Aula 8

Professores:

Lúcia M. A. Drummond

Simone de Lima Martins

Conteúdo:

Execução de Programas

- Introdução
- Linguagens de programação
- Montagem e Compilação
- Ligação ou Linkedição
- Execução de programas em código de máquina

Fundação CECIERJ
Consórcio cederj

Introdução

- Programa de computador: conjunto de instruções ou comandos organizados em uma certa seqüência para realizar uma tarefa específica

- Atualmente, é raro escrever um programa em linguagem de máquina: problemas de manutenção e dificuldade de organizar as instruções sem erro.

Linguagens de Programação

- Linguagem de programação: Linguagem criada para instruir um computador a realizar tarefas

- Código: programa completo, escrito em uma linguagem de programação

Linguagem de Programação

Linguagem de máquina:

- Tipo mais primitivo de linguagem de programação, com instruções diretamente executadas pela UCP
- Um programa de máquina é uma longa seqüência de algarismos binários

0010	0100	1001	0001	2 4 9 1
0100	0100	1001	1111	4 4 9 F
0100	0100	1001	0011	4 4 9 3
0001	0100	1001	0010	1 4 9 2
1000	0100	1001	1000	8 4 9 8
1110	0100	1001	1001	E 4 9 9
0011	0100	1001	0101	3 4 9 5
0100	0100	1001	1110	4 4 9 E
1111	0100	1001	1010	F 4 9 A
0000	0000	0000	0000	0 0 0 0

(a) Programa em linguagem binária **(b)** Programa em hexadecimal

(Fig. 9.1 do livro texto)

Linguagem de Programação

Dificuldades de utilização de linguagem de máquina:

- O programador deverá conhecer todas as instruções disponíveis para aquela máquina - códigos de operação e formatos, registradores da UCP, endereços de células de memória onde estão os dados e instruções

- Programa poderá conter milhares de instruções - tarefa tediosa e difícil

Linguagens de Programação

Linguagem Assembly:

- Códigos de operação como 0101 são mais fáceis de ser lembrados como ADD
- O programador não precisa mais guardar os endereços reais em memória onde dados e instruções estão armazenados - usa símbolos
- Linguagem de máquina: 1110 01001001 1001 ou E499
  Linguagem Assembly: ADD SALARIO

Linguagens de Programação

Assembler (montador)

- Traduz o programa assembly em código de máquina para ser executado pela UCP

- Lê cada instrução em linguagem de montagem e cria uma instrução em linguagem de máquina

```
ORG     ORIGEM
LDA     SALARIO - 1
ADD     SALARIO - 2
ADD     SALARIO - 3
SUB     ENCARGO
STA     TOTAL
HLT
DAD     SALARIO - 1
DAD     SALARIO - 2
DAD     SALARIO - 3
```

(Fig. 9.2 do livro texto)

Linguagens de Programação

Linguagem Assembly:

- Escrever programas ainda é uma tarefa árdua, tediosa e complexa
- Ainda é mais atraente do que escrever um programa em linguagem de máquina

Linguagens de Programação

Linguagens de alto nível:

- Refletem os procedimentos utilizados na solução de problemas, sem preocupação com o tipo de UCP ou de memória onde o programa será executado
- Mais simples, menos instruções do que Assembly
- Estruturadas de acordo com a compreensão e intenção do programador - linguagem de alto nível, nível afastado da máquina

Linguagens de Programação

Linguagem	Data	Observações
FORTRAN	1957	FORmula TRANslation - primeira linguagem de alto nível. Desenvolvida para realização de cálculos numéricos.
ALGOL	1958	ALGOrithm Language - linguagem desenvolvida para uso em pesquisa e desenvolvimento, possuindo uma estrutura algorítmica.
COBOL	1959	COmmon Business Oriented Language - primeira linguagem desenvolvida para fins comerciais.
LISP	1960	Linguagem para manipulação de símbolos e listas.
PL/I	1964	Linguagem desenvolvida com o próposito de servir para emprego geral (comercial e científico). Fora de uso.
BASIC	1964	Desenvolvida em Universidade, tornou-se conhecida quando do lançamento do IBM-PC, que veio com um interpretador da linguagem, escrito por Bill Gates e Paul Allen.
PASCAL	1968	Primeira linguagem estruturada - designação em homenagem ao matemático francês Blaise Pascal que, em 1642, foi o primeiro a planejar e construir uma máquina de calcular.
C	1967	Linguagem para programação de sistemas operacionais e compiladores.
ADA	1980	Desenvolvida para o Departamento de Defesa dos EUA.
DELPHI	1994	Baseada na linguagem Object Pascal, uma versão do Pascal orientada a objetos.
JAVA	1996	Desenvolvida pela Sun, sendo independente da plataforma onde é executada. Muito usada para sistemas Web.

(Tabela 9.1 do livro texto)

Linguagens de Programação

```
DETAIL PARAGRAPH.
        READ CARD-FILE AT END GO TO END-PARAGRAPH.
        MOVE CORRESPONDING CARD-IN TO LINE-OUT.
        ADD CURRENT-MONTH-SALES IN CARD-IN, YEAR-TO-DATE-SALES IN
                CARD-IN GIVING TOTAL-SALES-OUT IN LINE-OUT.
                WRITE LINE-OUT BEFORE ADVANCING 2 LINES AT EOP
                PERFORM HEADER-PARAGRAPH
```

(a) Trecho de programa em COBOL.

