EDIÇÕES DE OURO
CATEGORIA ÁGUIA
8087

Programação Cobol

Eli Rozendo Moreira dos Santos
Professor de Processamento de Dados
Mestrado de Comunicação-Informática
e Cibernética Pela UFRJ

Programação Cobol

Livrarias

EDIÇÕES DE OURO

RIO DE JANEIRO

Bonsucesso:
- Rua Nova Jerusalém, 345 (sede)

Centro:
- Av. Rio Branco, 156 – loja 4 (Ed. Avenida Central)

Copacabana:
- Rua Santa Clara, 33-D

Jóquei Clube (Jardim Botânico):
- Entre a Rua Jardim Botânico e Marques de São Vicente (Praça Santos-Dumont, 66)

Méier:
- Rua Dias da Cruz, 188 – loja 103

Mariz e Barros:
- Rua Mariz e Barros, 290 (em frente ao Instituto de Educação)

Madureira:
- Rua D. Clara, 11 – loja F (esquina de Domingos Lopes)

NOVA IGUAÇU

- Av. Floriano Peixoto, 1784 – loja 2 (Galeria Fluminense)

NITERÓI

- Rua Gavião Peixoto, 92 – loja 103 (Beco do Príncipe) – Icaraí

DUQUE DE CAXIAS

- Av. Plínio Casado, 58 – loja E (em frente à passarela do mercado)

SÃO PAULO

Centro:
- Rua Conselheiro Crispiniano, 403

Centro:
- Rua Benjamim Constant, 162

Penha:
- Av. Penha de França, 771

BELO HORIZONTE

- Av. Afonso Penna, 1707 (perto do Palácio das Artes)

CURITIBA

- Rua Jesuíno Marcondes, 33 – loja 1 (quase esquina da Pç. Gal. Osório)

PORTO ALEGRE

- Av. Ipiranga, 2821 (em frente ao Supermercado Zaffari)

FORTALEZA

- Rua Major Facundo, 680

RECIFE

- Rua do Hospício, 202 – loja 2 – Ed. Olympia

SALVADOR

- Av. 7 de Setembro, esquina da Rua Politeama de Cima (Mercês)

BRASÍLIA

- Super Center Venâncio 2000 – 1.º Subsolo – loja 42-F

EDIÇÕES DE OURO
Sede: Dep. de Vendas e Expedição
Dep. de Direitos e Exame de Originais
Rua NOVA JERUSALÉM, 345 – Caixa Postal 1880
RIO DE JANEIRO – CEP 20.000

Eli Rozendo Moreira dos Santos

Professor de Processamento de Dados
Mestrado de Comunicação — Informática e
Cibernética pela UFRJ

Programação Cobol

Ilustrações:
Ricardo Meirelles

EDIÇÕES DE OURO

As nossas edições reproduzem
integralmente os textos originais

História ou Estória?

As Edições de Ouro e o Coquetel grafam a palavra ***história*** e não ***estória*** por julgar a primeira forma mais correta, conforme dicionários mais categorizados, que julgam a segunda forma imitação do inglês ***story,*** sem correspondente com raízes em nossa língua.

EDITORA TECNOPRINT LTDA.

À
Neli,
minha esposa,
por todo apoio e colaboração recebidos.

*

* *

NOTA DE RECONHECIMENTO

Conforme determinação da American National Standards Institute (ANSI) reproduzimos abaixo, em inglês, a seguinte nota de reconhecimento:

ACKNOWLEDGMENT

The following extract from Government Printing Office Form Number 1965-0795689 is presented for the information and guidance of the user:

"Any organization interested in reproducing the COBOL report and specification in whole or in part, using ideas taken from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organization are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention 'COBOL' in acknowledgment of the source, but need not quote this entire section.

"COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organization.

"No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the Committee, in connection therewith.

"Procedures have been established for the maitenance of COBOL. Inquiries concerning the procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

"The authors and copyright holders of the copyrighted material used herein

> FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the UNIVAC (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator, Form. No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

have specifically authorized the use of this material in whole or in part, in the COBOL specifications in programming manuals or similar publications."

SUMÁRIO

Capítulo 1 — Generalidades

O que é um programa 15
O que é um programador 15
O que é linguagem de programação 15
Linguagem de programação 16
O que é COBOL 16
O que é um compilador 17
Fluxogramas 17
Os símbolos gráficos do fluxograma 17

Capítulo 2 — Fluxogramas

Exercícios 31

Capítulo 3 — Conceitos Básicos

As divisões do COBOL 43
Conjunto dos caracteres usados no COBOL 44
Regras para formação de palavras em COBOL 45
A folha de codificação 45
Convenções 50

Capítulo 4 — Identification Division

ESTRUTURA DA IDENTIFICATION DIVISION 57

Capítulo 5 — Environment Division

ESTRUTURA DA ENVIRONMENT DIVISION 63
Configuration Section 63
Source-computer 64
Object-computer 64
Special-names 64
Nome-mnemônico 65
Currency sign is 65
Decimal-point is comma 65
Input-Output section 67
File-control 67
Select 68
Assign 68
Reserve 71
Access Mode 72
Actual key 74
Nominal key 74
Record key 75
I-O-Control 75
Rerun 75
Same 76

Capítulo 6 — Data Division

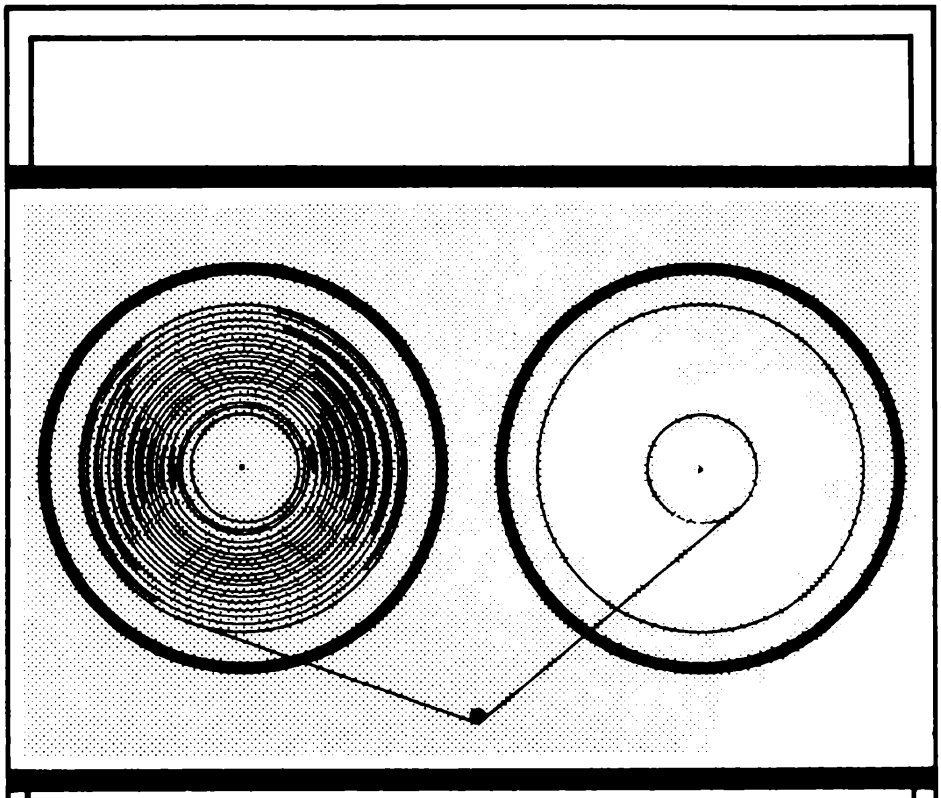
ESTRUTURA DA DATA DIVISION 81
File section 81
File description 82
Block contains 82
Record contains 84
Recording mode 85
Label records 86
Value of 86
Data records 86
Record description 88
Número-de-nível 88
Filler 90
Redefines 91
Blanck when zero 94

Justified 94
Occurs 94
Depending on 96
Picture 97
Edição 102
Synchronized 108
Usage 108
Opção display da cláusula usage 109
Opção computational da cláusula usage 109
Opção computational-1 da cláusula usage 109
Opção computational-2 da cláusula usage 109
Opção computational-3 da cláusula usage 109
Value 110
Literais 111
Constantes figurativas 113
Nome-de-condição 116
Renames 117
WORKING-STORAGE SECTION 119
Número de nível 77 119

Capítulo 7 — Procedure Division

PROCEDURE DIVISION 123
Expressões aritméticas 125
Condicionais 125
Condição de classe 126
Teste de nome-de-condição 126
Condição de relação 127
Condição de sinal 127
Condições compostas 128
Comando IF 128
Comando ADD 130
ROUNDED 132
ON SIZE ERROR 132
Comando compute 134
Comando divide 136
Comando multiply 137
Comando subtract 139
Comando GO TO 141
Comando alter 143
Comando perform 144

Comando stop 147
Comando exit 148
Comando move 150
Comando open 153
Comando read 155
Comando write 156
Comando ACCEPT 159
Comando display 160
Comando close 162
Exercício final 163



CAPÍTULO 1

Generalidades

*

* *

O QUE É UM PROGRAMA

O computador eletrônico é capaz de efetuar operações aritméticas, comparações de valores e tarefas necessárias à execução de um serviço, sem a intervenção do homem. Para isso, no entanto, o computador precisa ser "instruído" previamente. O conjunto de todas as instruções fornecidas ao computador para que ele possa resolver um determinado problema é chamado de *programa.*

O QUE É UM PROGRAMADOR

Alguém precisa indicar ao computador, por meio de um programa, quais são todas as etapas necessárias para resolver um problema. A pessoa que cria o programa, isto é, a pessoa que instrui a máquina, é conhecida como *programador.*

O QUE É LINGUAGEM DE PROGRAMAÇÃO

O computador identifica as instruções que lhe são dadas, por meio de códigos. As regras para utilização desses códigos,

conhecidos como códigos de máquina, são muito complexas, sendo difícil ao programador fornecer as instruções por meio desses códigos de máquina. Para tornar mais fácil, rápida e eficiente a tarefa do programador, as instruções não são dadas ao computador diretamente através desses códigos de máquina. Foram criadas várias formas de instruir o computador, por meio de códigos e regras mais simples do que os códigos de máquina e as regras que regem estes códigos de máquina. O conjunto destes códigos e regras simplificadas é conhecido como linguagem de programação.

Linguagem de programação

Foram criadas várias linguagens de programação. As mais empregadas no Brasil são COBOL, ASSEMBLER e FORTRAN.

O FORTRAN, abreviação de Formula Translator, é muito empregado em processamento científico. O ASSEMBLER tem seu emprego restringido ao tipo de computador para o qual foi feito o programa. Por exemplo, se foi feito um programa em ASSEMBLER para um computador IBM, este programa não poderá ser utilizado em computadores Burroughs.

O COBOL é a mais empregada destas linguagens de programação citadas, entre outros fatores, pela sua flexibilidade em relação aos computadores onde o programa terá de ser executado. Um programa escrito em COBOL para um computador de um determinado fabricante poderá, com poucas alterações, ser processado em um computador de outro fabricante.

O QUE É COBOL

A palavra COBOL é abreviação de COMMON BUSINESS ORIENTED LANGUAGE.

O COBOL é uma linguagem de programação dirigida, principalmente, para processamento de problemas comerciais. Por ser uma linguagem criada para ser utilizada em vários tipos, modelos e marcas de computadores, o programador fica liberado de muitas restrições de máquinas impostas aos programadores de outras linguagens.

O QUE É UM COMPILADOR

A máquina para resolver um problema, como vimos, precisa ser instruída por meio de códigos de máquina.

O programador, para sua facilidade, escreve os programas em linguagem tais como ASSEMBLER, FORTRAN, e COBOL, linguagens simbólicas, as quais não empregam os códigos de máquina. Há necessidade, portanto, de fazer uma tradução do programa escrito numa linguagem simbólica para uma linguagem de máquina, isto é, há necessidade de transformar o programa escrito pelo programador para uma forma que empregue os códigos de máquina.

O programador não precisa se preocupar com esta tradução. O computador faz esta tradução sozinho, pois existem programas cuja função é, exatamente, fazer este tipo de transformação. Estes programas são os "compiladores". Um compilador COBOL, por exemplo, é um programa que tem por função transformar um programa escrito na linguagem COBOL para linguagem de máquina.

FLUXOGRAMAS

Se você está familiarizado com fluxogramas passe para o capítulo 3. Caso contrário, acompanhe as explicações dadas a seguir.

Um programa é composto de muitas instruções. Estas instruções têm de ser fornecidas ao computador numa seqüência lógica rigorosa. Basta, às vezes, uma única instrução, entre milhares, ficar fora de seqüência para que o resultado do programa não seja o desejado pelo programador. Para facilitar o trabalho de colocação das instruções na ordem desejada, foi criada uma linguagem gráfica, chamada *fluxograma.*

OS SÍMBOLOS GRÁFICOS DO FLUXOGRAMA

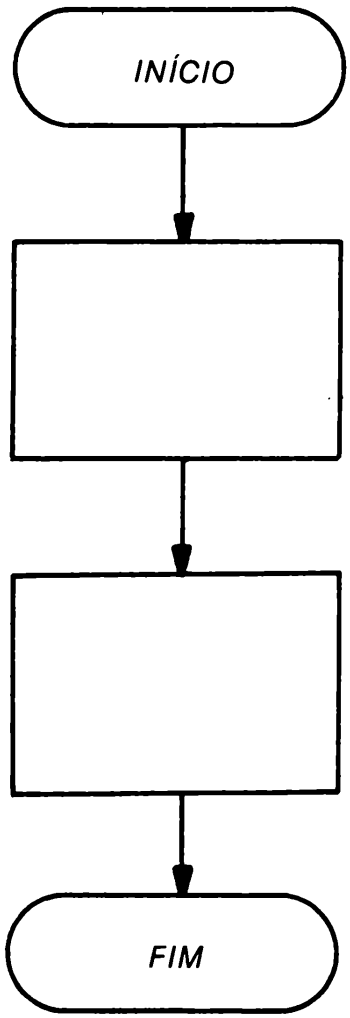
O fluxograma tem símbolos próprios a fim de evitar que cada programador criasse os seus próprios símbolos e convenções. Com a padronização dos símbolos gráficos usados nos

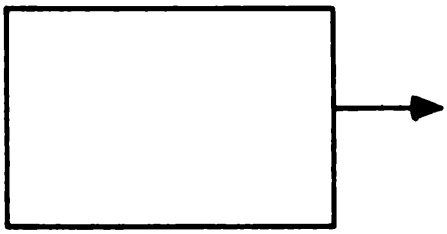
fluxogramas, um programador pode facilmente analisar um fluxograma feito por outro programador. Os fluxogramas não são usados apenas em programação de computador. Eles são também muito empregados em organização e métodos, análise de sistemas etc.

Indicaremos a seguir os símbolos usados nos fluxogramas de programação de computador.

Este símbolo indica início ou fim do fluxograma.

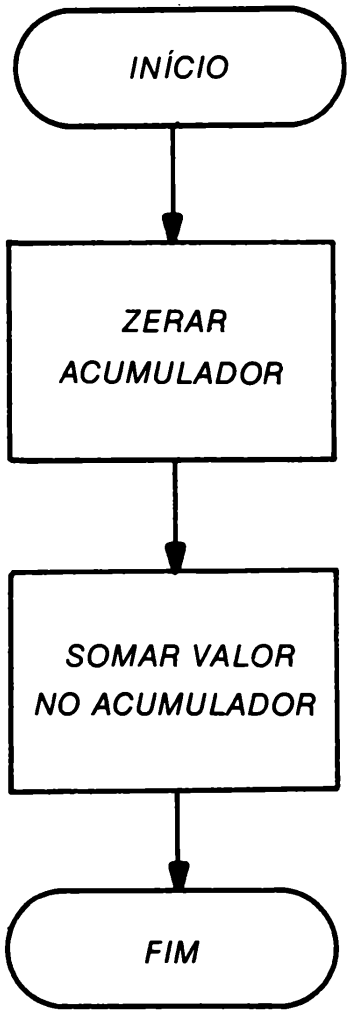
Exemplo:

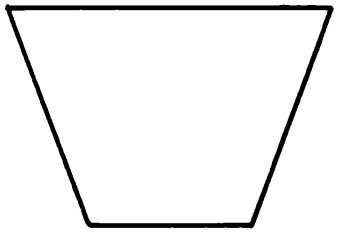




Este símbolo indica um processamento.

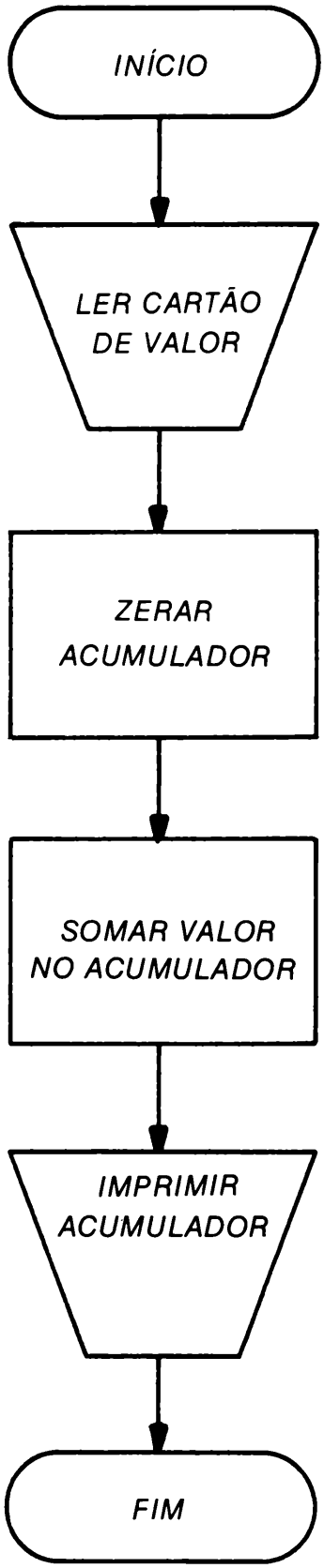
Exemplo:

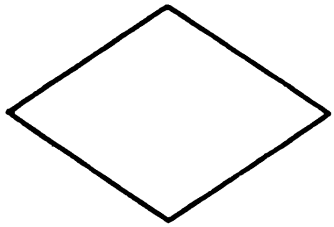




Este símbolo indica uma função de entrada/saída.

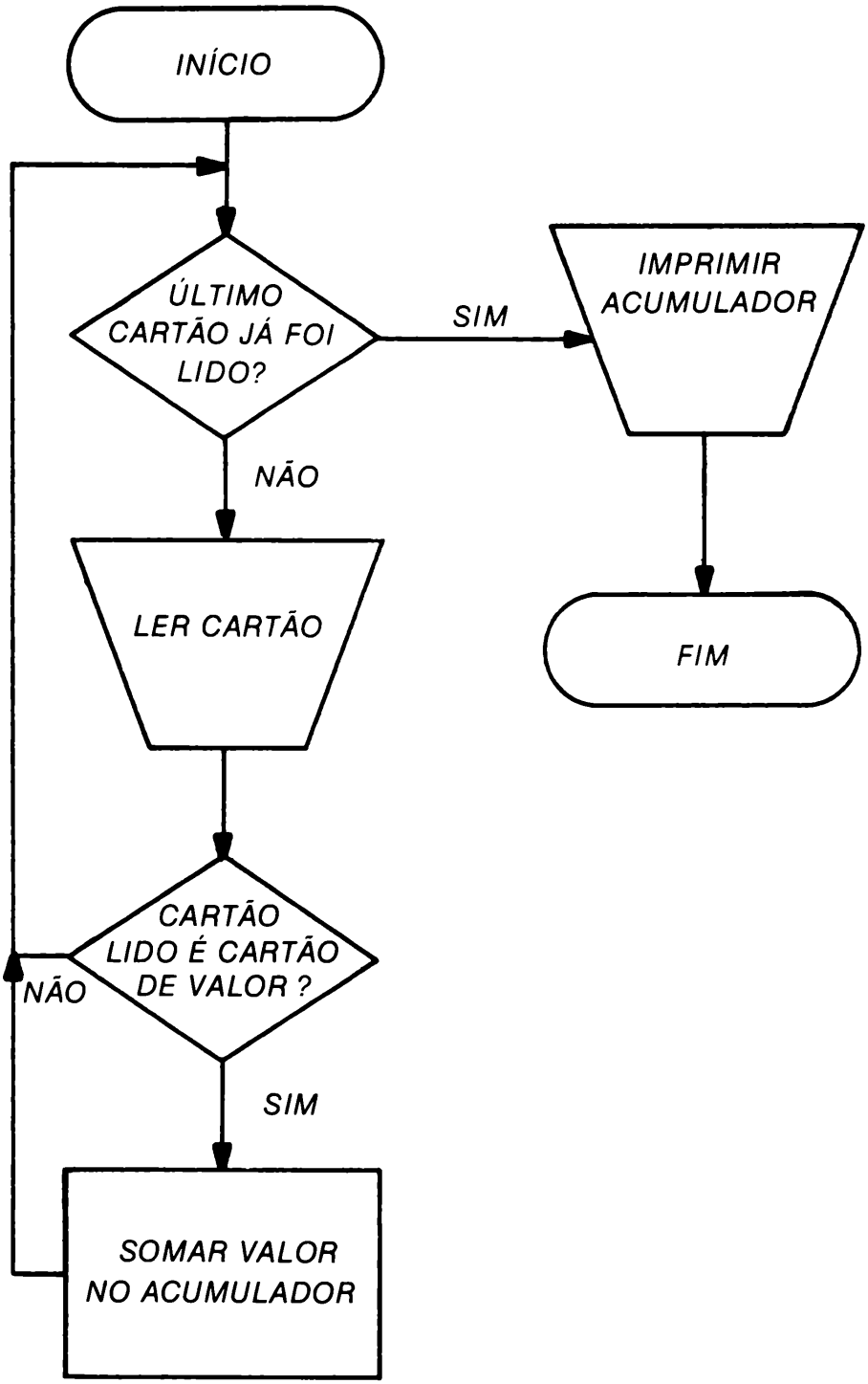
Exemplo:

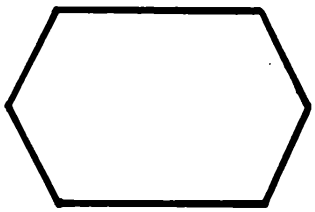




Este símbolo indica uma tomada de decisão.

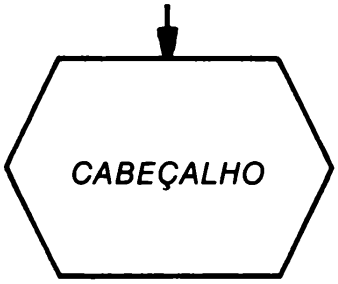
Exemplo:

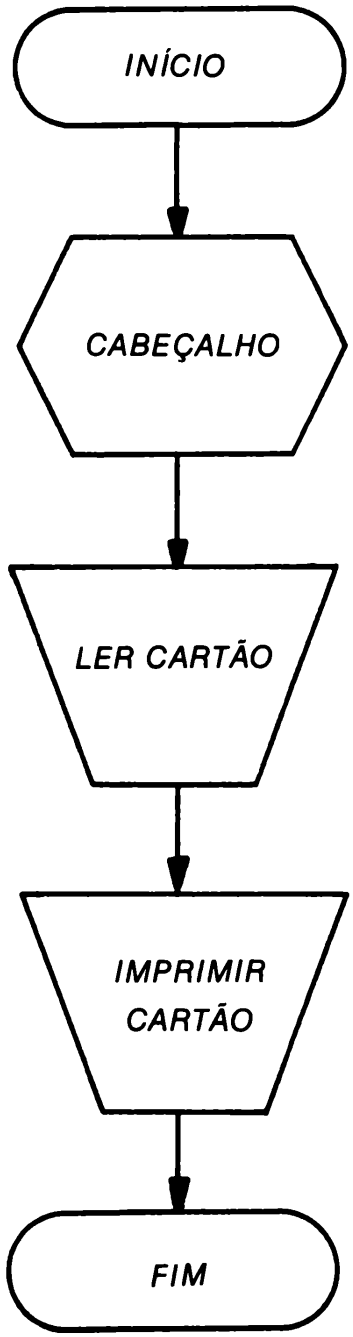


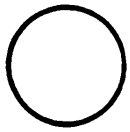


Este símbolo indica uma rotina completa do programa que não foi detalhada no fluxograma. Por exemplo, no fluxograma apresentado abaixo, a rotina de cabeçalho não foi detalhada, mas sabemos que o cabeçalho geralmente não pode ser feito com apenas uma instrução. O cabeçalho implica em numeração de folhas, impressão de datas, nomes de relatórios etc.

O símbolo subentende todas estas etapas.

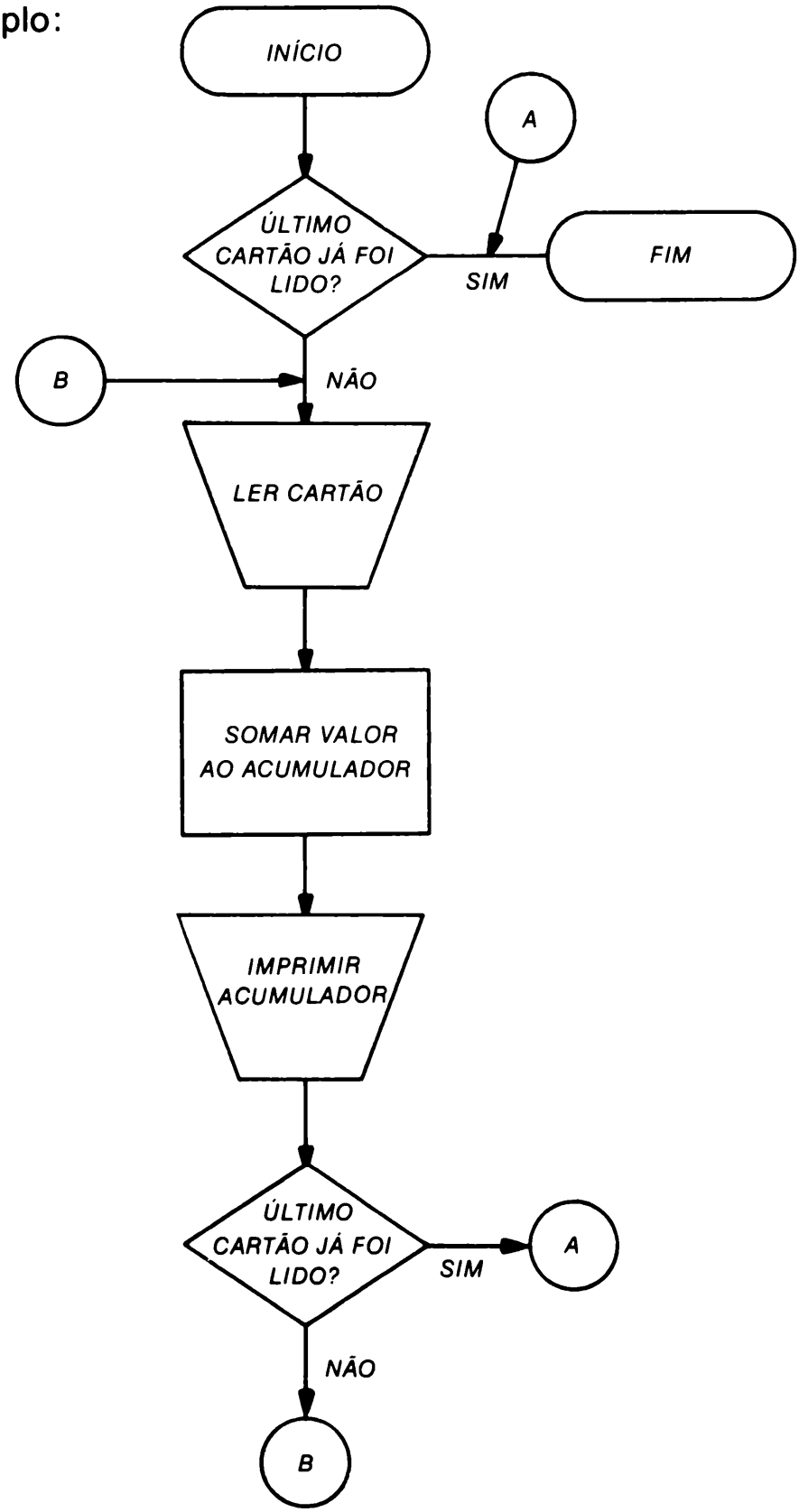


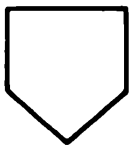




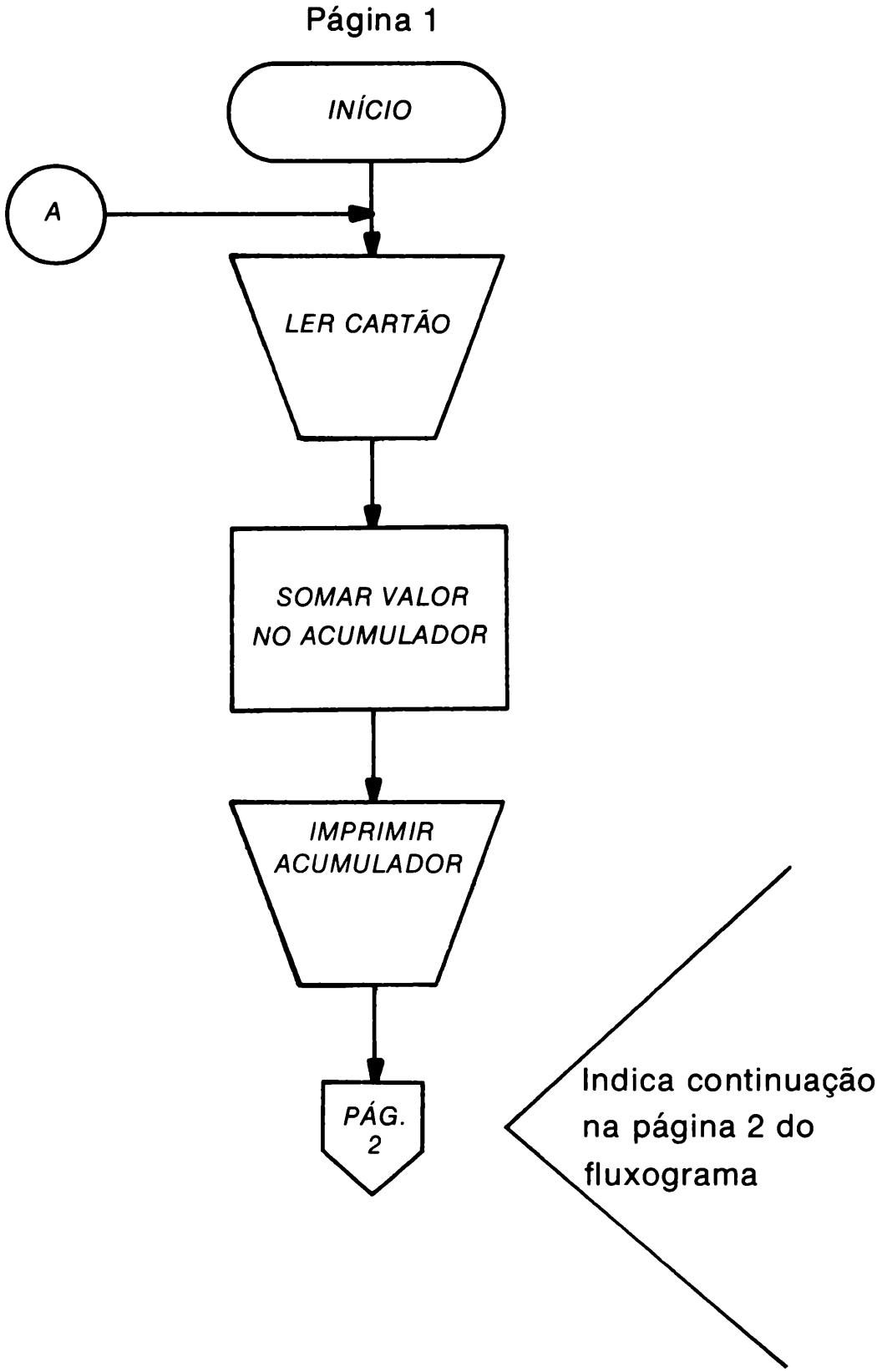
Este símbolo indica uma conexão.

