PCS 3216
Sistemas de Programação

João José Neto

Aula 11 – Montadores de dois passos – Parte 2

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.

PARA TRABALHAR NO PROJETO - RECOMENDAÇÕES GERAIS

Desenvolvimento do projeto

- Nesta altura do progresso da disciplina já temos todo o material essencial para desenvolver e implementar o montador:
  - O conhecimento dos detalhes do funcionamento de um sistema de programação e de seus módulos mais importantes, e dos vínculos principais entre eles.
  - Aa especificação detalhada do conjunto de instruções e pseudos da máquina a ser usada.
  - O método de simulação guiada por eventos, associado à modelagem baseada em regras.
  - A ferramenta básica de apoio ao emprego do método adotado de implementação baseada no uso do motor de eventos.

Detalhamento do montador

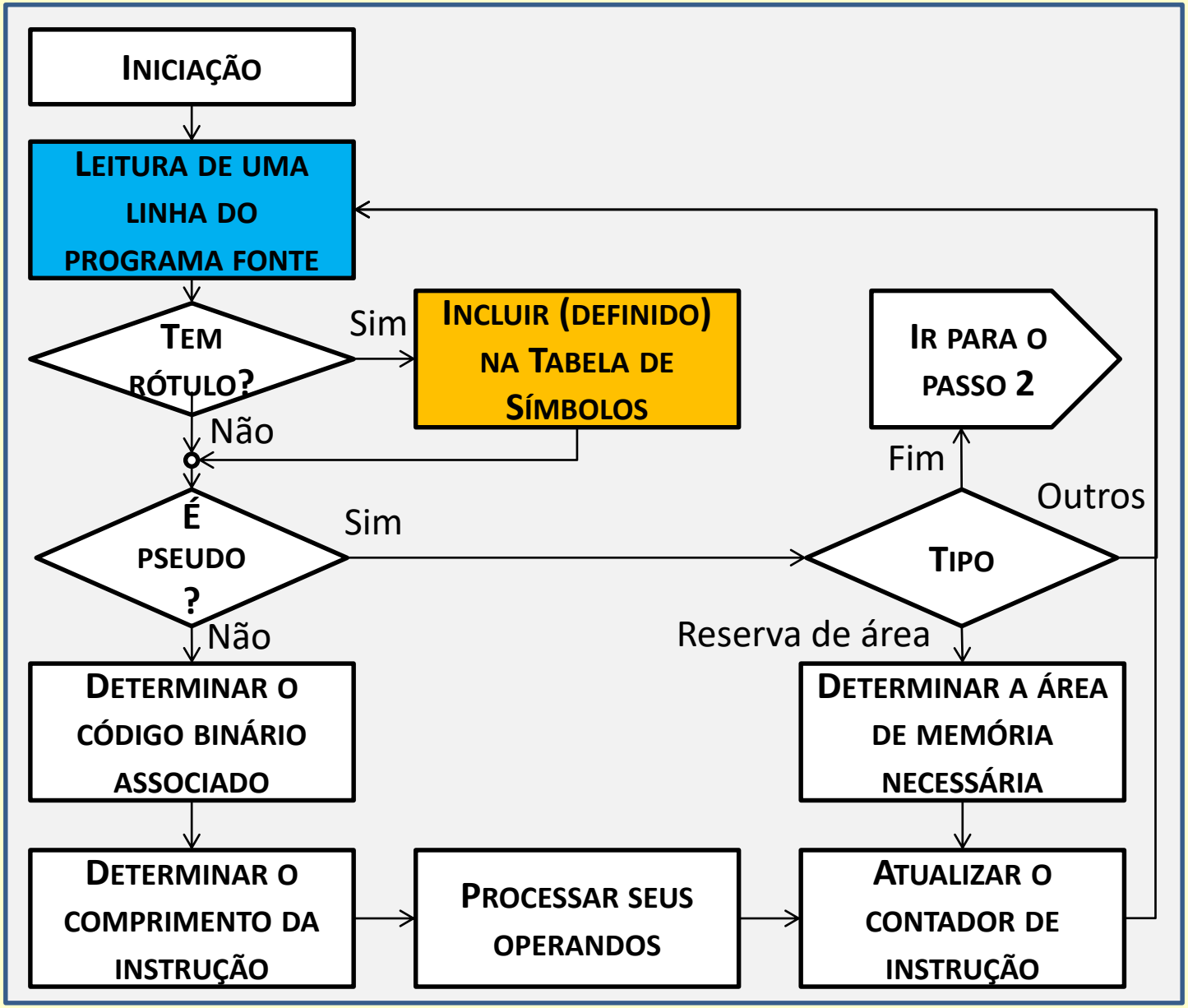
- Elabore esse detalhamento para ser usado como guia de implementação de um montador absoluto de dois passos para a linguagem simbólica desta máquina virtual.
- Evite saltar etapas, ou simplificar demais, ou estender muito o problema a resolver, para que os conceitos e fundamentos associados a cada uma das fases do projeto possam ser adequadamente assimilados e consolidados.

