capítulo 4

# máquinas de registradores – Norma

Intuitivamente, um algoritmo tem descrição finita e não ambígua e consiste em passos discretos, executáveis mecanicamente em um tempo finito. O conceito de programa introduzido satisfaz essa noção intuitiva. Entretanto, é necessário definir a máquina a ser considerada. Se for possível representar qualquer algoritmo como um programa em tal máquina, então esta é uma máquina universal. Com esse objetivo são introduzidas as máquinas de registradores. Em particular, é estudada a máquina Norma que possui três tipos de instruções/teste (ad, sub e zero) sobre registradores naturais, mostrando que uma máquina simples e rudimentar pode ser tão poderosa quanto qualquer outra.


Até o momento, o termo *algoritmo* foi intuitivamente usado como solução de um problema, ou seja, como uma forma de descrever se determinada propriedade é verificada ou não para uma dada classe de entrada. Portanto, o estudo da solucionabilidade de um problema é a investigação da existência de um algoritmo capaz de resolvê-lo. Entretanto, se a noção de algoritmo é intuitiva (não formal), qualquer afirmação sobre a não solucionabilidade de determinado problema é questionável.

Em relação à noção intuitiva de algoritmo, o seguinte pode ser afirmado:

- sua descrição deve ser finita e não ambígua;
- deve consistir de passos discretos, executáveis mecanicamente em um tempo finito.

Limitações de tempo ou de espaço podem, eventualmente, determinar se um algoritmo pode ou não ser utilizado na prática. Entretanto, estas não são restrições teóricas, pois a inexistência de limitações *não* implica recursos ou descrições infinitas. Assim, recursos de tempo e de espaço são "tanto quanto necessários". O correto entendimento dessa observação é de fundamental importância para todo o estudo que segue.

O estudo é restrito aos algoritmos naturais, ou seja, definidos sobre o conjunto dos números naturais. Na realidade, qualquer *conjunto contável* (existe uma bijeção com **N**) poderia ser equivalentemente considerado. Mesmo para os tipos cujos valores possuem descrição finita, uma grande variedade de tipos de dados deve ser considerada como, por exemplo, inteiros, cadeia de caracteres, valores-verdade, etc., bem como tipos baseados em conjuntos estruturados como vetores. Adiante, é exemplificado como alguns destes tipos de dados podem ser codificados como naturais.

O conceito de programa, como introduzido anteriormente, satisfaz à noção intuitiva de algoritmo. Entretanto, é necessário definir a máquina a ser considerada, a qual deve ser suficientemente:

- *simples*, para permitir estudos de propriedades sem a necessidade de considerar características não relevantes, bem como permitir estabelecer conclusões gerais sobre a classe de funções computáveis;
- *poderosa*, capaz de simular qualquer característica de máquinas reais ou teóricas, de tal forma que os resultados provados fossem válidos para modelos aparentemente com mais recursos e para que qualquer função computável possa ser nela representada.

Se for possível representar qualquer algoritmo como um programa em tal máquina, então esta é denominada máquina universal. As evidências de que uma máquina é, de fato, uma máquina universal podem ser classificadas como:

**a** *Evidência interna*. Consiste na demonstração de que qualquer extensão das capacidades da máquina proposta computa, no máximo, a mesma classe de funções, ou seja, não aumenta o seu poder computacional;

**b** *Evidência externa*. Consiste no exame de outros modelos que definem a noção de algoritmo, juntamente com a prova de que são, no máximo, computacionalmente equivalentes.

Ao longo deste e dos próximos capítulos, a noção de máquina universal é construída. No que segue, é estudada a máquina universal denominada máquina Norma, proposta por Richard Bird (1976), a qual, resumidamente, possui um conjunto de registradores naturais e somente três tipos de instruções sobre os registradores: adição e subtração do valor um e teste se o valor armazenado é zero. Trata-se de um exemplo de *máquina de registradores*. Como ilustração, diversas características de máquinas reais são simuladas usando a máquina Norma, reforçando as evidências internas de que, de fato, trata-se de uma máquina universal.

## 4.1 ⋯→ codificação de conjuntos estruturados

Nesta seção, é considerado, de forma breve e por meio de exemplos, o problema da *codificação de conjuntos estruturados*, onde elementos de tipos de dados estruturados são representados como números naturais. Para um dado conjunto estruturado X, a ideia básica é definir uma função injetora:

$$c: X \to \mathbf{N}$$

ou seja, uma função tal que, para todo $x, y \in X$, tem-se que:

$$\text{se } c(x) = c(y), \text{ então } x = y$$

Neste caso, o número natural $c(x)$ é a codificação do elemento estruturado x.

**exemplo 4.1** – Número de Gödel

Suponha que é desejado codificar, de forma unívoca, elementos de $\mathbf{N}^n$ como números naturais, ou seja, deseja-se uma função injetora:

$$c: \mathbf{N}^n \to \mathbf{N}$$

Uma codificação simples é a seguinte:

- lembre-se que, pelo *teorema fundamental da aritmética*, cada número natural é univocamente decomposto em seus fatores primos;
- suponha os n primeiros números primos denotados por $p_1 = 2$, $p_2 = 3$, $p_3 = 5$ e assim sucessivamente. Então, a codificação $c: \mathbf{N}^n \rightarrow \mathbf{N}$ definida a seguir é unívoca (suponha que $(x_1, x_2, ..., x_n) \in \mathbf{N}^n$ e que o símbolo $*$ denota a operação de multiplicação nos naturais):

$$c(x_1, x_2, ..., x_n) = p_1^{x1} * p_2^{x2} * ... * p_n^{xn}$$

Deve-se reparar que esta codificação, conhecida como *número de Gödel*, não constitui uma função bijetora, ou seja, nem todo número natural corresponde a uma n-upla (qual seria um exemplo?). Entretanto, todo número natural decomponível nos n primeiros números primos corresponde a uma n-upla. ❑

**exemplo 4.2** – Codificação de programas monolíticos

De forma análoga, um programa monolítico (fluxograma) pode ser codificado como um número natural. Suponha um programa monolítico $P = (I, r)$ com m instruções rotuladas, onde $\{ F_1, F_2, ..., F_f \}$ e $\{ T_1, T_2, ..., T_t \}$ são os correspondentes conjuntos de identificadores de operações e de testes, respectivamente. Seja, $P' = (I', 1)$ como P, exceto pelos rótulos, os quais são renomeados como números naturais, onde 1 é o rótulo inicial e 0 o único rótulo final (se existir). Assim, uma instrução rotulada pode ser de uma das duas seguintes formas:

**a** *Operação*.

$$r_1: \text{faça } F_k \text{ vá\_para } r_2$$

**b** *Teste*.

$$r_1: \text{se } T_k \text{ então vá\_para } r_2 \text{ senão vá\_para } r_3$$

Cada instrução rotulada pode ser denotada por uma quádrupla ordenada como segue:

**a** *Operação*.

$$(0, k, r_2, r_2)$$

**b** *Teste*.

$$(1, k, r_2, r_3)$$

onde a primeira componente identifica o tipo da instrução; a segunda, o k-ésimo teste ou operação; e as duas últimas são os rótulos sucessores (no caso de operação, são iguais).

