UNIVERSIDADE FEDERAL DA PARAIBA

CENTRO DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO

PROJETO DE UM CROSS-FORTRAN PARA MICROPROCESSADORES

BERNARDO LULA JUNIOR

CAMPINA GRANDE - PARAIBA

JANEIRO - 1979

L955p Lula Junior, Bernardo.
Projeto de um cross-fortran para microprocessdores / Bernardo Lula Junior. - Campina Grande, 1979.
92 f.

Dissertação (Mestrado em Ciências) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1979.
"Orientação : Prof. José Homero Feitosa Cavalcanti".
Referências.

1. Softwares - Desenvolvimento. 2. Engenharia de Software. 3. Desenvolvimento de Software. 4. Dissertação - Ciências. I. Cavalcanti, José Homero Feitosa. II. Universidade Federal da Paraíba - Campina Grande (PB). III. Título

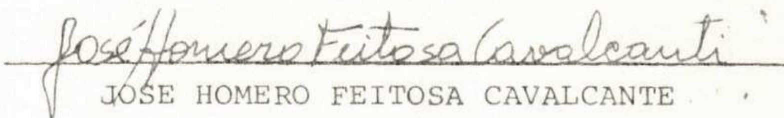
CDU 004.41(043)

PROJETO DE UM CROSS-FORTRAN PARA MICROPROCESSADORES

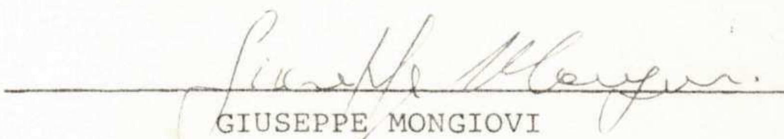
BERNARDO LULA JUNIOR

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

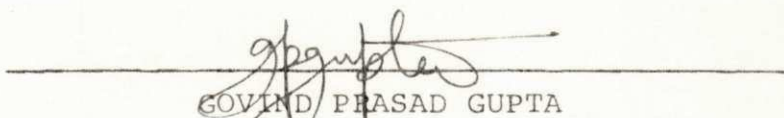
Aprovada por:



JOSÉ HOMERO FEITOSA CAVALCANTE
- Presidente -



GIUSEPPE MONGIOVI
- Examinador -



GOVIND PRASAD GUPTA
- Examinador -

CAMPINA GRANDE
ESTADO DA PARAÍBA - BRASIL
JANEIRO - 1979

Aos meus pais e sogros

AGRADECIMENTOS

O autor agradece :

Ao seu orientador e amigo Prof. JOSE HOMERO FEITOSA CAVALCANTI pela atenção, incentivo e amizade, não só na orientação deste trabalho, mas também no decorrer do curso de mestrado, extensivo, em número e grau, a todos os colegas do departamento de sistemas e computação.

R E S U M O

Este trabalho apresenta um projeto de um sistema de desenvolvimento de *software* para microprocessadores. O sistema foi concebido para operar na modalidade de *cross-software* e com o objetivo de possibilitar o desenvolvimento de programas, em uma linguagem de alto-nível, para um conjunto de diferentes microprocessadores.

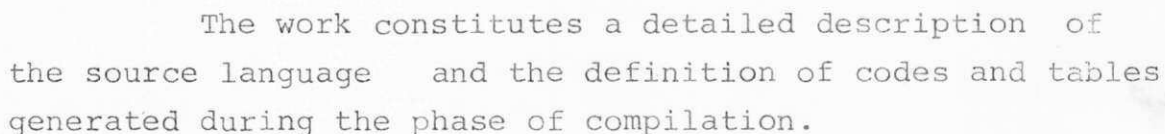
Os principais aspectos considerados no projeto foram: a valiação de qual linguagem seria mais conveniente como linguagem-fonte do sistema, considerando a atual fase de desenvolvimento de *software* para microprocessadores, e, a forma econômica de implementação dessa linguagem para uma classe numerosa desses dispositivos.

Consta do projeto uma descrição detalhada da linguagem fonte considerada e a definição dos códigos e tabelas gerados nas fases do processo de tradução.

ABSTRACT

This work describes a software development system for microprocessors. The system is viewed to operate in cross-software mode with the objective of development of programs in a high level language for a set of different microprocessors.

The two principal aspects considered in the design are: selection of a source language of the system considering the actual development phase of software, and a economic form to implement the language for a class of microprocessors.



The work constitutes a detailed description of the source language and the definition of codes and tables generated during the phase of compilation.

Í N D I C E

CAPITULO 1 : INTRODUÇÃO................................ 1

1.0 - INTRODUÇÃO.......................... 1

1.1 - LINGUAGEM DE ALTO NÍVEL............. 2

1.2 - O SISTEMA CFOR...................... 5

1.3 - DESENVOLVIMENTO DO SISTEMA.......... 8

CAPITULO 2 : A LINGUAGEM FONTE......................... 10

2.0 - INTRODUÇÃO.......................... 10

2.1 - ESCOLHA DO FORTRAN COMO LINGUAGEM FONTE............................. 11

2.1.1 - SIMPLICIDADE E ACEITAÇÃO.... 11

2.1.2 - EFICIÊNCIA DE EXECUÇÃO...... 12

2.1.3 - TRANSPORTABILIDADE.......... 13

2.1.4 - FACILIDADE DE IMPLEMENTAÇÃO. 13

2.2 - DESCRIÇÃO DA LINGUAGEM.............. 14

2.2.1 - CARACTERES BÁSICOS E PALA_ VRAS RESERVADAS............ 14

2.2.2 - TIPOS DE DADOS.............. 16

2.2.3 - CONSTANTES E VARIAVEIS...... 17

2.2.4 - OPERAÇÕES................... 19

2.2.5 - DECLARAÇÕES E COMANDOS...... 21

2.2.6 - SUBPROGRAMAS................ 26

2.2.7 - ENTRADA E SAIDA........... 28

2.2.8 - INTERRUPÇÃO............... 30

CAPITULO 3 : ORGANIZAÇÃO DO PROCESSO DE TRADUÇÃO...... 32

3.0 - INTRODUÇÃO......................... 32

3.1 - ORGANIZAÇÃO PADRÃO................. 32

3.2 - NIVEL INTERMEDIARIO................ 35

3.3 - SIMPLIFICAÇÃO DO GERADOR........... 39

CAPITULO 4 : ANALISADOR............................. 41

4.0 - INTRODUÇÃO......................... 41

4.1 - CODIGO INTERMEDIARIO.............. 41

4.1.1 - FORMATO DAS INSTRUÇÕES..... 41

4.1.2 - ORGANIZAÇÃO DO PROGRAMA INTERMEDIARIO............... 44

4.1.3 - INSTRUÇÕES NI.............. 45

4.2 - TABELAS DE INFORMAÇÃO............. 52

4.2.1 - TABELA DE VARIAVEIS (TV)... 53

4.2.2 - TABELA DE VARIAVEIS TEMPORARIAS (TVT)............... 54

4.2.3 - TABELA DE CONSTANTES NUMERICAS E LOGICAS (TCNL)..... 54

4.2.4 - TABELA DE CONSTANTES DE CARACTERES.(TCC)............. 55

CAPITULO 5 : MACROS E SUBROTINAS.................... 57

5.0 - INTRODUÇÃO......................... 57

5.1 - MACROS............................. 58

5.2 - TABELA DE DEFINIÇÃO DE MACROS(TDM). 62

5.3 - SUBROTINAS.......................... 64

5.4 - TABELA DE DEFINIÇÃO DE SUBROTINA (TDS)............................ 66

5.5 - TABELA DE VARIAVEIS INTERNAS (TVI). 67

CAPITULO 6 : GERADOR DE CODIGO........................ 69

6.0 - INTRODUÇÃO.......................... 69

6.1 - ORGANIZAÇÃO DO GERADOR............. 69

6.2 - O PROCESSADOR DE MACROS............ 73

6.2.1 - TABELAS E VARIAVEIS........ 73

6.2.2 - FLUXOGRAMA DO PROCESSADOR.. 76

6.3 - GERADOR DE DECLARAÇÕES............. 80

CAPITULO 7 : CONCLUSÕES............................ 81

7.0 - INTRODUÇÃO.......................... 81

7.1 - SUMÁRIO............................. 81

7.2 - IMPLEMENTAÇÃO DO SISTEMA........... 83

7.3 - SUGESTÕES........................... 84

APÊNDICE A : A.1 - PROGRAMA FONTE EXEMPLO............. 86

A.2 - PROGRAMA INTERMEDIARIO EQUIVALENTE. 87

A.3 - EQUIVALÊNCIA DE VARIAVEIS E CONSTANTES.......................... 88

APÊNDICE B : B.1 - TABELA DE VARIAVEIS............... 89

B.2 - TABELA DE VARIAVEIS TEMPORARIAS.... 89

B.3 - TABELA DE CONSTANTES............... 89

BIBLIOGRAFIA : 90

UNIVERSIDADE FEDERAL DA PARAÍBA
Pró-Reitoria Para Assuntos do Interior
Coordenação Setorial de Pós-Graduação
Rua Aprigio Veloso, 882 Tel (083) 321-7222-R 355
58.100 - Campina Grande - Paraíba

Capítulo I

INTRODUÇÃO

1.0 - INTRODUÇÃO

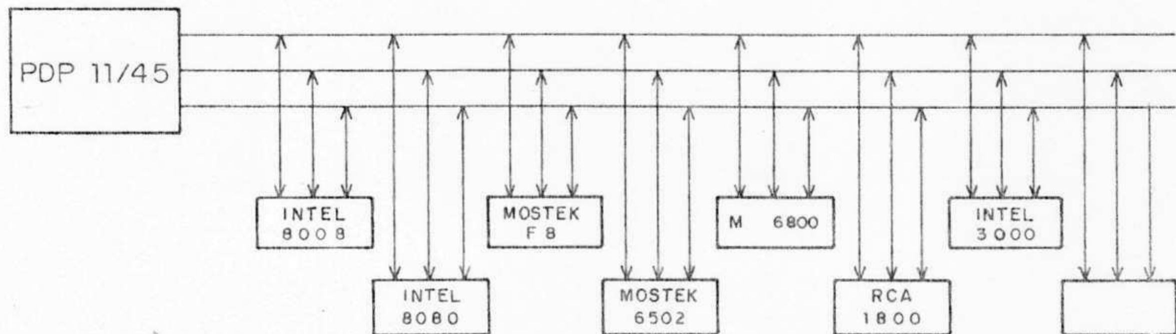
A grande variedade de microprocessadores presentes no mercado (mais de 40 tipos), proporciona aos interessados no desenvolvimento de sistemas baseados nesses dispositivos, a possibilidade de escolha do que melhor se adapte às especificações do projeto, escolha essa, baseada nas considerações de arquitetura, velocidade de operação, conjunto de instruções, capacidade de entrada e saída, e custo (1 e 2). No entanto, essa flexibilidade de seleção do *hardware* fica comprometida quando se considera os aspectos relativos ao *software*. A maioria dos sistemas de apoio existentes para o seu desenvolvimento, quer na forma de microcomputadores especializados nesta tarefa (*TWIN, EXORCISE, MCSIM, etc*), ou na forma de *cross-software* (residente em uma máquina com maiores recursos), apresentam dois grandes inconvenientes ou restrições:

- São dedicados a uma particular máquina.

- Se restringem à linguagem *assembly* do específico microprocessador a que se destinam atender.

Em alguns sistemas, como o desenvolvido pela *CASE WESTERN RESERVE UNIVERSITY, CLEVELAND, OH*, apresentado na figura 1.1 ,

embora a primeira restrição seja relaxada (o usuário conta com um editor de texto, e um *cross-assembler* para cada microprocessador que o sistema suporta), não apresentam uma flexibilidade efetiva, desde que o usuário deve aprender e bem utilizar os diversos *assemblys* para usufruir da capacidade oferecida de escolha do *hardware*, levando-o a tender conservar o microprocessador a que já está afeito, mesmo que não seja o mais adequado para os fins em vista (1).



Sistema da *CWR University*

Figura 1.1

1.1 - LINGUAGEM DE ALTO-NÍVEL

Linguagens de alto-nível projetadas especialmente para uso em microprocessadores têm sido desenvolvidas, tanto por fabricantes como por fontes independentes, e oferecidas aos usuários principalmente na forma de *cross-software* (3). A primeira dessas linguagens anunciada foi a PL/M, derivada da linguagem PL/I, introduzida pela INTEL em 1974 para auxiliar a programação do seu microprocessador 8008 e posteriormente modificada para suportar os novos recursos introduzidos no seu

sucessor, o 8080 (4). As mesmas razões que, em parte, determinaram o sucesso de várias linguagens de programação de computadores - o domínio do mercado por alguns fabricantes e a grande quantidade de recursos por eles investido na preparação de sofisticadas implementações e extensiva documentação - estão contribuindo para a adoção dêsse produto da INTEL, com a consequente proliferação de linguagens tipo PL/M, desenvolvidas para outros tipos de microprocessadores (4 e 5):

MPL/I - desenvolvida pela MOTOROLA para seu microprocessador M6800

SMPL - desenvolvida pela NATIONAL SEMICONDUCTOR para o IMP-16 e PACE

PL/M6800 - desenvolvida pela INTERMETRICS para o M6800

PL/W - desenvolvida pela WINTEK CORPORATION para o M6800

PLμs - desenvolvida por KILDALL para o SIGNETICS 2650

PL/Z - desenvolvida pela ZILOG para o Z-80

Para uma determinada classe de usuários, essas linguagens atendem perfeitamente às suas necessidades, proporcionando um aumento efetivo na taxa de produção do *software*, como também permitindo uma melhor documentação, em relação ao uso da correspondente linguagem *assembly*. A INTEL, por exemplo, tem desenvolvido diversos projetos utilizando a linguagem PL/M, entre êsses, um montador avançado, um compilador PL/M e um compilador FORTRAN IV para o sistema INTELLEC 8, que exigiram um reduzido tempo de desenvolvimento e testes, estimado em 5 vêzes menos do que o necessário com a utilização da linguagem

assembly, com um eficiente código objeto resultante, além da facilidade de manutenção e alteração proporcionada pela boa documentação (3 e 6). Essas linguagens, entretanto, enquanto sejam ótimas ferramentas nas mãos de "engenheiros de *software*", não encontram a mesma receptividade por parte dos usuários não-profissionais em programação ou engenheiros de *hardware* envolvidos em projetos de sistemas digitais utilizando microprocessadores (7):

- São linguagens de "implementação de sistemas", de alto-nível, mas, voltadas ao contrôle de uma particular máquina e suas funções, refletindo sua arquitetura, sendo dêsse modo incompatíveis em vários graus. Enquanto que, pelas suas características, produzam um eficiente código objeto, não oferecem ao usuário a necessária portabilidade(3 e 8)

- é de responsabilidade do usuário o desenvolvimento de todas as rotinas necessárias ao funcionamento do sistema (inicialização, entrada e saída, contrôle de interrupção, etc), exigindo a manipulação de *bits* e o conhecimento da maneira específica como a máquina realiza essas funções.

- a maior parte dêsses usuários são familiarizados com algumas poucas linguagens de programação, e entre essas, não se encontra de fato, o PL/I (4).

Embora essas linguagens tenham adquirido uma prioridade inicial por parte dos fabricantes de microprocessadores, linguagens de alto-nível orientadas para aplicação estão emergindo, versões das populares FORTRAN e BASIC, implementadas nos mais novos sistemas de desenvolvimento de *software* para microprocessadores (*INTELLEC 8, MILLENNIUM SYSTEM, ZENO SYSTEM, etc*), dotando os usuários de uma ferramenta de programação "universal" e familiar (4 e 6).

1.2 - O SISTEMA CFOR

Este trabalho apresenta o projeto de um sistema de apoio para desenvolvimento de programas para aplicação em microprocessadores. O sistema foi projetado para oferecer aos usuários as vantagens da utilização de uma linguagem de alto-nível voltada para aplicação (maior produtividade, maior facilidade de manutenção, e portabilidade de um microprocessador para outro), e, em função dos recursos existentes no NPD do CCT-UFPb*, apresentando as seguintes características:

1 - Linguagem fonte

A linguagem fonte do sistema CFOR é um subconjunto do FORTRAN IV do sistema IBM/370, com a supressão, simplificação e uso de declaração FORTRAN com sentido diferente ao atribuido pelo FORTRAN IV para possibilitar o desenvolvimento de programas típicos de microprocessadores e adaptação ao *hardware* dos microprocessadores de finalidade geral, classe mais numerosa e de maior espectro de aplicação entre os diversos tipos desses dispositivos. Embora modificações tenham sido introduzidas, foi mantida total compatibilidade com a sintaxe do FORTRAN IV, permitindo inclusive a utilização do compilador FORTRAN IV para uma primeira fase de depuração de êrros.

