A **Universidade de São Paulo**
**Instituto de Matemática e Estatística**
DCC – Leônidas O. Brandão

# Computador à Gaveta

O objetivo deste texto é introduzir os primeiros conceitos de algoritmos a partir de um modelo de instruções elementares, mais próximas à *linguagem de máquina*. Para isso introduziremos um modelo simplificado de computador, baseado materiais concretos, como um arquivo com várias gavetas numeradas e uma calculadora simples. Utilizaremos este modelo para ilustrar funcionamento de um computador real, explorando a linguagem de comunicação que deve ser empregada, principalmente para diferenciarmos as linguagens de programação *baixo nível* e *alto nível*, dentre outras questões. Em função deste modelo concreto das gavetas, denominaremos este "computador" de `CG-1`.

Ao indivíduo que utiliza o computador `CG-1` denominaremos por **usuário**. Este usuário dispondo de um conjunto de instruções, que representem a solução computacional de algum problema, as insere no computador por algum **dispositivo de entrada** e dá um comando de início de execução. O `CG-1` passa a processar as instruções, sem qualquer interferência do usuário até que existe uma instrução que requeira interferência do usuário (entrada de dados), ou ocorra algum evento inesperado (como uma divisão por zero), ou termine a execução das instruções.

A esta lista de instruções da-se o nome de **programa**.

Uma implementação em `Java` deste modelo pode ser encontrada em

`http://www.matematica.br/programas/icg`.

## 1 Componentes

O computador à gaveta `CG-1` é composto pelos seguintes dispositivos:

1. Um gaveteiro com 100 gavetas (memória).
2. Um porta cartões (dispositivo de entrada).
3. Uma lousa de saída (dipositivo de saída).
4. Uma calculadora manual (unidade de processamento aritmético).
5. Uma lousa denominada EPI (registrador de endereço de próxima instrução).
6. Um operador do sistema, um ser "não pensante", dispondo de giz, e apto a executar, apenas, um pequeno número de instruções (*chip* central - unidade central de processamento).

### 1.1 Gaveteiro (Memória)

O gaveteiro consiste de 100 gavetas, numeradas de 00 a 99, que denominaremos **memória** (equivalente a **RAM - "Random Acess Memory"** das máquinas comerciais). À cada um desses números denomina-se **endereço** (da gaveta correspondente) e às gavetas, **posições de memória**. Deste modo, as gavetas fazem o papel das palavras num computador real, é a menor unidade de memória que pode ser endereçada diretamente.

Cada gaveta contém uma lousa para informações que comporta três dígitos (como por exemplo: 010, 143, etc) ou dados com até seis dígitos (se negativo, cinco). Veremos que dependendo do estado do sistema esta informação será interpretada como **dado** ou **instrução**.

Apenas o operador pode manusear o gaveteiro, que é contituído de modo que:

i. em cada instante não mais que uma gaveta pode estar aberta;

ii. é permitida a leitura da lousa (por parte do operador) de alguma gaveta aberta;

iii. o operador pode escrever uma nova informação na lousa de uma gaveta aberta, após apagar a informação original;

iv. ao ligar o sistema, o operador escreve qualquer informação (número com três dígitos) em cada uma das gavetas.

### 1.2 Porta cartões (Dispositivo de entrada)

É um dispositivo de entrada, onde o usuário deposita cartões contendo exatamente três dígitos, com estrutura de fila, ou seja, pode-se inserir novos cartões apenas na extremidade superior do porta cartões e as retiradas são feitas exclusivamente da extremidade inferior.

Inserir cartões é um privilégio do usuário e apenas ao operador é permitida a retirada dos mesmos. As informações contidas nestes cartões serão interpretadas como dados ou instruções, dependendo do contexto do sistema.

### 1.3 Lousa de saída (Dispositivo de saída)

Este será o **dispositivo de saída**, que possibilitará ao usuário obter alguma resposta do sistema, na forma de três dígitos. É permitido apenas ao operador escrever nesta lousa.

