PCS 3216
Sistemas de Programação

João José Neto

Aula 10 – Montadores de dois passos

Montadores


Source Program
• Mnemonic opcode
• Symbol
Assembler
Object code

TIPOS MAIS FREQUENTES DE MONTADORES

Conceitos

- Passo da montagem – leituras completas do fonte
- Montadores load and go – ger. de código na mem.
- Backtracking – preencher lacunas a posteriori
- Montadores de dois passos
- Montadores de um passo
- Montadores usuais: Geração de código objeto absoluto compatível com bootstrap loader

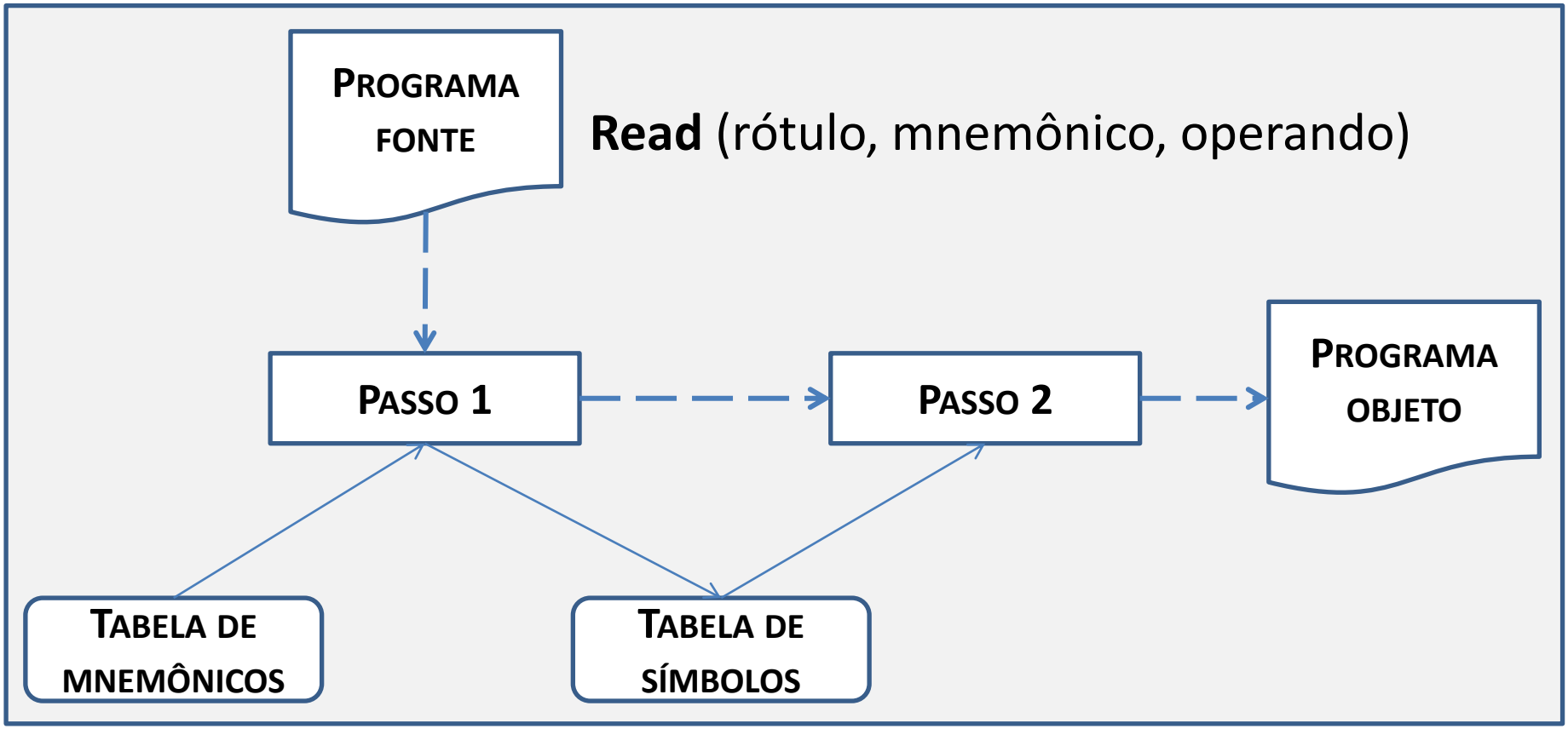
Resolução de endereços simbólicos

- Há duas soluções clássicas para o problema da resolução dos endereços simbólicos:
  - A primeira consiste em se efetuar a coleta dos símbolos e o cálculo dos endereços a eles associados, sem a preocupação em montar o código de máquina das instruções, o que é feito numa segunda etapa.
  - A segunda consiste em efetuar simultaneamente as duas tarefas, sempre que possível, apenas postergando a montagem das instruções com ***referências à frente***, e efetuando esta montagem no instante em que ficar definido o endereço associado ao símbolo em questão.

