Module 4:

# Design of the instruction format

Introduction to computers II

**José Manuel Mendías Cuadros**
*Dpto. Arquitectura de Computadores y Automática*
*Universidad Complutense de Madrid*

## Outline

- ✓ RISC-V instruction formats.
- ✓ Field encoding.
- ✓ From assembly to machine code.
- ✓ From machine code to assembly.

These slides are based on:

- S.L. Harris and D. Harris. *Digital Design and Computer Architecture. RISC-V Edition.*
- D.A. Patterson and J.L. Hennessy. *Computer Organization and Design. RISC-V Edition.*

# Instruction formats

- The instruction formats determine the location and encoding of the fields in a machine instruction.
  - The greater the regularity, the simpler the digital circuit to decode it.

- The RISC-V instruction formats have the following fields:
  - Operation code (op): indicates the kind of instruction.
  - Function code (funct3 y funtc7): determines the specific instruction within its kind.
  - Register operands (rs1, rs2, rd): encodes specific registers.
  - Immediate operand (imm): contains an immediate operand represented either in pure binary or C2, depending on the instruction.
    - This field may be split in two separate parts within the instruction.

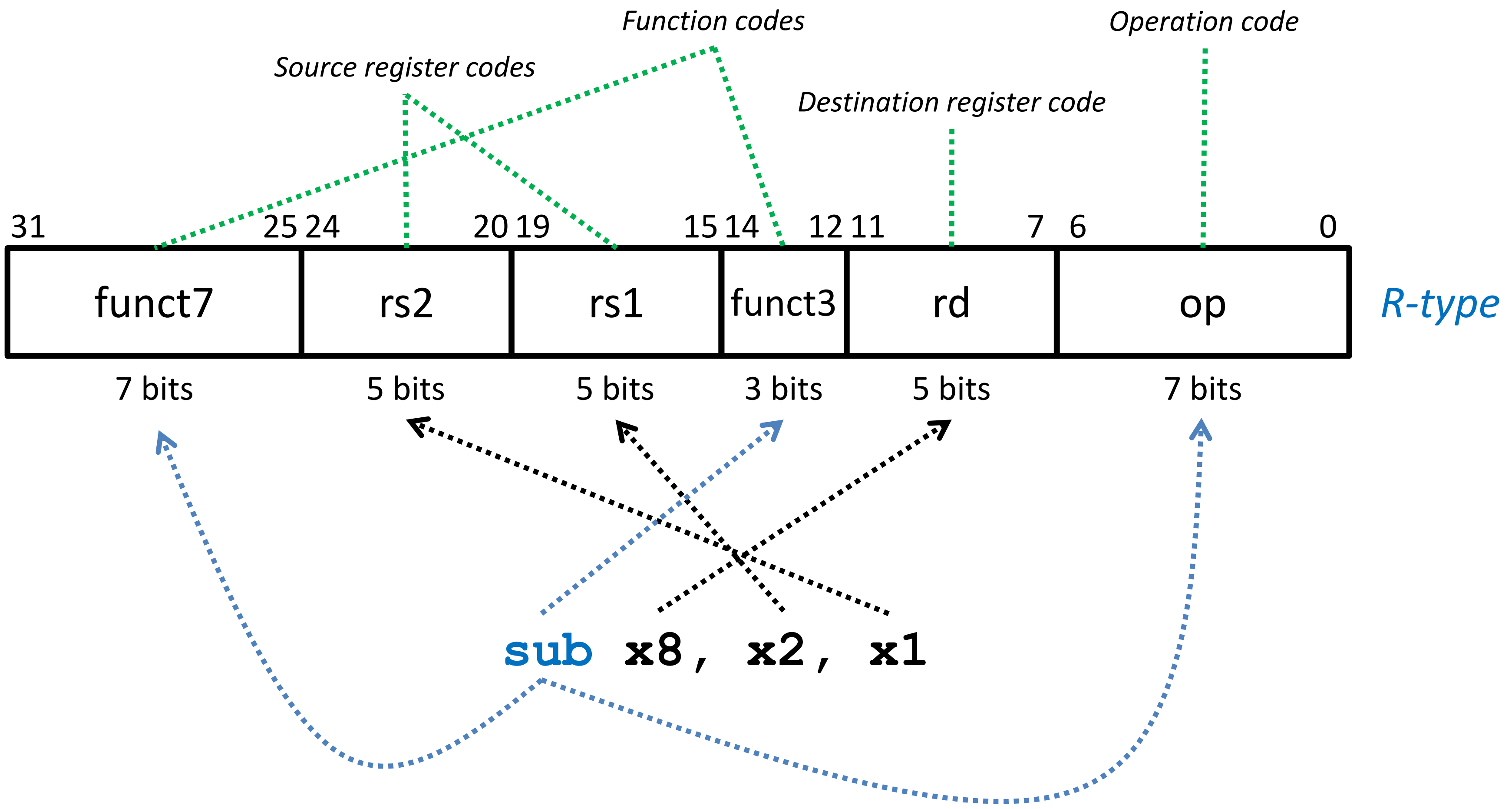
# Instruction formats

- There are only 4 formats in RISC-V:
  - R-type: for instructions with 3 register operands (2 sources y 1 destination).
    - Arithmetic-logic and shift.
  - I-type: for instructions with 2 register operands (one source and one destination), and 1 short immediate source operand (12 bits).
    - Arithmetic-logic and shift, with immediate operand, load and **`jalr`**.
  - S/B-type: for instructions with 2 register source operands and 1 short immediate operand (12/13 bits).
    - Store (S) and branch (B).
  - U/J-type: for instructions with 1 register destination operand and 1 long immediate operand (20/21 bits).
    - **`lui`** (U) , **`auipc`** (U) and **`jal`** (J).

- All the formats have a fixed width of 32 bits.

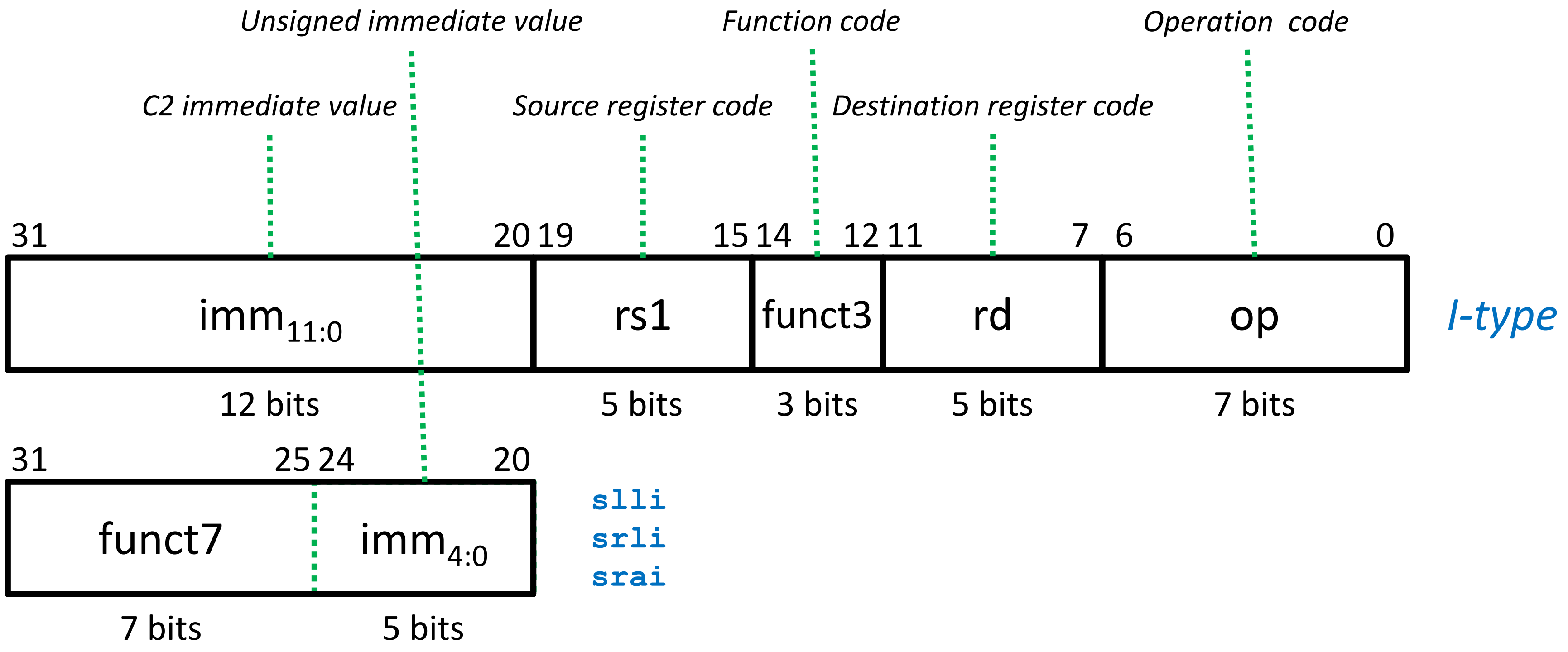
# R-type format

- Used to encode instructions with 3 register operands (2 sources y 1 destination).



# I-type format

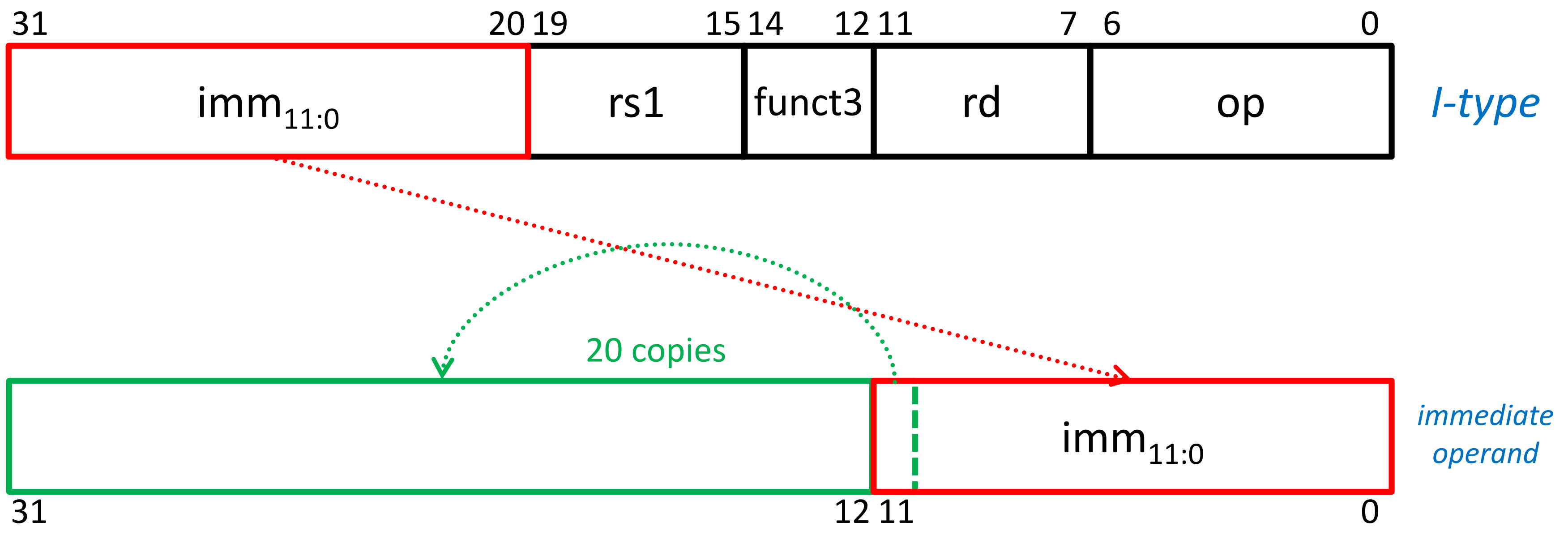
- Used to encode instructions with 2 register operands (one source and one destination), and 1 short immediate source operand (12 bits).
  - In arithmetic, logical, and load instructions, the immediate value sign will be extended to 32 bits.
  - In shift instructions, only the 5 least significant bits of the immediate value are used; the 7 most significant bits are used as a function field.



# I-type format

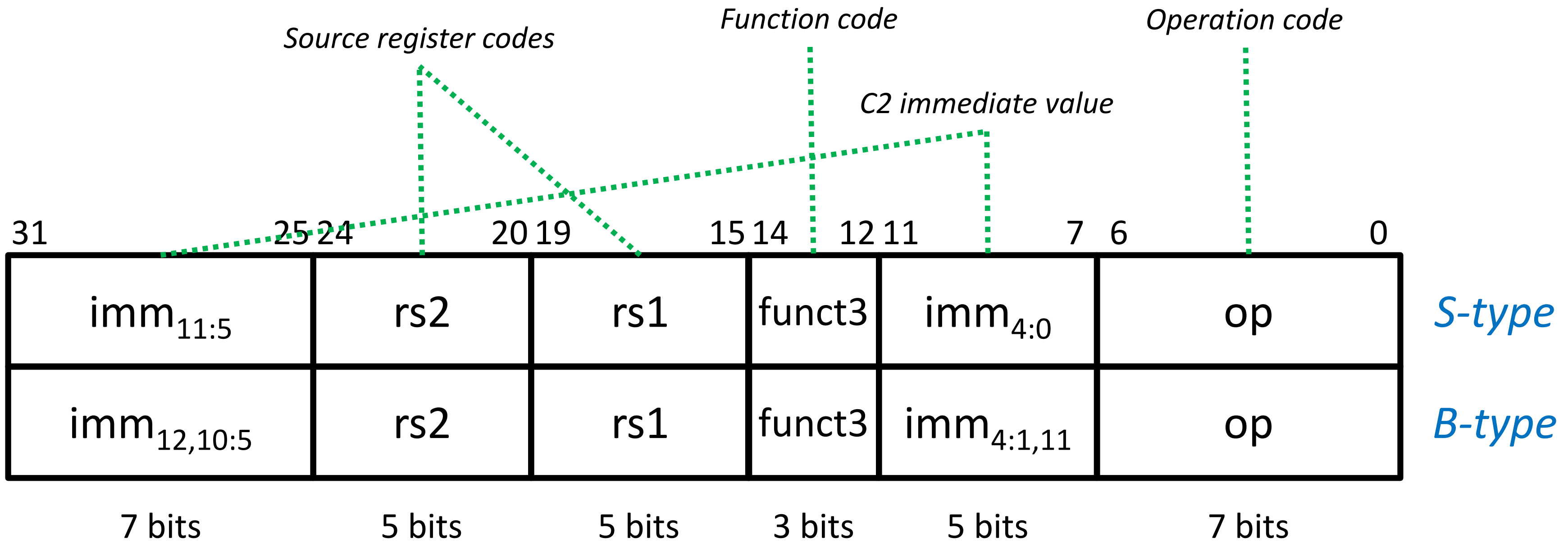
## Immediate operand encoding

- In order to get the 32b effective immediate operand in an I-type instruction:
  - The sign is extended to 32 bits (adding 20 bits).



## S/B-type format

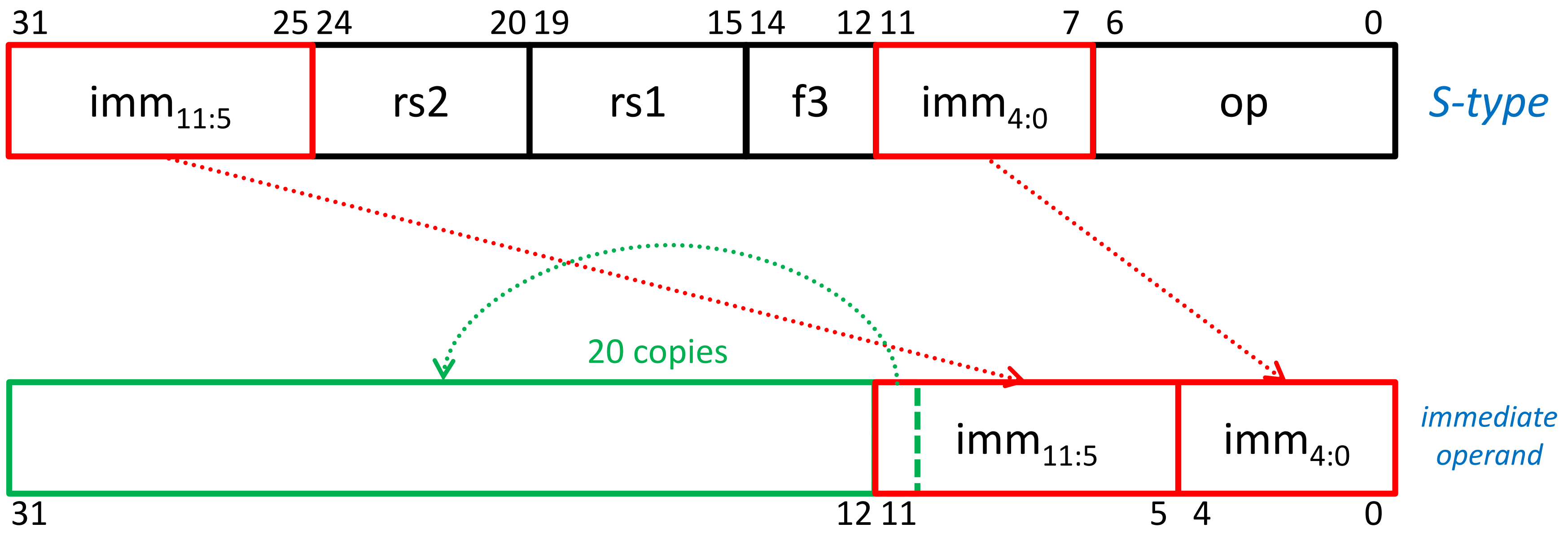
- Used to encode instructions with 2 register source operands and 1 short immediate operand (12/13 bits).
  - In both formats, the immediate value is split in 2 fields and its sign is extended to 32 bits.
  - In the B-type format, only the 12 most significant bits (of the 13) are stored in the instruction.
    - Instructions are placed in multiple-of-4 addresses, which end with 2 zeros. One 0 is removed for compatibility with the RVC extension (16b instructions).



# S-type format

## Immediate operand encoding

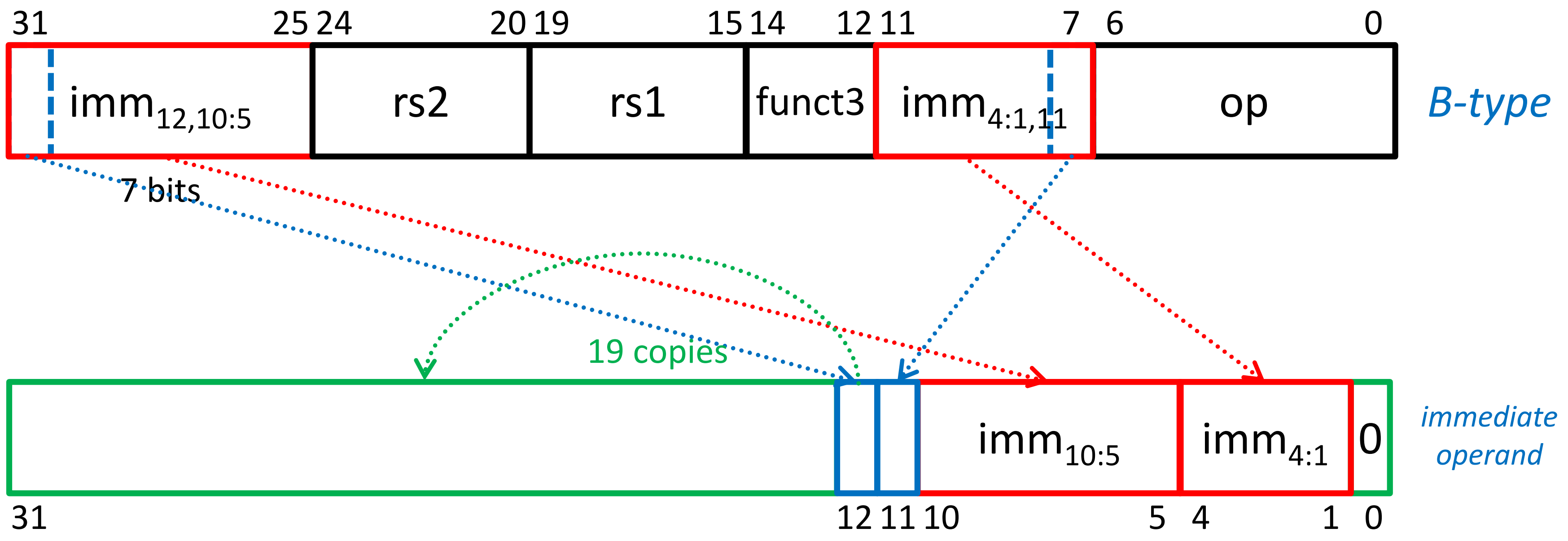
- In order to get the 32b effective immediate operand in a S-type instruction:
  - The two immediate fields are concatenated.
  - The sign is extended to 32 bits (adding 20 bits).



# B-type format

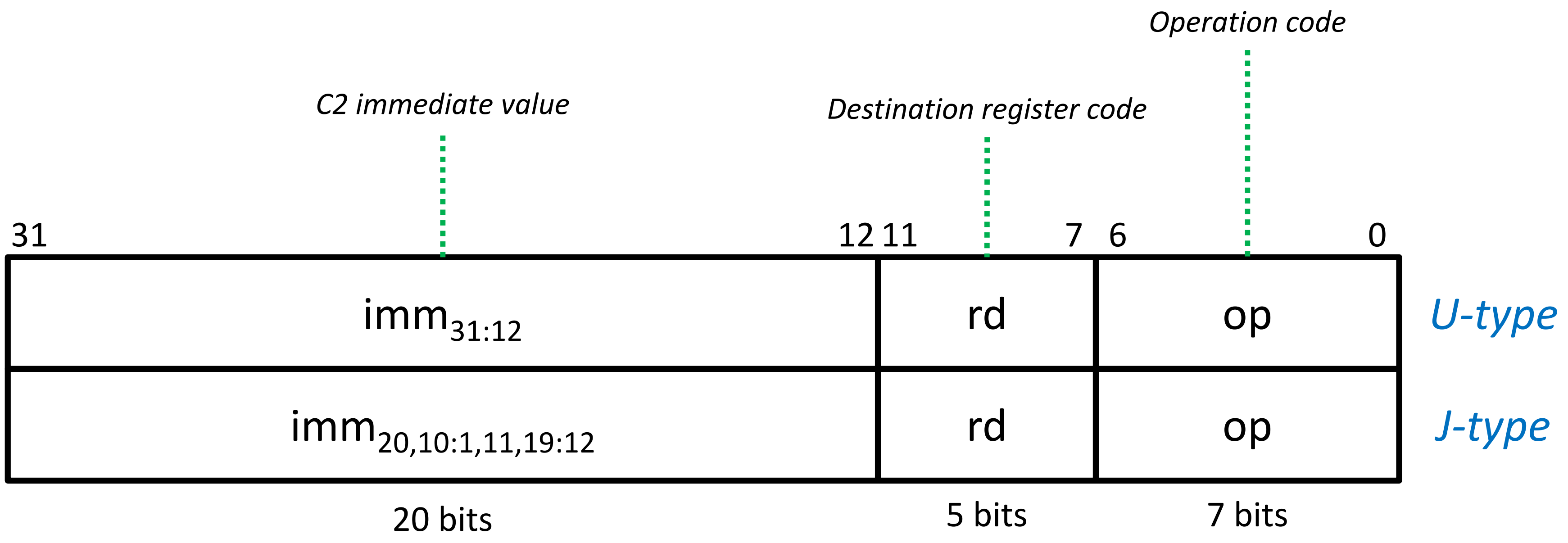
## Immediate operand encoding

- In order to get the 32b effective immediate operand in a B-type instruction:
  - The two immediate fields are concatenated and reordered
  - The implicit zero is added to the right.
  - The sign is extended to 32 bits (adding 19 bits).



# U/J-type format

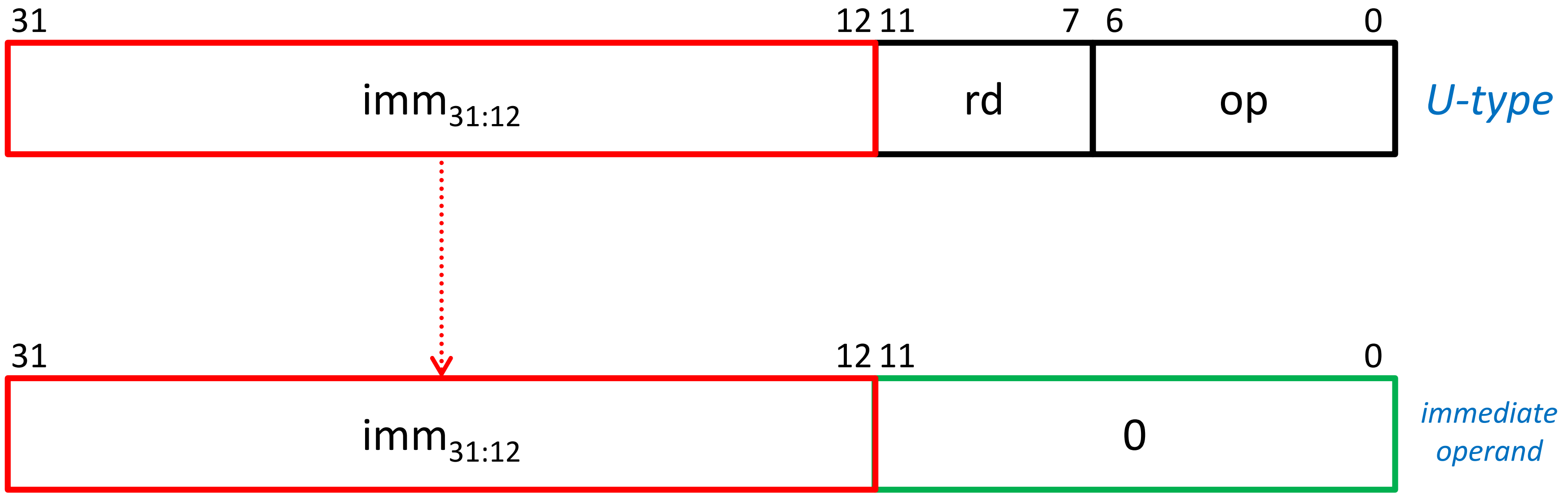
- Used to encode instructions with 1 register destination operand and 1 long immediate operand (20/21 bits).
  - In the U-type format, the immediate value is completed by adding 0 to the right, up to 32 bits.
  - In the J-type format, only the 20 most significant bits (of the 21) are stored in the instruction and its sign is extended to 32 bits.
    - Instructions are placed in multiple-of-4 addresses, which end with 2 zeros. One 0 is removed for compatibility with the RVC extension (16b instructions).



## U-type format

### Immediate operand encoding

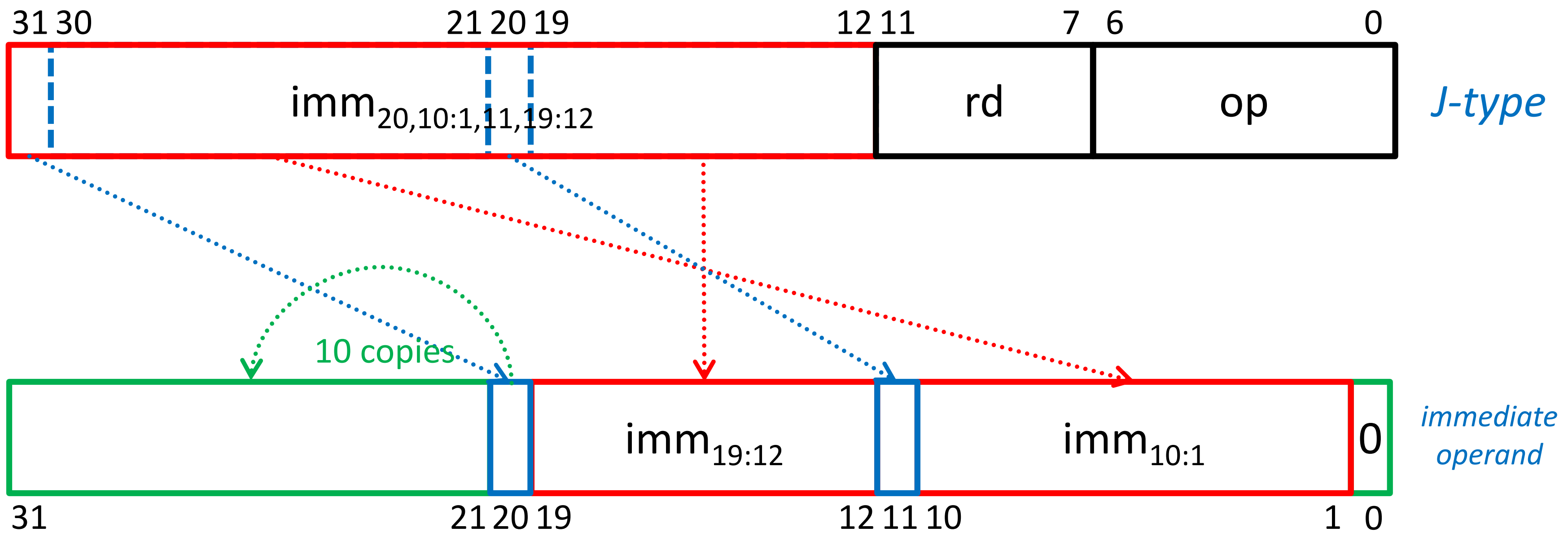
- In order to get the 32b effective immediate operand in a U-type instruction:
  - Twelve zeros are added to the right of the immediate field.



# J-type format

## Immediate operand encoding

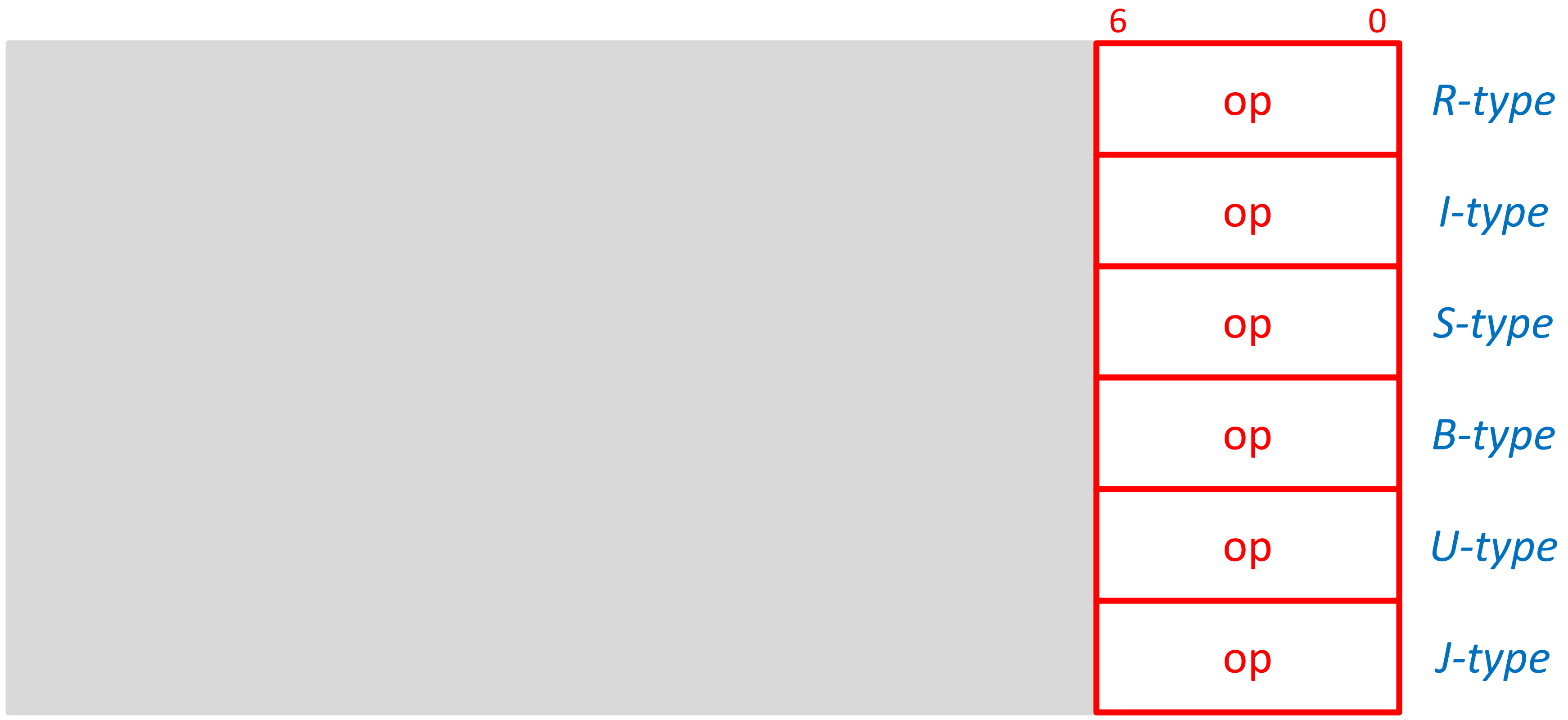
- In order to get the 32b effective immediate operand in a J-type instruction:
  - The immediate field is reordered.
  - The implicit zero is added to the right.
  - The sign is extended to 32 bits (adding 10 bits).



# Instruction formats

## Summary (i)

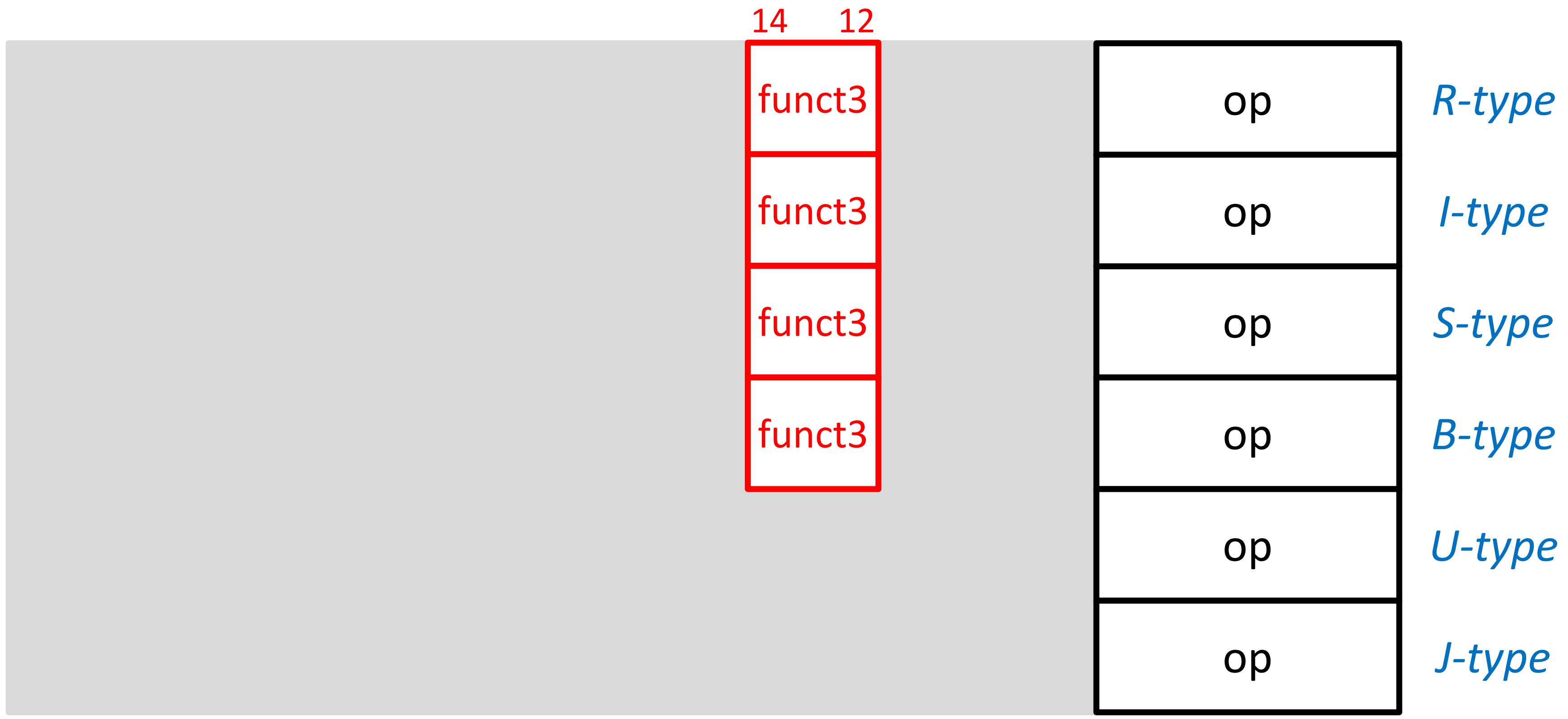
- The RISC-V formats are very regular:
  - The same fields are always in the same positions.



# Instruction formats

## Summary (i)

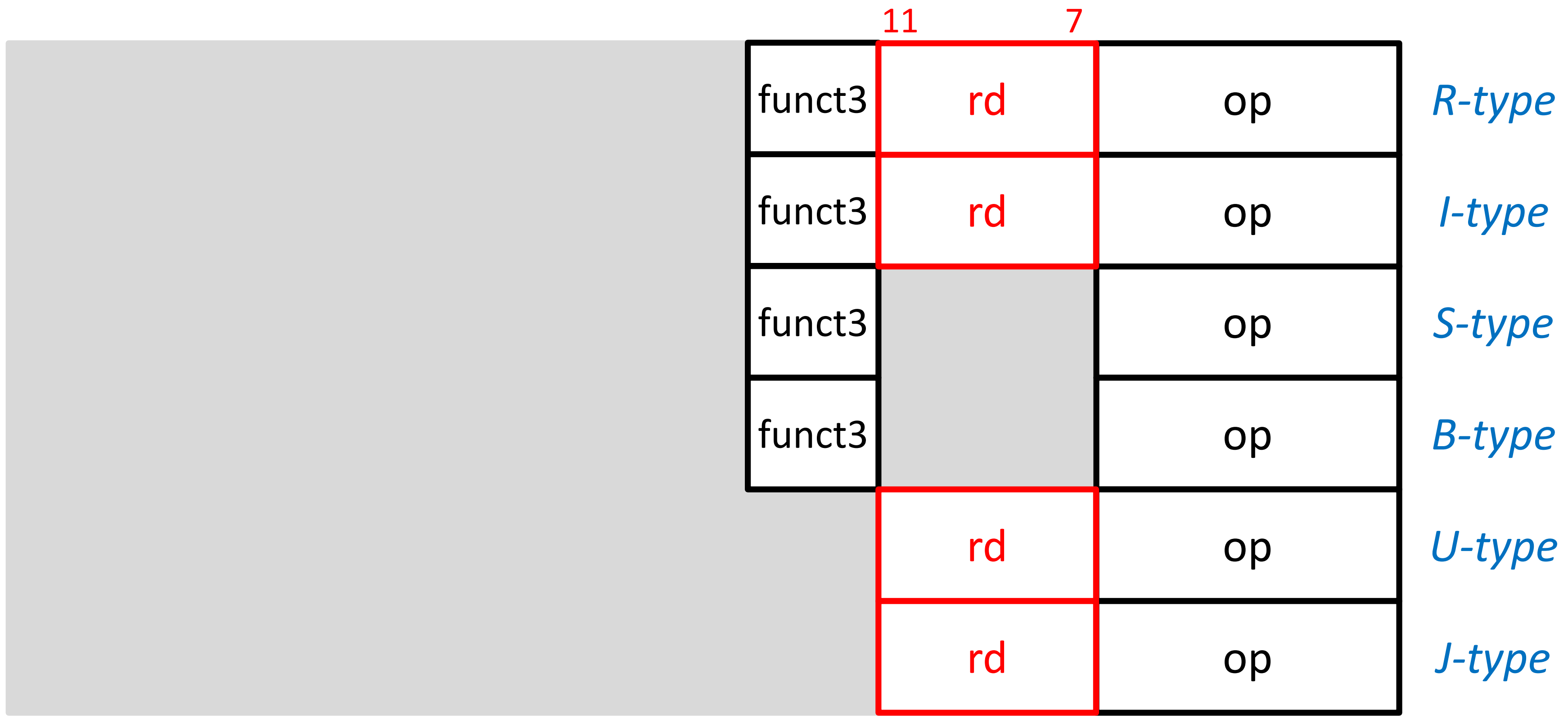
- The RISC-V formats are very regular:
  - The same fields are always in the same positions.