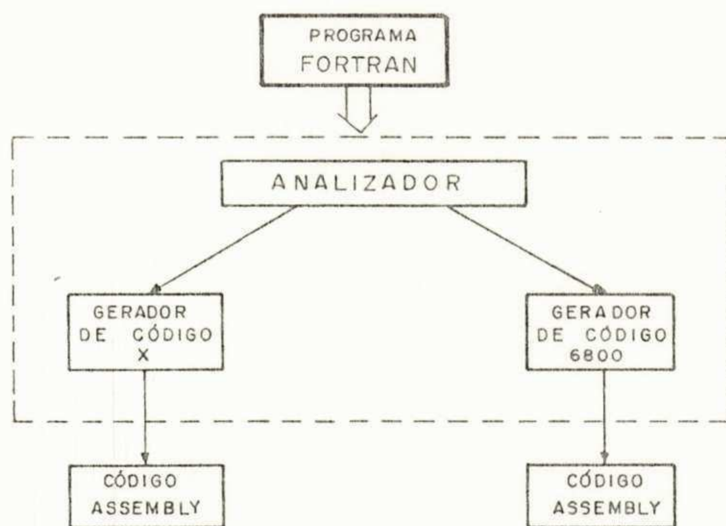
2 - Capacidade de programação "universal"

Teoricamente, a linguagem fonte apresenta essa capa_

---

* *Núcleo de Processamento de Dados do Centro de Ciências e Tecnologia da Universidade Federal da Paraíba.*

cidade, entretanto, para que ela se transfira efetivamente para o sistema, é necessário a sua implementação para os diversos microprocessadores da classe considerada. Evidentemente, um compilador para cada microprocessador, embora tecnicamente viável, não corresponderia do ponto de vista econômico pelo alto custo envolvido na geração dêsse *software*. Para que o sistema ofereça essa capacidade a um custo aceitável, foi adotada a técnica sugerida por MONGIOVI (9) e utilizada por CASILLI na implementação do μBASIC no MILLENNIUM SYSTEM (7), onde grande parte do processo de tradução é padronizado e compartilhado pelos compiladores, como apresentado na figura abaixo:



Sistema CFOR

Figura 1.2

3 - Cross-software

O sistema foi planejado para operar na forma de *cross-software* pelos seguintes motivos:

- Disponibilidade de um computador IBM/370-145 no NPD do CCT-UFPb com todos os recursos necessários ao desenvolvimento e implementação do sistema.

- Maior rapidez do processo de tradução pelo aproveitamento da velocidade de operação da CPU do 370-145.

- Possibilidade de vários usuários utilizarem o sistema ao mesmo tempo, aproveitando a capacidade de multi-programação oferecida pelo OS-VSI do computador do NPD.

- Praticamente nenhuma restrição de memória no desenvolvimento de um compilador é introduzida, ao contrário do que seria exigido caso o sistema fosse projetado para ser residente em microcomputador.

Dêsse modo, o sistema além de usufruir da eficiência do processamento da máquina indicada, não exige nenhum custo adicional em *hardware*, aproveitando integralmente os recursos já existentes na instalação do NPD.

4 - Geração de código *assembly*

O sistema CFOR não foi idealizado para ser um sistema completo de desenvolvimento de *software* para microprocessadores (capacidade de montagem, depuração, simulação, etc), como o são os sistemas similares aos

referenciados na seção 1.0 (*EXOCISER, TWIN, CWR UNIVERSITY*), e sim, dotar os usuários de tais sistemas com a capacidade e vantagens da programação em uma linguagem de alto-nível, aproveitando integralmente o *software* de suporte existente em cada (*assembler* residente ou *cross-assembler*, *loader*, depurador, simulador, etc). Dessa forma, o sistema CFOR gera o código *assembly* equivalente ao programa fonte FORTRAN, como apresentado na figura 1.2, na forma de cartão perfurado e/ou listagem impressa, que deve ser passado para a entrada do *assembler* ou *cross-assembler* para produzir o código objeto para o específico microprocessador.

Embora essa configuração acrescente um passo extra no processo de tradução, facilita o desenvolvimento e implementação do sistema, além de permitir ao usuário possiveis refinamentos em nível de *assembly* no caso das restrições de memória e perfomance se tornarem críticas.

1.3 - DESENVOLVIMENTO DO SISTEMA

Como visto na seção anterior, o desenvolvimento do sistema consiste basicamente na definição da linguagem fonte e na organização do processo de tradução de modo a possibilitar o atendimento a toda uma classe de microprocessadores a um custo aceitável. Assim, o texto foi organizado como segue:

O capítulo II faz uma abordagem inicial acerca dos critérios adotados para a escolha do FORTRAN como linguagem fonte do sistema, seguido da descrição dos seus elementos constituintes, ressaltando as modificações introduzidas em relação à linguagem FORTRAN IV do sistema IBM/370.

O capítulo III apresenta a propriedade do processo de compilação utilizada para viabilizar a implementação do sistema, as características do código gerado pelo ANALISADOR para suportar todos os microprocessadores da classe considerada sem prejuizo da capacidade particular de qualquer dêles,e, a simplificação resultante do desenvolvimento do GERADOR como um PROCESSADOR DE MACROS.

O capítulo IV apresenta a definição do código gerado pelo ANALISADOR, sua estrutura, instruções e tabelas de informação, de acôrdo com os objetivos do sistema e das características apresentadas no capítulo anterior.

O capítulo V descreve regras e facilidades a serem seguidas e utilizadas no desenvolvimento das macros, subrotinas e demais tabelas necessárias à implementação do GERADOR .

O capítulo VI apresenta o GERADOR DE CÓDIGO como um PROCESSADOR DE MACROS, mostrando suas entradas, processamento e saída através de um fluxograma elucidativo.

O capítulo VII apresenta um resumo e comentários acêrca do expôsto, algumas considerações referentes à implementação do sistema e algumas sugestões para o seu aperfeiçoamento.

O apêndice "A" apresenta um exemplo da transformação de um programa FORTRAN para a forma intermediária, simulando a execução do ANALISADOR.

O apêndice "B" mostra as tabelas resultantes da simulação do ANALISADOR apresentada no apêndice anterior.

Capítulo II

A LINGUAGEM FONTE

2.0 - INTRODUÇÃO

Uma linguagem de programação para microprocessadores basicamente em nada difere de uma linguagem para uso nos equipamentos convencionais de computação. Deve conter os conceitos fundamentais e mais frequentemente encontrados de programação, permitir a expressão dêsses conceitos da maneira mais natural e clara possível, além de possibilitar um processamento eficiente e econômico (10).

Diversas linguagens de programação de uso generalizado como o FORTRAN, ALGOL, PL/I, BASIC, etc, possuem, em maior ou menor grau, essas características, e, podem ser utilizadas para o desenvolvimento de programas para um amplo espectro de aplicações dêsses dispositivos.

A opção pela definição de uma nova linguagem não foi considerada, dando-se preferência ao aproveitamento de uma linguagem já estabelecida, principalmente pelos inconvenientes que adveriam para os usuários do sistema: aprender a nova linguagem, dificuldade de obtenção e troca de informações, e, impossibilidade de transferência de seus programas para um outro sistema de desenvolvimento.

2.1 - ESCOLHA DO FORTRAN COMO LINGUAGEM FONTE

A escolha do FORTRAN como linguagem fonte do sistema se deu em decorrência desta linguagem satisfazer em sua totalidade uma série de requisitos ou critérios de ordem prática, considerados como essenciais tanto para a viabilidade como para a utilização efetiva do sistema no desenvolvimento de programas para microprocessadores:

- simplicidade e aceitação
- eficiência de execução
- transportabilidade
- facilidade de implementação

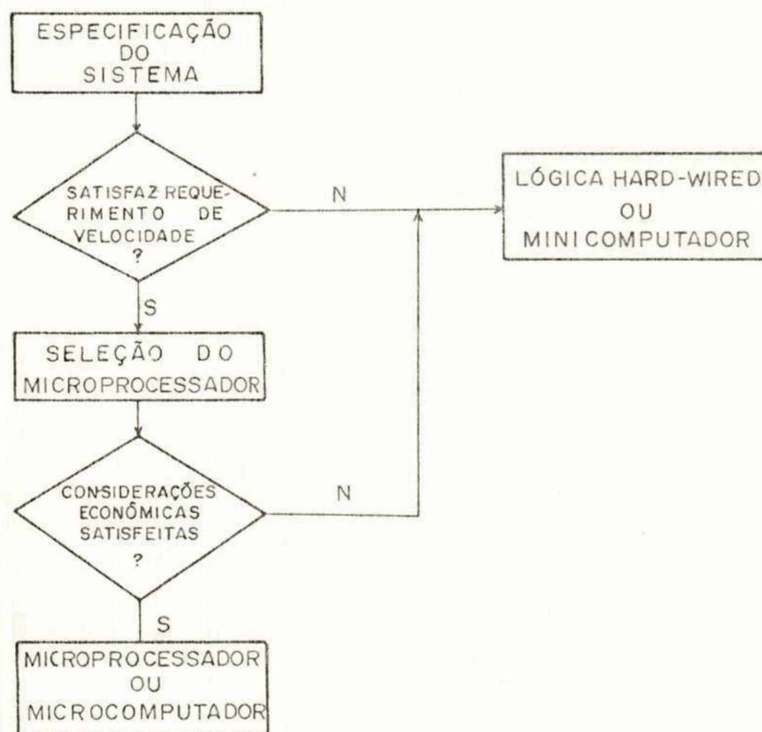
2.1.1 - SIMPLICIDADE E ACEITAÇÃO

O FORTRAN, não obstante suas estruturas de controle não permitir a expressão de algoritmos de uma maneira muito clara e natural como o é possível em linguagens tipo ALGOL e PL/I, apresenta uma definição precisa, estrutura simples e racional, além de ser uma das linguagens de maior difusão e penetração nos diversos campos de aplicação dos computadores, principalmente no meio técnico-científico onde é matéria obrigatória na maioria dos cursos de engenharia.

A sua condição de linguagem de largo uso vem determinando a sua utilização como a linguagem de desenvolvimento dos *cross-assemblers* e *cross*-compiladores oferecidos pelos fabricantes de microprocessadores, como também da maioria dos pacotes de *software* para os mais diversos fins (6 e 11).

2.1.2 - EFICIÊNCIA DE EXECUÇÃO

No processo de decisão a respeito da utilização ou não do microprocessador em substituição à lógica *hard-wired* ou, onde um microcomputador é muito dispendioso ou desnecessariamente poderoso, o principal aspecto técnico a ser considerado são os requerimentos de velocidade de operação exigidos pelo sistema em projeto (1). Como mostra o fluxograma do processo de decisão abaixo, as restrições de tempo podem se tornar tão severas a ponto de não recomendarem o seu uso:



Fluxograma do processo de decisão

Figura 2.1

É importante notar que, os parametros que servirão de apoio a esta tomada de decisão não estão ligados apenas às considerações de *hardware* (tempo de ciclo, nº de ciclos das instruções, etc), mas também à qualidade do *software* disponível. A utilização de uma linguagem de alto-nível faz com que o conceito de velocidade da máquina real evolua para o de eficiência de execução da "máquina abstrata", definida pela linguagem, na máquina real.

Eficiência de execução é a característica mais forte do FORTRAN, devido às suas estruturas simples e ao esquema estritamente estático de alocação de memória, não necessitando de nenhuma rotina de administração e alocação de memória em tempo de execução (5).

2.1.3 - TRANSPORTABILIDADE

Dentre as linguagens de programação consideradas, afora o BASIC, apenas o FORTRAN oferece essa capacidade, desde que, um grande número de novos sistemas de desenvolvimento de *software* para microprocessadores lançados recentemente no mercado apresentam o FORTRAN como linguagem fonte, com amplas perspectivas de padronização. Êste fato, além de encorajar a troca e venda de informação entre as diversas instalações, tornará possível aos usuários do sistema CFOR a transferência de seus programas para um outro sistema sem nenhuma ou quase nenhuma alteração, e vice-versa.

2.1.4 - FACILIDADE DE IMPLEMENTAÇÃO

Além dos três fatores apresentados e analizados nos paragrafos anteriores, um outro, considerado também de importância no processo de escolha da linguagem, desde que afeta dire

tamente o custo e a viabilidade do sistema, é o grau de complexidade envolvido na sua implementação. FORTRAN é uma das linguagens de maior uso que apresenta maior facilidade de desenvolvimento do compilador. A tradução de seus comandos é direta e pode ser levada a efeito na simples base de linha por linha. A única estrutura de maior complexidade é o DO, onde o compilador deve checar para que as restrições sôbre a estrutura e "ninhos de DO'S" sejam satisfeitas. O compilador faz uso intensivo de tabelas e não necessita de estruturas mais elaboradas e sofisticadas para levar a cabo o processo de tradução (5).

2.2 - DESCRIÇÃO DA LINGUAGEM

A linguagem FORTRAN considerada baseia-se na linguagem FORTRAN IV desenvolvida pela IBM para os sistemas IBM/360 e IBM/370 (12), contendo a maioria dos seus principais elementos e comandos mais utilizados, com algumas restrições introduzidas para adaptação à arquitetura e uso dos microprocessadores da classe considerada, sem no entanto, descaracterizar a sintaxe original.

Na descrição dos comandos e declarações da linguagem foi utilizada a mesma notação apresentada no documento GC 28-6515-10 publicado pela IBM para descrever a linguagem FORTRAN IV e que servirá como referência para toda e qualquer característica padrão da linguagem que não esteja explicitamente considerada no texto.

2.2.1 - CARACTERES BÁSICOS E PALAVRAS RESERVADAS

a - Conjunto de caracteres

Os caracteres válidos na linguagem consistem dos alfa-numéricos:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

dos caracteres especiais:

branco = + - / * . , ' ( )

e de qualquer dos caracteres ASCII de 21 (!) a 5A (Z) , permitidos apenas na formação de constantes de caracte_ res.

b - Palavras reservadas

Da observação de programas FORTRAN constata - se que em raríssimas ocasiões as palavras-chaves da lin_ guagem são utilizadas para dar nomes à variáveis, *arrays*, subrotinas, etc. Então, para facilitar a tarefa de reconhecimento de comandos e operações, e, possibi_ litar algumas extensões necessárias, os seguintes no_ mes são considerados palavras reservadas:

INTEGER	CALL	END	DABS
REAL	READ	EOF	FLOAT
LOGICAL	WRITE	INTRPT	DFLOAT
DIMENSION	FORMAT	DEVICE	IFIX
COMMON	CONTINUE	INICIO	HFIX
DATA	RETURN	IDENT	
DO	SUBROUTINE	IABS	
GO TO	STOP	ABS	

Além de todos os nomes válidos iniciados pela letra Z, inclusive.

2.2.2 - TIPOS DE DADOS

Os tipos de dados foram definidos em função das aplicações e da arquitetura dos microprocessadores considerados. Utilizam a mesma denominação do FORTRAN IV, porém, com representação interna diferente da original. A linguagem suporta cinco tipos elementares e um estruturado, além das constantes de caracteres permitidas apenas em mensagens programadas de saída.

2.2.2.1 - TIPOS SIMPLES

a - Inteiro-simples

Ocupa um *byte* de memória, representado internamente como um numero binário em complemento de 2.

b - Inteiro-duplo

Ocupa dois *bytes* consecutivos de memória, utilizando a representação em complemento de 2.

c - Real-simples

Os microprocessadores considerados possuem capacidade para aritmética decimal para ambos os tipos de números BCD, compactados e descompactados. Embora os dois tipos descritos anteriormente satisfaçam a maioria das aplicações numéricas, se achou por bem aproveitar a capacidade oferecida e consequentemente dar maiores opções ao usuário.

O tipo real-simples é representado internamente por digitos BCD descompactado, ocupando 5 *bytes* contiguos de memória, sendo os quatro primeiros para armazenar os digitos do número e o último para representar o sinal nos 4 *bits* de menor ordem.

d - Real-duplo

Representado internamente por digitos BCD compactados, ocupando 5 *bytes* consecutivos de memória, com o sinal representado nos 4 *bits* de menor ordem do último *byte*.

e - Lógico

Ocupa um *byte* de memória, com os dois valores lógicos representados internamente pelos hexadecimais 00 e 01.

