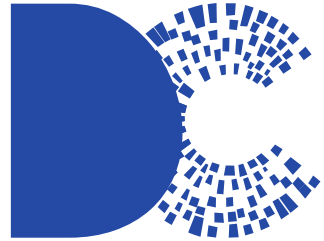
UNIVERSIDADE FEDERAL DE SERGIPE



DEPARTAMENTO DE COMPUTAÇÃO

Conjunto de instruções da arquitetura

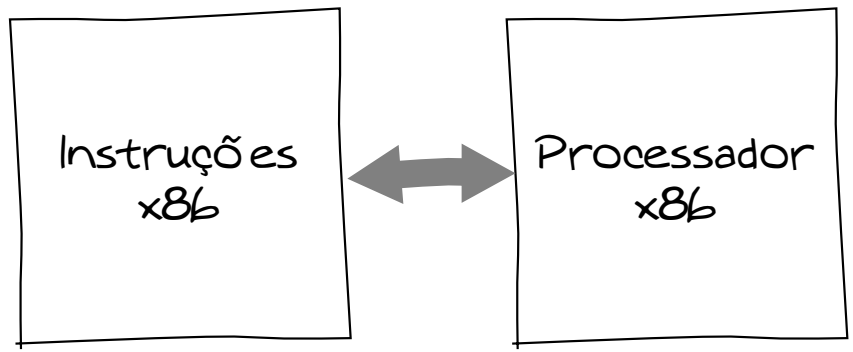
Arquitetura de Computadores

Bruno Prado

Departamento de Computação / UFS

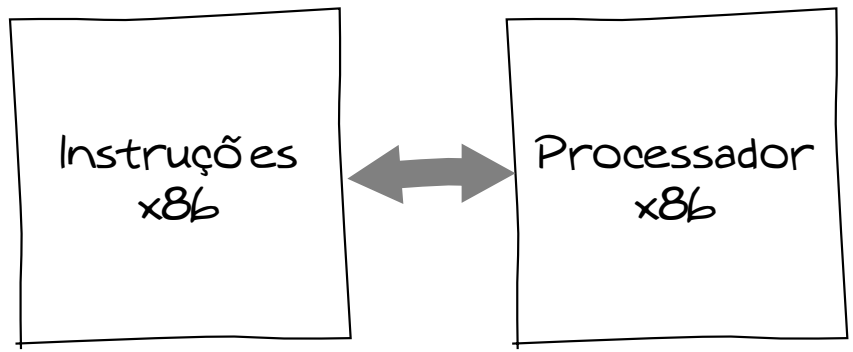
Introdução

- 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

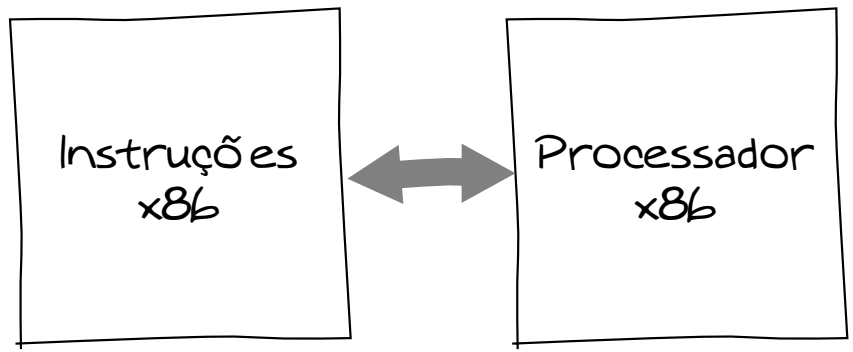
- O que é o conjunto de instruções da arquitetura?
  - Um conjunto de instruções da arquitetura é o idioma que um computador é capaz de interpretar e executar o comportamento



  - 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

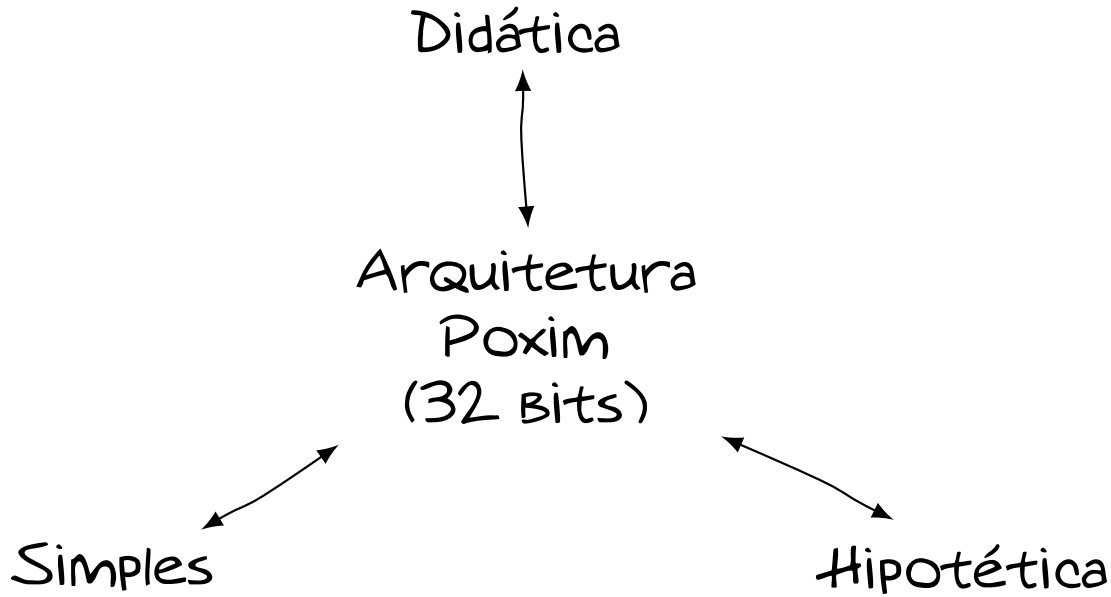
Instruction Set Architecture (ISA)

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



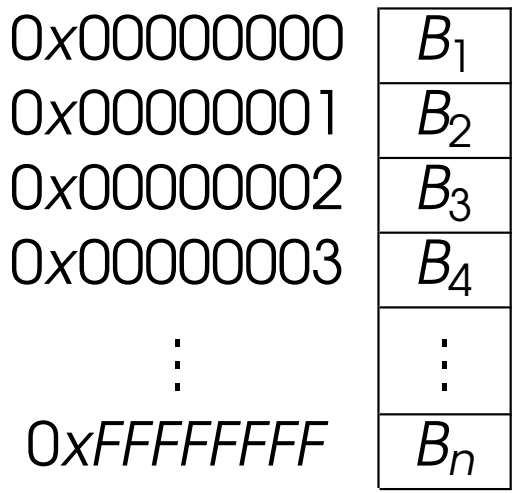
Introdução

- Como as informações são representadas?
  - Cada *nibble* (4 bits) corresponde a um dígito hexadecimal (base 16, notação 0*x*)

Decimal	Binário	Hexadecimal
0_{10}	0000_2	0_{16}
1_{10}	0001_2	1_{16}
2_{10}	0010_2	2_{16}
3_{10}	0011_2	3_{16}
⋮	⋮	⋮
12_{10}	1100_2	C_{16}
13_{10}	1101_2	D_{16}
14_{10}	1110_2	E_{16}
15_{10}	1111_2	F_{16}

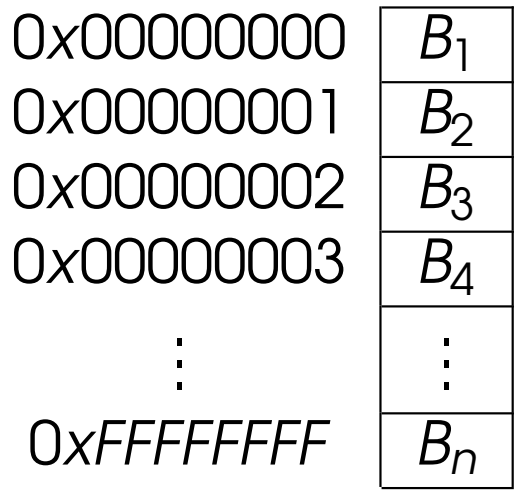
Introdução

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



Introdução

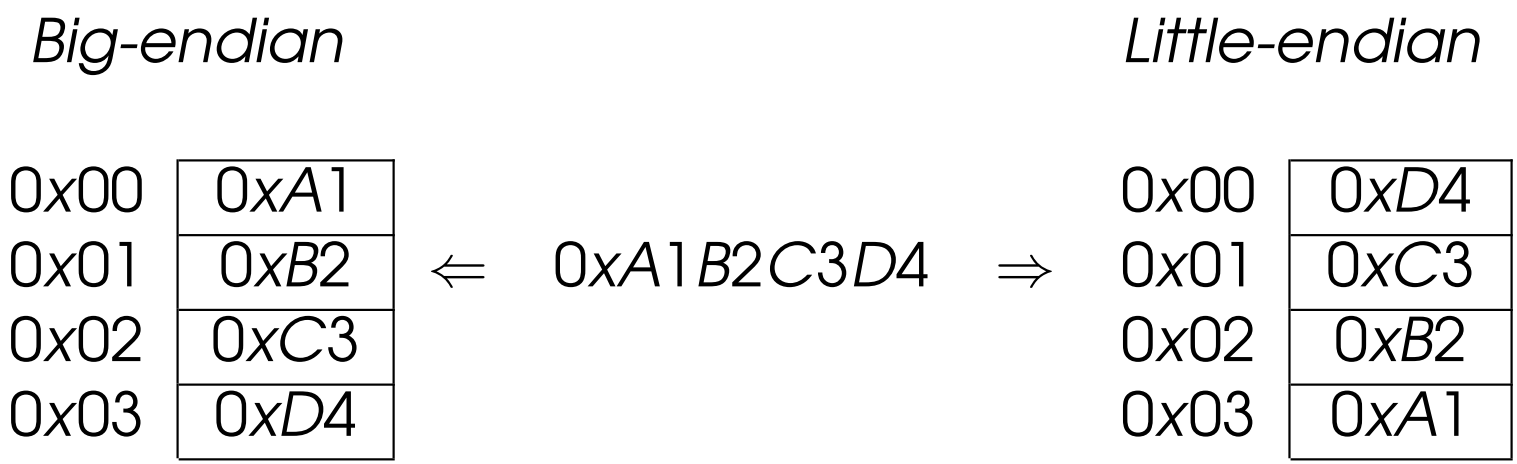
- Como é organizada a memória?
  - Os dados são divididos em bytes



$$4\ GiB \longrightarrow n = 2^{32} = 4.294.967.296\ bytes$$

Introdução

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



Introdução

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

Memória de 4 GiB

0x00000000	B_1	B_2
0x00000002	B_3	B_4
⋮	⋮	⋮
0xFFFFFFFC	B_{n-3}	B_{n-2}
0xFFFFFFFE	B_{n-1}	B_n

16 bits (2 bytes)

Introdução

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

Memória de 4 GiB

0x00000000	B_1	B_2	B_3	B_4
0x00000004	B_5	B_6	B_7	B_8
⋮	⋮	⋮	⋮	⋮
0xFFFFFFF8	B_{n-7}	B_{n-6}	B_{n-5}	B_{n-4}
0xFFFFFFFC	B_{n-3}	B_{n-2}	B_{n-1}	B_n

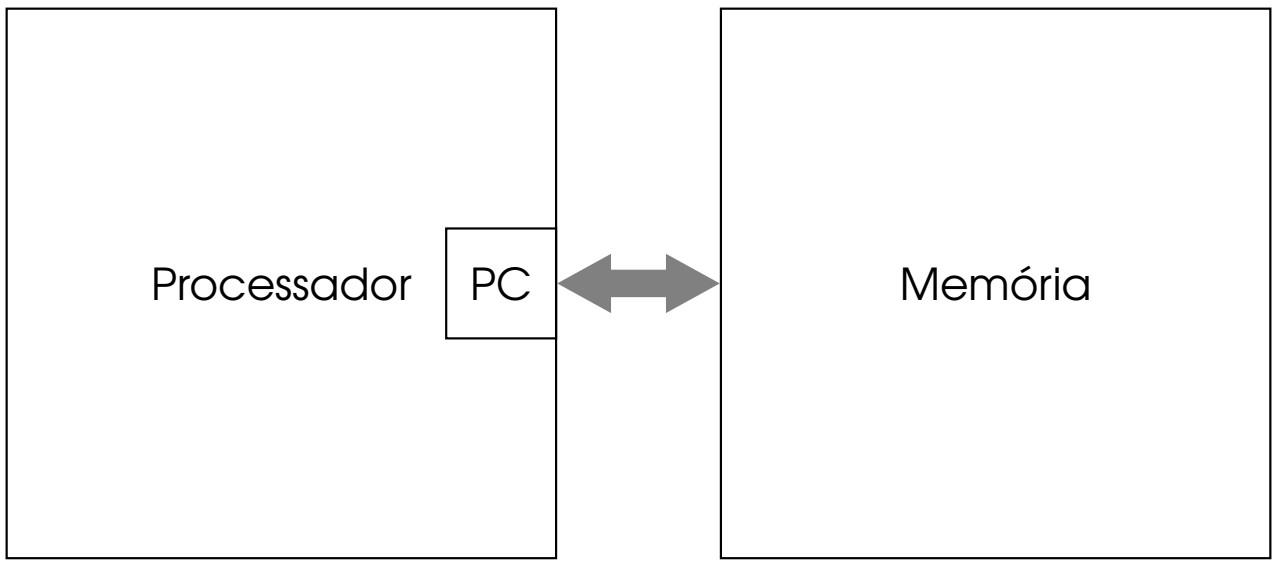
32 bits (4 bytes)