Um e dois passos

- Aos montadores que funcionam usando a técnica da primeira solução dá-se o nome de ***montadores de dois passos***, porque, para completar a montagem de um programa, o montador necessita efetuar duas leituras completas do mesmo: uma para montar a tabela de símbolos e de atributos, e outra, para efetuar a construção do programa traduzido, em linguagem de máquina, a partir do texto e das tabelas construídas no primeiro passo.
- Os montadores que seguem o segundo esquema são denominados ***montadores de um passo***, exigindo apenas uma leitura do programa e a manutenção de listas de pendências;

Uma organização simples de montador em dois passos



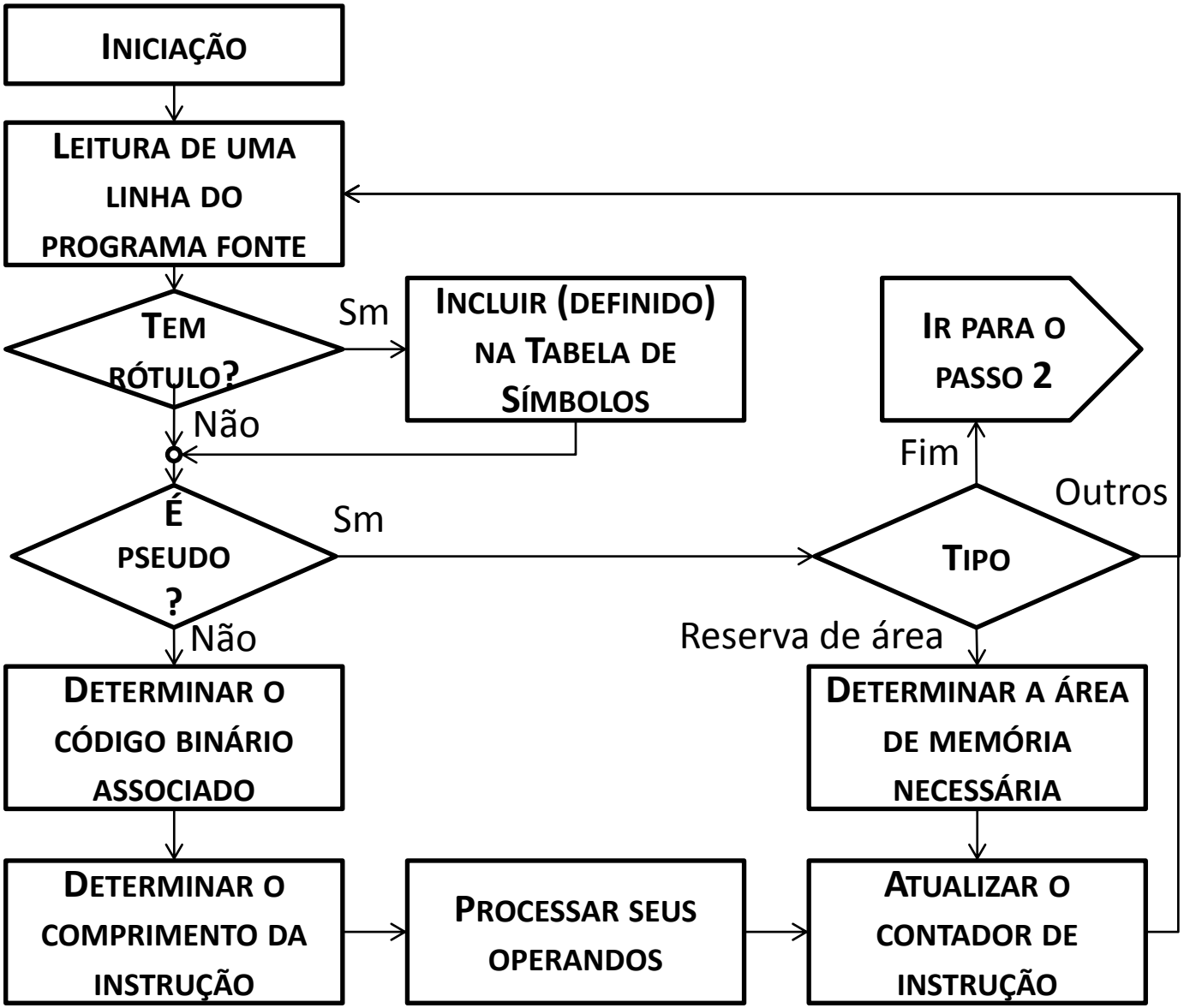
Implementação de montadores de dois passos

