Macro Montadores

PCS 3216 – Sistemas de Programação – Aula 15

Extraído do cap.7 do livro

D. W. Barron "Assemblers and Loaders" – 2nd ed.

McDonald/Elsevier, 1972

Macro Montadores

- É muito frequente que determinados padrões de código se repitam nos programas, com pequenas variações de uma instância para outra.
- Isso é particularmente verdadeiro quando se utiliza alguma operação comum cuja implementação exige o emprego de diversas instruções de baixo nível.
- Um exemplo marcante são as sequências de chamada, necessárias à ativação de subrotinas, especialmente quando tais chamadas envolverem a passagem de parâmetros.
- **Macros** são recursos que podem ser incluídos nos montadores para permitirem ao seu usuário a criação e a instanciação de tais *abreviaturas* em seus programas.

Macro Montadores

- Assim, a chamada da subrotina **SUB** com parâmetros **A** e **B**, que poderia ser confortavelmente denotada como:

  CALL SUB,A,B

  em linguagens de montagem convencionais costuma assumir o aspecto seguinte:

  LDX 4,*
  TFR SUB
  NOP A
  NOP B

- A notação abreviada é sem dúvida muito mais atraente.
- Montadores que tenham sido programados para converter automaticamente as abreviaturas para a forma extensa são conhecidos como **macro montadores**.

Vantagens

- Há três vantagens a destacar para o uso de macros:
  - O programador escreve menos código
  - Os programas, agora mais curtos, ficam mais legíveis, e portanto, sua manutenção fica mais simples
  - Se em outra ocasião houver alguma alteração na estrutura da sequência de chamada, bastará modificar apenas uma região confinada do código (onde são definidas as macros) para corrigir todo o programa, sem que seja necessário localizar e alterar individualmente cada uma das suas instâncias.

Um macro montador simples

- Vamos descrever agora um macro montador simples.
- Estaremos interessados apenas em estudar os novos aspectos, pressupondo que os recursos já estudados dos montadores convencionais estejam todos disponíveis.
- Inicialmente, vamos especificar uma **pseudo-instrução** adicional, destinada a **definir uma nova macro**.
- Seu **mnemônico** será **MACRO**, que recebe no campo de **rótulo** o **nome da macro** que estiver sendo definida, e no campo de **operandos,** uma **lista de parâmetros** formais separados por vírgulas:

CALL MACRO A,B,C

Um macro montador simples

- O programador deve então redigir uma sequência de linhas de código (denominada **corpo da macro**), que poderá incluir referências aos **parâmetros formais**, e deverá ser encerrada através de uma **pseudo-instrução** especial, representada pelo mnemônico **MEND**, que delimita o **final da declaração** dessa macro (de forma similar ao mnemônico END, que delimita o final do texto do programa-fonte).

Um macro montador simples

- Em sequência a essa declaração, o programador pode passar a utilizar a nova macro no seu programa, escrevendo para isso o nome da macro no campo de mnemônicos, e uma lista de argumentos no campo de operando da linguagem de montagem. Assim,

  CALL BLOGGS, FO, FUM

  Representará a seguinte sequência:

