Introdução à Computação

Prof. Dr. Marcos Paulino Roriz Junior (marcosroriz@ufg.br)

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

ENGENHARIA DE TRANSPORTES

FCT
FACULDADE DE CIÊNCIAS E TECNOLOGIA

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Recapitulando

Algoritmo

- Conjunto de idéias precisas para a solução de um problema;
- Sequência precisa, sistemática e finita de passos ou instruções para solução de algum problema;
- Um algoritmo não representa, necessariamente, um programa de computador;
- **Em resumo: sequência de passos para resolver um problema.**

Dispositivos de Entrada

Teclado

Mouse

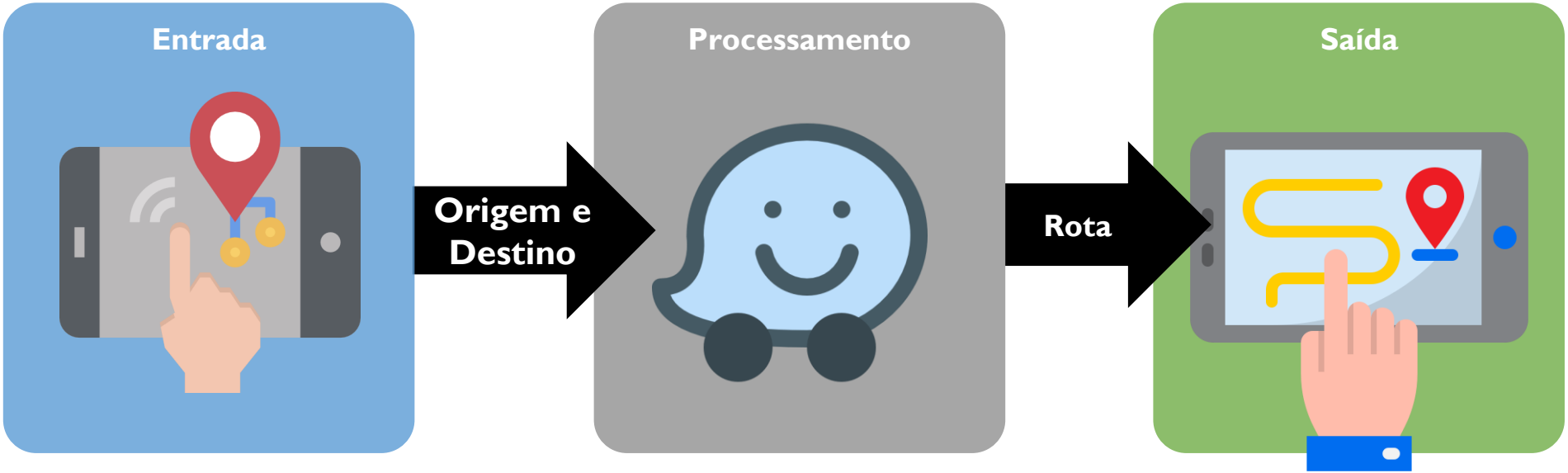
Tela do Celular

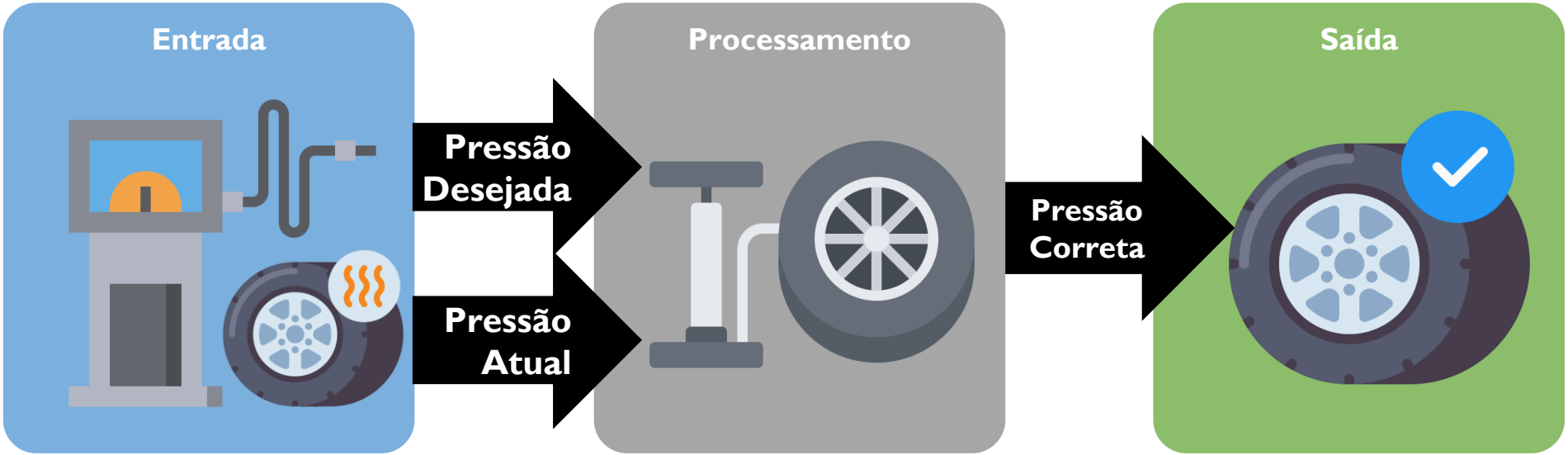
Dispositivos de Saída

Monitor

Impressora

Tela do Celular


Entrada
Origem e Destino
Processamento
Rota
Saída

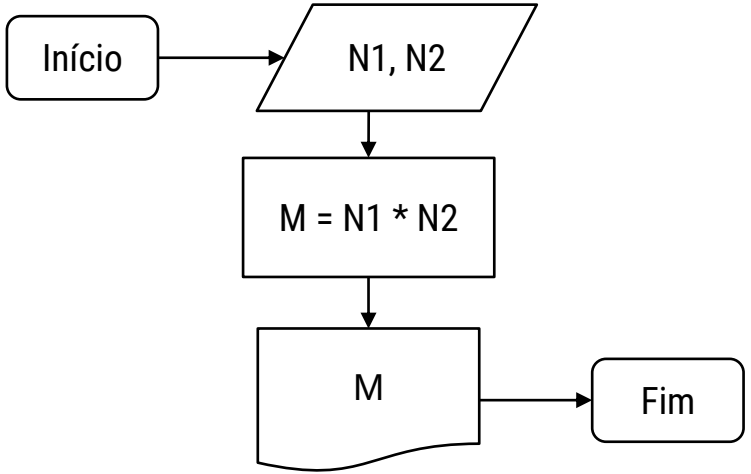

Entrada
Processamento
Saída
Pressão Desejada
Pressão Atual
Pressão Correta

Exemplos de Algoritmos

Algoritmo

- Faça um algoritmo para mostrar o resultado da multiplicação de dois números quaisquer.

1. ALGORITMO
2. DECLARE N1, N2, M NUMÉRICO
3. ESCREVA “DIGITE DOIS NÚMEROS”
4. LEIA N1, N2
5. M <- N1 * N2
6. ESCREVA "MULTIPLICAÇÃO = “, M
7. FIM_ALGORITMO



Algoritmo

- Escreva um algoritmo que lê uma medida em pés e uma medida em polegadas e imprime a medida correspondente em metros (número real).
  - Um pé equivale a 30,48cm e uma polegada equivale a 2,54cm.