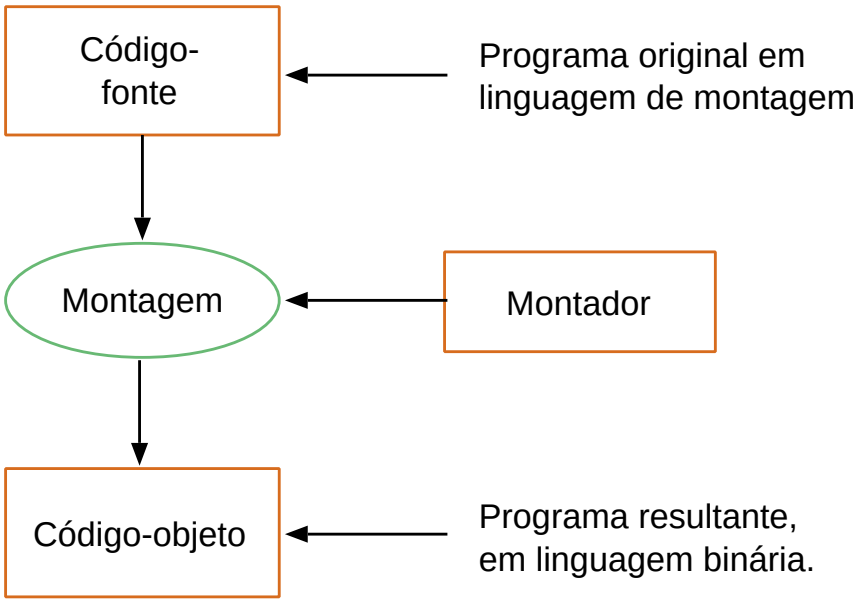
```
detail_procedure ( )
{
        int file_status, current_line, id_um;
        char *name;
        float current_sales, ytd_sales, total_sales;

        current_line=64;
        file_status=fscanf (card_file, "%d%s%f%f, &id_num, &name, &current_sales,
                &ytd_sales);
        while (file_status !=EOF) {
        if (current_line > 63) {
                header_procedure ( );
                current_line=3;
        }
        file_status=fscanf (card_file, "%d%s%f%f", &id_num, &name, &current_sales,
                &ytd_sales);
        total_sales = current_sales + ytd_sales;
        printf ("%4d%30s%6.2f%7.2f\n\n", id_num, name, current_sales,
                ytd_sales, total_sales);
        current_line ++; current_line ++;
        }
end_procedure ( );
```

(b) Trecho de programa em C.

(Fig. 9.3 do livro texto)

Montagem

- Tradução mais rápida e simples
- Realizada pelo Montador - Assembler
- Utilizada para traduzir um programa em linguagem de montagem (assembly) para seu equivalente em binário



(Fig. 9.4 do livro texto)

Montagem

Funções de um montador:

- Substituir códigos de operação simbólicos por valores numéricos.
  Ex: LOAD - 00101101, STR - 100010, ADD - 00001110,
  MOV - 11111100
- Substituir nomes simbólicos de endereços por valores numéricos dos endereços.
  Ex: ADD SOMA por 00001110 1011100011010
- Converter valores de constantes para valores binários.
- Examinar a correção de cada instrução. Ex: LDA pode ser uma instrução correta, mas LDB não. O montador não pode gerar código de operação inexistente

Montagem

```
0000                    ADDS        PROC        NEAR

0000    03 C3                       ADD         Ax, Bx
0002    03 C1                       ADD         Ax, Cx
0004    03 C2                       ADD         Ax, Dx
0006    C3                          RET
0007                    ADDS        ENDP
```

(Fig. 9.5 do livro texto)

Montagem

C. Op.	Descrição
HLT	Parar
INC	ACC ← ACC + 1
DCR	ACC ← ACC - 1
LDA Op.	ACC ← M (Op.)
STR Op.	M (Op.) ← ACC
ADD Op.	ACC ← ACC + M (Op.)
SUB Op.	ACC ← ACC - M (Op.)
JMP Op.	CI ← Op.
JP Op.	Se ACC > 0, então: CI ← Op.
JZ Op.	Se ACC = 0, então: CI ← Op.

(a) Subconjunto de instruções

```
Início :  ORG     ZERO          ; Origem do programa. Endereço relativo ∅.
          LDA     CONTADOR      ; Carregar valor do contador no ACC.
          JZ      FIM           ; Se contador = 0, então PARAR (desvia para FIM).
          LDA     Parcela 1
          ADD     Parcela 2     ; Realizar operação aritmética com dados.
          STR     Resultado
          LDA     Contador      ; Ler valor do contador para ACC.
          DCR     Zero          ; Substituir 1 do contador.
          JMP     Início        ; Voltar para início do loop.
Fim :     HLT                   ; Parar.
          DAD     Parcela 1
          DAD     Parcela 2
          DAD     Resultado
          DAD     Contador
```

(b) Programa em linguagem de montagem

(Fig. 9.6 do livro texto)

Montagem

Dois tipos básicos de símbolos:

- Códigos de operação. Ex: LDA, STR, JZ
- Endereços de dados ou de instruções. Ex: INICIO, FIM, CONTADOR, PARCELA 1

Montagem

Montador de 2 passos:

- O programa é examinado instrução por instrução duas vezes
- **Primeiro passo:** detecta erros, monta tabela de símbolos de endereços
- **Segundo passo:** cria o código objeto

Montagem

Montador de um passo:

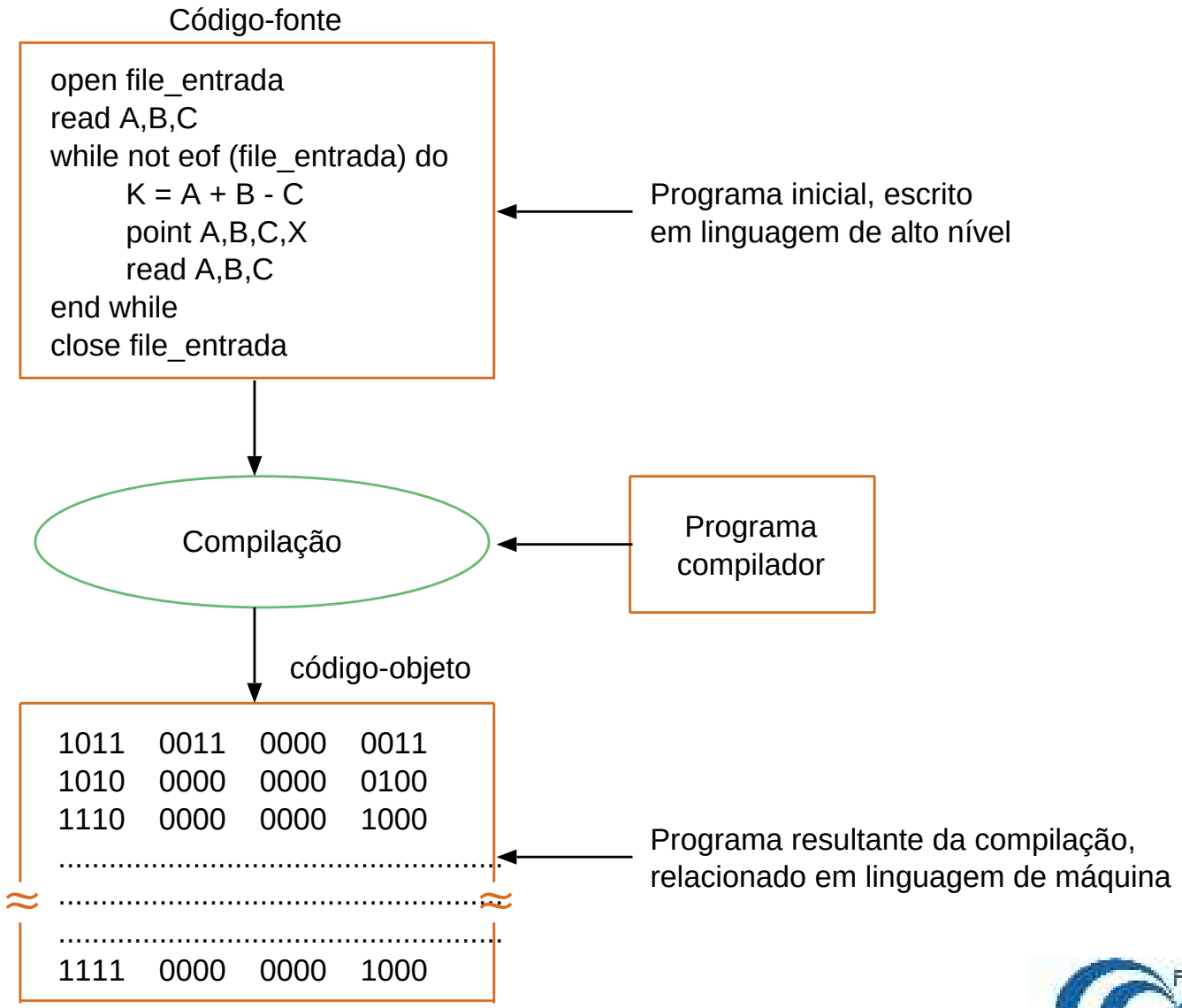
- Não tem a mesma clareza de execução do anterior
- Muitas instruções não podem ser montadas porque os endereços não são conhecidos e precisam ser completadas depois
- Se a tabela de endereços desconhecidos for grande (muitas referências a endereços inexistentes), a busca à suas diversas entradas será demorada, tão demorada quanto se tivesse realizado um segundo passo.

Compilação

- Processo de análise de um programa escrito em linguagem de alto nível (programa fonte) e sua tradução em um programa em linguagem de máquina (programa objeto)

- Compilador: programa que realiza tal tarefa

Compilação



(Fig. 9.7 do livro texto)

Compilação

Inicialmente ocorre a análise do código fonte (front-end):

- Análise léxica
- Análise sintática
- Análise semântica

Compilação

Análise Léxica

- Consiste em decompor o programa fonte em seus elementos individuais (comandos, operadores, variáveis, etc), que são verificados de acordo com as regras da linguagem, gerando erros se for encontrada alguma incorreção

Compilação

Análise Sintática

- Consiste basicamente na criação das estruturas de cada comando, na verificação da correção dessas estruturas e na alimentação da tabela de símbolos com as informações geradas

- Monta uma árvore de análise (parse tree) para o programa fonte por inteiro de acordo com as regras gramaticais da linguagem

