1. Publicação nº	2. Versão	3. Data	5. Distribuição
INPE-3269-RPI/108		*Set., 1984*	☒ Interna ☐ Externa ☐ Restrita

4. Origem	Programa
DCA/DEA	*LASIDA/PSDA*

6. Palavras chaves - selecionadas pelo(s) autor(es)
LINGUAGEM DE MICROPROGRAMAÇÃO *TRADUTOR*
MICROPROGRAMAÇÃO
MICROCÓDIGO

7. C.D.U.: *681.322.0LMP*

8. Título *INPE-3269-RPI/108*

LINGUAGEM DE MICROPROGRAMAÇÃO LMP:
MANUAL DE USUÁRIO
VERSÃO 1.0

10. Páginas: *50*

11. Última página: *A.11*

12. Revisada por

Marcos A. Cardoso Cruz

9. Autoria *Wilson Yamaguti*

13. Autorizada por

Nelson de Jesus Parada
Diretor Geral

Assinatura responsável

14. Resumo/Notas

Apresesenta-se o subconjunto da linguagem de micro programação LMP aceito pelo tradutor LMP versão 1.0 disponível no computador Burroughs B6800. A linguagem LMP é descrita utilizando a notação BNF, com exemplos visando auxiliar a geração de microcódigos pelo usuário.

15. Observações

ABSTRACT

The LMP microprogramming language is presented in this work, according to the translator LMP version 1.0 available at Burroughs B6800 computer. The LMP language is described using BNF notation with some examples for user's reference purpose.

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

A linguagem de microprogramação LMP (Yamaguti, 1981) foi elaborada com o intuito de prover uma ferramenta de geração e documentação de microcódigos para o desenvolvimento de sistemas microprogramados.

Este manual de referência descreve os recursos da linguagem LMP aceitos pelo tradutor LMP versão 1.0, residente no computador Burroughs B6800/INPE.

1.1 - ESTRUTURA DA LINGUAGEM LMP

Um microprograma descrito em linguagem LMP é constituído por declarações (DECL), comandos (CMD) e microcomandos (MCMD), conforme mostra a Figura 1.

```
BEGIN
    %
    % MICROPROGRAMA FONTE
    %
    % DECLARAÇÃO DE MEMÓRIA
    %
    DECLMEM;
    %
    % DEMAIS DECLARAÇÕES
    %
    DECL; DECL;
    :
    DECL;
    %
    % COMANDOS E MICROCOMANDOS
    %
    CMD;  CMD;  ...  CMD;
    MCMD, MCMD, ...  MCMD;
    :
    MCD

END
```

Fig. 1 - Estrutura de um microprograma em LMP.

A linguagem pode ser encarada como um instrumento que serve para transferir configurações binárias a registros de tamanhos variáveis (campos). Cada uma dessas transferências deve corresponder a uma microoperação executável pelos recursos funcionais do processador que deve ser microprogramado. Um dado conjunto de transferências (cada uma delas é um microcomando) pode ser agrupado e delimitado por ";", gerando uma microinstrução composta de microoperações executáveis simultaneamente.

Recursos de macro foram inseridos na linguagem LMP através da declaração DEFINE e sua invocação.

1.2 - CONSTANTES E EXPRESSÕES

As constantes podem representar tanto uma configuração binária como um número decimal equivalente. Elas são utilizadas em vários pontos do microprograma fonte. Estas constantes podem ser expressas em várias bases:

1) binária;

2) octal;

3) decimal;

4) hexadecimal.

A diferença básica entre a configuração binária e o número é que a primeira representa uma configuração de bits com tamanho definido, enquanto a segunda é vista como um valor numérico que tem uma representação equivalente em bits, cujo tamanho não é definido de forma explícita.

As expressões foram inseridas na linguagem de forma a oferecer facilidades quanto ao manuseio de endereços de microprograma via rótulos, de identificadores de campo, ou mesmo de valores numéricos.

1.3 - DECLARAÇÕES

O tradutor LMP versão 1.0 é uma ferramenta de uso geral que visa a geração de microcódigos para diversos sistemas microprogramados. Essa diversidade de aplicações faz com que seja necessária a descrição de certas características do circuito a ser microprogramado.

Assim sendo, as declarações se destinam à descrição dessas características, bem como à das entidades que descrevem as facilidades de programação inseridas na linguagem.

A descrição da memória de controle é feita pela declaração de memória de controle, onde implicitamente se define o tamanho da palavra de controle. No caso, uma palavra de controle corresponde a uma microinstrução.

A nomenclatura dos campos de uma microinstrução é feita através da declaração de campo. A declaração de microinstrução permite associar um identificador a uma dada microinstrução.

As demais declarações da linguagem LMP procuram oferecer as facilidades encontradas usualmente nas linguagens de programação. Essas facilidades se referem às declarações de rótulos e de definições.

1.4 - COMANDOS E MICROCOMANDOS

Os comandos são as entidades que realmente geram os microcódigos, fazendo uso das entidades declaradas. A cada comando corresponde a geração de uma palavra de controle.

O termo microcomando foi introduzido com a finalidade de associar, ao nível de linguagem, uma ou mais microoperações. Desta for

ma um comando é constituído por um conjunto de microcomandos, de maneira análoga a uma microinstrução, que é constituída pelo conjunto de microoperações simultâneas. Cada microoperação atua sobre um recurso específico do sistema a ser microprogramado.

Em síntese, cada microcomando provê recursos de atribuição de valores a campos da palavra de controle.

1.5 - OPÇÕES DE CONTROLE

As opções de controle permitem o gerenciamento da memória de controle quanto à alocação dos microcódigos gerados. Essas opções permitem, ainda, o controle de listagem do microprograma fonte.

A execução dessas opções foram atribuídas ao Analisador Léxico, que executa a chamada de rotinas de tratamento de cada uma das opções. A partir daí, pode-se dizer que há possibilidade de elas aparecerem em qualquer ponto do microprograma fonte.

2. DESCRIÇÃO DA LINGUAGEM LMP

A linguagem LMP é definida, em termos de sua sintaxe, através da notação BNF ("Backus Naur Form"), que possibilita a definição de uma linguagem de maneira formal e clara.

Comentários e exemplos são acrescentados ao final de cada declaração ou comando.

2.1 - COMPONENTES DA LINGUAGEM

Sintaxe:

```
<componentes da linguagem> :: = <símbolos básicos>|
                                <palavras reservadas>|
                                <constantes>|
```

<comentários>|
<identificador>|
<microprograma>|
<invocação de definição>|
<opções de controle>

2.1.1 - SÍMBOLOS BÁSICOS E PALAVRAS RESERVADAS

Sintaxe:

<símbolos básicos> :: = <letra>|<dígito>|<ponto>|
<caracteres especiais>

<letra> :: = A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|X|Y|W|Z|
<dígito> :: = 0|1|2|3|4|5|6|7|8|9|
<ponto> :: = .
<caracteres especiais> :: = ; |<espaço>| , | + | - | / | * |
: | = | $ | " | % | (|) | [|] | # |@
<espaço> :: = <espaço simples>|<espaço> <espaço simples>
<espaço simples> :: = {uma posição em branco}
<palavras reservadas> :: = BEGIN|END|MEMORY|LABEL|FIELD|FORMAT|DEFINE|
MICRO|PROCEDURE|COMMENT|RESERVE|ORIGIN|SET|
LIST|RESET|SOURCE|INTER|TABSIMB|BIN|OCT|
DEC|HEX|MAP|INVERT|ALL|DONTCARE|EQL|FOR|
FMAG|INTERNAL|VAR|LSM|TABLE

OBS.: Para o tradutor, a linguagem é encarada como uma sequência de símbolos básicos e de palavras reservadas.

