PCS 3216
Sistemas de Programação

Aula 14

Aspectos da Implementação de Montadores Relocáveis

INTRODUÇÃO

Objetivo

- O objetivo principal deste material é o de levantar os principais aspectos necessários ao encaminhamento da implementação de um **montador** para linguagem simbólica **relocável.**

Compatibilidades

- Programas de sistema são sempre utilizados conjuntamente, por isso é indispensável que suas **interfaces** sejam totalmente **compatíveis** entre si.
- No âmbito dos sistemas de programação, é necessário haver **compatibilidade de formatos** para que possa haver intercâmbio de informação entre os programas de sistema.
- Destacam-se os **formatos dos textos-fonte** e dos **códigos-objeto absoluto e relocável**, tanto binários como nos **formatos numéricos** ASCII, decimal, hexadecimal e octal.

Formato fonte

- É bastante evidente que, como alguns dos módulos do sistema de programação também geram informação na forma de **textos-fonte**, os formatos adotados devam ser os mesmos.
- Mais que uma **uniformidade de formato**, é preciso destacar a necessidade de **uniformidade de linguagem**, de modo que os comandos de qualquer programa sejam **correta e igualmente interpretados** por todos os programas do sistema de programação.
- É o que acontece no caso dos **compiladores**, os quais, entre outras possibilidades, podem também **gerar programas-fonte como saídas**.

Linguagens-fonte simbólicas

- Uma **compatibilidade** muito desejável é aquela que se pode estabelecer **entre** as linguagens simbólicas **absoluta e relocável**.
- Isto pode ser obtido facilmente:
  - **Padronizando a notação** usada para a representação das instruções de máquina nos dois casos.
  - **Identificando** claramente **o tipo dos operandos** utilizados (constante, endereço simbólico, absoluto ou relocável etc.)
  - **Unificando o programa montador**, para que possa tratar igualmente programas-fonte absolutos e relocáveis.
  - Adotando um conjunto de **pseudo-instruções bem escolhido**, de modo que essa unificação se viabilize.

Para especificar a linguagem-fonte

- As afirmações anteriores sobre compatibilidade e sobre padronização da linguagem fonte para linguagens simbólicas relocável e absoluta, bem como considerações sobre a versatilidade e a praticidade da linguagem permite impor para a linguagem-fonte simbólica do montador
- Pode-se criar, para a linguagem-fonte relocável, uma especificação que viabilize, das especificações do montador para linguagem absoluta:
  - Preservar tudo o que se mantém invariável ou compatível
  - Acrescentar à linguagem apenas novidades que preservem a uniformidade da linguagem
  - Evitar a introdução de elementos destoantes ou conflitantes com os que já estejam em uso com sucesso.

Número de passos

- Sabe-se bem que, além de tradicionalmente serem os mais utilizados na prática, **montadores de dois passos são menos complexos e menos artificiosos** que os equivalentes de um só passo.
- Por outro lado, como se trata de um software composto de **dois programas**, isso sem dúvida onera seu desenvolvimento e sua manutenção, além de poder torná-lo mais lento.
- Todavia, embora não seja obrigatória, neste caso a escolha de efetuar a **montagem em dois passos pode ser uma boa opção** para este projeto.

Linguagem de implementação

- Desejando-se construir um **montador auto-residente e auto-compilável**, uma possibilidade é a de implementá-lo na própria **linguagem da máquina virtual**.
- Todavia, não havendo muito suporte disponível para isso, convém elaborar um **montador cruzado**, escrito em uma linguagem de alto nível qualquer, para ser **executado em outra máquina**, gerando para a máquina virtual um **código-objeto compatível** com os que já foram construídos.
- A grande disponibilidade de compiladores, montadores e outras ferramentas de desenvolvimento também dá ao usuário a **liberdade de escolha do ambiente e da linguagem de implementação** considerados mais adequados.

Montadores para programas relocáveis

- Montadores são programas de sistema responsáveis pela geração correta do programa-objeto, nos moldes apresentados ao longo da disciplina.
- Devem ainda gerar listagens contendo informações sobre referências externas e sobre o tipo de relocação associado a cada endereço de memória referenciado no programa.
- Essas listagens devem incluir uma lista de todos os pontos de acesso ao programa, devidamente acompanhados dos correspondentes endereços.

ESPECIFICAÇÕES GERAIS

A Linguagem Simbólica

- O uso de endereçamento absoluto em linguagens simbólicas vincula o código a posições fixas de memória.
- Programas escritos com endereçamento absoluto só funcionam nos endereços para os quais foram codificados, devido às referências absolutas neles contidas.
- Por essa razão, não é prática a construção de bibliotecas de uso geral usando endereçamento absoluto.
- Contorna-se o problema introduzindo-se o conceito de **relocabilidade**.
- Programas relocáveis apresentam referências a endereços não resolvidos.
- Neles, a resolução dos endereços (vinculação a posições fixas) é postergada.
- Endereços não resolvidos (relocáveis) são endereços relativos à posição de memória a partir da qual o programa irá ser alocado.
- Isto viabiliza a construção de bibliotecas relocáveis de programas e rotinas.
- A programação relocável propicia o desenvolvimento de programas em módulos separados relativamente independentes.
- Introduzindo-se o conceito de **endereçamento simbólico**, é possível, através da mútua referência dos módulos através de rótulos simbólicos, compô-los de acordo com a necessidade, obtendo assim os programas desejados.

Pseudo Instruções

- Como sabemos, **pseudo-instruções** são linhas, na linguagem simbólica do programa fonte, que têm aparência muito semelhante à daquelas que representam instruções de máquina, porém em vez de instruções, fornecem metadados, ou seja, orientações para o correto funcionamento do montador.
- M**ontadores absolutos** costumam permitir ao programador fornecer essas informações ao montador através de **pseudo-instruções** como as seguintes. Naturalmente, os mnemônicos costumam variar de um montador para outro.
  - **ORG** – (define nova origem para o código a ser montado em seguida)
  - **BLOC** – (reserva na memória um vetor a ser usado como área de trabalho)
  - **DB, DW, DA** – (preenche uma ou mais posições da área de código ou de dados com uma constante ou um endereço (absoluto ou relocável)
  - **EQU** – (define um novo símbolo como sinônimo de outro)
  - **END** – (demarca o final físico do programa fonte)
- **Montadores relocáveis** em geral oferecem, além destas, um repertório mais variado de pseudo-instruções, direcionadas especificamente para os aspectos da programação simbólica relocável.

Pseudo-instruções p/ montador relocável

- As diferenças entre textos simbólicos absolutos e relocáveis de linguagem simbólica reside no conjunto de pseudo-instruções disponíveis e sua interpretação.
- Pseudos encontradas na maior parte dos montadores relocáveis são as seguintes:
  - **ENTRY** – define pontos de acesso (*entry-points*) a um módulo
  - **EXTERNAL** – indica *entry-points* de outros módulos, referenciados no texto
  - **NAME** – define o nome do módulo corrente.
  - **ORG** – define nova origem do código, usualmente relocável.
  - **END** – indica o final físico do módulo, e o endereço de execução (prog.principal)
  - **EQU** – define sinônimos, atribui nome a endereços (expressões).
- As pseudos seguintes permitem processamento em tempo de montagem, pouco frequentes na maior parte dos casos.
  - **SET** – dá nome e altera o valora de variáveis em tempo de montagem.
  - **AIF** – efetua desvio condicional testando variáveis em tempo de montagem.
  - **ANOP** – define rótulos, específicos para desvios em tempo de montagem.
  - **AGO** – executa desvio incondicional de montagem para diante.
  - **AGOB** – executa desvio incondicional de montagem para trás.
  - **OPDEF** – redefine mnemônicos.
- Conforme o montador, essas pseudos podem ser usadas em programação absoluta.
- O montador converte linguagem simbólica relocável em código-objeto relocável.