# Instruction formats

## Summary (i)

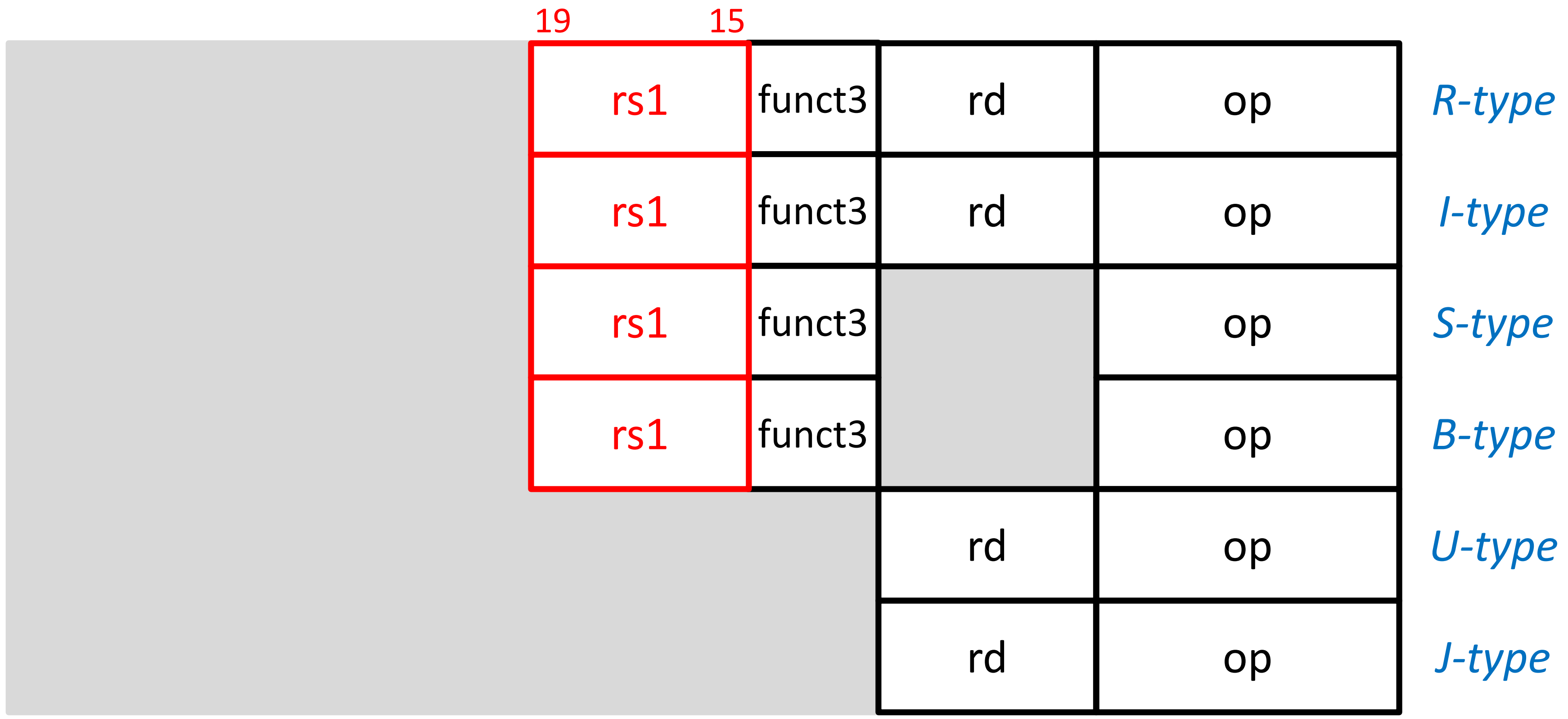
- The RISC-V formats are very regular:
  - The same fields are always in the same positions.



# Instruction formats

## Summary (i)

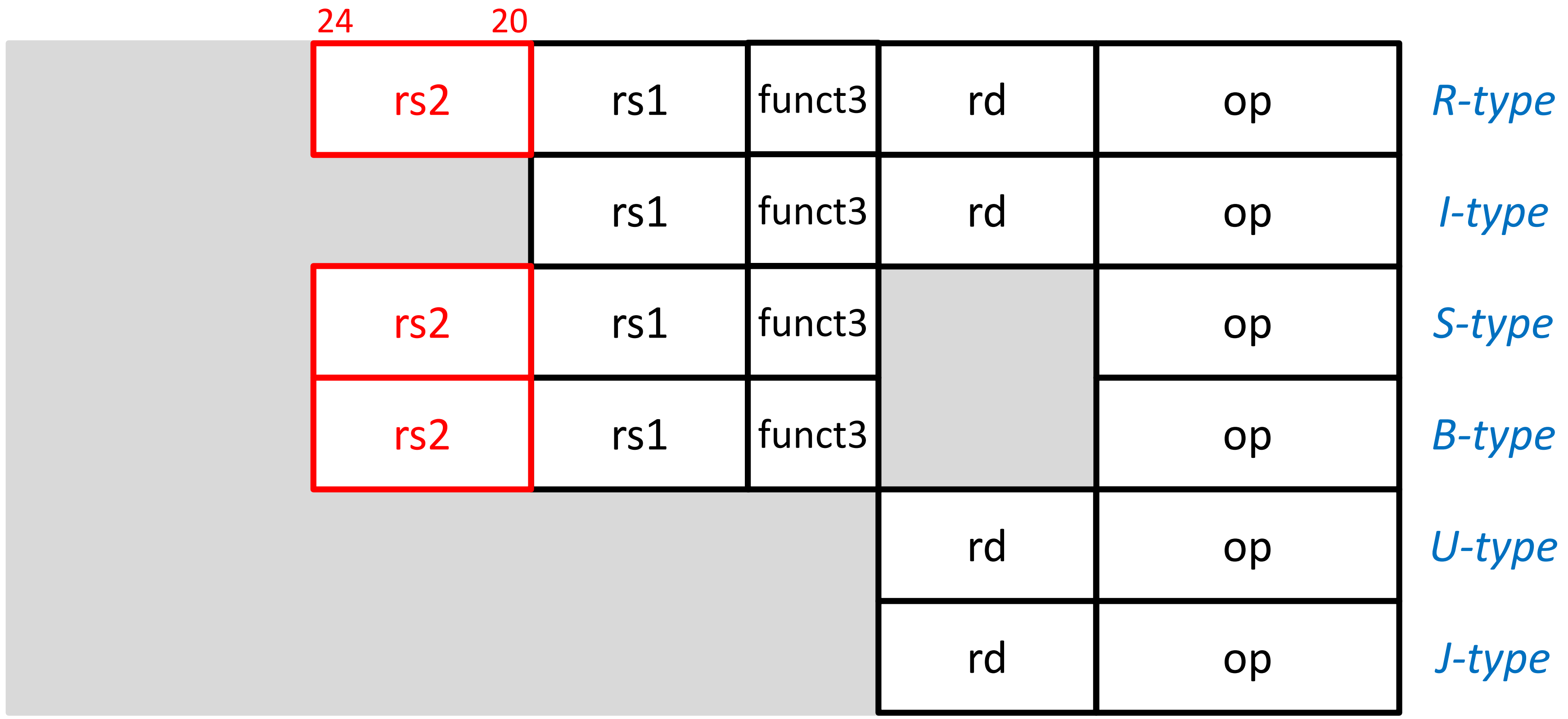
- The RISC-V formats are very regular:
  - The same fields are always in the same positions.



# Instruction formats

## Summary (i)

- The RISC-V formats are very regular:
  - The same fields are always in the same positions.



# Instruction formats

## Summary (i)

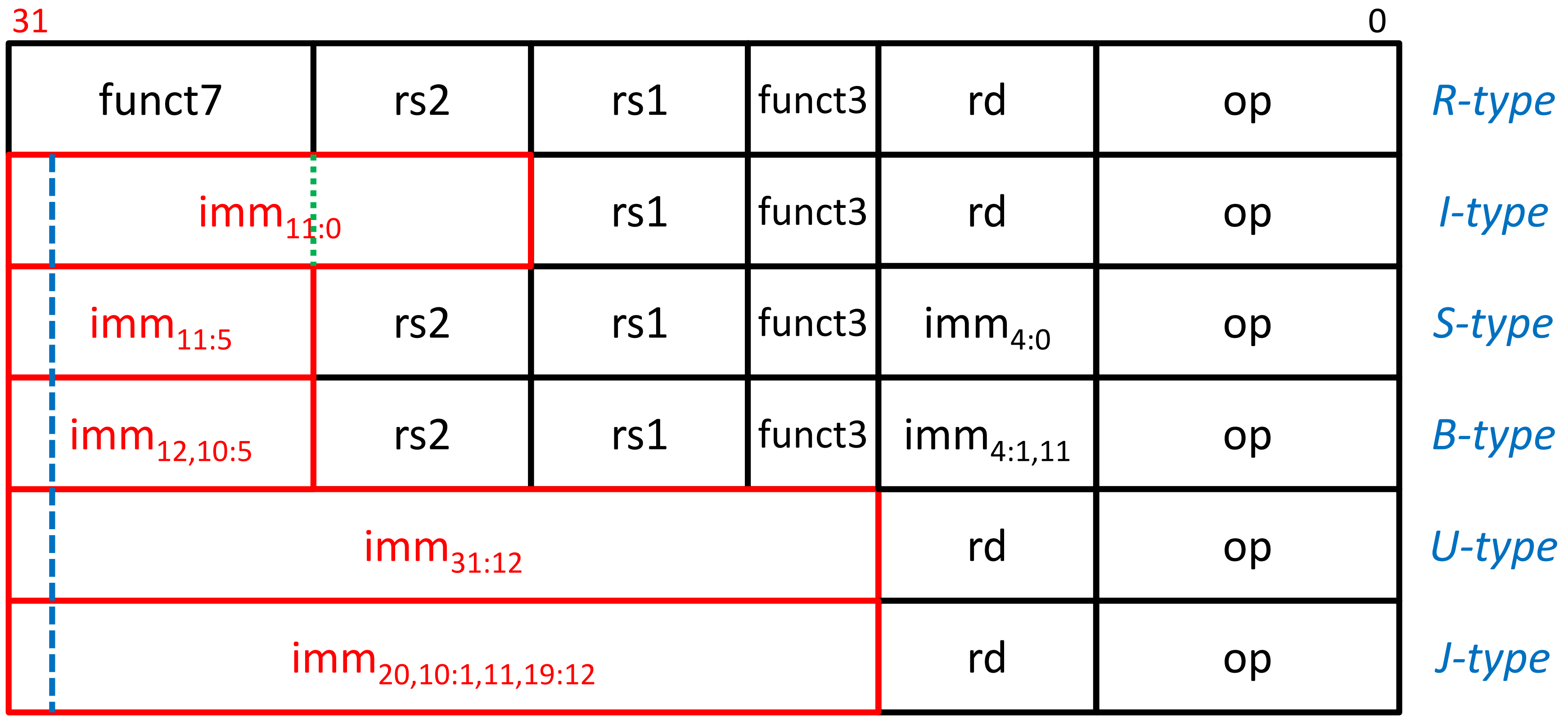
- The RISC-V formats are very regular:
  - The same fields are always in the same positions.
  - The unused fields are utilized for other purposes.

| 31 25 | | | | 11 7 | | |
|---|---|---|---|---|---|---|
| funct7 | rs2 | rs1 | funct3 | rd | op | *R-type* |
| | | rs1 | funct3 | rd | op | *I-type* |
| | rs2 | rs1 | funct3 | $imm_{4:0}$ | op | *S-type* |
| | rs2 | rs1 | funct3 | $imm_{4:1,11}$ | op | *B-type* |
| | | | | rd | op | *U-type* |
| | | | | rd | op | *J-type* |

# Instruction formats

## Summary (i)

- The RISC-V formats are very regular:
  - The same fields are always in the same positions.
  - The unused fields are utilized for other purposes.
  - The immediate value sign is always in position 31 of the instruction.



| 31 | | | | | 0 | |
|---|---|---|---|---|---|---|
| funct7 | rs2 | rs1 | funct3 | rd | op | *R-type* |
| $imm_{11:0}$ | | rs1 | funct3 | rd | op | *I-type* |
| $imm_{11:5}$ | rs2 | rs1 | funct3 | $imm_{4:0}$ | op | *S-type* |
| $imm_{12,10:5}$ | rs2 | rs1 | funct3 | $imm_{4:1,11}$ | op | *B-type* |
| $imm_{31:12}$ | | | | rd | op | *U-type* |
| $imm_{20,10:1,11,19:12}$ | | | | rd | op | *J-type* |

# Instruction formats

## Summary (i)

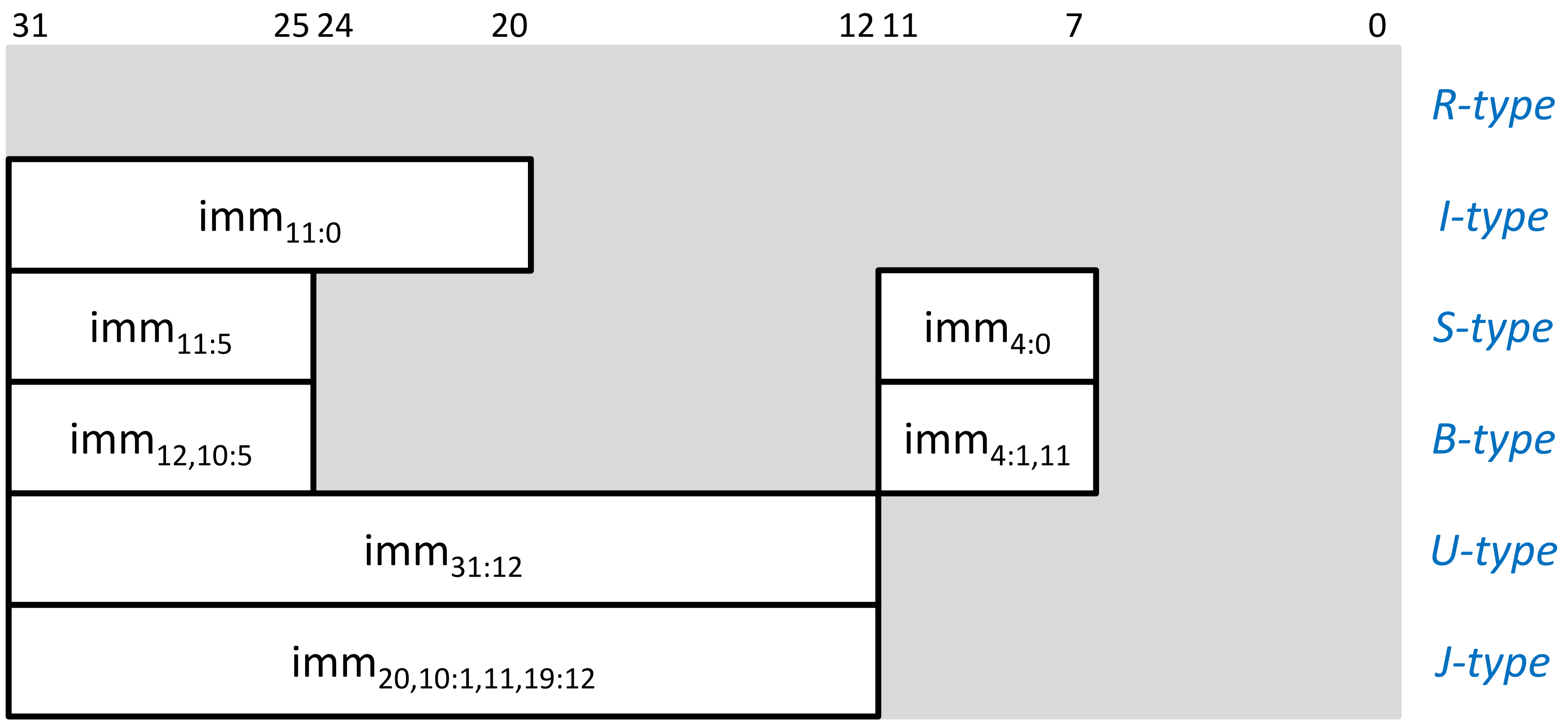
- The RISC-V formats are very regular:
  - The same fields are always in the same positions.
  - The unused fields are utilized for other purposes.
  - The immediate value sign is always in position 31 of the instruction.

| 31 25 | 24 20 | 19 15 | 14 12 | 11 7 | 6 0 | |
|---|---|---|---|---|---|---|
| funct7 | rs2 | rs1 | funct3 | rd | op | *R-type* |
| $imm_{11:0}$ | | rs1 | funct3 | rd | op | *I-type* |
| $imm_{11:5}$ | rs2 | rs1 | funct3 | $imm_{4:0}$ | op | *S-type* |
| $imm_{12,10:5}$ | rs2 | rs1 | funct3 | $imm_{4:1,11}$ | op | *B-type* |
| $imm_{31:12}$ | | | | rd | op | *U-type* |
| $imm_{20,10:1,11,19:12}$ | | | | rd | op | *J-type* |

# Instruction formats

## Summary (ii)

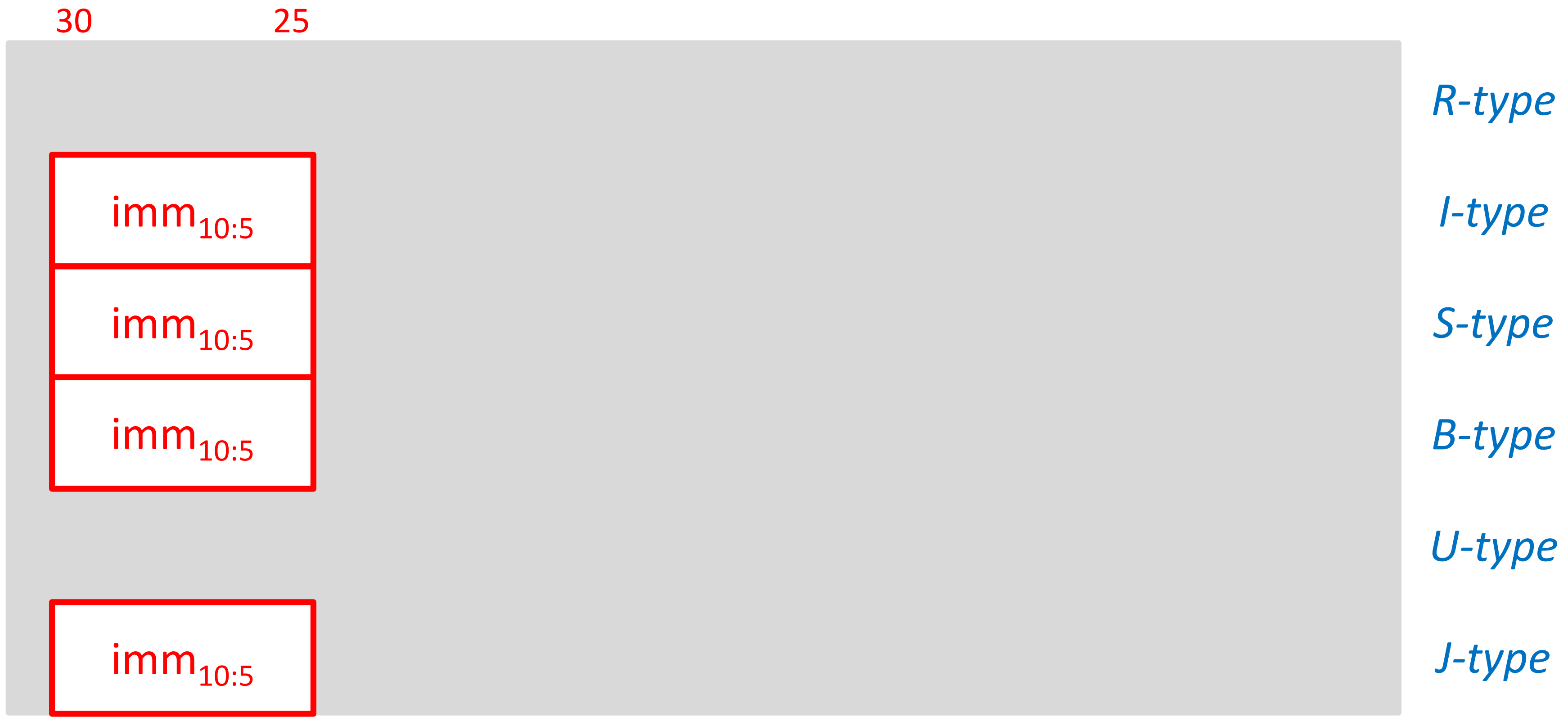
- The location of the immediate operands has been chosen in order to simplify the logic that performs the sign extension.
    - The same fragments are usually in the same position.



# Instruction formats

## Summary (ii)

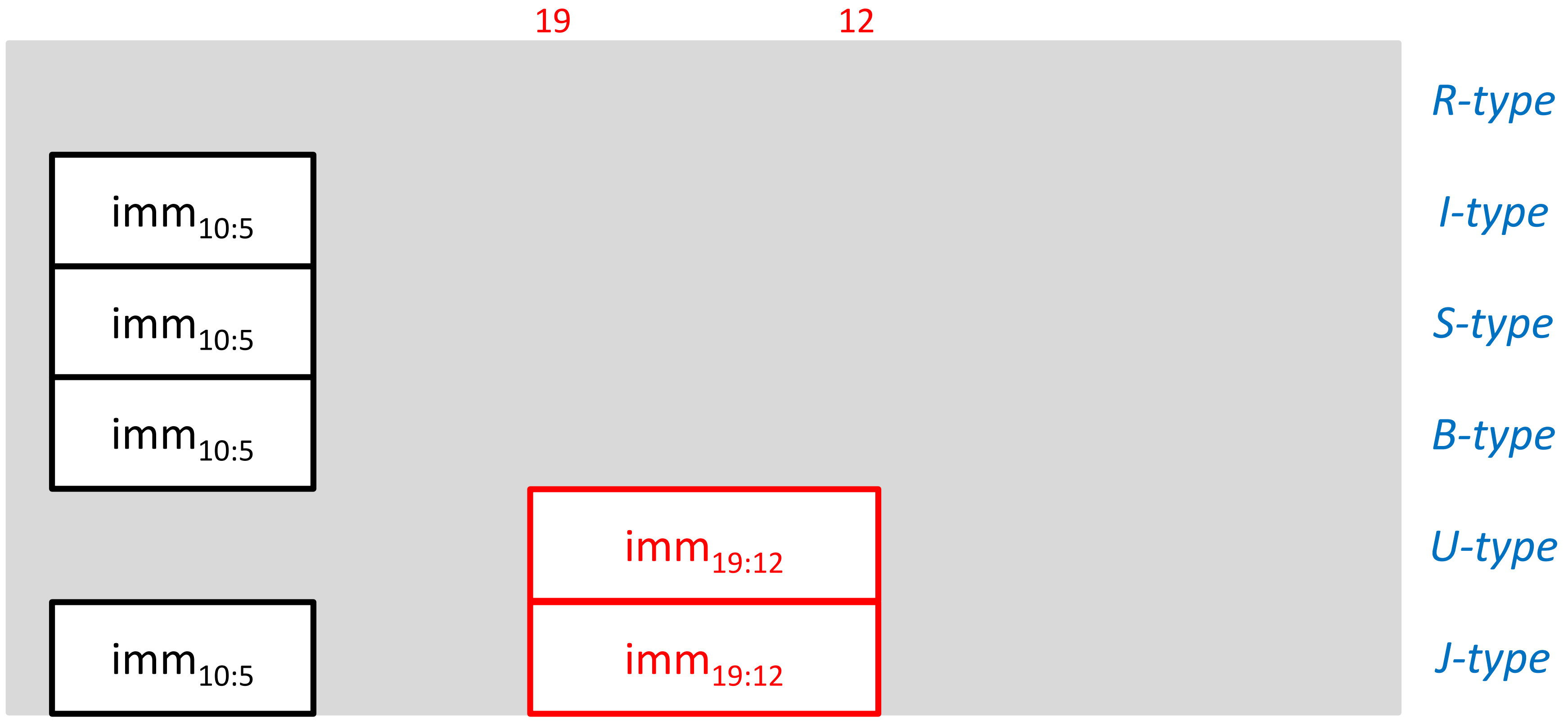
- The location of the immediate operands has been chosen in order to simplify the logic that performs the sign extension.
  - The same fragments are usually in the same position.



# Instruction formats

## Summary (ii)

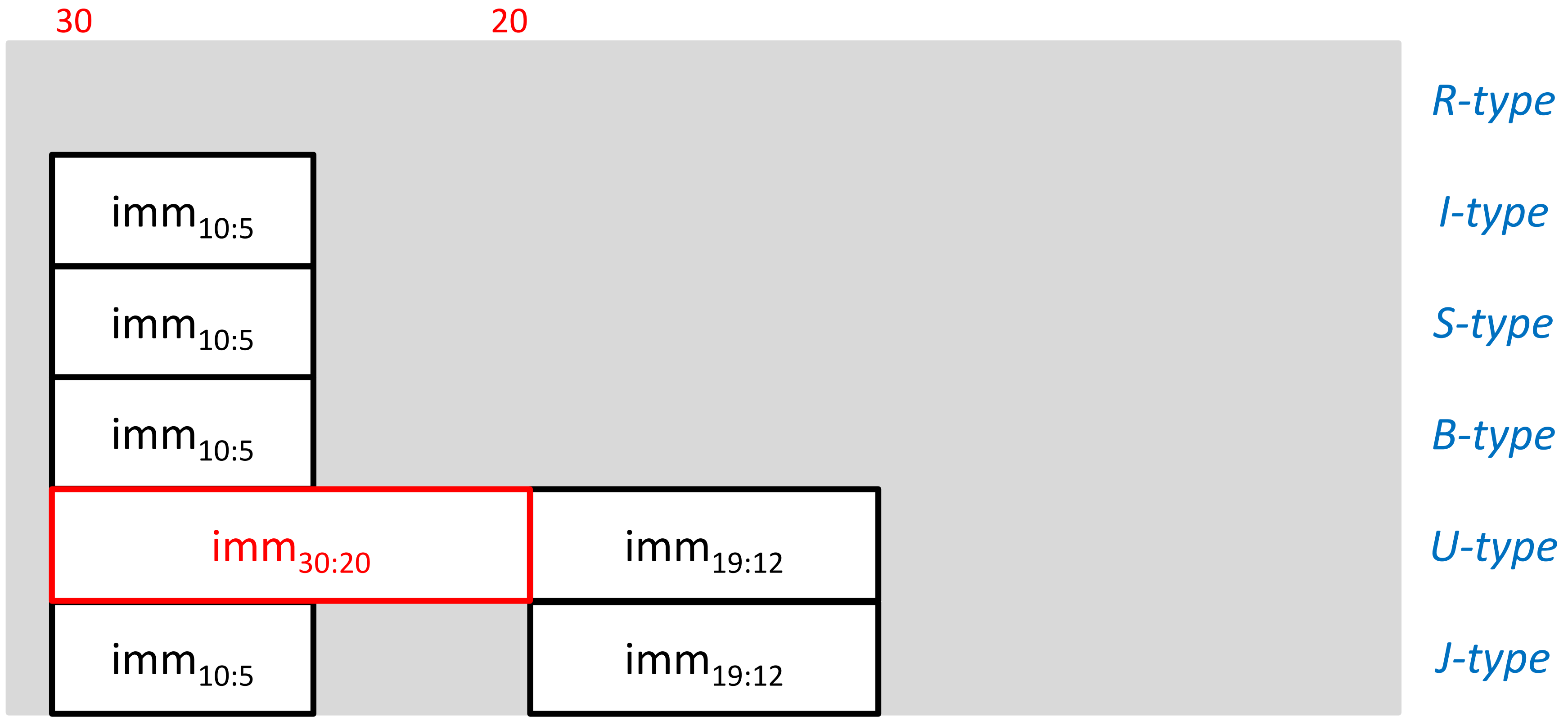
- The location of the immediate operands has been chosen in order to simplify the logic that performs the sign extension.
  - The same fragments are usually in the same position.



# Instruction formats

## Summary (ii)

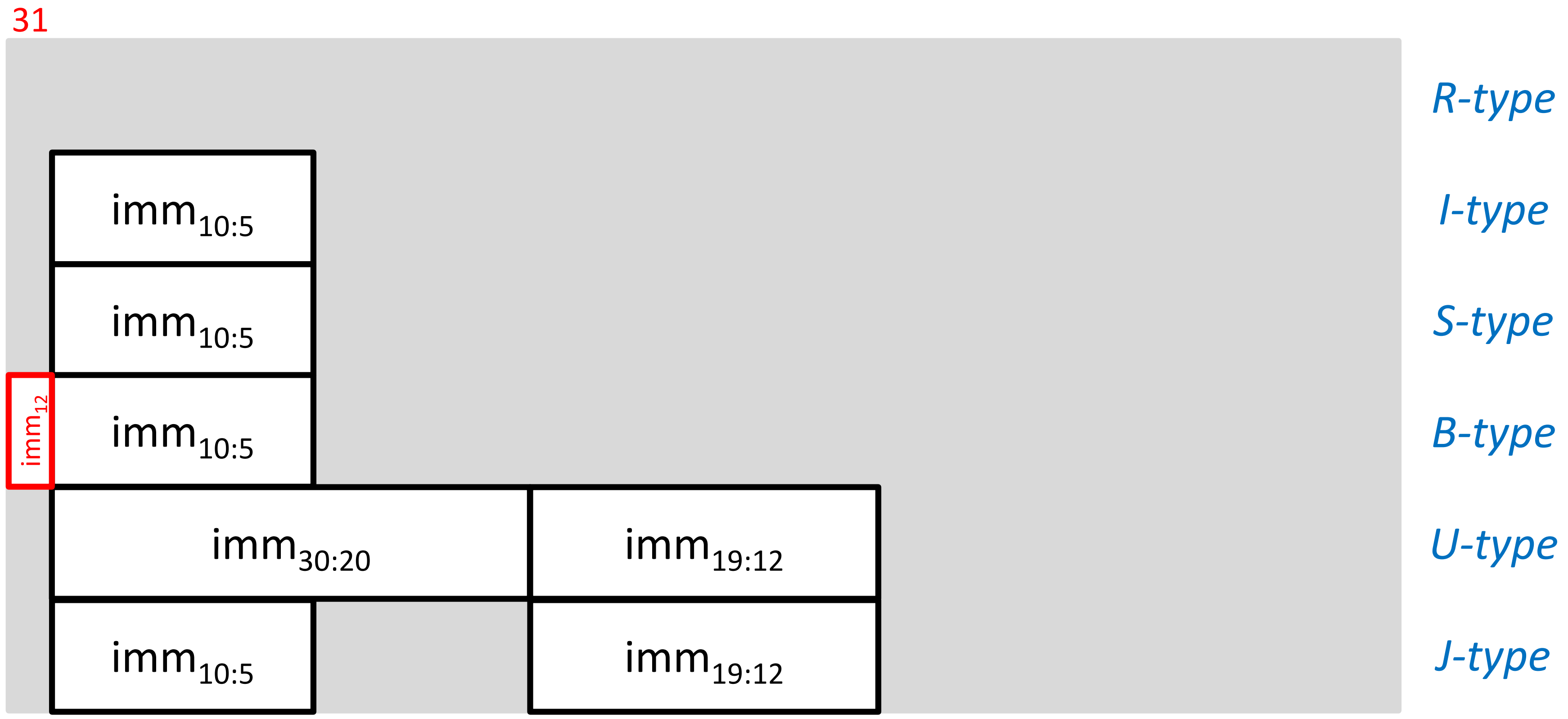
- The location of the immediate operands has been chosen in order to simplify the logic that performs the sign extension.
  - The same fragments are usually in the same position.



# Instruction formats

## Summary (ii)

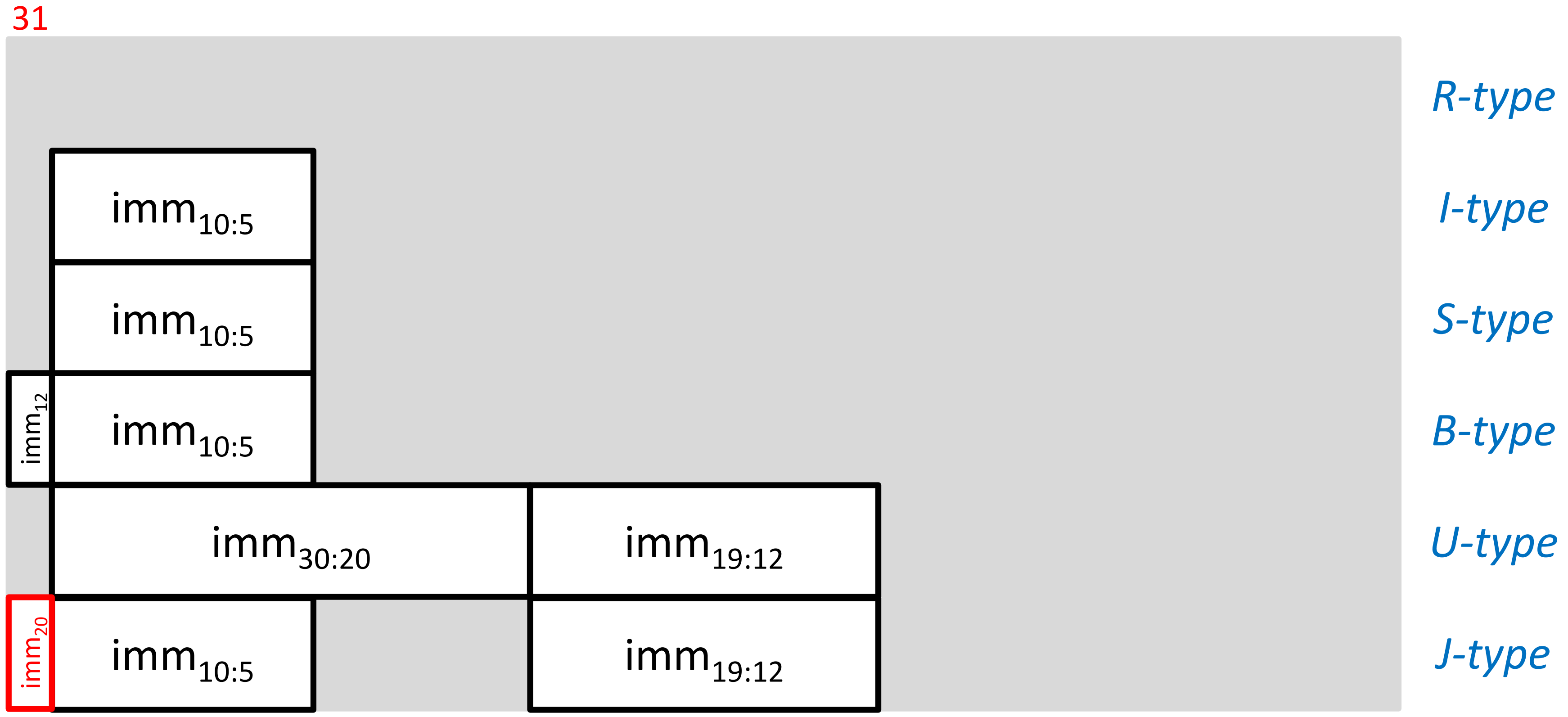
- The location of the immediate operands has been chosen in order to simplify the logic that performs the sign extension.
  - The same fragments are usually in the same position.
  - If not, they are distributed in as few different positions as possible.



# Instruction formats

## Summary (ii)

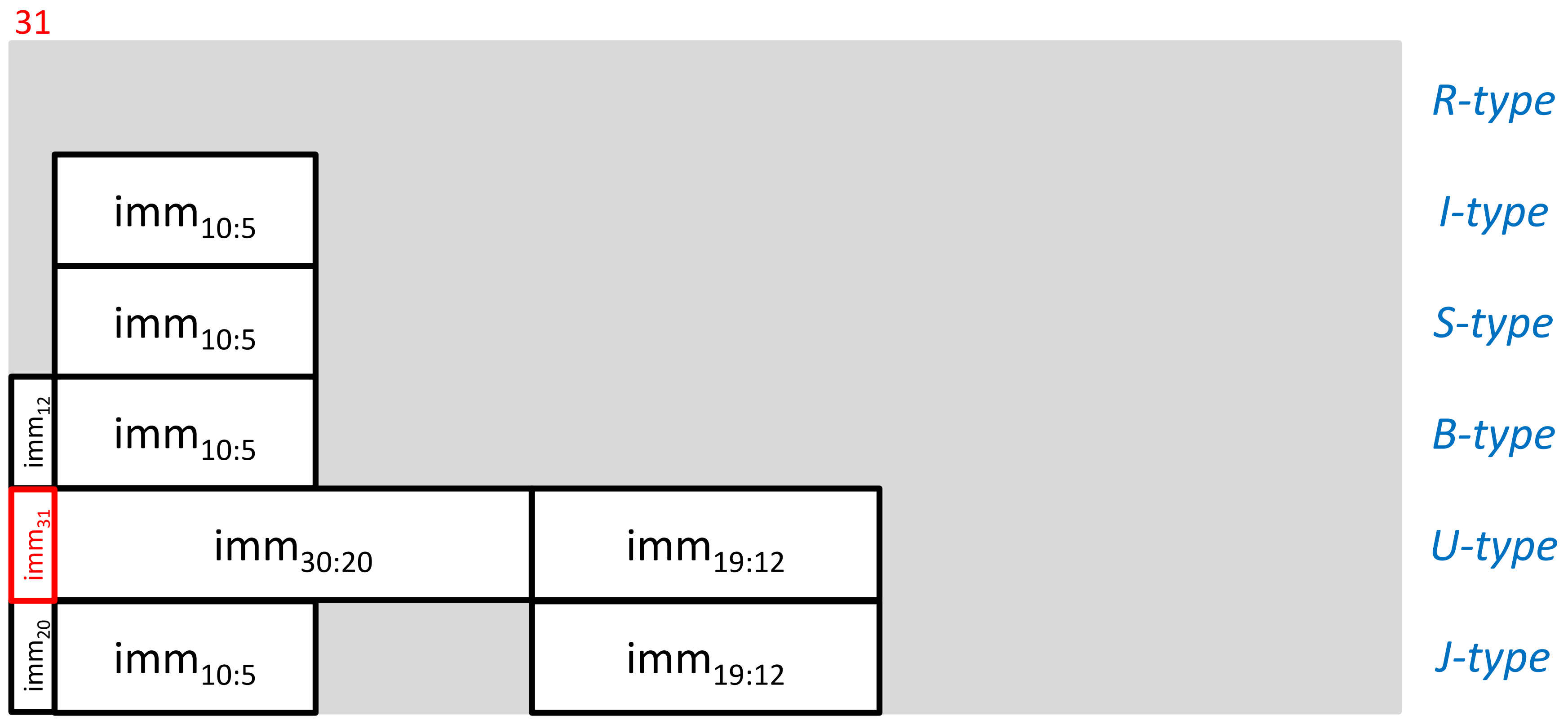
- The location of the immediate operands has been chosen in order to simplify the logic that performs the sign extension.
  - The same fragments are usually in the same position.
  - If not, they are distributed in as few different positions as possible.