Exemplo:

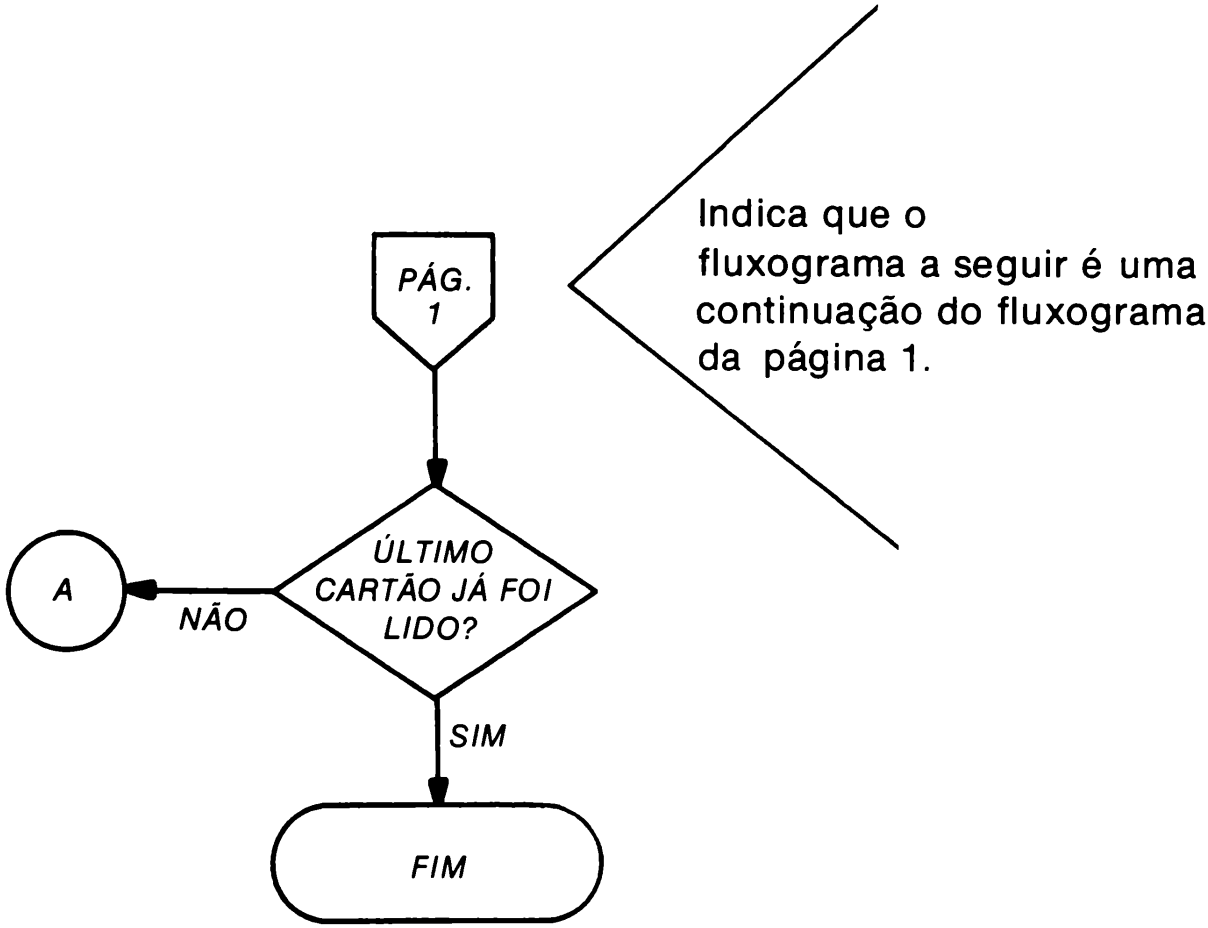


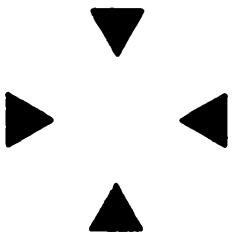


Este símbolo indica saída de uma página ou entrada em uma página.
Exemplo: Suponhamos que a primeira página de um fluxograma fosse a seguinte:



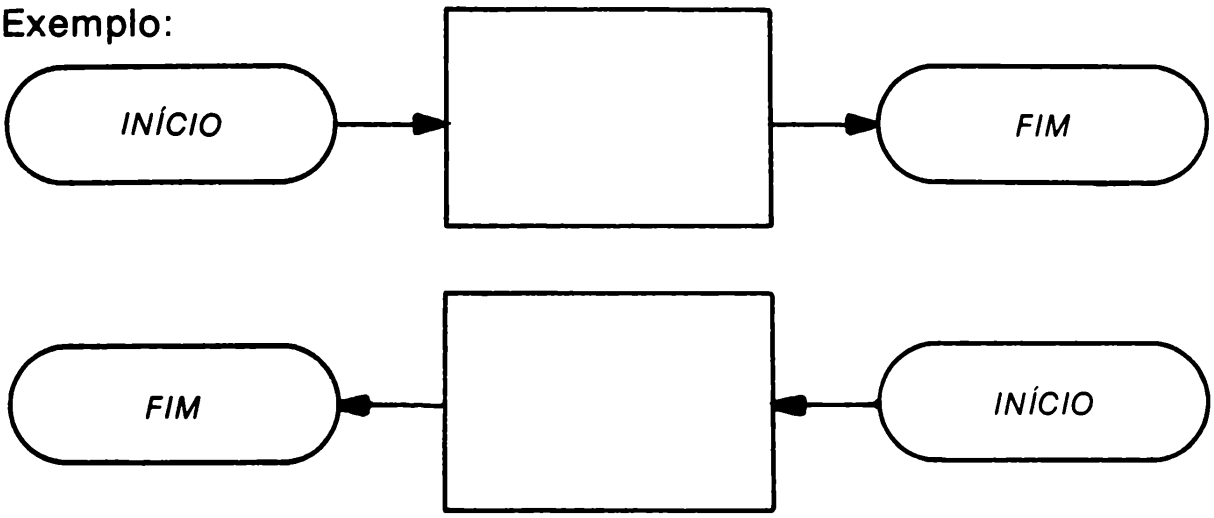
A página 2 do fluxograma no nosso exemplo seria:



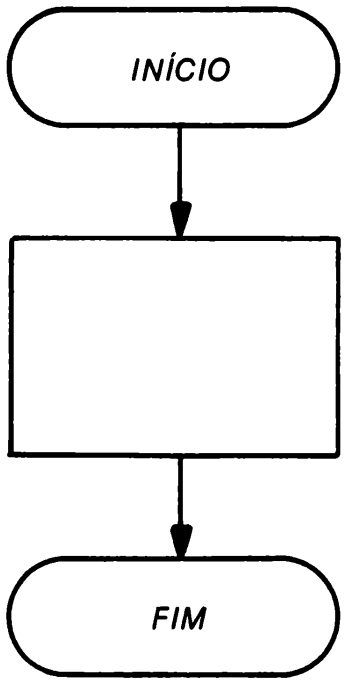


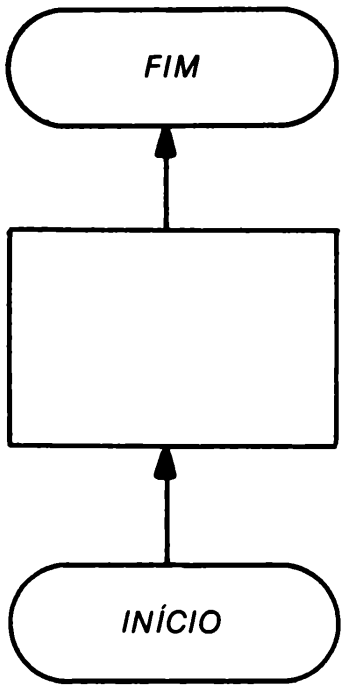
Estes símbolos indicam a direção do fluxo.

Exemplo:

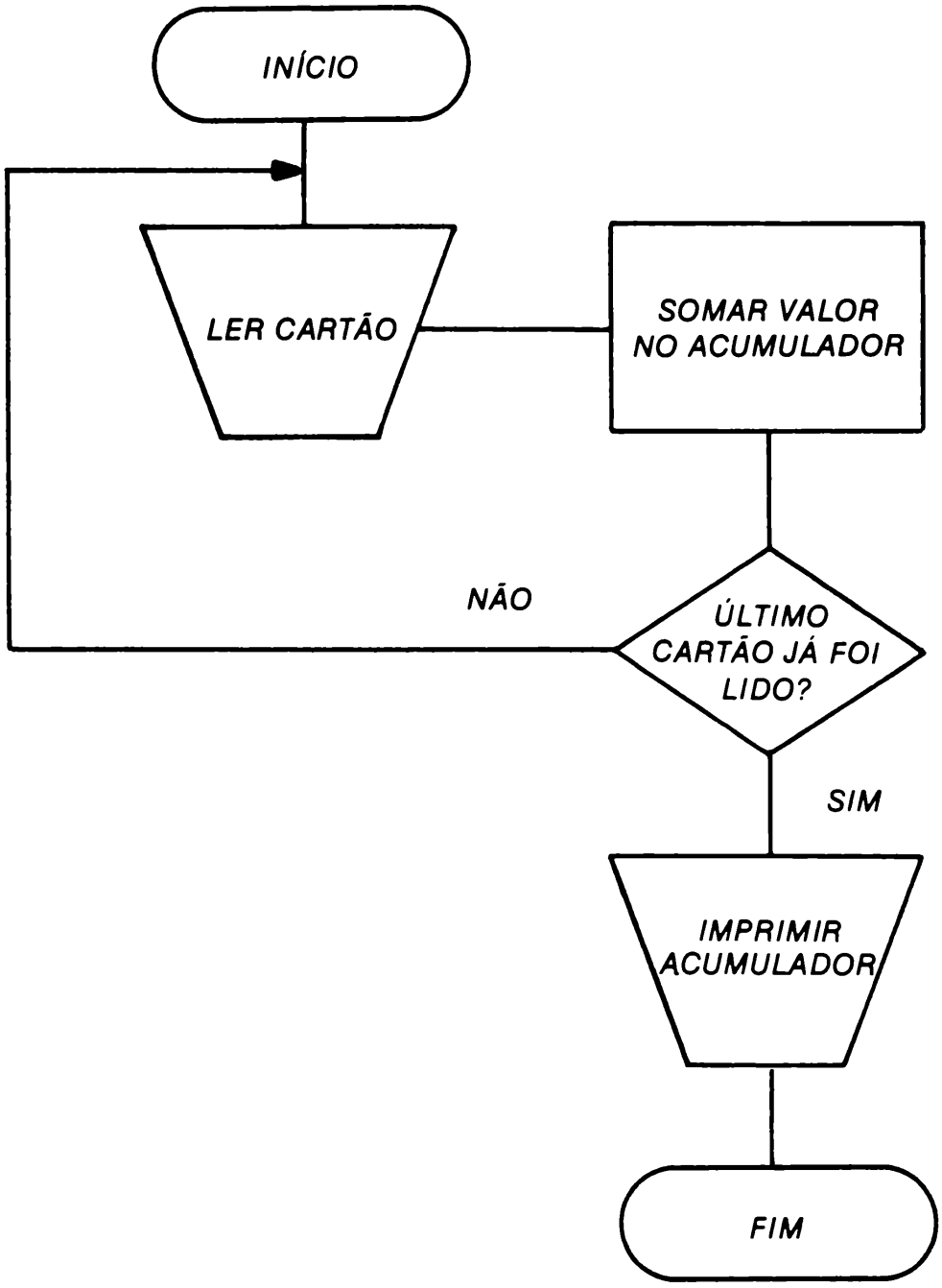


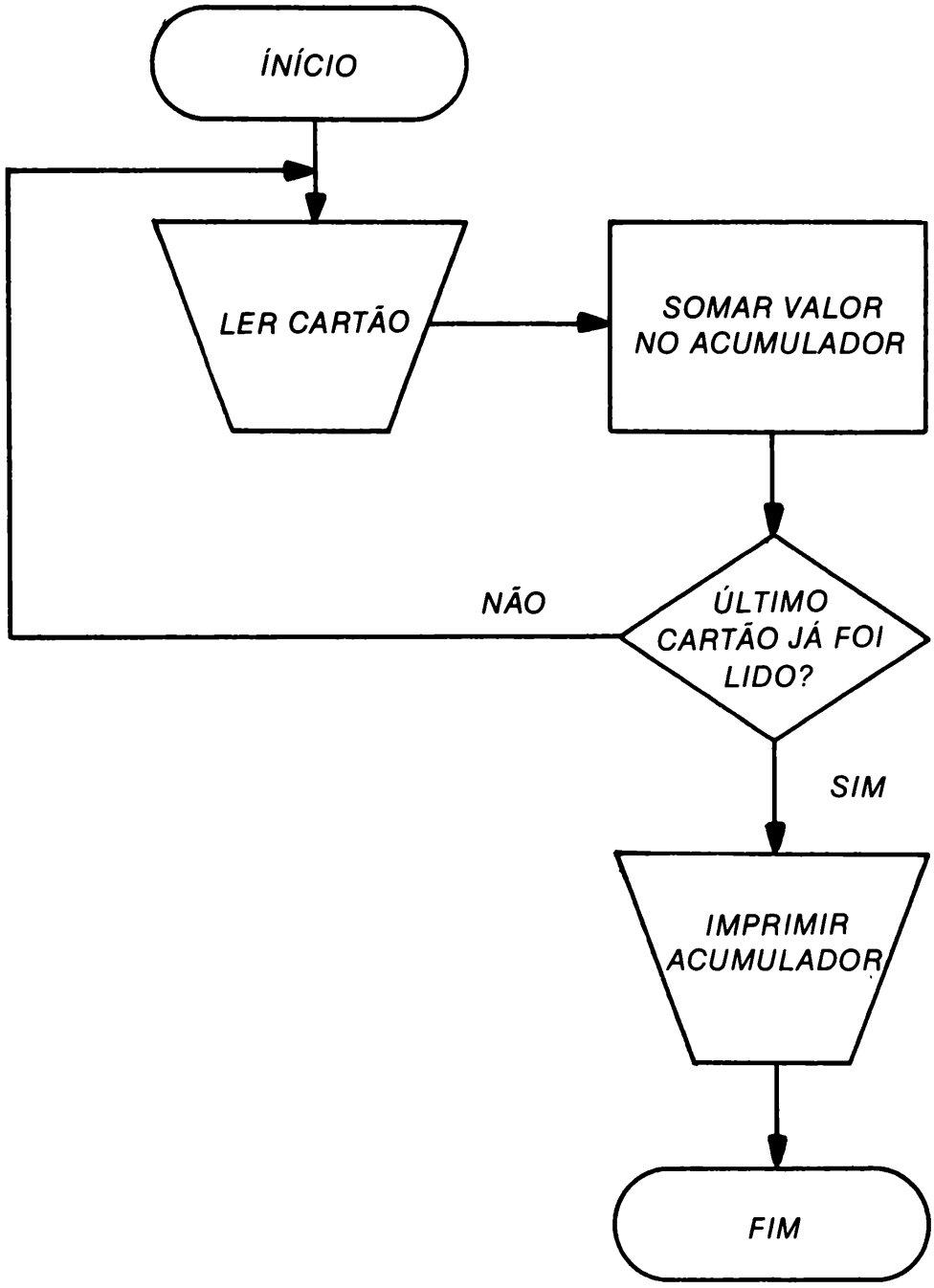
Exemplo:

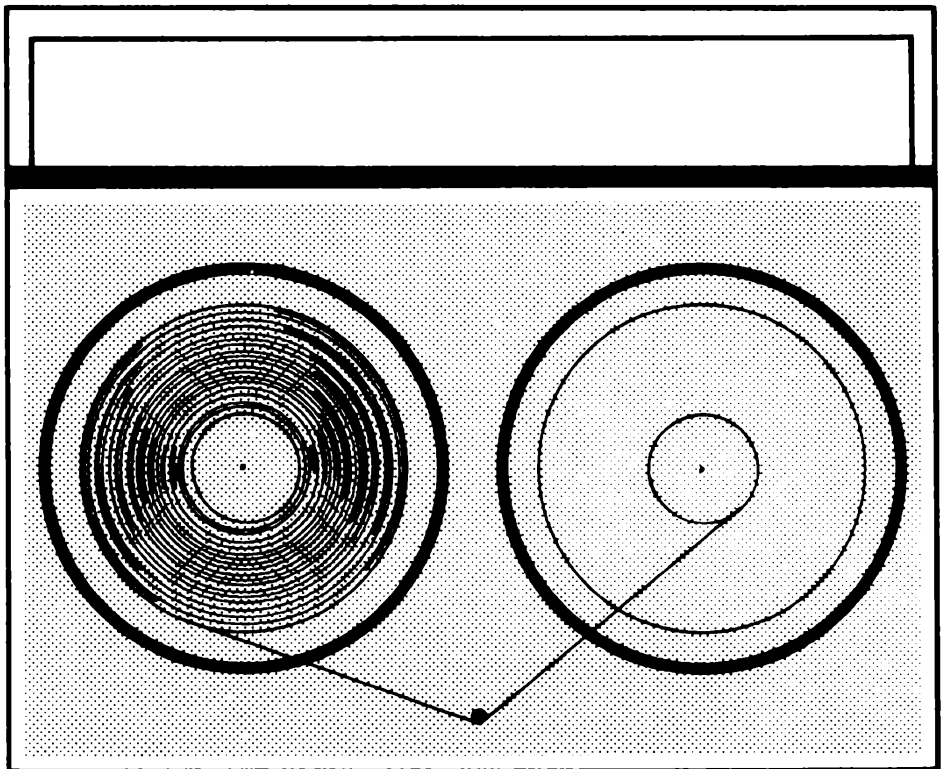




A direção normal do fluxo no fluxograma é da esquerda para a direita ou de cima para baixo. Quando a direção for normal, isto é, da esquerda para a direita ou de cima para baixo, não há necessidade dos símbolos ▶ ou ▼. Assim, os dois fluxogramas a seguir se equivalem.


INÍCIO
LER CARTÃO
SOMAR VALOR NO ACUMULADOR
NÃO
ÚLTIMO CARTÃO JÁ FOI LIDO?
SIM
IMPRIMIR ACUMULADOR
FIM


ÍNÍCIO
LER CARTÃO
SOMAR VALOR NO ACUMULADOR
NÃO
ÚLTIMO CARTÃO JÁ FOI LIDO?
SIM
IMPRIMIR ACUMULADOR
FIM



CAPÍTULO 2

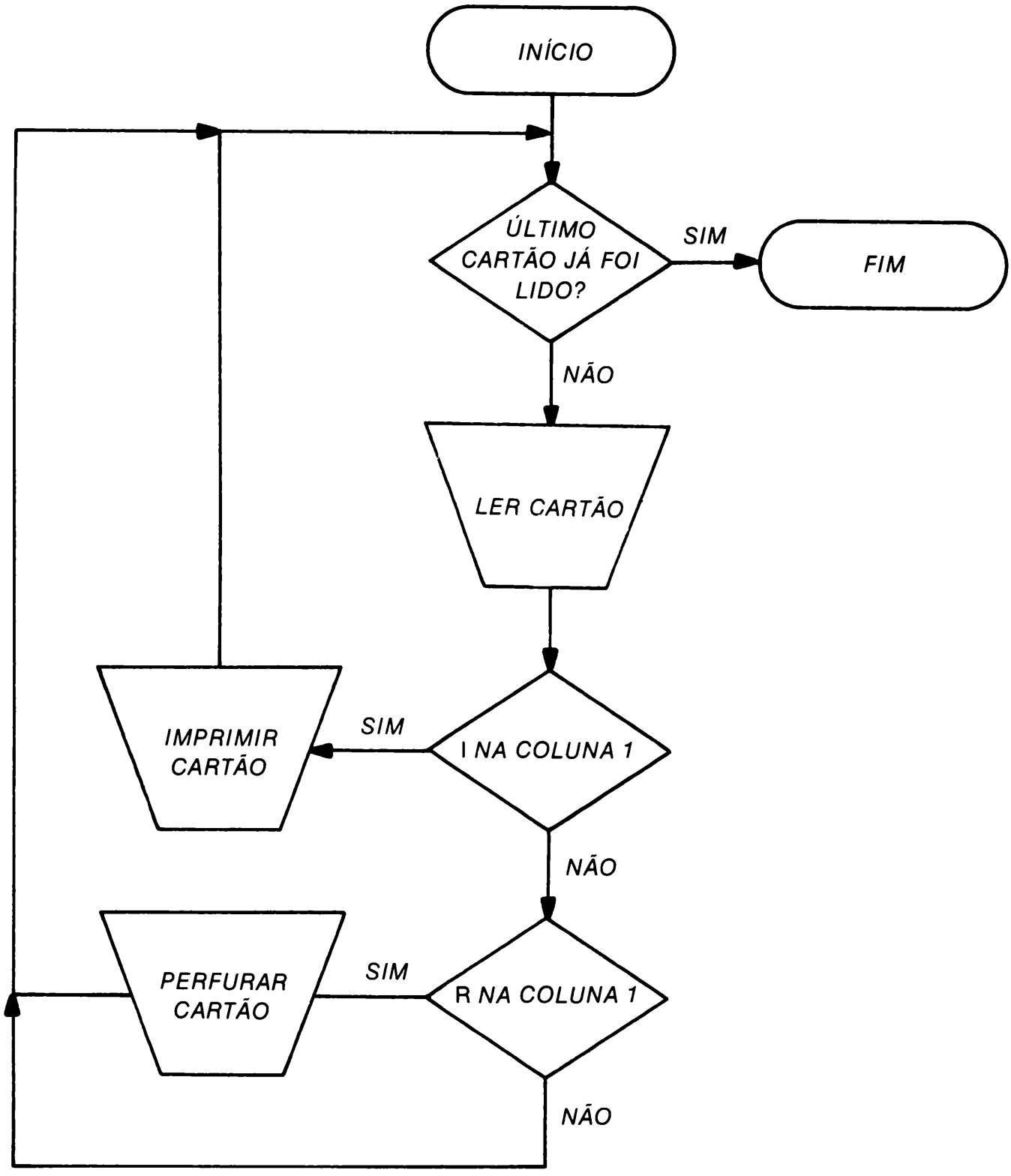
Fluxogramas

*

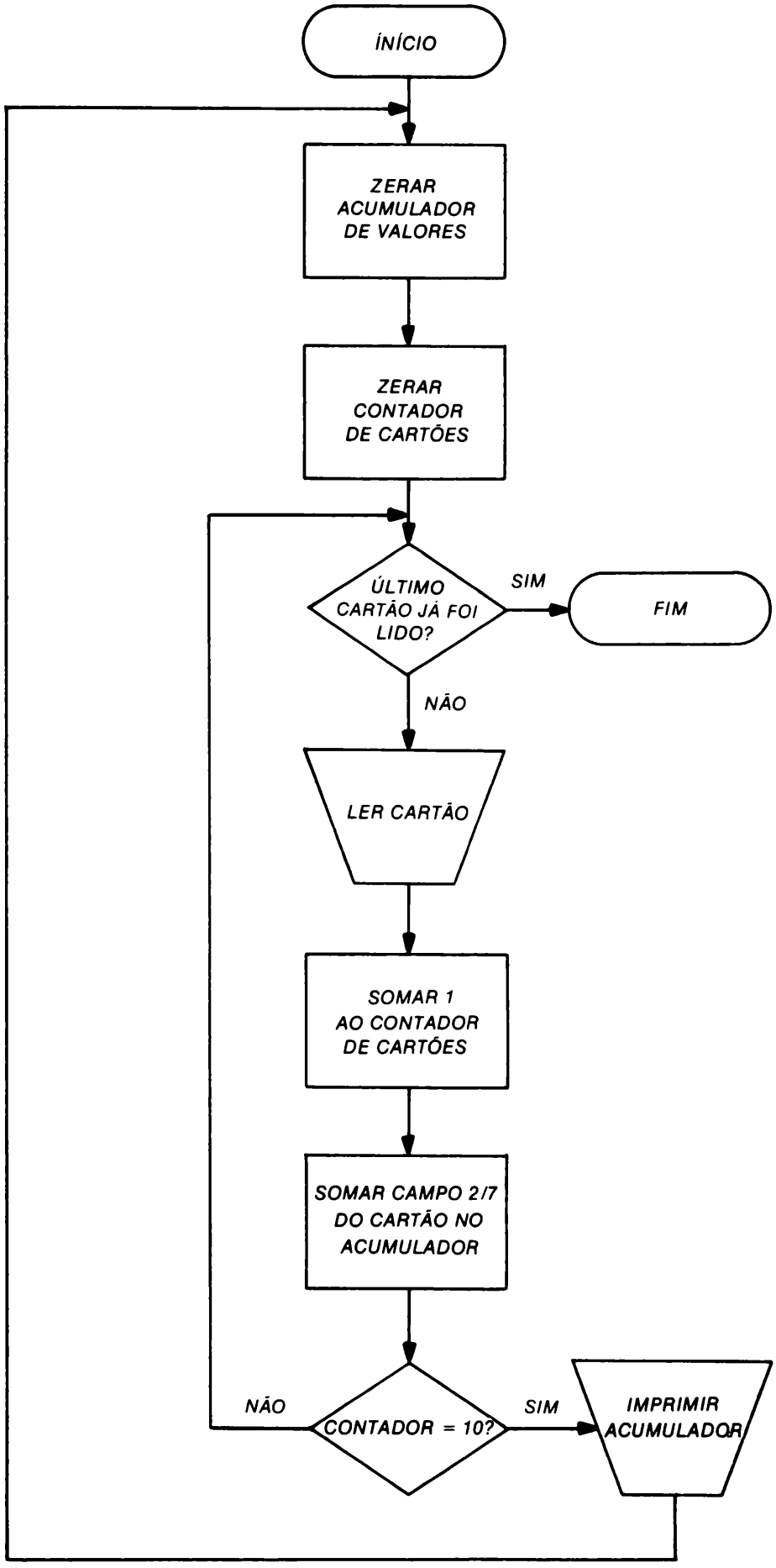
* *

Exercícios

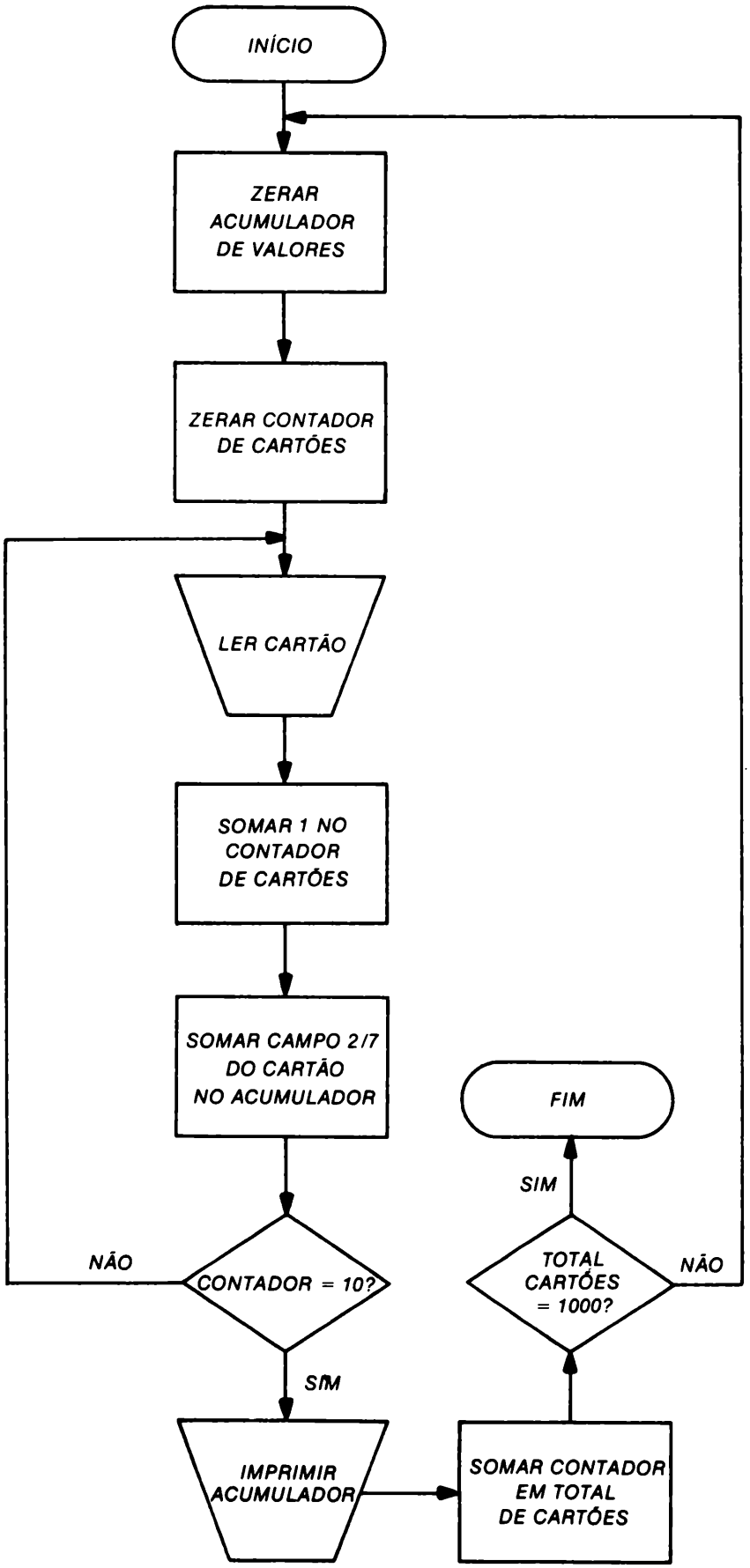
1) Fazer fluxograma do programa abaixo:
Ler uma massa de cartões. Todo cartão é identificado por uma letra perfurada na coluna 1. Quando o cartão tiver uma letra "I" perfurada na coluna 1, imprimir o conteúdo do cartão. Quando a perfuração da coluna 1 for a letra *R*, perfurar outro cartão idêntico ao cartão lido. Não haverá nenhum processamento para os cartões cujas letras perfuradas na coluna 1 forem diferentes de *I* ou *R*.
Compare o seu fluxograma com o apresentado na página seguinte.


INÍCIO
ÚLTIMO CARTÃO JÁ FOI LIDO?
SIM
FIM
NÃO
LER CARTÃO
I NA COLUNA 1
SIM
IMPRIMIR CARTÃO
NÃO
R NA COLUNA 1
SIM
PERFURAR CARTÃO
NÃO

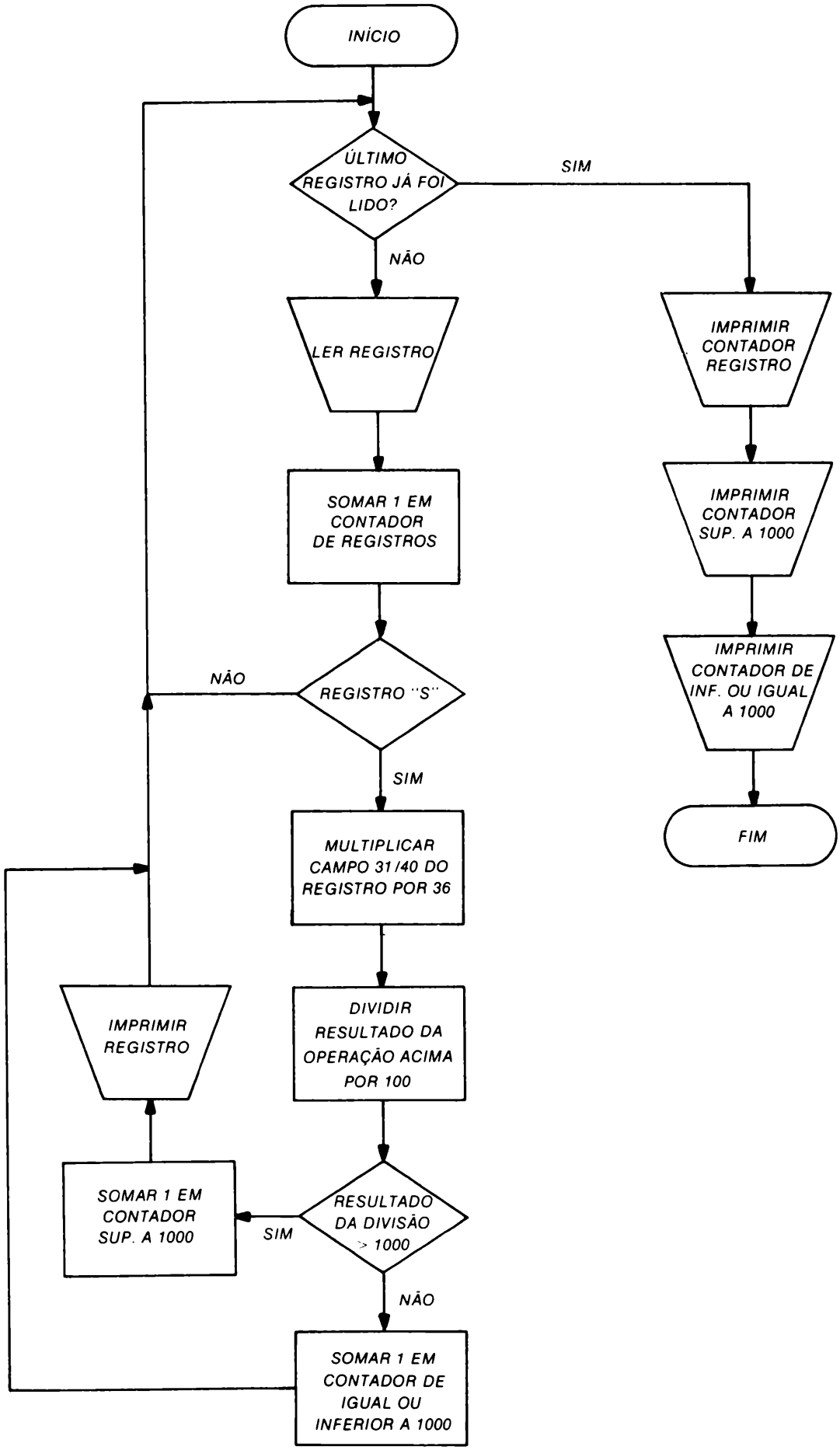
2) Fazer o fluxograma do programa abaixo:
Ler uma massa de 1 000 cartões. A cada 10 cartões lidos, imprimir o total da soma dos valores do campo 2/7 de todos os 10 cartões.
Compare o seu fluxograma com o apresentado na página seguinte.


ÍNICIO
ZERAR ACUMULADOR DE VALORES
ZERAR CONTADOR DE CARTÕES
ÚLTIMO CARTÃO JÁ FOI LIDO?
SIM
FIM
NÃO
LER CARTÃO
SOMAR 1 AO CONTADOR DE CARTÕES
SOMAR CAMPO 2/7 DO CARTÃO NO ACUMULADOR
NÃO
CONTADOR = 10?
SIM
IMPRIMIR ACUMULADOR

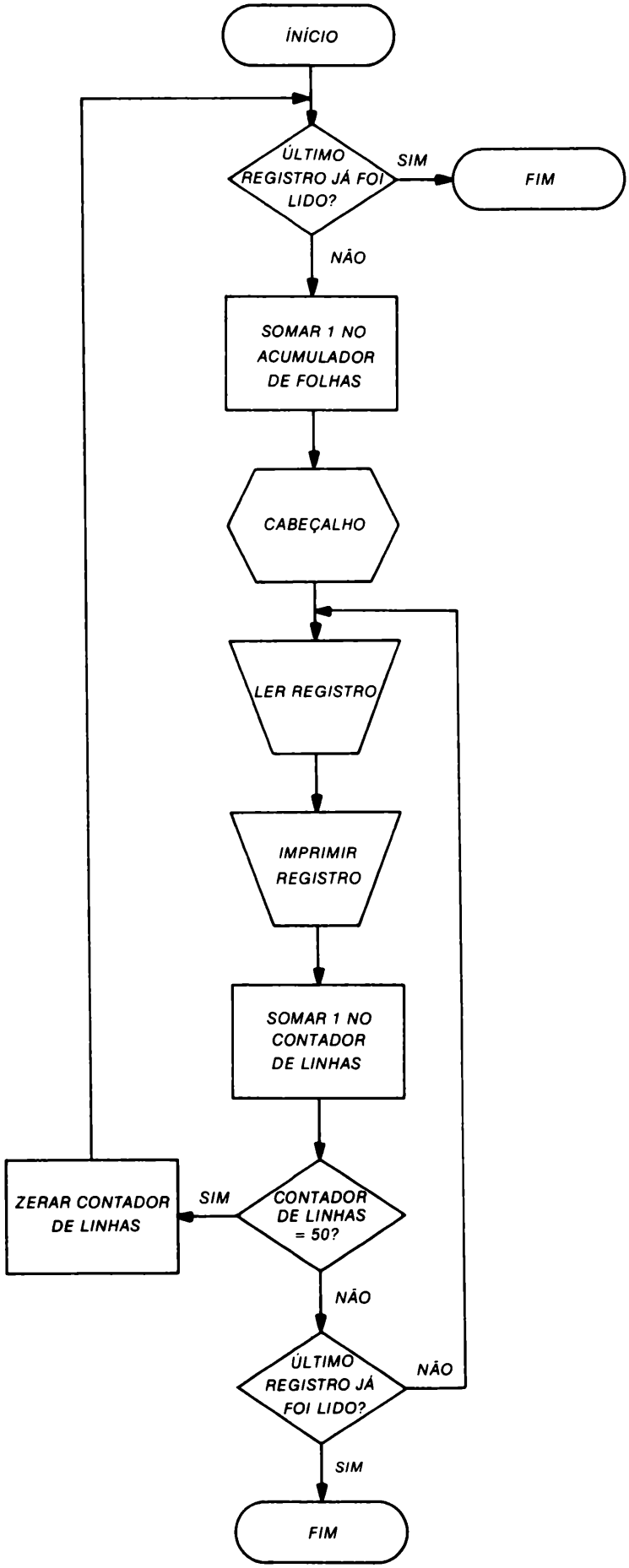
Veja outra solução para o mesmo problema.



3) Fazer o fluxograma do programa abaixo:
Ler um arquivo gravado em fita magnética. O arquivo contém vários tipos de registros, todos identificados por uma letra na posição 30 do registro. Para este programa só interessam os registros que tiverem a letra *S* na posição 30 do registro. Ao encontrar um registro "S", calcular 36% do campo 31/40 do registro. Se o resultado for superior a 1 000 imprimir o conteúdo do registro. No final do processamento imprimir a quantidade total de registros lidos, a quantidade de registros "S" cujo resultado do cálculo dos 36% foi superior a 1 000 e a quantidade de registros "S" cujo resultado do cálculo dos 36% foi igual ou inferior a 1 000.
Compare o seu fluxograma com o da página seguinte.

