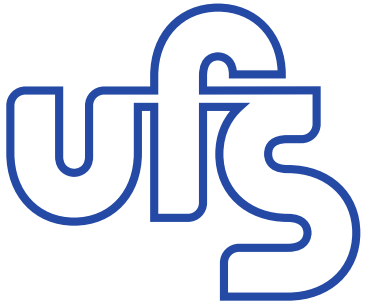
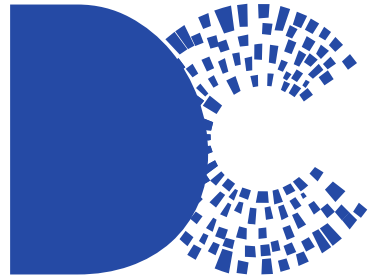




Operações bit a bit e lógicas

Arquitetura de Computadores

Bruno Prado

Departamento de Computação / UFS

Introdução

- Para que servem as operações bit a bit e lógicas?
  - Manipulação individual dos bits

```c
// Bibliotecas padronizadas
#include <stdio.h>
#include <stdint.h>
// Função principal
int main() {
    // Declaração das variáveis
    uint32_t a = 0x1234ABCD;
    // Selecionar os bits 19:12
    uint32_t b = (((a & 0b11111111) << 12) >> 12);
    // Limpar o bit 13
    uint32_t c = (a & ~(1 << 13));
    // Setar o bit 22
    uint32_t d = (a | (1 << 22));
    // Zerar todos os bits
    uint32_t e = (a ^ a);
    // Retornando 0 (sucesso)
    return 0;
}
```

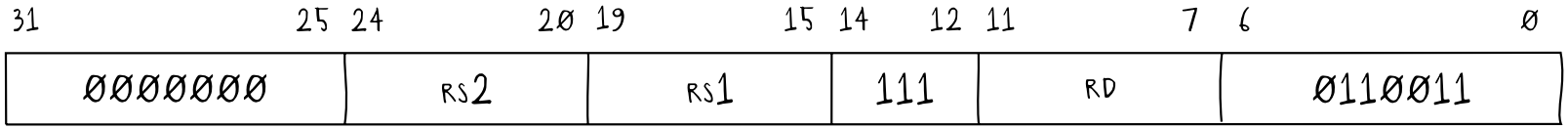
Introdução

- Para que servem as operações bit a bit e lógicas?
  - Comparações relacionais

```
1  // Bibliotecas padronizadas
2  #include <stdio.h>
3  #include <stdint.h>
4  // Função principal
5  int main() {
6      // Declaração das variáveis
7      uint32_t a = 1, b = 2;
8      // a == b
9      uint32_t c = (a == b);
10     // a != b
11     uint32_t d = (a != b);
12     // a < b
13     uint32_t e = (a < b);
14     // a > b
15     uint32_t f = (a > b);
16     // Retornando 0 (sucesso)
17     return 0;
18 }
```

Instruções bit a bit e lógicas

- E (and)




```
and rd, rs1, rs2:
    rd = rs1 & rs2
```

Ex: AND x9, x13, x22

⇩

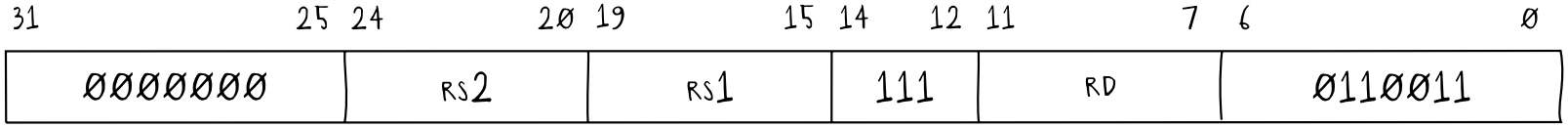
0000000 10110 01101 111 01001 0110011

⇩

B3 F4 66 01

Instruções bit a bit e lógicas

- E (and)




```
and rd, rs1, rs2:
    rd = rs1 & rs2
```

Ex: AND x9, x13, x22

⇩

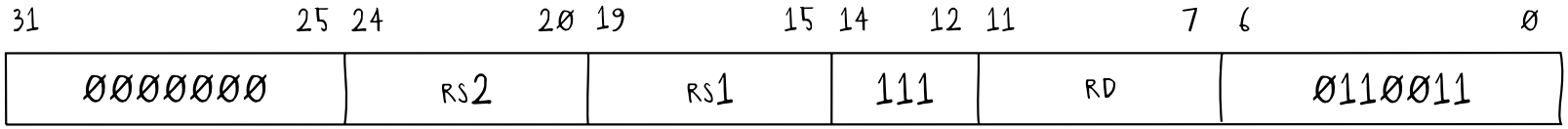
0000000 10110 01101 111 01001 0110011

⇩

B3 F4 66 01

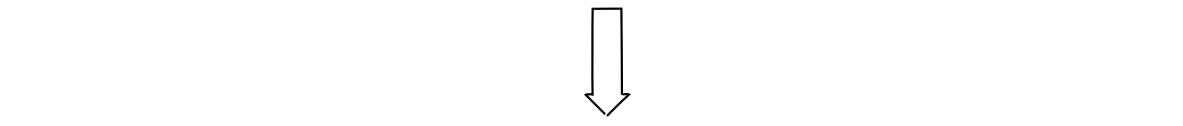
Instruções bit a bit e lógicas

- E (and)




```
and rd, rs1, rs2:
    rd = rs1 & rs2
```

Ex: AND x9, x13, x22

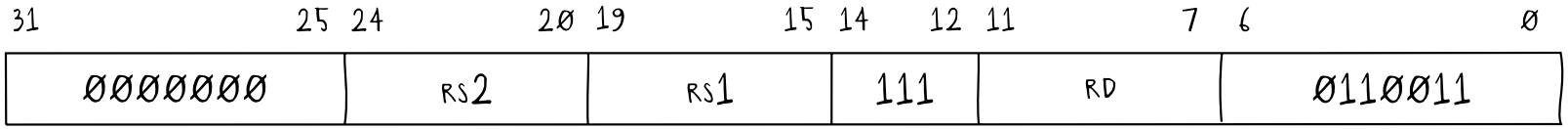


0000000 10110 01101 111 01001 0110011

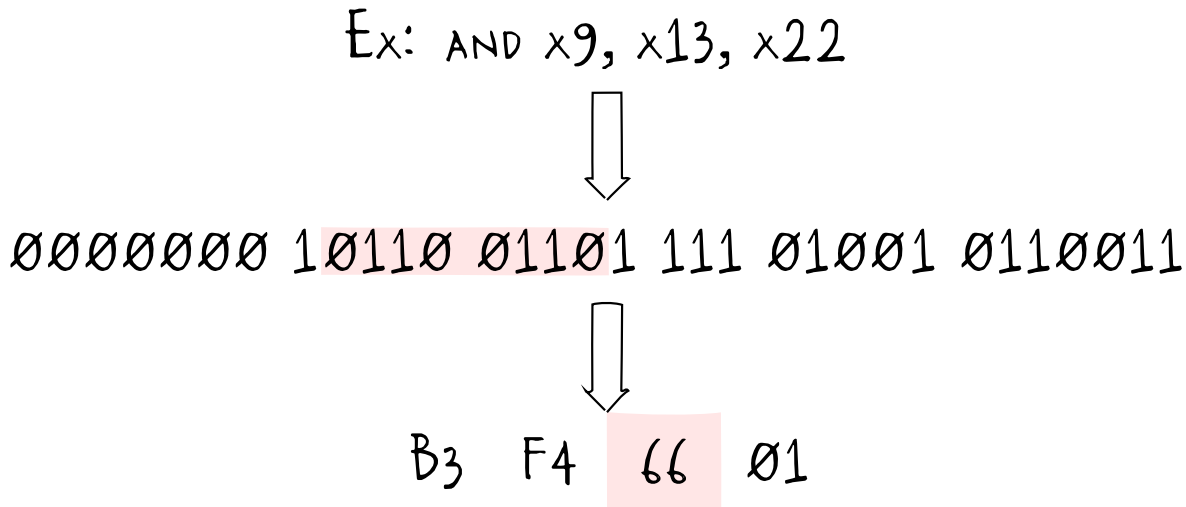
B3 F4 66 01

Instruções bit a bit e lógicas

- E (and)

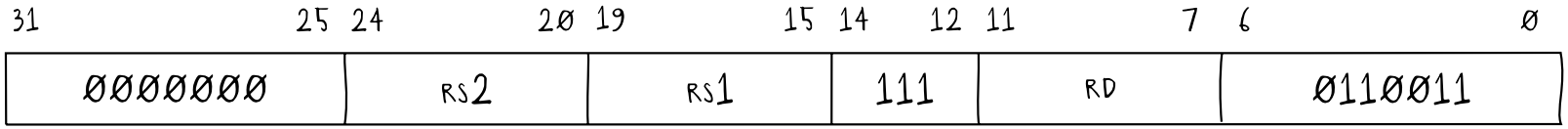



```
and rd, rs1, rs2:
    rd = rs1 & rs2
```



Instruções bit a bit e lógicas

- E (and)




```
and rd, rs1, rs2:
    rd = rs1 & rs2
```

Ex: AND x9, x13, x22

⇩

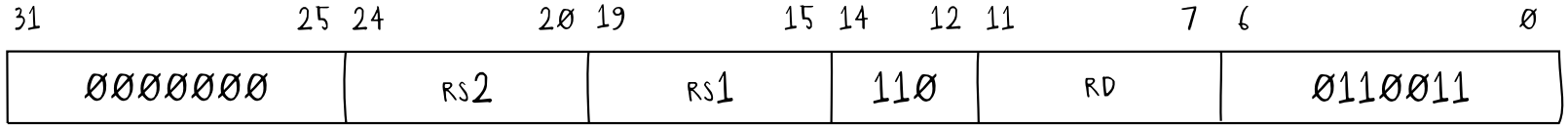
0000000 10110 01101 111 01001 0110011

⇩

B3 F4 66 01

Instruções bit a bit e lógicas

- Ou (or)




```
or rd, rs1, rs2:
    rd = rs1 | rs2
```

Ex: OR x31, x15, x7

⇩

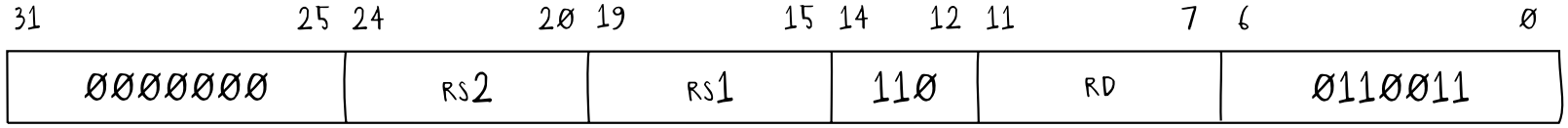
0000000 00111 01111 110 11111 0110011

⇩

B3 EF 77 00

Instruções bit a bit e lógicas

- Ou (or)




```
or rd, rs1, rs2:
    rd = rs1 | rs2
```

Ex: OR x31, x15, x7

⇩

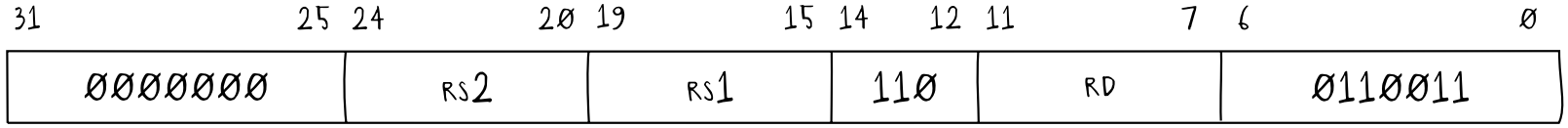
0000000 00111 01111 110 11111 0110011

⇩

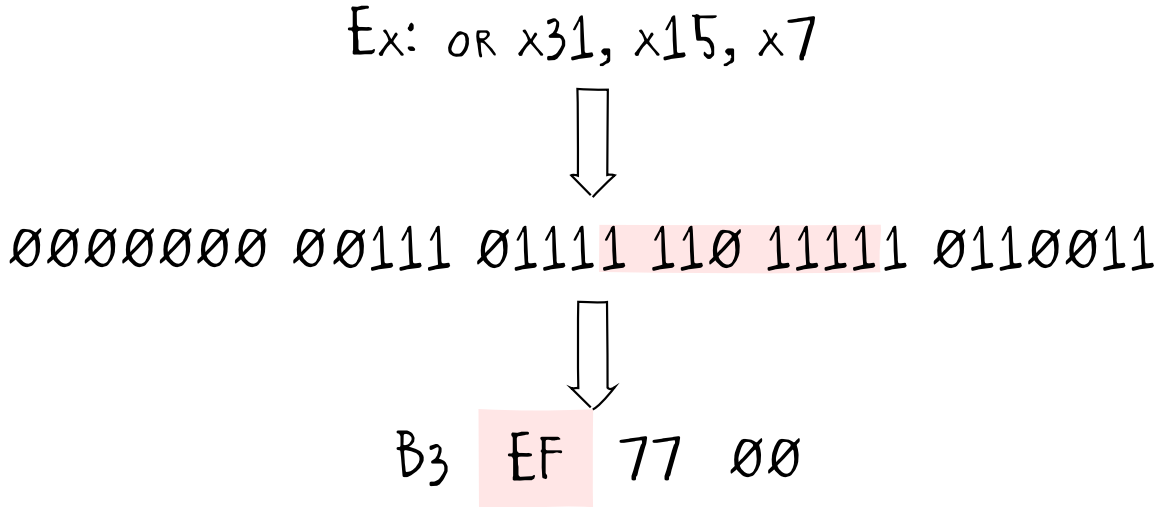
B3 EF 77 00

Instruções bit a bit e lógicas

- Ou (or)

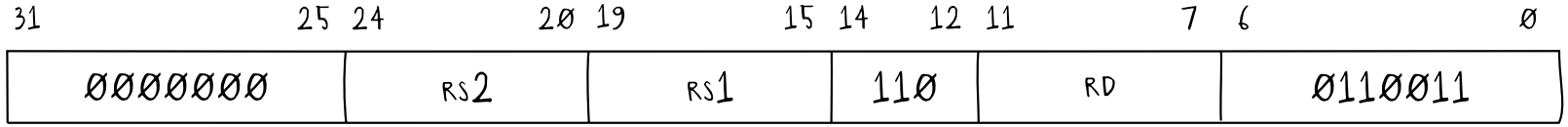



```
or rd, rs1, rs2:
    rd = rs1 | rs2
```

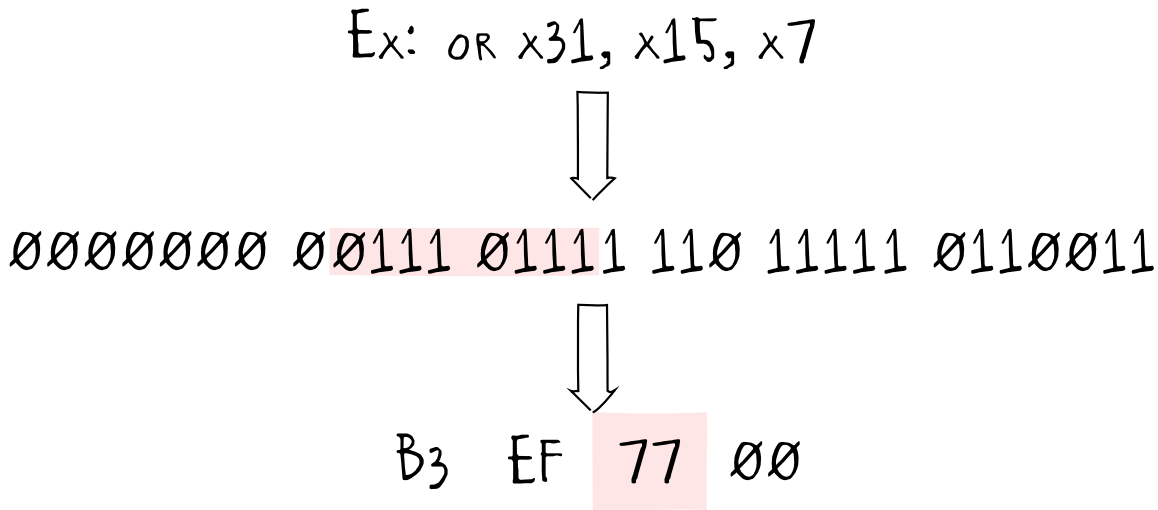


Instruções bit a bit e lógicas

- Ou (or)