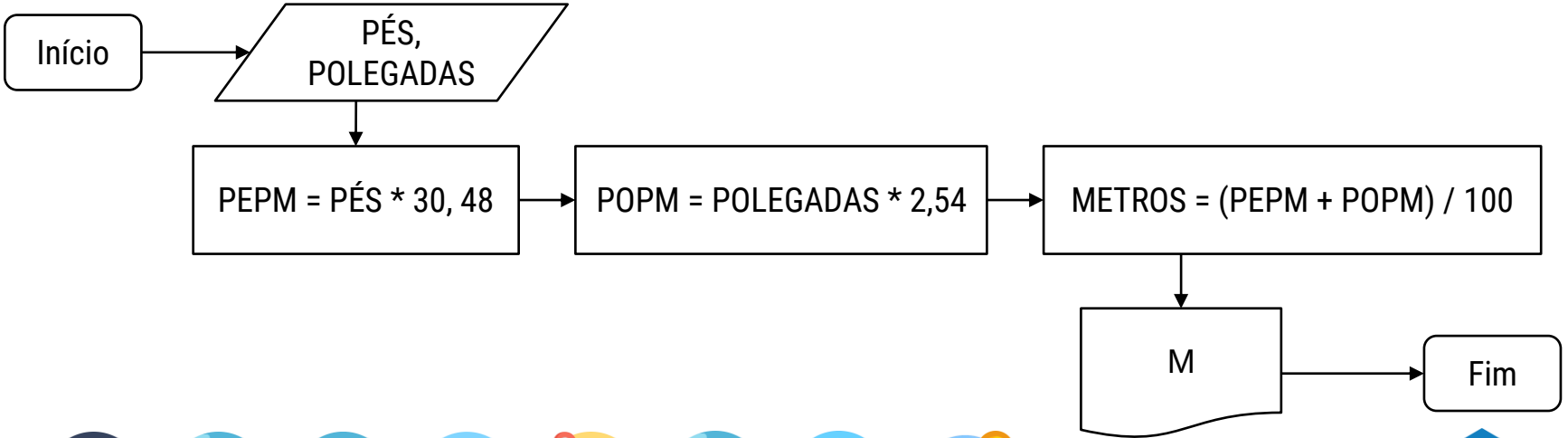
Algoritmo

- Escreva um algoritmo que lê uma medida em pés e uma medida em polegadas e imprime a medida correspondente em metros (número real).
  - Um pé equivale a 30,48cm e uma polegada equivale a 2,54cm.

```
ALGORITMO
    DECLARE PÉS, POLEGADAS, PEPM, POLPM, METROS COMO NUMÉRICO
    LEIA PÉS, POLEGADAS
    PEPM <- PÉS * 30,48
    POLPM <- POLEGADAS * 2,54
    METROS <- (PEPM + POLPM) / 100
    ESCREVA METROS
FIM_ALGORITMO
```

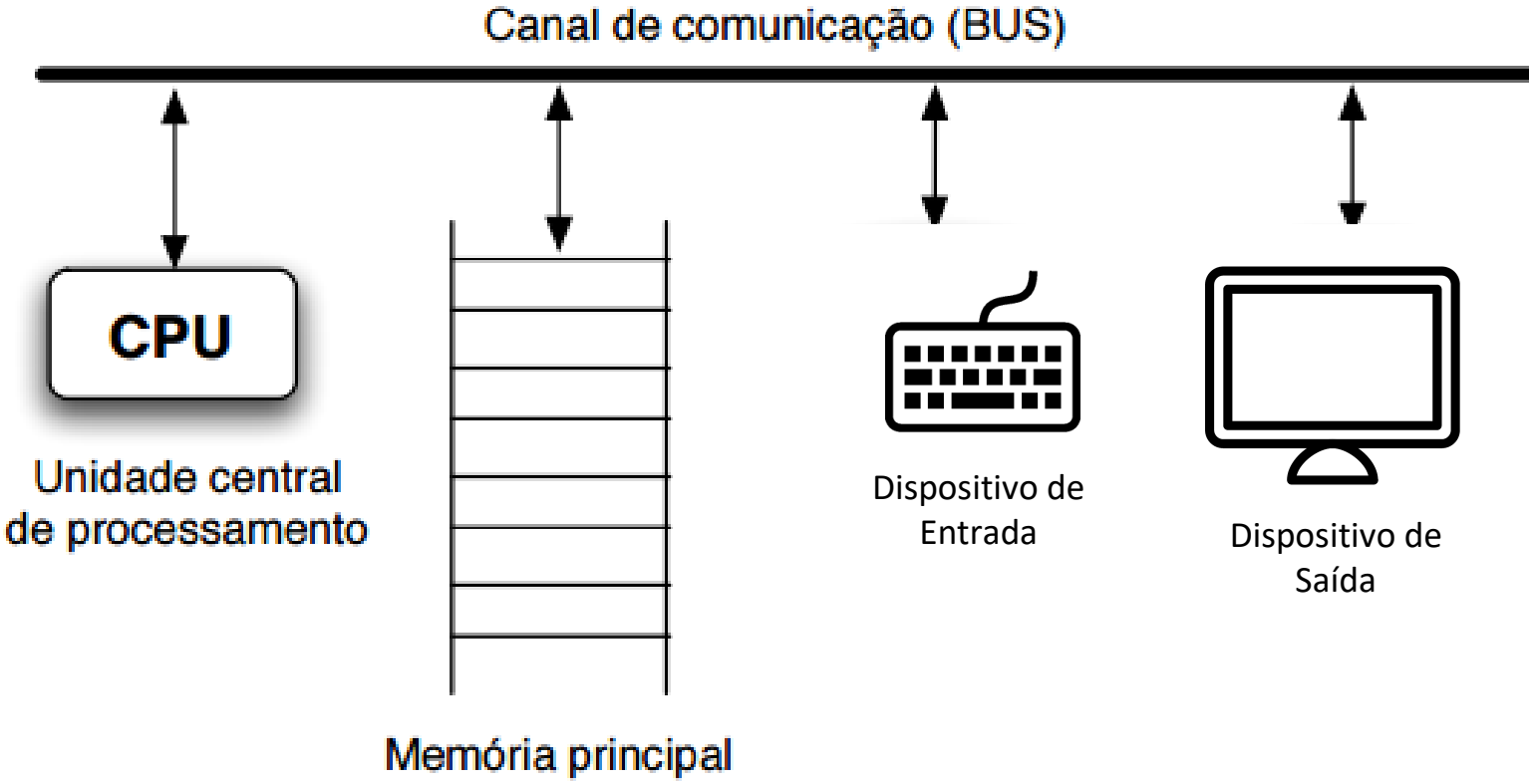
Algoritmo

- Escreva um algoritmo que lê uma medida em pés e uma medida em polegadas e imprime a medida correspondente em metros (número real).
  - Um pé equivale a 30,48cm e uma polegada equivale a 2,54cm.