Quando o operador escrever na lousa de saída o número 100, isto deverá ser entendido pelo usuário como tendo ocorrido um erro (não distinguiremos erros).

### 1.4 Calculadora manual (Unidade aritmética - "chip")

Esta é uma calculadora simples, que permite realizar apenas as operações aritméticas básicas, sem ponto ou vírgula decimal, sendo manuseada exclusivamente pelo operador. Ao mostrador ("display") da calculadora denominaremos por **acumulador (AC)**.

Este dispositivo é comandado somente pelo operador, utilizando no máximo seis dígitos. No caso de uma expressão resultar número com mais de seis dígitos, o resultado pode ser muito diferente do que esperaríamos (vide o quadro Q2 abaixo).

Toda operação aritmética será efetuada entre o acumulador e uma posição de memória, como nas calculadoras manuais não programáveis. Mais especificamente, representando por $\oplus$ uma operação aritmética qualquer reconhecida pelo `CG-1`, os cálculos aritméticos terão a forma

AC ← conteúdo de AC ⊕ conteúdo de EE.

Deste modo, para se efetuar o cálculo de uma expressão aritmética, será necessário decompô-la em várias partes elementares, como ilustrado no exemplo a seguir.

**Exemplo 1** *Cálculo da expressão aritmética 10*9, supondo que o valor 10 está na gaveta de endereço $EE_1$ e o 9 em $EE_2$.*

| Instrução | Valor do AC antes da execução da instrução |
|---|---|
| 1. coloque o conteúdo da gaveta $EE_1$ (10) no acumulador | ? |
| 2. some ao acumulador o conteúdo da gaveta $EE_2$ (9) | 10 |
| 3. (neste ponto o AC tem o valor da expressão 10*9) | 90 |

A seguir apresentamos mais dois exemplos, mas neles eliminamos os passos de tratamento do acumulador/memória, para destacarmos apenas a sequência em que os cálculos são efetuados e eventual problema de arredondamento/truncamento. Além disso simplificaremos também em relação à quantidade de dígitos na memória, limitando a apenas três.

**Exemplo 2** *Cálculo da expressão aritmética ( (10*9)-89 ) * 30, utilizando no máximo três dígitos.*

| AC | Operação | Operando | Situação |
|---|---|---|---|
| 010 | * | 009 | OK |
| 090 | - | 089 | 0K |
| 001 | * | 030 | OK |
| **030** | | | Resultado |

Q1. Expressão aritmética com resultado válido

**Exemplo 3** *Cálculo de uma expressão aritmética ( ( (-2 * 10) - 10 ) * 2 * 3) que provoca "sobrecarga" ("overflow"), utilizando no máximo três dígitos.*

| AC | Operação | Operando | Situação |
|---|---|---|---|
| -02 | * | 010 | OK |
| -20 | - | 010 | OK |
| -30 | * | 002 | OK |
| -60 | * | 003 | ERRO |
| **180** | | | Resultado |

Q2. Seqüência de operações com truncamento: ( (-2 * 10) - 10 ) * 2 * 3 = 180 ≠ -180.

Devemos notar que o resultado da seqüência de operações, representadas no quadro acima, é $\boxed{1}\boxed{8}\boxed{0}$.

### 1.5 EPI

É uma lousa, que comporta apenas dois dígitos, utilizada pelo operador. O nome desta lousa advém de **Endereço de Próxima Instrução**. Veremos na seção seguinte a razão desse nome.

### 1.6 Operador

O operador do sistema é um indivíduo que executa um número limitado de ações ou **instruções** (que descreveremos mais claramente na seção 2), dentre as quais, manusear a memória, retirar cartões do dispositivo de entrada, escrever no dispositivo de saída, alterar o acumulador (através de operações na calculadora) e o EPI.