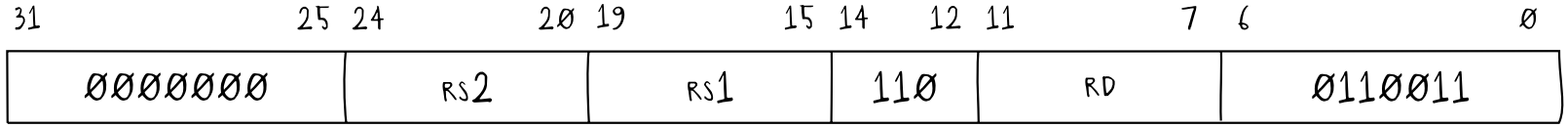



```
or rd, rs1, rs2:
    rd = rs1 | rs2
```

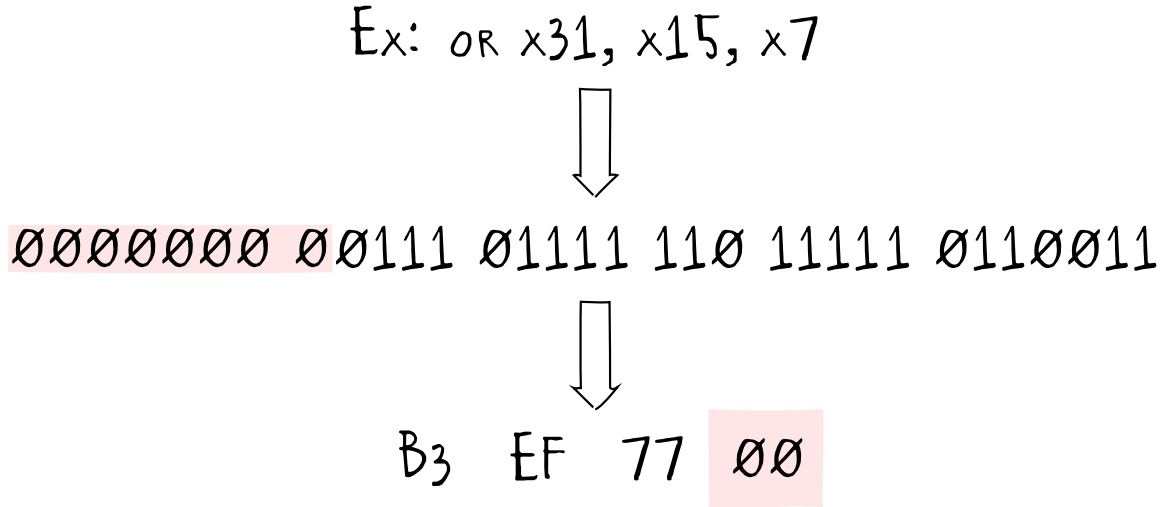


Instruções bit a bit e lógicas

- Ou (or)

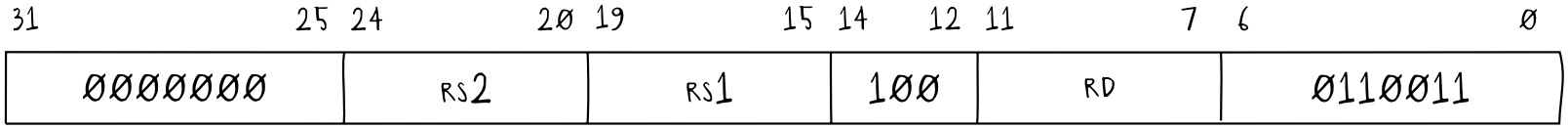



```
or rd, rs1, rs2:
    rd = rs1 | rs2
```



Instruções bit a bit e lógicas

- Ou exclusivo (*xor*)




```
xor rd, rs1, rs2:
    rd = rs1 ^ rs2
```

Ex: xor x1, x2, x3

⇩

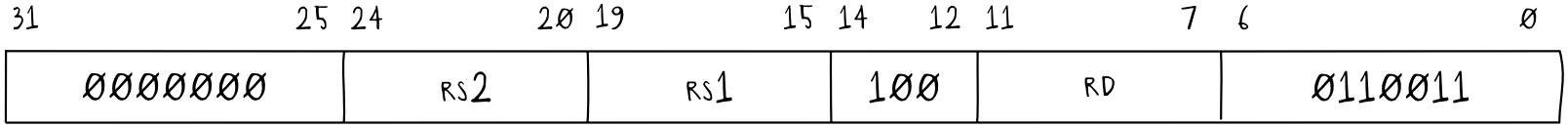
0000000 00011 00010 100 00001 0110011

⇩

B3 40 31 00

Instruções bit a bit e lógicas

- Ou exclusivo (*xor*)




```
xor rd, rs1, rs2:
    rd = rs1 ^ rs2
```

Ex: xor x1, x2, x3

⇩

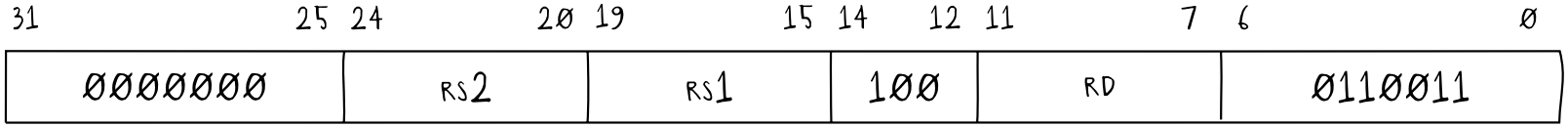
0000000 00011 00010 100 00001 0110011

⇩

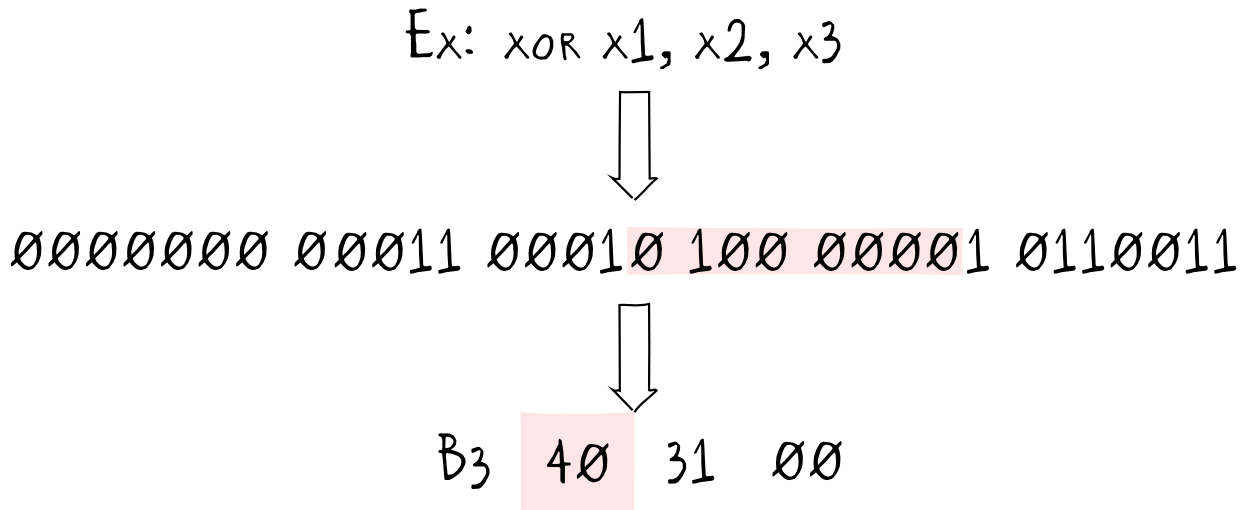
B3 40 31 00

Instruções bit a bit e lógicas

- Ou exclusivo (*xor*)

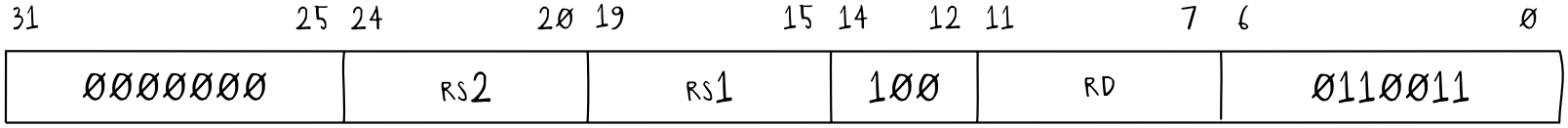



```
xor rd, rs1, rs2:
    rd = rs1 ^ rs2
```

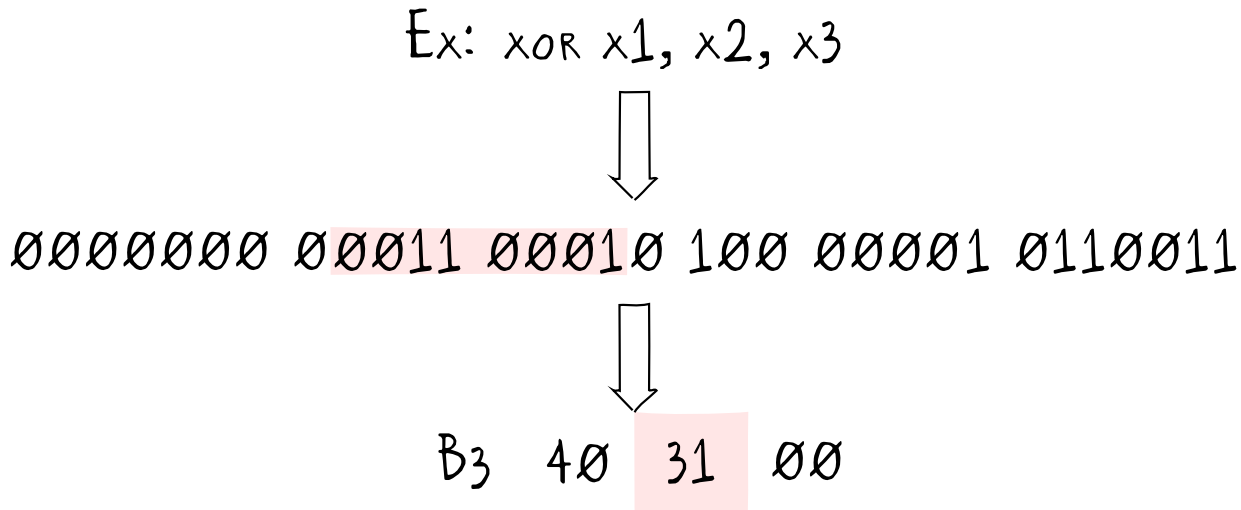


Instruções bit a bit e lógicas

- Ou exclusivo (*xor*)

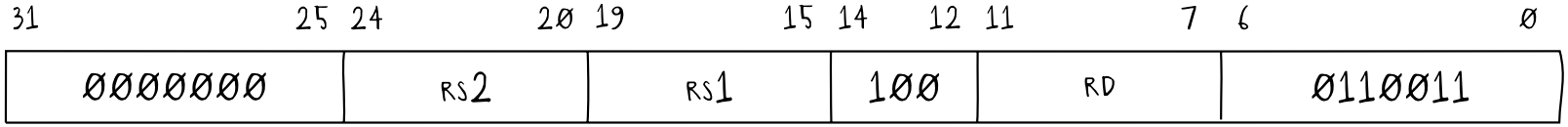



```
xor rd, rs1, rs2:
    rd = rs1 ^ rs2
```



Instruções bit a bit e lógicas

- Ou exclusivo (*xor*)




```
xor rd, rs1, rs2:
    rd = rs1 ^ rs2
```

Ex: xor x1, x2, x3

⇩

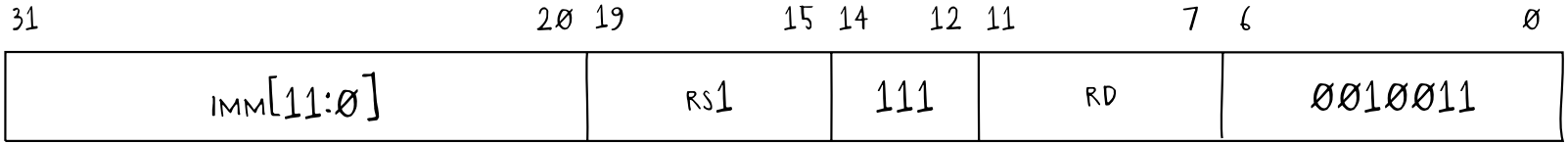
0000000 00011 00010 100 00001 0110011

⇩

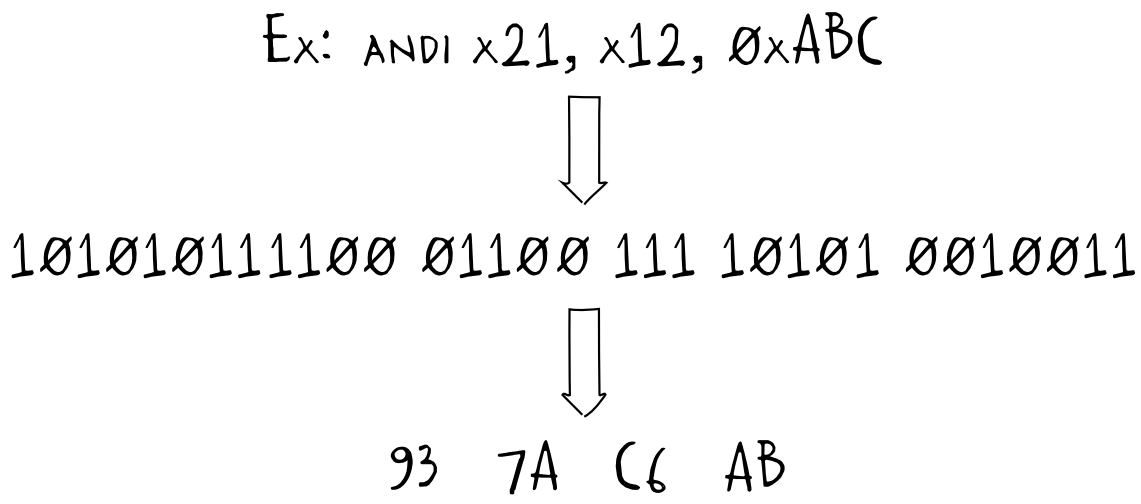
B3 40 31 00

Instruções bit a bit e lógicas

- E imediato (*and immediate*)

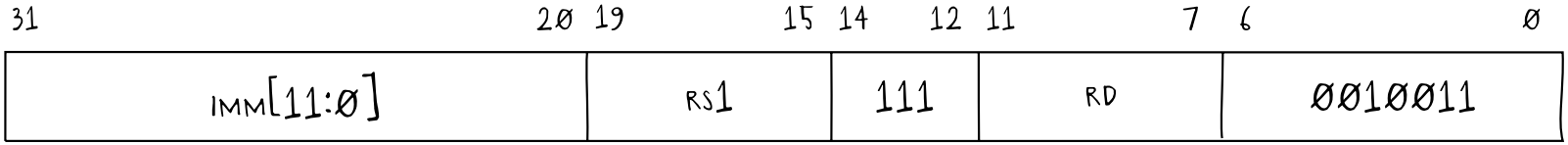



```
andi rd, rs1, imm:
    rd = rs1 & sign_extension(imm)
```