Estas quádruplas são codificadas usando os quatro primeiros números primos, seguindo um raciocínio análogo ao do exemplo 4.1 – Número de Gödel. Desta forma, a codificação de P' é realizada em duas etapas:

- cada quádrupla (instrução rotulada) é codificada como um número natural. Assim, o programa monolítico P' com m instruções rotuladas pode ser visto como uma m-upla;
- por sua vez, a m-upla correspondente ao programa monolítico P' é codificada como um número natural conforme o exemplo 4.1 – Número de Gödel. ❑

**exemplo 4.3** – Codificação de programas monolíticos

Considere o programa monolítico Q da figura 4.1. Então, a correspondente codificação c(i1, i2, i3, i4, i5, i6, i7, i8) é o numero natural q dado por:

$$q = 2^{i1} \cdot 3^{i2} \cdot 5^{i3} \cdot 7^{i4} \cdot 11^{i5} \cdot 13^{i6} \cdot 17^{i7} \cdot 19^{i8}$$

sendo que (tomando F como 1, G como 2 e o teste T como 1 na determinação de k, ou seja, o expoente do número primo 3):

| | |
|---|---|
| $i1 = 2^0 \cdot 3^1 \cdot 5^2 \cdot 7^2$ | obtido da quádrupla: (0, 1, 2, 2) |
| $i2 = 2^1 \cdot 3^1 \cdot 5^3 \cdot 7^5$ | (1, 1, 3, 5) |
| $i3 = 2^0 \cdot 3^2 \cdot 5^4 \cdot 7^4$ | (0, 2, 4, 4) |
| $i4 = 2^1 \cdot 3^1 \cdot 5^1 \cdot 7^0$ | (1, 1, 1, 0) |
| $i5 = 2^0 \cdot 3^1 \cdot 5^6 \cdot 7^6$ | (0, 1, 6, 6) |
| $i6 = 2^1 \cdot 3^1 \cdot 5^7 \cdot 7^2$ | (1, 1, 7, 2) |
| $i7 = 2^0 \cdot 3^2 \cdot 5^8 \cdot 7^8$ | (0, 2, 8, 8) |
| $i8 = 2^1 \cdot 3^1 \cdot 5^6 \cdot 7^0$ | (1, 1, 6, 0) |

❑

**Programa monolítico Q**

```
1:    faça F vá_para 2
2:    se T então vá_para 3 senão vá_para 5
3:    faça G vá_para 4
4:    se T então vá_para 1 senão vá_para 0
5:    faça F vá_para 6
6:    se T então vá_para 7 senão vá_para 2
7:    faça G vá_para 8
8:    se T então vá_para 6 senão vá_para 0
```

**figura 4.1** Programa monolítico Q.

Analogamente ao exemplo 4.1 – Número de Gödel, a codificação de programas monolíticos apresentada não é uma função bijetora (por quê?).

**exemplo 4.4** – Decodificação de programas monolíticos

A decodificação de um número natural que denota um programa monolítico em seu correspondente programa é ilustrada. Suponha o número natural p decomposto em fatores primos como segue:

$$p = (2^{150}) \cdot (3^{105})$$

Portanto, o programa possui duas instruções rotuladas correspondentes aos expoentes dos números primos 2 e 3, respectivamente, 150 e 105. Decompondo-os em seus fatores primos, tem-se que:

$$150 = 2^1 \cdot 3^1 \cdot 5^2 \cdot 7^0 \quad \text{e} \quad 105 = 2^0 \cdot 3^1 \cdot 5^1 \cdot 7^1$$

o que corresponde às quádruplas:

$$(1, 1, 2, 0) \quad \text{e} \quad (0, 1, 1, 1)$$

Logo, as instruções rotuladas decodificadas são como segue:

```
1:    se T1 então vá_para 2 senão vá_para 0
2:    faça F1 vá_para 1
```

Este exemplo ajuda a inferir que nem todo número natural corresponde a um programa codificado por esta função de codificação.

Sugere-se como exercício a generalização do caso ilustrado para qualquer número natural correspondente à codificação de um programa monolítico. ❑

## 4.2 ⇢ definição da máquina Norma

A *máquina Norma* (*Number TheQretic Register MAchine*), proposta por Richard Bird (1976), é uma máquina de registradores. Máquinas de registradores são modelos propostos mais recentemente (em comparação com a máquina de Turing e com outros formalismos universalmente conhecidos) e são definidas de forma a lembrar a arquitetura básica dos computadores atuais. A máquina Norma é especialmente interessante, pois distingue as noções de programa e máquina (tal fato ficará evidente quando do estudo dos demais modelos). Assim, por razões didáticas, a máquina Norma é o primeiro formalismo de máquina universal introduzido nesta publicação.

A máquina Norma possui um conjunto infinito ("tantos quantos forem necessários") de registradores naturais como memória e três instruções sobre cada registrador:

- adição do valor um;
- subtração do valor um;
- teste se o valor armazenado é zero.

No que se refere aos registradores naturais, entende-se que os registradores da máquina Norma podem assumir "qualquer valor natural tão grande quanto necessário".

No texto que segue, $\mathbf{N}^{\infty}$ denota o conjunto de todas as uplas com infinitos (mas contáveis) componentes sobre o conjunto dos números naturais. Por exemplo, as seguintes uplas são elementos de $\mathbf{N}^{\infty}$:

$$(0, 1, 2, 3, \ldots) \quad \text{e} \quad (5, 5, 5, 5, \ldots)$$

Para evitar subscritos, as componentes das uplas são denotadas por letras maiúsculas, como X, Y, A, B,... as quais denotam os registradores na máquina Norma.

