



UNIVERSIDADE  
FEDERAL DE  
SERGIPE



DEPARTAMENTO  
DE COMPUTAÇÃO

# Conjunto de instruções da arquitetura

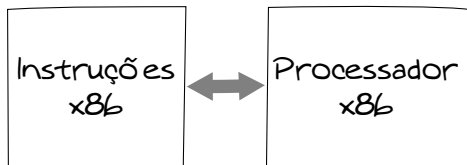
## Arquitetura de Computadores

Bruno Prado

Departamento de Computação / UFS

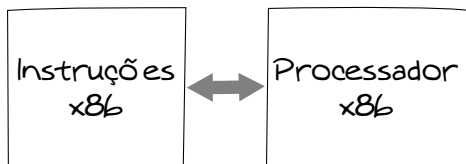
# Introdução

- ▶ O que é o conjunto de instruções da arquitetura?
  - ▶ Um conjunto de instruções da arquitetura é o idioma que um computador é capaz de interpretar e executar o comportamento



# Introdução

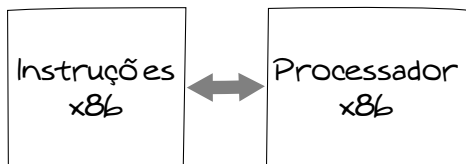
- ▶ O que é o conjunto de instruções da arquitetura?
  - ▶ Um conjunto de instruções da arquitetura é o idioma que um computador é capaz de interpretar e executar o comportamento



- ▶ As instruções são equivalentes às palavras de um texto e cada arquitetura possui pelo menos uma linguagem que o processador é capaz de entender

# Introdução

- ▶ O que é o conjunto de instruções da arquitetura?
  - ▶ Um conjunto de instruções da arquitetura é o idioma que um computador é capaz de interpretar e executar o comportamento



- ▶ As instruções são equivalentes às palavras de um texto e cada arquitetura possui pelo menos uma linguagem que o processador é capaz de entender

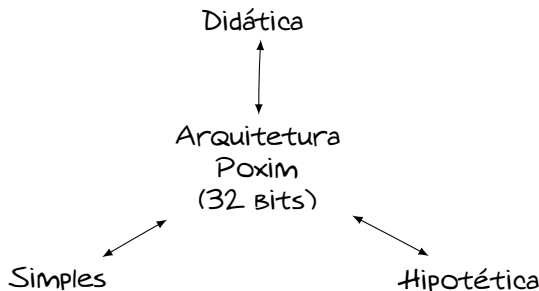
*Instruction Set Architecture (ISA)*

# Introdução

- ▶ Como dizer ao computador o que deve ser feito?
  - ▶ É preciso conhecer a linguagem de máquina
    - ▶ Códigos para operações
    - ▶ Parâmetros de entrada e de saída
    - ▶ Comportamento de cada operação
  - ▶ Carregar o programa na memória

# Introdução

- ▶ Como dizer ao computador o que deve ser feito?
  - ▶ É preciso conhecer a linguagem de máquina
    - ▶ Códigos para operações
    - ▶ Parâmetros de entrada e de saída
    - ▶ Comportamento de cada operação
  - ▶ Carregar o programa na memória



# Introdução

- ▶ Como as informações são representadas?
  - ▶ Cada *nibble* (4 bits) corresponde a um dígito hexadecimal (base 16)

Decimal	Binário	Hexadecimal
$0_{10}$	$0000_2$	$0_{16}$
$1_{10}$	$0001_2$	$1_{16}$
$2_{10}$	$0010_2$	$2_{16}$
$\vdots$	$\vdots$	$\vdots$
$13_{10}$	$1101_2$	$D_{16}$
$14_{10}$	$1110_2$	$E_{16}$
$15_{10}$	$1111_2$	$F_{16}$

# Introdução

- ▶ Como é organizada a memória?
  - ▶ Os dados são divididos em bytes

Endereço	Byte
0x00..00	B <sub>1</sub>
0x00..01	B <sub>2</sub>
0x00..02	B <sub>3</sub>
⋮	⋮
0xFF..FE	B <sub>n-1</sub>
0xFF..FF	B <sub>n</sub>



# Introdução

- ▶ Como é organizada a memória?
  - ▶ Os dados são divididos em bytes

Endereço	Byte
0x00..00	B <sub>1</sub>
0x00..01	B <sub>2</sub>
0x00..02	B <sub>3</sub>
⋮	⋮
0xFF..FE	B <sub>n-1</sub>
0xFF..FF	B <sub>n</sub>