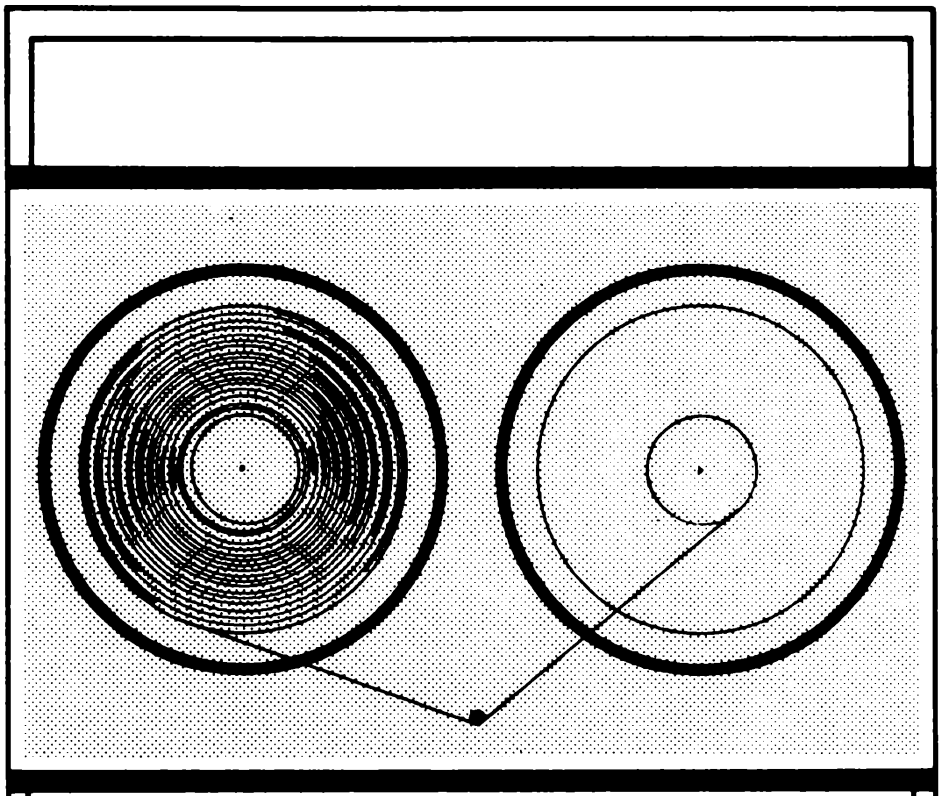

INÍCIO
ÚLTIMO REGISTRO JÁ FOI LIDO?
SIM
NÃO
LER REGISTRO
IMPRIMIR CONTADOR REGISTRO
SOMAR 1 EM CONTADOR DE REGISTROS
IMPRIMIR CONTADOR SUP. A 1000
IMPRIMIR CONTADOR DE INF. OU IGUAL A 1000
NÃO
REGISTRO "S"
SIM
FIM
MULTIPLICAR CAMPO 31/40 DO REGISTRO POR 36
IMPRIMIR REGISTRO
DIVIDIR RESULTADO DA OPERAÇÃO ACIMA POR 100
SOMAR 1 EM CONTADOR SUP. A 1000
SIM
RESULTADO DA DIVISÃO > 1000
NÃO
SOMAR 1 EM CONTADOR DE IGUAL OU INFERIOR A 1000

4) Fazer o fluxograma do programa abaixo:
Ler um arquivo gravado em fita magnética.
Para cada registro da fita magnética imprimir uma linha no relatório. Cada folha do relatório deverá ter 50 linhas impressas. Imprimir, no início de cada folha, um cabeçalho onde deverá constar o número da folha.
Compare o seu fluxograma com o apresentado na página seguinte.


ÍNICIO
ÚLTIMO REGISTRO JÁ FOI LIDO?
SIM
FIM
NÃO
SOMAR 1 NO ACUMULADOR DE FOLHAS
CABEÇALHO
LER REGISTRO
IMPRIMIR REGISTRO
SOMAR 1 NO CONTADOR DE LINHAS
ZERAR CONTADOR DE LINHAS
SIM
CONTADOR DE LINHAS = 50?
NÃO
ÚLTIMO REGISTRO JÁ FOI LIDO?
NÃO
SIM
FIM

*

* *



CAPÍTULO 3

Conceitos Básicos

*

* *

AS DIVISÕES DO COBOL

Um programa COBOL compõe-se de quatro DIVISÕES:
— IDENTIFICATION DIVISION;
— ENVIRONMENT DIVISION;
— DATA DIVISION;
— PROCEDURE DIVISION.

As quatro divisões do programa são escritas na ordem apresentada acima e cada uma delas tem funções específicas.

A IDENTIFICATION DIVISION contém informações que servem para documentar o programa. Tais informações não são traduzidas para linguagem de máquina. Elas são tratadas pelo compilador COBOL como simples comentários.

As informações que podem ser indicadas na IDENTIFICATION DIVISION são:

— nome do programa;
— nome do programador;
— empresa ou CPD usuário do programa;
— data da codificação do programa;
— data da compilação do programa;

— comentários sobre os controles de segurança do programa e/ou arquivos;
— observações gerais sobre o programa.

A ENVIRONMENT DIVISION serve para descrever a configuração do computador onde será feita a compilação do programa, para descrever a configuração do computador onde será executado o programa e também para identificar os arquivos a serem usados no programa e fazer a ligação entre estes arquivos e as unidades de entrada e saída.

A DATA DIVISION serve para:

— fazer uma descrição detalhada de todos os dados a serem usados no programa;
— informar ao compilador os tipos de registros de entrada/ saída;
— associar nomes-de-dados (data-names) para cada item de dados a ser usado;
— informar ao compilador o formato dos arquivos e o formato dos registros usados no programa;
— descrever os registros e itens de dados que não fazem parte dos arquivos de entrada/ saída (áreas de trabalho), mas são usados durante o processamento do programa-objeto.

A PROCEDURE DIVISION contém as instruções necessárias para resolver o problema proposto.

CONJUNTO DOS CARACTERES USADOS NO COBOL

Os 51 caracteres que podem ser usados na codificação de um programa COBOL são:

Dígitos:

0 1 2 3 4 5 6 7 8 9

(O zero geralmente é representado por Ø para diferençá-lo da letra O.)

Letras:

A B C D E F G H I J K
L M N O P Q R S T U V
W X Y Z

Caracteres especiais:

ϕ branco ou espaço
+ sinal de adição
* sinal de multiplicação ou asterisco
/ sinal de divisão ou barra
= sinal de igual
$ sinal de dólar
, vírgula
; ponto-e-vírgula
. ponto
" ou ' aspas ou apóstrofo
(parêntese à esquerda
) parêntese à direita
> símbolo de "maior que"
< símbolo de "menor que"

REGRAS PARA FORMAÇÃO DE PALAVRAS EM COBOL

As palavras em COBOL podem ser formadas pelos caracteres do conjunto: 0 a 9, A a Z e hífen (-).

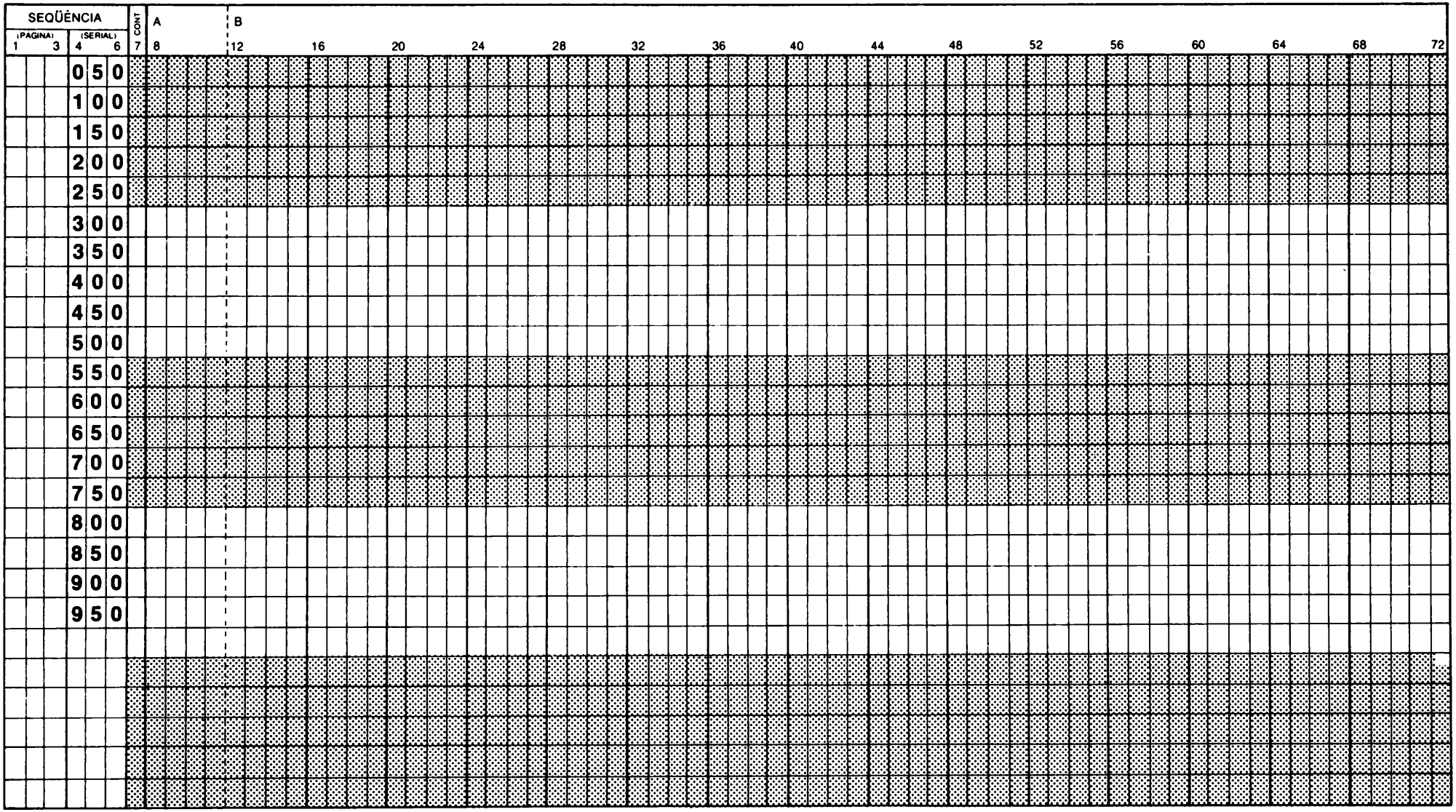
Uma palavra pode ser formada com até 30 caracteres do conjunto acima. Não pode, no entanto, começar nem terminar com hífen (-).

A FOLHA DE CODIFICAÇÃO

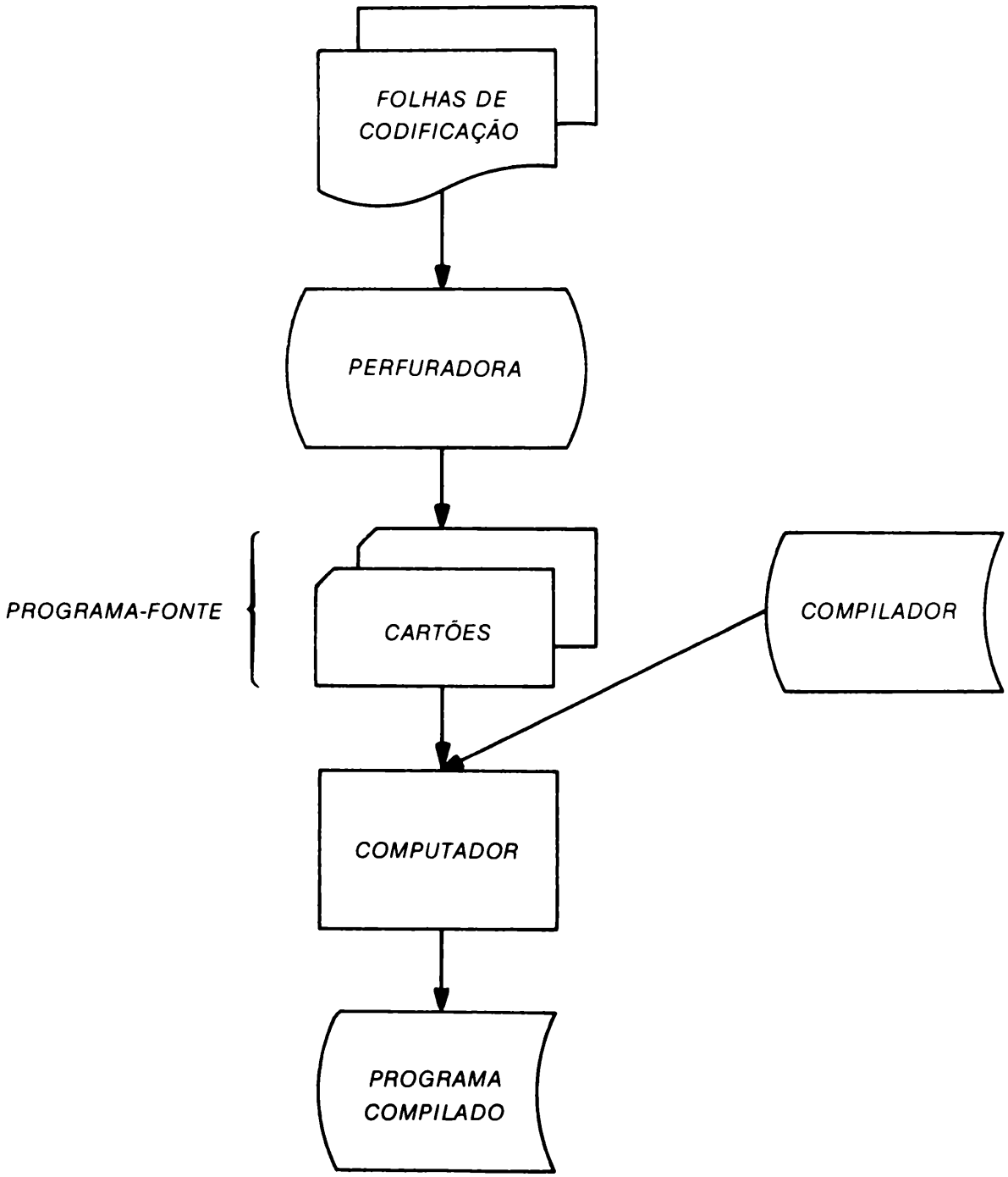
O programador ao codificar o programa em COBOL faz uso de uma folha própria para codificação. Esta folha é conhecida como folha de codificação COBOL. A seguir apresentamos o modelo da folha de codificação.

CODIFICAÇÃO COBOL

SISTEMA:	INSTRUÇÕES DE PERFURAÇÃO:	FOLHA DE
PROGRAMA:		IDENTIFICAÇÃO 73 80
PROGRAMADOR: DATA:		

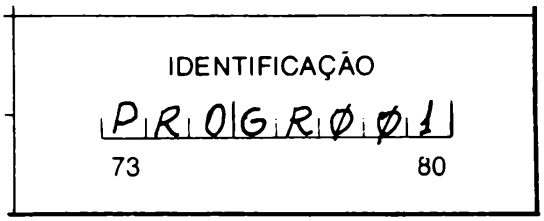


A folha de codificação é dividida em várias linhas e cada linha é dividida em 80 retângulos. O motivo da divisão da linha em 80 retângulos é que será perfurado um cartão para cada linha do formulário. Assim, dividindo a linha em 80 retângulos, cada retângulo corresponderá, exatamente, a cada uma das 80 colunas do cartão. Os cartões serão usados como entrada para o computador fazer a compilação. A massa de cartões perfurados é conhecida como programa-fonte COBOL.



Examine o modelo da folha de codificação. Repare que os retângulos 73 a 80 estão no canto superior à direita do formulário. Não há necessidade de repetir os retângulos 73/80 em todas as linhas do formulário, porque os dados a serem perfurados nas colunas 73/80 dos cartões serão sempre os mesmos em todos os cartões. Isto porque as colunas 73/80 serão usadas para a identificação do programa. Logo, a identificação será sempre a mesma em todos os cartões.

Por exemplo, se for desejado identificar o programa pela sigla PROGRØØ1, deveremos escrever esta sigla no campo 73/80 do formulário.



A perfuradora, ao perfurar os cartões, perfurará PROGRØØ1 nas colunas 73/80 de todos os cartões.

Se for encontrado um cartão extraviado, poderemos saber a que programa ele pertence, verificando a identificação perfurada nas colunas 73/80.

Os demais dados do cabeçalho do formulário (sistema, programa, programador, data, instruções de perfuração e folha) não serão perfurados nos cartões. Estes dados do cabeçalho serão usados apenas para documentação do programa e como orientação para a pessoa que for perfurar os cartões.

Colunas 1 a 6

As colunas 1 a 6 serão usadas para perfurar a seqüência dos cartões do programa. Isto porque os cartões deverão ficar numa ordem crescente rigorosa. Os cartões de um programa-fonte COBOL têm de estar, na hora da compilação, na ordem indicada pelo programador. Um ou mais cartões fora de ordem poderão provocar resultados imprevisíveis na execução ou compilação do programa.

As colunas 1 a 3 serão preenchidas com o número da página. Deste modo, todos os cartões correspondentes a uma mesma página terão os mesmos dígitos perfurados nas colunas 1 a 3. As colunas 4 a 6 serão preenchidas de forma a fazer com que os cartões perfurados fiquem numa ordem crescente rigorosa. No modelo de formulário apresentado as colunas 4 a 6 já têm pré-impresso o seu conteúdo. Há, no modelo apresentado, um intervalo de 050 entre a numeração de uma linha e de outra. Este intervalo tem a finalidade de permitir a inclusão de uma outra linha, se necessária, entre duas linhas consecutivas. Por exemplo, se depois de preenchida toda a folha 005 de codificação de um programa, for verificada a necessidade de incluir um comando entre as linhas 005250 e 005300, basta criar uma nova linha cuja seqüência seja qualquer uma maior do que 005250 e menor do que 005300. Após a perfuração dos cartões e antes da compilação do programa, o cartão seria colocado na devida ordem.

Coluna 7

A coluna 7 é usada para indicar linhas de continuação. Em certas situações, que veremos mais adiante, há necessidade de informar ao compilador que a linha seguinte é uma continuação da linha anterior. Nestes casos, será perfurado um hífen (-) na coluna 7 da linha-continuação.

A coluna 7 é usada, também, para indicar que a linha serve apenas como linha de comentário. Neste caso, o programador deve colocar um * (asterisco) na coluna 7 e toda a linha não afetará o programa, servindo apenas como ilustração.

Colunas 8/11

O campo 8/11 é conhecido como "margem A". É importante a fixação deste nome porque vários dados de um programa têm de ser perfurados na margem A. Toda vez que for indicada a necessidade de perfuração na margem A, a perfuração deverá começar em qualquer uma das colunas 8, 9, 10 ou 11.

Colunas 12/72

O campo 12/72 é conhecido como "margem B". É importante a fixação deste nome porque vários dados de um pro-

grama têm de ser perfurados na margem B. Toda vez que for indicada a necessidade de perfuração na margem B, a perfuração deverá começar em qualquer uma das colunas 12 a 72.

CONVENÇÕES

Os diversos elementos de um programa COBOL serão apresentados neste livro em seus formatos básicos.
Exemplo:

FORMATO

PROGRAM-ID. nome-do-programa.

Apresentamos agora as convenções para a compreensão dos formatos.

— *Palavras escritas com letras maiúsculas*
Estas palavras são conhecidas como palavras reservadas.
Todas as palavras de um formato escritas em letras maiúsculas terão de ser escritas pelo programador exatamente como aparecem no formato. Se o programador omitir, acrescentar ou trocar uma letra dessas palavras, provocará um erro no seu programa.

— *Palavras sublinhadas*
Estas palavras são conhecidas como palavras-chaves. Todas as palavras sublinhadas num formato são obrigatórias. Uma palavra reservada (escrita com letra maiúscula num formato) só será obrigatória se estiver sublinhada.

Exemplo:

FORMATO

RECORD KEY IS nome-do-dado.

No formato anterior as palavras RECORD, KEY e IS são palavras reservadas. Mas a única palavra obrigatória é a palavra RECORD porque ela está sublinhada. As palavras KEY e IS são palavras reservadas mas não são obrigatórias. Elas são chamadas de opcionais.

— *Os caracteres* + − < > =

Estes caracteres, quando aparecem num formato, são obrigatórios, mesmo que não estejam sublinhados.

— *Pontuação*

Todo sinal de pontuação quando aparecer num formato será obrigatório. Outros sinais de pontuação, mesmo que não obrigatórios, poderão ser escritos num programa COBOL, com a finalidade de facilitar a leitura e compreensão do programa, desde que as regras abaixo sejam seguidas rigorosamente:

a) Ponto (.), vírgula (,) ou ponto-e-vírgula (;) quando usados não devem ser precedidos de espaço, mas têm de ser seguidos de um espaço.

b) Um parêntese à esquerda não pode ser seguido imediatamente por um espaço; um parêntese à direita não pode ser imediatamente precedido por um espaço.

c) Deve existir pelo menos um espaço entre palavras e/ou expressões dentro de parênteses.

d) Um operador aritmético deve ser precedido e seguido por um espaço.

e) Uma vírgula pode ser usada como separador entre operandos sucessivos de um comando.

f) Uma vírgula ou ponto-e-vírgula podem ser usados para separar uma série de cláusulas.

```
Exemplo: DATA RECORD IS TRANSACTION,
         RECORD CONTAINS 8Ø CHARACTERS;
         LABEL RECORD IS OMITTED.
```

g) Um ponto-e-vírgula pode ser usado para separar uma série de comandos.

```
Exemplo: ADD A TO B;
         SUBTRACT C FROM D.
```

h) A palavra *THEN* pode ser usada para separar uma série de comandos.

```
Exemplo: ADD A TO B THEN
         ADD C TO D.
```

— *Palavras com letras minúsculas*

Todas as palavras escritas com letras minúsculas num formato representam informações que devem ser fornecidas pelo programador.

Exemplo:

FORMATO

PROGRAM-ID. nome-do-programa.

Neste formato, o nome-do-programa deve ser fornecido pelo programador.

— *Colchetes* [ ]

Todos os elementos que estiverem dentro de colchetes num formato são opcionais.

Exemplo:

FORMATO

[*AUTHOR.* [Comentário] ...]

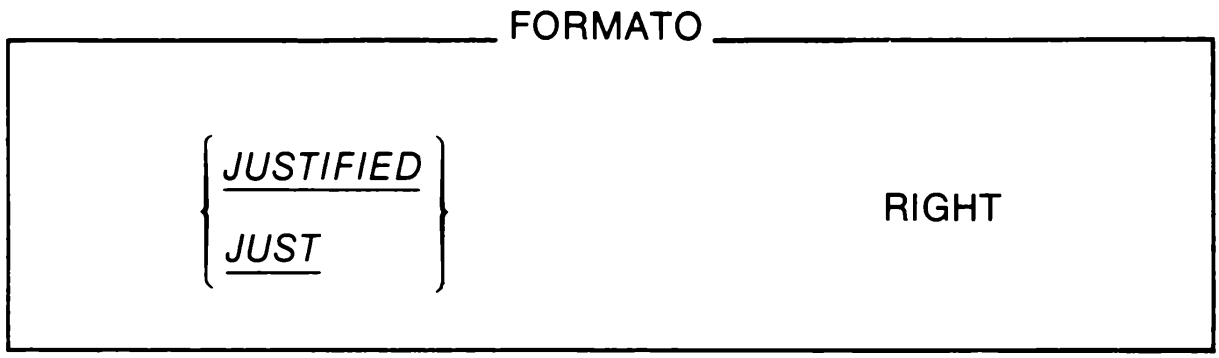
Os colchetes que envolvem a palavra comentário indicam que o comentário, tal como nome do programador, por exemplo, é opcional.

Os colchetes que envolvem todo o parágrafo, isto é, o primeiro e o último colchete, indicam que todo o parágrafo é opcional.

— *Chaves* { }

As chaves servem para indicar que apenas um dos itens que estão envolvidos por elas é obrigatório.

Exemplo:



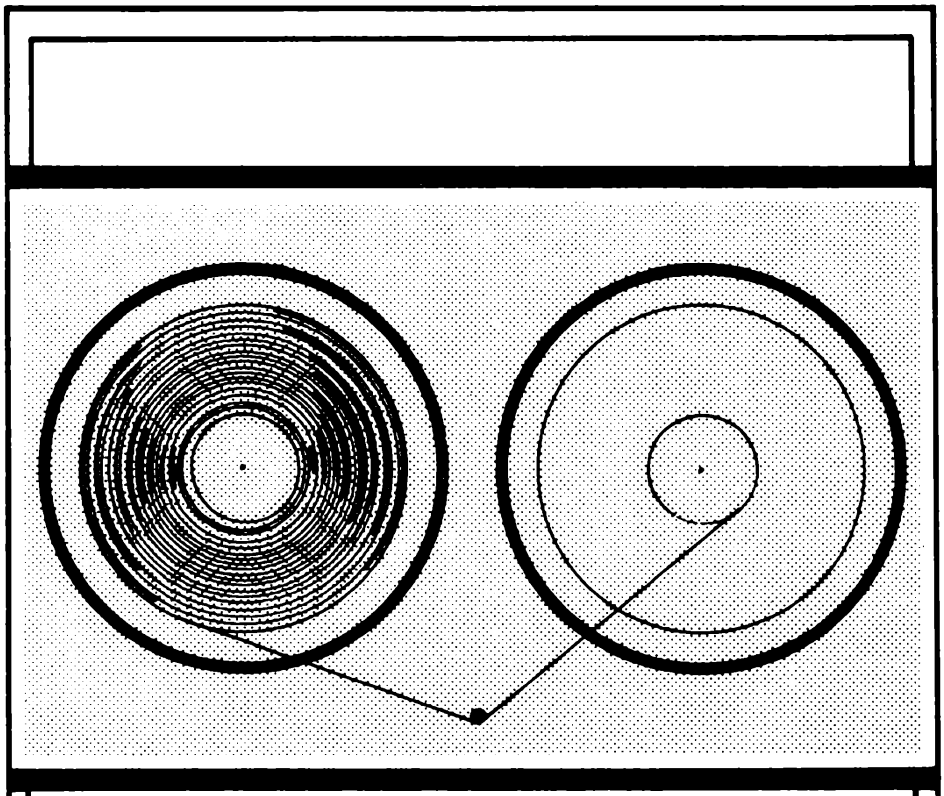
Neste exemplo, a chave está indicando que o programador deve optar entre a palavra JUSTIFIED e a palavra JUST. Apenas uma delas deve ser escrita.

— *Reticências (...)*

As reticências indicam que o item precedente pode ser repetido. Quando os três pontos seguem uma palavra, significa que aquela palavra pode ser repetida quantas vezes forem necessárias. Quando os três pontos seguem colchetes, significa que os itens dentro dos colchetes podem ser repetidos. O mesmo se aplica a itens dentro de chaves.

*

* *



CAPÍTULO 4

Identification Division

*

* *

ESTRUTURA DA IDENTIFICATION DIVISION

{*IDENTIFICATION DIVISION.*
ID DIVISION.}

PROGRAM-ID. nome-do-programa.
[*AUTHOR.* nome-do-programador.]
[*INSTALLATION.* nome-da-empresa.]
[*DATE-WRITTEN.* data.]
[*DATE-COMPILED.*]
[*SECURITY.* [comentários] ...]
[*REMARKS.* [comentários] ...]

Esta divisão tem como entradas obrigatórias apenas IDENTIFICATION DIVISION ou ID DIVISION e PROGRAM-ID. Todas as demais entradas são opcionais.

Descrição das entradas

IDENTIFICATION DIVISION ou ID DIVISION — Entrada obrigatória. Ela indica o início da divisão e o início do programa.

PROGRAM-ID — Obrigatória. Indica o nome do programa.

AUTHOR — Indica o nome do programador.

INSTALLATION — Indica a empresa a que pertence o programa.

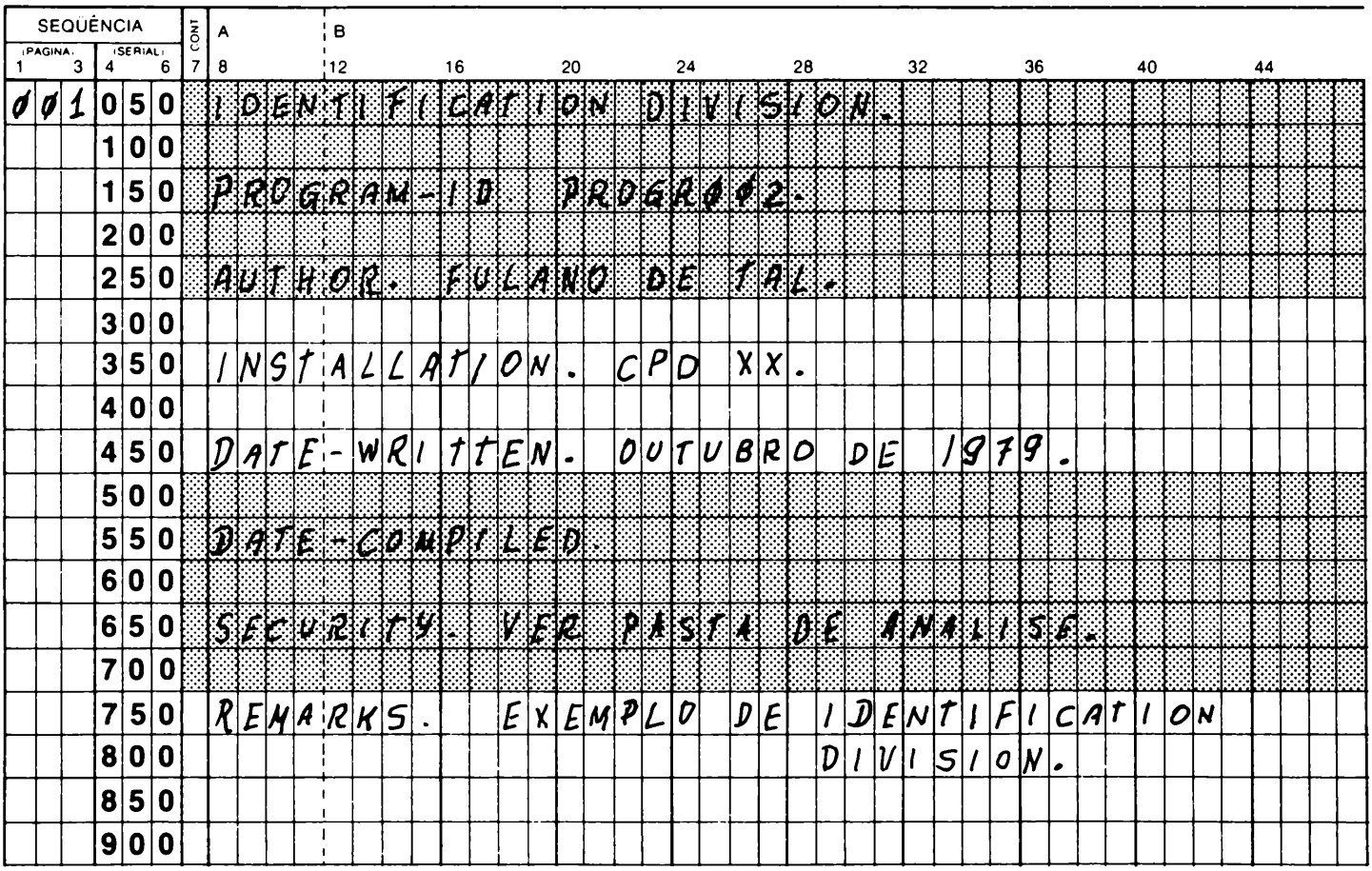
DATE-WRITTEN — Indica a data em que foi escrito o programa.

DATE-COMPILED — Indica a data em que o programa foi compilado. Quando for usado o COBOL ANSI a data será fornecida pelo compilador.

SECURITY — Referências com respeito à segurança de arquivos, programa etc.

REMARKS — Comentários em geral sobre o programa.

Exemplo 1:



Observações:

a) A identificação do programa (PROGRAM-ID) tem de começar por uma letra e pode ter até 8 caracteres.

b) A data da compilação (DATE-COMPILED) é fornecida pelo compilador, quando for usado o COBOL ANSI.

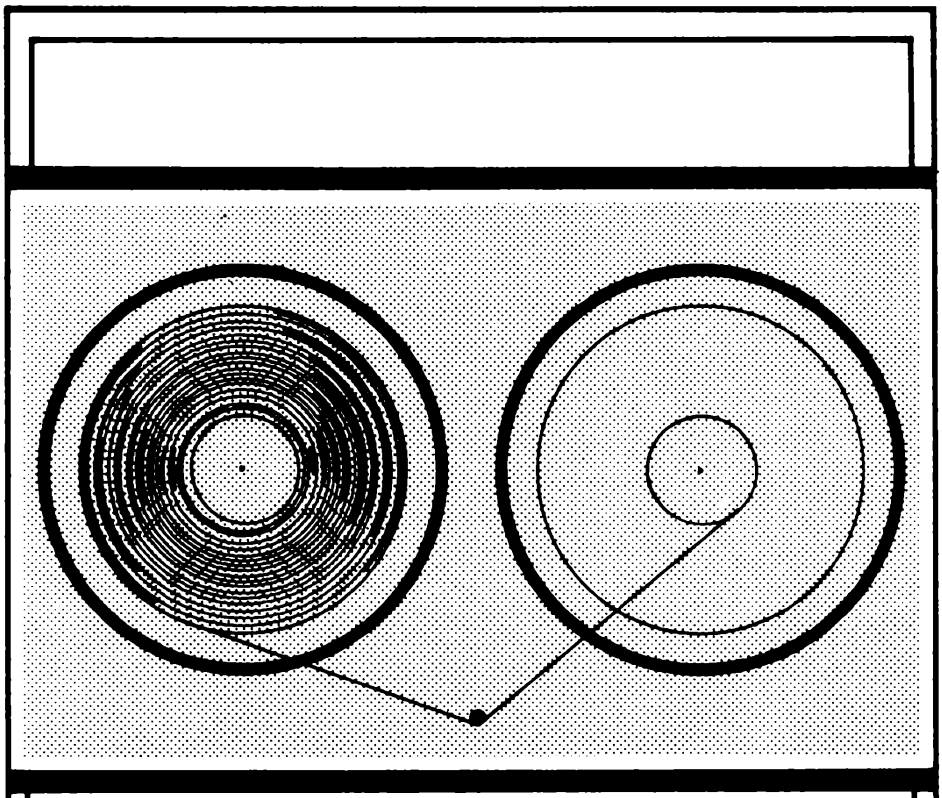
c) O número da página (colunas 1/3) não precisa ser repetido em todas as linhas do formulário. Basta orientar a perfuradora para que ela repita o mesmo número da página em todos os cartões correspondentes a todas as linhas da mesma página.

Exemplo 2:

SEQÜÊNCIA (PAGINA) 1 3	(SERIAL) 4 6	CONT 7	A 8	B 12 16 20 24 28 32 36
ØØ1	050		ID DIVISION.	
ØØ1	100		PROGRAM-ID. PROGRØØ2.	
ØØ1	150		AUTHOR. FULANO DE TAL.	
ØØ1	200		DATE COMPILED.	
	250			
	300			

*

* *



CAPÍTULO 5

Environment Division

*

* *

ENVIRONMENT DIVISION

A ENVIRONMENT DIVISION é dividida em duas seções: CONFIGURATION SECTION e INPUT-OUTPUT SECTION.