Más como isso funciona?

Computador Simples

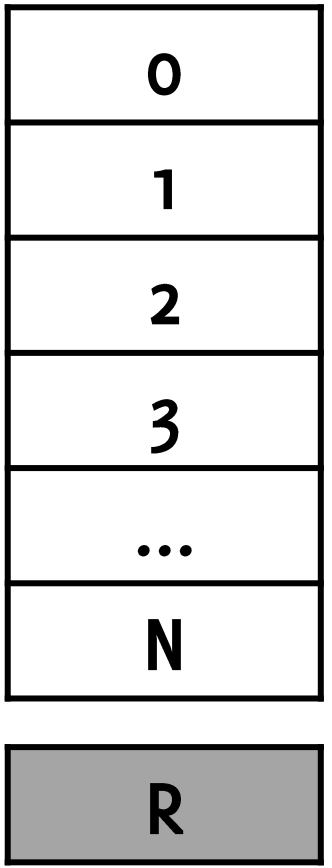


Computador Simples

- **Um computador simples (hipotético).**
  - Um dispositivo de entrada (teclado);
  - Um de saída (uma tela);
  - Memória Simples;
  - CPU com uma memória especial (registrador).

Memória

- Sequência de células de armazenamento endereçadas consecutivamente
- Cada célula possui um endereço (número entre 0 e n) e é capaz de armazenar um número inteiro
- Além das células anteriores, o C-HIP possui um único registrador, célula de armazenamento especial R



Instruções (Algoritmo)

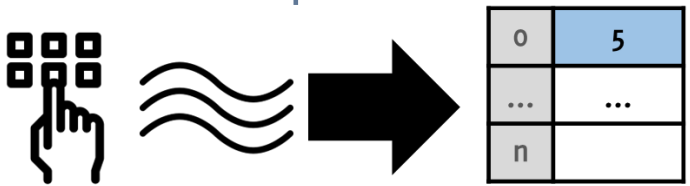
- Instruções: **ação** + *informação*
- Primeiras Instruções (Entrada/Saída):

`read` *x*	Captura/Lê um número do teclado e armazena em m[*x*].
`write` *x*	Escreve o valor armazenado na posição de memória *x* na tela.

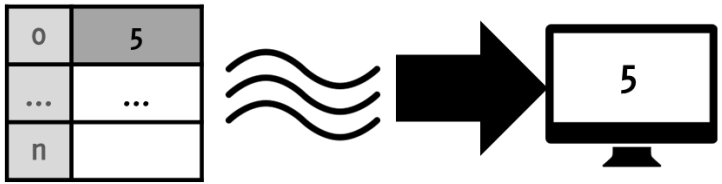
Exemplo

- Sequência de instruções
- O processador executa cada programa passo a passo
- O processamento termina após a última instrução ser executada

1. `read 0`



2. `write 0`



Instruções

- Instruções: **ação** + *informação*

`add` *x y*	Calcula a soma dos valores em *x* e *y* (m[x] + m[y]) e armazena em *R*.
`sub` *x y*	Calcula a dif. dos valores em *x* e *y* (m[x] - m[y]) e armazena em *R*.
`mul` *x y*	Calcula a mult. dos valores em *x* e *y* (m[x] * m[y]) e armazena em *R*.
`div` *x y*	Calcula a div. dos valores em *x* e *y* (m[x] / m[y]) e armazena em *R*.
`store` *x*	Armazena o valor do registrador (memória) *R* em m[x].
`storeconst` *num x*	Armazena o valor estático *num* na posição de memória m[x].

Instruções (AMD64)

ADD

Adds the value in a register or memory loca
a register or memory location (second opera

The instruction has two operands:

ADD *dest*, *src*

The instruction cannot add two memory ope
the length of the destination register or mem

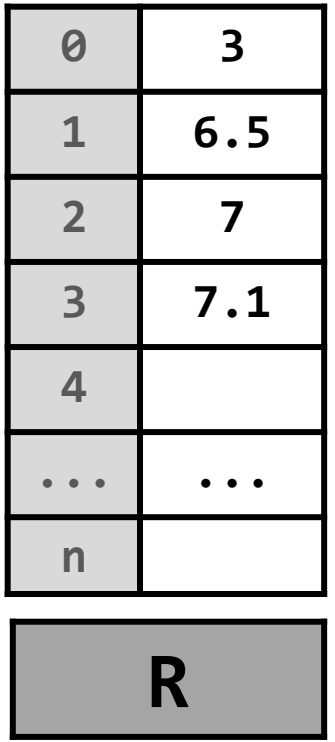
This instruction evaluates the result for bot
flags to indicate a carry in a signed or unsig
sign of a signed result.

BZHI . . . . . .
CALL (Near) . . . . . .
CALL (Far) . . . . . .
CBW
CWDE
CDQE . . . . . .
CWD
CDQ
CQO . . . . . .
CLC . . . . . .
CLD . . . . . .
CLFLUSH . . . . . .
CLFLUSHOPT . . . . . .
CLZERO . . . . . .
CMC

https://www.informit.com/articles/article.aspx?p=481859&seqNum=20
https://www.amd.com/system/files/TechDocs/24594.pdf

Calcular a média de 3 valores (e.g.: medir pH em três pontos)

```
1.  storeconst 3 0  # m[0] ← 3
2.  read 1          # m[1] ← read
3.  read 2          # m[2] ← read
4.  read 3          # m[3] ← read
5.  add 1 2         # R ← m[1] + m[2]
6.  store 4         # m[4] <- R
7.  add 3 4         # R ← m[3] + m[4]
8.  store 4         # m[4] ← R
9.  div 4 0         # R <- m[4] / m[0]
10. store 4         # m[4] <- R
11. write 4         # tela ← m[4]
```



O que esse programa faz?

```
storeconst 10 0
storeconst 20 1
storeconst 30 2
add 0 1
store 0
add 2 0
store 1
write 1
```

`read` *x*	Captura/Lê um número do teclado e armazena em m[*x*].
`write` *x*	Escreve o valor armazenado na posição de memória *x* na tela.
`add` *x y*	Calcula a soma dos valores em *x* e *y* (m[x] + m[y]) e armazena em *R*.
`sub` *x y*	Calcula a dif. dos valores em *x* e *y* (m[x] - m[y]) e armazena em *R*.
`mul` *x y*	Calcula a mult. dos valores em *x* e *y* (m[x] * m[y]) e armazena em *R*.
`div` *x y*	Calcula a div. dos valores em *x* e *y* (m[x] / m[y]) e armazena em *R*.
`store` *x*	Armazena o valor do registrador (memória) *R* em m[x].
`storeconst` *num x*	Armazena o valor estático *num* na posição de memória m[x].

Exercício

- Para converter graus Fahrenheit em graus Celsius, subtraia 32 e divida por 1,8.
- Faça um programa que converta uma temperatura de entrada em Fahrenheit em Celsius.

read *x*	Captura/Lê um número do teclado e armazena em m[*x*].
write *x*	Escreve o valor armazenado na posição de memória *x* na tela.
add *x y*	Calcula a soma dos valores em *x* e *y* (m[x] + m[y]) e armazena em *R*.
sub *x y*	Calcula a dif. dos valores em *x* e *y* (m[x] - m[y]) e armazena em *R*.
mul *x y*	Calcula a mult. dos valores em *x* e *y* (m[x] * m[y]) e armazena em *R*.
div *x y*	Calcula a div. dos valores em *x* e *y* (m[x] / m[y]) e armazena em *R*.
store *x*	Armazena o valor do registrador (memória) *R* em m[x].
storeconst *num x*	Armazena o valor estático *num* na posição de memória m[x].

