PCS3616

Programação de Sistemas

(Sistemas de Programação)

Semana 6, Aula 10

Introdução

Programação em linguagem de Montagem

Escola Politécnica da Universidade de São Paulo

Construção de Programas em Linguagem de Máquina (1)

- Escrever um programa usando diretamente codificação binária não é uma tarefa simples, e tampouco agradável.
- Entretanto, foi este tipo de codificação que permitiu a construção dos primeiros programas do curso.
- Naturalmente, se um programa é muito grande ou se lida com diversas estruturas complexas (listas, etc.), a sua codificação se torna ainda mais difícil e complexa.

Construção de Programas em Linguagem de Máquina (2)

- Por conta disso, torna-se imprescindível construir alguma **abstração** que facilite a programação e a verificação dos programas.
- A primeira idéia, mais natural, é utilizar o modelo de máquina existente e, a partir dele, definir nomes (mnemônicos) para cada instrução da máquina.
- Posteriormente, verifica-se que somente isso não basta, pois é necessário lidar com os endereços dentro de um programa (rótulos, operandos, sub-rotinas), com a reserva de espaço para tabelas, com valores constantes.
- Enfim, é necessário definir uma **linguagem simbólica**.

Linguagem Simbólica

- Uma instrução de máquina tem usualmente o aspecto seguinte em sua imagem mnemônica:

0012 JZ 042 ; 1042 0012=rótulo **J**Z=mnemônico 042=operando numérico

- A mesma instrução, em linguagem simbólica, pode ser escrita com ou sem um rótulo simbólico, e pode também referenciar um operando através de um rótulo simbólico ou numérico:

Q JZ R ; Q=rótulo JZ=mnemônico R=operando simbólico

JZ R ; rótulo omitido JZ=mnemônico R=operando simbólico

Q JZ 042 ; Q=rótulo JZ=mnemônico 042=operando numérico

JZ 042 ; rótulo omitido JZ=mnemônico 042=operando numérico

Linguagem Simbólica

- Convenciona-se que sempre o primeiro elemento da linha é um rótulo; caso o rótulo for omitido deverá haver uma instrução
- Entre os elementos de uma linha deve haver ao menos um espaço
- Cada linha deve conter uma instrução/pseudo-instrução completa
- À direita de um ponto-e-vírgula, todo texto é ignorado (=comentário)
- Mnemônicos e significado das pseudo-instruções:

  - **@** (Operando numérico: define endereço da instrução seguinte)
  - **$** (Reserva de área de dados)
  - # (Final físico do texto-fonte. Operando=endereço de execução)
  - **K** (Constante. Operando numérico = valor da constante, em hexadecimal)

Programa Exemplo em linguagem simbólica

O programa abaixo, que foi dado como exemplo na aula 7:

```
0100          8F00   Obtém o endereço para onde se deseja mover o dado
0102          4F02   Compõe o endereço com o código de operação Move
0104          9106   Guarda instrução montada para executar em seguida
0106          9000   Executa a instrução recém-montada
0108          ....   Provavelmente, o código seguinte altera o conteúdo de 0F00
....
015C          0100   Volta a repetir o procedimento, para outro endereço.
....
0F00          034C   Endereço (34C) para onde se deseja mover o dado
0F02          9000   Código de operação Move, com operando 000
```

codificado em linguagem simbólica, fica com o seguinte aspecto:

```
  @ /0100     ; @=origem do código 0100=posição de memória (em hexadecimal)
P LD E        ; P=rótulo LD=load E=endereço simbólico da constante 034C
  +  M        ; +=add M=rótulo de onde está uma instrução Move 0000
  MM X        ; MM=move X=endereço da instrução seguinte
X MM /0       ; reservado para guardar a instrução recém-montada
  ...
  JP P        ; JP=jump (desvio) P=rótulo da primeira instrução deste programa
  ...
E K  /034C    ; E=rótulo K=constante 034C=operando numérico, em hexadecimal
M MM /0000    ; M=rótulo MM=move 0000=operando zero
  # P         ; #=final físico P=rótulo da primeira instrução a ser executada
```

Tabela de mnemônicos para a MVN (de 2 caracteres)