2.2.2.2 - TIPO ESTRUTURADO

Idêntico ao *array* homogêneo do FORTRAN IV, com algumas restrições para permitir seu uso com eficiência:

- Limitado a um máximo de 2 dimensões

- Máximo número de elementos igual a 127

2.2.3 - CONSTANTES E VARIÁVEIS

a - CONSTANTES

As constantes podem ser formadas de qualquer dos ti

pos simples descritos no item 2.2.2.1 e de sequências de caracteres.

inteiro-simples: sequência de digitos decimais precedida ou não por sinal (+ ou -). Se o sinal fôr omitido, o valor é considerado positivo.

$$-128 \leq \text{valor} \leq 127$$

inteiro-duplo: extensão da anterior, com a faixa de valores se alargando para o máximo de 32767 a um mínimo de - 32768.

real-simples: sequência de até 4 digitos decimais precedido ou não por sinal (+ ou -). Se omitido o sinal, o valor é considerado positivo.

$$-9999 \leq \text{valor} \leq 9999$$

real-duplo: extensão da anterior, com a sequência se estendendo para até 9 digitos decimais.

$$-999999999 \leq \text{valor} \leq 999999999$$

lógico: .TRUE. e .FALSE.

caracteres: qualquer sequência de caracteres válidos (máximo de 255). Seu uso é restrito à declaração FORMAT e pode ser escrita nas duas formas permitidas pelo FORTRAN IV.

b - VARIÁVEIS SIMPLES

A lei de formação de nome de variáveis é idêntica a definida pelo FORTRAN IV, com restrição apenas aos caracteres básicos permitidos e indicados em 2.2.1.

c - VARIÁVEIS SUBSCRITAS

A formação do nome segue a mesma lei acima e os subscritos podem ser expressos nas seguintes formas:

$$v$$
$$c$$
$$v \pm c$$
$$c * v$$
$$c_1 * v \pm c_2$$

Onde v é uma variável inteiro-simples, sem sinal, e, c, c_1 e c_2 são constantes inteiro-simples sem sinal.

2.2.4 - OPERAÇÕES

A linguagem oferece um conjunto de operações aritméticas, relacionais e lógicas, aplicadas aos tipos de dados descritos em 2.2.2.1, além de um extensivo conjunto de operações de conversão de tipos e valor absoluto.

a - OPERAÇÕES ARITMÉTICAS

ADIÇÃO, SUBTRAÇÃO, MULTIPLICAÇÃO e DIVISÃO aplicadas a todos os tipos numéricos com o auxílio dos operadores +, -, *, /, com os operandos de um mesmo tipo.

b - OPERAÇÕES RELACIONAIS

Mesmas operações do FORTRAN IV, com os operandos de um mesmo tipo numérico e resultado do tipo lógico. Operadores:

.EQ., .NE., .LT., .GT., .LE., .GE.

c - OPERAÇÕES LÓGICAS

Idêntico ao FORTRAN IV, com os operadores .NOT., .AND., e .OR.

d - OPERAÇÕES DE CONVERSÃO DE TIPOS E VALOR ABSOLUTO

Embora os tipos de dados sejam diferentes, foi utilizada a mesma nomenclatura do FORTRAN IV para a definição das operações de conversão e valor absoluto:

OPERADOR	OPERANDO	OPERAÇÃO
FLOAT	ID	ID → RS
DFLOAT	ID	ID → RD
IFIX	RS	RS → ID
HFIX	RD	RD → ID
ABS	RS/RD	VA
IABS	IS	VA
DABS	ID	VA

* - ID - Inteiro-duplo
RS - Real-simples
RD - Real-duplo
IS - Inteiro-simples
VA - Valor absoluto

Na formação de expressões não é permitido a mistura de tipos e as mesmas leis de parentização e precedência do FORTRAN IV é assumida, com os operadores distribuidos na seguinte ordem (mais alta para mais baixa prioridade):

FLOAT, DFLOAT, IFIX, HFIX, ABS, IABS, DABS

* , /

+ , -

.EQ. , .NE. , .LT. , .GT., .LE., .GE.

.NOT.

.AND.

.OR.

2.2.5 - DECLARAÇÕES E COMANDOS

Os comandos e declarações da linguagem têm basicamente a mesma forma e significado do FORTRAN IV. As restrições e modificações estão explicitamente consideradas na descrição.

a - DECLARAÇÕES DE ESPECIFICAÇÃO DE TIPOS

As regras de declaração de variáveis do FORTRAN IV foram ligeiramente modificadas para acompanhar as diferenças de tipos entre as linguagens:

declaração implicita: se a primeira letra do nome for I, J, K, L, M ou N, a variável é considerado do tipo inteiro-simples caso contrário, real-simples.

declaração explicita: variáveis e *arrays* podem ser declaradas utilizando-se as seguintes declarações:

TIPO LISTA

onde:

TIPO : INTEGER, INTEGER*2, REAL, REAL*2, LOGICAL

LISTA : Nomes de variáveis simples ou de *arrays* (dimensionados ou não) separados por virgulas.

b - DECLARAÇÃO DIMENSION

FORMA GERAL:

DIMENSION $\underline{a}_1$ ($\underline{k}_1$), $\underline{a}_2$ ($\underline{k}_2$), $\underline{a}_3$ ($\underline{k}_3$),...,$\underline{a}_n$ ($\underline{k}_n$)

onde:

$\underline{a}_1$, $\underline{a}_2$,...$\underline{a}_n$: Nomes de *arrays*

$\underline{k}_1$, $\underline{k}_2$,...$\underline{k}_n$: formados por uma ou duas constantes inteiro-simples sem sinal, separadas por vírgulas e obedecendo as restrições estabelecidas em 2.2.2.2.

c - DECLARAÇÃO COMMON

FORMA GERAL :

COMMON/NOME DO BLOCO/LISTA

onde:

NOME DO BLOCO: Qualquer nome válido

LISTA : Nomes de variáveis simples ou de *arrays* (dimensionados ou não) separados por vírgulas. Deve concordar em número, ordem e tipo com a lista do bloco de mesmo nome declarado em uma subrotina.

d - COMENTÁRIOS

Da mesma forma que o FORTRAN IV.

e - COMANDO DE ATRIBUIÇÃO

FORMA GERAL:

$$\underline{a} = \underline{b}$$

onde:

$\underline{a}$: nome de variável simples ou subscrita

$\underline{b}$: expressão aritmética, relacional ou lógica

As seguintes regras de compatibilidade de atribuição devem ser obedecidas:

1) Se $\underline{b}$ é do tipo lógico, $\underline{a}$ deve ser necessariamente desse mesmo tipo.

2) Se $\underline{b}$ é inteiro-simples ou duplo, $\underline{a}$ deve ser de um desses tipos, com a conversão, se necessária, realizada automaticamente.

3) Se $\underline{b}$ é real-simples ou duplo, $\underline{a}$ deve ser de um desses tipos, com a conversão realizada automaticamente.

UNIVERSIDADE FEDERAL DA PARAÍBA
Pró-Reitoria Para Assuntos do Interior
Coordenação Setorial de Pós-Graduação
Rua Aprigio Veloso, 882 Tel (083) 321-7222-R 355
58.100 - Campina Grande - Paraíba

f - COMANDO DATA

FORMA GERAL:

DATA $\underline{k}_1/\underline{d}_1/,\underline{k}_2/\underline{d}_2/,\ldots,\underline{k}_n/\underline{d}_n/$

onde:

$\underline{k}_1,\underline{k}_2,\ldots,\underline{k}_n$: Lista de nomes de variáveis simples, subscritas ou de *arrays*.

$\underline{d}_1,\underline{d}_2,\ldots,\underline{d}_n$: Lista de constantes numéricas ou lógicas. Qualquer dessas constantes podem ser precedidas por i*, onde i é uma constante inteiro-simples sem sinal. Quando a forma i* aparece antes de uma constante, indica que esta deve ser especificada i vêzes.

g - COMANDOS DE CONTRÔLE DE SEQUÊNCIA

IF LÓGICO

FORMA GERAL:

IF (a) c

onde:

a : qualquer expressão relacional ou lógica

c : qualquer comando executável exceto o comando DO ou outro comando IF.

IF ARITMÉTICO

FORMA GERAL:

IF ($\underline{a}$) $\underline{n}_1$, $\underline{n}_2$, $\underline{n}_3$

onde:

$\underline{a}$: qualquer expressão aritmética.

$\underline{n}_1$,$\underline{n}_2$, $\underline{n}_3$: números de comandos da unidade contendo o comando IF

GO TO

FORMA GERAL:

GO TO $\underline{n}$

onde:

$\underline{n}$: número de um comando da mesma unidade de programação.

STOP

FORMA GERAL:

STOP

END e EOF

END indica ao compilador fim de uma unidade de programação (programa principal ou subprograma).

EOF indica fim do programa fonte.

h - COMANDO DE CONTRÔLE DE LOOP

FORMA GERAL:

DO n i = m_1, m_2, m_3

onde:

n : número do último comando do loop.

i : nome de variável simples (inteiro-simples ou duplo)

m_1, m_2, m_3 : constantes ou variáveis simples concordando em tipo com i.

2.2.6 - SUBPROGRAMAS

A linguagem aceita as duas formas principais de subprogramas do FORTRAN IV, FUNÇÃO e SUBROTINA, com restrições ao número e tipo de parametros permitidos. A transmissão de parametros obedece a mesma filosofia, por referência. Dados podem ser compartilhados através do uso da declaração COMMON.

a - DECLARAÇÃO DE FUNÇÃO

FORMA GERAL:

TIPO FUNCTION NOME * 2 (LISTA DE ARGUMENTOS SEM EFEITO)

onde:

TIPO: INTERGER, REAL ou LOGICAL

Nome: Nome da função, formado a partir da regra indicada em 2.2.3 item (b).

* 2 : essa forma só deve ser usada se a função for do tipo inteiro-duplo ou real-duplo.

LISTA : Nomes de variáveis simples ou de *arrays*, separados por virgulas. No máximo 8 argumentos são permitidos.

b - REFERÊNCIA A FUNÇÃO

A referência é feita numa expressão, na forma :

NOME (LISTA DE PARÂMETROS)

onde:

NOME : mesmo nome da declaração

LISTA: nomes de variáveis simples, subscritas ou de *arrays*, separadas por virgulas. Deve concordar em número, ordem e tipo com os correspondentes argumentos sem efeito da declaração.

c - DECLARAÇÃO DE SUBROTINA

FORMA GERAL:

SUBROUTINE NOME(LISTA DE ARGUMENTOS SEM EFEITO)

onde:

NOME : nome da subrotina formado a partir da regra indicada em 2.2.3 item (b).

LISTA : da mesma forma que indicada na declaração de FUNÇÃO. É opcional.

d - REFERÊNCIA A SUBROTINA

FORMA GERAL:

CALL NOME (LISTA DE PARÂMETROS)

onde:

NOME : mesmo nome da declaração

LISTA : da mesma forma que indicada na referência a FUNÇÃO. É opcional.

e - RETÔRNO DE SUBPROGRAMA

FORMA GERAL:

RETURN

OBS.: A mesma lei de escopo de nomes do FORTRAN IV é assumida, isto é, nomes de variáveis e *arrays* nos subprogramas e programa principal podem coincidir pois são unidades de programação independentes.

2.2.7 - ENTRADA E SAÍDA

A linguagem providencia um conjunto de subrotinas de entrada e saída desenvolvidas para cada tipo de periférico normalmente usado em aplicação de microprocessadores. Assim, o usuário não precisa se preocupar com essas operações, bastando selecionar o periférico desejado e utilizar os comandos especiais da linguagem para êsse fim, da mesma forma que nas instalações convencionais do FORTRAN. No processo de tradução, êsses comandos são convertidos em uma sequência de códigos relativamente simples, que representam basicamente uma invocação da apropriada subrotina de E/S.

UNIVERSIDADE FEDERAL DA PARAÍBA
Pró-Reitoria Para Assuntos do Interior
Coordenação Setorial de Pós-Graduação
Rua Aprígio Veloso, 882 - Tel (083) 321-7222-R 355
58.100 - Campina Grande - Paraíba

a - COMANDO DE ENTRADA DE DADOS

FORMA GERAL:

READ (n) LISTA

onde:

n : constante inteiro-simples sem sinal indicativa do periférico.

LISTA : Nomes de variáveis simples, subscritas ou de *arrays* separados por vírgulas.

b - COMANDO DE SAÍDA DE DADOS

FORMA GERAL:

WRITE (n) LISTA
ou
WRITE (n, m)

onde:

n : constante inteiro-simples sem sinal indicativa do periférico.

LISTA: Nomes de variáveis simples, subscritas ou de *arrays* separados por vírgulas.

m : Número da declaração FORMAT onde consta o texto da mensagem a ser transmitida ao periférico n.

c - DECLARAÇÃO FORMAT

FORMA GERAL:

FORMAT ('TEXTO')

ou

FORMAT (n H TEXTO)

onde:

n : número de caracteres do texto

TEXTO: sequência de caracteres válidos na linguagem.

2.2.8 - INTERRUPÇÃO

Uma extensão ao FORTRAN IV foi criada para suportar a facilidade de interrupção disponível no *hardware* dos microprocessadores considerados e frequentemente utilizada para controlar transferências de dados entre periféricos e o microprocessador.

A linguagem permite ao usuário definir uma subrotina especial de tratamento de interrupção, para a qual, após o reconhecimento realizado por uma rotina interna do periférico que requer atenção, é desviado a execução do programa. Essa subrotina dá acesso a uma variável pré-declarada DEVICE (inteiro-simples), cujo valor corrente é o número de identificação do dispositivo de E/S que está solicitando atenção. O usuário, testando essa variável, decide a ação a ser tomada através dos comandos normais da linguagem.

FORMA GERAL:

SUBROUTINE INTRPT

onde:

INTRPT : palavra reservada que identifica a subrotina.

A subrotina INTRPT não têm argumentos e cria um novo nível de hierarquia no programa. Não pode ser invocada pelo comando CALL e nem ramificar explicitamente para nenhuma unidade anterior à sua definição. Todos os subprogramas declarados após só obtêm acesso através de uma chamada ou referência localizada no seu interior, atuando assim, como um segundo programa principal. A comunicação de dados entre êsses dois níveis é estabelecido através da declaração COMMON. Na execução dessa subrotina ou de suas "dependentes" não será reconhecido nenhum pedido de interrupção, que deve aguardar até que o contrôle seja passado de volta ao ponto onde aconteceu a interrupção que originou a execução da subrotina INTRPT.

A estrutura de um programa utilizando êsse recurso fica:

```
PROGRAMA PRINCIPAL
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
END
{SUBPROGRAMAS DO
{PROGRAMA PRINCIPAL

SUBROUTINE INTRPT
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
END
{SUBPROGRAMAS DO
{"PROGRAMA PRINCIPAL"
{INTRPT
EOF
```

Capítulo III

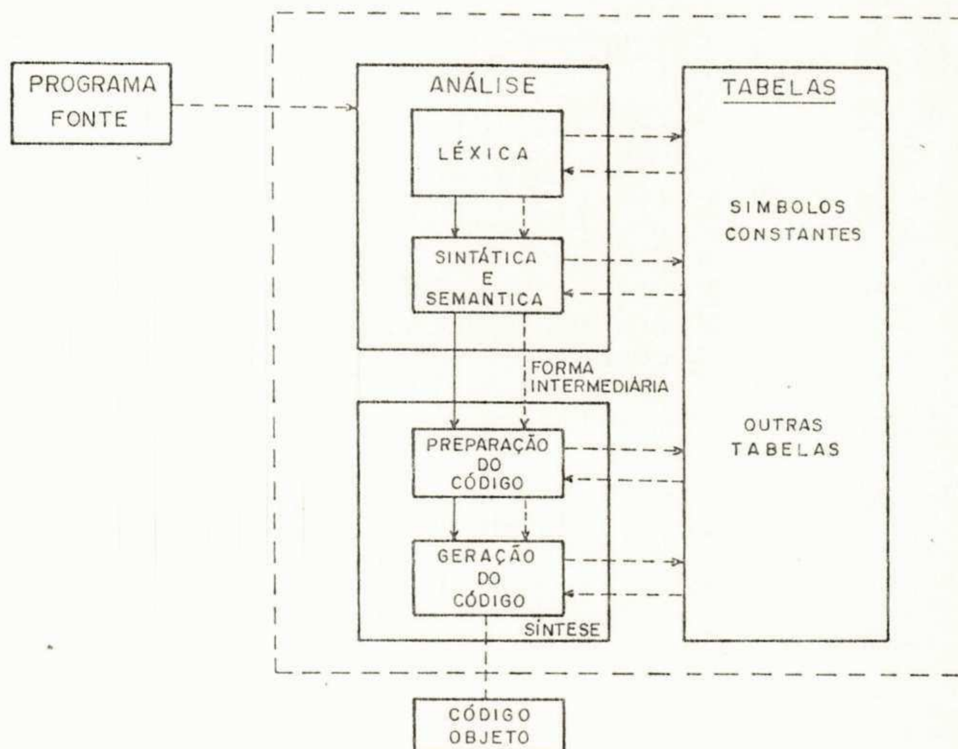
ORGANIZAÇÃO DO PROCESSO DE TRADUÇÃO

3.0 - INTRODUÇÃO

Definida a linguagem fonte do sistema, o passo seguinte é estabelecer um procedimento que torne possível a sua implementação para a classe de microprocessadores envolvida, sem a necessidade dispendiosa de se desenvolver integralmente um compilador para cada microprocessador. A solução encontrada para evitar o disperdício de um total reinvestimento foi dividir o processo de tradução em duas etapas, de forma a se obter uma padronização de grande parte do processo, acompanhada de uma simplificação considerável na sua elaboração.

