

---

# **Chapter 1: Computer Abstractions and Technology**

**1.1: Introduction**

**1.2: Great ideas**

**1.3: Below programs**

**Arquitetura e Organização  
de Computadores  
UVV**

---

# A Revolução da Computação

---

- **Agrícola – Industrial – Computação/Informação**



# A Revolução da Computação

- Agrícola – Industrial – Computação/Informação



# A Revolução da Computação

---

- **Agrícola – Industrial – Computação/Informação**



# A Revolução da Computação

---

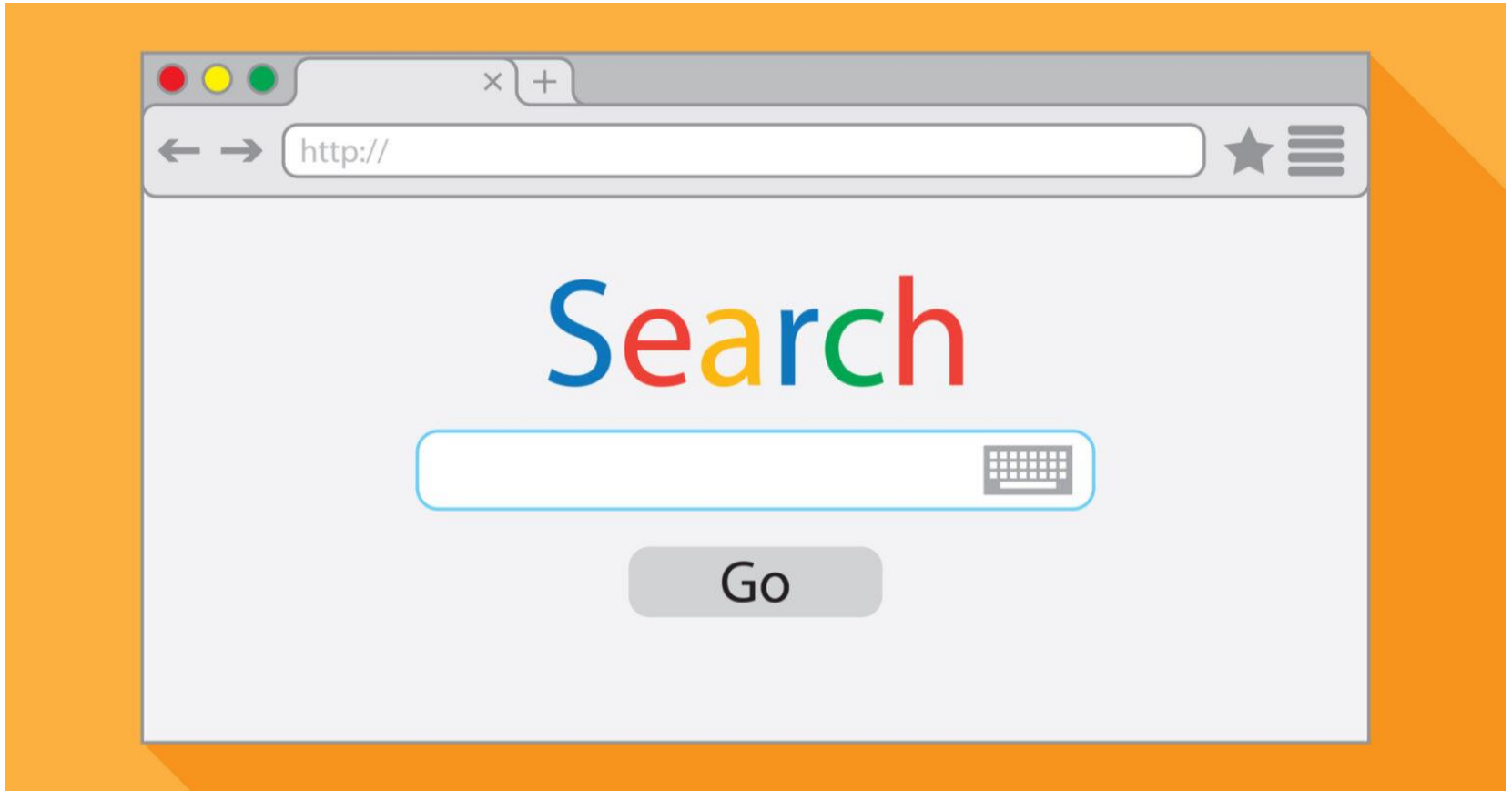
- **Agrícola – Industrial – Computação/Informação**



# A Revolução da Computação

---

- **Agrícola – Industrial – Computação/Informação**



# A Revolução da Computação

---



- **Futuro?**

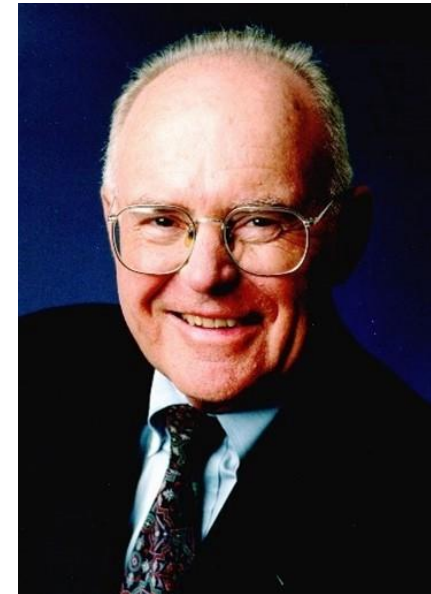
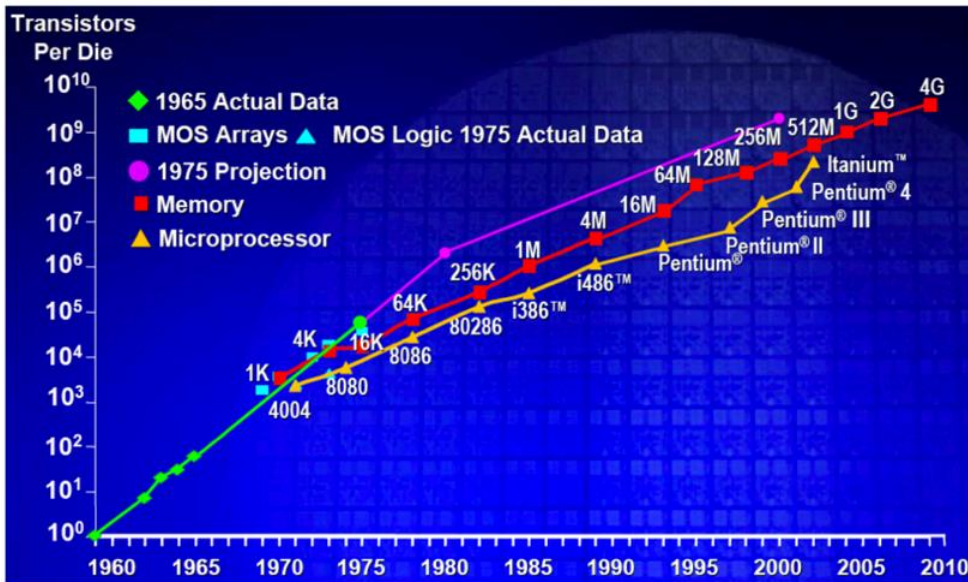
“O **gerador de improbabilidade infinita** é uma nova e maravilhosa invenção que possibilita atravessar imensas distâncias interestelares num simples zerézimo de segundo, sem toda aquela complicação e chatice de ter que passar pelo hiperespaço. (...)