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 }
```

$\Longrightarrow$

```
...
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 }
```

$\Longrightarrow$

```
...
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 }
```

$\Longrightarrow$

```
...
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 }
```

$\Longrightarrow$

```
...
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 }
```

$\Longrightarrow$

```
...
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 }
```

$\Longrightarrow$

```
...
muli x, a, 3
muli 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 }
```

$\Longrightarrow$

```
...
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 }
```

$\Longrightarrow$

```
...
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 }
```

$\Longrightarrow$

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

Operações aritméticas

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

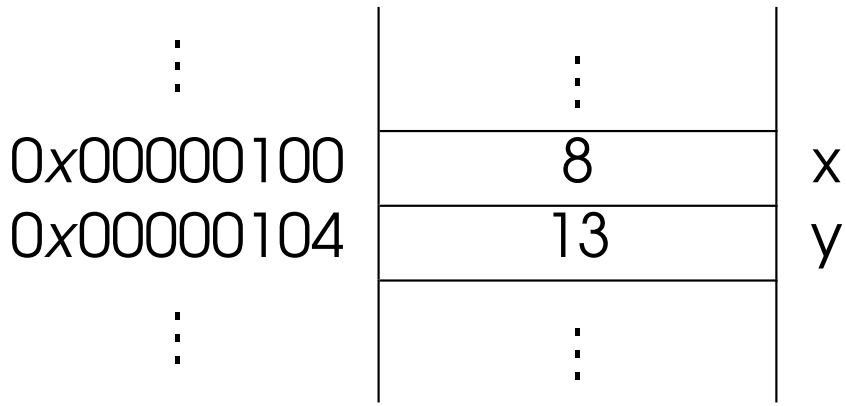
```
1  // Inteiros com tamanho fixo
2  #include <stdint.h>
3  // Função principal
4  int main() {
5      // Variáveis em memória
6      uint32_t x = 8, y = 13;
7      // Operações de aritméticas
8      x = x - 5;
9      y = y / x;
10     // Retorno sem erros
11     return 0;
12 }
```

$\Longrightarrow$

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

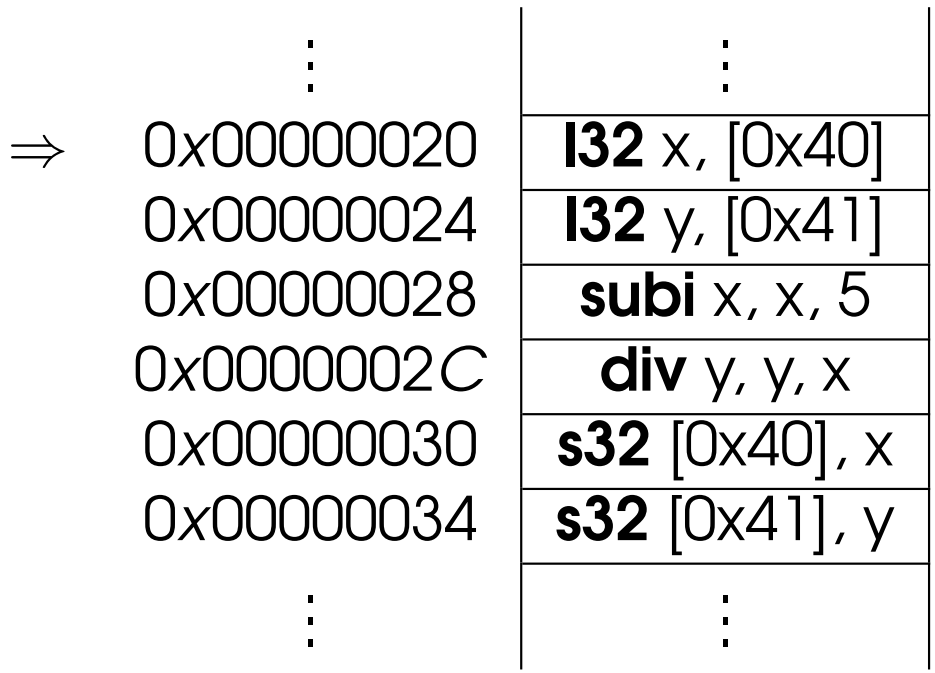
Operações aritméticas

- Instruções com operandos em memória
  - Visualização dos dados



Operações aritméticas

- Instruções com operandos em memória
  - Visualização do programa



Operações aritméticas

- Instruções com operandos em memória
  - Visualização do programa

	⋮	⋮
	0x00000020	**l32** x, [0x40]
⇒	0x00000024	**l32** y, [0x41]
	0x00000028	**subi** x, x, 5
	0x0000002C	**div** y, y, x
	0x00000030	**s32** [0x40], x
	0x00000034	**s32** [0x41], y
	⋮	⋮

Operações aritméticas

- Instruções com operandos em memória
  - Visualização do programa

	⋮	⋮
	0x00000020	**l32** x, [0x40]
	0x00000024	**l32** y, [0x41]
⇒	0x00000028	**subi** x, x, 5
	0x0000002C	**div** y, y, x
	0x00000030	**s32** [0x40], x
	0x00000034	**s32** [0x41], y
	⋮	⋮

Operações aritméticas

- Instruções com operandos em memória
  - Visualização do programa

	⋮	⋮
	0x00000020	**l32** x, [0x40]
	0x00000024	**l32** y, [0x41]
	0x00000028	**subi** x, x, 5
⇒	0x0000002C	**div** y, y, x
	0x00000030	**s32** [0x40], x
	0x00000034	**s32** [0x41], y
	⋮	⋮

Operações aritméticas

- Instruções com operandos em memória
  - Visualização do programa

	⋮	⋮
	0x00000020	**l32** x, [0x40]
	0x00000024	**l32** y, [0x41]
	0x00000028	**subi** x, x, 5
	0x0000002C	**div** y, y, x
⇒	0x00000030	**s32** [0x40], x
	0x00000034	**s32** [0x41], y
	⋮	⋮

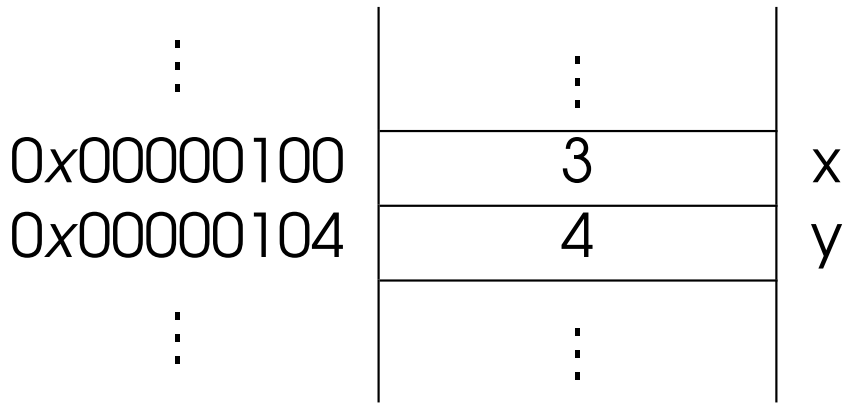
Operações aritméticas

- Instruções com operandos em memória
  - Visualização do programa

	⋮	⋮
	0x00000020	**l32** x, [0x40]
	0x00000024	**l32** y, [0x41]
	0x00000028	**subi** x, x, 5
	0x0000002C	**div** y, y, x
	0x00000030	**s32** [0x40], x
⇒	0x00000034	**s32** [0x41], y
	⋮	⋮

Operações aritméticas

- Instruções com operandos em memória
  - Visualização dos resultados



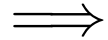
Operações bit a bit

- 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 }
```



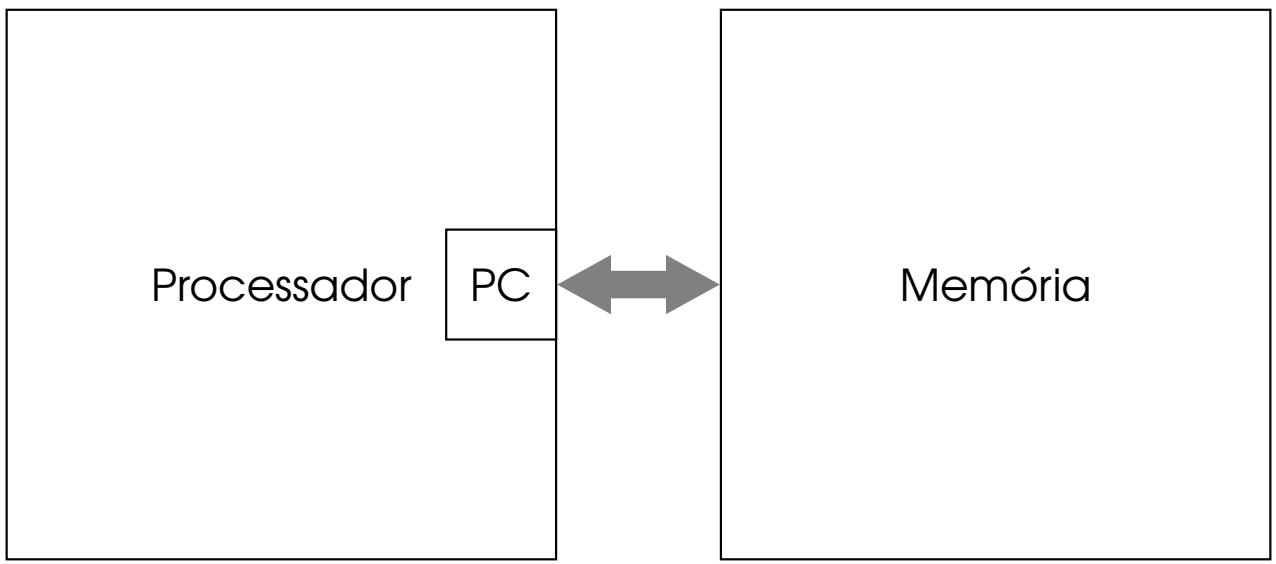
```
...
l32 x, [0x40]
l32 y, [0x41]
and a, x, y
or b, x, y
xor c, x, y
not d, x
srl e, x, y
...
```

Operações de controle de fluxo

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

Operações de controle de fluxo

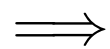
- Chamadas de funções e procedimentos
  - Quando é ligado ou reiniciado, o sistema começa a executar a programação armazenada em memória
  - Cada arquitetura define um valor inicial para o contador de programa (PC) que aponta para as rotinas de inicialização do software (*bootloader*)



Operações de controle de fluxo

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

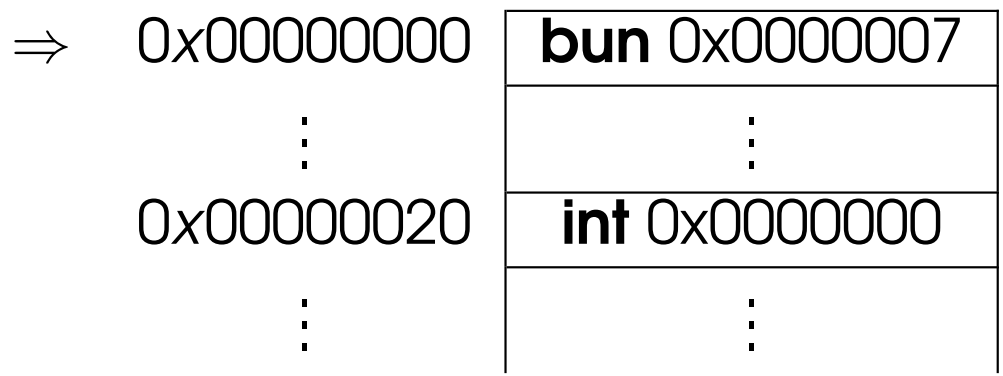
```
1  // Função principal
2  int main() {
3      // Retorno sem erros
4      return 0;
5  }
```