Assim que o sistema é ligado o operador entra em **estado de carga**, fazendo as seguintes operações:

- escreve qualquer número em cada uma das posições de memória e AC;
- escreve 000 no EPI;
- recolhe um a um, os cartões no dispositivo de entrada, copiando seus valores, respectivamente, nas lousas das gavetas 00, 01, 02 e assim por diante.[1]

O operador pára o recolhimento quando encontra um cartão especial com a seqüência de dígitos 000, que também é carregado para uma posição de memória. Na hipótese do usuário esquecer o cartão 000 o operador espera indefinidamente pelo mesmo. Se o número de cartões, no estado de carga, exceder 100, ocorre um erro de transbordamento ("overflow"). Neste caso o operador interrompe o recolhimento, escrevendo no dispositivo de saída o número

| 1 | 0 | 0 | Erro: Transboradamento |
|---|---|---|---|

descartando os cartões restantes e reiniciando um estado de carga, para que outro programa possa ser executado (podendo ser o programa anterior).

A única operação conhecida pelo operador é somar uma unidade ao conteúdo do EPI.

Após ler o cartão 000, o operador o escreve na gaveta correspondente, entrando em **estado de execução**, que detalharemos a seguir.

## 2 Programação do Sistema

A questão que mais interessa é:

"Como usar um computador para solucionarmos um dado problema?"

[1] Se houver algum número anteriormente escrito, este será removido.

Mais especificamente, estamos interessados em escrever uma seqüência de ordens, ou intruções, numa forma aceita pelo computador, que após a entrada dos dados nos forneça a resposta adequada. A esta seqüência de instruções denominamos **programa**[2].

No `CG-1`, os programas são inseridos através da leitura dos cartões do dispositivo de entrada, quando o operador está em estado de carga. As instruções, como dito anteriormente, são armazenadas a partir da gaveta 00, eventualmente até a 99, constituindo o que denominamos **programa armazenado**.

Um exemplo de problema, simples, que os alunos se deparam constantemente é

**Exemplo 4** *Conhecido o critério de avaliação de uma determinada disciplina e o conjunto de notas obtidas na mesma, desejamos computar a média final.*

Para resolver o problema do exemplo acima, seguiríamos os seguintes passos,

- Desenvolveríamos um **algoritmo**[3] adequado à resolução do problema: computar a nota final de acordo com o critério de avaliação;
- Codificaríamos este algoritmo em uma seqüência de instruções (programa) na linguagem da máquina que dispomos (ou poderíamos codificar em uma linguagem alto nível e posteriormente traduzir para a de máquina);
- "Rodaríamos", ou executaríamos, o programa com as notas (dados) obtidas pelo aluno.

Devido ao programa, referido no segundo item acima, poder ser executado na máquina utilizada, denominamos o mesmo por **programa executável** (nesta máquina).

Antes de definirmos o conjunto de instruções conhecidas pelo operador, vejamos como se dá a **execução** de um programa, ou seja, qual o procedimento seguido pelo operador quando esse entra em estado de execução:

1. Consulta o conteúdo do EPI;
2. Sendo EE o conteúdo do EPI, o operador passa a abrir a gaveta de número EE. Seja I (um número de três dígitos) o conteúdo desta gaveta;
3. Fecha a gaveta EE;
4. Incrementa em 1 o conteúdo do EPI;
5. Executa a instrução I;
6. Se a instrução I não for final de programa (000) o operador volta ao passo 1.

Apresentamos a seguir dois exemplos de "algoritmos", com os quais nos deparamos comumente.

[2] Uma seqüência de instruções, de um conjunto "pequeno" de regras, que sob os mesmos dados de entrada provoca sempre a mesma saída, em um número finito de passos

[3] Simplificadamente, uma seqüência de instruções que após um número finito de passos fornece uma resposta.

