

Topic V06

Logic Instructions

Reading: (Section 2.6)

Logic Operations in RISC-V

shift left logical immediate

 $s_{11} t_2, s_0, 4$
$$s0 \quad \boxed{0x00000002} = 2_{10}$$
$$2^4 = 16$$
[illegible]

bit 1 is 1

$$2 \times 16 = 32$$
[illegible]
$$2^5 = 32$$

bit 5 is 1

Logic Operations in RISC-V

shift left logical immediate

$$s_{11}i \ t_2, \ s_0, \ 2$$
$$s0 \quad \boxed{0x00000002} = 2_{10}$$

Diagram illustrating a 32-bit register `s0`. The register contains 31 zeros followed by a 1 at the least significant bit (bit 0). A red arrow points to the 1 at bit 0.

[illegible]

$$2^2 = 4$$

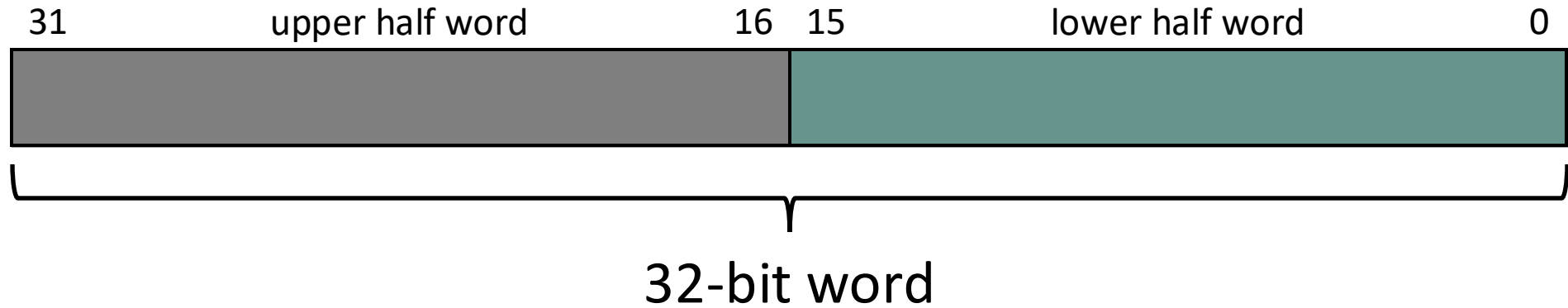
$$2 \times 4 = 8$$

$$2^3 = 8$$

Using Shift Left for Multiplication

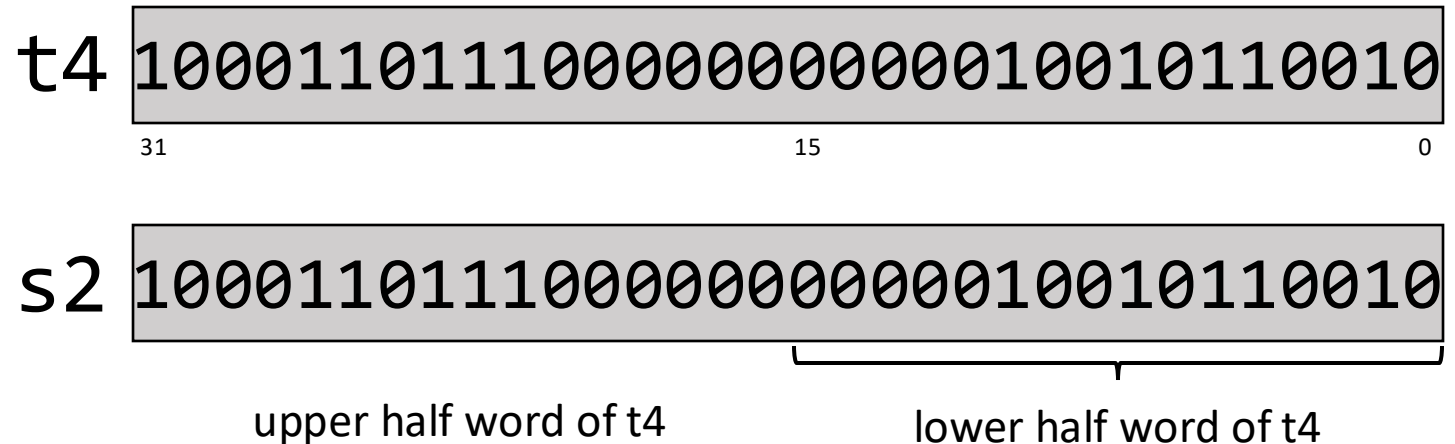
`slli t2, s0, k` is equivalent to: $t2 \leftarrow s0 \times 2^k$

Moving Lower Half Word to Upper Half Word



QUIZ: Write a RISC-V program to move the lower half word of t4 into the upper half word of s2 making the lower half word of s2 equal zero

```
slli s2, t4, 16
```



QUIZ: What would be the value stored in s4 after the execution of the following instruction?

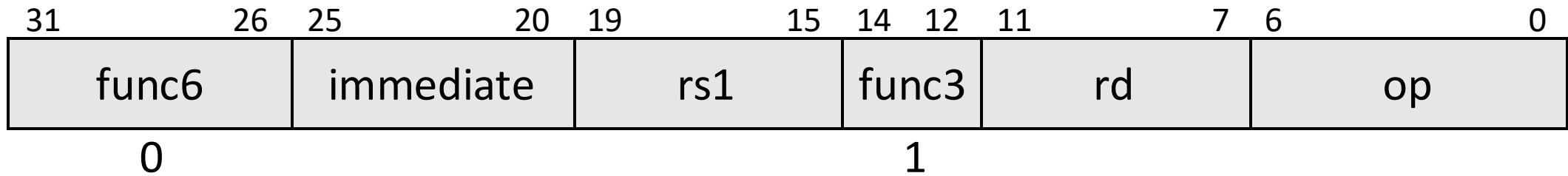
slli s4, t1, 32

operand out of range



Answer: The value would be zero because all 32 bits of t1 would be shifted out to the left and 32 zero bits would be shifted in from the right

Shifts With an Immediate Use a Modified I-Type Format



In RISC-V we can only shift up to 31 bits

⇒ a shift only needs the lower 6 bits of the I-type format's 12-bit immediate

6 upper bits form an additional opcode field: funct6

Attempting to `slli/srli` by more than 31:

⇒ RARS throws an 'operand out of range' error

Shift Left by an Amount in a Register

What happens when we don't know by how many bits we need to shift when we are writing the code?

