



MC404AE - Organização Básica de Computadores e Ling. Montagem

Apresentação da Disciplina

Prof. Allan M. de Souza

Agenda

— — — —

- Quem sou eu?
- Visão geral da disciplina
- Por que estudar assembly
- Conceitos básicos
- ISA
- Computadores
- Linguagem de programação
- Compilação
- Montagem
- Desmontagem

\$whoami

Swhoami

- **Allan Mariano de Souza**

- Professor @ IC Unicamp
- Pesquisador Associado H.IAAC & LRC
- <https://allanmsouza.github.io/>

- **Hobbie**

- Jogar MMORPGs



H.IAAC | HUB DE INTELIGÊNCIA
ARTIFICIAL E ARQUITETURAS
COGNITIVAS



- **Temas de Pesquisa**

- Inteligência Artificial
- Sistemas Distribuídos
- Redes de Computadores

Swwhoami - Allan M. de Souza

- **Bacharelado de 2010 a 2013**
 - Ciência da Computação @ UNIVEM

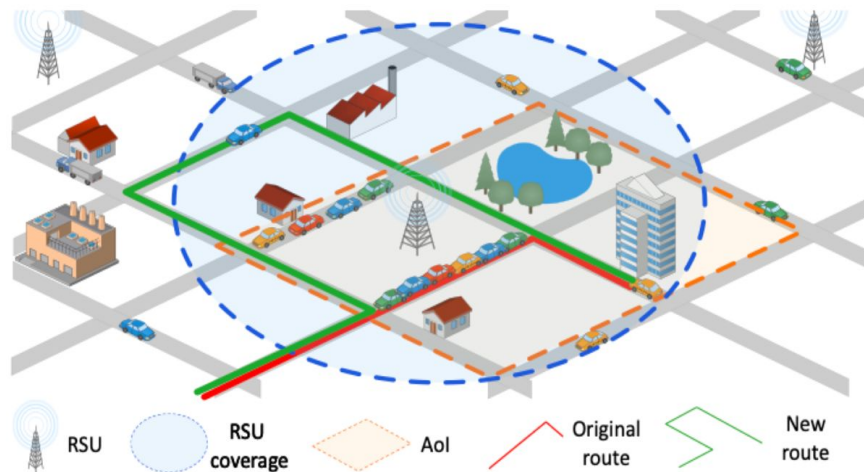
2010



Swwhoami - Allan M. de Souza

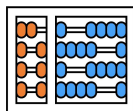
• Mestrado de 2014 a 2016

- Ciência da Computação @ Unicamp
- Leandro A. Villas
- Controle de congestionamentos



2010

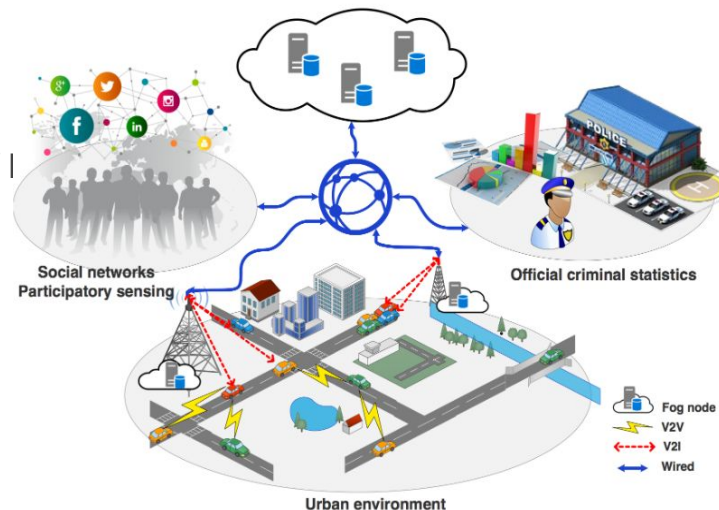
2014



Swwhoami - Allan M. de Souza

● Ph.D de 2017 a 2021

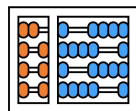
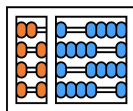
- Ciência da Computação @ Unicamp and Unil
- Leandro A. Villas & Torsten Braun
- Controle com múltiplos objetivos
- Predição de aspectos urbanos