  LDX 4,*
  TFR BLOGGS
  NOP FO
  NOP FUM

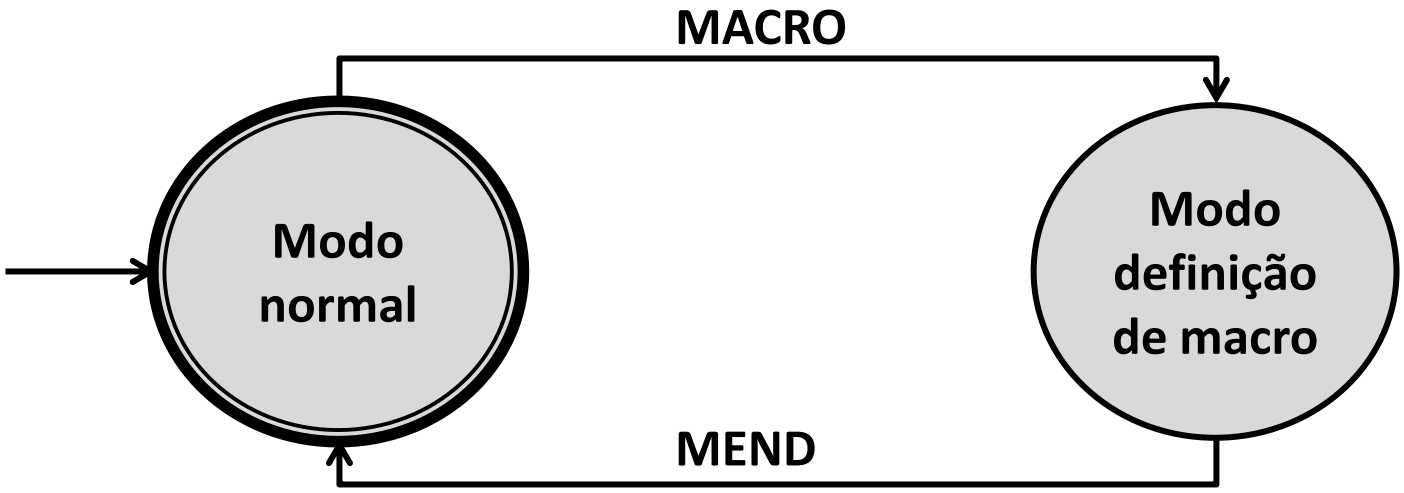
- Pode-se agora aplicar a esse resultado o processo normal de montagem, como se o programador o tivesse pessoalmente inserido em seu programa.

- Dessa forma, o montador deverá manter uma tabela com as macros declaradas no programa e as respectivas definições (parâmetros formais e corpo da macro)
- O montador, ao analisar o campo de mnemônicos
  - Se encontrar a pseudo MACRO, deverão ser coletados os parâmetros formais declarados bem como o corpo da macro (linhas subsequentes, até ser encontrada a pseudo MEND.
  - Se encontrar um nome de macro, esta deverá ser expandida antes de ser aplicado o tratamento usual às linhas de código simbólico resultantes da expansão

IMPLEMENTAÇÃO

Definição de macro

- As definições de macro são processadas no primeiro passo de um montador de 2 passos.
- Ao encontrar a pseudo MACRO no passo 1, o montador altera seu modo de operação para *modo de definição de macro,* do qual retornará ao *modo normal* ao encontrar a pseudo MEND.



Definição de macro

- Ao entrar no modo de definição de macro, o montador cria inicialmente mais um elemento na sua **tabela de definições de macros**, e:
  - Preenche o campo de nome desse elemento com o nome atribuído à macro pelo programador, ou seja, aquele declarado no campo de rótulos da pseudo MACRO.
  - Os parâmetros formais, extraídos do campo de operandos da pseudo MACRO, são incluídos em uma *lista auxiliar de parâmetros formais.*
  - As linhas seguintes de código são lidas e copiadas para a tabela de definições de macros.
  - Durante esse processo de cópia, buscam-se ocorrências de instâncias dos parâmetros formais nas linhas lidas, e tais instâncias são substituídas por identificadores especiais da forma @n (que significa “o n-ésimo parâmetro formal”).

Definição de macro

- A aparência da macro CALL na tabela de definições de macros torna-se:
  - Nome = **CALL**
  - Corpo =

```
LDX   4,*
TFR   @1
NOP   @2
NOP   @3
```

EXPANSÃO DE MACROS

Expansão de macro

- Quando o montador encontra uma chamada de macro, cabe-lhe substituir essa chamada pelo texto que a define (corpo da macro), devidamente modificado pela inserção dos argumentos utilizados nessa particular chamada, em substituição aos parâmetros formais.
- Esse processo de expansão de macros deve acontecer tanto no passo 1 como no passo 2.
- Como o passo 1 costuma inserir textos expandidos na sua cadeia de saída, para serem processados no passo 2, em geral convém garantir que tal expansão elimine a necessidade de execução de quaisquer expansões adicionais no passo 2.

Expansão de macro

- Como a expansão da macro envolve busca de cadeias de caracteres na cadeia de entrada, em geral é mais econômico ponto de vista de tempo de processamento fazer isso uma única vez.
- Esse método assume que o montador disponha de área em disco para o armazenamento de informações transferidas entre um passo e outro.
- Quando for inevitável que o programa fonte seja lido a partir de mídias externas em cada um dos passos, (isso era comum em máquinas pequenas e antigas), então não haverá como impedir que a expansão seja feita mais de uma vez.