3.1 - ORGANIZAÇÃO PADRÃO

Observando a organização lógica de um compilador, apresentada na figura 3.1, verifica-se que, a parte de análises léxica, sintática e semântica, com a transformação do programa fonte em uma forma intermediária, pode ser desenvolvida e implementada independentemente da máquina a ser considerada, desde que, os processos lógicos se sucedam na mesma ordem apresentada na figura (8 e 13). Obedecida esta restrição, as características de máquina (conjunto de instruções, organização dos registradores, entrada, saída, etc) só se fazem necessárias na etapa de geração do código objeto.



Organização lógica de um compilador

Figura 3.1

Aproveitando esta propriedade, se estabeleceu uma organização padrão a ser observada no desenvolvimento dos compiladores, com a divisão do processo de tradução em duas etapas:

Etapa 1: ANALISADOR

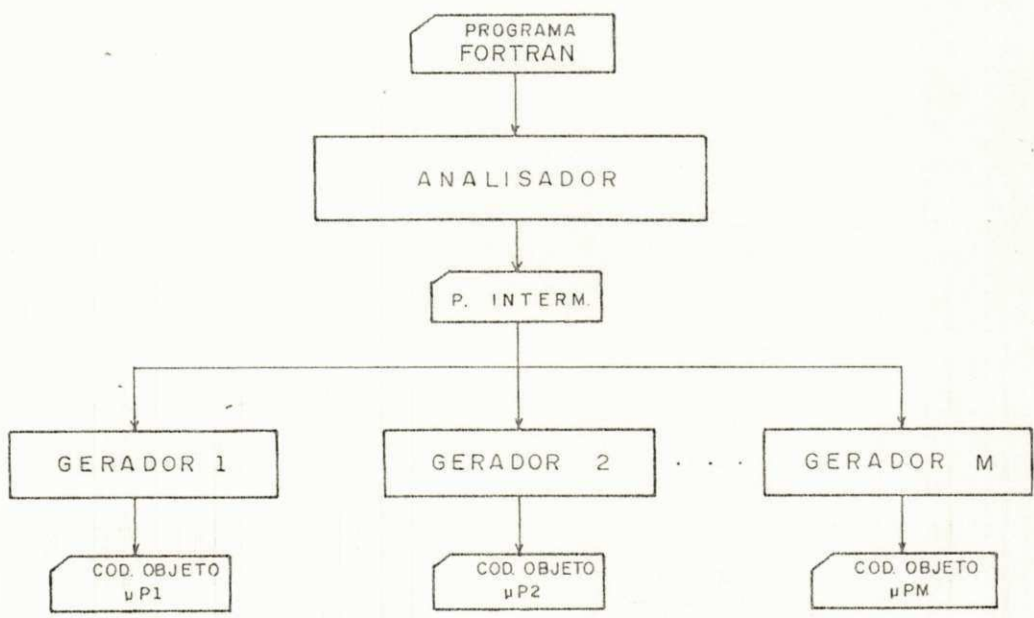
Módulo comum a todos os compiladores, responsável pela análise e decomposição do programa fonte em uma forma intermediária, com a criação das tabelas de varáveis, constantes, etc.

Etapa 2: GERADOR DE CÓDIGO

Módulo específico para cada compilador, responsável pela geração do código objeto a partir do programa intermediário e das tabelas criadas pelo ANALISADOR.

Com essa organização, o esfôrço dedicado à implantação de um compilador, à exceção do primeiro, é reduzido à elaboração da etapa de geração de código objeto, como mostra a figura 3.2 abaixo:

UNIVERSIDADE FEDERAL DA PARAÍBA
Pró-Reitoria Para Assuntos do Interior
Coordenação Setorial de Pós-Graduação
Rua Aprigio Veloso, 832 - Tel (083) 321 7222-R 355
58.100 - Campina Grande - Paraíba



Compiladores

Figura 3.2

3.2 - NÍVEL INTERMEDIÁRIO

A eficácia dessa organização depende diretamente da definição apropriada do nível intermediário. O ANALISADOR deve ser padronizado para os compiladores sem perda da flexibilidade proporcionada pelo uso dos recursos particulares de cada máquina. Para que êsse objetivo seja alcançado, o nível intermediário deve manter generalidade com respeito aos microprocessadores considerados (transparente às diferenças de arquitetura), ressaltando porém as características comuns. As principais diferenças de interêsse e que caracterizam êsses dispositivos são: conjunto de instruções (número e capacidade), organização dos registradores, e, entrada/saída.

1 - Conjunto de instruções

O conjunto de instruções e seus associados modos de endereçamento variam significativamente de um microprocessador para outro, refletindo a sua capacidade de operação e de acesso aos dados. Essa diversidade é evidenciada na comparação entre os dois extremos da faixa de microprocessadores de finalidade geral, o RCA COSMAC (CDP 1801 D e R) e o Z-80:

RCA COSMAC: Conjunto de 59 instruções operando sôbre dados exclusivamente de 8 *bits*.

Z-80 : Conjunto de 158 instruções que, quando associado aos 13 modos de endereçamento, perfazem um total de 696 diferentes códigos de operação. Além das instruções que operam com 8 *bits*, possui instruções aritméticas (adição e subtração) e de transferência que operam sôbre dados de 16 *bits* (DADD, DADH,

UNIVERSIDADE FEDERAL DA PARAÍBA
Pró-Reitoria Para Assuntos do Interior
Coordenação Setorial de Pós-Graduação
Rua Aprigio Veloso 882 Tel (083) 321 7222-R 355
58.100 - Campina Grande - Paraíba

DADB, SBC, SHLD, LHLD), e, de uma poderosa instrução de transferência interna de dados (LDDR) que permite mover qualquer número de *bytes* de uma região para outra da memória (até 32K *bytes*).

2 - Organização de registradores

Basicamente todos os microprocessadores possuem registradores de dados e de endereço, e, no mínimo um registrador apontador de pilha (*STACK-POINTER*), porém, apresentam diferenças significativas em sua organização, como mostrado na tabela abaixo:

μP / TIPO	MOS 6501	MOSTEK F8	MOSTEK 5065-1	M6800	8080	PPS-8	SIGNETICS 2650	Z-80
ACUMULADOR	1	1	3	2	1	0	0	2
INDEXADOR	2	2	0	1	0	0	0	2
USO GERAL	0	65	0	0	6	6	7	12

Organização de registradores

Tabela 3.1

Em alguns microprocessadores, como o 8080 e o o Z-80, os registradores podem ser usados como simples unidades de 8 *bits* ou como pares conjugados de 16 *bits*.

3 - Entrada/saída

Os microprocessadores variam bastante no modo e capacidade de realização das operações de entrada e saída. Alguns, como o 8080, F8, Z-80, etc, têm instruções específicas para êste fim. Outros, como o M6800 e SCMP,

utilizam as instruções normais de acesso a memória para realizar essas operações. A grande maioria tem possibilidades de transferir apenas 1 *byte* por instrução, enquanto os mais recentes, como o Z-80 por exemplo, têm instruções que permitem transferir blocos de *bytes* diretamente para qualquer região da memória, substituindo DMA (DIRECT MEMORY ACCESS) para taxas de transferências de até 125K*bytes/s*.

De acôrdo com essas considerações, uma linguagem de definição dêsse nível (linguagem intermediária) deve apresentar as seguintes características:

1 - Estrutura linear

O contrôle de sequência a nível de *assembly* de modo a que o "programa intermediário" seja uma versão linearizada do programa fonte, com as instruções localizadas na mesma ordem em que devem ser executadas.

2 - Instruções sem referência a registradores

Para evitar o estabelecimento de qualquer disciplina que iniba a particular maneira que cada máquina possui e utiliza registradores, as instruções de nível intermediário (NI) não devem fazer referência a essas unidades de trabalho, que devem ser alocadas, de acôrdo com cada caso, pelo GERADOR.

3 - Instruções de três operandos

Um código de três operandos, além de suportar o conjunto de instruções de microprocessadores mais poderosos como o Z-80 por exemplo, concentra a criação de temporárias no ANALISADOR, facilitando o trabalho do GERADOR.

4 - Operação e dados

A maioria das operações e dados da linguagem fonte não são definidos no *hardware* dos microprocessadores, devendo ser simulados por *software*. Como essa simulação não pode ser realizada com eficiência nêsse nível, pela necessidade de definição dos registradores e instruções da máquina, a linguagem intermediária deve manter a estrutura de dados elementares da linguagem fonte, isto é, instruções aritméticas, lógicas e de transferência operando sôbre dados de 1, 2 e 5 *bytes*. Essa estrutura permite tirar proveito, na etapa seguinte, das características próprias de cada máquina para realizar essas operações.

Como os microprocessadores consideram a memória como uma matriz unidimensional, a operação de acesso a elementos de *arrays* de duas dimensões da linguagem fonte deve ser substituida nêsse nível por operação de acesso linear a seus elementos.

5 - Instruções de entrada e saída

Para manter generalidade com respeito aos diferentes modos como os microprocessadores realizam essas operações, a linguagem intermediária deve conter instruções bem gerais de entrada e saída, de forma a abranger todos os tipos existentes. O código de três operandos permite a formação de tais instruções.

Com o nível intermediário apresentando essas características, o ANALISADOR pode ser padronizado para os diversos compiladores sem comprometer as potèncialidades de qualquer das máquinas consideradas, sendo então responsável pela análise e decomposição das estruturas da linguagem fonte em operações mais simples, que podem ser implementadas para to

dos os microprocessadores aproveitando todos os recursos disponíveis em cada.

3.3 - SIMPLICAÇÃO DO GERADOR

Uma simplificação considerável no desenvolvimento de um GERADOR, etapa mais longa e trabalhosa de um compilador, pode ser obtida pelo uso de uma técnica simples, que consiste em estabelecer cada instrução NI como uma chamada de uma macro previamente definida na linguagem *assembly* do específico microprocessador (8 e 14). Essa técnica só é usada com proveito se a linguagem a ser implementada apresentar uma estrutura linear, o que é o caso (8). Dessa maneira, a etapa de geração de código objeto é reduzida ao nível de um processador de macros, consistindo basicamente na substituição textual de cada instrução NI por uma instrução ou grupo de instrução equivalentes, desenvolvidas no *assembly* específico, e previamente catalogadas e arquivadas em uma tabela de definição de macros (TDM). Nenhuma rotina de administração e alocação de registradores é necessária, com o programador (implementador) com inteira liberdade de usar os registradores disponíveis da maneira mais conveniente e eficiente no desenvolvimento das instruções de nível intermediário.

Embora as instruções possam ser definidas com o mínimo código requerido para cada máquina, a implementação por macros pode levar a um código global extenso e pouco eficiente no uso da memória. Para evitar êsse disperdício, as instruções NI que apresentarem uma alta taxa de incidência e/ou exigirem um extenso código, devem ser implementadas com o auxílio de uma subrotina. A macro, nêsses casos, se limita a transferir os dados para a subrotina, que executará a operação requerida. As subrotinas devem ser definidas e armazenadas em uma tabela de definição de subrotinas (TDS), ao lado das rotinas internas de inicialização do sistema e identifica

ção de interrupção, apresentadas no capítulo V do texto.

Com a adoção dêsse método, o esforço dedicado ao desenvolvimento de um GERADOR se concentra principalmente na elaboração das tabelas TDM e TDS, visto que, um processador de macros não apresenta maiores complicações para seu desenvolvimento.

Capítulo IV

ANALISADOR

4.0 - INTRODUÇÃO

O principal aspecto considerado na definição do ANALISADOR foi o código intermediário e as tabelas de informação a serem produzidos por êste módulo. O método a ser adotado para levar a cabo as análises léxica, sintática e semântica, verificação e correção de êrros, otimização, etc, foi deixado a cargo do implementador, de acôrdo com suas conveniências e ferramentas disponíveis, tendo ainda a opção de utilização do compilador FORTRAN IV para uma fase inicial de verificação de êrros.

Êste capítulo apresenta as instruções elementares em que os comandos do programa fonte devem ser decompostos (formato e tipo de operandos), a organização do programa intermediário equivalente, e, as tabelas com as informações necessárias para a etapa seguinte de geração de código *assembly*. Ao lado da apresentação das instruções NI é feita uma abordagem acêrca da conveniência de sua implementação exclusivamente por uma macro ou com auxílio de uma subrotina.

4.1 - CÓDIGO INTERMEDIÁRIO

4.1.1 - FORMATO DAS INSTRUÇÕES

O ANALISADOR deve decompor os comandos fonte em instruções primitivas e sequenciais da forma:

LABEL	OPERADOR	OPERANDOS

onde:

LABEL : Identificador de instrução formado a partir do prefixo Z, seguido de até 5 digitos. Instruções que requerem LABEL devem ter êsse campo em branco.

OPERADOR : Mneumônico representativo da operação

OPERANDOS: Até três operandos, dependendo da instrução, com seus tipos definidos abaixo e seus formatos apresentados na figura 4.1.

1 - Nome gerado pelo ANALISADOR para as variáveis e *arrays* do programa fonte, em substituição aos nomes originais. Formados a partir do prefixo ZV, seguido de 4 digitos gerados sequencialmente, obedecendo as seguintes regras:

a - Variáveis e *arrays* com mesmo nome em unidades diferentes, levam nomes diferentes.

b - Variáveis relacionadas por COMMON levam um mesmo nome.

c - A variável DEVICE mantêm o nome original.

d - Os argumentos sem efeito mantêm o nome original.

e - As variáveis de mesmo nome de uma função, declaradas tanto na unidade em que é feita a referência quanto na própria unidade de declaração de função , levam um mesmo nome.

2 - Nome formado pelo ANALISADOR para as constantes do programa fonte. Para as constantes numéricas e lógicas, o nome deve ser formado a partir do prefixo ZC seguido de 4 digitos gerados sequencialmente. Para as constantes de caracteres das declarações FORMAT, o nome deve ser formado a partir do prefixo ZF.

3 - Nome de variáveis temporárias criadas pelo ANALISADOR, formado a partir do prefixo ZT, seguido de 4 digitos em sequência.

4 - Identificador de instrução.

5 - Constante indicativa de número de *bytes* a serem transferidos, de periférico a ser ativado ou de posição do parâmetro na lista de parâmetros do comendo CALL ou de referência a função (0, 2, 4, 6, 8, 10, 12, 14).

6 - Representação de elemento de *array*, com o nome do *array* formado segundo o ítem 1, seguido do nome do subscrito que pode ser: nome de variável, nome de constante ou de temporária, formado segundo os itens 1, 2 e 3 respectivamente.

1	D	NOME VARIÁVEL/ARRAY

2		NOME DE CONSTANTE

3		NOME TEMPORÁRIA

4		IDENTIF:DE INSTRUÇÃO

5		CONSTANTE

6	D	NOME DO ARRAY	NOME SUBSCRITO

Formato dos operandos

Figura 4.1

O campo D diferente de branco indica argumento sem efeito. Nêsse caso, pode ser : 0, 2, 4, 6, 8, 10, 12 ou 14, dependendo da posição do argumento na lista de argumentos sem efeito de uma declaração SUBROUTINE ou FUNCTION.

As variáveis subscritas de duas dimensões, por exemplo A(i, j), devem ser representadas pelo novo nome do *array* , seguido do nome de uma temporária contendo o valor da expressão inteiro-simples:

$$i * d_2 + j - d_2$$

onde d_2 é o valor limite da 2a. dimensão do *array*.

Dêsse modo, na fase seguinte, todos os *arrays* são considerados vetores, facilitando a tarefa de determinação do endereço de seus elementos.

4.1.2 - ORGANIZAÇÃO DO PROGRAMA INTERMEDIÁRIO

O programa intermediário deve manter a mesma divisão em unidades (programa principal e subprogramas) do programa fonte, porém, de acôrdo com a estrutura de um programa *assembly*, com os ENTRY-POINTS das unidades estabelecidas pelo ANALISADOR como segue:

- Programa principal: a primeira instrução N1 da sequência de códigos equivalentes ao programa principal fonte deve ter um identificador de instrução formado pela letra Z.

- Subprogramas: No nível intermediário não há distinção entre função e subrotina, com ambos os tipos se comportando exatamente como uma subrotina. A entrada dessas unidades é estabelecida por um identificador de instrução na primeira instrução da unidade, formado pelo próprio nome dado ao subprograma pelo usuário.

A última instrução do programa deve ser a pseudo-instrução END.