4 GiB  $\rightarrow n = 2^{32} = 4.294.967.296$  Bytes

# Introdução

- ▶ Como armazenar dados com mais de 1 byte?
  - ▶ Mais significativo primeiro (*big-endian*)
  - ▶ Menos significativo primeiro (*little-endian*)

Endereço	Byte
0x00	0xAA
0x01	0xBB
0x02	0xCC
0x03	0xDD

Big-endian

Endereço	Byte
0x00	0xDD
0x01	0xCC
0x02	0xBB
0x03	0xAA

Little-endian

0xAABBCCDD

# Introdução

- ▶ Como é feito o endereçamento na memória?
  - ▶ Definido pelo alinhamento dos dados

Endereço	Dado	
0x00..00	$B_1$	$B_2$
0x00..02	$B_3$	$B_4$
$\vdots$	$\vdots$	$\vdots$
0xFF..FC	$B_{n-3}$	$B_{n-2}$
0xFF..FE	$B_{n-1}$	$B_n$

16 Bits (2 Bytes)

# Introdução

- ▶ Como é feito o endereçamento na memória?
  - ▶ Definido pelo alinhamento dos dados

Endereço	Dado			
0x00..00	$B_1$	$B_2$	$B_3$	$B_4$
0x00..04	$B_5$	$B_6$	$B_7$	$B_8$
⋮	⋮	⋮	⋮	⋮
0xFF..F8	$B_{n-7}$	$B_{n-6}$	$B_{n-5}$	$B_{n-4}$
0xFF..FC	$B_{n-3}$	$B_{n-2}$	$B_{n-1}$	$B_n$

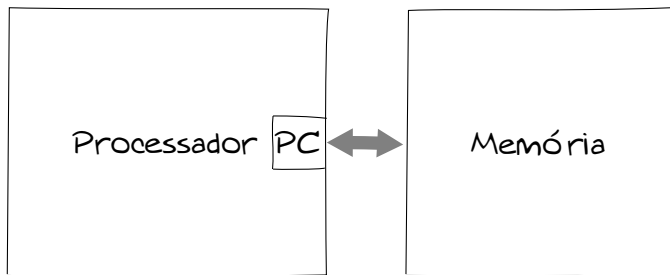
32 Bits (4 Bytes)

# Fluxo de execução

- ▶ O processador executa as instruções em um ciclo infinito de busca-decodificação-execução
  1. Buscar instrução armazenada na memória
  2. Decodificar a operação da instrução
  3. Obter os operandos necessários
  4. Executar o comportamento
  5. Armazenar os resultados
  6. Voltar para o passo 1

# Fluxo de execução

- ▶ Ciclo de busca-decodificação-execução
  - ▶ A programação armazenada em memória é indexada pelo contador de programa (PC) que controla o fluxo de execução da aplicação



# Operações e operandos

- ▶ Quais são as operações principais?
  - ▶ Aritméticas
  - ▶ Bit a bit e lógica
  - ▶ Controle do fluxo de execução

# Operações e operandos

- ▶ Quais são as operações principais?
  - ▶ Aritméticas
  - ▶ Bit a bit e lógica
  - ▶ Controle do fluxo de execução
- ▶ Como os operandos são utilizados?
  - ▶ Registradores
  - ▶ Imediatos ou constantes
  - ▶ Armazenados na memória



# Operações aritméticas

- ▶ Principais operações aritméticas
  - ▶ Adição
  - ▶ Subtração
  - ▶ Multiplicação
  - ▶ Divisão
  - ▶ Deslocamento

# Operações aritméticas

- ▶ Instrução de adição
  - ▶ Operandos em registradores

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em registrador
6     uint32_t x, y, a = 3, b = 5;
7     // Operações de adição
8     x = a + b;
9     y = b + x;
10    // Retorno sem erros
11    return 0;
12 }
```



```
...
add x, a, b
add y, b, x
...
```

# Operações aritméticas

- ▶ Instrução de adição
  - ▶ Operandos em registradores e imediato

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em registrador
6     uint32_t x, y, a = 3, b = 5;
7     // Operações de adição
8     x = a + 2;
9     y = b + 3;
10    // Retorno sem erros
11    return 0;
12 }
```



```
...
addi x, a, 2
addi y, b, 3
...
```