Tabelas para pseudo instruções

- Normalmente não se costuma ter uma tabela exclusiva de **pseudo instruções (ou *pseudos*)**, pois, devido à similaridade física e de tratamento por parte do montador, os mnemônicos das pseudo instruções **compartilham a tabela de mnemônicos** que já conhecemos, que contém as representações simbólicas das instruções de máquina.
- No caso particular das **pseudos**, é conveniente que haja na tabela, em alguma das colunas de **atributos** associados ao mnemônico, uma informação de que se trata de uma pseudo instrução, e outra, que indique a qual das pseudo-instruções do repertório disponível esta se refere.

Constantes

- Analogamente, **constantes** que são utilizadas pelo programa que está sendo montado necessitam ser repetidamente processadas pelo montador, ou então, tabeladas para uma consulta mais eficiente durante o trabalho de montagem.
- Assim, as **constantes** podem também **partilhar**, com os demais nomes simbólicos utilizados no programa, **espaço na tabela de símbolos**, em cujas colunas devem neste caso ser registrados os **atributos associados** à constante em questão: tipo, valor, endereço na memória, etc.

CÓDIGO OBJETO RELOCÁVEL

Formato do código-objeto relocável

- Entre os **ligadores** e os **montadores** de linguagens simbólicas **relocáveis**, deve haver uma total compatibilidade de formatos, e para isso deve-se adotar um **formato único para os códigos-objeto relocáveis**, pois estes constituem o veículo usado tanto para a representação da saída dos montadores de linguagem simbólica relocável, como para a entrada dos ligadores e dos desmontadores relocáveis.
- Os slides a seguir esquematizam o formato aqui adotado para o código-objeto relocável.

Informações estabelecidas para os formatos

Uma FITA-OBJETO RELOCÁVEL típica é composta pelos 5 tipos acima de blocos de informações, contendo: identificação, pontos de acesso, símbolos externos, sequência de dados e finalização, cujos formatos são detalhados adiante.

Estrutura geral de um BLOCO DE INFORMAÇÕES de um programa-objeto relocável. As informações do bloco variam caso a caso, conforme o tipo (um dos cinco acima) do bloco a que pertencem.

Blocos de identificação e de *entry points*

BLOCO DE IDENTIFICAÇÃO - Conforme o sistema, certas informações das áreas de programas, dados, apontadores ou pilhas podem ser omitidas. Eventualmente, alguns sistemas incluem alguma área adicional, como, por exemplo, a de *common*. O tipo do módulo indica tratar-se de programa principal, sub-rotina, *overlay* etc.

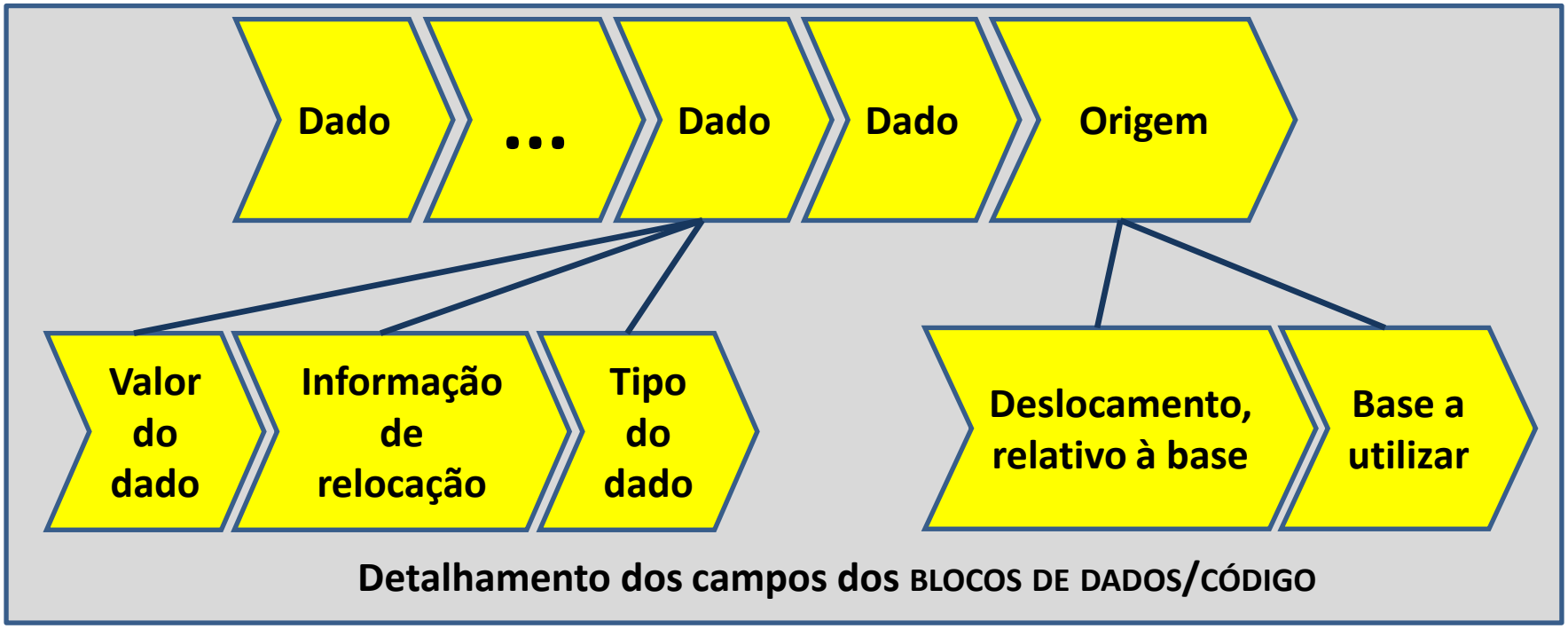
Um BLOCO DE PONTOS DE ACESSO é formado, em seu campo de informações do bloco, por uma sequência de estruturas de informações, uma para cada ponto de acesso, e cada qual tendo o formato esboçado na figura: o nome, o tipo do ponto de acesso (programa principal, sub-rotina, dado etc.) e seu endereço relativo (indicação da base de relocação a utilizar e deslocamento).

Formato para passo único e Bloco de *Externals*

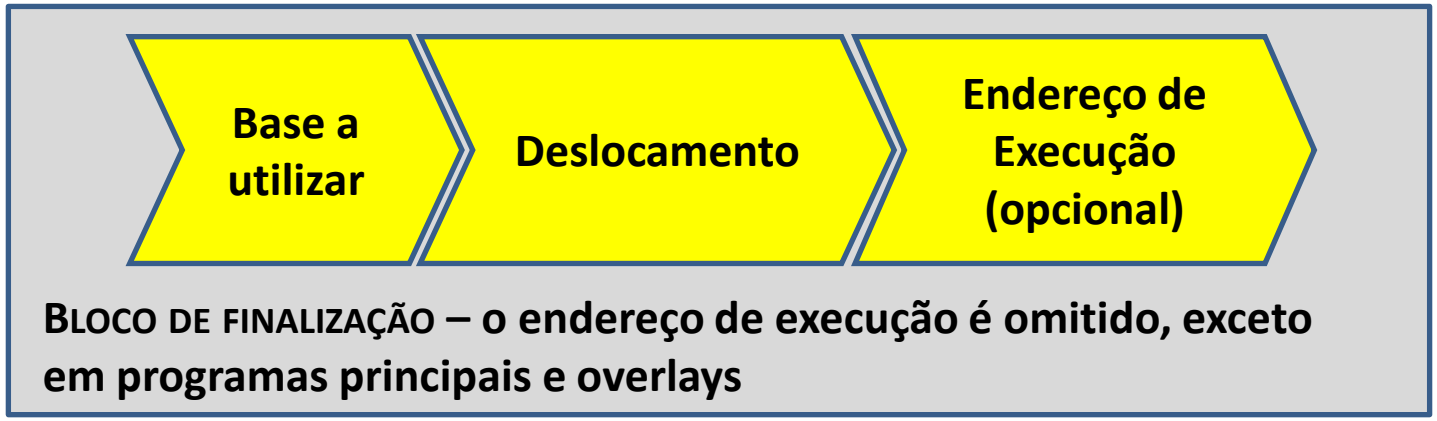
FORMATO PARA PASSO ÚNICO - Observar a inversão na sequência dos blocos em relação ao esquema em dois passos, para facilitar a geração. Notar a separação física inserida propositalmente entre o último bloco de dados e o restante das informações da fita, para facilitar o manuseio desta na época da ligação e relocação.