Expansão de macro

- A expansão de macros no Passo 1 é necessária a fim de permitir ao montador que mantenha atualizados seus contadores de instrução e outros ponteiros.
- Aparentemente, isso poderia ser evitado guardando-se o comprimento da macro na tabela de definições, porém isso acarretaria como consequência a inviabilização de dois recursos importantes, que exigem reinstanciamentos:
  - o uso de macros aninhadas.
  - o uso de rótulos locais nas macros.

Expansão de macro

- Vamos admitir que a expansão das macros seja restrita apenas ao Passo 1.
- O processo de montagem é modificado da seguinte forma:
  - Quando a o código de operação estiver sendo processado, os símbolos que ocorrem são primeiro colocados na tabela de definição de macros.
  - Se ele não for aí encontrado, as tabelas de pseudos e de mnemônicos são examinadas como de costume.
  - Essa sequência é importante, pois essa sequência de operações permite que o programador substitua , em suas linhas de código fonte, operações básicas de máquina por pequenos grupos de comandos.

Expansão de macro

- Por exemplo, a título de ajuda ao diagnóstico de operandos-destino é possível definir todos os comandos de armazenamento na forma de macros, as quais se expandirão em grupos de comandos que verifiquem a validade ou não dos endereços de armazenamento utilizados.
- Se o símbolo encontrado no campo de mnemônicos for localizado na tabela de definição de macros, os seus argumentos deverão ser memorizados em uma lista auxiliar, e o montador chaveia seu modo de operação para o *modo de expansão de macros*.

Expansão de macro

- Nesse modo de operação, o montador deixa de ler o programa-fonte, e passa a ser alimentado com o conteúdo da tabela de definição de macros, e sempre que for encontrada a sequência *@n*, o argumento adequado é obtido a partir da lista auxiliar de argumentos e inserido em seu lugar, na sequência de entrada.
- Ao final do corpo da macro, o montador encontra a pseudo MEND e nessa ocasião, chaveia de volta para o seu modo de operação normal.
- Considerando que as definições de macro são processadas completamente no primeiro passo, elas podem ser removidas do texto do programa fonte a ser processado pelo segundo passo de montagem.

Expansão de macro

- Muitos montadores disponibilizam certas macros predefinidas, as quais costumam ser denominadas *macros de sistema*.
- A fim de permitir que o programador tenha a opção de criar suas próprias definições para macros de sistema, as consultas à tabela de definições de macros devem ser efetuadas a partir dos seus elementos mais recentes, através de buscas lineares.