Estrutura da ENVIRONMENT DIVISION

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. descrição do computador.
OBJECT-COMPUTER. descrição do computador.
[*SPECIAL-NAMES.*]
[*INPUT-OUTPUT SECTION.*
FILE-CONTROL.
[*I-O-CONTROL*]]

Cada uma das entradas acima será discutida a seguir.
ENVIRONMENT DIVISION — Entrada obrigatória. Indica o início da divisão.
CONFIGURATION SECTION — Entrada obrigatória.
Indica início da seção.

SOURCE-COMPUTER — Entrada obrigatória. Indica o computador onde será feita a compilação do programa. Embora seja uma entrada obrigatória, o compilador a considera apenas como comentário. Não afeta a compilação nem a execução do programa.

OBJECT-COMPUTER — Entrada obrigatória. Indica o computador onde será *executado* o programa. Embora seja uma entrada obrigatória, o compilador a considera apenas como comentário, não afetando nem a compilação, nem a execução do programa.

SPECIAL-NAMES — Entrada opcional.
São três as funções desta entrada:
1. Associar nomes mnemônicos criados pelo programador com funções padronizadas pelo compilador COBOL.
2. Substituir o ponto decimal, usado em alguns países, pela vírgula decimal usada em outros países, como o Brasil.
3. Substituir o símbolo do dólar por outro símbolo.
O formato da entrada SPECIAL-NAMES é:

FORMATO

```
SPECIAL-NAMES.
    [nome-da-função IS nome-mnemônico] ...
    [CURRENCY SIGN IS literal]
    [DECIMAL-POINT IS COMMA].
```

Observações:

Pode existir vírgula ou ponto-e-vírgula para separar as diversas entradas. O ponto final é obrigatório.

NOME-DA-FUNÇÃO

O nome-da-função pode ser qualquer um dos seguintes:

CSP, C01, C02, C03, C04, C05, C06, C07, C08, C09, C10, C11, C12, S01 ou S02.

A função tomada pelo compilador para cada um dos nome-da-função é:

C01 a C12 — Salto para canal 1 até canal 12, respectivamente.

S01 e S02 — Seleção para escaninho 1 ou 2 na IBM 1442 ou seleção para escaninho P1 e P2 na IBM 2540.

NOME-MNEMÔNICO

Nome que o programador deve fornecer para substituir um dos nome-da-função. Tal substituição é feita para tornar a leitura do programa mais fácil por outro programador.

Exemplos: C01 IS CANAL-1
S01 IS ESCANINHO-1

CURRENCY SIGN IS

Esta cláusula é usada quando se deseja substituir o símbolo do dólar por outro símbolo.

LITERAL (Na cláusula SPECIAL-NAMES)

É o caráter não numérico que o programador deve indicar para substituir o símbolo do dólar. Este caráter *não* pode ser nenhum dos seguintes:
0 1 2 3 4 5 6 7 8 9 A B C D P R S V X Z ø * — , . ; ( ) + ' ' '

DECIMAL-POINT IS COMMA

Esta cláusula indica ao compilador que deve substituir a função do ponto decimal usado na notação inglesa pela vírgula, usada na nossa notação.
Exemplo:

Inglês:	1.00	10,000.00	100,000.00
Português:	1,00	10.000,00	100.000,00

Exemplos da CONFIGURATION SECTION da ENVIRONMENT DIVISION

Exemplo 1:

```
      350
      400
      450
001   500  ENVIRONMENT DIVISION.
      550
      600  CONFIGURATION SECTION.
      650
      700  SOURCE-COMPUTER.    IBM-370.
      750
      800  OBJECT-COMPUTER.    IBM-370.
      850
      900  SPECIAL-NAMES.      C01 IS CANAL-1
      950                      DECIMAL-POINT IS COMMA.
```

Exemplo 2:

```
      350
001   400  ENVIRONMENT DIVISION.
      450  CONFIGURATION SECTION.
      500  SOURCE-COMPUTER.  IBM-360.
      550  OBJECT-COMPUTER.  IBM-360.
      600
```

INPUT-OUTPUT SECTION

A INPUT-OUTPUT SECTION trata da definição de cada arquivo, da ligação entre os arquivos e as unidades de entrada/saída e das informações necessárias para otimizar as operações de entrada e saída.

O formato da INPUT-OUTPUT SECTION é:

FORMATO

```
[INPUT-OUTPUT SECTION.
 FILE-CONTROL. {entradas da file-control}
[I-O-CONTROL. entradas da i-o-control]]
```

FILE-CONTROL

O parágrafo FILE-CONTROL é usado para fazer a ligação do arquivo, que será descrito na DATA DIVISION, com as unidades de entrada/saída.

O formato da FILE-CONTROL é:

FORMATO

```
FILE-CONTROL.
     {SELECT
      ASSIGN
       [RESERVE]
       [FILE-LIMIT]
       [ACCESS MODE]
       [PROCESSING MODE]
       [ACTUAL KEY]
       [NOMINAL KEY]
       [RECORD KEY]. } ...
```

SELECT

Cada cláusula SELECT deve ser seguida por uma cláusula ASSIGN. As demais cláusulas podem ser escritas em qualquer ordem.

Para cada arquivo do programa deverá ser escrita uma cláusula SELECT.

O formato da cláusula SELECT é:

FORMATO

SELECT *OPTIONAL* nome-do-arquivo

OPTIONAL

Indica que o arquivo apontado pela cláusula SELECT poderá estar ou não alimentado quando for executado o programa. O parâmetro OPTIONAL só é válido para arquivos de acesso seqüencial.

Se, por ocasião da leitura do arquivo, ele não tiver sido alimentado e a palavra-chave OPTIONAL tiver sido fornecida na cláusula SELECT, no primeiro comando READ ocorrerá a condição AT END (o comando READ e o parâmetro AT END serão estudados na PROCEDURE DIVISION).

NOME-DO-ARQUIVO

O nome do arquivo que será descrito na DATA DIVISION.

ASSIGN

É através da cláusula ASSIGN que um arquivo é ligado à unidade de entrada/saída.

O formato da cláusula ASSIGN é:

FORMATO

ASSIGN TO [inteiro-1] nome-do-sistema-1
[nome-do-sistema-2) ...
[FOR *MULTIPLE*

{*REEL* / UNIT}

Inteiro-1 — indica a quantidade de unidade de entrada/saída reservada para o arquivo.

Nome-do-sistema-1 — indica a unidade a ser ligada ao arquivo apontado na cláusula SELECT. A estrutura do nome-do-sistema-1 é:
classe-dispositivo-organização-nome

Classe: pode ser DA
UT
UR

Dispositivo: A classe será DA se o dispositivo for uma das unidades abaixo:
2301, 2302, 2303, 2311, 2314 e 2321.

A classe será UT se o dispositivo for uma das unidades abaixo:
2301, 2302, 2311, 2314, 2321, ou 2400.

A classe será UR se o dispositivo for uma das unidades abaixo:
1403, 1404, 1442R, 1442P, 1443, 1445, 2501, 2520R, 2520P, 2540R ou 2540P (R indica leitora, P indica perfuradora).

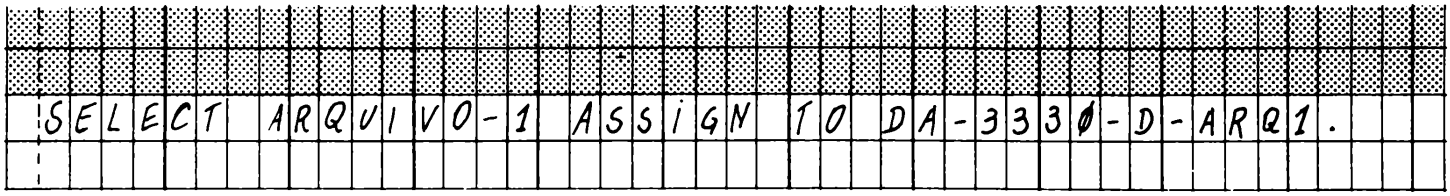
Organização: A organização será indicada por um caráter como indicado abaixo:
S — para arquivos com organização seqüencial.
D — para arquivos com organização direta.
W — para arquivos com organização direta quando for usado REWRITE.

R — para arquivos com organização relativa.
I — para arquivos com organização indexada.

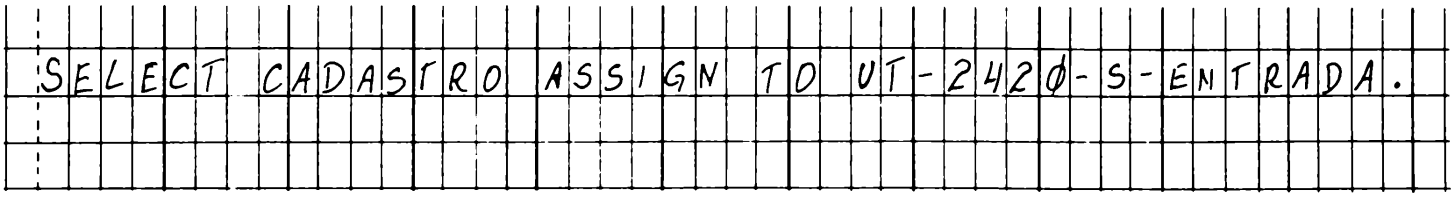
Nome: É um nome com até 8 caracteres, começando obrigatoriamente com um caráter alfabético. É o nome do cartão DD dos cartões de JCL.

Exemplos de cláusulas SELECT e ASSIGN

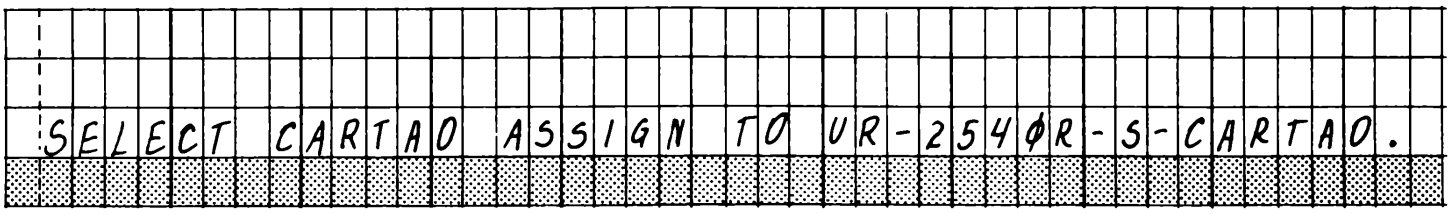
Exemplo 1:



Exemplo 2:



Exemplo 3:



Exemplo 4:

```
SELECT PERFURA ASSIGN TO UR-2540P-S-SAIDA.
```

Exemplo 5:

```
SELECT RELAT ASSIGN TO UR-1403-S-RELAT.
```

RESERVE

A cláusula RESERVE permite ao programador modificar o número de áreas de entrada e saída (buffers) alocados normalmente pelo compilador.

O formato da cláusula RESERVE é:

FORMATO

```
RESERVE  { NO      }  ALTERNATE  [ AREA  ]
         { inteiro }              [ AREAS ]
```

Esta cláusula só deve ser usada para arquivos seqüenciais ou indexados. Para arquivos diretos ou relativos ela é ignorada.

NO

Indica que só deve ser reservada uma área de trabalho para o arquivo, não sendo reservado nenhum buffer adicional.

Inteiro

Indica a quantidade de buffers a ser reservada para o arquivo, até a quantidade de 254 no máximo.

Observações:

a) Se a cláusula RESERVE for omitida e a cláusula SAME AREA (no parágrafo I-O-CONTROL) for usada para o mesmo arquivo, serão reservadas 2 áreas.

b) Se a cláusula RESERVE for omitida e a cláusula SAME AREA não for usada para o mesmo arquivo, o número de áreas será tomado do cartão DD do arquivo, na hora da execução. (O cartão DD é um cartão de JCL.) Se a quantidade de áreas não for indicada no cartão DD, serão reservadas duas áreas.

FILE-LIMIT

Esta é uma cláusula sem finalidade prática e que é considerada como comentário pelo compilador. Podemos desconsiderá-la.

ACCESS MODE

Indica como os registros do arquivo deverão ser acessados.

O formato da cláusula ACCESS MODE é:

FORMATO

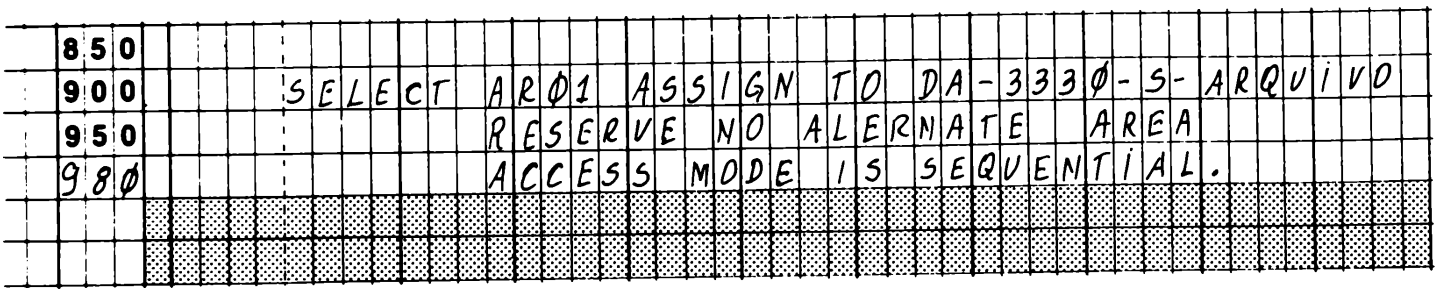
ACCESS MODE *IS* { *SEQUENTIAL* | *RANDOM* }

Se esta cláusula for omitida será assumida a opção SEQUENTIAL.

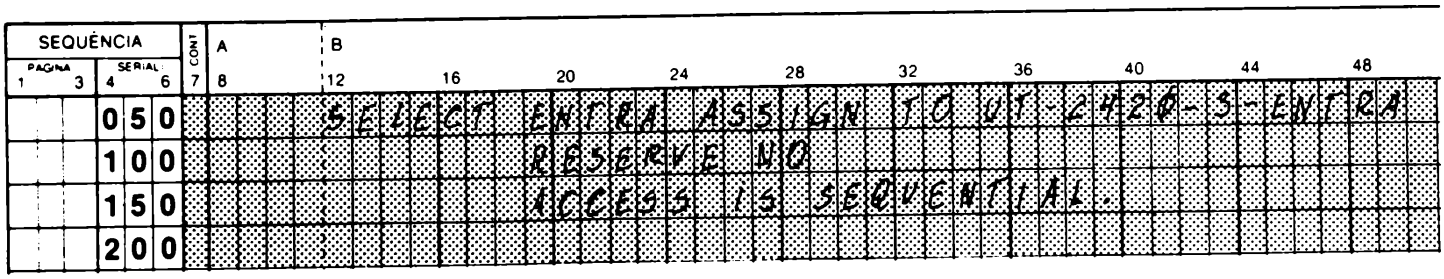
Se for indicada a opção RANDOM, o acesso será em função de uma ACTUAL KEY ou de uma NOMINAL KEY associada com cada registro. (As cláusulas ACTUAL KEY e NOMINAL KEY são estudadas neste capítulo.)

Exemplos de cláusulas RESERVE e ACCESS

Exemplo 1:



Exemplo 2:



PROCESSING MODE

É uma cláusula sem finalidade prática e que é considerada como comentário pelo compilador. Por este motivo podemos desconsiderá-la.

ACTUAL KEY

A cláusula ACTUAL KEY permite a localização de um registro lógico num arquivo de acesso direto. Ela deve ser fornecida sempre que for usada a cláusula ACCESS MODE IS RANDOM.

O formato da cláusula ACTUAL KEY é:

FORMATO

ACTUAL KEY *IS* nome-dos-dados

Nome-dos-dados

É um campo de 5 a 259 bytes de comprimento. Os primeiros 4 bytes devem conter o número relativo da trilha onde a pesquisa do registro deve começar ou onde deverá ser colocado o novo registro. Os bytes seguintes representam o campo de controle do registro.

NOMINAL KEY

Esta cláusula é usada para arquivos indexados ou relativos.

Para arquivos indexados, a cláusula especifica uma identidade simbólica para um registro lógico específico.

Para arquivos relativos especifica o número relativo do registro, em relação ao início do arquivo.

O formato da NOMINAL KEY é:

FORMATO

NOMINAL KEY IS nome-dos-dados

RECORD KEY

Esta cláusula é usada para acessar um arquivo indexado. Ela especifica o campo dentro do registro de entrada que contém a chave seqüencial a usar na identificação do registro no disco.

O formato da RECORD KEY é:

FORMATO

RECORD KEY IS nome-dos-dados

I-O-CONTROL

O parágrafo I-O-CONTROL é usado para otimizar as operações de entrada/saída.

Todo o parágrafo I-O-CONTROL é opcional.

O formato do parágrafo I-O-CONTROL é:

FORMATO

```
I-O-CONTROL.
    [RERUN] ...
    [SAME AREA] ... .
```

As entradas acima indicadas podem ser escritas em qualquer ordem.

RERUN

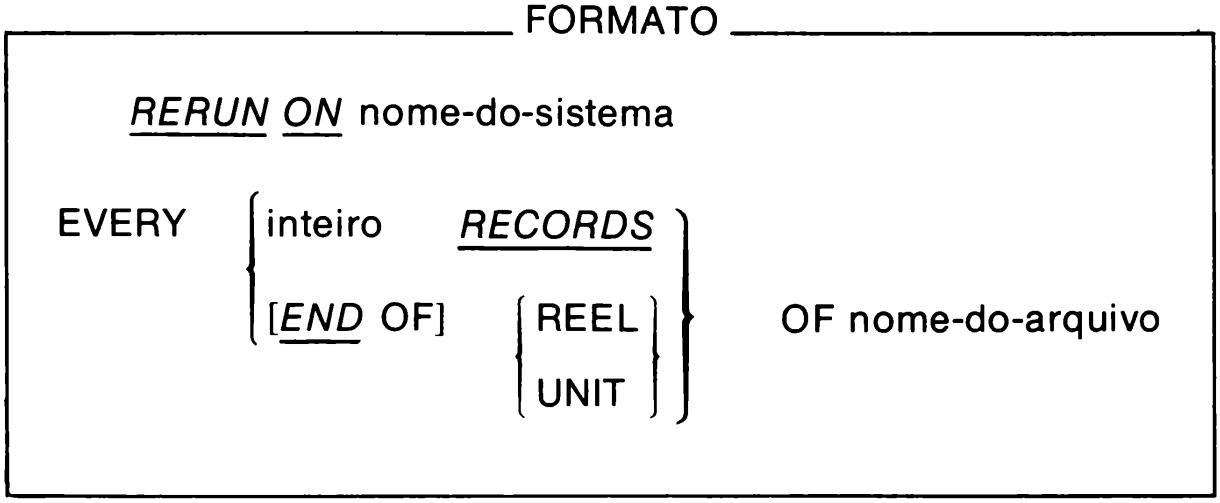
A cláusula RERUN indica que devem ser gravados registros de *checkpoint.*

Os registros de *checkpoint* são registros da situação do programa e da máquina gravados a intervalos regulares especificados pelo programador. Em caso de necessidade, o programa

pode ser reiniciado a partir de um determinado ponto, sem ter de reprocessar todo o programa novamente.

O formato da cláusula RERUN é:

FORMATO



Os registros de *checkpoint* podem ser gravados em fita ou disco magnético.

Nome-do-sistema

Especifica onde devem ser gravados os registros de *checkpoint*. Não pode ser o mesmo nome-de-sistema usado na cláusula ASSIGN mas segue as mesmas regras de formação.

Nome-do-arquivo

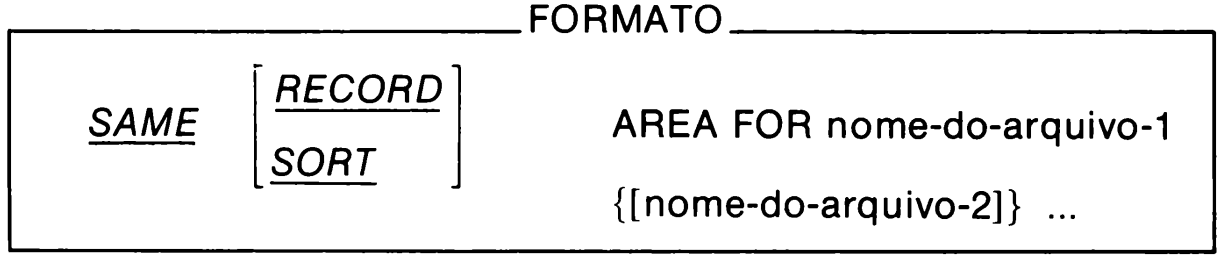
Indica para qual arquivo o *checkpoint* deve ser gravado.

SAME

A cláusula SAME serve para indicar que dois ou mais arquivos devem usar a mesma área de memória na hora do processamento.

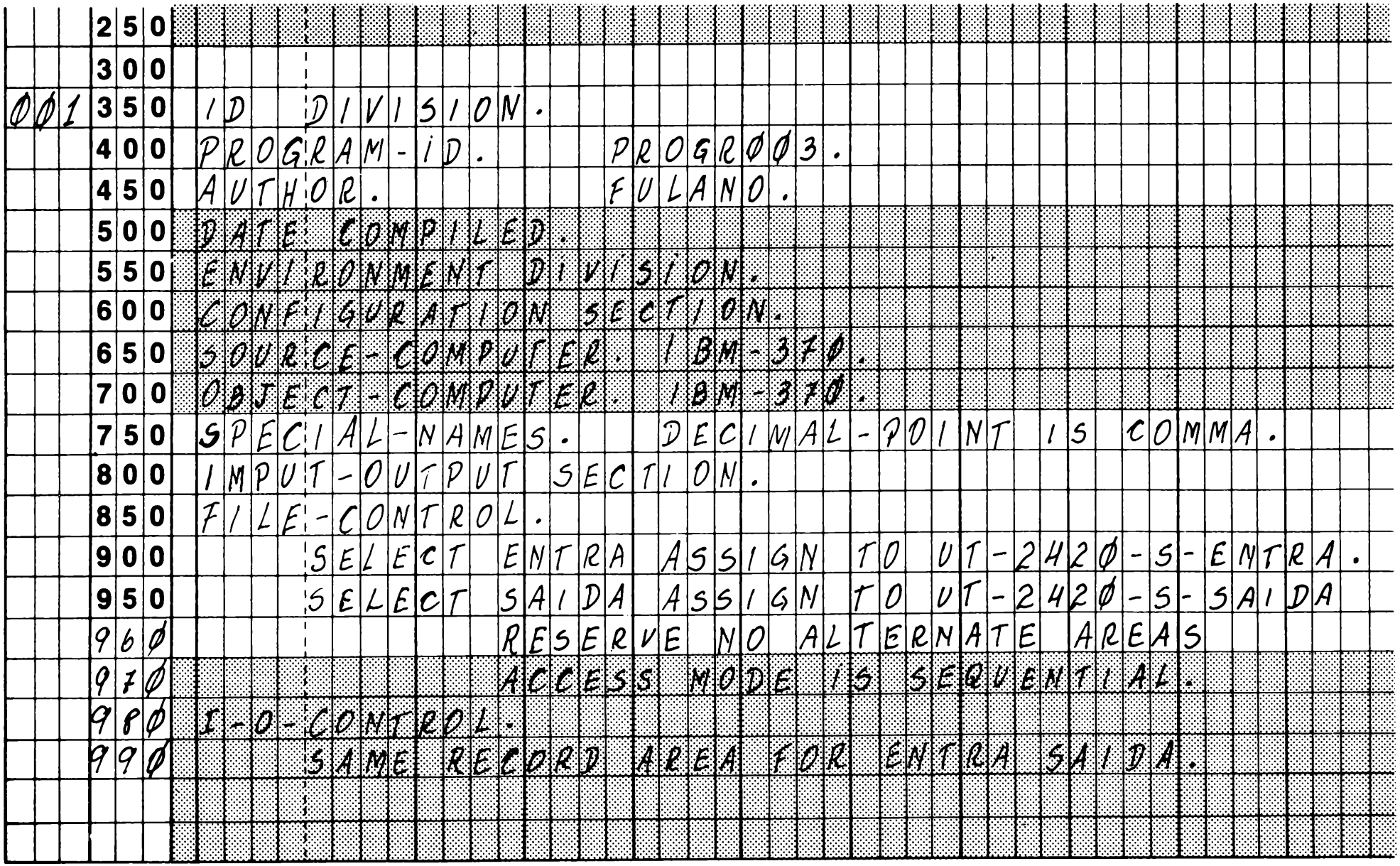
O formato da cláusula SAME é:

FORMATO



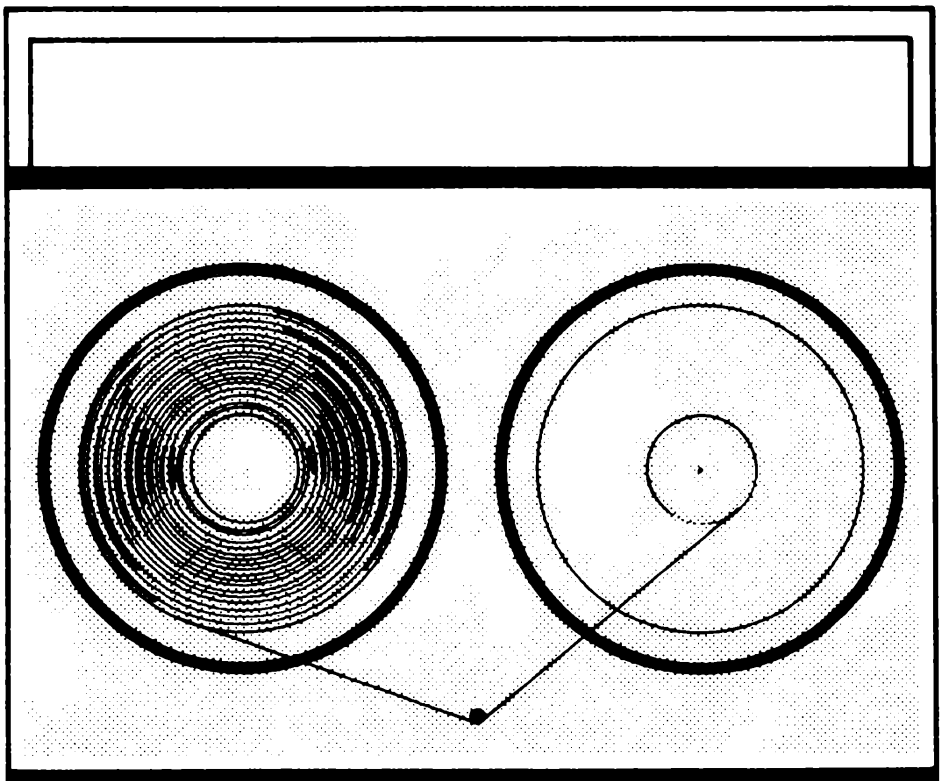
Exemplos de IDENTIFICATION DIVISION e ENVIRONMENT DIVISION

Exemplo 1:



Exemplo 2:

```
SEQÜÊNCIA  (PÁGINA) 1-3  (SERIAL) 4-6   CONT 7   A 8   B 12 16 20 24 28 32 36 40 44 48 52
ØØ1 050    IDENTIFICATION DIVISION.
           100    PROGRAM-ID.                                  PROGRØØ3.
           150    AUTHOR.                                      FULANO.
           200    DATE-COMPILED.
           250    *
           300    *** AQUI COMEÇA A ENVIRONMENT DIVISION ****
           350    *
           400    ENVIRONMENT DIVISION.
           450    *
           500    * O CARTAO ØØ14ØØ ESTA ERRADO. NAO PODE COME-
           550    * CAR NA COLUNA 7. TEM DE COMEGAR NA COLUNA 8
           600    *
           650    CONFIGURATION SECTION.
           700    SOURCE-COMPUTER.                             IBM-36Ø.
           750    OBJECT-COMPUTER.                             IBM-36Ø.
           800    *
           850    INPUT-OUTPUT SECTION.
           900    FILE-CONTROL
           950        SELECT ENTRA ASSIGN TO UT-2420-S-ENTRA.
           96Ø        SELECT SAIDA ASSIGN TO UT-2420-S-SAIDA.
```



CAPÍTULO 6

Data Division

*

* *

DATA DIVISION

A DATA DIVISION é composta das seções:
FILE SECTION
WORKING-STORAGE SECTION
LINKAGE-SECTION

Estrutura da DATA DIVISION

DATA DIVISION.
FILE SECTION.
  {descrição do arquivo
  {descrição do registro} ... } ...
WORKING-STORAGE SECTION
  [descrição dos dados] ...
  [descrição dos registros] ...
LINKAGE SECTION.
  [descrição dos dados] ...
  [descrição dos registros] ...

FILE SECTION

A FILE SECTION é usada para descrever os arquivos de entrada/saída.

O formato da FILE SECTION é:

FORMATO

```
{descrição do arquivo
{descrição do registro} ... } ...
```

FILE DESCRIPTION

Estudaremos a seguir as descrições dos arquivos de entrada/saída.

O formato da FILE DESCRIPTION (descrição dos arquivos) é:

FORMATO

```
FD  nome-do-arquivo
[BLOCK CONTAINS]
[RECORD CONTAINS]
[RECORDING MODE]
[LABEL RECORDS]
[VALUE OF]
[DATA RECORDS].
```

BLOCK CONTAINS

A cláusula BLOCK CONTAINS é usada para especificar o tamanho do bloco. Ela pode ser omitida se o fator-de-bloco for 1, isto é, se em cada bloco houver apenas um registro lógico.

O formato da cláusula BLOCK CONTAINS é:

FORMATO

```
BLOCK CONTAINS [inteiro-1  TO] inteiro-2 { CHARACTERS }
                                         { RECORDS    }
```

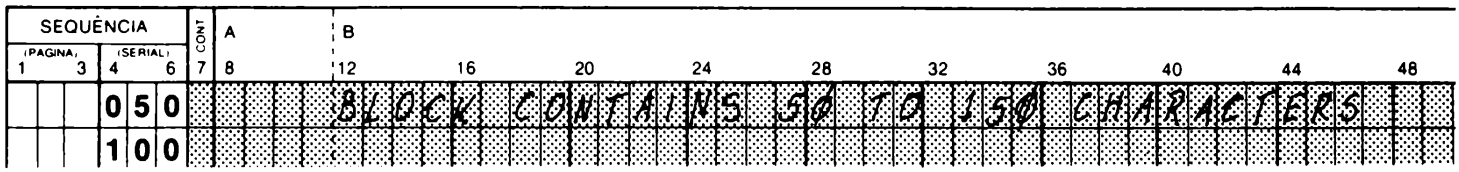
Não há necessidade de especificar a cláusula BLOCK CONTAINS para:

— Arquivos de acesso direto com registros F, V ou U.
— Arquivos com registros U.
— Arquivos relativos.
— Arquivos de acesso direto quando a cláusula RECORDING MODE for especificada para registros S.

INTEIRO-1

Usado quando o bloco for de tamanho variável. Ele indica o tamanho mínimo do bloco. Neste caso, há necessidade de codificar inteiro-2 (tamanho máximo do bloco) e codificar CHARACTERS.

Exemplo: A cláusula BLOCK CONTAINS para um bloco que pode variar de 50 a 150 bytes seria:

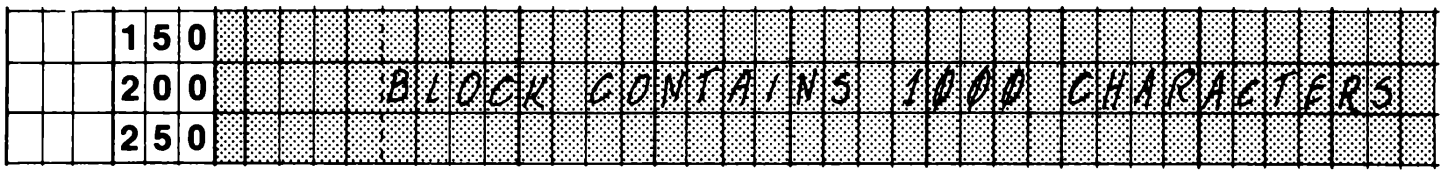


INTEIRO-2

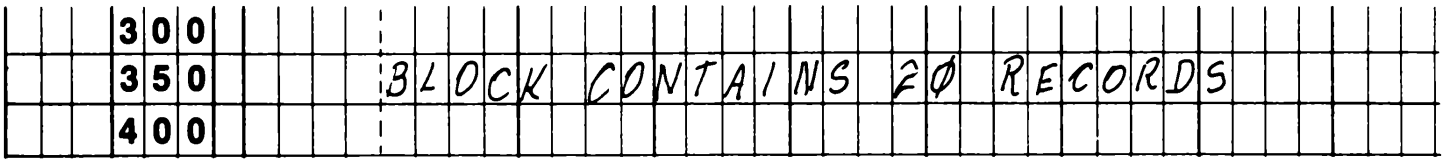
a) Se usado com o parâmetro CHARACTERS e com o parâmetro inteiro-1, indica o tamanho máximo do bloco. Veja exemplo anterior.

b) Se usado com o parâmetro CHARACTERS e o parâmetro inteiro-1 não tiver sido usado, indica o tamanho exato do bloco.

Exemplo: A cláusula BLOCK CONTAINS para um bloco de 1000 caracteres seria:



c) Se usado com o parâmetro RECORDS, indica o fator-de-bloco.
Exemplo: A cláusula BLOCK CONTAINS para um bloco cujos registros são agrupados de 20 em 20 (fator de bloco 20) seria:



Se não for especificado CHARACTERS nem RECORDS, o compilador assumirá CHARACTERS.

RECORD CONTAINS

A cláusula RECORD CONTAINS é usada para especificar o tamanho de cada registro de um arquivo.

Esta cláusula não é obrigatória, pois cada registro terá de ser completamente definido numa RECORD DESCRIPTION (estudada neste capítulo). Embora não obrigatória, ela deve ser usada para facilitar a manutenção do programa, por evitar que tenhamos de calcular o tamanho do registro através da RECORD DESCRIPTION, sempre que tivermos necessidade de saber o tamanho do registro.

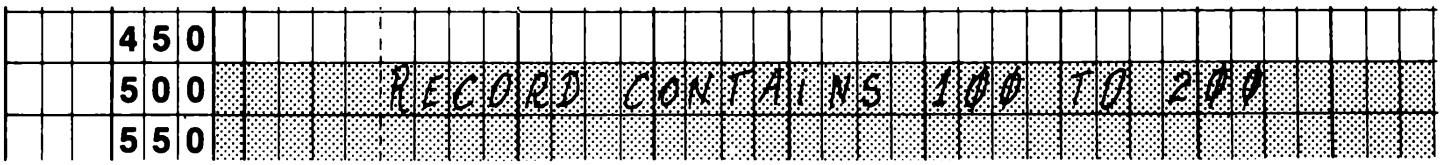
O formato da cláusula RECORD CONTAINS é:

FORMATO

RECORD CONTAINS [inteiro-1 TO] inteiro-2 CHARACTERS

INTEIRO-1

Indica o menor tamanho do registro. Neste caso, há necessidade de codificar inteiro-2 para indicar o maior tamanho do registro.
Exemplo: A cláusula RECORD CONTAINS para um registro que possa variar de 100 a 200 caracteres seria:

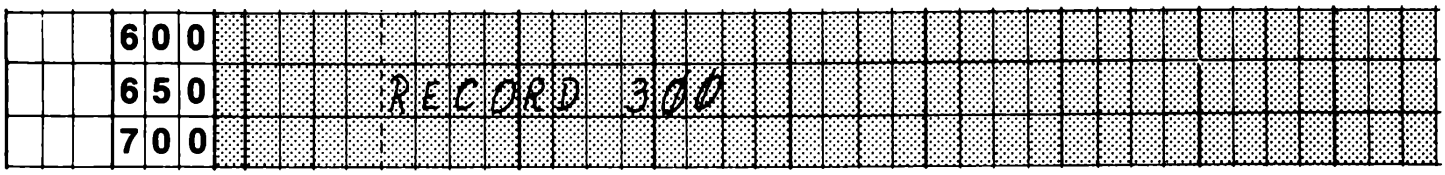


INTEIRO-2

Se usado com inteiro-1, indica o maior tamanho do registro.