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

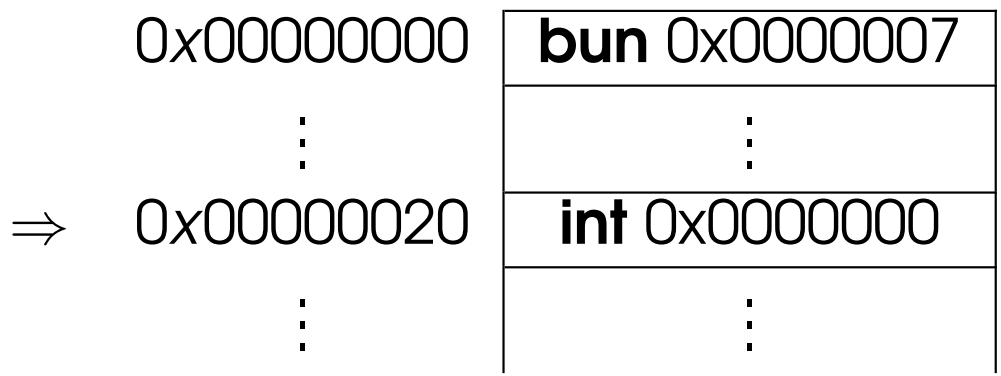
Operações de controle de fluxo

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



Operações de controle de fluxo

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



Operações de controle de fluxo

- Controle condicional
  - Sentença *if-else*

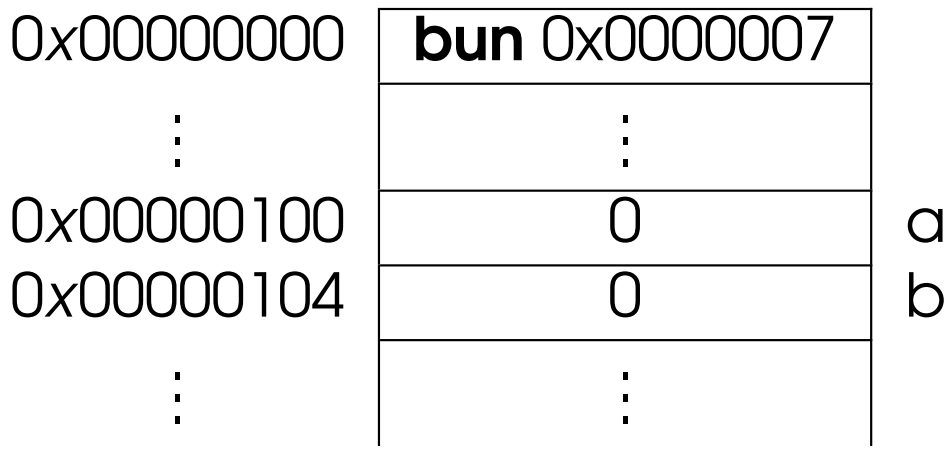
```
1  // Inteiros com tamanho fixo
2  #include <stdint.h>
3  // Função principal
4  int main() {
5      // Variáveis em memória
6      uint32_t a, b;
7      // Controle condicional
8      if(a == b) a = 11;
9      else a = 0;
10     b = a * 5;
11     // Retorno sem erros
12     return 0;
13 }
```

⟹

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

Operações de controle de fluxo

- Controle condicional
  - Visualização dos dados



Operações de controle de fluxo

- Controle condicional
  - Visualização do programa

⇒	0x00000000	**bun** 0x0000007
	⋮	⋮
	0x00000020	**l32** a, [0x40]
	0x00000024	**l32** b, [0x41]
	0x00000028	**cmp** a, b
	0x0000002C	**bne** 0x0000002
	0x00000030	**addi** a, a, 11
	0x00000034	**bun** 0x0000001
	0x00000038	**xor** a, a, a
	0x0000003C	**muli** b, a, 5
	0x00000040	**s32** [0x40], a
	0x00000044	**s32** [0x41], b
	0x00000048	**int** 0x0000000
	⋮	⋮

Operações de controle de fluxo

- Controle condicional
  - Visualização do programa

	0x00000000	**bun** 0x0000007
	⋮	⋮
⇒	0x00000020	**l32** a, [0x40]
	0x00000024	**l32** b, [0x41]
	0x00000028	**cmp** a, b
	0x0000002C	**bne** 0x0000002
	0x00000030	**addi** a, a, 11
	0x00000034	**bun** 0x0000001
	0x00000038	**xor** a, a, a
	0x0000003C	**muli** b, a, 5
	0x00000040	**s32** [0x40], a
	0x00000044	**s32** [0x41], b
	0x00000048	**int** 0x0000000
	⋮	⋮

Operações de controle de fluxo

- Controle condicional
  - Visualização do programa

	0x00000000	**bun** 0x0000007
	⋮	⋮
	0x00000020	**l32** a, [0x40]
⇒	0x00000024	**l32** b, [0x41]
	0x00000028	**cmp** a, b
	0x0000002C	**bne** 0x0000002
	0x00000030	**addi** a, a, 11
	0x00000034	**bun** 0x0000001
	0x00000038	**xor** a, a, a
	0x0000003C	**muli** b, a, 5
	0x00000040	**s32** [0x40], a
	0x00000044	**s32** [0x41], b
	0x00000048	**int** 0x0000000
	⋮	⋮

Operações de controle de fluxo

- Controle condicional
  - Visualização do programa

	0x00000000	**bun** 0x0000007
	⋮	⋮
	0x00000020	**l32** a, [0x40]
	0x00000024	**l32** b, [0x41]
⇒	0x00000028	**cmp** a, b
	0x0000002C	**bne** 0x0000002
	0x00000030	**addi** a, a, 11
	0x00000034	**bun** 0x0000001
	0x00000038	**xor** a, a, a
	0x0000003C	**muli** b, a, 5
	0x00000040	**s32** [0x40], a
	0x00000044	**s32** [0x41], b
	0x00000048	**int** 0x0000000
	⋮	⋮

Operações de controle de fluxo

- Controle condicional
  - Visualização do programa

	0x00000000	**bun** 0x0000007
	⋮	⋮
	0x00000020	**l32** a, [0x40]
	0x00000024	**l32** b, [0x41]
	0x00000028	**cmp** a, b
⇒	0x0000002C	**bne** 0x0000002
	0x00000030	**addi** a, a, 11
	0x00000034	**bun** 0x0000001
	0x00000038	**xor** a, a, a
	0x0000003C	**muli** b, a, 5
	0x00000040	**s32** [0x40], a
	0x00000044	**s32** [0x41], b
	0x00000048	**int** 0x0000000
	⋮	⋮

Operações de controle de fluxo

- Controle condicional
  - Visualização do programa

	0x00000000	**bun** 0x0000007
	⋮	⋮
	0x00000020	**l32** a, [0x40]
	0x00000024	**l32** b, [0x41]
	0x00000028	**cmp** a, b
	0x0000002C	**bne** 0x0000002
⇒	0x00000030	**addi** a, a, 11
	0x00000034	**bun** 0x0000001
	0x00000038	**xor** a, a, a
	0x0000003C	**muli** b, a, 5
	0x00000040	**s32** [0x40], a
	0x00000044	**s32** [0x41], b
	0x00000048	**int** 0x0000000
	⋮	⋮

Operações de controle de fluxo

- Controle condicional
  - Visualização do programa

	0x00000000	**bun** 0x0000007
	⋮	⋮
	0x00000020	**l32** a, [0x40]
	0x00000024	**l32** b, [0x41]
	0x00000028	**cmp** a, b
	0x0000002C	**bne** 0x0000002
	0x00000030	**addi** a, a, 11
⇒	0x00000034	**bun** 0x0000001
	0x00000038	**xor** a, a, a
	0x0000003C	**muli** b, a, 5
	0x00000040	**s32** [0x40], a
	0x00000044	**s32** [0x41], b
	0x00000048	**int** 0x0000000
	⋮	⋮

Operações de controle de fluxo

- Controle condicional
  - Visualização do programa

	0x00000000	**bun** 0x0000007
	⋮	⋮
	0x00000020	**l32** a, [0x40]
	0x00000024	**l32** b, [0x41]
	0x00000028	**cmp** a, b
	0x0000002C	**bne** 0x0000002
	0x00000030	**addi** a, a, 11
	0x00000034	**bun** 0x0000001
	0x00000038	**xor** a, a, a
⇒	0x0000003C	**muli** b, a, 5
	0x00000040	**s32** [0x40], a
	0x00000044	**s32** [0x41], b
	0x00000048	**int** 0x0000000
	⋮	⋮

Operações de controle de fluxo

- Controle condicional
  - Visualização do programa

	0x00000000	**bun** 0x0000007
	⋮	⋮
	0x00000020	**l32** a, [0x40]
	0x00000024	**l32** b, [0x41]
	0x00000028	**cmp** a, b
	0x0000002C	**bne** 0x0000002
	0x00000030	**addi** a, a, 11
	0x00000034	**bun** 0x0000001
	0x00000038	**xor** a, a, a
	0x0000003C	**muli** b, a, 5
⇒	0x00000040	**s32** [0x40], a
	0x00000044	**s32** [0x41], b
	0x00000048	**int** 0x0000000
	⋮	⋮

Operações de controle de fluxo

- Controle condicional
  - Visualização do programa

	0x00000000	**bun** 0x0000007
	⋮	⋮
	0x00000020	**l32** a, [0x40]
	0x00000024	**l32** b, [0x41]
	0x00000028	**cmp** a, b
	0x0000002C	**bne** 0x0000002
	0x00000030	**addi** a, a, 11
	0x00000034	**bun** 0x0000001
	0x00000038	**xor** a, a, a
	0x0000003C	**muli** b, a, 5
	0x00000040	**s32** [0x40], a
⇒	0x00000044	**s32** [0x41], b
	0x00000048	**int** 0x0000000
	⋮	⋮

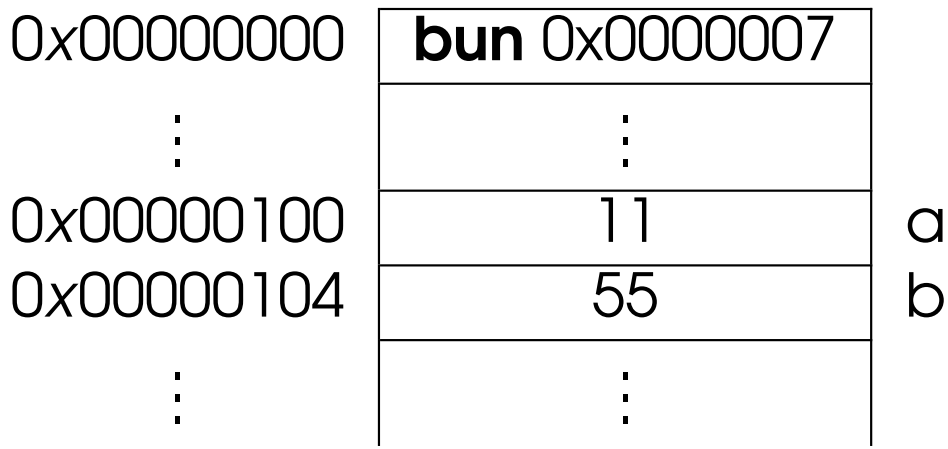
Operações de controle de fluxo

- Controle condicional
  - Visualização do programa

	0x00000000	**bun** 0x0000007
	⋮	⋮
	0x00000020	**l32** a, [0x40]
	0x00000024	**l32** b, [0x41]
	0x00000028	**cmp** a, b
	0x0000002C	**bne** 0x0000002
	0x00000030	**addi** a, a, 11
	0x00000034	**bun** 0x0000001
	0x00000038	**xor** a, a, a
	0x0000003C	**muli** b, a, 5
	0x00000040	**s32** [0x40], a
	0x00000044	**s32** [0x41], b
⇒	0x00000048	**int** 0x0000000
	⋮	⋮

