CAPÍTULO 7 – NÍVEL DE LINGUAGEM DE MONTAGEM

7.1 Introdução

- Difere dos níveis inferiores por ser implementado por tradução.
- A tradução é usada quando um processador está disponível para uma mensagem fonte mas não para uma linguagem alvo
- O programa tradutor converte um programa fonte (escrito numa linguagem fonte) para um programa equivalente, o programa objeto (na linguagem de máquina do processador disponível).
- A tradução é feita em duas etapas, as quais são realizadas em seqüência:
  - Geração de um programa em linguagem alvo (programa objeto).
  - Execução do programa gerado.
- Existem dois tipos de tradutores:
  - Compilador: linguagem fonte = linguagem de alto nível; linguagem objeto = linguagem de máquina.
  - Montador (*Assembler*): linguagem fonte = linguagem de montagem (*Assembly*) representação simbólica da linguagem de máquina; linguagem objeto = linguagem de máquina.

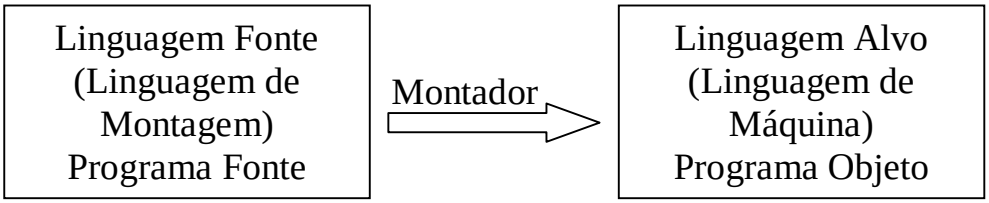

Figura 7.1. Processo de montagem.

7.2 Linguagem de Montagem

- Razão de uso da linguagem de montagem:
  - Melhorar o desempenho.
  - Algumas máquinas podem não ter um compilador disponível.
- Cada comando em linguagem resulta em um comando em linguagem de máquina: uma linha de programa fonte ≡ uma linha de programa objeto.
- Representação simbólica da linguagem de máquina: Códigos de operação e operandos (números) substituídos por mnemônicos e rótulos.
- Ao contrário da programação em alto nível, o programador de linguagem de montagem dispõe de todos os recursos do nível ISA.
- Programa não portável (válido apenas para uma mesma família de processadores).

7.3 Formato de um Comando em Linguagem de Montagem

- Uma instrução em linguagem de montagem contém, pelo menos, um campo de operação e outros campos que a especificam.

Label	Operação	Operando(s)	Comentários
K:	DD	2	; Define Dado K reservando 4 bytes iniciando com 2.
DESTINO	ADD	EAX, K	; Adicione K a EAX
	JUMP	DESTINO	; Desvie para DESTINO

Figura 7.2. Exemplo de formato de comandos em linguagem de montagem.

- Campo de Rótulo (*Label*): nome simbólico atribuído a um endereço. Permite que destinos de desvios sejam identificados facilmente pelo programador.
- Campo de Operação: abreviatura simbólica do OPCODE. Deve lembrar o tipo de operação realizada pela instrução. Exemplo: ADD = soma.
- Campo de operando(s): especifica simbolicamente o(s) endereço(s), registrador(es) ou constante(s) utilizados como operandos.
- Campo de comentários: documento o programa (o qual seria ilegível sem este campo). Ignorado pelo montador.

7.4 Linguagem de Montagem *versus* Linguagem de Alto Nível

- Um programa em linguagem de montagem é mais eficiente, é mais difícil de programar e tem de 5 a 10 vezes mais instruções do que o seu equivalente em linguagem de alto nível. Envolve mais programadores (a um custo maior)
- Independente da linguagem, um programador pode produzir um determinado número de linhas de código por mês.
- Programas em linguagem de montagem são difíceis de entender. Mudanças de programadores são bastante prejudiciais.
- Num programa, uma pequena percentagem do código é responsável por grande parcela do tempo de execução. Solução: Afinação = implementação dos trechos críticos do programa em linguagem de máquina. Trechos não críticos desenvolvidos em linguagem de alto nível.