O princípio de gerar pequenas quantidades de improbabilidade finita simplesmente ligando os circuitos lógicos de um Cérebro Subméson Bambleweeny 57 a uma impressora de vetor atômico suspensa num produtor de movimentos brownianos intensos (por exemplo, uma boa xícara de chá quente) já era, naturalmente, bem conhecido – e tais geradores eram freqüentemente usados para quebrar o gelo em festas, fazendo com que todas as moléculas da calcinha da anfitriã se deslocassem 30 centímetros para a direita, de acordo com a Teoria da Indeterminação.

Muitos físicos respeitáveis afirmavam que não admitiam esse tipo de coisa – em parte porque era uma avacalhação da ciência, mas principalmente porque eles não eram convidados para essas festas.”

# A Revolução da Computação

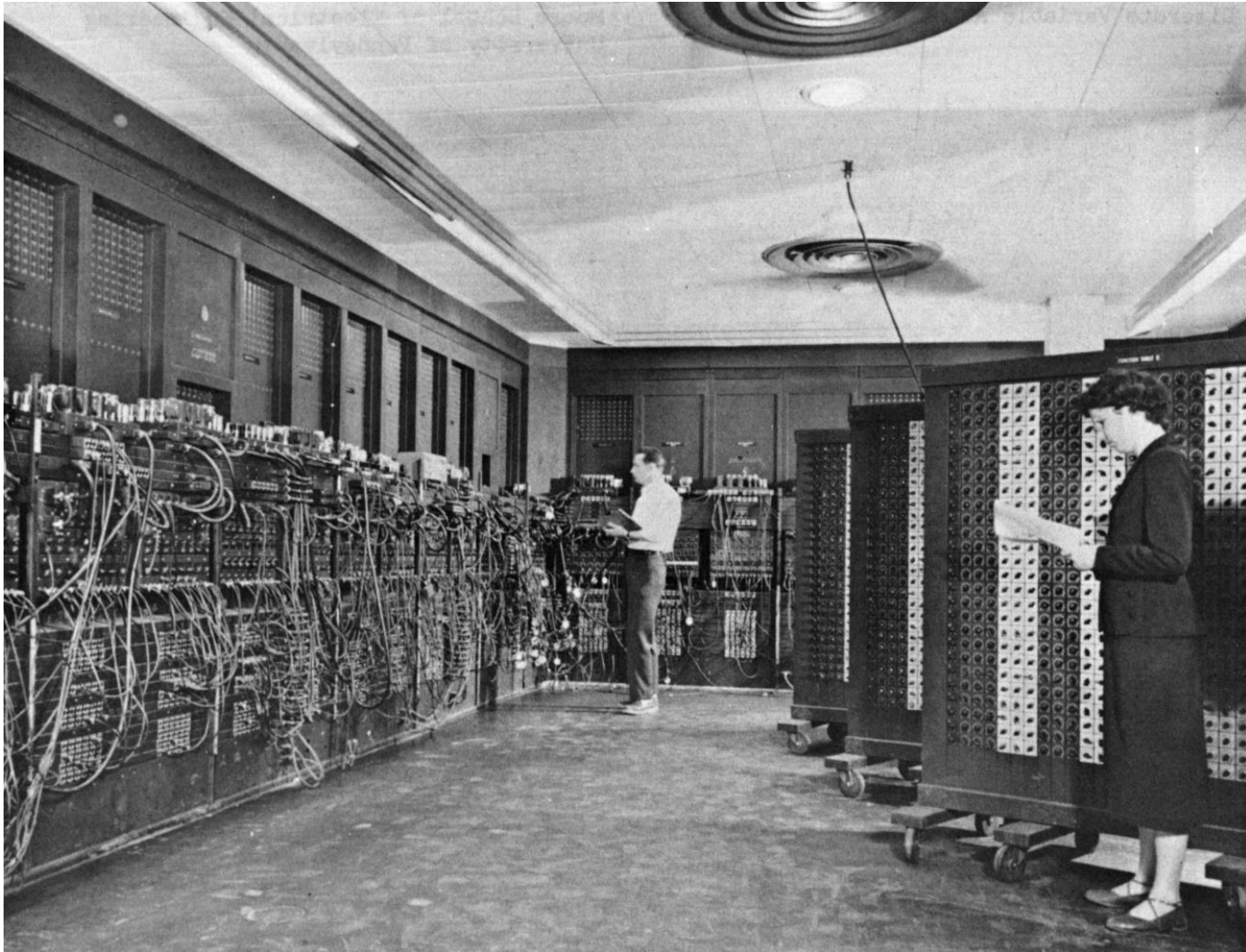
- Lei de Moore (Gordon Moore)
  - A cada 2 anos a densidade dos circuitos dobra (frequência, performance), mantendo o custo.