Exercício

- Para converter graus Fahrenheit em graus Celsius, subtraia 32 e divida por 1,8.
- Faça um programa em C-HIP que converta uma temperatura de entrada em Fahrenheit em Celsius.

```
1. storeconst 32 0    # m[0] ← 32
2. storeconst 1.8 1   # m[1] ← 1.8
3. read 2             # m[2] ← read
4. sub 2 0            # R ← m[2] – m[0]
5. store 3            # m[3] <- R
6. div 3 1            # R <- m[3] / m[1]
7. store 3            # m[3] <- R
8. write 3            # tela ← m[3]
```

0	32
1	1.8
2	
3	
...	...
n	
R	

Instruções

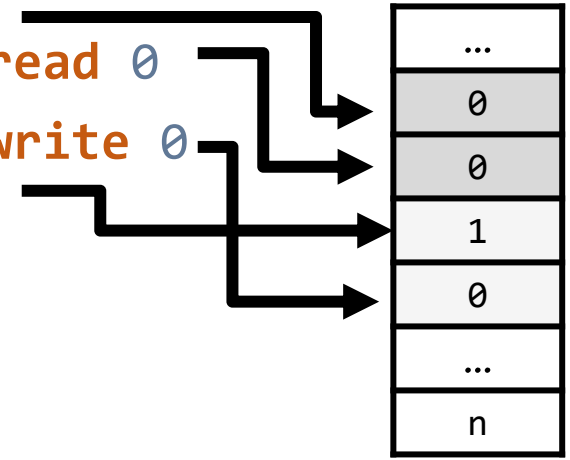
- Instruções manipulam o conteúdo da memória, mas onde elas estão?
- Resposta: Também na memória, codificadas como números

Instrução	Código
read	0
write	1
add	2
sub	3
mul	4
div	5
store	6
storeconst	7

Instruções

- Na prática, um programa é uma sequência de códigos de instrução e operandos armazenados na memória

- `read 0`
- `write 0`

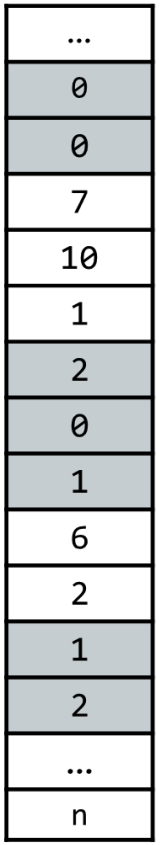


Instrução	Código
read	0
write	1
add	2
sub	3
mul	4
div	5
store	6
storeconst	7

Exemplo

- Veja o seguinte programa:

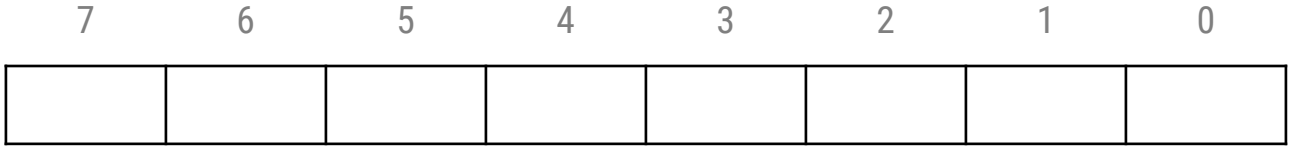
```
read 0
storeconst 10 1
add 0 1
store 2
write 2
```



Instrução	Código
read	0
write	1
add	2
sub	3
mul	4
div	5
store	6
storeconst	7

Memória

- A capacidade da memória é finita
- Cada célula armazena oito dígitos (*byte*) (0 a 9)



- O dígito mais significativo (7) armazena o sinal do número
  - 0 se positivo, 1 se negativo
- Os demais dígitos armazenam o número propriamente dito

Memória

- Exemplos

	7	6	5	4	3	2	1	0
355	0	0	0	0	0	3	4	5
-432	1	0	0	0	0	4	3	2
7012	0	0	0	0	7	0	1	2
0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0

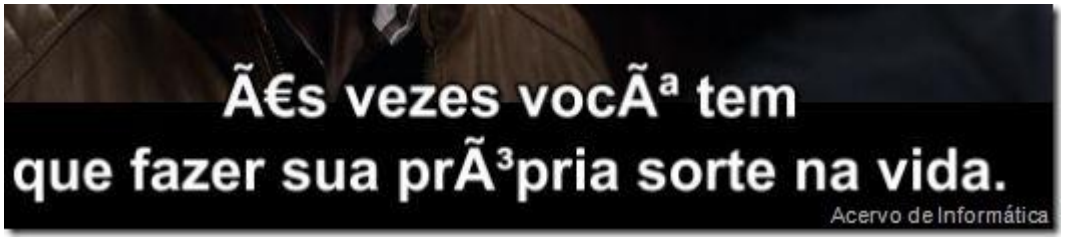
Tabela ASCII

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Imagem: https://en.wikipedia.org/wiki/ASCII

Problema em legendas



- UTF-8
- ISO 8859-1

- *Fonte: http://acervodeinformatica.blogspot.com/2015/12/legendas-com-problema-na-acentuacao.html*

- Qual é o maior número que podemos representar? E o menor?
- Qual o valor de R ao final dos seguintes trechos de código?
  - Overflow vs Underflow

	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0
9999999	0	9	9	9	9	9	9	9
-9999999	1	9	9	9	9	9	9	9

Limitações

- Números inteiros
- Ponto Fixo (Implícito)

	7	6	5	4	3	2	1	0	
355	0	0	0	0	0	3	4	5	. 0
-432	1	0	0	0	0	4	3	2	. 0

- Como representar números reais? Ex: 3.14 e -257.4

Solução: Notação Científica

- Notação científica
- $-257.4 = -2.574 \times 10^2$ ou -0.2574×10^3
- $3.14 = 3.14 \times 10^0$ ou 0.314×10^1

Indica posição do ponto

	7	6	5	4	3	2	1	0
-257.4	1	2	5	7	4	0	0	3
3.14	0	3	1	4	0	0	0	1

- **Ponto Flutuante**

Ponto Flutuante

- Permite “flutuar” a posição do ponto → representar números reais
- $x = 20.423$

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0

0	2	0	4	2	3	0	2

Computadores binários

- Números que estamos trabalhando: Base 10
- $415_{10} = (4 \times 10^2) + (1 \times 10^1) + (5 \times 10^0)$
- $4_{10} = (4 \times 10^0)$