Operação **0** **Jump** Mnemônico **JP**	Operação **1** Jump if **Zero** Mnemônico **JZ**	Operação **2** Jump if **Negative** Mnemônico **JN**	Operação **3** Load **Value** Mnemônico **LV**
Operação **4** **Add** Mnemônico **+**	Operação **5** **Subtract** Mnemônico **–**	Operação **6** **Multiply** Mnemônico *****	Operação **7** **Divide** Mnemônico **/**
Operação **8** **Load** Mnemônico **LD**	Operação **9** Move to **Memory** Mnemônico **MM**	Operação **A** **Subroutine** Call Mnemônico **SC**	Operação **B** **Return** from Sub. Mnemônico **RS**
Operação **C** **Halt** Machine Mnemônico **HM**	Operação **D** **Get** Data Mnemônico **GD**	Operação **E** **Put** Data Mnemônico **PD**	Operação **F** **Operating System** Mnemônico **OS**

Exercício

Desenvolva um programa que, dada uma sequência de 5 dados, verifica se ela está escrita em ordem decrescente. No caso da sequência estar ordenada, o valor na variável de saída deve ser 0001, em caso contrário, o valor deve ser 0000. Use auto-modificação do código para ler a sequência de dados e desenvolva sub-rotina **Menor** (dados x e y verifica se x<y).

Nos comentários do código documente as condições, premissas e limitações consideradas.

Endereço de início do programa principal: 0000
Endereço da variável de saída: 0002 (resultado)
Endereços da lista de dados: 0004 a 000C

USE LINGUAGEM SIMBÓLICA (ASSEMBLY)

Bibliografia (Programação de Sistemas)

Relíquias Preciosas

Barron, D. W. ***Assemblers and Loaders*** (3rd. ed.) MacDonald/Elsevier, 1978

Beck, L. L. ***System Software - An Introduction to Systems Programming*** Addison-Wesley, 1996

Calingaert, P. ***Assemblers, Compilers and Program Translation*** Computer Science Press, 1979

Donovan, J. J. ***Systems Programming*** McGraw-Hill, 1972

Duncan, F.G. ***Microprocessor Programming and Software Development*** Prentice Hall, 1979.

Freeman, P. ***Software System Principles*** SRA, 1975

Gear, C. W. ***Computer Organization and Programming (3rd. ed.)*** McGraw-Hill, 1980

Graham, R. M. ***Principles of Systems Programming*** John Wiley & Sons, 1975

Gust, P. ***Introduction to Machine and Assembly Language Programming*** Prentice Hall, 1985

Maginnis, J. B. ***Elements of Compiler Construction*** Appleton-Century-Crofts, Meredith Co., 1972

Presser, L. and White, J. R. ***Linkers and Loaders*** ACM Comp. Surveys, vol. 4, n. 3, pp. 149-168, 1972

Rosen, S. (ed.) ***Programming Systems and Languages*** McGraw-Hill, 1967

Tseng, V. (ed.) ***Microprocessor Development and Development Systems*** McGraw-Hill, 1982

Ullman, J. D. ***Fundamental Concepts of Programming Systems*** Addison-Wesley, 1976

Wegner, P. ***Progr. Languages, Inf. Structures and Machine Organization*** McGraw-Hill, 1968.

Welsh, J. and McKeag, M. ***Structured System Programming*** Prentice-Hall,1980

Referências Bibliográficas

Bryant R. E. and O'Hallaron, D. R. ***Computer Systems: A Programmer's Perspective***, 2010.

DONOVAN, J. ***Systems Programming***, 1972.

Leitura complementar:

UM SIMULADOR-INTERPRETADOR PARA A LINGUAGEM DE MÁQUINA DO PATINHO FEIO.

(João José Neto, Aspectos do Projeto de Software de um Minicomputador, Dissertação de Mestrado, EPUSP, S. Paulo, 1975, cap.3)

Transparências extraídas e alteradas de:

José Neto, J., Sichman, J. S., Silva, P.S.M., Rocha, R.L.A. ***Material didático da disciplina PCS 2024 – Laboratório de Fundamentos da Engenharia de Computação***, PCS/EPUSP, São Paulo, SP. 2005-2015.