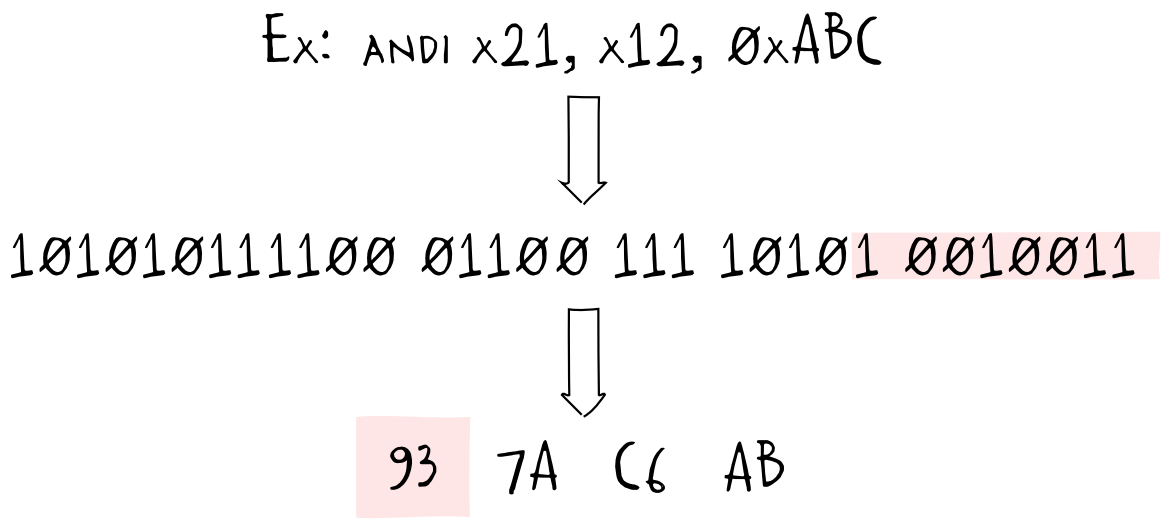


Instruções bit a bit e lógicas

- E imediato (*and immediate*)

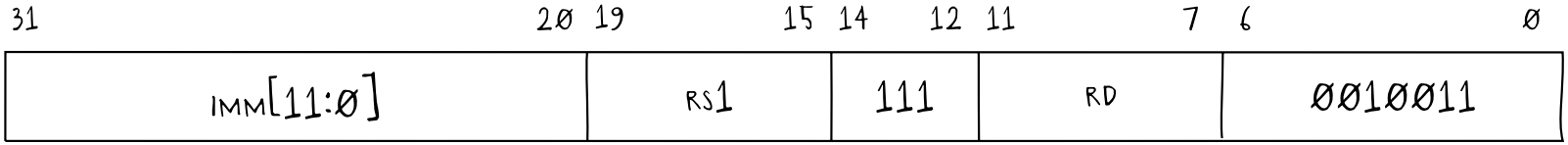



```
andi rd, rs1, imm:
    rd = rs1 & sign_extension(imm)
```

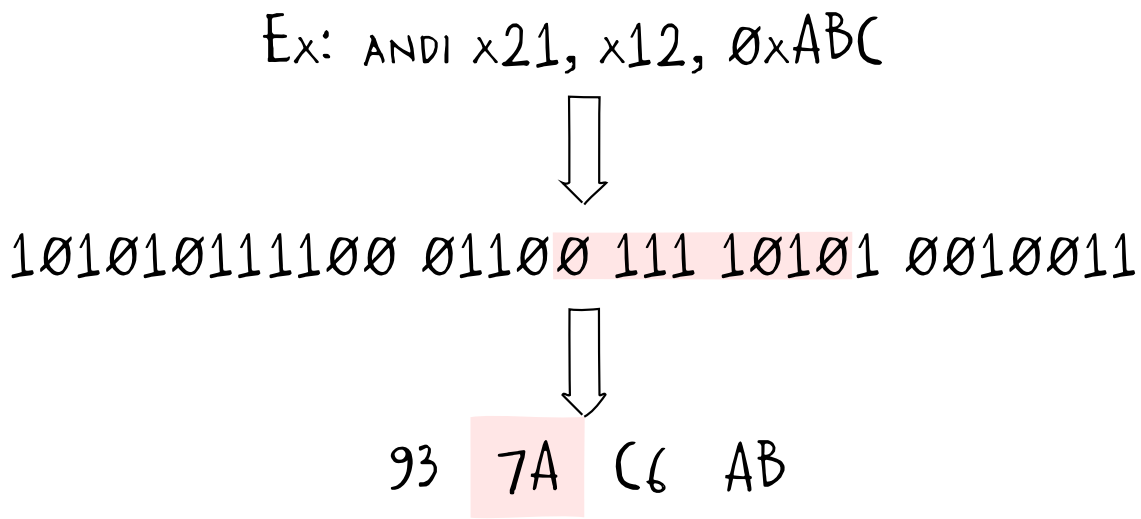


Instruções bit a bit e lógicas

- E imediato (*and immediate*)

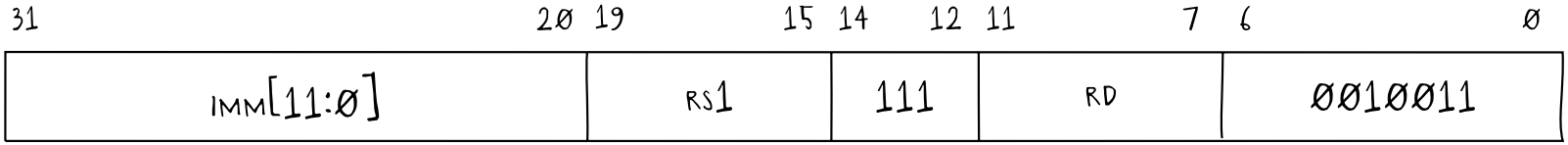



```
andi rd, rs1, imm:
    rd = rs1 & sign_extension(imm)
```

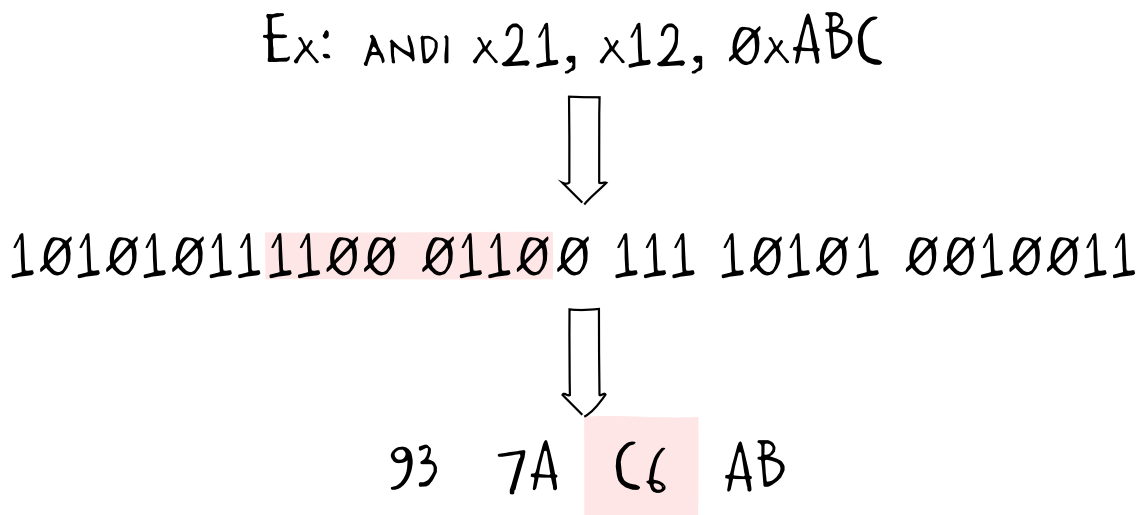


Instruções bit a bit e lógicas

- E imediato (*and immediate*)

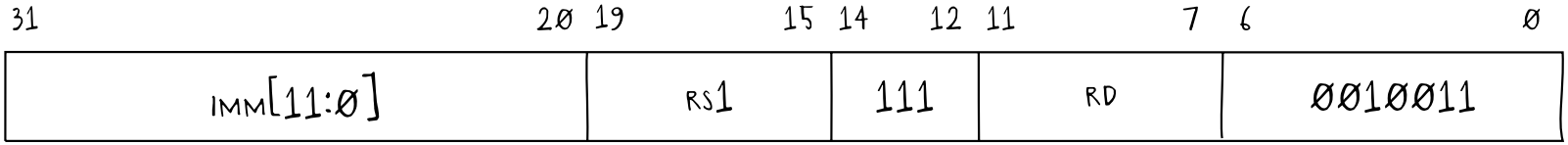



```
andi rd, rs1, imm:
    rd = rs1 & sign_extension(imm)
```



Instruções bit a bit e lógicas

- E imediato (*and immediate*)




```
andi rd, rs1, imm:
    rd = rs1 & sign_extension(imm)
```

Ex: ANDI x21, x12, 0xABC

⇩

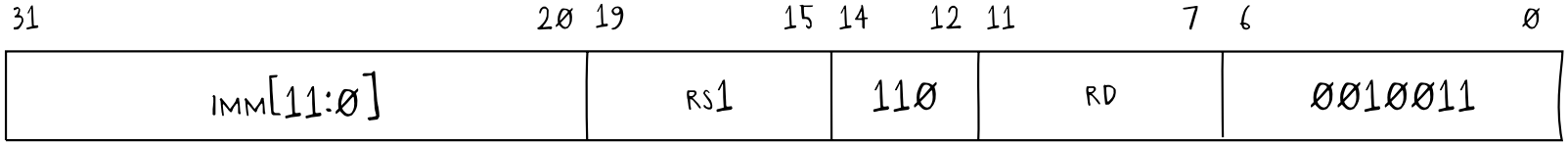
10101011110000 01100 111 10101 0010011

⇩

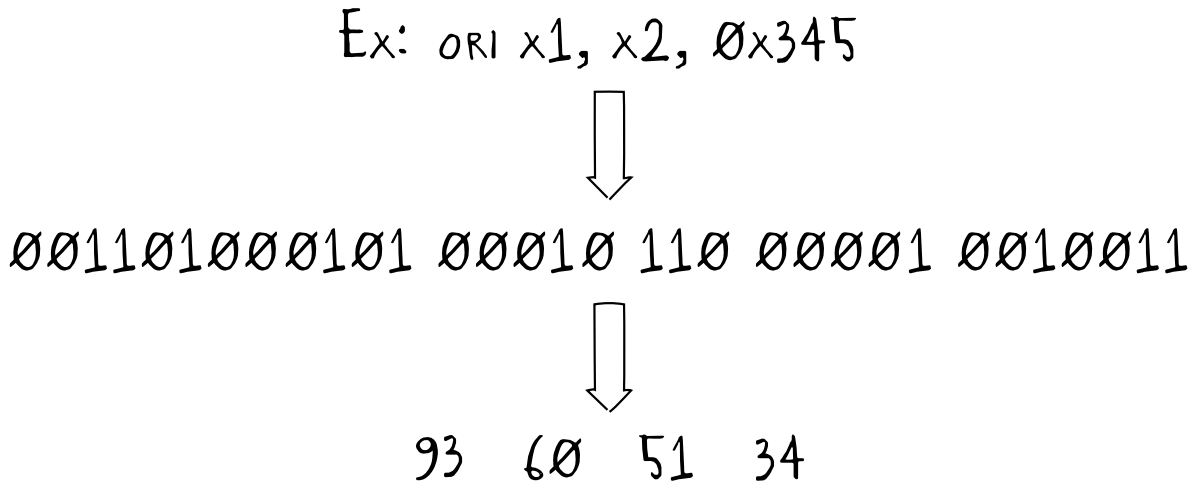
93 7A C6 AB

Instruções bit a bit e lógicas

- Ou imediato (*or immediate*)

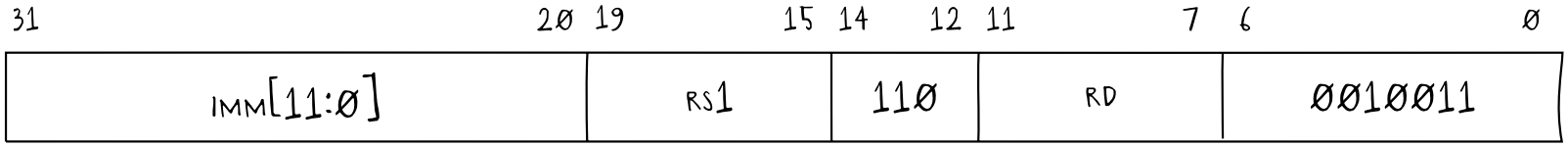



```
ori rd, rs1, imm:
    rd = rs1 | sign_extension(imm)
```



Instruções bit a bit e lógicas

- Ou imediato (*or immediate*)




```
ori rd, rs1, imm:
    rd = rs1 | sign_extension(imm)
```

Ex: ORI x1, x2, 0x345

⇩

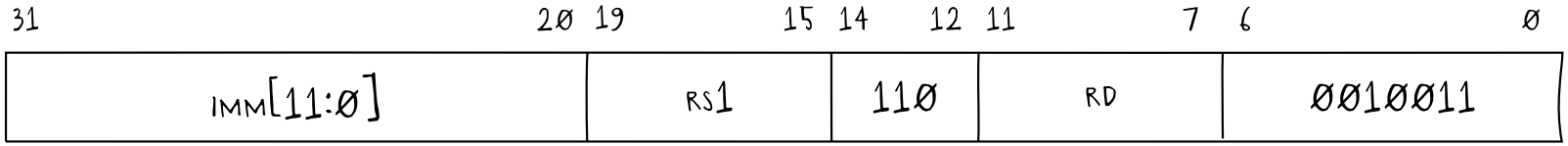
0011010000101 00010 110 00001 0010011

⇩

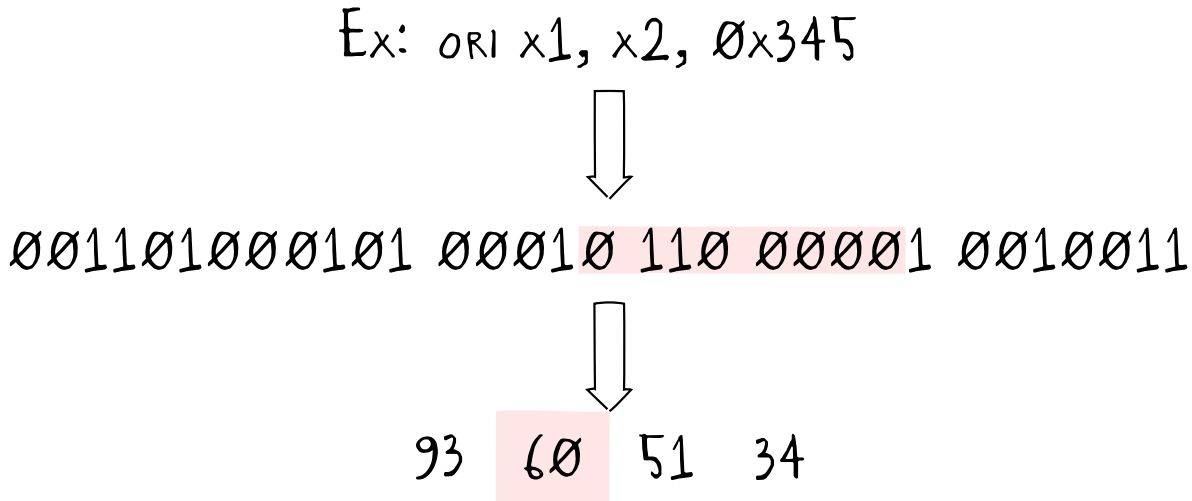
93 60 51 34

Instruções bit a bit e lógicas

- Ou imediato (*or immediate*)