# Operações aritméticas

- ▶ Instrução de subtração
  - ▶ Operandos em registradores

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em registrador
6     uint32_t x, y, a = 3, b = 5;
7     // Operações de subtração
8     x = a - b;
9     y = b - x;
10    // Retorno sem erros
11    return 0;
12 }
```



```
...
sub x, a, b
sub y, b, x
...
```

# Operações aritméticas

- ▶ Instrução de subtração
  - ▶ Operandos em registradores e imediato

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em registrador
6     uint32_t x, y, a = 3, b = 5;
7     // Operações de subtração
8     x = a - 5;
9     y = b - 8;
10    // Retorno sem erros
11    return 0;
12 }
```



```
...
subi x, a, 5
subi y, b, 8
...
```

# Operações aritméticas

- ▶ Instrução de multiplicação
  - ▶ Operandos em registradores

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em registrador
6     uint32_t x, y, a = 3, b = 5;
7     // Operações de multiplicação
8     x = a * b;
9     y = b * x;
10    // Retorno sem erros
11    return 0;
12 }
```



```
...
mul x, a, b
mul y, b, x
...
```

# Operações aritméticas

- ▶ Instrução de multiplicação
  - ▶ Operandos em registradores e imediato

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em registrador
6     uint32_t x, y, a = 3, b = 5;
7     // Operações de multiplicação
8     x = a * 3;
9     y = b * 9;
10    // Retorno sem erros
11    return 0;
12 }
```



```
...
mulh x, a, 3
mulh y, b, 9
...
```

# Operações aritméticas

- ▶ Instrução de divisão
  - ▶ Operandos em registradores

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em registrador
6     uint32_t x, y, a = 3, b = 5;
7     // Operações de divisão
8     x = a / b;
9     y = b / x;
10    // Retorno sem erros
11    return 0;
12 }
```



```
...
div x, a, b
div y, b, x
...
```



# Operações aritméticas

- ▶ Instrução de divisão
  - ▶ Operandos em registradores e imediato

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em registrador
6     uint32_t x, y, a = 3, b = 5;
7     // Operações de divisão
8     x = a / 3;
9     y = b / 7;
10    // Retorno sem erros
11    return 0;
12 }
```



```
...
divi x, a, 3
divi y, b, 7
...
```

# Operações aritméticas

- ▶ Instruções de deslocamento
  - ▶ Direita ( $\div 2$ ) e esquerda ( $\times 2$ )

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em registrador
6     int32_t x, y, a = -2, b = 5;
7     // Operações de deslocamento
8     x = a >> 1;
9     y = b << 2;
10    // Retorno sem erros
11    return 0;
12 }
```



```
...
sra x, a, 1
sla y, b, 2
...
```

# Operações aritméticas

- ▶ Instruções com operandos em memória
  - ▶ Carregamento dos dados para registradores

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em memória
6     uint32_t x = 8, y = 13;
7     // Operações de aritméticas
8     x = x - 5;
9     y = y / x;
10    // Retorno sem erros
11    return 0;
12 }
```



```
...
l32 x, [0x40]
l32 y, [0x41]
subi x, x, 5
div y, y, x
s32 [0x40], x
s32 [0x41], y
...
```

# Operações aritméticas

- ▶ Instruções com operandos em memória
  - ▶ Visualização dos dados

Endereço	Memória	Variável
⋮	⋮	⋮
0x000000100	8	x
0x000000104	13	y
⋮	⋮	⋮

# Operações aritméticas

- ▶ Instruções com operandos em memória
  - ▶ Visualização do programa

Endereço	Instrução
⋮	⋮
0x000000020	l32 x, [0x40]
0x000000024	l32 y, [0x41]
0x000000028	subi x, x, 5
0x00000002C	div y, y, x
0x000000030	s32 [0x40], x
0x000000034	s32 [0x41], y
⋮	⋮

# Operações aritméticas

- ▶ Instruções com operandos em memória
  - ▶ Visualização do programa

Endereço	Instrução
⋮	⋮
0x000000020	l32 x, [0x40]
0x000000024	l32 y, [0x41]
0x000000028	subi x, x, 5
0x00000002C	div y, y, x
0x000000030	s32 [0x40], x
0x000000034	s32 [0x41], y
⋮	⋮

# Operações aritméticas

- ▶ Instruções com operandos em memória
  - ▶ Visualização do programa

Endereço	Instrução
⋮	⋮
0x000000020	l32 x, [0x40]
0x000000024	l32 y, [0x41]
0x000000028	subi x, x, 5
0x00000002C	div y, y, x
0x000000030	s32 [0x40], x
0x000000034	s32 [0x41], y
⋮	⋮