2010

2014

2017

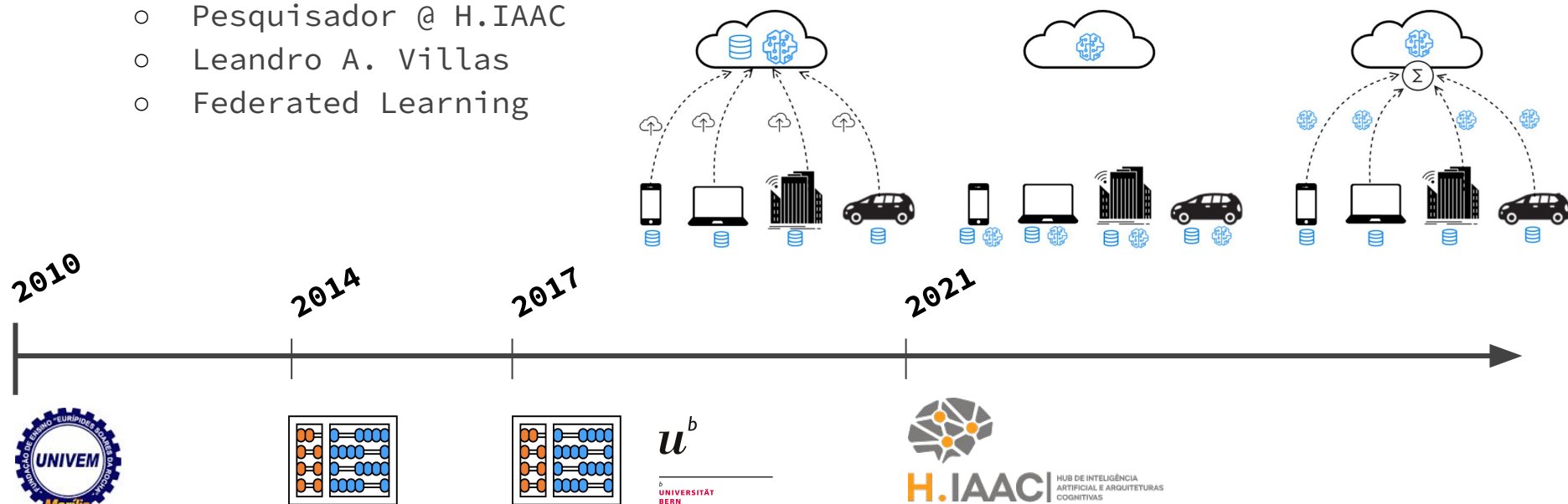

 u^b

 b
UNIVERSITÄT
BERN

Swwhoami - Allan M. de Souza

• Pós-doc de 2021 a 2023

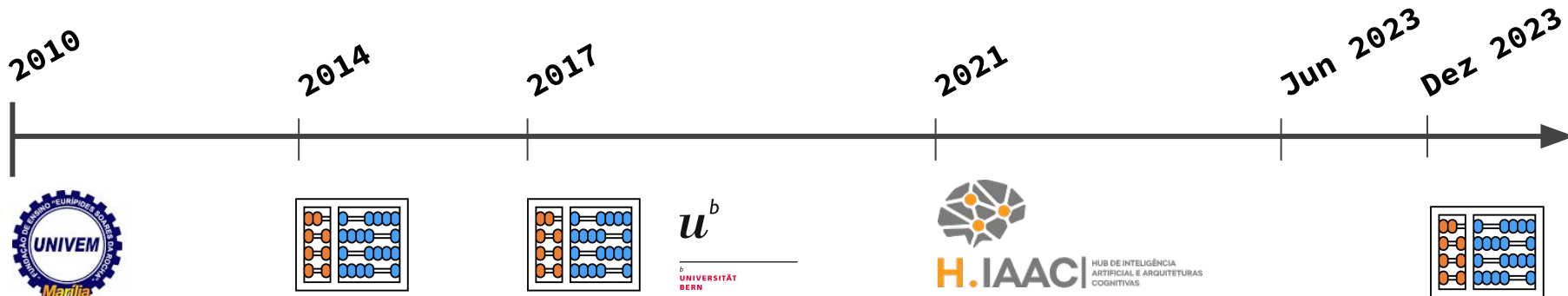
- Pesquisador @ H.IAAC
- Leandro A. Villas
- Federated Learning



Swwhoami - Allan M. de Souza

● De Junho 2023 a Atualmente

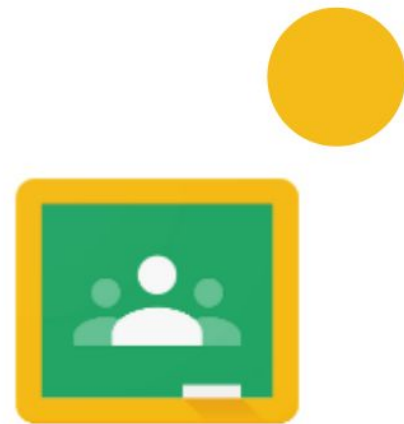
- 4 de Junho fui aprovado como Professor @ IC Unicamp
- Professor da disciplina de Redes de computadores - M0832A
- Professor responsável por esta disciplina - MC404



Visão Geral

Tópicos

- Organização básica de computadores
- Memória e endereçamento
- Representação de dados na memória
- Introdução Arquitetura de processadores
- Conjunto de instruções
- Programação em linguagem de montagem
- Instruções de I/O
- Interrupções
- Pilha, procedimentos e funções
- Passagem de parâmetros
- Montadores e ligadores



Google Classroom



Cronograma Previsto

Conteúdo

- 12 Aulas teóricas
- 02 Aulas para revisão
- 02 Aulas para provas
- 14 Aulas de laboratório

Feriados

- 29/03 - Sexta-feira
- 01/05 - Quarta-feira
- 31/05 - Sexta-feira

Ajustes para sincronizar Turmas

- 01/04 - Sem Aula
- 03/06 - Sem Aula

Avaliação

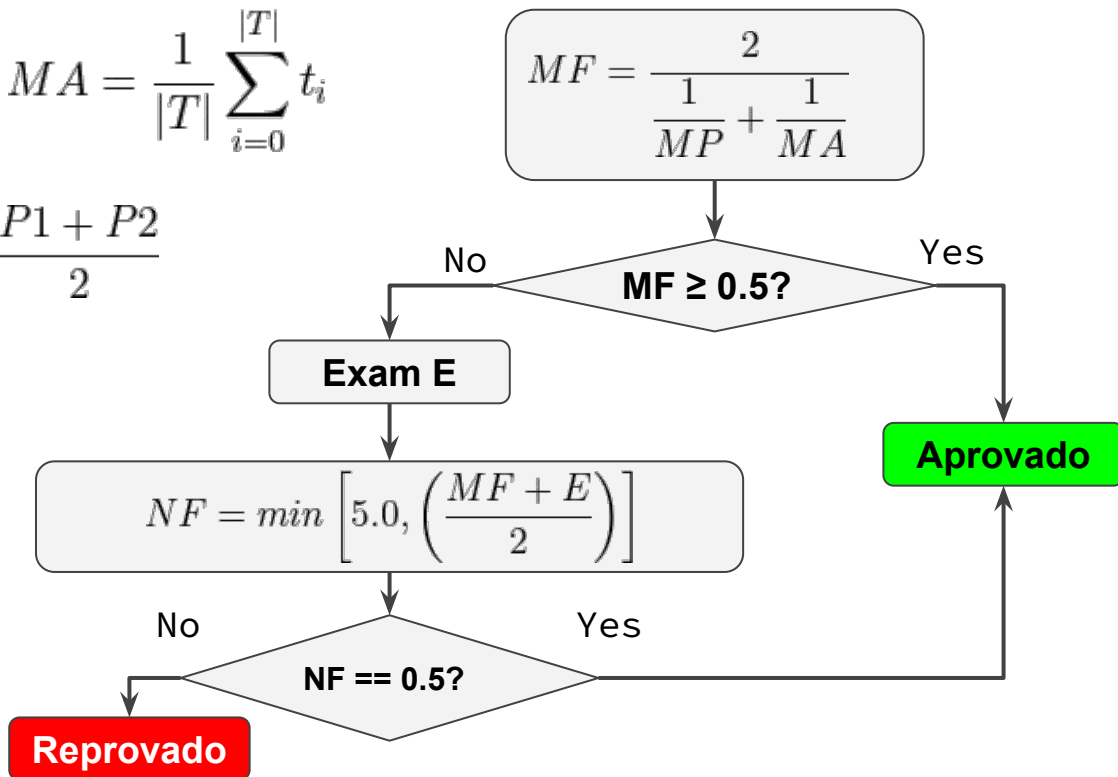
- Média Atividades (MA) $MA = \frac{1}{|T|} \sum_{i=0}^{|T|} t_i$