7.5 O processo de Montagem

- Montagem: tradução de programa fonte em programa objeto.
- Problema da Referência Futura: desvios para posições de destino situadas adiante da posição da instrução de desvio são representados por símbolos ainda não definidos. Conseqüência: não é possível converter o programa diretamente, linha após linha.
- Solução - Tradução em Dois Passos:
    1. Definição de símbolos, armazenados em uma tabela.
    2. Tradução do programa usando os símbolos definidos no passo 1.

    - Passo 1:

        - O montador analisa as instruções, uma a uma, até encontrar uma pseudo-instrução de fim de programa (END).
        - No processo de análise das instruções, usando uma Tabela de Códigos de Operação, o montador constrói uma Tabela de Símbolos, a qual será usada no Passo 2.

        - Tabela de Códigos:
            - A Tabela de Códigos contém uma entrada para cada código de operação (*opcode*) simbólico (mnemônico) da linguagem de montagem.
            - A tabela tem vários campos que permitem, a partir do mnemônico, definir o seu valor em linguagem de máquina, o tamanho e tipo dos operandos, o tamanho da instrução, etc.:
                - Campo de Código de Operação Simbólico: contém o mnemônico associado à instrução.
                - Campos de Operandos: contêm o tipo de operandos manipulados pela instrução.
                - Campo de Código de Operação: contém o valor numérico do código de operação em linguagem de máquina. Para cada tipo de operandos (modos de endereçamento diferentes), utiliza-se um código de operação diferente.
                - Campo de Tamanho de Instrução: contém o tamanho da instrução correspondente (usado para incrementar o ILC).
                - Campo de Classe de Instrução: contém um número de classe que separa códigos de operação em grupos dependentes do número e do tipo de operandos. Permite ao montador tratar da mesma maneira instruções semelhantes.

Código de operação simbólico	Tipo do primeiro operando	Tipo do segundo operando	Código de operação em Hexa	Tamanho da Instrução em bytes	Classe da instrução

Figura 7.3. Exemplo de entrada de uma Tabela de Códigos de Operação.

- Construção da Tabela de Símbolos:
  - Instruções são analisadas, uma a uma, em seqüência.
  - Contador de Localização de Instrução (ILC - *Instruction Location Counter*), zerado no início do Passo1, é incrementado do comprimento da instrução corrente a cada instrução analisada.
  - O ILC provê o endereço "de execução" da instrução montada.
  - Com base no ILC, uma Tabela de Símbolos é montada.
  - A Tabela de Símbolos contém os seguintes campos:
    - Campo de símbolo: contém o próprios *labels*, (ou um ponteiro para os mesmos).
    - Campo de Valor: contém o valor numérico atribuído ao símbolo (o valor do ILC da instrução).
    - Outras informações (campos opcionais). Exemplo:
      - Comprimento dos dados associados ao símbolo.
      - Bits de relocação, que especificam se o símbolo muda de valor se o programa é carregado em um endereço diferente daquele assumido pelo montador.
      - Informações sobre se o símbolo é ou não acessível fora do procedimento.

Símbolo	Valor do ILC	Outras informações

Figura 7.4. Uma entrada da Tabela de Símbolos.

- Passo 2:

  - Gera o programa objeto. Produz a expansão binária da instrução a partir das tabelas.
  - Produz informações necessárias ao procedimento de Ligação.
  - Gera mensagens de erro caso estes existam no programa fonte.

7.6 Macros

- Macros são abreviaturas simbólicas para trechos do programa fonte.
- Usadas para diminuir o tamanho do código quando um mesmo trecho de linhas de código aparece repetido várias vezes no programa fonte.

- Definição de uma Macro:
  - Só aparece uma vez no programa fonte. Possui a seguinte estrutura:
    - Cabeçalho da Macro: contém um nome mnemônico da Macro.
    - Corpo da Macro: contém as linhas de código, propriamente ditas, que executam a tarefa definida para a Macro.
    - Fim de Macro: pseudo-instrução de fim de Macro informa ao montador onde a Macro termina.
  - O montador deve manter uma tabela de Macros com o nome de cada uma delas e ponteiros correspondentes para uma tabela de definições das mesmas, onde estão armazenados os corpos das Macros.