Operações de controle de fluxo

- Controle condicional
  - Visualização dos resultados



Operações de controle de fluxo

- Controle iterativo
  - Sentença *while*

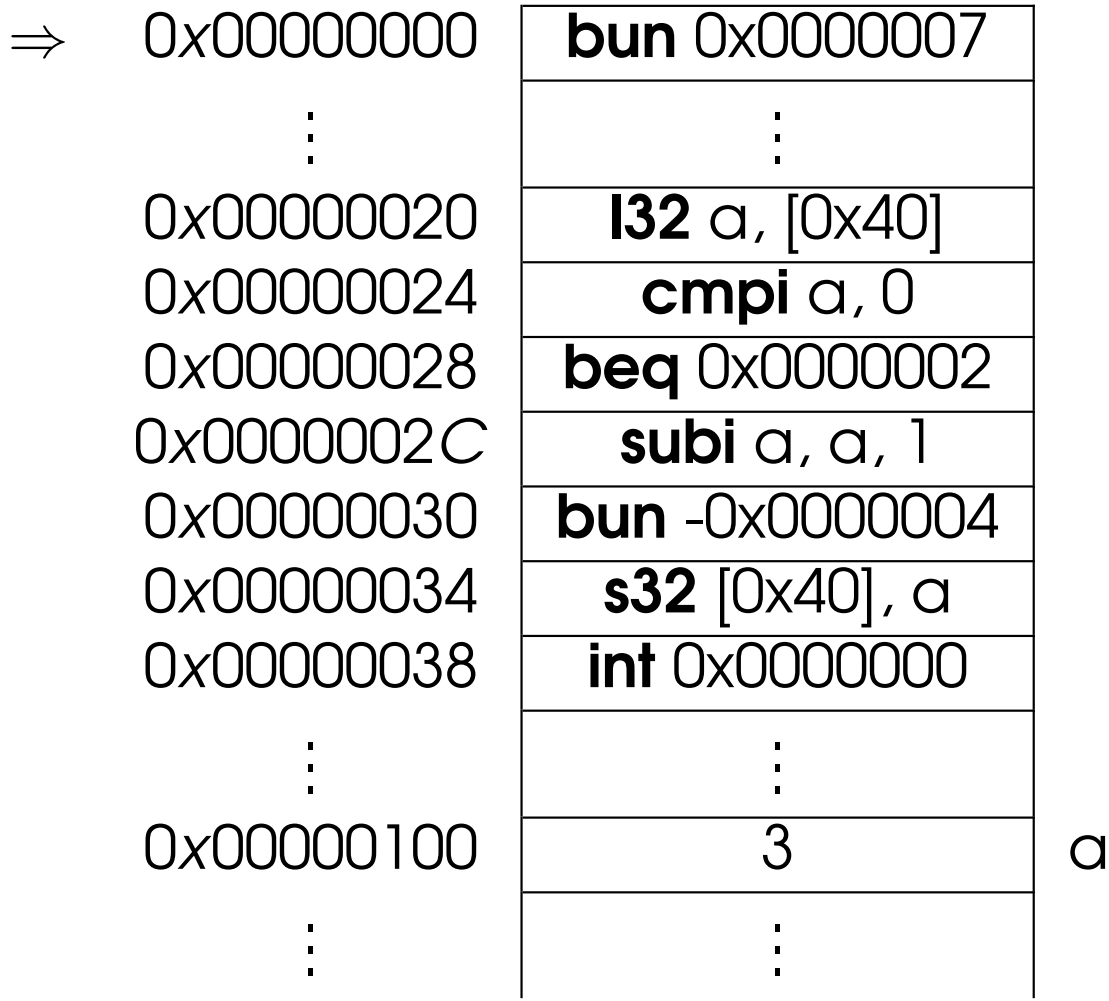
```
1  // Inteiros com tamanho fixo
2  #include <stdint.h>
3  // Função principal
4  int main() {
5      // Variável em memória
6      uint32_t a = 3;
7      // Controle iterativo
8      while(a > 0) {
9          a--;
10     }
11     // Retorno sem erros
12     return 0;
13 }
```

$\Longrightarrow$

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

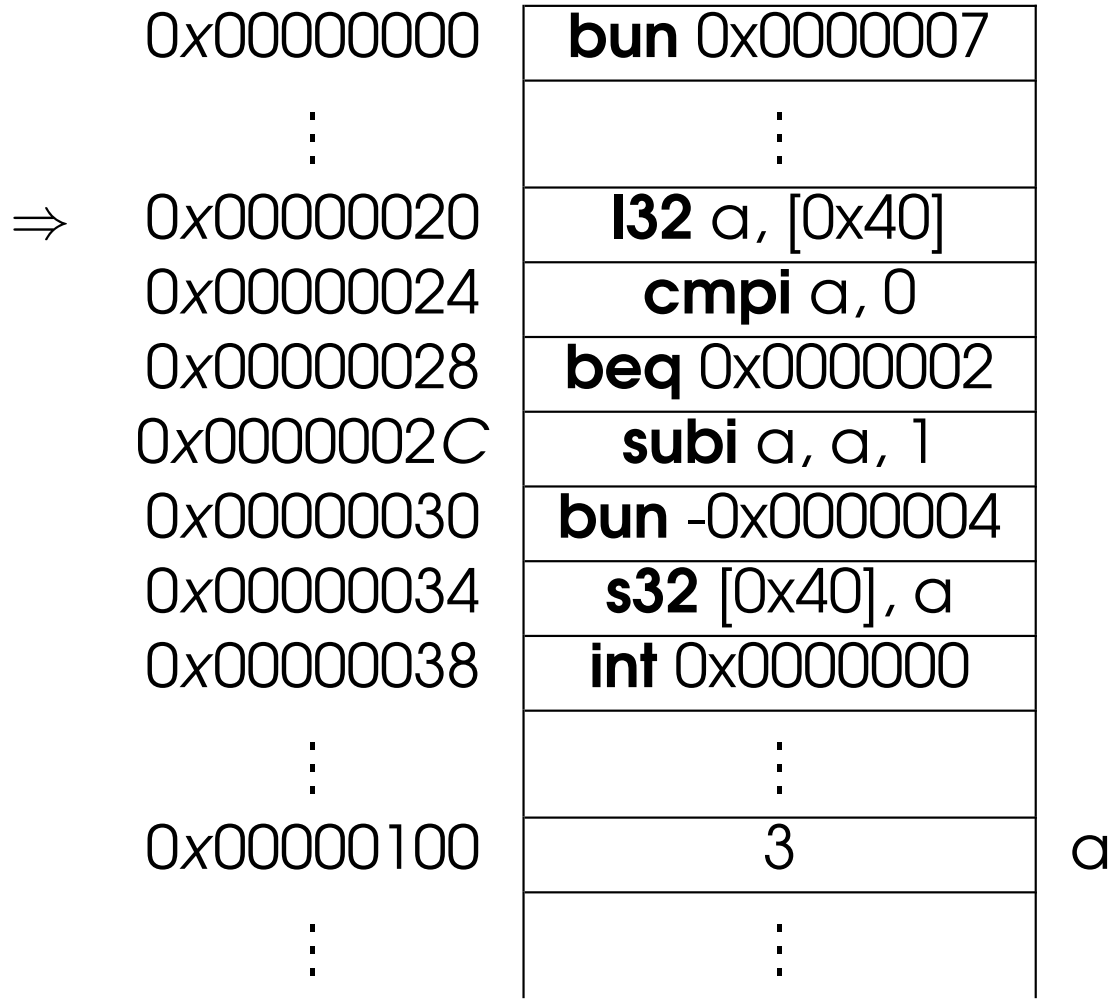
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



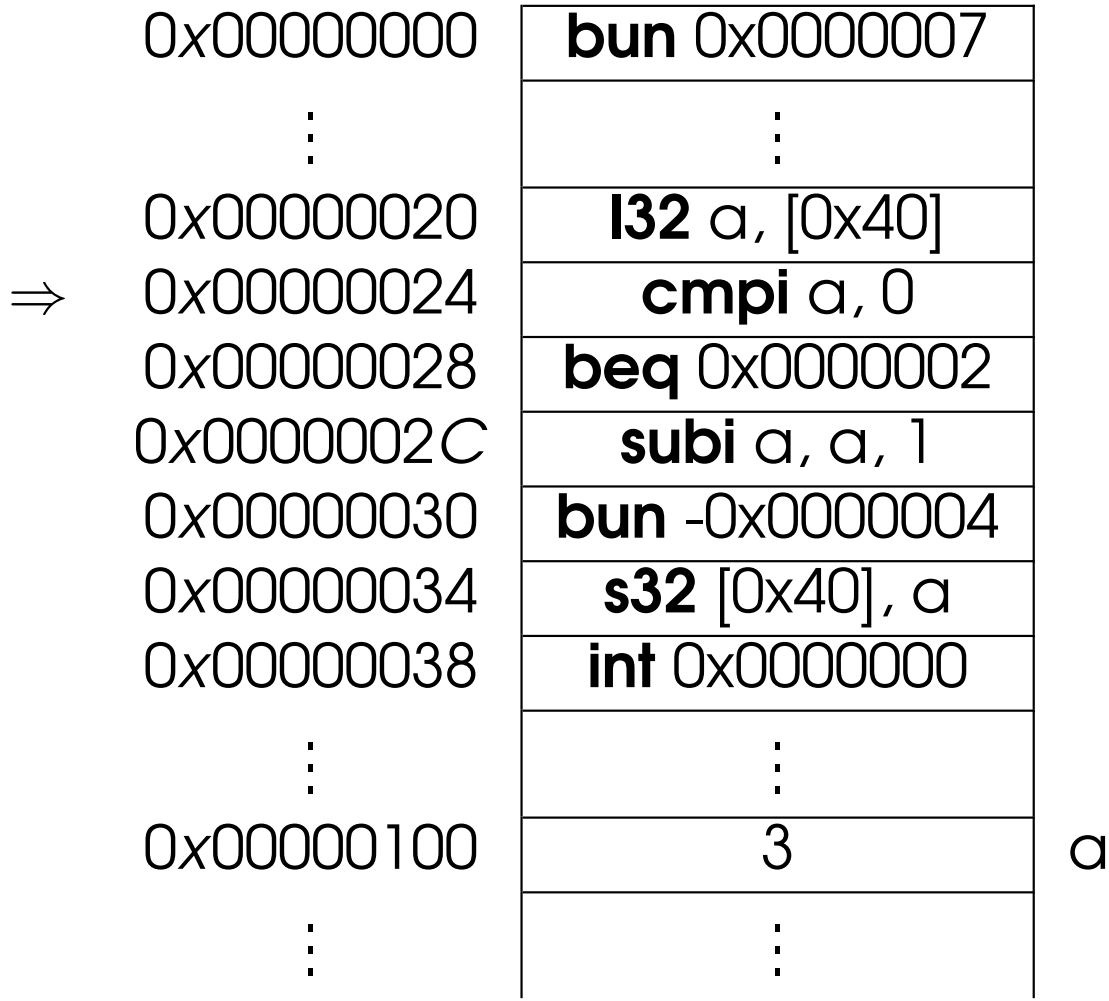
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



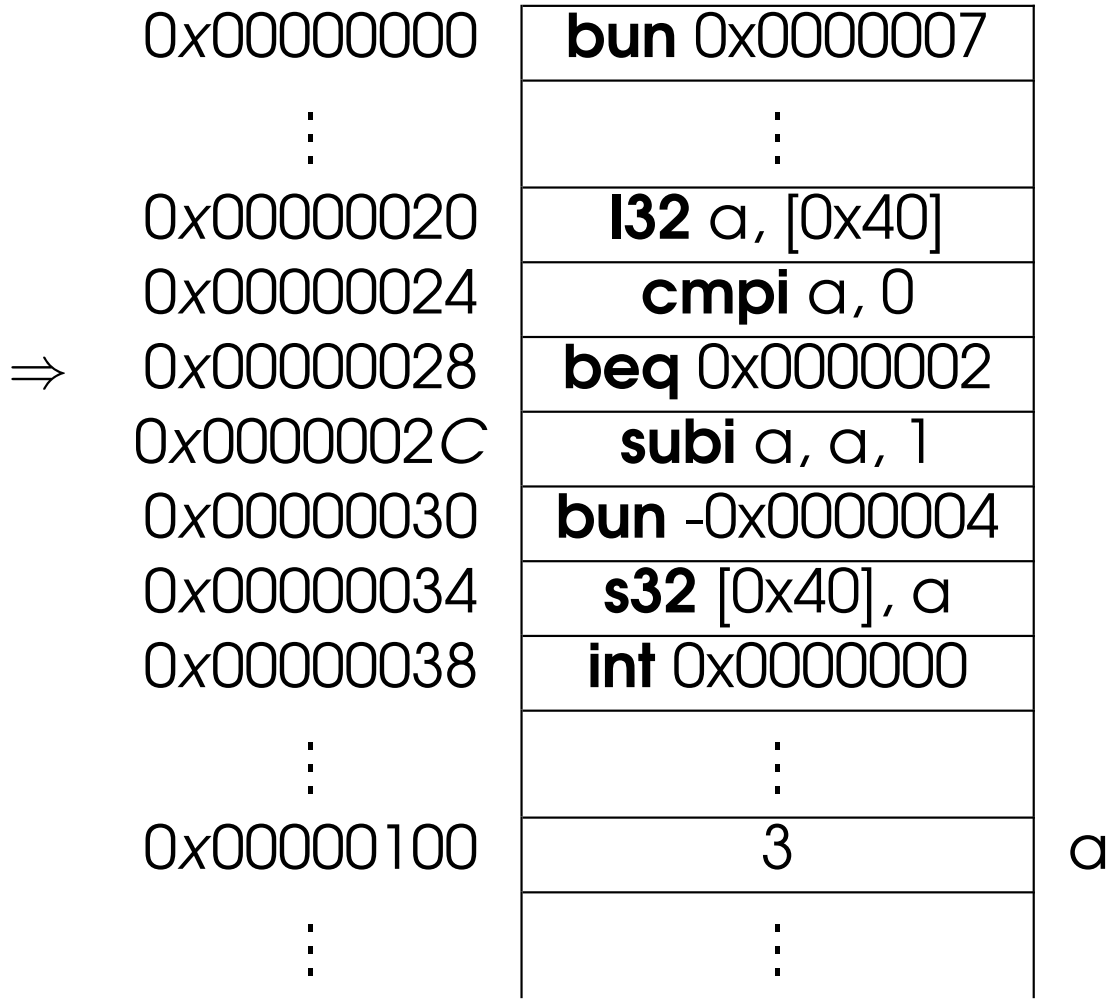
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