Se não for usado inteiro-1, indica o tamanho exato do registro.

Exemplo: A cláusula RECORD CONTAINS para um registro de 300 caracteres seria:



RECORDING MODE

A cláusula RECORDING MODE é usada para especificar o formato dos registros lógicos no arquivo.

Esta cláusula não é obrigatória. O seu uso, no entanto, facilita o trabalho de manutenção do programa.

O formato da cláusula RECORDING MODE é:

FORMATO

RECORDING MODE IS modo

Modo: F — indica registro de tamanho fixo.
V — indica registro de tamanho variável.
U — indica registro de tamanho indefinido.

LABEL RECORDS

A cláusula LABEL RECORDS indica se há ou não LABEL no arquivo. Em caso positivo, identifica o tipo de LABEL.

O formato da cláusula LABEL RECORDS é:

FORMATO

```
LABEL  { RECORD IS  }        { OMITTED  }
       { RECORDS ARE }       { STANDARD }
```

OMITTED

Indica que o arquivo não possui *label.* Deve ser usado sempre para arquivos em cartões perfurados e relatórios a serem impressos em impressoras.

STANDARD

Indica que o arquivo possui *label standard.*

VALUE OF

A cláusula VALUE OF particulariza a descrição de um item no registro *label.* Como o compilador a considera apenas como comentário, podemos desconsiderá-la.

DATA RECORDS

A cláusula DATA RECORDS identifica cada registro do arquivo por um nome. Embora não obrigatória, deve ser usada para facilitar a manutenção do programa.

O formato da cláusula DATA RECORDS é:

FORMATO

```
DATA  { RECORD IS  }  nome-do-registro-1
      { RECORDS ARE }
                      [nome-do-registro-2]
```

Exemplos:

1) A cláusula DATA RECORDS para um arquivo que possui apenas um tipo de registro poderia ser:

```
750
800     DATA RECORD IS REGISTRO
850
```

2) A cláusula DATA RECORDS para um arquivo que possui registros de nome, registros de endereço e registros de valor, poderia ser:

```
DATA RECORDS ARE REG-NOME REG-END REG-VALOR
```

Exemplo completo de FILE DESCRIPTION

Escrever a *File Description* de um cadastro gravado em fita magnética com *label standard,* com registros de tamanho fixo de 100 caracteres. Cada bloco do arquivo contém 20 registros. Existem dois tipos de registros no cadastro: registros de dados pessoais e registros de dados empresariais.

```
SEQUÊNCIA  PAGINA 1 3  SERIAL 4 6   CONT 7   A 8   B 12 16 20 24 28 32 36 40 44
050        FD  CADASTRO
100            BLOCK CONTAINS 20 RECORDS
150            RECORD CONTAINS 100 CHARACTERS
200            RECORDING MODE F
250            LABEL RECORDS ARE STANDARD
300            DATA RECORDS ARE PESSOAIS EMPRESA.
350
```

A descrição de cada registro de um arquivo segue as regras gerais para descrição de quaisquer dados. Por esse motivo, veremos a seguir, simultaneamente, a descrição de registros e a descrição de dados.

O formato para uma descrição de registros ou de outros dados é:

FORMATO

```
número-de-nível  { nome-de-dados }
                 { FILLER     }

[REDEFINES]
[BLANK WHEN ZERO]
[JUSTIFIED]
[OCCURS]
[PICTURE]
[SYNCHRONIZED]
[USAGE]
[VALUE]
```

NÚMERO-DE-NÍVEL

É um dado a ser fornecido pelo programador. Pode variar de 01 a 49.

O número-de-nível é usado para estruturar um registro, de forma a obter subdivisões que possam ser referenciadas. Por exemplo: Se num programa houver necessidade de se referir a um campo de data, podemos fazer uma descrição assim: 01 DATA. Esta descrição permitirá fazer referência à data completa (dia, mês e ano). Se houver, no entanto, necessidade de se fazer referência, separadamente, ao dia, ao mês e ao ano, poderemos dividir o campo de data em 3 subcampos com um outro número-de-nível, assim:

```
01  DATA.
  02  DIA.
  02  MÊS.
  02  ANO.
```

Os itens que possuem subdivisões são chamados ITENS DE GRUPO. No exemplo acima, o item 01 é um item de grupo, pois possui as subdivisões DIA, MÊS e ANO.

Os itens que não possuem subdivisões são chamados de itens elementares. No exemplo acima, são itens elementares: DIA, MÊS e ANO.

Podem ser feitas várias subdivisões numa mesma descrição. Neste caso, teremos vários itens de grupo numa mesma descrição.

Por exemplo:

```
01 REGISTRO.
   03 EMPREGADO.
     05 NOME.
     05 ENDEREÇO.
     05 NASCIMENTO.
       07 DIA.
       07 MÊS.
       07 ANO.
   03 SALÁRIO.
```

Neste exemplo, o registro é dividido em dois campos (nível 03): EMPREGADO e SALÁRIO.

O campo de empregado foi subdividido em 3 outros campos (nível 05): nome, endereço e nascimento.

O campo de nascimento foi subdividido, ainda, em três campos (nível 07): dia, mês e ano.

Graficamente, ficaria assim o nosso registro:

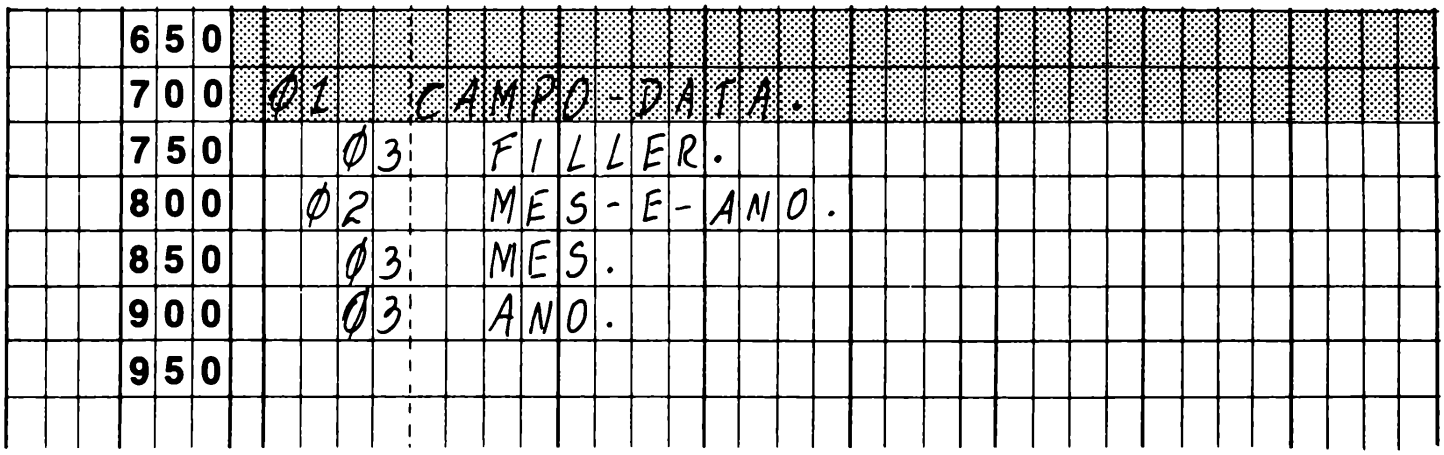
<table>
<tr><td>Nível 01</td><td colspan="6">REGISTRO</td></tr>
<tr><td>Nível 03</td><td colspan="5">EMPREGADO</td><td>SALÁRIO</td></tr>
<tr><td>Nível 05</td><td>NOME</td><td colspan="2">ENDEREÇO</td><td colspan="2">NASCIMENTO</td><td rowspan="2"></td></tr>
<tr><td>Nível 07</td><td></td><td colspan="2"></td><td>DIA</td><td>MÊS</td><td>ANO</td></tr>
</table>

Observações:
— A descrição de um registro ou de uma área deverá começar com um nível 01.
— O nível 01 deve ser perfurado na margem A (colunas 8 a 11) do cartão. Os níveis 02 a 49 devem ser perfurados na margem A ou B (colunas 12 a 82) do cartão.
— O número-de-nível de um item subordinado a outro item tem de ser maior que o número-de-nível do item que o subordina. Não precisa, no entanto, ser o número imediatamente superior.
Exemplo: Se um item de grupo tiver o número-de-nível igual a 05, o item elementar subordinado não tem de ser necessariamente 06. Pode ser 06, 07, 08 ou outro qualquer até 49.

— Existem números-de-nível especiais, com funções específicas. Os números-de-nível especiais são:
66 — Usado com a cláusula RENAMES, estudada mais adiante.
77 — Usado no WORKING-STORAGE, estudada mais adiante.
88 — Usado com a cláusula VALUE, estudada mais adiante.

NOME-DE-DADOS
Um nome qualquer criado pelo programador para identificar um registro ou área.
Exemplos: 1) 01 REGISTRO.
2) 01 CAMPO.

FILLER
Palavra usada quando o campo, embora necessário, não será referenciado pelo programador.
Por exemplo: Suponhamos um programa em que, num campo de data, teremos de nos referir a:
1) à data completa;
2) ao mês e ano em conjunto;
3) ao mês isoladamente;
4) ao ano isoladamente.
O campo de dia não será solicitado isoladamente em nenhum ponto do programa. Assim, o campo correspondente ao dia poderá ser descrito como FILLER. A definição do campo de data poderia ser:



Graficamente, o campo ficaria assim:

Nível 01	DATA		
Nível 02		MÊS-E-ANO	
Nível 03	FILLER	MÊS	ANO

A palavra FILLER não substitui o conteúdo do campo. No caso acima, o dia não seria alterado. O FILLER só indica que o campo de dia não será referido isoladamente.

REDEFINES

A cláusula REDEFINES permite mais de uma descrição para o mesmo grupo de dados.

Por exemplo: Poderemos ter necessidade de descrever um mesmo campo de datas de três maneiras diferentes, como indicadas abaixo:

```
1) 01 DATAS-1.
      03 DIA.
      03 MÊS.
      03 ANO.

2) 01 DATAS-2.
     02 DIA-MÊS.
       03 DIA.
       03 MÊS.
     02 FILLER.
       03 ANO.
```

3) 01 DATAS-3.
    03 DIA.
  02 MÊS-ANO.
    03 MÊS.
    03 ANO.

Se fizéssemos a definição dos campos tal como apresentada anteriormente estaríamos definindo três campos e não apenas um campo com três descrições, como estaríamos necessitando. Para resolver tal tipo de problema usamos a cláusula REDEFINES, cujo formato é o seguinte:

FORMATO

número-de-nível nome-de-dados-1 REDEFINES nome-de-dados-2

NÚMERO-DE-NÍVEL
O número-de-nível do campo que está sendo redefinido.

NOME-DE-DADOS-1
Um nome qualquer, diferente do nome do campo que está sendo redefinido.

NOME-DE-DADOS-2
O nome do campo que está sendo redefinido.

As cláusulas REDEFINES para o exemplo das datas citado acima, seriam:

SEQÜÊNCIA (PÁGINA) 1–3	(SERIAL) 4–6	CONT 7	A 8 / B 12 … 40

```
050  Ø1  DATAS-1.
100      Ø3  DIA.
150      Ø3  MES.
200      Ø3  ANO.
250
300  Ø1  DATAS-2 REDEFINES DATAS-1.
350      Ø2  DIA-MES.
400          Ø3  DIA.
450          Ø3  MES.
500      Ø2  FILLER.
550          Ø3  ANO.
600
650  Ø1  DATAS-3 REDEFINES DATAS-2.
700          Ø3  DIA.
750      Ø2  MES-ANO.
800          Ø3  MES.
850          Ø3  ANO.
900
950
```

BLANK WHEN ZERO

Esta cláusula especifica que um item é para ser preenchido com brancos (espaços) sempre que o seu valor for zero.

JUSTIFIED

Em COBOL, ao ser movimentado o conteúdo de um campo alfabético ou alfanumérico, o alinhamento é feito pela esquerda. Por exemplo:

Conteúdo do campo A: 

Se movermos o conteúdo do campo A para dois campos B e C com capacidade para 6 e 10 caracteres, respectivamente, o conteúdo dos campos B e C serão:

conteúdo do campo B: 

conteúdo do campo C: 

A cláusula JUSTIFIED serve para que o alinhamento dos campos alfabéticos e alfanuméricos seja feito pela direita.

Usando-se a cláusula JUSTIFIED no movimento de dados do exemplo anterior, os campos B e C ficariam assim:

Conteúdo do campo B: 

Conteúdo do campo C: 

Observações:

1) A cláusula JUSTIFIED só pode ser especificada para itens elementares.

2) A cláusula JUSTIFIED não pode ser especificada para níveis 66 ou 88.

OCCURS

Esta é uma cláusula para definir tabelas. A cláusula OCCURS elimina a necessidade de descrever várias vezes um determinado tipo de dado que seja repetitivo.

Por exemplo: Na descrição de um registro onde haja um campo para número do contrato e 7 campos, um para cada dia da semana, normalmente a sua definição seria assim:

```
SEQÜÊNCIA          CONT.  A         B
(PÁGINA)  (SERIAL)        8         12        16        20        24        28        32
1     3   4     6    7    ....|....|....|....|....|....|....|....|....|....|....|....|
0 5 0     01   REGISTRO.
1 0 0       03 CONTRATO.
1 5 0       03 DIA-1.
2 0 0       03 DIA-2.
2 5 0       03 DIA-3.
3 0 0       03 DIA-4.
3 5 0       03 DIA-5.
4 0 0       03 DIA-6.
4 5 0       03 DIA-7.
5 0 0
```

Com o uso da cláusula OCCURS a descrição pode ser simplificada como mostra o exemplo abaixo.

```
5 5 0
6 0 0     01   REGISTRO.
6 5 0       03 CONTRATO.
7 0 0       03 DIAS OCCURS 7 TIMES.
7 5 0
```

Apresentaremos a seguir os dois formatos mais usados na cláusula OCCURS.

FORMATO 1

OCCURS inteiro-2 TIMES.

Este foi o formato usado no exemplo anterior.

Inteiro-2: Indica a quantidade de vezes que o dado se repete.

FORMATO 2

OCCURS inteiro-1 *TO* inteiro-2 TIMES

[*DEPENDING* ON nome-de-dados-1]

INTEIRO-1
Indica o menor número de vezes que pode-se repetir o dado. Pode variar de zero a 32.766.

INTEIRO-2
Indica a maior quantidade de vezes que pode-se repetir o dado. Pode variar de 1 a 32.767 e precisa ser maior do que inteiro-1.

DEPENDING ON
Indica que a quantidade de vezes que o dado se repete pode variar. A quantidade de vezes que o dado se repete deverá ser fornecida no campo nome-de-dados-1.

NOME-DE-DADOS-1
Indica a quantidade de vezes que o dado se repete, quando usado o parâmetro DEPENDING ON.

No exemplo abaixo, o campo VALOR se repetirá de 1 a 100 vezes, dependendo do conteúdo do campo QUANT. Se QUANT contiver 5, o campo VALOR se repetirá 5 vezes. Se o campo QUANT contiver 83, o campo VALOR se repetirá 83 vezes e assim por diante.

```
01  EXEMPLO.
    03  VALOR  OCCURS  1  TO  100  DEPENDING  ON  QUANT.
```

PICTURE

A cláusula PICTURE descreve as características gerais, tais como tamanho, tipo e formato de impressão de um item elementar. A cláusula PICTURE não pode ser usada para um item de grupo.

O formato da cláusula PICTURE é:

FORMATO

{ *PICTURE* / *PIC* }	IS	formato

FORMATO

O formato consiste de algumas combinações permitidas de alguns caracteres. O número máximo de caracteres permitidos no formato é 30.

O formato indica a categoria de um item elementar. Há 5 categorias de dados que podem ser descritos com uma cláusula PICTURE:

1 — alfabética;
2 — numérica;
3 — alfanumérica;
4 — alfanumérica editada;
5 — numérica editada.

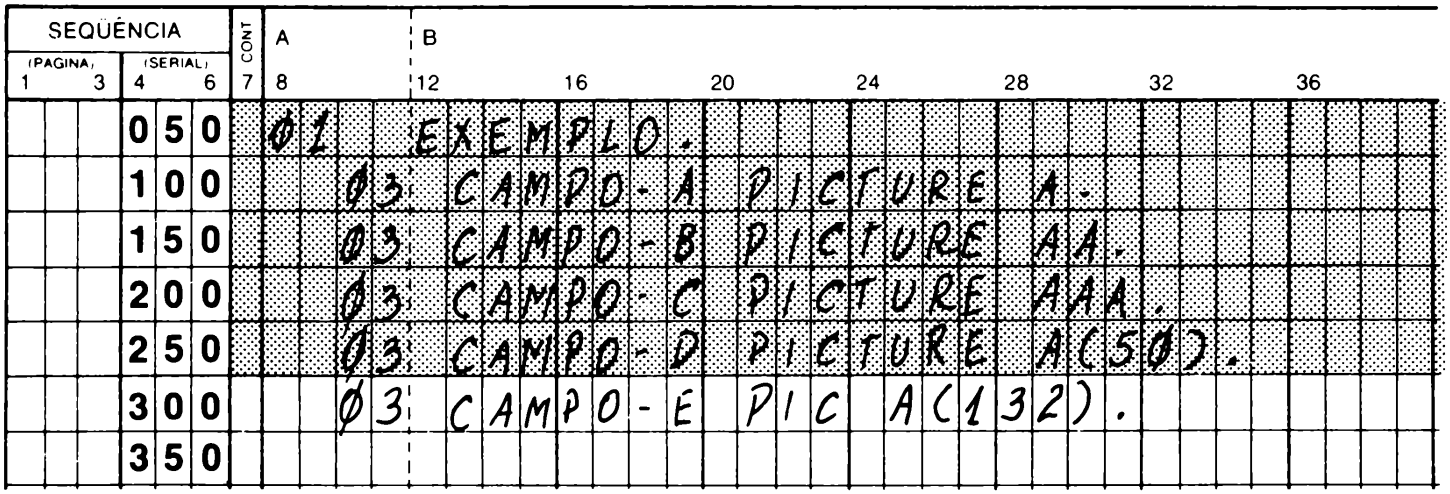
Os caracteres que podem ser usados no formato são:
A B S V X Z 9 0 , . + – CR DB * S

A cláusula PICTURE é usada, entre outras funções, para indicar o tamanho do item de dados. Veremos a seguir qual a função de cada caráter usado no campo formato e como indicar o tamanho do item que está sendo definido.

CARÁTER A

Cada caráter A usado no campo formato da cláusula PICTURE indica que aquela posição pode conter uma letra ou um espaço.

Exemplo:



Neste exemplo, o CAMPO-A foi definido como um campo que pode conter um dado alfabético. O fato de ter sido indicado apenas um caráter A significa que o campo é de uma posição, isto é, o tamanho do campo é de 1 byte.

O CAMPO-B, por ter sido definido como AA (dois caracteres A) poderá conter dois caracteres alfabéticos. Fica fácil compreender, após estes dois exemplos, que a quantidade de A indicados na cláusula PICTURE indica o tamanho do campo que está sendo definido.

O CAMPO-C, definido no exemplo como AAA, poderá conter 3 caracteres alfabéticos.

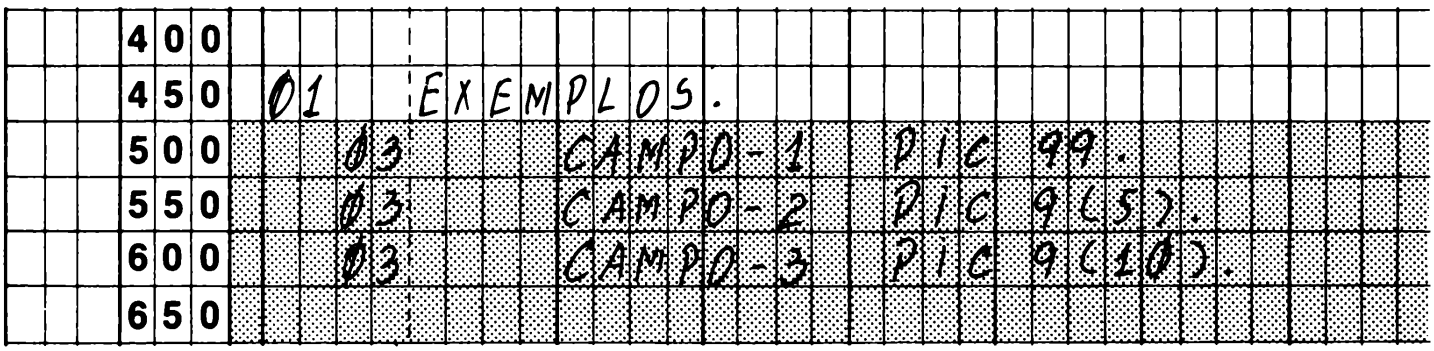
Caso seja necessário definir um campo de tamanho muito grande não há necessidade de escrever uma fileira de A corres-

pondente à quantidade de caracteres que o campo poderá conter. Para evitar este trabalho ao programador, o COBOL permite que a quantidade de vezes que o caráter deve ser repetido seja colocado dentro de parênteses. No nosso exemplo, o número 50 escrito dentro dos parênteses indica que o CAMPO-D poderá conter 50 caracteres alfabéticos. O CAMPO-E será um campo alfabético de 132 posições.

CARÁTER 9

O caráter 9, quando usado no campo formato da cláusula PICTURE, indica que aquela posição do campo deverá conter um dado numérico. As observações feitas a respeito do tamanho nos exemplos do caráter A são válidos para o caráter 9.

Exemplo:



Neste exemplo, o CAMPO-1 foi definido como um campo que pode conter dois caracteres numéricos. O CAMPO-2 foi definido como um campo de 5 caracteres numéricos. O CAMPO-3 foi definido como um campo de 10 caracteres numéricos.

O CARÁTER X

O caráter X, quando usado no campo formato da cláusula PICTURE, indica que o campo poderá conter qualquer caráter alfanumérico, isto é, poderá conter quaisquer caracteres do conjunto EBCDIC.

Exemplo:

```
700
750    01  EXEMPLO-X.
800      03  CAMPO-1  PIC  X.
850      03  CAMPO-2  PIC  XXXXX.
900      03  CAMPO-3  PIC  X(80).
950
```

Neste exemplo, o CAMPO-1 foi definido como um campo de uma posição capaz de conter um dado alfabético, numérico ou um caráter especial.

O CAMPO-2 foi definido como um campo de 5 posições capaz de conter qualquer dado alfanumérico. O CAMPO-3 foi definido como um campo capaz de conter 80 caracteres alfanuméricos.

CARÁTER S

O caráter S serve para indicar se o campo será ou não sinalizado, isto é, se ele deverá ser considerado positivo ou negativo. Maiores detalhes serão fornecidos no estudo da cláusula VALUE.

Exemplo:

```
SEQÜÊNCIA (PAGINA) 1 3 (SERIAL) 4 6   CONT 7   A 8   B 12   16   20   24   28   32   36
050    01  SINALIZA.
100      03  CAMPO-A  PIC  999.
150      03  CAMPO-B  PIC  9.
200
```

Neste exemplo, o CAMPO-A foi definido como um campo numérico de 3 posições, sinalizado. O CAMPO-B foi definido como um campo numérico de uma posição, sinalizado.

Observações:

a) O caráter S deverá ser escrito na posição mais à esquerda do campo formato.
Exemplos: S9, S999, S9(5), S9(10)

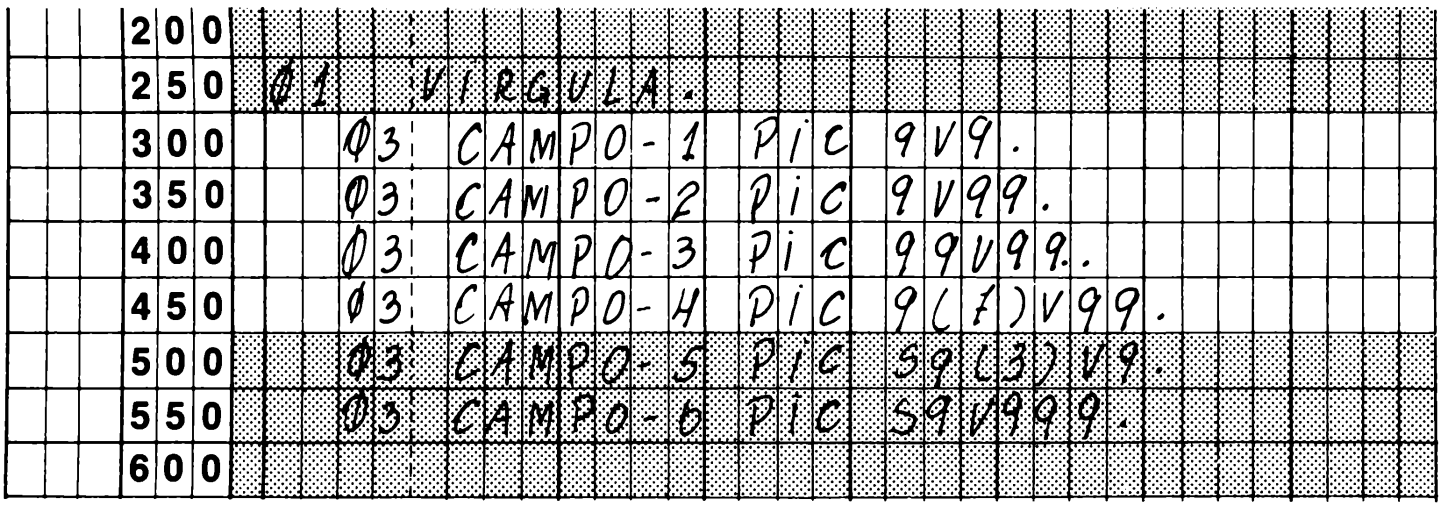
b) Um campo numérico, sem sinal, de uma posição, isto é, PIC 9, poderá conter valores de 0 a 9. Se a PICTURE for 999, poderá conter valores de 0 a 999. Se for 9(4), ou seja, 9999, poderá conter valores de 0 a 9999 e assim por diante.

c) Se os campos anteriormente citados fossem sinalizados, isto é, se a cláusula PICTURE fosse escrita com S, os valores que os campos poderiam conter seriam:

S9	— de −9 a +9.
S99	— de −99 a +99.
S999	— de −999 a +999.
S9(4)	— de −9999 a +9999.

CARÁTER V

O caráter V quando usado no campo formato da cláusula PICTURE serve para indicar a posição da vírgula decimal. A vírgula decimal não ocupará espaço na memória nem influirá no tamanho do campo definido. Ela ficará apenas subentendida.



Segue abaixo uma descrição dos diversos campos definidos neste exemplo:

CAMPO-1: Definido como um campo de um inteiro e um decimal. Pode conter de 0,0 a 9,9.

CAMPO-2: Definido como um campo de um inteiro e dois decimais. Pode conter de 0,00 a 9,99.

CAMPO-3: Definido como um campo de dois inteiros e dois decimais. Pode conter de 0,00 a 99,99.

CAMPO-4: Definido como um campo de 7 inteiros e dois decimais. Pode conter de 0,00 a 9.999.999,99. A posição da vírgula decimal fica indicada pela posição do caráter V. Os pontos não fazem parte do campo-4. O computador trabalhará, sempre, com os valores sem considerar os pontos. Os pontos só serão necessários no caso de os valores serem impressos em relatórios. Neste caso há uma operação especial para a inserção de pontos, vírgulas, sinais etc. Esta operação especial é chamada de *edição.* Estudaremos a EDIÇÃO de campos logo após o final da explicação dos CAMPOS 5 e 6.

CAMPO-5: Definido como um campo sinalizado de 3 inteiros e um decimal. Pode conter de −999,9 a +999,9.

CAMPO-6: Definido como um campo sinalizado de um inteiro e 3 decimais. Pode conter de −9,999 a +9,999.

EDIÇÃO

Nem sempre os dados são processados no mesmo formato que os vemos impressos. Por exemplo, um acumulador de valores é processado sem os pontos que vemos impressos nos relatórios. Além disso, se fôssemos imprimir o conteúdo do acumulador diretamente, sem nenhum cuidado especial, é possível que os valores fossem impressos com vários zeros à esquerda, o que não é comum em relatórios a serem manuseados pelo homem. Para imprimir os dados no formato apropriado para a leitura humana, há uma operação especial chamada EDIÇÃO.

A edição é feita por meio de símbolos especiais chamados símbolos de edição.

O processo é muito simples: cria-se um campo por meio de uma cláusula PICTURE onde, no campo de formato, são colocados os símbolos de edição. Ao mover os dados a serem impressos para o campo onde estão os símbolos de edição, estes símbolos vão operando uma transformação na disposição dos dados a serem impressos, sem, no entanto, alterar o conteúdo do campo original.

Exemplo:

Suponhamos um CAMPO-A cujo conteúdo é 0003757800. Suponhamos, ainda, que o CAMPO-A contivesse o resultado de uma série de operações com valores em cruzeiros. Se fôssemos imprimir o CAMPO-A sem editá-lo, no relatório sairia impresso 003757800. Devemos editá-lo para que saia impresso no relatório: 37.578,00.

Vimos anteriormente que os caracteres que podem ser usados no campo de formato de uma cláusula PICTURE são:
A B S V X Z 9 0 , . + − CR DB * $

Vimos, também, a função dos caracteres A 9 X S V. Os demais caracteres são símbolos de edição, usados no campo de formato da cláusula PICTURE quando desejamos fazer a edição de um campo, isto é, quando desejamos imprimir um campo num formato apropriado para a leitura humana.

Veremos a seguir a função de cada símbolo de edição, quando usado no campo de formato da cláusula PICTURE.

CARÁTER B

O caráter B provoca a inserção de um espaço no campo editado.

Exemplos:

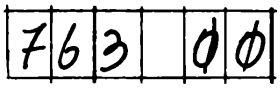
1) Conteúdo do CAMPO-A: 

CAMPO-B: 

Se movêssemos o conteúdo do CAMPO-A para o CAMPO-B, o resultado seria: 

2) Conteúdo de VALOR: 

CAMPO-H: 

Se movêssemos o conteúdo do campo VALOR para o CAMPO-H, o resultado seria: 

CARÁTER Z

O caráter Z indica supressão de zeros à esquerda. Para cada zero a ser suprimido deverá ser escrito um Z na PICTURE de edição. Se na posição correspondente ao Z houver um zero a ser suprimido, ele será suprimido. Se não for zero, o caráter será mantido.

Exemplos:
1) Conteúdo do CAMPO-A: 000360.
CAMPO-E: ZZZ999.
Se movêssemos o CAMPO-A para o CAMPO-E, o resultado seria: 360.

2) Conteúdo de FIELD-60: 0368375
FIELD-3: ZZZZZZ9.
Se movêssemos o FIELD-60 para o FIELD-3, o resultado seria: 368375.

CARÁTER Ø

O caráter Ø provoca a inserção de um caráter Ø na posição correspondente no campo impresso.

Exemplos:
1) Conteúdo do CAMPO-J: PALAVRA
CAMPO-K: ØØØXXXXXXX
Se movêssemos o CAMPO-J para o CAMPO-K, o resultado seria: ØØØPALAVRA

2) Conteúdo de AA: 7615
CAMPO-BB: 9Ø9Ø9Ø9ØØ
Se movêssemos o campo AA para CAMPO-BB, o resultado seria: 7Ø6Ø1Ø5ØØ

CARÁTER (,)

O caráter (,) provoca a inserção de um caráter (,) na posição correspondente no campo impresso.
Exemplo:

Conteúdo do CAMPO1: 76333
CAMPO33: 999,99
Se movêssemos o CAMPO1 para o CAMPO33, o resultado seria: 763,33

CARÁTER (.)

O caráter (.) provoca a inserção de um caráter (.) na posição correspondente no campo impresso.
Exemplos:

1) Conteúdo do CAMPOA: 8736813
CAMPOK: 99.999,99.
Se movêssemos o CAMPOA para o CAMPOK, o resultado seria: 87.368,13

2) Conteúdo do CAMPO1: 1234567
CAMPO-B: 9.999.999
Se movêssemos o CAMPO1 para o CAMPO-B, o resultado seria: 1.234.567

CARÁTER (+)

O caráter (+) provoca a inserção de um caráter (+) na posição correspondente no campo impresso, desde que o campo de origem seja POSITIVO. Se o campo de origem for negativo, haverá a inserção do caráter (−).

Exemplos:

1) Conteúdo do CAMPO-1: +360
CAMPO-2: 999+
Se movêssemos o CAMPO-1 para o CAMPO-2, o resultado seria: 360+

2) CAMPO-1: −360
CAMPO-2: 999+
Após o movimento do CAMPO-1 para o CAMPO-2, o resultado seria: 360−

CARÁTER (−)

O caráter (−) provoca a inserção de um caráter (−) na posição correspondente no campo impresso, desde que o campo de origem seja NEGATIVO. Se o campo de origem for positivo, haverá a inserção de um espaço.

Exemplos:

1) CAMPO-1: 1000−
CAMPO-2: 9999−
Após o movimento do CAMPO-1 para o CAMPO-2, o resultado seria: 1000−

2) Conteúdo do CAMPO-1: 31+
CAMPO-2: 99−
Após o movimento do CAMPO-1 para o CAMPO-2, o resultado seria: 31

CARACTERES CR e DB

Os caracteres DB e CR provocam a inserção dos caracteres DB e CR, respectivamente, desde que o campo de origem seja NEGATIVO. Se o campo de origem for positivo, haverá a inserção de 2 espaços.

Exemplos:

1) CAMPO-1: 300−
CAMPO-2: 999CR
Após o movimento do CAMPO-1 para o CAMPO-2, o resultado seria: 300CR

2) CAMPO-1: 61+
CAMPO-2: 99DB
Após o movimento do CAMPO-1 para o CAMPO-2, o resultado seria: 61

CARÁTER (*)

O caráter (*) provoca a substituição de zeros à esquerda por (*). Cada (*) escrito provocará a substituição de um zero à esquerda.

Exemplos:

1) Conteúdo do CAMPO-1: 000301
CAMPO-2: *****9
Após o movimento do CAMPO-1 para o CAMPO-2, o resultado seria: ***301

2) Conteúdo do CAMPO-1: 008000
CAMPO-2: *99999
Após o movimento do CAMPO-1 para o CAMPO-2, o resultado seria: *08000

CARÁTER $

O caráter $ provoca a inserção do caráter $ na posição correspondente do campo impresso.

Exemplo:

Conteúdo do CAMPO-1: 10000
CAMPO-2: $999,99
Após o movimento do CAMPO-1 para o CAMPO-2, o resultado seria: $100,00

Se for desejada a troca do símbolo $ por outro símbolo, basta escrever a cláusula CURRENCY SIGN no parágrafo SPECIAL-NAMES da ENVIRONMENT DIVISION.

SYNCHRONIZED

A cláusula SYNCHRONIZED é opcional. Ela tem por função otimizar operações aritméticas. Quando usada, os itens elementares são colocados no alinhamento adequado na memória, quando da execução do programa.

