COM 3

MENS1 EBC
MENS2 ASC

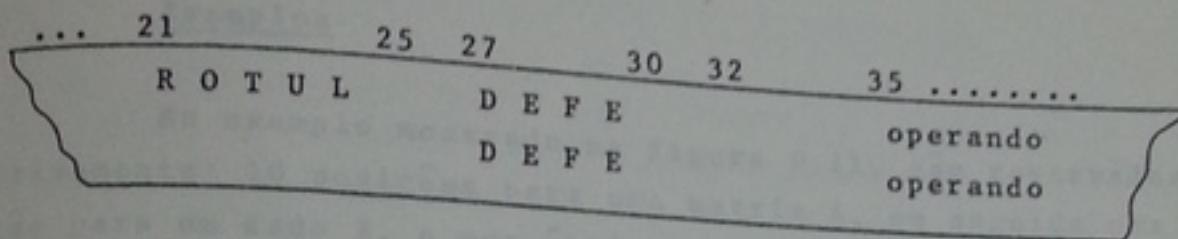
• CADEIA EBCDIC.
• CADEIA ASCII.

DEFE - DEFINE ENDEREÇO

B.37

Uso: Por meio deste comando pode-se definir um endereço, isto é, esta "pseudo" reserva duas posições da memória, destinadas ao armazenamento do endereço, que vem codificado no operando. As 4 primeiras posições vagas da palavra mais significativa são preenchidas com zeros.

Formato:



Neste caso, o operando pode ser um endereço absoluto, simbólico ou relativo. Não pode, entretanto, ser endereço indireto.

Exemplos:

A figura B.11, apresenta exemplos de uso desta "pseudo", em EX3.

Neste caso, verificar os exemplos da instrução PLA sem levar em conta porém endereçamento indireto.

COM - DEFINE SÍMBOLO EM ÁREA COMUM

Uso: Por meio desta "pseudo" pode-se, definir símbolo em Área Comum, a diversos programas e rotinas. A equivalência entre símbolos definidos em área comum, em diversos programas se faz de maneira posicional. A Área Comum se situa no fim da memória.

A vantagem deste processo é não necessitar a transferência de parâmetros entre um programa e uma subrotina por ele chamada.

Formato:

... 21 25 27 30 32 35
R O T U L C O M n

Onde "n" é um número decimal.

Exemplos

No exemplo mostrado na figura B.11, são reservadas respectivamente: 10 posições para uma matriz A, em seguida uma posição para um dado B, e por último 3 posições para uma matriz C.

EBC - DEFINE CADEIA DE CARACTERES EBCDIC
ASC - DEFINE CADEIA DE CARACTERES ASCII

Uso: Estas "pseudos" são usadas para reservar área para uma cadeia de caracteres EBCDIC ou ASCII dentro do programa. Cadeias deste tipo podem ser muito úteis para o uso de mensagens. A primeira faz a reserva de caracteres EBCDIC e a segunda de caracteres ASCII.

Formato:

... 21 25 27 30 32 35
R O T U L E B C .caracteres.
 E B C .caracteres.
R O T U L A S C . caracterес.
 A S C . caracterес.

Exemplos:

No exemplo mostrado na figura B.11, são reservadas áreas para duas cadeias, a primeira preenchida com caracteres EBCDIC, e rotulada MENS1, e a segunda, para uma cadeia de caracteres ASCII, rotulada MENS2.

B.5.b - "Pseudos" de controle de paginação

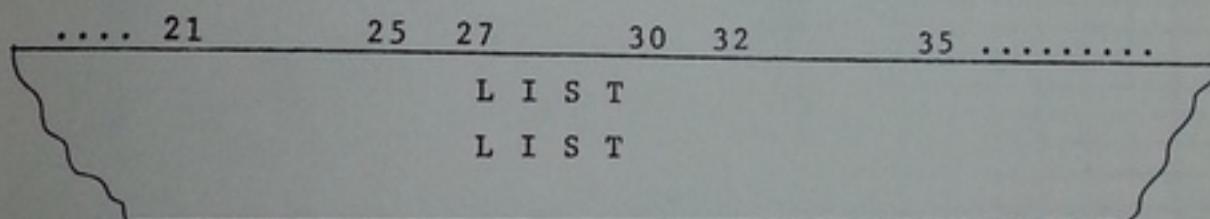
São "pseudos" usadas para controlar a listagem do programa. Não são traduzidas em linguagem de máquina e portanto não causam reserva de área.

LIST - LISTAR COMANDOS FONTE

NLST - INIBIR LISTAGEM DE COMANDOS FONTE

Uso: Estas "pseudos" são usadas para pedir a listagem, ou retirá-la de um trecho do programa. LIST liga o indicador que faz a listagem do programa, e NLST, desliga este indicador.

Formato:



Exemplo:

No exemplo apresentado na figura B.12, EX1, o trecho do programa compreendido entre os comandos LIST e NLST, não aparece na listagem de saída.

formulário para perfuração de cartões

"CROSS-MONTADOR"

Nome do cartão

FIGURA

B12 - Pseudos para controle de paginação

Nome do comando

EX1 NLST A

CAR B

ARM

LIST

EX2 NCTR A

PAGE

CAR CNT R

LINE

EX3 CAB

PROGRAMA DE TESTE

peg. B.39.A

Data

LINE - PULAR LINHA
PAGE - PULAR PÁGINA

Uso: Estes comandos são usados respectivamente para pedir mudança de linha e mudança de página, o que equivale a pular uma linha na listagem de saída, e avançar para a página seguinte na impressora. Tais comandos podem ser importantes para a estética da listagem de saída do programa.

Formato:

A diagram of a punched card. At the top, there are three columns of holes labeled 21, 25, and 27. Below these are two rows of characters: 'L I N E' and 'P A G E'. To the left of the first row is a wavy line indicating a blank column. To the right of the second row is a wavy line indicating another blank column. The number 30 is positioned above the 'L' in 'LINE', and the number 32 is positioned above the 'P' in 'PAGE'. The number 35 is positioned below the 'E' in 'PAGE' and extends to the right, followed by several dots representing continuation.

CNTR - LISTAR COMANDOS DE CONTROLES

NCTR - INIBIR COMANDOS DE CONTROLE

Uso: Estas "pseudos" são usadas para impedir ou permitir a listagem dos outros cartões de controle, vistos acima. A primeira pede a impressão de tais cartões, e a segunda, impede a sua impressão.

Formato:

A diagram of a punched card. At the top, there are three columns of holes labeled 21, 25, and 27. Below these are two rows of characters: 'N C T R' and 'C N T R'. To the left of the first row is a wavy line indicating a blank column. To the right of the second row is a wavy line indicating another blank column. The number 30 is positioned above the 'N' in 'NCTR', and the number 32 is positioned above the 'C' in 'CNTR'. The number 35 is positioned below the 'R' in 'NCTR' and extends to the right, followed by several dots representing continuation.

Exemplo:

A figura B.12, exemplifica o uso de tais comandos.

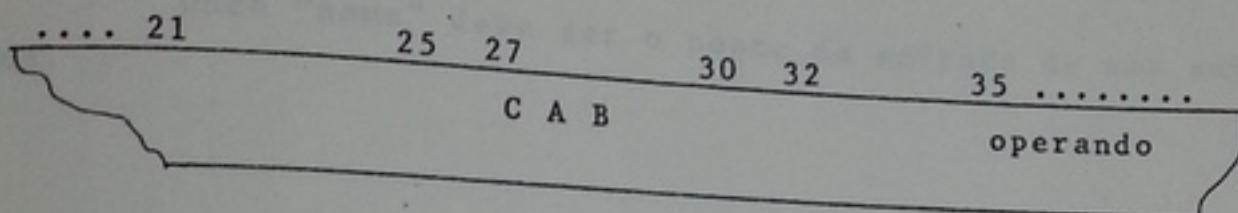
No exemplo, o comando PAGE, não será impresso na

listagem de saída, enquanto o comando LINE o será.

CAB - INICIALIZA PÁGINA COM CABEÇALHO

Uso: Esta "pseudo" faz a impressão de um cabeçalho na listagem de saída, com consequente avanço para uma nova página. O cabeçalho é dado pelo operando da "pseudo".

Formato:



Exemplo:

No exemplo apresentado na figura B.12, a listagem passará a ser impressa numa nova folha onde se lerá o cabeçalho PROGRAMA DE TESTE.

B.5. c - Definições de pontos de entrada de sub-rotinas e suas chamadas

Uma sub-rotina é um grupo de comandos que são escritos para reduzir tempo de codificação e minimizar o uso da memória principal. Quando um determinado cálculo deve ser realizado vários pontos do programa é melhor escrever uma sub-rotina que se incumba deste cálculo. Uma vez pronta e armazenada, a sub-rotina pode ser chamada pelo programa e executada. Desta maneira o grupo de instruções é escrito uma única vez e ocupa um único espaço na memória principal, ao invés de ocupar de várias vezes, este espaço no programa.

CALL - CHAMADA DE SUBROTINA

Uso: Esta "pseudo" é usada para se fazer a transferência entre um programa e uma sub-rotina, isto é, transfere o comando do programa onde é empregada, para a sub-rotina cujo nome comparece no operando da instrução.

Formato:

....	21	25	27	30	32	35
	ROTUL		C A L L			nome	
			C A L L			nome	

Onde "nome" deve ser o ponto de entrada de uma sub-rotina.

ENT - DEFINE PONTO DE ENTRADA DE SUB-ROTINA
ISS - DEFINE PONTO DE ENTRADA DE SUB-ROTINA DE INTERUPÇÃO

Uso: Estas duas "pseudos" são utilizadas para definir pontos de entrada de sub-rotinas. As primeiras definem um ponto de entrada de uma sub-rotina simples, enquanto a segunda define o ponto de entrada de uma sub-rotina de entrada e saída. O ponto de entrada é o comando onde se deseja iniciar a execução daquela sub-rotina. No caso de uma sub-rotina simples, são permitidos, no máximo 12 pontos de entrada. No caso de uma sub-rotina de entrada e saída é permitido somente um ponto de entrada.

Formato:

....	21	25	27	30	32	35
			ENT			nome	
			ISS			n, nome	

Onde "nome" especifica o ponto de entrada da sub-rotina e deve ser o rótulo de uma instrução desta. "n" corresponde ao número de um dispositivo lógico permitido para o sistema.

formulário para perfuração de cartões

Sistema *CROSS - MONTADOR*

FIGURA B13 - Sub-rotinas e programas - Nomes

Nome do cartão

	10.11	12.21	30.31	40.41	50.51	60.61	70.71	80.81
*								
EXEMPLO 1								
ENT A								
DEF C 0								
*								
EXEMPLO 2								
ISS 10, INT								
INT								
*								
EXEMPLO 3								
CALL A								
*								
EXEMPLO 4								
EQU B								
EQU D+4								
EQU 10								
EQU /10								
EQU *+3								
A								
C								
F								
G								
E								

Exemplo:

A figura B.13 mostra o uso destas instruções (EXEMPLO 1 e EXEMPLO 2).

O primeiro exemplo, EX1, mostra a definição de um ponto de entrada de uma sub-rotina simples, enquanto EX2 mostra a definição do ponto de entrada de uma sub-rotina de entrada e saída para o dispositivo lógico 10.

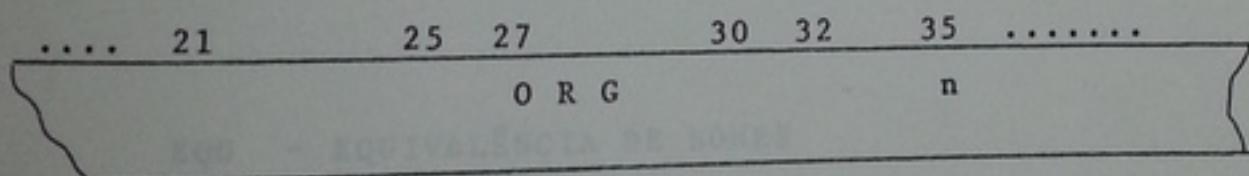
B.5.d - Comandos de origem e fim de programa

São comandos incumbidos de posicionar o início e o fim do programa, isto é, indicar a posição onde o programa deve se iniciar, e qual o último comando válido do programa fonte. Figura ainda neste grupo, a instrução de definição de nome de um programa.