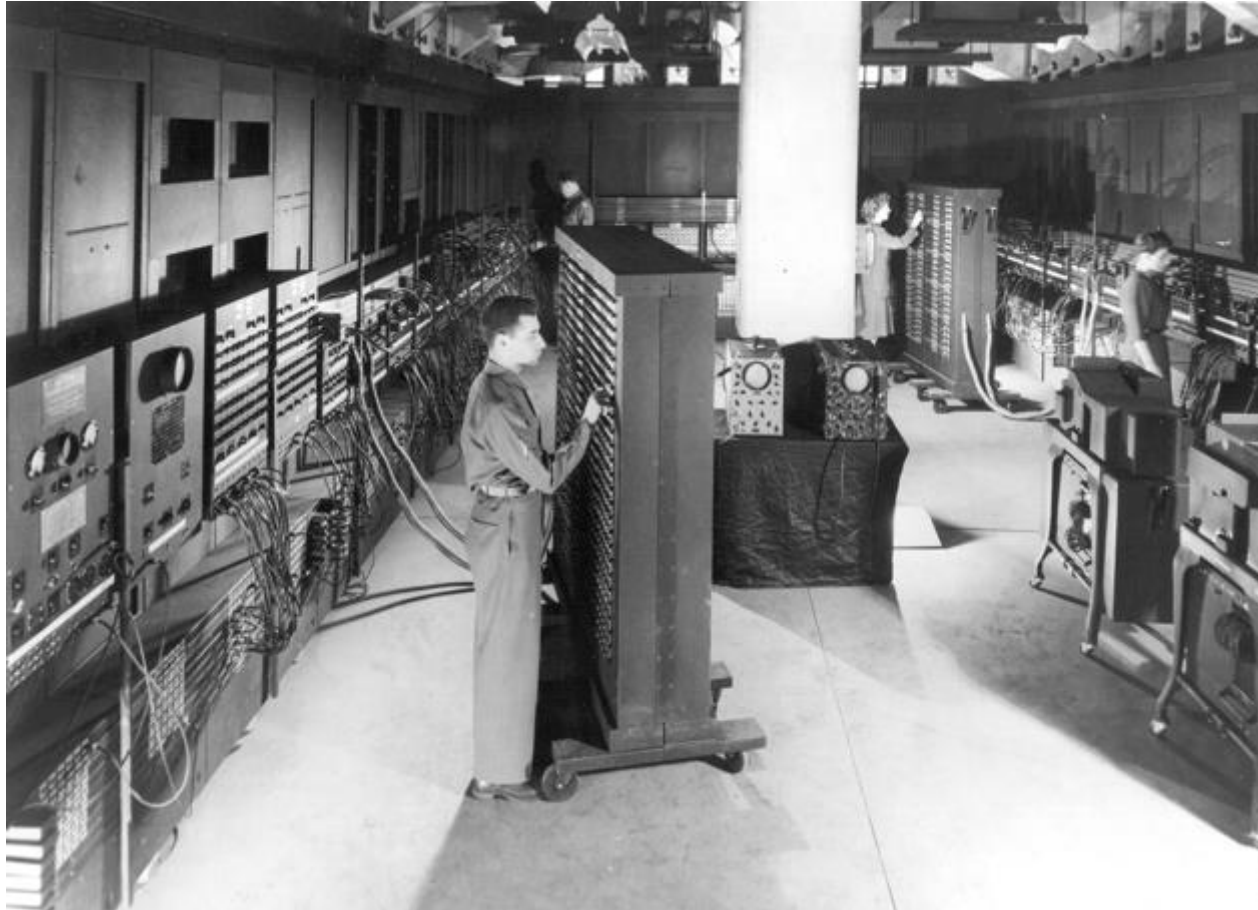
# A Revolução da Computação

---

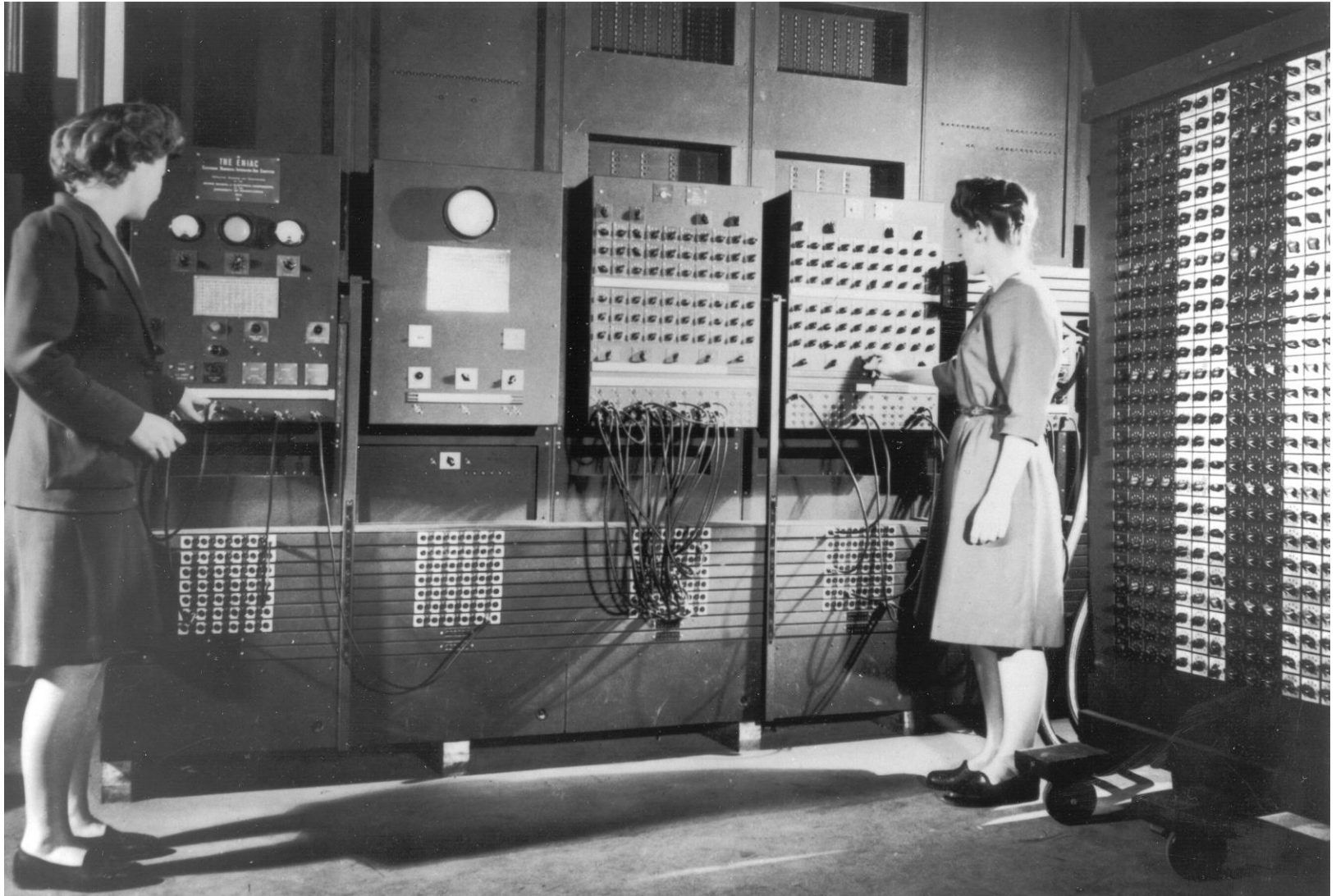


# A Revolução da Computação

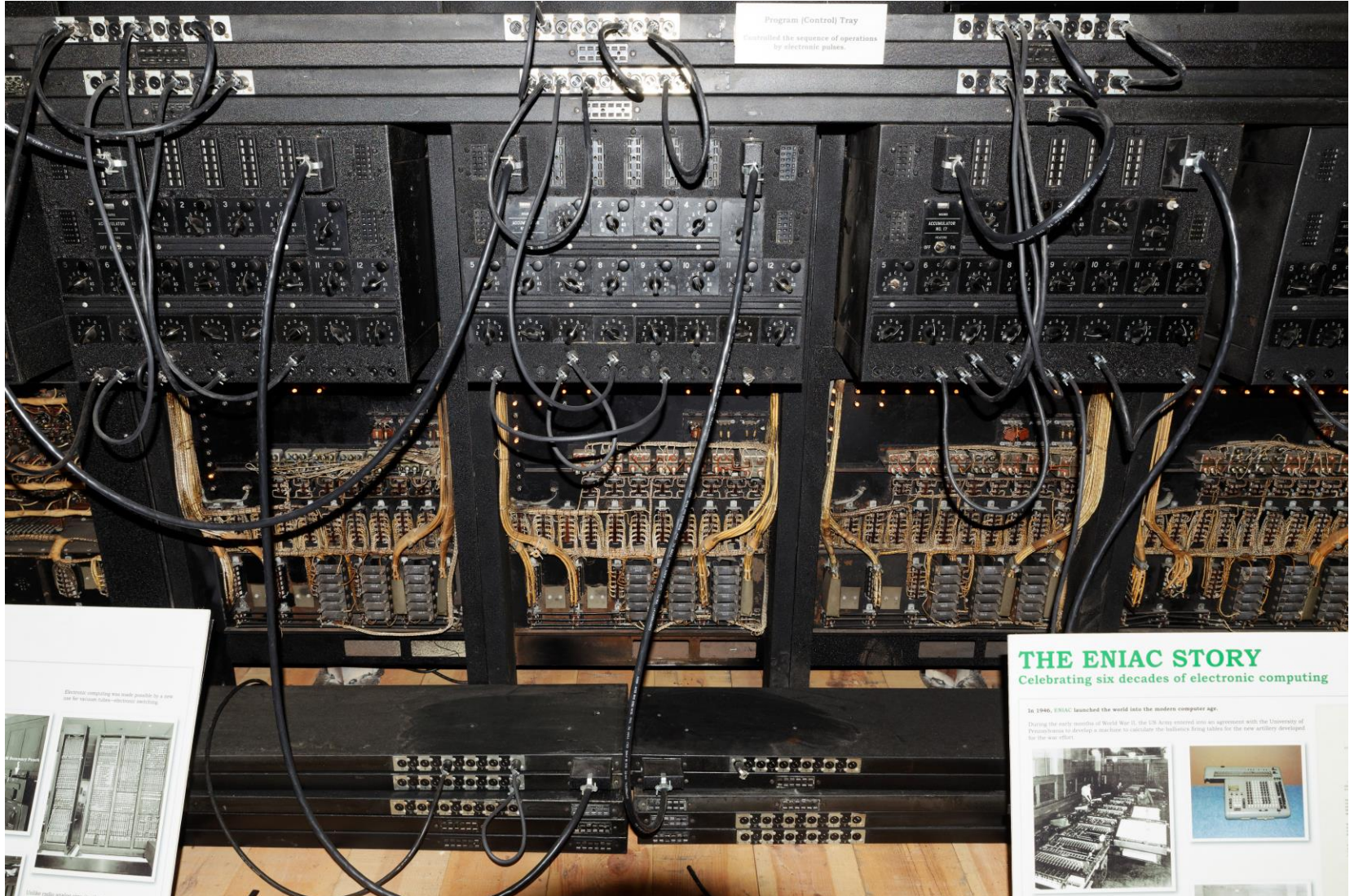
---



# A Revolução da Computação

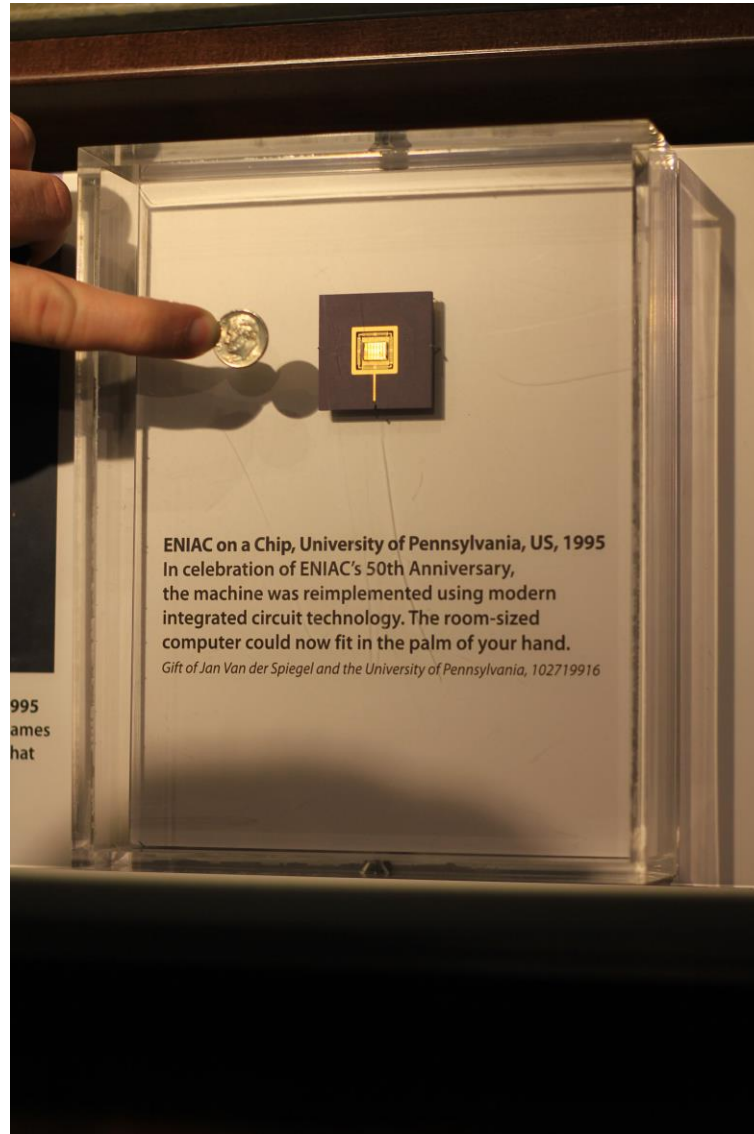


# A Revolução da Computação

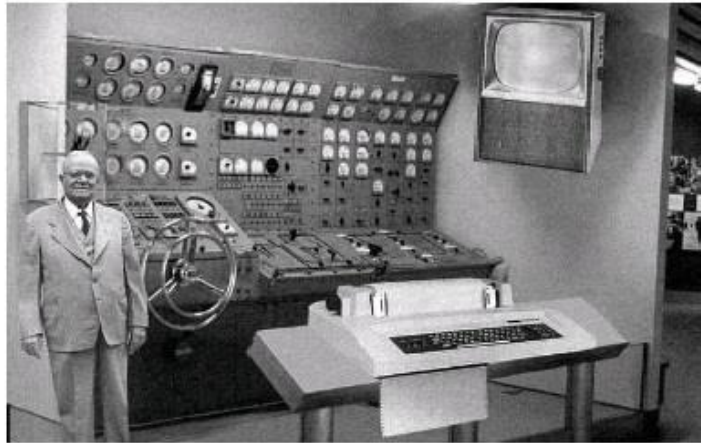


# A Revolução da Computação

---



# A Revolução da Computação



First Generation



Second Generation



Third Generation



Fourth Generation



Fifth Generation

# “Classes” de Computadores

---

- Computadores Pessoais (PC)
  - Uso geral, variedade de software
  - Custo/performance
- Servidores
  - Acesso em rede
  - Alta capacidade
  - Alta performance
  - Alta confiabilidade