CHAMADAS ANINHADAS DE MACROS

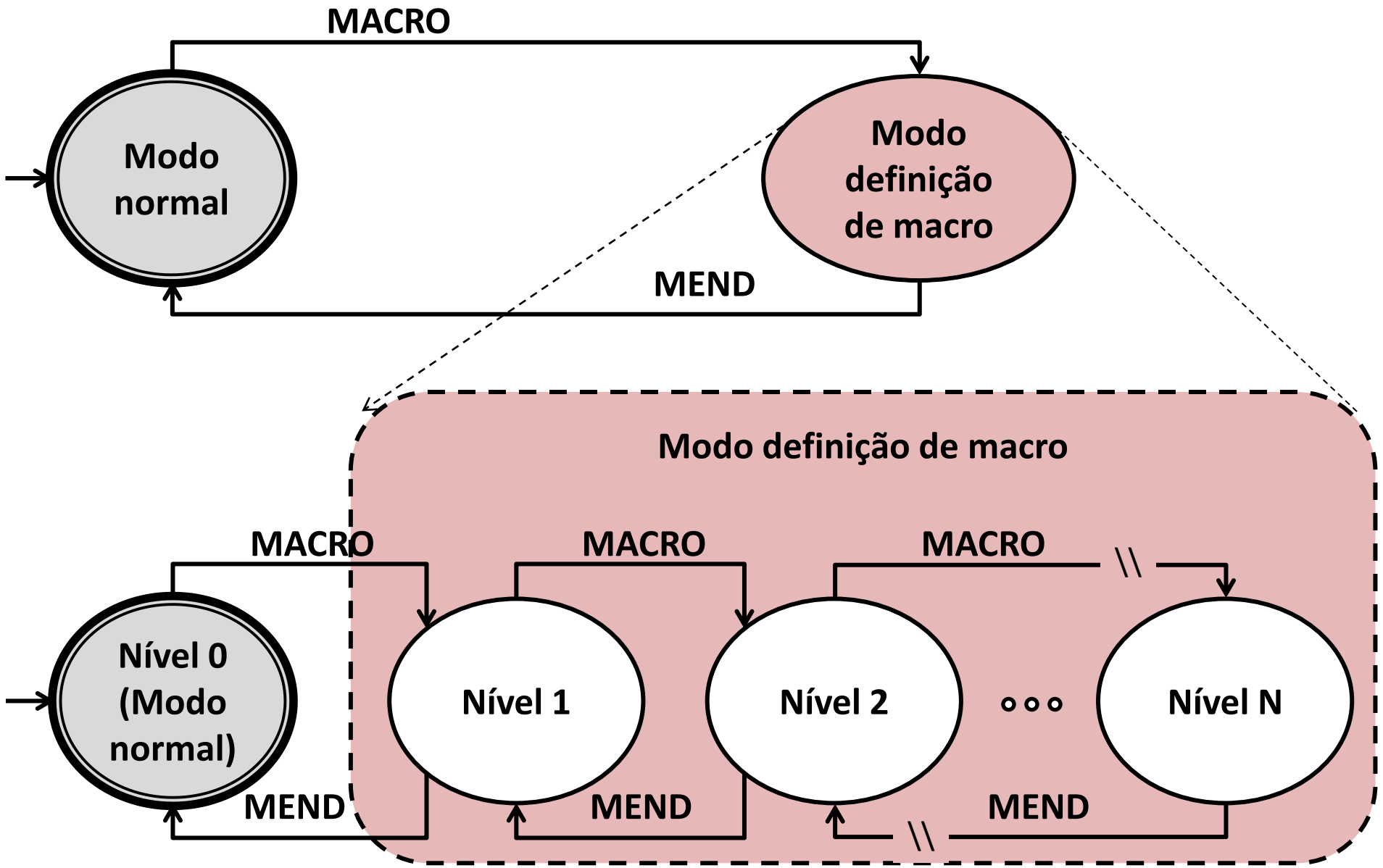
Macros aninhadas

- Existe uma vantagem evidente que pode ser obtida permitindo-se que o corpo da macro contenha chamadas de outras macros .
- De fato, sem essa possibilidade de aninhar chamadas de macros, perde-se uma boa parte de sua utilidade.
- Para a incorporação desse recurso, não é muito grande a complexidade adicional necessária à sua implementação no montador.

Macros aninhadas

- No lugar de simplesmente distinguir entre *modo normal* e *modo expansão de macro*, é necessário associar, ao estado do montador, um **nível de aninhamento** associado à expansão de macros em cada momento.
- O nível zero corresponde ao antigo modo normal e o nível de aninhamento é incrementado de uma unidade para cada chamada de macro cuja expansão tenha sido iniciada mas não ainda completada.

Modelo de estados p/ Macros aninhadas



Macros aninhadas

- Portanto, quando o nome de uma macro é reconhecido no campo de mnemônicos, a ação do montador é a seguinte:
  - Se o nível for nulo, incrementá-lo, e entrar no módulo de expansão de macros como era feito anteriormente
  - Se o nível for diferente de zero, então:
    - empilhar o conjunto corrente de argumentos e um ponteiro para a última posição da tabela de definição de macros
    - incrementar o número do nível de aninhamento
    - copiar os novos argumentos para a lista auxiliar e iniciar a expansão da nova macro

Macros aninhadas

- Quando o final da definição da macro é atingido, se o nível de aninhamento for 1, ele deve ser zerado e o modo normal deve ser retomado.
- Caso contrário, o nível é decrementado, os argumentos da macro anterior e o ponteiro são restaurados a partir da pilha e a expansão macro anterior é retomada.
- Convém notar que quando chamadas de macros são aninhadas, seus argumentos podem ser utilizados do nível mais externo para o mais interno

Macros aninhadas

- Por exemplo seja uma macro definida como segue:

```
LARGER    MACRO    A,B
          LAC      A
          SUB      B
          TFP      *+2    (desviar se positivo)
          ZAC             (zerar acumulador)
          ADD      B
          MEND
```

- Essa macro seleciona e deixa no acumulador o maior dentre dois argumentos.