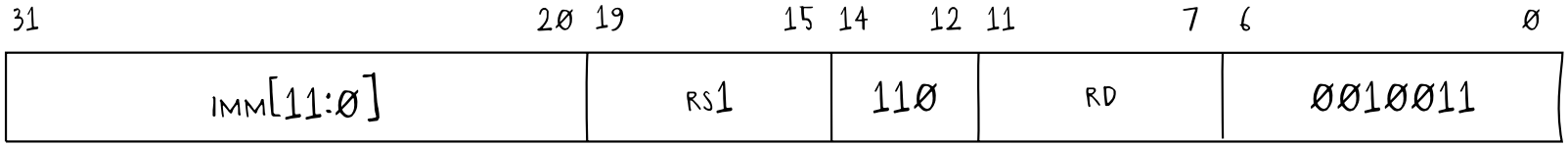



```
ori rd, rs1, imm:
    rd = rs1 | sign_extension(imm)
```

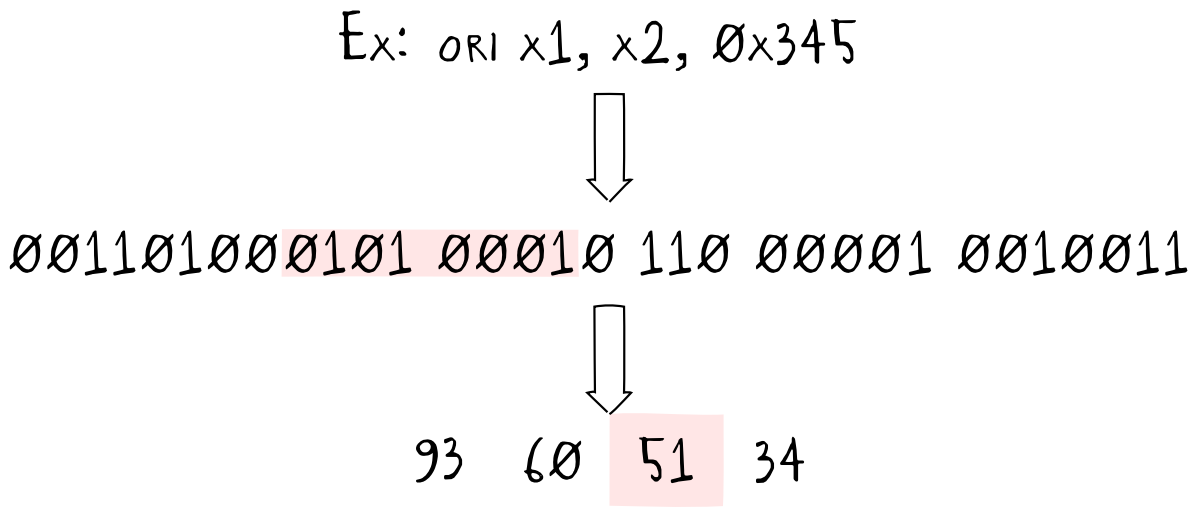


Instruções bit a bit e lógicas

- Ou imediato (*or immediate*)

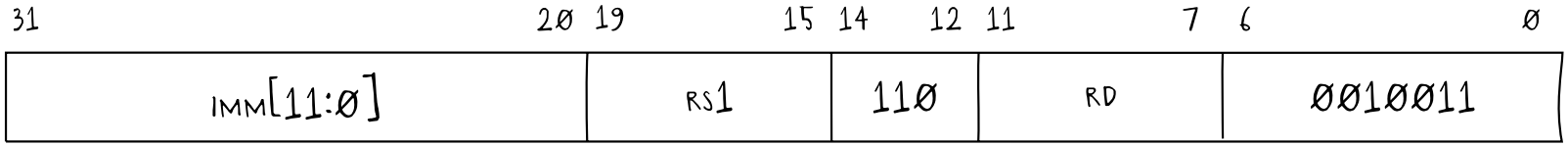



```
ori rd, rs1, imm:
    rd = rs1 | sign_extension(imm)
```



Instruções bit a bit e lógicas

- Ou imediato (*or immediate*)




```
ori rd, rs1, imm:
    rd = rs1 | sign_extension(imm)
```

Ex: ORI x1, x2, 0x345

⇩

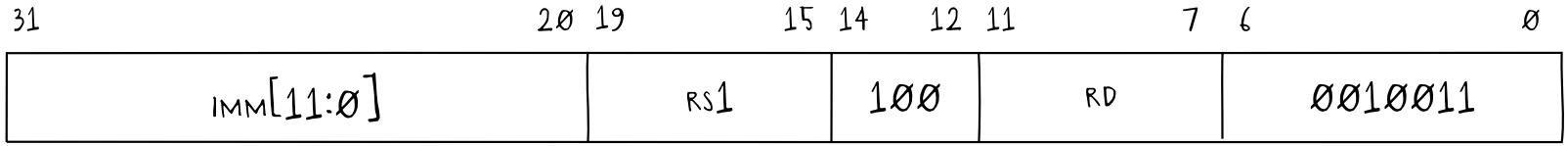
000110100101 00010 110 00001 0010011

⇩

93 60 51 34

Instruções bit a bit e lógicas

- Ou exclusivo imediato (*xor immediate*)




```
xori rd, rs1, imm:
    rd = rs1 ^ sign_extension(imm)

not rd, rs1:
    xori rd, rs1, -1
```

Ex: XORI x31, x31, 0xDEF

⇩

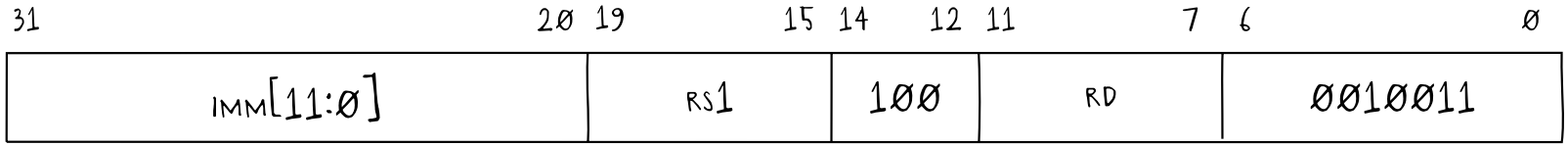
11011111011111111100111111110011

⇩

93 CF FF DE

Instruções bit a bit e lógicas

- Ou exclusivo imediato (*xor immediate*)




```
xori rd, rs1, imm:
    rd = rs1 ^ sign_extension(imm)

not rd, rs1:
    xori rd, rs1, -1
```

Ex: XORI x31, x31, 0xDEF

⇩

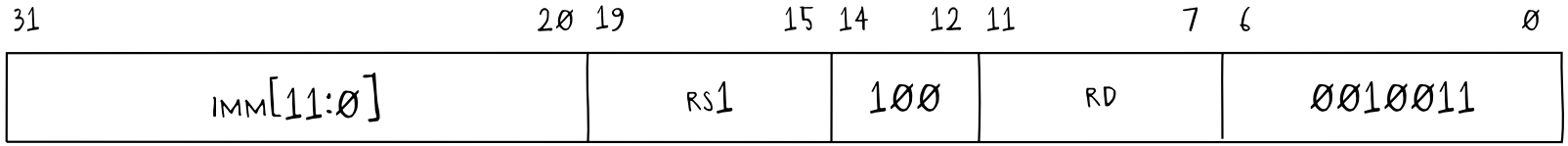
11011110111 11111 100 11111 0010011

⇩

93 CF FF DE

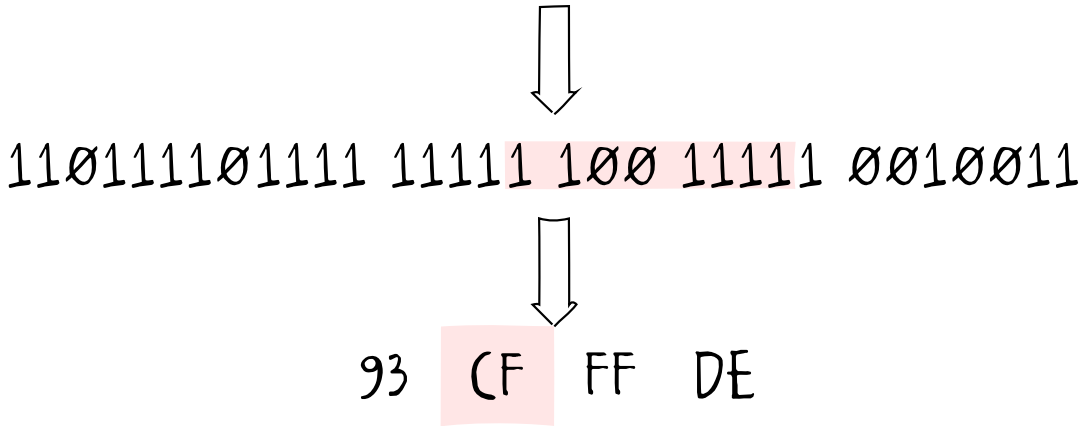
Instruções bit a bit e lógicas

- Ou exclusivo imediato (*xor immediate*)



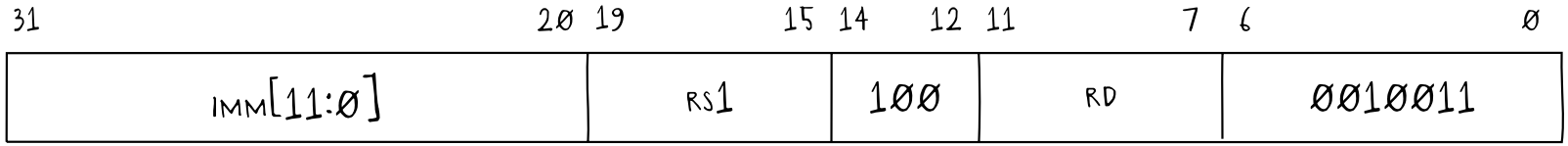

```
xori rd, rs1, imm:
    rd = rs1 ^ sign_extension(imm)

not rd, rs1:
    xori rd, rs1, -1
```

Ex: XORI x31, x31, 0xDEF



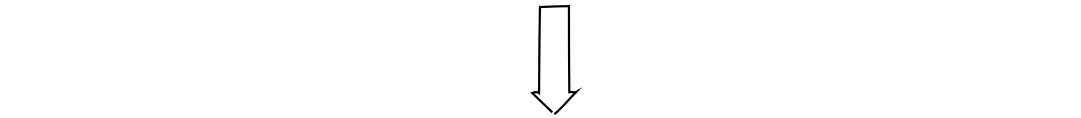
Instruções bit a bit e lógicas

- Ou exclusivo imediato (*xor immediate*)

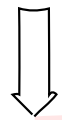



```
xori rd, rs1, imm:
    rd = rs1 ^ sign_extension(imm)

not rd, rs1:
    xori rd, rs1, -1
```

Ex: XORI x31, x31, 0xDEF



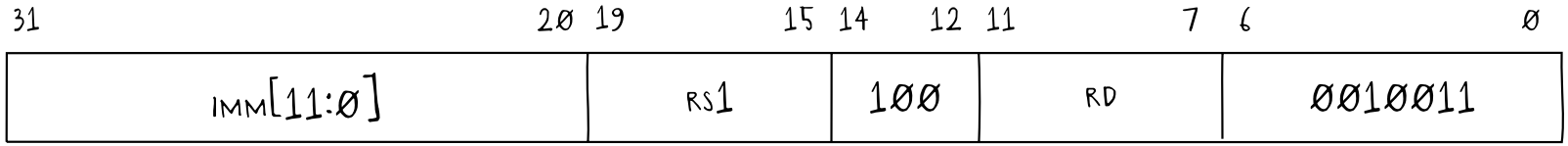
11011110 1111 11111 100 11111 0010011



93 CF FF DE

Instruções bit a bit e lógicas

- Ou exclusivo imediato (*xor immediate*)




```
xori rd, rs1, imm:
    rd = rs1 ^ sign_extension(imm)

not rd, rs1:
    xori rd, rs1, -1
```

Ex: XORI x31, x31, 0xDEF

⇩

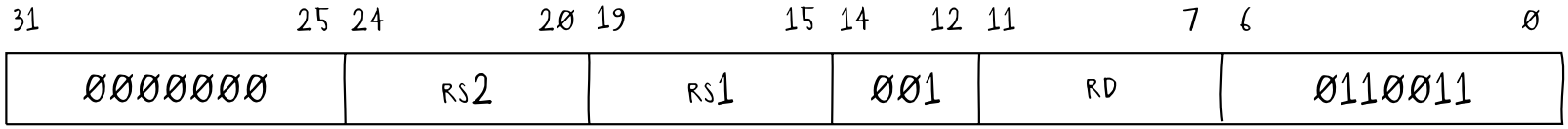
11011110 1111 11111 100 11111 0010011

⇩

93 CF FF DE

Instruções bit a bit e lógicas

- Deslocamento para esquerda lógico (*shift left logical*)




```
sll rd, rs1, rs2:
    rd = rs1 << get_4_0(rs2)
```

Ex: SLL x5, x8, x13

⇩

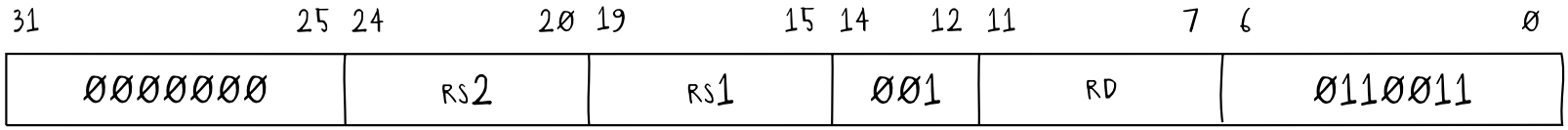
0000000 01101 01000 001 00101 0110011

⇩

B3 12 D4 00

Instruções bit a bit e lógicas

- Deslocamento para esquerda lógico (*shift left logical*)




```
sll rd, rs1, rs2:
    rd = rs1 << get_4_0(rs2)
```

Ex: SLL x5, x8, x13

⇩

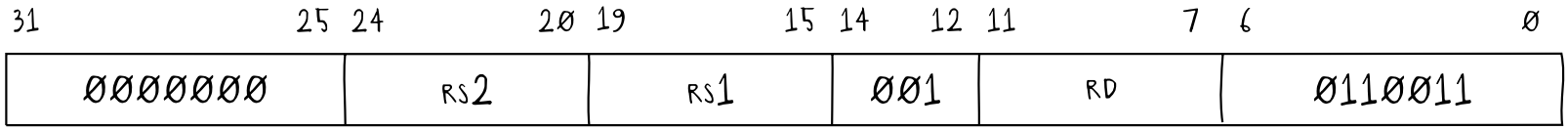
0000000 01101 01000 001 00101 0110011

⇩

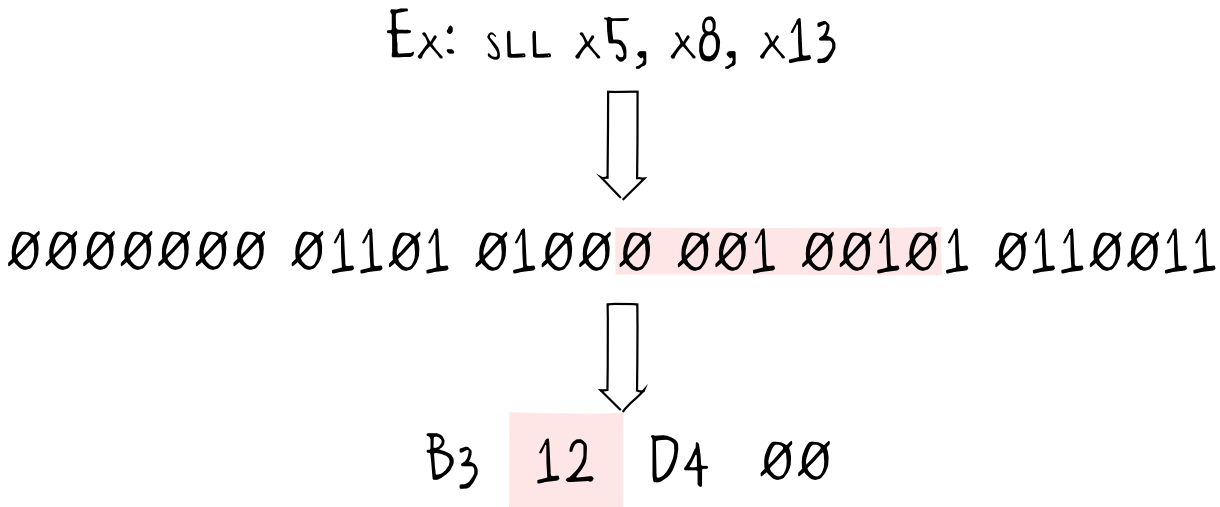
B3 12 D4 00

Instruções bit a bit e lógicas

- Deslocamento para esquerda lógico (*shift left logical*)