- Média Prova (MP) $MP = \frac{P1 + P2}{2}$

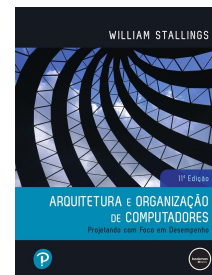
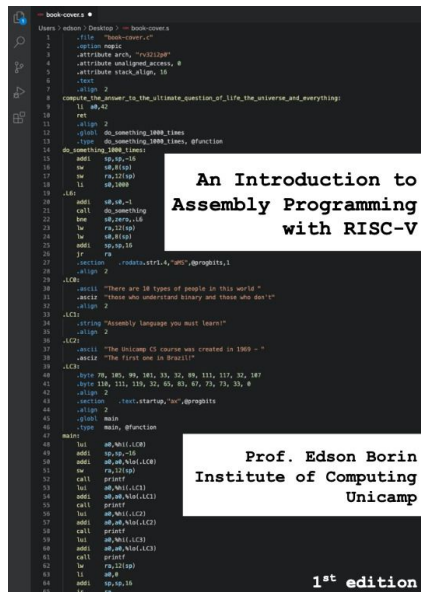
- Média Final (MF)

- Datas

- P1 24/04/2024
- P2 19/06/2024
- Exame 10/07/2024



Bibliografia da disciplina



- An Introduction to Assembly Programming with RISC-V, 1a. Edição Edson Borin 2023
- Computer Organization and Design 4a. Edição. David A. Patterson e John L. Hennessy
- Arquitetura e Organização de Computadores. 5a. Edição. William Stallings

Comentários Gerais & Dúvidas?

Por Que Aprender Linguagem de Montagem?

Por que aprender linguagem de montagem?

Permite compreender o funcionamento da CPU

Utilizado na:

- Programação de máquinas baseadas em microcontroladores
- Programação de sistemas embarcados
- Programação de trechos críticos (tempo/memória)
- Acesso a recursos não disponível em alto nível

A linguagem de montagem é absolutamente ligada ao hardware, depende de cada máquina específica (diferentemente das linguagens de alto nível, como C, C++ e Java)

Por que aprender linguagem de montagem?

Permite entender como programas escritos em linguagens de alto nível, como C ou Java, são traduzidos para a linguagem de máquina

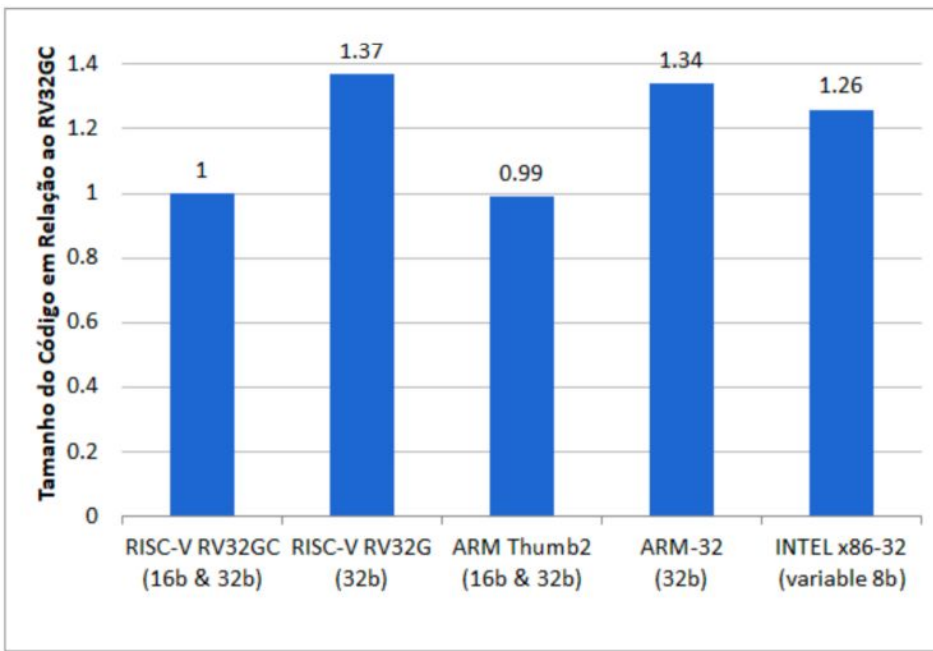
Por que RISC-V?