Example: Write RISC-V assembly to shift the value of **x** (which is in s3) to the left by **k** bits (**k** is in s7)

```
sll s3, s3, s7           # x ← x << k
```


Shifting Right

shift right logical immediate

$$\text{srli } t2, s0, 3$$

`s0` `0x00000040` = 64_{10}

Diagram illustrating a 32-bit register `s0`. The register contains 32 bits, all 0s except for the 16th bit (index 15 from the left, index 1 from the right) which is 1. A red arrow points to this bit.

[illegible]

$$2^3 = 8$$

$$\left\lfloor \frac{64}{8} \right\rfloor = 8$$

$$2^3 = 8$$

A quick way to find
the decimal value of
some binary numbers

In Decimal

$$\begin{aligned} 999 &= 1000 - 1 \\ &= 10^3 - 1 \end{aligned}$$

$$\begin{aligned} 99999 &= 100000 - 1 \\ &= 10^5 - 1 \end{aligned}$$

In Decimal

digit 6 digit 4 digit 1 digit 0

↓ ↓ ↓ ↙

$$9,990,099 = 10,000,000 - 10,000 + 100 - 1$$
$$= 10^7 - 10^4 + 10^2 - 10^0$$

In Binary

What is the decimal value of the following binary number?

bit 14



$$x = 0000\ 0111\ 1111\ 1111\ 1111$$

$$x = 2^{15} - 1 = 2^{10} \times 2^5 - 1 = 1024 \times 32 - 1$$

$$\begin{array}{r} 0000\ 1000\ 0000\ 0000\ 0000 = 2^{15} \\ - 0000\ 0000\ 0000\ 0000\ 0001 = 1 \\ \hline 0000\ 0111\ 1111\ 1111\ 1111 \end{array}$$

In Binary

What is the decimal value of the following binary number?




Diagram showing bit positions 16, 10, 6, and 2 with arrows pointing to the corresponding bits in the binary number 0001 1111 1100 0111 1100.

$$\begin{aligned}x &= 0001\ 1111\ 1100\ 0111\ 1100 \\x &= 2^{17} - 2^{10} + 2^7 - 2^2 \\&= (2^7 - 1) \times 2^{10} + (2^5 - 1) \times 2^2 \\&= (128 - 1) \times 1024 + (32 - 1) \times 4 \\&= 127 \times 1024 + 31 \times 4 \\&= 130,048 + 124 \\&= 130,172\end{aligned}$$

Shifting Right

shift right logical immediate

$$\text{srli } t2, s0, 2$$
$$s0 \quad \boxed{0xFFFFFFFF0} = -16_{10}$$

$$2^{30} - 2^2$$

$$= 2^{10} \times 2^{10} \times 2^{10} - 4$$

$$= 1024 \times 1024 \times 1024 - 4$$

= 1,073,741,820

t2 111111111111111111111111111111110000

bit 30 bit 2

Shifting Right

shift right arithmetic immediate

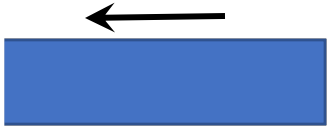
How do I perform this computation?

$$\text{srai } t_2, s_0, 2$$
$$s0 \quad \boxed{0xFFFFFFFF0} = -16_{10}$$

$$\left\lfloor \frac{-16}{4} \right\rfloor = -4$$

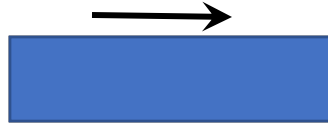
$$t_2 \text{ 111111111111111111111111110000} = -4_{10}$$