ALGUNS SLIDES DA AULA ANTERIOR

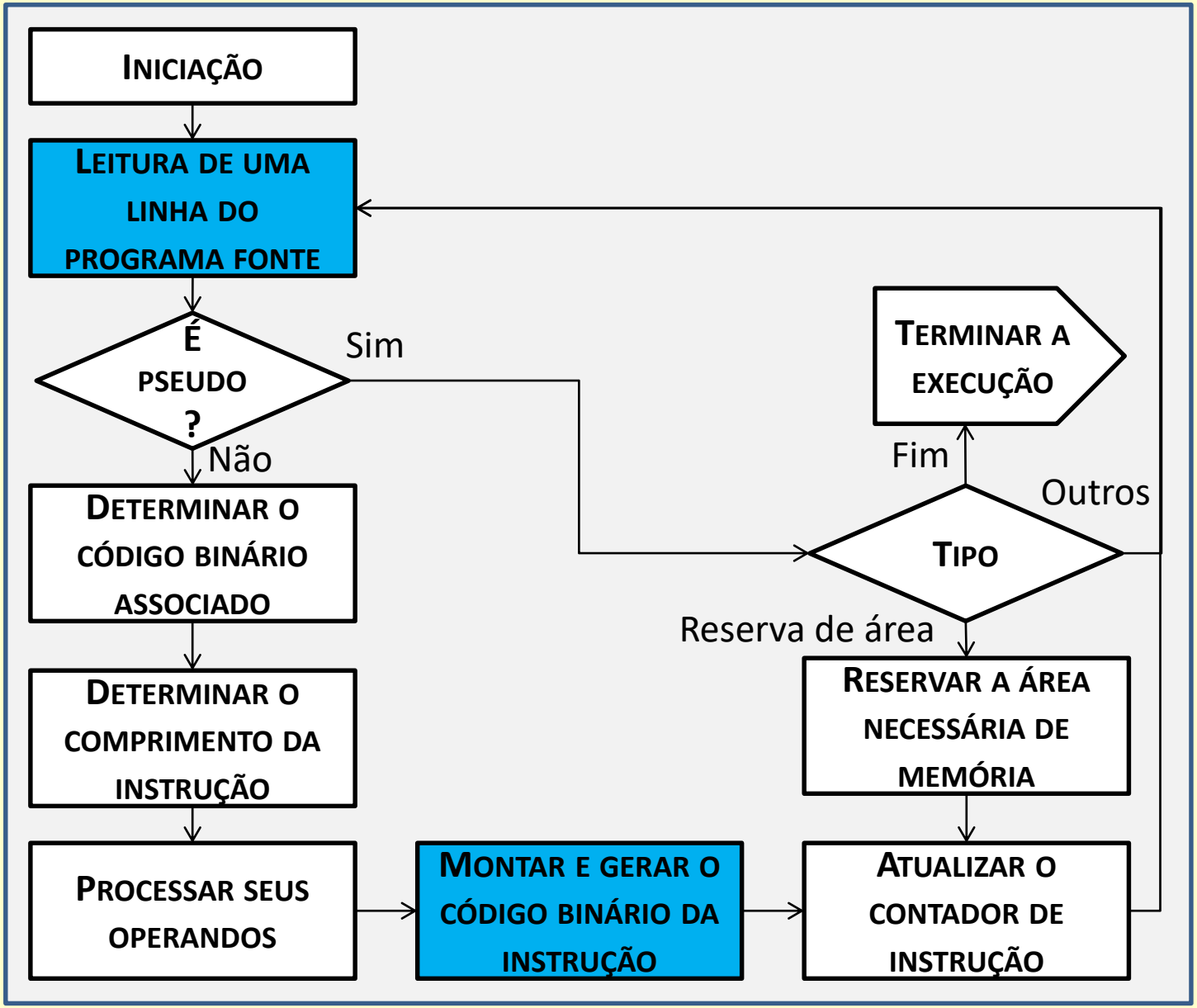
Tabela de Instruções e Pseudo-instruções

```
; MNEMÔNICOS    CÓDIGO    INSTRUÇÃO DA MÁQUINA VIRTUAL
;                   OBS.1: y= número do banco de memória (hexadecimal,4 bits)
;                   OBS.2: x= dígito (hexadecimal, 4 bits)
;   JP    J     /0xxx     JUMP UNCONDITIONAL desvia para endereço /yxxx
;   JZ    Z     /1xxx     JUMP IF ZERO idem, se acumulador contém zero
;   JN    N     /2xxx     JUMP IF NEGATIVE idem, se acumulador negativo
;   CN    C     /3x       * instruções de controle (/x = operação desejada)
;   +     +     /4xxx     ADD soma ac. + conteúdo do endereço /yxxx (8bits)
;   -     -     /5xxx     SUBTRACT idem, subtrai (operação em 8 bits)
;   *     *     /6xxx     MULTIPLY idem, multiplica (operação em 8 bits)
;   /     /     /7xxx     DIVIDE idem, divide (operação em 8 bits)
;   LD    L     /8xxx     LOAD FROM MEMORY carrega ac. com dado de /yxxx
;   MM    M     /9xxx     MOVE TO MEMORY move para /yxxx o conteúdo do ac.
;   SC    S     /Axxx     SUBROUTINE CALL guarda end.de retorno e desvia
;   OS    O     /Bx       * OPERATING SYSTEM CALL - 16 chamadas do sistema
;   IO    I     /Cx       * INPUT/OUTPUT – entrada, saída, interrupção
;               /D        * combinação disponível
;               /E        * combinação disponível
;               /F        * combinação disponível
;
; MNEMÔNICOS    OPERANDO  PSEUDO-INSTRUÇÃO DO MONTADOR
;   @     @     /yxxx     ORIGIN define end. inicial do código a seguir
;   #     #     /yxxx     END define final físico do prog. e end. de partida
;   $     $     /xxx      ARRAY define área de trabalho c/tamanho indicado
;   K     K     /xx       CONSTANT preenche byte corrente c/ constante
```