  - Diversos tamanhos e aplicações



# “Classes” de Computadores

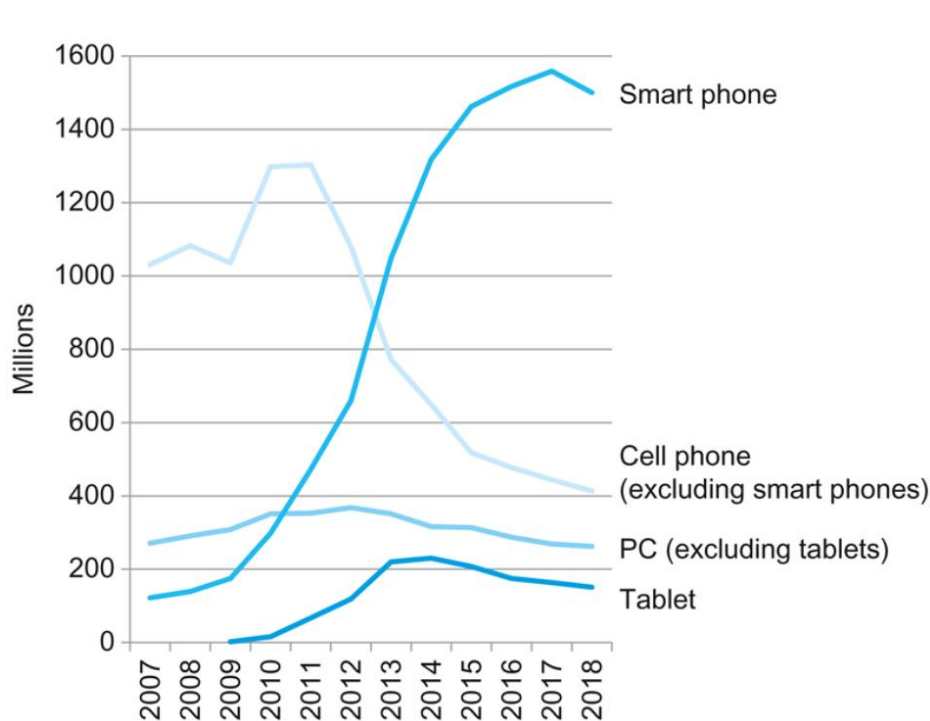
- Supercomputadores
  - Aplicações científicas e de engenharia de alta complexidade
- Embarcados
  - “Escondidos” dentro do sistema/dispositivo
  - Grandes limites de energia, capacidade, performance
  - Faça uma coisa bem feita
  - Confiabilidade
  - Alta tolerância à falhas





# “Classes” de Computadores

- Era pós-PC:
  - Dispositivo Móvel Pessoal (PMD)
    - Internet, bateria, barato
  - Computação em Nuvem
    - IaaS, PaaS, SaaS



# Você aprenderá:

---

- Como os programas são **traduzidos** de linguagens de alto nível para linguagem de máquina
  - E como o hardware executa os programas
- A **interface** hardware/software
  - E como o software instrui o hardware para realizar as operações necessárias
- O que determina a **performance** de um programa
  - E como ela pode ser melhorada pelo programador
- Como os projetistas de hardware melhoram a **performance**
- Como os projetistas de hardware melhoram e **eficiência energética**
- Razões e vantagens do **processamento paralelo**
- **Grandes idéias** que formaram os fundamentos da computação moderna

**Tudo isso que o tornará um profissional melhor e mais capacitado.**

# O que é “performance”?

- **Performance:**
  - **Hardware: frequência**
  - **Aplicação: experiência do usuário**
- Parece complicado, mas é “formada” por poucos ingredientes:

Hardware or software component	How this component affects performance	Where is this topic covered?
Algorithm	Determines both the number of source-level statements and the number of I/O operations executed	Other books!
Programming language, compiler, and architecture	Determines the number of computer instructions for each source-level statement	Chapters 2 and 3
Processor and memory system	Determines how fast instructions can be executed	Chapters 4, 5, and 6
I/O system (hardware and operating system)	Determines how fast I/O operations may be executed	Chapters 4, 5, and 6

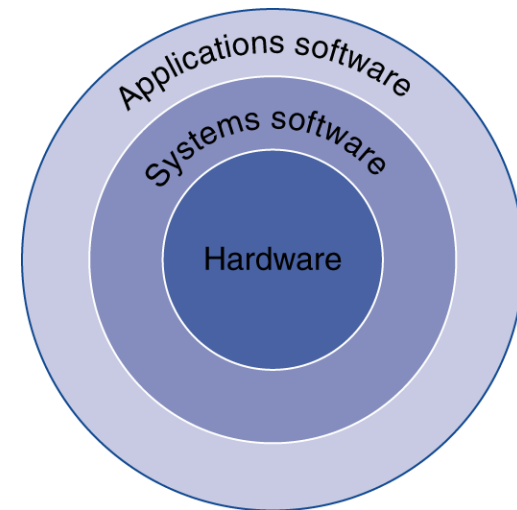
# As 7 Grandes Idéias

- Usar **abstrações** para simplificar o design
- Performance do **caso comum**
- Performance por **paralelismo**
- Performance por **pipelining**
- Performance por **predição**
- **Hierarquia** de memórias
- **Confiança** por redundância



# Abaixo de Seu Programa

- Software aplicativo
  - Linguagem de alto nível
- Software de sistema
  - Compilador: traduz alto nível para linguagem de máquina
  - Sistema Operacional: serviços
    - Entrada/saída
    - Gerenciamento de memória e armazenamento
    - Agendamento de tarefas e compartilhamento de recursos
- Hardware
  - Processador, memória, dispositivos de I/O, etc...



# Alto Nível para Linguagem de Máquina

---

1001010100101110

# Alto Nível para Linguagem de Máquina

---

1001010100101110

- **INSTRUÇÃO**: um comando que o hardware do computador entende e obedece (realiza alguma coisa)
  - São apenas coleções de bits
- Ponto chave: usamos números (binários) para duas coisas diferentes:
  - INSTRUÇÕES
  - DADOS
- Os números acima são uma INSTRUÇÃO que comanda o processador a somar 2 números.

# Alto Nível para Linguagem de Máquina

---

- Codificar em binário?

1001010100101110





# Alto Nível para Linguagem de Máquina

---

- **Assembler:**

- Programa que traduz uma versão simbólica das instruções em sua versão binária (linguagem de máquina)

add A, B

- **Linguagem Assembly:**

- A linguagem formada pelas representações simbólicas das instruções

1001010100101110

- **Linguagem de Máquina:**

- Uma representação binária das instruções

# Alto Nível para Linguagem de Máquina

- **Linguagem de alto nível:**
  - Compreensível
  - Produtividade
  - Portabilidade
- **Linguagem Assembly:**
  - Representação textual simbólica das instruções
  - Interface entre o HW e o SW
- **Linguagem de Máquina:**
  - Usa dígitos binários
  - Codifica INSTRUÇÕES e DADOS

High-level  
language  
program  
(in C)

```
swap(size_t v[], size_t k)  
{  
    size_t temp;  
    temp = v[k];  
    v[k] = v[k+1];  
    v[k+1] = temp;  
}
```

Compiler

Assembly  
language  
program  
(for RISC-V)

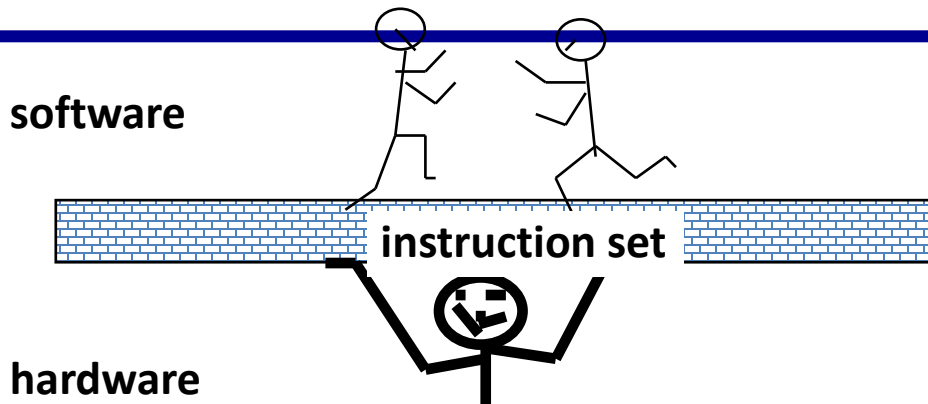
```
swap:  
    slli x6, x11, 3  
    add  x6, x10, x6  
    ld   x5, 0(x6)  
    ld   x7, 8(x6)  
    sd   x7, 0(x6)  
    sd   x5, 8(x6)  
    jalr x0, 0(x1)
```

Assembler

Binary machine  
language  
program  
(for RISC-V)

```
00000000001101011001001100010011  
00000000011001010000001100110011  
00000000000000110011001010000011  
00000000100000110011001110000011  
00000000011100110011000000100011  
00000000010100110011010000100011  
0000000000000000100000001100111
```

# Alto Nível para Linguagem de Máquina



- As **INSTRUÇÕES** formam as palavras da linguagem do computador
- O **DICIONÁRIO** das instruções forma o conjunto de instruções (instruction set – IS)
  - É a interface de mais baixo nível de software, para usuários ou programadores de compiladores
- **ARQUITETURA DO CONJUNTO DE INSTRUÇÕES (ISA):**
  - Um tipo de computador

High-level language program (in C)

```
swap(size_t v[], size_t k)
{
    size_t temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

Compiler

Assembly language program (for RISC-V)

```
swap:
    slli x6, x11, 3
    add x6, x10, x6
    ld x5, 0(x6)
    ld x7, 8(x6)
    sd x7, 0(x6)
    sd x5, 8(x6)
    jalr x0, 0(x1)
```

Assembler

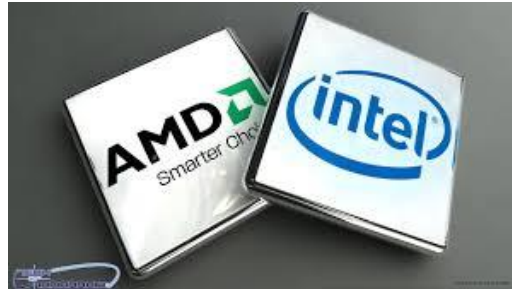
Binary machine language program (for RISC-V)

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
0000000011100110011000000100011
0000000010100110011010000100011
000000000000000100000001100111
```

# Principais tipos de Arquitetura de Conjunto de Instruções

---

- X86: Intel, AMD, Desktop, laptop, servidores



- ARM: embarcados, telefones, calculadores, etc.