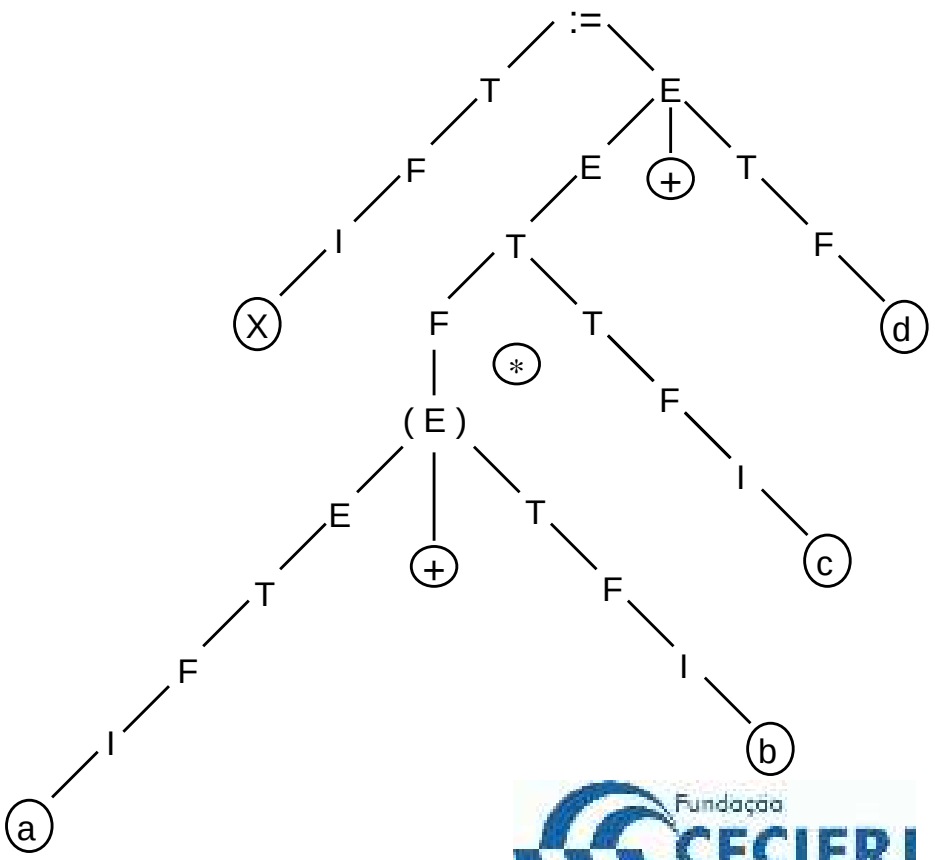
Compilação

Análise Sintática

a) Comando: X:= (a + b) ∗ c + d ;

b) Regras gramaticais		Abreviatura	
Expressão	⟶ Expressão + Termo	E	⟶ E + T
Expressão	⟶ Termo	E	⟶ T
Termo	⟶ Fator ∗ Termo	T	⟶ F ∗ T
Fator	⟶ (Expressão)	F	⟶ (E)
Termo	⟶ Fator	T	⟶ F
Fator	⟶ Identificador	F	⟶ I

c) Árvore de análise



(Fig. 9.8 do livro texto)

Compilação

Análise Sintática

- Com a árvore, é permitida a criação de código de máquina e verificação da correção do comando conforme as regras gramaticais especificadas na definição da linguagem

- A tabela de símbolos contém entradas para cada identificador e cada literal usado no programa fonte. Ex: os nomes das variáveis terão como atributo seu tipo, isto é se é inteiro, se é fracionário, etc... De modo que a operação aritmética a ser realizada com ele siga o algoritmo apropriado.

Compilação

Análise Semântica

- Verifica as regras semânticas estáticas, podendo produzir mensagens de erros

- Regras semânticas estáticas: regras que podem ser verificadas durante o processo de compilação, não é necessária a execução

Exemplo: atribuição de dado em uma expressão. O tipo de dado tem que ser coerente com o que foi declarado

Compilação

- Módulo front-end (analisadores léxico, sintático e semântico) cria tabela de símbolos

- Módulo back-end aloca espaço de memória, define que registradores serão usados e que dados serão armazenados nele, gerando o código objeto em linguagem de máquina

Compilação

Exemplo:

Trecho de data division de programa em Cobol:

77 A PIC 9(6)

77 B PIC 9(6)

77 C PIC 9(6)

Nome	Tipo	Tamanho	Endereço
A	Inteiro	4	1000
B	Inteiro	4	1004
C	Inteiro	4	1008

(Fig. 9.9 do livro texto)

Compilação

Considerações:

- O código objeto pode ser absoluto, endereços reais de memória, ou relocável, endereços são relativos ao início do programa, transformando-se em endereços reais apenas na execução

- Na prática os compiladores ignoram muitas das instruções de máquina existentes, sendo este fato uma das desvantagens alegadas para máquinas CISC em relação às arquiteturas RISC

Ligação ou Linkedição

- O código binário para algumas operações já existe armazenado no sistema. Exemplos: Comandos de entrada/saída, rotinas matemáticas especiais como seno
- Rotinas externas são organizadas em arquivos que constituem diretórios chamados de bibliotecas (libraries)
- Ligador ou linkeditor: faz a interpretação à chamada a uma rotina e respectiva conexão entre o código objeto principal e o código da rotina

Ligação ou Linkedição

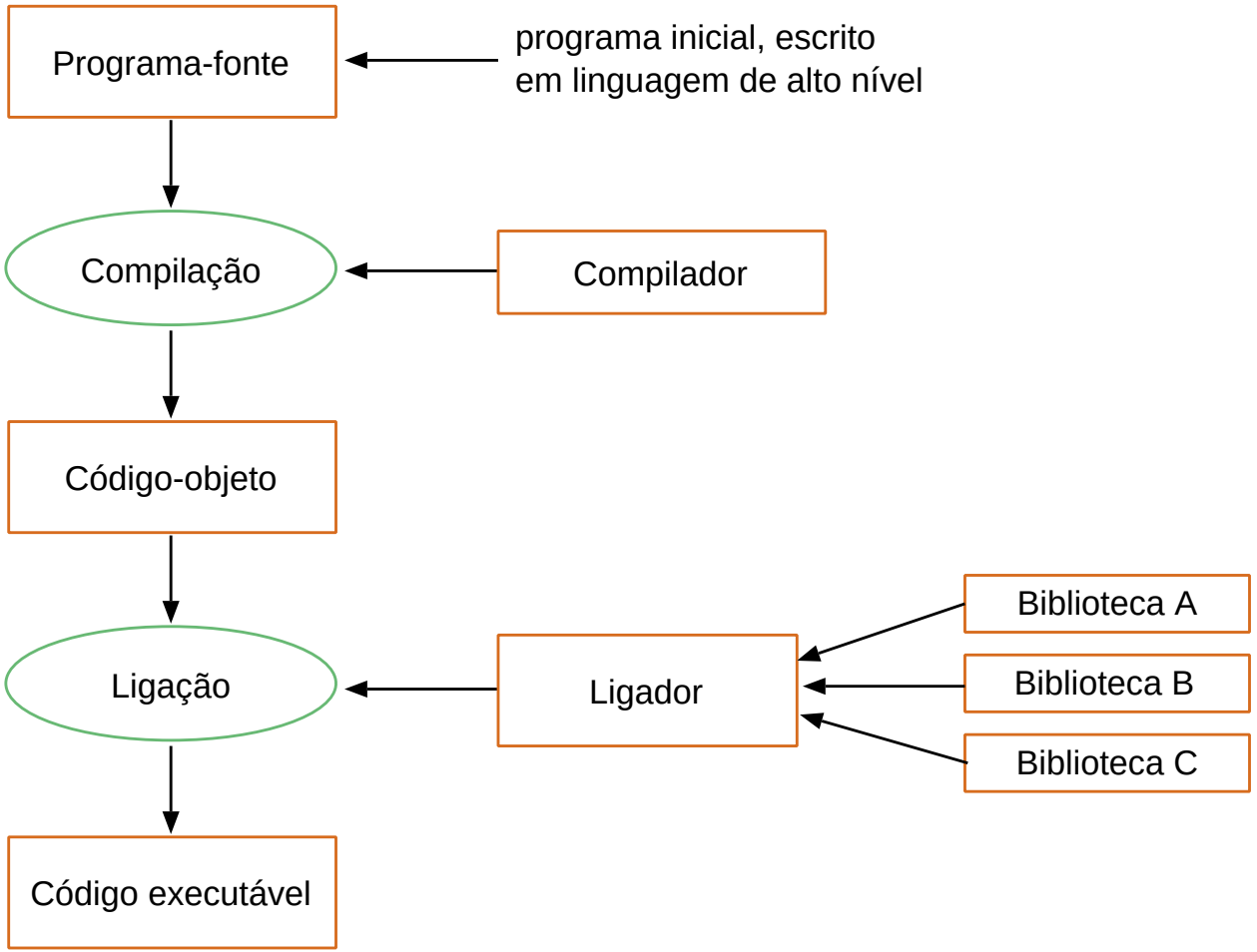
Considere o seguinte comando PASCAL:

X : = A * B

- Suponha que A, B e X sejam números representados em ponto flutuante e que o compilador somente suporte aritmética de ponto-flutuante via rotina externa. O compilador irá inserir:

  CALL MPY_FP(1AB5, 1AB9, 1ABF), onde MPY_FP é um índice para biblioteca, onde se encontra um programa já compilado e os números entre parênteses são endereços de A, B e X

Ligação ou Linkedição



(Fig. 9.10 do livro texto)

Ligação ou Linkedição

Ligador:

- examina o código-objeto
- procura referências externas não resolvidas
- procura suas localizações nas bibliotecas
- substitui a linha de chamada pelo código da rotina
- emite mensagem de erro em caso de não encontrar a rotina

Ligação ou Linkedição

Execução de Programa:

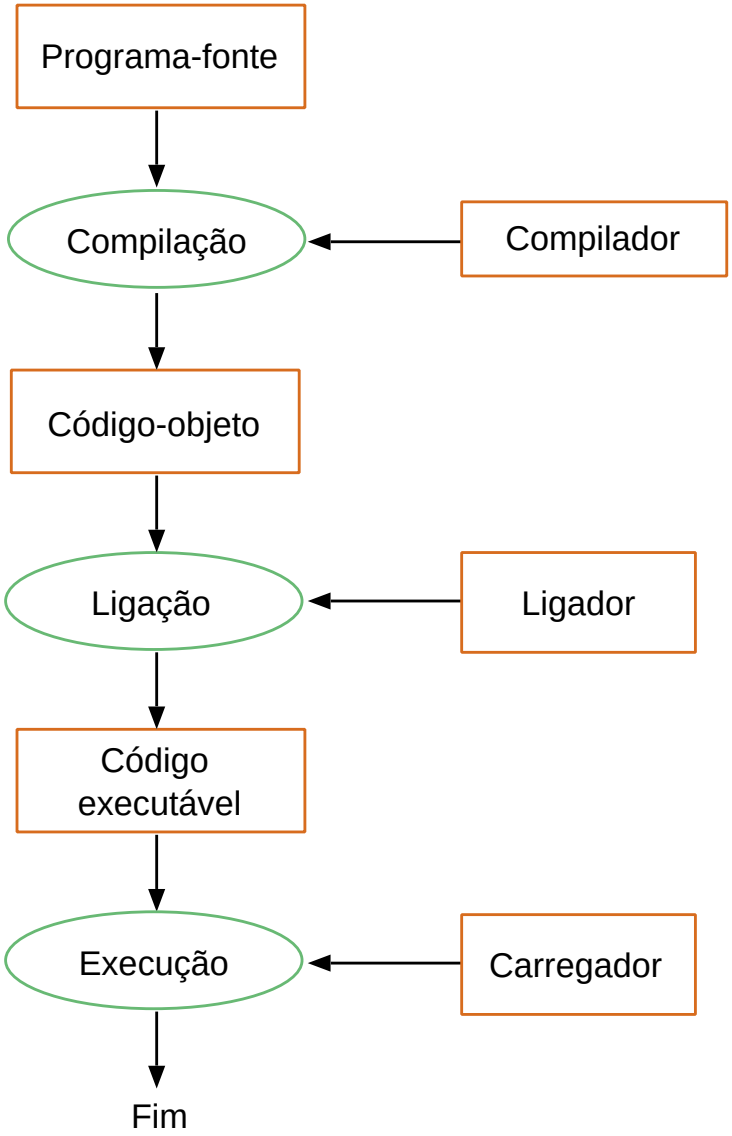
1. Armazenamento na MP do código fonte e compilador
2. O compilador gera o código objeto
3. Este arquivo pode ficar em memória secundária para ser ligado mais tarde ou ser imediatamente carregado na memória com o ligador
4. Após a ligação está formado o executável que pode ficar em memória secundária para posterior execução ou em MP para execução imediata.