Números binários

- Números descritos na base 2
- Dois símbolos: 0 e 1

1011_2

$1 \times 2^0 = 1$

$1 \times 2^1 = 2$

$0 \times 2^2 = 0$

$1 \times 2^3 = 8$

SOMA

11_{10}

Números binários

- Números descritos na base 2
- Dois símbolos: 0 e 1

0101_2

$1 \times 2^0 = 1$

$0 \times 2^1 = 0$

$1 \times 2^2 = 4$

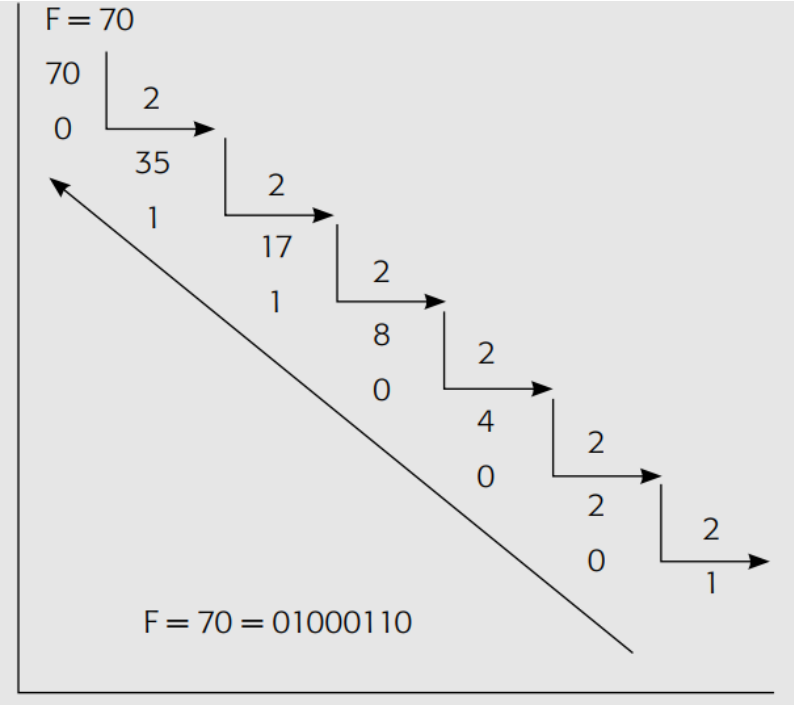
$0 \times 2^3 = 0$

SOMA

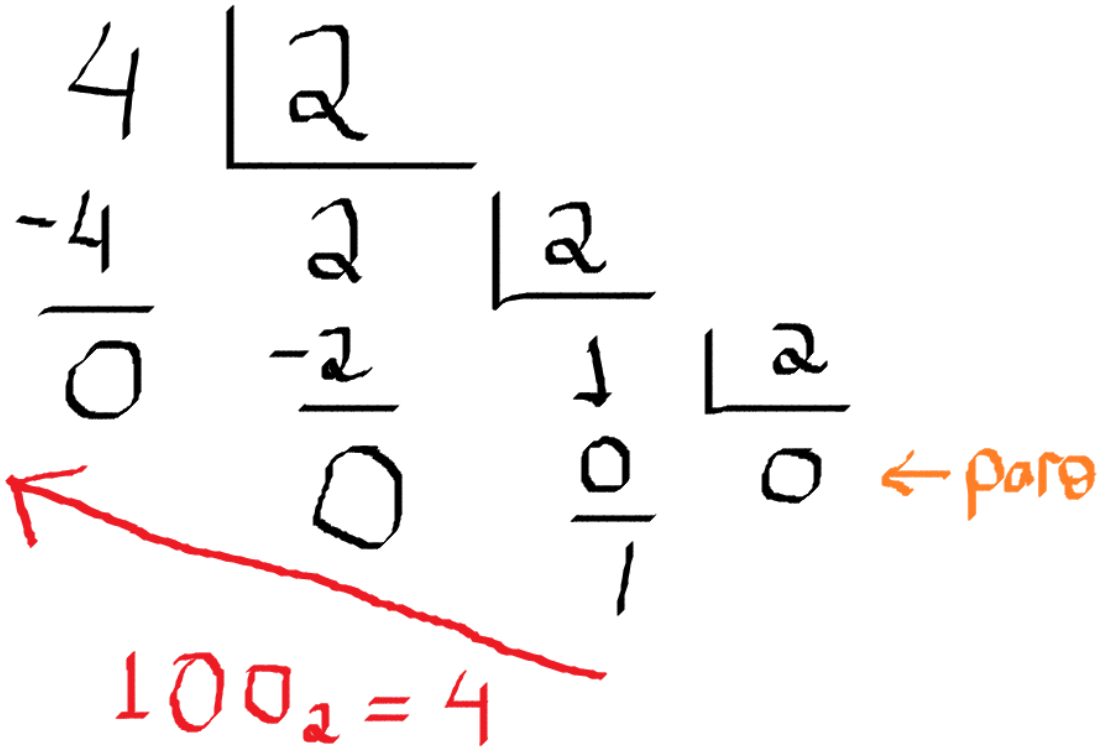
5_{10}

Números Binários

- Como converter de decimal para binário?
- Divide-se o número até obter o quociente igual a 0
- O número binário é igual aos restos obtidos (em ordem contrária)