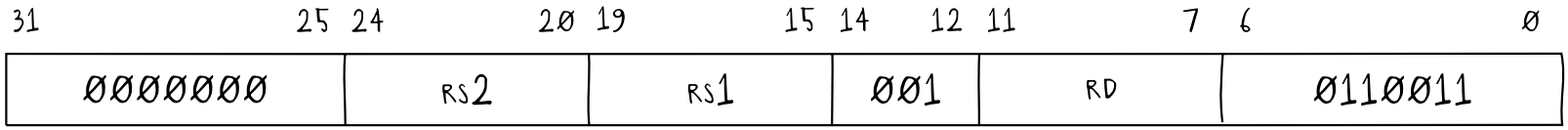



```
sll rd, rs1, rs2:
    rd = rs1 << get_4_0(rs2)
```

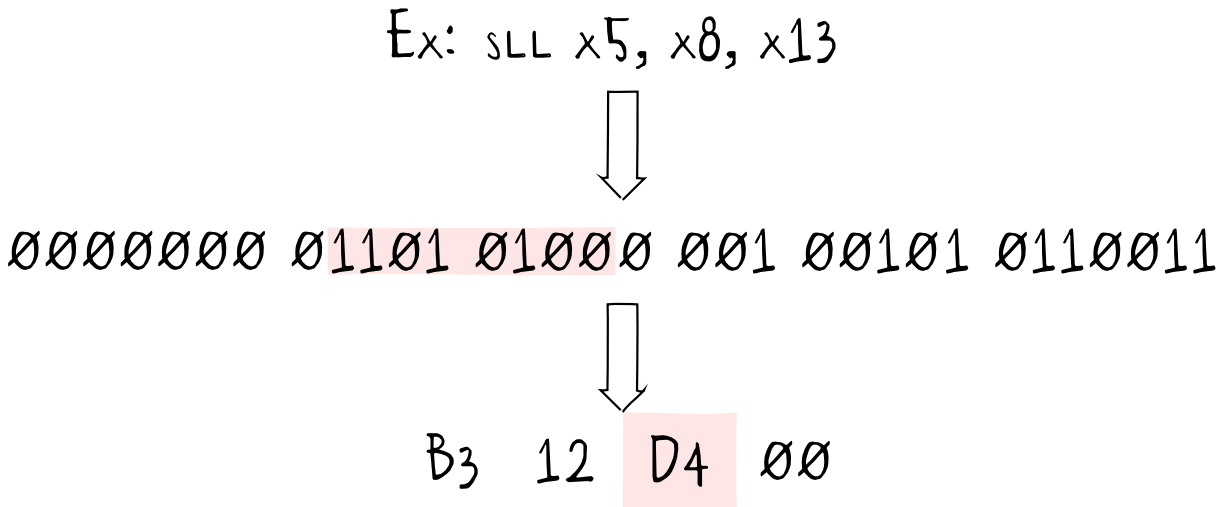


Instruções bit a bit e lógicas

- Deslocamento para esquerda lógico (*shift left logical*)

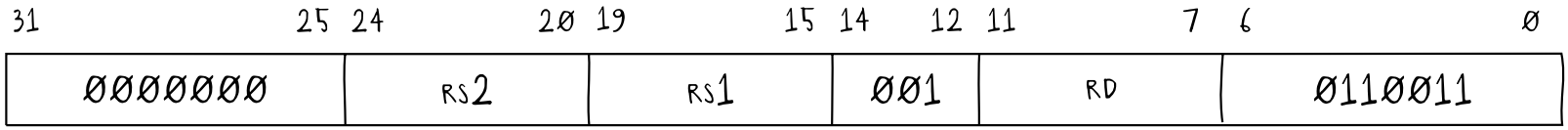



```
sll rd, rs1, rs2:
    rd = rs1 << get_4_0(rs2)
```



Instruções bit a bit e lógicas

- Deslocamento para esquerda lógico (*shift left logical*)




```
sll rd, rs1, rs2:
    rd = rs1 << get_4_0(rs2)
```

Ex: SLL x5, x8, x13

⇩

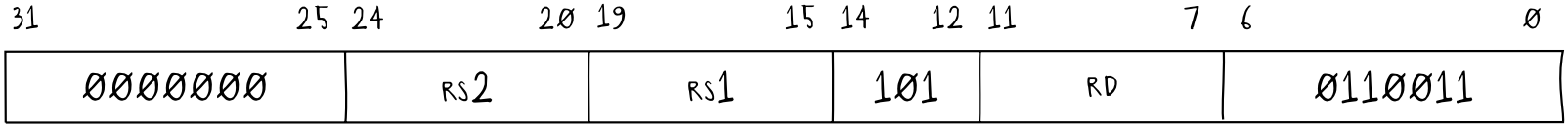
0000000 01101 01000 001 00101 0110011

⇩

B3 12 D4 00

Instruções bit a bit e lógicas

- Deslocamento para direita lógico (*shift right logical*)




```
srl rd, rs1, rs2:
    rd = rs1 >> get_4_0(rs2)
```

Ex: SRL x8, x13, x21

⇩

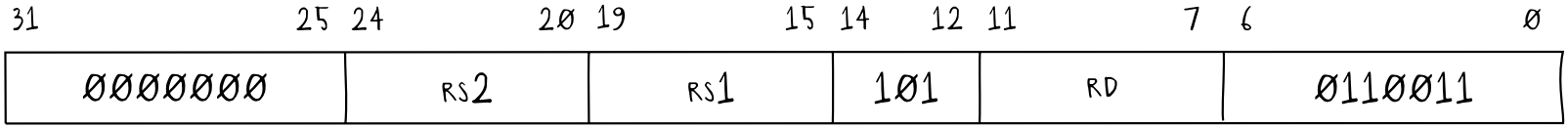
0000000 10101 01101 101 01000 0110011

⇩

33 D4 56 01

Instruções bit a bit e lógicas

- Deslocamento para direita lógico (*shift right logical*)




```
srl rd, rs1, rs2:
    rd = rs1 >> get_4_0(rs2)
```

Ex: SRL x8, x13, x21

⇩

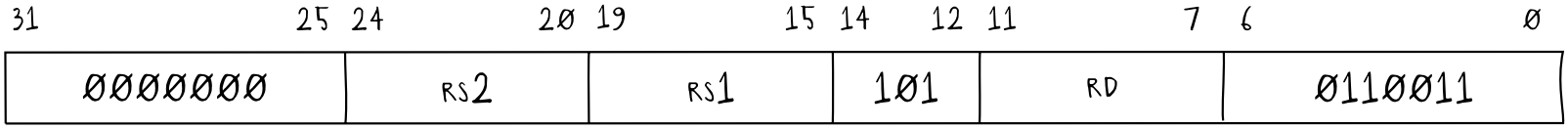
0000000 10101 01101 101 01000 0110011

⇩

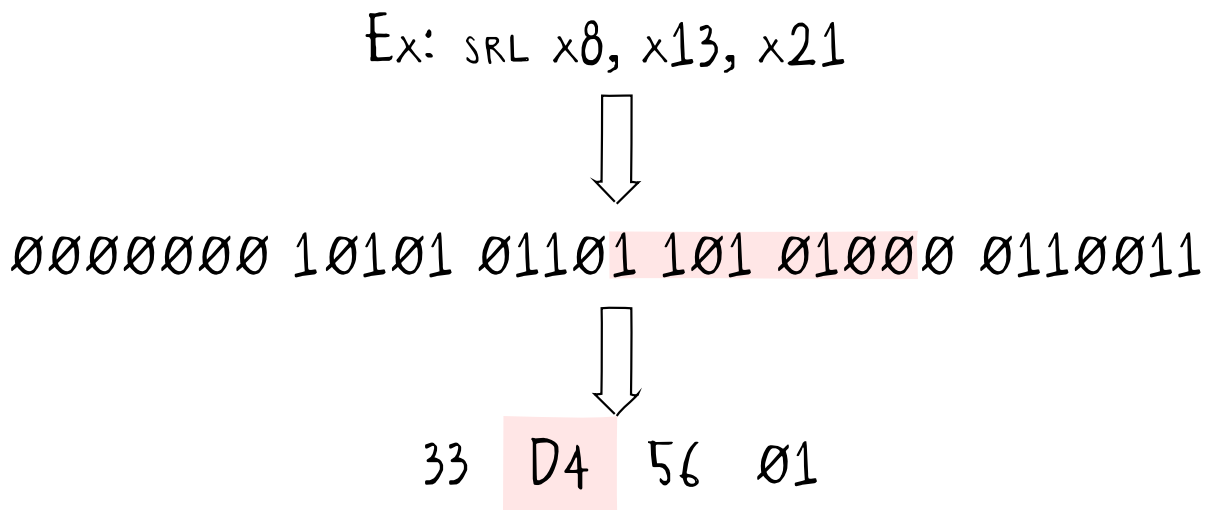
33 D4 56 01

Instruções bit a bit e lógicas

- Deslocamento para direita lógico (*shift right logical*)

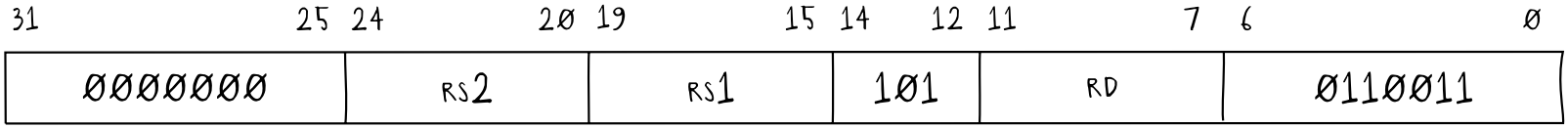



```
srl rd, rs1, rs2:
    rd = rs1 >> get_4_0(rs2)
```



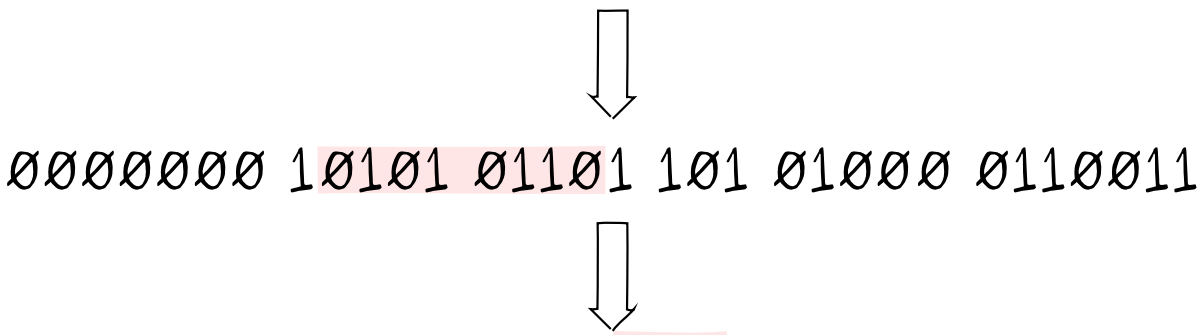
Instruções bit a bit e lógicas

- Deslocamento para direita lógico (*shift right logical*)



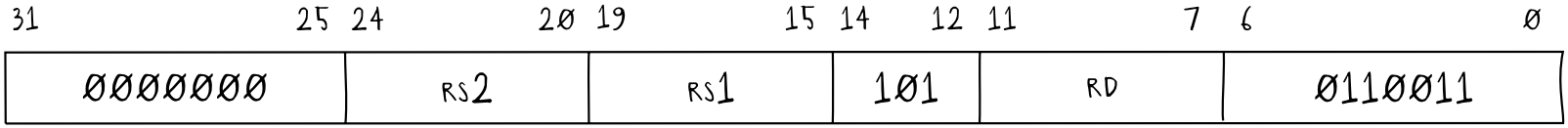

```
srl rd, rs1, rs2:
    rd = rs1 >> get_4_0(rs2)
```

Ex: SRL x8, x13, x21



Instruções bit a bit e lógicas

- Deslocamento para direita lógico (*shift right logical*)




```
srl rd, rs1, rs2:
    rd = rs1 >> get_4_0(rs2)
```

Ex: SRL x8, x13, x21

⇩

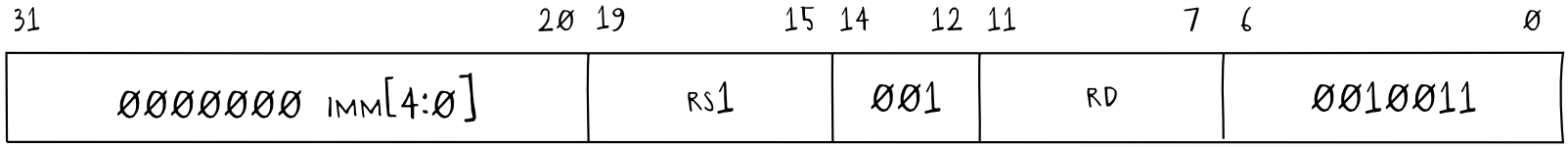
0000000 10101 01101 101 01000 0110011

⇩

33 D4 56 01

Instruções bit a bit e lógicas

- Deslocamento para esquerda lógico imediato (*shift left logical immediate*)




```
slli rd, rs1, imm:
    rd = rs1 << imm
```

Ex: SLLI x13, x21, 31

⇩

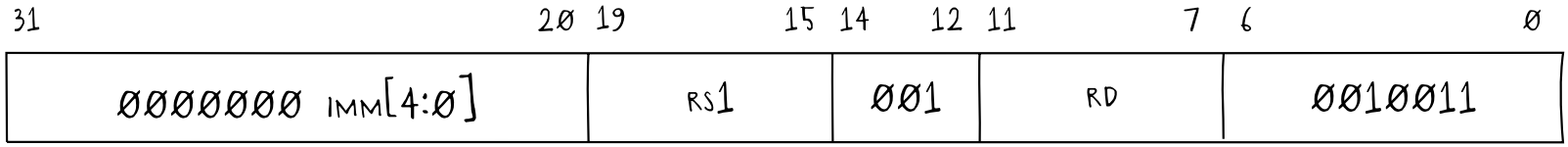
0000000 11111 10101 001 01101 0010011

⇩

93 96 FA 01

Instruções bit a bit e lógicas

- Deslocamento para esquerda lógico imediato (*shift left logical immediate*)




```
slli rd, rs1, imm:
    rd = rs1 << imm
```