- Chamada de Macro:
  - É feita no programa fonte através do nome da Macro.
  - O nome da Macro substitui o corpo da Macro em todos os lugares onde este deveria aparecer dentro do programa fonte.

```
MOV  EAX, P
MOV  EBX, Q
MOV  Q, EAX
MOV  P, EBX

      .
      .
      .

MOV  EAX, P
MOV  EBX, Q
MOV  Q, EAX
MOV  P, EBX
```

(a)

```
TROCA   MACRO
        MOV  EAX, P
        MOV  EBX, Q
        MOV  Q, EAX
        MOV  P, EBX
        ENDM
              .
              .
              .
        TROCA
              .
              .
              .
        TROCA
```

(b)

Figura 7.5. Código em Assembly. a) Sem Macro. b) Com Macro.

- Expansão de Macro:
  - Substituição da chamada de Macro pelo corpo da Macro.
  - É realizada pelo montador no início do Passo 1 da montagem.
  - Após a expansão, o corpo da Macro pode aparecer repetidas vezes no programa.
  - O programa objeto não tem chamadas de Macro.
  - Diferentemente das chamadas de procedimentos, as chamadas de Macro não precisam de instrução de retorno.

Parâmetro de Comparação	Chamada de Macro	Chamada de Procedimento
Quando é feita a chamada?	Durante a montagem	Durante a execução
O corpo é inserido no programa objeto em todo lugar em que aparece?	Sim	Não
Uma instrução de chamada é inserida e executada no programa objeto?	Não	Sim
Precisa de instrução de retorno para retomar a instrução seguinte à chamada?	Não	Sim
Quantas cópias do corpo aparecem no programa objeto?	Uma para cada chamada	Uma

Figura 7.6. Comparação entre Macros e Procedimentos.

- Macros com parâmetros:

  - Algumas Macros podem receber parâmetros.
  - Na definição da Macro, utilizam-se parâmetros formais (genêricos). Exemplo de cabeçalho: SOMA3 MACRO A, B, C.
  - Na chamada da Macro, utilizam-se os parâmetros reais. Exemplo: SOMA3 X, Y, Z; SOMA3 K, L, M; etc.
  - Quando a Macro é expandida, cada parâmetro formal é substituído pelos correspondentes parâmetros reais.
  - Os parâmetros reais são colocados no campo de operandos da chamada da Macro correspondente.

7.7 Ligação e Carga

- Programas podem ser constituídos de vários procedimentos.
- Antes da execução, todos os procedimentos traduzidos devem ser ligados apropriadamente, em um único programa, a ser carregado na memória principal para execução.
- O *software* que faz a ligação é denominado Ligador (*Linker*).
- O *linker* produz um único programa, o Módulo Absoluto de Carga, a partir dos vários módulos objeto obtidos da tradução dos vários procedimentos (módulos fonte).
- O Módulo de Carga contém todos os módulos ligados em um único programa executável.
- Um *software* denominado Carregador (*Loader*) é responsável por carregar o Módulo de Carga na memória principal para executá-lo.