Macros aninhadas

- Podemos definir outra macro, LARGEST, que selecione o maior dentre três argumentos:

```
LARGEST   MACRO    A,B,C
          LARGER   A,B
          SUB      C
          TFP      *+2
          ZAC
          ADD      C
          MEND
```

- Suponhamos agora que encontremos em um programa uma chamada à macro LARGEST:

```
          LARGEST  X,Y,Z
```

Macros aninhadas

- A expansão dessa chamada da macro LARGEST se inicia com os argumentos X, Y e Z.
- LARGER é reconhecido como nome de macro, e o nível de aninhamento é incrementado, empilhando-se os argumentos de LARGEST.
- Logo, os parâmetros de LARGER aparecem na definição de LARGEST como @1 e @2 e, nesse contexto, devem ser substituídos por valores extraídos da pilha, e não da lista auxiliar.

Macros aninhadas

- Convém ainda observar que esse mecanismo funciona corretamente mesmo se o corpo de uma macro contiver chamadas para a própria macro que estiver sendo definida.
- No entanto, isso só pode ter uso prático se o corpo da macro incluir alguns comandos de montagem condicional que garantam a finalização do processo recursivo de expansão.

RECURSOS ADICIONAIS DO MACRO MONTADOR

Valores “*default*”

- É frequentemente útil que se possa especificar um valor a ser adotado por omissão de um parâmetro da macro. Por exemplo:

```
EXAMPLE   MACRO   A,B=5,C
          LAC     A
          ADD     B
          STO     C
          MEND
```

Valores "*default*"

- Chamando EXAMPLE com os argumentos 1,2,3 ela será expandida como

```
LAC     1
ADD     2
STO     3
```

- Para os argumentos 1,,3 teremos

```
LAC     1
ADD     5
STO     3
```

- Para os argumentos 1,2 porém, o montador gerará apenas uma mensagem de erro.

- A implementação desse recurso exige apenas que poucas informações adicionais sejam mantidas na tabela de definição de macros.

Palavras-chave

- No sistema de macros descrito até aqui, parâmetros são identificados por sua posição em uma lista, de modo que se um parâmetro for omitido, isso será indicado por duas vírgulas sucessivas.
- Um recurso conveniente para o usuário, que permite que os argumentos apareçam em qualquer ordem, e até que sejam totalmente omitidos, pode ser obtido denotando os argumentos como:

Parâmetro formal = Valor do argumento

Palavras-chave

- Por exemplo, para uma chamada da macro LARGEST será possível escrever:

  LARGEST A=PIG, B=DOG, C=CAT

  LARGEST C=CAT, A=PIG, B=DOG

- Para implementar esse recurso, é necessário preservar os nomes dos parâmetros formais e dos eventuais valores “default”.
- Quando a macro é chamada, esses nomes são comparados com as palavras-chave que estiverem na lista de argumentos, com a finalidade de atualizar a tabela de argumentos

Rótulos locais em macros

- Seja a seguinte definição de macro:

```
STORE   MACRO   A
        STO     A
        TFP     L
        STZ     A
L       NOP
        MEND
```

- Essa macro tem o efeito de uma instrução de armazenamento se o acumulador for positivo, mas guarda um zero em caso contrário.
- Se essa macro for chamada mais de uma vez, haverá um problema, pois ocorrerá o erro de redefinição do rótulo L, portanto não será possível utilizar a macro da forma como está.

Rótulos locais em macros

- É desejável que o montador modifique sistematicamente os rótulos definidos no corpo da macro, para garantir que cada instância de cada um deles assuma um valor único e inédito a cada chamada da macro.
- Isso é fácil de implementar:
  - mantendo-se uma variável N que efetue a contagem do número de chamadas de macro (incrementado a cada chamada de macro encontrada no programa), e
  - concatenando-se esse número a todas as ocorrências e todos os rótulos definidos no corpo da macro.

Rótulos locais em macros

- Assim, supondo que o programa contenha apenas duas chamadas de macro STORE, e que estas sejam as duas primeiras chamadas de macros no programa, o código gerado pela expansão de macros deverá conter:
- Na primeira expansão:

```
                STO     X
                TFP     L0001
                STZ     X
L0001           NOP
```

Rótulos locais em macros

- E na segunda expansão:

```
                STO     Y
                TFP     L0002
                STZ     Y
L0002           NOP
```

- Um esquema similar pode ser adotado para gerar nomes únicos para posições de memória utilizadas como áreas temporárias de rascunho pelo programa.