slli t2, s0, k

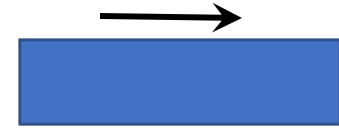


$t2 \leftarrow s0 \times 2^k$

srli t2, s0, k



srai t2, s0, k



$t2 \leftarrow \left\lfloor \frac{s0}{2^k} \right\rfloor$

sll t2, s0, t4

srli t2, s0, t4

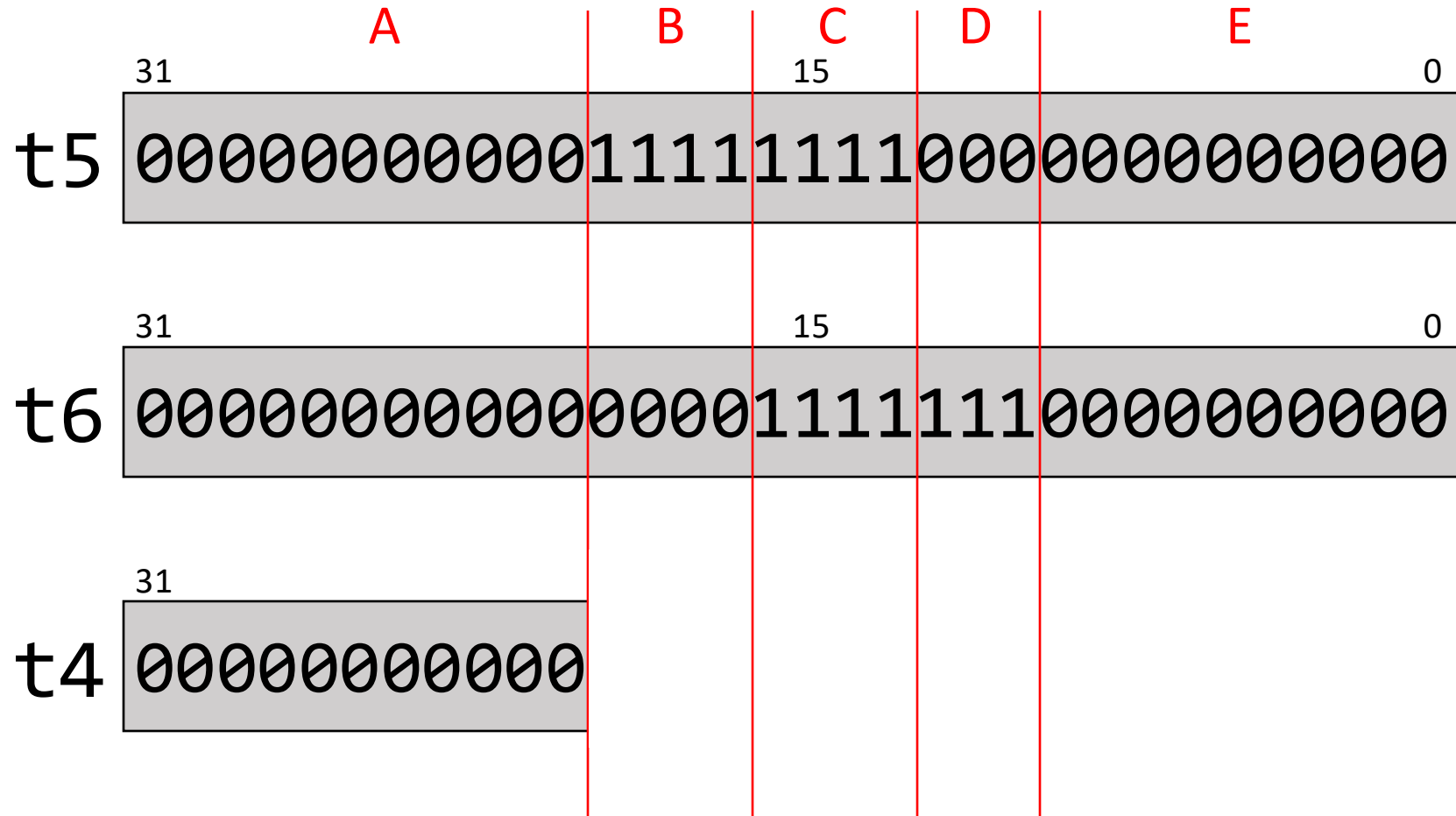
sra t2, s0, t4

Recap of shift
operations

Bitwise Logical Operations

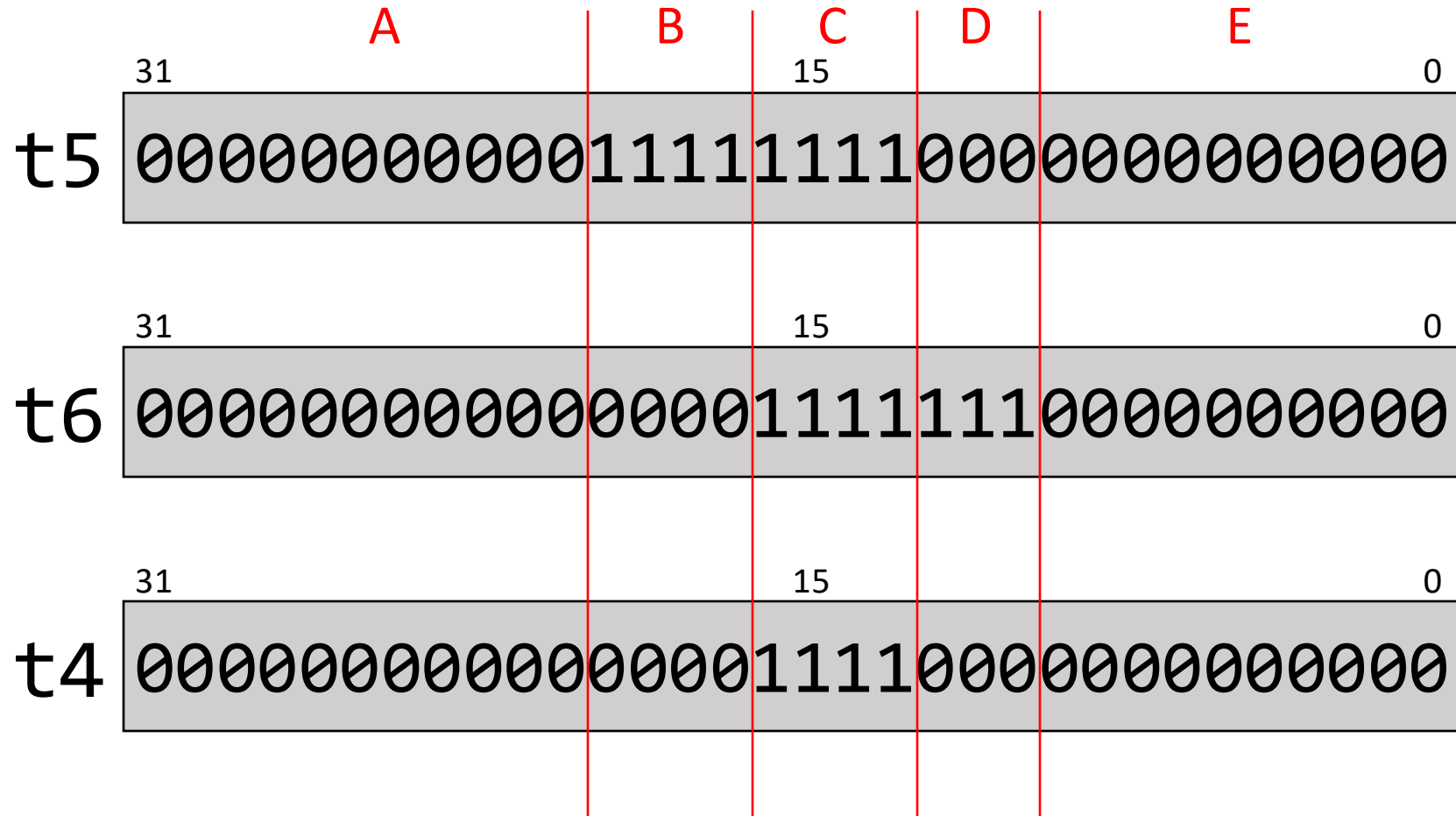