Ex: SLLI x13, x21, 31

⇩

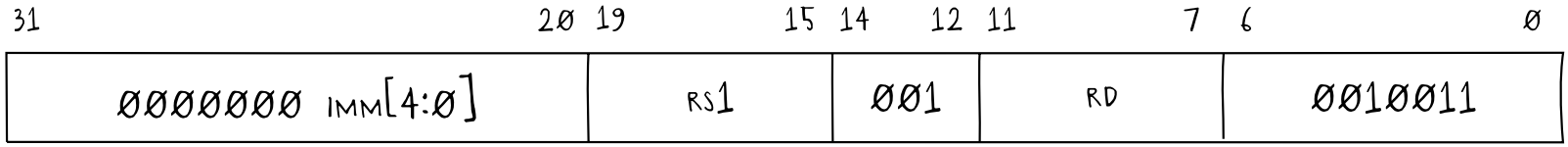
0000000 11111 10101 001 01101 0010011

⇩

93 96 FA 01

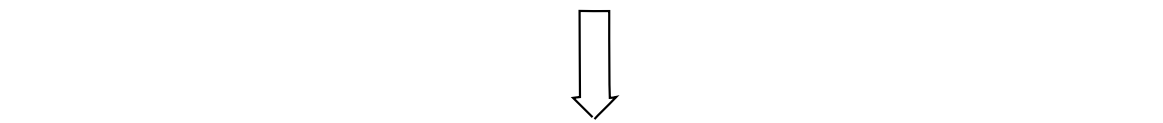
Instruções bit a bit e lógicas

- Deslocamento para esquerda lógico imediato (*shift left logical immediate*)

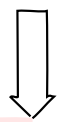



```
slli rd, rs1, imm:
    rd = rs1 << imm
```

Ex: SLLI x13, x21, 31



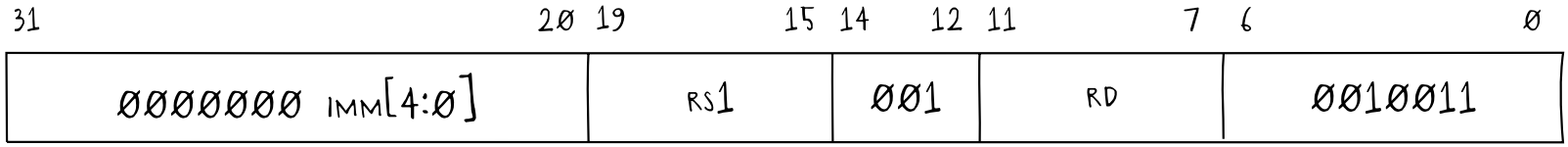
0000000 11111 10101 001 01101 0010011



93 96 FA 01

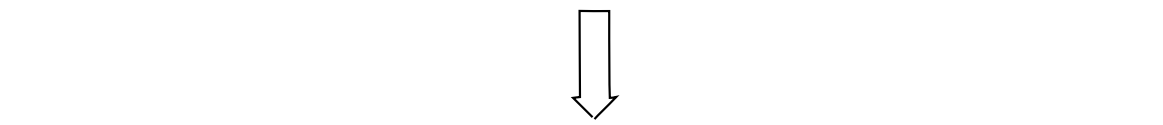
Instruções bit a bit e lógicas

- Deslocamento para esquerda lógico imediato (*shift left logical immediate*)




```
slli rd, rs1, imm:
    rd = rs1 << imm
```

Ex: SLLI x13, x21, 31

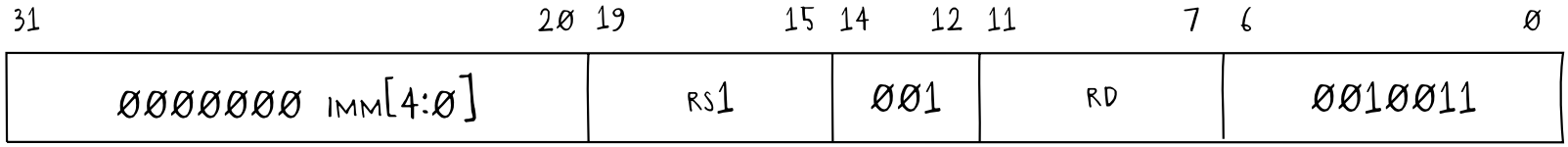


0000000011111 10101 001 01101 0010011

93 96 FA 01

Instruções bit a bit e lógicas

- Deslocamento para esquerda lógico imediato (*shift left logical immediate*)




```
slli rd, rs1, imm:
    rd = rs1 << imm
```

Ex: SLLI x13, x21, 31

⇩

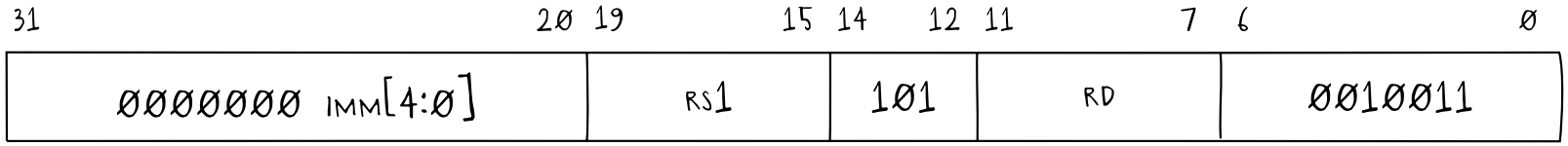
00000001 1111 1010 1001 0011 0101 0010011

⇩

93 96 FA 01

Instruções bit a bit e lógicas

- Deslocamento para direita lógico imediato (*shift right logical immediate*)




```
srli rd, rs1, imm:
    rd = rs1 >> imm
```

Ex: SRLI x21, x13, 10

⇩

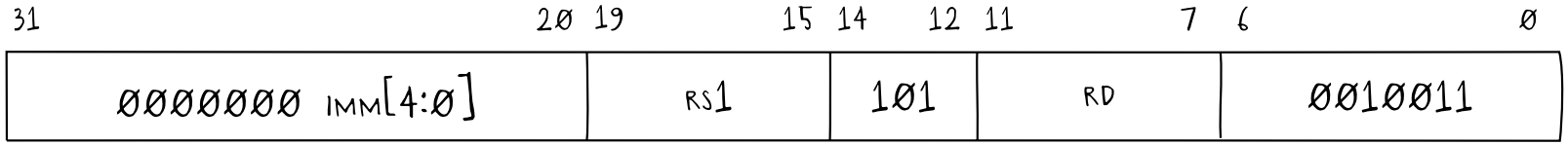
0000000010101101101010100010011

⇩

93 DA A6 00

Instruções bit a bit e lógicas

- Deslocamento para direita lógico imediato (*shift right logical immediate*)




```
srli rd, rs1, imm:
    rd = rs1 >> imm
```

Ex: SRLI x21, x13, 10

⇩

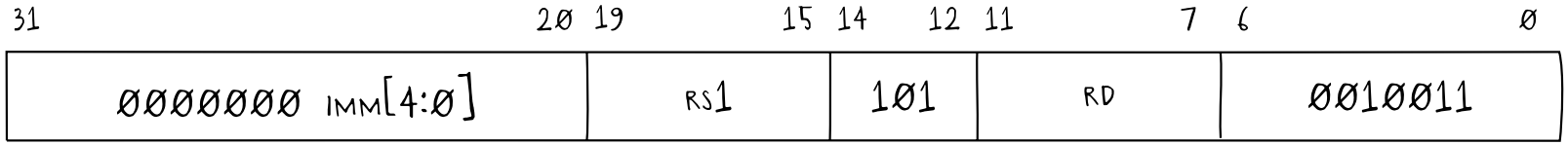
0000000010 10 01101 101 10101 0010011

⇩

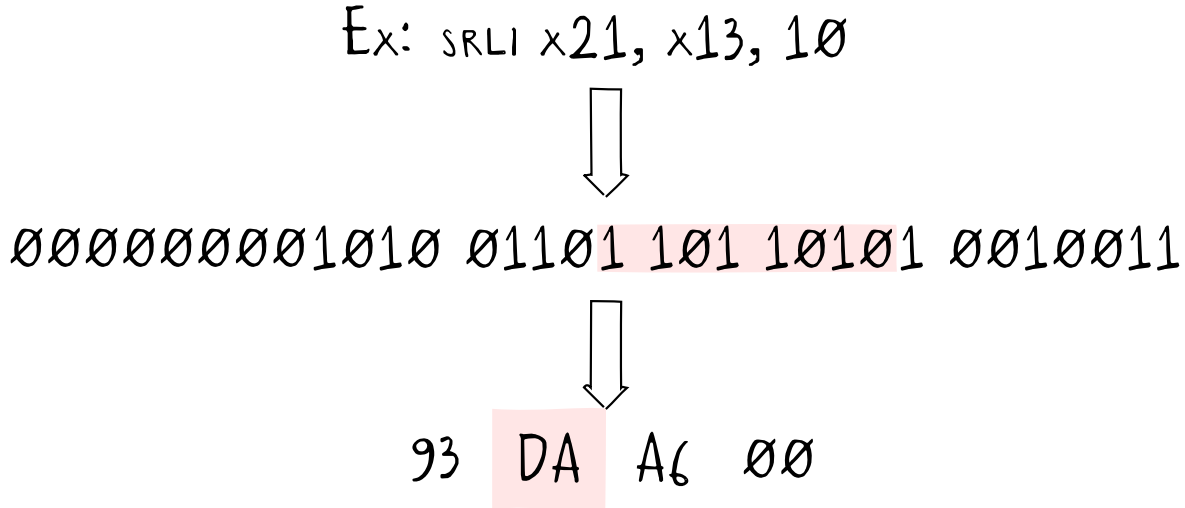
93 DA A6 00

Instruções bit a bit e lógicas

- Deslocamento para direita lógico imediato (*shift right logical immediate*)

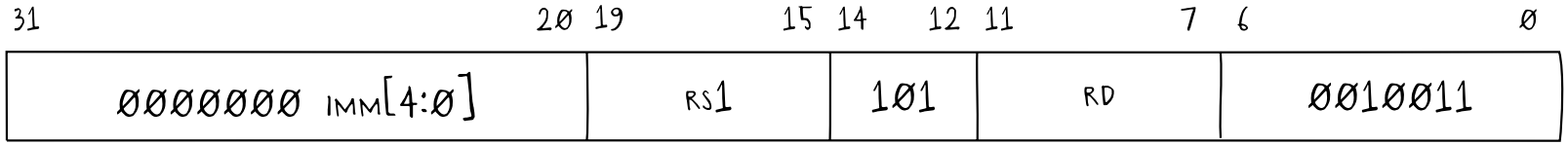



```
srli rd, rs1, imm:
    rd = rs1 >> imm
```

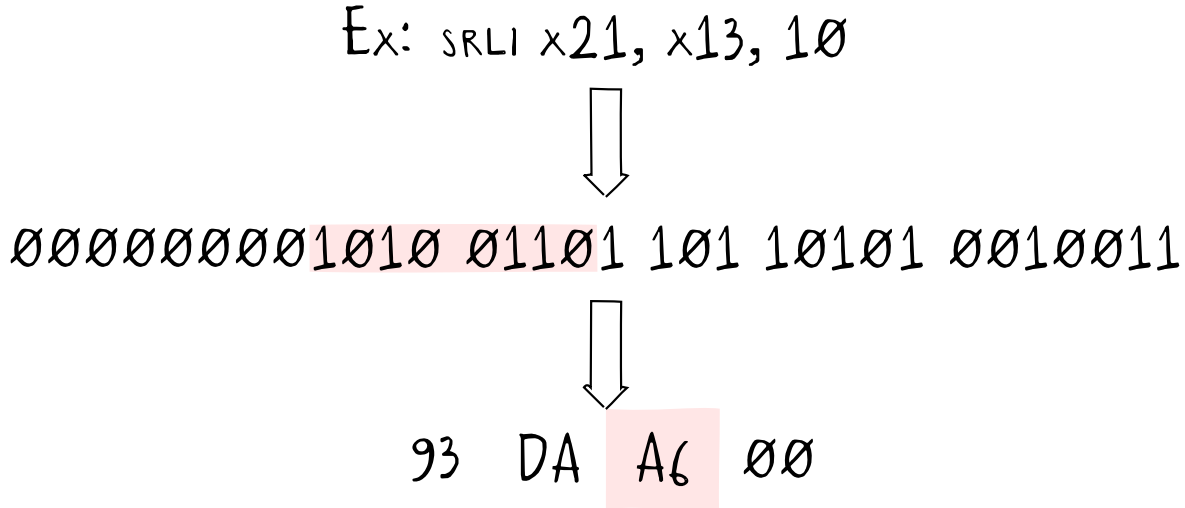


Instruções bit a bit e lógicas

- Deslocamento para direita lógico imediato (*shift right logical immediate*)

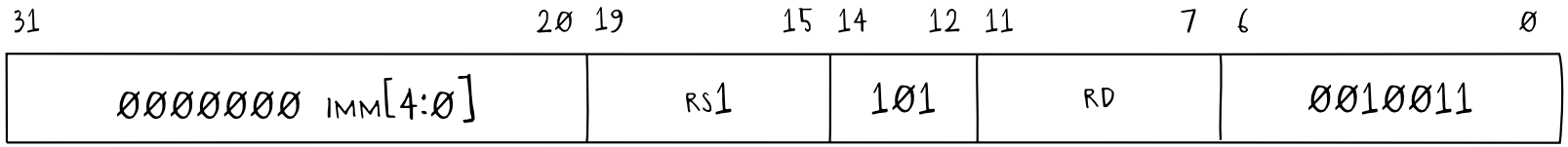



```
srli rd, rs1, imm:
    rd = rs1 >> imm
```

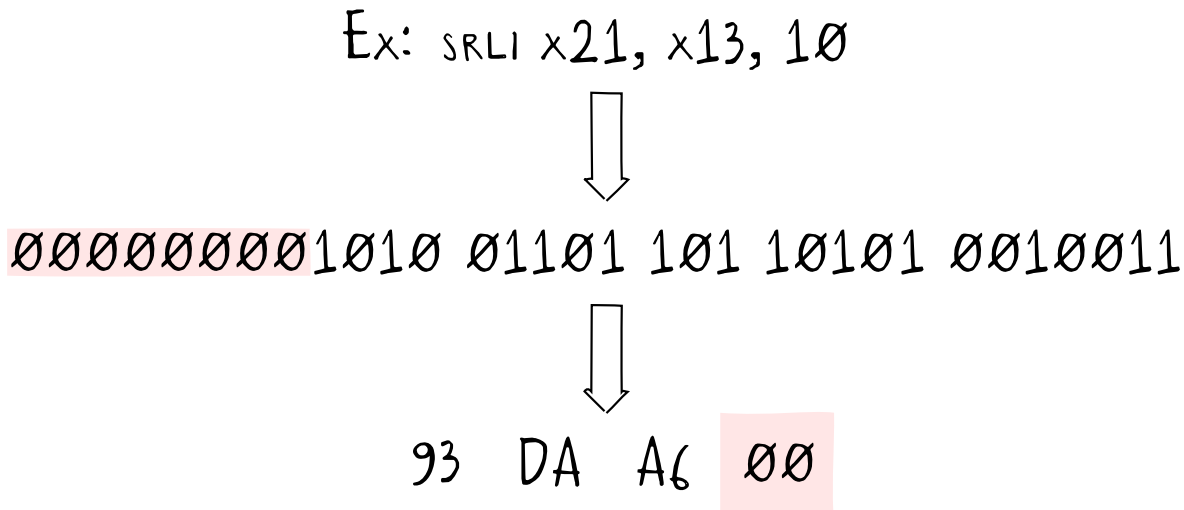


Instruções bit a bit e lógicas

- Deslocamento para direita lógico imediato (*shift right logical immediate*)