O formato da cláusula SYNCHRONIZED é:

FORMATO

```
{SYNCHRONIZED}        [LEFT ]
{SYNC        }        [RIGHT]
```

USAGE

A cláusula USAGE indica o modo em que os dados serão representados na memória. É através desta cláusula que informamos ao compilador se os dados estarão em decimal compactado, binário, display etc.

O formato da cláusula USAGE é:

FORMATO

```
[USAGE IS] { DISPLAY
           { COMPUTATIONAL  }
           { COMP           }
           { COMPUTATIONAL-1 }
           { COMP-1          }
           { COMPUTATIONAL-2 }
           { COMP-2          }
           { COMPUTATIONAL-3 }
           { COMP-3          }
```

Se não for fornecida nenhuma das opções anteriores o compilador assumirá DISPLAY.

OPÇÃO DISPLAY DA CLÁUSULA USAGE

Indica que os dados estarão no formato que corresponde à entrada por cartão perfurado ou impressos num relatório através de impressora. Isto é, cada caráter ocupará um byte e será lido de uma coluna do cartão ou impresso numa posição do relatório. Os dados neste formato são conhecidos por decimal zonado.

Exemplo:

```
01 CARTÃO.
  03 CAMPO1 PICTURE X(7) USAGE IS DISPLAY.
  03 CAMPO2 PIC X(7).
```

No exemplo acima CAMPO1 e CAMPO2 possuem a mesma USAGE porque, quando for omitida a opção, o compilador assumirá DISPLAY.

OPÇÃO COMPUTATIONAL DA CLÁUSULA USAGE

Esta opção é especificada para indicar itens binários. A quantidade de memória ocupada por um item binário dependerá do número de dígitos decimais definidos na cláusula PICTURE, como apresentado abaixo:

Cláusula PICTURE	Memória ocupada
de 1 a 4 dígitos	2 bytes
de 5 a 9 dígitos	4 bytes
de 10 a 18 dígitos	8 bytes

O bit mais à esquerda na memória será o sinal.

Exemplo: (1) 1 1 1 0 0 1 1

SINAL

OPÇÕES COMPUTATIONAL-1, COMP-1, COMPUTATIONAL-2 e COMP-2

Estas opções são usadas para itens de ponto flutuante.

OPÇÃO COMPUTATIONAL-3 ou COMP-3 DA CLÁUSULA USAGE

Esta opção é especificada para indicar itens que conterão dados em decimal compactado.

A PICTURE será obrigatoriamente composta de noves e não poderá exceder 18 dígitos decimais.
Exemplo: 77 ACUMULADOR PIC 9(7) USAGE IS COMP-3.

O exemplo está especificando um campo chamado ACUMULADOR de 7 bytes que deverá conter dados em decimal compactado.

Exemplos da cláusula USAGE:
1) 01 CAMPOS.
03 A PIC X(9).
03 B PIC S9(3) USAGE IS COMP-3.
03 C PIC 9(6) USAGE IS COMP.
Neste exemplo foram apresentados 3 campos (A, B e C) com cláusulas USAGE diferentes (DISPLAY, COMP-3 e COMP respectivamente).
2) 01 CAMPOS.
03 A PIC 999 USAGE IS COMP-3.
03 B PIC 9999 USAGE IS COMP-3.
03 C PIC 9(5) USAGE IS COMP-3.

3) 01 CAMPOS USAGE IS COMP-3.
03 A PIC 999.
03 B PIC 9999.
03 C PIC 9(5).

Os exemplos 2 e 3 se equivalem, porque a cláusula USAGE quando usada num item de grupo (como no exemplo 3) ela é válida para todos os itens elementares do mesmo grupo.

VALUE

A cláusula VALUE é usada para definir o valor inicial de um item da WORKING-STORAGE (estudada adiante) ou o valor associado a um nome-de-condição (condition-name).

A cláusula VALUE admite os dois formatos que seguem:

FORMATO 1

VALUE IS literal

FORMATO 2

```
{VALUE IS  }
{VALUES ARE}  literal-1      [THRU literal-2]

              [literal-3     [THRU literal-4]] ...
```

A cláusula só pode ser especificada item com tamanho fixo.

LITERAL

No formato 1 da cláusula VALUE, literal será o valor inicial a ser atribuído ao campo.

Certos tipos de dados constantes, invariáveis durante todo o processamento do programa, são chamados de LITERAL. Outros tipos de dados constantes são chamados de CONSTANTES FIGURATIVAS.

As constantes figurativas podem substituir qualquer literal na cláusula VALUE.

Seguem abaixo algumas explicações sobre literais e constantes figurativas.

LITERAIS

Os literais são de dois tipos: numéricos e não numéricos.

LITERAL NUMÉRICO

Um literal numérico é definido por um conjunto de caracteres escolhidos entre os dígitos ∅ a 9, o sinal (+), o sinal (−) e a vírgula decimal.

Estas regras se aplicam aos literais numéricos:

1) Deve conter de 1 a 18 dígitos.

Exemplos:

literal a) 73
literal b) 807
literal c) 8473156
literal d) 8019487656712
literal e) 3

2) Não pode conter mais de um caráter de sinal. Se o sinal for usado, ele deve aparecer na posição mais à esquerda do literal.

Exemplos:
literal a) + 300
literal b) + 7
literal c) − 9
literal d) − 10
literal e) − 300

3) Se um literal numérico não for sinalizado, o literal será considerado positivo.

Exemplos:
literal a) + 300 (positivo)
literal b) − 300 (negativo)
literal c) 300 (positivo, por não ter sinal)

4) Não pode ter mais de uma vírgula decimal e a vírgula decimal não pode aparecer na posição mais à direita do literal. Se o literal não possuir vírgula decimal, será considerado inteiro.

Exemplos:
literal a) 77,6 (correto)
literal b) 80,00 (correto)
literal c) 80, (errado)
literal d) 0,8 (correto)
literal e) 800 (inteiro, correto)
literal f) 08 (inteiro, correto)

LITERAL NÃO NUMÉRICO

Um literal não numérico é definido por um conjunto de caracteres escolhidos entre todos os caracteres EBCDIC, exceto o apóstrofo. Um literal não numérico pode ser constituído de 1 a 120 caracteres. *O literal não numérico deve ser escrito entre apóstrofos.* Qualquer espaço dentro dos apóstrofos é considerado parte do literal não numérico.

Exemplos:
a) 'PÁGINA'
b) 'RELATÓRIO'

c) 'NOME DO FUNCIONÁRIO * VALOR'
d) 'A'

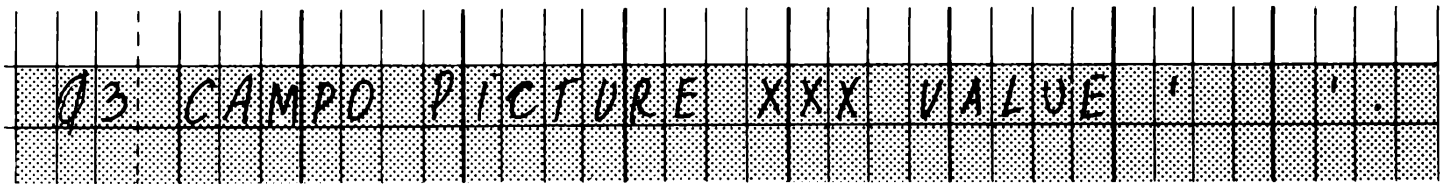
CONSTANTES FIGURATIVAS

Alguns dados que são constantes em todo o processamento podem ser definidos através de palavras reservadas especialmente para este fim. Estas palavras são as CONSTANTES FIGURATIVAS.

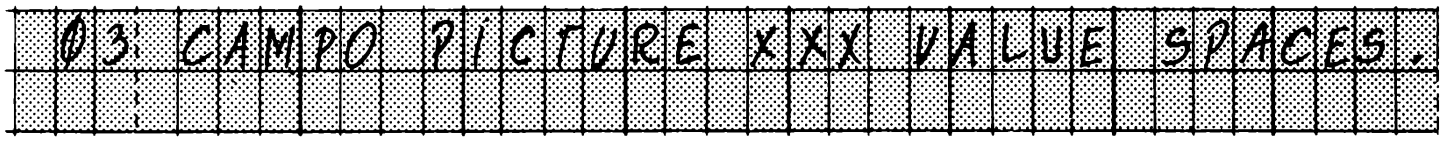
Por exemplo: Se for desejado colocar espaços num campo qualquer de 3 posições, poderemos recorrer a um dos recursos abaixo:

a) Usar o literal não numérico.
b) Usar uma constante figurativa.

Para usar o literal não numérico, escreveríamos:



Para usar a constante figurativa escreveríamos:



A palavra SPACES é uma constante figurativa que representa um ou mais espaços.

Segue abaixo uma relação das constantes figurativas e suas funções. A forma singular ou plural são equivalentes e podem ser usadas com o mesmo resultado.

Uma constante figurativa pode ser usada no lugar de um literal sempre que um literal aparecer num formato, exceto quando o literal tiver de ser numérico. Neste caso, a única constante figurativa permitida será ZERO (ZEROES ou ZEROS).

CONSTANTE FIGURATIVA — ZERO ou ZEROES ou ZEROS

FUNÇÃO

Representa um ou mais zeros, dependendo do tamanho do campo.

Exemplos:

a) 03 CAMPO PIC 9 VALUE ZERO.
b) 03 CAMPO PIC 9 (7) VALUE ZERO.
c) 03 CAMPO PIC 9 (7) VALUE ZEROS.

No exemplo *a* a constante figurativa ZERO representa ∅. Equivale ao literal ∅. No exemplo *b* a constante figurativa ZERO representa ∅∅∅∅∅∅∅ equivalente ao literal ∅∅∅∅∅∅∅. No exemplo *c* a constante figurativa ZEROS tem o mesmo significado da constante figurativa do exemplo *b.*

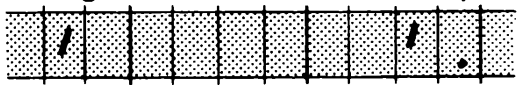
CONSTANTE FIGURATIVA — SPACE ou SPACES

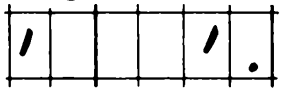
FUNÇÃO

Representa um ou mais espaços, dependendo do tamanho do campo.

Exemplos:

a) 03 CAMPO PIC X(7) VALUE SPACES.
b) 03 CAMPO PIC XXX VALUE SPACE.

No exemplo *a* a constante figurativa SPACES representa 7 espaços. Equivale ao literal 

No exemplo *b* a constante figurativa SPACE representa 3 espaços, equivalendo ao literal 

CONSTANTE FIGURATIVA — HIGH-VALUE ou HIGH-VALUES

FUNÇÃO

Provoca o preenchimento dos bytes do campo com o valor FF, que é o maior valor para efeito de SORT.

LOW-VALUE ou LOW-VALUES

Provoca o preenchimento dos bytes do campo com o valor binário ØØ, que é o menor valor para efeito de SORT.

CONSTANTE FIGURATIVA — QUOTE ou QUOTES

FUNÇÃO

Representa um ou mais apóstrofos, dependendo do tamanho do campo.

ALL literal

Representa a ocorrência de um ou mais caracteres especificados no literal, dependendo do tamanho do campo.

Exemplos:

a) 03 CAMPO PIC X(30) VALUE ALL '*'.
Neste exemplo, a constante figurativa ALL representa 30 asteriscos, que preencherão o CAMPO.

b) 03 CAMPO PIC X(10) VALUE ALL 'AB'.
Neste exemplo, a constante figurativa ALL representa os caracteres ABABABABAB, que preencherão o CAMPO. Equivale ao literal 'ABABABABAB'.

Vejamos agora o formato 2 da cláusula VALUE:

FORMATO 2

```
{ VALUE IS  }   literal-1      [THRU literal-2]
{ VALUES ARE}
                [literal-3     [THRU literal-4]]...
```

O formato 2 da cláusula VALUE é usado para descrever um nome-de-condição (condition-name).

Um nome-de-condição é um nome qualquer que pode ser diretamente relacionado a condições previamente estabelecidas.

Por exemplo:

Podemos estabelecer que a palavra MAIORIDADE seja válida num comando qualquer para substituir todas as idades superiores a 20 anos. Assim, ao dar uma instrução não haverá necessidade de testar a idade. Bastará escrever: IF MAIORIDADE... (IF, em inglês, significa SE).

Outro exemplo:

Um analista poderia definir que a ocorrência de números ímpares entre 0 e 10 num determinado campo provocaria a necessidade de ativar uma determinada rotina do programa. Se não houvesse o recurso do nome-de-condição uma das soluções seria testar:

```
IF CAMPO = 1
IF CAMPO = 3
IF CAMPO = 5
IF CAMPO = 7
IF CAMPO = 9
```

Com o uso do nome-de-condição bastaria comandar: IF IMPAR ...

NOME-DE-CONDIÇÃO

Um nome-de-condição é identificado por um número de nível 88.

Um nome-de-condição pode ser associado a qualquer item elementar ou item de grupo, exceto a:

1. Um número de nível 66.

2. Um grupo com descrições que contenha cláusulas JUSTIFIED, SYNCHRONIZED ou USAGE diferente de DISPLAY.

Exemplos de nome-de-condição:

```
03 CAMPO PICTURE 9.
 88 MASCULINO VALUE 1.
 88 FEMININO VALUE 2.
```

RENAMES

Às vezes há necessidade de o programador se referir, por outro nome, a um grupo de dados definidos previamente.

A cláusula RENAMES supre esta necessidade.

O formato da cláusula RENAMES é:

FORMATO

```
66 nome-de-dados RENAMES nome-de-dados-2

                         [THRU nome-de-dados-3]
```

Podem ser escritas mais de uma entrada RENAMES para um mesmo registro lógico.

Todas as cláusulas RENAMES, associadas a um registro lógico, devem ser escritas imediatamente após a descrição do último campo do registro.

Devem ser tomados os seguintes cuidados:

1. Nome-de-dados-2 e nome-de-dados-3 devem pertencer ao mesmo registro lógico e não podem ter o mesmo nome.

2. Nome-de-dados-3 não pode ser subordinado ao nome-de-dados-2.

3. Quando nome-de-dados-3 não for especificado:

a) Se nome-de-dados-2 for um item de grupo, nome-de-dados será considerado um item de grupo.

b) Se nome-de-dados-2 for um item elementar, o nome-de-dados será considerado um item elementar.

4. Quando nome-de-dados-3 for especificado:
a) Nome-de-dados é um grupo item que incluirá todos os itens elementares:
— começando com nome-de-dados-2, se este item for um item elementar.
b) Começando com o primeiro item elementar dentro do nome-de-dados-2, se este for um item de grupo.
c) Terminando com nome-de-dados-3, se este for um item elementar.
d) Terminando com o último item elementar dentro do nome-de-dados-3, se este for um item de grupo.

5. Um número de nível 66 (cláusula RENAMES) não pode renomear outro nível 66, nem nível 77, nem nível 88, nem nível 01.

Exemplos de cláusula RENAMES:

Exemplo:

```
a) 01 REGISTRO.
     03 NOME PIC X(30).
     03 ENDEREÇO.
      05 RUA PIC X(20).
      05 BAIRRO PIC X(10).
     03 CAMPOS-DIVERSOS.
      05 CAMPO1 PIC 9(7).
      05 CAMPO2 PIC 9(5).
66 NOME-E-ENDEREÇO RENAMES NOME THRU ENDEREÇO.
```

Neste exemplo, o campo NOME-E-ENDEREÇO engloba os campos NOME, RUA e BAIRRO.

Exemplo:

b) Usando a definição do REGISTRO apresentado no exemplo *a*:

```
66 OUTRO-DADO RENAMES ENDEREÇO.
```

Neste exemplo, o campo ENDEREÇO pode ser referenciado por "ENDEREÇO" ou por "OUTRO-DADO", pois ambos os nomes representam a mesma área.

WORKING-STORAGE SECTION

Nesta seção são descritas as áreas de processamento interno, que não são utilizadas em arquivos de entrada e saída.

O nome de seção WORKING-STORAGE SECTION deve ser perfurado na margem A (colunas 8/11).

Os itens de grupo de nível 01 também devem ser perfurados na margem A, bem como os itens cujo número de nível seja 77. Os itens que devem ter número de nível 77 são estudados a seguir.

NÚMERO DE NÍVEL 77

Os itens que devem ter número de nível 77 são os itens elementares que não estejam diretamente relacionados a nenhum outro item.

Por exemplo: Um acumulador usado para numeração de folhas de um relatório. O acumulador não faz parte de nenhum arquivo de entrada ou saída. Ele é apenas uma área auxiliar onde são acumulados os números das folhas, os quais serão editados numa linha a ser impressa.

A definição do acumulador de folhas poderia ser assim:

```
77 ACUM-DE-FOLHA PICTURE 9(5) VALUE ∅.
```

Observe que foi fornecido um valor inicial para o acumulador. No caso do exemplo foi indicado o valor zero para inicializar o acumulador. Qualquer área reservada na WORKING-STORAGE SECTION cujo valor inicial possa influir no resultado do processamento, deve ser inicializada por meio do parâmetro VALUE.

Exemplos:

```
77 EMPRESA PIC X(13) VALUE 'EMPRESA X S/A'.
77 VALOR PIC 9(5)V99 VALUE ∅.
77 ÁREA-AUXILIAR PIC X(10) VALUE SPACES.
```

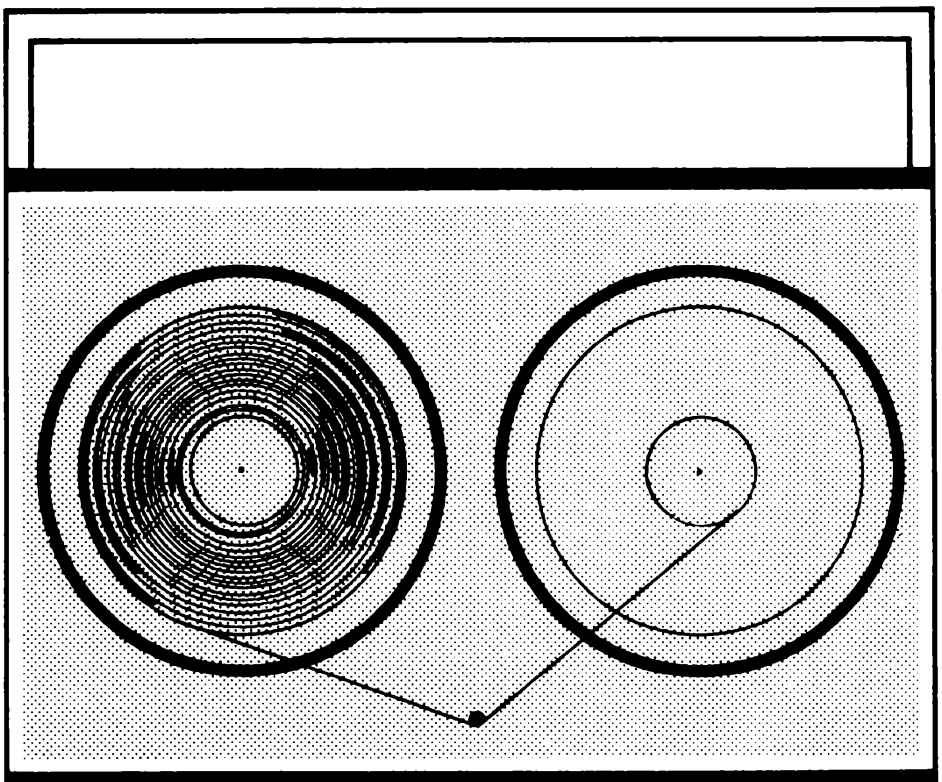
Há outro meio, menos prático, de inicializar as áreas auxiliares. Em vez de escrever a cláusula VALUE, o dado inicial do campo seria colocado através de um comando MOVE, a ser estudado na PROCEDURE DIVISION.

ITENS DE GRUPO NA WORKING-STORAGE SECTION

É permitida a codificação de itens de grupo na WORKING-STORAGE SECTION. Os itens de número de nível 77 devem, no entanto, ser escritos, todos eles, antes dos itens de grupo.

Exemplo:

```
WORKING-STORAGE SECTION.
77 ACUM-PÁGINA PIC 9(5) VALUE Ø.
77 EMPRESA PIC X(13) VALUE 'EMPRESA X S/A'.
77 ÁREA-AUXILIAR PIC X(1Ø) VALUE SPACES.
Ø1 CABEÇALHO.
  Ø3 NOME-EMPRESA PIC X(3Ø).
  Ø3 NOME-RELATÓRIO PIC X(2Ø).
  Ø3 FILLER PIC XXX VALUE 'FL.'.
  Ø3 NÚMERO-FOLHA PIC ZZ.Z99.
Ø1 LINHA.
  Ø3 SALTO PIC X.
  Ø3 DADOS PIC X(132).
```



CAPÍTULO 7

Procedure Division

*

* *

PROCEDURE DIVISION

A PROCEDURE DIVISION é usada para indicar ao computador quais as instruções que deverão ser executadas para resolver o problema proposto. Com o uso de um fluxograma (ver capítulo 2) fica fácil codificar a PROCEDURE DIVISION de um programa.

É na PROCEDURE DIVISION que serão dados os comandos de leitura, movimento de dados de um campo para outro, operações aritméticas e lógicas, gravação de arquivos, perfuração de cartões etc.

A PROCEDURE DIVISION é dividida em parágrafos. Cada parágrafo é dividido em comandos ou declarações.

Exemplo:

```
PROCEDURE DIVISION.
PARÁGRAFO-1.                                PARÁGRAFO-2.
     COMANDO A.                                  COMANDO E.
     COMANDO B.   COMANDO C.                     COMANDO F.
     COMANDO D.                             PARÁGRAFO-3.
                                                 COMANDO G.
```

As palavras PROCEDURE DIVISION têm de ser escritas na margem A, no início da codificação desta divisão. Os nomes dos parágrafos (no exemplo chamados de PARÁGRAFO-1, PARÁGRAFO-2 e PARÁGRAFO-3) podem ser escolhidos à vontade pelo programador, desde que siga as regras de formação de palavras já estudadas.

Os nomes dos parágrafos têm de ser escritos na margem A.

Os comandos têm de ser escritos na margem B.

Podem ser escritos mais de um comando na mesma linha.

Um comando pode ter uma ou mais linhas de continuação. Não há necessidade de perfurar o hífen (-) na coluna 7 da linha continuação. Não pode, no entanto, dividir nenhuma palavra do comando.

Exemplo: (O comando MOVE apresentado no exemplo será estudado mais adiante. No momento basta compreender que um comando pode ser continuado em outra linha, desde que não se divida nenhuma palavra do comando.)

Exemplo 1:

```
MOVE CAMPO-A TO CAMPO-B. MOVE
CAMPO-C TO CAMPO-D.
```

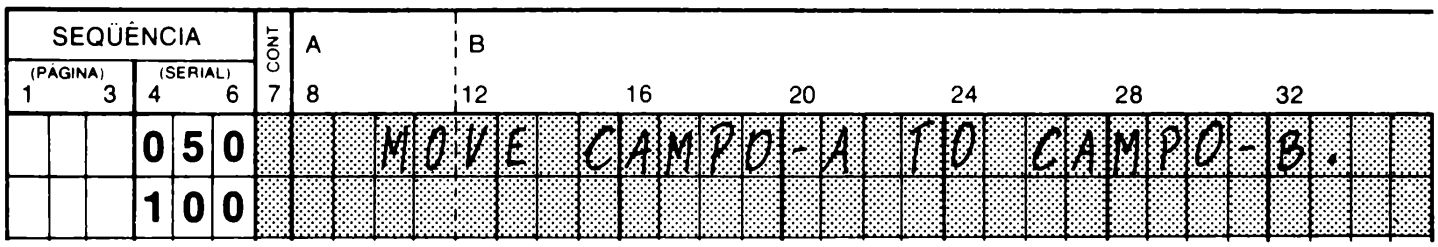
O exemplo acima está correto porque não infringe nenhuma das normas indicadas.

Exemplo 2:

```
MOVE CAMPO-A TO CAM-
    PO-B.
```

O exemplo acima está incorreto porque a palavra CAMPO-B foi dividida.

Exemplo 3:



O exemplo 3 está incorreto porque a perfuração não pode começar na margem A.

Para facilidade de manutenção do programa, geralmente cada comando é escrito em uma linha, o que torna mais clara a apresentação do programa.
Exemplo:

```
MOVE CAMPO-A TO CAMPO-B.
MOVE CAMPO-C TO CAMPO-D.
MOVE CAMPO-E TO CAMPO-F.
```

EXPRESSÕES ARITMÉTICAS

Alguns comandos usam expressões aritméticas como operandos.

Uma expressão aritmética pode ser:

1. Um item elementar.
2. Um literal numérico.
3. Itens elementares e literais numéricos separados por sinais aritméticos.
   Os sinais aritméticos permitidos são:
   + (adição)
   − (subtração)
   * (multiplicação)
   / (divisão)
   ** (exponenciação)

É permitido, também, o uso de parênteses para especificar a ordem em que os operandos deverão ser processados.

4. Qualquer expressão pode ser precedida do sinal (+) ou do sinal (−).

CONDICIONAIS

O uso de uma condição indica ao programa-objeto que será feita uma escolha entre várias opções, dependendo do resultado de um teste.
Por exemplo: O programador poderá ter necessidade de ativar uma rotina A ou uma rotina B, dependendo de um resultado ser positivo ou negativo.

Os testes de condição são usados nos comandos IF e PERFORM estudados mais adiante.

As condições podem ser:

— Condição de classe.
— Condição de nome-de-condição.
— Condição de relação.
— Condição de sinal.

O resultado de um teste poderá ser verdadeiro ou falso, dependendo das circunstâncias existentes no momento em que a instrução for processada pelo computador.

CONDIÇÃO DE CLASSE

O teste de classe determina se o conteúdo de um campo é numérico ou alfabético.

O seu formato é:

FORMATO

```
IF identificador    IS [NOT]  { NUMERIC      }
                              { ALPHABETIC   }
```

Exemplos:

a) IF CAMPO IS NUMERIC

Neste exemplo, se o conteúdo do campo chamado CAMPO for numérico, o resultado do teste será verdadeiro. Se for alfabético, o resultado do teste será falso.

b) IF AREAS NOT NUMERIC

Neste exemplo, se o conteúdo do campo AREAS for numérico, o resultado do teste será falso. Se o conteúdo do campo *não* for numérico, o resultado do teste será verdadeiro.

TESTE DE NOME-DE-CONDIÇÃO

Já estudamos este tipo de teste quando foi descrita a cláusula VALUE.

CONDIÇÃO DE RELAÇÃO

A condição de relação causa uma comparação de dois operandos.

Um símbolo de relação especifica o tipo de comparação a ser feito.

O formato da condição de relação é:

FORMATO

identificador-1 literal-1 expressão-aritmética-1	símbolo de relação	identificador-2 literal-2 expr. aritmética-2

OS SÍMBOLOS DE RELAÇÃO SÃO:

a) IS [*NOT*] *GREATER* THAN ([não] maior que)
IS [NOT] >

b) IS [*NOT*] *LESS* THAN ([não] menor que)
IS [*NOT*]>

c) IS [*NOT*] *EQUAL* TO ([não] igual a)
IS [*NOT*] =

Exemplos:

```
IF CAMPO-A IS GREATER THAN CAMPO-B ...
IF CAMPO-A IS NOT GREATER THAN A * B...
IF CAMPO-A > CAMPO-B ...
IF CAMPO-A > 700 ...
IF CAMPO-B LESS THAN A/B ...
IF CAMPO-H NOT EQUAL TO A - B ...
```

CONDIÇÃO DE SINAL

A condição de sinal determina se o valor de um campo numérico é positivo, negativo ou zero.

O seu formato é:

FORMATO

```
{identificador          }          {POSITIVE }
{expressão-aritmética   }  IS [NOT] {NEGATIVE}
                                    {ZERO     }
```

Exemplos:
IF CAMPO NOT POSITIVE ...
IF CAMPO IS ZERO ...

CONDIÇÕES COMPOSTAS

Com o uso dos símbolos AND e OR, é possível escrever condições compostas.

Exemplos: IF CAMPO + CAMPO-C > CAMPO-R OR
(A + B) IS POSITIVE

IF A = B AND (A − B) > C

IF (A < B) OR (C < D)

O COMANDO IF

O comando IF testa uma condição. A ação seguinte do programa dependerá do resultado do teste da condição. Se o resultado for verdadeiro, a ação será uma. Se o resultado for falso, a ação será outra.

O formato do comando IF é:

FORMATO

```
                  {comando-1       }{ELSE     }{comando-2       }
IF condição       {NEXT SENTENCE   }{OTHERWISE}{NEXT SENTENCE   }
```

As ações tomadas pelo programa são:

1. Se a condição for verdadeira, o comando que segue imediatamente à condição será executado. Se este comando não for um "desvio" (comando para fugir à seqüência normal do programa) o controle do programa será passado para o comando seguinte ao IF.

2. Se a condição for falsa, o controle passa para o comando seguinte ao IF.

A função do parâmetro NEXT SENTENCE é, também, passar o controle do programa para o comando seguinte ao IF.

Exemplos:

a) IF CAMPO-A = CAMPO-B ADD 1 TO C.
ADD 2 TO CAMPO-D.

(O comando ADD, estudado mais adiante, indica uma adição.)

Neste exemplo, se o CAMPO-A for igual ao CAMPO-B, o programa somará 1 ao campo C e depois passa para o comando seguinte, ou seja, somará 2 ao campo D.

Se o CAMPO-A não for igual ao CAMPO-B, o programa passará imediatamente para o comando seguinte ao IF, ou seja, *não* somará 1 ao campo C e passará imediatamente para o comando de somar 2 ao campo D.

b) IF A > B ADD 1 TO C ELSE ADD 5 TO D.
ADD 10 TO E.

Neste exemplo, se A for maior que B:

1. somará 1 ao campo C.
2. somará 10 ao campo E.

Se A for igual ou menor que B:

1. somará 5 ao campo D.
2. somará 10 ao campo E.

c) IF A = B AND A < C NEXT SENTENCE ELSE
ADD 1 TO A.

ADD 3 TO B.
Neste exemplo, só deixará de ser somado 1 ao campo A (ADD 1 TO A) se:
a) o campo A for igual ao campo B e, simultaneamente,
b) o campo A for menor do que o campo C.

COMANDOS
Os principais comandos em COBOL são:

IF (já estudado)
COMPUTE
DIVIDE
MULTIPLY
SUBTRACT
GO TO
ALTER
PERFORM
STOP
EXIT
MOVE
OPEN
READ
WRITE
ACCEPT
DISPLAY
CLOSE

O COMANDO ADD

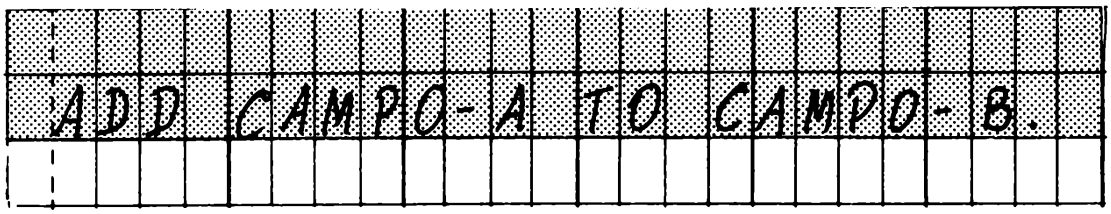
O comando ADD provoca a soma de 2 ou mais operandos. Existem dois formatos principais para o comando ADD:

FORMATO 1

```
ADD { identificador-1 } [ identificador-2 ]
    { literal-1       } [ literal-2       ]  ... TO identificador-m

    [ROUNDED]  [identificador-n  [ROUNDED]]...
    [ON SIZE ERROR  comando]
```

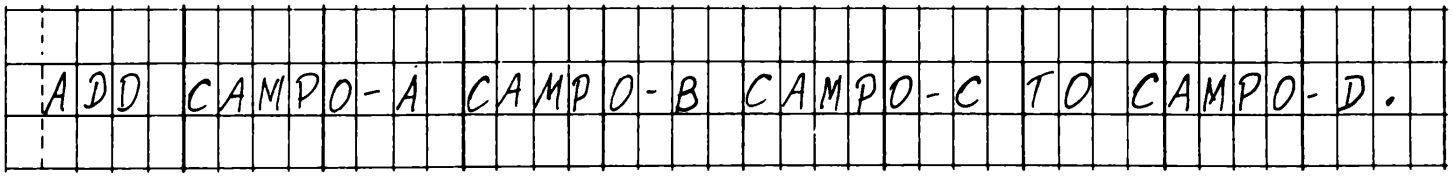
Neste formato, os valores dos operandos que precedem a palavra *TO* são somados juntos e a soma é adicionada aos valores do identificador-m, identificador-n. O resultado é colocado no identificador-m (e nos demais identificadores — identificador-n, identificador-o etc., se forem escritos).
Exemplos:

a)



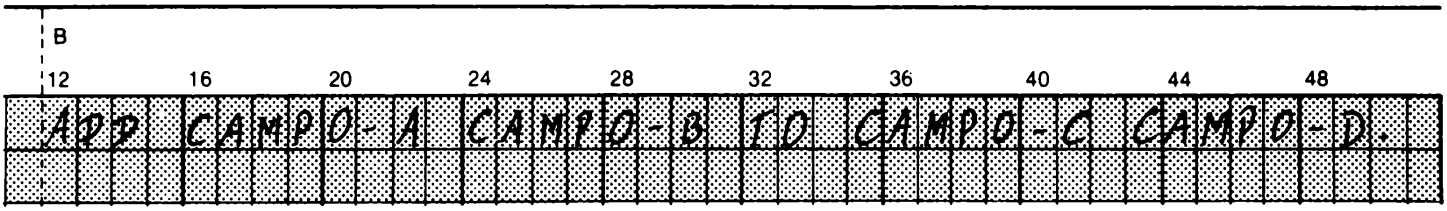
Neste exemplo o conteúdo do CAMPO-A é somado ao conteúdo do CAMPO-B. O resultado fica no CAMPO-B.

b)



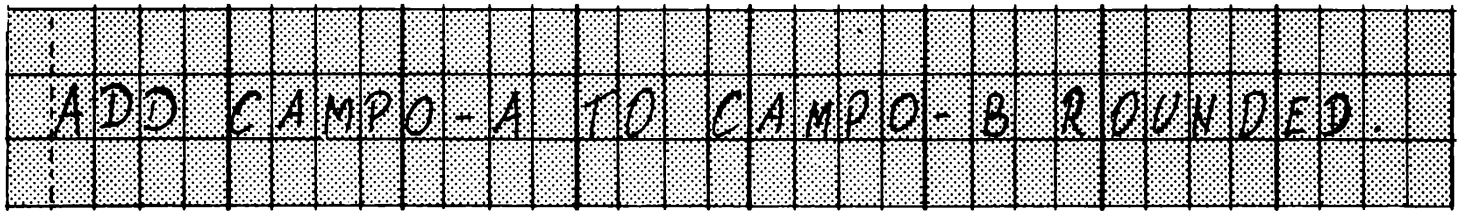
Neste exemplo, os conteúdos dos CAMPO-A, CAMPO-B e CAMPO-C são somados ao conteúdo do CAMPO-D. O resultado fica no campo-D.

c)



Neste exemplo, os conteúdos dos CAMPO-A e CAMPO-B são somados ao conteúdo do CAMPO-C. O resultado fica no CAMPO-C. Simultaneamente, os conteúdos dos CAMPO-A e CAMPO-B são somados ao conteúdo do CAMPO-D.

d)



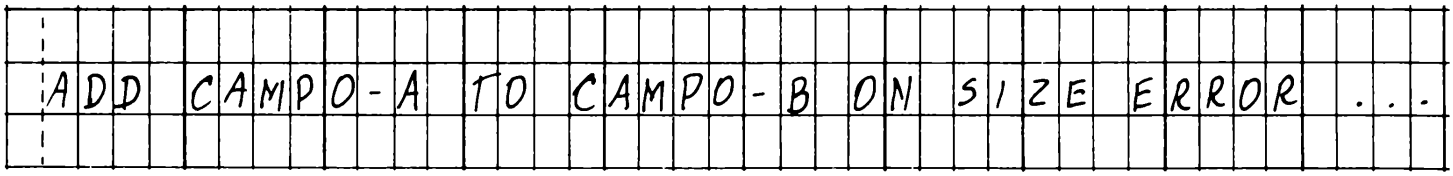
Antes de explicar este exemplo, explicaremos a função do parâmetro ROUNDED.