Bitwise Logical Operations

and t4, t5, t6



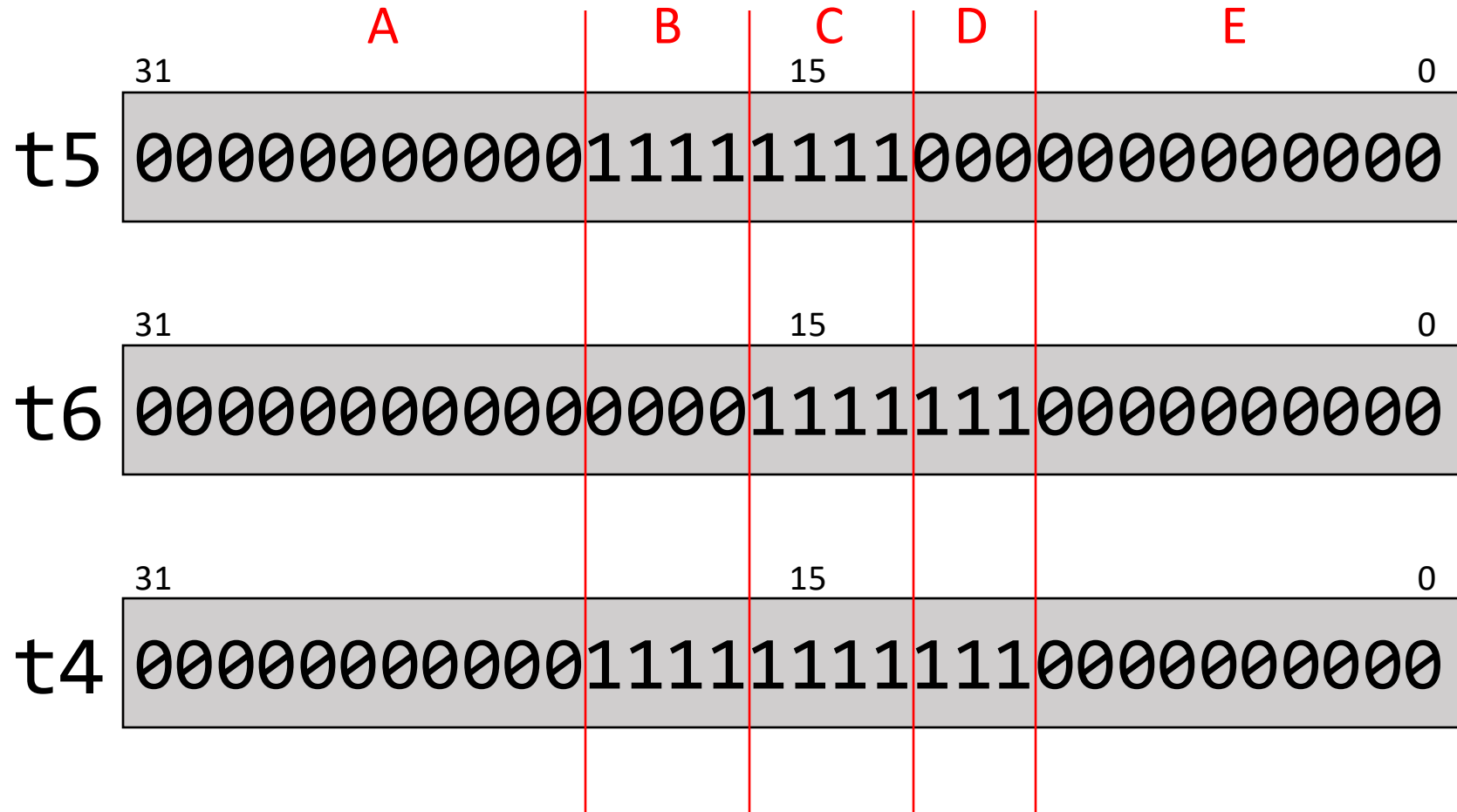
Bitwise Logical Operations

and t4, t5, t6



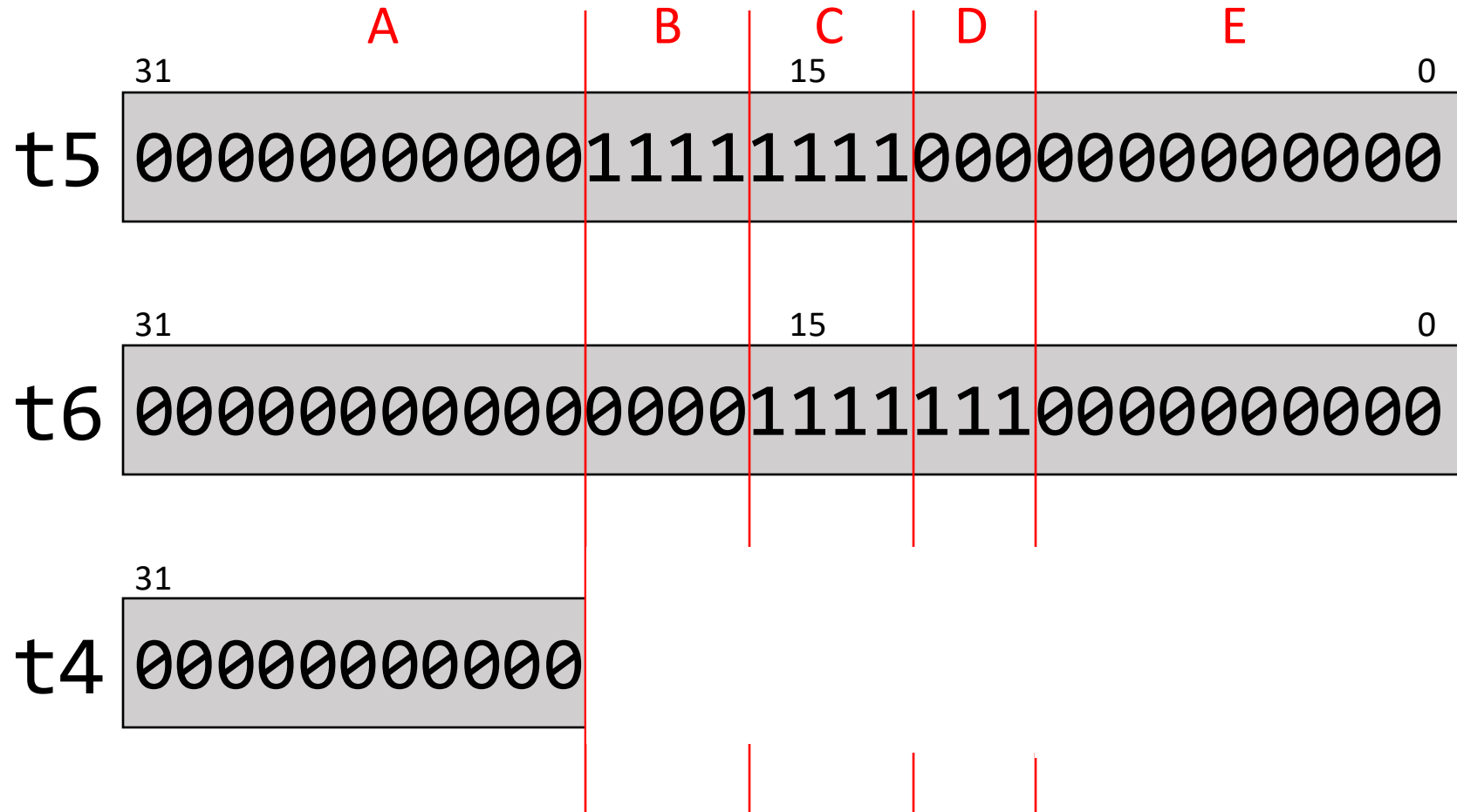
Bitwise Logical Operations

or t4, t5, t6



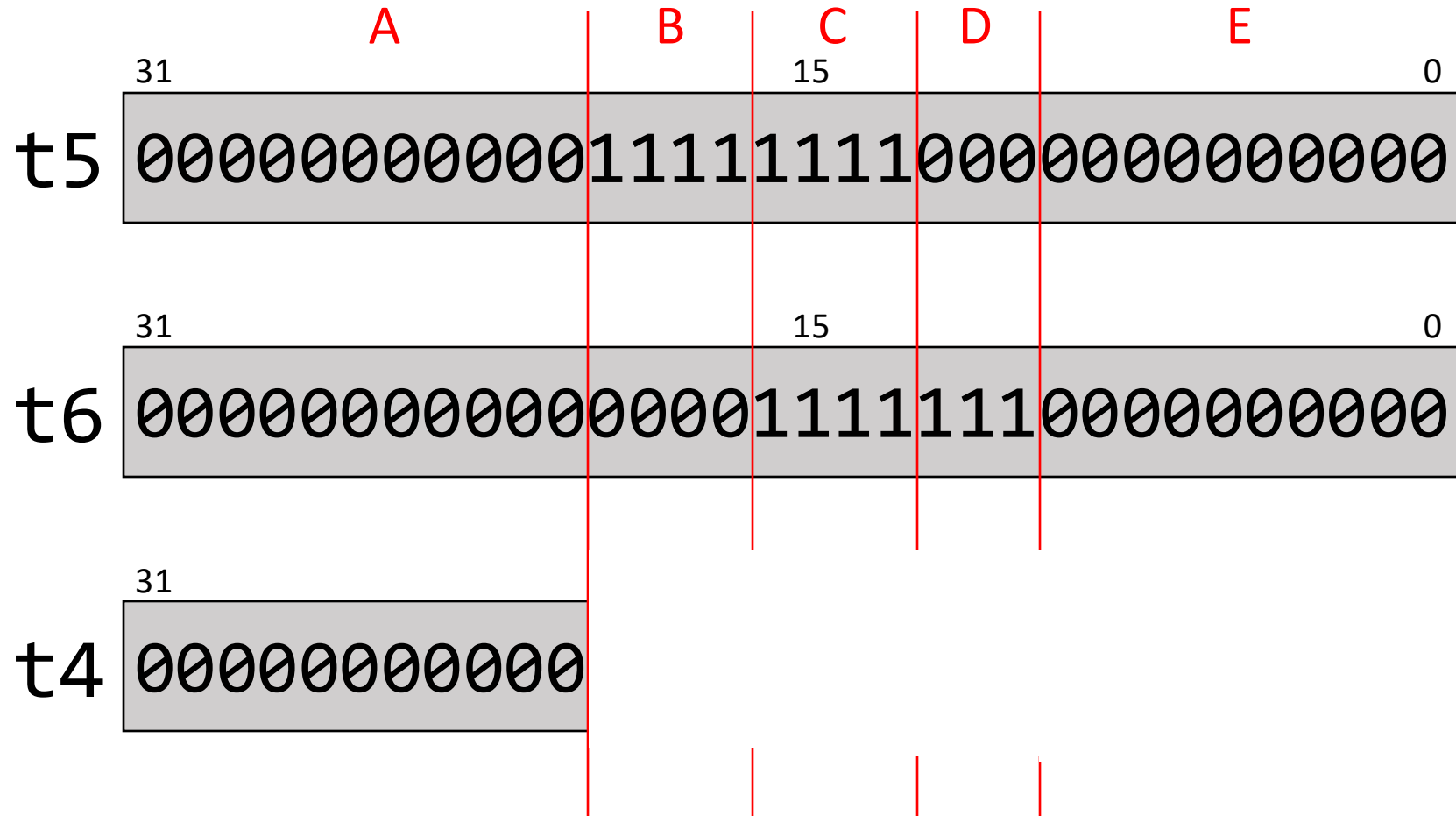
Bitwise Logical Operations

or t4, t5, t6



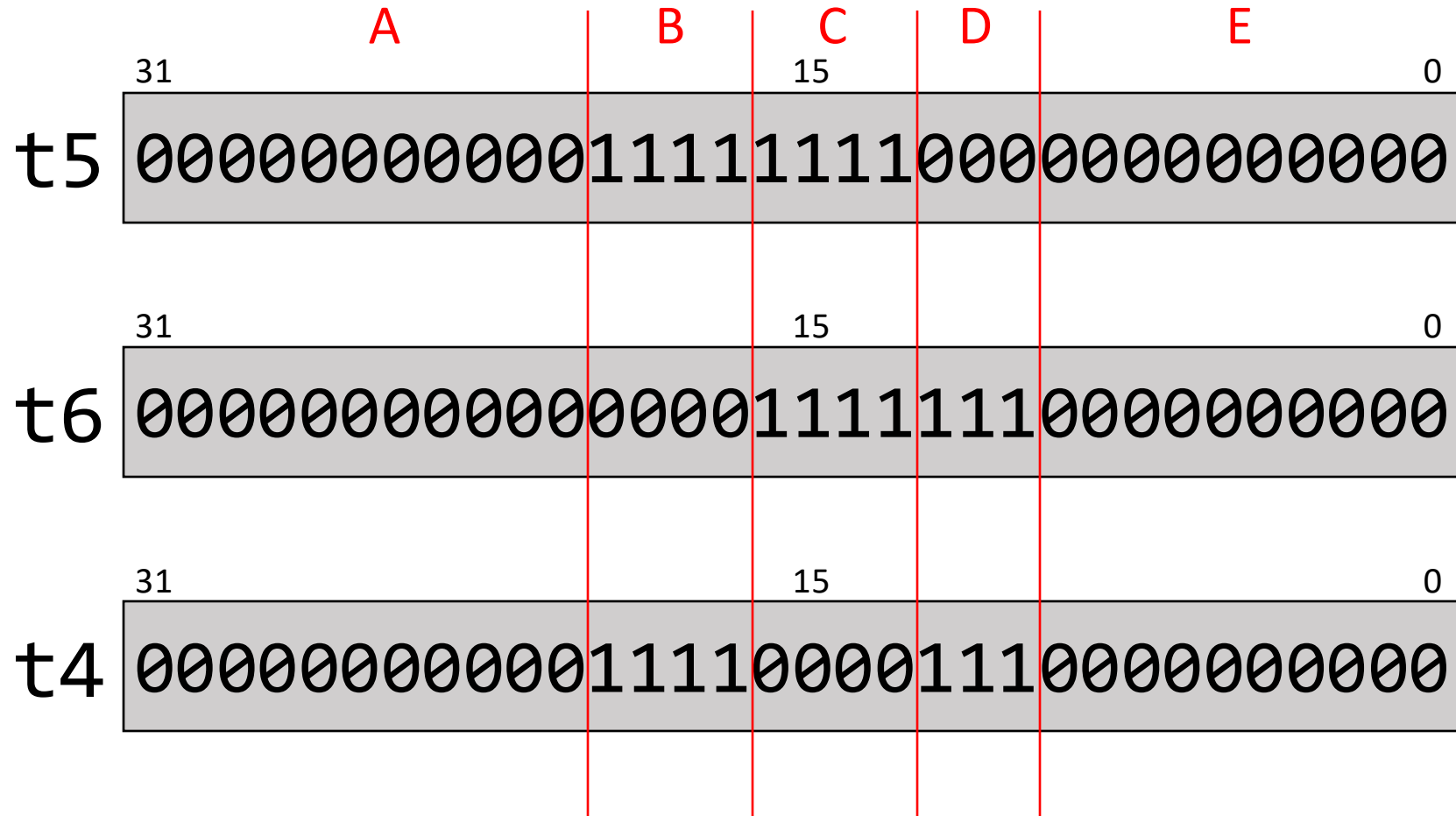
