PCS 3216
Sistemas de Programação

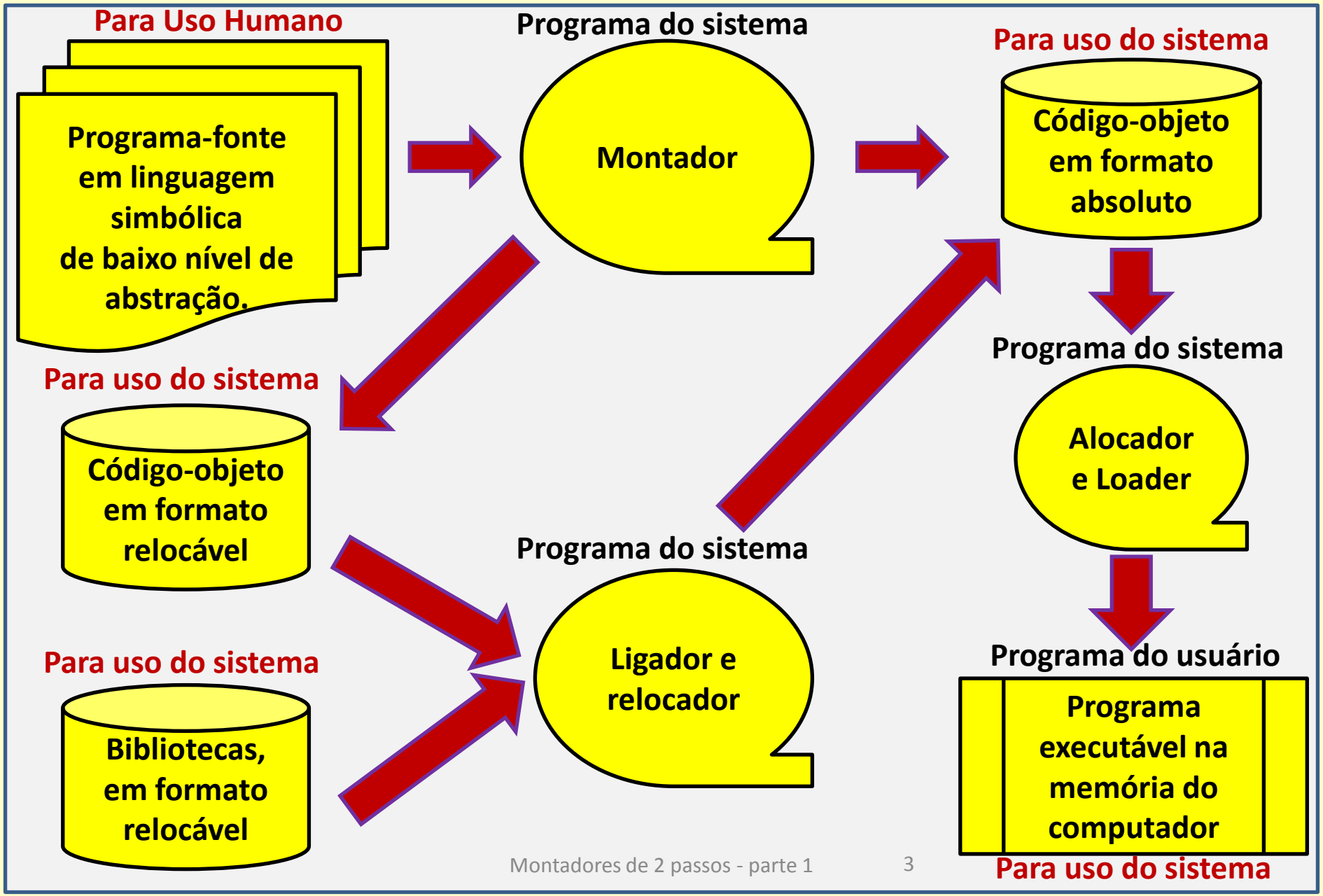
João José Neto

Aula 09 – Montadores de dois passos

Introdução

- Nesta aula:
  - Contextualiza-se de forma mais precisa o montador no âmbito dos sistemas de programação
  - Apresentam-se os principais termos técnicos da área
  - Comparam-se conceitual e estruturalmente os montadores de 2 e de 1 passo
  - Levanta-se a especificação básica de requisitos de um montador para a máquina virtual estudada na disciplina
  - Inicia-se a elaboração de um anteprojeto de montador de dois passos para a máquina virtual.