Ligação ou Linkedição

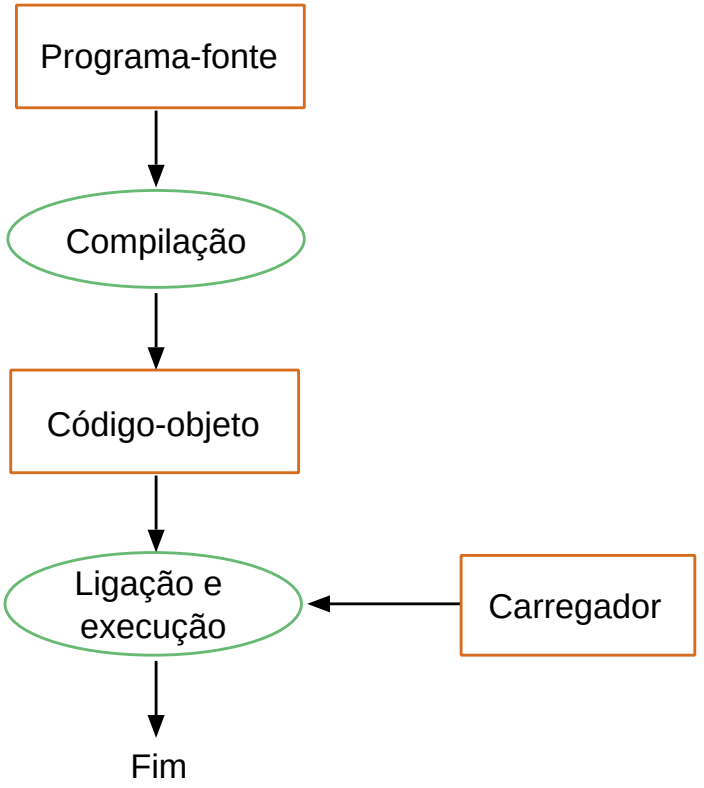
Loader:

- Realiza em seqüência imediata as duas tarefas - ligação e execução do código de máquina, sem geração de código executável permanente
- Cria o executável sem armazená-lo e imediatamente inicia a execução

Ligação ou Linkedição



(a) Com programas compilador, ligador e carregador distintos



(b) Com programas compilador e carregador apenas

(Fig. 9.11 do livro texto)

Interpretação

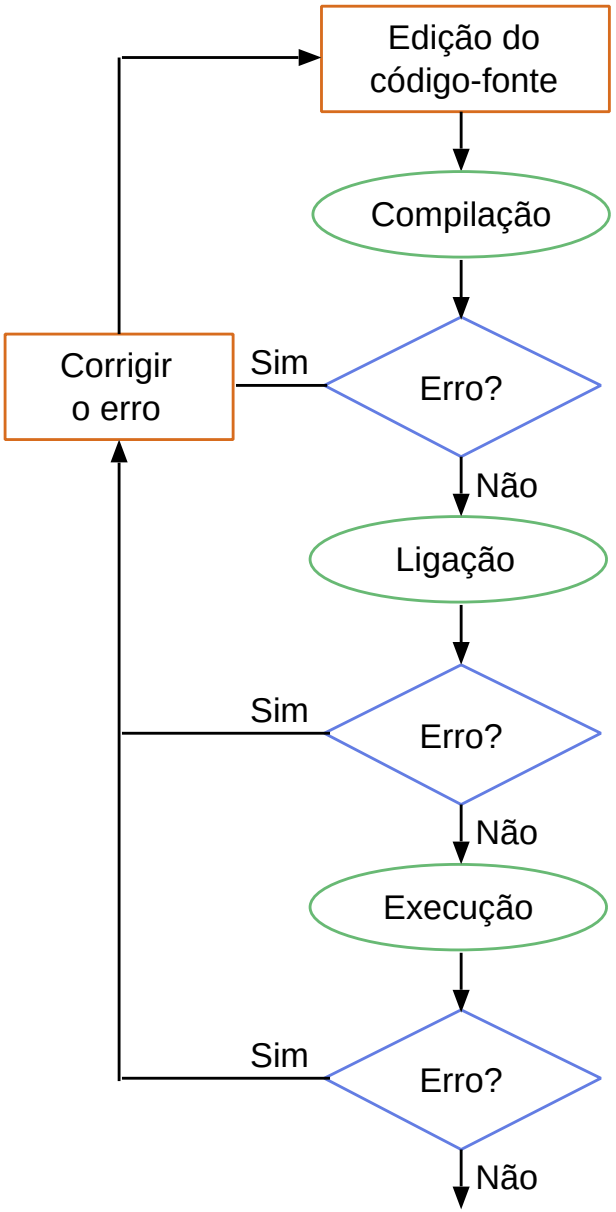
- Realiza as 3 fases, compilação, ligação e execução, comando a comando do programa-fonte
- Não há um processo explícito de compilação e ligação
- Não há produtos intermediários: código-objeto e código executável
- Um programa-fonte é diretamente interpretado pelo interpretador e produz o resultado
- Cada comando do código fonte é lido pelo interpretador, convertido em código executável e imediatamente executado antes do próximo comando

Compilação X Interpretação

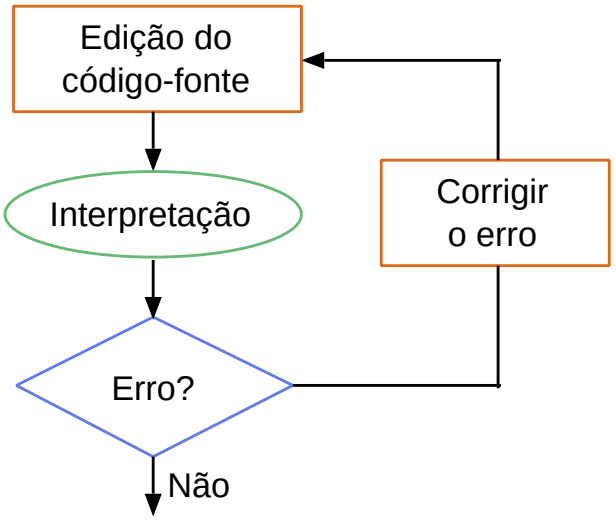
Recursos	Compilação	Interpretação
Uso da memória (durante a execução)		
- Interpretador ou compilador	Não	Sim
- Código-fonte	Não	Parcial
- Código executável	Sim	Parcial
- Rotinas de bibliotecas	Só as necessárias	Todas
Instruções de máquina (durante a execução)		
- Operações de tradução	Não	Sim
- Ligação de bibliotecas	Não	Sim
- Programa de aplicação	Sim	Sim