A definição formal de $\mathbf{N}^{\infty}$ é omitida, mas pode ser resumidamente entendida como segue:

- suponha a upla $(a_0, a_1, a_2, \ldots)$ em $\mathbf{N}^{\infty}$;
- pode-se afirmar que, para cada $n \in \mathbf{N}$, $a_n$ é a n-ésima componente da upla $(a_0, a_1, a_2, \ldots)$;

- tal fato pode ser formalmente denotado como uma função $f: \mathbf{N} \to \mathbf{N}$ tal que, para qualquer $n \in \mathbf{N}$, $f(n) = a_n$;
- portanto, um elemento de $\mathbf{N}^{\infty}$ pode ser visto como uma função nos naturais, ou seja:

$$\mathbf{N}^{\infty} = \{ f: \mathbf{N} \to \mathbf{N} \mid f \text{ é função} \}$$

No caso específico das uplas (0, 1, 2, 3, ...) e (5, 5, 5, 5, ...), as correspondentes funções são as seguintes, respectivamente:

- função identidade de **N**, ou seja, $id_{\mathbf{N}}: \mathbf{N} \to \mathbf{N}$ tal que, para qualquer $n \in \mathbf{N}$, tem-se que $id_{\mathbf{N}}(n) = n$;
- função constante 5, ou seja, $const_5: \mathbf{N} \to \mathbf{N}$ tal que, para qualquer $n \in \mathbf{N}$, tem-se que $const_5(n) = 5$.

**definição 4.1** - Máquina Norma

A *máquina Norma* é uma 7-upla (suponha que $K \in \{ X, Y, A, B, \ldots \}$):

$$\text{Norma} = (\mathbf{N}^{\infty}, \mathbf{N}, \mathbf{N}, \text{ent}, \text{sai}, \{ ad_K, sub_K \}, \{ zero_K \})$$

onde:

**a** Cada elemento do conjunto de valores de memória $\mathbf{N}^{\infty}$ denota uma configuração de seus infinitos *registradores*, os quais são denotados por:

$$X, Y, A, B, \ldots$$

**b** A *função de entrada*:

$$\text{ent}: \mathbf{N} \to \mathbf{N}^{\infty}$$

é tal que carrega no registrador denotado por X o valor de entrada, inicializando todos os demais registradores com zero;

**c** A *função de saída*:

$$\text{sai}: \mathbf{N}^{\infty} \to \mathbf{N}$$

é tal que retorna o valor corrente do registrador denotado por Y;

**d** O *conjunto de interpretações de operações* é uma família de operações indexada pelos registradores onde, para cada registrador K, tem-se que:

$$ad_K: \mathbf{N}^{\infty} \to \mathbf{N}^{\infty}$$

adiciona um à componente de memória correspondente ao registrador K, deixando as demais com seus valores inalterados, e:

$$sub_K: \mathbf{N}^{\infty} \to \mathbf{N}^{\infty}$$

subtrai um da componente de memória correspondente ao registrador K, se o seu valor for maior que zero (caso contrário, mantém o valor zero), deixando as demais com seus valores inalterados;

**e** O *conjunto de interpretações de testes* é uma família de testes indexada pelos registradores onde, para cada registrador K, tem-se que:

$$zero_K: \mathbf{N}^{\infty} \to \{ \text{verdadeiro, falso} \}$$

resulta em **verdadeiro**, se a componente de memória correspondente ao registrador K for zero e, caso contrário, resulta em **falso**. ❑

Por simplicidade, para um registrador K, as correspondentes operações ou teste $ad_K$, $sub_K$, $zero_K$ são denotadas, respectivamente, como segue:

K := K + 1
K : = K - 1
K = 0

## 4.3 ···→ máquina Norma como máquina universal

A máquina Norma é uma máquina extremamente simples, de tal forma que parece difícil acreditar que o seu poder computacional é, no mínimo, o de qualquer computador moderno. Inclusive, é suficiente considerar somente os programas monolíticos. Como ilustração, diversas características de máquinas reais são simuladas usando a máquina Norma, reforçando as evidências de que, de fato, trata-se de uma máquina universal. As seguintes simulações por Norma são exemplificadas:

**a** *Operações e testes*. Definição de operações e de testes mais complexos como adição, subtração, multiplicação e divisão de dois valores e tratamento de valores diversos como testes sobre números primos;

**b** *Valores numéricos*. Armazenamento e tratamento de valores numéricos de diversos tipos como inteiros (negativos e não negativos) e racionais;

**c** *Dados estruturados*. Acesso, armazenamento e tratamento de dados estruturados como arranjos (vetores uni e multidimensionais), pilhas, etc.;

**d** *Endereçamento indireto e recursão*. Desvio para uma instrução determinada pelo conteúdo de um registrador e simulação de mecanismos de recursão;

**e** *Cadeia de caracteres*. Definição e manipulação de cadeias de caracteres.

### 4.3.1 operações e testes

As seguintes operações e testes não definidos na máquina Norma são exemplificados a seguir (não necessariamente nesta ordem):

- atribuição de valor a um registrador;
- adição de dois registradores;
- atribuição do valor de um registrador a outro registrador;
- multiplicação de dois registradores;
- atribuição de um número primo a um registrador;
- teste se o valor de um registrador é menor que o valor de outro registrador;
- teste se o valor de um registrador é múltiplo do valor de outro registrador (ou teste de divisão inteira com resto zero);

- teste se o valor de um registrador é um número primo.

**exemplo 4.5** – Atribuição do valor zero a um registrador

A atribuição do valor zero para um registrador A, denotada por:

$$A := 0$$

pode ser obtida com o seguinte programa monolítico:

```
1:    se A = 0 então vá_para 3 senão vá_para 2
2:    faça A := A - 1 vá_para 1
```

ou com seu equivalente programa iterativo:

```
até     A = 0
faça    (A := A - 1)
```

❑

Dessa forma, pode-se tratar a operação A := 0 como uma *macro*, ou seja, um trecho de programa que é substituído pela sua definição sempre que referenciado.

Usando a macro A := 0, é fácil construir macros para definir operações de atribuição de um valor qualquer.

**exemplo 4.6** – Atribuição de um valor natural a um registrador

A seguinte macro de atribuição de um valor natural n a um registrador A, denotada por:

$$A := n$$

pode ser definida pelo seguinte programa monolítico, exemplificado para n = 3:

```
1:    faça A := 0 vá_para 2
2:    faça A := A+1 vá_para 3
3:    faça A := A+1 vá_para 4
4:    faça A := A+1 vá_para 5
```

ou pelo seu programa fortemente equivalente iterativo:

```
A := 0;
A := A+1;
A := A+1;
A := A+1
```

❑

Apesar da máquina Norma só processar programas monolíticos, para facilitar a apresentação dos exemplos, os programas ilustrativos são apresentados na forma de programas iterativos. Lembre-se de que, a partir de um programa iterativo, sempre é possível obter o seu correspondente programa monolítico fortemente equivalente .

**exemplo 4.7** – Adição de dois registradores

Uma macro correspondente à operação de adição do valor do registrador B ao de A, denotada por:

$$A := A + B$$

pode ser obtida com o seguinte programa iterativo:

```
até     B = 0
faça    (A := A + 1;  B := B - 1)
```

❑

Note-se que, no exemplo acima, ao somar o valor de B ao de A, o registrador B é zerado. Para preservar o valor original de B, é necessário utilizar um registrador de trabalho.