2.1.2 - CONSTANTES

Sintaxe:

<constantes> :: = <configuração de bits>|
<número>

<configuração de bits> :: = <configuração binária>|
<configuração octal>|
<configuração decimal>|
<configuração hexadecimal>

<configuração binária> :: = "<cordão binário>" B
<configuração octal> :: = "<cordão octal>" Q
<configuração decimal> :: = "<cordão decimal>" D
<configuração hexadecimal> :: = "<cordão hexadecimal>" H

<número> :: = <número binário>|
<número octal>|
<número decimal>|
<número hexadecimal>

<número binário> :: = <cordão binário> B
<número octal> :: = <cordão octal> Q
<número decimal> :: = <cordão decimal>|
<cordão decimal> D
<número hexadecimal> :: = <dígito> <cordão hexadecimal> H|
<dígito> H|
0 <letra hexadecimal> H|
0 <letra hexadecimal> <cordão hexadecimal> H

<cordão binário> :: = <dígito binário>|
<cordão binário> <dígito binário>
<cordão octal> :: = <dígito octal>|
<cordão octal> <dígito octal>
<cordão decimal> :: = <dígito>|
<cordão decimal> <dígito>
<cordão hexadecimal> :: = <caractere hexadecimal>|
<cordão hexadecimal> <caractere hexadecimal>

<dígito binário> :: = 0|1
<dígito octal> :: = 0|1|2|3|4|5|6|7
<letra hexadecimal> :: = A|B|C|D|E|F
<caractere hexadecimal> :: = 0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F

Observações:

As constantes podem ser expressas nas seguintes bases: binária, caracterizada por B; octal, caracterizada por Q; decimal, caracterizada por D; e hexadecimal, caracterizada por H. Estas bases foram inseridas na linguagem LMP com o intuito de oferecer flexibilidade e clareza na documentação do microprograma.

As constantes do tipo configuração de bits representam implicitamente um conjunto de bits, cujo tamanho é função da base utilizada. As constantes do tipo número são utilizadas com a finalidade de fornecer o valor correspondente à base associada. Exemplos de configuração de bits são dados na Tabela 1.

Exemplos de constantes válidas:

123, 01101111B (= 111_{10}), 377Q (= 255_{10}),

"01"B, "C3"H, 0A5H (= 165_{10}), 1023D, 1023

Exemplos de constantes inválidas:

A5H, 130112B, 139Q, 01A5, "1A5"D,"016A"Q

O tamanho máximo de uma constante é função do seu tipo e da capacidade do tradutor implementado. No caso da versão 1.0 efetuada, a constante poderá ser representada por até 48 bits, que correspondem a uma palavra do B6800.

TABELA 1

EXEMPLOS DE CONFIGURAÇÕES DE BITS E CONFIGURAÇÕES BINÁRIAS EQUIVALENTES

CONSTANTES	BASE	CONFIGURAÇÃO BINÁRIA EQUIVALENTE	TAMANHO EM BITS
"01101111"B	Binária	01101111	8
"01"B	Binária	01	2
"147"Q	Octal	001100111	9
"120"D	Decimal	000100100000	12
"C3"H	Hexadecimal	11000011	8
"02"H	Hexadecimal	00000010	8

2.1.3 - COMENTÁRIOS

Sintaxe:

<comentários> :: = <comentário do tipo COMMENT>|
<comentário do tipo "%">

<comentário do tipo COMMENT> :: = COMMENT {qualquer sequência de caracteres sem ";"}

<comentário do tipo "%"> :: = % {qualquer sequência de caracteres até o final do registro lógico do microprograma fonte}

Semântica:

Os comentários assumem importância fundamental quanto à documentação do microprograma. A utilização deste recurso é recomendada, com muita ênfase, para facilitar a manutenção dos microprogramas e melhorar a sua legibilidade. Os comentários podem ser colo

cados em qualquer posição entre as várias entidades do microprograma, quer seja nas declarações quer seja nos comandos, ou mesmo dentro de um comando, no caso do comentário do tipo "%".

Exemplo de uso de comentários:

```
BEGIN
  COMMENT:- MICROPROGRAMA DE TESTE 01
            UNIDADE DE CONTROLE DO MODEM 4800 BPS
            PROJETO SISMAG/PSDA - INPE
            06.NOV.1980;
  %
  % DECLARAÇÃO DA MEMÓRIA DE CONTROLE
  %
  MEMORY WC [0:511, 0:71];
  %
  % DEFINIÇÃO DOS CAMPOS DA PALAVRA DE CONTROLE
  %
  FIELD OPCODE = WC [0:5] ("100110"B), % CÓDIGO DE OPERAÇÃO
        A = WC [6:9],      % ENDEREÇO A
        B = WC [10:13],    % ENDEREÇO B
        FLAG = WC [14:15], % FLAG DE CONDIÇÃO
        ADR = WC [16:24];  % ENDEREÇO DE MEMÓRIA
  .
  .
  .
  FORMAT F1 (OPCODE, A, ALU, ADR); % FORMATO ÚNICO
  .
  .
  .
END
```

2.1.4 - IDENTIFICADORES

Sintaxe:

```
<identificador  :: = <letra>|
                     <identificador> <letra>|
                     <identificador> <dígito>|
                     <identificador> <ponto>
```

Semântica:

Os identificadores não têm significados intrínsecos como as palavras reservadas e são utilizadas para identificação das várias entidades declaradas na linguagem.

Exemplos:

Identificadores válidos:

CONTROLE, MUX1, MUX2, SOMA.A.NB, ALU.SHIFT

Identificadores inválidos:

125, $MUX, %CONTR, 3MUX

2.1.5 - MICROPROGRAMA

Sintaxe:

```
<microprograma> :: = BEGIN
                     <declaração de memória de controle>;
                     <corpo do microprograma>;
                     END

<corpo do microprograma> :: = <lista de comandos>|
                              <lista de declarações>; <lista de
                               comandos>

<lista de declarações> :: = <declaração>|
                            <lista de declarações>; <declarações>

<lista de comandos> :: = <comando>|
                         <lista de comandos>; <comando>
```

Semântica:

Todo microprograma é delimitado pelas palavras reservadas BEGIN e END.

Em todo microprograma deve-se inicialmente declarar a memória de controle, onde residirá o microprograma objeto ou os microcódigos gerados a partir do microprograma fonte. Com esta declaração ficam determinadas as dimensões da memória de controle e o tamanho da microinstrução ou palavra de controle.

Todo identificador referenciado num comando deverá ter sido declarado anteriormente.

Na versão implementada do tradutor LMP, um microprograma é basicamente constituído por duas etapas distintas. A primeira etapa é dada pela lista de declarações que descrevem as características ligadas ao circuito em desenvolvimento e algumas facilidades de programação. A segunda etapa compreende a sequência de comandos que geram, em última análise, os microcódigos desejados.

2.1.6 - INVOCAÇÃO DE DEFINIÇÃO

A invocação de definição faz uso de entidades ainda não definidas, mas que serão apresentadas nas próximas seções. Assim, a invocação de definição será apresentada na Seção 2.2.4, onde é definida a declaração de definição.

2.1.7 - OPÇÕES DE CONTROLE

Sintaxe:

```
<opções de controle> :: = $RESERVE <número>|
                          $ORIGIN <número>|
                          $SET LIST <opção de listagem>|
                          $RESET LIST <opção de listagem>
```

<opção de listagem> :: = SOURCE|
INTER|
TABSIMB|
MEMORY

Semântica:

As opções de controle permitem o gerenciamento da memória de controle quanto à alocação dos microcódigos gerados. Essas opções possibilitam, ainda, o controle de listagem do microprograma fonte, dos códigos gerados na forma intermediária, das tabelas internas e da memória de controle.

O controle de alocação dos microcódigos gerados é feita pelas opções $RESERVE e $ORIGIN.

A opção $RESERVE incrementa o contador de alocação de microinstrução (MILC) pelo valor fornecido em <número>; isto é, reserva tantas posições de memória quanto forem estabelecidas pelo número dado.

Como exemplo de utilização, têm-se:

$RESERVE 5

$RESERVE 12

$RESERVE 20Q

A opção $ORIGIN carrega o contador de alocação de microinstrução com o valor fornecido pelo número dado. Desta forma, a opção $ORIGIN se assemelha à instrução ORG em uma linguagem "assembly".