Alguns montadores registram neste BLOCO DE SÍMBOLOS EXTERNOS apenas símbolos externos efetivamente referenciados no programa, e não todos os declarados nas pseudo-instruções de definição de símbolos externos. Outros geram um bloco separado para cada símbolo.

Blocos de dados/código e de finalização



Detalhamento dos campos dos BLOCOS DE DADOS/CÓDIGO



BLOCO DE FINALIZAÇÃO – o endereço de execução é omitido, exceto em programas principais e overlays

ESTRUTURAS DE DADOS

Principais Estruturas de Dados

- **Tabela de símbolos** (símbolo - endereço - tipo - 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

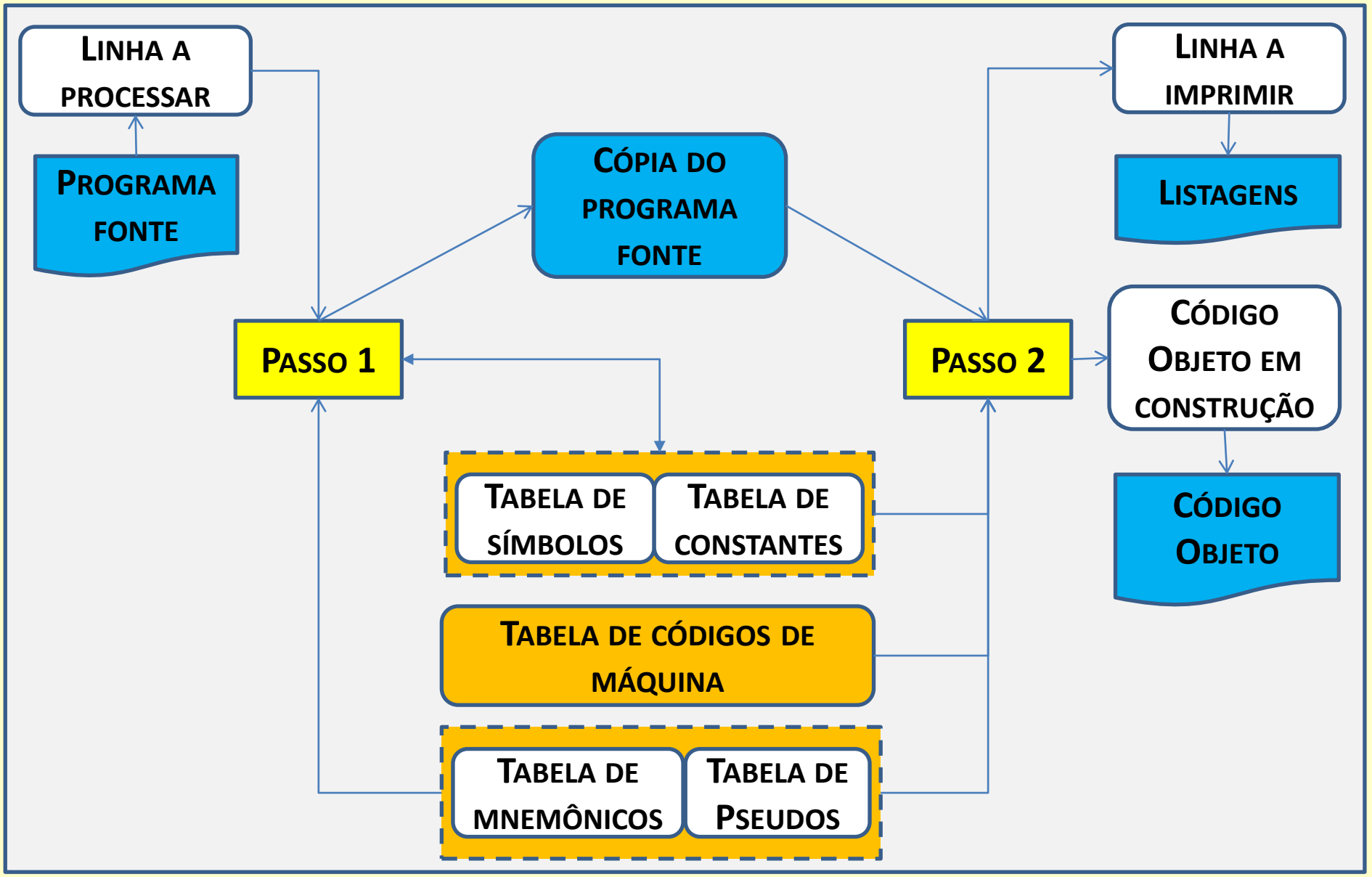


Tabela de Símbolos

- O montador constrói a tabela de símbolos para que o programador possa referenciar por **nome** as **posições de memória** em seus programas em linguagem simbólica
- O montador se incumbe de associar cada **nome simbólico** ao correspondente **endereço (absoluto ou relocável)** na memória, incluindo um tag que indica o tipo de endereço.
- Quando a localização física das posições ocupadas pelo código é conhecido, **endereços absolutos** são registrados na tabela de símbolos, associados ao endereço simbólico.
- Para endereçamento **relativo**, os endereços associados aos diversos rótulos (endereços simbólicos) designam localizações (**distâncias**) relativas à posição alocada na memória à primeira posição da área por ele ocupada, e permanecem relativos até o momento em que o endereço de alocação física dessa área se tornar conhecido.

Estrutura da Tabela de Símbolos para Montadores de Linguagens Simbólicas Relocáveis

- A tabela de símbolos deve ser estendida para conter, além das informações já estudadas para os montadores das linguagens simbólicas absolutas, a indicação da base de relocação associada a cada um dos endereços referenciados.
- A tabela de símbolos deve ainda memorizar o tipo referente a cada símbolo utilizado, indicando tratar-se de um ponto de acesso (público, *entry-point*), se é um símbolo externo (*external*- referência a um símbolo público declarado em outro programa), se é um endereço simbólico local, utilizado apenas internamente a este programa, etc.
- A tabela a seguir ilustra o aspecto dessa estrutura de dados, apresentando o conteúdo da tabela de símbolos gerada para o programa do exemplo anterior.

Coleta de Informação sobre os Símbolos

- As tabelas de símbolos são **criadas** durante a execução do **primeiro passo** do montador, e guardam, sobre os rótulos referenciados no programa, informações a serem usadas no segundo passo, tais como:
  - **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 organização física** para a tabela de símbolos memorizar a coleção de pares conceituais do tipo ***(símbolo-atributos)***: vetores de registros, tabelas bidimensionais, listas ligadas etc.