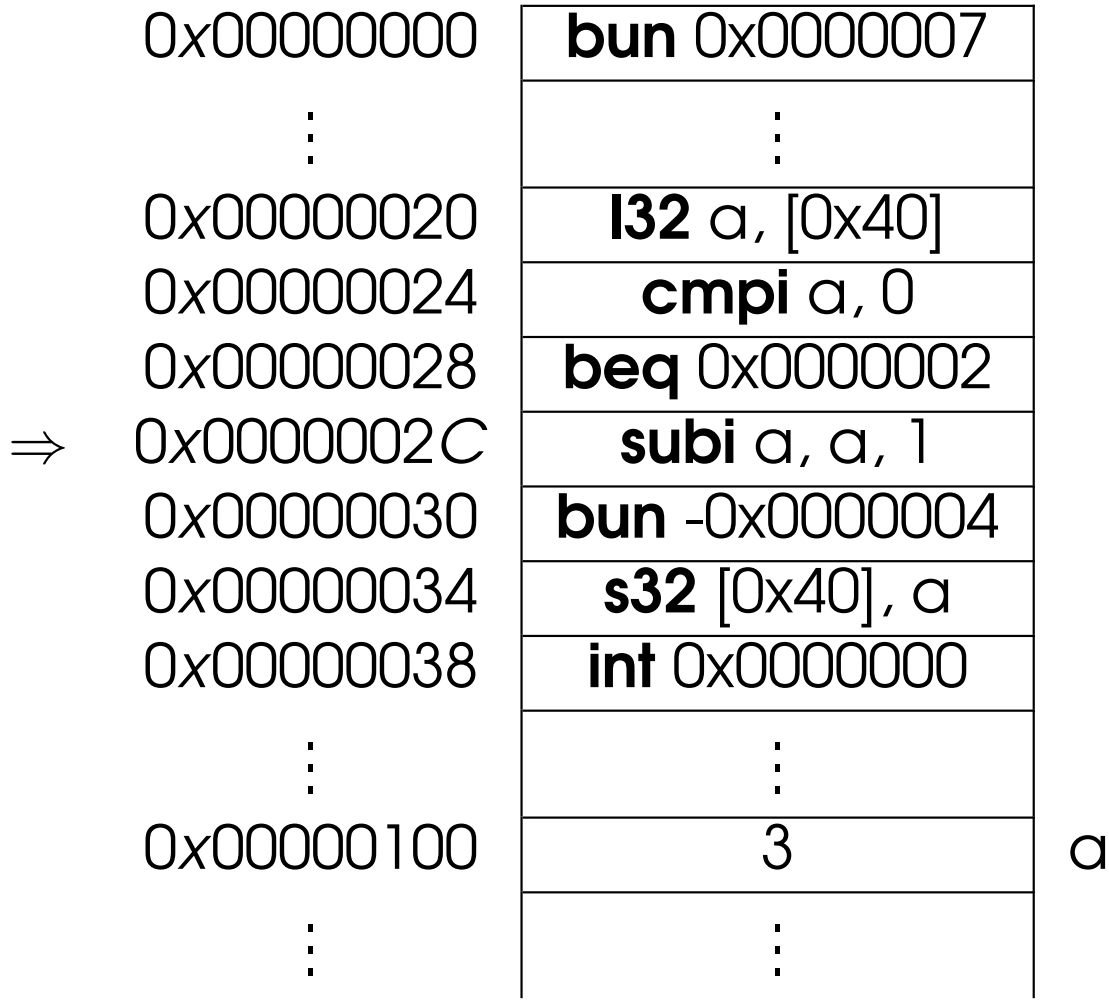
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



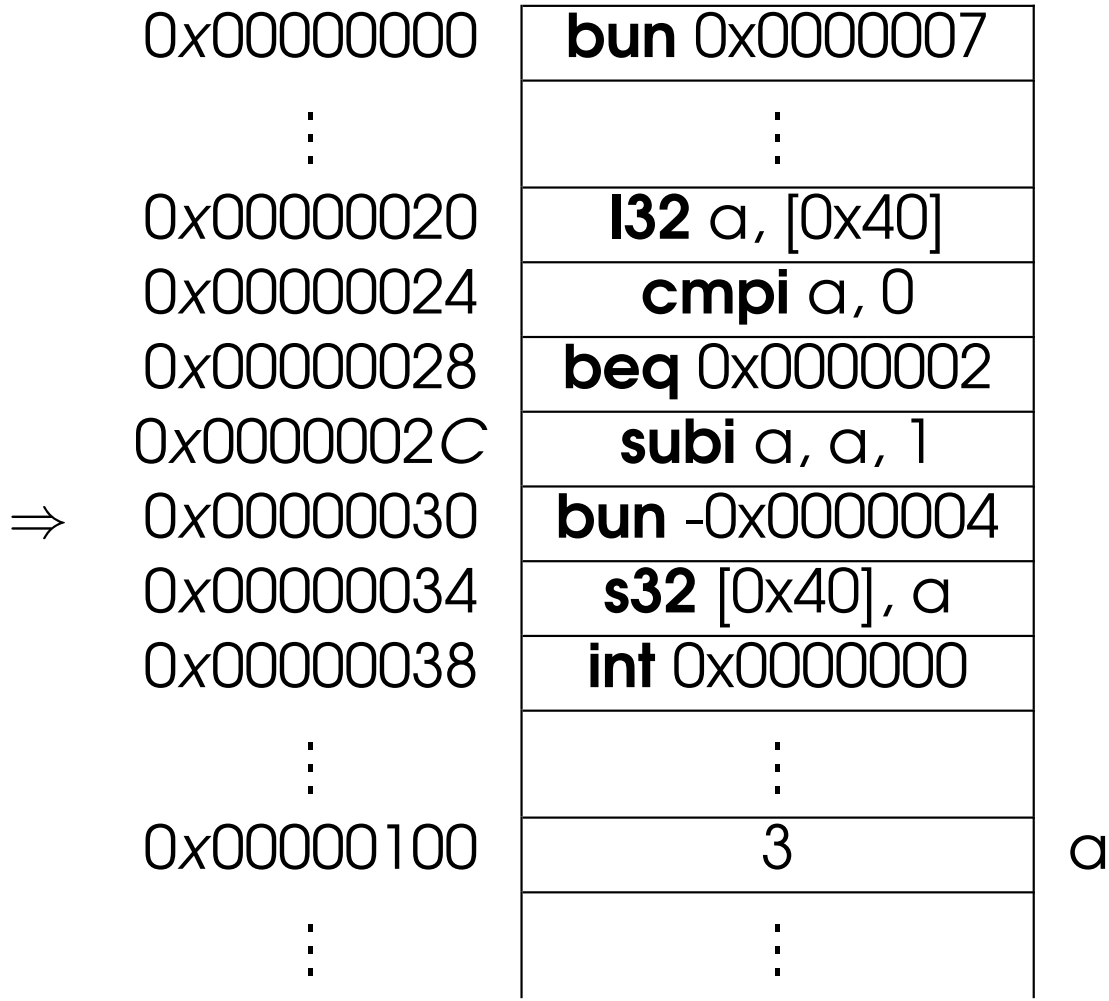
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



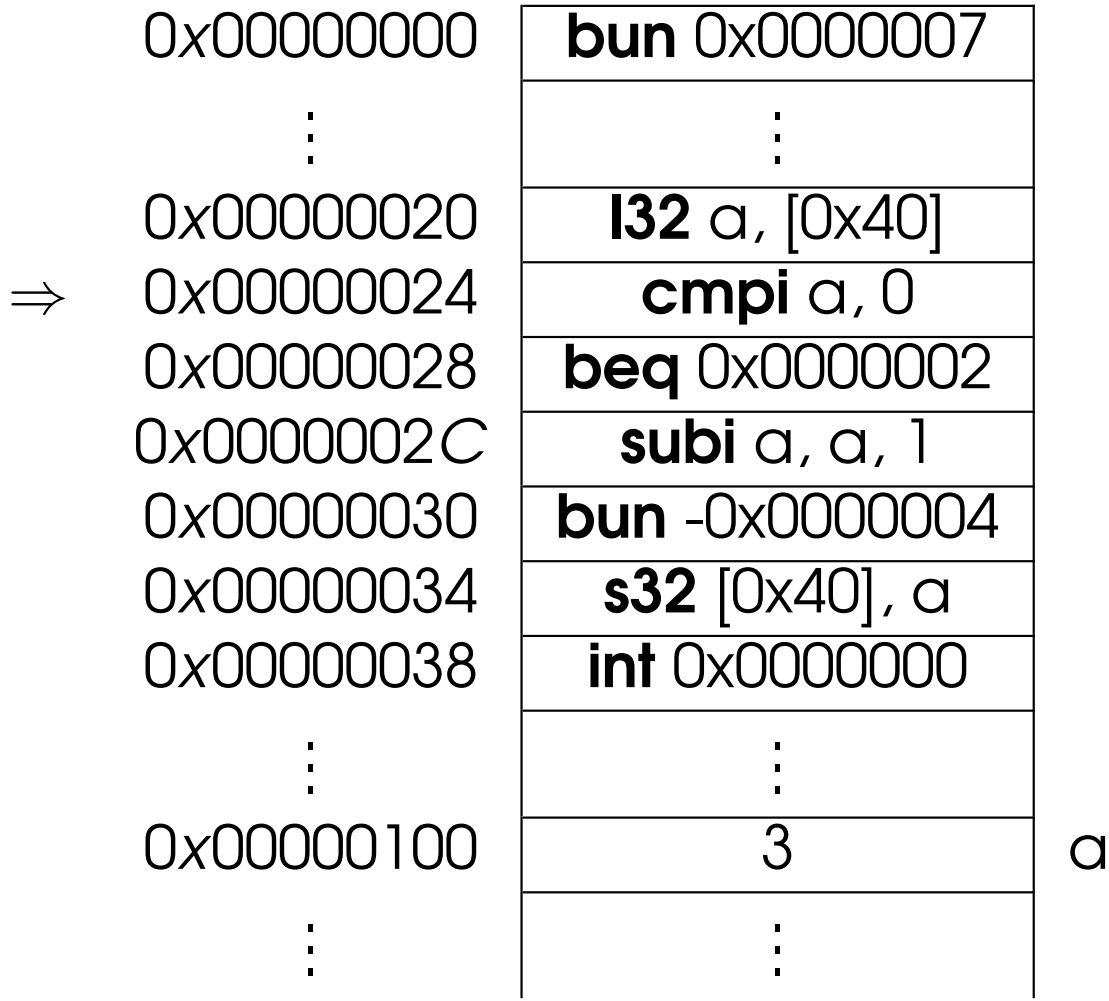
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