ISA (Instruction Architecture Set) aberta

Modular para ajustar processadores as aplicações

Instruções compactas -> reduz tamanho do código

Amplamente adotado por sistemas embarcados e dispositivos IoT

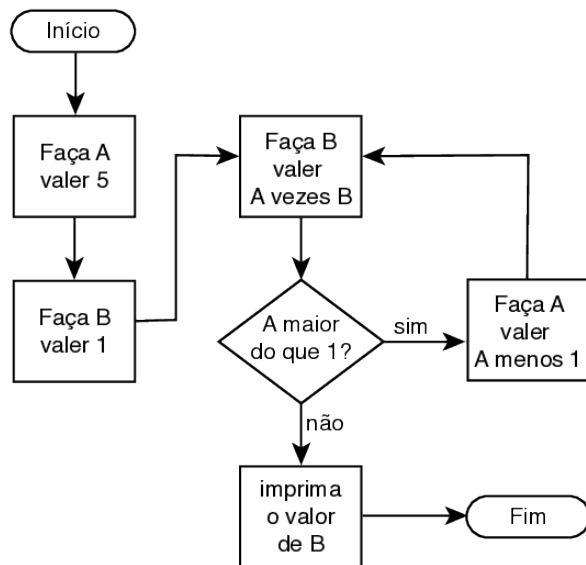


Conceitos Básicos

Resolução de Problemas com Computadores

Problema

Resolução de Problemas com Computadores



Problema

Algoritmo

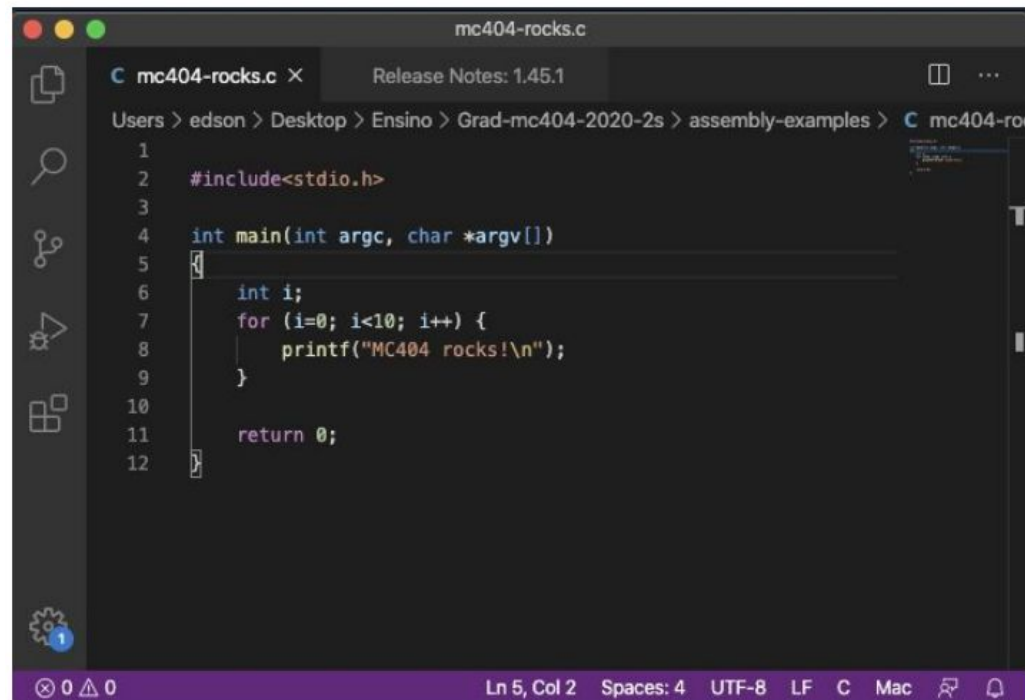
Modelo computacional

Algoritmo 1 Exemplo de Pseudocódigo.

```

leia (x, y) {Esta linha é um comentário}
se x > y então
    escreva ("x é maior")
senão
    se y > x então
        escreva ("y é maior")
    senão
        escreva ("x e y são iguais")
    fim-se
fim-se
  
```

Resolução de Problemas com Computadores



```
mc404-rocks.c
Release Notes: 1.45.1

Users > edson > Desktop > Ensino > Grad-mc404-2020-2s > assembly-examples > mc404-rocks.c

1
2  #include<stdio.h>
3
4  int main(int argc, char *argv[])
5  {
6      int i;
7      for (i=0; i<10; i++) {
8          printf("MC404 rocks!\n");
9      }
10
11     return 0;
12 }
```

Problema

Algoritmo

Programa Ling. Alto Nível

Sintaxe e Semântica da Ling.

Resolução de Problemas com Computadores

```

mc404-rocks.hexdump
mc404-rocks.c  mc404-rocks.s  mc404-rocks.hexdump X
ers > edson > Desktop > Ensino > Grad-mc404-2020-2s > assembly-examples > mc404-rocks.he
1  00000000 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00
2  00000100 01 00 f3 00 01 00 00 00 00 00 00 00 00 00 00
3  00000200 90 02 00 00 00 00 00 00 34 00 00 00 00 00 28 00
4  00000300 0b 00 0a 00 13 01 01 fd 23 26 11 02 23 24 81 02
5  00000400 13 04 01 03 23 2e a4 fc 23 2c b4 fc 23 26 04 fe
6  00000500 6f 00 00 02 b7 07 00 00 13 85 07 00 97 00 00 00
7  00000600 e7 80 00 00 83 27 c4 fe 93 87 17 00 23 26 f4 fe
8  00000700 03 27 c4 fe 93 07 90 00 e3 de e7 fc 93 07 00 00
9  00000800 13 85 07 00 83 20 c1 02 03 24 81 02 13 01 01 03
10 00000900 67 80 00 00 4d 43 34 30 34 20 72 6f 63 6b 73 21
11 00000a00 00 00 47 43 43 3a 20 28 47 4e 55 29 20 39 2e 32
12 00000b00 2e 30 00 41 2f 00 00 00 72 69 73 63 76 00 01 25
13 00000c00 00 00 00 04 10 05 72 76 33 32 69 32 70 30 5f 6d
14 00000d00 32 70 30 5f 61 32 70 30 5f 66 32 70 30 5f 64 32
15 00000e00 70 30 00 00 00 00 00 00 00 00 00 00 00 00 00
16 00000f00 00 00 00 00 01 00 00 00 00 00 00 00 00 00 00
17 00001000 04 00 f1 ff 00 00 00 00 00 00 00 00 00 00 00
18 00001100 03 00 01 00 00 00 00 00 00 00 00 00 00 00 00
19 00001200 03 00 03 00 00 00 00 00 00 00 00 00 00 00 00
  