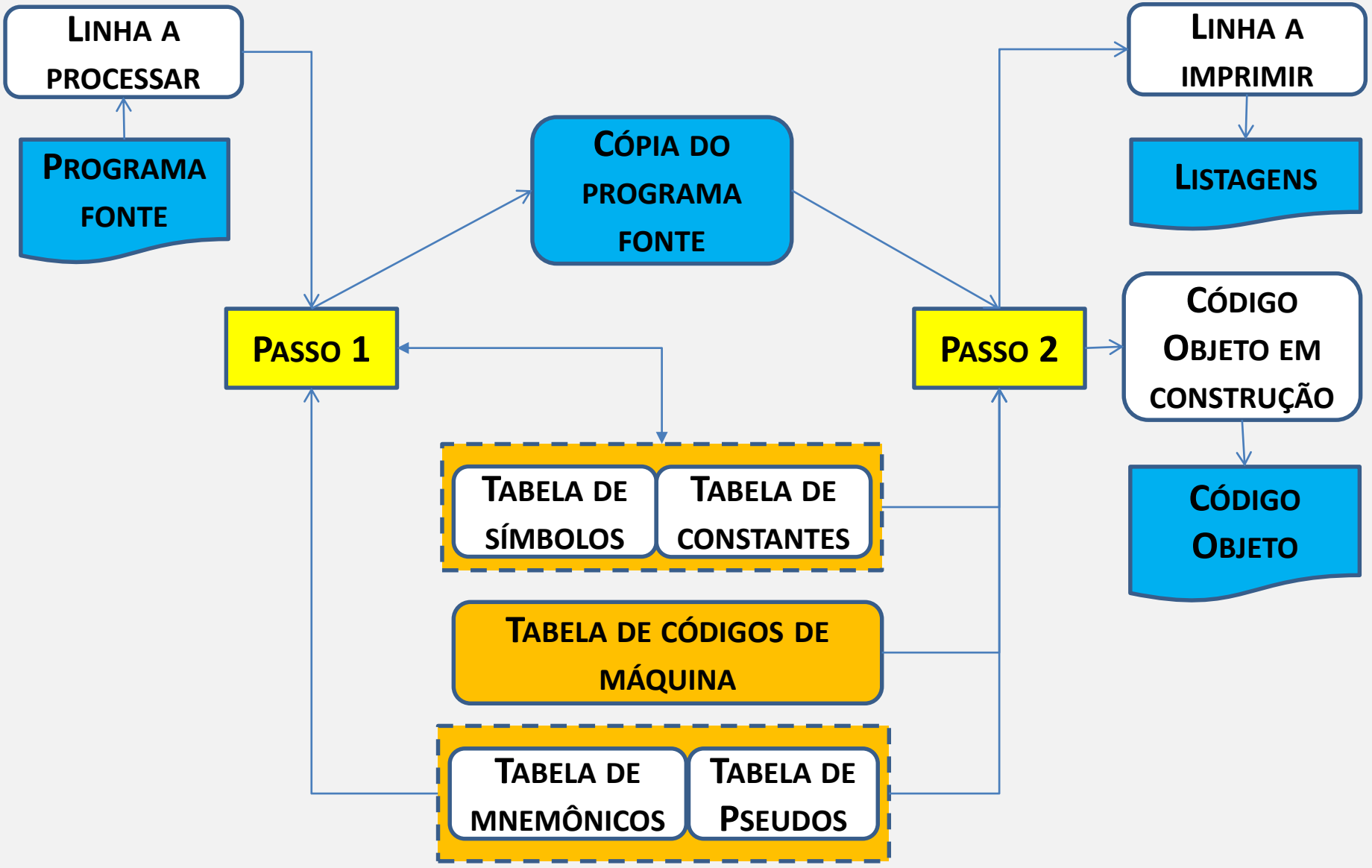
Lógica resumida do passo 1 do montador



Lógica resumida do passo 2 do montador



Áreas de Dados usadas pelo montador



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

Aspecto típico de Tabelas de Mnemônicos

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

Variáveis mais importantes do montador

- Além das duas grandes tabelas, algumas importantes informações foram sugeridas pelo contexto:
  - **Passo** – indica se é o 1º ou o 2º passo da montagem
  - **Linha fonte em uso** – vetor com a parte em uso do texto fonte
  - **Contador de Instruções** – apontador que indica o próximo endereço de memória a ser preenchido com código objeto
  - **Primeira posição livre na tabela de símbolos** – aponta a linha seguinte à última das linhas preenchidas da tabela de símbolos
  - **Comprimento da tabela de mnemônicos** – indica o total de mnemônicos nela contidos
  - **Tamanho limite para o programa** – indica a quantidade total de memória disponível para abrigar o código objeto
  - **Bloco de saída do código objeto** – vetor contendo a imagem (geralmente incompleta) do código objeto (ainda a preencher como resultado da montagem do programa fonte, a partir da linha correntemente em uso e seguintes).

Tabela de Instruções e Pseudo-instruções

```
; MNEMÔNICOS   CÓDIGO   INSTRUÇÃO DA MÁQUINA VIRTUAL
;                 OBS.1: y=número do banco de memória (hexadecimal,4 bits)