Exemplificando, podem-se ter:

```
$ORIGIN 0
$ORIGIN 256
$ORIGIN 1000B
```

As opções $SET LIST e $RESET LIST atuam na habilitação ou não de listagem de: microprograma fonte, no caso da opção de listagem SOURCE; forma intermediária, no caso de INTER; tabela de símbolos, no caso de TABSIMB; e memória de controle com estados don't care (indicado pelo ".") no caso de MEMORY.

A Tabela 2 mostra, de forma condensada, essas opções de listagem e a sua ação tomada por omissão ("default").

TABELA 2

OPÇÕES DE LISTAGEM

OPÇÃO	OMISSÃO	Ação tomada
SOURCE	ATIVADO	Lista o microprograma fonte
INTER	DESATIVADO	Lista a forma intermediária correspondente ao microprograma fonte
TABSIMB	DESATIVADO	Lista a tabela de símbolos e as tabelas auxiliares
MEMORY	DESATIVADO	Lista a memória de controle com estados "don't care" representados pelo caractere "."

Exemplos de utilização das opções $SET LIST e $RESET LIST

```
BEGIN
  :
$RESET LIST SOURCE   % Obs.: A listagem do microprograma fonte é desati
                     % vada até que seja novamente ativada pela opção
                     % $SET LIST SOURCE
$SET LIST INTER
$SET LIST MEMORY
  :
END
```

2.2 - DECLARAÇÕES

Sintaxe

```
<declaração LMP> :: = <declaração de memória de controle>|
                      <declaração>

<declaração> :: = <declaração de campo>|
                  <declaração de formato>|
                  <declaração de definição>|
                  <declaração de rótulo>|
```

As declarações podem ser classificadas em dois tipos:

1) aquelas que descrevem as características do processador para o qual se estão gerando os microcódigos, isto é, as declarações dos campos da microinstrução e a declaração dos formatos dessas microinstruções;

2) aquelas que auxiliam na microprogramação através da linguagem, ou seja, as declarações de definição e de rótulo. Na declaração de definição está contida a potencialidade da linguagem na geração de microcódigos, bem como na documentação e na facilidade de leitura do microprograma.

2.2.1 - DECLARAÇÃO DE MEMÓRIA DE CONTROLE

Sintaxe:

<declaração de memória de controle> :: = MEMORY <identificador de memória> <especificação de memória>

<identificador de memória> :: = <identificador>

<especificação de memória> :: = [<tamanho da memória>, <comprimento da palavra>]

<tamanho da memória> :: = <par delimitador>

<comprimento da palavra> :: = <par delimitador>

<par delimitador> :: = <número>:<número>

Semântica:

Através desta declaração, associa-se um identificador à memória de controle, bem como especifica-se a área disponível e a nomenclatura dos bits da palavra de controle.

É conveniente notar que esta declaração é necessária em qualquer microprograma e deve ser a primeira declaração a constar no bloco principal.

Exemplos:

```
MEMORY CS [0:511, 0:71]
          ↑       ↑
          |       palavra de 72 bits
          área de memória disponível: 512 posições

MEMORY MEMCONT [40H:1000H, 0:31]
               ↑           ↑
               |           microinstrução com 32 bits, onde o bit 0
               |           é o mais significativo e o bit 31 o menos significativo
               área disponível a partir de 40H a 1000H
```

2.2.2 - DECLARAÇÃO DE CAMPO

Sintaxe:

<declaração de campo> :: = FIELD <lista de especificação de campo>

<lista de especificação de campo> :: = <especificação de campo> |
<lista de especificação de campo>, <especificação de campo>

<especificação de campo> :: = <identificador de campo> = <identificador de memória de controle> [<par delimitador de campo>] |
<identificador de campo> = <identificador de memória de controle> [<par delimitador de campo>] (<constante>)

<identificador de campo> :: = <identificador>

<par delimitador de campo> :: = <número> : <número>

Semântica:

Através da declaração de campo, associa-se a cada campo da microinstrução um identificador. Basicamente, a linguagem LMP deverá fornecer meios para o preenchimento destes campos, com a configuração adequada de bits.

Como exemplo de uma declaração de campo, pode-se dar uma palavra de controle, como na Figura 2, pertencente a uma memória de controle indicada por CS. A declaração de campo correspondente é apresentado a seguir.

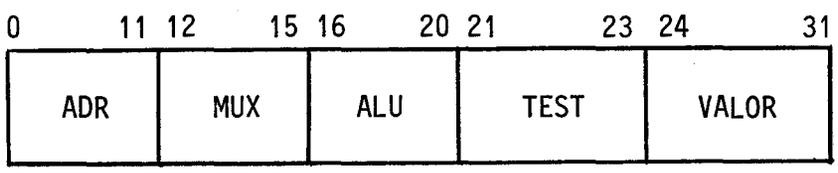

Fig. 2 - Exemplo de uma microinstrução e seus campos.

```
FIELD  ADR   = CS [0:11],            % Campo de endereço
       MUX   = CS [12:15] (0111B),  % Controle do multiplex
                                    % Valor por omissão = 0111B
       ALU   = CS [16:20] (0),      % Controle da ALU
                                    % Valor por omissão igual a zero
       TEST  = CS [21:23] ("XXX"B), % Teste de condição
                                    % Valor por omissão é "don't
                                    % care"
       VALOR = CS [24:31]           % Constante
```

É permitido ainda atribuir um valor por omissão a um dado do campo através de uma constante entre parênteses situada logo após a declaração correspondente.

Os campos que não tiverem sido associados com quaisquer valores durante o corpo do microprograma serão considerados como contendo estados "don't care" com respectiva mensagem do tradutor (ver declaração de formato).

É importante notar a ordem em que se especifica o par delimitador de campo. Os valores mais significativos de cada campo são aqueles que apresentam os valores mais baixos, como convencionado na declaração de memória de controle para a descrição da palavra de controle.

A codificação de uma microinstrução pode estar associado mais que um formato (ver declaração de formato), sendo permitido declarar vários identificadores diferentes entre si, associados a um mesmo conjunto de bits da palavra de controle.

Na Figura 3 são apresentados os formatos de microinstrução do minicomputador HP1000, com o propósito de exemplificar a utilização da declaração de campos.

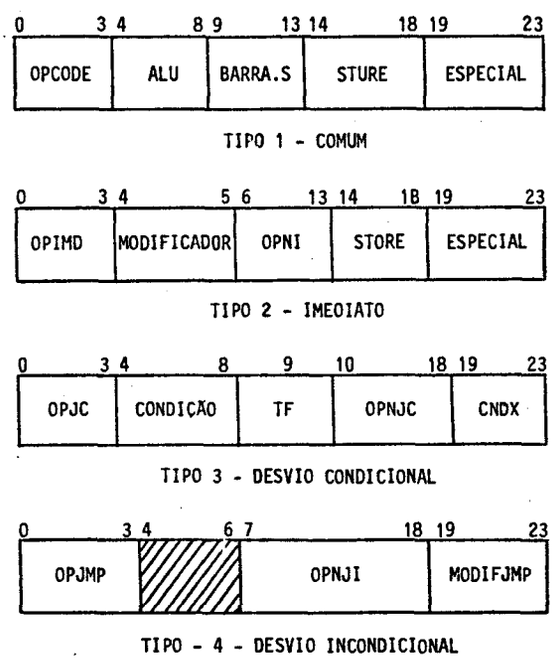

Fig. 3 - Formatos de microinstrução do HP1000, baseados em Hewlett-Packard (1977).

```
FIELD
   %
   % Tipo 1
   %
   OPCODE   = CS [0:3],        % Operação
   ALU      = CS [4:8],        % Controle da ALU
   BARRA.S  = CS [9:13],       % Controle do Barramento S
   STORE    = CS [14:18],      % Armazenamento
   ESPECIAL = CS [19:23],      % Microoperações especiais
   %
   % Tipo 2
   %
   OPIMD    = CS [0:3],        % Operação do tipo imediato
   MODIFICADOR = CS [4:5],     % Modificador de Função
   OPNI     = CS [6:13,        % Operando imediato
   %
   % Tipo 3
   %
   OPJC     = CS [0:3],        % Operação de desvio
   CONDIÇÃO = CS [4:8],        % Condição de desvio
   TF       = CS [9:9],        % Verdadeira ou falsa
   OPNJC    = CS [10:18],      % Operando de desvio condicional
   CNDX     = CS [19:23]       % Código especial
   %
   % Tipo 4
   %
   OPJMP    = CS [0:3],        % Operação de desvio incondicional
   MODIFJMP = CS [19:23],      % Modificador de desvio
   OPNJI    = CS [7:18],       % Operando de desvio incondicional
```