```

Problema

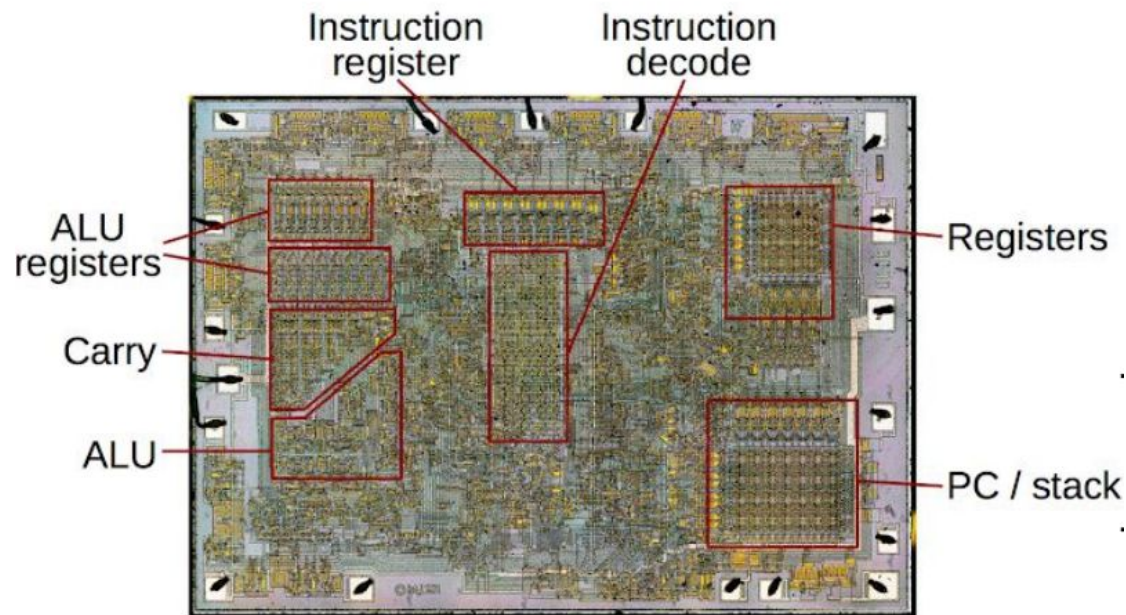
Algoritmo

Programa Ling. Alto Nível

Programa Ling. Máquina

Organização e Arquitetura

Resolução de Problemas com Computadores



Problema

Algoritmo

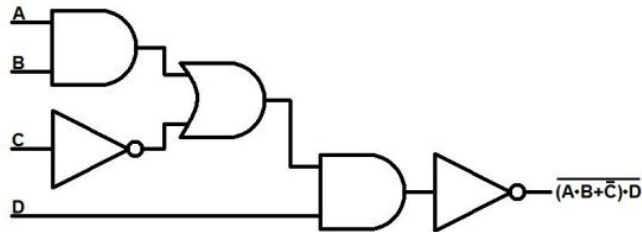
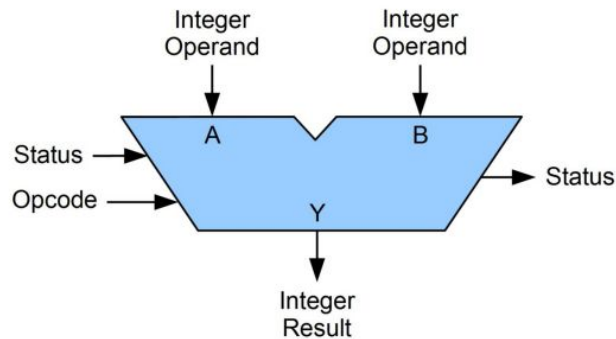
Programa Ling. Alto Nível

Programa Ling. Máquina

Microarquitetura do Proc.

Dispositivos lógicos,
registradores, ULA, ...

Resolução de Problemas com Computadores



Problema

Algoritmo

Programa Ling. Alto Nível

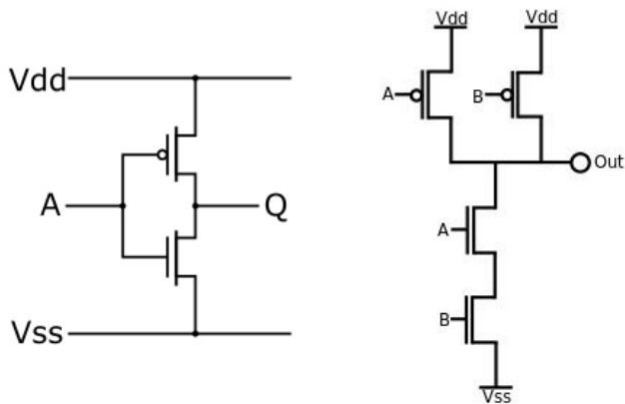
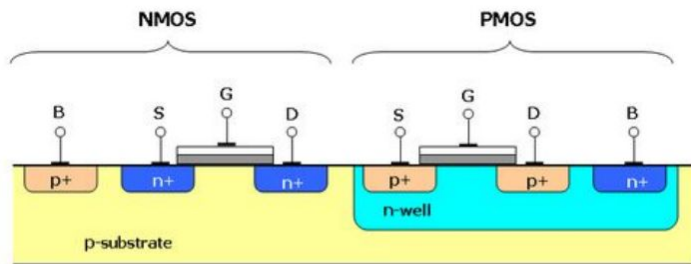
Programa Ling. Máquina

Microarquitetura do Proc.

Circuitos Lógicos

Portas lógicas

Resolução de Problemas com Computadores



Problema

Algoritmo

Programa Ling. Alto Nível

Programa Ling. Máquina

Microarquitetura do Proc.

Circuitos Lógicos

Portas Lógicas

Resolução de Problemas com Computadores

Níveis de Abstração

Problema

Algoritmo

Programa Ling. Alto Nível

Programa Ling. Máquina

Microarquitetura do Proc.