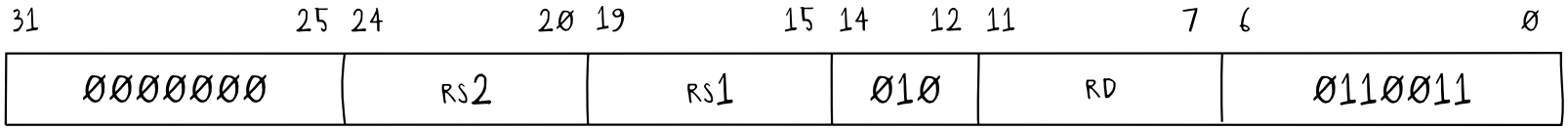



```
srli rd, rs1, imm:
    rd = rs1 >> imm
```



Instruções bit a bit e lógicas

- Setar se for menor com sinal (*set less than*)




```
slt rd, rs1, rs2:
    rd = rs1 < rs2

sltz rd, rs1:
    slt rd, rs1, x0

sgtz rd, rs2:
    slt rd, x0, rs2
```

Ex: SLT x10, x20, x30

⇩

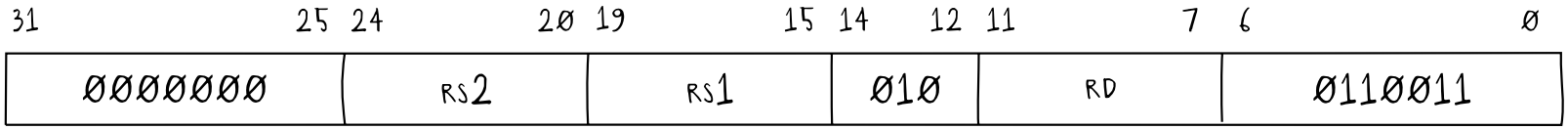
0000000 11110 10100 010 01010 0110011

⇩

33 25 EA 01

Instruções bit a bit e lógicas

- Setar se for menor com sinal (*set less than*)




```
slt rd, rs1, rs2:
    rd = rs1 < rs2

sltz rd, rs1:
    slt rd, rs1, x0

sgtz rd, rs2:
    slt rd, x0, rs2
```

Ex: SLT x10, x20, x30

⇩

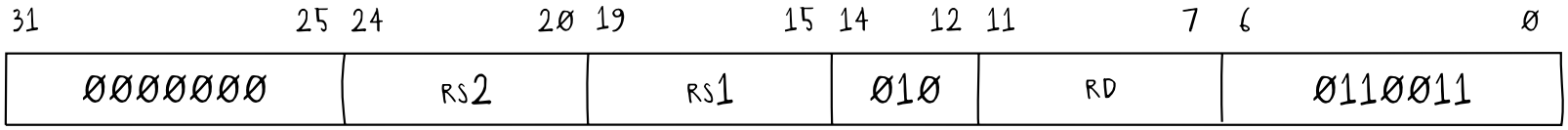
0000000 11110 10100 010 01010 0110011

⇩

33 25 EA 01

Instruções bit a bit e lógicas

- Setar se for menor com sinal (*set less than*)




```
slt rd, rs1, rs2:
    rd = rs1 < rs2

sltz rd, rs1:
    slt rd, rs1, x0

sgtz rd, rs2:
    slt rd, x0, rs2
```

Ex: SLT x10, x20, x30

⇩

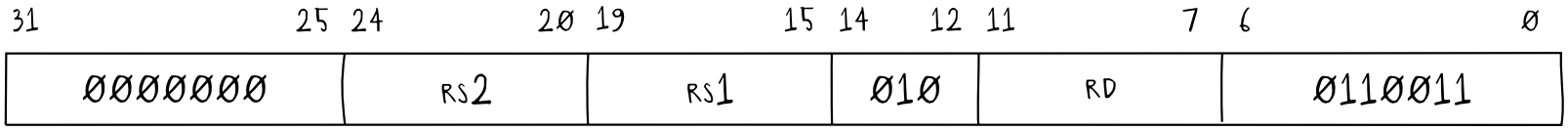
0000000 11110 10100 010 01010 0110011

⇩

33 25 EA 01

Instruções bit a bit e lógicas

- Setar se for menor com sinal (*set less than*)




```
slt rd, rs1, rs2:
    rd = rs1 < rs2

sltz rd, rs1:
    slt rd, rs1, x0

sgtz rd, rs2:
    slt rd, x0, rs2
```

Ex: SLT x10, x20, x30

⇩

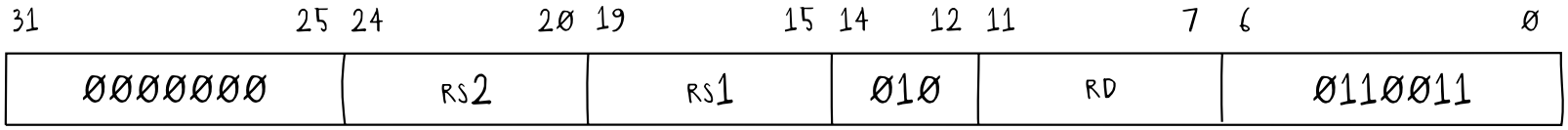
0000000 11110 10100 010 01010 0110011

⇩

33 25 EA 01

Instruções bit a bit e lógicas

- Setar se for menor com sinal (*set less than*)




```
slt rd, rs1, rs2:
    rd = rs1 < rs2

sltz rd, rs1:
    slt rd, rs1, x0

sgtz rd, rs2:
    slt rd, x0, rs2
```

Ex: SLT x10, x20, x30

⇩

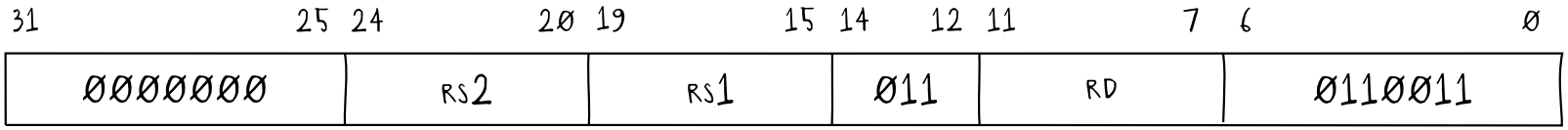
0000000 11110 10100 010 01010 0110011

⇩

33 25 EA 01

Instruções bit a bit e lógicas

- Setar se for menor sem sinal (*set less than unsigned*)




```
sltu rd, rs1, rs2:
    rd = rs1 < rs2

snez rd, rs2:
    sltu rd, x0, rs2
```

Ex: sltu x20, x10, x5

⇩

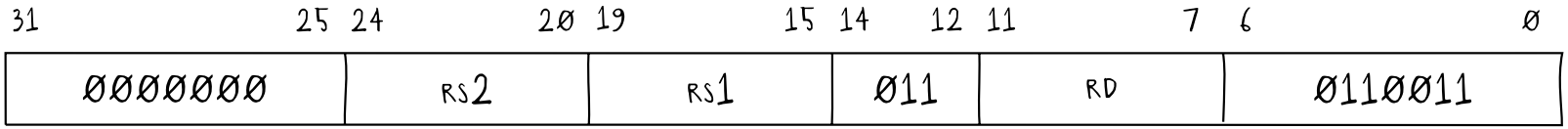
0000000 00101 01010 011 10100 0110011

⇩

33 3A 55 00

Instruções bit a bit e lógicas

- Setar se for menor sem sinal (*set less than unsigned*)




```
sltu rd, rs1, rs2:
    rd = rs1 < rs2

snez rd, rs2:
    sltu rd, x0, rs2
```

Ex: SLTU x20, x10, x5

⇩

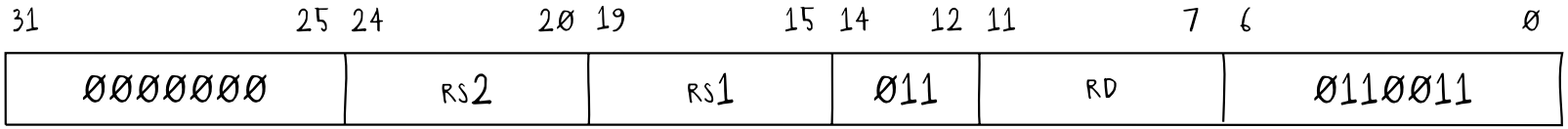
0000000 00101 01010 011 10100 0110011

⇩

33 3A 55 00

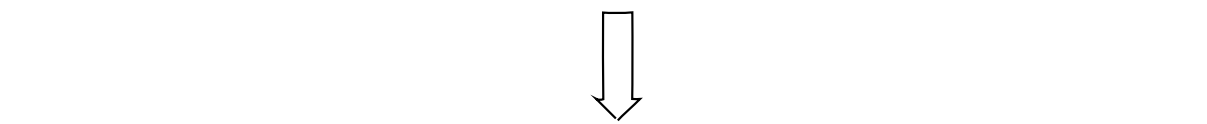
Instruções bit a bit e lógicas

- Setar se for menor sem sinal (*set less than unsigned*)

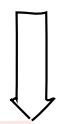



```
sltu rd, rs1, rs2:
    rd = rs1 < rs2

snez rd, rs2:
    sltu rd, x0, rs2
```

Ex: SLTU x20, x10, x5



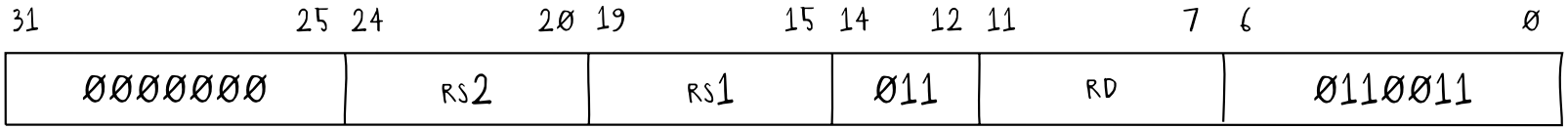
0000000 00101 01010 011 10100 0110011



33 3A 55 00

Instruções bit a bit e lógicas

- Setar se for menor sem sinal (*set less than unsigned*)




```
sltu rd, rs1, rs2:
    rd = rs1 < rs2

snez rd, rs2:
    sltu rd, x0, rs2
```

Ex: SLTU x20, x10, x5

⇩

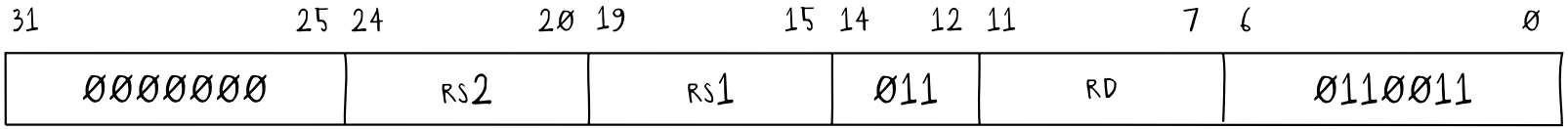
0000000 00101 01010 011 10100 0110011

⇩

33 3A 55 00

Instruções bit a bit e lógicas

- Setar se for menor sem sinal (*set less than unsigned*)




```
sltu rd, rs1, rs2:
    rd = rs1 < rs2

snez rd, rs2:
    sltu rd, x0, rs2
```

Ex: SLTU x20, x10, x5

⇩

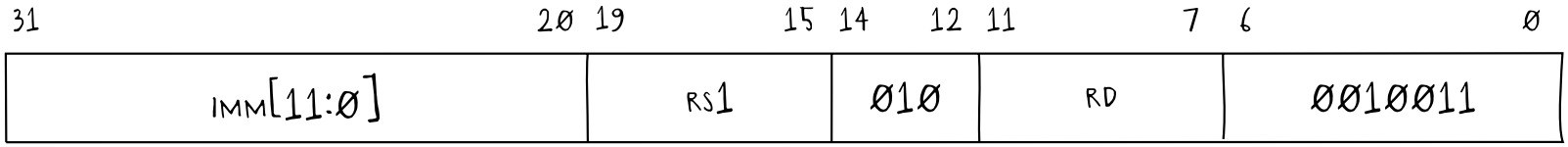
0000000 00101 01010 011 10100 0110011

⇩

33 3A 55 00

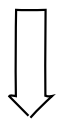
Instruções bit a bit e lógicas

- Setar se for menor com sinal e imediato (*set less than immediate*)

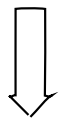



```
slti rd, rs1, imm:
    rd = rs1 < sign_extension(imm)
```

Ex: SLTI x7, x11, 0x567



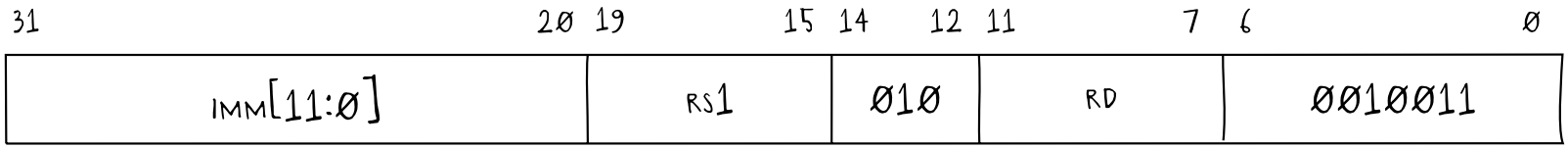
0101011 00111 01011 010 00111 0010011



93 A3 75 56

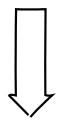
Instruções bit a bit e lógicas

- Setar se for menor com sinal e imediato (*set less than immediate*)

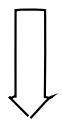



```
slti rd, rs1, imm:
    rd = rs1 < sign_extension(imm)
```

Ex: SLTI x7, x11, 0x567



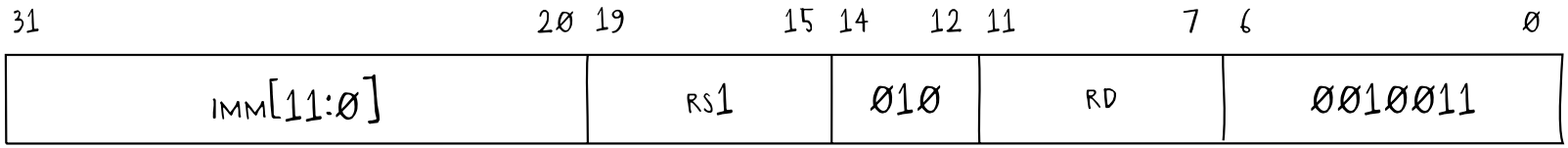
01010110 0111 01011 010 00111 0010011



93 A3 75 56

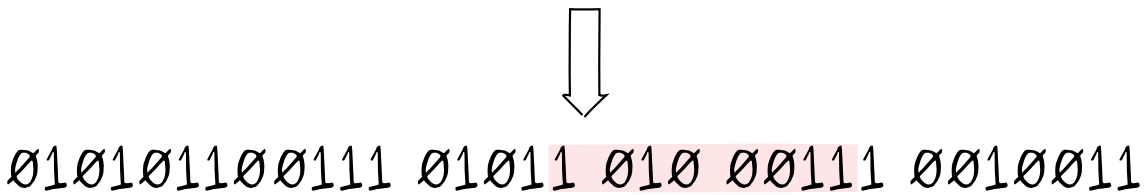
Instruções bit a bit e lógicas

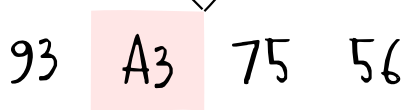
- Setar se for menor com sinal e imediato (*set less than immediate*)




```
slti rd, rs1, imm:
    rd = rs1 < sign_extension(imm)
```