# Instruction formats

## Summary (ii)

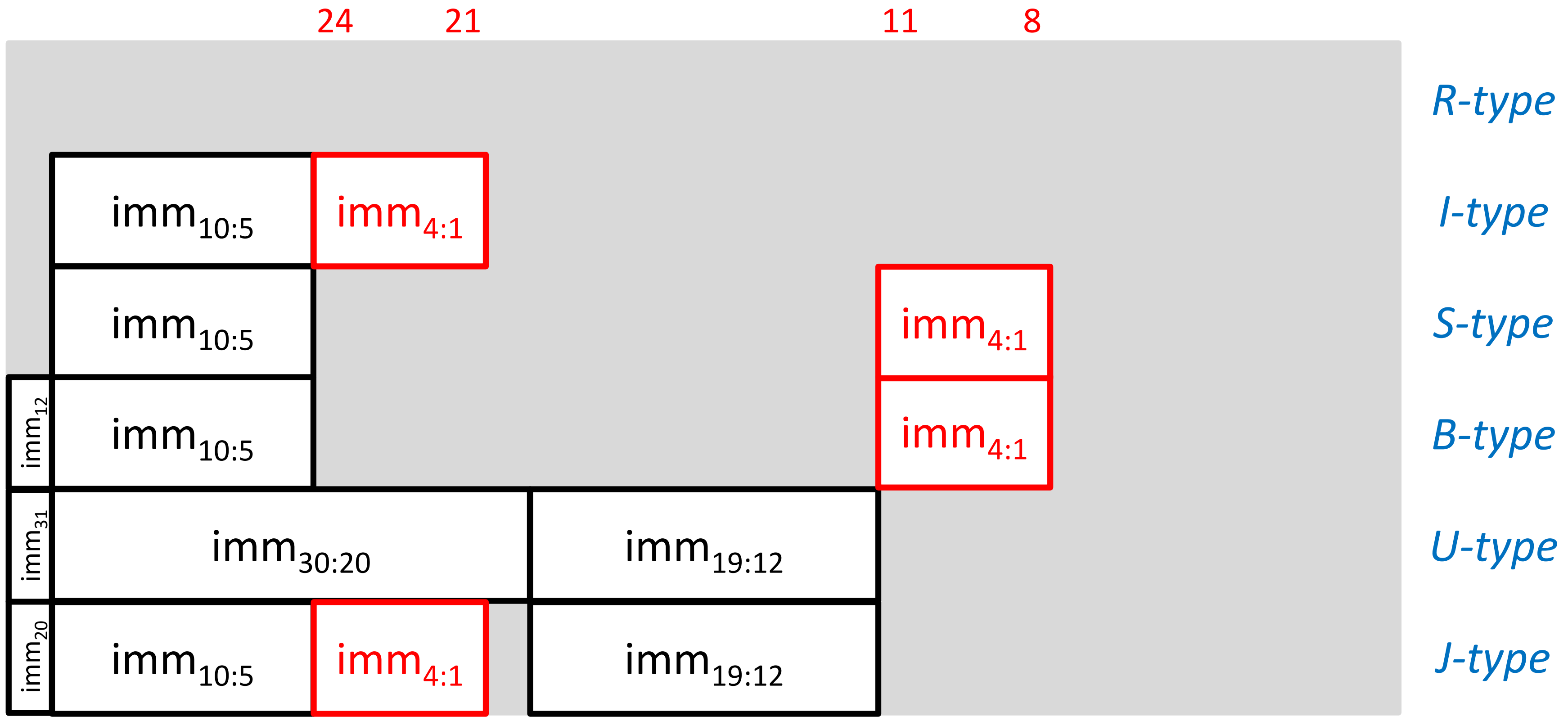
- The location of the immediate operands has been chosen in order to simplify the logic that performs the sign extension.
  - The same fragments are usually in the same position.
  - If not, they are distributed in as few different positions as possible.



# Instruction formats

## Summary (ii)

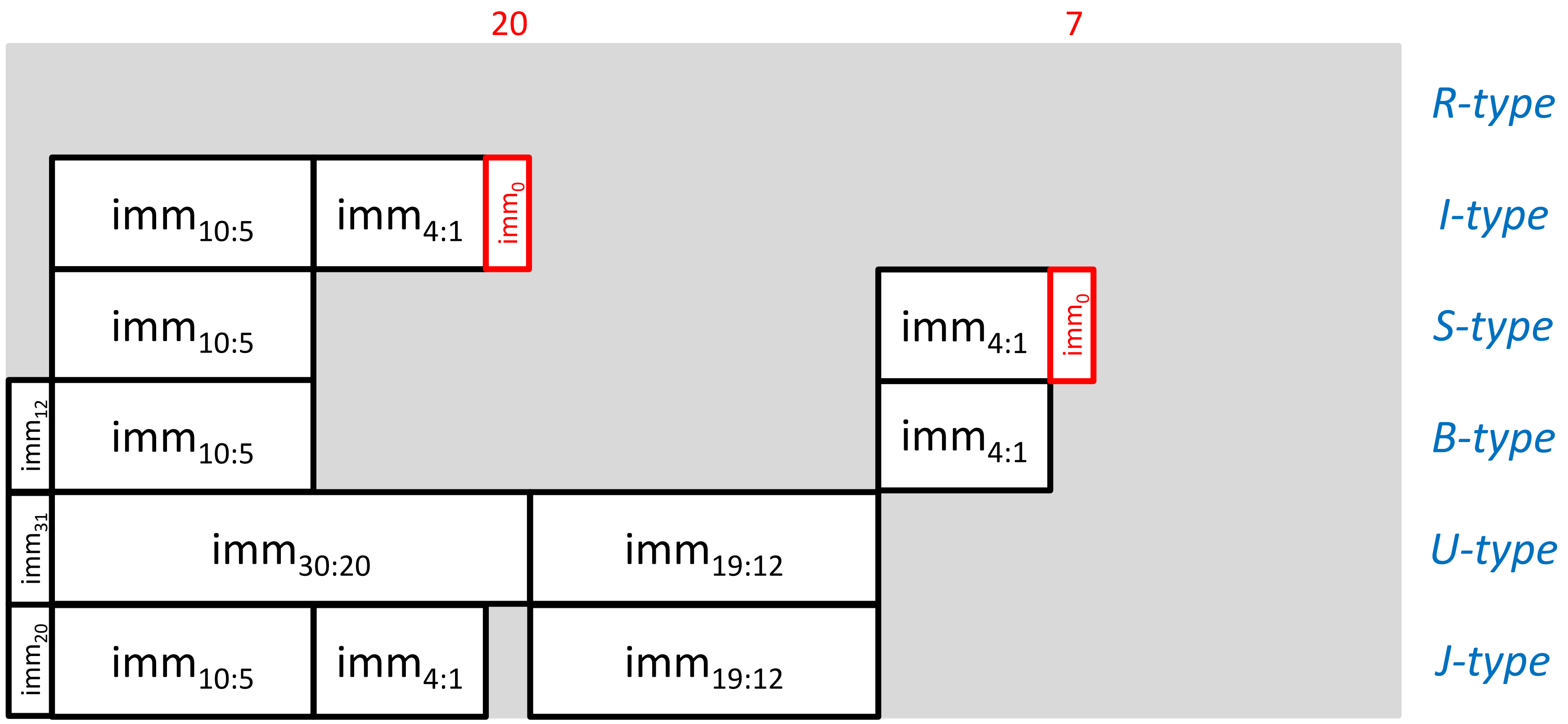
- The location of the immediate operands has been chosen in order to simplify the logic that performs the sign extension.
  - The same fragments are usually in the same position.
  - If not, they are distributed in as few different positions as possible.



# Instruction formats

## Summary (ii)

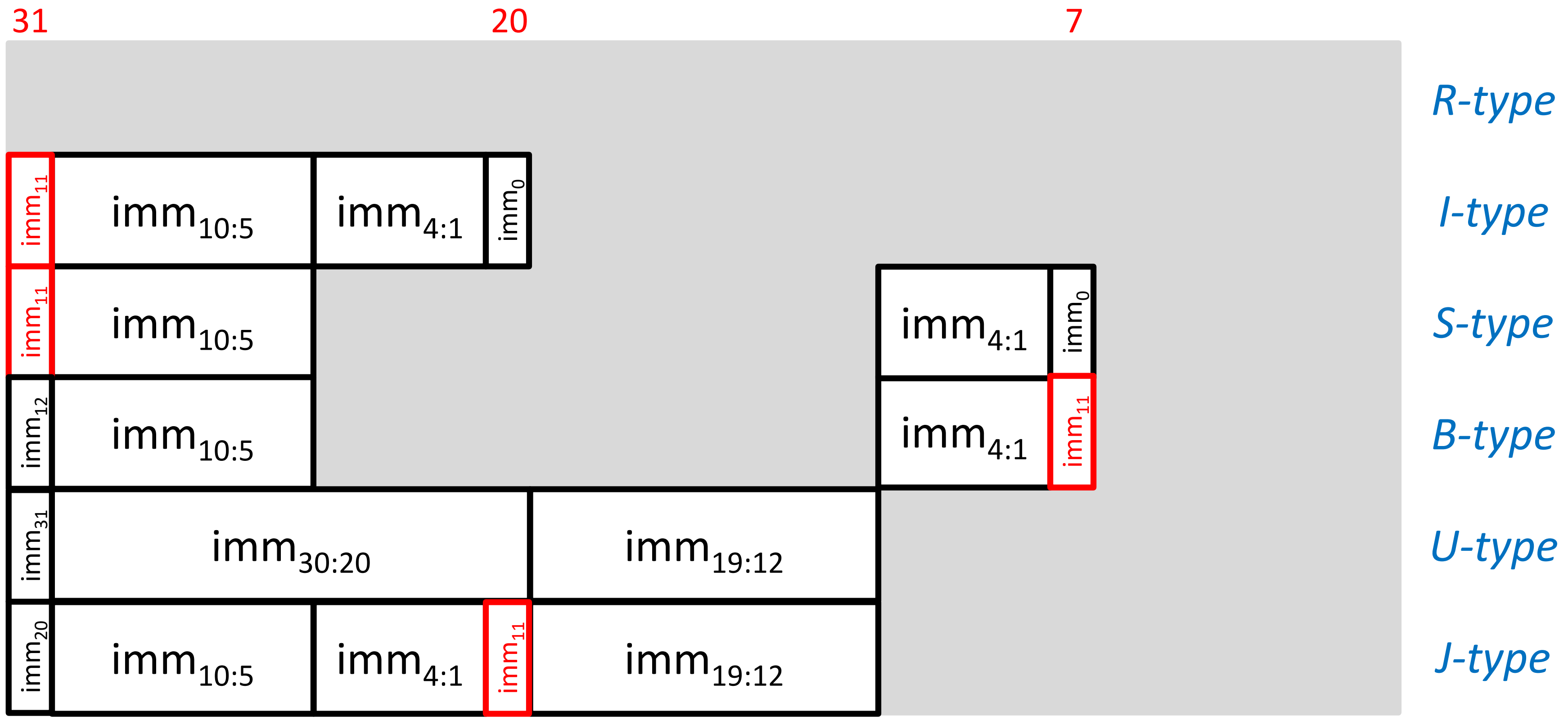
- The location of the immediate operands has been chosen in order to simplify the logic that performs the sign extension.
  - The same fragments are usually in the same position.
  - If not, they are distributed in as few different positions as possible.



# Instruction formats

## Summary (ii)

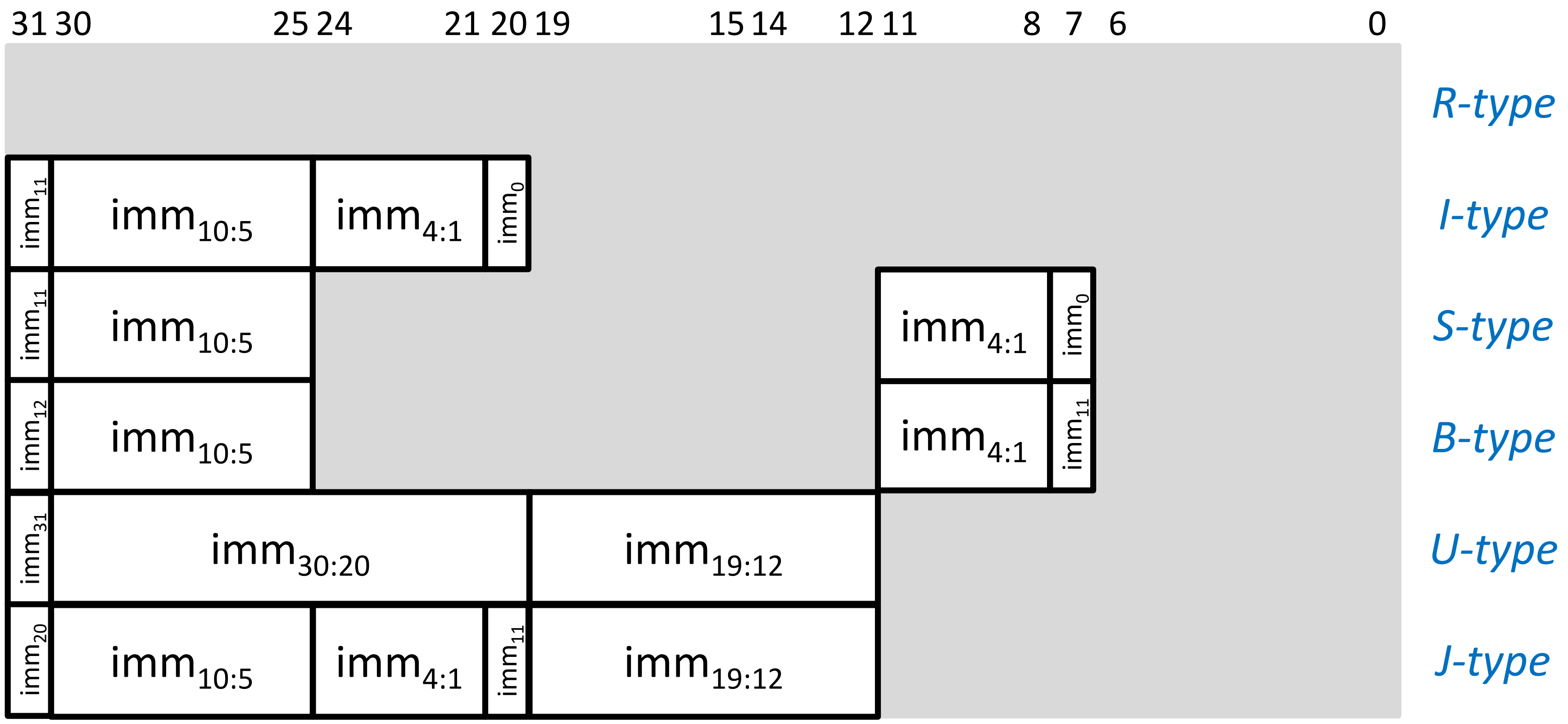
- The location of the immediate operands has been chosen in order to simplify the logic that performs the sign extension.
  - The same fragments are usually in the same position.
  - If not, they are distributed in as few different positions as possible.



# Instruction formats

## Summary (ii)

- The location of the immediate operands has been chosen in order to simplify the logic that performs the sign extension.
  - The same fragments are usually in the same position.
  - If not, they are distributed in as few different positions as possible.



31 30
25 24
21 20 19
15 14
12 11
8 7 6
0
R-type
I-type
S-type
B-type
U-type
J-type
$imm_{11}$ $imm_{10:5}$ $imm_{4:1}$ $imm_0$
$imm_{11}$ $imm_{10:5}$ $imm_{4:1}$ $imm_0$
$imm_{12}$ $imm_{10:5}$ $imm_{4:1}$ $imm_{11}$
$imm_{31}$ $imm_{30:20}$ $imm_{19:12}$
$imm_{20}$ $imm_{10:5}$ $imm_{4:1}$ $imm_{11}$ $imm_{19:12}$

# Field encoding

## Operation codes

| op | Instruction | Type |
|---|---|---|
| 0000011 | load | I |
| 0010011 | arithmetic-logic and shift with immediate operand | I |
| 0010111 | **`auipc`** | U |
| 0100011 | store | S |
| 0110011 | arithmetic-logic and shift with register operands | R |
| 0110111 | **`lui`** | U |
| 1100011 | branch | B |
| 1100111 | **`jalr`** | I |
| 1101111 | **`jal`** | J |

## Field encoding

### Function codes (i)

#### Load instructions

| op | funct3 | Instruction | Type |
|---|---|---|---|
| 0000011 | 000 | `lb` | I |
| | 001 | `lh` | I |
| | 010 | `lw` | I |
| | 011 | `lbu` | I |
| | 100 | `lhu` | I |

#### Store instructions

| op | funct3 | Instruction | Type |
|---|---|---|---|
| 0100011 | 000 | `sb` | S |
| | 001 | `sh` | S |
| | 010 | `sw` | S |

## Field encoding

### Function codes (ii)

**Arithmetic-logic and shift instructions with immediate operand**

| op | funct3 | funct7* | Instruction | Type |
|---|---|---|---|---|
| 0010011 | 000 | - | `addi` | I |
| | 001 | 0000000* | `slli` | I |
| | 010 | - | `slti` | I |
| | 011 | - | `sltiu` | I |
| | 100 | - | `xori` | I |
| | 101 | 0000000* | `srli` | I |
| | 101 | 0100000* | `srai` | I |
| | 110 | - | `ori` | I |
| | 111 | - | `andi` | I |

*Encoded in the 7 most significant bits of the imm field

# Field encoding

## Function codes (iii)

### Arithmetic-logic and shift instructions with register operands

| op | funct3 | funct7 | Instruction | Type |
|---|---|---|---|---|
| 0110011 | 000 | 0000000 | `add` | R |
| | 000 | 0100000 | `sub` | R |
| | 001 | 0000000 | `sll` | R |
| | 010 | 0000000 | `slt` | R |
| | 011 | 0000000 | `sltu` | R |
| | 100 | 0000000 | `xor` | R |
| | 101 | 0000000 | `srl` | R |
| | 101 | 0100000 | `sra` | R |
| | 110 | 0000000 | `or` | R |
| | 111 | 0000000 | `and` | R |

# Field encoding

## Function codes (iv)

### Multiplication and division instructions*

| op | funct3 | funct7 | Instruction | Type |
|---|---|---|---|---|
| 0110011 | 000 | 0000001 | `mul` | R |
| | 001 | 0000001 | `mulh` | R |
| | 010 | 0000001 | `mulhsu` | R |
| | 011 | 0000001 | `mulhu` | R |
| | 100 | 0000001 | `div` | R |
| | 101 | 0000001 | `divu` | R |
| | 110 | 0000001 | `rem` | R |
| | 111 | 0000001 | `remu` | R |

*Defined in the RVM extension

# Field encoding

## Function codes (v)

### Condition branch instructions

| op | funct3 | Instruction | Type |
|---|---|---|---|
| 1100011 | 000 | **`beq`** | B |
| | 001 | **`bne`** | B |
| | 100 | **`blt`** | B |
| | 101 | **`bge`** | B |
| | 110 | **`bltu`** | B |
| | 111 | **`bgeu`** | B |

# Field encoding

## Register codes

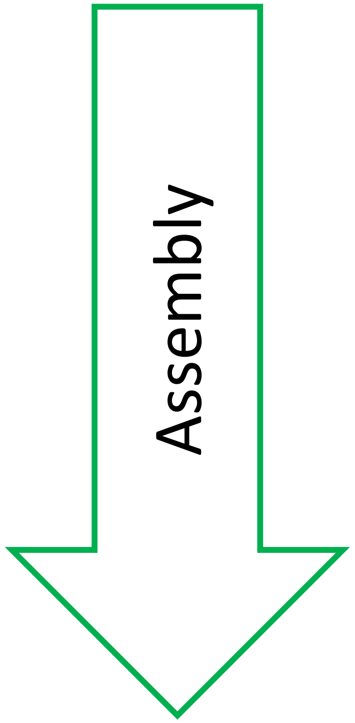
| Name | Number | Code |
|---|---|---|
| zero | x0 | 00000 |
| ra | x1 | 00001 |
| sp | x2 | 00010 |
| gp | x3 | 00011 |
| tp | x4 | 00100 |
| t0 | x5 | 00101 |
| t1 | x6 | 00110 |
| t2 | x7 | 00111 |
| s0/fp | x8 | 01000 |
| s1 | x9 | 01001 |
| a0 | x10 | 01010 |
| a1 | x11 | 01011 |
| a2 | x12 | 01100 |
| a3 | x13 | 01101 |
| a4 | x14 | 01110 |
| a5 | x15 | 01111 |

| Name | Number | Code |
|---|---|---|
| a6 | x16 | 10000 |
| a7 | x17 | 10001 |
| s2 | x18 | 10010 |
| s3 | x19 | 10011 |
| s4 | x20 | 10100 |
| s5 | x21 | 10101 |
| s6 | x22 | 10110 |
| s7 | x23 | 10111 |
| s8 | x24 | 11000 |
| s9 | x25 | 11001 |
| s10 | x26 | 11010 |
| s11 | x27 | 11011 |
| t3 | x28 | 11100 |
| t4 | x29 | 11101 |
| t5 | x30 | 11110 |
| t6 | x31 | 11111 |

## From assembly to machine code

### Example: R-type instruction

```
sub x5, x6, x7
```

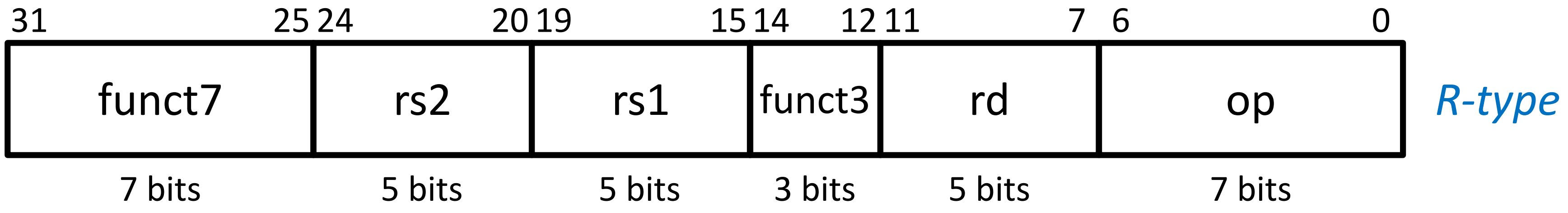


```
0x407302b3
```

# From assembly to machine code

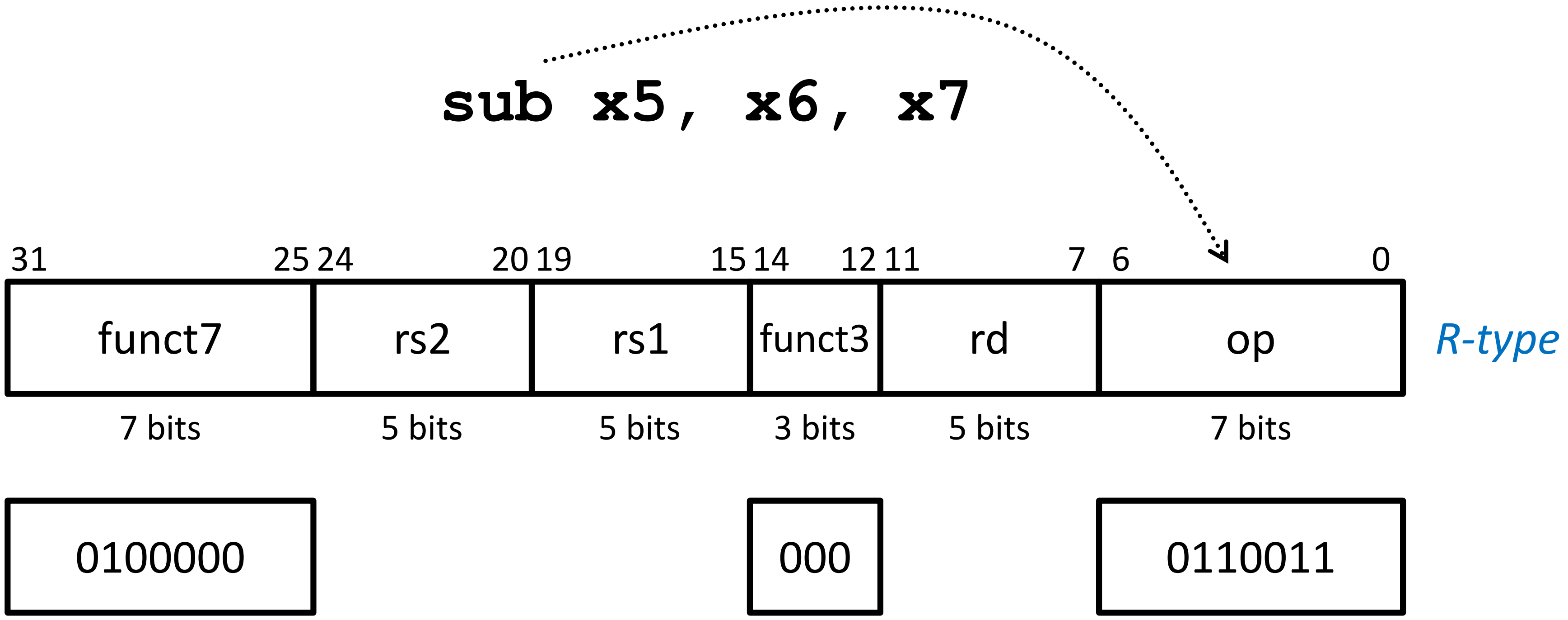
## Example: R-type instruction

```
sub x5, x6, x7
```



# From assembly to machine code

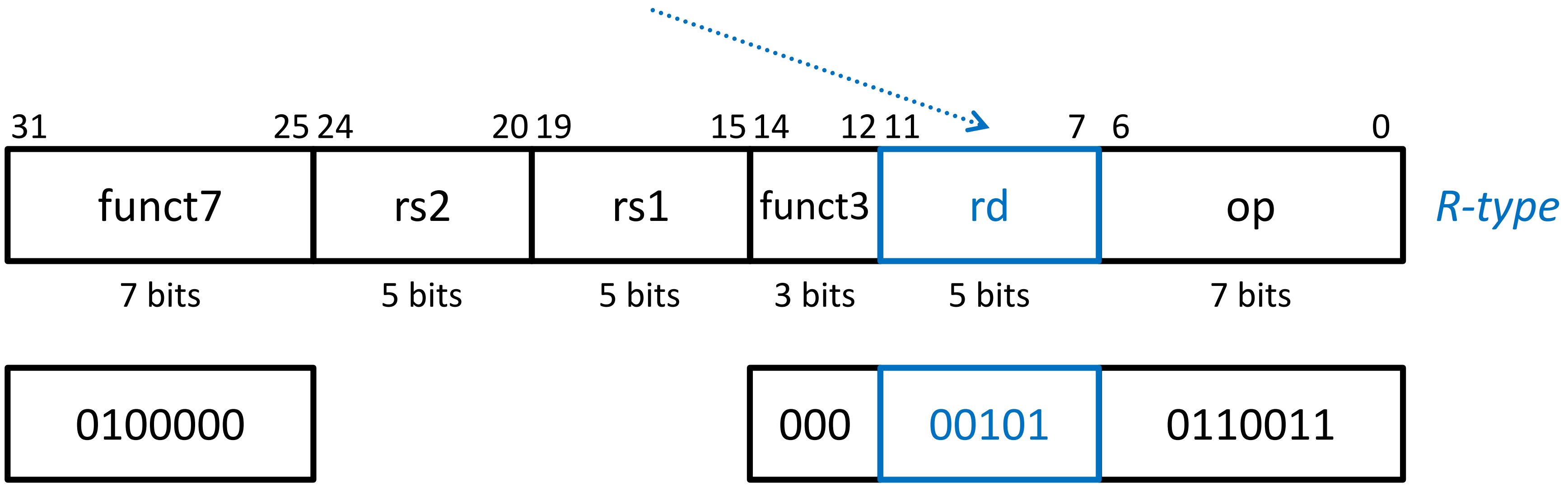
## Example: R-type instruction



# From assembly to machine code

## Example: R-type instruction

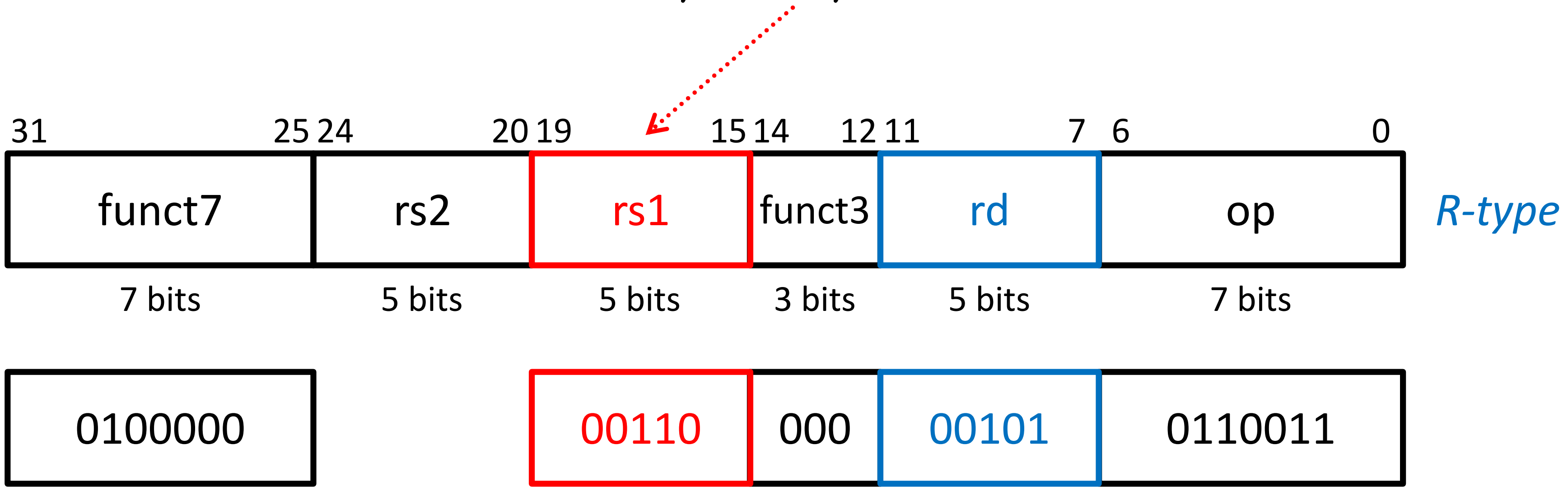
```
sub x5, x6, x7
```



# From assembly to machine code

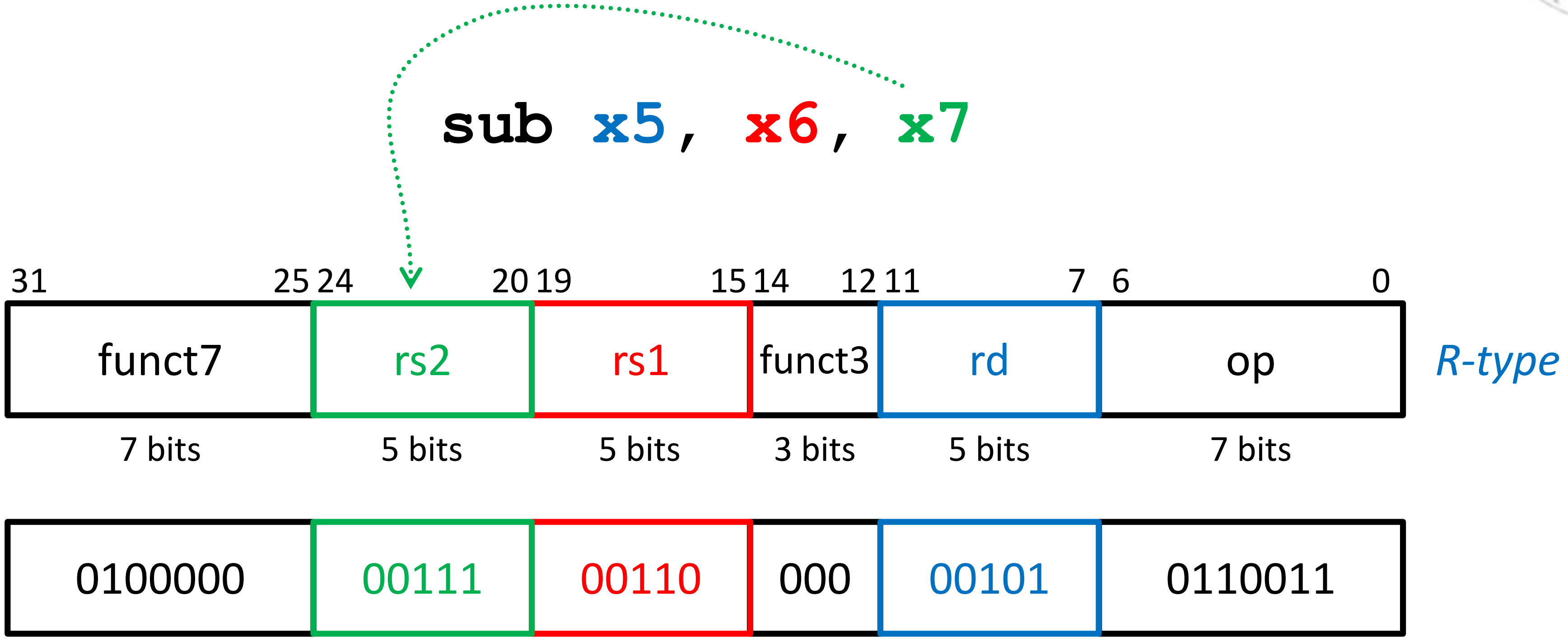
## Example: R-type instruction

```
sub x5, x6, x7
```



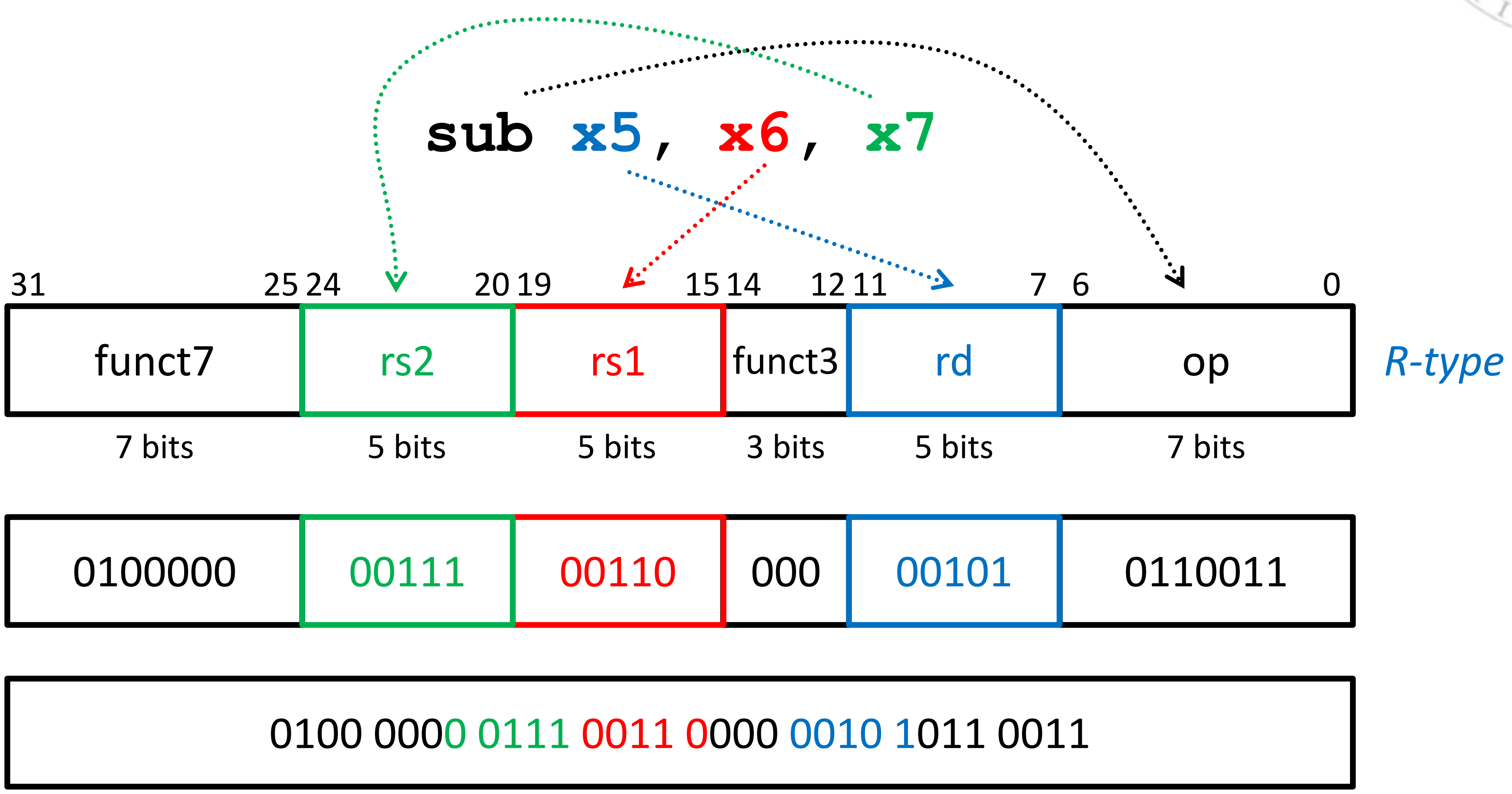
# From assembly to machine code

## Example: R-type instruction



# From assembly to machine code

## Example: R-type instruction



`0x407302b3`

## From assembly to machine code

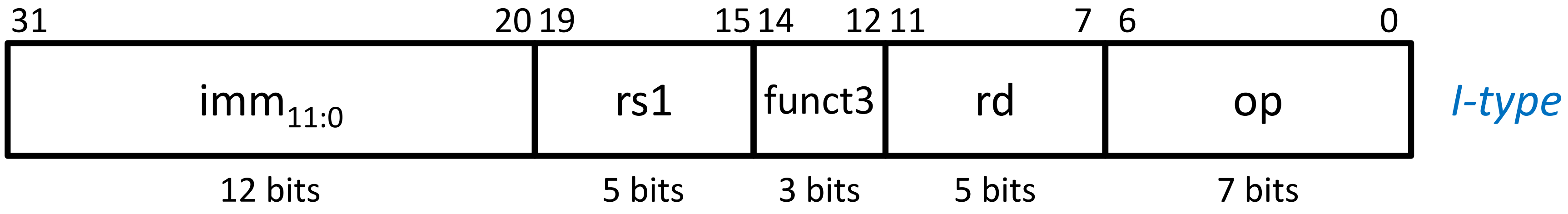
Example: I-type instruction

$$\begin{tikzcd}
\texttt{addi s0, s1, 12} \arrow[d, "\text{Assembly}"] \\
\texttt{0x00c48413}
\end{tikzcd}$$