ORG - DEFINE ORIGEM DE PROGRAMA

Uso: Este comando indica a origem de um programa absoluto, ou seja o endereço da primeira instrução do programa. O Contador de Instruções recebe o endereço, dado pelo operando da instrução.

Formato:

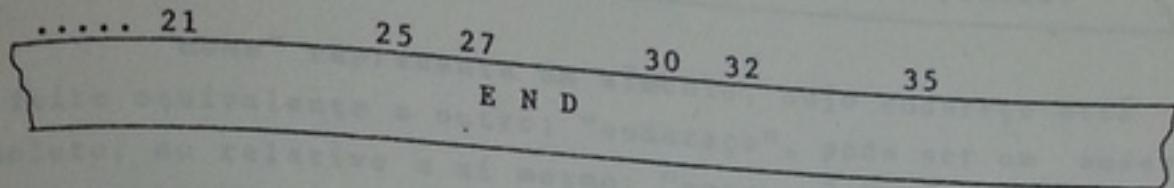


Onde "n" é um número decimal entre 0 e 4095.

END - FIM DE PROGRAMA FONTE

Uso: Esta "pseudo" marca o fim do programa fonte, e deve portanto ser o último comando do programa.

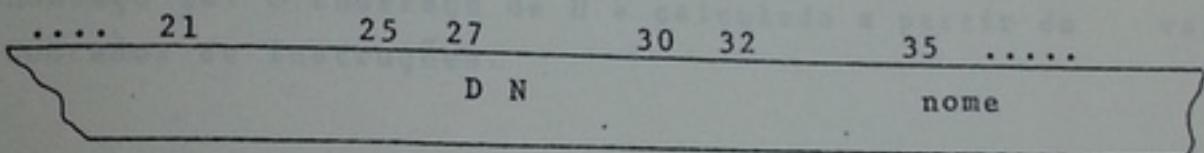
Formato:



DN - DEFINE NOME DE PROGRAMA

Uso: Esta "pseudo" é usada para definir um nome para o programa. Esta deve ter no máximo cinco caracteres, e constitui o operando da "pseudo".

Formato:



Onde "nome", é o nome que se quer definir para o programa.

B.5. e - Comandos de equivalência de endereços

EQU - EQUIVALÊNCIA DE NOMES

Uso: Este comando tem por função fazer a equivalência entre símbolos, ou atribuir a estes símbolos endereços.

Formato:

...	21	25	27	30	32	35
	nome 1		E Q U			nome 2	
	nome		E Q U			endereço	
	nome		E Q U			expressão	

"nome" representa um símbolo, cujo endereço está sendo feito equivalente a outro; "endereço", pode ser um endereço absoluto, ou relativo a si mesmo; "expressão", é uma expressão relativa a um outro símbolo, onde este símbolo já deve estar endereçado.

Exemplo:

No primeiro exemplo 4 da figura B.13, A deve ter o mesmo endereço de B. No segundo, o endereço do símbolo C é calculado a partir da expressão, porém D deve estar endereçado. No terceiro comando F recebe o endereço 10, e no quarto G recebe o endereço 16. O endereço de H é calculado a partir do valor do Contador de Instruções.

A P E N D I C E C

Utilização do Sistema

C.1. - Utilização do "Cross-Montador"	C.1
C.2. - Utilização da Biblioteca	C.2
C.2.1. - Cartões de controle da Biblioteca	C.4
C.2.2. - Esquemas de utilização da Biblioteca	C.11
C.3. - Utilização do "Cross-Carregador"	C.13
C.3.1. - Cartão de Opções	C.14
C.3.2. - Esquema de utilização do "Cross-Carregador"	C.15

O cartão de 17 X 90 milímetros que o "Cross-Montador" traz é inserido no disco onde reside a rotina na memória do computador IBM-1130 para processamento. No final desse processo, que é feito da maneira interativa, (o programa é executado em sequência, se houver erros), o programa produzido, agora já com todos os dados, é colocado em um arquivo de disco, de modo podendo ser utilizado na sua biblioteca, ou ainda composto pelo próprio usuário.

Na figura 1, ver-se uma listação da Tabela 1, de cárteis posicionados e chave 10 do painel de comando. A Tabela 1 é a tabela de rotinas existentes, quando não produzida, o resultado do programa é dar um erro. No caso da figura 1, ao término do programa se dar um erro. No caso de não existir a Tabela 1, o programa é encarregado de gerar a sua estrutura, momento a listagem, que deixa a de rotinas vazia, ou seja, dependendo da porta onde portaria o erro.

A P E N D I C E

c

Utilização do Sistema

C.1. - Utilização do "Cross-Montador"

Para se utilizar o "Cross-Montador", na compilação de um programa, deve-se colocar o "deck" de cartões a compilar, após um cartão de chamada do programa:

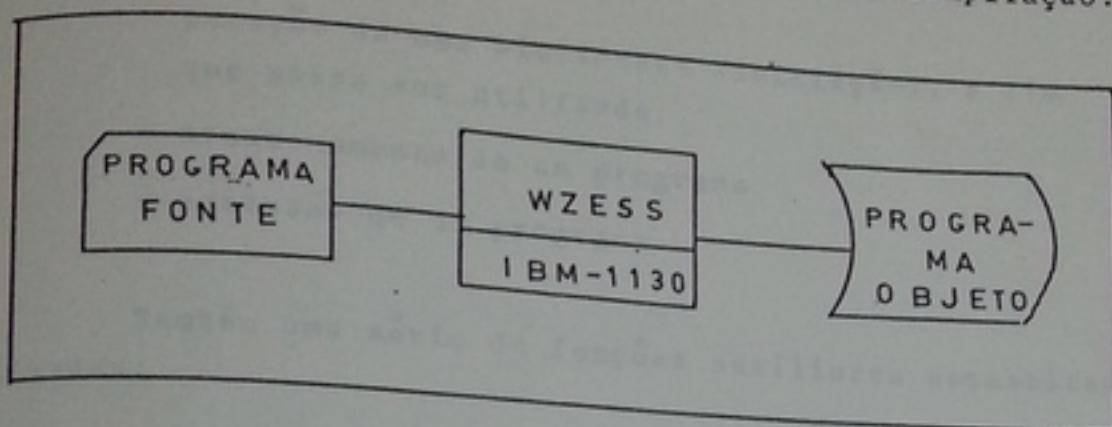
coluna 1

```
// XEQ WZESS
      |
      programa a compilar
// *
```

O cartão de // XEQ WZESS faz com que o "Cross-Montador" seja trazido do disco onde reside, e colocado na memória do computador IBM-1130 para processamento. No fim da execução, se esta se der de maneira satisfatória, (o programa a compilar não apresentar erros), o programa produzido, agora na forma relocável, será colocado em um arquivo de disco, de onde poderá ser guardado em uma biblioteca, ou então composto pelo programa "Cross-Carregador".

Se for desejável, ter-se uma listagem da Tabela de Símbolos, deve-se posicionar a chave 10 do painel do computador IBM-1130. A Tabela de Símbolos, entretanto, somente será produzida, se o término do programa se der sem erros. No caso de ocorrer um erro, o programa é encerrado de maneira a que seja impressa, somente a listagem, quer seja a da primeira fase, ou a da segunda, dependendo do ponto onde ocorreu o erro.

O esquema abaixo mostra o processo de compilação:



A figura C.1., mostra um exemplo do processo, apresentando os cartões fonte, e de controle, bem como uma saída da tabela de símbolos.

C.2. - Utilização da Biblioteca

A Biblioteca é configurada com dois propósitos:

- armazenar programas em formato relocável, que podem ser programas principais ou sub-rotinas.
- armazenar programas já compostos, isto é, em formato absoluto.

As bibliotecas podem ser do sistema ou do usuário, e são definidas para uso de um programa, através de cartões de FILES. Uma biblioteca somente pode ser utilizada após ter sido iniciada convenientemente, o que é feito através de um dos programas do sistema de biblioteca. As bibliotecas residem sempre em área do usuário.

As funções básicas do Sistema de Biblioteca são:

- geração de uma biblioteca (iniciação), a fim de que possa ser utilizada.
- armazenamento de um programa
- exclusão de um programa

Também uma série de funções auxiliares necessitam ser realizadas:

- compactação da biblioteca, uma vez que, programas excluídos deixam espaços desperdiçados no disco.
- extensão do índice, quando o número de programas cresce, e este não pode mais conter informações sobre eles.
- listagem da biblioteca, a fim de se fazer a manutenção da mesma.
- preparação do arquivo de disco, contendo um programa absoluto, para posterior perfuração em cartões.

Uma série de comandos são permitidos para a utilização da biblioteca. O "deck" de cartões contendo comandos deve ser colocado entre um cartão do // XEQ e um de // *.

A sequência correta destes cartões é a seguinte:

Coluna 1 17

// XEQ WZW 1
*FILES(1,N08),(11,WZN11),(12, BBB12),(13, BBB13)

cartões de comandos para a biblioteca

// *

C.2.1. - Cartões de controle da biblioteca

Os cartões de controle da biblioteca são apresentados a seguir, mostrando a sua sintaxe:

Cartão de Geração da Biblioteca

Este comando inicia uma biblioteca, para que esta possa ser utilizada a seguir. A iniciação da biblioteca é feita uma única vez.

Coluna	Conteúdo	Função
1	*BIBL	Identifica o comando como sendo de geração
6 - 7	branco	
8	BBBBBB	Seis caracteres alfanuméricos, identificando a biblioteca.
14 - 15	branco	
16	PPPPP	Cinco caracteres alfanuméricos, identificando o nome do programa
21	branco	
22 - 80	n	Número de "disk blocks", que deverá conter a biblioteca. (1 "disk block" = 20 palavras).
(em formato livre)	ns	Número de setores que deverão ser reservados para índice. (1 setor = 16 "disk-blocks")
	nd	Número de arquivo lógico onde deverá ser gerada a biblioteca (12 ou 13). Este número deverá ser coincidente com o que aparecer no cartão de FILES

Exemplo:

1
*GERA BIBL12

60 1 12

Este cartão fará a geração de uma biblioteca de nome BIBL12, contendo 60 "disk blocks", com um setor (16 "disk-blocks") reservado para índice, e cujo número lógico será 12. Se se considerar o cartão de FILES apresentado na página anterior, esta biblioteca será gerada no arquivo, referido externamente como BBB12.

Cartão de Inclusão de um programa na biblioteca

Este comando inclui um programa na biblioteca, o qual pode estar em formato relocável ou absoluto. O formato do comando é o seguinte:

Coluna	Conteúdo	Função
1	*INCL	Identifica o comando como sendo de inclusão
6 - 7	branco	
8	BBBBBB	Seis caracteres alfanuméricos, identificando a biblioteca
14 - 15	branco	
16	PPPPP	cinco caracteres alfanuméricos, identificando o nome com que o programa deve ser incluído. Se for uma sub-rotina o nome deve ser "entry point" válido da sub-rotina. Neste caso todos os demais "entry points" são incluídos no índice da biblioteca.
21	branco	

Coluna	Conteúdo	Função
22 - 80	n	<p>Se "n" for o caracter "A", então o programa se encontra em formato <u>absoluto</u>, já composto pelo "Cross-Carregador".</p> <p>Se "n" for qualquer outro caracter, o formato é <u>relocável</u>, isto é, ainda deverá ser composto pelo "Cross-Carregador".</p>

Cartão de exclusão de programa da Biblioteca

Este comando, exclui de uma biblioteca determinada, um programa especificado. O formato deste comando é o seguinte:

Coluna	Conteúdo	Função
1	* EXCL	Identifica o comando, como sendo de exclusão
6 - 7	branco	
8	BBBBBB	Seis caracteres alfanuméricos, identificando a biblioteca
14 - 15	branco	
16	PPPPP	Cinco caracteres alfanuméricos, identificando o programa a ser excluído. Se se tratar de uma sub-rotina com múltiplos "entry-points", todos eles serão excluídos.
20 - 80		não utilizados

Cartão de pedido de extensão do índice da biblioteca

Este comando faz a extensão do índice da biblioteca de um número qualquer de setores (desde que possível), determinado pelo comando de cartão.