2.2.3 - DECLARAÇÃO DE FORMATO

Sintaxe:

<declaração de formato> :: = FORMAT <lista de especificações de formato>

<lista de especificações de formato> :: =<especificação de formato>|
<lista de especificação de formato>, <especificação de formato>

<especificação de formato> :: = <identificador de formato> (<lista de identificadores de campo>)

<identificador de formato> :: = <identificador>

<lista de identificadores de campo> :: = <identificador de campo>|
<lista de identificadores de campo>,
<identificador de campo>

Semântica:

Com o intuito de diminuir a palavra de controle, a codificação em certos campos é realizada de modo que estes sejam compartilhados. Desta forma, é possível mais de um formato de microinstrução.

Esta declaração tem a finalidade de caracterizar a coerência dos campos de um dado formato, bem como a atribuição de valores por omissão aos campos de um formato reconhecido de forma automática. Ela também possibilita a verificação e a deteção de possíveis erros de microprogramação.

Deve-se observar que todos os identificadores de campo utilizados na declaração de formato foram declarados anteriormente. Caso contrário, haverá emissão de mensagem de erro pelo tradutor LMP V1.0.

Exemplos:

```
FORMAT
      TIPO.1 (OPCODE, ALU, BARRA.S, STORE, ESPECIAL), %comum
      TIPO.2 (OPIMD,MODIFICADOR, OPNI, STORE, ESPECIAL), % imediato
      TIPO.3 (OPJC, CONDIÇÃO, TF, OPNJC, CNDX), % desvio condicional
      TIPO.4 (OPJMP, OPNJI, MODIFJMP) % desvio incondicional.
```

2.2.4 - DECLARAÇÃO DE DEFINIÇÃO

Sintaxe:

<declaração de definição> :: = DEFINE <lista de especificações de definição>

<lista de especificações de definição> :: = <especificação de definição>|
<lista de especificações de definição>,
<especificação de definição>

<especificação de definição> :: = <identificador de definição> = <contexto> #

<identificador de definição> :: = <identificador>

<contexto> :: = {qualquer sequência de caracteres válidos, diferentes de # , que inserida no microprograma fonte adquire significado válido}

<invocação de definição> :: = <identificador de definição> <parâmetro de texto real>

<parâmetro de texto real> :: = <vazio>|
(<lista de segmentos de texto real>)

<vazio :: = {conjunto vazio de caracteres}

<lista de segmentos de texto> :: = <segmento de texto real>|
<lista de segmentos de texto real>,
<segmento de texto real>

<segmento de texto real> :: = {texto real sem conter parênteses descasados ou vírgulas parentizadas}

Semântica:

A idéia básica da declaração de definição é a de possibi tar o recurso de macro ao nível de microprograma fonte, através de subs tituição ou inserção de texto quando é feita uma invocação de defini ção.

Com esta declaração, o usuário poderá associar um identi ficador a uma sequência de caracteres, que pode ter sido definida no contexto. Quando este identificador for referenciado no microprograma fonte, o contexto por ele identificado passará a ser analisado, neste pon to, como constituinte real do microprograma fonte.

Existem dois tipos de declaração de definição:

a) *Paramétrica*: Ela apresenta no contexto, uma ou mais vezes, a sequência de caracteres ?nn. O índice nn é um número que repre senta a ordem do parâmetro formal. Na invocação de definição, o parâmetro real, alocado mais à esquerda da lista de parâme tros reais, substitui o parâmetro formal ?nn de menor valor, e assim sucessivamente.

b) *Simples:* Esta não apresenta parâmetros formais, ou melhor, ca racteriza-se pela ausência da sequência de caracteres ?nn no contexto.

Exemplos

```
DEFINE   R0 = 00H # ,
         R1 = 01H # ,
         R2 = 02H # ,
         STATUS = 11B # ;

DEFINE   IFCOND = TEST : = ?01, JMPADR : = ?02, CLT : = 1 # ,
         CALL   = ADR : = ?01, MUX # ;
```

Na invocação de definição correspondente, poder-se-ia ter:

```
IFCOND (STATUS, LOOP),     % onde LOOP é um identificador de rótulo
CALL (FETCH),              % FETCH é um identificador de rótulo.
```

A restrição no segmento de texto real, onde não é permitida a existência de parênteses descasados ou de vírgulas com parênteses, visa a eliminação de ambiguidades que poderiam ocorrer. O exemplo abaixo ilustra situações deste tipo.

Na invocação de TEXTO (AB, TIPO (ALU, ADR), 3) poder-se-iam ter as seguintes interpretações:

```
1) Parâmetros: AB
               TIPO (ALU, ADR)
               3
```

ou

```
2) Parâmetros: AB
               TIPO (ALU
               ADR)
               3
```

2.3 - EXPRESSÕES

Sintaxe:

```
<expressão> :: = <termo>|
                 <expressão> + <termo>|
                 <expressão> - <termo>
<termo> :: = <fator>|
             <termo> * <fator>|
             <termo> / <fator>
```

<fator> :: = (<expressão>)|
<número>|
$|
<identificador de rótulo>|
<identificador de campo>

Semântica:

As expressões permitem algumas facilidades quanto às operações aritméticas e quanto às referências aos identificadores de rótulo, de campo, ou mesmo do contador de alocação de microinstrução.

Os operadores aritméticos +, -, * e / têm significado convencional de soma, subtração, multiplicação e divisão, respectivamente. No caso do operador divisão, a operação é realizada como sendo entre variáveis do tipo INTEGER da linguagem ALGOL.

Quanto à referência ao identificador de rótulo, o valor associado se refere ao endereço da memória de controle que representa o rótulo.

No caso de identificador de campo, certo cuidado é necessário. O valor associado corresponde ao último valor atribuído a este campo, podendo até mesmo corresponder ao valor por omissão ("default"), por ocasião de sua declaração. Porém, se o campo estiver no estado "don't care", a expressão será inválida.

Quanto ao contador de alocação de microinstrução, o seu valor é representado pelo símbolo $. O endereço de microprograma corresponde ao endereço onde será armazenada, pelo respectivo comando, a microinstrução.

Exemplo de expressões válidas:

```
10 + 5 * (3+2)
LOOP + 3          % LOOP é identificador de rótulo
$ - 2             % a expressão equivale ao valor do contador de
                    alocação de microinstrução menos dois
ALU               % ALU é um identificador de campo e possui um
                    valor associado
```

Exemplo de expressões inválidas:

```
-10 + 5 * (3 + 2)   % expressão não pode ser iniciada com o operador menos (-)
ALU                 % ALU é um identificador de campo,
                    % porém contém estados "don't care"
```

2.4 - COMANDOS E MICROCOMANDOS

Sintaxe:

```
<comando> :: = <comando simples>|
               <comando rotulado>

<comando rotulado> :: = <identificador de rótulo> : <comando simples>

<comando simples> :: = <lista de microcomandos>

<lista de microcomandos> :: = <microcomando>|
                              <lista de microcomandos>,
                              <microcomando>

<microcomando> :: = <atribuição>|
                    <configuração de campo anterior>
```

Os comandos são as entidades que na realidade geram a memória de controle. A cada comando corresponde a geração de uma microinstrução, que é composta de microoperações simultâneas. Os microcomandos correspondem a estas microoperações ao nível de linguagem.