# From assembly to machine code

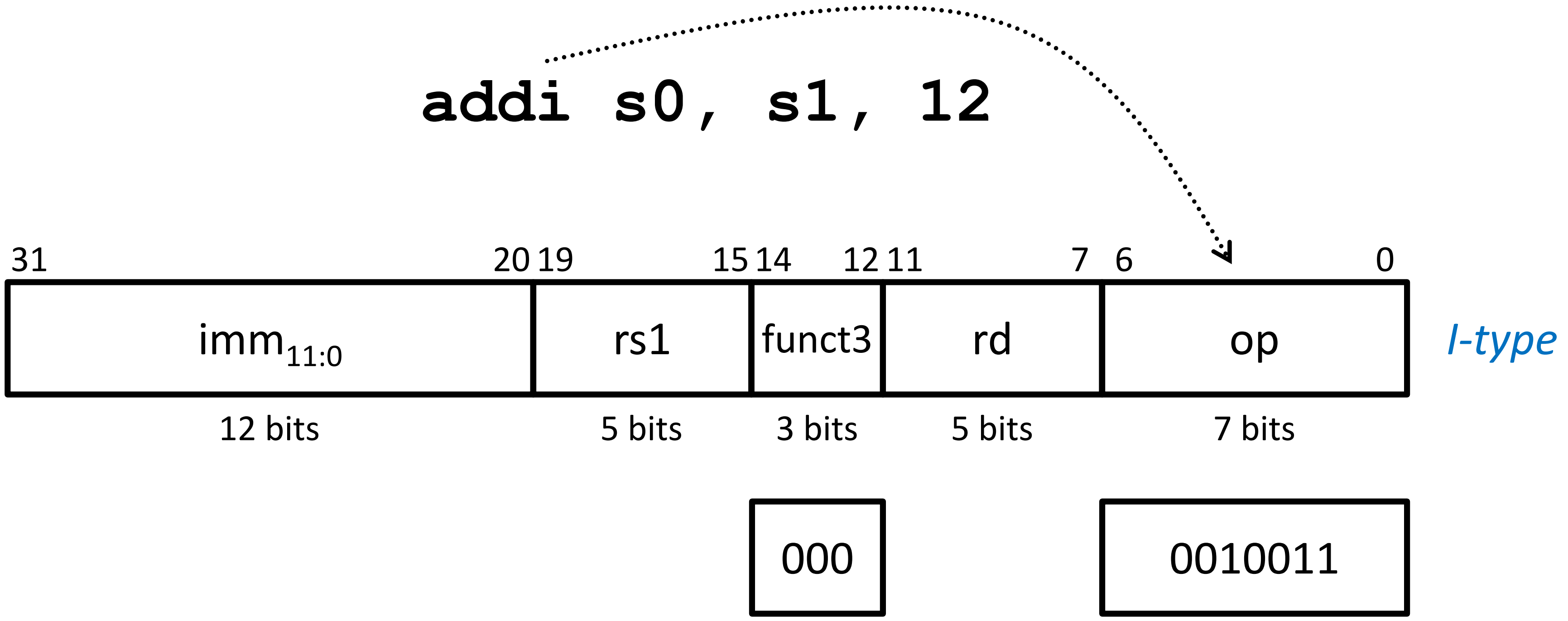
## Example: I-type instruction

```
addi s0, s1, 12
```



## From assembly to machine code

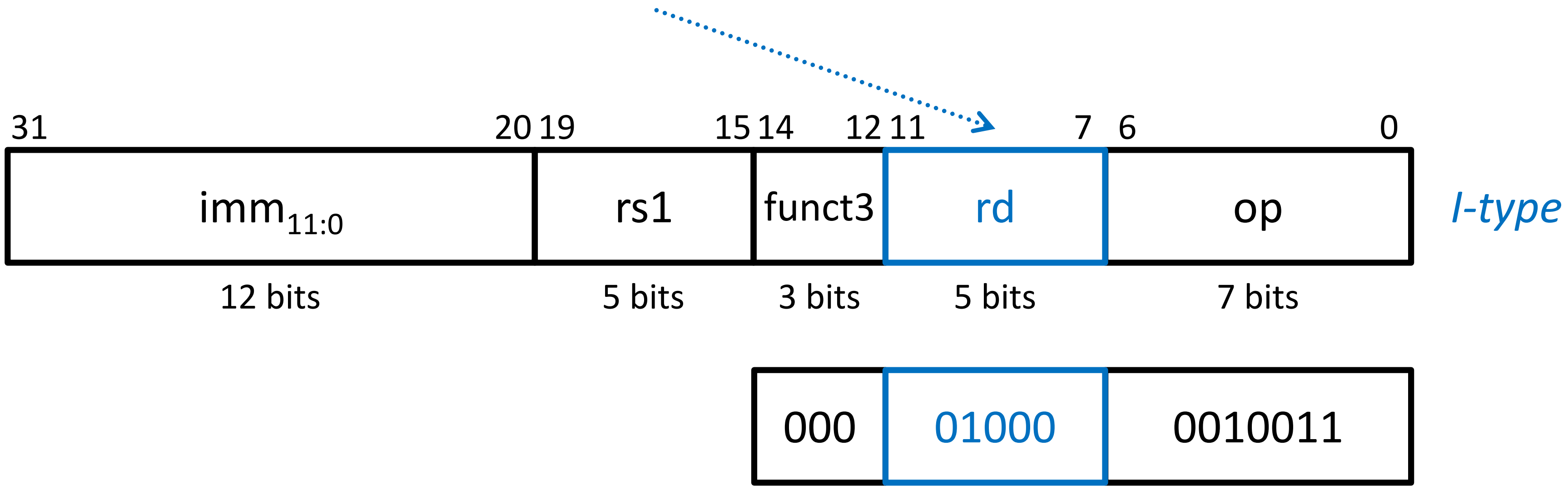
### Example: I-type instruction



# From assembly to machine code

## Example: I-type instruction (i)

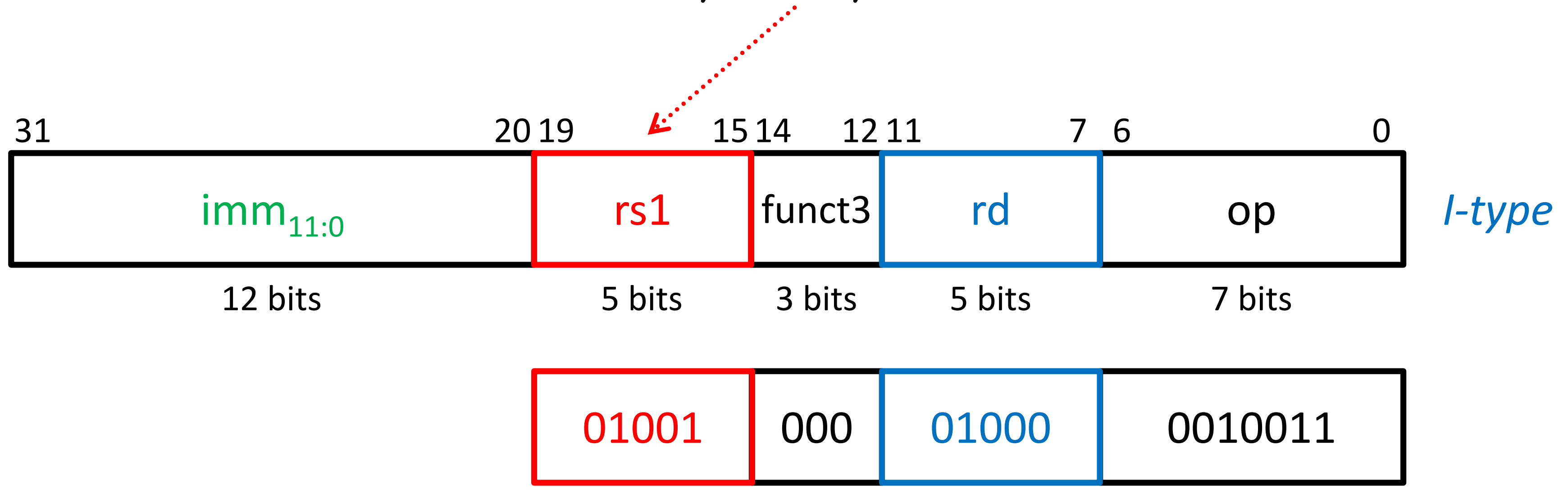
```
addi s0, s1, 12 ≡ addi x8, x9, 12
```



# From assembly to machine code

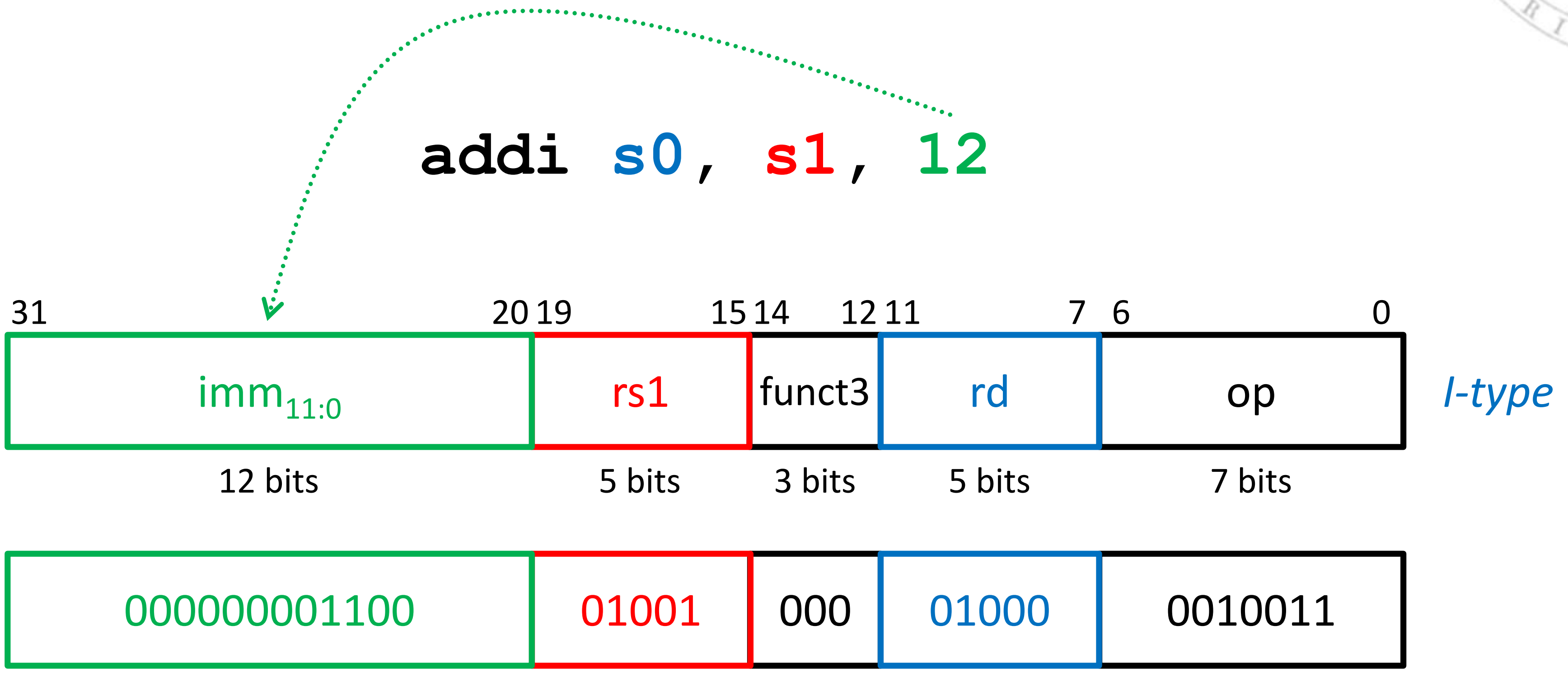
## Example: I-type instruction (i)

```
addi s0, s1, 12 ≡ addi x8, x9, 12
```



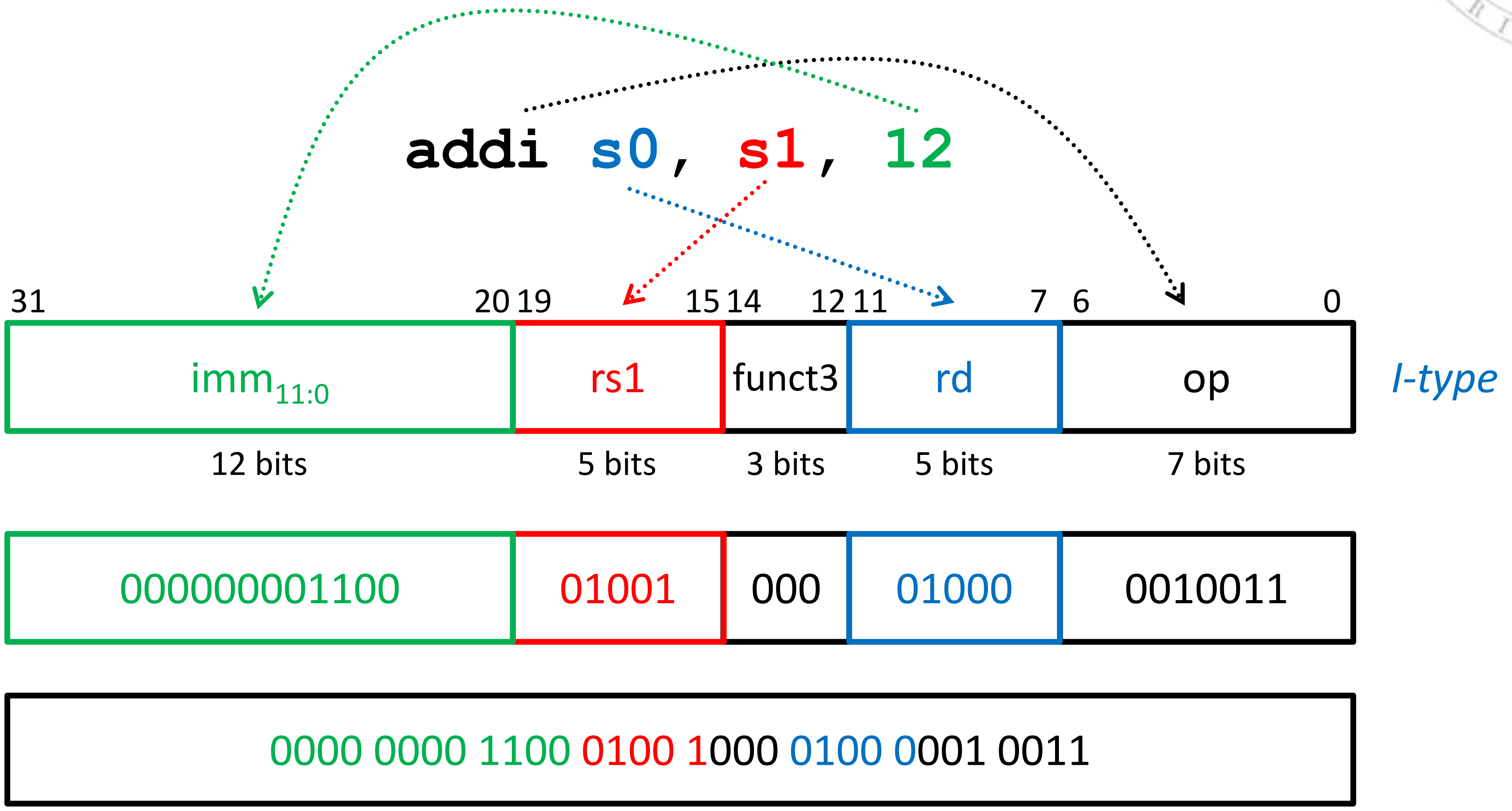
# From assembly to machine code

## Example: I-type instruction (i)



# From assembly to machine code

## Example: I-type instruction (i)

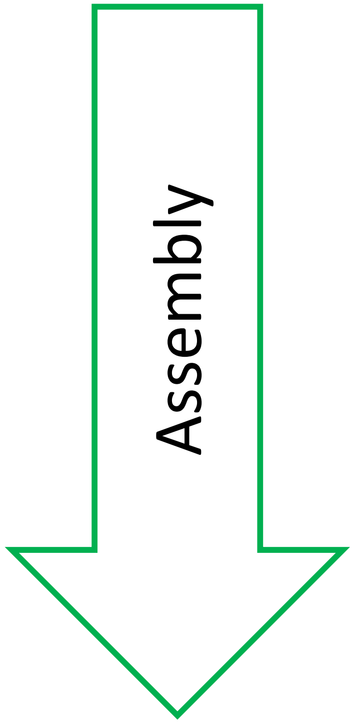


**0x00c48413**

## From assembly to machine code

### Example: I-type instruction (ii)

```
lw t2, -6(s3)
```

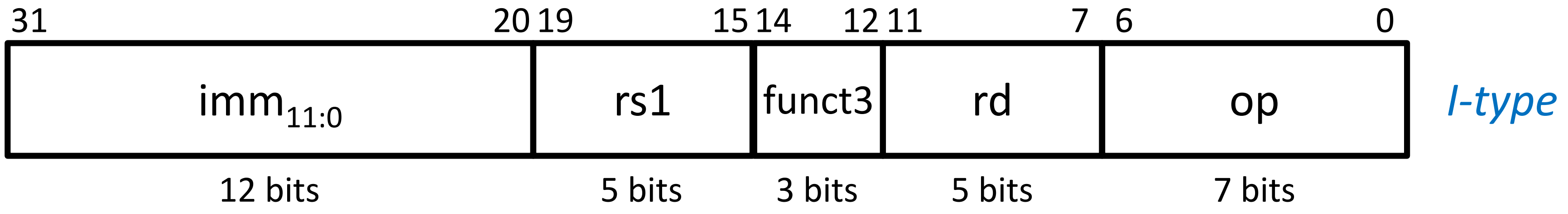


```
0xffa9a383
```

# From assembly to machine code

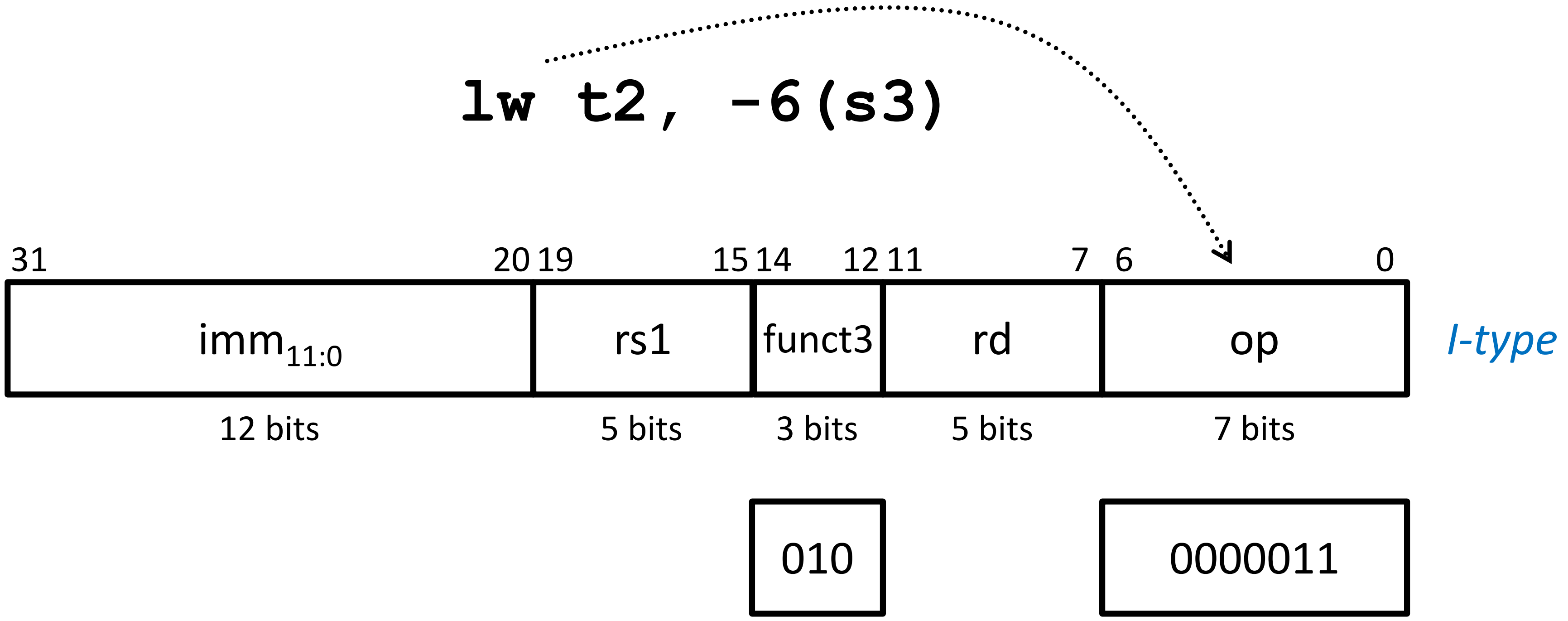
## Example: I-type instruction (ii)

```
lw t2, -6(s3)
```



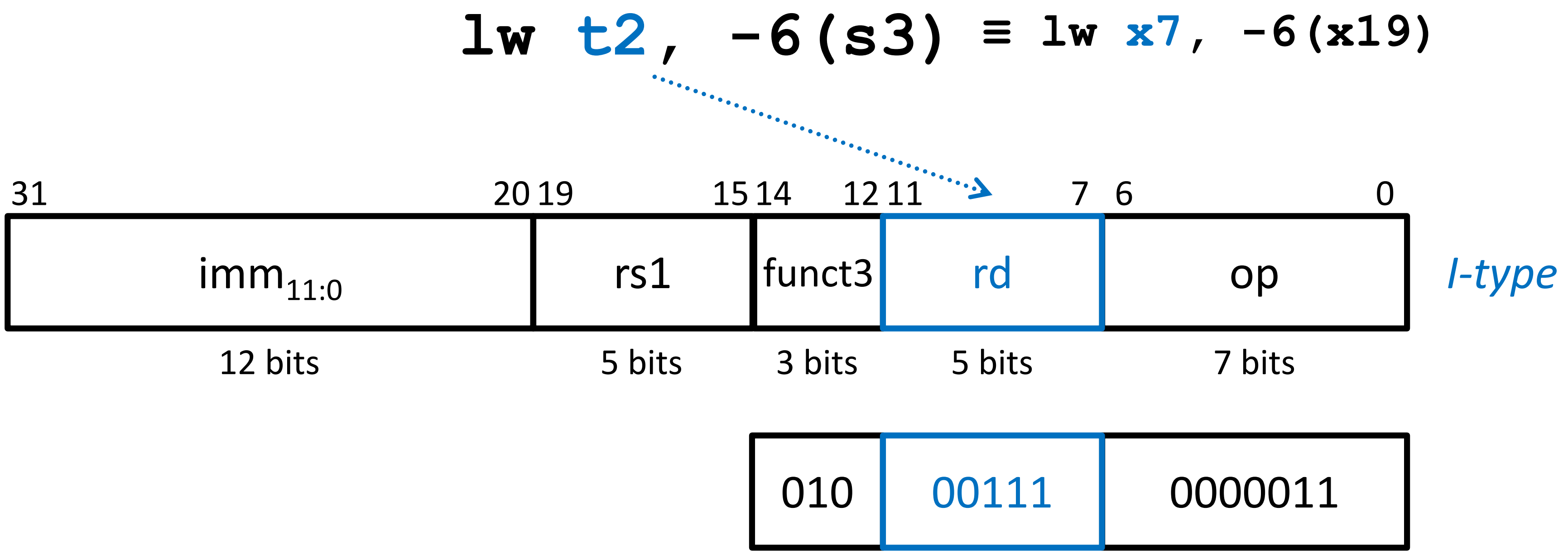
# From assembly to machine code

## Example: I-type instruction (ii)



# From assembly to machine code

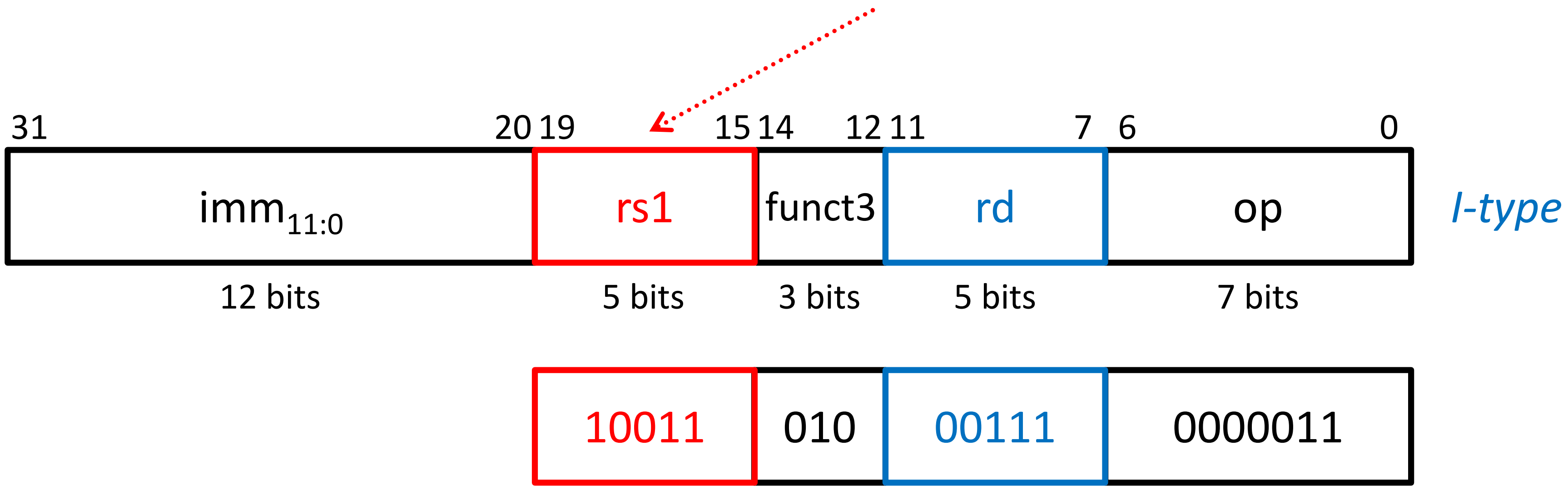
## Example: I-type instruction (ii)



# From assembly to machine code

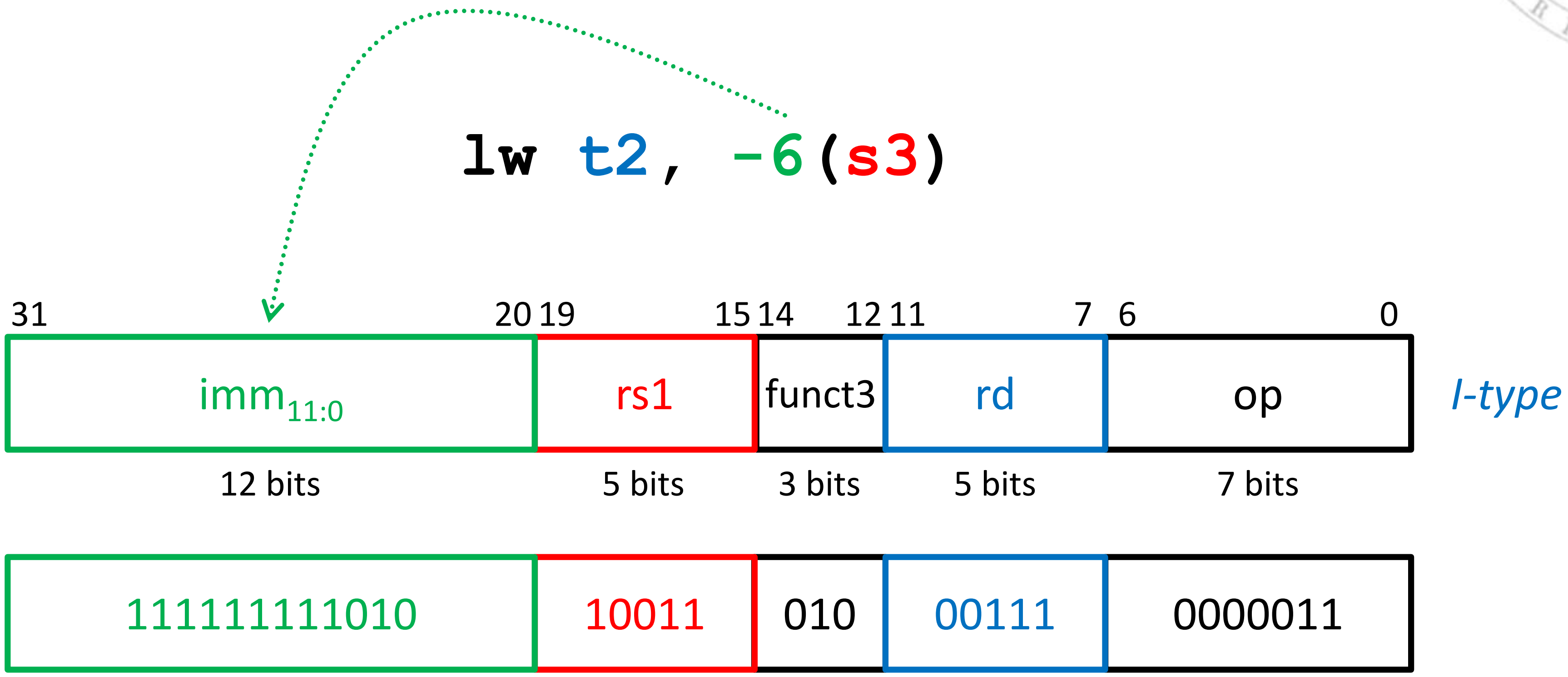
## Example: I-type instruction (ii)

```
lw t2, -6(s3) ≡ lw x7, -6(x19)
```



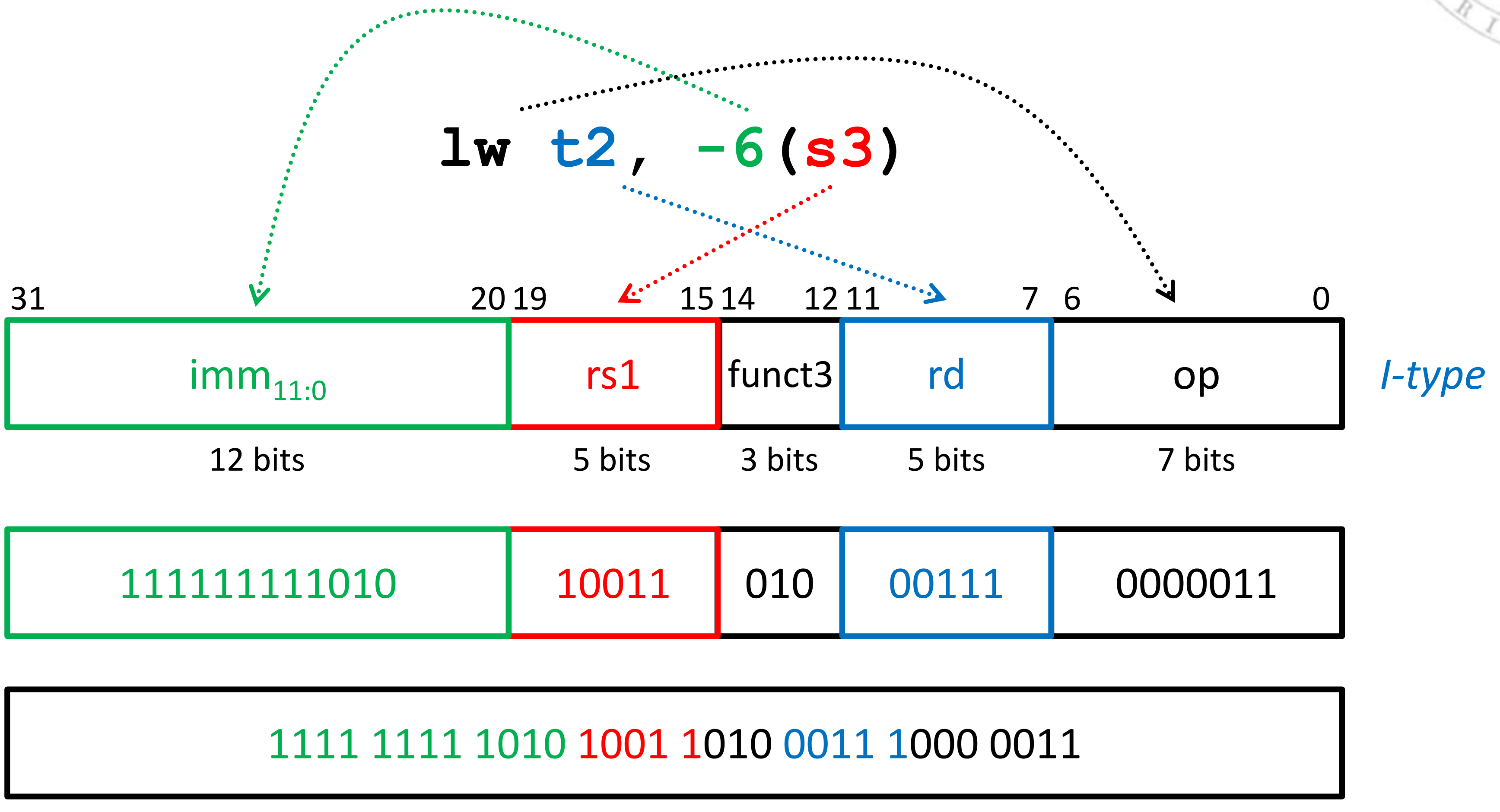
# From assembly to machine code

## Example: I-type instruction (ii)



# From assembly to machine code

## Example: I-type instruction (ii)

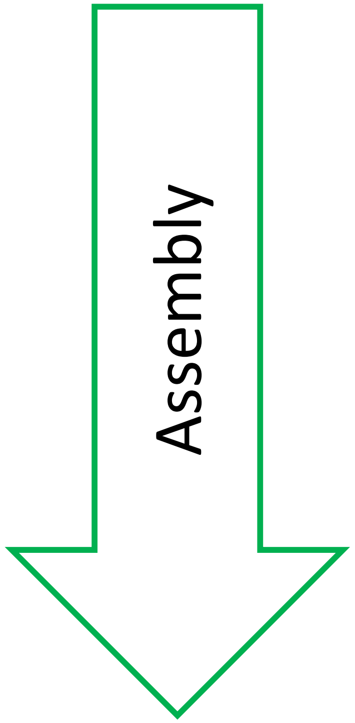


**0xffa9a383**

# From assembly to machine code

## Example: I-type instruction (iii)

```
srai t1, t2, 29
```

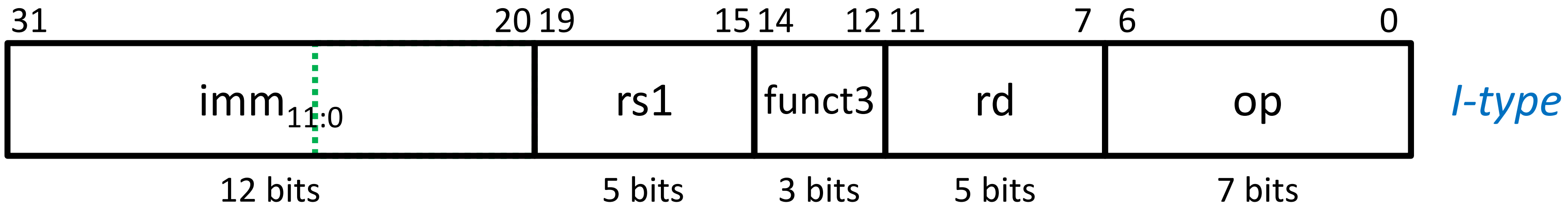


```
0x41d3d313
```

# From assembly to machine code

## Example: I-type instruction (iii)

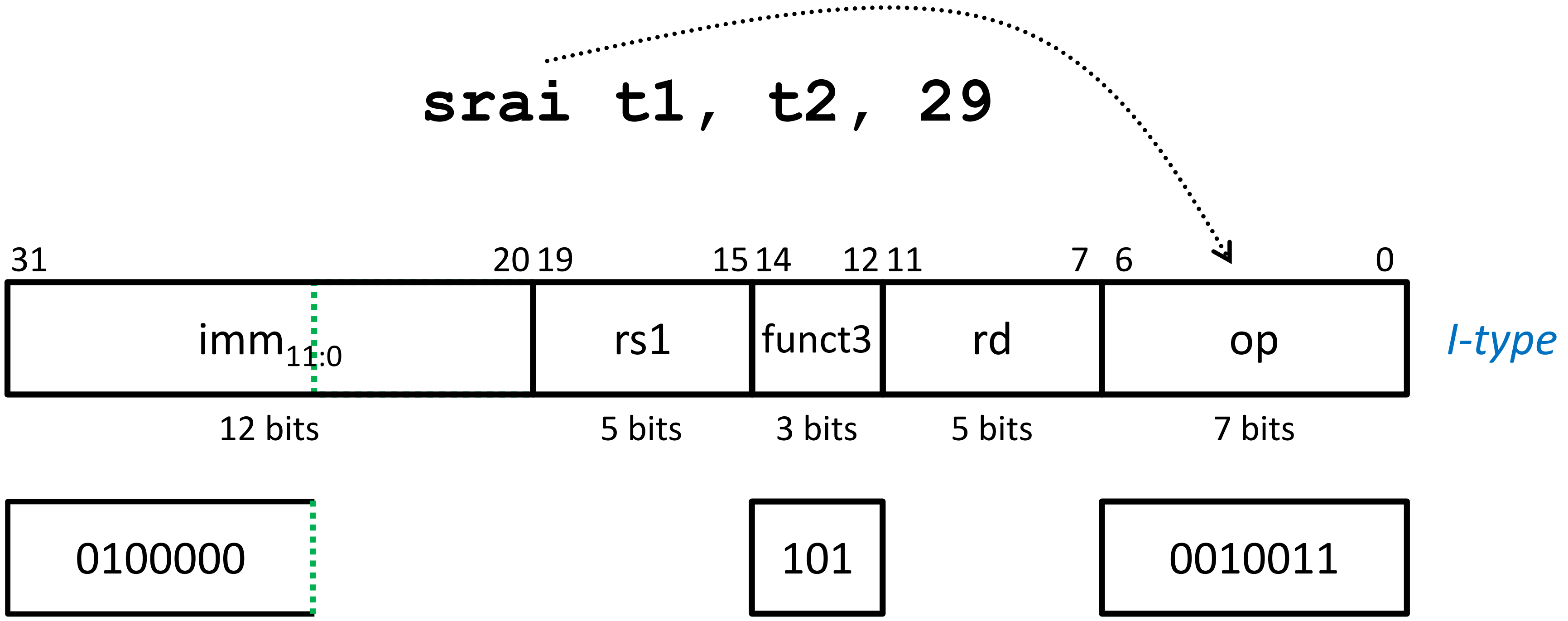
```
srai t1, t2, 29
```



# From assembly to machine code

## Example: I-type instruction (iii)

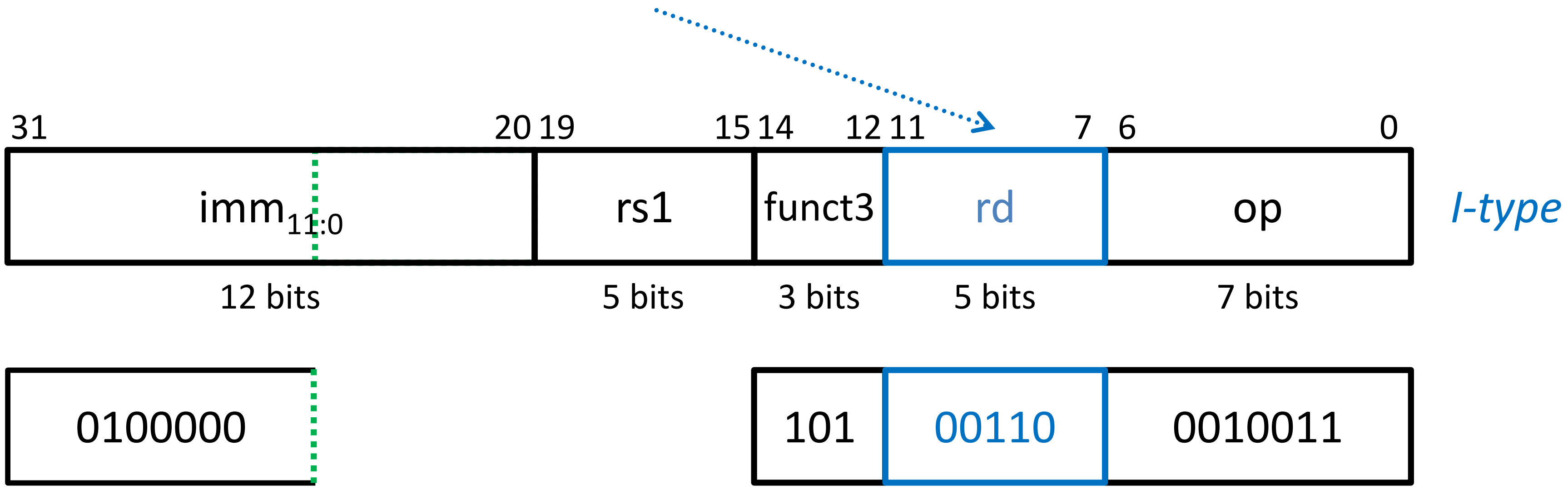
```
srai t1, t2, 29
```



# From assembly to machine code

## Example: I-type instruction (iii)

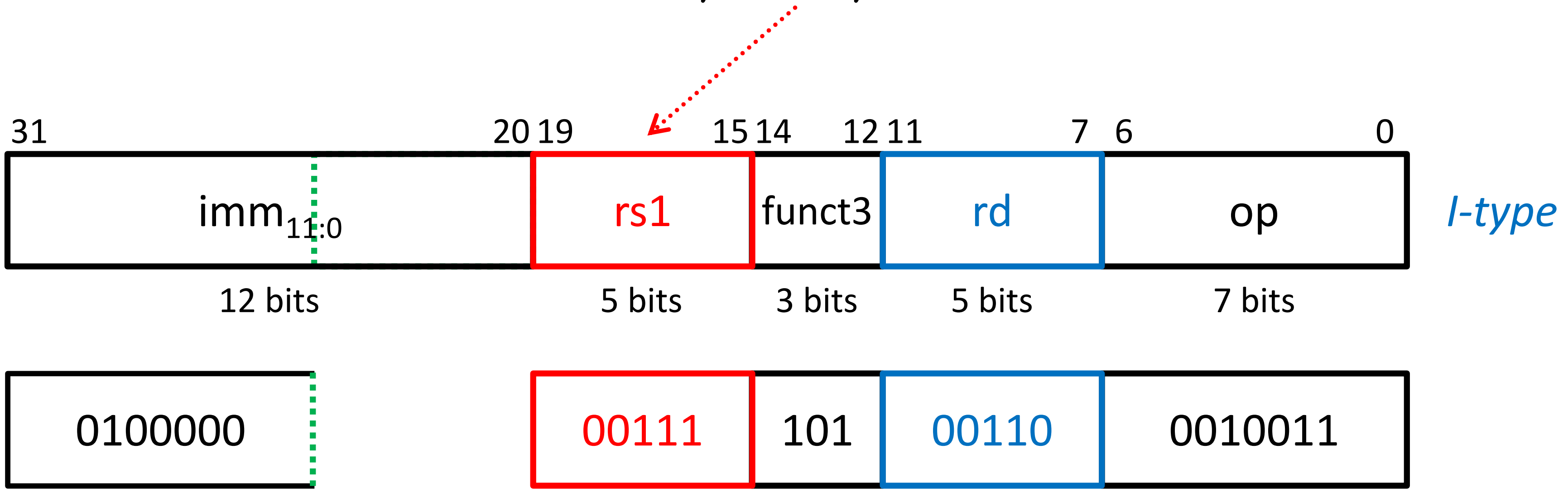
```
srai t1, t2, 29 ≡ srai x6, x7, 29
```