Preparação de código em linguagem de baixo nível



TIPOS MAIS FREQUENTES DE MONTADORES

Alguns Conceitos

- **Passo de montagem** – corresponde ao processamento que envolve uma leitura completa do programa fonte
- Montadores ***load and go*** – geram diretamente na memória o código objeto, para execução imediata
- ***Backtracking*** – técnica de gerar código objeto incompleto, cujas lacunas vão sendo preenchidas oportunamente
- Montadores de **dois passos** – são os que precisam ler o programa fonte duas vezes para gerar o código objeto
- Montadores de **um passo** – são aqueles que, para gerar o código objeto, precisam ler apenas uma vez o programa fonte
- Montadores que constroem em meio externo o **programa-objeto absoluto** devem gerá-lo no formato de um código binário que possa ser carregado pelo programa ***loader***

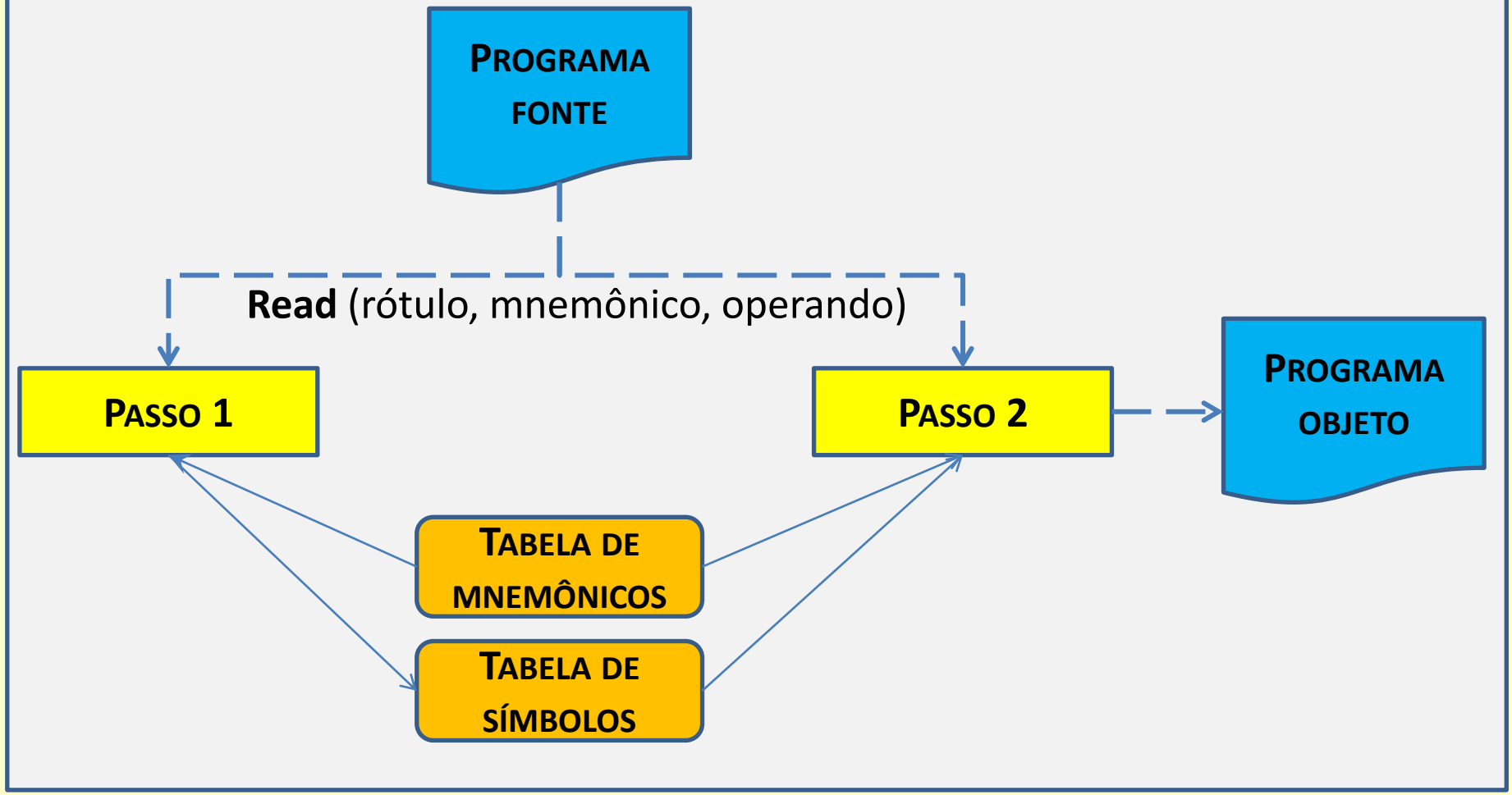
Resolução de endereços simbólicos

- Há duas técnicas clássicas para o problema da ***resolução*** (= conversão para valores numéricos) dos nomes que representam endereços simbólicos:
  - **Em dois passos** – a primeira técnica consiste em efetuar inicialmente a **coleta dos símbolos e o cálculo dos endereços a eles associados**, sem a preocupação em montar o código de máquina das instruções, pois isso é feito em uma segunda etapa.
  - **Em um passo** – a segunda consiste em, sempre que possível, efetuar simultaneamente essas duas tarefas, postergando somente a montagem de instruções que contenham ***referências à frente***, de modo que a montagem venha a ocorrer apenas quando se tornar possível a determinação exata do endereço associado ao símbolo em questão.

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



Implementação de montadores de dois passos

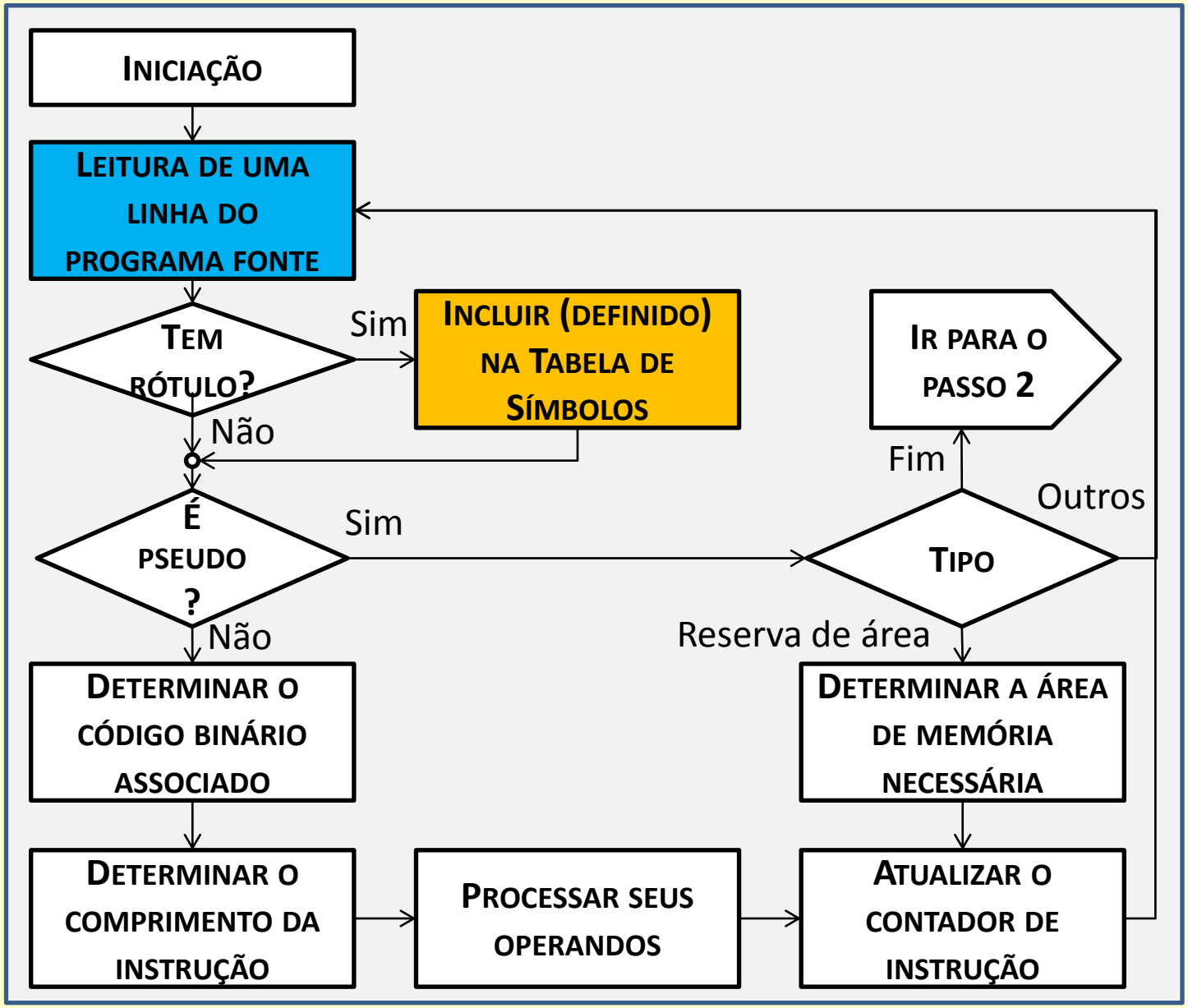
- É a arquitetura de montador mais difundida
- De um lado, tem necessidade de uma ***área menor de armazenamento*** (em cada passo)
- Porém, exige a ***releitura*** completa do programa fonte
- Estruturalmente ***mais simples***, propicia uma implementação menos artificiosa
  - **Primeiro passo**:
 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