2.4.1 - MICROCOMANDO DE ATRIBUIÇÃO

Sintaxe

<atribuição> :: = <atribuição lado esquerdo> : = <expressão> |
<atribuição lado esquerdo> : = <configuração de bits>

<atribuição lado esquerdo> :: = <identificador de campo> |
<identificador de memória de controle>
[<par delimitador>]

Semântica:

O microcomando de atribuição permite o preenchimento de um dado campo ou de uma região da palavra de controle, com uma configuração de bits determinada pela expressão correspondente.

No cálculo da configuração de bits a ser armazenada no campo ou região da palavra de controle, a partir da expressão, duas situações podem ocorrer:

1) A configuração de bits calculada é menor que o tamanho do campo ou da região da palavra de controle. Neste caso, a configuração de bits é armazenada, justificando-se à direita, isto é, nas posições mais significativas são inseridos bits iguais a zero.

2) A configuração de bits calculada é maior que o tamanho do campo ou da região da palavra de controle. Neste caso, haverá emissão de mensagem notificando o usuário.

Exemplos:

ADR : = 2 * 4 + 16

CS [0:4] := 3

SOURCE := R1

2.4.2 - MICROCOMANDO DE CONFIGURAÇÃO DE CAMPO ANTERIOR

Sintaxe:

<configuração de campo anterior> :: = <identificador de campo>

Semântica:

Este microcomando foi introduzido para facilitar a micro programação de certos campos que são modificados com pouca frequência.

O tradutor LMP armazena, para cada campo, a informação da última configuração assumida. A simples referência a um certo campo dentro de um comando equivalerá a um microcomando de atribuição com o último valor assumido.

3. PROCEDIMENTOS DE UTILIZAÇÃO DO TRADUTOR LMP V1.0 E EXEMPLOS

O tradutor LMP versão 1.0 foi escrita em linguagem ALGOL (Burroughs, 1974) e está disponível no computador B6800 do INPE.

Esta versão implementa um subconjunto da linguagem LMP (Yamaguti, 1981), gerando, a partir do microprograma fonte, a listagem em impressora dos microcódigos correspondentes e a listagem em forma binária com os bits "don't care" representados pelo caractere ".".

Para manipulação de arquivos e comandos do CANDE, ver o respectivo manual (Burroughs, 1977), pois supõe-se que sejam do conhecimento prévio do usuário.

3.1 - PROCEDIMENTOS DE UTILIZAÇÃO

Nesta versão têm-se basicamente dois passos:

a) criação do arquivo de entrada;

b) tradução e geração do arquivo de saída.

A criação e edição do arquivo de entrada pode ser feita através do CANDE (Burroughs, 1977), gerando um arquivo em disco do tipo SEQ ou DATA, que contêm o microprograma fonte desejado.

O arquivo objeto do tradutor LMP versão 1.0 é denominado (SDA)LMP e é disponível para execução por qualquer terminal ligado ao B6800 do INPE.

O arquivo de entrada para o tradutor é denominado CARD (tipo REMOTE) e o de saída LINHA (tipo REMOTE), os quais podem ser definidos na execução (ver comandos WFL/Burroughs), como apresentado no exemplo abaixo:

```
RUN (SDA) LMP; FILE CARD (KIND = DISK, TITLE = <nome do arquivo
  de entrada>, FILETYPE = 7), LINHA (KIND = PRINTER)
```

Isto é, o arquivo de entrada contêm o microprograma fonte, e as saídas do tradutor LMP versão 1.0 são direcionadas para a impressora.

3.2 - EXEMPLO DE UM MICROPROGRAMA EM LMP

Um exemplo de utilização da linguagem LMP é apresentada no Apêndice A. Este exemplo é baseado em um exemplo da ADVANCED MICRO DEVICES (1977).

4. CONSIDERAÇÕES FINAIS

A aplicabilidade da linguagem LMP em termos de microprogramação é muito vasta, uma vez que a linguagem em si não depende especificamente do sistema para a qual são gerados os microcódigos. Usualmente, linguagens de microprogramação têm sido desenvolvidas especificamente para processadores específicos e implementadas em baixo nível, principalmente as destinadas à microprogramação do tipo horizontal.

Com a disponibilidade de recursos de pós-processamento prevista para as próximas versões do tradutor, acredita-se poder operacionalizar a geração dos microcódigos com a depuração no emulador de memórias de microcontrole EMMAC (Amaral, 1979) de forma automática e mais efetiva em relação à versão atualmente disponível.

REFERÊNCIAS BIBLIOGRÁFICAS

ADVANCED MICRO DEVICES. (AMD). *"AMDASM/80 reference manual - MSD resident microassembler"*. Sunnyvale, Ca., 1977.

———. *"Microprogramming handbook and Am 2900 emulation"*. 2ed. Sunnyvale, Ca., c1976.

AMARAL, P.F.S. *"Emulador de memórias de microcontrole auxiliado por computador"*. São José dos Campos, INPE, maio 1979. (INPE-1489-TDL/009).

BURROUGHS. *"B6600/B7700 Algol language reference manual"*. Detroit, 1974.

BURROUGHS. *"B7000/B6000 Series CANDE user's manual"*. Detroit, 1977.

HEWLETT PACKARD (HP). *"21MX Series Computers BCS and DOS microprogramming reference manual"*. Cupertino, Ca., 1977.

YAMAGUTI, W. *"LMP, uma linguagem de microprogramação"*. São José dos Campos, INPE, abril 1981. (INPE-2031-TDL/049).

APÊNDICE A

EXEMPLO DE APLICAÇÃO

O exemplo que se segue baseia-se em um outro fornecido pela ADVANCED MICRO DEVICES (1977) e ilustra o emprego da linguagem LMP, após adequada adaptação.

A Figura A.1 apresenta o diagrama de blocos simplificado da unidade de controle, para o qual serão gerados os microcódigos do exemplo. A Figura A.2 apresenta a palavra de controle e sua especificação por campos.

O algoritmo da Figura A.3 foi utilizado como o exemplo de aplicação da linguagem LMP, apresentado a seguir.