# From assembly to machine code

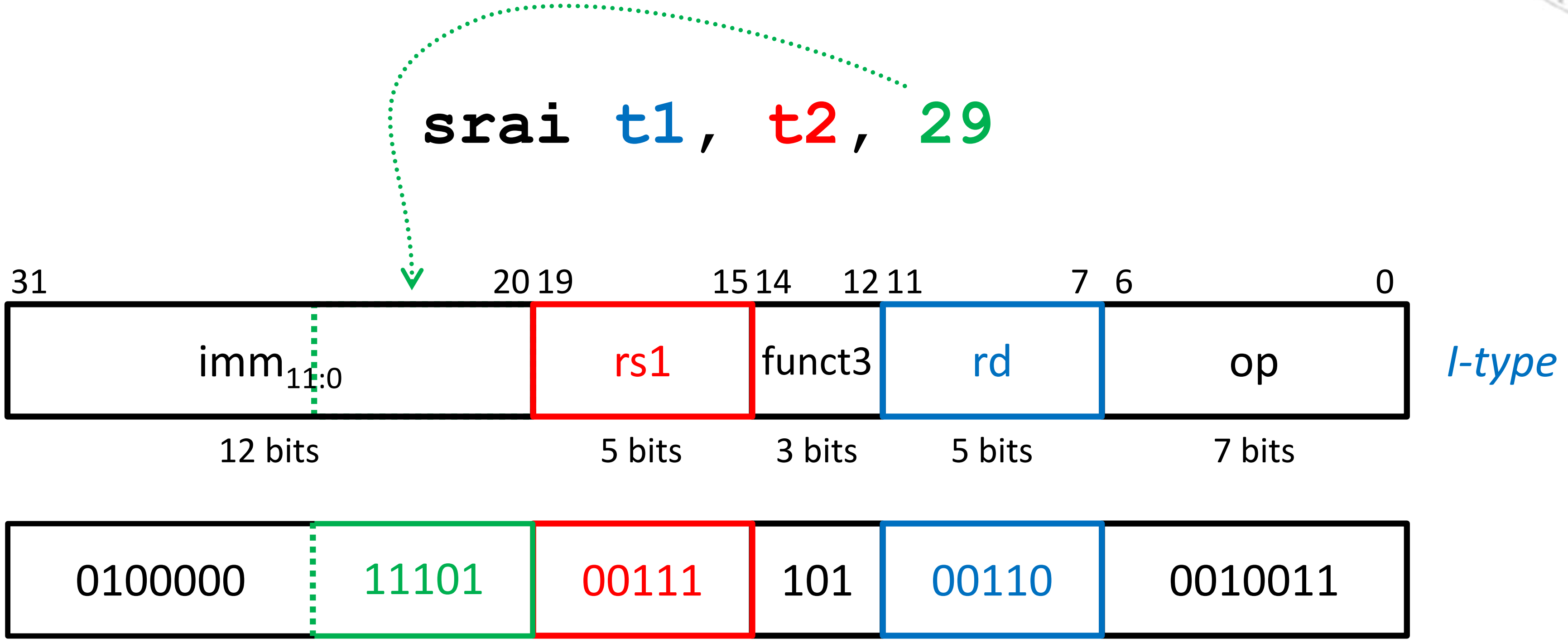
## Example: I-type instruction (iii)

`srai t1, t2, 29 ≡ srai x6, x7, 29`



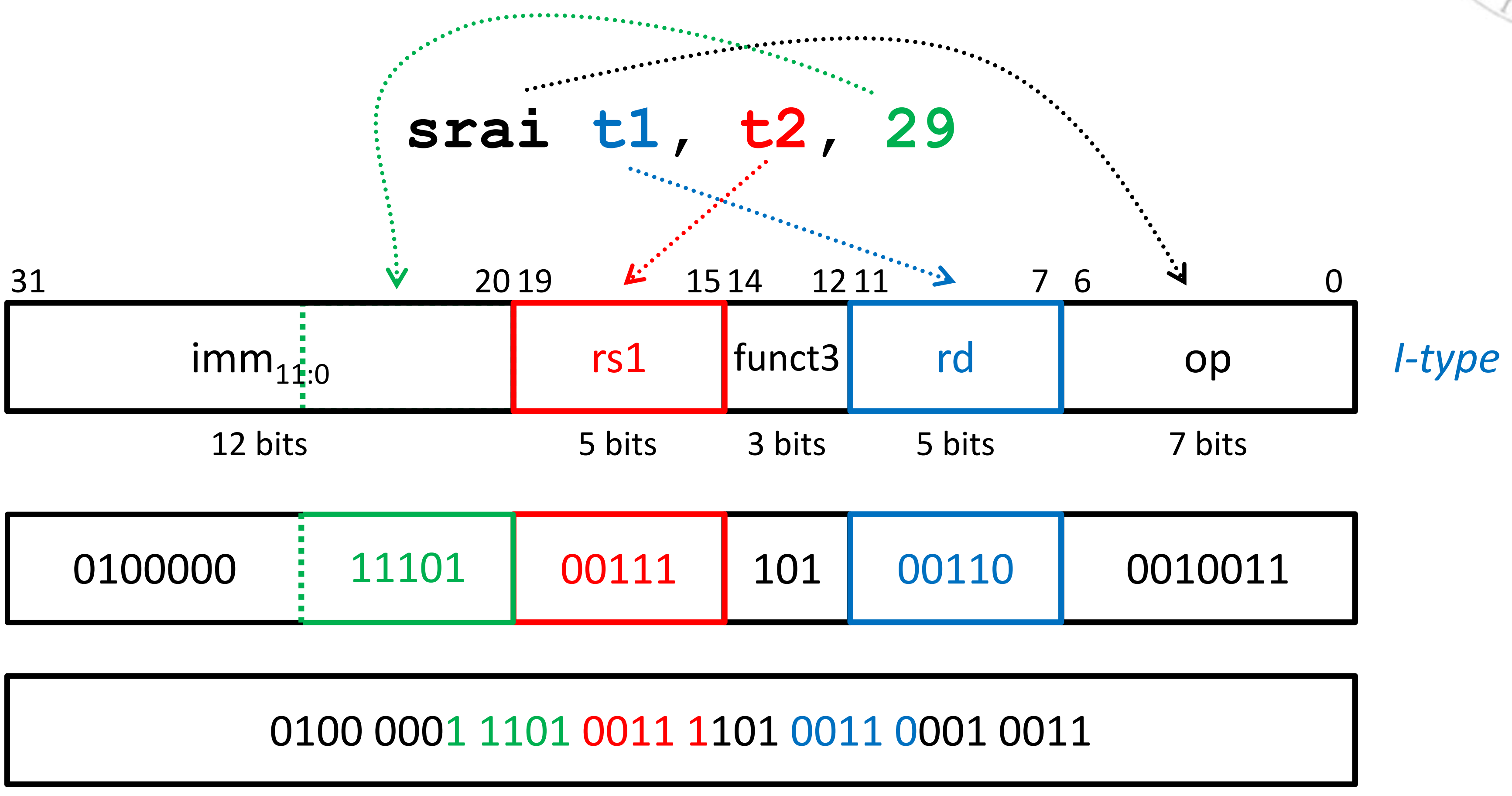
# From assembly to machine code

## Example: I-type instruction (iii)



# From assembly to machine code

## Example: I-type instruction (iii)

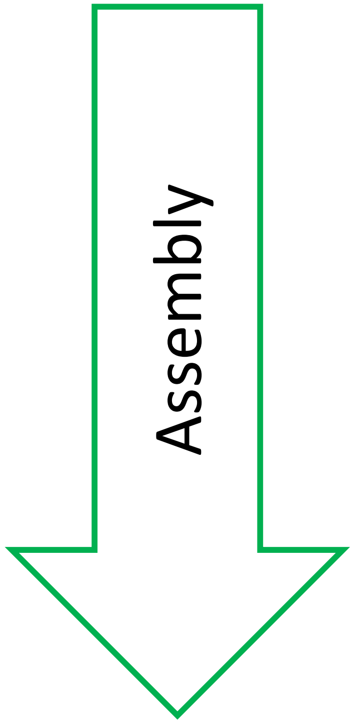


**0x41d3d313**

# From assembly to machine code

## Example: S-type instruction

```
sb t5, 45(zero)
```

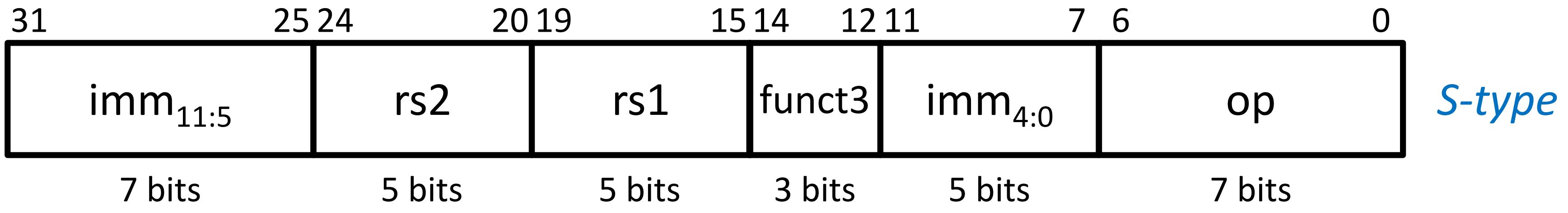


```
0x03e006a3
```

# From assembly to machine code

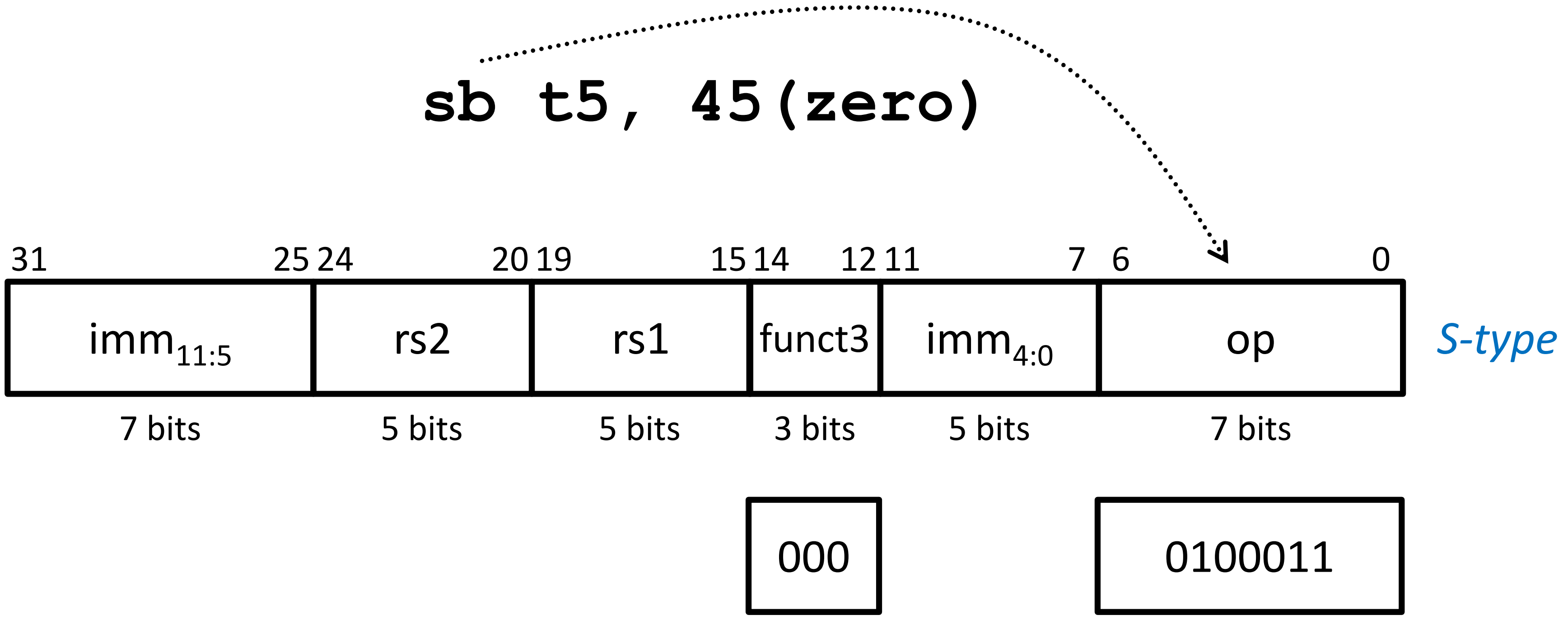
## Example: S-type instruction

```
sb t5, 45(zero)
```



# From assembly to machine code

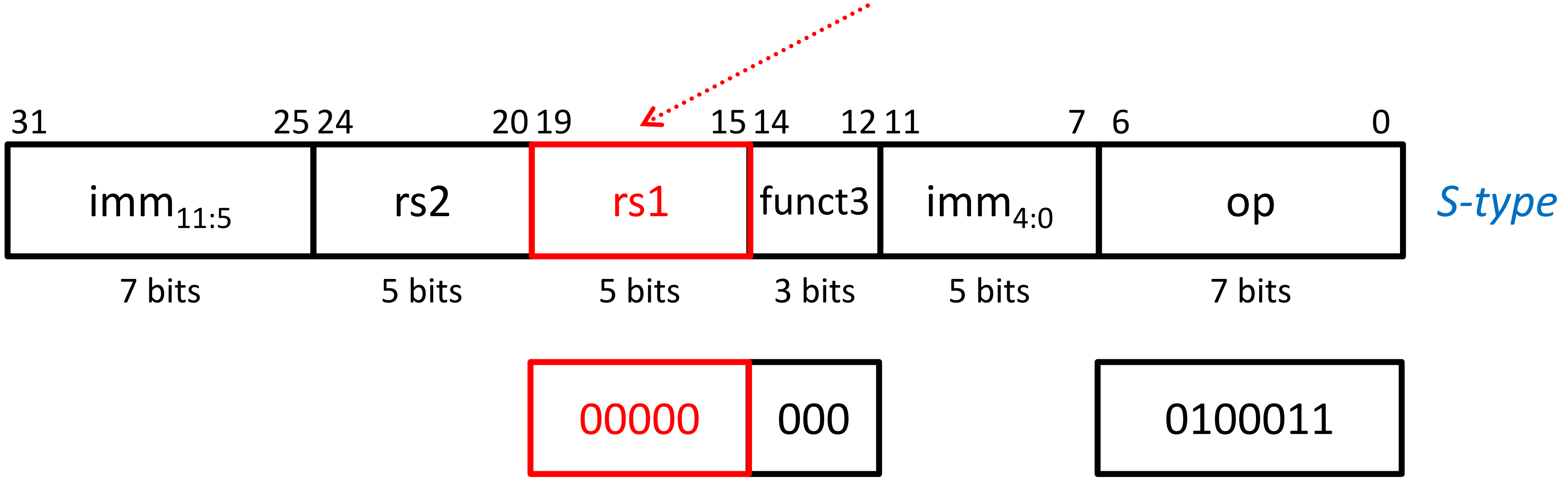
## Example: S-type instruction



# From assembly to machine code

## Example: S-type instruction

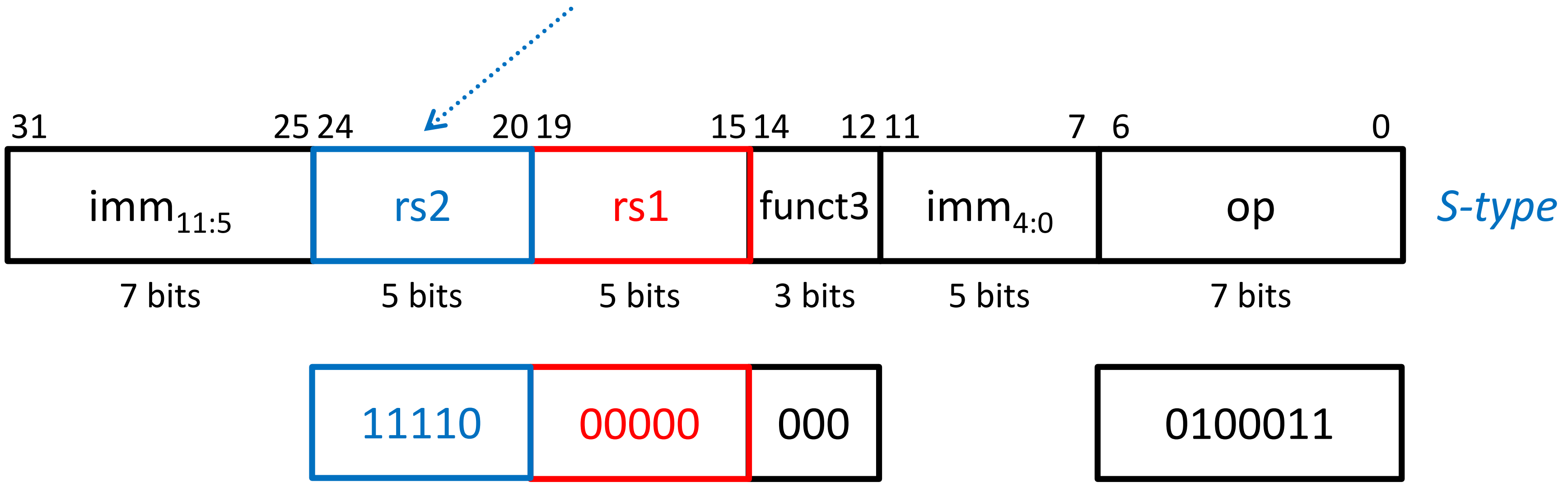
```
sb t5, 45(zero) ≡ sb x30, x0, 45
```



# From assembly to machine code

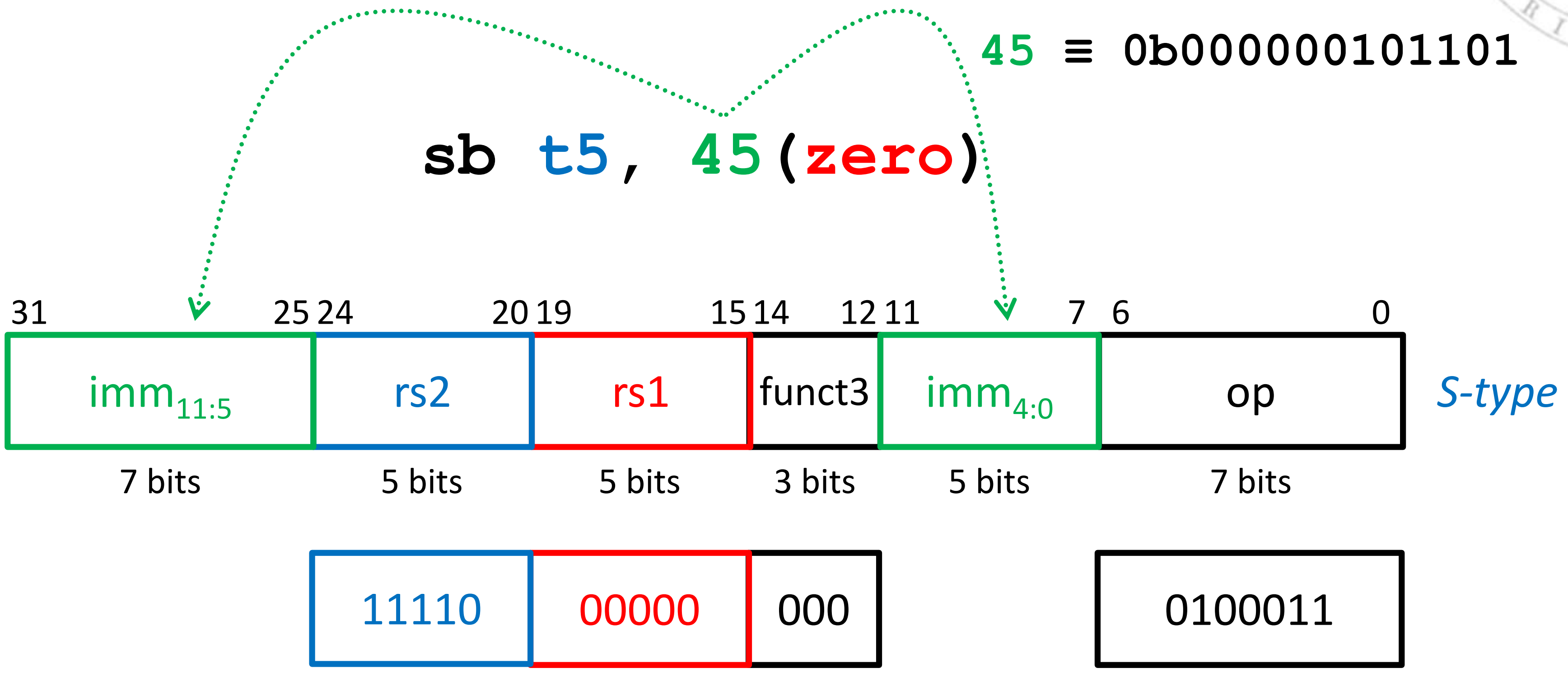
## Example: S-type instruction

```
sb t5, 45(zero) ≡ sb x30, x0, 45
```



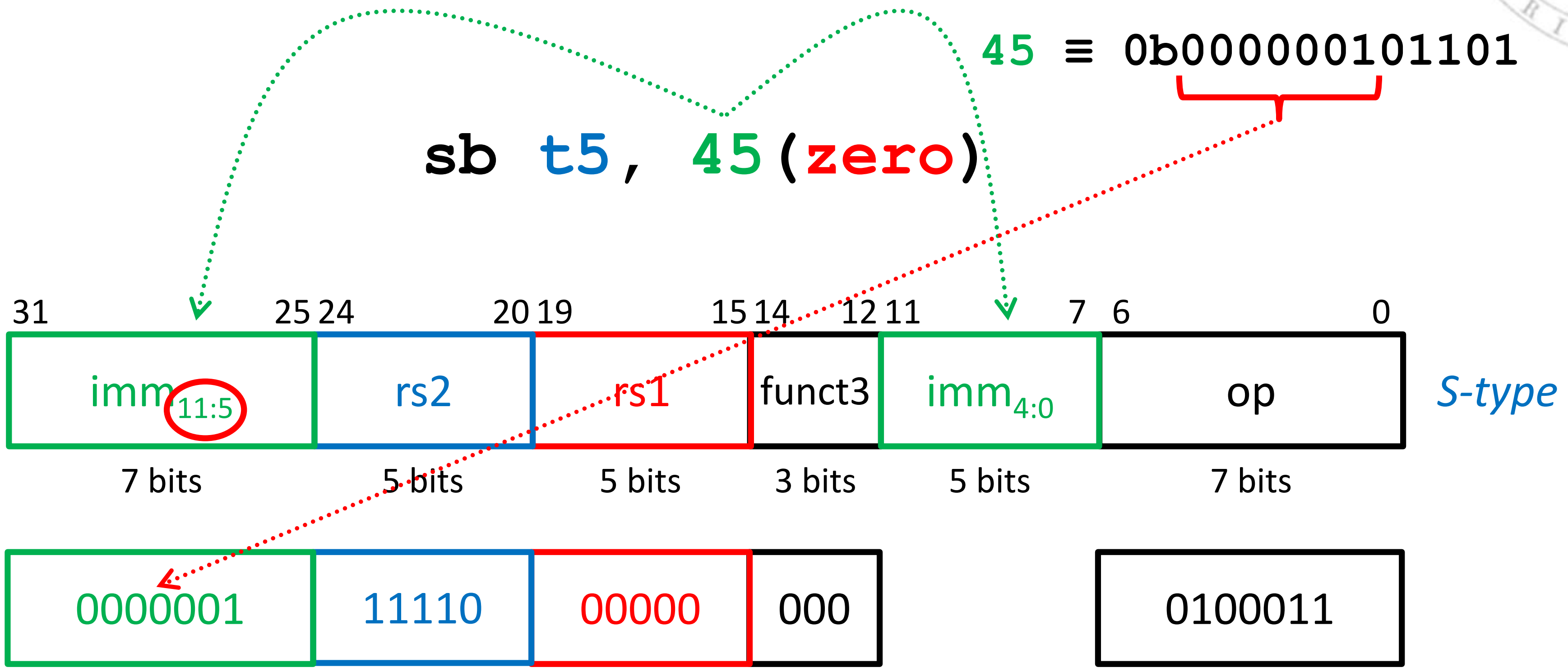
# From assembly to machine code

## Example: S-type instruction



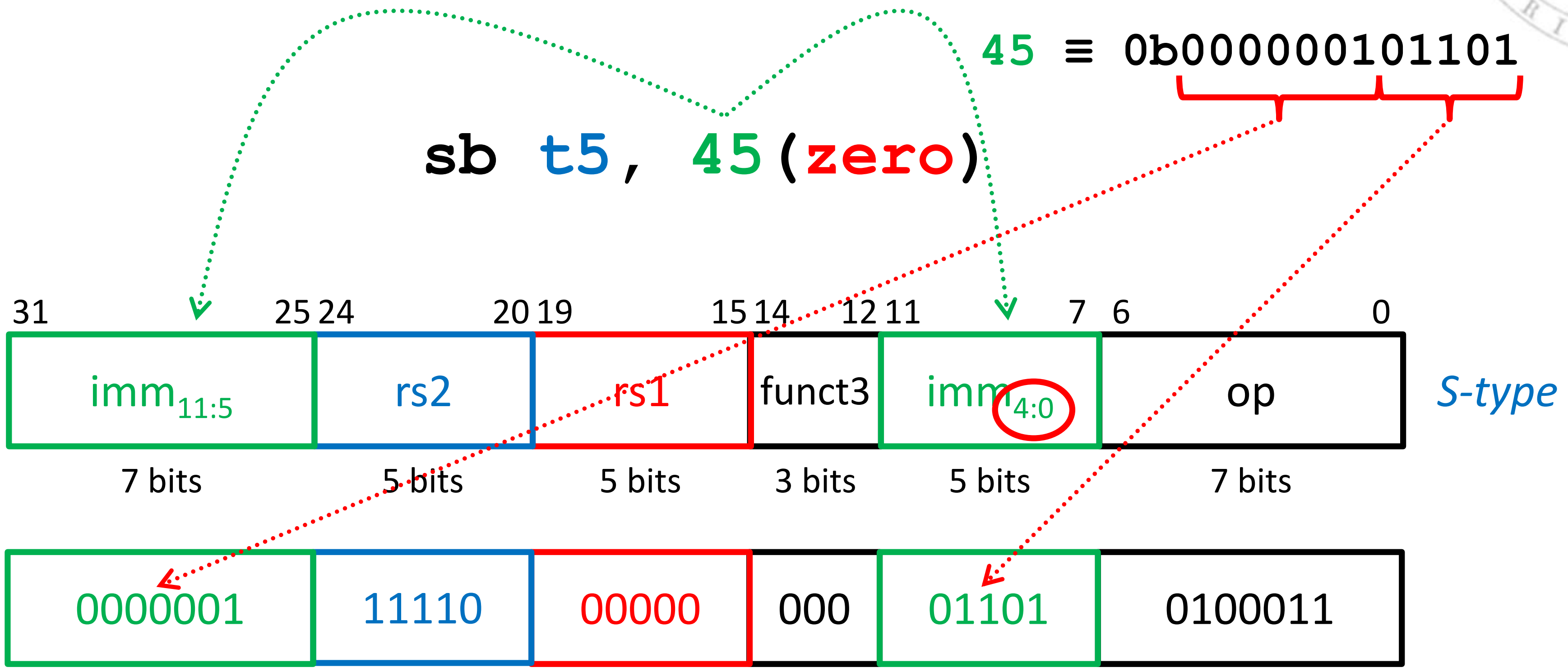
# From assembly to machine code

## Example: S-type instruction



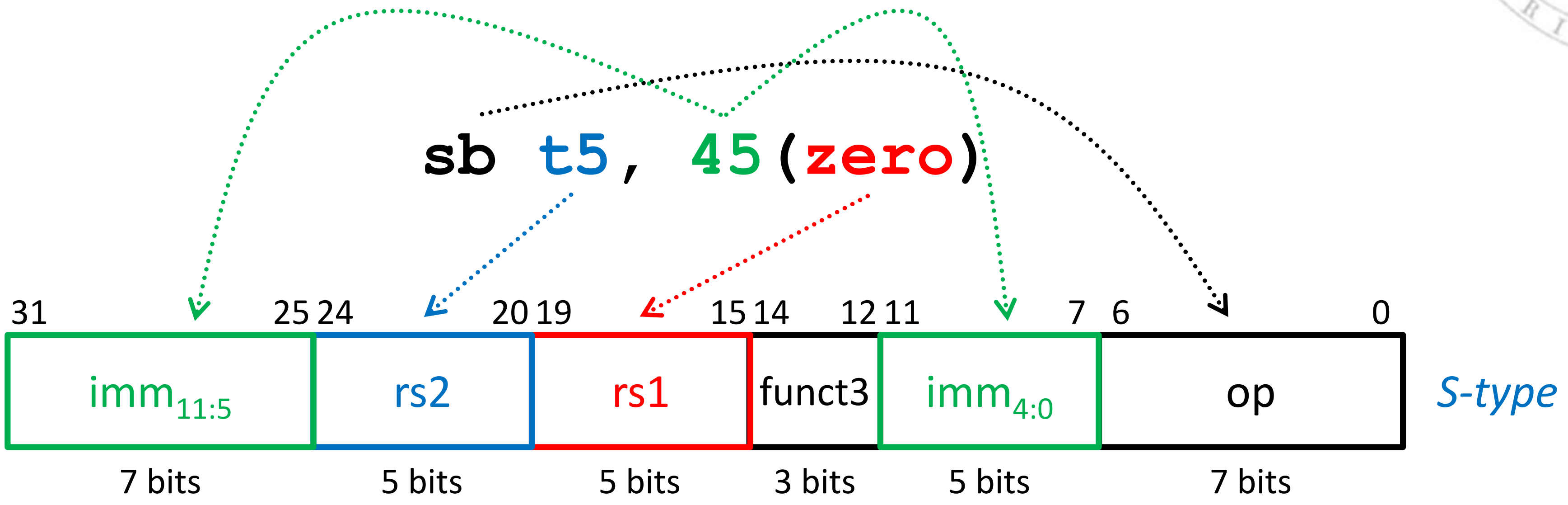
# From assembly to machine code

## Example: S-type instruction



# From assembly to machine code

## Example: S-type instruction



| | | | | | |
|---|---|---|---|---|---|
| 0000001 | 11110 | 00000 | 000 | 01101 | 0100011 |

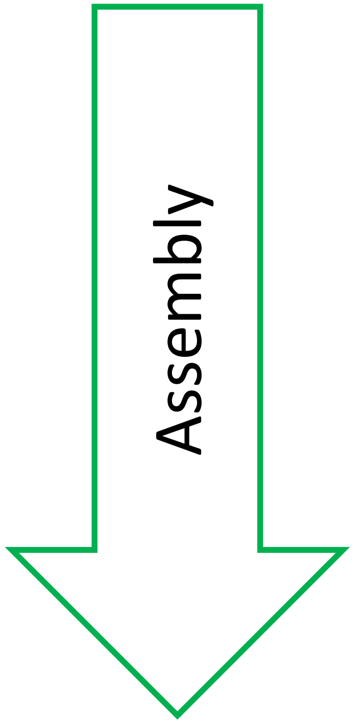
0000 0011 1110 0000 0000 0110 1010 0011

**0x03e006a3**

# From assembly to machine code

## Example: B-type instruction

```
beq s0, t5, 0x10
```



```
0x01e40863
```

## From assembly to machine code

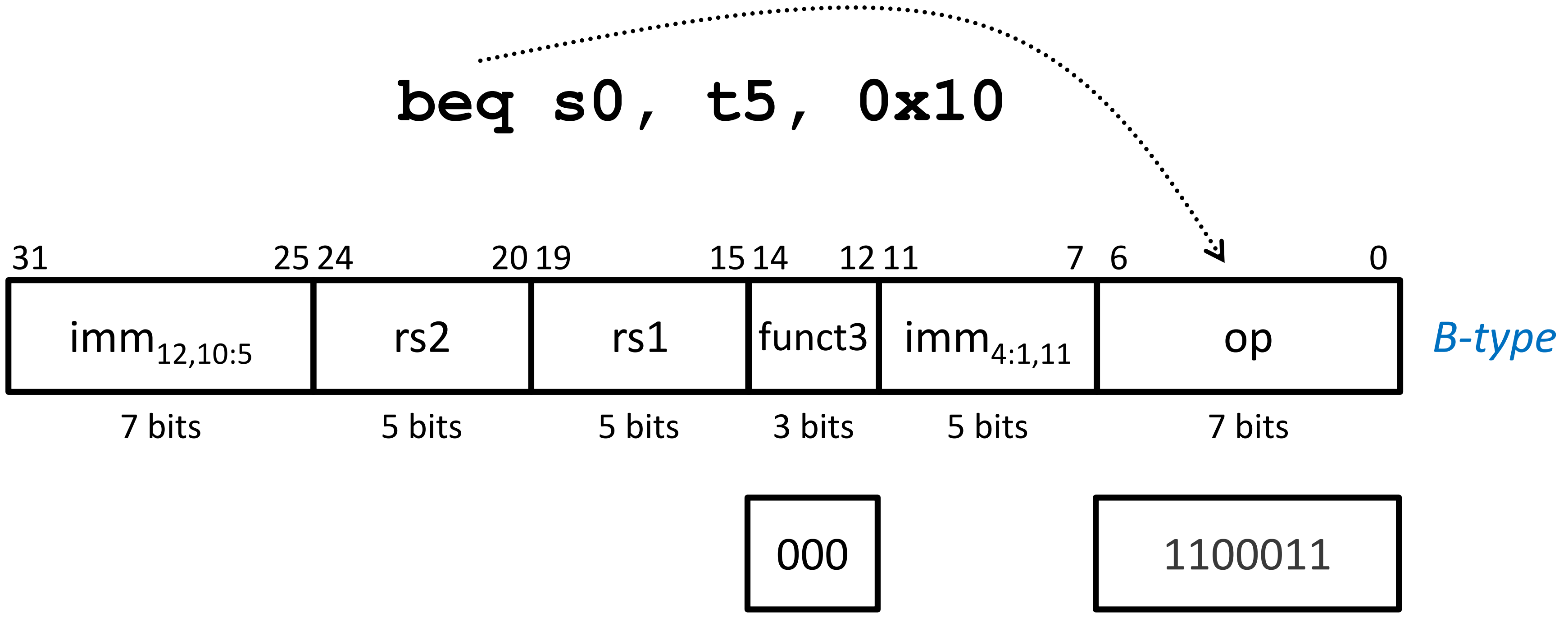
### Example: B-type instruction

```
beq s0, t5, 0x10
```

| 31 – 25 | 24 – 20 | 19 – 15 | 14 – 12 | 11 – 7 | 6 – 0 | |
|---|---|---|---|---|---|---|
| $\text{imm}_{12,10:5}$ | rs2 | rs1 | funct3 | $\text{imm}_{4:1,11}$ | op | *B-type* |
| 7 bits | 5 bits | 5 bits | 3 bits | 5 bits | 7 bits | |