O formato do comando é o seguinte:

Coluna	Conteúdo	Função
1	*EXTS	Identifica o comando como sendo de extensão do índice.
6 - 7	branco	
8	BBBBBB	Biblioteca, cujo índice deve ser <u>ex</u> tendido
14 - 20	branco	não utilizadas
21	branco	
22 - 80 (em forma to livre)	n	Número de setores de que deve ser <u>ex</u> tendido o índice da biblioteca

A extensão do índice, pode ainda ser feita através do comando *INCL.

Neste caso ela é feita automaticamente, no caso de se fazer necessária à inclusão do programa, pedida.

Cartão de pedido de compactação da biblioteca

Este comando faz a compactação dos programas na biblioteca, isto é, elimina os espaços vazios do disco, na área de armazenamento de programas. O seu formato é o seguinte:

Coluna	Conteúdo	Função
1	* COMP	Identifica este comando como sendo de compactação
6 - 7	branco	
8	BBBBBB	Nome da biblioteca a compactar
14 - 80		Não utilizadas

A compactação da biblioteca é feita automaticamente no caso de um comando * INCL, quando se faz necessária à inclusão de um programa.

Cartão de pedido de perfuração de cartões

Este comando permite fazer-se a perfuração de um programa em formato absoluto que se encontre armazenado na biblioteca. O seu formato é o seguinte:

Coluna	Conteúdo	Função
1	* FIT	Identifica este comando como sendo de pedido de perfuração de cartões
6 - 7	branco	
8	BBBBB	Seis caracteres alfanuméricos, identificando a biblioteca que contém o programa
14 - 15	branco	
16	PPPPP	Nome do programa a ser perfurado em cartões
21 - 80	não	não utilizados

Este comando deve ser seguido de uma série de cartões em branco, nos quais será feita a perfuração do programa.

Cartões de pedido de carregamento de programa

Este comando pede o carregamento de um programa, que se encontra ou no arquivo da biblioteca, ou então no arquivo de saída do "Cross-Montador". A finalidade do comando é identificar o programa a ser composto, e as opções requisitadas. Seu formato é o seguinte:

Coluna	Conteúdo	Função
1	*LOAD	Identifica este comando como sendo de pedido de composição do programa
6 - 7	branco	
8	BBBBBB	Se comparecer no comando, indica em qual biblioteca se encontra o programa. Se for deixado em branco o programa está no arquivo de saída do "Cross-Montador".
14 - 15		
16	PPPPP	Se comparecer, indica qual programa da Biblioteca BBBBB deve ser composto.
21	branco	
22 - 80	M C n IIIII	Se comparecer no comando, indica pedido de listagem do mapa de carregamento. Se comparecer no comando, indica pedido de saída de cartões perfurados. É obrigatório o comparecimento, e indica o número de bibliotecas à disposição do "Cross-Carregador" Se 2, indica duas bibliotecas Se 1, indica uma biblioteca Se comparecer no comando, indica qual a rotina de interrupção a ser utilizada pelo sistema.

Este comando deve ser seguido de cartões em branco, para serem utilizados na perfuração, se a opção "C" for pedida.

Cartão de pedido de listagem do índice da biblioteca

Este cartão requisita a listagem do índice da biblioteca, isto é, os nomes, tipos, tamanhos e endereços dos programas armazenados na biblioteca. O seu formato é o seguinte:

Coluna	Conteúdo	Função
LIST	*LIST	Identifica este comando como sendo de pedido de listagem do índice.
6 - 7	branco	
8	BBBBBB	Identificam estes seis caracteres alfanuméricos, a biblioteca, cujo índice deve ser listado.
14 a 80		Não utilizados

Cartão de encerramento do programa de biblioteca

Este comando encerra um "deck" de cartões de biblioteca. Entretanto não deve ser usado, quando os programas de perfuração de cartões são especificados. O seu formato é o seguinte:

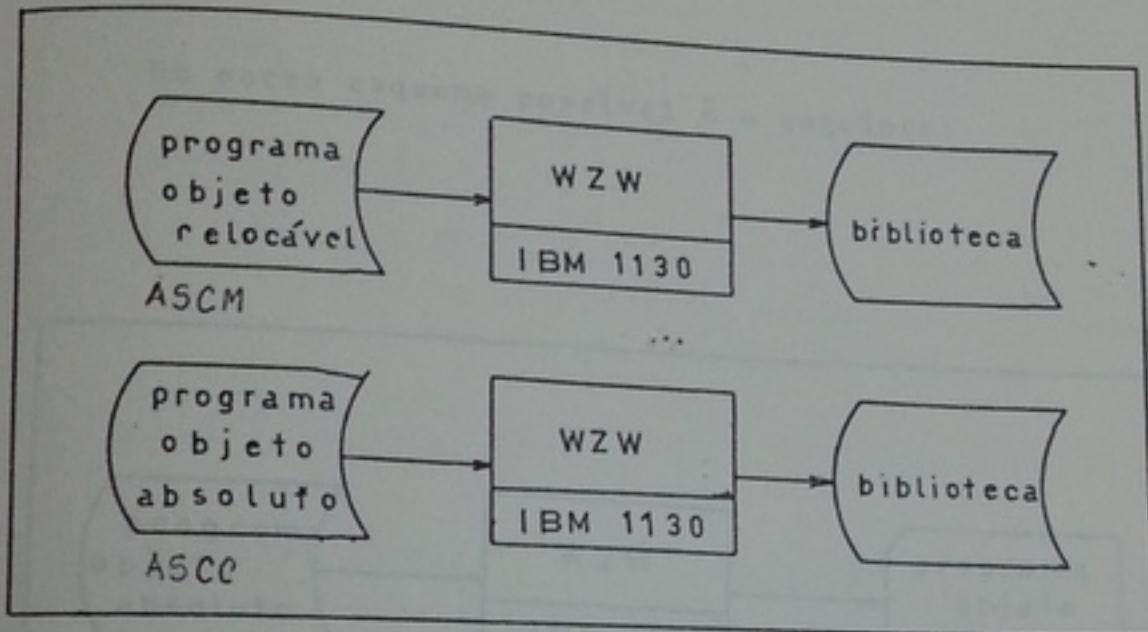
Coluna	Conteúdo	Função
1	*FIM	Identifica o comando como de "fim" não utilizados.

A figura C.2., exemplifica o processo.

C.2.2. - Esquemas de utilização da biblioteca

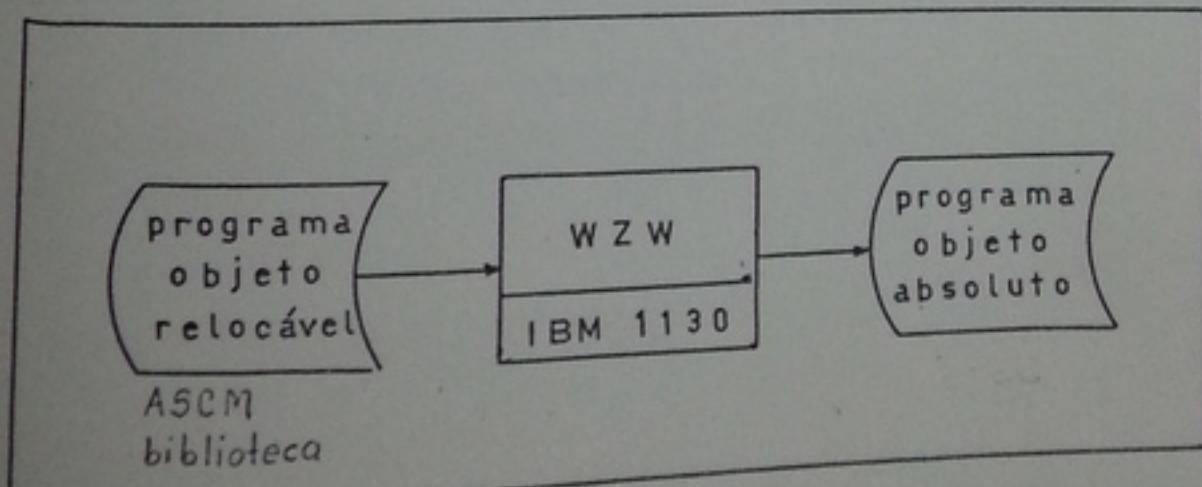
Os esquemas apresentados abaixo, mostram algumas utilizações da biblioteca.

ARMAZENAMENTO DE UM PROGRAMA



O esquema apresentado, mostra a inclusão de um programa relocável, na Biblioteca. Este programa é incluído a partir do arquivo de saída do Programa "Cross-Montador". O segundo esquema, mostra a inclusão de um programa absoluto, já composto pelo "Cross-Carregador", a partir do arquivo de saída do "Cross-Carregador".

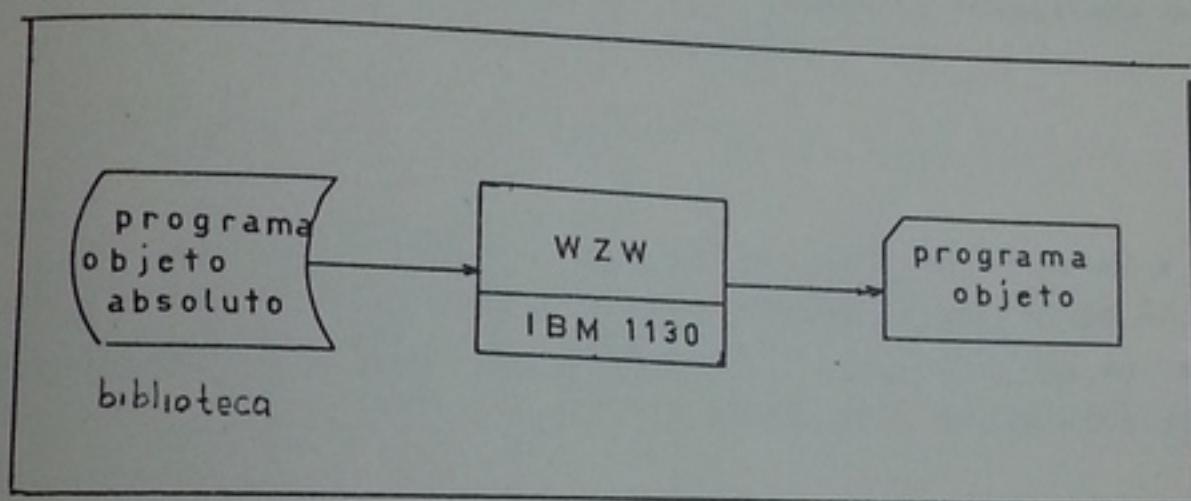
COMPOSIÇÃO DE UM PROGRAMA



Neste esquema, através deste sistema, é possível utilizar-se o "Cross-Carregador" para compor um programa relocável, que se encontra na Biblioteca, ou no arquivo de saída do "Cross-Carregador".

PERFURAÇÃO DE UM PROGRAMA QUE SE ENCONTRA NA BIBLIOTECA

Um outro esquema possível é o seguinte:



Neste último esquema, o programa de biblioteca faz a geração dos cartões de saída, a partir do arquivo de biblioteca, o qual continha um programa absoluto (composto pelo "Cross-Carregador").

C.3. - Utilização do "Cross-Carregador"

Um programa pode ser composto pelo "Cross-Carregador", a partir da Biblioteca, ou a partir do Arquivo de Saída do "Cross-Montador". As opções de carregamento são indicadas por um cartão de opções, o qual deve seguir os cartões de controle.