PCS3616

Programação de Sistemas

(Sistemas de Programação)

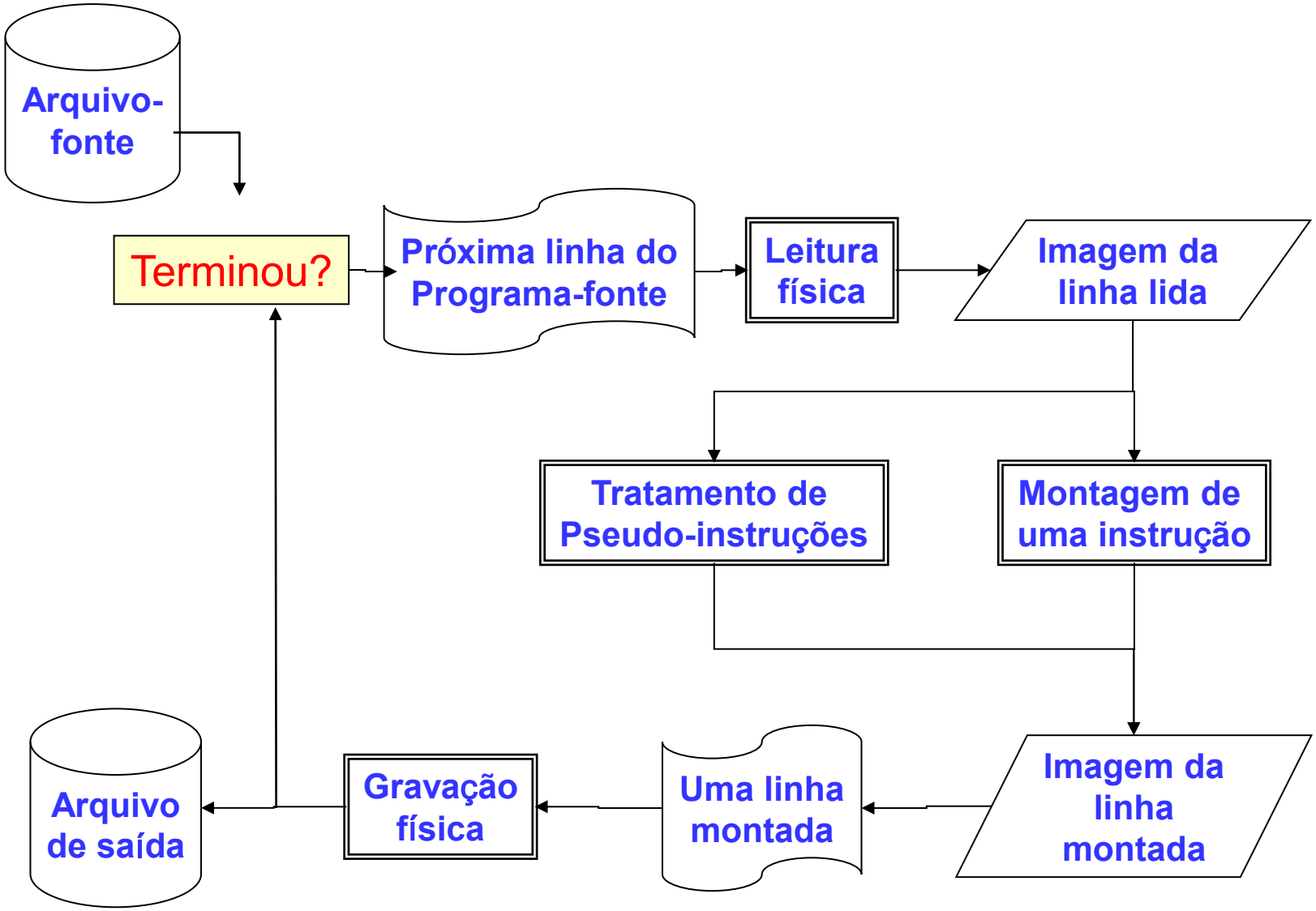
Semana 6, Aula 11

Introdução

Programação em linguagem de Montagem

Escola Politécnica da Universidade de São Paulo

Esquema geral de um montador



Construção de um Montador

- Para a construção de um montador pressupõe-se que sejam tratadas as seguintes questões:
  - **definição das instruções**: determinar os mnemônicos que as representam simbolicamente;
  - **definição das pseudo-instruções**: determinar os mnemônicos que as representam, bem como sua função para o montador
- Durante a execução de um montador, pressupõe-se que sejam resolvidos os seguintes problemas:
  - **alocação dos rótulos**: determinar qual será o endereço efetivo de um nome encontrado;
  - **geração de código:** gerar um arquivo com o código correspondente em linguagem de máquina
- Para cumprir esta tarefa é necessário completar, em primeiro lugar, as definições dos mnemônicos (instruções e pseudo-instruções), para se pensar posteriormente, nos algoritmos.