4.1.3 - INSTRUÇÕES NI

As instruções de nível intermediário foram definidos em função dos elementos básicos constituintes dos comandos executáveis FORTRAN. Apenas 5 tipos de instruções elementares são necessárias para a decomposição dos comandos fonte:

- Aritméticas e lógicas
- Transferência interna de dados
- Contrôle de sequência
- Contrôle de subprograma (transferência de dados e contrôle)
- Entrada e saída

1 - Aritméticas e lógicas

a - Aritméticas

Como as expressões aritméticas do programa fonte operam com diversos tipos de dados (inteiro-simples e duplo, real simples e duplo), um grupo de instrução básicas para cada é necessário:

INTEIRO SIMPLES	SIGNIFICADO
ADIS OP1 OP2 OP3	OP3 ← OP1 + OP2
SUIS	OP3 ← OP1 - OP2
MUIS	OP3 ← OP1 * OP2
DVIS	OP3 ← OP1 / OP2

INTEIRO DUPLO	SIGNIFICADO
ADID OP1 OP2 OP3	OP3 ← OP1 + OP2
SUID	OP3 ← OP1 - OP2
MUID	OP3 ← OP1 * OP2
DVID	OP3 ← OP1 / OP2

REAL SIMPLES

ADRS OP1 OP2 OP3	OP3 ← OP1 + OP2
SURS	OP3 ← OP1 - OP2
MURS	OP3 ← OP1 * OP2
DVRS	OP3 ← OP1 / OP2

REAL DUPLO

ADRD OP1 OP2 OP3	OP3 ← OP1 + OP2
SURD	OP3 ← OP1 - OP2
MURD	OP3 ← OP1 * OP2
DVRD	OP3 ← OP1 / OP2

Além das instruções para conversão de tipos e valor absoluto:

IFIX	OP1	OP2	OP2 ← IFIX(OP1)
HFIX	OP1	OP2	OP2 ← HFIX(OP1)
FLOAT	OP1	OP2	OP2 ← FLOAT(OP1)
DFLOAT	OP1	OP2	OP2 ← DFLOAT(OP1)
CONVD	OP1	OP2	OP2 ← CONVD(OP1)
CONVS	OP1	OP2	OP2 ← CONVS(OP1)
ABS	OP1	OP2	OP2 ← ABS(OP1)
IABS	OP1	OP2	OP2 ← IABS(OP1)
DABS	OP1	OP2	OP2 ← DABS(OP1)

OBS.1 - CONVD e CONVS, convertem inteiro-simples em inteiro-duplo, e, inteiro-duplo em inteiro-simples, respectivamente.

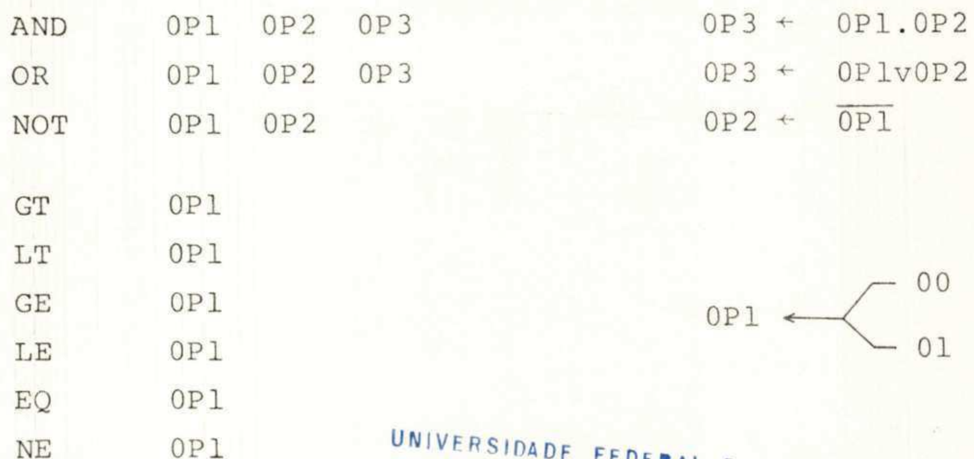
OBS.2 - Tôdas as instruções modificam o estado dos registradores de condição.(implicitamente)

OBS.3 - As instruções inteiro simples, ADID e SUID , CONVD e CONVS e as instruções de valor absoluto devem ser implementadas por macros, pois o número de instruções *assembly* necessário para a transferência dos dados para a subrotina é comparável com o número requerido para a realização da operação. As demais, pelo extenso código, requerem o auxílio de uma subrotina.

OBS.4 - A maior parte dessas rotinas já foram desenvolvidas para os mais populares microprocessadores e se encontram apresentado em publicação do gênero (15, 16 e 17).

b - Lógicas

Além das instruções lógicas básicas, algumas outras são necessárias para a decomposição das expressões relacionais do programa fonte:



AND	OP1	OP2	OP3	OP3 ← OP1.OP2
OR	OP1	OP2	OP3	OP3 ← OP1vOP2
NOT	OP1	OP2		OP2 ← $\overline{OP1}$
GT	OP1			
LT	OP1			
GE	OP1			OP1 ← 00 / 01
LE	OP1			
EQ	OP1			
NE	OP1			

UNIVERSIDADE FEDERAL DA PARAÍBA
Pró-Reitoria Para Assuntos do Interior
Coordenação Setorial de Pós-Graduação
Rua Aprigio Veloso, 882 - Tel (083) 321-7222-R 355
58.100 - Campina Grande - Paraíba

Essas instruções testam o estado dos registradores de condição e dependendo dêste, armazenam no endereço especificado por OP1 o hexadecimal 00 ou 01.

OBS.1 - Tôdas as instruções modificam o estado dos registradores de condição.

OBS.2 - Essas instruções operam com dados de 1 *byte* e exigem pouco código *assembly* (de 3 a 5 instruções), devendo ser implementados exclusivamente por macros.

2 - Transferência de dados

Embora as transferências internas possam ser de 1,2 ou 5 *bytes*, apenas uma instrução é requerida para realizar essas operações visto que o código intermediário permite instrução de três operandos:

MOVE OP1 OP2 OP3

onde:

OP1 - especifica o endereço de origem dos dados
OP2 - especifica o endereço de destino dos dados
OP3 - constante indicativa do nº de *bytes* a serem transferidos

OBS - Essa instrução deve ser implementada exclusivamente por Macro, em vista de:

1 - Pouca incidência no programa intermediário otimizado

2 - A possibilidade de expansão condicional permite que apenas o código estritamente necessário para cada tipo seja selecionado.

3 - Contrôle de sequência

a - Desvio condicional

BRPO	OP1	Desvio se $>$ 0
BRNE	OP1	$<$ 0
BRZE	OP1	$=$ 0
BRNZ	OP1	$\leq$ 0
BRPZ	OP1	$\geq$ 0

O desvio depende do resultado da operação aritmética ou lógica anterior, indicado pelo estado dos registradores de condição.

b - Desvio incondicional

BRA OP1

Desvia incondicionalmente para a instrução identificada por OP1.

c - Especiais

NOP

STOP

OBS - Essas instruções devem ser implementadas por macros face a correspondência 1 a 1 com instruções de máquina.

4 - Contrôle de subprogramas

a - Ativação de subprograma

PARM OP1 OP2

```
SALVA
CALL    OP1
```

A instrução PARM armazena o endereço descrito por OP1 numa área pré-declarada ZLISTA de 16 *bytes*. O segundo operando, OP2, indica a posição em ZLISTA em que o endereço deve ser armazenado. Como um endereço ocupa 2 *bytes*, OP2 pode ser 0, 2, 4, 6, 8, 10, 12 ou 14, dependendo da posição do parâmetro na lista de parâmetro do comando CALL ou de referência à função.

A instrução SALVA, salva o conteúdo de ZLISTA em uma pilha pré-declarada ZSTACK. Essa instrução só é necessária se houver parâmetros a serem transmitidos, e, se isso acontece, deve ser gerada antes das instruções PARM requeridas, permitindo que um subprograma referencie um outro, sem perda dos endereços da chamada anterior.

A instrução CALL transfere o contrôle para o subprograma de nome especificado por OP1, salvando o endereço de retôrno na pilha ZSTACK.

OBS - As instruções CALL e PARM devem ser implementadas exclusivamente por macros pois na maioria dos casos exigem de 1 a 2 instruções de máquina, respectivamente.

A instrução SALVA, embora requera apenas de 7 a 10 códigos de máquina consome um tempo muito maior que o necessário para ramificar e retornar de uma subrotina. Como o tempo de ativação e desativação é irrelevante em face do tempo de execução, o espaço de memória deve ser economizado através da implementação com auxílio de subrotina.

DFPb / BIBLIOTECA / PRAI

b - Desativação de subprograma

RESTRA

RETS

RETI

A instrução RESTRA realiza o inverso de SALVA, isto é, restaura o último conteúdo de ZLISTA armazenado em ZSTACK, antes do retôrno.

As instruções RETS e RETI substituem o comando fonte RETURN em contextos diferentes:

RETS - Retorna o contrôle para o ponto subsequente à ativação da subrotina.

RETI - Retôrno da subrotina INTRPT, restaurando todos os registradores salvos na pilha quando da ativação dessa subrotina por uma interrupção. Não exige a instrução RESTRA.

OBS - As instruções RETS e RETI na maioria dos microprocessadores são realizadas apenas por 1 instrução de máquina, devendo ser implementadas por macros. A instrução RESTRA, pelos mesmos motivos que SALVA deve ser implementada com auxílio de subrotina.

5 - Entrada e saída

A possibilidade de três operandos numa instrução NI, permite a formação de instrução de entrada e saída transparentes tanto às limitações dos microprocessadores referentes a essas operações, quanto aos diferentes

modos como as realizam, também permitindo que transferências possam ser realizadas sob contrôle do periférico. As instruções para entrada/saída especificam um bloco de dados, seu endereço e o periférico a receber ou transmitir o bloco:

READ OP1 OP2 OP3

WRITE OP1 OP2 OP3

onde:

OP1 - constante indicativa do periférico
OP2 - especifica o endereço do bloco
OP3 - constante indicativa do número de *bytes* do bloco

OBS - A macro READ ou WRITE apenas passa o endereço da área descrito por OP2 e o número de *bytes* especificado por OP3 para a subrotina selecionada pela constante em OP1 que vai controlar ou simplesmente ativar, dependendo do dispositivo, a transferência de dados entre êste e a memória do sistema ou vice-versa. A expansão da macro é condicional, dependendo do valor da constante, uma chamada para a determinada subrotina é anexada ao programa.

4.2 - TABELAS DE INFORMAÇÕES

O ANALISADOR deve fornecer na saída, tabelas contendo informações sôbre as variáveis, *arrays*, constantes e temporárias, necessárias para a etapa de geração de código *assembly*.

4.2.1 - TABELA DE VARIÁVEIS (TV)

Nessa tabela devem ser declarados todas as variáveis e *arrays* do programa fonte, com a seguinte organização:

NOME	# BYTES	TIPO

Tabela de variáveis
Tabela 4.1

onde:

NOME - Nome formado pelo ANALISADOR para as variáveis simples e *arrays*, como descrito em 4.1.1.

BYTES - Número de *bytes* de uma variável simples ou número de *bytes* ocupado por um *array*.

TIPO - Êsse campo deve ser preenchido apenas para *arrays*, indicando o tipo de seus elementos. Caso contrário, branco.

1 se interio-simples ou lógico.

2 se inteiro-duplo.

5 se real-simples ou duplo.

4.2.2 - TABELA DE VARIÁVEIS TEMPORÁRIAS (TVT)

Nessa tabela deve constar tôdas as variáveis temporárias criadas pelo ANALISADOR, com a simples organização:

NOME	# BYTES

Tabela de variáveis temporárias
Tabela 4.2

onde:

NOME : Nome formado pelo ANALISADOR como descrito em 4.1.1

BYTES: Número de *bytes* da variável

4.2.3 - TABELA DE CONSTANTES NUMÉRICAS E LÓGICAS (TCNL)

Tabela contendo tôdas as constantes numéricas e lógicas do programa fonte, com a seguinte organização:

NOME	# BYTS	VALOR	TIPO

Tabela de constantes numéricas e lógicas

Tabela 4.3

onde:

NOME - Nome formado pelo ANALISADOR como descrito em 4.1.1

BYTES- Número de *bytes* da constante

VALOR - Valor da constante (se lógica, 0 ou 1)

TIPO - 1 se inteiro-simples ou lógica

2 se inteiro-duplo

3 se real-simples

4 se real-duplo

UNIVERSIDADE FEDERAL DA PARAÍBA
Pró-Reitoria Para Assuntos do Interior
Coordenação Setorial de Pós-Graduação
Rua Aprigio Veloso, 882 - Tel (083) 321-7222-R 355
58.100 - Campina Grande - Paraíba

4.2.4 - TABELA DE CONSTANTES DE CARACTERES(TCC)

Tabela formada a partir das constantes de caracteres das declarações FORMAT:

NOME	# CARACTERES	TEXTO

Tabela de constantes de caracteres
Tabela 4.4

onde:

NOME : Nome formado pelo ANALISADOR como descrito em 4.1.1

#CARACTERES: Número de caracteres da constante

TEXTO: Sequência de caracteres da mensagem

Um exemplo da transformação de um programa fonte em seu correspondente programa intermediário, com a equivalência das variáveis e constantes e tabelas de informação geradas, é apresentado nos apêndices "A" e "B".

Capítulo V

MACROS E SUBROTINAS

5.0 - INTRODUÇÃO

O grau de complexidade envolvido na programação das macros e subrotinas, que substituirão as instruções NI na etapa de geração de código, depende diretamente da potência do conjunto de instruções do especifico microprocessador. Para alguns, como o Z-80 por exemplo, o desenvolvimento das instruções NI na linguagem *assembly* do microprocessador não apresenta maiores dificuldades, visto que, grande parte dessas instruções são implementadas diretamente na máquina, necessitando apenas a transferência dos dados para os registradores exigidos pela instrução, ao contrário de outros, onde o manuseio exclusivamente com 8 *bits* acarreta um grau maior de dificuldade no desenvolvimento das instruções NI e a necessidade de introdução de alguns recursos extras para permitir ao processador de Macros a seleção apenas do código estritamente necessário à sua execução.

Êste capítulo apresenta as regras que devem ser seguidas e os recursos a serem utilizados, se necessário, no desenvolvimento das macros e subrotinas, como também a organização das tabelas TDM, TDS e tabelas auxiliares. Para uma melhor compreensão do texto, alguns exemplos são apresentado desenvolvidos na linguagem *assembly* do microprocessador M6800.

5.1 - MACROS

No desenvolvimento das macros, o programador deve seguir as regras e utilizar os recursos descritos abaixo:

1 - Os argumentos a serem substituidos devem ser escritos com um símbolo especial não-existente no *assembly* específico (&, por exemplo para o M6800), seguido do número que indica sua posição na macro-instrução (NI). As posições são estabelecidas como:

Instrução NI :	LABEL	OPERADOR	OP1	OP2	OP3
Posição :	1		2	3	4

As instruções *assembly* do texto que podem ser a primeira do grupo a ser expandido, devem ser identificadas com o símbolo especial (&) seguido do número 1 no campo de label. Assim, a macro ADIS é escrita como:

```
&1 LDA A  &2
   ADD A  &3
   STA A  &4
```

2 - As macros que fazem uso de desvios internos devem utilizar para tanto identificadores de instrução formado a partir do símbolo (&) seguido de uma letra ou sequência de letras (máximo 5).

Ex.:

```
       - - - - - - - -
       - - - - - - - -
       - - - - - - - -
&LABEL - - - - - - - -
       - - - - - - - -
       BCC    &LABEL
       - - - - - - - -
```

Êsses identificadores serão substituidos pelo Processador de Macros por identificadores padronizados.

3 - Qualquer literal que se faça necessário ao desenvolvimento da Macro pode ser criado pelo programador, de acordo com as regras da linguagem *assembly* utilizada.

4 - Além da variável ZLISTA utilizada no desenvolvimento da macro PARM, o programador pode criar variáveis internas, formadas a partir do prefixo ZVI, e, utilizadas na transferência de dados para a subrotina associada. Essas variáveis correspondem às áreas de dados das subrotinas e são definidas em uma tabela de variáveis internas (TVI), ao lado de ZSTACK, ZLISTA e DEVICE.

Ex.: A macro MUID passa os dados para a subrotina associada SUB1 e recebe o resultado da operação através das variáveis ZVI1, ZVI2 e ZVI3, respectivamente:

```
&1  LDA A   &2      }
    STA A   ZVI1    }
    LDA A   &2+1    }  dados são transferidos pa
    STA A   ZVI1+1  }  ra a subrotina através de
    LDA A   &3      }  ZVI1 e ZVI2
    STA A   ZVI2    }
    LDA A   &3+1    }
    STA A   ZVI2+1  }
    JSR     SUB1    }  contrôle passa p/subrotina
    LDA A   ZVI3    }
    STA A   &4      }  resultado é transferido
    LDA A   ZVI3+1  }  para a macro através   de
    STA A   &4+1    }  ZVI3
```

A subrotina SUB1 é desenvolvida considerando os dados em ZVI1 e ZVI2 e armazenando o resultado da operação em ZVI3.