Coluna	1	17
	// XEQ WZV	1
	*FILES(1,N08),(11,WZN11),(14,WZNOQ),(12,bbb12),(13,bbb13)	
	----- cartão de opções	

	cartões em branco, se solicitada saída por cartões	

	// *	

O cartão de FILES, especifica quais os arquivos a serem utilizados na composição de um programa. O cartão de // XEQ WZV pede que o programa "Cross-Carregador" seja colocado na memória do computador para execução. Os arquivos a serem utilizados são:

N08 - número lógico 1 - arquivo de saída do "Cross-Montador", onde ficam os programas compilados na forma objeto relocável.

WZN11 - número lógico 11 - arquivo de saída do "Cross-Carregador", no caso de não ser solicitada a saída através de cartões perfurados.

WZNOQ - número lógico 14 - arquivo que contém os erros do "Cross-Carregador".

BBB12 e BBB13 - arquivos de números lógicos 12 e 13
são arquivos de Biblioteca, que deverão ser con
sultados na montagem do programa. O nome externo
destes arquivos (no caso BBB12 e BBB13), pode ser
qualquer, desde que se obedeça aos números lógi
cos na locação dos mesmos. O arquivo de número
lógico 12 é sempre pesquisado primeiro numa bus
ca, sendo portanto a biblioteca principal do sis
tema. No caso de um programa, que não necessita de
nenhuma rotina compilada separadamente, estes dois
arquivos não necessitam ser referenciados no car
tão de FILES.

O arquivo de número lógico 11, também pode deixar de
aparecer no cartão FILES, sempre que a saída for feita através
de cartões perfurados.

C.3.1. - Cartão de opções

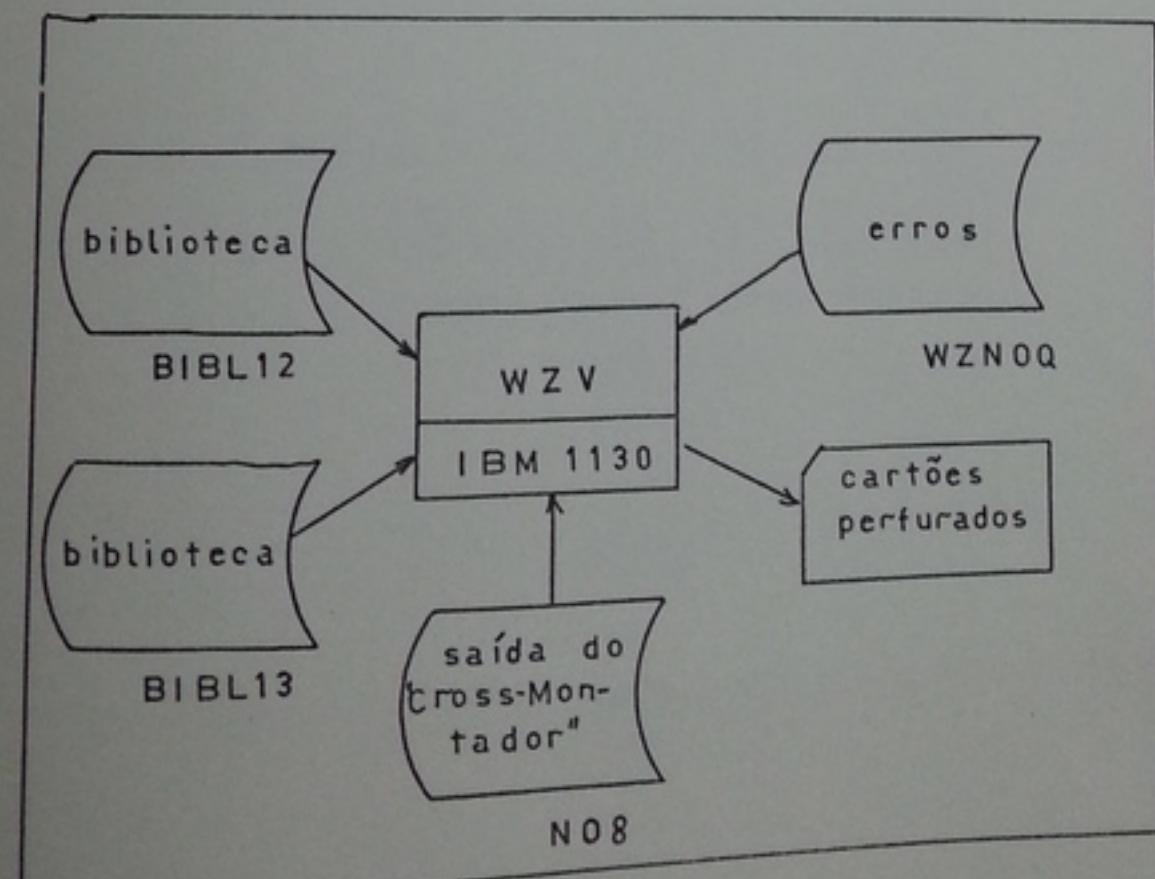
Este cartão tem o seguinte formato:

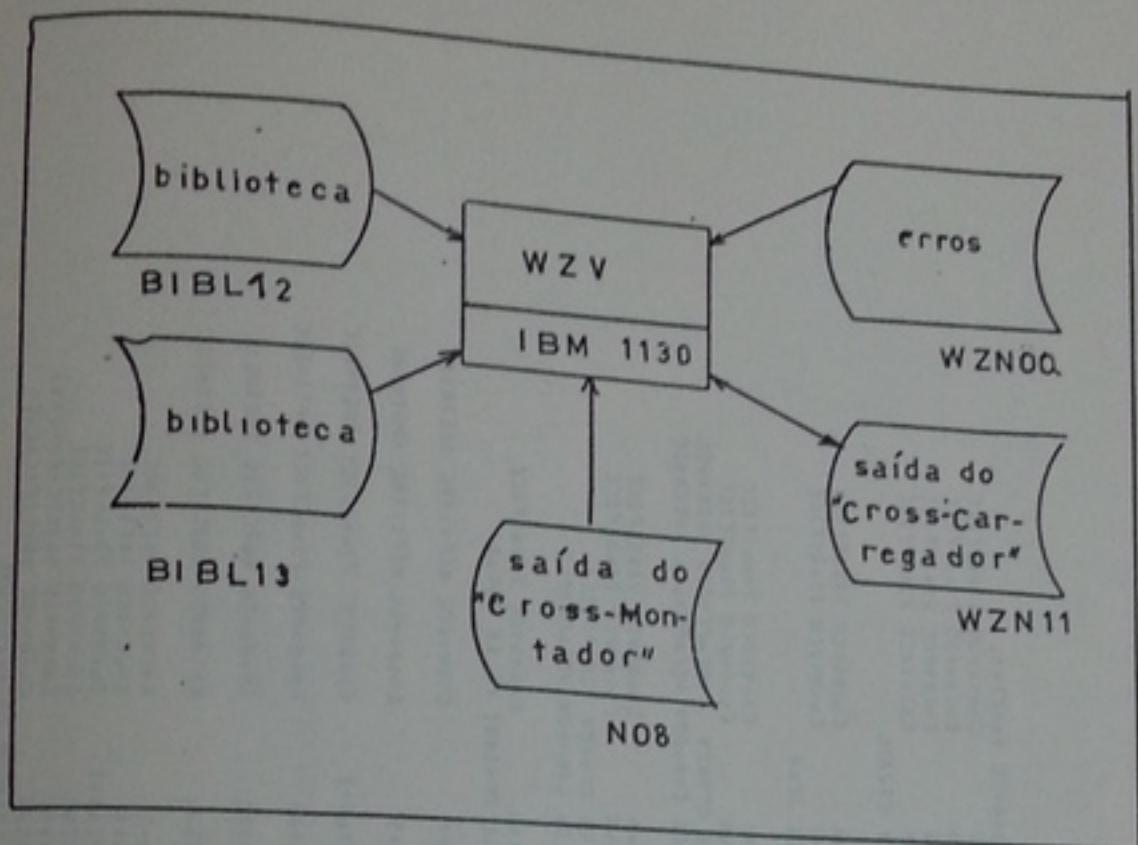
Coluna	Conteúdo	Função
1 - 7	branco	
8	BBBBBB	Nome da biblioteca que contém o <u>pro</u> <u>grama</u> a ser composto. (deve ser <u>deixa</u> <u>do</u> em branco se o programa se <u>encon</u> <u>trar</u> no arquivo de saída do "Cross - Montador").
14 - 15	branco	
16	PPPPPP	Se comparecer, indica qual programa da biblioteca BBBB deve ser composto.
21	branco	
22 - 80	M	Se comparecer no comando, indica pedi do de listagem do mapa de carregamen to.
	C	Se comparecer, indica pedido de saída através de cartões.

Coluna	Conteúdo	Função
	n IIIII BIBL13	Indica o número de Bibliotecas que serão utilizadas pelo programa (1, indica uma biblioteca sendo usada; a de número lógico 12, e 2, indica duas bibliotecas sendo usadas, as de números lógicos 12 e 13. Neste caso a de número 12 é pesquisada primeiro). Se comparecer no comando, indica a rotina de interrupção, que deverá ser usada pelo programa que está sendo composto.

C.3.2.-Esquema de utilização do "Cross-Carregador"

São apresentados a seguir dois esquemas de utilização do "Cross-Carregador". O primeiro produz uma saída em disco, enquanto o segundo gera uma saída em cartões. O programa a ser composto pode se encontrar tanto no arquivo de saída do "Cross-Montador", como no Sistema de Bibliotecas. As rotinas a serem inseridas devem se encontrar no Sistema de Bibliotecas.





-228-

CC	CI	INSTRUCAO		CCPAGACC	
1			FAT	VFT	
2	0709	2306 7240	VEZ	CALL TAXI	
4			*	INSTALACAO DE PULA PARA ENCHEREC CC CCPAGACC E	
5	0002	F4CE	*	GLANDA E CICATRIZ DE INSTALACAO	
6	0004	F410	INIC	OPERANDO ABSOLUTICO(FEX)	
7	0006	F042	PUG 1230	OPERANDO ABSOLUTICO(FEX)	
8	0009	F043	PUG /A10	OPERANDO SIMPLES	
9	001A	F70C	PUG SIMB	OPERANDO SIMPLES	
10	001B	F010	PUG S1W8+1	OPERANDO SIMPLES	
11	001C	F010	PUG ++2	OPERANDO RELATIVO	
12	001E	3F	PUG +-1	OPERANDO RELATIVO	
13	001F	F4CE	PUG 1 1230	OPERANDO ABSOLUTO INCIRETC	
14	0012	F410	PUG 1 /A10	OPERANDO ABSOLUTO INCIRETC	
15	0014	9F	PUG 1 S1W8	OPERANDO SIMPLES INCIRETC	
16	0015	F042	PUG 1 S1W8-1	OPERANDO SIMPLES INCIRETC	
17	0017	3F	PUG 1 ++2	OPERANDO RELATIVO INCIRETC	
18	0018	F041	PUG 1 +-1	OPERANDO RELATIVO INCIRETC	
19			*	INSTALACAO DE SISTEMA EM SALTA	
20	0020	F042	SUS SIMP	OPERANDO SIMPLES	
21			*	INSTALACAO DE CARREGA ACUMULACAO	
22			*	INSTALACAO DE GUARDA ACUMULACAO	
23	0022	4042	CAR SIMP	OPERANDO SIMPLES	
24	0022	4042	ANV SIMP	OPERANDO SIMPLES	
25	0024	2042	*	INSTALACAO DE CARREGA ACUMULACAO INCXACC	
26			*	INSTALACAO DE GLARCA ACUMULACAO INCXACC	
27			*	INSTALACAO DE GUARDA ACUMULACAO INCXACC	
28	0026	5042	CARX SIMP	OPERANDO SIMPLES	
29	0026	5042	ANVX SIMP	OPERANDO SIMPLES	
30	0028	3042	*	INSTALACAO DE SCRA	
31			*	INSTALACAO DE SCRA	
32			*	INSTALACAO DE LESVIO	
33	002A	6042	SCP SIMP	OPERANDO SIMPLES	
34	002C	7042	SCWX SIMP	OPERANDO SIMPLES	
35			*	INSTALACAO DE LESVIO	
36			*	PLAX SIMP	OPERANDO SIMPLES
37	002E	1042	PLAN SIMP	OPERANDO SIMPLES	
38	0030	4042	PLAZ SIMP	OPERANDO SIMPLES	
39	0132	F042	PLA SIMP	OPERANDO SIMPLES	
40	0034	0042	*	ENDERECAMENTO INFELATC	
41			*		