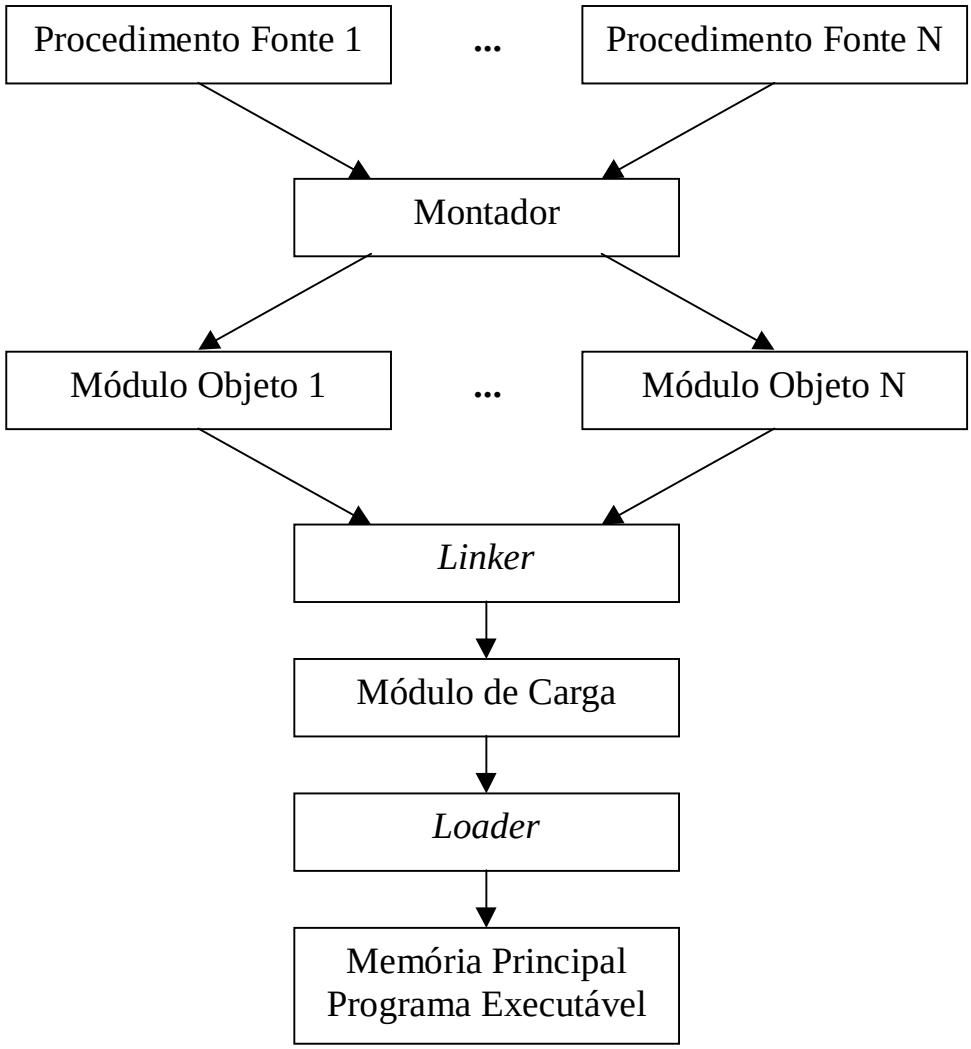

Figura 7.7. O processo de Montagem, Ligação e Carga.

- Para criar o Módulo de Carga, o *Linker* une todos os Módulos Objeto em um único espaço de endereçamento:
  - As referências a endereços devem ser todas atualizadas (Problema de Relocação). Este problema é inexistente quando a memória é segmentada.
  - Quando existe um Procedimento A que chama a um Procedimento B, o endereço absoluto de B só é conhecido após a ligação (Problema de Referência Externa).

- Tarefas do *Linker*:

  i. Construir uma Tabela com todos os Módulos Objeto e seu respectivos comprimentos.
  ii. Atribuir um Endereço de Carga a cada Módulo Objeto.
  iii. Relocar todas as instruções que contêm um endereço adicionando uma Constante de Relocação (endereço inicial de cada módulo).
  iv. Encontrar todas as instruções que referenciam outros procedimentos e inserir nelas o endereço absoluto dos mesmos.