**exemplo 4.8** – Adição de dois registradores, preservando o conteúdo

Uma macro correspondente à operação de adição do valor do registrador B ao de A, preservando o valor em B, necessita usar um registrador auxiliar C, como no seguinte programa iterativo:

```
C := 0;
até     B = 0
faça    (A := A + 1;  C := C + 1;  B := B - 1);
até     C = 0
faça    (B := B + 1;  C := C - 1)
```

Entretanto, como este programa não preserva o conteúdo original do registrador de trabalho C, faz-se necessário explicitar o uso deste registrador. Portanto, a seguinte notação é adotada para a macro de adição de dois registradores, preservando o conteúdo:

$$A := A + B \text{ usando } C$$ ❑

É importante destacar que, ao executar na máquina Norma um programa P de outra máquina que contenha uma macro da forma A := A + B, é necessário escolher um registrador de trabalho C que não seja referenciado em P. Desta forma, quando da execução da macro A := A + B usando C, a alteração do valor armazenado em C não terá efeito em qualquer outra parte do programa P.

**exemplo 4.9** - Atribuição do conteúdo de um registrador

Usando a macro A := A + B usando C, pode-se facilmente construir a seguinte macro de atribuição entre registradores:

$$A := B \text{ usando } C$$

a qual pode ser definida pelo seguinte programa iterativo:

```
A := 0;
A := A + B usando C
```

❑

A definição de uma macro de multiplicação requer dois registradores de trabalho.

**exemplo 4.10** – Multiplicação de dois registradores

Uma macro correspondente à operação de multiplicação do valor do registrador B ao de A, usando dois registradores de trabalho C e D, denotada por:

**A := A • B usando C, D**

pode ser definida pelo seguinte programa iterativo:

```
C := 0;
até     A = 0
faça    (C := C + 1; A := A - 1);
até     C = 0
faça    (A := A + B usando D; C := C - 1)
```

❑

**exemplo 4.11** – Teste se o valor de um registrador é menor que outro registrador

Uma macro de teste que verifica se o valor de um registrador A é menor que o valor de um registrador B, denotada por:

**A < B usando C, D, E**

```
C:= A usando E;
D:= B usando E;
até C = 0
faça (se D = 0
      então falso
      senão C:= C-1; D:= D-1);
     (se D = 0
      então falso
      senão verdadeiro)
```

❑

**exemplo 4.12** – Teste se o valor da divisão inteira é zero

Uma macro de teste que verifica se o resto da divisão inteira do valor de um registrador A pelo valor de um registrador B é zero, denotada por:

**teste_mod(A, B) usando C, D, E, C', D', E'**

```
C:= A usando E;
(se B = 0
então falso
senão (se A = 0
       então verdadeiro
       senão até C < B usando C', D', E'
             faça (C := C -D usando E);
      (se C = 0
       então verdadeiro
       senão falso)))
```

Por simplicidade, no texto que segue, na referência a uma macro definida anteriormente, são omitidos os registradores usados. Por exemplo:

**teste_mod(A, B) usando C, D, E, C', D', E'** é abreviada por **teste_mod(A, B)**

**exemplo 4.13** – Teste se o valor de um registrador é um número primo

Uma macro de teste que verifica se o valor de um registrador A é um número primo, usando um registrador de trabalho C, denotada por:

**teste_primo(A)**

pode ser obtido pelo seguinte programa iterativo, o qual retorna o valor **verdadeiro**, se o valor de A for primo, e, caso contrário, o valor **falso**:

```
(se     A := 0;
então falso
senão C := A;
      C := C - 1;
      (se     C = 0
      então verdadeiro
      senão até    teste_mod(A, C)
            faça   (C := C – 1)
            C := C - 1;
            (se     C = 0
            então verdadeiro
            senão falso) ) )
```

onde **teste_mod(A, C)** é o do exemplo 4.12 - Teste se o valor da divisão inteira é zero. ❑

**exemplo 4.14** – Atribuição do n-ésimo número primo a um registrador

A atribuição do n-ésimo número primo a um registrador A, usando um registrador de trabalho D, denotada por (suponha que o conteúdo de B é n):

**A := primo(B)**

pode ser obtida pelo seguinte programa iterativo, o qual usa a macro **teste_primo** construída acima (suponha que 1 é o 0-ésimo número primo):

```
A := 1;
D := B;
até    D = 0
faça   (D := D - 1;
       A := A + 1;
       até    teste_primo(A)
       faça   (A := A + 1) )
```

❑

É sugerido como exercício desenvolver os programas iterativos para as seguintes operações e testes:

- fatorial;
- potenciação;
- teste "maior";
- teste "maior ou igual";
- teste "menor ou igual".

### 4.3.2 valores numéricos

Os seguintes tipos de dados numéricos não definidos na máquina Norma são exemplificados:

- inteiros (negativos e não negativos);
- racionais.

**exemplo 4.15** – Inteiros

Um valor inteiro m pode ser representado como um par ordenado:

$$(s, |m|)$$

onde:

- $|m|$ denota a *magnitude* dada pelo valor absoluto de m;
- s denota o sinal de m, como segue:

$$\text{se } m < 0, \text{ então } s = 1; \text{ senão } s = 0$$

Duas formas simples de representar tal par ordenado na máquina Norma podem ser:

- usando a codificação de n-uplas naturais como introduzido em codificação de conjuntos estruturados;
- usando dois registradores: o primeiro referente ao sinal, e o segundo, à magnitude (valor absoluto).

No segundo caso (dois registradores), a simulação da operação inteira:

$$A := A + 1$$

é exemplificada na seguinte tabela da figura 4.2. Observando a tabela, percebe-se que primeiro é necessário verificar se $A_1$ (sinal) é zero para então analisar $A_2$ (magnitude). Assim, a operação pode ser realizada pelo seguinte programa iterativo (suponha zero com sinal positivo):

```
(se      A1 = 0
então    A2 := A2 + 1
senão    A2 := A2 - 1;
         (se     A2 = 0
         então A1 := A1 - 1
         senão ✔) )
```

Analogamente, a operação inteira de subtração (A := A - 1) é exemplificada na tabela da figura 4.2 e pode ser realizada pelo seguinte programa iterativo:

```
(se     A1 = 0
então (se    A2 = 0
       então A1 := A1 + 1;
             A2 := A2 + 1
       senão A2 := A2 - 1)
senão A2 := A2 + 1)
```