Circuitos Lógicos

Portas Lógicas

Resolução de Problemas com Computadores

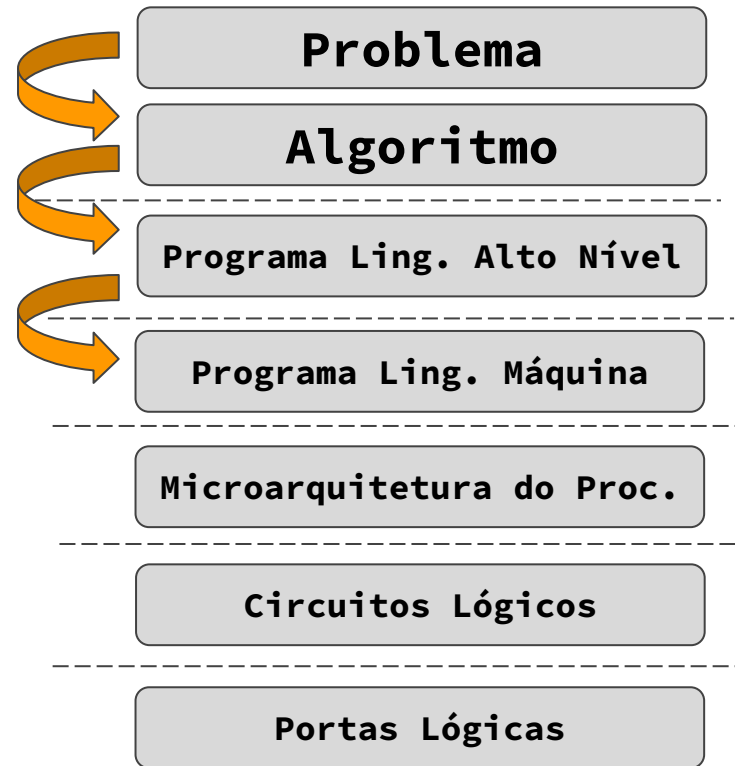
MC102 e MC202

Projeto de Software: Definir algoritmos e estrutura de dados

Programação: Implementar um projeto com uma linguagem

MC404 e MC910

Compilação e Interpretação: Converter linguagem para instruções de máquina



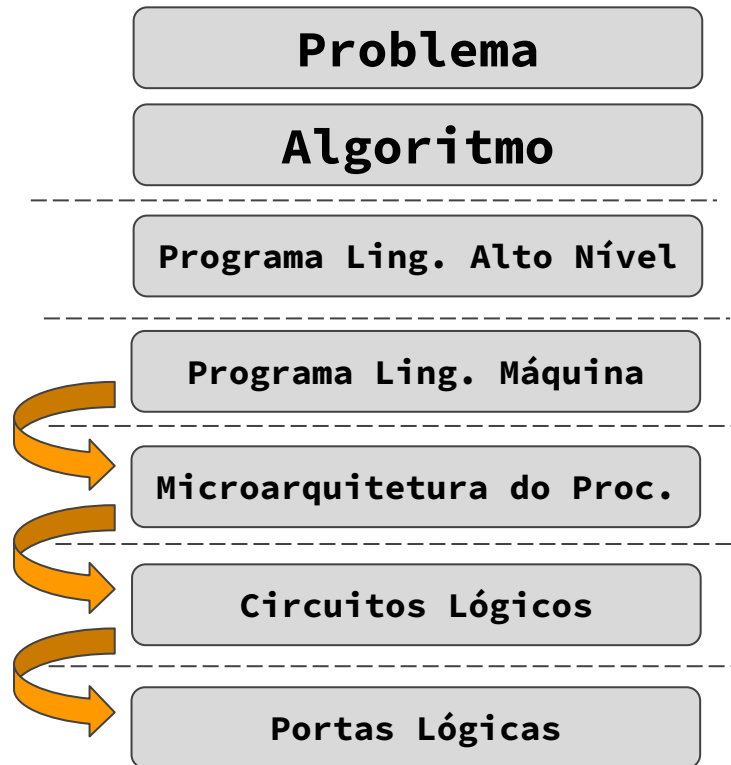
Resolução de Problemas com Computadores

MC722

Projeto de processadores: *Escolher estruturas para implementar ISA*

MC602

Projeto de Circuitos Lógicos:
Projeto a nível de gates e componentes



Conceitos Básicos - ISA

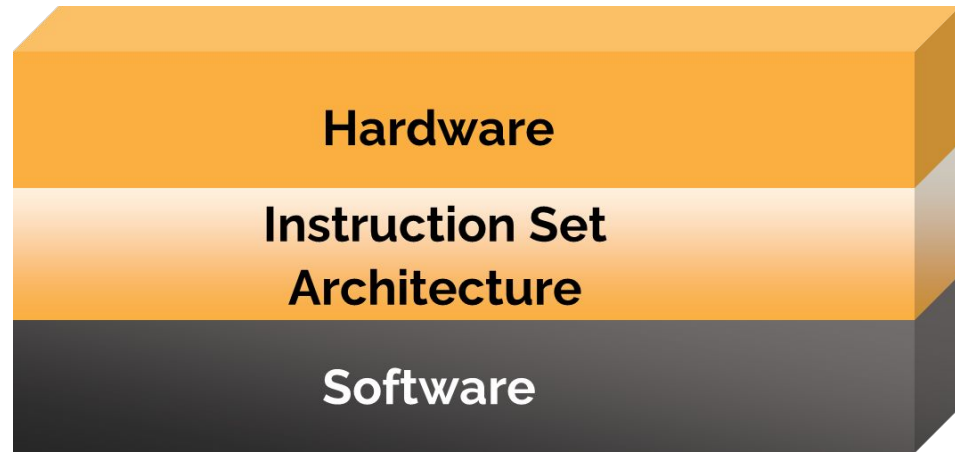
Conceitos Básicos - ISA

ISA

Instruction Set Architecture:

Define a interface entre software e hardware

Conjunto de instruções que podem ser usadas pelo computador para realizar as operações



Conceitos Básicos - ISA



Instruction Set Architecture



Conceitos Básicos - Arquitetura vs Microarquitetura

Arquitetura vs Microarquitetura

— — — —

Arquitetura é o modelo

- x86, ARM, RISC-V, Power

Microarquitetura é a implementação

- Intel i7 geração 14, AMD Ryzen 9, ARM Cortex-A53, RISC-V RV32I, PowerPC 970

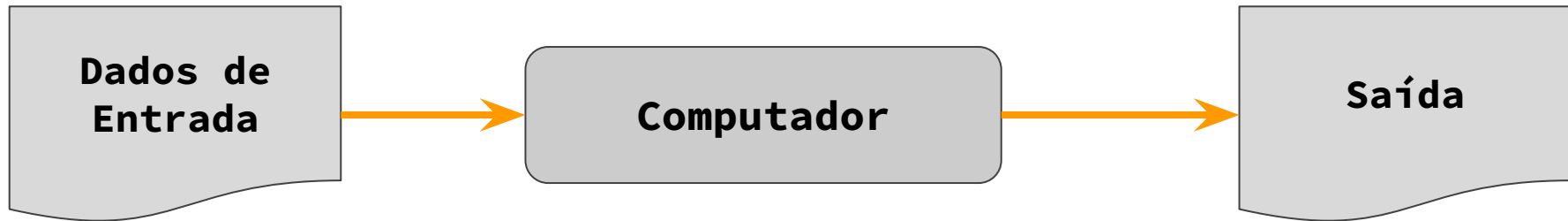
Arquiteturas de um mesmo modelo **precisam** utilizar um **conjunto base de instruções**, porém instruções **adicionais** podem ser incluídas.