# Operações aritméticas

- ▶ Instruções com operandos em memória
  - ▶ Visualização do programa

Endereço	Instrução
⋮	⋮
0x000000020	l32 x, [0x40]
0x000000024	l32 y, [0x41]
0x000000028	subi x, x, 5
0x00000002C	div y, y, x
0x000000030	s32 [0x40], x
0x000000034	s32 [0x41], y
⋮	⋮



# Operações aritméticas

- ▶ Instruções com operandos em memória
  - ▶ Visualização do programa

Endereço	Instrução
⋮	⋮
0x000000020	l32 x, [0x40]
0x000000024	l32 y, [0x41]
0x000000028	subi x, x, 5
0x00000002C	div y, y, x
0x000000030	s32 [0x40], x
0x000000034	s32 [0x41], y
⋮	⋮

# Operações aritméticas

- ▶ Instruções com operandos em memória
  - ▶ Visualização do programa

Endereço	Instrução
⋮	⋮
0x000000020	l32 x, [0x40]
0x000000024	l32 y, [0x41]
0x000000028	subi x, x, 5
0x00000002C	div y, y, x
0x000000030	s32 [0x40], x
0x000000034	s32 [0x41], y
⋮	⋮

# Operações aritméticas

- ▶ Instruções com operandos em memória
  - ▶ Visualização do programa

Endereço	Instrução
⋮	⋮
0x000000020	l32 x, [0x40]
0x000000024	l32 y, [0x41]
0x000000028	subi x, x, 5
0x00000002C	div y, y, x
0x000000030	s32 [0x40], x
0x000000034	s32 [0x41], y
⋮	⋮

# Operações aritméticas

- ▶ Instruções com operandos em memória
  - ▶ Visualização dos resultados

Endereço	Memória	Variável
⋮	⋮	⋮
0x000000100	3	x
0x000000104	4	y
⋮	⋮	⋮

# Operações bit a bit

- ▶ Principais operações bit a bit
  - ▶ E (and)
  - ▶ Ou (or)
  - ▶ Complemento (not)
  - ▶ Ou-exclusivo (xor)
  - ▶ Deslocamento lógico

# Operações bit a bit

## ► Instruções and, or, xor, not e deslocamento

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em memória
6     uint32_t x = 13, y = 21;
7     // Variáveis em registradores
8     uint32_t a, b, c, d, e;
9     // Operações bit a bit
10    a = x & y;
11    b = x | y;
12    c = x ^ y;
13    d = ~x;
14    e = x >> y;
15    // Retorno sem erros
16    return 0;
17 }
```



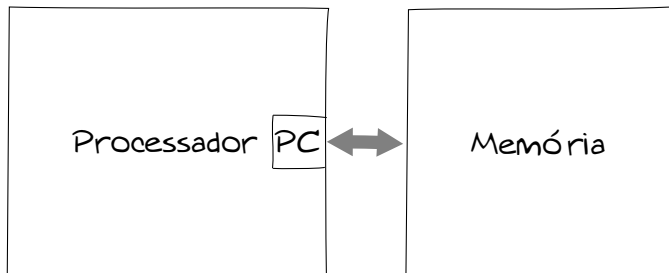
```
...
132 x, [0x40]
132 y, [0x41]
and a, x, y
or b, x, y
xor c, x, y
not d, x
srl e, x, y
...
```

# Operações de controle de fluxo

- ▶ Controle do fluxo de execução
  - ▶ Chamadas de funções e procedimentos
  - ▶ Controles condicionais e iterativos

# Operações de controle de fluxo

- ▶ Chamadas de funções e procedimentos
  - ▶ Quando é ligado ou reiniciado, o sistema começa a executar a programação armazenada em memória
  - ▶ Cada arquitetura define um valor inicial para o contador de programa (PC) que aponta para as rotinas de inicialização do software (*bootloader*)





# Operações de controle de fluxo

- ▶ Chamadas de funções e procedimentos
  - ▶ Desvio incondicional: função principal (*main*)

```
1 // Função principal
2 int main() {
3     // Retorno sem erros
4     return 0;
5 }
```