Seguindo o mesmo raciocínio, para o teste se é zero (A = 0) basta verificar se $A_2$ (magnitude) é zero, como realizado pelo seguinte programa iterativo:

```
(se     A2 = 0
então verdadeiro
senão falso)
```

❑

| A | $A_1$ (sinal) | $A_2$ (mag.) | A + 1 | $A_1$ (sinal) | $A_2$ (mag.) | A - 1 | $A_1$ (sinal) | $A_2$ (mag.) |
|---|---|---|---|---|---|---|---|---|
| -2 | 1 | 2 | -1 | 1 | 1 | -3 | 1 | 3 |
| -1 | 1 | 1 | 0 | 0 | 0 | -2 | 1 | 2 |
| 0 | 0 | 0 | 1 | 0 | 1 | -1 | 1 | 1 |
| 1 | 0 | 1 | 2 | 0 | 2 | 0 | 0 | 0 |
| 2 | 0 | 2 | 3 | 0 | 3 | 1 | 0 | 1 |

**figura 4.2** Operações inteiras A := A + 1 e A := A - 1.

**exemplo 4.16** - Racionais

Um valor racional r pode ser denotado como um par ordenado:

(a, b)

tal que b > 0 e r = a/b. A representação não é única pois, por exemplo, o valor racional 0.75 pode ser representado pelos pares (3, 4) e (6, 8), entre outros (na realidade, os pares (3, 4) e (6, 8) pertencem à classe de equivalência que denota o número racional 0.75 - por simplicidade, tal questão não será discutida). Neste contexto, as operações de adição, subtração, multiplicação e divisão, bem como o teste de igualdade, podem ser definidos como segue:

(a, b) + (c, d) = (a•d + b•c, b•d)
(a, b) - (c, d) = (a•d - b•c, b•d)
(a, b) × (c, d) = (a•c, b•d)
(a, b) ÷ (c, d) = (a•d, b•c)  com c ≠ 0
(a, b) = (c, d)  se, e somente se,  a•d = b•c

A construção dos correspondentes programas na máquina Norma é sugerida como exercício. ❑

### 4.3.3 dados estruturados

A seguir, é exemplificado como uma estrutura do tipo arranjo unidimensional pode ser definida na máquina Norma. Sugere-se como exercício a generalização para arranjos multidimensionais. Adicionalmente, é esboçada uma solução para pilhas, usando arranjos.

**exemplo 4.17** - Arranjo unidimensional

Uma estrutura do tipo arranjo unidimensional da forma A(1), A(2), ..., pode ser definida por um único registrador A, usando a codificação de n-uplas naturais como introduzido em 4.1 – Codificação de conjuntos estruturados. Exemplificando, se um arranjo unidimensional é tal que:

A(1) = 5
A(2) = 2
A(3) = 0
A(4) = 3
A(5) = 1

então o valor a armazenada em A é o resultado de:

$$a = 2^5 \cdot 3^2 \cdot 5^0 \cdot 7^3 \cdot 11^1 \cdot 13^0 \cdot 17^0 \ldots$$

Note-se que o arranjo não necessita ter tamanho máximo (número de posições indexáveis) predefinido. Lembre-se de que a função de entrada é tal que carrega o valor da entrada no registrador X, zerando todos os demais, incluindo, portanto, o arranjo.

Em uma estrutura do tipo arranjo, é desejável indexar as suas posições de forma direta (número natural) ou indireta (conteúdo de um registrador). Para ambos os casos e para manter a coerência com a definição da máquina Norma, é importante definir as operações de adição e subtração do valor 1, bem como do teste se o valor é zero, como segue, supondo que:

- o arranjo é implementado usando o registrador A;
- $p_n$ denota o n-ésimo número primo;
- o teste **teste_mod(A, C)** que retorna o valor **verdadeiro**, se o conteúdo do registrador C é um divisor do conteúdo do registrador A e **falso**, caso contrário, implementado no exemplo 4.12 - Teste se o valor da divisão inteira é zero;
- a seguinte macro (cuja definição é sugerida como exercício) denota a divisão de dois registradores:

A := div(A, C)

**a** *Indexação direta*. As macros:

$\mathbf{ad}_{A(n)}$
$\mathbf{sub}_{A(n)}$
$\mathbf{zero}_{A(n)}$

onde A(n) denota a n-ésima posição do arranjo A, podem ser definidas pelos seguintes programas iterativos, respectivamente:

**Programa iterativo** adA(n)

```
C := p_n;
A := A • C
```

**Programa iterativo** $sub_{A(n)}$

```
C := p_n;
(se      teste_mod(A, C)
então  A := div(A, C)
senão  ✔)
```

**Programa iterativo** $zero_{A(n)}$

```
C := p_n;
(se      teste_mod(A, C)
então  falso
senão  verdadeiro)
```

**b** *Indexação indireta*. As macros:

$\mathbf{ad}_{A(B)}$
$\mathbf{sub}_{A(B)}$
$\mathbf{zero}_{A(B)}$

onde A(B) denota a b-ésima posição do arranjo A, onde b é o conteúdo do registrador B, podem ser definidas pelos seguintes programas iterativos, respectivamente:

**Programa iterativo** $ad_{A(B)}$

```
C := primo(B)
A := A • C
```

**Programa iterativo** $sub_{A(B)}$

```
C := primo(B)
(se      teste_mod(A, C)
então  A := div(A, C)
senão  ✔)
```

**Programa iterativo** $zero_{A(B)}$

```
C := primo(B)
(se      teste_mod(A, C)
então  falso
senão  verdadeiro)
```

□


**observação 4.2** – Arranjo unidimensional × Norma com 2 registradores

Um resultado interessante (que será usado adiante) é que, usando a estrutura de arranjo unidimensional com indexação direta, como ilustrado no exemplo 4.17 - Arranjo unidimensional, pode-se mostrar que os registradores X e Y são suficientes para realizar qualquer processamento (ou seja, os registradores A, B,... não são necessários). De fato, é suficiente usar X para armazenar um arranjo unidimensional onde cada posição corresponde a um registrador (por exemplo: X, Y, A, B,... correspondem às posições do arranjo indexadas por 0, 1, 2, 3,...). Assim, para um determinado registrador K, as operações e testes de Norma:

$\mathbf{ad}_K$
$\mathbf{sub}_K$
$\mathbf{zero}_K$

podem ser simulados pelas operações e testes indexados introduzidos no exemplo 4.17 - Arranjo unidimensional:

$\mathbf{ad}_{X(k)}$
$\mathbf{sub}_{X(k)}$
$\mathbf{zero}_{X(k)}$

onde X(k) denota a k-ésima posição do arranjo em X. ❑

**exemplo 4.18** – Pilha

Estruturalmente, a principal característica de uma *pilha* é que o último valor gravado é o primeiro a ser lido, como ilustrado na figura 4.3. A *base* de uma pilha é fixa e define o seu início, o *topo* é variável e define a posição do último símbolo gravado. Uma pilha pode facilmente ser simulada usando um arranjo (como definido) e um registrador de índice (indexação indireta do arranjo) que aponta para o topo da pilha. Sugere-se como exercício o detalhamento desta solução, bem como a definição das operações *empilha* (adiciona

o conteúdo de um registrador ao topo da pilha) e *desempilha* (retira o valor do topo e armazena-o em um registrador). ❑

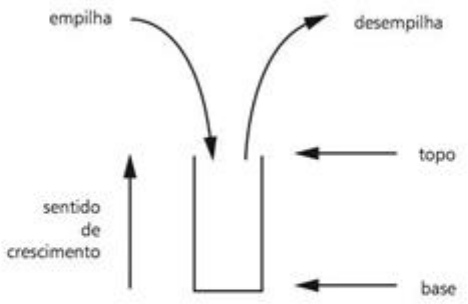


**figura 4.3** Estrutura do tipo pilha.

De forma análoga ao exemplificado, outras estruturas de dados podem ser construídas, como vetores, matrizes, filas, listas e listas encadeadas.

### 4.3.4 endereçamento indireto e recursão

A seguir, é exemplificado, em programas do tipo monolítico, como definir desvios usando endereçamento indireto (determinado pelo conteúdo de um registrador).

**exemplo 4.19** – Endereçamento indireto em um programa monolítico

Uma operação com endereçamento indireto da seguinte forma, onde A é um registrador:

```
r:   faça F vá_para A
```

pode ser definida pelo seguinte programa monolítico:

```
r:   faça F vá_para End_A
```

onde a macro End_A, representada na forma de fluxograma na figura 4.4, trata o endereçamento indireto de A (e onde: cada circunferência rotulada representa um desvio incondicional para a correspondente instrução; entenda-se A+k como uma notação simplificada de A+1 repetida k vezes). De forma análoga (sugere-se como exercício), é possível definir um teste com endereçamento indireto como segue, onde A e B são registradores:

```
r:   se T então vá_para A senão vá_para B
```

—

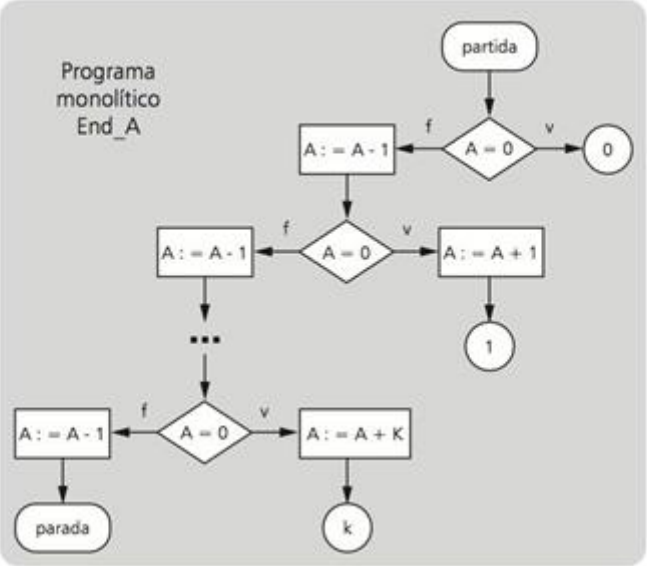


**figura 4.4** Fluxograma para tratar endereçamento indireto.

**observação 4.3** – Programa monolítico × programa recursivo.

Em Norma, sub-rotinas e mecanismos de recursão como os definidos nos programas recursivos podem ser simulados por programas monolíticos, usando endereçamento indireto. Tais demonstrações não serão estudadas neste livro. ❑

### 4.3.5 cadeias de caracteres

Cadeia de caracteres é um tipo de dado não predefinido na máquina Norma. O tratamento da definição e da manipulação de cadeias de caracteres será realizado por meio de outra máquina universal, denominada máquina de Turing, estudada adiante, a qual se prova ser equivalente à máquina Norma.

## 4.4 ⇢ conclusões

Neste capítulo, foram introduzidas as máquinas de registradores. É especialmente interessante observar como uma máquina tão simples e rudimentar pode ser tão poderosa. De fato, os recursos necessários para computar são bem poucos. Todos os demais recursos existentes

em máquinas reais como nos computadores modernos são apenas artifícios para facilitar a programação da computação.

Em particular, foi estudada a máquina Norma que possui três tipos de instruções/teste (**ad**, **sub** e **zero**) sobre registradores naturais. Evidências internas de que Norma é uma máquina universal foram apresentadas, evidências externas serão vistas nos próximos capítulos.

## 4.5 ···→ exercícios

**exercício 4.1** Analise o programa monolítico abaixo e assinale a opção que contém a codificação correta (considere a seguinte ordem de operações F=1, G=2, H=3 e dos testes $T_1$=1, $T_2$=2).

**Programa monolítico**

```
1:   faça F vá_para 2
2:   se T1 então vá_para 3 senão vá_para 1
3:   faça G vá_para 4
4:   se T2 então vá_para 1 senão vá_para 5
5:   faça H vá_para 6
6:   se T2 então vá_para 0 senão vá_para 5
```

**a** $2^{3675} \cdot 3^{5250} \cdot 5^{135025} \cdot 7^{151260} \cdot 11^{496371875} \cdot 13^{30256}$
**b** $2^{375} \cdot 3^{525} \cdot 5^{1350625} \cdot 7^{151630} \cdot 11^{4963311875} \cdot 13^{3025}$
**c** $2^{3675} \cdot 3^{5250} \cdot 5^{13505625} \cdot 7^{1512630} \cdot 11^{49633171875} \cdot 13^{302526}$
**d** $2^{375} \cdot 3^{5250} \cdot 5^{1355625} \cdot 7^{151260} \cdot 11^{49633171875} \cdot 13^{30256}$
**e** $2^{3675} \cdot 3^{5250} \cdot 5^{13505625} \cdot 7^{152630} \cdot 11^{4963171875} \cdot 13^{3526}$

**exercício 4.2** Codifique o seguinte programa monolítico e escolha a alternativa que representa o resultado de forma correta.