Figura C.1.: - Saída impressa do programa "Cross-Montador"

CC	C1	INSTRUCAO	C2	INSTRUCAO
43			44	INSTRUÇÕES DE OPERAÇÕES LOGICAS - NAND E XCR
45	UN36	E400	45	NAND
46	UN32	U470	46	SEI OPERAÇÕES
47	UN34	U4FF	47	OPERANDO DECIMAL
48			48	OPERANDO HEXADECIMAL
49	U03C	U211	49	XCR /11 OPERAÇÕES HEXADECIMAL
50			50	INSTRUÇÕES CF SCV1 CCP OPERAÇÕES INICIAIS
51			51	SCV1 /10 OPERAÇÕES INICIAIS
52	U93E	0810	52	INSTRUÇÕES CF CARREGA ACUMULADOR INICIAIS
53			53	CAR1 /1G OPERAÇÕES INICIAIS
54	U040	0810	54	INSTRUÇÕES CF CARREGA ACUMULADOR ESPECIAIS
55			55	CAR1 /1G OPERAÇÕES ESPECIAIS
56			56	R EQU 15C
57			57	R EQU 15C
58			58	SIWA BLCC IN
59	U182		59	ASC ASCD.
60	U04C	41	60	
60	U04C	42	60	
60	U04E	43	60	
61	U04F	44	61	ETL EHC .ACC.
61	U170	C1	61	
61	U051	C2	61	
61	U052	C3	61	
61	U053	C4	61	
62			62	END VFT
63			63	

Figura C.1.: - Saída impressa do programa "Cross-Montador"
(continuação)

TABELA DE SÍMBOLOS

SÍMBOLO	ENDEREÇO
VIZ	0000
INIC	0012
SIMP	0042
R	0096
BFL	0054

1	0010	0000	0054	0000	0033	0000	0000	CCCC	FFFF
11	0010	0000	0040	C004	2306	7240	F4CE	CCCC	F410
21	0010	F042	0000	F043	0003	F00C	0000	F01P	CCCC
31	0F10	0010	0000	0000	9E00	FA10	0000	FC00	CCCC
41	4E10	4104	F041	0000	9E10	F01D	0000	FF00	FC42
51	4E44	E'42	0000	4042	0003	2042	0003	FF00	CCCC
51	3042	0000	6042	0000	7042	0000	1042	0042	CCCC
71	0010	8042	0000	0042	0070	C400	0000	0000	4444
91	0010	FF10	1100	D610	1000	0000	0000	0000	A042
71	0010	0000	4100	4200	4300	4400	C100	C200	CCCC
131	0054	0000	2423	5704	0000	0000	0000	0000	CCCC

OBSERVAÇÃO : Abaixo da tabela de símbolos a figura apresenta a configuração das 12 primeiras palavras do arquivo de disco : a saída de disco correspondente ao programa gerado.

Figura C.1.: Saída impressa do programa "Cross-Montador"
(continuação)

ERRATA

PÁGINA

TEXTO

CORREÇÃO

I.2	A quantidade de memória disponível pode variar segundo a versão em apreço e o <u>modelo</u> do computador.	modelo
I.6	Quanto ao número de operandos, as instruções podem ser: sem operandos, de um, dois ou <u>tres</u> operandos.	três
I.6	De uma maneira geral podem ser divididas em <u>tres</u> grupos: , deve-se levar em consideração não apenas a linguagem, mas <u>tambem</u> o compilador existente para esta linguagem,	três
I.19		também
I.23	Em tais casos estes usuários devem se contentar com os demais recursos <u>apresentador</u> pelo minicomputador e esquecer o "Cross-Montador".	apresentados
II.5	Por outro lado, o uso de endereçamentos simbólicos relativos é de grande importância para a <u>linguagem</u> ;.....	linguagem;
I.23	A maior parte dos "Cross-Montadores existente na prática estão <u>codificados</u> em linguagens de alto nível,.....	codificados
II.7	...de uma única "pseudo". Neste caso, o nomede uma única "pseudo". Pode ainda ser chamada pela colocação de seu nome no campo de operando. Neste caso, o nome
III.4	neste programa: o caso dos símbolos <u>externos</u> : neste caso, pode-se	externos. Neste caso,..
III.15	Os operandos absolutos são analisados na segunda fase se de-	

BIBLIOTECA
Dept. Eng. ELETRICIDADE
E. P. U. S. P.

O E D U C A O

PÁGINA

	TEXTO	CORREÇÃO
III.14	<u>terminando-se</u> neste	determinando-se neste....
III.16	Fig. III.3.2	Fig. III.3.1
III.19	Figura III.3.1	Figura III.3.2
III.27	INCT(end):..... Os grupos de tratamento para as "pseudos" estão apresenta- dos nas tabelas.	INST(end): "pseudos" estão apresenta- dos nas tabelas corres- pondente(figura III.9.2).
III.43 e		Estão com os números tro- cados.
III.44		
III.43 (a- tual)	III.4.h-Processamento de EBC e ASC	III.4.i - Processamento de EBC e ASC
III.44 (a- tual)	III.4.i-Processamento de ISS	III.4.h-Processamento de ISS
III.50	A figura III.4.12 mostra o tratamento dado a estas "pse- udos" em ambas as fases.	A figura III.4.12 mostra o tratamento dado a estas "pseudos" no segundo pas- so.
III.52	, é não se analisar as par- tes de um comando, na Fase de Geração, <u>que já forma analisa</u> dos na Fase de Locação.	que já foram analisados...
III.55	...numa forma compactada, em duas palavras do computador <u>hospedeiro ("name code")</u> .	hospedeiro.
III.56	Figura III.9.1	Figura III.9.2
III.57	Figura III.9.2	Figura III.9.1
III.62	, onde se encontram as instru- ções geradas, se compõe de três dígitos hexadecimais (000 a FFF),	de 4 dígitos hexadecimais (0000 a 0FFF)
IV.4	Uma vez determinada a posição dos pontos de entrada chama- dos, <u>sobrepor nestas porções</u> os endereços corretos.....	posições
IV.5	Neste caso, um dos pontos de <u>entrada</u> recebe indicador pa- ra carga,.....	<u>entrada</u>

BIBLIOTECA
Dept. Eng. ELETRICIDADE
E. P. U. S. P.

0 0 0 0 0 0 0

PÁGINA

IV.5

TEXTO

O programador deve fornecer essa informação ao programa "Cross-Carregador".

CORREÇÃO

IV.9

Esta é composta de um cabeçalho cuja configuração é mostrada na figura por um conjunto de "disk blocks".....

"Cross-Carregador".

IV.10

,os quais são utilizados pelo "Cross-Carregador", na montagem de um programa relocável.

montagem

A.8

Máquinas deste tipo apresentam na instrução o endereço explícito de um operando

explícito

B.2

O modo indireto é proibido....

proibido

B.5

No caso das instruções CAR e CARX os operandos podem ser endereços imbutidos, absolutos ou relativos.....

simbólicos, absolutos,

.....

B.13

São funções de controle do dispositivo, conforme.....

controle

B.22

A instrução SLVM, no caso da condição testa ser verdadeira, faz a complementação do valor "flip-flop".

do valor do "flip-flop"

B.44

Formato

21	25	27	30	32	35	39
----	----	----	----	----	----	----

E N D

21	25	27	30	35
----	----	----	----	----

E N D

nome

onde "nome" é o rótulo do ponto de execução do programa/rotina

C.2

A figura C.1, mostra um exemplo do processo, apresentando os cartões fonte, e de controle, bem como uma saída da tabela de símbolos.

.....
do processo, apresentando a listagem dos cartões fonte e código gerado, bem como uma saída da tabela de símbolos.

C.3

*FILES(1,NO8),(11,WZN11),(12,BBB12),(13,BBB13),....
*FILES(1,NO8),(11,WZN11),(12,BBB12),(13,BBB13),(14,WZN00)

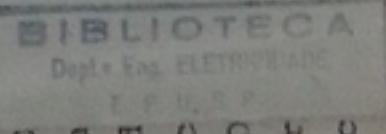


Figura C2 - Exemplo de uma saída do Cross-Carregador.
programa compilado - ROCA -
O programa requisita 2 subrotinas : CAD e

ESTUDOS DO CTI

O conjunto é alto, nos trai de controle, cartão de op os dois cart res, mostram cartões prod

CENANDO

X	SCM	8
	CAR	
	ARM	
	SCM	C+1
	PLA	X
Y	CARX	C
	TRI	
	TRE	
	CAR	1
	CALL	RCC
	CALL	CAD
	PARE	
ABCD	CCN	100
C	DEF0	10
A	DEF0	1
B	DEF0	C
	END	X

bloco de uma saída do Cross-Carregador.

rama compilado - ROCA -

ograma requisita 2 subrotinas : CAD e ROC.

F1 para C2 -
saída do Cr

O conjunto
alto, nostr,
de controle
cartão de O
Os dois car
res, mostra
cartões pro

```
// XEQ LSTDR      1
*FILESET8,WZNCS1
```

```
1      LISTAGEM DA SAIDA DE DISCO DO CROSS-MONTADOR
11     0000    0000    0200    0018    0064    0003    0000
11     9540    0000    0200    0010    4444    6016    0000
21     0000    0000    0000    4400    0000    0000    5015
31     4001    6016    0000    1958    3000    C304    4000
41     0000    0000    CC00    4040    4040    4040    4040
51     4040    0018    0000    4040    4040    4040    4040
61     4040    4040    4040    4040    4040    4040    4040
// EJECT
```

Figura C2 - Saída em disco do Programa ROCA .

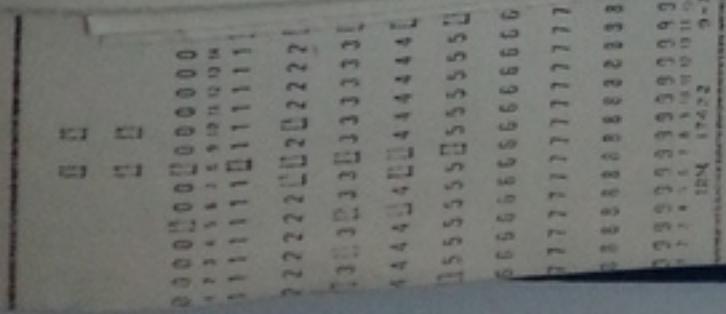


Figura C2 -
saída do CR

O conjunto é
alto, mostra
de controle,
cartão de CE
Os dois cart
ões, mostram
cartões proc

DE CRCSS-MONTADOR

0018	0064	0003	0000	0003	0000	1404
0010	4444	6016	0000	4017	0000	2015
4400	0000	0000	5015	0000	9E00	9900
1958	3000	0304	4000	9000	0400	0100
4040	4040	4040	4040	4040	4040	4040
4040	4040	4040	4040	4040	4040	4040

disco do Programa ROCA.

Figura C2 -
saída do Cr.