```
bun 0x7
...
int 0
```

# Operações de controle de fluxo

- ▶ Chamadas de funções e procedimentos
  - ▶ Desvio incondicional: função principal (*main*)

Endereço	Instrução
0x00000000	Bun 1
⋮	⋮
0x00000020	int 0
⋮	⋮

# Operações de controle de fluxo

- ▶ Chamadas de funções e procedimentos
  - ▶ Desvio incondicional: função principal (*main*)

Endereço	Instrução
0x00000000	Bun 1
⋮	⋮
0x00000020	int 0
⋮	⋮

# Operações de controle de fluxo

- ▶ Chamadas de funções e procedimentos
  - ▶ Desvio incondicional: função principal (*main*)

Endereço	Instrução
0x00000000	Bun 1
⋮	⋮
0x00000020	int 0
⋮	⋮

# Operações de controle de fluxo

## ► Controle condicional

### ► Sentença *if-else*

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em memória
6     uint32_t a, b;
7     // Controle condicional
8     if(a == b) a = 11;
9     else a = 0;
10    b = a * 5;
11    // Retorno sem erros
12    return 0;
13 }
```



```
bun 0x7
...
l32 a, [0x40]
l32 b, [0x41]
cmp a, b
bne 0x2
addi a, a, 11
bun 0x1
xor a, a, a
mulh b, a, 5
s32 [0x40], a
s32 [0x41], b
int 0
```

# Operações de controle de fluxo

- ▶ Controle condicional
  - ▶ Visualização dos dados

Endereço	Memória	Variável
⋮	⋮	⋮
0x000000100	0	a
0x000000104	0	B
⋮	⋮	⋮

# Operações de controle de fluxo

- ▶ Controle condicional
  - ▶ Visualização do programa

⋮	⋮
0x000000020	132 a, [0x40]
0x000000024	132 B, [0x41]
0x000000028	cmp a, B
0x00000002C	bne 2
0x000000030	addi a, a, 11
0x000000034	bun 1
0x000000038	xor a, a, a
0x00000003C	mulh B, a, 5
0x000000040	s32 [0x40], a
0x000000044	s32 [0x41], B
0x000000048	int 0
⋮	⋮

# Operações de controle de fluxo

- ▶ Controle condicional
  - ▶ Visualização do programa

⋮	⋮
0x000000020	132 a, [0x40]
0x000000024	132 B, [0x41]
0x000000028	cmp a, B
0x00000002C	bne 2
0x000000030	addi a, a, 11
0x000000034	bun 1
0x000000038	xor a, a, a
0x00000003C	mulh B, a, 5
0x000000040	s32 [0x40], a
0x000000044	s32 [0x41], B
0x000000048	int 0
⋮	⋮



# Operações de controle de fluxo

- ▶ Controle condicional
  - ▶ Visualização do programa

⋮	⋮
0x000000020	132 a, [0x40]
0x000000024	132 B, [0x41]
0x000000028	cmp a, B
0x00000002C	Bne 2
0x000000030	addi a, a, 11
0x000000034	Bun 1
0x000000038	xor a, a, a
0x00000003C	mul B, a, 5
0x000000040	s32 [0x40], a
0x000000044	s32 [0x41], B
0x000000048	int 0
⋮	⋮

# Operações de controle de fluxo

- ▶ Controle condicional
  - ▶ Visualização do programa

⋮	⋮
0x000000020	132 a, [0x40]
0x000000024	132 B, [0x41]
0x000000028	cmp a, B
0x00000002C	bne 2
0x000000030	addi a, a, 11
0x000000034	bunl
0x000000038	xor a, a, a
0x00000003C	mulh B, a, 5
0x000000040	s32 [0x40], a
0x000000044	s32 [0x41], B
0x000000048	int 0
⋮	⋮

# Operações de controle de fluxo

- ▶ Controle condicional
  - ▶ Visualização do programa

⋮	⋮
0x000000020	132 a, [0x40]
0x000000024	132 B, [0x41]
0x000000028	cmp a, B
0x00000002C	bne 2
0x000000030	addi a, a, 11
0x000000034	bun 1
0x000000038	xor a, a, a
0x00000003C	mulh B, a, 5
0x000000040	s32 [0x40], a
0x000000044	s32 [0x41], B
0x000000048	int 0
⋮	⋮

# Operações de controle de fluxo

- ▶ Controle condicional
  - ▶ Visualização do programa

⋮	⋮
0x000000020	132 a, [0x40]
0x000000024	132 B, [0x41]
0x000000028	cmp a, B
0x00000002C	bne 2
0x000000030	addi a, a, 11
0x000000034	bun 1
0x000000038	xor a, a, a
0x00000003C	mulh B, a, 5
0x000000040	s32 [0x40], a
0x000000044	s32 [0x41], B
0x000000048	int 0
⋮	⋮