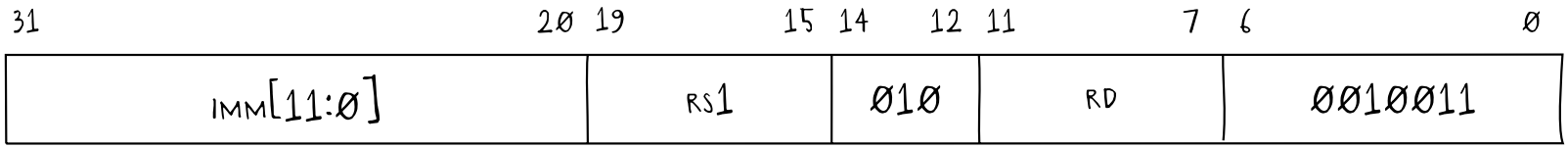
Ex: SLTI x7, x11, 0x567





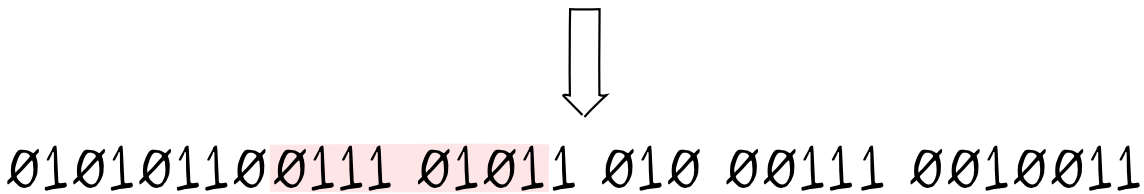
Instruções bit a bit e lógicas

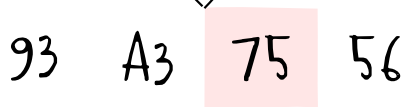
- Setar se for menor com sinal e imediato (*set less than immediate*)




```
slti rd, rs1, imm:
    rd = rs1 < sign_extension(imm)
```

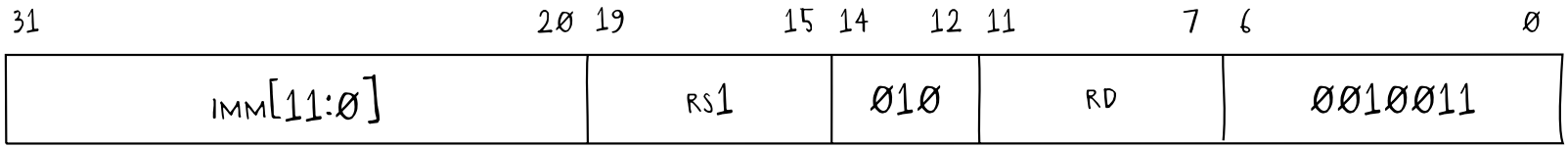
Ex: SLTI x7, x11, 0x567





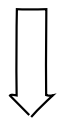
Instruções bit a bit e lógicas

- Setar se for menor com sinal e imediato (*set less than immediate*)

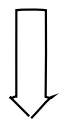



```
slti rd, rs1, imm:
    rd = rs1 < sign_extension(imm)
```

Ex: SLTI x7, x11, 0x567



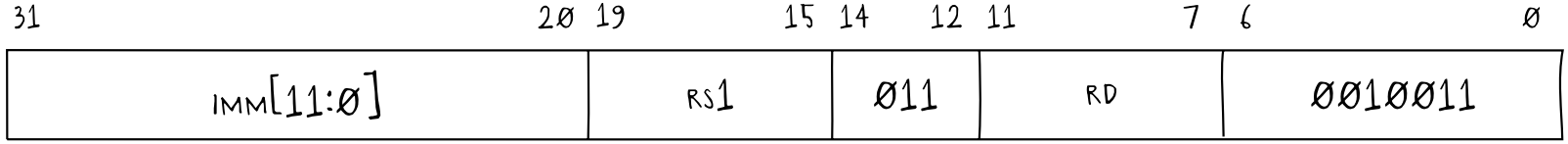
0101011 00111 01011 010 00111 0010011



93 A3 75 56

Instruções bit a bit e lógicas

- Setar se for menor sem sinal e imediato (*set less than immediate unsigned*)




```
sltiu rd, rs1, imm:
    rd = rs1 < sign_extension(imm)

seqz rd, rs1:
    sltiu rd, rs1, 1
```

Ex: SLTIU x5, x6, 0x789

⇩

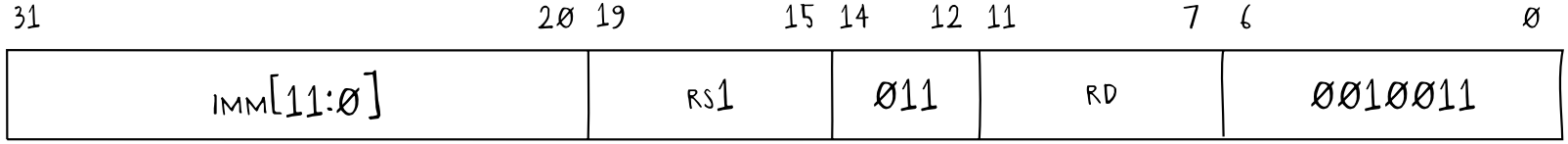
011110001001 00110 011 00101 0010011

⇩

93 32 93 78

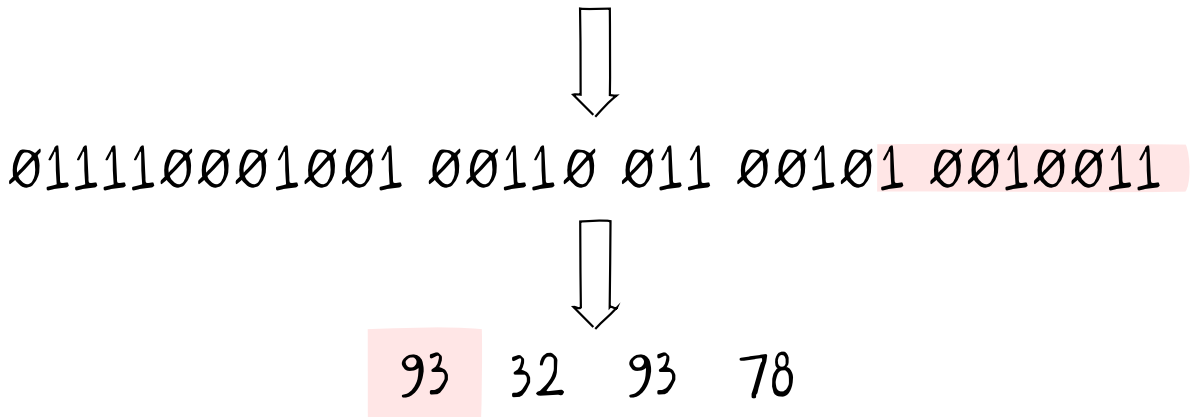
Instruções bit a bit e lógicas

- Setar se for menor sem sinal e imediato (*set less than immediate unsigned*)



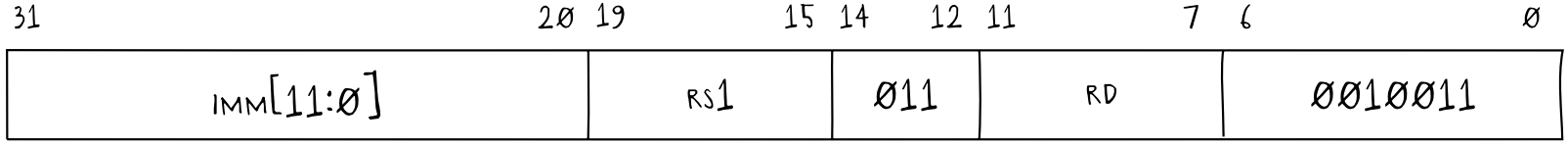

```
sltiu rd, rs1, imm:
    rd = rs1 < sign_extension(imm)

seqz rd, rs1:
    sltiu rd, rs1, 1
```

Ex: SLTIU x5, x6, 0x789

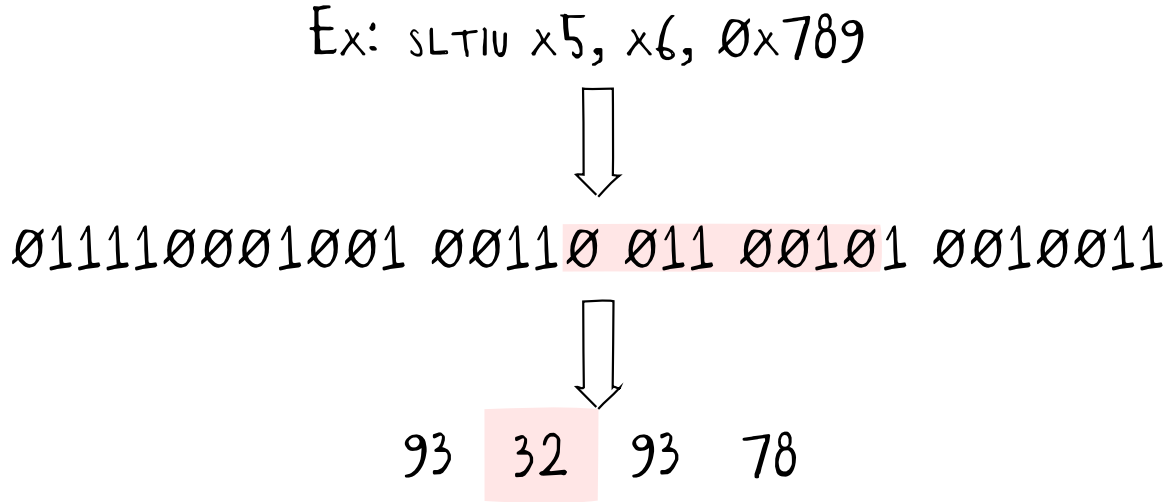


Instruções bit a bit e lógicas

- Setar se for menor sem sinal e imediato (*set less than immediate unsigned*)

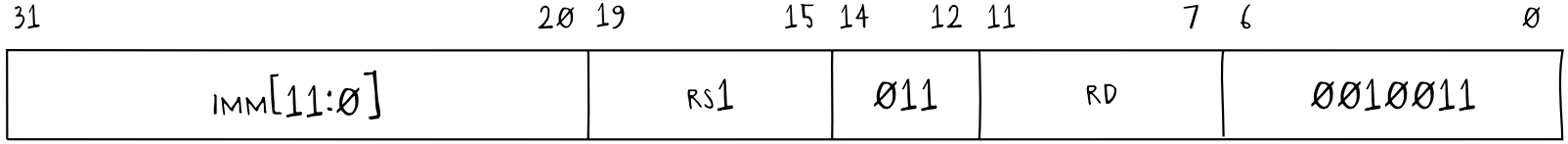



```
sltiu rd, rs1, imm:
    rd = rs1 < sign_extension(imm)

seqz rd, rs1:
    sltiu rd, rs1, 1
```

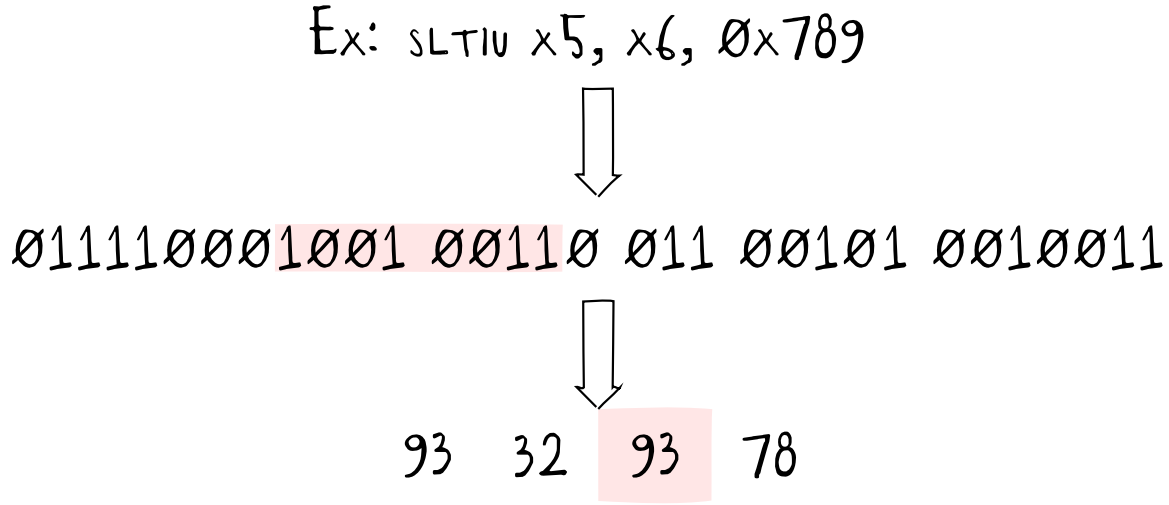


Instruções bit a bit e lógicas

- Setar se for menor sem sinal e imediato (*set less than immediate unsigned*)

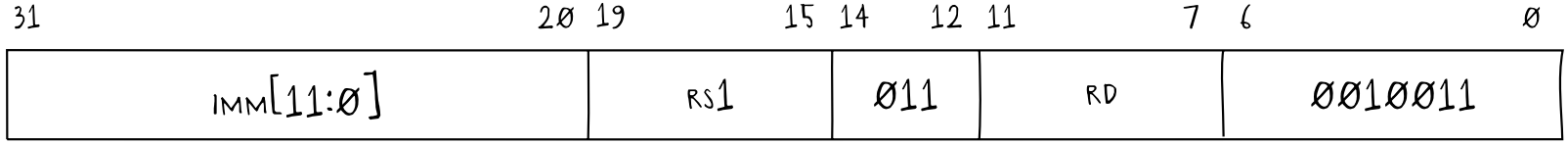



```
sltiu rd, rs1, imm:
    rd = rs1 < sign_extension(imm)

seqz rd, rs1:
    sltiu rd, rs1, 1
```



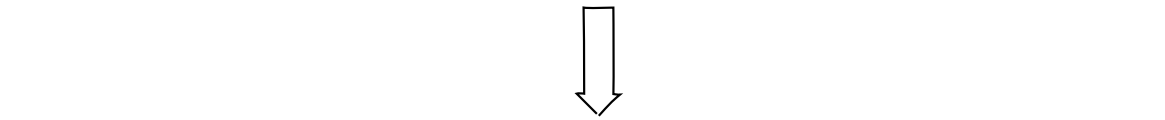
Instruções bit a bit e lógicas

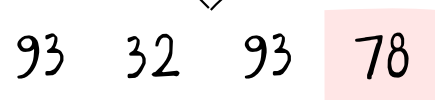
- Setar se for menor sem sinal e imediato (*set less than immediate unsigned*)




```
sltiu rd, rs1, imm:
    rd = rs1 < sign_extension(imm)

seqz rd, rs1:
    sltiu rd, rs1, 1
```

Ex: SLTIU x5, x6, 0x789





Exercício

- Simule os exemplos disponibilizados sobre as instruções de acesso à memória e operações bit a bit e lógicas
  - Verifique a instalação do QEMU para RISC-V de 32 bits
  - Algumas instruções ainda serão vistas, mas são necessárias para execução do programa, por isso, apenas considere as operações realizadas na função *main*
  - O montador pode realizar otimizações que alteram a emissão de algumas instruções, logo é importante entender estas mudanças e como elas afetam o comportamento do código-fonte

Exercício

- Entenda a diferença entre os resultado das operações realizadas sobre o mesmo valor no código-fonte abaixo

```c
// Bibliotecas padronizadas
#include <stdio.h>
#include <stdint.h>
// Função principal
int main() {
    // Declaração das variáveis
    int32_t s = -1;
    uint32_t u = -1;
    // Operações de deslocamento para direita
    uint32_t rs = s >> 31;
    uint32_t ru = u >> 31;
    // Impressão dos resultados
    printf("r >> 31 = %08x, u >> 31 = %08x\n", rs, ru);
    // Retornando 0 (sucesso)
    return 0;
}
```