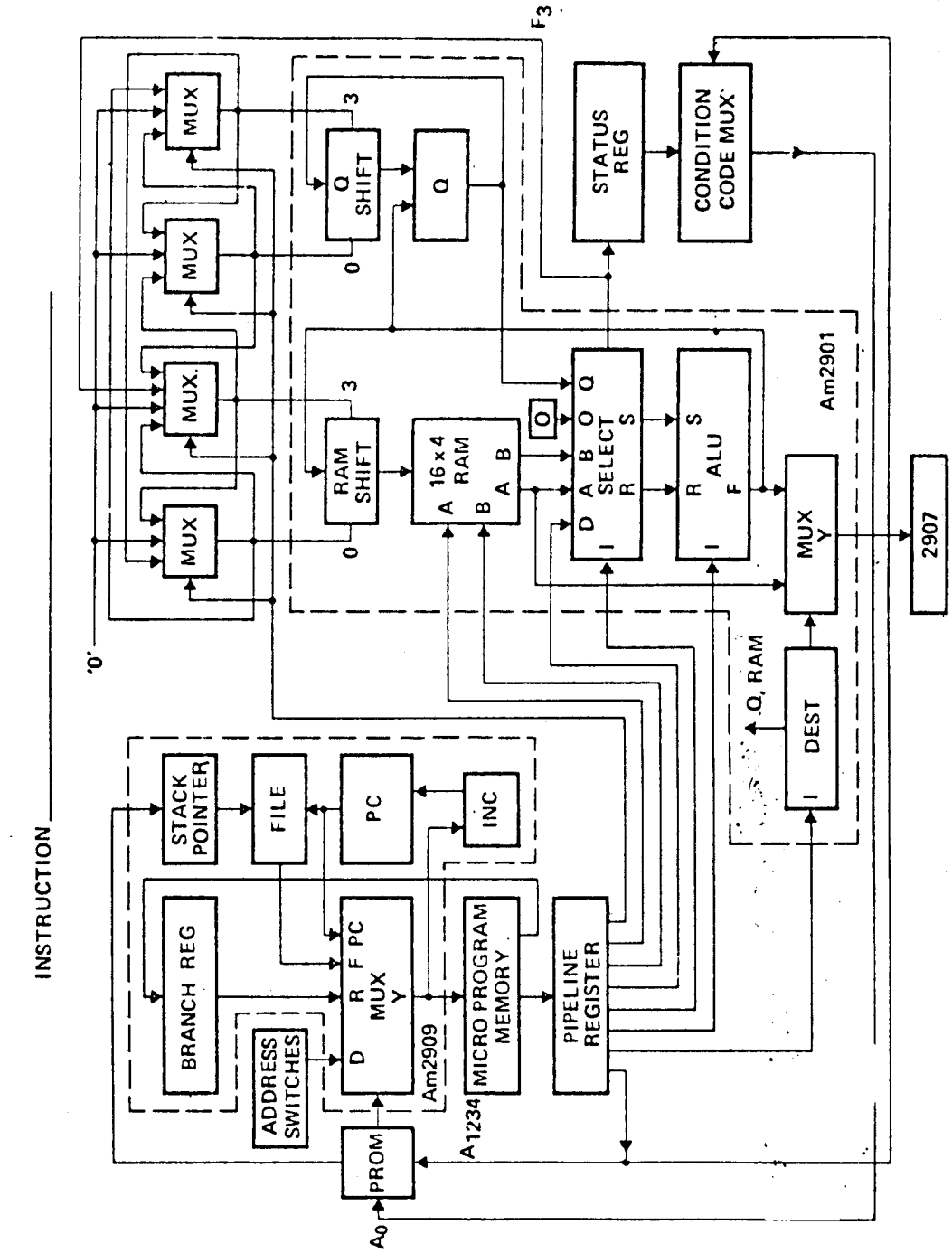

Fig. A.1 - Diagrama de blocos simplificado da unidade de controle para o exemplo de aplicação.

FONTE: Advanced Micro Devices (1976), p. 25.

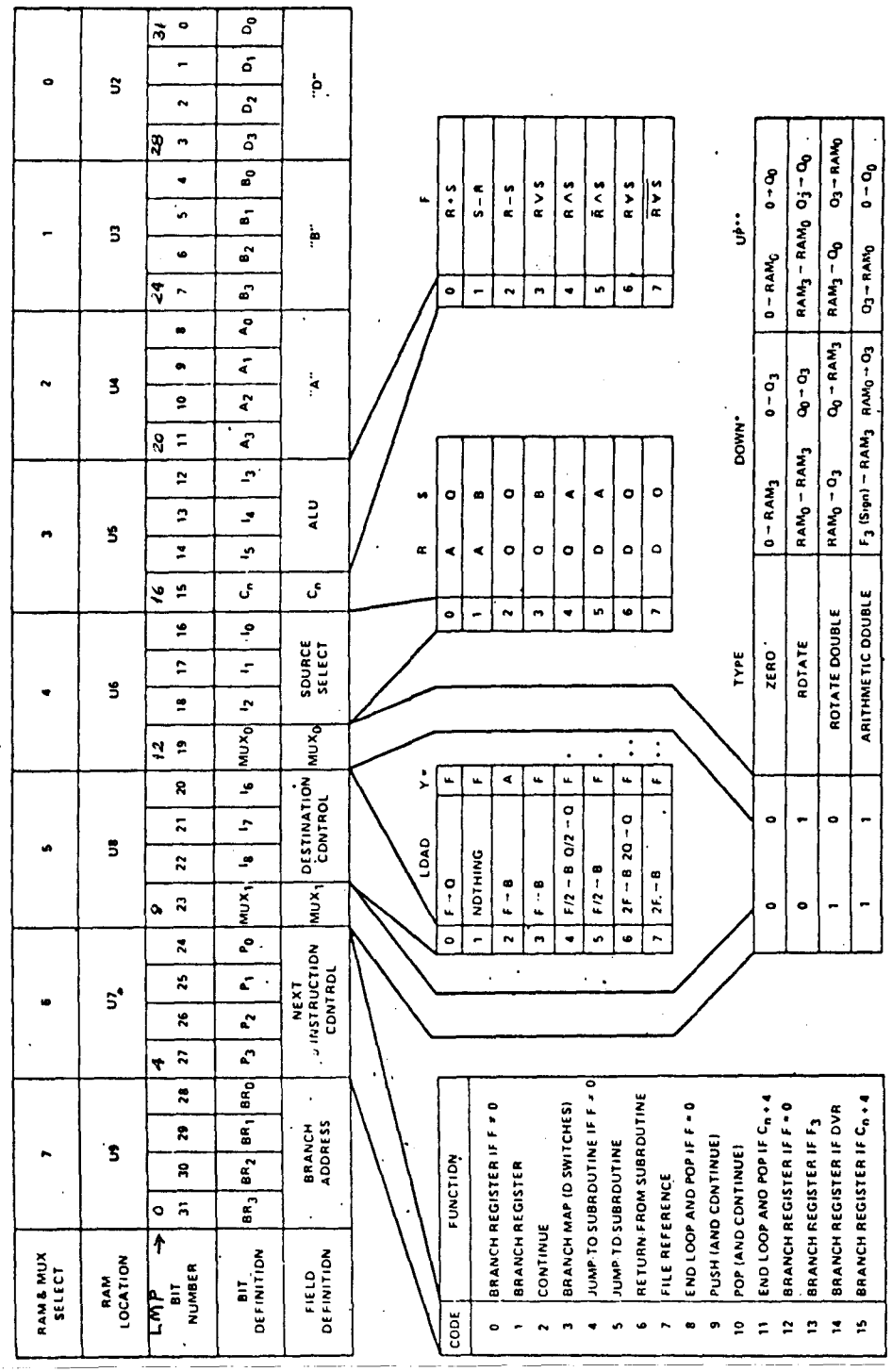

Fig. A.2 - Formatação da palavra de controle.

FONTE: Advanced Micro Devices (1976), p. 26.

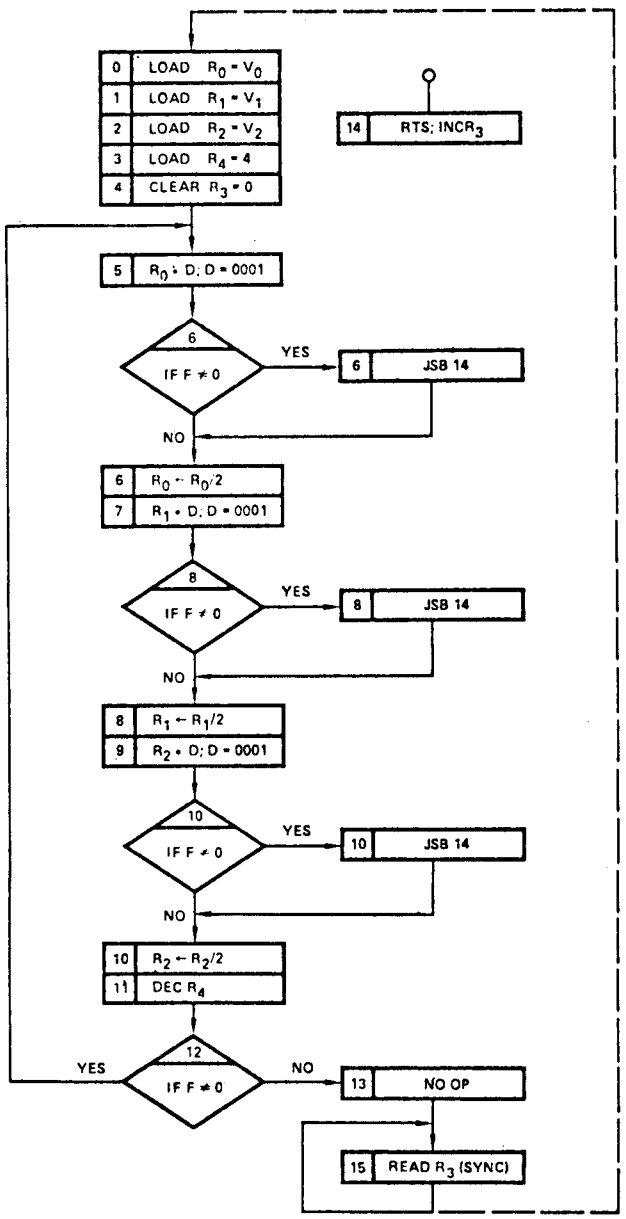

Fig. A.3 - Fluxograma do exemplo.