**Exemplo 5** *Um indivíduo que nunca utilizou um telefone público, dispondo da informação que é necessário adquirir fichas ou um cartão da TELESP,*[4] *poderá efetuar sua ligação consultando apenas o conjunto de informações (ou "programa") disposto junto ao aparelho. Neste caso a execução de "programa" seria a inserção da ficha (ou cartão) no aparelho e discagem do número desejado.*

**Exemplo 6** *A seqüência de operações que efetuamos para fazer a multiplicação de dois números é um algoritmo que pode ser formalizado e colocado na forma de um programa, executável em algum computador, cujos dados serão os fatores da multiplicação.*

# 3 Instruções Válidas

Podemos perceber das definições anteriores que os valores armazenados em uma gaveta podem ser considerados tando dados quanto instruções. Apesar disto, o operador jamais ficará sem saber como interpretá-los, pois como dissemos anteriormente, o estado em que o operador se encontra (ou simplesmente, **estado do sistema**) resolverá esta situação, como veremos mais adiante.

Por simplicidade de notação utilizaremos as seguintes convenções:

| | |
|---|---|
| **AC** | acumulador |
| **cAC** | conteúdo (valor) de AC |
| **EE** | endereço da gaveta genérica EE |
| **cEE** | conteúdo da gaveta EE. |

Podemos inferir da descrição dos componentes do computador à gaveta e do exemplo das médias que existem instruções para a entrada e saída de dados (informações) no sistema. De fato o computador à gaveta dispõe destas instruções, a primeira tem a forma:

"leia um cartão, armazenando seu conteúdo na gaveta EE",

fazendo com que o operador se dirija ao porta cartões, retirando o cartão a mais tempo na fila (no início da fila)[5] copiando a informação que este traz para a lousa da gaveta EE; a saída de dados tem a forma

"escreva (imprima) o cEE no dispositivo de saída"

fazendo com que o operador copie o número contido na gaveta EE para a lousa do dispositivo de saída.

Ou seja, entrada e saída de dados são efetuadas apenas com a memória (gaveteiro).

As operações aritméticas também farão uso da memória, porém um dos operadores deverá estar sempre no acumulador. Não apenas isto, o resultado ficará sempre no acumulador (conforme dito na subseção 1.4) como podemos notar na tabela 1 à frente.

Feitas estas considerações podemos apresentar esquematicamente o conjunto de instruções aceitas pelo `CG-1`.

[4] Ou de qualquer outra empresa de telefonia pública, como TELERJ, TELEMIG, etc.

[5] Se a fila de cartões estiver vazia o operador espera, indefinidamente, pela inserção, por parte do usuário, de um cartão.

| Código | "Assembler" | Instrução |
|---|---|---|
| 0EE | `AC <- cEE` | copie valor na gaveta EE (cEE) para AC |
| 1EE | `EE <- cAC` | copie valor no AC (cAC) na gaveta EE |
| 2EE | `AC <- cAC+cEE` | some cEE ao AC |
| 3EE | `AC <- cAC-cEE` | subtraia de AC o cEE |
| 4EE | `AC <- cAC*cEE` | multiplique o cAC por cEE |
| 5EE | `AC <- cAC/cEE` | divida o cAC por cEE |
| 6EE | `cAC>0, EPI <- EE` | se cAC $> 0$ vá para EE; |
| 7EE | `leia(EE)` | leia um valor e guarde na gavea EE |
| 8EE | `escreva(cEE)` | escreva cEE no dispositivo de saída |
| 9EE | `EPI <- cEE` | vá para cEE; |
| 0−N | `AC <- N` | AC recebe uma constante (truque) |
| 0 0 0 | `fim` | final de programa |

Tabela 1: instruções de máquina

As instruções do tipo 0 e 1 são utilizadas para se fazer atribuições de valores. A instrução 6EE é um **desvio condicional** (para o endereço EE) e a 9EE é um **desvio incondicional** (para EE).