Aspecto típico de Tabelas de Símbolos

IDENTIFICADOR SIMBÓLICO	VALOR	ENDEREÇO INICIAL	TAMANHO EM BYTES	INFORMAÇÃO DE RELOCAÇÃO
ABCD	-	/0030	1	ABSOLUTO
XYZ	-	/0123	50	ABSOLUTO
A1	-	INDEFINIDO	100	RELOCÁVEL
B	-	/0000	20	ABSOLUTO
-	150	/05B2	1	ABSOLUTO
-	/0123	/05B3	2	ABSOLUTO

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
  - Indicação dos **operandos exigidos** pela instrução
  - Indicação dos **tipos de operando** permitidos
  - Valor numérico **binário** associado a seu **código**
  - **Número de bytes ocupados** pela instrução
  - Classe da instrução – **número e tipo de operandos**

Aspecto típico de Tabelas de Mnemônicos

ESTRUTURA DO CÓDIGO DE MÁQUINA	TIPO	MNEMÔNICO	NOME	VALOR	OPERANDO
0000xxxxxxxxxxx	inst.ref.mem.	JMP	JUMP	-	ENDEREÇO
0001xxxxxxxxxxx	inst.ref.mem.	LDA	LOAD	-	ENDEREÇO
0010xxxxxxxxxxx	inst.ref.mem.	STA	STORE	-	ENDEREÇO
	...	...			
XXXXXXXX	constante-8	K	BYTE	OPERANDO	CONST. BYTE
XXXXXXXXXXXXXXXX	Endereço	ADDR	POINTER	OPERANDO	CONST. ADDR
	...	...			

INSTRUÇÕES

Tratamento de referências à memória

- Instruções simbólicas cujos operandos simbolizam posições de memória têm conotações variadas nas linguagens simbólicas relocáveis:
  - Referindo-se a uma posição física específica conhecida, denominam-se referências **absolutas**, podendo ser **simbólicas** (referindo-se a um símbolo que represente uma referência absoluta) ou então **numéricas** (referindo-se diretamente a um endereço conhecido)
  - Referindo-se a uma posição relativa a uma base de relocação cujo valor é desconhecido na época da codificação do programa.
  - Conforme a arquitetura da máquina e do montador, pode haver mais de uma base disponível. Referências **relocáveis** são referências relativas a alguma delas

Tratamento das demais instruções

- Instruções que não referenciam a memória sempre geram código absoluto, pois independem dos tipos de endereçamento tratados pelo montador.
- Por isso, o montador não sofre alterações, em sua forma de tratamento, em relação à adotada na versão voltada à linguagem simbólica absoluta.

Tratamento das Instruções

- O tratamento das instruções de máquina pelo montador consiste essencialmente em:
  - Registrar, quando presente, o rótulo na Tabela de Símbolos, definindo-o com o endereço contido na ocasião na variável Contador de Instruções
  - Consultando a Tabela de Mnemônicos, obter
    - o padrão binário associado ao mnemônico da instrução,
    - o número de bytes ocupado por seu código,
    - o tipo de operando que exige, e
    - o tratamento necessário para a sua montagem
  - Aplicar a rotina de tratamento correspondente para construir e gerar no meio de saída o código objeto associado à instrução

Exemplo de montagem de uma instrução

- Supondo que **CI** = /0126 seja o endereço corrente de geração do código objeto, e que **DADO** seja uma referência simbólica ao endereço /0100:
  - Linha do programa-fonte, contendo uma instrução:

```
LOOP    LD    DADO        ;  esta instrução copia DADO para o acumulador
```

  - O rótulo **LOOP** é inserido na Tabela de Símbolos, e associado ao endereço /0126 (valor corrente de **CI**)
  - O mnemônico **LD** é identificado na Tabela de Mnemônicos como sendo de instrução de máquina, com código de operação /8
  - O nº de bytes desta instrução é 2: atualiza-se **CI** para /0128
  - **DADO**=/0100 é uma referência absoluta à memória
  - Portanto, o código objeto binário associado será /8100

```
0126   81 00    LOOP   LD  DADO    ;  esta instrução copia DADO para o acumulador
```

Exemplo da estrutura do bloco de saída

- O trecho abaixo representa um fragmento de um programa MVN, com a finalidade de ilustrar algumas situações típicas que costumam ser encontradas nos programas em notação simbólica absoluta, e que devem ser tratadas pelo montador.

Endereço	Código	Rótulo	Mnemônico	Operando	Comentários
			@	/0100	
0100	00	DADO	K	0	
0101	00	AUX	K	0	
0102	81 01	INIC	LD	AUX	; primeira instrução executável do programa
⋮	⋮		⋮		
0124	91 01		MM	AUX	; salva acumulador em AUX
0126	81 00	LOOP	LD	DADO	; esta instrução copia DADO para o acumulador
⋮	⋮		⋮		
01AF	01 26		JP	LOOP	; esta instrução fecha o loop
0102			#	INIC	; indica INIC como endereço de partida do prog.

Observações sobre o bloco de saída

- Como já deve ter sido notado, é **incremental** a **geração do código objeto** à imagem da memória, alguns bytes para cada linha do programa fonte.
- Para que o programa objeto seja **compatível com o loader**, seu formato não é à imagem da memória, mas, blocado em partes referentes a áreas relativamente pequenas de código, e incluindo um byte de checksum, para reduzir a probabilidade de leitura incorreta dos dados.
- Assim sendo, e levando em conta que blocos muito pequenos acarretam o aumento desnecessário do tamanho do código objeto, opta-se por **acumular na memória do montador** uma série de códigos vizinhos à imagem da memória até que estejam acumulados bytes suficientes para formar um bloco de tamanho aceitável. Nessa ocasião, o bloco é **transferido para o arquivo de saída**, e um novo bloco, vazio, é criado e passa a ser preenchido pelo montador, a partir de uma nova origem.
- Durante a montagem, o aparecimento de instruções ou de pseudo instruções que **modificam a origem do código gerado** devem, portanto, **iniciar um novo bloco a partir da nova origem**, sendo portanto necessário **forçar o encerramento de um eventual bloco** parcialmente formado na memória do montador, para que o código já gerado, nele contido, não seja sobrescrito, e portanto perdido.

PSEUDO-INSTRUÇÕES

Tratamento de pseudo-instruções

- Somente serão consideradas aqui as pseudos do slide anterior pertencentes ao primeiro grupo.
  - **ENTRY** – Se passo=1. gerar bloco de pontos de acesso (*entry-points*) a um módulo.
  - **EXTERNAL** – Se passo=1, gerar bloco dos *externals* referenciados no texto.
  - **NAME** –Se passo=1, gerar bloco de nome do módulo corrente.
  - **ORG** – define nova origem do código, absoluta ou relocável conforme especificado no operando.
  - **END** – Se passo = 1, fazer passo ← 2. Se não, gerar bloco de finalização do programa e terminar a montagem.
  - **EQU** – Se passo = 1, atualizar a tabela de equivalências.
  - **DB,DW** – Se passo =2, gerar o código objeto associado.

Tratamento de Pseudo Instruções

- Em **montadores absolutos**, costumam ser encontradas as **pseudo-instruções** seguintes:
  - **ORG** – (nova origem) modifica o contador de instruções conforme o valor do operando **(@)**
  - **BLOC** – (reserva de área) modifica contador de instruções para contabilizar a área reservada **($)**
  - **DB, DW, DA** – (preenche memória com constante) se passo igual 2, gerar código objeto **(K)**
  - **EQU** – (define sinônimos) se passo igual a 1, atualizar a tabela de equivalências
  - **END** – (indica final físico do programa fonte) se passo for igual a 1, fazer passo igual a 2. Se não, encerrar os trabalhos do montador **(#)**
- Voltar à leitura de nova linha.