O conjunto é
alto, mostra
de controle,
cartão de or
Os dois car
res, mostra
cartões pro

LABORATORIO DE SISTEMAS DIGITAIS

CC	CI	INSTRUCAO
1		
2	0000	000A
3	0002	0000
4	0004	002A
5	0006	
6	0026	32
7	0027	10
8	0028	702C
9	002A	0004
10	002C	0004
11		

	COMANDO
ENT	MART
DEFE	10
DEFE	4-4
MART	PLA
RTI	RTI
BLCC	32
DEFC	/32
DEFC	16
SCMX	A
RTI	PLA
A	PLA
	MART
	END

Figura C2 - (continuação) - Exemplo de uma saída
programa compilado - MART-
Esta subrotina é considerada como se

Figura C2 -
saída do CR

O conjunto (alto, no topo, de controle cartão de O Os dois carres, mostra cartões pro

CENANCC		
MART	ENT	NART
	DEFB	10
	DEFB	4-9
	PLA	RTI
	BLOC	32
	DEFB	132
	DEFB	16
	SCMX	A
RTI	PLA	NART
A	PLA	NART
	END	

(continuação) - Exemplo de uma saída do Cross-carregador.
programa compilado - MART-
Esta subrotina é considerada como sendo de interrupção para ROCA-

Figura C2 -
saída do CR

O conjunto é
alto, mostrando
de controle,
cartão de op.
Os dois cartões
res, mostram
cartões pro-

LABORATORIO DE SISTEMAS DIGITAIS

-234-

CC	CI	INSTRUCAO	COMANDO
1			
2			
3	0000	00	ENT CAD
4	0001	10	ENT MCR
5	0002	10	MOR 0
6			CAD DEF C /10
7	0003	C580	A DEF C /10
8	0005	C540	X EQU 13
9			SAI 5
			ENTR 15
			END

Figura C2 - (continuação) - Exemplo de um programa compilado - CAD
Esta subrotina é chamada pelo p

Figura C2 -
saída do Cr

O conjunto, alto, mostra de controle, cartão de op Os dois cart res, mostrar cartões prod

```
CCNANDO
ENT CAD
ENT NCR
MON DEF0 0
CA0 DEF0 /10
A DEF0 /10
X EQU 13
SAI 5
ENTR 5
END
```

2 - (continuação) - Exemplo de uma saída do Cross-Carregador:
programa compilado - CAD
Esta subrotina é chamada pelo programa ROCA

PI para C2 -
saída do CR:

O conjunto é
alto, mostra
de controle,
cartão de op
Os dois cart
res, mostrai
cartões proc

```
// XEQ LSTDR    1
```

```
*FILE$18,WZNQ8)
```

```
1      LISTAGEM DA SAÍDA DE DISCO DO CROSS-MONTADOR
11     0000    0000    0400    002E    0000    0003    0000
21     98C0    0004    0000    0009    0040    CCC8    0000
31     0000    0026    0008    0444    3200    1000    702C
41     0004    0000    002E    0000    4040    4040    4040
// XEQ WZK    1
```

```
*FILE$12,BBB12),(1,WZNQ8)
```

Figura C2 - (continuação) - Exemplo de uma saída do cr
saída de disco do programa mART.

0000000000000000
1111111111111111
2222222222222222
3333333333333333
4444444444444444
5555555555555555
6666666666666666
7777777777777777
8888888888888888
9999999999999999
1111111111111111
124 17422 9-7

Figura C2 -
saída do Cr.

O conjunto
alto, mostra
de controle,
cartão de op
Os dois cart
res, mostran
cartões prod

CROSS-MONTADOR

02E	0000	0003	0000	0002	0000	1405
009	0040	0004	0000	0000	0000	0026
444	3200	1000	7020	0000	0004	0000
000	4040	4040	4040	4040	4040	4040

- Exemplo de uma saída do cross-Carregador
do programa mART.

// XEQ LSTDR 1

*FILESET8,NZNC8)

```

1      LISTAGEM DA SAIDA DE DISCO DO CROSS-MONTADOR
11     0000    0000    0000    0000    0000    0000    0000
21     9000    0000    0400    0008    0000    0006    0000
31     1000    0000    0304    4000    0001    0000    0004
41     4040    C500    8000    C500    4000    0008    0000
41     4040    4040    4040    4040    4040    4040    4040
// EJECT

```

Figura C2 - (continuação) - Exemplo de uma saída de disco do programa EAD

Figura C2 -
saída do Cr.

O conjunto é
alto, mostrando
de controle,
cartão de op.
Os dois cartas,
mostram cartões prod

CD DO CROSS-MONTADOR

0008	0000	0006	0004	0002	0000	1459
4000	0001	0000	0000	0000	0000	1000
C500	4000	0008	4040	0540	C440	4040
4040	4040	4040		4040	4040	4040

- (continuação) - Exemplo de uma saída do Cross-Carregador.
saída de disco do programa EAD

F1 figura C2 -
saída do Cr.

O conjunto é
alto, mostra
de controle,
cartão de op
os dois cart
res, mostra
cartões proc

LABORATORIO DE SISTEMAS DIGIT

-237-

```

CC   CI INSTRUCAO
1   0000  0A
2
3
4
5
6
7   0001  C780
8
9   0003  C740
10
11  0005  C725
12
13  0007  C520
14
15  0009  C710
16
17  000B  C715
18
19  000D  C510
20
21
22
23
24
25
26  000F  D188
27  0011  D18C
28  0013  D18E
29  0015  D18F
30
31
32  0017  D108
33
34
35  0019  D148
36
37
38  0018  D118
39
40
41  001D  D158
42
43
44
45
46  001F  D128
47
48
49  0021  D168
50
51
52  0023  D138
53

```

		COMANDO
ROC	ISS	10, ROC
*	DEF.C	10
*	INSTRUCCIONES DE ENTRADA	
*	16 DISPOSITIVOS	
*	16 FUNCIONES POR D.	
*	SAI	7
*	ENTR	7
*	SAL	7,W
*	SAL	W,0
*	FNC	7,0
*	FNC	7,W
*	FNC	W,0
*	ECU	5
*	INSTRUCCIONES DE DESLOCAMIENTO	
*	DESLOCAMIENTO A DIREITA	
*	SIN	
*	DDS	1
*	DDS	2
*	DDS	3
*	DDS	4
*	DESLOCAMIENTO A DIREITA	
DD	DD	1
*	DESLOCAMIENTO A ESQUERDA	
DE	DE	1
*	DESLOCAMIENTO A DIREITA	
DDV	DDV	1
*	DESLOCAMIENTO A ESQUERDA	
DEV	DEV	1
*	INSTRUCCIONES DE GIROS	
*	GIRO A DIREITA	
GD	GD	1
*	GIRO A ESQUERDA	
GE	GE	1
*	GIRO A DIREITA CI	
GDV	GDV	1

Figura C2 -
saída do Cr.

O conjunto é
alto, mostra
de controle,
cartão de op.
Os dois cart
ões, mostran
cartões prod

COMANDO

ROC	ISS	10	ROCK
	DEF.C	10	
*	INSTRUÇÕES DE ENTRADA/SAÍDA		
*	16 DISPOSITIVOS DE ENTRADA/SAÍDA.		
*	16 FUNÇÕES POR DISPOSITIVO.		
*	SAI	7	SAÍDA DE DADOS PELO DISPOSITIVO
*		7.	7.
*	ENTR	7	ENTRADA DE DADOS PELO DISPOSITIVO
*		7.	7.
*	SAL	7,W	SALTO COM TESTE W PARA C DISPOSITIVO
*		7.	7.
*	SAL	W,O	SALTO COM TESTE O PARA C DISPOSITIVO
*		W.	W.
*	FNC	7,O	FUNÇÃO TIPO O PARA C DISPOSITIVO
*		7.	7.
*	FNC	7,W	FUNÇÃO TIPO W PARA C DISPOSITIVO
*		7.	7.
*	FNC	W,O	FUNÇÃO TIPO O PARA C DISPOSITIVO
*		W.	W.
*	EQU	5	

INSTRUÇÕES DE DESLOCAMENTOS

DESLOCAMENTO A DIREITA COM DUPLICACAO DO SINAL

DDS	1
DDS	2
DDS	3
DDS	4

DESLOCAMENTO A DIREITA

DD	1
----	---

DESLOCAMENTO A ESQUERDA

DE	1
----	---

DESLOCAMENTO A DIREITA COM V

DDV	1
-----	---

DESLOCAMENTO A ESQUERDA COM V

DEV	1
-----	---

INSTRUÇÕES DE GIROS

GIRAR A DIREITA

GD	1
----	---

GIRAR A ESQUERDA

GE	1
----	---

GIRAR A DIREITA COM V

GDV	1
-----	---

Figura C2 -
saída do CR

O conjunto é
alto, mostrando
de controle
cartão de Q.
Os dois cartões
mostra cartões PRC

LABORATORIO DE SISTEMAS DIGIT

-238-

```

CC   CI INSTRUCAO
54
55  0025 D178
56
57
58
59  0027 0F
60  0028 00
61  0029 00
62  002A 10
63  002B 002C
64  002D 0000
65  002F 04CE
66  0031 0101
67  0033 0032
68
    
```

COMANDO GIRO A ESQUERDA		
*	GEV	1
A	COM	10
B	COM	20
D	DEFC	15
F	DEFC	*-*
	DEFC	0
	DEFC	/10
	DEFE	*+1
	DEFE	*-*
	DEFE	1230
	DEFE	/101
	DEFE	*-1
	END	

Figura C2 - (continuação) - Exemplo de uma saída do programa compilado - ROC
Esta sub-rotina é chamada pelo programa

Figura C2 -
saída do CR

O conjunto c
alto, nostr
de controle,
cartão de OP
Os dois cart
res, mostran
cartões prod

COMANDO
GIRAR A ESQUERDA COM V

```

* GEV 1
* CCM 10
* CCM 20
* DEF0C 15
* DEF0C *-* 
* DEF0C 0
* DEF0C /10
* DEF0E *+1
* DEF0E *-* 
* DEF0E 1230
* DEF0E /101
* DEF0E *-1
END

```

continuação) - Exemplo de uma saída do Cross-carregador.

grama compilado - ROC

a sub-rotina é chamada pelo programa ROCA.

Figura C2 -
saída do Cr

O conjunto (alto, mostra de controle, cartão de opções, mostram cartões prod

PAGE 3

// *

// XEQ LSTDR 1

#FILES(8,WZN08)

LISTAGEM DA SAIDA DE DISCO DO CROSS-MONTADOR

1	0000	0000	0600	0036	001E	0008	0000
11	3000	0000	003C	000A	0000	0000	0000
21	0A00	C700	8000	C700	4000	C700	2500
31	C700	1000	C700	1500	C500	1000	D100
41	8C00	D100	8E00	D100	8F00	D100	0000
51	D100	1800	D100	5800	D100	0000	2800
61	3800	D100	7800	0F00	0100	0000	0000
71	0000	0000	04CE	0100	0000	0101	0000
81	0000	0000	0010	0001	0000	0000	1008
91	0000	0000	11D5	0572	0000	0000	0000
101	0000						

// EJECT

Figura C2 - (Continuação) - Exemplo de
Saída de disco de ROC

Figura C2 -
saída do CR

O conjunto é
alto, mostrando
de controle,
cartão de op.
Os dois cartões,
mostram cartões prod

O CROSS-MONTADOR

0036	001E	0008	0000	0005	0000	1958
000A	0000	0000	0000	0000	003E	0000
C700	4000	C700	2500	C500	0000	2000
1500	C500	1000	C100	0000	8800	D100
D100	8F00	D100	0000	0800	D100	4800
5800	D100	0000	2800	D100	6800	D100
0F00	0100	0000	0000	1000	002C	0000
0100	0000	0101	0000	0032	0000	0036
0001	0000	0000	1008	0010	0001	0000
0572	0000	0000	0000	1047	571E	0000

a C2 - (Continuação) - Exemplo de uma saída do Cross-Carregador
Saída de disco de ROC

Pá qura C2 -
saída do Cr.

O conjunto é
alto, mostrando
de controle o
cartão de O
Os dois car-
res, mostrando
cartões pr-

LABORATORIO DE SISTEMAS DIGITAIS

MEMORIA NAO OCUPADA 0E80
ENDERECO DE EXECUCAO 000A
ENDERECO DE INTERRUPCAO 0026

TABELA DE INTERRUPCAO
CANAL

00	01	02	03	04	05	06	07	08	09	10	11	12	13
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	F054	0000	0000	0000

CONFIGURACAO

NOME	ENDERECO
PRINC	000A
MART	0026
REC	0050
CAD	0087
MCR	0086

Figura C2 - (continuação) - Exemplo de saída do Cr
Mapa de carregamento impresso pelo Pro

0000000000000000
1111111111111111
2222222222222222
3333333333333333
4444444444444444
5555555555555555
6666666666666666
7777777777777777
8888888888888888
9999999999999999
124 17622 3-7

Figura C2 -
saída do Cr

O conjunto é
alto, mostrando
de controle,
cartão de op
Os dois cart
res, mostrar
cartões prod

0-

C DE SISTENAS DIGITAIS - L.S.C. - PAC. I

0880
000A
CAC 0026

RELA DE INTERRUPCAO
CANAL

06	07	08	09	10	11	12	13	14	15
0000	0000	0000	0000	FC54	0000	0000	0000	0000	0000

continuação) - Exemplo de saída do Cross-Carregador.
apá de carregamento impresso pelo Programa cross-Carregador.