Lógica resumida do passo 1 do montador

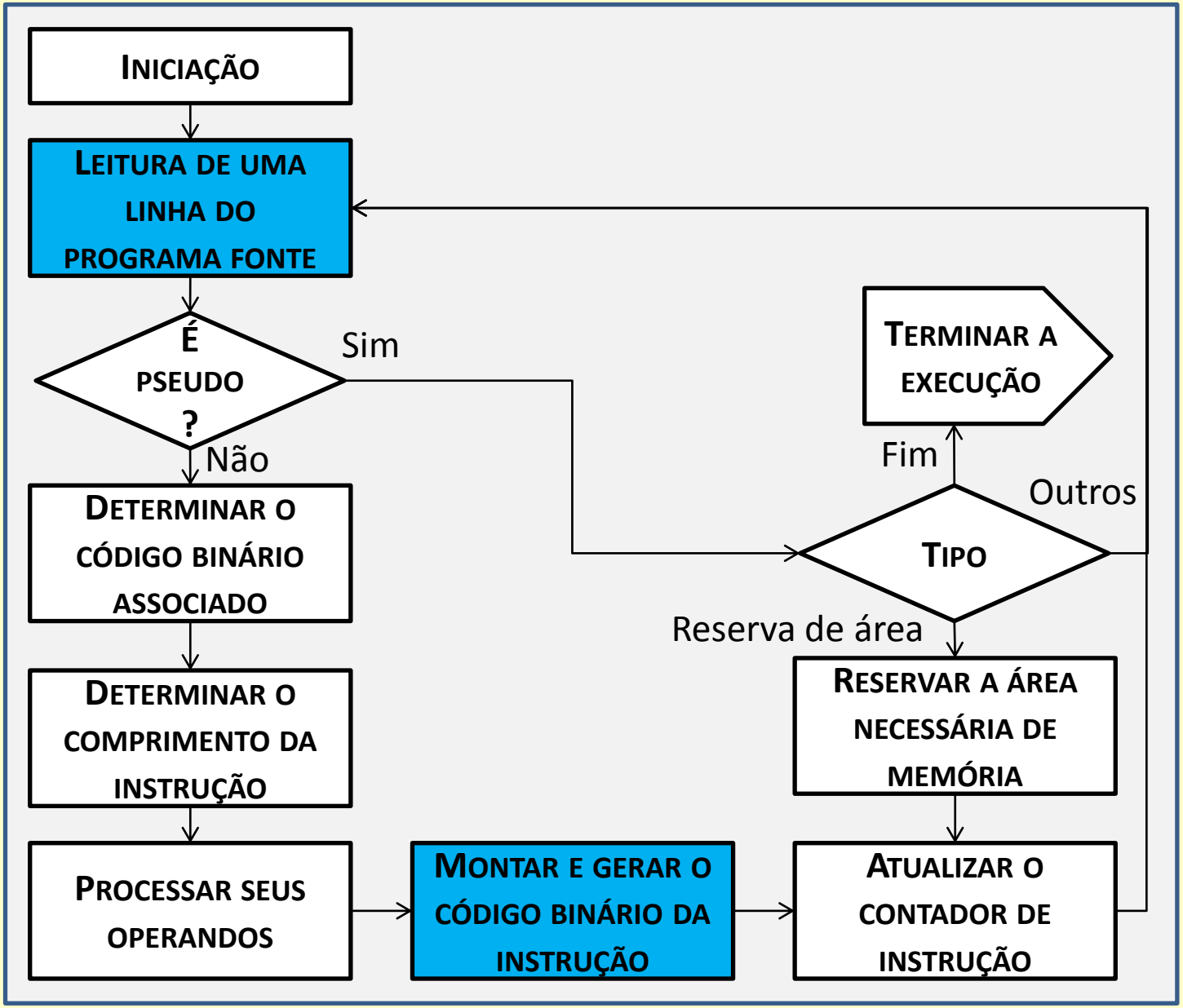


PASSO 2

Atividades principais

- 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



Principais Estruturas de Dados

- **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

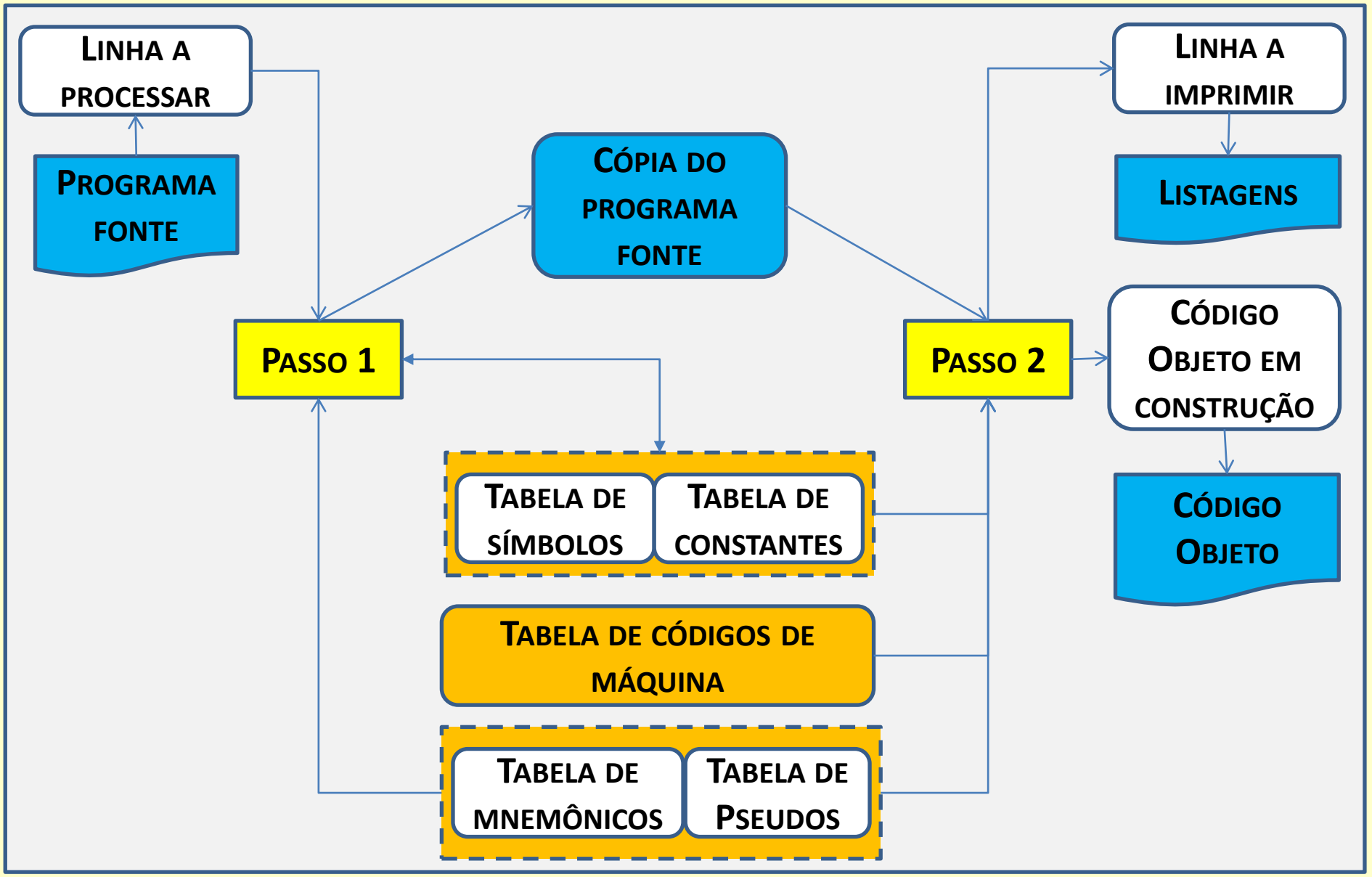


Tabela de Símbolos

- 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 físico** na memória
- Quando a localização física das posições ocupadas pelo código é conhecido, **endereços absolutos** são registrados na tabela de símbolos, associados ao endereço simbólico.
- Em montadores para programas com endereçamento **relativo**, os endereços associados aos diversos rótulos (endereços simbólicos) designam localizações (**distâncias**) relativas à posição ocupada na memória pela primeira posição do programa, e permanecem relativos até o momento em que o endereço de alocação física na memória se tornar conhecido.

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
  - 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