ORG (origin)

- Determina nova origem para o código a ser gerado em seguida pelo montador:
  - Em montadores absolutos, o operando deve ser obrigatoriamente absoluto.
  - Em montadores relocáveis, pode ser relocável, absoluto, simbólico, relativo
  - Tratamento:
    Se passo for igual a 2 e houver no bloco de saída do código objeto algum código ainda não gerado em meio externo, gerar o bloco devidamente, e esvaziá-lo;
    Modificar o contador de instruções do montador, de acordo com o valor do operando que estiver sendo especificado.

BLOC (define memory block)

- Esta pseudo-instrução determina a reserva de uma área de memória de comprimento estabelecido, sem preenchimento de dados, disponibilizando-a para uso pelo programa objeto.
- O operando deve ter um valor numérico inteiro não negativo, pois refere-se ao número de palavras de memória a ser reservado.
- Tratamento:
  Equivalente à definição de nova origem no endereço obtido adicionando-se ao contador de instruções o tamanho da área que estiver sendo reservada.
  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 for igual a 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 uma unidade.

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 for igual a 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 duas unidades.

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 for igual a 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 duas unidades.

EQU (define equivalence)

- Esta pseudo-instrução permite determinar a equivalência (sinônimos) entre novos nomes e endereços associados a um ou mais símbolos definidos no programa-fonte.
- Seu operando precisa ser obrigatoriamente uma expressão simbólica que represente algum endereço de memória, de qualquer tipo.
- Tratamento: se passo for igual a 1, atualizar a tabela de equivalências

END (end mark for source program)

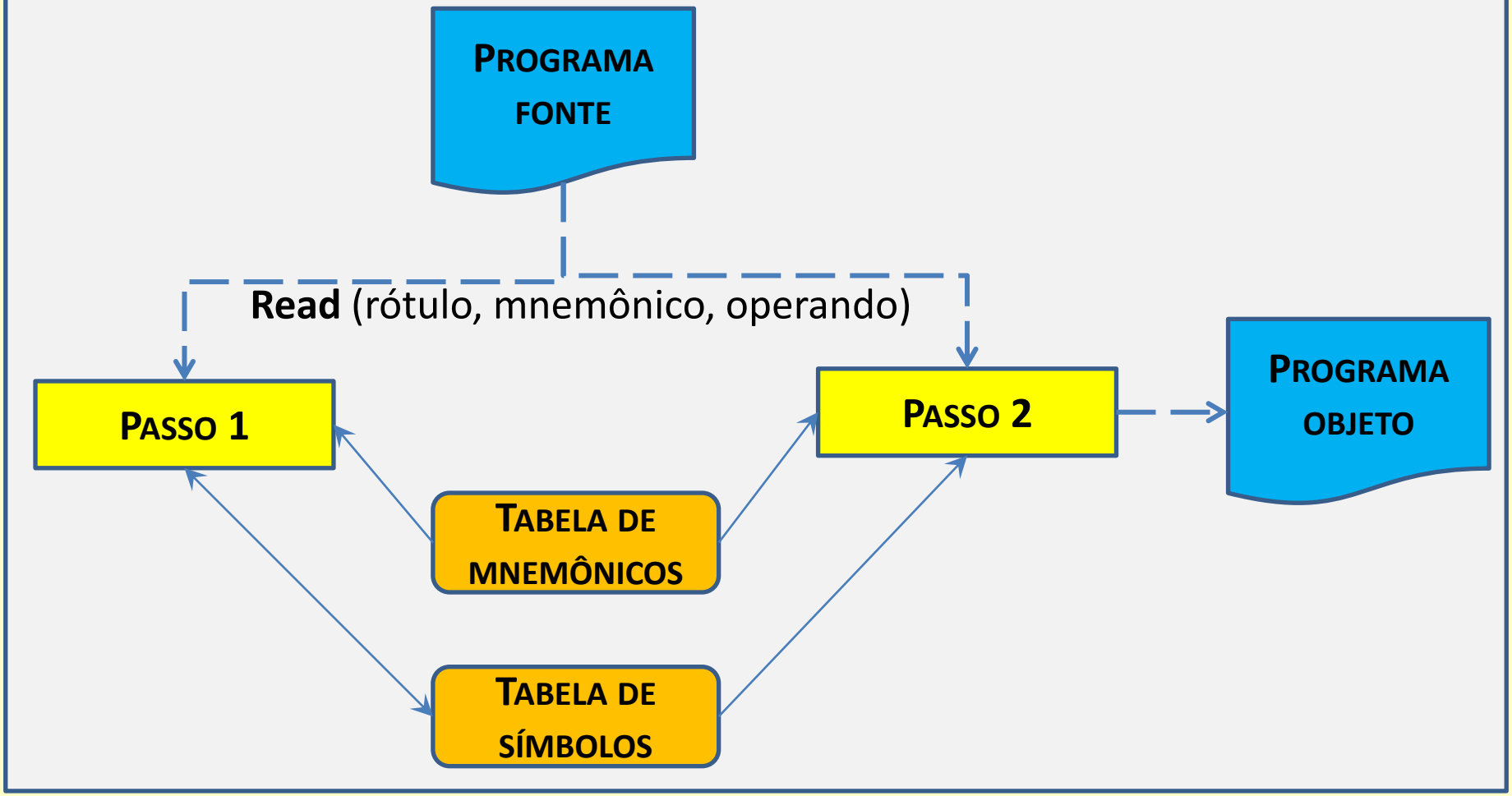
- Esta pseudo-instrução permite ao programador informar ao montador que foi atingido o final físico do programa-fonte.
- Seu operando deve ser um rótulo definido no programa, e deve referir-se a um endereço de memória, relativo a um rótulo do programa, que fornece a informação do endereço a partir do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

ARQUITETURA E IMPLEMENTAÇÃO

Montagem em dois e em um passo

- 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-fonte que foi lido e das tabelas construídas no primeiro passo.
- Os montadores que seguem o segundo esquema são denominados ***montadores de um passo***, e realizam **apenas uma leitura** do programa fonte, mas exigem para isso a manutenção de uma ***lista de pendências***;