F1 figura C2 -
saída do Cr

O conjunto
alto, nostr
de controle
cartão de O
Os dois car
res, mostra
cartões PRC

ACCUMULATOR 7847

ACDR	EXTENSION				XRI				XR2		XR3	
	****0	****1	****2	****3	****4	****5	****6	****7	****8	****9	****A	****B
3800	405F	4C00	C026	C004								
3810	0100	000A	CCCC	CCCC	FF00	6020	4021	201F	6020	CCCC		
3820	4C20	76FC	6304	6F00	0000	0000	0000	0000	0000	0000	0000	
3830	8801	8C01	8E01	8FD1	3210	704E	0026	0026	0026	0026	0026	
3840	CE01	0100	8223	C010	0801	4801	1801	5801	2801	6801	6801	
					10C5	80C5	4064	5F16	5F00	5F00	5F00	

// EJECT

figura C2 - (continuação) - Exemplo de saída
"Dump" da memória simulada.

0 0
0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7

8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7

FI figura C2 -
saída do Cr

O conjunto,
alto, mostra,
de controle,
cartão de or
Os dois cart
res, mostra
cartões prod

XRI FF80	XR2 0219	XR3 33C6	OVERFLOW CN	CARRY CFF						
FFFFS	FFFF6	FFFF7	FFFF8	FFFF9	FFFFA	FFFFB	FFFFC	FFFFD	FFFFE	FFFFF
6020	4021	201F	6020	000A	501F	9E55	4001	F05C	F0E7	9DCA
0000	0000	0000	0000	0000	0000	0000	0000	0000	FC54	ECF3
704E	0026	0026	CAC7	8007	40C7	25C5	20C7	10C7	15C5	1001
48D1	18D1	5801	28D1	6801	3801	780F	0000	1000	7000	0004
80C5	4064	5F16	5F00	5F00	5F60	1580	5F64	5F16	5F00	5F00

inuação) - Exemplo de saída do cross-Carregador.
* da memória simulada.

ALDAG SBBIS ROCHE M D T MBSI

FILESC12,85812),14,UNIBO(,11,UNIBO(,11,UNIBO(,

// XEQ 420

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

--

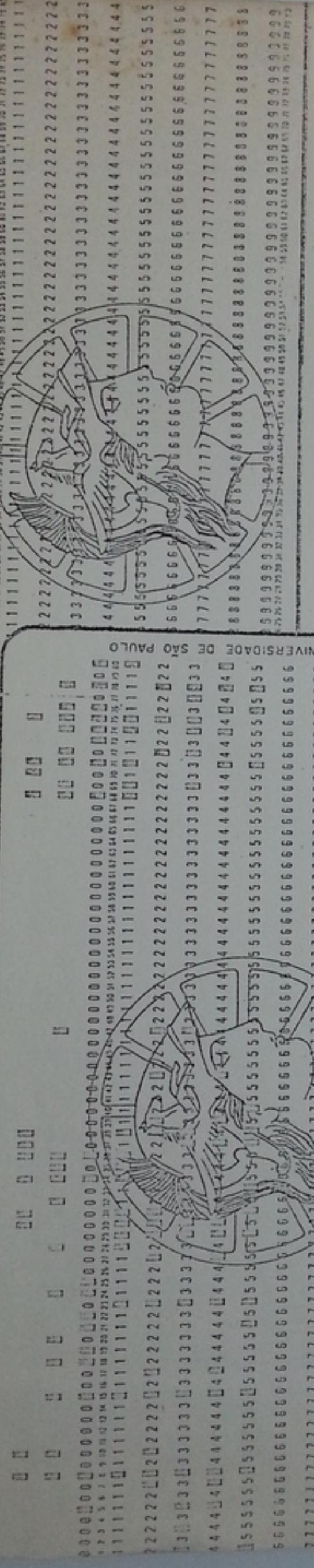
--

--

--

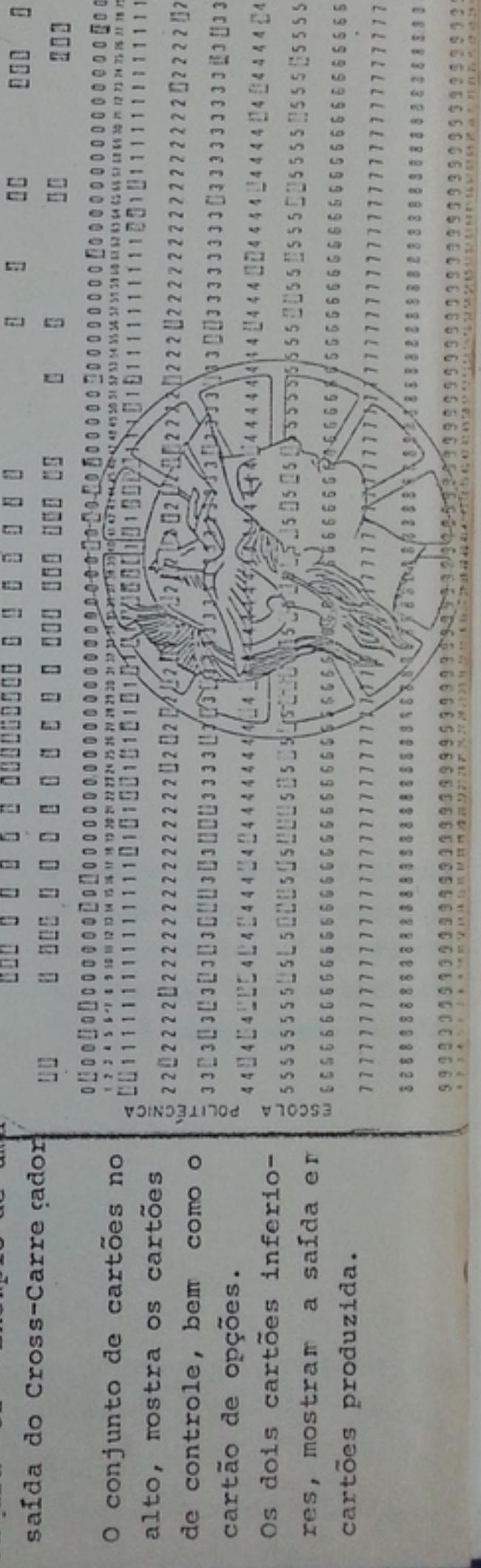
--

--



-242-

Figura C2 - Exemplo de uma saída do Cross-Carre sadon



O conjunto de cartões no alto, mostra os cartões de controle, bem como o cartão de opções.
Os dois cartões inferiores, mostrar a saída e os cartões produzida.

PAGE 11

// XEQ WZW 1

*FILES(12,888I2),(1,WZN08)

Figura C3 - Exemplos de uso da Biblioteca
A figura mostra os cartões para o seu uso. É apresentado de comandos para a inclusão teca. É apresentado também

21 Símbolos equivalentes já foram definidos e contém endereços diferentes

22 Há impossibilidade de endereçar 2 símbolos equivalentes

3 - Exemplos de uso da Biblioteca.

A figura mostra os cartões de controle da Biblioteca, seguidos de comandos para o seu uso. É apresentado o comando de geração da Biblioteca, seguido de comandos para a inclusão de programas, e um pedido de listagem da Biblioteca. É apresentado também um comando para a exclusão de programas.

am definidos e contém end

car 2 símbolos equivalentes

-244-

LABORATORIO DE SISTEMAS DIGITA

*BIBL 88812 48 1 12
BIBLIOTECA 88812
TOTAL BLOCOS BLOCOS BLOCO NUM.DE SETCRES
BLOCOS USADOS REST. LIVRE PROG. INDICE
48 17 31 18 0 1

*INCL 88812 ROC R

PROGRAMA PEDIDO NAO SE ENCONTRA NO ARQUIVO

*INCL 88812 MOR R

PROGRAMA MOR INCLUIDO NA BIBLIOTECA 88812 ENDERECO

*LIST 88812

----- não apresenta endereço de execução

21 Símbolos equivalentes já foram definidos e contém endereços diferentes

22 Há impossibilidade de endereçar 2 símbolos equivalentes

4-

IO DE SISTEMAS DIGITAIS

L.S.D. - PAG. 1

48 1 12

COS BLOCO NUM. DE SETCRES
T. LIVRE PROG. INDICE
31 18 0 1

R

ENCONTRA NO ARQUIVO

R

O NA BIBLIOTECA BBB12

ENDERECO 18 EXTENSAO EM CB 2

ao apresenta endereço de execução

s já foram definidos e contém ende

: endereçar 2 símbolos equivalentes

LABORATORIO DE SISTEMAS DIGITA

BIBLIOTECA BBB12

TOTAL	BLOCOS	BLOCOS	BLOCO	NUM. DE	SETORES
BLOCOS	USADOS	REST.	LIVRE	PROG.	INDICE
48	19	29	20	2	1

NOME PROG	COMP DB	END
--------------	------------	-----

MOR	2	18
-----	---	----

NOME PROG	COMP DB	END
--------------	------------	-----

CAD	2	18
-----	---	----

NOME PROG	COMP DB
--------------	------------

*FIM

o---- para o par não apresenta endereço de execução

21 Símbolos equivalentes já foram definidos e contém endereços diferentes

22 Há impossibilidade de endereçar 2 símbolos equivalentes

EMAS DIGITAIS

- L.S.D. - PAG. 2

JM.DE SETORES
ROG. INDICE
2 1

NOME CCMP
PROG DB END

NOME CCMP
PROG DB END

ndereço de execu,
inidos e contém ende

LABORATORIO DE SISTEMAS DIGITA

*EXCL BBB12 ROCA
PROGRAMA ROCA FOI EXCLUIDO DA BIBLIOTECA BBB12
*INCL BBB12 ROCA R
PROGRAMA ROCA INCLUIDO NA BIBLIOTECA BBB12 ENDERECO
*FIM

- 21 Programa principal não apresenta endereço de execução
 Símbolos equivalentes já foram definidos e contém endereços diferentes
- 22 Há impossibilidade de endereçar 2 símbolos equivalentes

E SISTEMAS DIGITAIS - L.S.D. - PAG. 1

BIBLIOTECA 88812

LIBRERIA 88812 ENDEREÇO 30 EXTENSÃO EM DB 3

esenta endereço de execução

oram definidos e contém ender

reçar 2 símbolos equivalentes

A P E N D I C E D

TABELA DE ERROS DO "CROSS - MONTADOR"

- 01 Símbolo se inicia com caractér pribido
- 02 Símbolo contém caractér proibido
- 03 Perfuração inválida entre fim de rótulo e início de mnemonico
- 04 Instrução que não deve conter operando contém um operando
- 05 Perfuração inválida em coluna de indireto
- 06 Pseudos ORG é precedido por outros comandos diferentes de "pseudo" de controle
- 07 Instrução que exige operando não o contém
- 08 Instrução de deslocamento apresenta operando inválido
- 09 Instrução de salto ou LIMP contém operando inválido
- 10 Sintaxe errada em comando de Entrada e Saída
- 11 Operando imediato é maior que 255
- 12 Erro de sintaxe de "pseudo" EBC ou ASC
- 13 Número contém dígito inválido
- 14 "Pseudo" contém um rótulo proibido
- 15 Instrução com endereçamento relativo a si mesmo inválido
- 16 Comando EQU apresenta endereço fora dos limites de memória
- 17 Instrução ou "pseudo" indireto proibida
- 18 Mnemônico não foi encontrado nas tabelas de busca
- 19 "Pseudo" de FIM de PROGRAMA apresenta rótulo
- 20 Programa principal não apresenta endereço de execução
- 21 Símbolos equivalentes já foram definidos e contém endereços diferentes
- 22 Há impossibilidade de endereçar 2 símbolos equivalentes

- 23 "Pseudo" ORG apresenta origem maior que 4095
- 24 Endereço simbólico de Entrada e Saída não definido
- 25 Num comando E/S tipo FNC ou SAL falta o delimitador ","
- 26 "Pseudo" EQU não contém rótulo
- 27 "Pseudo" DEFE com endereçamento indireto
- 28 Operando de EQU está indefinido numa expressão
- 29 Símbolo definido de maneira imprópria para ponto de entrada já definido
- 30 Comando de painel inválido
- 31 Constante definida é inválida
- 32 Dispositivo endereçado em "pseudo" ISS é maior que 15
- 33 Numa instrução de endereçamento a memória o endereço é > que 4095 ou < que zero
- 34 Ponto de entrada de sub-rotina proibido, ou por ter mais de 12 pontos já definidos ou devido a existência de um ponto de entrada de sub-rotina de interrupção.
- 35 "Pseudo" não contém o operando necessário, ou apresenta operando proibido
- 36 "Pseudo" ISS não é o primeiro comando diferente de "pseudo" de controle
- 37 Sintaxe errada em comando ISS

A P É N D I C E

TABELA ----- DE ----- ERROS ----- DO ----- CARREGADOR

Durante o processamento do "Cross-Carregador", a ocorrência de um erro é o suficiente para que se dê o encerramento do programa.