- Mais difundida
- Exige menos área de memória
- É mais simples conceitualmente
- Primeiro passo:
  - leitura do programa fonte para a montagem da tabela de símbolos
  - Análise da tabela de símbolos em busca de inconsistências
- Segundo passo:
  - Releitura do programa fonte para a montagem do código-objeto

PASSO 1

- Construção da tabela de símbolos
- Consulta à tabela de mnemônicos
- Construção da tabela de equivalências
- Cálculo do endereço de cada instrução
- Teste de consistência da tabela de símbolos
- Geração de tabelas de referências cruzadas

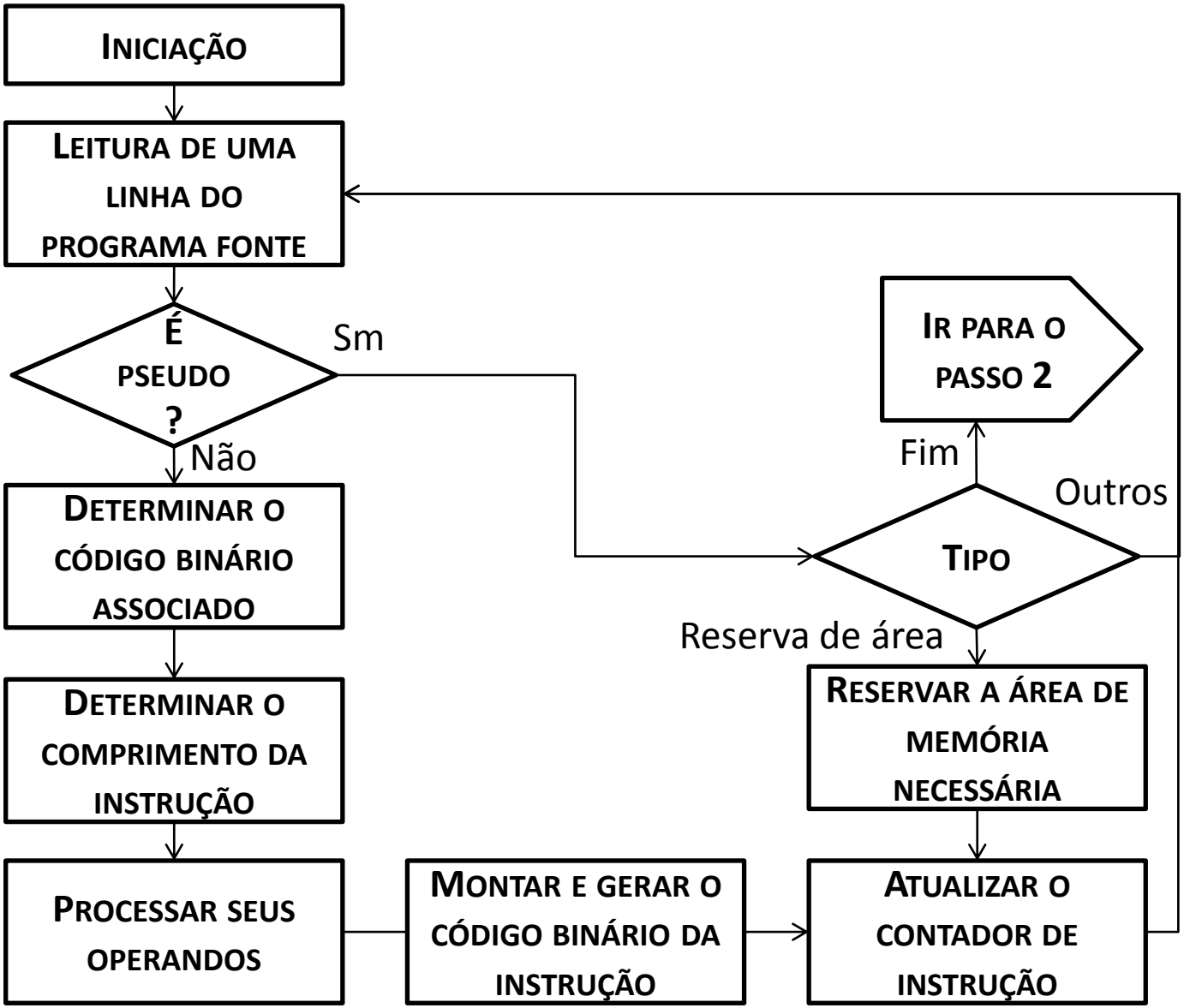
Lógica resumida do passo 1 do montador



PASSO 2

- Consulta à tabela de códigos
- Montagem do código objeto
- Avaliação das expressões dos operandos
- Geração da listagem formatada
- Geração do programa objeto