(Tabela 9.2 do livro texto)

Compilação X Interpretação



(a) Compilação



(b) Interpretação

(Fig. 9.12 do livro texto)

Compilação X Interpretação

Exemplo:

```
FOR J=1 TO 1000
    BEGIN
        READ A,B;
        X := A + B;
        PRINT X;
    END
END
```

Compilação X Interpretação

Compilação:

- os três comandos dentro do loop serão convertidos uma única vez para código executável
- Identificação do erro em tempo de execução é mais difícil

Interpretação:

- 1000 conversões do fonte para o executável, consumindo mais CPU.
- Outro problema: programas executados freqüentemente (ex: folha de pagamento), tradução a cada execução

Execução de Programas em Código de Máquina

```
Início do programa principal
X recebe 1
Y recebe 2
Z recebe a soma de X com Y
Se Z maior que zero
                Então: Z recebe o Quadrado (X)
                Senão: Z recebe o Quadrado (Y)
Fim do Se
Fim do programa principal

Função do Quadrado:
                Início função do Quadrado
                Retorne o produto do parâmetro da função com ele mesmo
Fim da função do Quadrado
```

Figura 9.13(a) Pseudocódigo de um algoritmo.

Execução de Programas em Código de Máquina

EXAMPLE1.PAS

```
(*
* EXAMPLE1.PAS          Exemplo em linguagem pascal
*)

Program Example1;
Var
x, y, z   : integer;
function func (w: integer) : integer;
Begin
        Func:= w * w;
End;
Begin
        x:= 1;
        y:= 2;
        z:= x+y;
        if (z > 0)
          then z:= func (x)
        else z:= func (y);
End
```

Figura 9.13(b) Programa em Turbo Pascal Versão 5.0, que implementa o algoritmo da Fig. 9.13(a)

Execução de Programas em Código de Máquina

```
int func (int w);

void main (void)
{
        int x, y, z;

x = 1;
y = 2;
z = x + y;
if (z > 0)
        z = func (x);
else z = func (y);
}
int func (int w);
{
return (w * w);
}
```

Figura 9.13(c) Implementação do algoritmo da Fig. 9.13(a) em Turbo C++.

Execução de Programas em Código de Máquina

```
[EX1_PAS.ASM]
_func: function func (w: integer): integer;
cs: 0000        55              PUSH    BP
cs: 0001        89E5            MOV     BP, SP
cs: 0003        B80200          MOV     AX, 0002
cs: 0006        9A4402800D      CALL    0D80:0244 # Rotina auxiliar de inicialização
cs: 000B        83EC02          SUB     SP, +02
cs: 000E        8B4604          MOV     AX, [BP + 04] # func: = w * w;
cs: 0011        F76E04          IMUL    WORD PTR [BP + 04]
cs: 0014        8946FE          MOV     [BP-02], AX
cs: 0017        8B46FE          MOV     AX, [BP-02]
cs: 001A        89EC            MOV     SP, BP
cs: 001C        5D              POP     BP
cs: 001D        C20200          RET     0002
_Example1: Program  Example1;
cs: 0020        9A0000800D      CALL    0D80:0000 # Rotina auxiliar de inicialização
cs: 0025        55              PUSH    BP
cs: 0026        89E5            MOV     BP, SP
cs: 0028        C7063E000100    MOV     WORD PTR [003E], 0001 # x: = 1
cs: 002E        C70640000200    MOV     WORD PTR [0040], 0002 # y: = 2
cs: 0034        A13E00          MOV     AX, [003E] # z: = x + y;
cs: 0037        03064000        ADD     AX, [0040]
cs: 003B        A34200          MOV     [0042], AX
cs: 003E        833E420000      CMP     WORD PTR [0042], + 00 # if (z > 0)
cs: 0043        7E0C            JLE     0051
cs: 0045        FF363E00        PUSH    [003E] # then z: = func (x)
cs: 0049        E8B4FF          CALL    _func (0000)
cs: 004C        A34200          MOV     [0042], AX
cs: 004F        EB0A            JMP     005B
cs: 0051        FF364000        PUSH    [0040] # else z: func (y);
cs: 0055        E8A8FF          CALL    _func (0000)
cs: 0058        A34200          MOV     [0042], AX
cs: 005B        89EC            MOV     SP, BP
cs: 005D        5D              POP     BP
cs: 005E        31C0            XOR     AX, AX
cs: 0060        9AD800800D      CALL    0D80:00DB # Rotina auxiliar de finalização
```

Figura 9.13(ab) Programa gerado pelo compilador Pascal relativo à compilação do programa mostrado na Fig. 9.13(b).