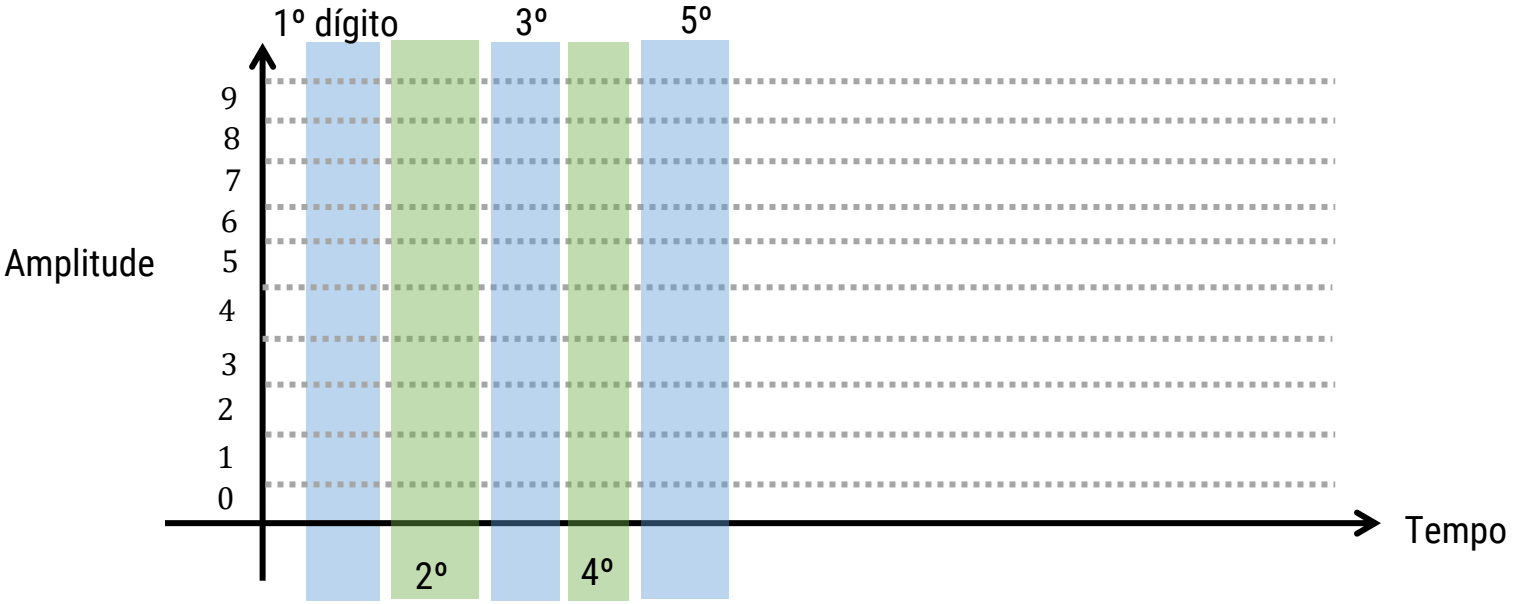


Números Binários



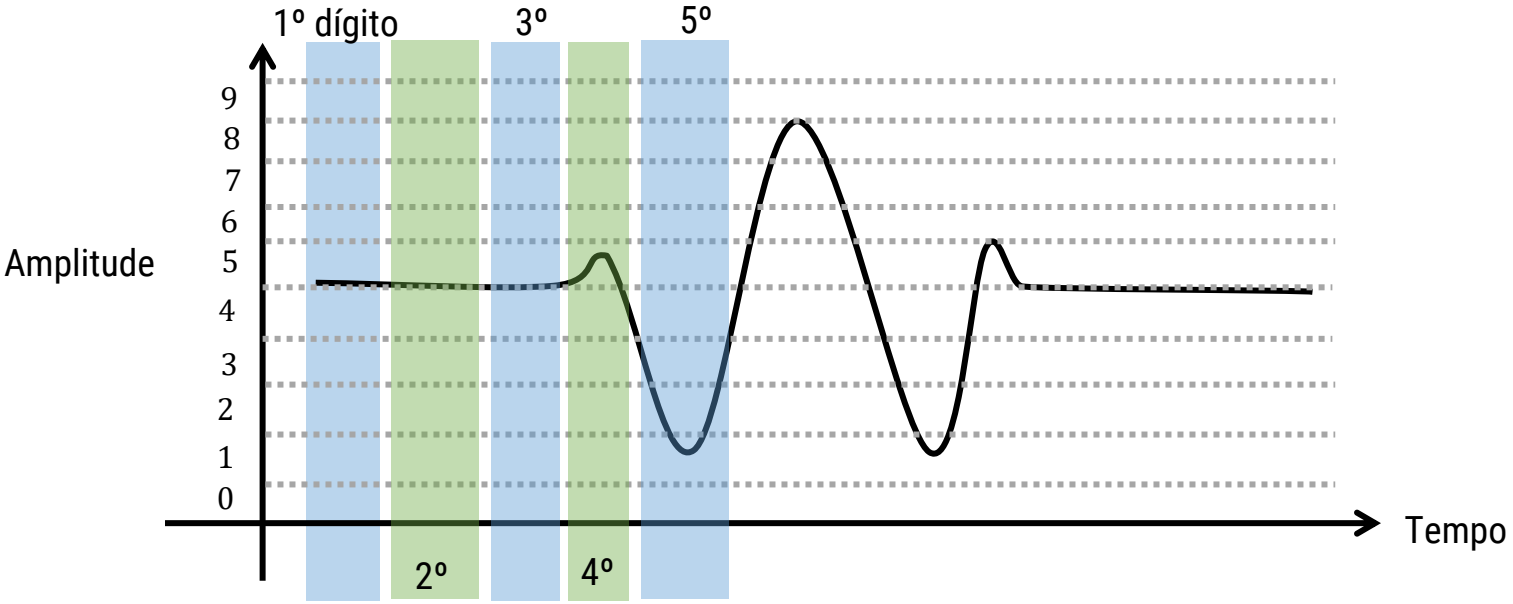
Por que binário?

- Computadores usam circuitos elétricos.
- Recebem pulsos de energia.



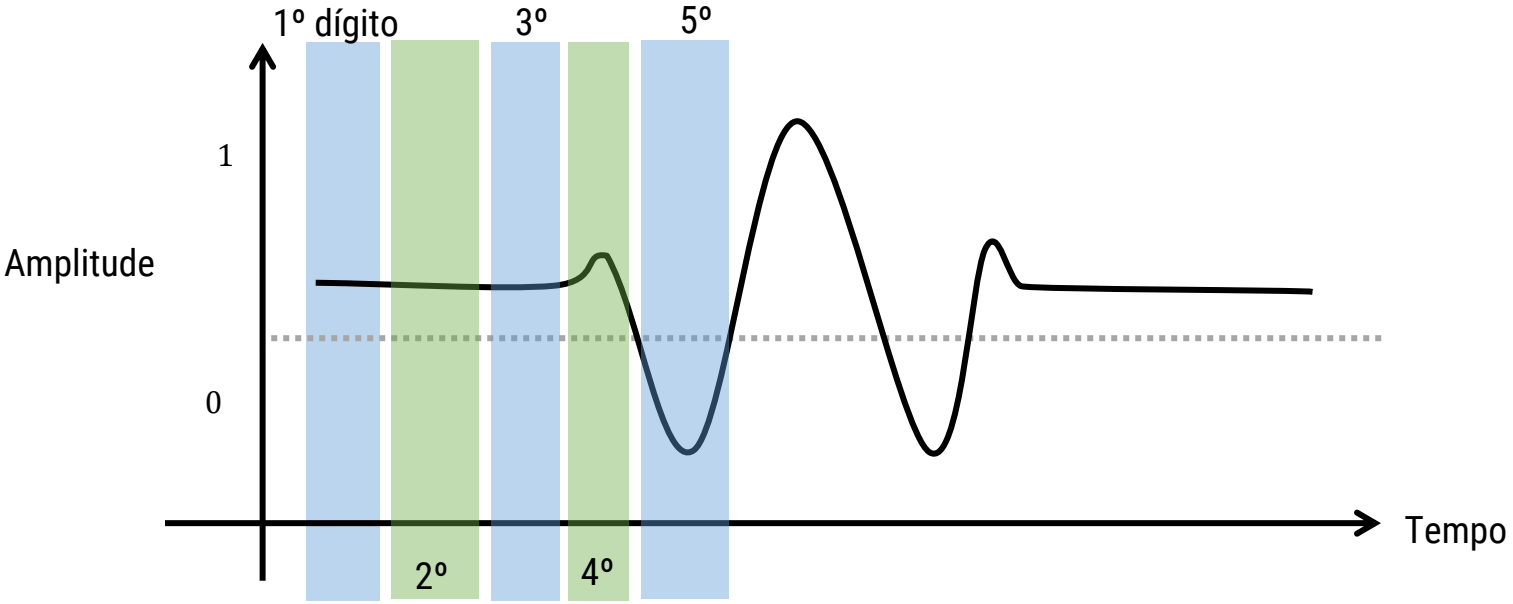
Por que binário?

- Computadores usam circuitos elétricos.
- Recebem pulsos de energia.