5 - A última instrução de uma definição de macro deve ser a pseudo-instrução MEND.

6 - Em algumas macros, como MOVE, READ e WRITE, as instruções da definição a serem executadas dependem do valor de um operando, como número do periférico, e número de *bytes* a serem transferidos. Se os testes e desvios forem realizados em tempo de execução, perde-se tempo e espaço de memória para armazenar códigos que nem sempre são necessários. Para se evitar êsse disperdício, o programador pode utilizar as pseudos instruções AIF e AGO que tornam possível a seleção do código no tempo de expansão, omitindo o desnecessário.

AIF (& ARG OPERADOR C) LABEL

AGO LABEL

onde:

OPERADOR: EQ ou NE (igual ou não-igual)

C : constante inteira sem sinal

LABEL : identificador de uma linha do texto, formado a partir de uma letra seguida de letras ou digitos (máximo 6)

A pseudo AGO dirige o processador incondicionalmente para a linha do texto identificada pelo LABEL.

A pseudo AIF dirige o processador para a linha do texto identificada pelo LABEL se a expressão relacional entre parênteses fôr verdadeira. Caso contrário, o processamento continua na linha seguinte.

Ex. 1: MACRO MOVE

```
      &1  LDA A    &2               } Transfere o primei-
          STA A    &3               } ro ou único byte
          AIF      (&4 EQ 1)FIM ──→ Se um byte,fim
          LDA A    &2+1             } Transfere o segundo
          STA A    &3+1             } byte
          AIF      (&4 EQ 2)FIM ──→ Se dois bytes,fim
          LDA A    &2+2             } Transfere os bytes
          STA A    &3+2             } restantes.
            .        .              } (real simples ou du-
          LDA A    &2+4             } plo)
          STA A    &3+4
  FIM     MEND
```

Ex. 2: MACRO READ

```
      &1  LDX      #&3              } Transfere o endereço da
          LDAA     &4               } área e o nº de bytes
          AIF      (&2 NE 1)Z1      }
          JSR      SUB1             }
          AGO      FIM              }
  Z1      AIF      (&2 NE 2)Z2      } Seleciona a subroti-
          JSR      SUB2             } na de acôrdo com o
          AGO      FIM              } periférico
  Z2      AIF      (&2 NE 3)Z3      }
          - - - - - - - - - - - - - }
          - - - - - - - - - - - - - }
  FIM     MEND
```

<u>OBS</u>: Êsses recursos são desnecessários no desenvolvimento dessas macros para microprocessadores como o Z-80, por exemplo, visto ter instruções de máquina equivalentes às instruções NI especificadas.

7 - As rotinas de determinação do endereço de elemento de *array* ou argumento sem efeito, podem ser sistematizadas e desenvolvidas como macros, visto que as informações necessárias são bem determinadas e disponíveis ao processador: nome do *array*, do subscrito, posição do endereço em ZLISTA e informações da própria linha sendo processada. O programador desenvolve a macro em função dessas informações. O processador ao analisar uma linha do texto e verificar que o argumento a ser substituido é um elemento de *array* ou *dummy argument*, coleta os dados necessários numa ordem estabelecida e gera implicitamente uma chamada para a macro específica de calculo de endereço. Ao terminar a expansão da nova macro, retorna à tarefa de expansão da macro original na linha subsequente.

Assim, duas macros são adicionadas às anteriormente definidas: macro ARRAY e a macro DUMMY.

8 - Um registrador (ou acumulador) deve ser escolhido para uso dedicado às macros ARRAY e DUMMY para evitar a destruição de qualquer valor intermediário, visto que essas macros são expandidas no interior de uma macro "normal".

5.2 - TABELA DE DEFINIÇÃO DE MACRO (TDM)

O programador deve escrever as macros de acôrdo com as regras definidas no parágrafo anterior, linha por linha, e considerando a organização abaixo:

LABEL DE LINHA	T E X T O			INDICADOR
	LABEL	CÓDIGO	OPERANDO	

Tabela de definição de macro
Tabela 5.1

onde:

LABEL DE LINHA : identificador de linha do texto. Usado em combinação com as pseudos AIF e AGO.

TEXTO : campo de definição das instruções *assembly*.

INDICADOR : Êsse campo deve ser preenchido com qualquer caracter diferente de branco se na linha correspondente deve haver uma substituição em qualquer dos sub-campos do texto. Caso contrário, ou se pseudo-instrução, deve ser branco.

A macro MOVE seria definida como:

	LABEL DE LINHA	TEXTO			INDICADOR
		LABEL	COD.OP.	OPERANDO	
1	.	.	.	.	.
2	.	.	.	.	.
	.	.	MEND	.	.
30	.	&1	LDAA	&2	*
31	.	.	STAA	&3	*
32	.	.	AIF	(&4 EQ 1)FIM	
33	.	.	LDAA	&2+1	*
34	.	.	STAA	&3+1	*
35	.	.	AIF	(&4 EQ 2)FIM	
36	.	.	LDAA	&2+2	*
37	.	.	STAA	&3+2	*
38	.	.	LDAA	&2+3	*
39	.	.	STAA	&3+3	*
40	.	.	LDAA	&2+4	*
41	.	.	STAA	&3+4	*
42	FIM	.	MEND	.	.
43	.	.	.	.	.

Associada a essa tabela, deve ser criada uma tabela de nome de macro (TNM) cuja chave é o nome da macro e a informação um apontador ou índice para o início da sua definição em TDM:

NOME	APONTADOR OU INDICE P/TDM
.	-
.	-
MOVE	30
.	-
.	-

Tabela de nome de macro

Tabela 5.2

5.3 - SUBROTINAS

As subrotinas, como não sofrem nenhum processamento , sendo anexados exatamente como se encontram definidas na tabela de definição de subrotinas (TDS), não exigem praticamente regras adicionais além das estabelecidas pelo *assembly* original. Porém, algumas disciplinas se fazem necessárias:

1 - Os identificadores utilizados para desvios de sequência devem ser formados a partir do prefixo ZLS e devem ser únicos. O programador deve ter o cuidado de não repetir um identificador em uma outra subrotina.

2 - Tôdas as subrotinas devem ser identificadas com um nome no campo de label da primeira instrução da definição, formado a partir do prefixo ZSB (exceção feita às subrotinas de inicialização e identificação de interrupção que têm nomes INICIO e IDENT, respectivamente).

3 - Apenas fazem referência às variáveis internas: ZSTACK, ZLISTA, DEVICE e as utilizadas como área de dados, ZVI ..., descritas no parágrafo 5.1. Em microprocessadores que endereçam periféricos como locação de memória, como o M6800 o prefixo ZP deve ser usado para formação de nome dos registradores de contrôle e de dados das interfaces.

4 - A última instrução de uma subrotina deve ser a pseudo-instrução MEND.

Além das subrotinas associadas às macros aritméticas, SALVA/RESTRA, e READ/WRITE, duas subrotinas adicionais são necessárias: inicialização do sistema e identificação de interrupção:

INICIALIZAÇÃO

A subrotina INICIO é a primeira parte do programa a ser executado quando o sistema é ativado e tem a função de inicializar os registradores de contrôle das interfaces, o STACK-POINTER e o registrador de interrupção.

Os registradores das interfaces são inicializados de acôrdo com os periféricos estabelecidos, o STACK-POINTER com o endereço de ZSTACK e o registrador de interrupção de forma a aceitar pedidos de interrupção. A última instrução *assembly* dessa subrotina deve ser um desvio incondicional para Z.

IDENTIFICAÇÃO DE INTERRUPÇÃO

A identificação de uma interrupção é efetuada pela subrotina IDENT, para onde deve ser desviado qualquer solicita_

ção de serviço dos periféricos. A subrotina identifica o periférico e armazena na variável DEVICE o seu número, ramificando então incondicionalmente para INTRPT.

A subrotina deve ser escrita de acôrdo com o modo particular em que cada microprocessador aceita e identifica as interrupções. Todos os registradores devem ser salvos na pilha e o registrador de interrupção deve ser colocado em estado de "não-aceita" novo pedido até que o serviço requisitado seja concluido.

Ex.: Se os periféricos 1 e 2 estão conectados aos lados A e B respectivamente de uma PIA, e os registradores de controle ZPACRA e ZPACRB indicam interrupção pela mudança de estado do seu primeiro *bit* (*bit 7*), a seguinte rotina de *POLLING* identifica o periférico:

```
INICIO   LDAA   ZPACRA
         BPL    ZLS1
         LDAA   #$1
         STAA   DEVICE
         JMP    INTRPT
ZLS1     LDAA   ZPACRB
         BPL    ZLS2
         LDAA   #$2
         STAA   DEVICE
         JMP    INTRPT
ZLS2     - - - - - - -
         - - - - - - -
```

5.4 - TABELA DE DEFINIÇÃO DE SUBROTINA (TDS)

As subrotinas são definidas em TDS de acôrdo com as regras do *assembly* especifico e as descritas no paragrafo anterior. A tabela tem a organização:

LABEL	COD.OP.	OPERANDO

Tabela de definição de subrotina
Tabela 5.3

Da mesma forma que para TDM, deve-se ter uma tabela de nome de subrotina (TNM), cuja chave é o nome e a informação um apontador ou índice para o início de sua definição em TDS. Um campo extra (requisição) é necessário, para o processador indicar qual subrotina foi requisitada para fazer parte do código objeto *assembly*.

NOME	REQUISIÇÃO	APONTADOR OU ÍNDICE P/TDS

Tabela de nome de subrotina
Tabela 5.4

O campo "requisição" inicialmente é branco e o processador indica com um caracter qualquer se a subrotina for exigida.

5.5 - TABELA DE VARIÁVEIS INTERNAS(TVI)

O programador, após o desenvolvimento de tôdas as macros e subrotinas necessárias, declara as variáveis internas utilizadas, além de ZLISTA,ZSTACK e DEVICE, em uma tabela de va

UNIVERSIDADE FEDERAL DA PARAÍBA
Pró-Reitoria Para Assuntos do Interior
Coordenação Setorial de Pós-Graduação
Rua Aprígio Veloso, 832 - Tel (083) 321-7222-R 355
58.100 - Campina Grande - Paraíba

riáveis internas. A declaração é de acôrdo com o específico *as sembly*.

EX: M6800

LABEL	COD.OP.	OPERANDO
ZLISTA	RMB	16
DEVICE	RMB	1
	RMB	49
ZSTACK	RMB	1
ZVI1	RMB	1
ZVI2	RMB	2

Tabela de variáveis internas
Tabela 5.5

Capítulo VI

GERADOR DE CÓDIGO

6.0 - INTRODUÇÃO

O algoritmo de geração de código pode ser sistematizado para todos os microprocessadores considerados, visto que as características de cada máquina são inteiramente introduzidas através das tabelas apresentadas no capítulo anterior (TDM, TDS, etc).

Êste capítulo apresenta a organização do gerador de código *assembly*, suas tabelas e a descrição de seu funcionamento através de um fluxograma detalhado e elucidativo.

6.1 - ORGANIZAÇÃO DO GERADOR

O código objeto *assembly* a ser entregue na saída do gerador é formado a partir das seguintes fontes:

1 - Substituição das instruções NI pelas correspondentes definições armazenadas em TDM. Essa tarefa requer um processador de macros.

2 - Subrotinas requisitadas pelas instruções NI, armazenadas em TDS. Não requer processamento, sendo anexadas diretamente ao código objeto, de acôrdo com a solicitação indicada em TNS.

3 - Declaração das variáveis internas armazenadas em TVI. Essas declarações se encontram na forma exigida pelo específico *assembly* e são anexadas diretamente ao código objeto.

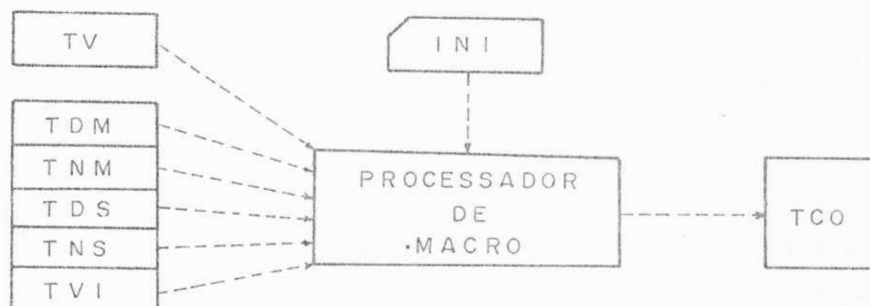
4 - Variáveis e constantes do programa fonte mais as variáveis temporárias criadas pelo ANALISADOR, armazenados nas tabelas TV, TCNL, TCC e TVT, respectivamente. Essas variáveis e constantes devem ser declaradas de acôrdo com as diretivas do específico *assembly* e anexadas ao código objeto, completando a sua formação.

Como os itens 2 e 3 não exigem processamento, o GERADOR DE CÓDIGO pode ser dividido em duas partes:

1 - Processador de Macros

Nessa parte as instruções NI são substituídas, uma a uma, pelas definições correspondentes em TDM, formando a tabela código objeto (TCO). Após o processamento, as subrotinas requisitadas e as declarações em TVI são diretamente anexadas à TCO, completando todo o código referente às instruções executáveis, mais o código de declaração das variáveis internas. A tabela TCO tem a mesma organização que TDS e TVI.

O processador faz uso das tabelas TDM, TDS, TNM e TNS, e, da tabela de variáveis (TV) formada pelo ANALISADOR (informação sôbre *arrays*).



Processador de macros
Figura 6.1

2 - Gerador de declaração

Essa parte tem a função de gerar as declarações das variáveis e constantes das tabelas TV, TVT. TCNL e TCC, a partir de padrões de diretivas do específico *assembly* fornecidas através de uma tabela de diretivas(TD), completando a formação da tabela de código objeto. O gerador de declarações, após concluida a tarefa, dá saída ao conteúdo de TCO através de uma listagem e/ou cartões perfurados.

As diretivas necessárias são: reservar áreas na memória, formar constantes numéricas de um e dois *bytes* (constantes lógicas são consideradas numéricas de 1 *byte*) e formar constantes de caracteres.

Os padrões para o M6800 seriam:

&NOME RMB &Nº de *bytes*

Utilizada para declaração das variáveis em TV e TVT.

&NOME FCB &VALOR

P/declaração das constantes de 1 *byte* em TCNL.

&NOME FDB &VALOR

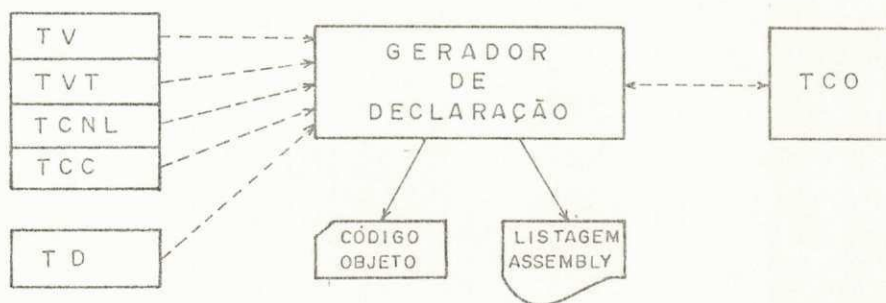
P/declaração das constantes de 2 *bytes* em TCNL.

&NOME FCC &Nº *bytes*, &TEXTO

P/declaração das constantes em TCC.

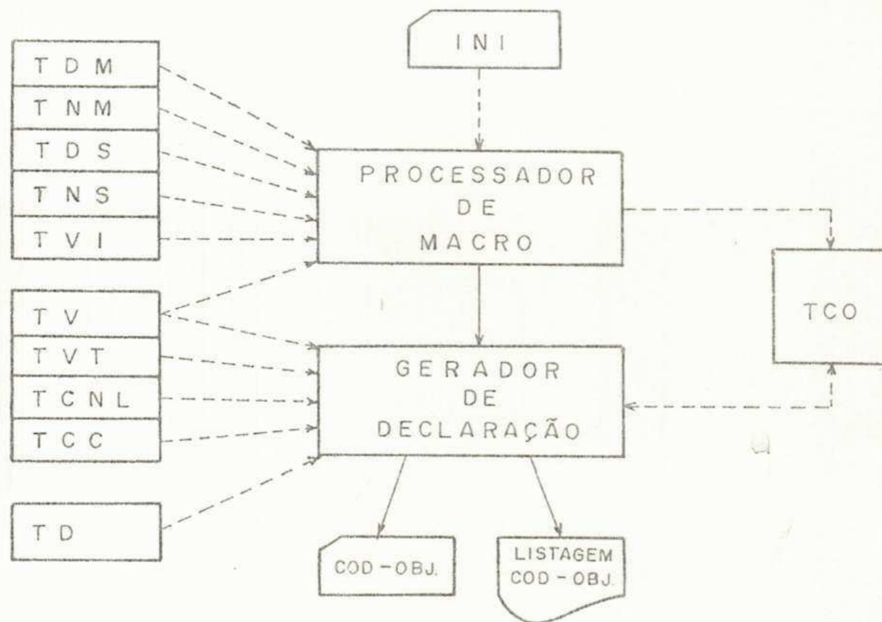
As constantes BCD (compactado e descompactado), não tendo diretivas específicas para suas declarações, seriam antes transformadas em hexadecimal ou binário e declaradas a partir do padrão:

&NOME FCB $*byte*5, $*byte*4, &*byte*3, $*byte*2, $*byte*1.