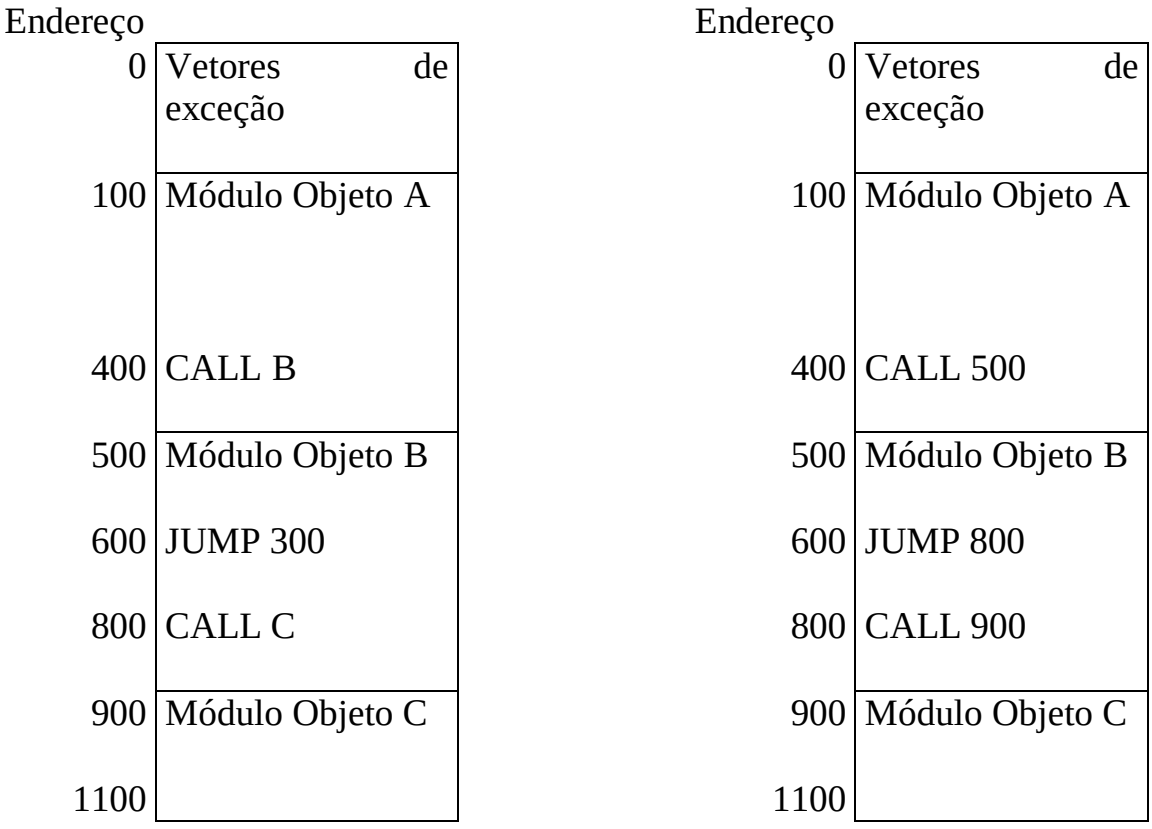

Figura 7.8. Módulos objeto. a) Antes da ligação. b) Depois da Ligação.

Módulo	Comprimento	Endereço Inicial
A	400	100
B	400	500
C	200	900

Figura 7.9. Exemplo de Tabela de Módulos Objeto.

- Estrutura de um módulo objeto:
  - A maioria dos *Linkers* requer dois passos:
    i. Ler todos os módulos objeto, construir uma Tabela de nomes e comprimentos de módulos e uma Tabela global de símbolos internos e externos.
    ii. Ler os módulos objeto, relocá-los e ligá-los para formar um único módulo.

Campo	Conteúdo
Identificação	Nome do módulo e comprimentos das suas partes
Tabela de Símbolos Internos	Lista de símbolos definidos no módulo corrente que outros módulos podem referenciar
Tabela de Símbolos Externos	Lista de símbolos definidos em outros módulos utilizados no módulo corrente. Lista de instruções e seus correspondentes símbolos externos
Instruções e constantes	Código montado e constantes (única parte do módulo objeto que será carregada na memória para execução)
Dicionário de Relocação	Lista de endereços a serem relocados
Fim de Módulo	Indicação de fim de módulo

Figura 7.10. Estrutura de um módulo objeto.

- Tempo de Mapeamento (*Binding Time*):
  - Tempo de Mapeamento: tempo no qual um endereço real da memória principal é atribuído a um símbolo. Exemplo: quando o programa é escrito, quando o programa é carregado, quando a instrução que usa o endereço é executada, etc.
  - Em Sistemas de Tempo Compartilhado, programas podem ser carregados em endereços diferentes, quando executados em instantes diferentes. Devem ser relocados dinamicamente.
  - Solução: mapear símbolos em endereços virtuais e alterar a tabela de páginas para realizar o novo mapeamento virtual-físico.

- Ligação Dinâmica:
  - Ligação de procedimentos compilados separadamente no momento em que estes são chamados pela primeira vez.
  - Procedimentos raramente utilizados só são ligados se necessário.
  - Melhor aproveitamento da memória virtual.