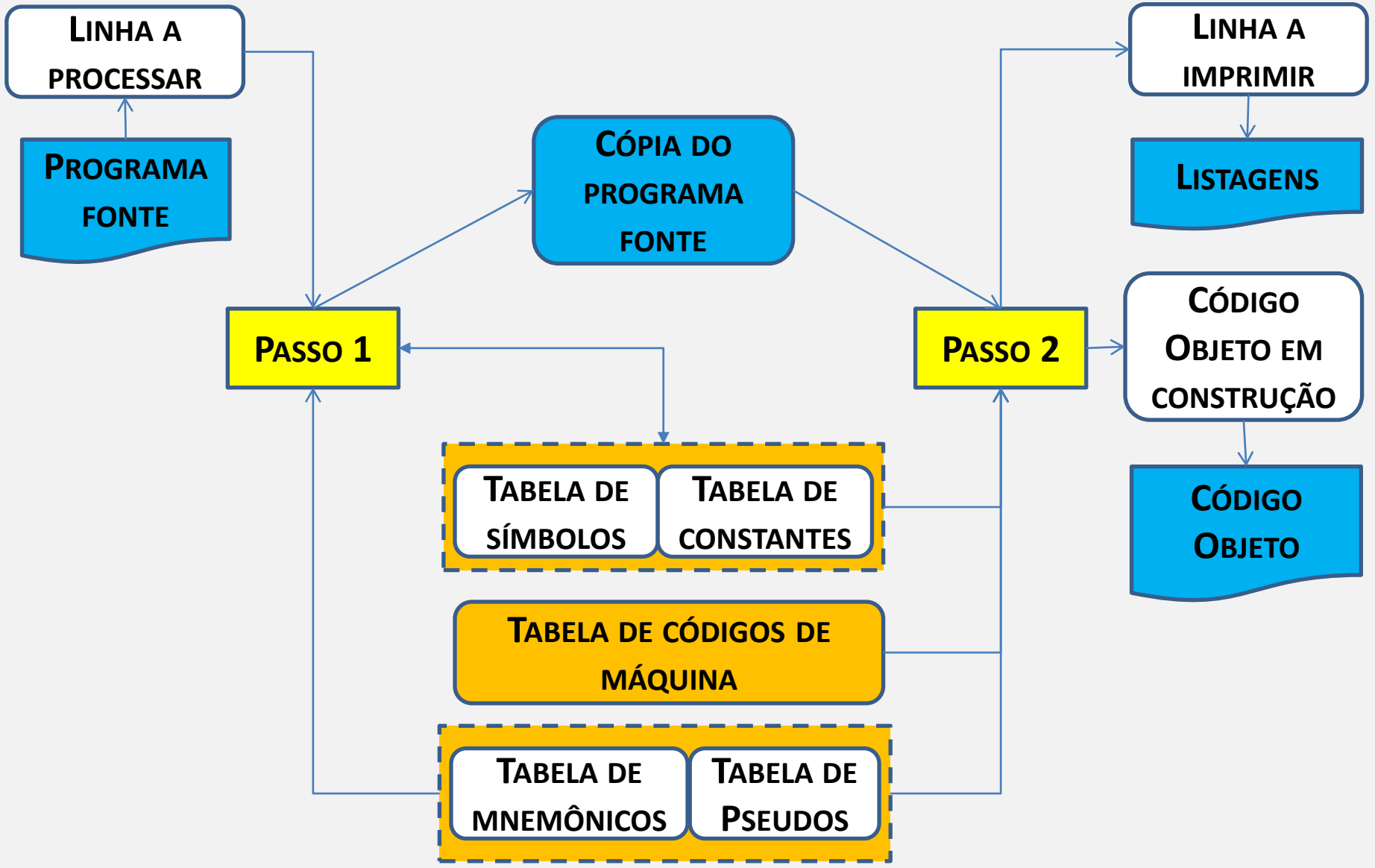
Organização de um montador simples, em dois passos



Principais Estruturas de Dados

- **Tabela de símbolos**
  (símbolo - endereço - definido - referenciado)
- **Extensão** da tabela de símbolos, para
  - considerar **endereços absolutos e relocáveis,** e
  - para a 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 - deslocamento - *link*)
- **Área de saída**
  (bloco contendo o código objeto (parcial) gerado)

Áreas de Dados usadas pelo montador



Estrutura da Tabela de Símbolos para Montadores de Linguagens Simbólicas Relocáveis

- A tabela de símbolos deve ser estendida para conter, além das informações já estudadas para os montadores das linguagens simbólicas absolutas, a indicação da base de relocação associada a cada um dos endereços referenciados.
- A tabela de símbolos deve ainda memorizar o tipo referente a cada símbolo utilizado, indicando tratar-se de um ponto de acesso (público, *entry-point*), se é um símbolo externo (*external*- referência a um símbolo público declarado em outro programa), se é um endereço simbólico local, utilizado apenas internamente a este programa, etc.
- A tabela a seguir ilustra o aspecto dessa estrutura de dados, apresentando o conteúdo da tabela de símbolos gerada para o programa do exemplo anterior.

Tabela de símbolos para um programa relocável

NOME	BASE	ENDEREÇO	D/I	TIPO
MODULO	1	1000	D	nome
SOMA	2		D	externo
VAR	2	0200	D	público
MAXIMO	2	0212	D	público
LOOP	1	1009	D	interno
INICIO	2	0210	D	interno
FIM	2	0211	D	interno

Tabela de símbolos para o programa-exemplo

PROJETO DO MONTADOR

Implementação de montadores de dois passos para linguagem simbólica relocável

- Os montadores de dois passos são os mais conhecidos e mais difundidos.
- Tais programas são os mais simples, exigindo menores esforços para sua construção e implantação.
- A seguir, especifica-se o projeto de um montador de dois passos para linguagem simbólica, compatível com ambas as formas de endereçamento (absoluta e relocável)

Montadores para programas relocáveis

- Montadores são os programas de sistema responsáveis pela geração correta do programa-objeto, nos moldes apresentados anteriormente.
- Devem gerar ainda listagens contendo informações sobre referências externas e sobre o tipo de relocação associado a cada endereço de memória referenciado no programa.
- Essas listagens devem incluir uma lista de todos os pontos de acesso ao programa, devidamente acompanhados dos correspondentes endereços.

Modificações no passo 1 do montador

- O passo 1 do montador relocável é responsável pela geração da tabela de símbolos para programas relocáveis.
- Como foi visto, diversas informações adicionais são necessárias, que se referem a elementos ausentes nos programas absolutos.
- Por isso, as modificações correspondentes devem ser introduzidas no algoritmo do montador absoluto para que o programa-objeto possa ser corretamente gerado.
- Tais modificações referem-se, basicamente, à alteração da funcionalidade de algumas pseudo-instruções (ORG, END) e também a introdução de novas pseudo-instruções (NAME, ENTRY, EXTERNAL, CSEG, DSEG, ASEG), destinadas a controlar ponteiros adicionais, referentes a cada uma das áreas do código relocável.
- Além disso, o passo 1 do montador é responsável pela geração do cabeçalho do programa-objeto.

Montador Relocável, passo 1

- O primeiro passo do montador destina-se especialmente às tarefas seguintes:
  - Construção da **tabela de símbolos**
  - Consulta à **tabela de mnemônicos**
  - Construção da **tabela de equivalências**
  - Cálculo dos **endereços** nos operandos das instruções
  - **Teste de consistência** da tabela de símbolos final
  - Detecção e diagnóstico de **erros**
  - Geração de **mapas de memória**
  - Geração de tabelas de referências cruzadas