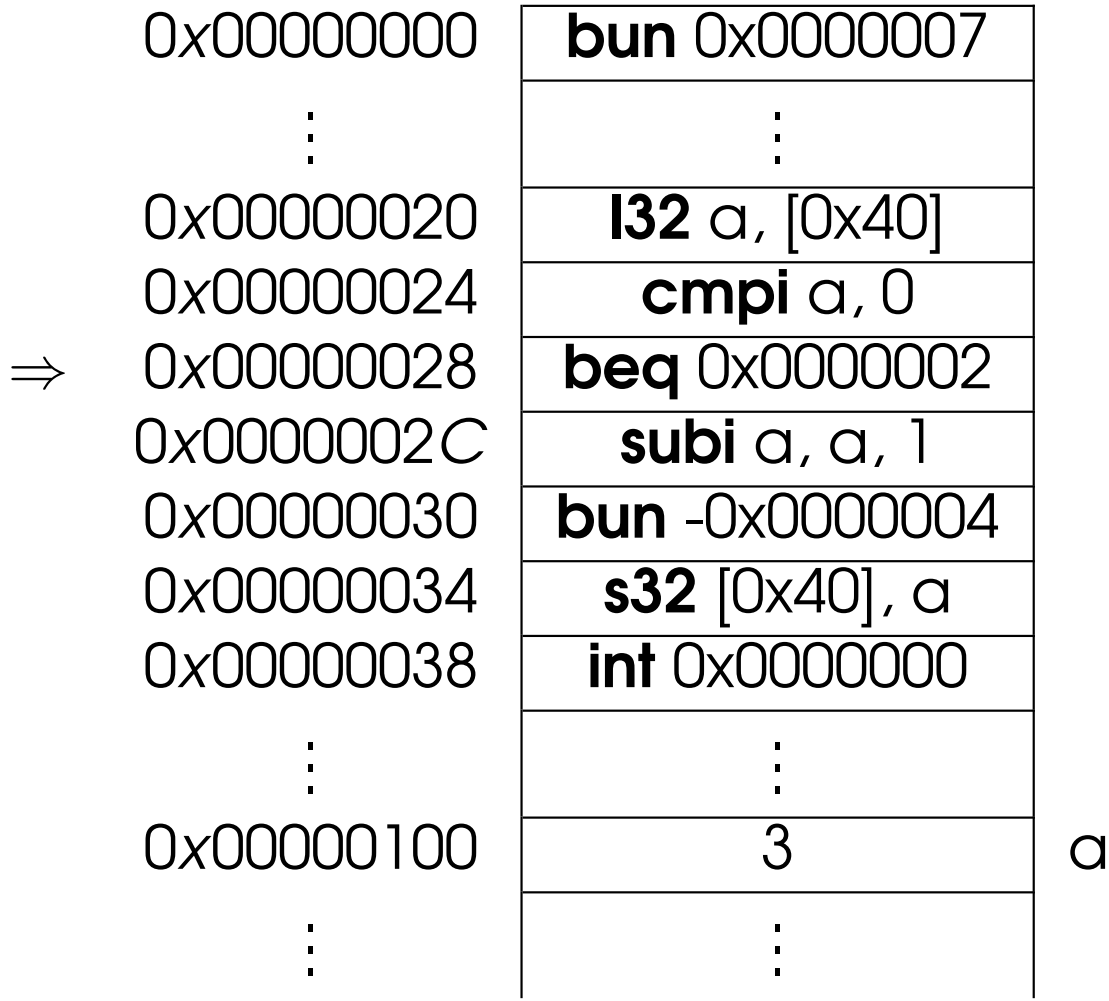
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



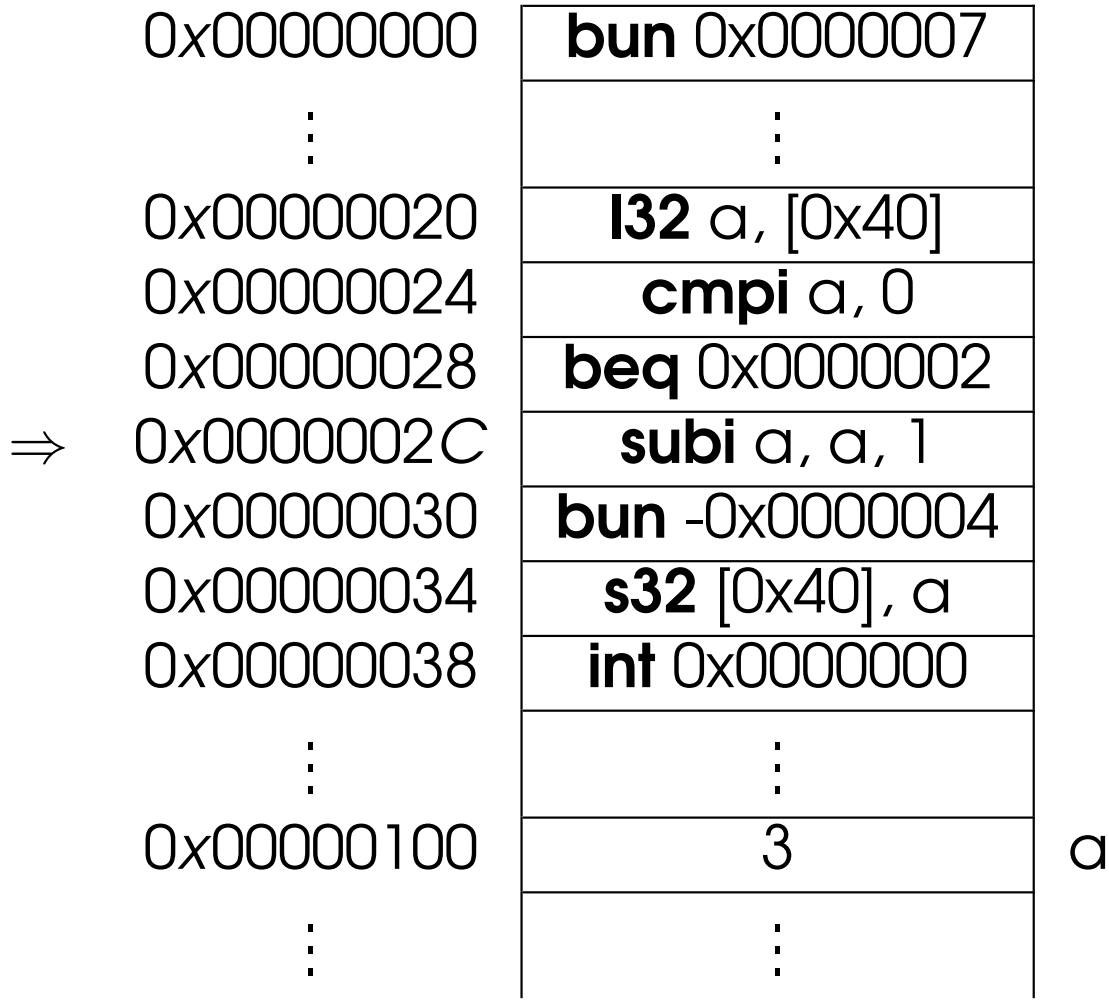
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



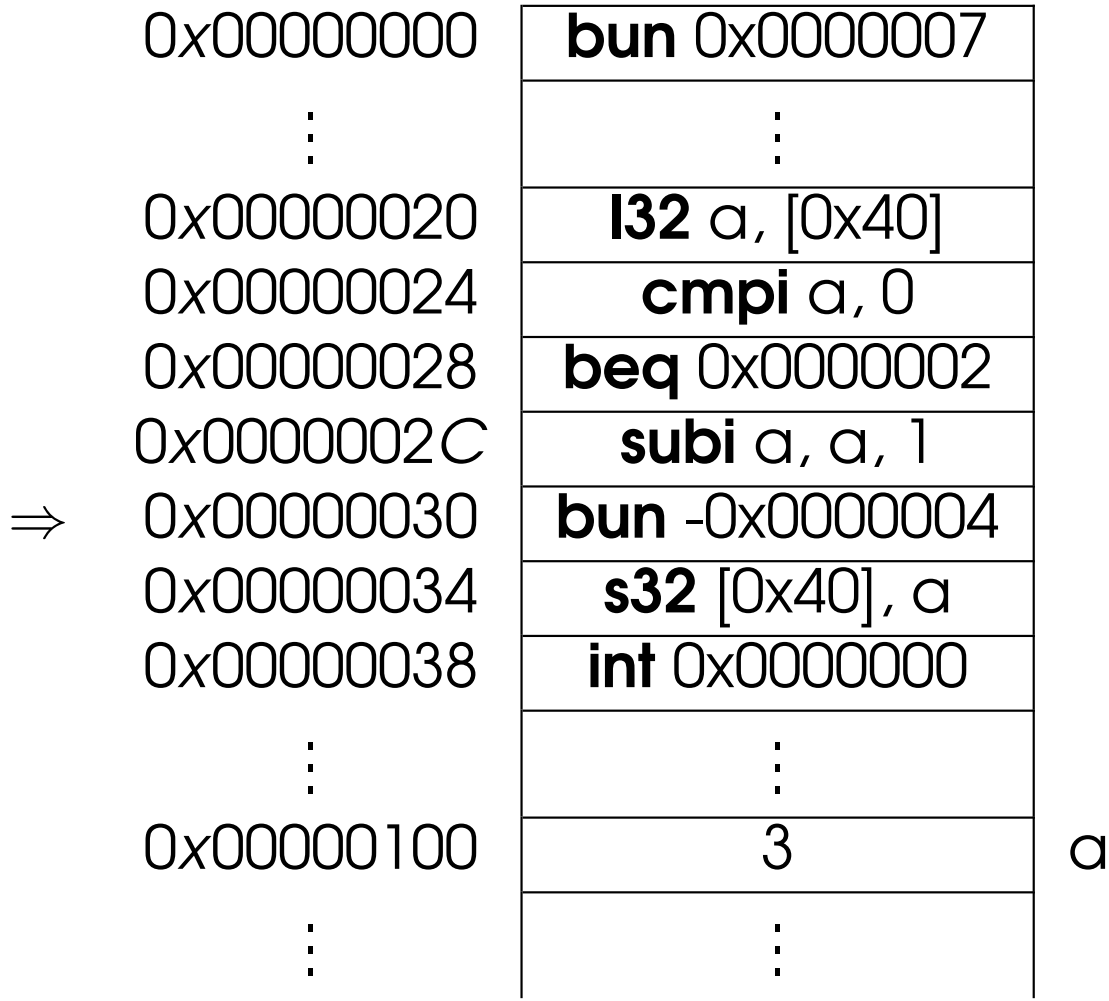
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



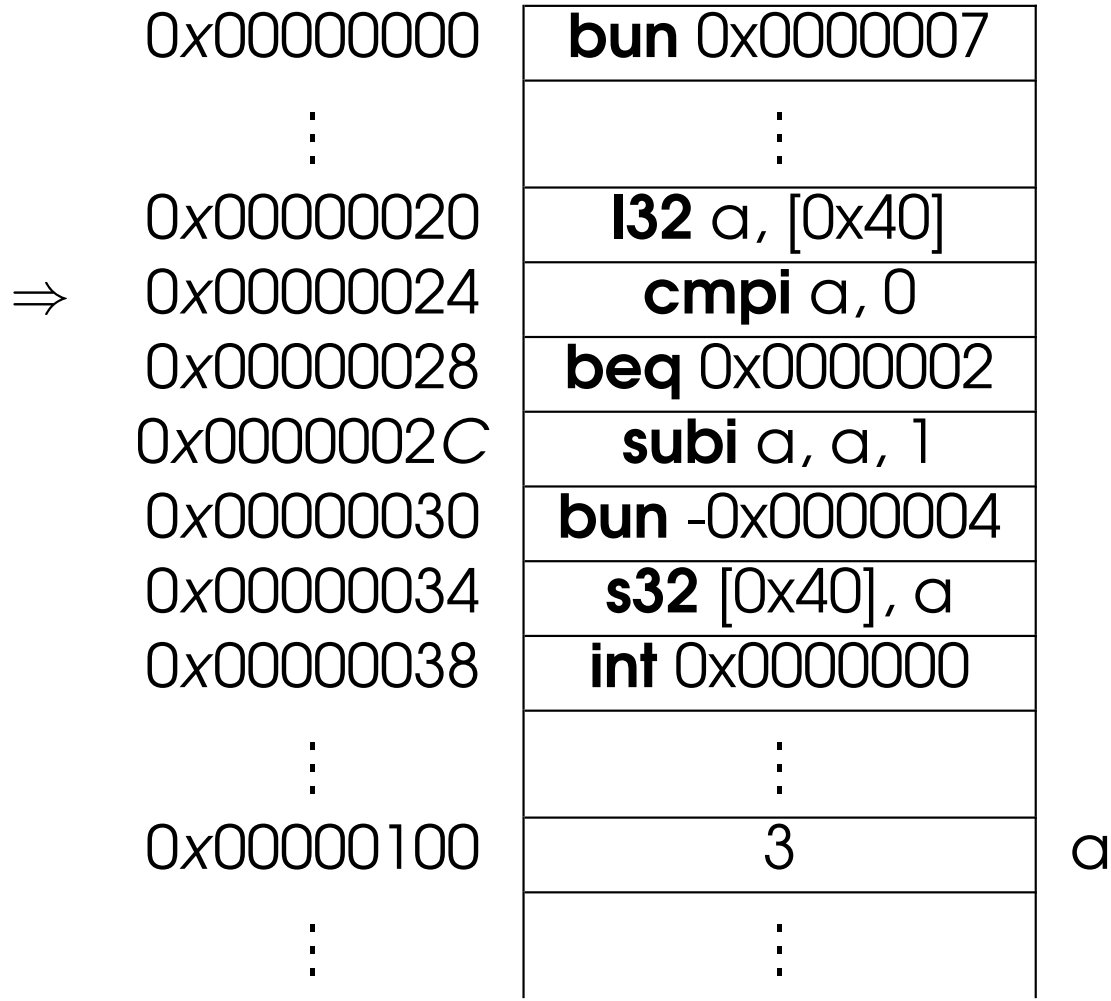
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