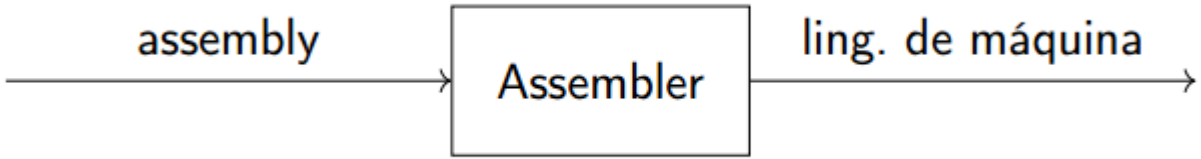
Por que binário?

- Computadores usam circuitos elétricos.
- Recebem pulsos de energia.



Linguagem assembly e de máquina

- Linguagem de máquina
  - Sequência de números (0101...) que representa o programa.
- **Linguagem de montagem (*assembly*)**
  - Representação simbólica e textual com as instruções do computador
- Assembler (montador)
  - Traduz um programa em linguagem de montagem para código de máquina



Computador simples vs moderno

- Computador moderno
  - Codificação binária (base 2)
  - Uma célula de memória armazena um byte (=8 bits, 1 bit=1 ou 0)
  - Possuem instruções para manipular números inteiros e números em ponto-flutuante (aproximação de número real)
  - Possuem mais registradores R (memória cache)
- Apesar das diferenças ...
  - O assembly mostrado não está distante do assembly de computares reais
  - Em tese, é possível tudo o que um computador real faz

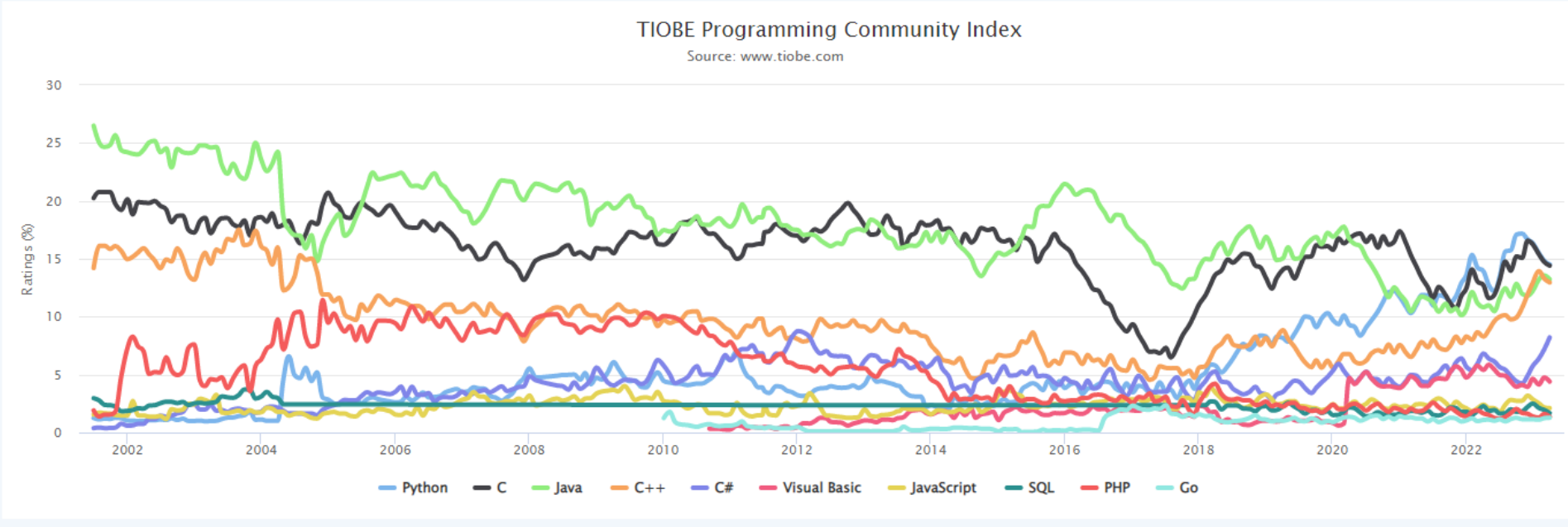
Assembly

- Programar em assembly é uma tarefa entediante e propensa a erros
- Um programa em assembly trabalha no nível de abstração de uma máquina em específica
- Queremos descrever processos algorítmicos num nível de abstração mais alto, e sem nos preocuparmos com detalhes de implementação de uma máquina específica, *e.g.*, $z = x^2 + 3y + 5$

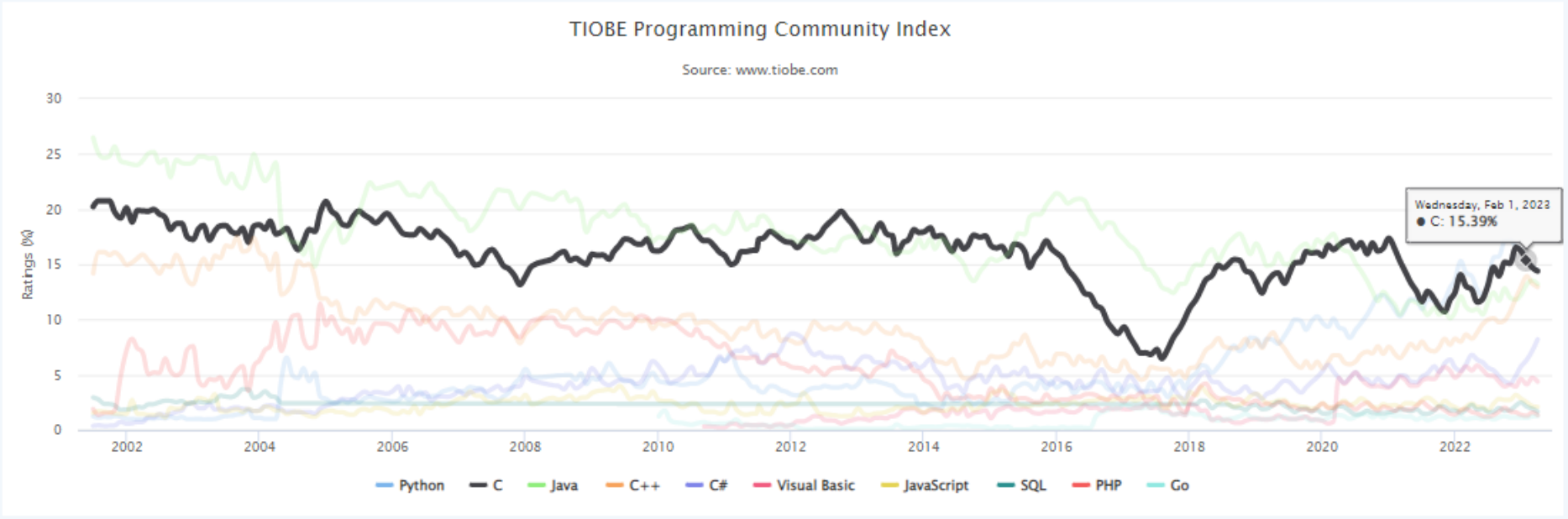
Linguagens de Alto Nível

- A partir dos anos 1950, linguagens de programação de alto nível foram criadas
- Nível de abstração mais alto e de forma independente de uma máquina particular
  - Fortran (1957)
  - COBOL (1960)
  - Pascal (1970)
  - **C (1972)**
  - C++ (1983)
  - Python (1991)
  - Lua (1993)
  - R (1993)