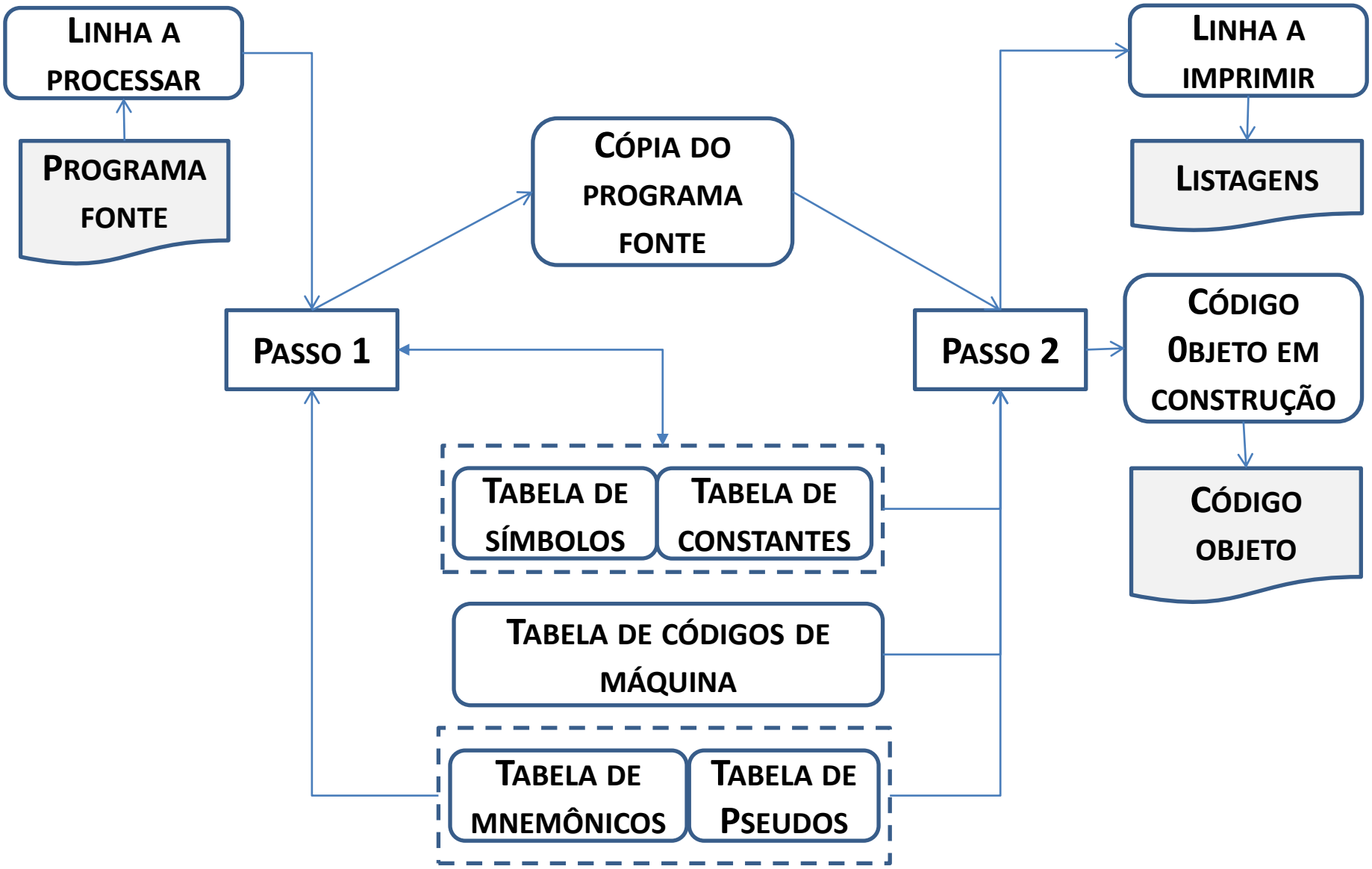
Lógica resumida do passo 2 do montador



Estruturas de Dados principais

- Tabela de símbolos (símbolo - endereço - definido - referenciado)
- Extensão da tabela de símbolo para geração de referências cruzadas
  - Linha de definição
  - Link para ordem alfabética
  - Ponteiro para lista de referências
- Lista de referências (para referências cruzadas)
  - Link
  - Número da Linha
- Tabela de mnemônicos e códigos
  - Mnemônico
  - Código
  - Classe
- Tabela de equivalências
  - Símbolo
  - Link
- Área de saída
  - Bloco de código objeto gerado

Áreas de Dados usadas pelo montador



Coleta de informação sobre os símbolos

- Criadas pelo primeiro passo do montador, guardam informações a serem usadas no segundo passo, sobre os rótulos referenciados no programa.
  - Nome do símbolo
  - Endereço ou valor numérico associado ao símbolo
  - Informação sobre o tipo de relocação necessário no caso de alteração do endereço do programa
  - Informação sobre a acessibilidade ao símbolo fora do módulo em que ele foi definido
- Há muitas formas alternativas de organizar a tabela de simbolos para memorizar a coleção de pares de pares do tipo (símbolo-atributos): vetores de registros, tabelas bidimensionais, listas ligadas etc.

Tabela de símbolos

- O montador constrói a tabela de símbolos
- O uso de tal tabela permite ao programador referenciar por nome as posições de memória
- O montador se incumbe de associar cada nome ao respectivo endereço na memória
- Os endereços associados às diversas variáveis são endereços relativos à posição ocupada na memória pela primeira variável do programa.

Aspecto de uma tabela de símbolos