Bitwise Logical Operations

xor t4, t5, t6



Bitwise Logical Operations

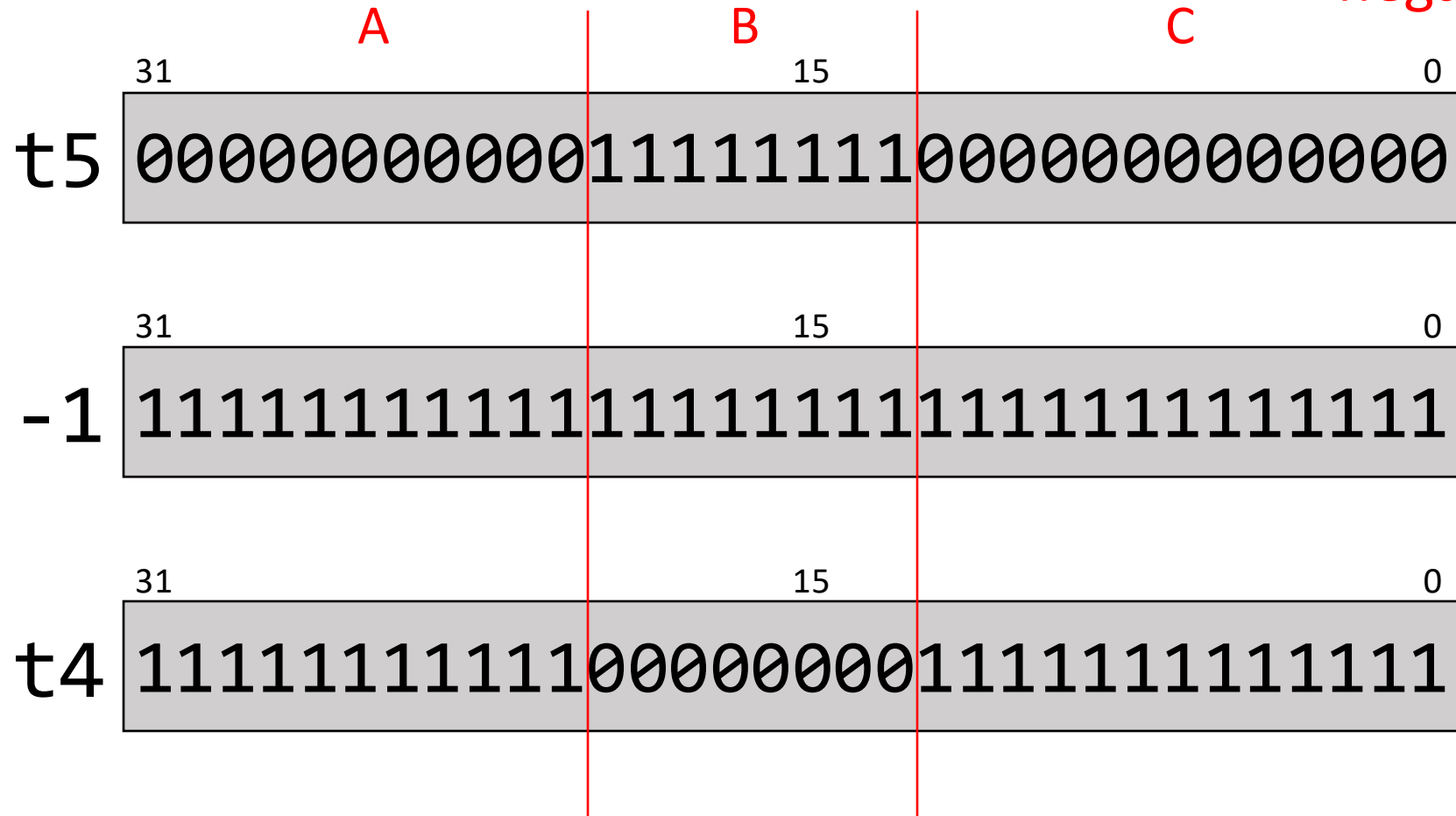
xor t4, t5, t6



Bitwise Logical Operations

`xori t4, t5, -1`

used for logical
negation

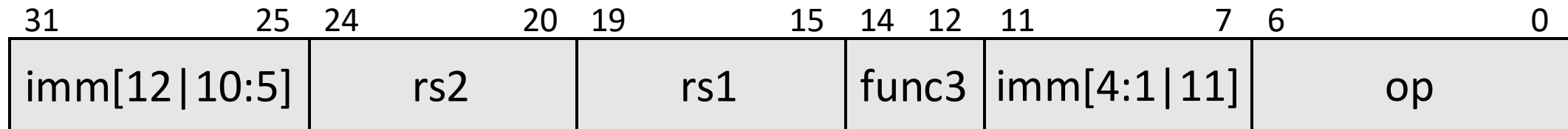


Warm Up

An unconditional branch can be created using a **beq** instruction with the same register for both source operands. Examples:

```
beq t3, t3, LeftSide
```

Binary format of a branch instruction:



Assume that **t0** has the binary representation of a **beq** instruction.

Write a sequence of RISC-V instructions that writes in **t1**:

- zero: if that **beq** is an unconditional branch
- non-zero: if that **beq** is a conditional branch

slli t2, t0, 7

t2 =

imm[12 10:5]	rs2	rs1	func3	imm[4:1 11]	op
----------------	-----	-----	-------	---------------	----

srli t3, t2, 27

t3 =

rs2	rs1	func3	imm[4:1 11]	op	000 0000
-----	-----	-------	---------------	----	----------

slli t4, t0, 12

t4 =

imm[12 10:5]	rs2	rs1	func3	imm[4:1 11]	op
----------------	-----	-----	-------	---------------	----