	...	...			

Pseudo Instruções

- **Pseudo-instruções** são linhas, na linguagem simbólica do programa fonte, que têm aparência muito semelhante à das linhas que representam instruções de máquina, porém não especificam instruções mas fornecem informações de orientação ao montador.
- M**ontadores absolutos** costumam permitir ao programador fornecer ao montador informações através de **pseudo-instruções** tais como as seguintes (naturalmente, os mnemônicos costumam ser escolhidos arbitrariamente):
  - **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 um valor fornecido)
  - **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.

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** já apresentada, 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 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.

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, 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.

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 for igual a 1, ignorar a linha lida
  - Se Passo for igual a 2, então listar a linha

Se a linha tiver rótulo

- Procurar o rótulo da linha na Tabela de Símbolos
- Se já **existe** na tabela, e está **definido**,
  - reportar **erro de múltipla definição** de rótulo
- Se já **existe** na tabela, 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 na Tabela de Símbolos como rótulo 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 for achado, **reportar erro: mnemônico inválido**
- Atualizar o contador de instruções, conforme o tipo de mnemônico encontrado
- Se o mnemônico exigir operando:
  - Analisar o campo de operando:
    - **Incluir** eventuais símbolos novos **na tabela de símbolos**
    - Os que não constarem na tabela, **marcar como indefinidos**
    - **Atualizar** a tabela de **referências cruzadas** (referências)
  - **Avaliar eventual expressão** encontrada no operando
  - Se for passo 2 e houver código-objeto associado a gerar
    - **Montar código objeto** correspondente

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.