Exemplo: AMD e Intel

Conceitos Básicos - Computadores

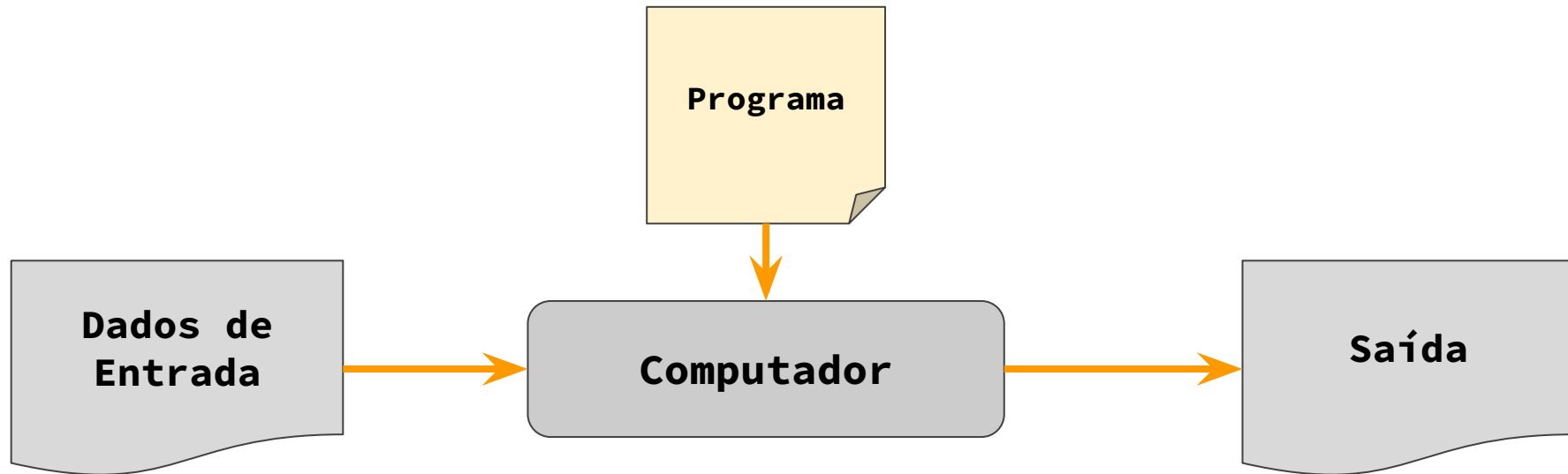
Conceitos Básicos - Computadores

Máquinas para **manipular** informações ou dados



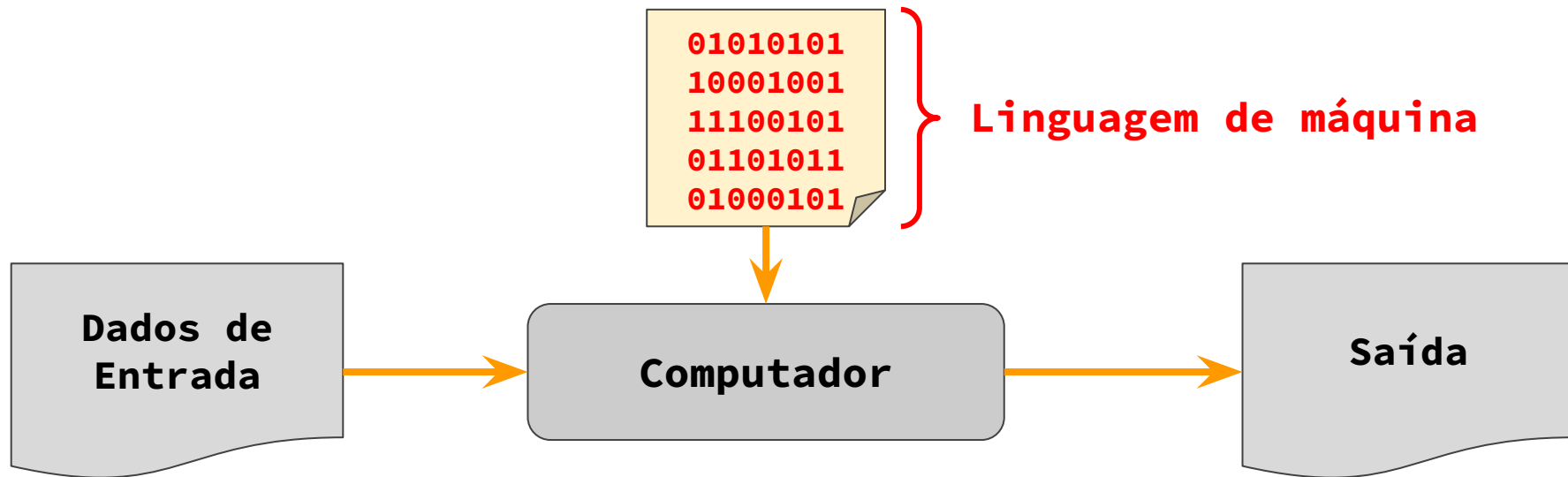
Conceitos Básicos - Computadores

Máquinas “**programáveis**” para **manipular** informações ou dados



Conceitos Básicos - Computadores

Máquinas “**programáveis**” para **manipular** informações ou dados



Conceitos Básicos - Linguagem de Programação

Conceitos Básicos - Linguagem de Programação

Linguagem de alto nível

```
int func(int a){  
    return a*113;  
}
```

Conceitos Básicos - Linguagem de Programação

Linguagem de alto nível

```
int func(int a){  
    return a*113;  
}
```

01010101
10001001
11100101
01101011
01000101



Computador

Saída

Conceitos Básicos - Linguagem de Programação

Compilador
gcc, ...

```
int func(int a){  
    return a*113;  
}
```

func:
slli a5,a0,3
sub a5,a5,a0

Montador
as, ...

```
01010101  
10001001  
01010101
```

Conceitos Básicos - Linguagem de Programação

Laços, variáveis, objetos, ...
Independente de máquina

Linguagem de baixo nível
Sequência de instruções,
registradores, posições de
memória