**Programa monolítico**

```
1:   se T1 então vá_para 2 senão vá_para 3
2:   faça F1 vá_para 3
3:   faça F2 vá_para 4
4:   se T1 então vá_para 1 senão vá_para 0
```

**a** $2^{51450} \cdot 3^{128625} \cdot 5^{13505625} \cdot 7^{30}$
**b** $2^{2} \cdot 3^{12825} \cdot 5^{13505625} \cdot 7^{10}$
**c** $2^{450} \cdot 3^{108625} \cdot 4^{13505625} \cdot 7^{20}$
**d** $2^{40532} \cdot 3^{128888} \cdot 4^{13505625} \cdot 7^{70}$
**e** $2^{2345} \cdot 3^{1555555} \cdot 5^{13505625} \cdot 7^{30}$

**exercício 4.3** Dado um programa monolítico abaixo, sabe-se que sua codificação é da forma $p = 2^x \cdot 3^y \cdot 5^z$.

**Programa monolítico**

```
1:    se T1 então vá_para 4 senão vá_para 3
2:    faça F1 vá_para 1
3:    se T1 então vá_para 2 senão vá_para 1
```

Marque a alternativa correta:

**a** x = 643125, y = 128625 e z = 26250
**b** x = 1286250, y = 105 e z = 1050
**c** x = 643125, y = 210 e z = 525
**d** x = 36750, y = 128625 e z = 26250
**e** x = 1286250, y = 128625 e z = 1050

**exercício 4.4** Considere o seguinte programa monolítico:

**Programa monolítico**

```
1:    se T então vá_para 2 senão vá_para 3
2:    faça F vá_para 6
3:    se U então vá_para 5 senão vá_para 4
4:    faça G vá_para 0
5:    faça F vá_para 0
6:    se U então vá_para 4 senão vá_para 1
```

Assinale a quádrupla ordenada que *não* corresponde a nenhuma codificação das instruções rotuladas existentes neste programa:

**a** (0, 1, 1, 0)
**b** (0, 2, 0, 0)
**c** (1, 1, 2, 3)
**d** (1, 2, 4, 1)
**e** (1, 2, 5, 4)

**exercício 4.5** Considere o seguinte programa monolítico:

**Programa monolítico**

```
1:    faça F vá_para 2
2:    se T1 então vá_para 1 senão vá_para 3
3:    faça G vá_para 4
4:    se T2 então vá_para 5 senão vá_para 1
```

Marque a correspondente codificação correta:

**a** $2^{(2^0 3^1 5^2 7^2)} + 3^{(2^1 3^1 5^1 7^3)} + 5^{(2^0 3^2 5^4 7^4)} + 7^{(2^1 3^2 5^5 7^1)}$
**b** $2^{(2^1 3^2 5^5 7^1)} \cdot 3^{(2^1 3^1 5^1 7^3)} \cdot 5^{(2^0 3^2 5^4 7^4)} \cdot 7^{(2^0 3^1 5^2 7^2)}$
**c** $2^{(2^0+3^1+5^2+7^2)} \cdot 3^{(2^1+3^1+5^1+7^3)} \cdot 5^{(2^0+3^2+5^4+7^4)} \cdot 7^{(2^1+3^2+5^5+7^1)}$
**d** $2^{(2^2 3^1 5^2 7^2)} \cdot 3^{(2^3 3^1 5^1 7^3)} \cdot 5^{(2^0 3^2 5^4 7^4)} \cdot 7^{(2^1 3^2 5^5 7^1)}$
**e** $2^{(2^0 3^1 5^2 7^2)} \cdot 3^{(2^1 3^1 5^1 7^3)} \cdot 5^{(2^0 3^2 5^4 7^4)} \cdot 7^{(2^1 3^2 5^5 7^1)}$

**exercício 4.6** Considere o seguinte programa monolítico:

**Programa monolítico**

```
1:   faça F vá_para 2
2:   se T1 então vá_para 1 senão vá_para 3
3:   faça F2 vá_para 0
```

Marque a alternativa que representa o número natural associado pela função de codificação codigo(P):

**a** 340341750
**b** $2^{7350} \cdot 3^{5145} \cdot 5^{18}$
**c** $2^{3675} \cdot 3^{10290} \cdot 5^{9}$
**d** 583200
**e** 77145

**exercício 4.7** Qual dos números abaixo representa a codificação de um programa monolítico?

**a** $2^{3675} \cdot 3^{18375} \cdot 5^{4410}$
**b** $2^{11025} \cdot 3^{1470} \cdot 5^{5292}$
**c** $2^{9261} \cdot 3^{6174} \cdot 5^{9450}$
**d** $2^{8505} \cdot 3^{13230} \cdot 5^{2835}$
**e** 9447840

**exercício 4.8** Qual dos números abaixo *não* representa a codificação de uma instrução de programa monolítico?

**a** 210
**b** 450
**c** 315
**d** 750
**e** 525

**exercício 4.9** Considere o seguinte programa monolítico codificado:

$$p = 2^{51450} * 3^{105} * 5^{9}$$

Marque a alternativa que representa a reescrita como um programa iterativo:

**a**

```
enquanto T faça (F;G)
```

**b**

```
até T faça (F);G
```

**c**

```
até T faça (F;G)
```

**d**

```
enquanto T faça (F);G
```

**e**

```
enquanto T faça (F)
```

**exercício 4.10** Codifique o seguinte programa iterativo e marque a opção correta:

```
enquanto T1 faça (F1);
F2;
(Se T2
então faça até T1 faça (F3;F1);✔
senão faça ✔)
```

**a** $2^{(2^1 3^1 5^2 7^3)} * 3^{(2^0 3^1 5^1 7^1)} * 5^{(2^1 3^2 5^4 7^4)} * 7^{(2^0 3^2 5^5 7^0)} * 11^{(2^1 3^3 5^6 7^6)} * 13^{(2^1 3^1 5^7 11^7)} * 17^{(2^0 3^1 5^0 7^5)}$

**b** $2^{(2^1 3^1 5^2 7^3)} * 3^{(2^0 3^1 5^1 7^1)} * 5^{(2^0 3^2 5^4 7^4)} * 7^{(2^1 3^2 5^5 7^0)} * 11^{(2^1 3^1 5^0 7^6)} * 13^{(2^0 3^3 5^7 7^7)} * 17^{(2^0 3^1 5^5 7^5)}$

**c** $2^{(2^0 3^1 5^2 7^3)} * 3^{(2^1 3^6 5^1 7^1)} * 5^{(2^1 3^2 5^4 7^4)} * 7^{(2^0 3^2 5^5 7^0)} * 11^{(2^1 3^3 5^6 7^6)} * 13^{(2^1 3^1 5^7 7^7)} * 17^{(2^0 3^1 5^0 7^7)}$

**d** $2^{(2^1 3^1 5^2 7^3)} * 3^{(2^0 3^1 5^1 7^1)} * 5^{(2^0 3^2 5^4 7^4)} * 7^{(2^1 3^2 5^5 7^0)} * 11^{(2^0 3^3 5^6 7^6)} * 13^{(2^0 3^1 5^7 7^7)} * 19^{(2^1 3^1 5^0 7^5)}$

**e** $2^{(2^0 3^1 5^2 7^3)} * 3^{(2^1 3^1 5^1 7^1)} * 5^{(2^1 3^2 5^4 7^4)} * 7^{(2^0 3^2 5^5 7^0)} * 11^{(2^1 3^3 5^6 7^6)} * 13^{(2^5 3^1 5^7 7^7)} * 19^{(2^0 3^1 5^0 7^5)}$

**exercício 4.11** Codifique o fluxograma ilustrado na figura 4.5 como um número natural, usando a codificação de programas introduzida no exemplo 4.2 – Codificação de programas monolíticos.