Gerador de declaração
Figura 6.2

O GERADOR DE CÓDIGO tem assim a seguinte configuração:



Gerador de código
Figura 6.3

6.2 - O PROCESSADOR DE MACROS

6.2.1 - TABELAS E VARIÁVEIS

Além das tabelas indicadas no parágrafo anterior, o processador de macros constroi e utiliza as seguintes tabelas e variáveis:

Tabela de argumentos (TA)

O processador ao analisar uma instrução NI, prepara

uma tabela de argumentos a partir dos campos de label e de operandos da instrução e, da informação sôbre o tipo do *array*, se o operando fôr um elemento de *array*. A tabela tem apenas 4 elementos, com a seguinte organização:

	ARGUMENTO	SUBSCRITO	D	TIPO
1				
2				
3				
4				

Tabela de argumentos
Tabela 6.1

onde:

ARGUMENTO : Preenchido com o label e operandos da instrução NI, na ordem estabelecida em 5.1.

SUBSCRITO : Nome do subscrito se o operando fôr elemento de *array*. Caso contrário permanece branco.

D : Campo D do operando

TIPO : Tipo do *array*, se o operando fôr elemento de *array*. Caso contrário, permanece branco.

O processador substitui os argumentos da definição a partir das informações em TA.

Tabela de argumentos simplificada(TAS)

O processador ao substituir um argumento da definição pela correspondente informação em TA, verifica se o operando é elemento de *array* ou argumento sem efeito. Caso uma dessas situações se verifique, coleta as informações necessárias em

TA e na linha da definição sendo analisada, e, prepara uma tabela de argumentos simplificada, a partir da qual, os argumentos de definição das macros *ARRAY* e *DUMMY* são substituídos. A tabela não tem campos adicionais ao de argumento, apresentando a simples organização:

	ARGUMENTO
1	
2	
-	-
-	-

Tabela de argumentos simplificada
Tabela 6.2

Tabela de Labels internos (TLI)

O processador ao encontrar um label interno em uma linha de definição, como descrito em 5.1, declara-o numa tabela de labels internos, gerando um label padrão equivalente a partir do prefixo ZLI. A tabela tem a organização:

LABEL INTERNO	LABEL PADRÃO

Tabela de labels internos
Tabela 6.3

Como o processador ao expandir uma macro pode suspender temporariamente sua expansão para processar uma macro de

endereço (*ARRAY* ou *DUMMY*), essa tabela deve ser salva em uma cópia (TLIC) e restaurada posteriormente para a continuação da expansão original.

Apontador (índice) para TDM(APTDM)

Indica a próxima linha do texto a ser analisada. Para cada Macro, é inicializada com o apontador (índice) p/TDM em TNM. Da mesma forma que TLI, deve ser salvo em APTDMC e restaurado após *ARRAY* ou *DUMMY*.

Apontador (índice) para TCO(APTCO)

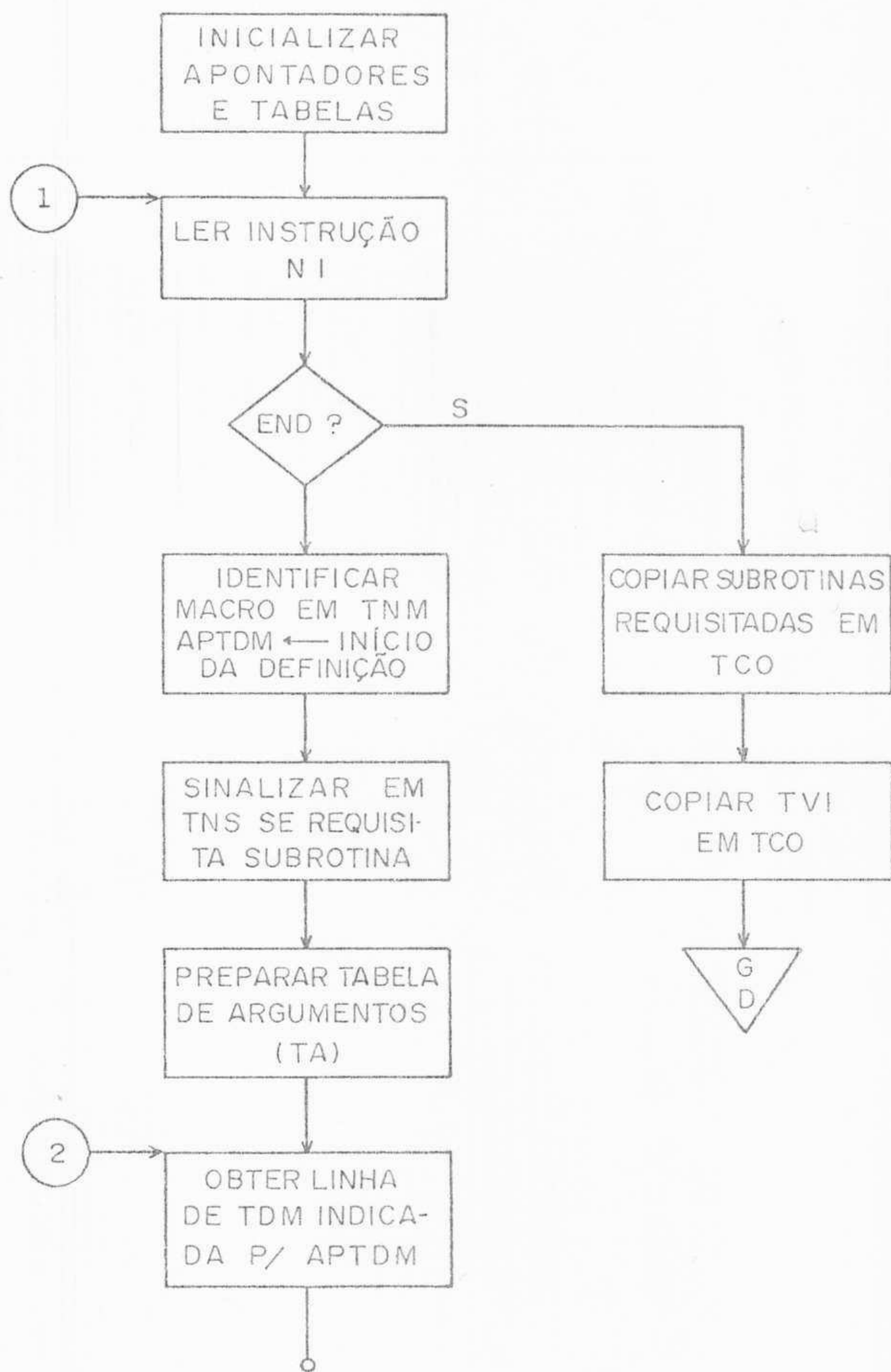
Indica a próxima entrada em TCO a ser preenchida. É inicializada para a primeira entrada e após cada inclusão, aponta para a entrada subsequente.

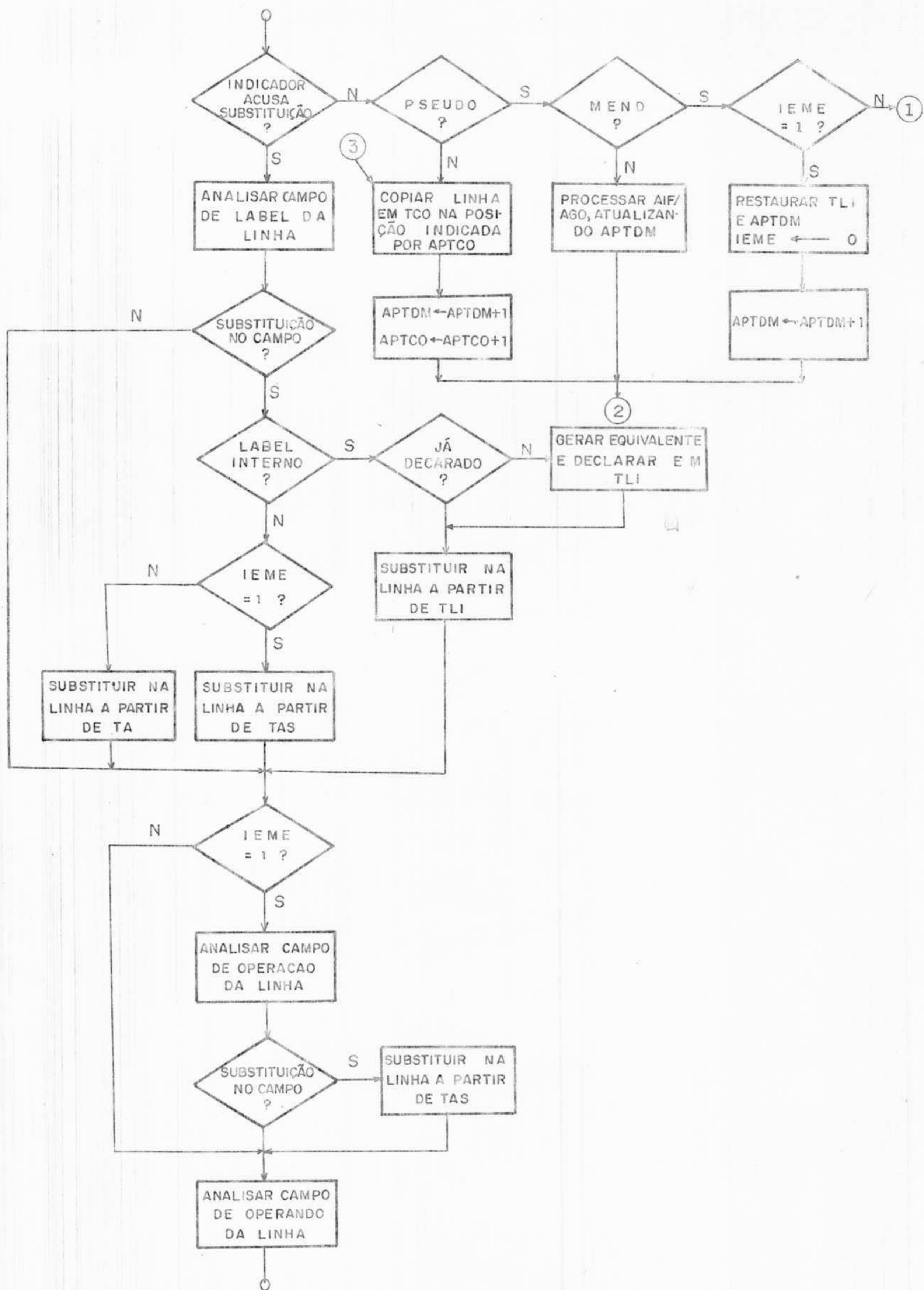
Indicador de expansão de macro de endereço(IEME)

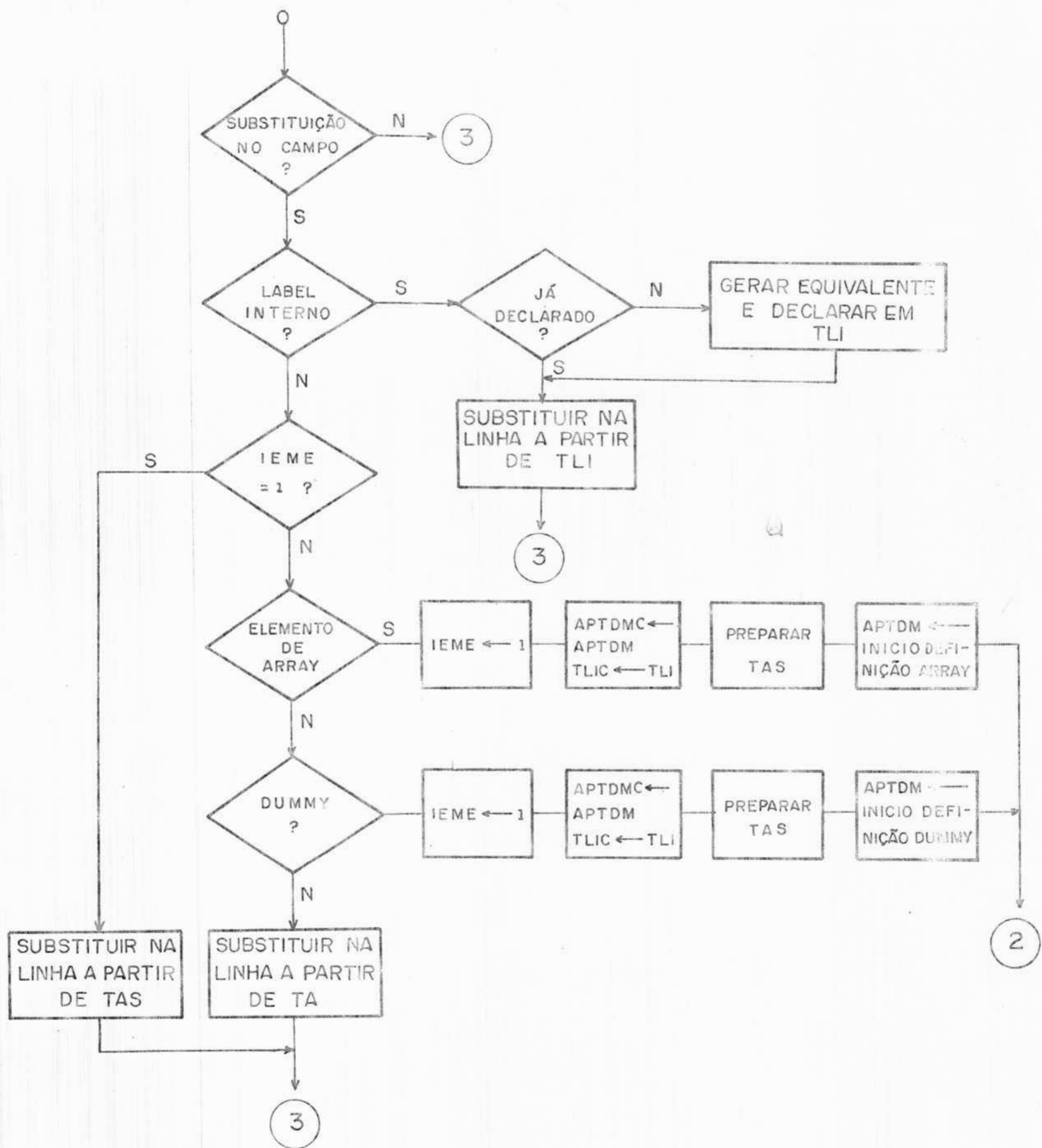
Essa variável apresenta dois estado (0 ou 1), de acôrdo com o tipo de macro sendo processada. É inicializada com zero. Ao suspender uma expansão para processar uma macro de endereço, o processador modifica o seu estado. Ao término da expansão de *ARRAY* e *DUMMY*, volta ao estado original, permanecendo assim até que uma das duas macros seja solicitada.