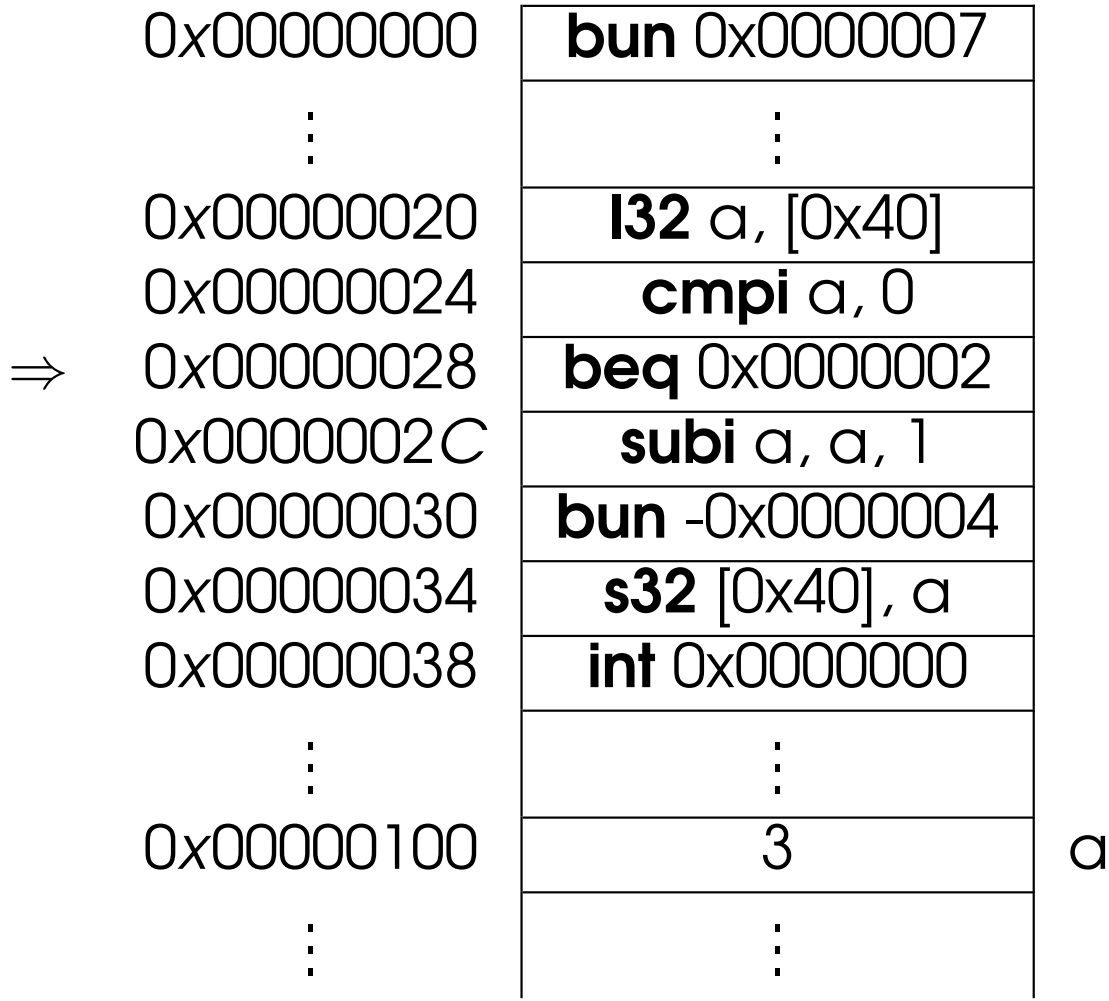
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



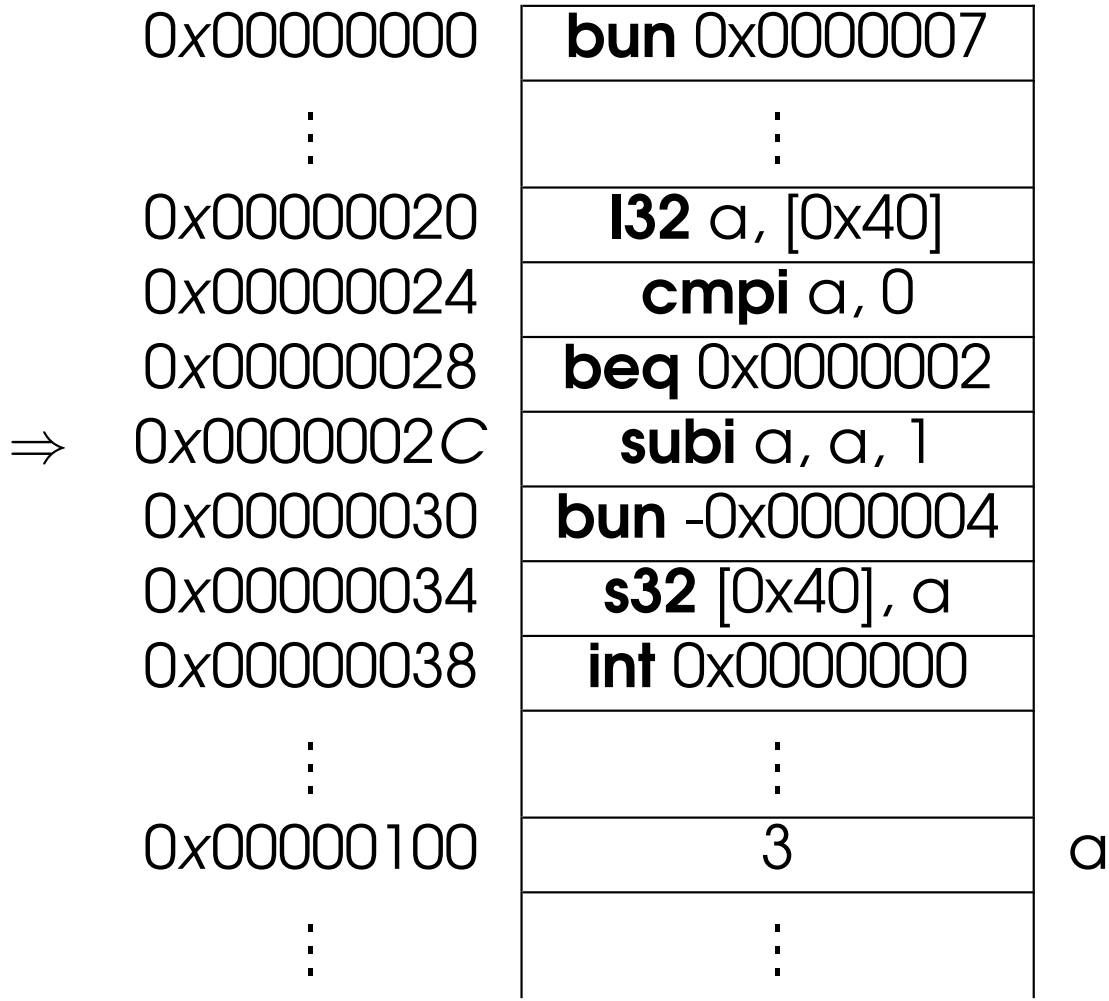
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



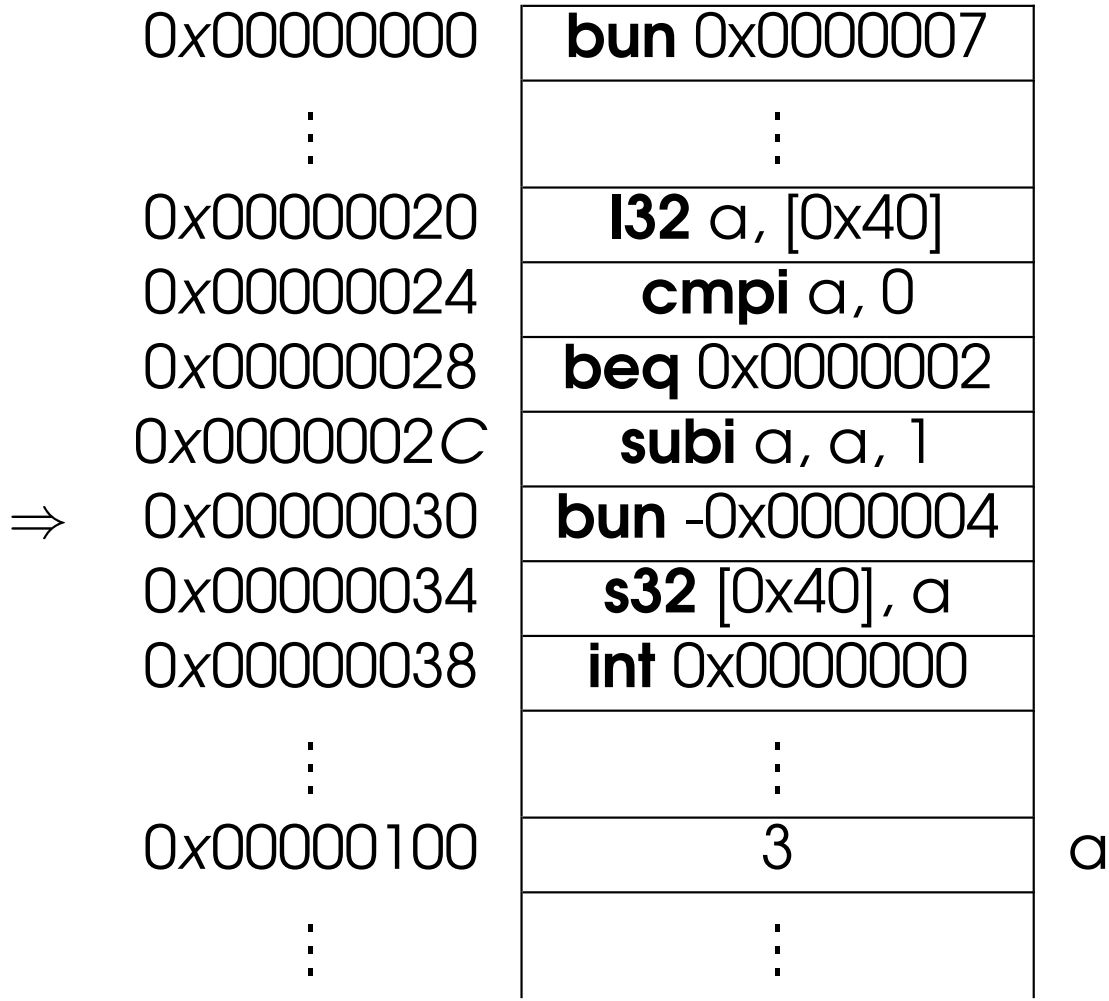
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