# Operações de controle de fluxo

- ▶ Controle condicional
  - ▶ Visualização do programa

⋮	⋮
0x000000020	132 a, [0x40]
0x000000024	132 B, [0x41]
0x000000028	cmp a, B
0x00000002C	bne 2
0x000000030	addi a, a, 11
0x000000034	bun 1
0x000000038	xor a, a, a
0x00000003C	mul B, a, 5
0x000000040	s32 [0x40], a
0x000000044	s32 [0x41], B
0x000000048	int 0
⋮	⋮

# Operações de controle de fluxo

- ▶ Controle condicional
  - ▶ Visualização do programa

⋮	⋮
0x000000020	132 a, [0x40]
0x000000024	132 B, [0x41]
0x000000028	cmp a, B
0x00000002C	bne 2
0x000000030	addi a, a, 11
0x000000034	bun 1
0x000000038	xor a, a, a
0x00000003C	mulh B, a, 5
0x000000040	s32 [0x40], a
0x000000044	s32 [0x41], B
0x000000048	int 0
⋮	⋮

# Operações de controle de fluxo

- ▶ Controle condicional
  - ▶ Visualização do programa

⋮	⋮
0x000000020	132 a, [0x40]
0x000000024	132 B, [0x41]
0x000000028	cmp a, B
0x00000002C	bne 2
0x000000030	addi a, a, 11
0x000000034	bun 1
0x000000038	xor a, a, a
0x00000003C	mulh B, a, 5
0x000000040	s32 [0x40], a
0x000000044	s32 [0x41], B
0x000000048	int 0
⋮	⋮

# Operações de controle de fluxo

- ▶ Controle condicional
  - ▶ Visualização do programa

⋮	⋮
0x000000020	132 a, [0x40]
0x000000024	132 B, [0x41]
0x000000028	cmp a, B
0x00000002C	bne 2
0x000000030	addi a, a, 11
0x000000034	bun 1
0x000000038	xor a, a, a
0x00000003C	mulh B, a, 5
0x000000040	s32 [0x40], a
0x000000044	s32 [0x41], B
0x000000048	int 0
⋮	⋮



# Operações de controle de fluxo

- ▶ Controle condicional
  - ▶ Visualização dos resultados

Endereço	Memória	Variável
⋮	⋮	⋮
0x000000100	11	a
0x000000104	55	B
⋮	⋮	⋮

# Operações de controle de fluxo

## ► Controle iterativo

### ► Sentença *while*

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variável em memória
6     uint32_t a = 3;
7     // Controle iterativo
8     while(a > 0) {
9         a--;
10    }
11    // Retorno sem erros
12    return 0;
13 }
```



```
bun 0x7
...
l32 a, [0x40]
cmpi a, 0
beq 0x2
subi a, a, 1
bun -0x4
s32 [0x40], a
int 0
```

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000

⋮

0x00000020

0x00000024

0x00000028

0x0000002C

0x00000030

0x00000034

0x00000038

⋮

0x00000000

⋮

Bun 7
⋮
l32 a, [0x40]
cmpi a, 0
beq 2
subi a, a, 1
Bun -4
s32 [0x40], a
int 0
⋮
3
⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	Beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	Beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a



# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	Beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	Beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	Beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 1
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x00000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	Beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	Beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a



# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	Beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	Beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	3
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	0
⋮	⋮

a

# Operações de controle de fluxo

- ▶ Controle iterativo
  - ▶ Visualização da memória

0x00000000	Bun 7
⋮	⋮
0x00000020	l32 a, [0x40]
0x00000024	cmpi a, 0
0x00000028	beq 2
0x0000002C	subi a, a, 1
0x00000030	Bun -4
0x00000034	s32 [0x40], a
0x00000038	int 0
⋮	⋮
0x000000100	0
⋮	⋮

a

## Exercício

- ▶ Faça a tradução do código fonte abaixo para o seu respectivo código de montagem equivalente
  - ▶ Utilize as instruções e o endereçamento vistos
  - ▶ Simule a execução passo a passo a passo

```
1 // Inteiros com tamanho fixo
2 #include <stdint.h>
3 // Função principal
4 int main() {
5     // Variáveis em memória
6     uint32_t a, i = 3;
7     // Controle condicional
8     if(i == 3) i = 5;
9     // Controle iterativo
10    while(i > 0) {
11        a = a + i--;
12    }
13    // Retorno sem erros
14    return 0;
15 }
```