6.2.2 - FLUXOGRAMA DO PROCESSADOR

O fluxograma a seguir descreve suscintamente as etapas de obtenção de dados e de execução do processador de macros:


INICIALIZAR APONTADORES E TABELAS
1
LER INSTRUÇÃO NI
END ?
S
IDENTIFICAR MACRO EM TNM APTDM ⟵ INÍCIO DA DEFINIÇÃO
COPIAR SUBROTINAS REQUISITADAS EM TCO
SINALIZAR EM TNS SE REQUISITA SUBROTINA
COPIAR TVI EM TCO
PREPARAR TABELA DE ARGUMENTOS (TA)
G D
2
OBTER LINHA DE TDM INDICADA P/ APTDM

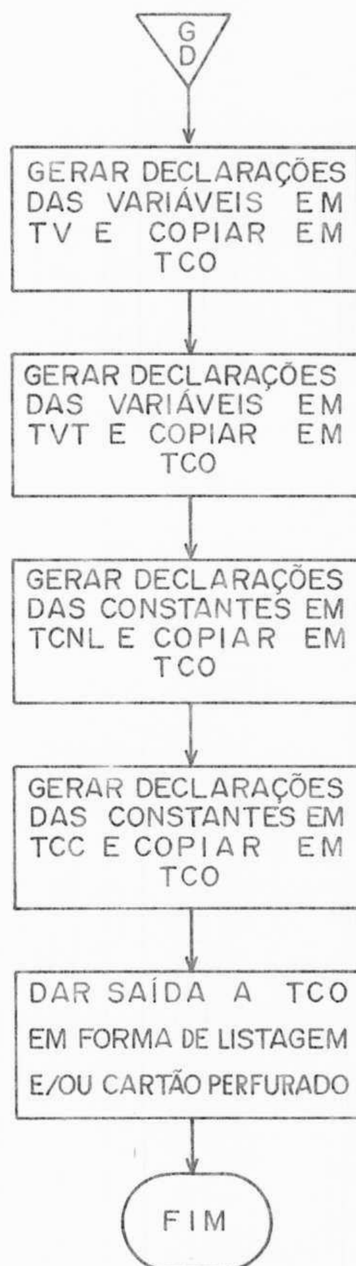

INDICADOR ACUSA SUBSTITUIÇÃO ?
N
S
PSEUDO ?
S
N
MEND ?
S
N
IEME = 1 ?
N
1
S
3
ANALISAR CAMPO DE LABEL DA LINHA
COPIAR LINHA EM TCO NA POSIÇÃO INDICADA POR APTCO
PROCESSAR AIF/AGO, ATUALIZANDO APTDM
RESTAURAR TLI E APTDM IEME ← 0
APTDM ← APTDM+1
APTCO ← APTCO+1
APTDM ← APTDM+1
SUBSTITUIÇÃO NO CAMPO ?
N
S
2
LABEL INTERNO ?
S
JÁ DECARADO ?
N
GERAR EQUIVALENTE E DECLARAR EM TLI
N
SUBSTITUIR NA LINHA A PARTIR DE TLI
IEME =1 ?
N
S
SUBSTITUIR NA LINHA A PARTIR DE TA
SUBSTITUIR NA LINHA A PARTIR DE TAS
IEME = 1 ?
N
S
ANALISAR CAMPO DE OPERACAO DA LINHA
SUBSTITUIÇÃO NO CAMPO ?
S
SUBSTITUIR NA LINHA A PARTIR DE TAS
ANALISAR CAMPO DE OPERANDO DA LINHA



Fluxograma do processador de macros
Figura 6.4

6.3 - GERADOR DE DECLARAÇÃO

Como visto em 6.1, o gerador de declaração apenas complementa a formação do código objeto, gerando as declarações apropriadas para as constantes e variáveis das tabelas TV, TVT, TCNL e TCC, de acôrdo com as diretivas padrões fornecidas em TD.



Fluxograma do gerador de declaração

Figura 6.5

Capítulo VII

CONCLUSÕES

7.0 - INTRODUÇÃO

Êste capítulo apresenta um sumário crítico do expôsto, tece algumas considerações acêrca da implementação do sistema projetado, e introduz algumas sugestões no intuito de seu aperfeiçoamento.

7.1 - SUMÁRIO

Nêste projeto nós temos demonstrado que, explorando a capacidade de uma máquina poderosa e dotada de inúmeros recursos de programação, fazendo uso objetivamente de propriedades e definições da teoria dos compiladores, e, aproveitando em grande parte experiências anteriores, um sistema de apoio para o desenvolvimento de programas para uma ampla faixa de aplicação e tipos de microprocessadores, oferecendo aos usuários a capacidade e vantagens de programação em uma linguagem de alto-nível, pode ser desenvolvido e implementado a um custo aceitável, em tempo e recursos humanos.

No capítulo II nós começamos com uma apreciação acêrca de qual linguagem seria mais conveniente para ser adotada como linguagem fonte do sistema e concluimos que o FORTRAN, entre outras citadas, reunia maiores condições para atender a uma série de requisitos propostos e considerados como fundamentais tanto para a comodidade do usuário, expressa na sua pretensa familiaridade com a linguagem e possibilidade de troca de informação e programas com outros sistemas, para maior

UNIVERSIDADE FEDERAL DA PARAÍBA
Pró-Reitoria Para Assuntos do Interior
Coordenação Setorial de Pós-Graduação
Rua Aprigio Veloso, 882 - Tel (083) 321 7222-R 355
58.100 - Campina Grande - Paraíba

eficiência de execução de programas típicos de microprocessa_ dores, como para maior facilidade de implementação do siste _ ma. Evidentemente, a linguagem descrita, um *sub-set* do FORTRAN IV, limita a faixa de aplicação do sistema, desde que não ofe rece tipos de dados e operações voltados para aplicação com caracteres (*strings*), porém, os tipos numéricos e lógicos de que dispõe possibilitam um grande espectro de aplicações. A extensão à linguagem FORTRAN IV introduzida para tratamento de interrupção, embora adicione um tempo extra na identifica_ ção do periférico, permite ao usuário a definição em alto- ní vel das ações a serem levadas a efeito.

No capítulo III nós definimos a organização do proces so de tradução de modo a que o custo de desenvolvimento dos compiladores não inviabilize o projeto. A organização assumi da decorre da possibilidade de separação do processamento das análises léxica, sintática e semântica da etapa de geração do código objeto, seccionando-se assim o processo de tradução em uma etapa independente e outra dependente da máquina, com a primeira sendo padronizada e compartilhada por todos os compi ladores, tornando o custo total de desenvolvimento do sistema tão mais baixo quanto maior fôr o número de microprocessado _ res implementados. Mostramos ainda, que para o sistema aten der a tôda a faixa de microprocessadores de finalidade geral, o código gerado na primeira fase (código intermediário) deve manter uma generalidade com respeito a todas as máquinas con sideradas e apresentar algumas características que tanto per mita o aproveitamento integral de tôdas as potencialidades de qualquer das máquinas como simplifique a etapa de geração de código objeto.

No capítulo IV apresentamos o formato e tipos de ope_ randos das instruções de nível intermediário, com a modifica_ ção ou substituição dos nomes das variáveis e *arrays* do pro grama fonte para nomes padronizados e definidos em função do objetivo de geração do código objeto *assembly*. A estrutura do programa intermediário com a definição dos seus pontos de

entrada foi apresentada, e a seguir, a definição das instruções necessárias para a decomposição dos comandos fonte .... FORTRAN em sequências de códigos com as características descritas no capítulo III e uma apreciação a respeito de sua implementação por macros ou com auxílio de subrotinas.

No capítulo V apresentamos as regras e recursos disponíveis ao programador das macros e subrotinas que, segundo o expôsto, segue em linhas gerais as mesmas diretrizes e orientações de qualquer texto de teoria do desenvolvimento de macros/subrotinas. A definição das macros/subrotinas dependem substancialmente do microprocessador utilizado, com bastante facilidade para aqueles dotados de um poderoso conjunto de instruções e maior complexidade para aqueles mais simples e de menor capacidade dentre a faixa considerada. As tabelas TDM, TDS e TVI apresentam uma organização bastante simplificada e um grande número de exemplos complementam a informação necessária para o seu desenvolvimento.

Finalmente, no capítulo VI definimos detalhadamente a organização do GERADOR DE CÓDIGO, com a apresentação de tôdas as suas tabelas e dos fluxogramas que descrevem o seu funcionamento.

7.2 - IMPLEMENTAÇÃO DO SISTEMA

O modo como o sistema foi definido permite que seja implementado em módulos, de acôrdo com a necessidade que naturalmente vai se apresentando de ampliar a capacidade de programação para novos microprocessadores. Inicialmente o módulo mais importante é o ANALISADOR, que, após a sua conclusão e catalogação na biblioteca de programas do sistema operacional, terá consumido a maior parte do esforço a ser dedicado à implementação do sistema. Os geradores de código, vão depender da disponibilidade de microprocessadores e de *assemblers ou cross-assemblers* existentes, não apresentando maio

res dificuldades no seu desenvolvimento e podem ser elaborados em paralelo com o desenvolvimento das macros, subrotinas e demais tabelas associadas.

7.3 - SUGESTÕES

Algumas modificações podem ser introduzidas no sentido de aperfeiçoar o sistema projetado, sem alterar os princípios e objetivos básicos propostos:

- Adição de maiores recursos à linguagem apresentada. Por exemplo, as estruturas de contrôle do WATFIV podem ser introduzidas para oferecer aos usuários a possibilidade de expressar seus algoritmos numa forma mais clara do que o permitido pelo FORTRAN, sem perda das principais vantagens desta:

  1 - O WATFIV é largamente difundido.

  2 - Não altera a eficiência de execução exigida.

  3 - Não aumenta o grau de complexidade da implementação.

  4 - O compilador WATFIV pode ser utilizado em substituição ao FORTRAN IV numa fase inicial de detecção de erros.

- Apesar de uma otimização intensiva ser realizada no nível do ANALISADOR, algumas redundâncias podem ocorrer na saida da etapa de geração de código objeto, como por exemplo pares tais como:

STORE X	ou	PUSH
LOAD X		POP

Sugerimos que seja feita uma otimização no código *assembly* gerado, através de um módulo com esta finalidade inserido na saída do PROCESSADOR DE MACROS, que, além da eliminação de instruções redundantes, deve verificar e eliminar variáveis temporárias desnecessárias, contribuindo para o aumento da performance e diminuição das necessidades de memória.

- A etapa de geração de código pode ser realizada apenas por um GERADOR, desde que se disponha de uma forma sistemática de definição das caractrísticas individuais de cada *assembly*, visto que as análises dos campos das instruções *assembly*, preparação de TAS (para as macros *DUMMY e ARRAY*) e, a otimização sugerida acima, dependem do *assembly* do específico microprocessador. À falta desta ferramenta, é preferível se desenvolver um GERADOR para cada máquina como estabelecido, mesmo porque o seu custo não é elevado e exige relativamente pouco tempo para elaboração do programa.

APÊNDICE A

A.1 - PROGRAMA FONTE EXEMPLO

```
      INTEGER X(10), Y(10), W(10,10), DESLCT, PE
      COMMON/MATRIZ/W
10    READ(1)W
      CALL VETORS(X,Y)
      READ(2)DESLCT
      PE = PRDESC(X,Y) + DESLCT
      IF(PE.LT.0)WRITE(20)PE
      GO TO 10
      END

      INTEGER FUNCTION PRDESC(X,Y)
      INTEGER X(10), Y(10)
      PRDESC = 0
      DO  10  I = 1, 10
10    PRDESC = PRDESC + X(I) * Y(I)
      RETURN
      END

      SUBROUTINE VETORS(A,B)
      INTEGER A(10), B(10), W(10,10)
      COMMON/MATRIZ/W
      DO  10  I = 1, 10
      J = 10 - I + 1
      A(I) = W(I,I)
10    B(I) = W(I,J)
      RETURN
      END
      EOF
```

A.2 - PROGRAMA INTERMEDIARIO EQUIVALENTE

```
Z       READ    1          ZV3       100
        SALVA
        PARM    ZV1        0
        PARM    ZV2        2
        CALL    VETORS
        READ    2          ZV4       1
        SALVA
        PARM    ZV1        0
        PARM    ZV2        2
        CALL    PRDESC
        ADIS    ZV6        ZV4       ZV5
        SUIS    ZV5        ZC1       ZT1
        BRPZ    Z15
        WRITE   20         ZV5       1
Z15     BRA     Z

PRDESC  MOVE    ZV6        ZC1       1
        MOVE    ZV7        ZC2       1
Z18     MUIS    X(ZV7)     Y(ZV7)    ZT1
        ADIS    ZV6        ZT1       ZV6
        ADIS    ZV7        ZC2       ZV7
        SUIS    ZV7        ZC3       ZT1
        BRNZ    Z18
        RESTRA
        RETS

VETORS  MOVE    ZV8        ZC2       1
Z26     SUIS    ZC3        ZV8       ZT1
        ADIS    ZC2        ZT1       ZV9
        MUIS    ZV8        ZC3       ZT1
        ADIS    ZV8        ZT1       ZT1
        SUIS    ZT1        ZC3       ZT1
        MOVE    ZV3(ZT1)   A(ZV8)    1
```

```
MUIS    ZV8        ZC3      ZT1
ADIS    ZV9        ZT1      ZT1
SUIS    ZT1        ZC3      ZT1
MOVE    ZV3(ZT1)   B(ZV8)   1
ADIS    ZV8        ZC2      ZV8
SUIS    ZV8        ZC3      ZT1
BRNZ    Z26
RESTRA
RETS
END
```

A.3 - EQUIVALÊNCIA DE VARIÁVEIS E CONSTANTES

ZV1 = X
ZV2 = Y
ZV3 = W
ZV4 = DESLCT
ZV5 = PE
ZV6 = Retôrno do valor da função PRDESC
ZV7 = I da função PRDESC
ZV8 = I da subrotina VETORS
ZV9 = J

ZC1 = constante 0
ZC2 = " 1
ZC3 = " 10

APÊNDICE B

B.1 - TABELA DE VARIÁVEIS

NOME	#BYTES	TIPO
ZV1	10	1
ZV2	10	1
ZV3	100	1
ZV4	1	
ZV5	1	
ZV6	1	
ZV7	1	
ZV8	1	
ZV9	1	

B.2 - TABELA DE VARIÁVEIS TEMPORÁRIAS

NOME	#BYTES
ZT1	1

B.3 - TABELA DE CONSTANTES NUMÉRICAS E LÓGICAS

NOME	#BYTES	VALOR	TIPO
ZC1	1	0	1
ZC2	1	1	1
ZC3	1	10	1

BIBLIOGRAFIA

1 - WEN C. LIN, *Microprocessor-Based Digital System Design Fundamentals and the Development Laboratory for Hardware Designers and Engineering Executives*, Proceedings of the IEEE, Vol. 65, nº 8, Agosto 1977.

2 - R.E. SEVIORA e J. HOMERO. CAVALCANTI, *Projeto Rápido de Software de Desenvolvimento para Microprocessadores*, Centro de Ciências e Tecnologia, Universidade Federal da Paraíba.

3 - GARY A. KILDALL, *High-Level Language Simplifies Microcomputer Programming*, Microprocessors, Electronics Book Series , 1975.

4 - L. ALTMAN e S.E. SCRUPSKI, *Software Becomes the Real Challenge*, Applying Microprocessors, Electronics Book Series, 1976.

5 - T.W. PRATT, *Programming Languages: Design and Implementation* , Prentice-Hall, 1975.

6 - JOHN DOERR, *Low-Cost Microcomputing: The Personal Computer and Single-Board Computer Revolutions*, Proceedings of the IEEE, Vol. 66, Nº 2, Fevereiro 1978.

7 - ROBERT D. CATTERTON e G.S. CASILLI, *Universal Development System is aim of Master-Slave Processors*, Applying Microprocessors, Electronics Book Series, 1976.

8 - DAVID GRIES, *Compiler Construction for Digital Computers*, John Wiley & Sons, 1971.

9 - GIUSEPPE MONGIOVI, *A Preliminary Investigation of Block Structured Languages for Several Microcomputers*, Tese de M.Sc. University of Waterloo, 1977.

10 - NIKLAUS WIRTH, *Programação Sistemática*, Editora Campus, 1978.

11 - ROBERT LEWANDOWISKI, *Preparation: The Key to Success with Microprocessors*, Electronics, Electronics Book Series, 1975.

12 - IBM, *IBM Systems/360 and System/370 FORTRAN IV Language*, GC 28-6515-10, Maio 15, 1974.

13 - A.V. AHO e J.D. ULLMAN, *The Theory of Parsing, Translation and Compiling*, Vol. 1, Prentice-Hall, 1972.

14 - JOHN J. DONOVAN, *Systems Programming*, McGraw-Hill Book Co., 1972.

15 - BRANKO SOURCEK, *Microprocessors and Microcomputers*, John Wiley & Sons Inc., 1976.

16 - JOHN B. PEATMAN, *Microcomputer-Based Design*, Mc-Graw-Hill Book Co., 1977.

17 - JOHN L. HILBURN, *Microcomputers, Microprocessors Hardware, Software and Applications*, Prentice-Hall, 1976.

18 - ARPAD BARNA e D.J. PORAT, *Introduction to Microcomputers & Microprocessors*, John Wiley & Sons, 1976.

19 - MOTOROLA, *M6800 Microprocessor Programming Manual*, Motorola Inc., 1975.

20 - MOTOROLA, *MPL/I Language Reference Manual*, Motorola Inc. , 1975.

21 - INTEL, *8008 and 8080 PL/M Programming Manual*, Intel Corporation, 1975.

22 - JACK EMMERICKS, *Implementing then Tiny Assembler*, BYTE , Vol. 2, Nº 5, Maio 1977.

23 - JAN LEMAIR e ROBERT NOBIS, *Complex Systems are Simple to Design*, Electronic Design, Vol. 26, Nº 18, Setembro 1, 1978.