ROUNDED

Nos comandos aritméticos, o parâmetro ROUNDED ocasiona um arredondamento do resultado. Quando o comprimento da parte fracionária do resultado for maior do que o campo reservado para ela, esta parte fracionária é truncada, a menos que seja especificado ROUNDED. Quando ROUNDED for especificado, o dígito menos significativo do resultado será aumentado de 1 sempre que o dígito mais significativo do excesso seja superior a 4.

No caso do exemplo *d* acima, deseja-se que o resultado a ser colocado no CAMPO-B seja arredondado.

e)



Antes de explicar este exemplo, explicaremos a função do parâmetro ON SIZE ERROR.

ON SIZE ERROR

Nas operações aritméticas, após o alinhamento da vírgula decimal, se o valor do resultado não couber no campo desti-

nado a ele, ocorrerá uma condição de SIZE ERROR (erro de comprimento). Divisão por zero sempre provoca SIZE ERROR. Nestes casos, duas situações podem existir:

1) Não foi codificado o parâmetro SIZE ERROR.
O resultado será imprevisível.
2) Foi codificado o parâmetro SIZE ERROR.
Nestas situações, o campo que deveria receber o resultado fica inalterado e o controle do programa passa para o comando escrito após o parâmetro SIZE ERROR.

FORMATO 2

```
ADD { identificador-1 } { identificador-2 } { identificador-3 } ...
    { literal-1     }   { literal-2     }   { literal-3     }

    GIVING identificador-m [ROUNDED] [ON SIZE ERROR comando]
```

Quando este formato for usado, os valores dos identificadores 1, 2 e 3 são somados e o resultado é levado para o identificador-m. As funções dos parâmetros ROUNDED e SIZE ERROR são semelhantes às do formato 1.

Exemplos:
a)

```
ADD A B C 8 GIVING RESULTADO.
```

Neste exemplo, os campos A, B e C são somados. A este resultado parcial é somado o valor 8. Todo o resultado é levado para o campo RESULTADO.

b)

```
ADD E F GIVING G ROUNDED.
```

Neste exemplo, o campo G conterá o resultado da soma dos campos E e F. O resultado será arredondado.

COMANDO COMPUTE

O comando COMPUTE coloca num campo o valor de um item de dados, de um literal ou de uma expressão aritmética. O formato é:

FORMATO

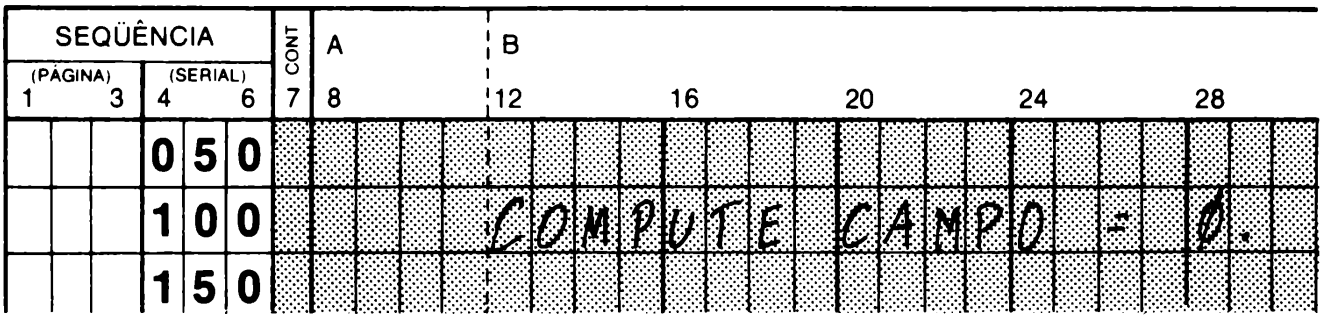
COMPUTE identificador-1 [ROUNDED] = { identificador-2 | literal-1 | expressão aritmética }

[ON *SIZE ERROR* comando]

— Literal-1 tem de ser literal numérico.
— Identificador-1: pode ser um item a ser editado.
— Identificador-2: deve ser um item elementar numérico.
— O comprimento de cada operando deve, como em todos os comandos aritméticos, ter 18 dígitos decimais no máximo.

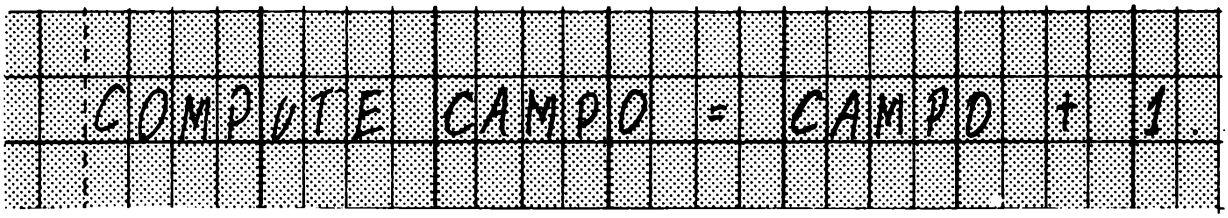
Exemplos:

a)



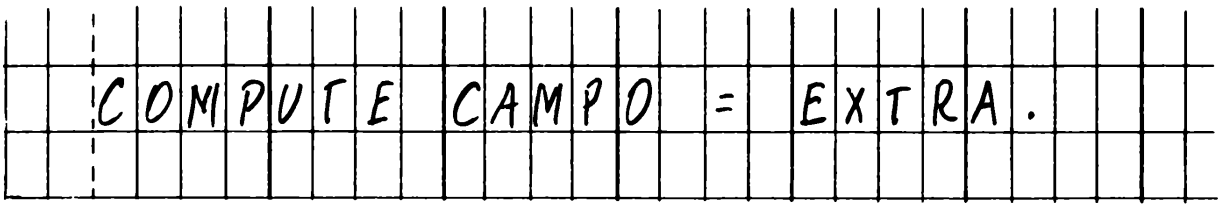
Neste exemplo, o CAMPO será preenchido com o valor ø.

b)



Neste exemplo, o CAMPO terá, como resultado final, o valor anterior mais 1. Ou seja, este comando serviria para somar 1 ao conteúdo de campo.

c)



Neste exemplo, o valor de CAMPO ficará igual ao valor de EXTRA.

COMANDO DIVIDE

O comando DIVIDE é usado para efetuar divisão de valores.

O seu formato é:

FORMATO

```
DIVIDE { identificador-1 } { INTO } { identificador-2 }
       { literal-1       } { BY   } { literal-2       }

       GIVING identificador-3 [ROUNDED]

       [REMAINDER identificador-4]

       [ON SIZE ERROR comando]
```

Observações:

1. O valor do identificador-2 ou literal-2 (dividendo) é dividido pelo identificador-1 (divisor). O quociente ficará:

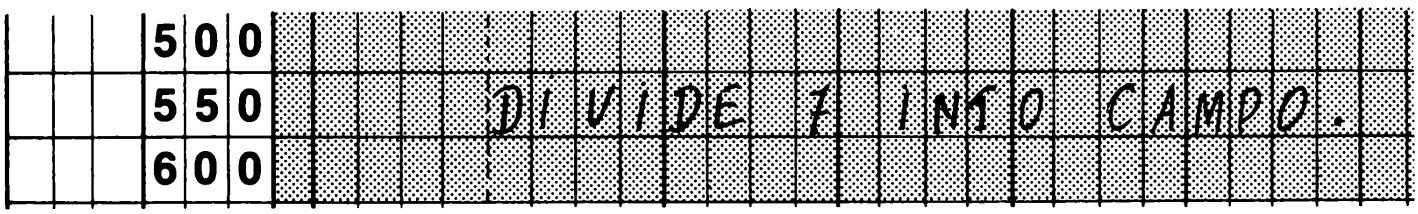
   a) no literal-2, se não for usada a opção GIVING.
   b) no identificador-3, se for usada a opção GIVING.

2. Se usado o parâmetro REMAINDER, o resto ficará no identificador-4. Caso contrário, o resto será ignorado.

3. Se usado o parâmetro ROUNDED, o quociente será arredondado.

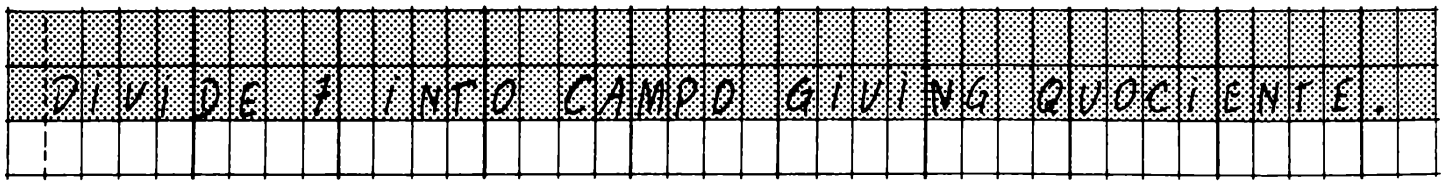
Exemplos:

a)



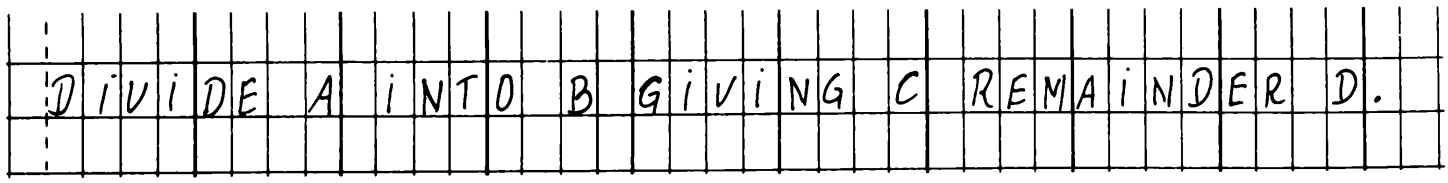
Neste exemplo, o CAMPO será dividido por 7 e o resultado ficará em CAMPO.

b)



Neste exemplo, o CAMPO será dividido por 7 e o resultado ficará em QUOCIENTE.

c)



Neste exemplo, o conteúdo do campo B será dividido pelo conteúdo do campo A, ficando o resultado no campo C. O resto ficará no campo D.

COMANDO MULTIPLY

O comando MULTIPLY é usado para efetuar multiplicações. O seu formato é:

FORMATO

```
MULTIPLY { identificador-1 }        BY { identificador-2 }
         { literal-1       }           { literal-2       }

          GIVING identificador-3

          [ROUNDED] [ON SIZE ERROR comando]
```

O identificador-1 (ou literal-1) é multiplicado pelo identificador-2 (ou literal-2) e o resultado ficará:

1. No lugar do identificador-2, se não for usada a opção GIVING.

2. No identificador-3, se for usada a opção GIVING.

Exemplos:

a)

```
MULTIPLY 10 BY VALOR.
```

Neste exemplo, o VALOR será multiplicado por 10 e o resultado ficará em VALOR.

b)

```
MULTIPLY 32,7 BY VALOR GIVING TOTAIS.
```

Neste exemplo, o VALOR será multiplicado por 32,7 e o resultado será colocado em TOTAIS.

O comando SUBTRACT é usado para efetuar subtrações. Existem dois formatos do comando SUBTRACT:

FORMATO 1

```
                    ⎧identificador-1⎫     ⎡identificador-2⎤
SUBTRACT            ⎩literal-1      ⎭     ⎣literal-2      ⎦ ...

          FROM identificador-m      [ROUNDED]

[identificador-n] [ROUNDED] ...   [ON SIZE ERROR comando]
```

FORMATO 2

```
SUBTRACT ⎧identificador-1⎫     identificador-2⎤
         ⎩literal-1      ⎭     literal-2      ⎦ ...

         ⎧identificador-m⎫
FROM     ⎩literal-m      ⎭     GIVING identificador-n

[ROUNDED]  [ON SIZE ERROR comando]
```

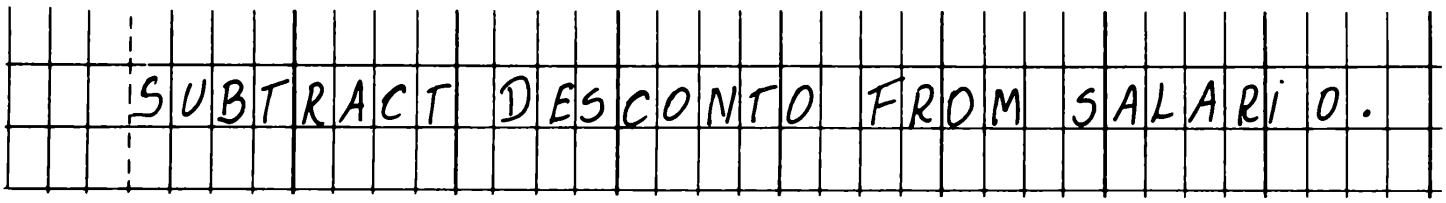
Formato 1:

Todos os literais e identificadores que precederem à palavra *FROM* serão somados juntos e este total será subtraído do identificador-m (e identificador-n etc., se forem escritos). O resultado ficará no identificador-m (e identificador-n etc., se escritos).

Formato 2:

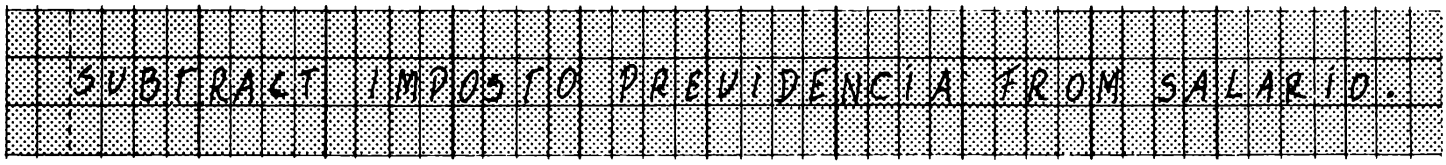
Todos os literais e identificadores que precederem à palavra *FROM* serão somados e este total será subtraído do literal-m ou identificador-m. O resultado final ficará no identificador-n.

Exemplos:
a)



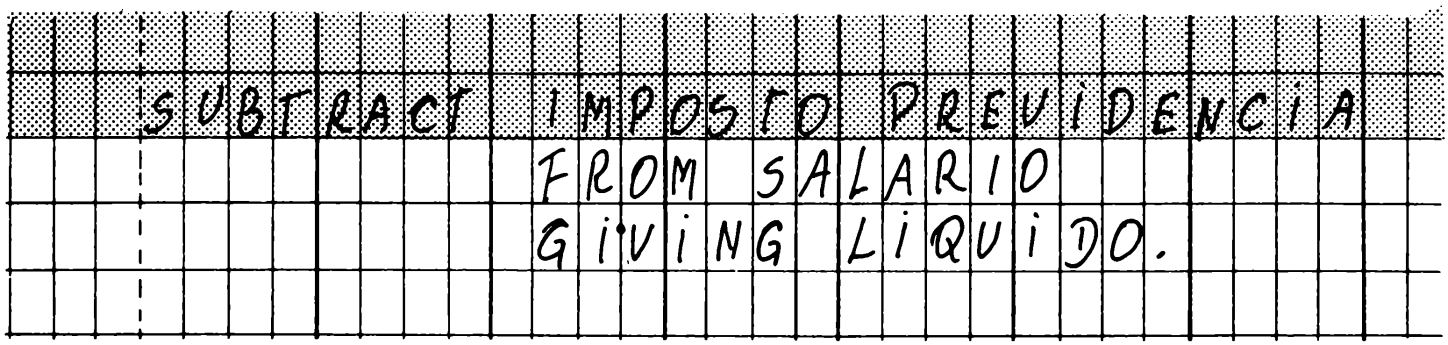
Neste exemplo, o valor de DESCONTO será subtraído de SALÁRIO. O resultado ficará em SALÁRIO.

b)



Neste exemplo, o campo IMPOSTO será somado ao campo PREVIDÊNCIA. O resultado desta soma será subtraído de SALÁRIO. Isto equivale a dizer que IMPOSTO e PREVIDÊNCIA serão subtraídos de SALÁRIO. O resultado ficará em SALÁRIO.

c)

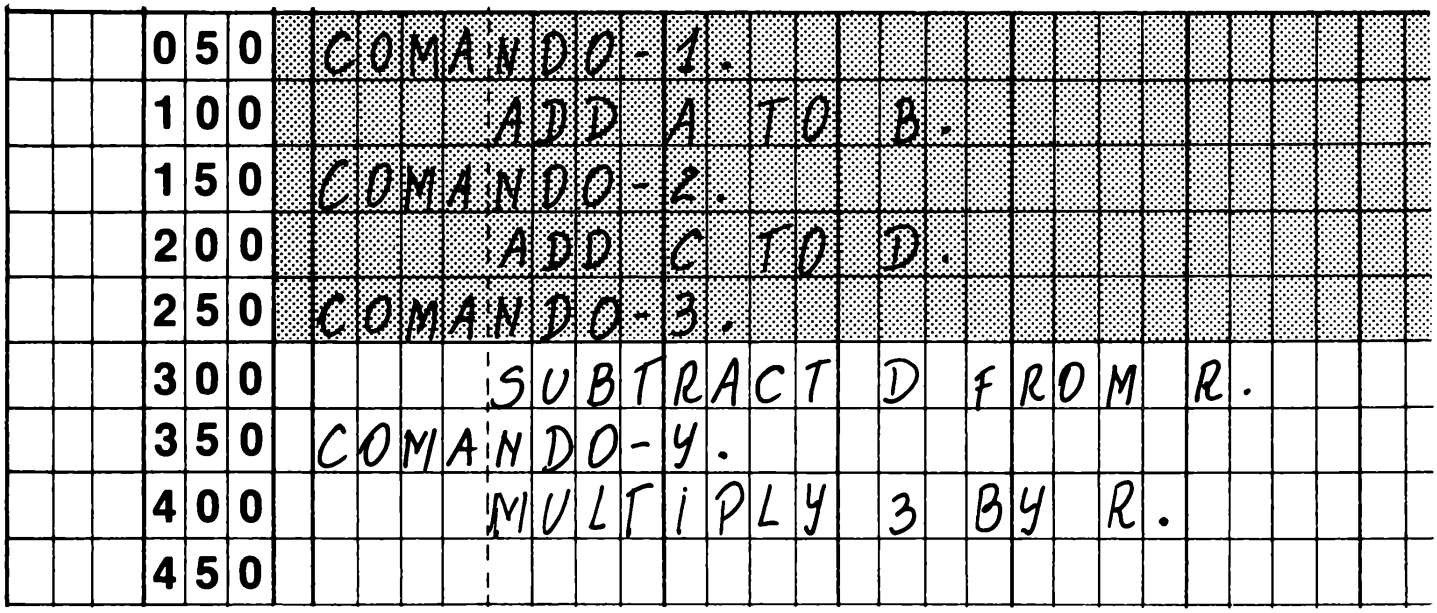


Neste exemplo, a operação será idêntica à do exemplo *b*, com a diferença apenas que o resultado ficará no campo LÍQUIDO.

COMANDO GO TO

Os comandos de um programa, normalmente, são executados em seqüência.

Por exemplo:



Neste exemplo, serão processados sucessivamente, na ordem:

1. O comando-1.
2. O comando-2.
3. O comando-3.
4. O comando-4.

Com o uso do comando GO TO podemos alterar a ordem de execução destes comandos.

Os formatos do comando GO TO são:

FORMATO 1

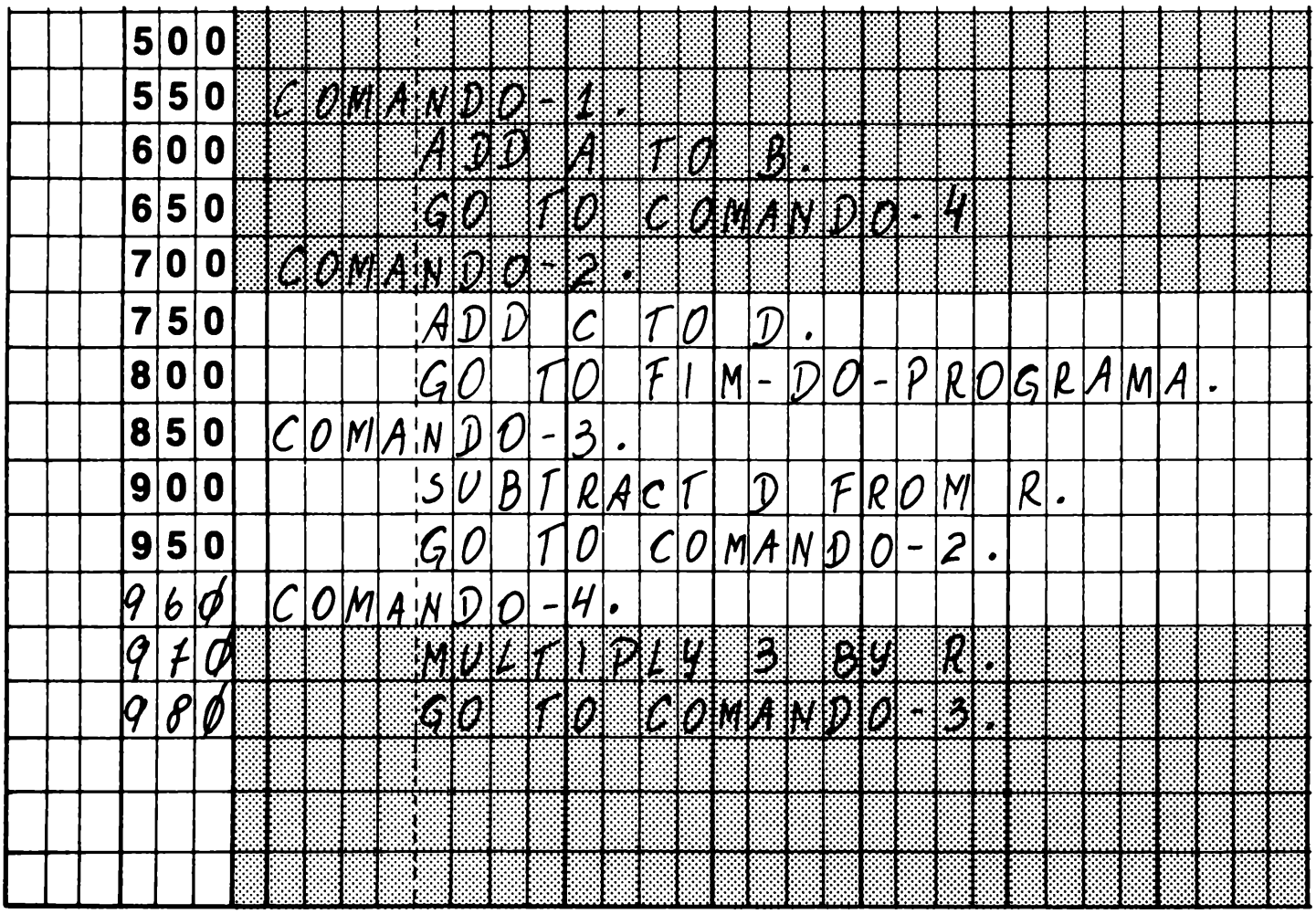
GO TO nome-do-parágrafo-1

FORMATO 2

GO TO nome-do-parágrafo-1 [nome-do-parágrafo-2]...
DEPENDING ON identificador.

Exemplos:

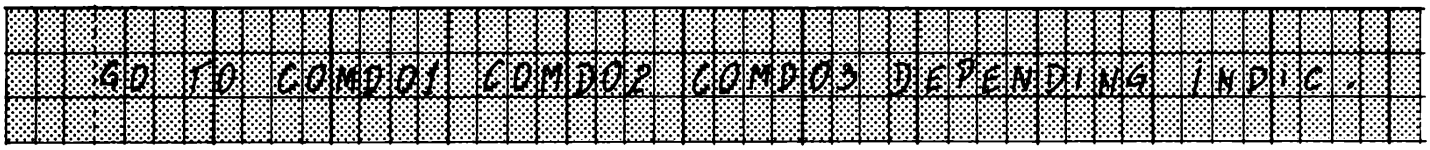
1)



Neste exemplo, após executar ADD A TO B, o comando GO TO COMANDO-4 fará com que o programa, em vez de passar o controle para o parágrafo COMANDO-2, passe o controle para o parágrafo COMANDO-4. Após executar MULTIPLY 3 BY R, o comando GO TO COMANDO-3 fará com que o controle passe para o COMANDO-3. Após executar SUBTRACT D FROM R, o

comando GO TO COMANDO-2 fará com que o programa passe a executar o comando ADD C TO D (COMANDO-2). Após executar ADD C TO D o programa desviará (passará o controle) para o primeiro comando do parágrafo chamado FIM-DO-PROGRAMA.

2)



Neste exemplo, o controle do programa passará para COMDO1, COMDO2 ou COMDO3, dependendo do valor encontrado no campo INDIC. Se o valor de INDIC for 1, o desvio ocorrerá para COMDO1. Se o valor de INDIC for 2, ocorrerá desvio para COMDO2. Se o valor de INDIC for 3, ocorrerá desvio para COMDO3. Se o valor de INDIC for diferente de 1, 2 ou 3, o GO TO será ignorado.

COMANDO ALTER

O comando ALTER é usado para modificar (alterar) um comando GO TO.

O seu formato é:

FORMATO

ALTER nome-do-parágrafo-1 *TO* [*PROCEED TO*] nome-do-parágrafo-2

[nome-do-parágrafo-3] *TO* [*PROCEED TO*] nome-do-parágrafo-4

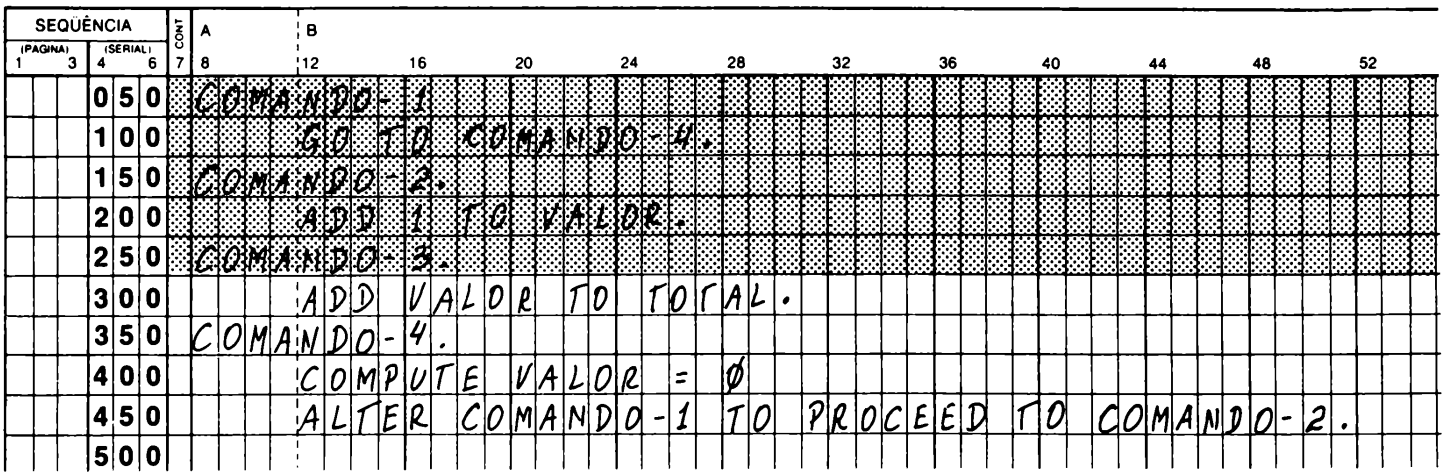
...

Nome-do-parágrafo-1, nome-do-parágrafo-3 etc., devem ser nomes de parágrafos que contenham somente um comando GO TO sem a opção DEPENDING.

Nome-do-parágrafo-2, nome-do-parágrafo-4 etc., devem ser nomes de parágrafos da PROCEDURE DIVISION.

O comando ALTER substitui, no comando GO TO, o nome-do-parágrafo-1 pelo nome-do-parágrafo-2, o nome-do-parágrafo-3 pelo nome-do-parágrafo-4 etc.

Exemplo:



Neste exemplo, a primeira vez que o COMANDO-1 for executado ocorrerá um desvio para o COMANDO-4, isto é, o controle do programa passará para o COMANDO-4, devido ao GO TO COMANDO-4.

Após a execução do ALTER, no parágrafo COMANDO-4, quando novamente o programa for executar o COMANDO-1, o desvio não mais ocorrerá para o COMANDO-4 e sim para o COMANDO-2. Isto porque o comando ALTER substitui o endereço de desvio (COMANDO-4) do parágrafo COMANDO-1 pelo parágrafo COMANDO-2.

COMANDO PERFORM

O comando PERFORM é usado para forçar a execução de comandos fora da seqüência normal do programa. Após a execução dos comandos, o controle passa para o comando seguinte ao comando PERFORM.

Existem 3 formatos do comando PERFORM:

FORMATO 1

```
PERFORM nome-do-parágrafo-1 · [THRU nome-do-parágrafo-2]
```

Quando este formato é usado, o parágrafo ou parágrafos indicados serão executados uma vez e o controle é passado para o comando seguinte ao comando PERFORM.

Exemplos:

a)

```
550
600 INICIO.
650     ADD 1 TO CAMPO.
700     PERFORM ROTINA.
750     COMPUTE CAMPO = Ø.
800     GO TO INICIO.
850 ROTINA.
900     COMPUTE CAMPO = CAMPO + VALOR.
950     COMPUTE TOTAIS = VALOR + CAMPO + 1Ø.
98Ø OUTRO.
990     SUBTRACT 1 FROM VALOR.
995 FIM.
```

Neste exemplo, após a execução do comando ADD 1 TO CAMPO, será processado o parágrafo ROTINA. Após os dois COMPUTE do parágrafo ROTINA, o controle passará para o COMPUTE CAMPO = Ø, logo abaixo do PERFORM. Neste exemplo, o comando SUBTRACT não seria executado.

b)

Se no exemplo anterior o PERFORM fosse escrito: PERFORM ROTINA THRU OUTRO, todo o processo seria semelhante. A diferença seria que o comando SUBTRACT também seria executado.

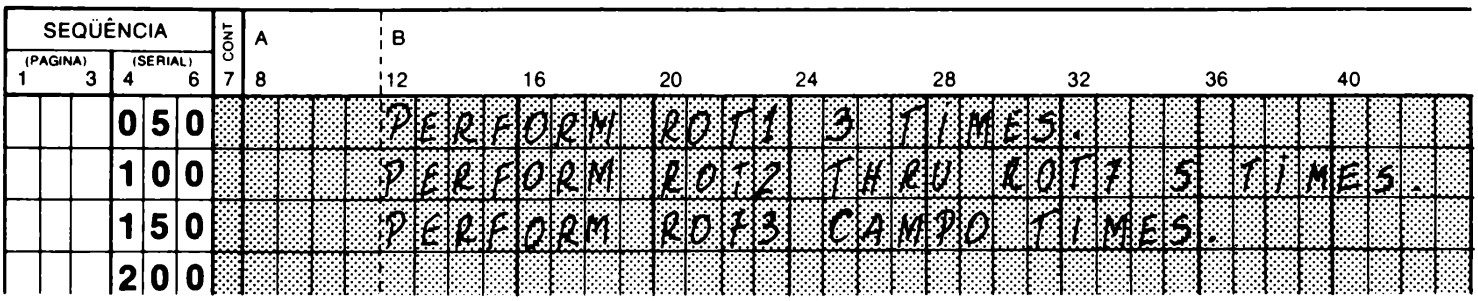
FORMATO 2

PERFORM nome-do-parágrafo-1 [*THRU* nome-do-parágrafo-2]

{identificador-1 / inteiro-1} TIMES

Este formato é usado quando desejamos que a rotina seja executada um determinado número de vezes. Neste caso, definimos o número de vezes pelo inteiro-1 ou pelo conteúdo identificador-1.

Exemplo:



Neste exemplo, o primeiro PERFORM indica que ROT1 deverá ser executada 3 vezes. O segundo PERFORM indica que a ROT2 e todas as rotinas entre ROT2 e ROT7 inclusive deverão ser executadas 5 vezes.

O terceiro PERFORM indica que a rotina chamada RO73 deverá ser executada o número de vezes que determinado pelo conteúdo da área chamada CAMPO. Se o conteúdo de CAMPO for 1, a RO73 será executada uma vez. Se for 2, a RO73 será executada 2 vezes e assim por diante.

FORMATO 3

PERFORM nome-do-parágrafo-1 [*THRU* nome-do-parágrafo-2]

UNTIL condição-1

Este formato é usado quando se deseja que uma rotina seja executada várias vezes até que uma determinada condição seja alcançada.

Exemplo:

```
PERFORM ROT1 UNTIL CAMPO = 36.
PERFORM ROT2 UNTIL CAMPOA < CAMPOB.
```

Neste exemplo, ROT1 será executada até que o conteúdo do CAMPO seja 36.

ROT2 será executada até que CAMPOA seja menor que CAMPOB.

COMANDO STOP

O comando STOP é usado para parar o processamento temporariamente ou definitivamente.

O formato é:

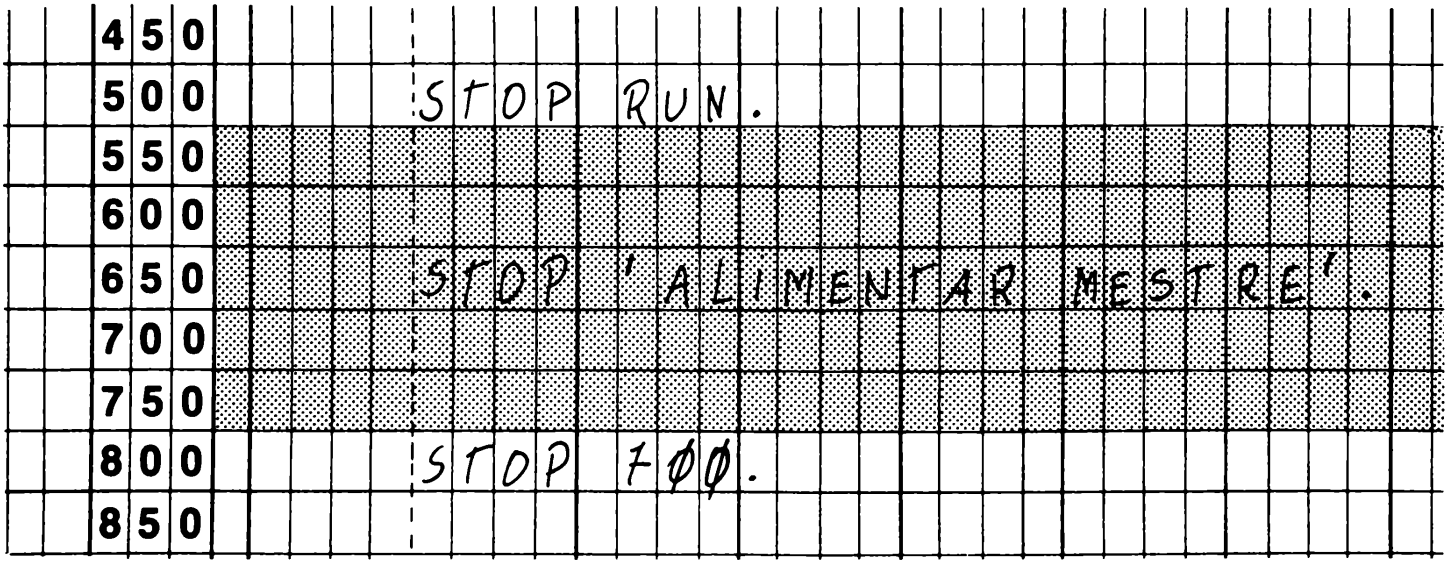
FORMATO

STOP { *RUN* / literal }

RUN: É usado para encerrar o programa e passar o controle para o sistema. Todos os arquivos devem ser encerrados com um comando CLOSE antes do comando STOP RUN.