# From assembly to machine code

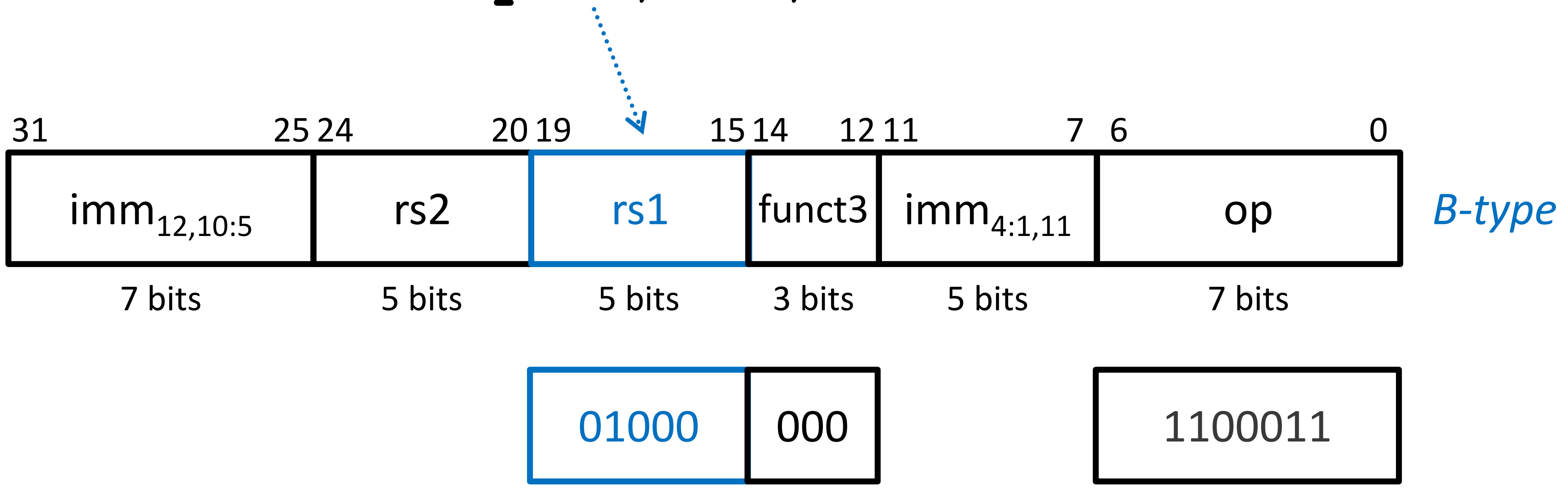
## Example: B-type instruction



# From assembly to machine code

## Example: B-type instruction

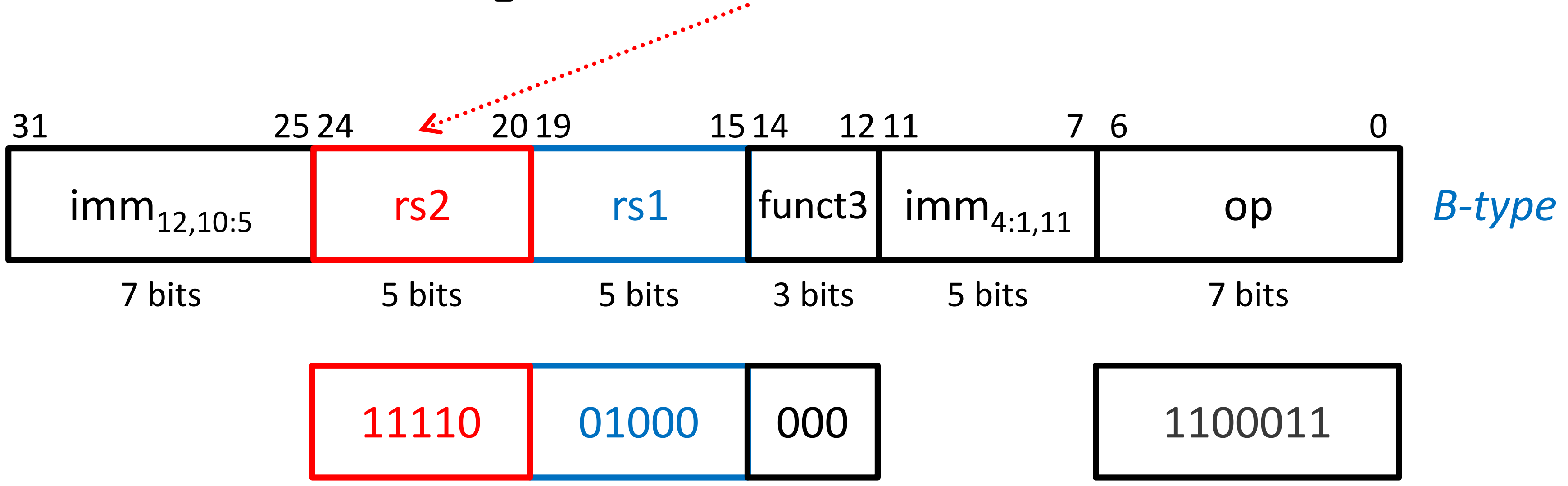
```
beq s0, t5, 0x10 ≡ beq x8, x30, 0x10
```



# From assembly to machine code

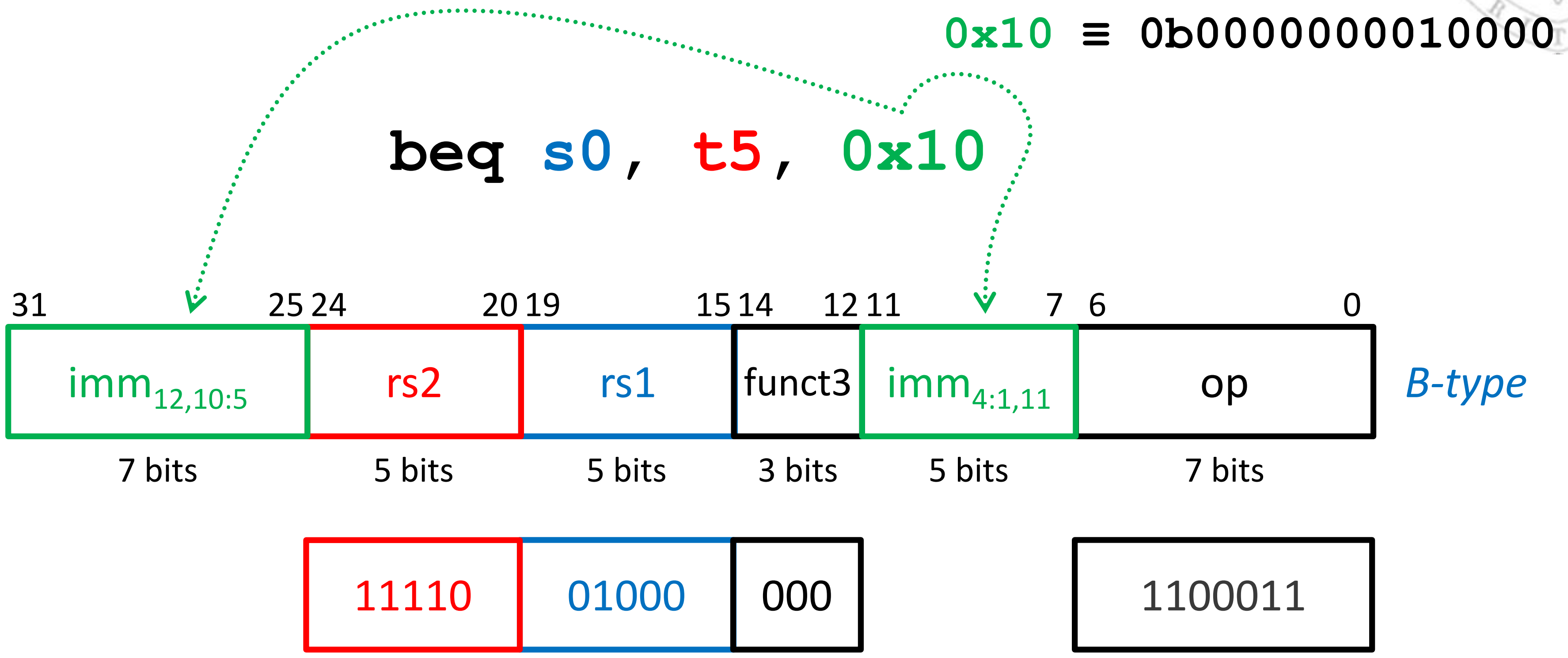
## Example: B-type instruction

```
beq s0, t5, 0x10  ≡ beq x8, x30, 0x10
```



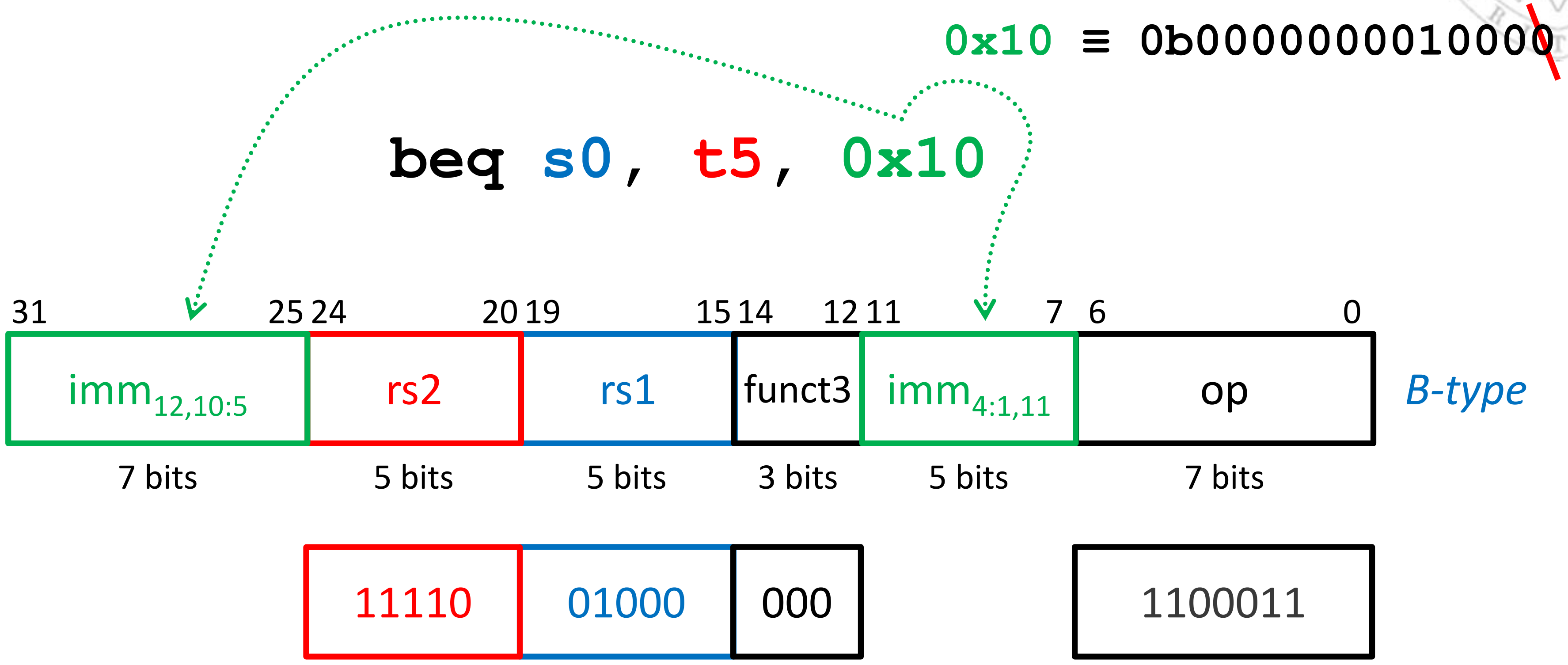
# From assembly to machine code

## Example: B-type instruction



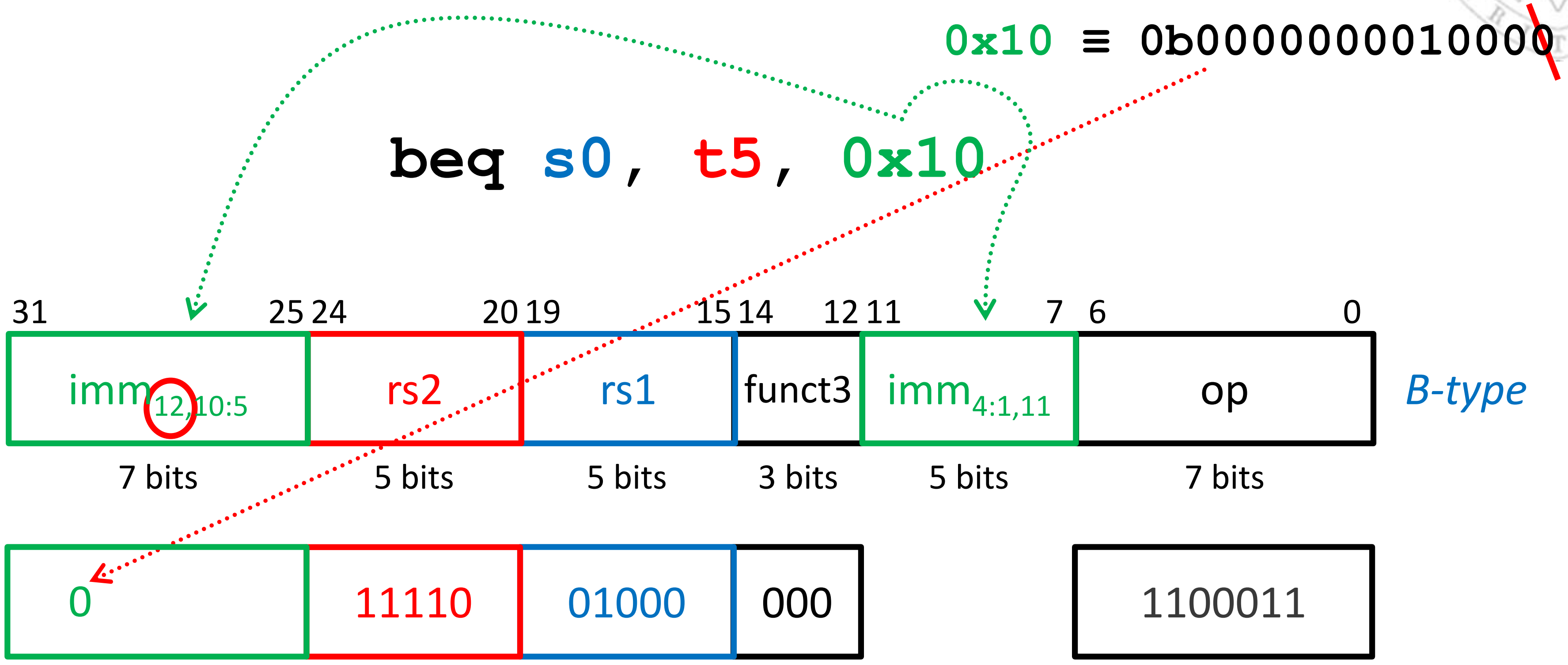
# From assembly to machine code

## Example: B-type instruction



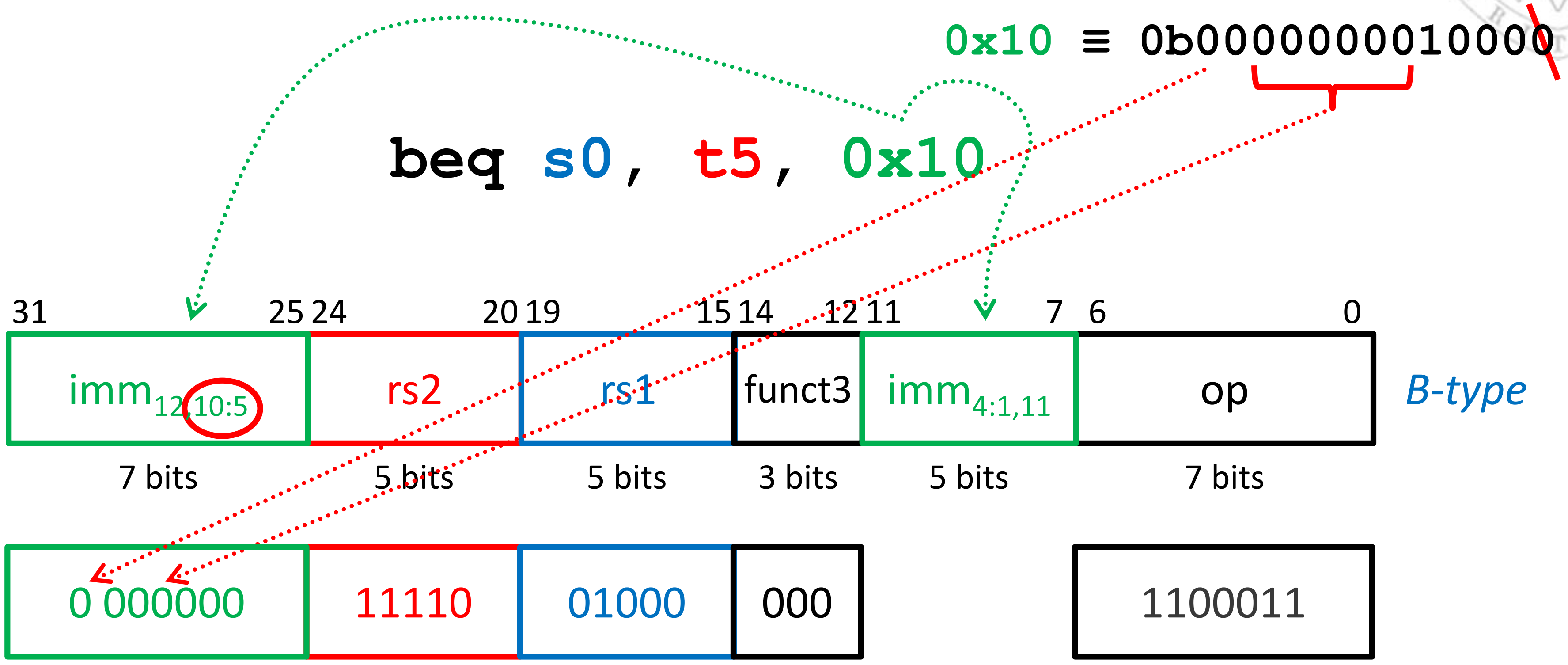
# From assembly to machine code

## Example: B-type instruction



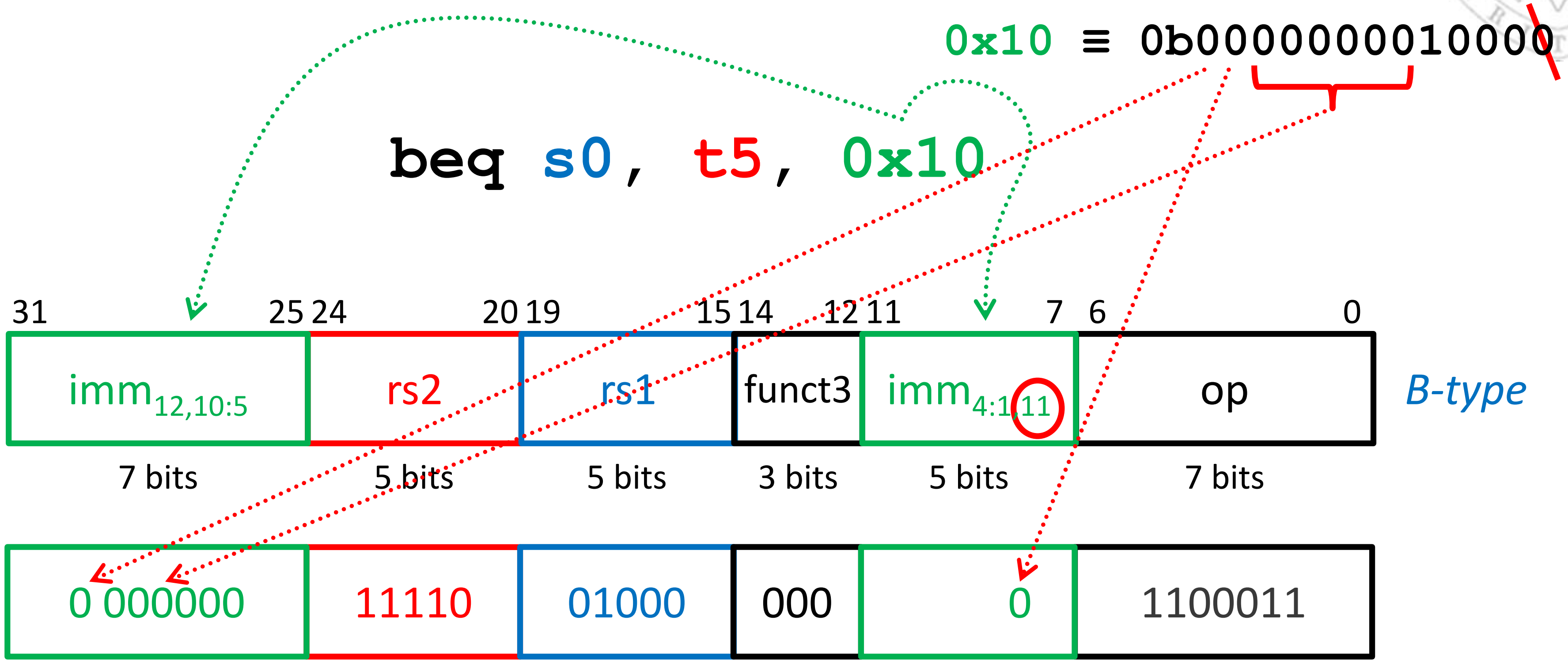
# From assembly to machine code

## Example: B-type instruction



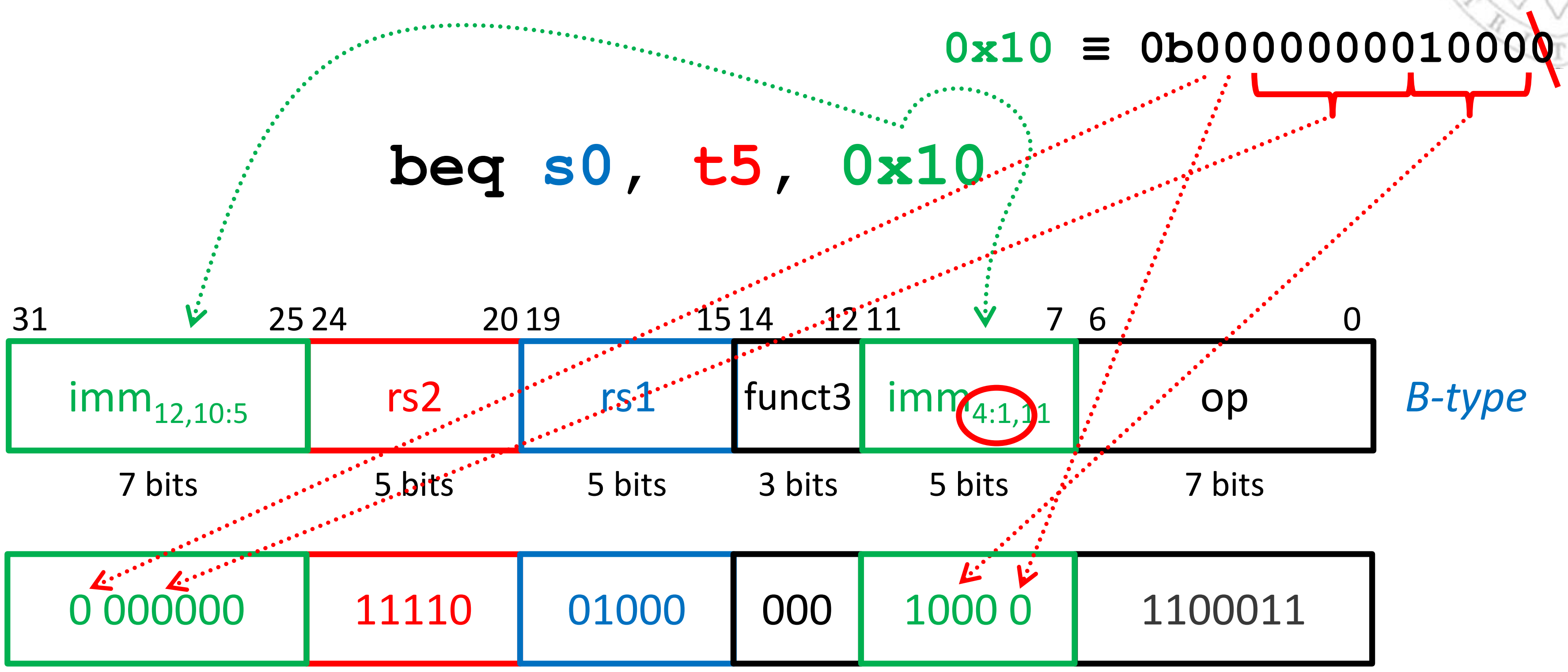
# From assembly to machine code

## Example: B-type instruction



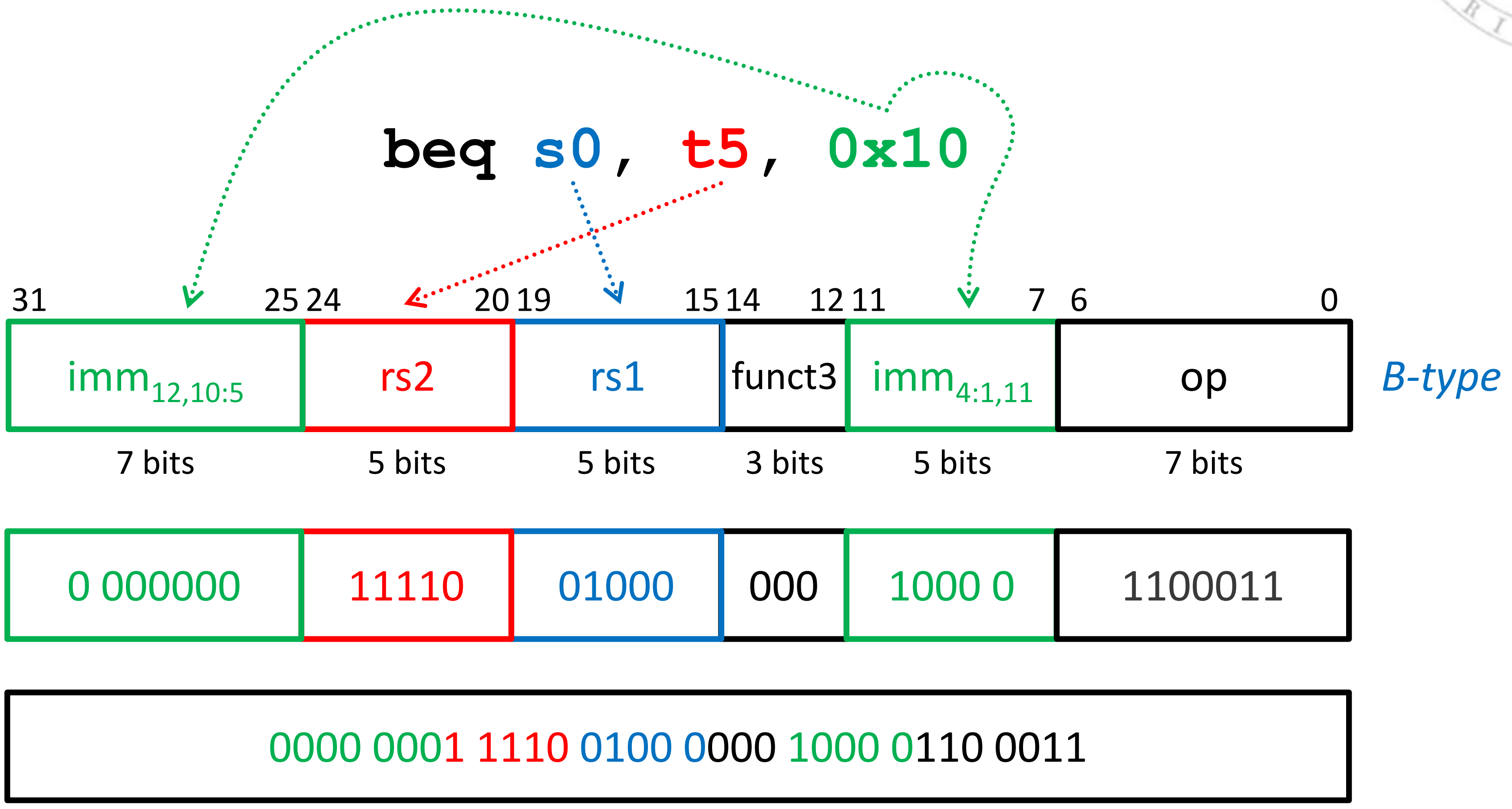
# From assembly to machine code

## Example: B-type instruction



# From assembly to machine code

## Example: B-type instruction

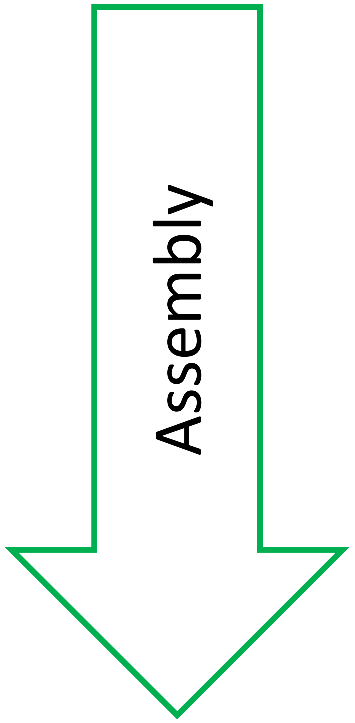


**0x01e40863**

## From assembly to machine code

Example: U-type instruction

```
lui s5, 0x8cdef
```

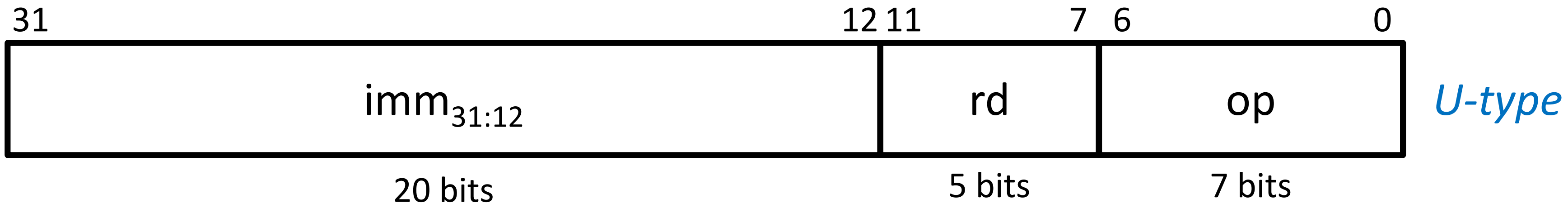


```
0x8cdefab7
```

# From assembly to machine code

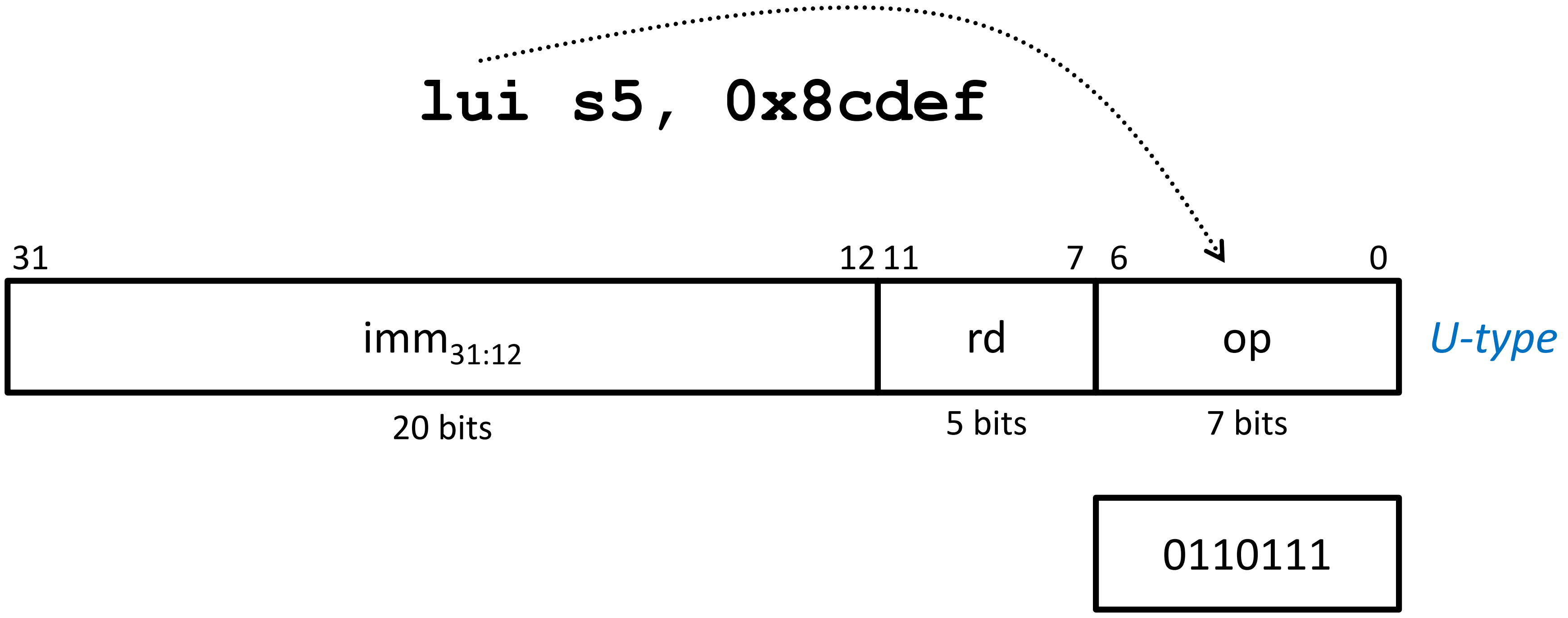
## Example: U-type instruction

```
lui s5, 0x8cdef
```



# From assembly to machine code

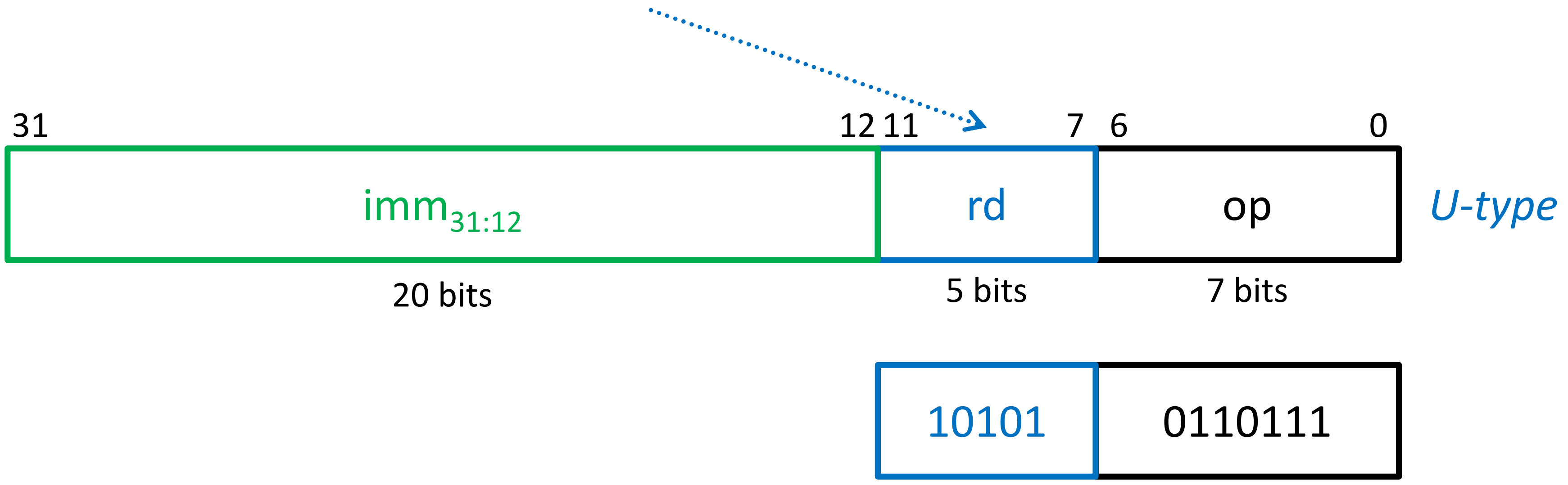
## Example: U-type instruction



# From assembly to machine code

## Example: U-type instruction

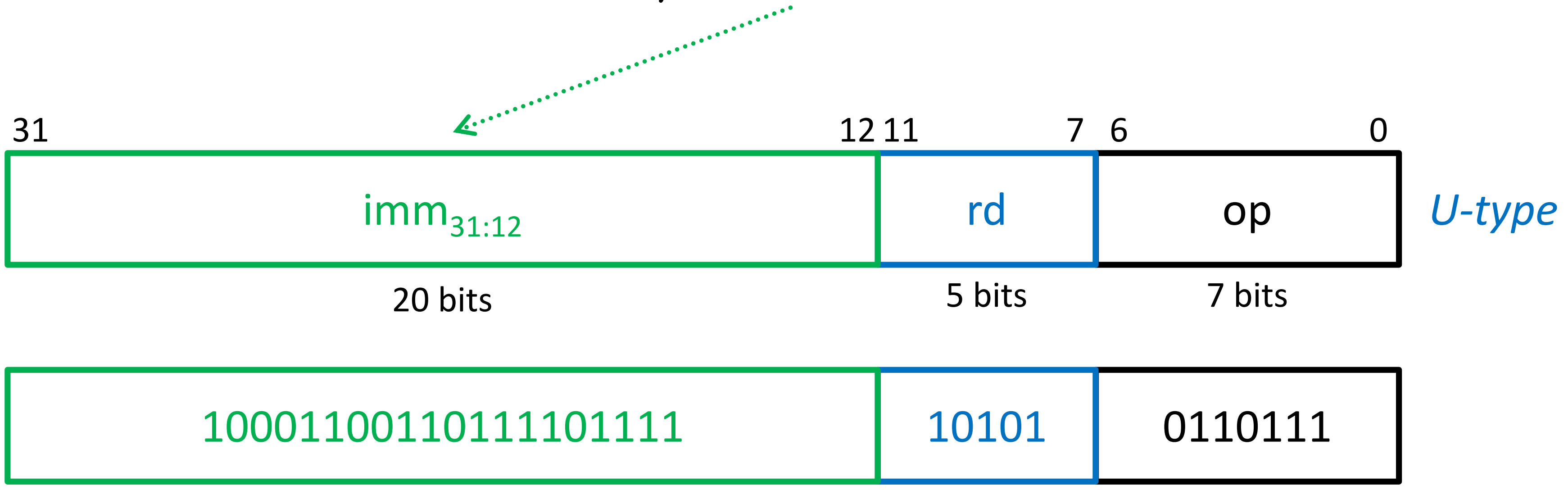
```
lui s5, 0x8cdef ≡ lui x21, 0x8cdef
```