Pseudo-Instruções

- Na aula anterior foi determinado o conjunto de mnemônicos para as instruções, nesta aula serão definidas aqueles relativos às pseudo-instruções.
- As pseudo-instruções utilizadas no montador desta aula (montador absoluto) são as seguintes:
  - **@** : Recebe um operando numérico, define o endereço da instrução seguinte;
  - **K** : Constante, o operando numérico tem o valor da constante (em hexadecimal);
  - **$** : Reserva de área de dados, o operando numérico define o tamanho da área a ser reservada;
  - **#** : Final físico do texto fonte.
    - Há ainda pseudo-instruções para a definição do tipo de dado recebido, '**/**' simboliza hexadecimal, ou seja '**/0100**' indica valor 0x0100 (em hexadecimal), ou 256 decimal. Há outras possibilidades de representação, como binário, etc.

Formas de Construção de um Montador

- Há mais de uma forma de se tratar ao construir um Montador. Pelo menos duas são imediatas:
  - Montador de um passo:
    - Lê o código fonte uma única vez;
    - Armazena dinamicamente os rótulos não definidos em uma lista de pendências;
    - Gera o código para cada linha de entrada completamente definida;
    - Resolve uma pendência caso a linha de entrada inicie com um rótulo pendente;
    - Ao final, completa as linhas de código que ainda não haviam sido completamente definidas, resolvendo todos os rótulos pendentes.
  - Montador de dois passos:
    - Lê o código fonte da primeira vez;
    - Num primeiro passo, trata todas as linhas apenas para resolver os endereços dos rótulos e tratar as pseudo-instruções;
    - Lê novamente o código fonte num segundo passo para gerar o código correspondente ao programa

Estruturas de Dados do Montador (1)

- O montador precisará de um conjunto de estruturas de dados que o permitirão conduzir a tarefa. Dentro deste conjunto, há as seguintes estruturas de dados:
  - **locationCounter**: define a localização atual (endereço corrente) de execução.
  - **Tabela de instruções**: define as instruções válidas (símbolo e valor).
  - **Tabela de pseudo-instruções**: define as pseudo-instruções válidas (símbolo e valor).
  - **Tabela de símbolos**: permite armazenar e recuperar os rótulos (símbolo e endereço real).

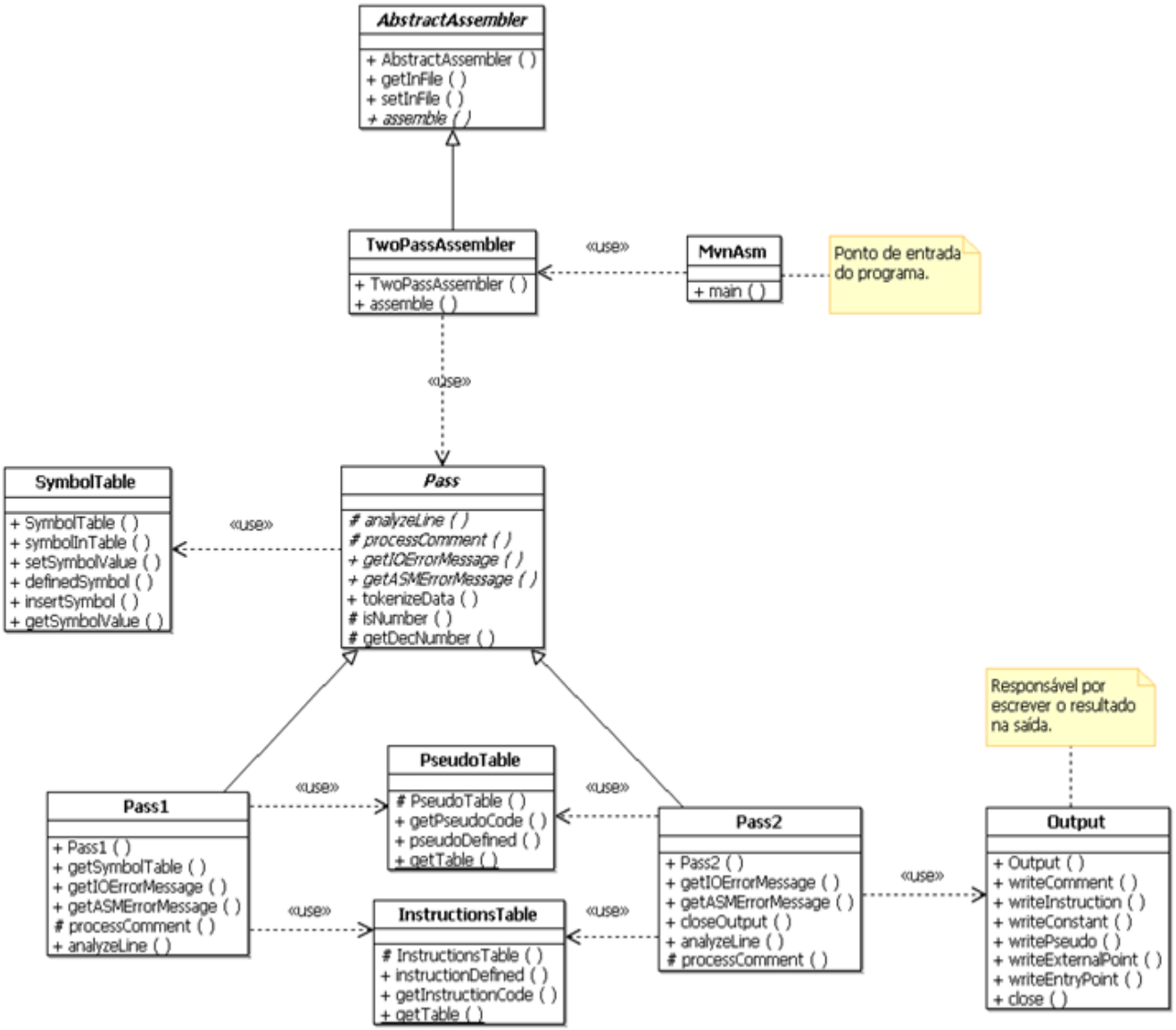
Estruturas de Dados do Montador (2)