Devemos notar que é um tanto quanto difícil nos lembrarmos de cada um destes códigos e seus respectivos significados, pois estes não matêm qualquer relação óbvia. Além disso, se necessitarmos escrever programas que resultem em um grande número de instruções, digamos 60 delas, será muito difícil, que outra pessoa ou o próprio programador, após passar um período sem ter contato com sua "obra"[6] entender o que aquela seqüência numérica significa.

Deste modo, para o desenvolvimento inicial de programas, lançaremos mão de um conjunto de signos mais próximo a nossa linguagem natural (colocando aí, as convenções usuais da matemática). Uma vez escrito o programa nesta linguagem, podemo traduzi-lo para a linguagem de máquina.

Usaremos o símbolo `;` para fazermos **comentários** ao programa, isto é, anotações que terão como finalidade tornar o programa mais **legível**, sendo desconsiderados na sua tradução para a linguagem de máquina.

# 4 Desenvolvimento de Programas

Uma vez conhecida a estrutura do `CG-1` e seu conjunto de instruções podemos atacar alguns problemas, procurando uma solução "implementável" no `CG-1`.

**Problema 1** *Desejamos somar os valores 3, 5 e* $-1$.

**Resolução**

(1) Inicialmente, podemos notar que a única instrução que soma é a **2EE**, ou seja,

$$\text{AC} \leftarrow \text{cAC} + \text{cEE}.$$

[6] Este é o **teste da gaveta**, para se certificar que seu programa pode ser entendido facilmente, deixe-o na "gaveta" (de sua cômoda) por um mês.

(2) Devemos inserir os valores 3, 5 e $-1$ no `CG-1`, e como no item (1) acima, dispomos de uma única instrução para este fim,

leia(EE); ler um valor e guardar na gaveta (memória) de endereço EE.

(3) Outra observação importante diz respeito aos dados que precisaremos armazenar, {3, 5, -1}. Como as instruções e dados são armazenados no mesmo "gaveteiro" e o operador coloca os primeiros seqüencialmente a partir da gaveta 00, podemos usar uma **variável**[7], que denotaremos por AP para indicar o endereço da primeira gaveta disponível após a última instrução de programa (000).

Trabalhando a partir das três observações acima, podemos construir a seguinte versão de programa,

```
1  ; AP será o endereço da primeira gaveta disponível após última instrução de programa
2  leia(AP)                    ; lê 3 para gaveta AP
3  leia(AP+1)                  ; lê 5 para a gaveta AP+1
4  leia(AP+2)                  ; lê -1 para a gaveta AP+2
5  AC ← cAP                    ; AC ← 3
6  AC ← cAC + c(AP + 1)        ; AC ← 3 + 5
7  AC ← cAC + c(AP + 2)        ; AC ← 3 + 5 − 1
8  (AP + 3) ← cAC              ; gaveta (AP + 3) ← 3 + 5 − 1
9  escreva(AP + 3)             ; imprime conteúda da gaveta 3 + 5 − 1
10 fim
```

**V.1**. Primeira versão de código para o `CG-1`

Ao codificarmos esse programa, descobrimos que existem 9 instruções (linhas 3 a 11 em V.1), sendo que a primeira instrução fica na gaveta 00. Portanto,

AP = 09.

Codificando este programa para o conjunto de instruções reconhecidos pelo `CG-1`, teremos 9 instruções e portanto *AP* deverá ser substituído por este valor. Deste, teremos a seguinte a seqüência de instruções para `CG-1` e os seguintes dados [8]

código: (709, 710, 711, 009, 210, 211, 112, 812, 000)
dados : (003, 005, -01).

Podemos notar que após o uso de cada um dos valores, 3 (na linha 6 de V.1), 5 (linha 7) e -1 (linha 8), as gavetas nas quais foram armazenados não são mais utilizadas, portanto existe a possibilidade usarmos uma única gaveta para armazenar todos eles. Vejamos como ficaria uma versão mais "econômica" no quanto ao gasto de memória.