SÍMBOLO	ENDEREÇO	TAMANHO	RELOCAÇÃO
ABCD	30	1	ABSOLUTO
XYZ	123	50	ABSOLUTO
A1	INDEFINIDO	100	RELOCÁVEL
B	0	20	ABSOLUTO
...			

LÓGICA DETALHADA DE UM MONTADOR DE 2 PASSOS

Início da lógica do montador

- Fazer Contador de Instruções (C.I.) igual a 0
- Fazer Passo igual a 1
- Ler uma linha do programa-fonte
- Se for linha de comentário
  - Se passo igual a 1, ignorar a linha lida
  - Se passo igual a 2, então listar a linha

Se a linha tiver rótulo

- Procurar o rótulo da linha na tabela dos símbolos
- Se já **existe** e está **definido**,
  - reportar **erro de dupla definição**
- Se já **existe** mas ainda está **indefinido**,
  - **defini-lo** c/endereço apontado pelo **contador de instruções**
- Se ainda **não existe**
  - **inserir na tabela** e **defini-lo** c/o endereço apontado pelo **contador de instruções**
- Marcar como definido
- Incluir o **número da linha** associando-a à **definição** do rótulo na tabela de **referências cruzadas**

Analisar o campo de mnemônico

- Procurar na tabela de mnemônicos
  - Se não achar, gerar mensagem de erro
- Atualizar o contador de instruções, conforme o tipo de mnemônico encontrado
- Se o mnemônico exigir operando:
  - Analisar o operando:
    - Incluir eventuais símbolos novos na tabela de símbolos
    - Se não constarem na tabela, marcar como indefinidos
    - Atualizar a tabela de referências cruzadas
  - Avaliar a expressão encontrada no operando
  - Se for passo 2 e houver código-objeto associado a gerar
    - Montar código objeto

Aspecto da Tabela de Mnemônicos

Binário	Mnemônico	Nome	Ação
0000xxxxxxxxxxx	Jmp	Jump	CI:=X
0001xxxxxxxxxxx	Lda	Load	Ac:=Mem[X]
0010xxxxxxxxxxx	Sta	store	Mem[X]:=Ac
...			