FONTE: Advanced Micro Devices (1976), p. 28.

```
================================================================================

                        INSTITUTO DE PESQUISAS ESPACIAIS
   LMP - LINGUAGEM DE MICROPROGRAMACAO SISTEMA B6700/B6800 - VERSAO 1.0
          TRADUCAO EXECUTADA EM 30/08/83 AS 14 HORAS E 60 MINUTOS

================================================================================

ARQUIVO FONTE : EXEMPLO.

BEGIN
COMMENT :- EXEMPLO DE UTILIZACAO DA LINGUAGEM LMP, BASEADO
           NA ARQUITETURA DO "KIT" EDUCATIVO AM2900 DA
           ADVANCED MICRO DEVICES;
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                             %
%                                                             %
%                 KIT AM2900/EXEMPLO LMP                      %
%                 ======================                      %
%                                                             %
%                 INPE - DCA / DEA                            %
%                 PROJETO : LASIDA/PSDA                       %
%                 EQUIPAMENTO : KIT AM2900                    %
%                 PLACA : EXEMPLO                              %
%                 AUTOR : WILSON YAMAGUTI                     %
%                 DATA : 08/AGOSTO/1983                       %
%                                                             %
%                                                             %
%                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                             %
%.............................................................%
%                                                             %
%        DECLARACAO DA MEMORIA DE MICROCONTROLE               %
%                                                             %
%.............................................................%
%
MEMORY
           CS[0:15,0:31];    % 16 PALAVRAS DE 80 BITS.
%..............................................................
%
%   DECLARACAO DOS CAMPOS DE MICROCONTROLE
%
%..............................................................
%
%
FIELD
      BADR      = CS[0:3]("XXXX"B), % ENDER DE DESVIO
      NEXT.ADDR = CS[4:7]("0010"B), % CONTROLE PROX. ENDER
                                    % DEFAULT: CONT
      MUX1      = CS[8:8]("X"B),    % CONTROLE MUX BIT1
      DEST      = CS[9:11]("001"B), % CONTROLE DE DESTINO
      MUX0      = CS[12:12]("X"B),  % CONTROLE MUX BIT0
      FONTE    = CS[13:15]("XXX"B),% REG. FONTE
      CN        = CS[16:16]("X"B),  % CARRY IN 2901
      ALU       = CS[17:19]("XXX"B),% OPER NA ALU 2901
      SELA      = CS[20:23]("XXXX"B), % SELA/RAM 16 X 4
      SELB      = CS[24:27]("XXXX"B), % SELB/RAM 16 X 4
      DIN       = CS[28:31]("XXXX"B); % ENTRADA D2901
```

```
51 %
52 %
53 %.....................................................................
54 %
55 %   DECLARACAO DE FORMATO (INICIO)
56 %
57 %.....................................................................
58 %
59 COMMENT:= A DECLARACAO DE FORMATO, MESMO QUE UNICA E NECESSARIA;
60    FORMAT
61          F1( BADR,        % ENDER DE DESVIO
62              NEXT.ADDR,   % CTL PROX. ENDER
63              MUX1,        % MUX1/SHIFT 2901
64              DEST,        % DESTINO
65              MUX0,        % MUX0/SHIFT 2901
66              FONTE,       % FONTE R-S
67              CN,          % CARRY IN 2901
68              ALU,         % OPER. NA ALU
69              SELA,        % RAM A
70              SELB,        % RAM B
71              DIN);        % ENTRADA D 2901
72 %
73 %.....................................................................
74 %
75 %   DEFINICAO DE REGISTROS
76 %
77 %.....................................................................
78 %
79   DEFINE
80          R0  = "0"4#,
81          R1  = "1"4#,
82          R2  = "2"4#,
83          R3  = "3"4#,
84          R4  = "4"4#,
85          R5  = "5"4#,
86          R6  = "6"4#,
87          R7  = "7"4#,
88          R8  = "8"4#,
89          R9  = "9"4#,
90          R10 = "A"4#,
91          R11 = "B"4#,
92          R12 = "C"4#,
93          R13 = "D"4#,
94          R14 = "E"4#,
95          R15 = "F"4#;
96 %
97 %.....................................................................
98 %
99 %  DEFINICAO DE OPERANDOS PARA O AM2901
100 %
101 %.....................................................................
102 %
103   DEFINE
104        AQ = "0"0#,      %      R       S
105        AB = "1"0#,      %      A       B
106        ZQ = "2"0#,      %     ZERO     Q
107        ZB = "3"0#,      %     ZERO     B
108        ZA = "4"0#,      %     ZERO     A
109        DA = "5"0#,      %      D       A
110        DQ = "6"0#,      %      D       Q
```

```
111         DZ = "7"Q#;      %        D       ZERO
112 %
113 %.....................................................
114 %
115 % DEFINICAO DE OPERADORES LOGICOS DA ALU
116 %
117 %.....................................................
118   DEFINE
119         .ADD.    =       0#,      %        R + S
120         .SUBR.   =       1#,      %        S - R
121         .SUBS.   =       2#,      %        R - S
122         .OR.     =       3#,      %        R OR S
123         .AND.    =       4#,      %        R AND S
124         .NOTRS.  =       5#,      %        R/ AND S
125         .EXOR.   =       6#,      %        R XOR S
126         .EXNOR.  =       7#;      %       (R XOR S)/
127 %
128 %.....................................................
129 %
130 %   DEFINICAO DE FUNCOES LOGICAS DA ALU
131 %
132 %.....................................................
133 %
134   DEFINE
135         ADD    = ALU:=.ADD.,      FONTE:=?01#,     % R+S
136         SUBR   = ALU:=.SUBR.,     FONTE:=?01#,     % S-R
137         SUBS   = ALU:=.SUBS.,     FONTE:=?01#,     % R-S
138         OR     = ALU:=.OR.,       FONTE:=?01#,     % R OR S
139         AND    = ALU:=.AND.,      FONTE:=?01#,     % R AND S
140         NOTRS  = ALU:=.NOTRS.,    FONTE:=?01#,     % R/ AND S
141         EXOR   = ALU:=.EXOR.,     FONTE:=?01#,     % R XOR S
142         EXNOR  = ALU:=.EXNOR.,    FONTE:=?01#;     % (R XOR S)/
143 %
144 %.....................................................
145 %
146 %  DEFINICAO DE FUNCOES SHIFT/ROTATE
147 %
148 %.....................................................
149 %
150 %MUX1  MUX0     TYPE              DOWN                        UP
151 %.....................................................
152 %  0     0    ZERO.SHIFT    0->RAM3, 0->Q3          0->RAM0, 0->Q0
153 %  0     1      ROTATE     RAM0->RAM3,Q0->Q3       RAM3->RAM0,Q3->Q0
154 %  1     0    ROTATE.DB    RAM0->Q3, Q0->RAM3      RAM3->Q0, Q3->RAM0
155 %  1     1     ARIT.DB   F3(SIGN)->RAM3,RAM0->Q3   Q3->RAM0, 0->Q0
156 %
157 %.....................................................
158 %
159   DEFINE