- Além destas estruturas, o montador utiliza um conjunto de arquivos (um de entrada e pelo menos dois de saída). Pode ser necessário gerar o texto objeto em algum formato específico, para que um programa *loader* possa carregá-lo na memória.
- Pode-se, ainda, armazenar o conteúdo do texto fonte durante o passo 1 para facilitar a execução do passo 2.

Construção do Montador

- Nesta disciplina foi escolhido realizar um montador de dois passos em linguagem de alto nível. Esta escolha nos conduz à definição das ações a serem realizadas em cada um dos dois passos do montador. Assim temos:
  - **Passo1**: O objetivo é definir os símbolos encontrados, sejam eles rótulos encontrados antes das instruções, ou ainda rótulos de destino de alguma instrução. Para isso deve-se:
    - Manter atualizado o endereço de execução corrente, chamado de **locationCounter**.
    - Armazenar os valores dos símbolos (rótulos) na Tabela de Símbolos (**TS**) para uso posterior no passo 2.
    - Processar as pseudo-instruções.
  - **Passo2**: O objetivo é gerar o código objeto e possivelmente um arquivo de listagem contendo além do código objeto, o texto fonte à direita do código objeto. Para isso, este passo deve:
    - Recuperar os valores dos símbolos (da **TS**).
    - Gerar as instruções.
    - Processar as pseudo-instruções.

Diagrama de Classes do Montador Java



Classes do Montador Absoluto

- O montador é definido a partir de uma classe abstrata (*AbstractAssembler*) porque o projeto deve prever a possibilidade de existência de um montador de um passo, ou de dois passos. A partir da decisão de projeto, o montador construído para esta disciplina é de dois passos (*TwoPassAssembler*). As demais classes são:
  - **Tabela de instruções** (*InstructionsTable*): define as instruções válidas (símbolo e valor).
  - **Tabela de pseudo-instruções** (*PseudoTable*): define as pseudo-instruções válidas (símbolo e valor).
  - **Tabela de símbolos** (*SymbolTable*): permite armazenar e recuperar os rótulos (símbolo e endereço real).
  - **Passo** (*Pass*): define a estrutura dos passos, que são derivados desta classe (*Pass1* e *Pass2*).
  - **Saída** (*Output*): responsável por toda saída de dados para os arquivos.
  - **Ponto de entrada** (*MvnAsm*): contém o programa principal que inicia o montador.