O erro ocorrido, é listado para informar a causa do cancelamento. Os seguintes erros podem ocorrer, durante a composição de um programa:

<u>Erro</u>	<u>Causa</u>
CÓDIGO INVÁLIDO	Este erro ocorre, quando se tenta compor um programa de tipo 4 ou 6, isto é, rotinas simples, ou de entrada e saída. O "Cross - Carregador", compõe apenas programas principais.
NÚMERO DE SUB-ROTINAS EX CEDIDO	O programa, sendo composto requisita, um número de rotinas excessivo o qual não pode ser causado, quando se pede a utilização de sub-rotinas com múltiplos "entry-points", os quais são todos colocados na tabela.
ORIGEM EM POSIÇÃO IMPAR	A origem especificada para um programa absoluto é uma posição ímpar de memória.
ORIGEM MENOR QUE 4	A origem especificada para um programa absoluto é menor que 4.
ORIGEM 4 E A ROTINA DE INTERRUPÇÃO	A origem de um programa absoluto é 4, e há uma rotina de interrupção mencionada no catálogo de pedido de opções.

Erro

SUB-ROTI_NA COM
CÓDIGO INVÁLIDO
NÚMERO DE CANAL
INVÁLIDO

DUAS SUB-ROTI_NAS
PARA O MESMO CA-
NAL

ROTI_NA DE INTER-
RUPÇÃO NÃO ESPE-
CIFICADA E HÁ
E/S

MEMÓRIA EXCEDIDA

CÓDIGO DE RELOCA-
ÇÃO INVÁLIDO

ROTI_NA DE INTER-
RUPÇÃO COM ESCRI-
TA ERRADA

ERRO DO CONSTRU-
TOR

COMMON DA SUB-ROTI_NA
EXCEDE O DO PRO-
GRAMA

Causa

Foi especificada uma sub-rotina de interrupção, a qual porém não é de tipo permitido. O número do canal especificado em uma rotina de entrada e saída é maior que 15.

Duas sub-rotinas de entrada e saída foram especificadas para o mesmo canal de entrada e saída.

Durante a composição do programa, encontrou-se uma chamada para uma rotina de entrada e saída, enenhuma rotina de tratamento de interrupção foi indicada no cartão de opções.

A memória disponível para programas foi excedida durante a composição de um programa.

Encontrou-se um indicador de relocação com valor 2, proibido pela programação.

A rotina de interrupção não obedece as normas de escritas preestipuladas.

Ocorreu uma condição anormal. O programa "Cross-Carregador" deve ser novamente compilado e montado para se continuar o processamento.

Durante uma composição, o "common" (área comum) do programa é menor do que o da sub-rotina em composição.

A P E N D I C E F

T A B E L A D E E R R O S D A B I B L I O T E C A

Durante a execução de um comando de Biblioteca podem ocorrer os seguintes erros:

ERRO	CAUSA
PROGRAMA PEDIDO NÃO SE ENCONTRA NO ARQUIVO	O arquivo especificado para uso não <u>con</u> tém um programa ou sub-rotina válidos.
NOME DA SUB-ROTINA DE INTERRUPÇÃO PROIBIDO	O nome da sub-rotina de interrupção é o mesmo do programa principal.
COMANDO INVÁLIDO	O comando de controle da biblioteca não é válido.
PROGRAMA PPPPP NÃO <u>FI</u> GURA NA BIBLIOTECA	Uma rotina pedida pelo programa não figura na biblioteca especificada para uso.
PROGRAMA PPPPP NÃO <u>ES</u> TÁ NA BIBLIOTECA	Foi requisitada a perfuração de um <u>pro</u> grama, em cartões, o qual não faz <u>par</u> te de nenhuma das bibliotecas <u>especifi</u> cadas.
PROGRAMA NÃO ESTÁ NO <u>FOR</u> MATO PEDIDO	Requisitou-se a perfuração de um <u>pro</u> grama em formato relocável.
IMPOSSÍVEL ESTENDER <u>IN</u> DICE-OVERFLOW	Foi pedida a extensão do índice de uma biblioteca, ou então esta extensão <u>de</u> veria ser feita automaticamente, mas não há espaço suficiente na biblioteca para isso.

ERRO GRAVE NA BIBLIOTECA

Durante a compactação da biblioteca, deu-se um erro grave nesta biblioteca, cuja correção não pode ser feita automaticamente.

BIBLIOTECA PEDIDA NÃO SE ENCONTRA NO AR

Indicou-se em um cartão de comando de biblioteca o nome de uma biblioteca, que não é válido.

NOME DO PROGRAMA JÁ CONSTA DA BIBLIOTECA

Pedi-se a inclusão de um nome na biblioteca entretanto, esta já possui um programa com o mesmo nome.

BIBLIOTECA ESGOTADA

A inclusão pedida de um programa não pode ser feita, pois não há "disk blocks" disponíveis na biblioteca em número suficiente.

NOME ESPECIFICADO NÃO É ENTRY POINT ACEITO

Na inclusão de uma sub-rotina na biblioteca o nome especificado no cartão de comando não é o de nenhum dos "entry points" da mesma.

NOME DO PROGRAMA NÃO FIGURA NA BIBLIOTECA

Foi pedida a exclusão de um programa, o qual não se encontra na biblioteca especificada.

BIBLIOGRAFIA

1. B1. Barron. D.W. - Assemblers and Loaders
Mc Donald: London and American Elsevier Inc.
New York, 1969
2. D1. Debraine. Paul - Machine de Traitement de l'information
Tome II - Programmation: Principes et
languages d'Assemblage.
3. D2. Donovan. John J. - Systems Programming
Mc Graw-Hill Book Co., 1972
4. F1. Fisher. F. Peter and George F. Swindle - Computer Pro
gramming System - Holt Rinehart and
Winston, 1964
5. F2. Flores. Ivan - Assembler and BAL
Prentice Hall, 1969
6. F3. Flores. Ivan - Computer Software
Prentice Hall, 1969
7. G1. Gries. David - Compiler Construction for Digital Com
puter - John Wiley & Sons, Inc., 1971
8. G2. Gear. C. William - Computer Organization and Programming
Mc Graw-Hill Book Co., 1969
9. H1. Hassit-Anthony - Computer Programming and Computer Systems
Academic Press - New York and London, 1967
10. H2. Hopgood-F.R.A. - Compiling Techniques
Mc Donald - London, 1969
11. H3. Hellerman - Herbert - Digital Computer Systems Principles
Mc Graw Hill - Book Company, 1967
12. K1. Knuth - Donald E. - The Art of Computer Programming - Fun
damental Algorithms - John Wiley & Sons

13. Pl. Price. Wilson T. - Elements of IBM - 1130 Programming
Holt, Rinehart and Winston, 1968
14. R1. Rosen, Saul - Programming Systems and Languages
Mc Graw Hill Book Co., 1967
15. S1. Sammet, Jean - Programming Languages: History and Fundamentals
Prentice Hall, Inc. Englewood Cliffs, 1969
16. S2. Stone. Harold S. - Introduction to Computer Organization and Data Structure
Mc Graw Hill Book Co., 1972
17. W1. Wegner. Peter -Programming Languages, information Structures and Machine Organization
Mc Graw Hill Book Co., 1967
18. B2. Bergeron R. Daniel and Gannon. John D. and Van Dam Andries
Language for Systems Development - ACM
Sigplan, Notices - Volume 6, Number 9, October, 1971
19. D4. David J. - A Survey of the Systematic Use of Macros in
Systems Building - ACM Sigplan, Notices - Volume 6, Number
9, October, 1971
20. L1. Lamb, Jr. Vincent - All About Cross Assemblers - Datamation
July, 1973 - pg. 77 a 80
21. S3. Sammet. Jean - Programming Languages: History and Future
Comm. of ACM - July, 1972 - Vol. 15 - Number 7.
22. S4. Sammet, Jean - Brief Survey of Languages used in Systems
Implementation - ACM Sigplan Notices - Volume 6, Number 9,
October, 1971
23. IBM-1130/1800 Assembler Languages System Reference Library
FN 1130/1800 Assembler Language System Reference Library
24. IBM 1130 Disk Monitor System, Operator's Guide System, Version
2, Programmer's and Library - FN 1130-36 ON-GC 26-3717-9
25. F4. Fregni - Projeto Lógico de Unidade de Controle de um Mini-computador - Dissertação de mestrado apresentada a Escola Politécnica da Universidade de São Paulo, 1972.

FD-85

Ferreira, Maria Alice Grigas Varella
Sistema de programação "Cross-Monta-
dor" para um minicomputador

BIBLIOTECA
Dept. Eng. ELETRICIDADE
E. P. U. S. P.

MARIA ALICE GRIGAS VARELLA FERREIRA
Eng^a Eletricista, Escola Politécnica da USP, 1968
Mestre em Engenharia, Escola Politécnica da USP, 1975

LINGUAGEM ESTR
PROCESSAMENTO C
DEFINIÇÃO E

Tese
Polí
obte
tor

FERREIRA

TESE

EPUSP

Suor

Orientador: Prof. Dr. Tamio Shimizu
Prof. Livre Docente do Departamento de
Engenharia de Produção da Escola Poli
técnica da Universidade de São Paulo.

São Paulo, 1979.

FT-132



Eng.^a Eletricista, Escola F
Mestre em Engenharia, Escola

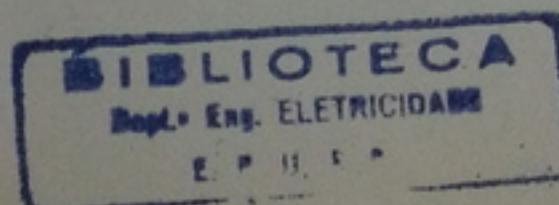
LINGUAGEM ESTRUTURADA PARA
PROCESSAMENTO CONCORRENTE:
DEFINIÇÃO E PROJETO

Tese apresentada à Escola
Politécnica da USP para a
obtenção do Título de Dou
tor em Engenharia.

Orientador: Prof. Dr. Tamio Shimizu
Prof. Livre Docente do Departamento de
Engenharia de Produção da Escola Poli
técnica da Universidade de São Paulo.

São Paulo, 1979.

FT-132



FD-85

Autor: Ferreira, Maria Alice G. Varella
Título: Sistema de programação "Cross-Montador" para um minicomputador

FD - 85

Ferreira, Maria Alice G. Varella
Sistema de programação " Cross-Montador" para um microcomputador.

Serviço de Bibliotecas
Biblioteca de Engenharia Elétrica