160         ZERA.SHIFT = MUX1:="0"B,  MUX0:="0"B#,
161         ROTATE     = MUX1:="0"B,  MUX0:="1"B#,
162         ROTATE.DB  = MUX1:="1"B,  MUX0:="0"B#,
163         ARIT.DB    = MUX1:="1"B,  MUX0:="1"B#;
164 %.....................................................
165 %
166 %
167 %  DEFINICAO DE CONTROLE DE DESTINO AM2901
168 %
169 %.....................................................
170 %
```

```
171   DEFINE
172         QREG   = DEST:=0#, % F->Q              , Y=F
173         NOP    = DEST:=1#, %                   , Y=F
174         RAMA   = DEST:=2#, % F->B              , Y=A
175         RAMF   = DEST:=3#, % F->B              , Y=F
176         RAMQD  = DEST:=4#, % F/2->B, Q/2->Q,  Y=F     (DOWN)
177         RAMD   = DEST:=5#, % F/2->B            , Y=F     (DOWN)
178         RAMQU  = DEST:=6#, % 2F->B, 2Q->Q      , Y=F     (UP)
179         RAMU   = DEST:=7#; % 2F->B,            , Y=F     (UP)
180 %
181 %.....................................................................
182 %
183 %  DEFINICAO DE OPERADORES DE CONTROLE PROXIMO ENDER
184 %
185 %.....................................................................
186 %
187   DEFINE
188         .BRFNO.    =      "0"H#, % BRANCH REG. IF F#0
189         .BR.       =      "1"H#,
190         .CONT.     =      "2"H#,
191         .BM.       =      "3"H#,
192         .JSRFNO.    =      "4"H#,
193         .JSR.      =      "5"H#,
194         .RTS.      =      "6"H#,
195         .STKREF.   =      "7"H#,
196         .LOOPFNO.  =      "8"H#,
197         .PUSH.     =      "9"H#,
198         .POP.      =      "A"H#,
199         .LOOPCOUT.=       "B"H#,
200         .BRF.EQ.0.=       "C"H#,
201         .BRF3.     =      "D"H#,
202         .BROVR.    =      "E"H#,
203         .BRCOUT.   =      "F"H#;
204
205
206
207 %
208 %.....................................................................
209 %
210 %  DEFINICAO DA FUNCAO DE CONTROLE DO PROXIMO ENDERECO
211 %
212 %.....................................................................
213 %
214   DEFINE
215         BRFNO     = NEXT.ADDR:=.BRFNO.    , BADR:=?01#,
216         BR        = NEXT.ADDR:=.BR.       , BADR:=?01#,
217         CONT      = NEXT.ADDR:=.CONT.#    ,
218         BM        = NEXT.ADDR:=.BM.#      ,
219         JSRFNO    = NEXT.ADDR:=.JSRFNO.   , BADR:=?01#,
220         JSR       = NEXT.ADDR:=.JSR.      , BADR:=?01#,
221         RTS       = NEXT.ADDR:=.RTS.#     ,
222         STKREF    = NEXT.ADDR:=.STKREF.#  ,
223         LOOPFNO   = NEXT.ADDR:=.LOOPFNO.#,
224         PUSH      = NEXT.ADDR:=.PUSH.#    ,
225         POP       = NEXT.ADDR:=.POP.#     ,
226         LOOPCOUT  = NEXT.ADDR:=.LOOPCOUT.#,
227         BRF.EQ.0  = NEXT.ADDR:=.BRF.EQ.0. , BADR:=?01#,
228         BRF3      = NEXT.ADDR:=.BRF3.     , BADR:=?01#,
229         BROVR     = NEXT.ADDR:=.BROVR.    , BADR:=?01#,
230         BRCOUT    = NEXT.ADDR:=.BRCOUT.   , BADR:=?01#;
```

```
231 %
232 %.....................................................
233 %
234 %  OUTRAS DEFINICOES
235 %
236 %.....................................................
237 %
238   DEFINE
239        CN0  = "0"B#,
240        CN1  = "1"B#,
241        LOW  = "0"B#,
242        HIGH = "1"B#,
243        ZERO = "0"B#,
244        ONE  = "1"B#;
245 %
246 %.....................................................
247 %
248 %  DEFINICAO DE MICROFUNCOES RELACIONADAS C/O ALGORITMO EXEMPLO
249 %
250 %.....................................................
251 %
252   DEFINE
253        LOAD   = RAMF,        % CARREGA REGISTRO COM VALOR IMEDIATO
254                 OR(DZ),      % CHAMADA LOAD(RI, VALOR)
255                 SELB:=?01,   % <==> RI:=VALOR
256                 DIN:=?02#,
257        IFCALL = JSRFNO(INCR3),        % DESVIO CONDICIONAL //.
258                 RAMD,        % RI <= RI/2
259                 OR(ZB),
260                 SELB:=?01#;
261 %
262 %
263 %
264 %.....................................................
265 %
266 %  DECLARACAO DE ROTULOS
267 %
268 %.....................................................
269 %
270    LABEL LOOP.1, LOOP.2, INCR3;
271 %
272 %.....................................................
273 %
274 %
275 %.....................................................
276 %
277 %
278 %=====================================================
279 %
280 %     MICROPROGRAMA  PRINCIPAL
281 %
282 %=====================================================
283 %
284    $ORIGIN 0
285        LOAD(R0,15);     % V0 =15
286        LOAD(R1,9);      % V1 = 9
287        LOAD(R2,0);      % V2 = 0
288        LOAD(R4,4);      % R4 = 4
289        RAMF,AND(ZB), SELB:=R3;       % CLEAR R3
290 LOOP.1:AND(DA), SELA:=R0, SELB:=R0, DIN:="1"H;
```

```
291                          % RO AND D E 0=0001
292                          % 5
293 IFCALL( R0);     % CALL SUBROTINA INCR3 SE F#0
294                                % R0 <= R0/2
295                                % 6
296 AND(DA), SELA:=R1, SELB:=R1, DIN:="1"H;  % 7
297 IFCALL(R1);                 % 8     R1 <= R1/2
298 AND(DA), SELA:=R2, SELB:=R2, DIN:="1"H;  % 9
299 IFCALL(R2);                 % 10
300 SUBR(ZB),RAMF,SELB:=R4,CN:=CN0;          % 11
301 BRFND(LOOP.1);                           % 12
302 BR(LOOP.2);                             % 13
303 INCR3: RTS,ADD(ZB),SELB:=R3,CN:=CN1;     % ROTINA R3:=R3+1
304                                          % 14
305 LOOP.2: BR(LOOP.2), OR(ZB), SELB:=R3;    % 15
306 $SET MEMORY
307 END
```

```
================================================================
NUMERO DE ERROS DETETADOS =  0.     NUMERO DE AVISOS EMITIDOS =  0.
NUMERO DE REGISTROS DO MICROPROGRAMA FONTE =  307.
FORMA INTERMEDIARIA = 519.          MICROINSTRUCOES GERADAS=   16.
TRADUTOR LMP COMPILADO EM 06.07.83 NO SISTEMA B6800/INPE
PROJ.LASIDA/PSDA/INPE-DEPTO ENG.COMPUTACAO EM APLICACOES ESPACIAIS
================================================================
```

```
*** LMP VERSAO 1.0 - INPE/B6800 ***

CARTAO  ENDERECO    MICROCODIGOS NA FORMA SEMI-FINAL
 FONTE  DEC  HEX

  285     0 0000    ....0010.011.111.011....00001111
  286     1 0001    ....0010.011.111.011....00011001
  287     2 0002    ....0010.011.111.011....00100000
  288     3 0003    ....0010.011.111.011....01000100
  289     4 0004    ....0010.011.011.100....0011....
  290     5 0005    ....0010.001.101.100000000000001
  293     6 0006    11100100.101.011.011....0000....
  296     7 0007    ....0010.001.101.100000100010001
  297     8 0008    11100100.101.011.011....0001....
  298     9 0009    ....0010.001.101.100001000100001
  299    10 000A    11100100.101.011.011....0010....
  300    11 000B    ....0010.011.0110001....0100....
  301    12 000C    01010000.001.....................
  302    13 000D    11110001.001.....................
  303    14 000E    ....0110.001.0111000....0011....
  305    15 000F    11110001.001.011.011....0011....
```