Lógica Geral do Montador

- O algoritmo utilizado é o seguinte:

```
begin
    Marque o endereço inicial de geração de código como 0 /* locationCounter */;
    abra o arquivo com o programa;
    while não encontra fim de arquivo do
        Passo1: leia uma linha preenchendo a Tabela de Símbolos (TS);
    end
    reinicie o arquivo;
    while não encontra fim de arquivo do
        Passo2:  begin
            leia uma linha;
            monte e gere o código objeto correspondente usando a TS;
            escreva o código gerado nos arquivos de saída (em arquivo de saída carregável);
        end
    end
    fecha os arquivos;
end
```

Leitura e Tratamento das Linhas

```
begin
    leia a linha até o final <EOL>;
    separe os tokens da linha (ou seja, palavras);
    while houver tokens do
        if não é comentário then
            armazena temporariamento o token;
        end
    end
    if há token armazenado then
        analiza a linha;
    end
    incremente o contador de linhas;
    pegue a próxima linha;
end
```

Análise de uma Linha

```
begin
    if há 3 tokens then
        // há um rótulo e deve ser tratado;
        if rótulo não está definido na TS then
            defina o valor do rótulo na TS como sendo o locationCounter;
            incremente o locationCounter de 2 (porque a abstração da MVN é Word);
        else
            erro! o rótulo já estava definido;
        end
    end
    Teste a validade do restante da linha (operador e operando);
end
```

Teste de Operador e de Operando

```
begin
    verifique se o operador é válido (se é instrução ou pseudo);
    verifique se o operando é válido (se é número ou rótulo);
end
```

Montagem e Geração de Código

```
begin
    if argumento é instrução then
        pegue e monte o código correspondente a partir da Tabela de instruções;
    else
        if é pseudo then
            trate a pseudo corretamente;
        end
    end
end
```

Bibliografia (Programação de Sistemas)

Relíquias Preciosas

Barron, D. W. ***Assemblers and Loaders*** (3rd. ed.) MacDonald/Elsevier, 1978

Beck, L. L. ***System Software - An Introduction to Systems Programming*** Addison-Wesley, 1996

Calingaert, P. ***Assemblers, Compilers and Program Translation*** Computer Science Press, 1979

Donovan, J. J. ***Systems Programming*** McGraw-Hill, 1972

Duncan, F.G. ***Microprocessor Programming and Software Development*** Prentice Hall, 1979.

Freeman, P. ***Software System Principles*** SRA, 1975

Gear, C. W. ***Computer Organization and Programming (3rd. ed.)*** McGraw-Hill, 1980

Graham, R. M. ***Principles of Systems Programming*** John Wiley & Sons, 1975

Gust, P. ***Introduction to Machine and Assembly Language Programming*** Prentice Hall, 1985

Maginnis, J. B. ***Elements of Compiler Construction*** Appleton-Century-Crofts, Meredith Co., 1972

Presser, L. and White, J. R. ***Linkers and Loaders*** ACM Comp. Surveys, vol. 4, n. 3, pp. 149-168, 1972

Rosen, S. (ed.) ***Programming Systems and Languages*** McGraw-Hill, 1967

Tseng, V. (ed.) ***Microprocessor Development and Development Systems*** McGraw-Hill, 1982

Ullman, J. D. ***Fundamental Concepts of Programming Systems*** Addison-Wesley, 1976

Wegner, P. ***Progr. Languages, Inf. Structures and Machine Organization*** McGraw-Hill, 1968.

Welsh, J. and McKeag, M. ***Structured System Programming*** Prentice-Hall,1980

Referências Bibliográficas

Bryant R. E. and O'Hallaron, D. R. ***Computer Systems: A Programmer's Perspective***, 2010.

DONOVAN, J. ***Systems Programming***, 1972.

Leitura complementar:

UM SIMULADOR-INTERPRETADOR PARA A LINGUAGEM DE MÁQUINA DO PATINHO FEIO.

(João José Neto, Aspectos do Projeto de Software de um Minicomputador, Dissertação de Mestrado, EPUSP, S. Paulo, 1975, cap.3)

Transparências extraídas e alteradas de:

José Neto, J., Sichman, J. S., Silva, P.S.M., Rocha, R.L.A. ***Material didático da disciplina PCS 2024 – Laboratório de Fundamentos da Engenharia de Computação***, PCS/EPUSP, São Paulo, SP. 2005-2015.