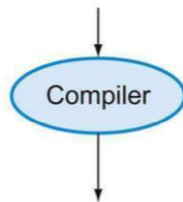
- Power (IBM) e SPARC (Oracle e Fujitsu): servidores
- **RISC-V: aberto, em crescimento, embarcado**



# De Programas para Arquitetura

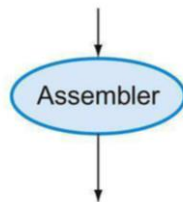
High-level language program (in C)

```
swap(size_t v[], size_t k)
{
    size_t temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```



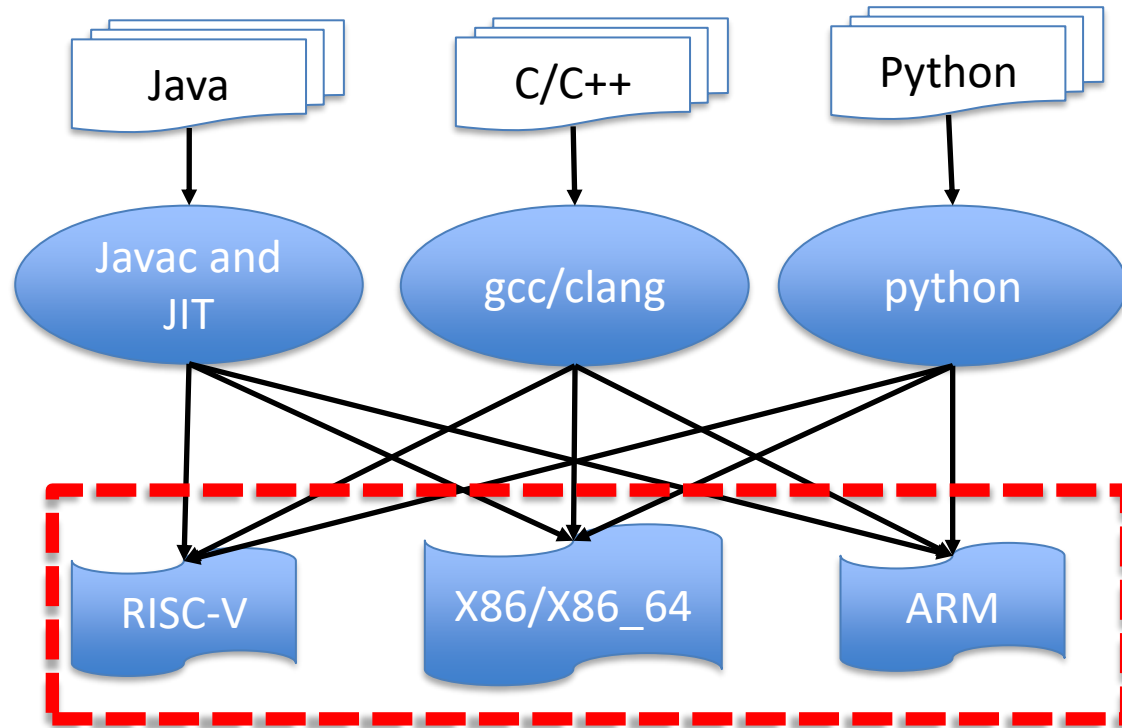
Assembly language program (for RISC-V)

```
swap:
    slli x6, x11, 3
    add x6, x10, x6
    ld x5, 0(x6)
    ld x7, 8(x6)
    sd x7, 0(x6)
    sd x5, 8(x6)
    jalr x0, 0(x1)
```



Binary machine language program (for RISC-V)

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000010100110011010000100011
0000000000000000100000001100111
```



# Exemplo de Assembly para X86\_64

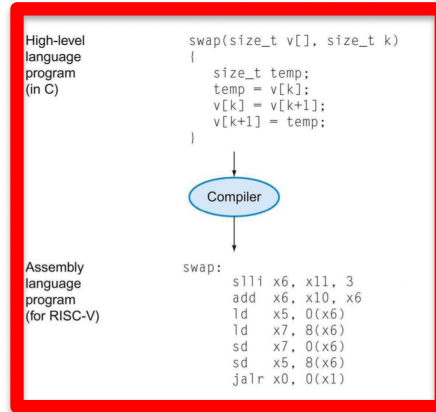
Usar a opção “-S” para traduzir alto nível para assembly

```
yanyh@vm:~$ uname -a
Linux vm 4.4.0-170-generic #199-Ubuntu SMP Thu Nov 14 01:45:04 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
yanyh@vm:~$ gcc -S swap.c
yanyh@vm:~$ cat swap.s
```

```
.file "swap.c"
.text
.globl swap
.type swap, @function
```

```
swap:
.LFB0:
```

```
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
movq %rdi, -24(%rbp)
movl %esi, -28(%rbp)
movl -28(%rbp), %eax
cltq
leaq 0(,%rax,4), %rdx
movq -24(%rbp), %rax
addq %rdx, %rax
movl (%rax), %eax
movl %eax, -4(%rbp)
movl -28(%rbp), %eax
cltq
leaq 0(,%rax,4), %rdx
movq -24(%rbp), %rax
addq %rax, %rdx
movl -28(%rbp), %eax
cltq
addq $1, %rax
leaq 0(,%rax,4), %rcx
movq -24(%rbp), %rax
```



```
Binary machine language program (for RISC-V)
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
000000001110011001100000110011
0000000010100110011010000100011
00000000000000001100000001100111
```

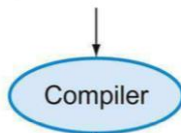
- X86\_64 é a ISA para a maioria das CPUs Intel e AMD
- RISC-V é uma ISA
- ARM é outra ISA
  - A maioria dos celulares são ARM