Literal: Quando é usada a opção literal, o literal é comunicado ao operador. O processamento do programa só é reiniciado com a intervenção do operador. O literal pode ser numérico ou não numérico e pode ser uma constante figurativa, exceto ALL.

Exemplo:



O primeiro STOP encerra o programa.

O segundo STOP interrompe o processamento e mostra a mensagem "Alimentar mestre" ao operador. O programa só é reiniciado com a intervenção do operador.

O terceiro STOP interrompe o processamento e mostra o número 700 ao operador. O operador poderá, com base num manual de operação, tomar uma providência relacionada com o número 700. O processamento só é reiniciado com a intervenção do operador.

COMANDO EXIT

O comando EXIT indica o ponto final de uma rotina. É usado, também, para servir de saída para várias rotinas.

Exemplo:

```
050
100 ROTINA-1.
150     ADD 1 TO ACUM.
200     GO TO FIM.
250 ROTINA-2.
300     ADD 2 TO ACUM.
350     GO TO FIM.
400 ROTINA-3.
450     ADD 3 TO ACUM.
500     GO TO FIM.
550 ROTINA-4.
600     ADD 4 TO ACUM.
650 FIM.
700     EXIT.
750
800     PERFORM ROTINA-1 THRU FIM.
850     PERFORM ROTINA-2 THRU FIM.
900     PERFORM ROTINA-3 THRU FIM.
950     PERFORM ROTINA-4 THRU FIM.
```

Neste exemplo, o parágrafo FIM serviu como ponto final para ROTINA-1, ROTINA-2, ROTINA-3 e ROTINA-4.

O EXIT permite o controle passar para o comando seguinte ao comando PERFORM.

COMANDO MOVE

O comando MOVE é usado para transferir dados de uma área para outra ou outras áreas.

Existem dois formatos do comando MOVE:

FORMATO 1

```
MOVE { identificador-1 }   TO identificador-2   [identificador-3] ...
     { literal         }
```

FORMATO 2

```
MOVE { CORRESPONDING } identificador-1 TO identificador-2
     { CORR          }
```

Formato 1: Os dados representados pelo identificador-1 (ou literal) são movidos para identificador-2, para identificador-3 (se especificado) etc.

Exemplo:

```
050
100 MOVE CAMPO-A TO CAMPO-B.
150
```

Neste exemplo, o conteúdo do CAMPO-A é movido para o CAMPO-B, isto é, o conteúdo do CAMPO-B é substituído pelo conteúdo do CAMPO-A. Os dois campos ficarão com conteúdos idênticos.

O formato 2 é usado para transferir dados entre itens de dados que tenham mesmo nome, simplesmente especificando a que grupo eles pertencem. Pelo menos um dos itens de dados de um par de itens com mesmo nome precisa ser item elementar.

Exemplo:

```
200
250  01  PAI.
300      03 NOME PIC X(30).
350      03 IDADE PIC 999.
400      03 SEXO  PIC X.
450
500  01  FILHO.
550      03 IDADE PIC 999.
600      03 NOME  PIC X(30).
650      03 SEXO  PIC X.
700
750
800      MOVE CORRESPONDING PAI TO FILHO.
850
```

Neste exemplo, os campos NOME, IDADE e SEXO do campo PAI serão movidos para os campos correspondentes do campo FILHO.

REGRAS APLICÁVEIS A QUALQUER COMANDO *MOVE*

1. Quando o campo receptor é um item alfabético, alfanumérico ou alfanumérico de edição:
a) Se o comprimento do campo receptor for maior do que o campo emissor, as posições excedentes serão preenchidas com espaços.

b) Se o comprimento do campo emissor for maior do que o comprimento do campo receptor, os caracteres em excesso serão truncados.

c) Se o item emissor tiver sinal, será usado o valor absoluto.

2. Quando o item receptor for numérico ou numérico de edição:

a) Haverá alinhamento do ponto (ou vírgula) decimal e as posições excedentes serão preenchidas com zeros.

b) Se o item receptor não tiver sinal, será usado o valor absoluto do item emissor.

c) Se houver excesso de dígitos no campo emissor em relação ao campo receptor, eles serão truncados.

d) Se existir um caráter não numérico no campo emissor, os resultados serão imprevisíveis.

3. Qualquer conversão de dados de uma forma de representação (display, computational, computational-3) para outra pode ser feita com o comando MOVE. O campo receptor determina a forma da representação que ficarão os dados, independentemente da forma em que estejam no campo emissor.

Existem algumas restrições no movimento de itens. Elas são mostradas no quadro a seguir.

CAMPO EMISSOR	CAMPO RECEPTOR							
	GR	AL	AN	DE	BI	NE	ANE	DC
Grupo (GR)	S	S	S1	S1	S1	S1	S1	S1
Alfabético (AL)	S	S	S	N	N	N	S	N
Alfanumérico (AN)	S	S	S	S4	S4	S	S	S4
Decimal (DE)	S1	N	S2	S	S	S	S2	S
Binário (BI)	S1	N	S2	S	S	S	S2	S
Numérico editado (NE)	S	N	S	N	N	N	S	N
Alfanumérico editado (ANE)	S	S	S	N	N	N	S	N
Zeros (numérico ou alfanum.)	S	N	S	S3	S3	S3	S	S3
SPACES (AL)	S	S	S	N	N	N	S	N
High-value, low-value, quote	S	N	S	N	N	N	S	N
All literal	S	S	S	S5	S5	S5	S	S5
Literal numérico	S1	N	S2	S	S	S	S2	N
Literal não numérico	S	S	S	S5	S5	S5	S	S5
Decimal compactado (DC)	S1	N	S2	S	S	S	S3	S

Convenções:

S — Sim, permitido.
N — Não permitido.
S1 — Permitido, mas não será feita conversão de um formato para outro.
S2 — Permitido, somente se a vírgula decimal estiver à direita do último dígito significativo.
S3 — Movimento numérico.
S4 — O campo alfanumérico é tratado como um campo inteiro sem sinal.
S5 — O literal deve ser constituído apenas de caracteres numéricos e será tratado como um campo inteiro decimal.

COMANDO OPEN

Todo arquivo antes de ser lido ou ser gravado precisa ser "aberto". O comando OPEN supre esta necessidade. Portanto, antes de comandar a primeira leitura ou gravação de um arquivo

é preciso escrever o comando OPEN. O comando OPEN confere *labels* em arquivos de entrada e grava label nos arquivos de saída. Ele serve, também, para indicar quais serão os arquivos de entrada (INPUT) e quais serão os de saída (OUTPUT).

O formato do comando OPEN é:

FORMATO

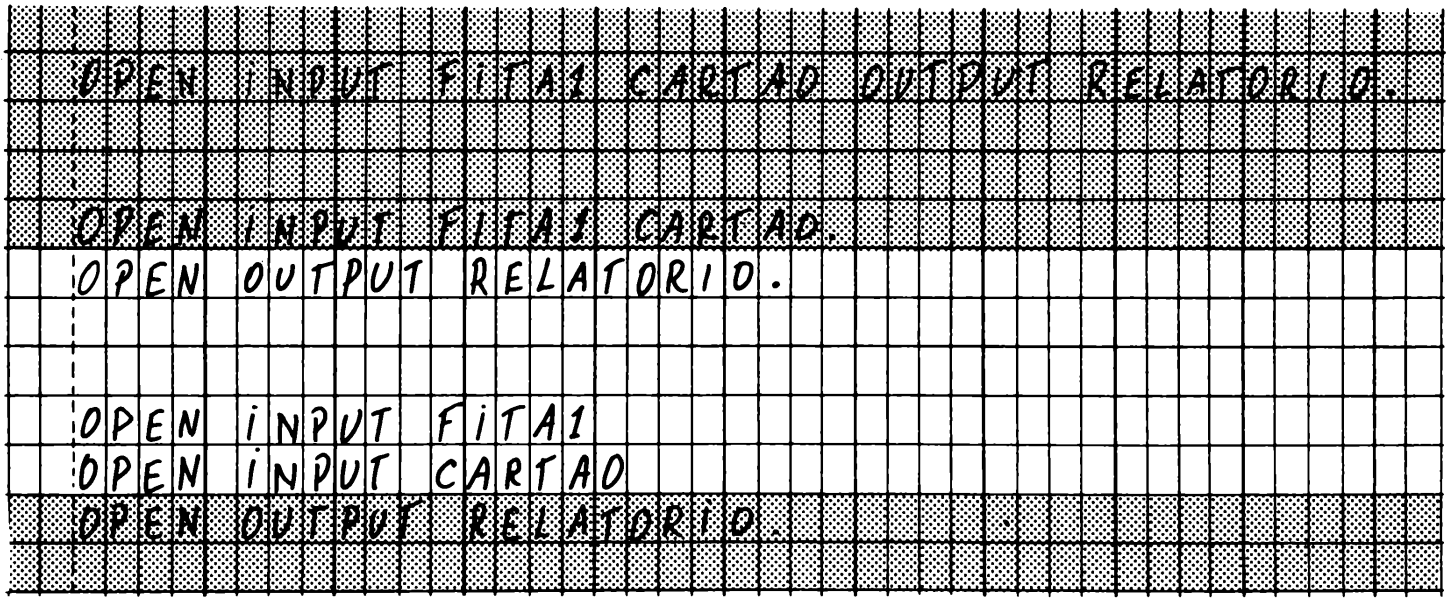
```
OPEN [INPUT  {nome-do-arquivo   [REVERSED          ]]
                                [WITH NO REWIND    ]} ...

     [OUTPUT  {nome-do-arquivo  [WITH NO REWIND]}...

     [I-O  {nome-do-arquivo}      . . .]
```

As opções REVERSED e NO REWIND só podem ser usadas com arquivos seqüenciais gravados num único volume. Alguns compiladores, no entanto, aceitam arquivos com mais de um volume.

REVERSED: Indica que a leitura deve ser feita do fim para o princípio do arquivo.

WITH NO REWIND: Alguns compiladores a consideram apenas como comentário. Outros compiladores usam-na para evitar que a fita, antes de ser processada, seja reenrolada. Neste caso, a leitura ou gravação terá início a partir do ponto em que a fita parou após o último processamento.

Exemplos:



Os três exemplos são equivalentes. O programador pode escolher qualquer dos três modelos apresentados.

COMANDO READ

Após o comando OPEN, um arquivo fica disponível para ser lido ou gravado. O comando READ tem por função ler o arquivo.

O formato do comando READ é:

FORMATO

READ nome-do-arquivo RECORD [*INTO* identificador]

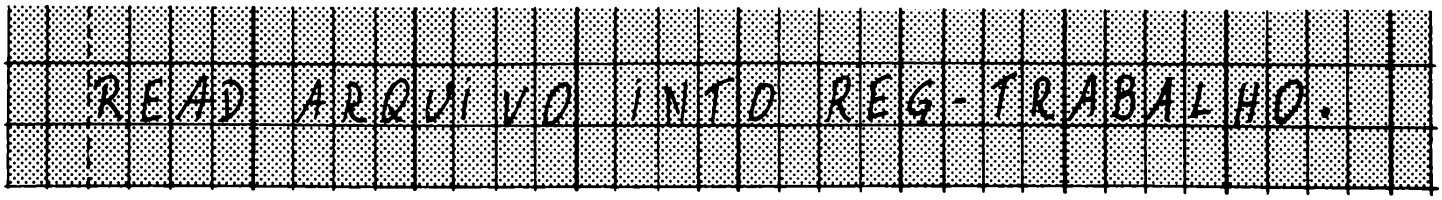
[*AT END* / INVALID KEY] comando

Quando um comando READ é executado, o próximo registro lógico do arquivo fica disponível na área definida na entrada RECORD DESCRIPTION.

O registro permanece disponível até ser executado outro comando de entrada para o mesmo registro.

INTO: a opção INTO faz com que o registro, além de ficar disponível na área descrita na RECORD DESCRIPTION, fique disponível, também, na área indicada pelo identificador.

Exemplo:



Neste exemplo, o registro ficará disponível em duas áreas:

a) Na área definida para o registro na RECORD DESCRIPTION (descrição do registro).

b) Na área chamada REG-TRABALHO, que deverá ter sido definida na WORKING-STORAGE SECTION.

AT END: esta opção tem de ser especificada para todos os arquivos seqüenciais. Ela indica para onde deve ser passado o controle do programa, caso ocorra, num comando READ, o fim do arquivo.

Exemplo:

Neste exemplo, quando chegar o final do ARQUIVO, o controle do programa passará para a rotina chamada FIM.

INVALID KEY: esta opção deve ser usada para arquivos de acesso direto. O controle é passado para o comando que segue a opção INVALID KEY, sempre que o conteúdo da ACTUAL KEY ou da NOMINAL KEY for inválido.

COMANDO WRITE

O comando WRITE libera um registro lógico para o arquivo de saída.

O comando WRITE tem dois formatos:

FORMATO 1

```
WRITE    nome-do-registro    [FROM identificador-1]

[{ BEFORE } ADVANCING        {identificador-2 LINES}]
 { AFTER  }                  {inteiro LINES        }
                             {nome-mnemônico       }
```

FORMATO 2

```
WRITE  nome-do-registro  [FROM identificador-1]

       INVALID  KEY  comando
```

Nome-do-registro: é o nome do registro lógico descrito na FILE SECTION da DATA DIVISION.

FROM: quando for usada a opção FROM, antes do registro ser gravado, haverá um movimento do conteúdo do identificador-1 para o registro.

ADVANCING: a opção ADVANCING permite controlar o espacejamento vertical do formulário numa impressora.

Quando for usada a opção ADVANCING para um registro do arquivo, todos os demais registros precisam também conter esta opção.

Quando a opção ADVANCING é usada, o compilador grava um caráter, para controle de espacejamento, como primeiro caráter do registro.

Identificador-2: deve conter um número inteiro menor que 100. Haverá um espacejamento de linhas correspondente ao número contido no identificador-2.

Inteiro: indica o número de linhas a ser espacejado na impressora. Deve ser menor que 100.

Nome-mnemônico: quando este parâmetro é usado na opção ADVANCING, ele deve ser definido como em nome-de-função no parágrafo SPECIAL-NAMES da ENVIRONMENT DIVISION. Ele é usado para saltar para canais 1 a 12 e para suprimir espacejamento.

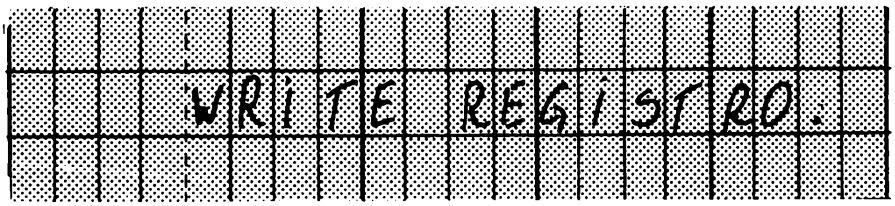
BEFORE ADVANCING: esta opção indica que o registro deve ser impresso ANTES da impressora espacejar.

AFTER ADVANCING: esta opção indica que o registro deve ser impresso APÓS a impressora espacejar.

INVALID KEY: esta opção deve ser especificada para arquivos gravados em discos magnéticos. O controle é passado para o comando que segue a opção INVALID KEY, sempre que houver um erro detectado na RECORD, na NOMINAL ou na ACTUAL KEY.

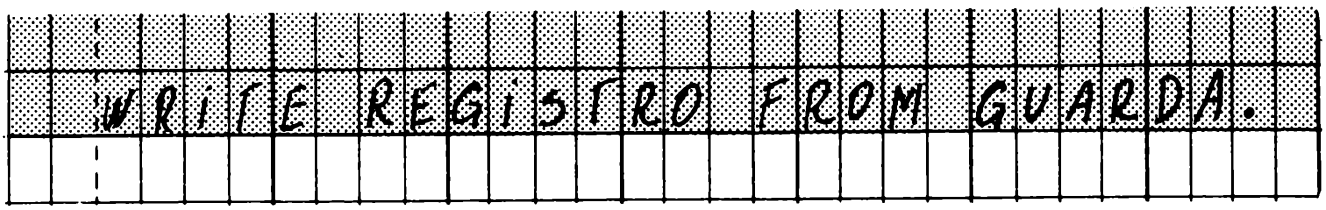
Exemplos:

a)



Neste exemplo, um registro será liberado para saída.

b)



Neste exemplo, antes do REGISTRO ser liberado, o conteúdo do campo GUARDA será movido para o campo REGISTRO.

c)

```
WRITE REGISTRO AFTER ADVANCING 3 LINES.
```

Neste exemplo, a impressora espacejará 3 linhas e após imprimirá os dados do REGISTRO.

d)

```
WRITE REGISTRO AFTER ADVANCING CØ1.
```

Neste exemplo, a impressora saltará para o canal 1 e em seguida imprimirá os dados do REGISTRO.

COMANDO ACCEPT

A função do comando ACCEPT é obter dados da SYSIN ou do CONSOLE.

O seu formato é:

FORMATO

ACCEPT identificador [FROM { SYSIN / CONSOLE }]

Identificador: será o campo para onde serão movidos os dados obtidos através da SYSIN ou do CONSOLE.

Se os dados forem obtidos através da SYSIN, deverá ter um comprimento de 80 caracteres. Se forem obtidos através do CONSOLE poderá ter até 114 caracteres.

Se não for especificado CONSOLE ou SYSIN, será assumida a opção SYSIN.

Exemplo:

a)

```
ACCEPT MESTRE.
```

Neste exemplo, será lido um cartão da SYSIN e gravado no campo MESTRE.

b)

```
ACCEPT ORIGEM FROM CONSOLE.
```

Neste exemplo, os dados que o operador digitar no CONSOLE serão gravados no campo ORIGEM.

COMANDO DISPLAY

A função do comando DISPLAY é gravar dados numa unidade de saída.

O seu formato é:

FORMATO

```
DISPLAY  { literal-1       }  [ literal-2       ] ...
         { identificador-1 }  [ identificador-2 ]

[UPON { CONSOLE  }
      { SYSPUNCH }  ]
      { SYSOUT   }
```

Quando a opção UPON for omitida, será assumida a opção SYSOUT.

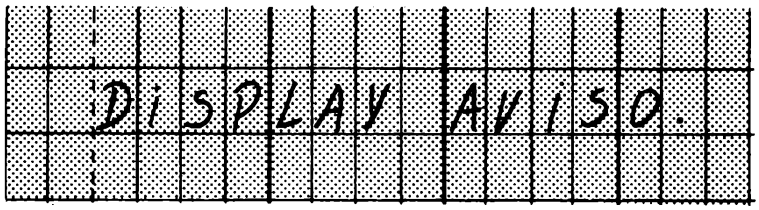
O tamanho máximo permitido para o registro é:

Para SYSOUT: 120 caracteres.
Para SYSPUNCH: 72 caracteres.
Para CONSOLE: 100 caracteres.

Identificador-1, identificador-2 etc.: se não estiverem no formato DISPLAY, serão convertidos automaticamente para este formato.

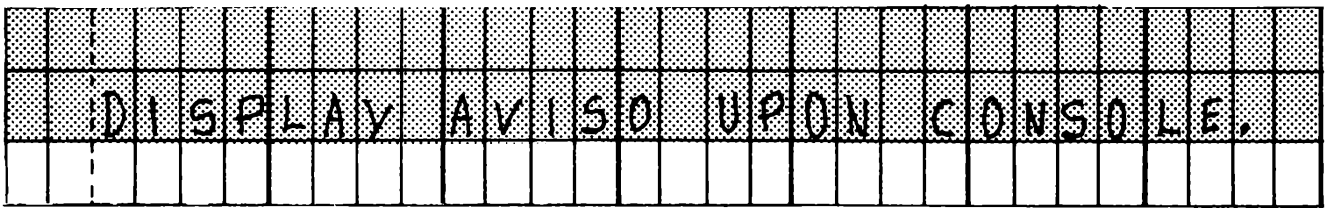
Exemplos:

a)



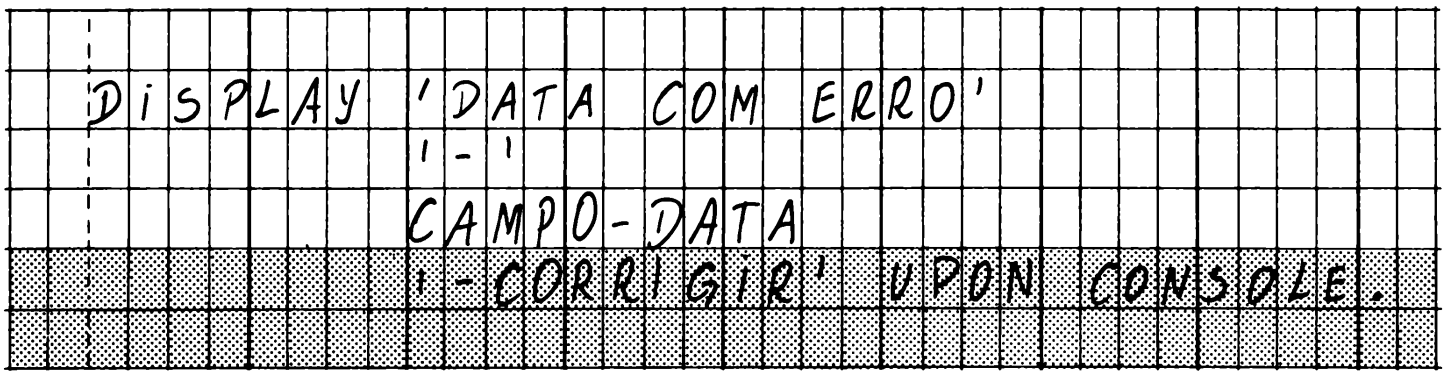
Neste exemplo, o conteúdo do campo AVISO será impresso na SYSOUT.

b)



Neste exemplo, o conteúdo do campo AVISO será mostrado no CONSOLE.

c)



Neste exemplo, serão impressos sucessivamente no console: o literal 'DATA COM ERRO'
o literal '-'
o conteúdo do campo CAMPO-DATA
o literal '-CORRIGIR'
Se o CAMPO-DATA contiver uma data e for de 6 caracteres, a linha mostrada no console será assim:
DATA COM ERRO-XXXXXX-CORRIGIR.

COMANDO CLOSE

O comando CLOSE encerra o processamento do arquivo de entrada/saída.

O seu formato é:

FORMATO

```
CLOSE    nome-do-arquivo-1   [WITH { NO REWIND }]
                               { LOCK      }

         [nome-do-arquivo-2  [WITH { NO REWIND }] ]...
                                   { LOCK      }
```

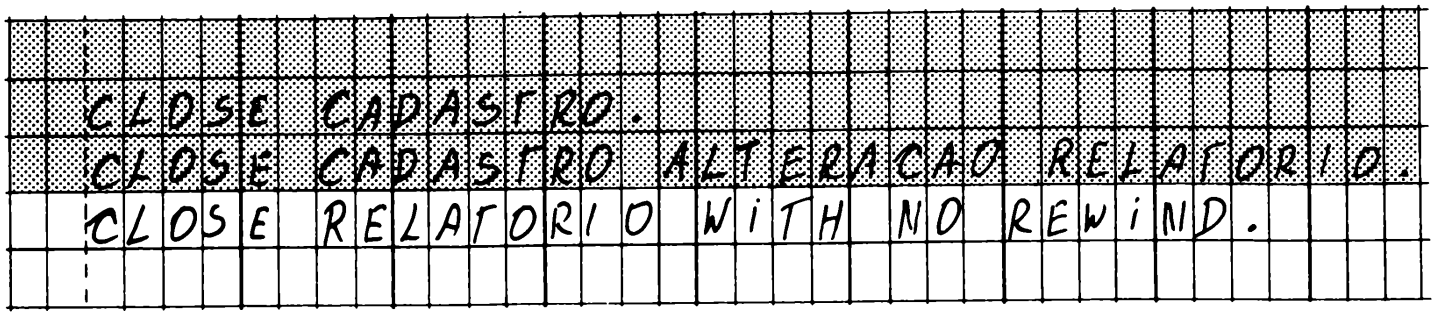
Nome-do-arquivo-1, nome-do-arquivo-2 etc.: são os arquivos sobre os quais devem operar o comando CLOSE. Eles devem ser definidos na FILE DESCRIPTION da DATA DIVISION.

A opção NO REWIND só é aplicável a arquivos em fita magnética.

NO REWIND: usado quando desejamos que a fita não seja reenrolada ao final do processamento.

WITH LOCK: usado quando desejamos que a fita seja desconectada (descarregada) ao final do processamento, isto é, quando desejamos que ao final do processamento a fita seja liberada para o operador retirá-la da máquina.

Exemplo:



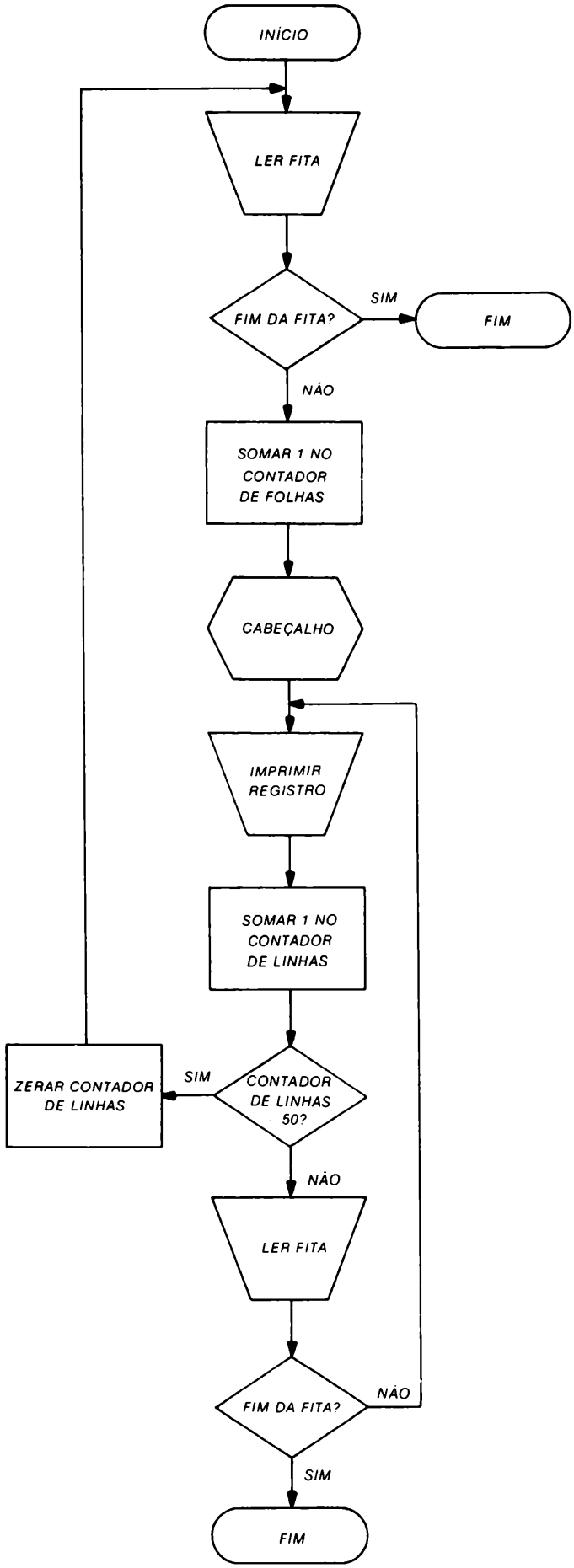
EXERCÍCIO FINAL

Escrever, em COBOL, um programa para executar a tarefa indicada abaixo:

Ler um arquivo gravado em fita magnética. O tamanho do bloco é de 1 320 caracteres. A fita foi gravada sem label.

Para cada registro lido da fita magnética, imprimir uma linha no relatório. Cada folha do relatório deverá ter 50 linhas impressas. Imprimir, no início de cada folha, um cabeçalho onde deverá constar o número da folha.

O fluxograma deste exercício poderia ser como o mostrado a seguir.


INÍCIO
LER FITA
FIM DA FITA?
SIM
FIM
NÃO
SOMAR 1 NO CONTADOR DE FOLHAS
CABEÇALHO
IMPRIMIR REGISTRO
SOMAR 1 NO CONTADOR DE LINHAS
CONTADOR DE LINHAS = 50?
SIM
ZERAR CONTADOR DE LINHAS
NÃO
LER FITA
FIM DA FITA?
NÃO
SIM
FIM

SISTEMA:		INSTRUÇÕES DE PERFURAÇÃO:	FOLHA 1 DE
PROGRAMA: EXERCÍCIO FINAL		Ø — ZERO	IDENTIFICAÇÃO: EXERCFIM (73–80)
PROGRAMADOR: FULANO	DATA: 1979		

SEQUÊNCIA PÁGINA (1–3)	SEQUÊNCIA SERIAL (4–6)	CONT (7)	A (8–11) / B (12–72)
ØØ1	050		IDENTIFICATION DIVISION.
	100		PROGRAM-ID. EXERCICIO FINAL.
	150		AUTHOR. FULANO.
	200		INSTALLATION. EMPRESA TAL.
	250		DATE-WRITTEN. 1979.
	300		DATE-COMPILED.
	350		ENVIRONMENT DIVISION.
	400		CONFIGURATION SECTION.
	450		SOURCE-COMPUTER. IBM-370.
	500		OBJECT-COMPUTER. IBM-370.
	550		SPECIAL-NAMES. DECIMAL-POINT IS COMMA.
	600	*	
	650		INPUT-OUTPUT SECTION.
	700		SELECT FITA ASSIGN TO UT-2420-S-ENTRA.
	750		SELECT IMPRESSORA
	800		ASSIGN TO UR-1403-S-SAIDA.
	850	*	
	900		DATA DIVISION.
	950		FILE SECTION.

SISTEMA		INSTRUÇÕES DE PERFURAÇÃO	FOLHA 2 DE
PROGRAMA EXERCÍCIO FINAL		Ø — ZERO	IDENTIFICAÇÃO EXERCFIM (73–80)
PROGRAMADOR FULANO	DATA 1979		

```
SEQUÊNCIA  CONT  A       B
PAGINA SERIAL  8       12        16        20        24        28        32        36        40        44        48        52        56        60        64        68        72
ØØ2    050     FD      FITA
               100     BLOCK CONTAINS 10 RECORDS
               150     RECORD CONTAINS 132 CHARACTERS
               200     RECORDING MODE IS F
               250     LABEL RECORDS OMITTED
               300             DATA RECORD IS REGISTRO.
               350     Ø1      REGISTRO.
               400         Ø3  REG PICTURE X(132).
               450 *
               500     FD      IMPRESSORA
               550             RECORD CONTAINS 132 CHARACTERS
               600             LABEL RECORDS OMITTED
               650             DATA RECORD IS LINHA.
               700     Ø1      LINHA PICTURE X(132).
               750 *
               800     WORKING-STORAGE SECTION.
               850     77      CONTA-FOLHAS PICTURE 9(5) VALUE ZERO.
               900     77      CONTA-LINHAS PICTURE 99   VALUE ZERO.
               950
               960     Ø1      LINHA-CABECALHO.
               970         Ø3  FILLER PICTURE X(9)       VALUE 'RELATORIO'.
               980         Ø3  FILLER PICTURE X(114)     VALUE SPACES.
               990         Ø3  FILLER PICTURE XXX        VALUE 'FL.'.
               995         Ø3  NUMERO-FOLHA PICTURE ZZ.ZZ9.
```

SISTEMA		INSTRUÇÕES DE PERFURAÇÃO	FOLHA 3 DE
PROGRAMA EXERCÍCIO FINAL		Ø-ZERO	IDENTIFICAÇÃO EXERCFIM (73–80)
PROGRAMADOR FULANO	DATA 1979		

SEQUÊNCIA (PAGINA) 1–3	(SERIAL) 4–6	CONT 7	A (8)	B (12–72)
ØØ3	050		PROCEDURE DIVISION.	
	100			OPEN INPUT FITA
	150			OUTPUT IMPRESSORA.
	200		AA.	READ FITA AT END GO TO FIM.
	250			ADD 1 TO CONTA-FOLHAS.
	300			PERFORM CABECALHO THRU SAI-CABECALHO.
	350		IMPRIME.	
	400			MOVE REGISTRO TO LINHA.
	450			WRITE LINHA AFTER ADVANCING 1 LINES.
	500			COMPUTE CONTA-LINHAS = CONTA-LINHAS + 1.
	550			IF CONTA-LINHAS IS EQUAL TO 50
	600			MOVE Ø TO CONTA-LINHAS
	650			GO TO AA.
	700			READ FITA AT END GO TO FIM.
	750			GO TO IMPRIME.
	800		FIM.	
	850			CLOSE FITA IMPRESSORA.
	900			STOP RUN.
	950			

SISTEMA		INSTRUÇÕES DE PERFURAÇÃO	FOLHA 4 DE
PROGRAMA EXERCÍCIO FINAL		Ø-ZERO	IDENTIFICAÇÃO EXERCFIM (73–80)
PROGRAMADOR FULANO	DATA 1979		

```
SEQUÊNCIA (PÁGINA 1-3, SERIAL 4-6)  CONT 7   A 8-11   B 12-72
ØØH 050   CABECALHO.
    100       MOVE CONTA-FOLHAS TO NUMERO-FOLHA.
    150       WRITE LINHA FROM LINHA-CABECALHO AFTER
    200                 ADVANCING 16 LINES.
    250   *                                              *
    300   *   OBSERVACAO - AS 16 LINHAS INDICADAS ACIMA  *
    350   *   SERAO ESPACEJADAS PARA COM-                *
    400   *   PLETAR A FOLHA ANTERIOR.                   *
    450   *                                              *
    500       SAI-CABECALHO.
    550       EXIT
    600
    650
    700
    750
    800
    850
    900
    950
```

O computador eletrônico é capaz de efetuar operações aritméticas, comparações de valores e tarefas necessárias à execução de um serviço, sem a intervenção do homem. Para isso, no entanto, o computador precisa ser "instruído" previamente.

Foram criadas várias formas de instruir o computador por meio de codigos e regras mais simples do que os códigos de márquina e as regras que regem estes códigos de máquina. O conjunto destes códigos e regras simplificadas é conhecido como linguagem de programação. O COBOL é a mais empregada destas linguagens e dirigida, principalmente, para processamento de problemas comerciais.

O livro contém exercícios onde o leitor poderá comparar os seus fluxogramas com os apresentados nas páginas seguintes aos mesmos.

EDIÇÕES DE OURO