Implementação de montadores de dois passos

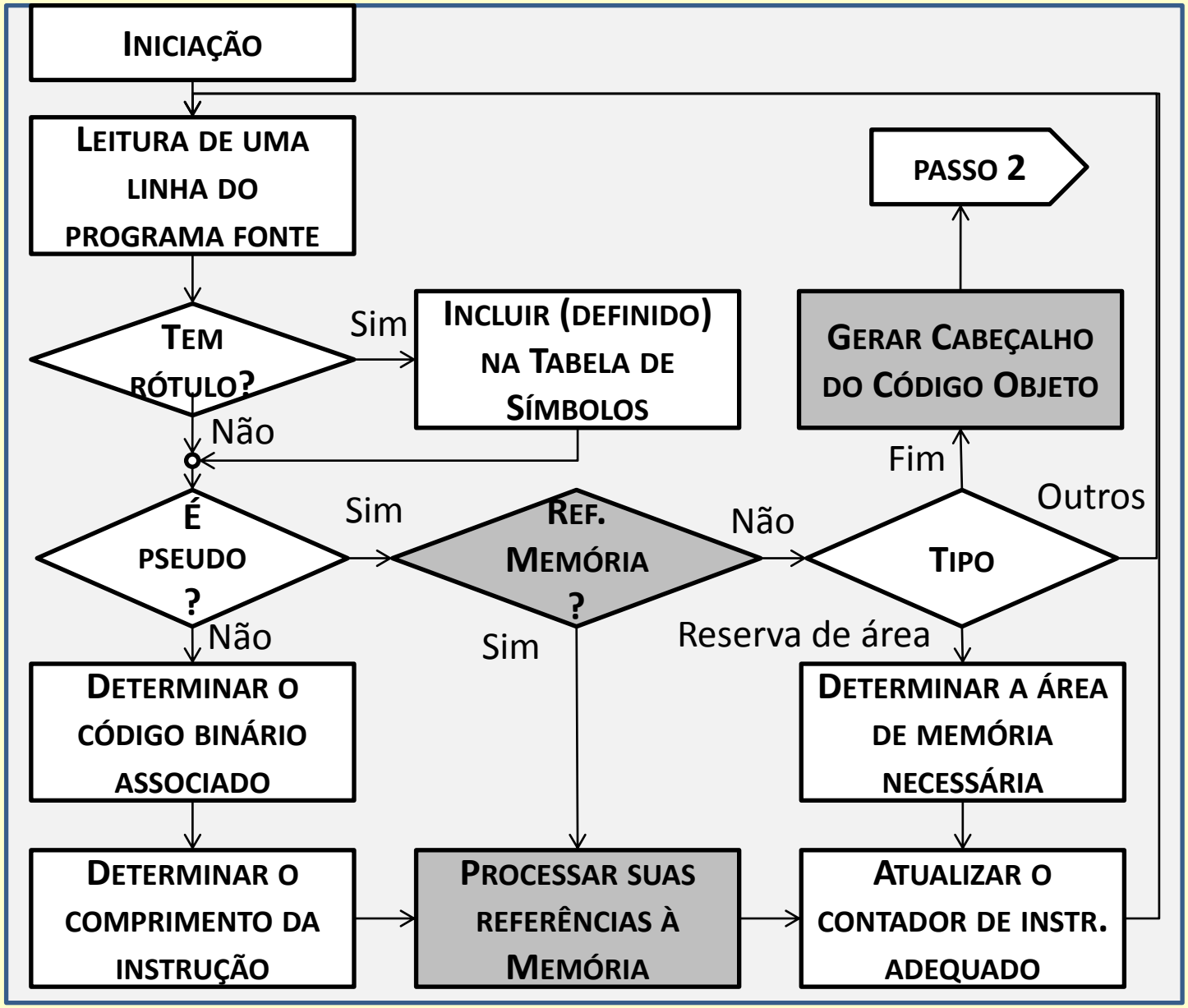
- É a arquitetura de montador mais difundida
- De um lado, tem necessidade de uma ***área menor de armazenamento*** (em cada passo)
- Porém, exige a ***releitura*** completa do programa fonte
- Estruturalmente ***mais simples***, propicia uma implementação menos artificiosa
  - **Primeiro passo**:
    - Leitura do texto-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**
    - **Geração** do programa-objeto **em formato carregável**

PASSO 1 do Montador

Atividades principais:

- 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 da tabela de referências cruzadas

Lógica resumida do passo 1 do montador



Tratamento de referências à memória

- Instruções simbólicas cujos operandos simbolizam posições de memória têm conotações variadas nas linguagens simbólicas relocáveis:
  - Referindo-se a uma posição física específica conhecida, denominam-se referências **absolutas**, podendo ser **simbólicas** (referindo-se a um símbolo que represente uma referência absoluta) ou então **numéricas** (referindo-se diretamente a um endereço conhecido)
  - Referindo-se a uma posição relativa a uma base de relocação cujo valor é desconhecido na época da codificação do programa.
  - Conforme a arquitetura da máquina e do montador, pode haver mais de uma base disponível. Referências **relocáveis** são referências relativas a alguma delas

Tratamento das demais instruções

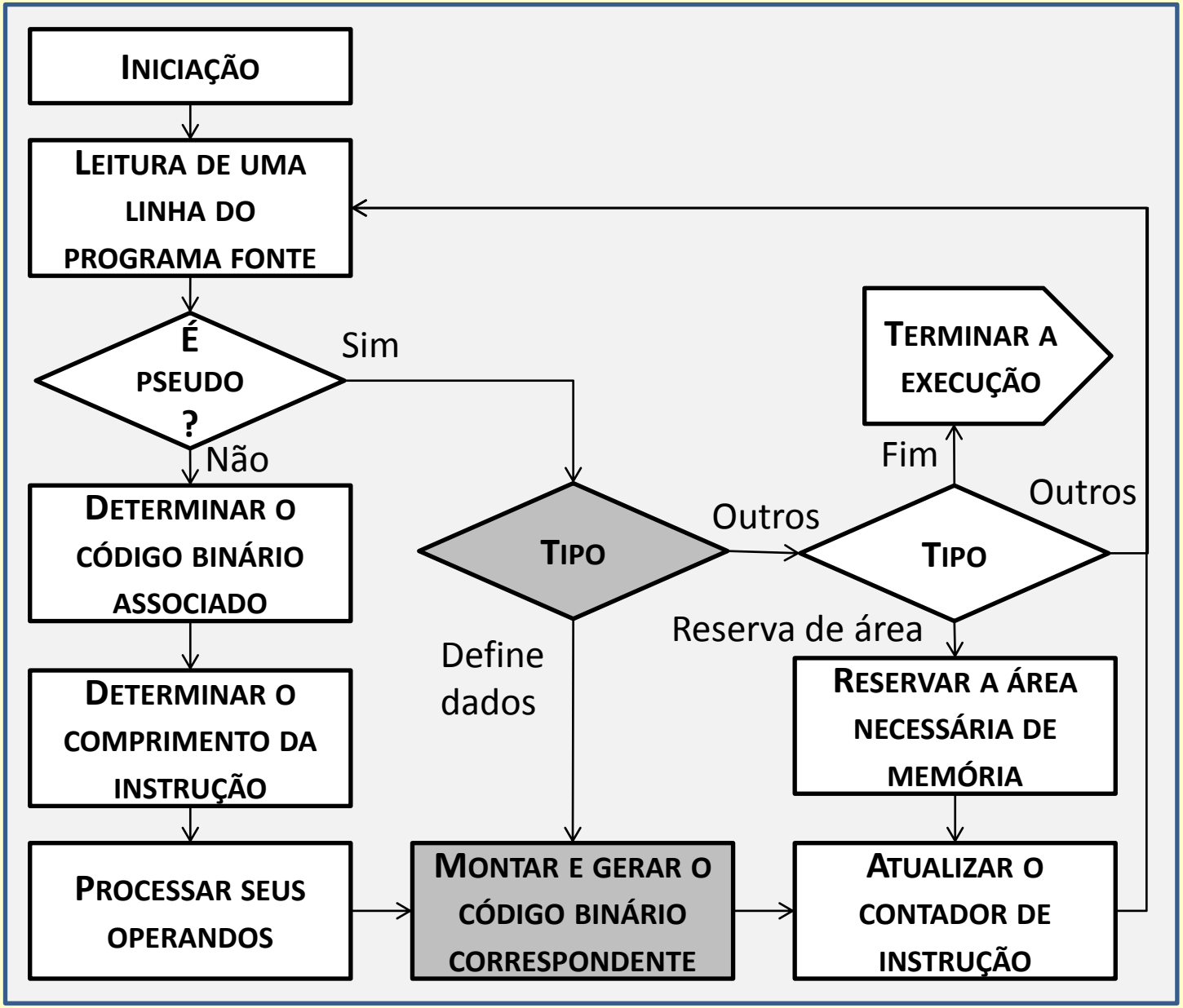
- Instruções que não referenciam a memória sempre geram código absoluto, pois independem dos tipos de endereçamento tratados pelo montador.
- Por isso, o montador não sofre alterações, em sua forma de tratamento, em relação à adotada na versão voltada à linguagem simbólica absoluta.