# From assembly to machine code

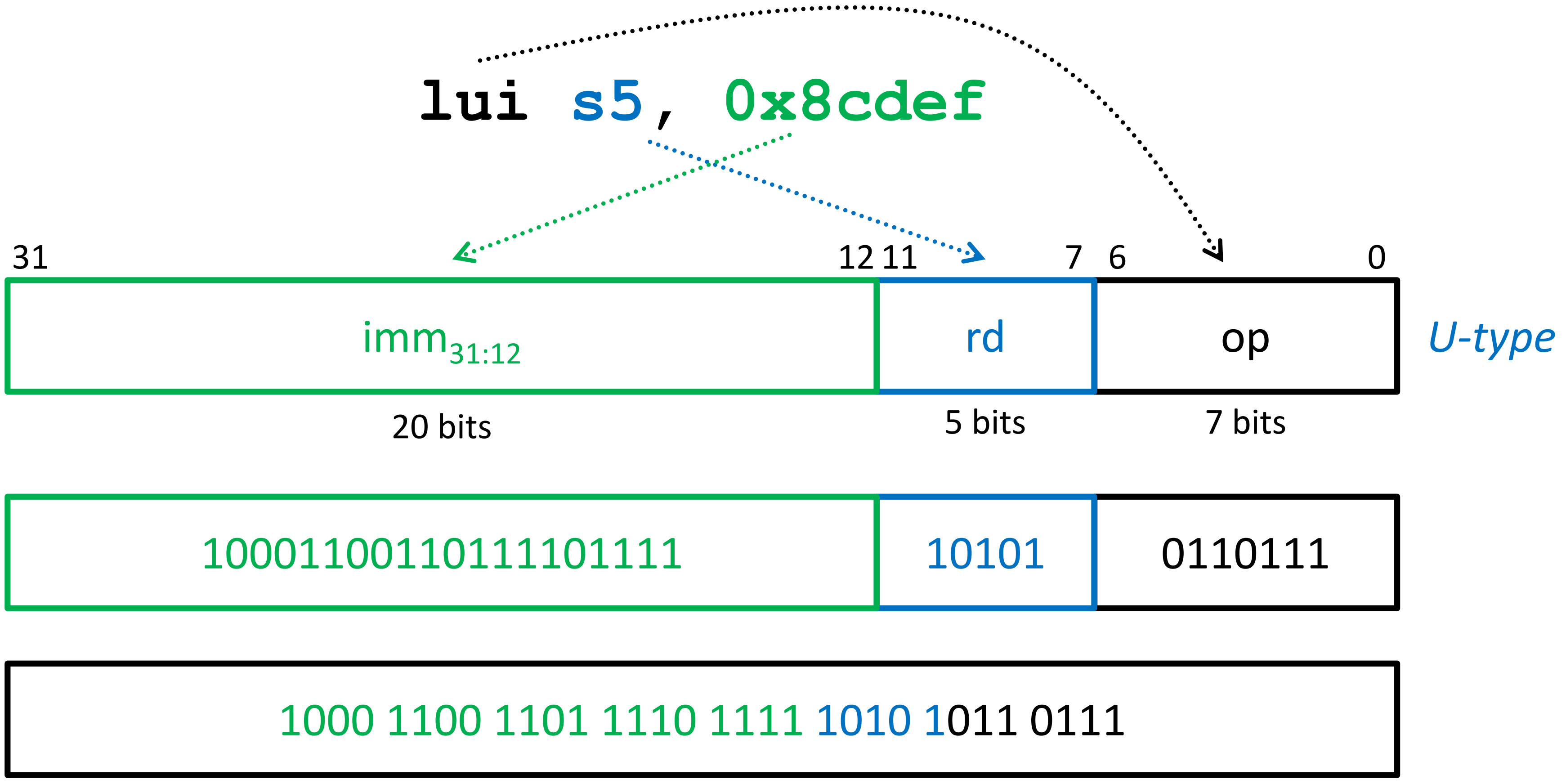
## Example: U-type instruction

```
lui s5, 0x8cdef
```



# From assembly to machine code

## Example: U-type instruction

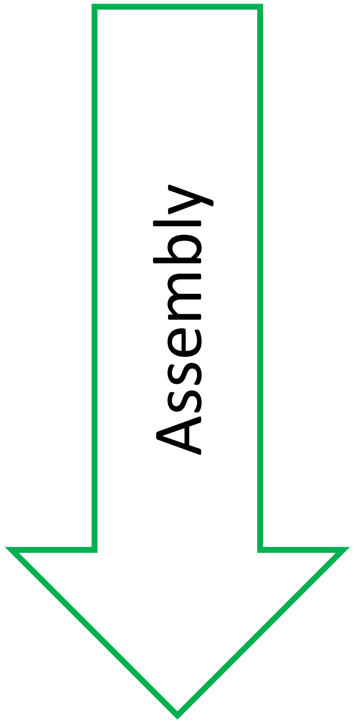


**0x8cdefab7**

## From assembly to machine code

Example: J-type instruction

```
jal ra, 0xa67f8
```

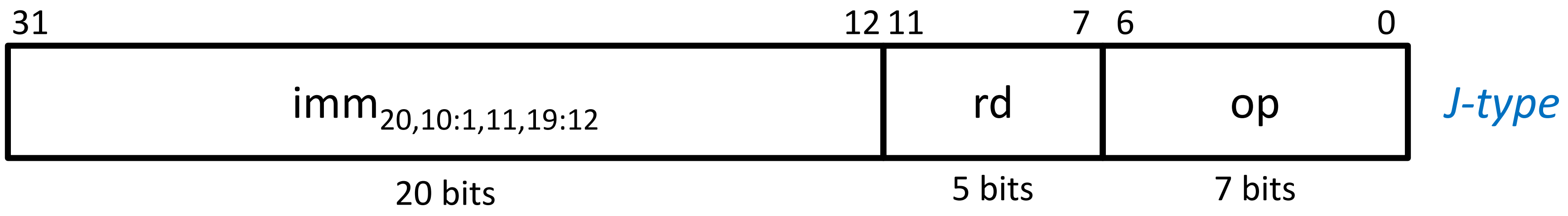


```
0x7f8a60ef
```

# From assembly to machine code

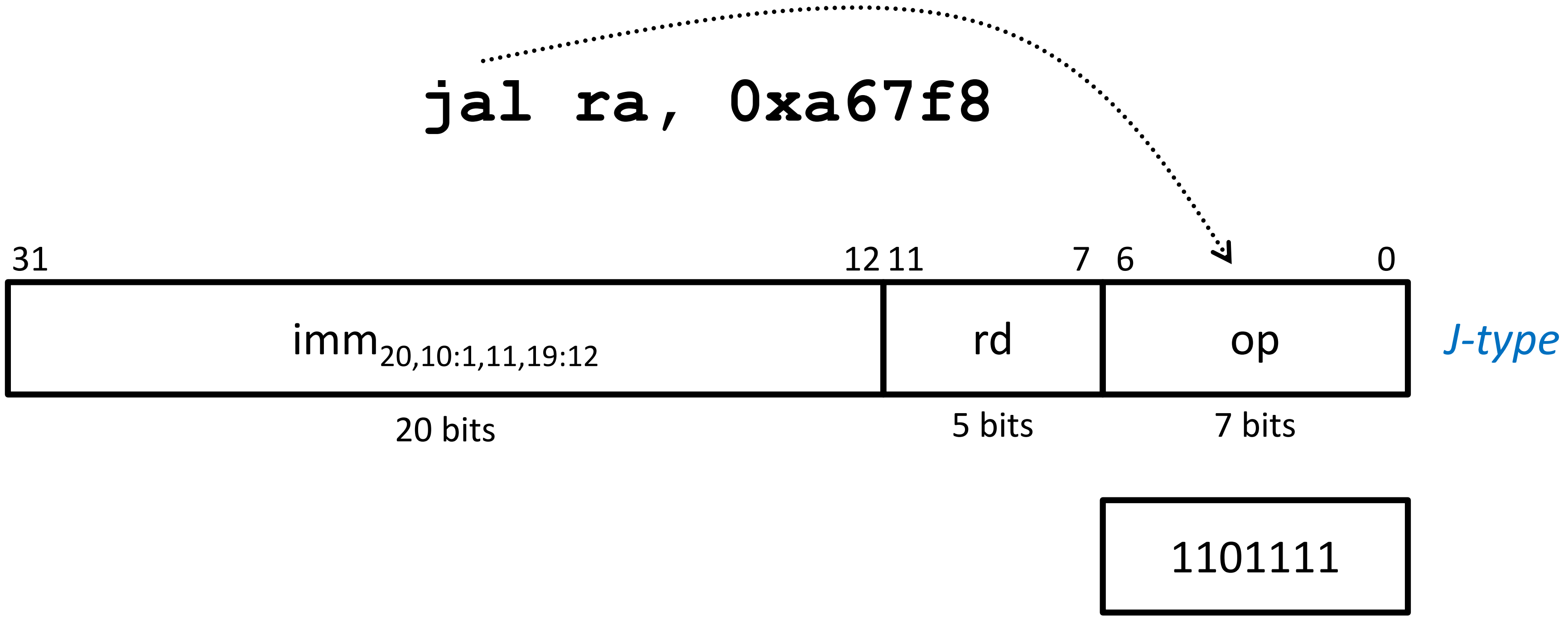
## Example: J-type instruction

```
jal ra, 0xa67f8
```



## From assembly to machine code

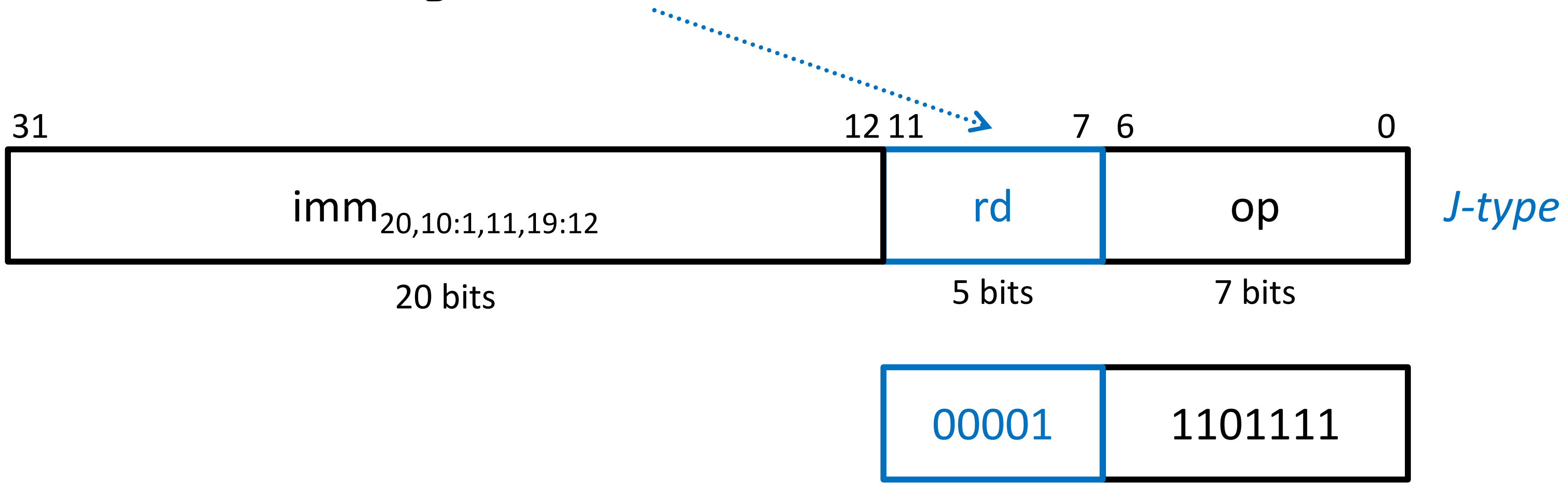
### Example: J-type instruction



# From assembly to machine code

## Example: J-type instruction

`jal ra, 0xa67f8` ≡ `jal x1, 0xa67f8`

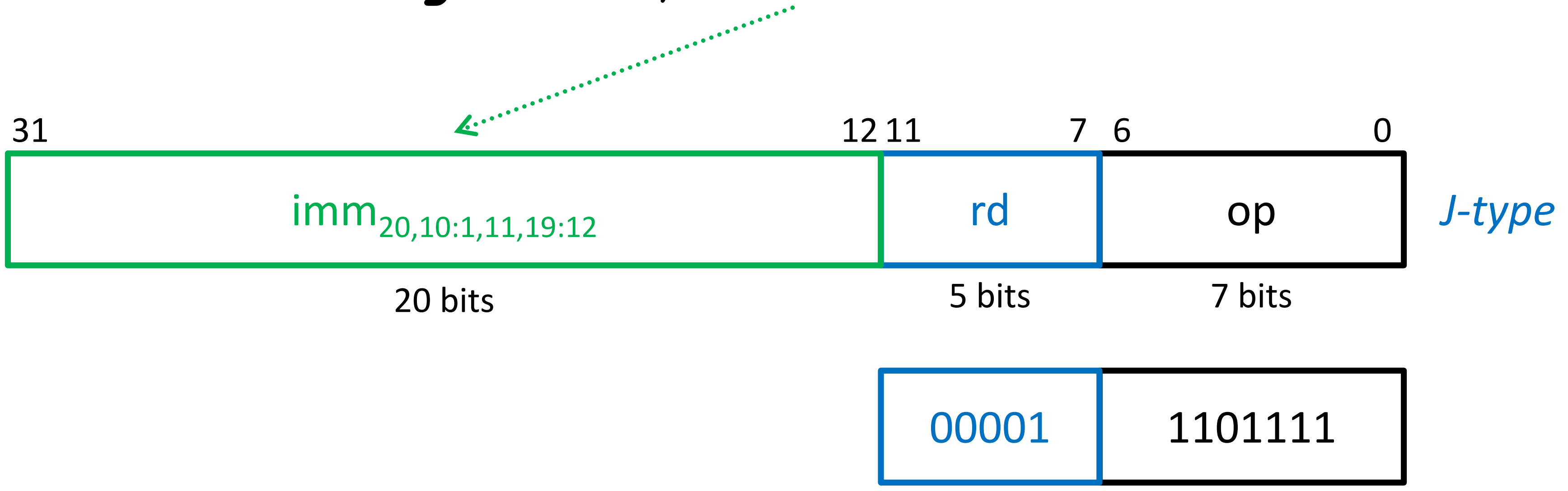


# From assembly to machine code

## Example: J-type instruction

`0xa67f8 ≡ 0b01010011001111111000`

`jal ra, 0xa67f8`

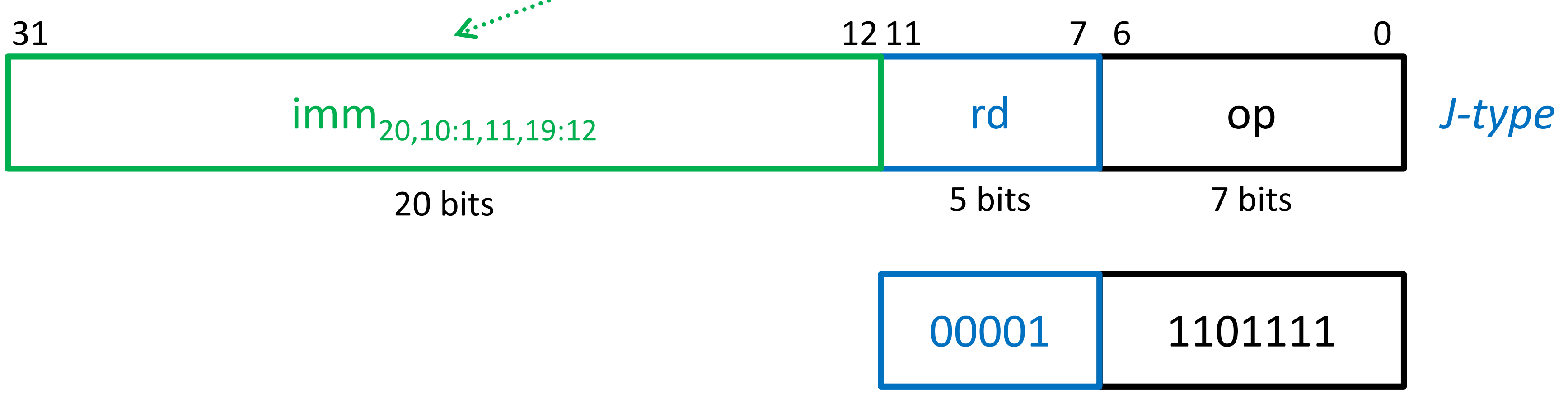


# From assembly to machine code

## Example: J-type instruction

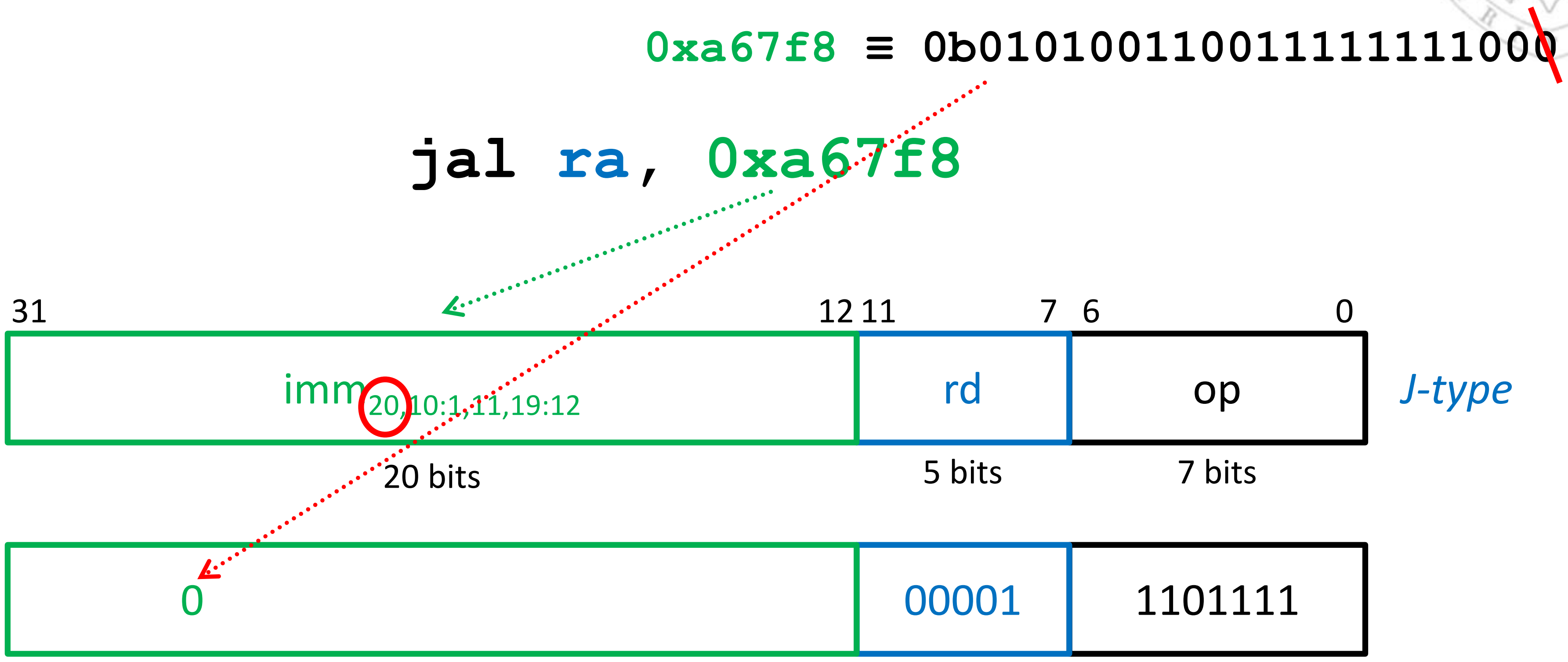
`0xa67f8` ≡ `0b01010011001111111100~~0~~`

```
jal ra, 0xa67f8
```



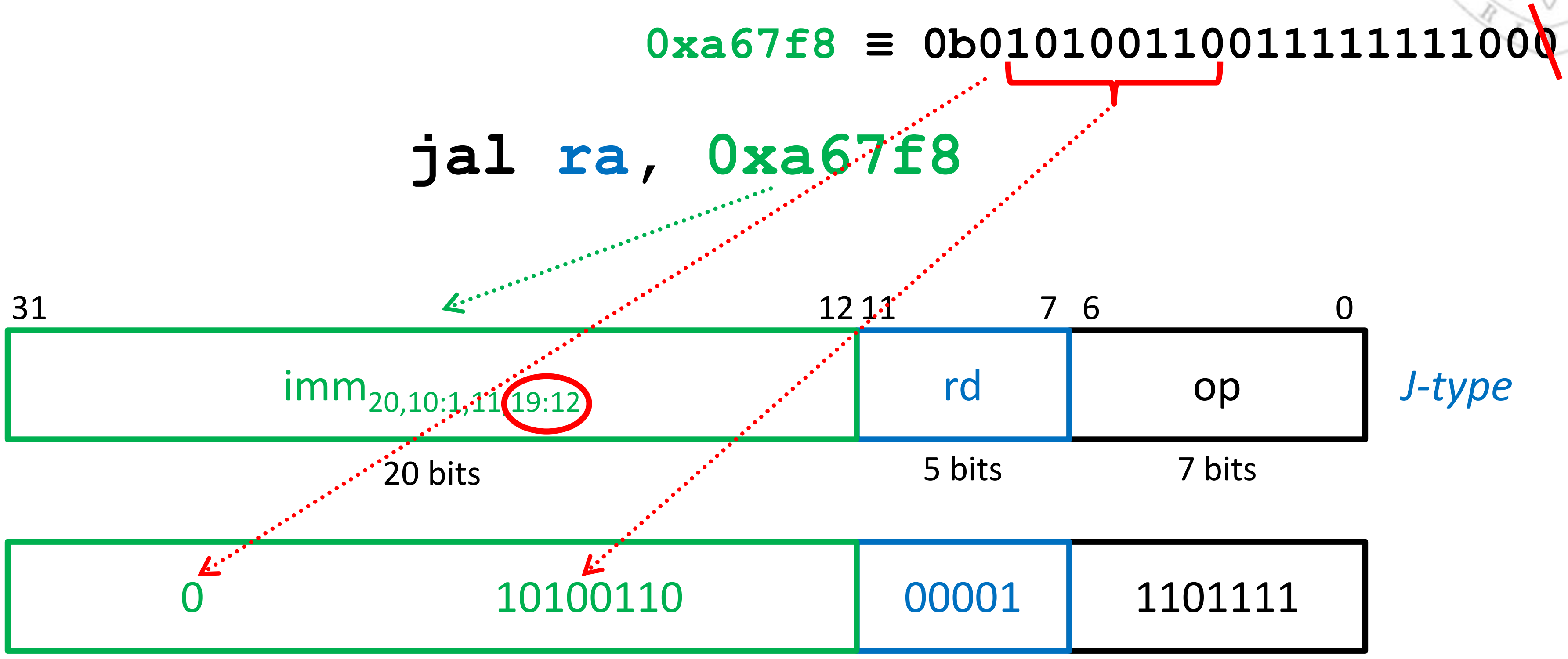
# From assembly to machine code

## Example: J-type instruction



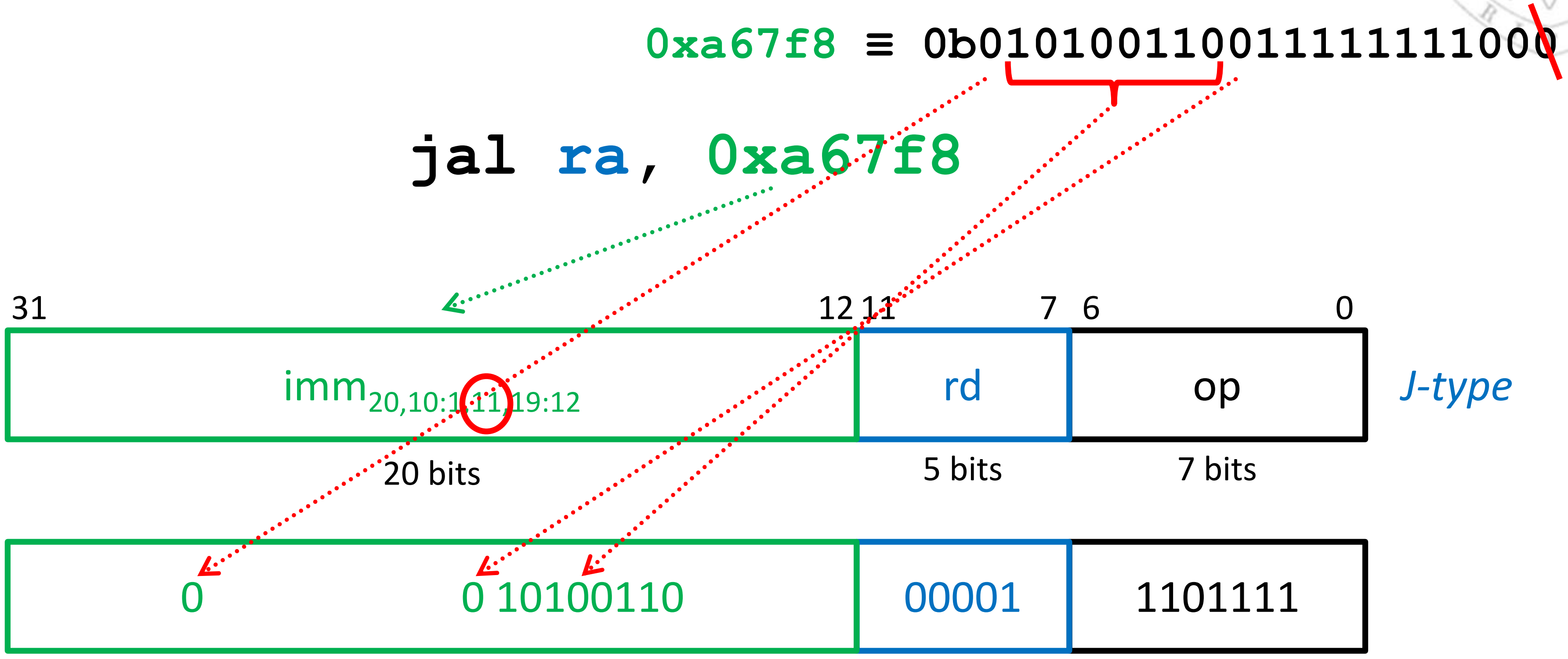
# From assembly to machine code

## Example: J-type instruction



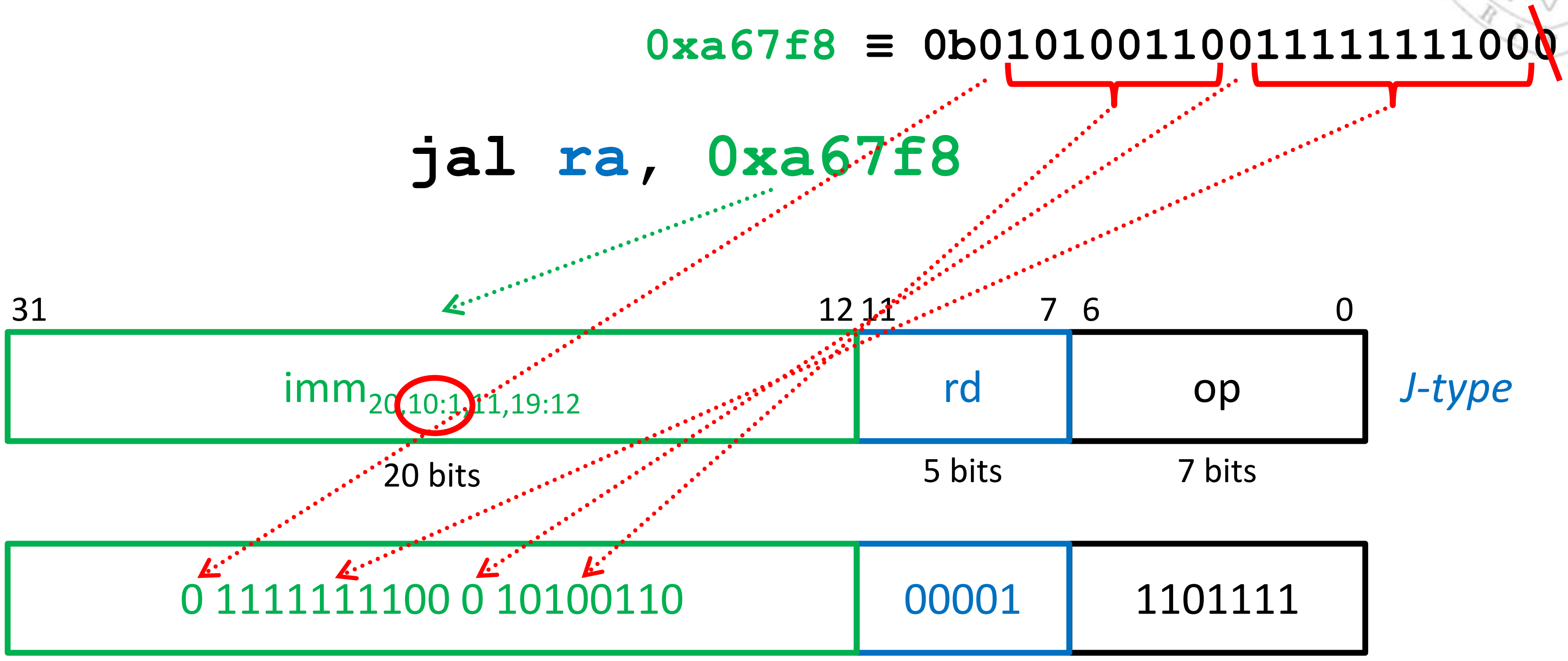
# From assembly to machine code

## Example: J-type instruction



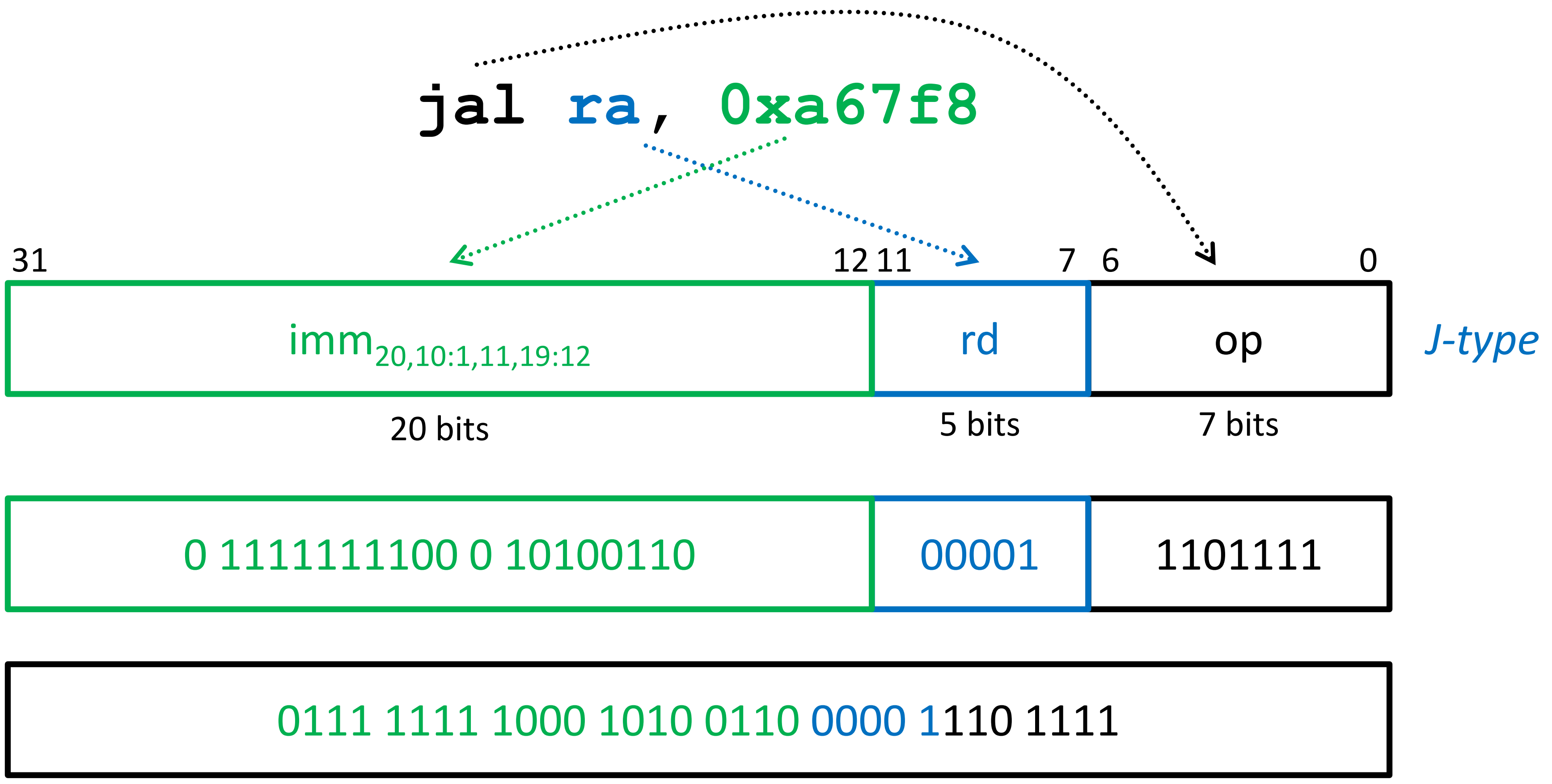
# From assembly to machine code

## Example: J-type instruction



# From assembly to machine code

## Example: J-type instruction

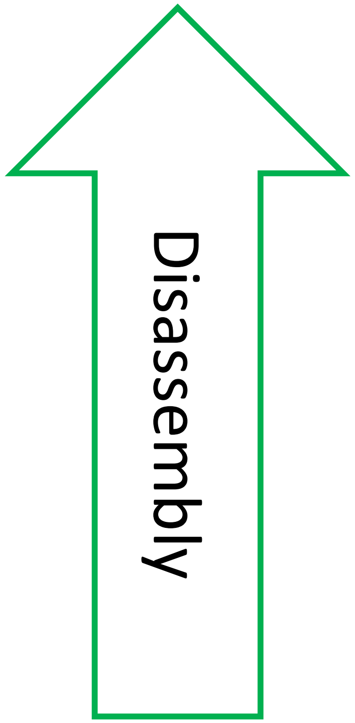


**`0x7f8a60ef`**

# From machine code to assembly

Example (i)

```
sub t2, t4, t6
```



```
0x41fe83b3
```

# From machine code to assembly

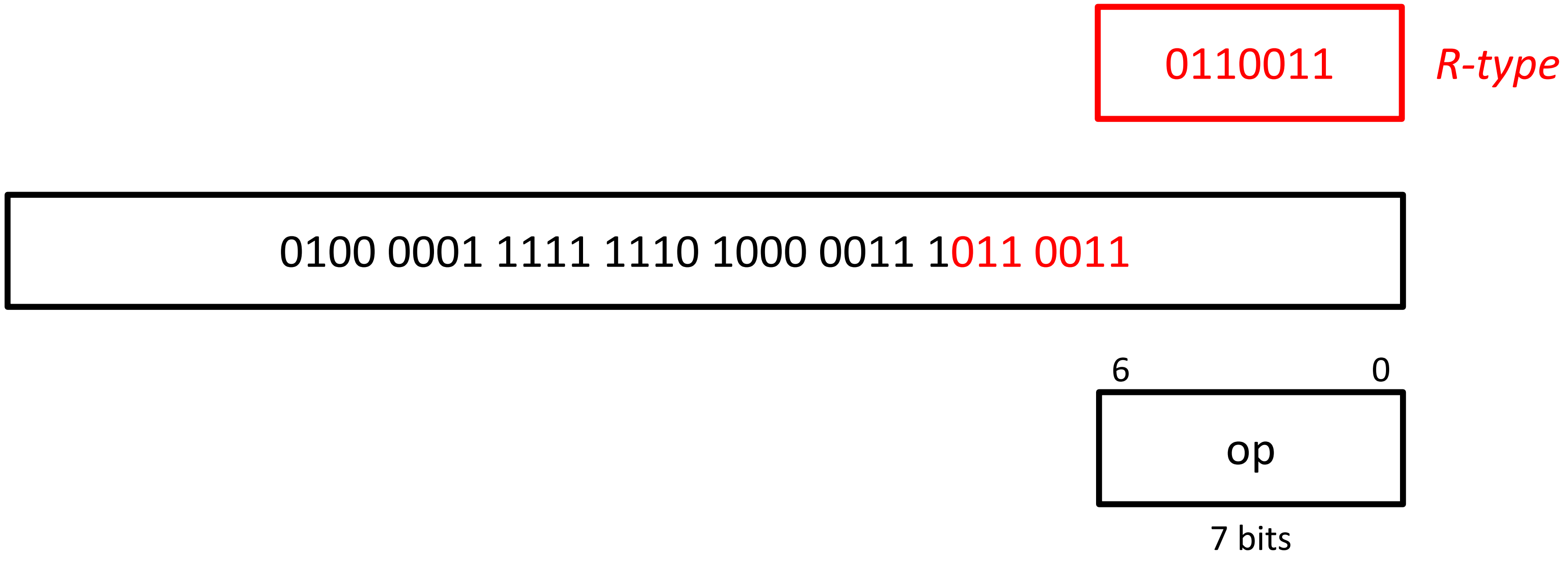
## Example (i)

```
0100 0001 1111 1110 1000 0011 1011 0011
```

```
0x41fe83b3
```

# From machine code to assembly

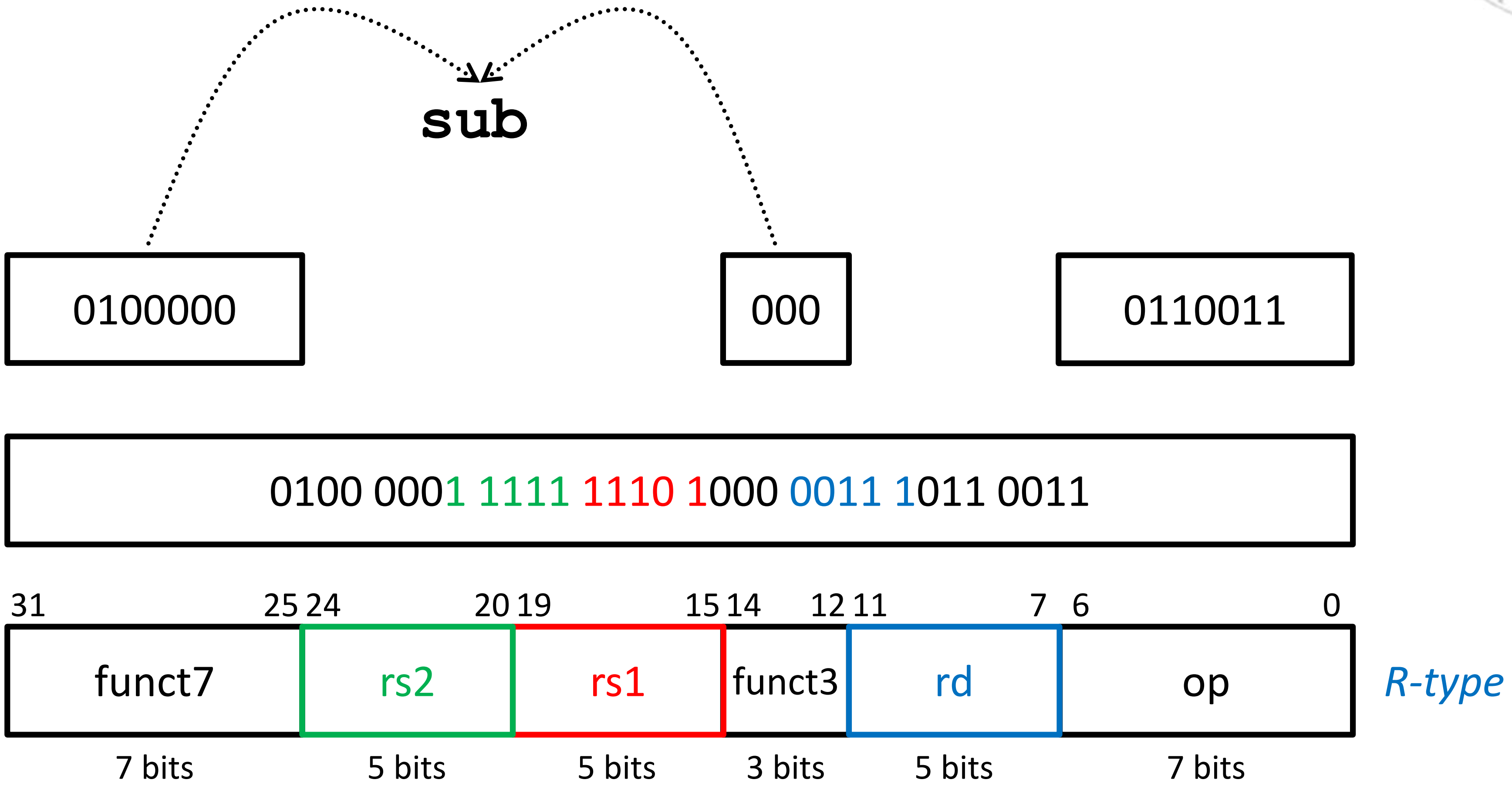
## Example (i)