;                 OBS.2: x=dígito (hexadecimal, 4 bits)
;   JP          /0xxx    JUMP (UNCONDITIONAL) desvia para endereço /yxxx
;   JZ          /1xxx    JUMP IF ZERO idem, se acumulador contém zero
;   JN          /2xxx    JUMP IF NEGATIVE idem, se acumulador negativo
;   CN          /3x      * instruções de controle (/x = operação desejada)
;   +           /4xxx    ADD soma ac. + conteúdo do endereço /yxxx (8bits)
;   -           /5xxx    SUBTRACT idem, subtrai (operação em 8 bits)
;   *           /6xxx    MULTIPLY idem, multiplica (operação em 8 bits)
;   /           /7xxx    DIVIDE idem, divide (operação em 8 bits)
;   LD          /8xxx    LOAD FROM MEMORY carrega ac. com dado de /yxxx
;   MM          /9xxx    MOVE TO MEMORY move para /yxxx o conteúdo do ac.
;   SC          /Axxx    SUBROUTINE CALL guarda end.de retorno e desvia
;   OS          /Bx      * OPERATING SYSTEM CALL - 16 chamadas do sistema
;   IO          /Cx      * INPUT/OUTPUT – entrada, saída, interrupção
;               /D       * combinação disponível
;               /E       * combinação disponível
;               /F       * combinação disponível
;
; MNEMÔNICOS   OPERANDO PSEUDO-INSTRUÇÃO DO MONTADOR
;   @           /yxxx    ORIGIN define end. inicial do código a seguir
;   #           /yxxx    END define final físico do prog. e end. de partida
;   $           /xxx     ARRAY define área de trabalho c/tamanho indicado
;   K           /xx      CONSTANT preenche byte corrente c/ constante
```