PASSO 2 do Montador

Atividades principais

- Consulta à tabela de códigos
- Montagem do código objeto
- Avaliação das expressões dos operandos
- Geração da listagem formatada
- Geração do programa-objeto
- Tratamento de referências relocáveis, quando necessário

Lógica resumida do passo 2 do montador



Modificações no passo 2 do montador

- O passo 2 é responsável pela geração dos blocos de dados do código-objeto, no formato especificado anteriormente.
- Para tanto, a tabela de mnemônicos deve informar se o operando de cada instrução faz referência ou não à memória.
- De posse dessa informação, o montador é capaz de decidir, consultando a Tabela de Símbolos, qual deve ser a base de relocação à qual o endereço em questão se refere, e incluir adequadamente essa informação na parte do programa-objeto associada àquele particular endereço.
- As pseudos ASEG, CSEG, DSEG e ORG são tratadas pelo passo 2 de forma análoga à do passo 1.
- Quando o passo 2 do montador relocável encontra qualquer uma destas pseudo-instruções, o bloco de saída do montador é finalizado e um novo bloco de programa-objeto é iniciado.
- As demais pseudo-instruções não sofrem modificações em seu tratamento no passo 2.

Montador Relocável, passo 2

- No segundo passo, o montador se dedica especialmente às seguintes atividades:
  - Consulta à **tabela de códigos**
  - **Montagem do código**-objeto
  - Avaliação das **expressões dos operandos**
  - Geração de **listagens** formatadas
  - Geração de **programas-objeto**

LÓGICA DO MONTADOR

Pseudocódigo do montador relocável

- Contador de instruções (C.I.) ← 0. Passo ← 1.
- **Leitura de uma linha**, ignorando comentários; se passo =2 então **listar a linha**;
- Se a linha tiver **rótulo**:
  - **Procurar** o rótulo na tabela dos símbolos.
  - Se já existe e foi definido, gerar **mensagem de erro** .
  - Se já existe Indefinido, **atribuir-lhe o endereço** do contador de instruções .
  - Se não existe, **inserir na tabela**, e **atribuir o endereço** do contador de instruções .
  - **Marcar como definido**.
  - **Atualizar** a tabela de **referências cruzadas**.
- Analisar o **mnemônico**:
  - **Procurar** na tabela de mnemônicos .
  - Se não achar, gerar **mensagem de erro**.
  - **Atualizar contador de instruções** conforme o tipo de mnemónico.
  - Se o mnemônico exigir **operando**:
    - analisar o operando
      - **Incluir eventuais símbolos novos** na tabela de símbolos .
      - Se não constaram na tabela, **marcar como indefinidos**.
      - **Atualizar** a tabela de **referências cruzadas**.
    - Avaliar operando (expressão)
    - Se passo = 2 e não for pseudo instrução, **Montar código objeto**.
  - Se for **pseudo** instrução, efetuar o tratamento (slides seguintes)
- Voltar à leitura de nova linha.

Tratamento das pseudo-instruções

- As linguagens simbólicas relocáveis têm pseudo-instruções que são utilizadas também em programas simbólicos absolutos.
- A menos das pseudo-instruções ORG e END, cuja atuação em programas relocáveis é mais ampla (pois acumulam outras funcionalidades), as demais são tratadas de forma similar, guardadas as peculiaridades dos tipos de endereçamento que devem acrescentar

Pseudo-instruções DW, DB, DA etc.

- Suando definem o preenchimento da memória com valores constantes, essas pseudo-instruções não sofrem modificações em seu tratamento.
- Se porém referenciarem endereços de memória, terão tratamento compatível com o aplicado às instruções de referência à memória, como foi apresentado anteriormente.

Tratamento da pseudo-instrução NAME

- Esta pseudo instrução associa um nome ao módulo.
- O nome, atribuído ao módulo pelo programador, deve ser registrado na tabela de símbolos, mas antes é preciso pesquisar a tabela para verificar se o símbolo em questão já foi definido anteriormente.
- Se o símbolo já tiver sido anteriormente inserido na tabela, e marcado como definido, trata-se de uma tentativa de redefinição do símbolo, e por isso, o montador deve gerar uma mensagem de erro.
- Caso o símbolo não esteja presente na tabela, nela deverá ser inserido, e marcado como definido.

Tratamento da pseudo-instrução ENTRY

- No primeiro passo do montador, esta pseudo-instrução é tratada atualizando-se na tabela de símbolos, para cada um dos símbolos encontrados em seu operando, os campos de tipo correspondentes .
- Se algum desses símbolos não estiver definido nesta ocasião, ele deve ser inserido na tabela de símbolos, e marcado como indefinido.

Tratamento da pseudo-instrução EXTERNAL

- Ao tratar a pseudo-instrução EXTERNAL, o montador deve marcar como EXTERNO, na tabela de símbolos, no campo referente ao tipo do símbolo, cada um dos símbolos do seu operando.
- Observe-se que, ao contrário do que ocorre com símbolos de outros tipos, não é condição de erro o fato de símbolos externos não serem definidos no escopo do programa corrente.
- Por essa razão, marcam-se como símbolos definidos todos aqueles que forem encontrados como operandos desta pseudo-instrução, já que, por terem sido declarados como externos, não poderão estar associados a qualquer endereço interno do programa.

Tratamento das pseudo-instruções de definição de área (CSEG, ASEG e DSEG)

- Estas pseudo-instruções causam a mudança da base de relocação em relação à qual são computados os endereços referenciados no código.
- Dessa forma, todas as referências à memória contidas no código subsequente passam a incrementar o ponteiro para essa nova base.
- Isso prossegue desta maneira até que uma próxima ocorrência de qualquer dessas pseudo-instruções efetue uma nova alteração.

Tratamento da pseudo-instrução ORG

- Esta pseudo-instrução é tratada da forma semelhante ao que foi feito no primeiro passo do montador absoluto.
- A única diferença é que, no caso de o operando ser um endereço relocável, o contador de instruções que recebe o valor que ele representa pode variar ao longo do programa, de acordo com o que for selecionado pelo programador através das pseudo-instruções de definição de área (ASEG, DSEG, CSEG).

Tratamento da pseudo-instrução END

- Se esta pseudo-instrução apresentar um parâmetro que indique o endereço de execução do módulo, tal endereço é registrado na tabela de símbolos, na posição correspondente ao nome do módulo.
- Se o montador não encontrar nenhum símbolo com atributo de nome, associa a ele um endereço de execução, que é o operando especificado na pseudo-instrução END, e por uniformidade, impõe artificialmente um nome para o módulo.
- Se a pseudo-instrução END for utilizada sem parâmetro, procede-se como no caso anterior, forçando, como operando, o endereço zero.

Consistência da Tabela de Símbolos e Geração do Cabeçalho do Programa-Objeto

- Quando o passo l termina o processamento das linhas do arquivo que contém o programa-fonte, deve ser verificada a consistência da Tabela de Símbolos.
- Para tanto, verifica-se, da mesma forma que é feito nos montadores absolutos, a situação de indefinição ou a super-definição de símbolos, ao final do primeiro passo do montador relocável.
- Verifica-se também se foi associado mais de um nome ao módulo.
- Após esse teste de consistência, cria-se o bloco de cabeçalho do código-objeto, a partir das informações previamente registradas na Tabela de Símbolos, e os valores finais assumidos pelos ponteiros associados a cada área.

FIM