Execução de Programas em Código de Máquina

```
                                        EX1_C.ASM
_main: void main (void)
cs: 02C2        55          PUSH        BP
cs: 02C3        88RC        MOV         BP, SP
cs: 02C5        83EC02      SUB         SP, + 02
cs: 02C8        56          PUSH        SI
cs: 02C9        57          PUSH        DI
# EXAMPLE1#11:  x = 1;
cs: 02CA        BF0100      MOV         DI, 0001
#EXAMPLE1#12:   y = 2;
cs: 02CD        C746FE0200  MOV         WORD PTR [BP - 02], 0002
#EXAMPLE1#13:   z = x + y;
cs: 02D2        8BC7        MOV         AX, DI
cs: 02D4        0346FE      ADD         AX, [BP - 02]
cs: 02D7        8BF0        MOV         SI, AX
#EXAMPLE1#14:   if (z > 0)
cs: 02D9        0BF6        OR          Si, SI
cs: 02DB        7E03        JLE         #EXAMPLE#16  (02E0)
#EXAMPLE1#15:   z = func (x);
cs: 02DD        57          PUSH        DI
cs: 02DE        EB03        JMP         02E3
#EXAMPLE#16:    z = func (y);
cs: 02E0        FF76FE      PUSH        [BP - 02]
cs: 02E3        E80900      CALL        _func (02EF)
cs: 02E6        59          POP         CX
cs: 02E7        8BF0        MOV         SI, X
#EXAMPLE#17: }
cs: 02E9        5F          POP         DI
cs: 02EA        5E          POP         SI
cs: 02EB        8BE5        MOV         SP, BP
cs: 02ED        5D          POP         BP
cs: 02EE        C3          RET
_func: int func (int w)
cs: 02EF        55          PUSH        BP
cs: 02F0        8BEC        MOV         BP, SP
cs: 02F2        8B5E04      MOV         BX, [BP + 04]
#EXAMPLE#21     return (w * w);
cs: 02F5        8BC3        MOV         AX, BX
cs: 02F7        F7EB        IMUL        BX
cs: 02F9        EB00        JMP         02FB
#EXAMPLE1#22: }
cs: 02FB        5D          POP         BP
cs: 02FC        C3          RET
```

Figura 9.13(ac) Programa gerado pelo compilador C relativo à compilação do programa mostrado na Fig. 9.13(c).

Execução de Programas em Código de Máquina

EX1_ASM.ASM

cs: 0100	50	PUSH	AX	#Salva os registradores a serem
cs: 0101	53	PUSH	BX	#usados na rotina
cs: 0102	51	PUSH	CX	
cs: 0103	52	PUSH	DX	
cs: 0104	B90100	MOV	CX, 0001	# x = 1
cs: 0107	BA0200	MOV	DX, 0002	# y = 2
cs: 010A	89C8	MOV	AX, CX	# z = x + y
cs: 010C	01D0	ADD	AX, DX	
cs: 010E	89C3	MOV	BX, AX	
cs: 0110	09DB	OR	BX, BX	#if (z > 0)
cs: 0112	7E03	JLE	0117	
cs: 0114	51	PUSH	CX	#Then x como parâmetro
cs: 0115	EB01	JMP	0118	
cs: 0117	52	PUSH	DX	#Else y como parâmetro
cs: 0118	E80600	CALL	0121	#Chama rotina int func (parâmetro)
cs: 011B	5B	POP	BX	#z = func (parâmetro)
cs: 011C	5A	POP	DX	#Restaura os registradores usados
cs: 011D	59	POP	CX	
cs: 011E	5B	POP	BX	
cs: 011F	58	POP	AX	
cs: 0120	C3	RET		#Fim da rotina
cs: 0121	55	PUSH	BP	#Início da rotina int func (parâmetro).
cs: 0122	89E5	MOV	BP, SP	
cs: 0124	8B4604	MOV	AX, [BP + 04]	#Recebe parâmetro
cs: 0127	F7E8	IMUL	AX	#Calcula o produto
cs: 0129	894604	MOV	[BP + 04], AX	#Prepara o valor do retorno
cs: 012C	5D	POP	BP	
cs: 012D	C3	RET		#Fim da rotina int func (parâmetro).

Figura 9.13(ad) Programa gerado pelo montador do DOS 5.0, relativo à montagem do programa mostrado na Fig. 9.13(c).

Execução de Programas em Código de Máquina

Consideremos a expressão:

$$X = Y + Z - T$$

O programa imprime X se este for diferente de zero

```
REAL        X, Y, Z, T
X:          X + Z - T
IF          X <> 0
THEN         PRINT X
ELSE         END
END
```

(Fig. 9.14 do livro texto)

Execução de Programas em Código de Máquina

```
        ORG
        LDA     Y
        ADD     Z
        SUB     T
        STR     X
        JZ      FIM
        PRT     X
FIM     HLT
```

Figura 9.15 Programa em linguagem de montagem para solucionar a expressão X = Y + Z - T.

Execução de Programas em Código de Máquina

Para converter o referido programa em linguagem de máquina assumiremos:

- O processador/MP utilizado possui as mesmas características definidas no capítulo 6 e as mesmas instruções
- As variáveis usadas no programa são:

Execução de Programas em Código de Máquina

variável	endereço	valor
Y	1F	051
Z	20	03E
T	21	003
X	22	01A

Execução de Programas em Código de Máquina

- O programa está armazenado na MP a partir do endereço 18, e no instante inicial vamos considerar que:

1. CI = 18
2. Os valores armazenados no RI e ACC são da instrução anterior, não importante para o início de nosso programa

Execução de Programas em Código de Máquina

```
        ORG
        LDA     Y
        ADD     Z
        SUB     T
        STR     X
        JZ      FIM
        PRT     X
FIM     HLT
```

(Fig. 9.15 do livro texto)

ENDEREÇOS	CONTEÚDOS
18	11F
19	320
1A	421
1B	222
1C	51E
1D	B22
1E	000
1F	051
20	03E
21	003
22	01A

(Fig. 9.16 do livro texto)

Execução de Programas em Código de Máquina

```
        ORG
        LDA     Y
        ADD     Z
        SUB     T
        STR     X
        JZ      FIM
        PRT     X
FIM     HLT
```

(Fig. 9.15 do livro texto)

ENDEREÇOS	CONTEÚDOS
18	11F
19	320
1A	421
1B	222
1C	51E
1D	B22
1E	000
1F	051
20	03E
21	003
22	01A

(Fig. 9.16 do livro texto)

Instrução 1

CI	RI	ACC
19	11F	051

Próxima Instrução

Voltar

Exercícios

Fazer todos os exercícios do capítulo 9 do livro texto