[7] Um símbolo que armazena um valor numérico qualquer, dentro de seu escopo de definição.
[8] Por simplicitade, também representaremos os dados com apenas três dígitos.

| gaveta | instrução "assembler" | instrução máquina |
|---|---|---|
| 0 | leia(AP) | ; 709 |
| 1 | AC ← cAP | ; 209 |
| 2 | leia(AP) | ; 709 |
| 3 | AC ← cAC + cAP | ; 209 |
| 4 | leia(AP) | ; 709 |
| 5 | AC ← cAC + cAP | ; 209 |
| 6 | AP ← cAC | ; 109 |
| 7 | escreva(AP) | ; 809 |
| 8 | fim | ; 000 |

Nesta versão **economizamos memória**, visto que usamos uma única gaveta para os dados, a gaveta AP≡09.

**V.2** Versão melhorada de código, utilizando apenas uma gaveta para cálculos intermediários

Note que os conjuntos de instruções, das versões V.1 e V.2, são análogos (mesmo conjunto de instruções, com outros parâmetros), porém em outra ordem. Verifique o que aconteceria se as instruções de V.2 fossem reordenadas de modo a ficarem como em V.1, ou seja, colocadas na seqüência leia(AP), leia(AP), leia(AP), AC←cAP, ...

**Exercício 1** *É possível reescrever a versão V.1, mantendo a ordem inicial de instruções (leia(AP), leia(AP), leia(AP), AC←cAP, ...) e utilizar apenas uma gaveta para dados, produzindo o mesmo resultado ?*
*Se não for possível, justifique e apresente uma solução que utilize um número mínimo de gavetas, senão apresente sua solução.*
*Teste a solução encontrada no emulador do iCG, que pode ser descarregado a partir do endereço Web* `http://www.matematica.br/programas/icg`.

Programa em linguagem de máquina (complete) : .....
Dados : (003, 005, -01).

Uma vez que o programa está pronto é conveniente fazermos uma **simulação** do mesmo, ou seja, executarmos seqüencialmente cada uma das instruções do programa, sem o uso de nossa memória ou intuição, para termos "alguma" garantia sobre seu funcionamento. Este processo deve ser efetuado sempre que você terminar uma versão de programa.

**Nota:** a simulação pode ser efetuada com maior ou menor rigor, dependendo de sua confiança no programa, paciência ou tempo disponível, porém este processo não garante corretude do programa, um programa que funciona para um conjunto de dados, pode falhar em outros conjuntos, cuidado!

Técnicas para se efetuar simulação podem ser melhor compreendidas através de exemplos.

**Exemplo 7** *Uma simulação do programa V.2, considerando como dados de entrada os valores 3, 5 e -1, dispondo por colunas as informações relevantes ao programa, conforme a tabela abaixo. O resultado da simulação propriamente dito está na tabela 4.*

| **Coluna** | **Significado** |
|---|---|
| Instante | os instantes em que alguma instrução é executada; |
| Dados | a seqüência de valores lidos, nos respectivos instantes de leitura; |
| AP | a seqüência de valores que AP recebe, no respectivo instante; |
| AC | análogo à coluna AP; |
| Saída | os valores impressos no respectivo instante. |

Tabela 3. Convenções para os quadros de simulação

| Instantes | Dados | AP | AC | Saídas |
|---|---|---|---|---|
| 0 | 3 | ? | ? | |
| 1 | | 3 | | |
| 2 | | | 3 | |
| 3 | 5 | | | |
| 4 | | 5 | | |
| 5 | | | 8 | |
| 6 | $-1$ | | | |
| 7 | | $-1$ | | |
| 8 | | | 7 | |
| 9 | | 7 | | |
| 10 | | | | 7 |

Tabela 4. Quadro com a simulação para o programa V.2, utilizando como entradas a sequência $(3, 5, -1)$

No exemplo anterior, devemos notar a relação de passos de execução (instantes) e conjunto de instruções, cada instrução é executada uma única vez, portanto em um único passo. Porém esta observação não é geral, como veremos no próximo exemplo.