srli t5, t4, 27

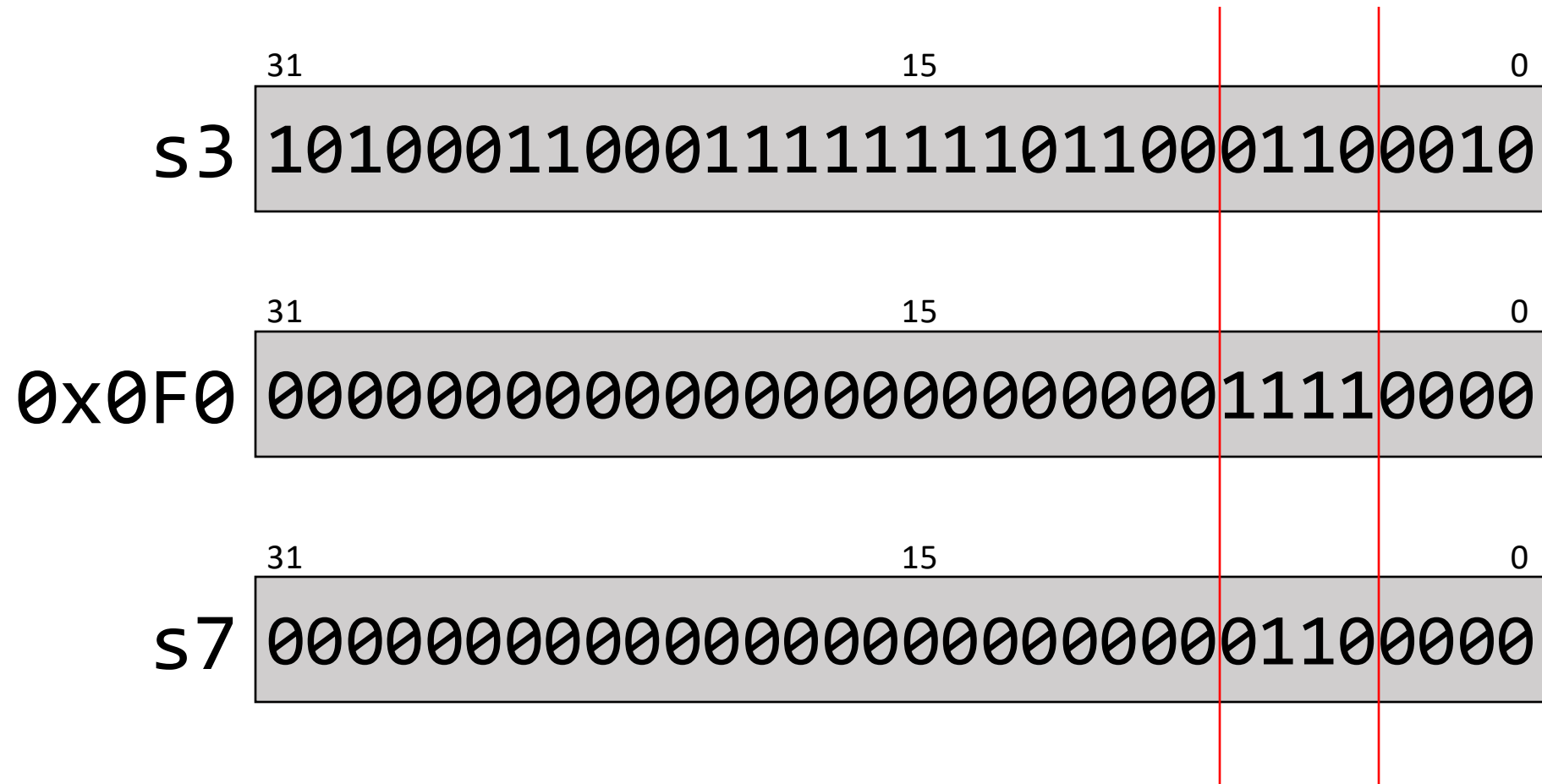
t5 =

rs1	func3	imm[4:1 11]	op	0000 0000 0000
-----	-------	---------------	----	----------------

xor t1, t4, t5

Bitwise Logical Operations With Immediate Operand

```
andi s7, s3, 0x0F0
```



A RARS Issue

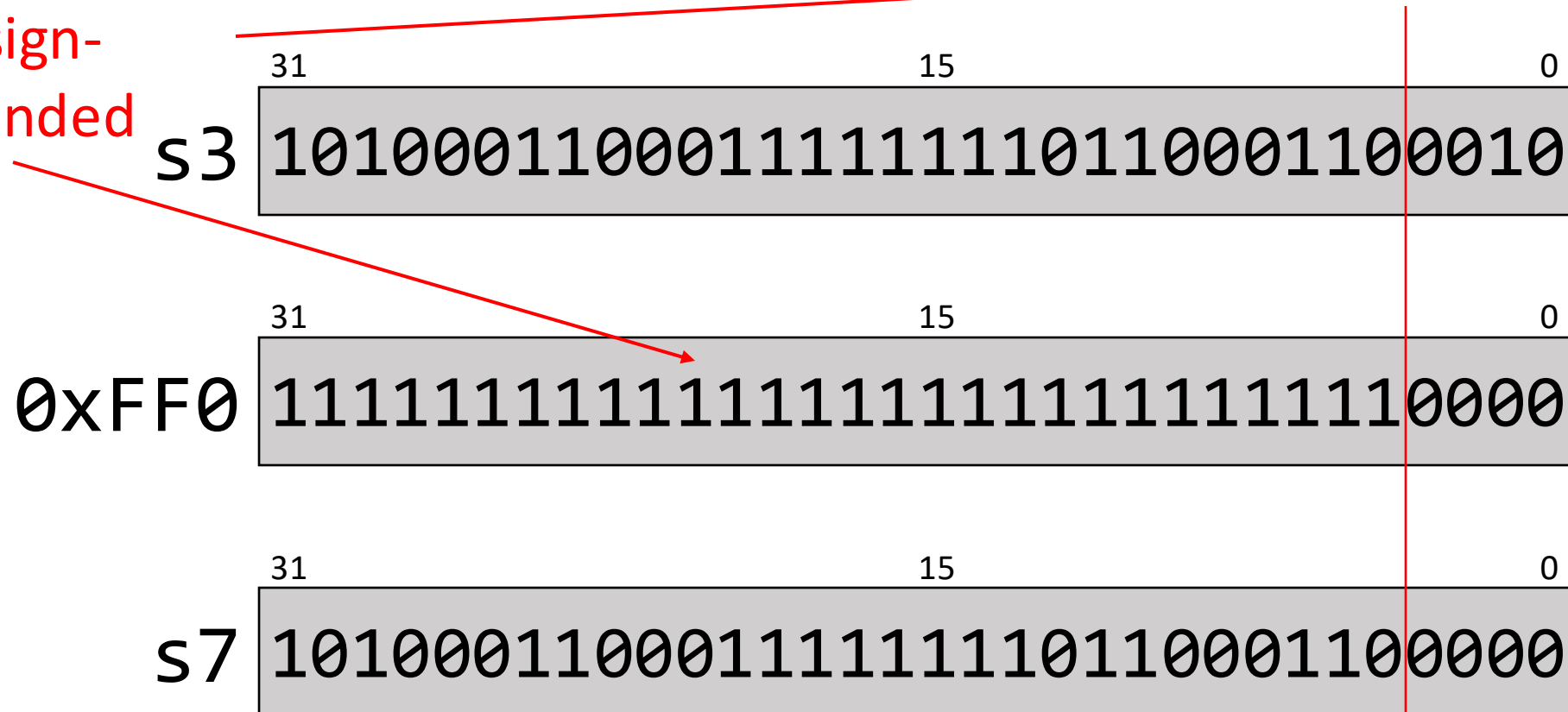
RARS, incorrectly, zero extends constants
expressed in hexadecimal.

Bitwise Logical Operations With Immediate Operand

12-bit constant
is sign-
extended

`andi s7, s3, 0xFF0`

results in an
error in RARS



A work around

Express the 12-bit constant in decimal.

`0xFFFFFFFF0 = -16`

`andi s7, s3, -16`

The Load Immediate Instruction (li)

li is a pseudo instruction implemented by an addi instruction.

The immediate value is sign extended to 32 bits.

```
li t0, 0x0FF # t0 ← 0x000000FF
```

```
li t0, 0xFF0 # t0 ← 0xFFFFFFFF0
```


RARS Bitwise Logic With Immediate

```
li    t0, 0xFF    # t0 ← 0x000000FF
```

```
slli t0, t0, 4    # t0 ← 0x0000FF0
```

```
and s7, s3, t0
```

	31	15	0
s3	1010001100011111110110001100010		

	31	15	0
t0	0000000000000000000011111110000		

	31	15	0
s7	00000000000000000000110001100000		