# Exemplo de Assembly X86\_64

## Reverter o código binário para o assembly

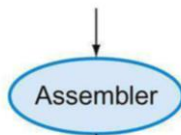
High-level language program (in C)

```
swap(size_t v[], size_t k)
{
    size_t temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```



Assembly language program (for RISC-V)

```
swap:
    slli x6, x11, 3
    add x6, x10, x6
    ld x5, 0(x6)
    ld x7, 8(x6)
    sd x7, 0(x6)
    sd x5, 8(x6)
    jalr x0, 0(x1)
```



Binary machine language program (for RISC-V)

```
000000000011010110010011000010011
00000000011001010000001100110011
00000000000000110011001010000011
00000000100000110011001110000011
00000000011100110011000000100011
00000000010100110011010000100011
0000000000000000100000001100111
```

Disassembly

```
yanyh@vm:~$ gcc -c swap.c
yanyh@vm:~$ objdump -D swap.o
```

swap.o: file format **elf64-x86-64**

Disassembly of section .text:

```
0000000000000000 <swap>:
 0: 55                push %rbp
 1: 48 89 e5          mov %rsp,%rbp
 4: 48 89 7d e8       mov %rdi,-0x18(%rbp)
 8: 89 75 e4          mov %esi,-0x1c(%rbp)
 b: 8b 45 e4          mov -0x1c(%rbp),%eax
 e: 48 98            cltq
10: 48 8d 14 85 00 00 00 lea 0x0(,%rax,4),%rdx
17: 00
18: 48 8b 45 e8       mov -0x18(%rbp),%rax
1c: 48 01 d0          add %rdx,%rax
1f: 8b 00            mov (%rax),%eax
21: 89 45 fc          mov %eax,-0x4(%rbp)
24: 8b 45 e4          mov -0x1c(%rbp),%eax
27: 48 98            cltq
29: 48 8d 14 85 00 00 00 lea 0x0(,%rax,4),%rdx
30: 00
31: 48 8b 45 e8       mov -0x18(%rbp),%rax
35: 48 01 c2          add %rax,%rdx
38: 8b 45 e4          mov -0x1c(%rbp),%eax
3b: 48 98            cltq
3d: 48 83 c0 01       add $0x1,%rax
41: 48 8d 0c 85 00 00 00 lea 0x0(,%rax,4),%rcx
48: 00
49: 48 8b 45 e8       mov -0x18(%rbp),%rax
4d: 48 01 c8          add %rcx,%rax
50: 8b 00            mov (%rax),%eax
```

# Exercício: Olá, Mundo!

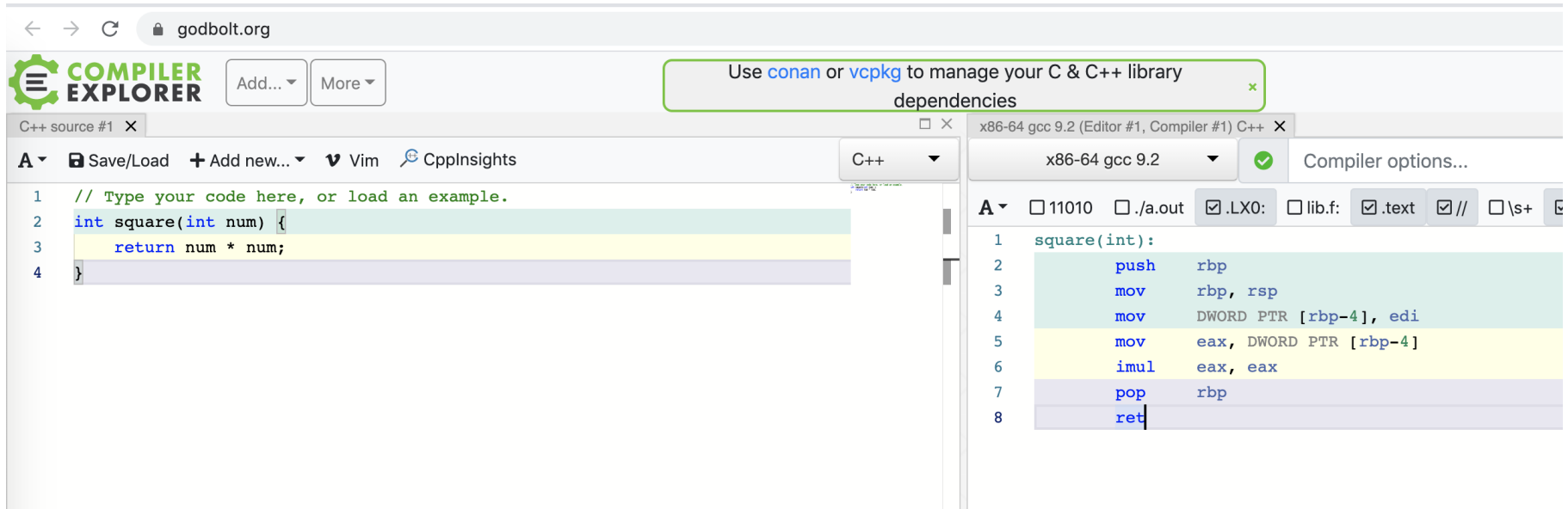
---

- Crie o “Olá, Mundo!”, em C
- Compile:
  - `gcc -s ola.c -o ola.x86_64.s`
  - `gcc -c ola.c`
  - `objdump -D ola.o > ola.x86_64_objdump.txt`
- Verifique:
  - `ola.x86_64.s`,
  - `ola.x86_64_objdump.txt`



# Exercício: explore outros ISA

- Explore outros ISA usando o Compiler Explorer:  
<https://godbolt.org/>



The screenshot displays the Compiler Explorer interface. On the left, the C++ source code is shown in a text editor:

```
1 // Type your code here, or load an example.
2 int square(int num) {
3     return num * num;
4 }
```

On the right, the assembly output for the x86-64 gcc 9.2 compiler is displayed:

```
1 square(int):
2     push    rbp
3     mov     rbp, rsp
4     mov     DWORD PTR [rbp-4], edi
5     mov     eax, DWORD PTR [rbp-4]
6     imul   eax, eax
7     pop     rbp
8     ret
```

The interface also includes a navigation bar with 'COMPILER EXPLORER' and 'Add...'/'More' buttons, a notification box for library dependencies, and a toolbar with 'Save/Load', 'Add new...', 'Vim', and 'Cpplnsights' options.

---

**Até a próxima!**