Linguagens de Alto Nível



Linguagens de Alto Nível



Linguagem C

- Linguagem de propósito geral
- Base para muitas outras linguagens (R, Python)
- Programação de diferentes tipos de sistemas e hardware.
- Programar para Windows/Linux/MAC
- Programar para Arduino

Exemplo Programa C

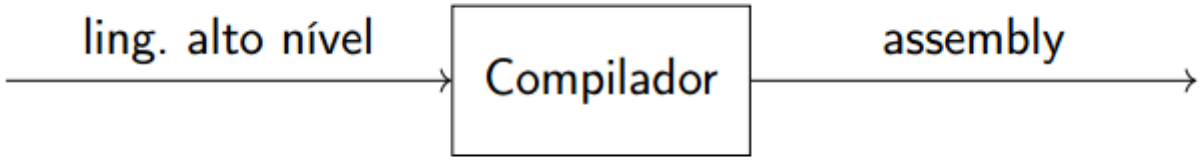
- Programa de conversão Fahrenheit para Celsius.

```c
#include <stdio.h>

int main () {
    float fahrenheit;
    float celsius;

    scanf("%f", &fahrenheit);                  // read
    celsius = (fahrenheit - 32) / 1.8;         // compute
    printf("Temp. Celsius: %f", celsius); // write
}
```

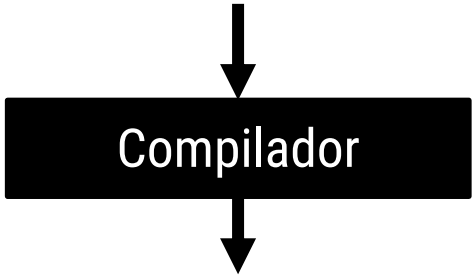
```
1.  storeconst 32 0
2.  storeconst 1.8 1
3.  read 2
4.  sub 2 0
5.  store 3
6.  div 3 1
7.  store 3
8.  write 3
```

Compilação

- Programas em linguagens de alto nível não são executados diretamente pela máquina
- Antes de executá-los é preciso traduzi-los para programas equivalentes em assembly ou linguagem de máquina (diretamente)
- Essa tradução (compilação) é realizada por um outro programa chamado compilador ***Compilador***



```c
#include <stdio.h>

int main () {
    float fahrenheit;
    float celsius;

    scanf("%f", &fahrenheit);              // read
    celsius = (fahrenheit - 32) / 1.8;     // compute
    printf("Temp. Celsius: %f", celsius); // write
}
```




```
1.  storeconst 32 0
2.  storeconst 1.8 1
3.  read 2
4.  sub 2 0
5.  store 3
6.  div 3 1
7.  store 3
8.  write 3
```

```
1.   storeconst 32 0
2.   storeconst 1.8 1
3.   read 2
4.   sub 2 0
5.   store 3
6.   div 3 1
7.   store 3
8.   write 3
```

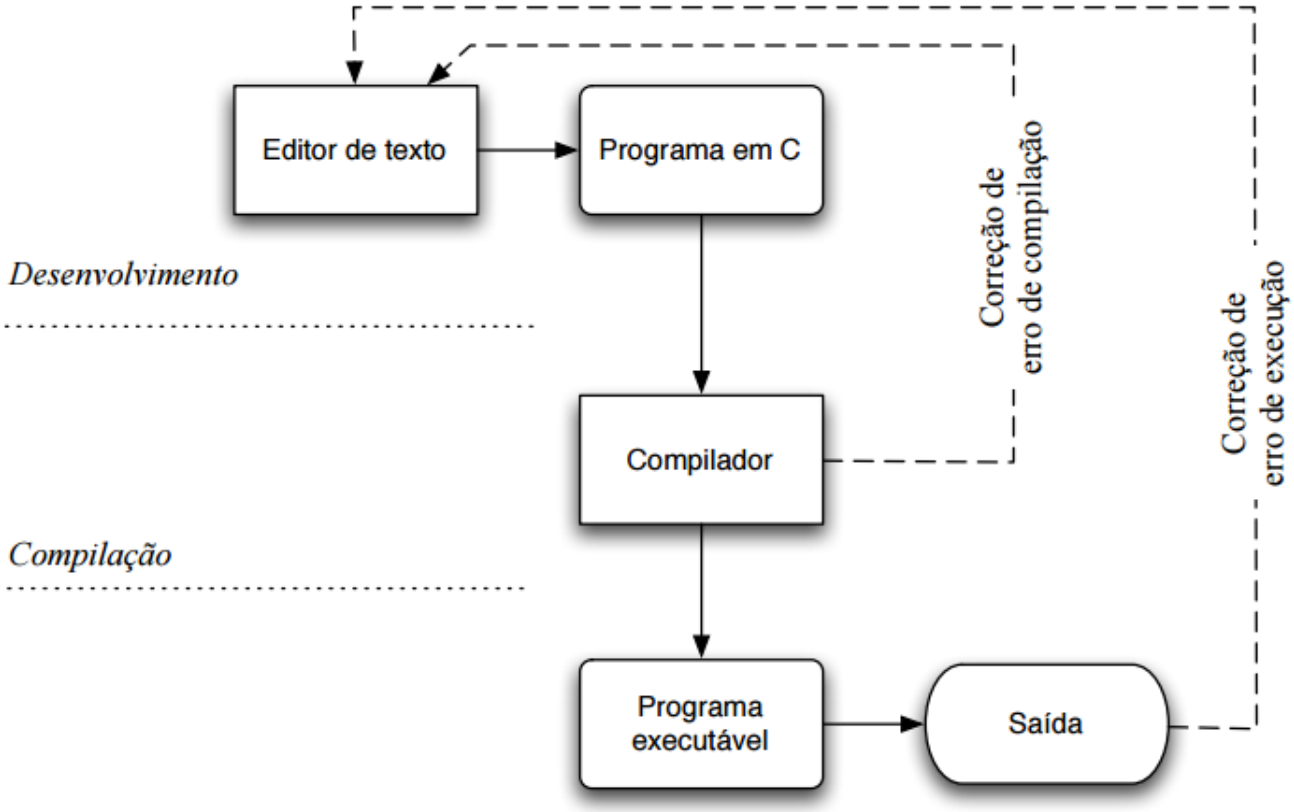
Assembler

```
01010101001010100101011111010010101010010101010101
01010100101010101010101001010101010100101011111110101
01001010101010101010100101010101010100101010010101
11110010101001001010101010010101010100101010110100
10101010111100000110110010100110110001
```

Compiladores

- A maioria dos compiladores modernos já incluem a etapa de geração de código de máquina internamente
- Além disso, tipicamente tentam otimizar o código gerado

Ciclo de Desenvolvimento



Próxima Aula

- Variáveis
- Primitivas básicas

Obrigado!

Perguntas?

marcosroriz@ufg.br

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

Fonte dos ícones: flaticons.com