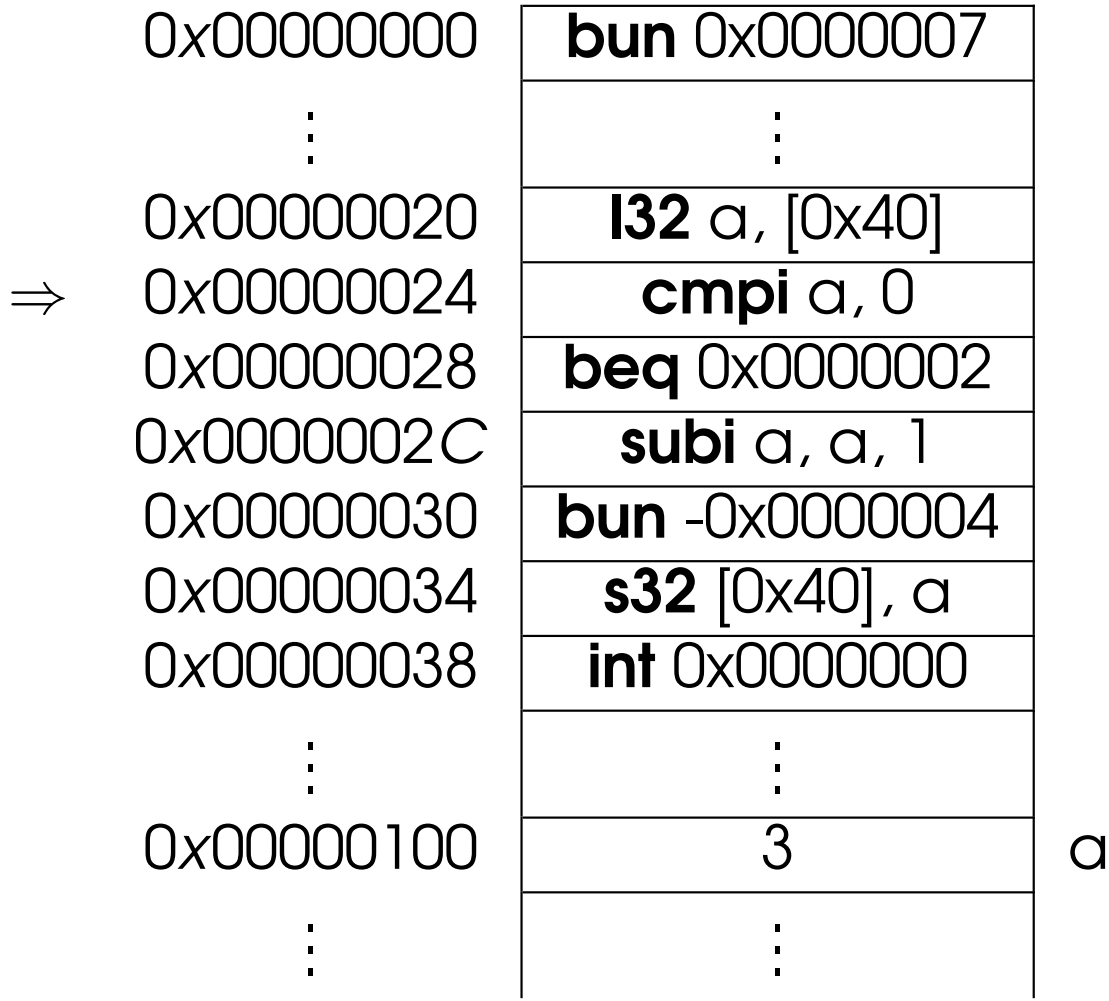
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



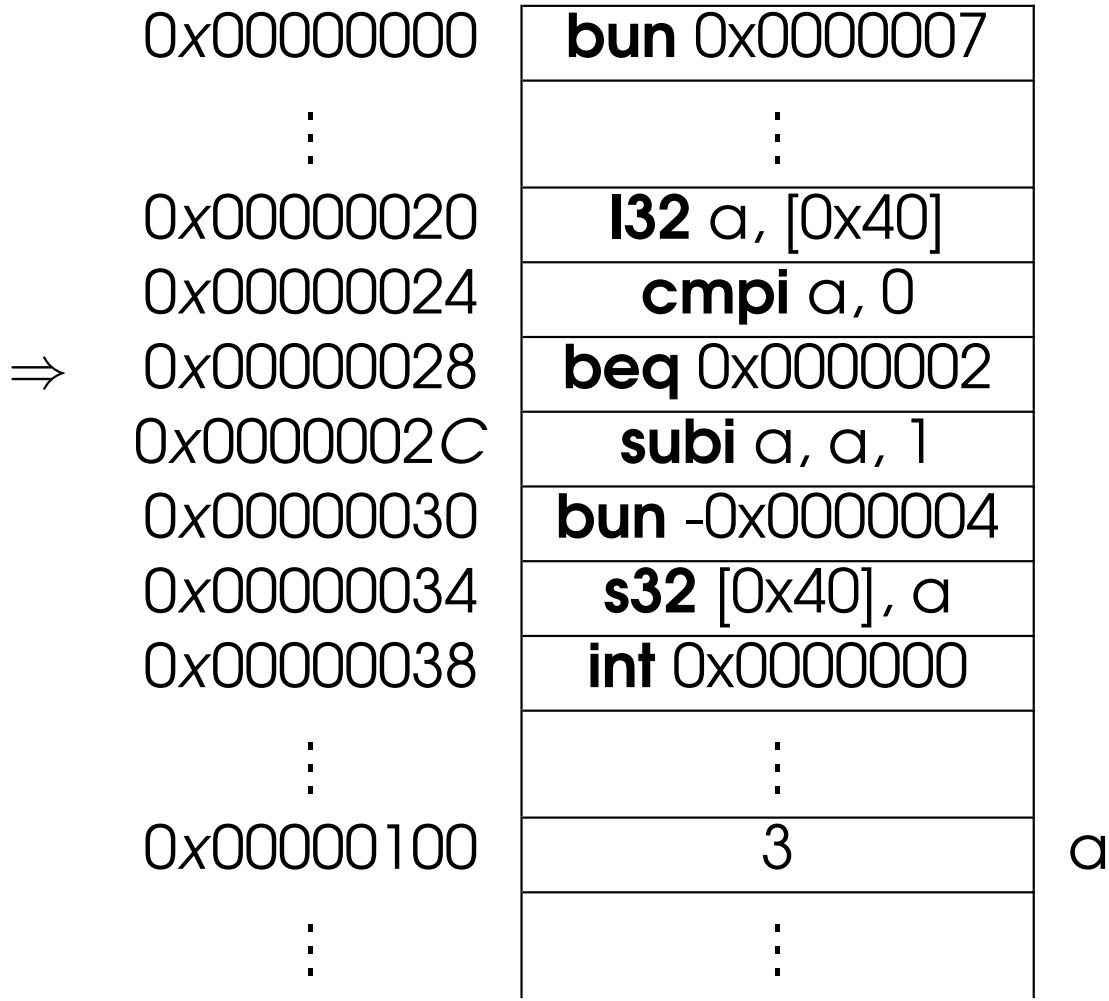
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



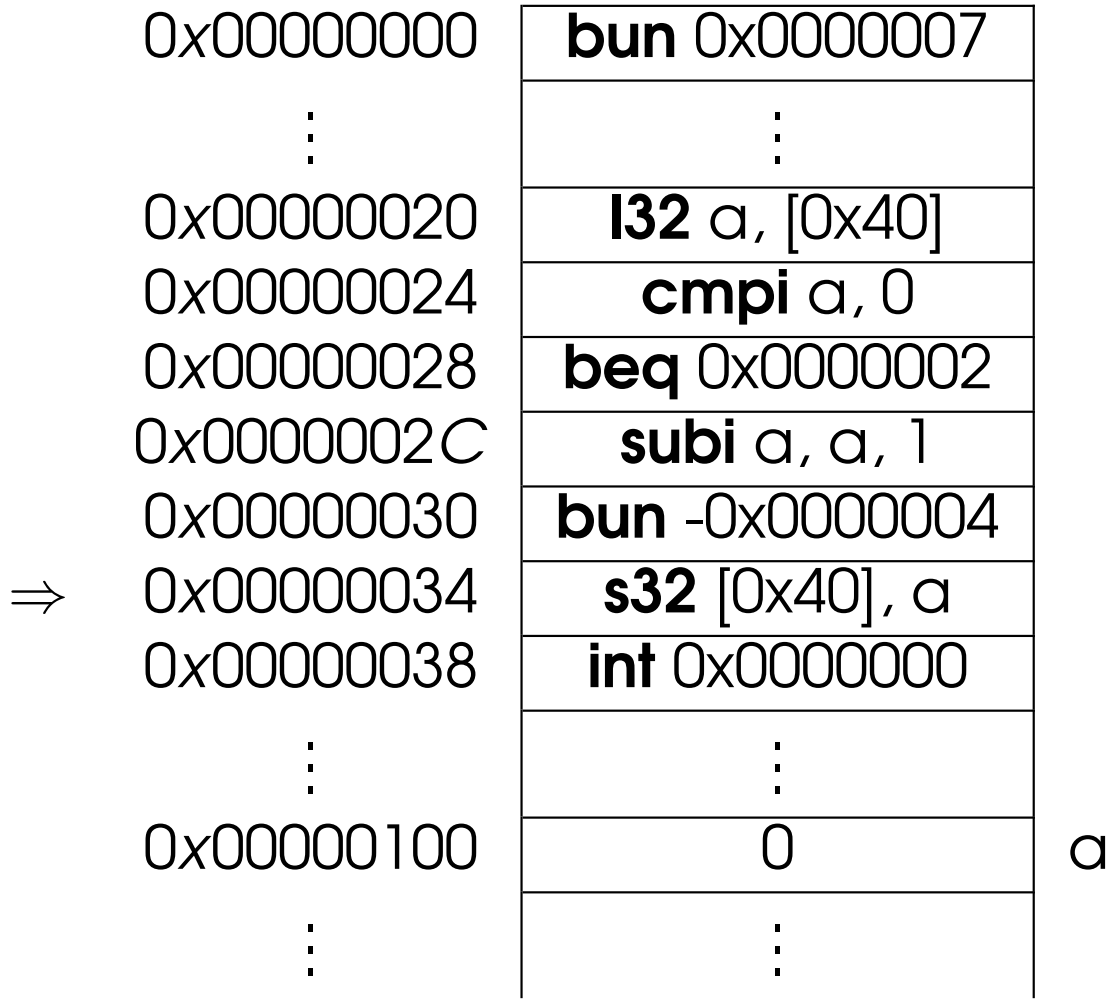
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



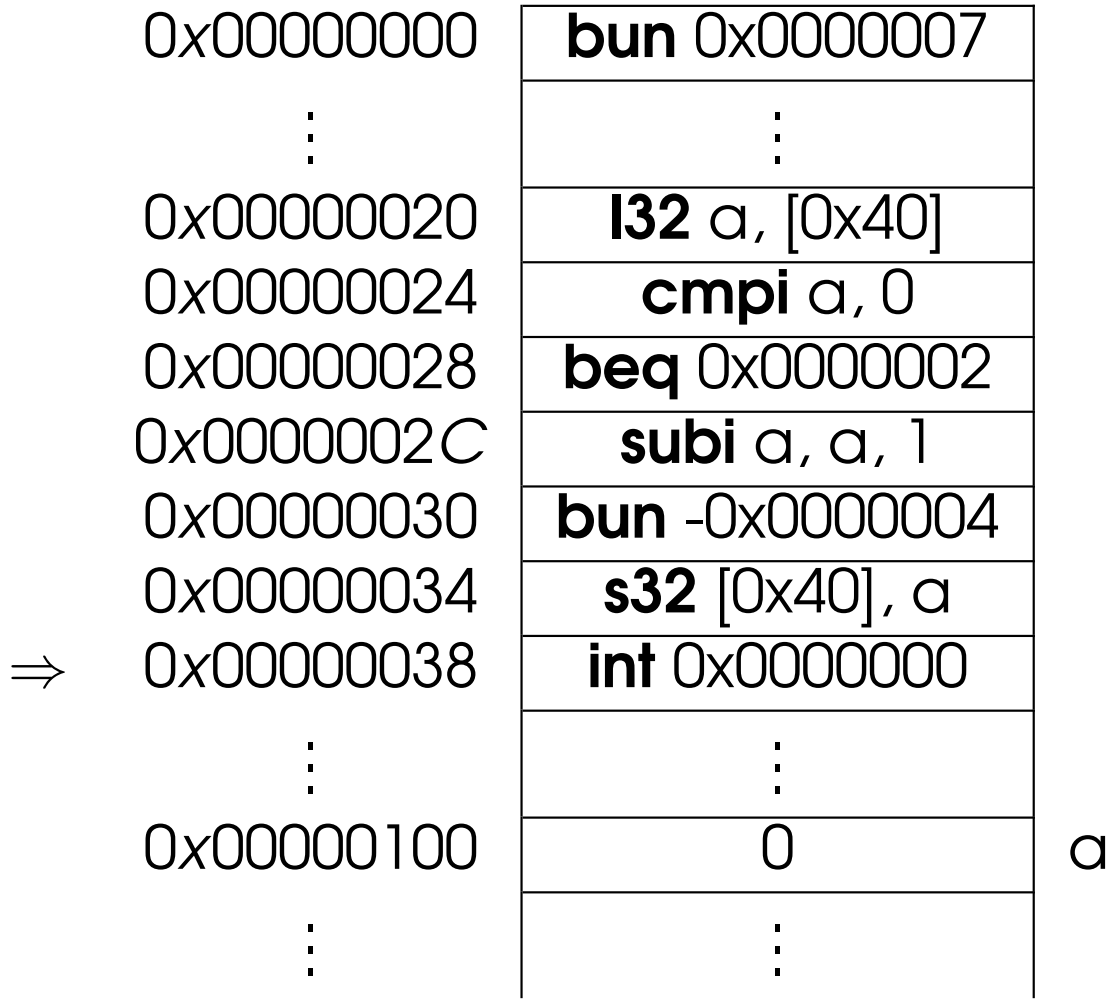
Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



Operações de controle de fluxo

- Controle iterativo
  - Visualização da memória



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

```
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 }
```