Dependente de máquina

Codificada de forma binária (0s
e 1s)

Dependente de máquina

Compilador
gcc, ...

```
int func(int a){  
    return a*113;  
}
```

Montador
as, ...

```
func:  
    slli a5,a0,3  
    sub a5,a5,a0
```

```
01010101  
10001001  
01010101
```

Exemplo

Compilador
gcc, ...



```
int func_1(int a, int b, int c) {  
    return (a + (113 * b)) * c;  
}
```

```
func_1:  
    push %ebp  
    mov %esp, %ebp  
    imul $113, 12(%ebp), %eax  
    add 8(%ebp), %eax  
    imul 16(%ebp), %eax  
    pop %ebp  
    ret
```

Exemplo

func_1:

```
push %ebp
mov %esp, %ebp
imul $113, 12(%ebp), %eax
add 8(%ebp), %eax
imul 16(%ebp), %eax
pop %ebp
ret
```

push %ebp

- **Opcode típico:** 55
- **Descrição:** Empilha o valor de ebp na pilha.
- **Binário:** 01010101

mov %esp, %ebp

- **Opcode típico:** 89 E5
- **Descrição:** Move esp para ebp.
- **Binário:** 10001001 11100101

imul \$113, 12(%ebp), %eax

- **Opcode típico:** 6B 45 0C 71
- **Descrição:** Multiplica o valor no endereço ebp+12 por 113, e o resultado vai para eax. Os opcodes podem variar bastante para instruções de multiplicação imediata, dependendo do montador e das otimizações.
- **Binário:** 01101011 01000101 00001100 01110001

Exemplo

func_1:

```
push %ebp
mov %esp, %ebp
imul $113, 12(%ebp), %eax
add 8(%ebp), %eax
imul 16(%ebp), %eax
pop %ebp
ret
```

ret

- **Opcode típico:** C3
- **Descrição:** Retorna da função.
- **Binário:** 11000011

add 8(%ebp), %eax

- **Opcode típico:** 03 45 08
- **Descrição:** Adiciona o valor no endereço ebp+8 a eax.
- **Binário:** 00000011 01000101 00001000

imul 16(%ebp), %eax

- **Opcode típico:** 0F AF 45 10
- **Descrição:** Multiplica eax pelo valor no endereço ebp+16, e o resultado vai para eax. Novamente, os detalhes exatos podem variar.
- **Binário:** 00001111 10101111 01000101 00010000

pop %ebp


- **Opcode típico:** 5D
- **Descrição:** Desempilha o valor do topo da pilha de volta para ebp.
- **Binário:** 01011101

Exemplo

func_1:

```
push %ebp
mov %esp, %ebp
imul $113, 12(%ebp), %eax
add 8(%ebp), %eax
imul 16(%ebp), %eax
pop %ebp
ret
```

**Montador
as, ...**



```
01010101
10001001
11100101
01101011
01000101
00001100
01110001
00000011
01000101
00001000
00001111
10101111
01000101
00010000
01011101
11000011
```

Linguagem de Máquina x86

Exemplo

func_1:

```

push %ebp
mov %esp, %ebp
imul $113, 12(%ebp), %eax
add 8(%ebp), %eax
imul 16(%ebp), %eax
pop %ebp
ret

```

Montador
as, ...

```

01010101
10001001
11100101
01101011
01000101
00001100
01110001
00000011
01000101
00001000
00001111
10101111
01000101
00010000
01011101
11000011

```

Linguagem de Máquina x86

Desmontador

Desmontador
objdump, ...



```
00000000 <_func_1>:
  0: 55 push %ebp
  1: 89 e5 mov %esp,%ebp
  3: 6b 45 0c 71 imul $0x71,0xc(%ebp),%eax
  7: 03 45 08 add 0x8(%ebp),%eax
 a: 0f af 45 10 imul 0x10(%ebp),%eax
 e: 5d pop %ebp
 f: c3 ret
```

```
01010101
10001001
11100101
01101011
01000101
00001100
01110001
00000011
01000101
00001000
00001111
10101111
01000101
00010000
01011101
11000011
```

Linguagem de Máquina x86

Outros Exemplos

Programa em Ling. C

```
int func_1(int a, int b, int c){
    return (a + (113 * b)) * c;
}
```

Ling. Montagem do RISC-V

```
func_1:
    slli a5,a1,3
    sub a5,a5,a1
    slli a5,a5,4
    add a5,a5,a1
    add a0,a5,a0
    mul a0,a0,a2
    ret
```

Ling. Montagem do ARM

```
_func_1:
    rsb r3, r1, r1, asl #3
    add r1, r1, r3, asl #4
    add r1, r1, r0
    mul r0, r2, r1
    bx lr
```

Ling. Montagem do x86

```
func_1:
    push %ebp
    mov %esp, %ebp
    imul $113, 12(%ebp), %eax
    add 8(%ebp), %eax
    imul 16(%ebp), %eax
    pop %ebp
    ret
```

Agradecimentos

Prof. Edson Borin
Prof. Rodolfo Azevedo



Próxima Aula

— — — —

Execução de Programas em Computadores

- Componentes de um computador
 - Memória principal
 - CPU
 - Memória secundária
 - Barramentos
 - Periféricos
- Codificação de programas de computador
- Geração de programas nativos
- Execução de programas nativos