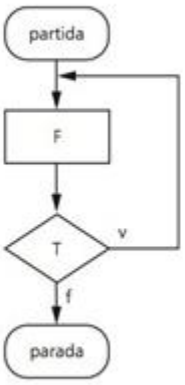


**figura 4.5** Fluxograma.

**exercício 4.12** Codifique o programa da figura 4.6 usando a codificação de programas monolíticos.

**Programa monolítico M**

```
1:    faça F vá_para 2
2:    se T então vá_para 3 senão vá_para 1
3:    faça G vá_para 4
4:    se T então vá_para 1 senão vá_para 0
```

**figura 4.6** Programa monolítico.

**exercício 4.13** Desenvolva um *software* que seja capaz de processar qualquer programa de máquina Norma. A entrada para o *software* contém:

- o programa no formato de instruções rotuladas simples;
- o valor de entrada a ser computado pelo programa;

A saída é a computação do programa para a dada entrada. Teste o *software* para o cálculo do fatorial de 3.

**exercício 4.14** Mostre como a instrução abaixo pode ser construída como uma macro em Norma:

```
r1: se A<2 então vá_para r2 senão vá_para r3
```

**exercício 4.15** Desenvolva os programas, em Norma, que realizam as operações e testes abaixo:

**a** A:= B - C
**b** div(B, C): divisão inteira de B por C
**c** fat(A): fatorial;
**d** pot(x, y): potência;
**e** teste(A > B)
**f** teste(A ≥ B)
**g** teste(A ≤ B)
**h** mdc(A, B): máximo divisor comum;
**i** mod(A, B): resto da divisão inteira;
**j** primos(A, B): todos os primos entre os valores de A e de B;
**k** teste_nperf(A): verifica se é um número perfeito.

**exercício 4.16** Desenvolva os programas, em Norma, que realizam as operações abaixo nos inteiros (utilize a representação sinal-magnitude introduzida no exemplo 4.15 - Inteiros):

**a** A:= B + C
**b** A:= B • C
**c** A:= B - C
**d** A:= div(B, C)

**exercício 4.17** Desenvolva os programas, em Norma, que realizam as operações abaixo nos números racionais (utilize a representação de racionais introduzida no exemplo 4.16 - Racionais):

**a** A:= B + C
**b** A:= B • C
**c** A:= B - C
**d** A:= div(B, C)

**exercício 4.18** Seja a *máquina Normaneg*, a qual é, em todos os aspectos, idêntica à Norma, exceto pelo fato de poder armazenar números negativos (inteiros) em seus registradores. Prove que, para cada fluxograma Normaneg, pode-se definir um fluxograma Norma equivalente sem o uso de registradores extras.

*Sugestão*: utilize a representação de inteiros baseada na função codificação e desenvolva os programas para as instruções de Norma que simulam as de Normaneg.

**exercício 4.19** Seja P = { I, 1 } definido na máquina Norma e suponha o conteúdo dos registradores A e B igual a 0 e 5 respectivamente. Marque o programa que realiza A := B na máquina Norma:

**a**

**Programa monolítico**

```
1:   se zero_B então vá_para 5 senão vá_para 2
2:   faça ad_D vá_para 3
3:   faça sub_B vá_para 4
4:   faça sub_A vá_para 1
```

**b**

**Programa monolítico**

```
1:   se zero_A então vá_para 4 senão vá_para 2
2:   faça ad_A vá_para 3
3:   faça ad_B vá_para 1
```

**c**

**Programa monolítico**

```
1:   se zero_A então vá_para 5 senão vá_para 2
2:   faça ad_A vá_para 3
3:   faça sub_B vá_para 1
```

**d**

**Programa monolítico**

```
1:   se zero_B então vá_para 5 senão vá_para 2
2:   faça ad_A vá_para 3
3:   faça sub_B vá_para 1
```

**e** Nenhuma das alternativas anteriores está correta.

**exercício 4.20** A máquina Norma é composta basicamente por um conjunto de registradores naturais e apenas três instruções sobre cada um deles (adição e subtração de uma unidade e verificação se o conteúdo do registrador é zero). Portanto, é correto afirmar que:

**a** Seus registradores podem assumir valores naturais tão grandes quanto necessários;
**b** Os registradores podem assumir qualquer valor;
**c** Seu modelo formal encontra-se baseado em uma estrutura chamada fita;
**d** Não é capaz de simular mecanismos de recursão;
**e** Pode ser definida por uma 8-upla da forma ($\sum$, Q, $\Pi$, $q_0$, F, V, $\beta$, **O**).

**exercício 4.21** Sobre a máquina Norma, analise as seguintes afirmações:

I. É uma máquina extremamente simples, porém com poder computacional igual ao de qualquer computador atual;
II. Em norma, sub-rotinas e mecanismos de recursão, como os definidos em programas recursivos, podem ser simulados por programas monolíticos, usando endereçamento indireto;
III. É impossível programar operações matemáticas complexas (radiciação, seno e cosseno, por exemplo) em Norma, visto que apenas as instruções de somar/diminuir 1 unidade a um registrador não dão suporte para que isso seja feito.

Marque a alternativa correta:

**a** Apenas I está correta;
**b** Apenas II está correta;
**c** Apenas III está correta;
**d** Apenas I e II estão corretas;
**e** Apenas II e III estão corretas.

**exercício 4.22** Sobre a máquina Norma, analise as seguintes afirmações:

I. Possui um conjunto finito de registradores naturais como memória;
II. Possui poder computacional, no mínimo, igual ao de qualquer computador moderno;
III. Sub-rotinas e mecanismos de recursão podem ser simulados por programas monolíticos, usando endereçamento indireto.

Marque a alternativa correta:

**a** Apenas I está correta;
**b** Apenas II está correta;
**c** Apenas I e III estão corretas;
**d** Apenas II e III estão corretas;
**e** I, II e III estão corretas.

## Termos-chaves

codificação de programas, p. 106
máquina universal, p. 105
máquina Norma, p. 105, p.109
Número de Gödel, p. 105