Montagem condicional em macros

- O recurso da montagem condicional pode ser muito útil quando usado no corpo de uma macro, especialmente se permitir o teste do valor dos argumentos.
- Por exemplo, suponha que seja necessário definir uma macro MOVE tal que MOVE A,B transfira o conteúdo de A para B; suponha que desejemos ser capazes de escrever AC (com seu sentido usual) como um argumento para a macro MOVE.

Montagem condicional em macros

- O código que desejamos gerar é ilustrado neste exemplo:

```
MOVE    X,Y
```

deve ser convertido em

```
STO     TEMP
LAC     X
STO     Y
LAC     TEMP
```

enquanto

```
MOVE    AC,Y
```

se transforma em

```
STO     Y
```

e

```
MOVE    X,AC
```

se traduz para

```
LAC     X
```

Montagem condicional em macros

- Isso se obtém com a seguinte definição:

```
MOVE    MACRO     A,B
        AIF       (A .NE. AC) .SS1
        STO       B
        AGO       .SS2
.SS1    AIF       (B .NE. AC) .SS3
        LAC       A
        AGO       .SS2
.SS3    STO       TEMP
        LAC       A
        STO       B
        LAC       TEMP
.SS2    MEND
```

Montagem condicional em macros

- A definição da macro MOVE, mostrada no slide anterior, poderia ser simplificada acrescentando-se uma pseudo-instrução MEXIT, cujo significado seria “finalizar a expansão da macro como se o final físico da definição tivesse sido atingido”.
- Esse recurso é particularmente útil quando se prepara um programa para que se torne compatível com uma série de máquinas, já que modificar um só parâmetro torna possível gerar códigos bastante diferentes, que venham a atender as exigências de diversas configurações.

Concatenação

- Argumentos podem não somente ser substituídos em campos especificados do corpo da macro, mas podem também ser concatenados com cadeias de caracteres nele contidos.
- O ponto (“.”) costuma ser usado como operador de concatenação: se no corpo da macro for encontrada a cadeia HEAD.BASE, e se HEAD for um parâmetro formal, então o valor do argumento correspondente a HEAD será concatenado com a cadeia “BASE” quando a macro for utilizada.

Concatenação

- Isso possibilita utilizar macros nas quais os argumentos são inseridos em expressões complexas. Por exemplo:

```
SKIP    MACRO   N
        TFR     N. + * + 1
        MEND
```

- A chamada desta macro:

```
        SKIP    3
```

será expandida como

```
        TFR     3 + * + 1
```

ou seja, um desvio relativo que salta as próximas três instruções.

Número variável de argumentos

- Alguns macro montadores disponibilizam uma variável de sistema cujo valor é o número de argumentos usado em uma particular chamada da macro.
- Em associação com recursos de montagem condicional, isso viabiliza o uso de chamadas com um número variável de argumentos
- Tal recurso costuma aparecer associado a uma notação que permita referenciar os argumentos segundo sua posição relativa na lista de argumentos.

Variáveis globais

- O recurso de montagem condicional emprega variáveis locais ao corpo das macros.
- Com isso, é possível fazer o código gerado para uma chamada particular depender dos efeitos da expansão de outras macros previamente chamadas.
- Isso viabiliza, por exemplo, escrever uma macro que consulte um argumento a partir de um registrador do processador, permitindo assim, por exemplo, impedir a geração de sequências STORE-LOAD redundantes.

Atributos

- O recurso dos atributos dá acesso a metadados a respeito de um parâmetro específico (por exemplo, se é um inteiro, uma cadeia, um número em ponto flutuante, etc.), dando ao programador a oportunidade de otimizar o código resultante da expansão das macros.

Macro montadores e macro processadores

- Embora os recursos de macros tenham feito parte integrante de um macro montador, é evidente a independência conceitual existente entre os processos de montagem e de expansão de macros.
- Isso se nota observando como o macro montador se comporta em modo expansão de macros (simplesmente chaveando a origem do texto de entrada).

Macro montadores e macro processadores

- Processadores de macros efetuam seu trabalho considerando-o uma pura manipulação de material textual, independente de qualquer linguagem de programação ou de seus tradutores.
- Usado como um passo extra para um montador ou compilador, um processador de macros pode ter papel bastante importante e poderoso, já que permite o preenchimento (automático), através da expansão de macros, de regiões selecionadas do texto de entrada.

FIM