**0x41fe83b3**

# From machine code to assembly

## Example (i)

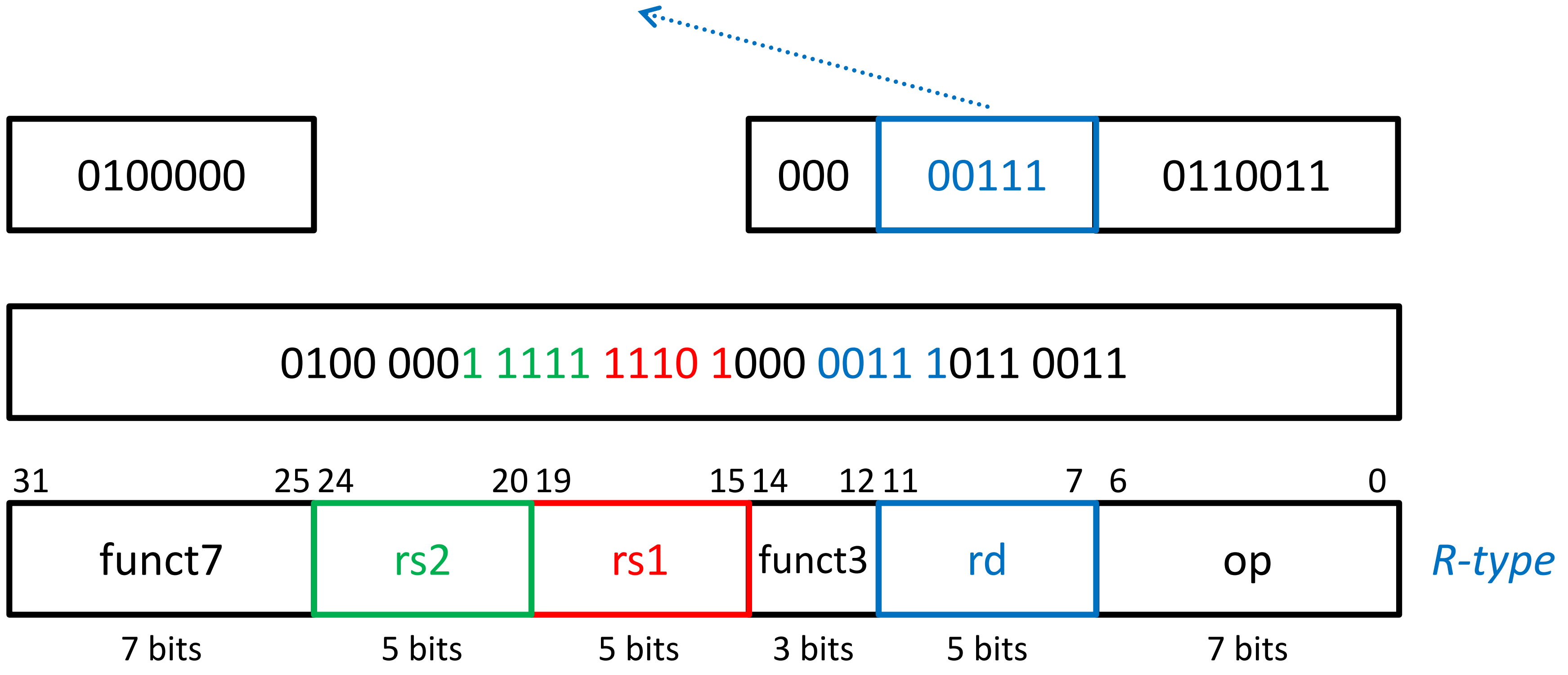


**`0x41fe83b3`**

## From machine code to assembly

Example (i)

`sub t2` ≡ `sub x7`

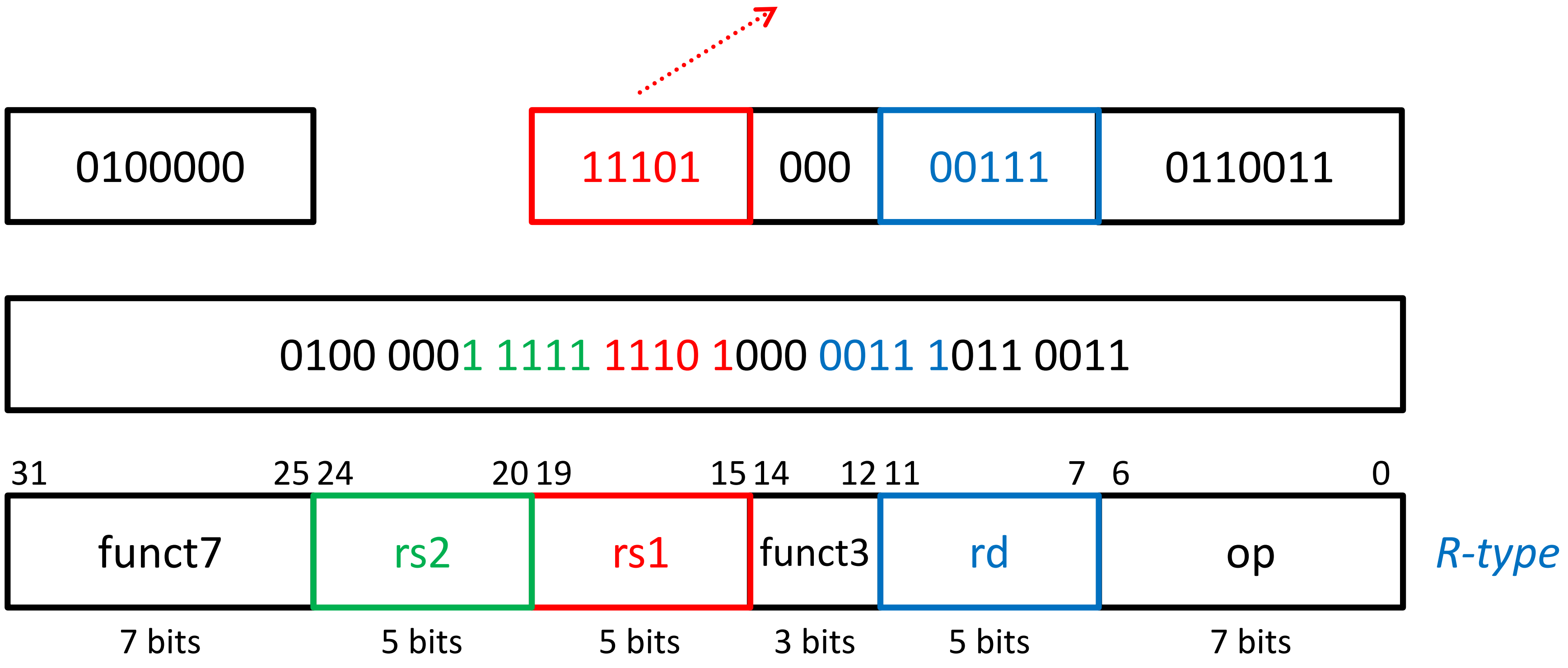


**`0x41fe83b3`**

# From machine code to assembly

## Example (i)

```
sub t2, t4          ≡ sub x7, x29
```



**0x41fe83b3**

# From machine code to assembly

## Example (i)

```
sub t2, t4, t6   ≡ sub x7, x29, x31
```

| | | | | | |
|---|---|---|---|---|---|
| 0100000 | 11111 | 11101 | 000 | 00111 | 0110011 |

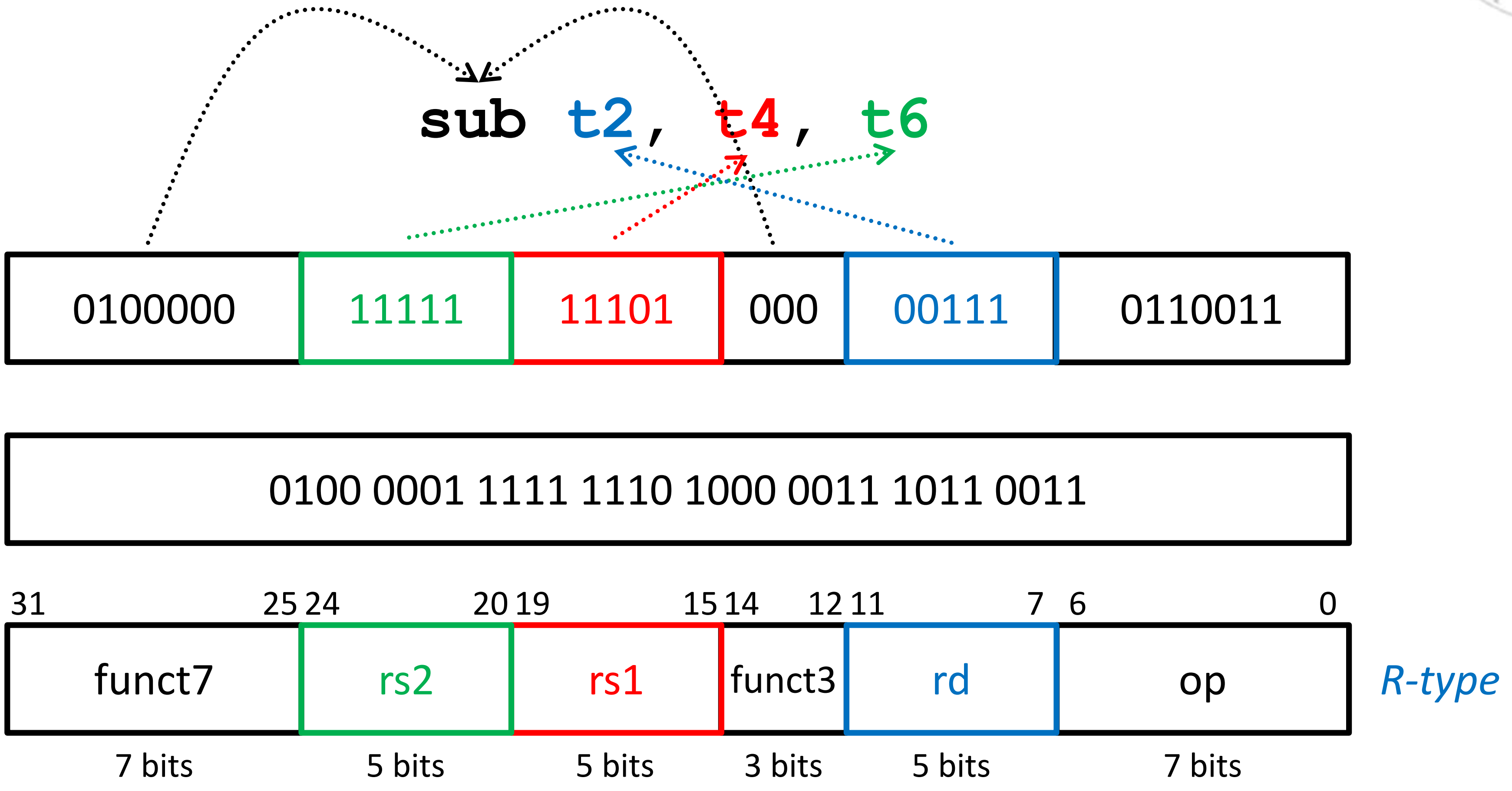
| |
|---|
| 0100 0001 1111 1110 1000 0011 1011 0011 |

| 31 … 25 | 24 … 20 | 19 … 15 | 14 … 12 | 11 … 7 | 6 … 0 | |
|---|---|---|---|---|---|---|
| funct7 | rs2 | rs1 | funct3 | rd | op | *R-type* |
| 7 bits | 5 bits | 5 bits | 3 bits | 5 bits | 7 bits | |

**0x41fe83b3**

# From machine code to assembly

## Example (i)

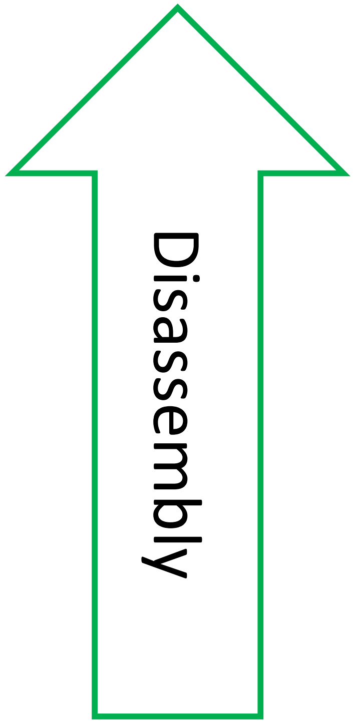


**0x41fe83b3**

# From machine code to assembly

## Example (ii)

```
addi t0, s1, -38
```



```
0xfda48293
```

# From machine code to assembly

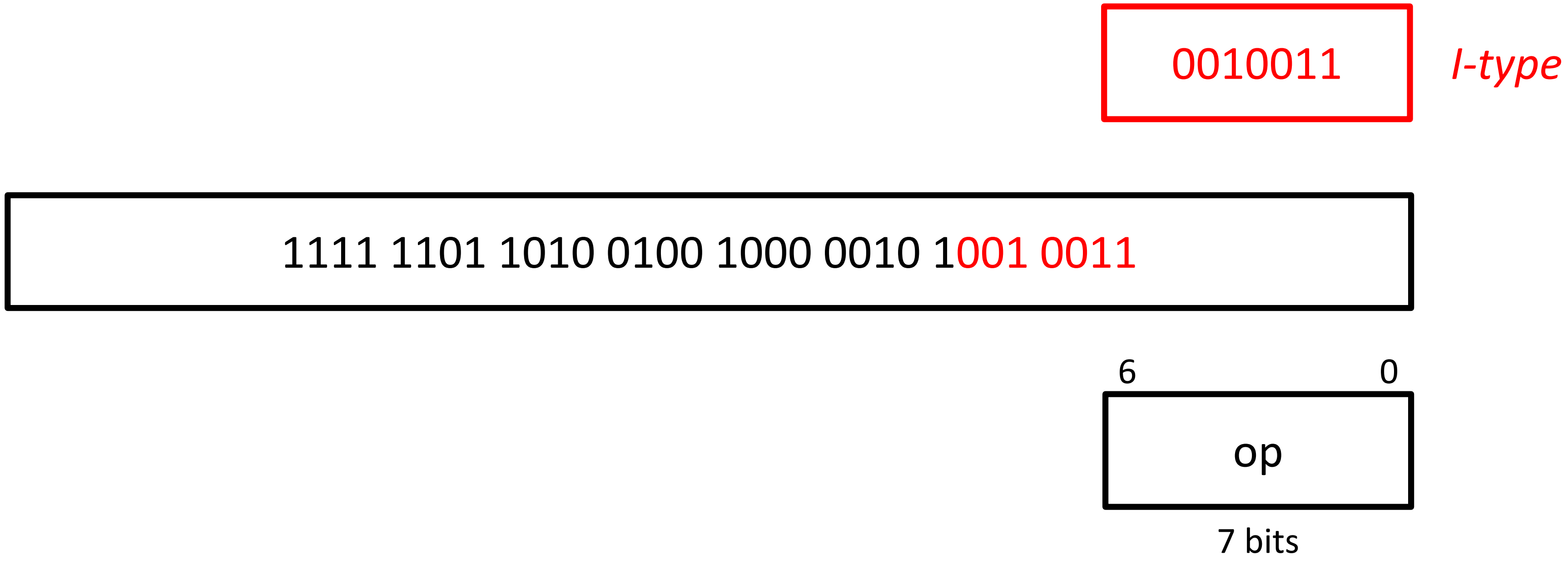
Example (ii)

1111 1101 1010 0100 1000 0010 1001 0011

**0xfda48293**

# From machine code to assembly

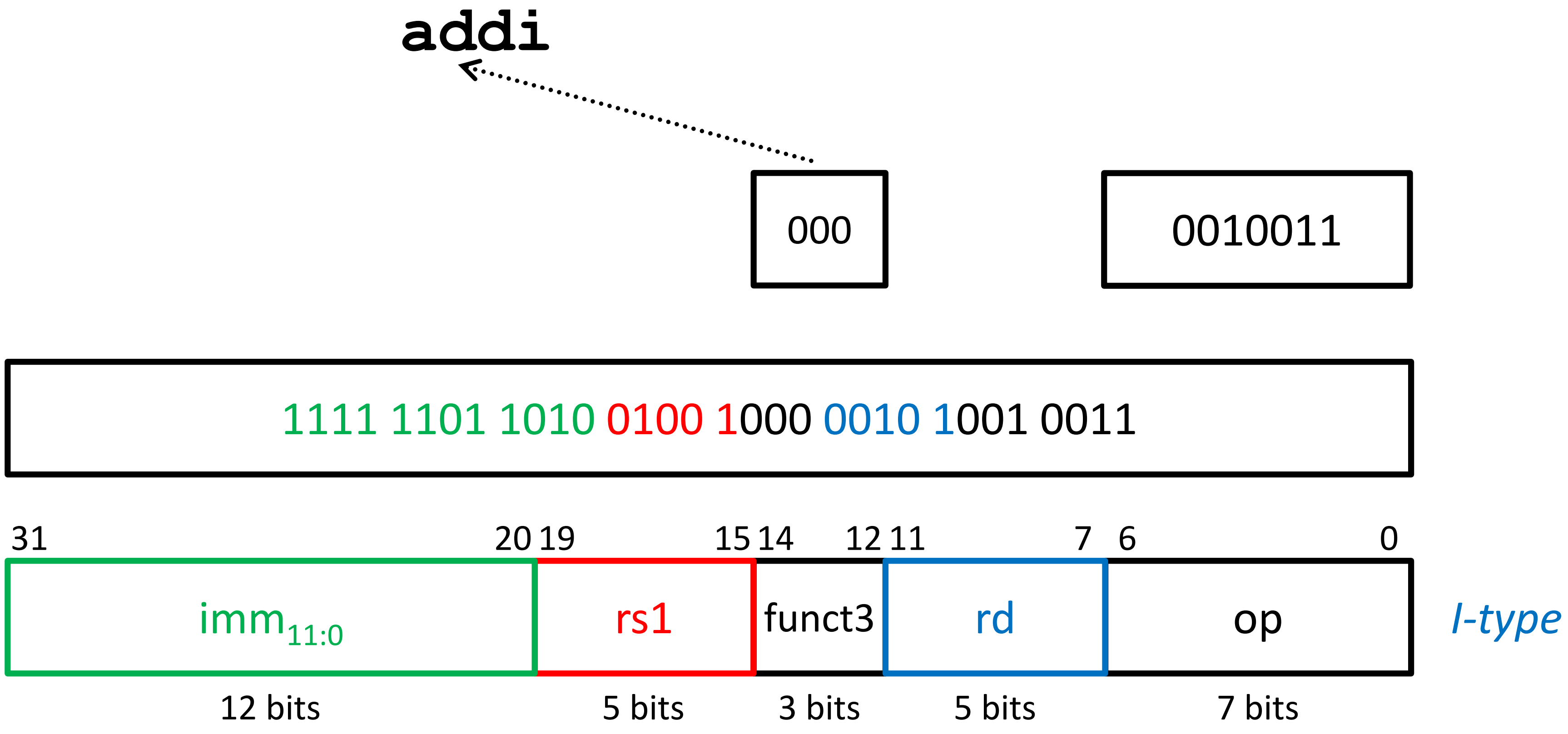
## Example (ii)



**0xfda48293**

## From machine code to assembly

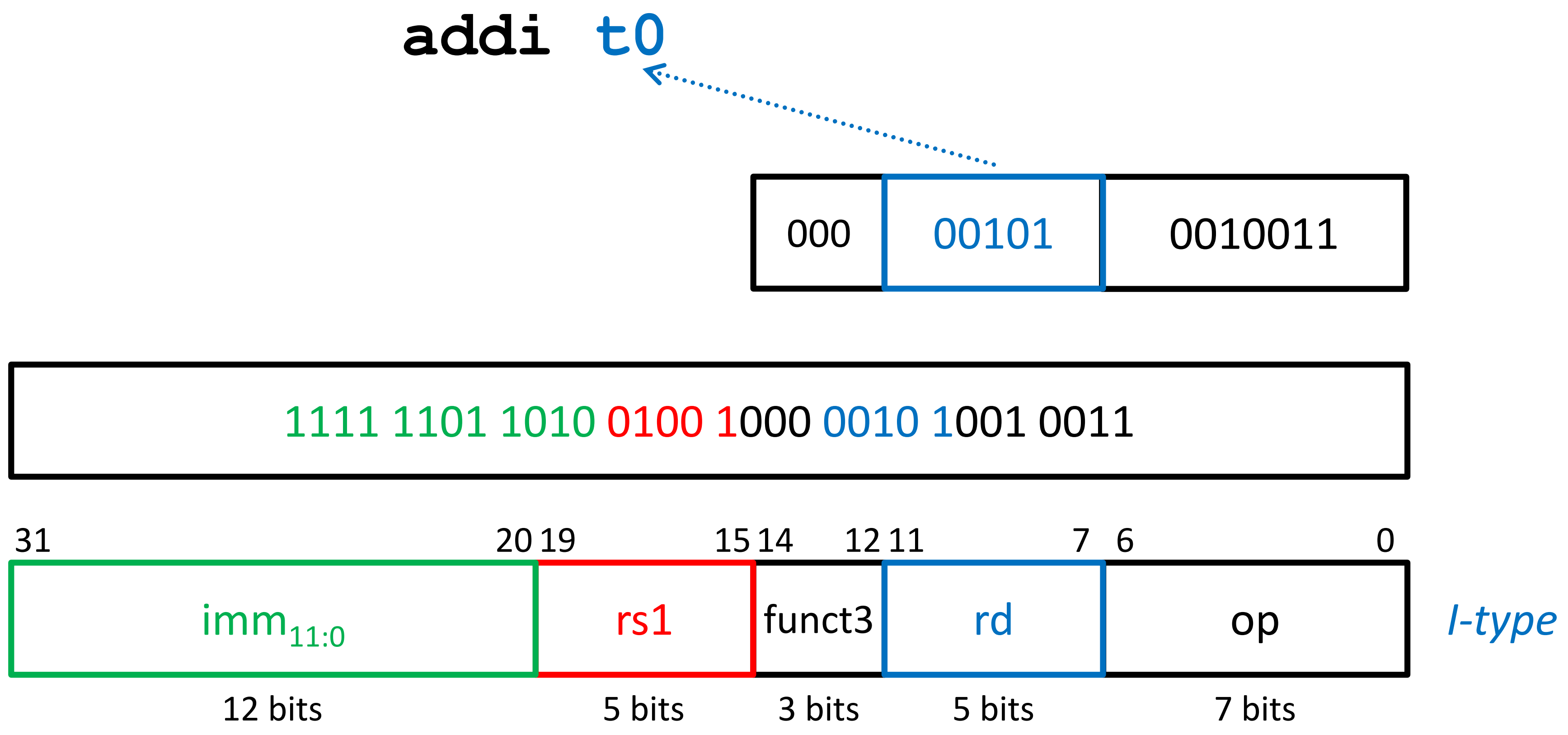
### Example (ii)



**0xfda48293**

## From machine code to assembly

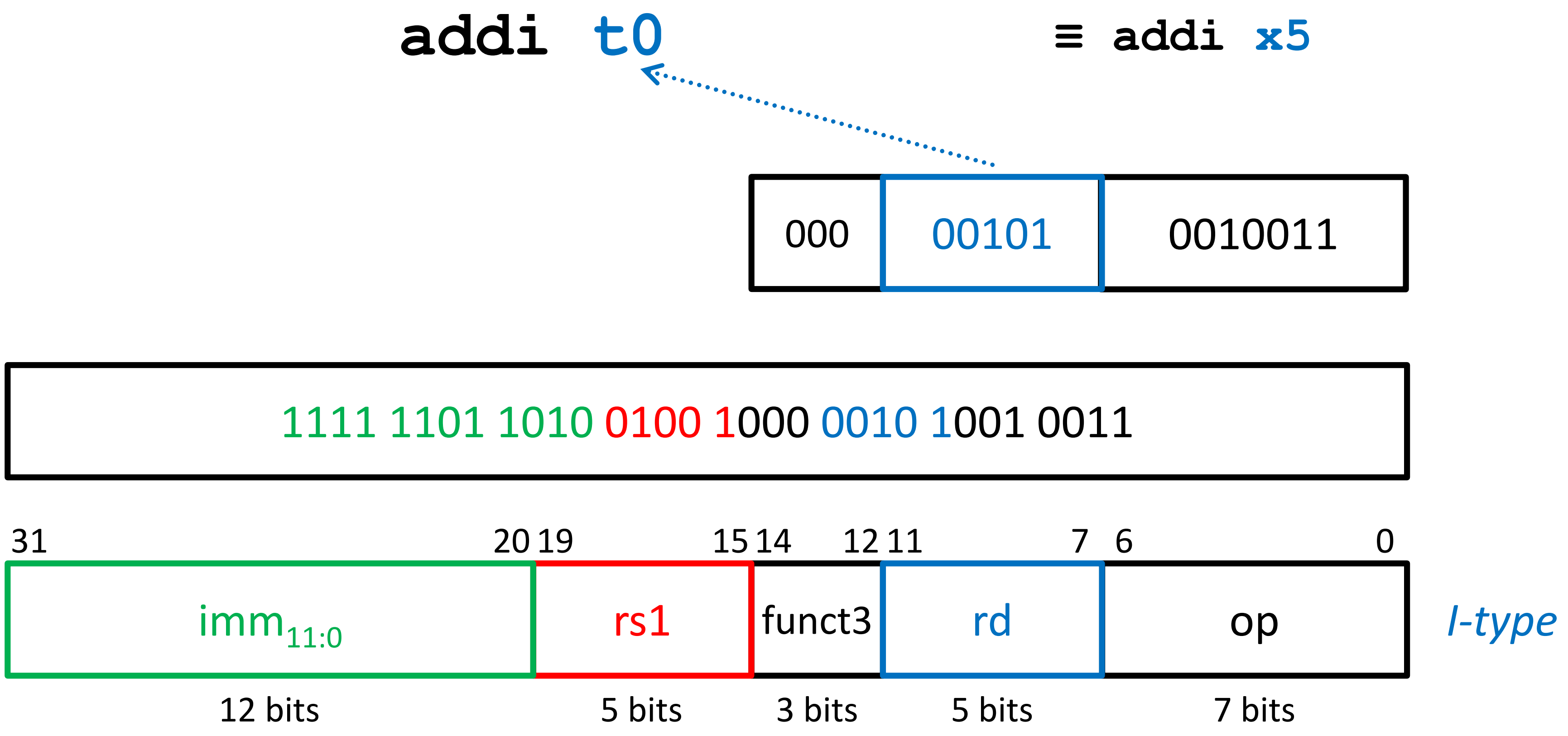
Example (ii)



**0xfda48293**

# From machine code to assembly

Example (ii)

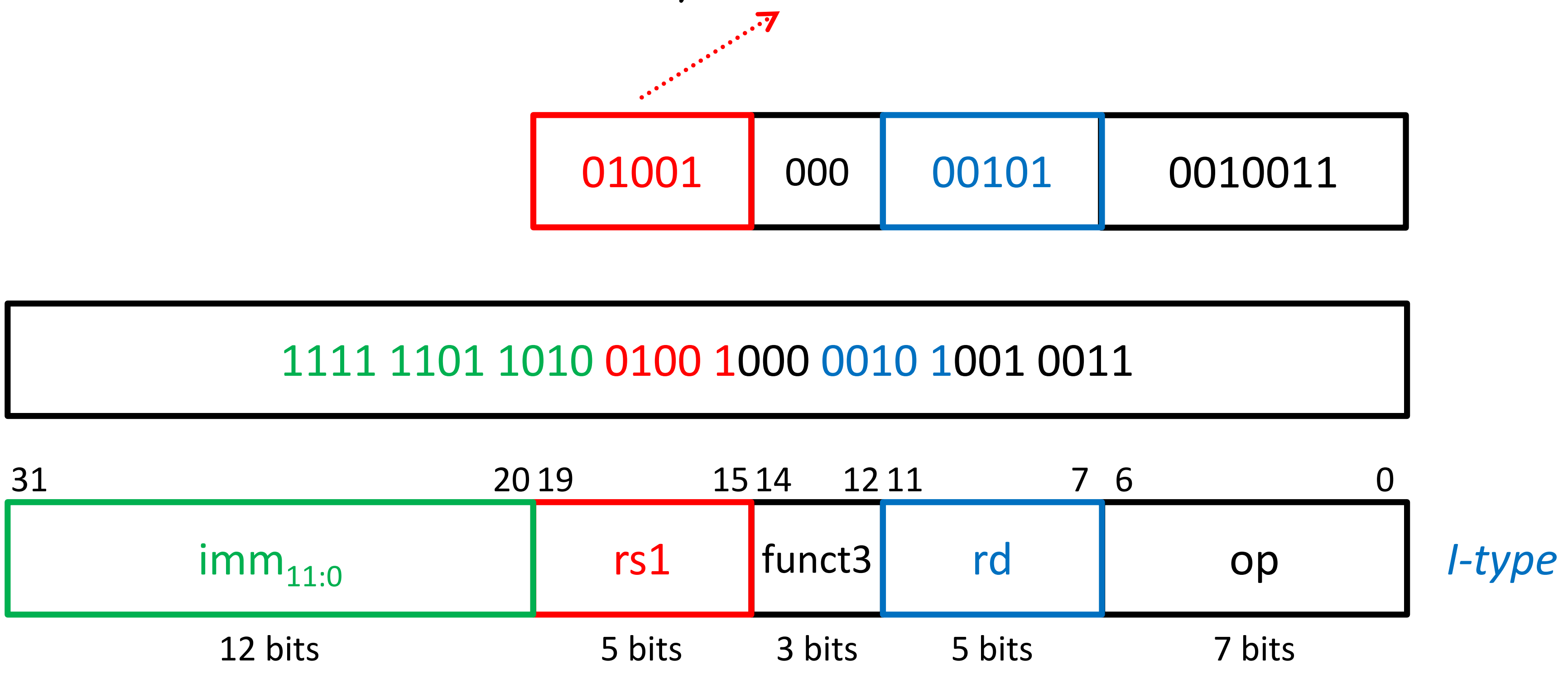


`0xfda48293`

# From machine code to assembly

## Example (ii)

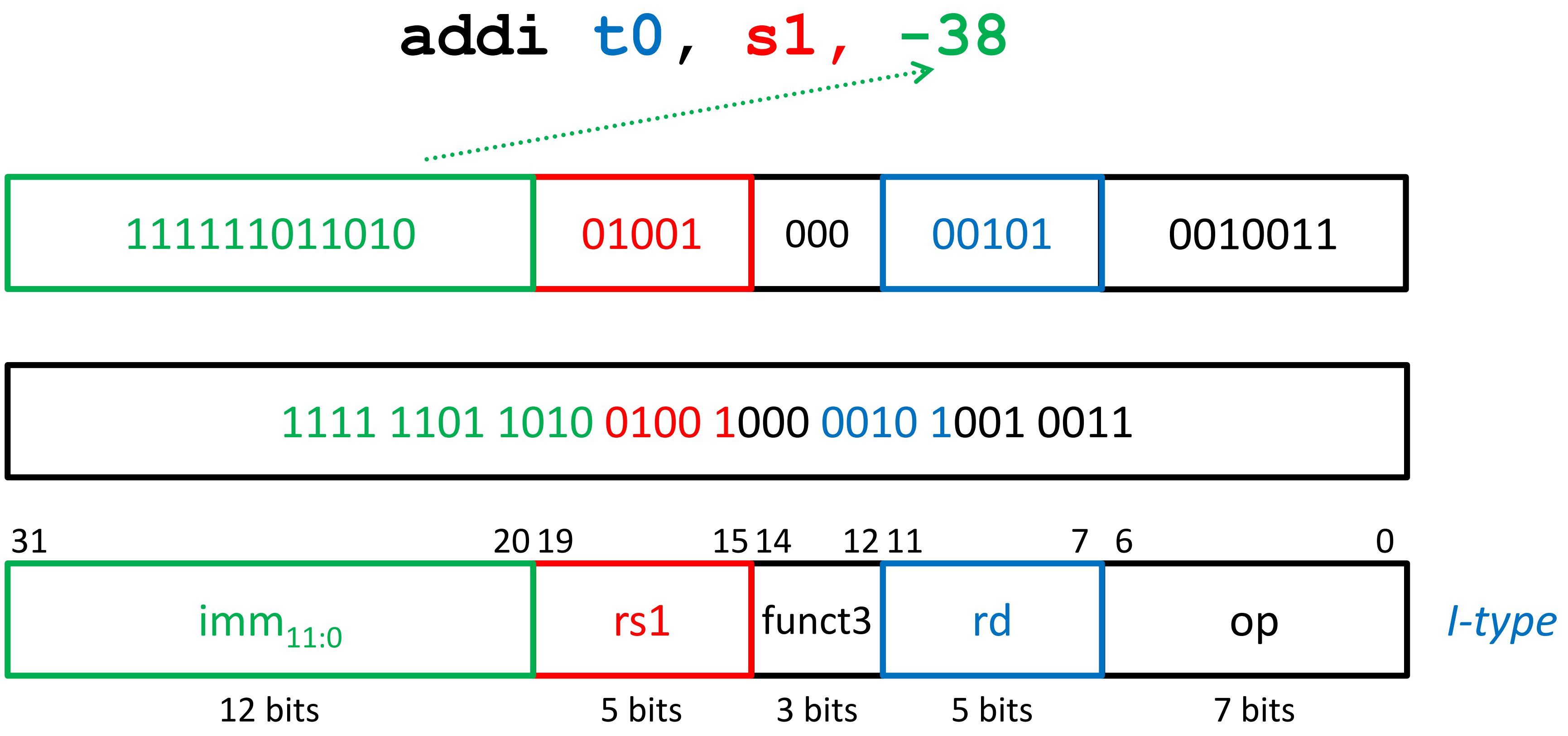
`addi t0, s1` ≡ `addi x5, x9`



`0xfda48293`

# From machine code to assembly

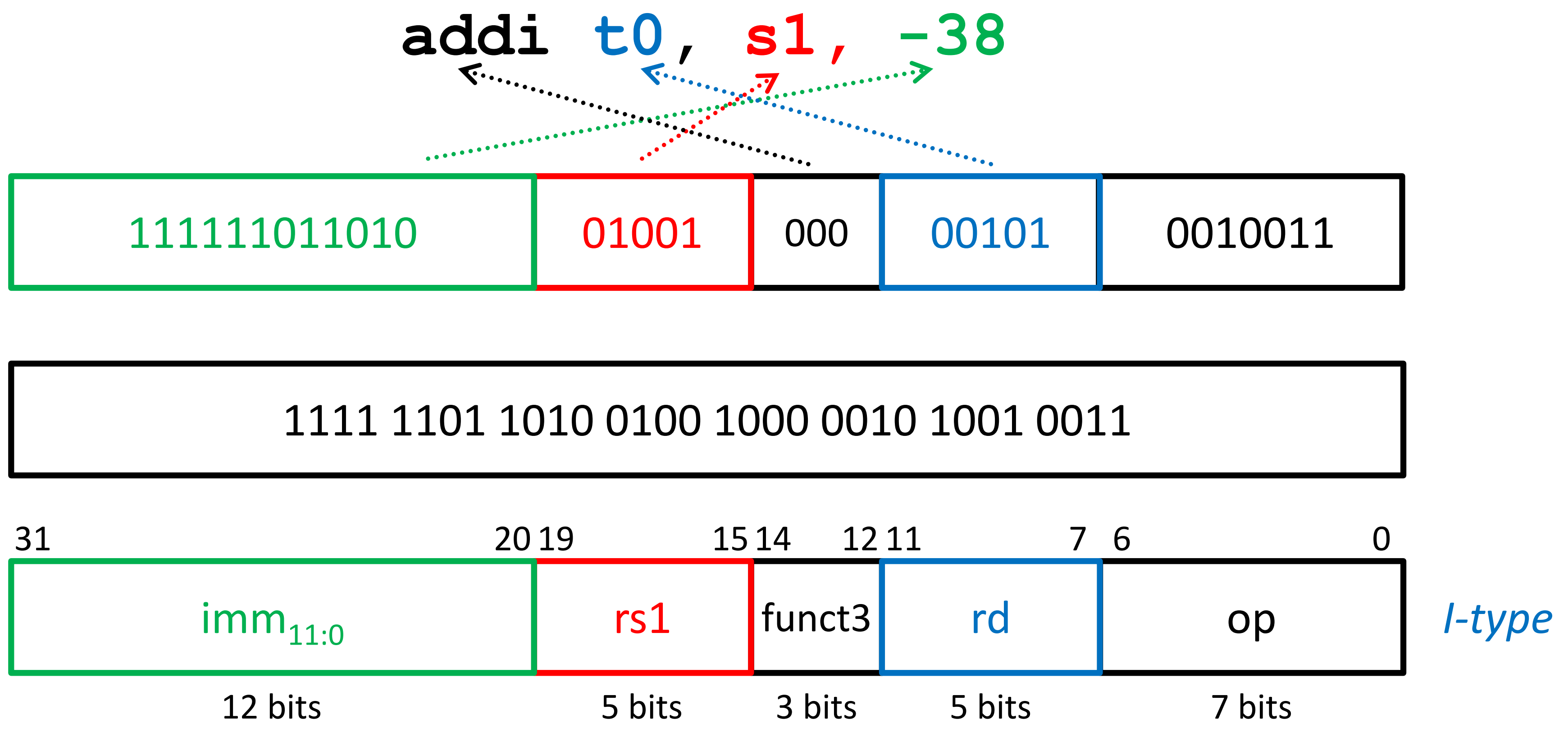
## Example (ii)



```
0xfda48293
```

# From machine code to assembly

## Example (ii)



**`0xfda48293`**

# About *Creative Commons*

- CC license (Creative Commons)
  - This license enables reusers to distribute, remix, adapt, and build upon the material in any medium or format for noncommercial purposes only, and only so long as attribution is given to the creator. If you remix, adapt, or build upon the material, you must license the modified material under identical terms:

Attribution:
Credit must be given to the creator.

Non commercial:
Only noncommercial uses of the work are permitted.

Share alike:
Adaptations must be shared under the same terms.

More information: https://creativecommons.org/licenses/by-nc-sa/4.0/