Tabela de mnemônicos

- Esta tabela é essencial para a montagem do código-objeto.
- Cada mnemônico do código simbólico tem associada uma linha desta tabela, contendo:
  - O mnemônico simbólico
  - Operandos exigidos pela instrução
  - Valor numérico binário associado a seu código
  - Número de bytes ocupado pela instrução
  - Classe da instrução – número e tipo de operandos

Tratamento de Pseudo Instruções

- Em montadores absolutos, costumam ser encontradas as Pseudo-instruções seguintes:
  - **ORG** – modificar contador de instruções conforme o valor do operando
  - **BLOC** – modificar contador de instruções para contabilizar a área reservada
  - **DB, DW, DA** – se passo igual 2, gerar código objeto
  - **EQU** – se passo igual a 1, atualizar a tabela de equivalências
  - **END** – se passo igual a 1, fazer passo igual a 2. Se não, encerrar os trabalhos do montador.
- Voltar à leitura de nova linha.

ORG (origem)

- Determina nova origem para o código a ser gerado em seguida
  - Em montadores absolutos, o operando deve ser obrigatoriamente absoluto.
  - Em montadores relocáveis, pode ser relocável, absoluto, simbólico, relativo
  - Tratamento: Modificar o contador de instruções conforme especificado no valor do operando

BLOC (reserva área de memória)

- Esta pseudo-instrução determina a ocupação de uma área de memória de comprimento estabelecido, sem preenchimento de dados, disponibilizando-a para uso pelo programa.
- O operando deve ter um valor numérico inteiro não negativo, pois refere-se ao número de palavras de memória a ser reservado.
- Tratamento: em ambos os passos da montagem, atualizar o contador de instruções, adicionando-lhe o valor declarado no seu operando.

DB (define byte)

- Esta pseudo instrução destina-se a preencher o endereço de memória apontado pelo contador de instruções corrente e o seguinte, com o valor (um byte) associado ao seu operando.
- O valor do operando dessa pseudo instrução deve ser numérico, e expresso como número binário de um byte (oito bits).
- Tratamento: se passo igual 2,
  - Gerar código-objeto preenchendo um byte, com o valor do operando, no endereço de memória apontado pelo contador de instruções.
  - Atualizar o contador de instruções incrementando-o de 1.

DW (define word)

- Esta pseudo instrução destina-se a preencher o endereço de memória apontado pelo contador de instruções corrente e o seguinte, com o valor (dois bytes) associado ao seu operando.
- O valor do operando dessa pseudo instrução deve ser numérico, e expresso como número binário de dois bytes.
- Tratamento: se passo igual 2,
  - Gerar código-objeto preenchendo dois bytes, com o valor do operando, nos endereços de memória apontado pelo contador de instruções e seguinte.
  - Atualizar o contador de instruções incrementando-o de 2.

DA (define address)

- Esta pseudo-instrução destina-se a preencher o endereço de memória apontado pelo contador de instruções corrente e o seguinte, com a representação binária de um endereço (dois bytes) associado ao seu operando, a ser usado como ponteiro pelo programa.
- O valor do operando dessa pseudo-instrução deve ser um endereço absoluto, e expresso como um número binário de dois bytes.
- Tratamento: se passo igual 2,
  - Gerar código-objeto preenchendo dois bytes, com o valor do operando, nos endereços de memória apontado pelo contador de instruções e seguinte.
  - Atualizar o contador de instruções incrementando-o de 2.

EQU (equate – definir equivalência)

- Esta pseudo-instrução permite determinar a equivalência entre novos nomes e endereços associados a outros nomes usados no programa-fonte.
- Seu operando precisa ser obrigatoriamente um endereço, de qualquer tipo.
- Tratamento: se passo igual a 1, atualizar a tabela de equivalências

END (final físico do programa-fonte)

- Esta pseudo-instrução permite ao programador informar ao montador que foi atingido o final do programa-fonte.
- Seu operando deve ser um rótulo definido no programa, e deve referir-se a um endereço de memória, relativo a um rótulo do programa, que fornece a informação do endereço a partir do qual está previsto o início da execução do programa.
- Tratamento da pseudo-instrução FIM
  - Se passo igual a 1, fazer passo igual a 2.
  - Se não, encerrar a execução do montador.