RISC-V
Logical Operations

Sparsh Mittal

Logical Operations (for bitwise manipulation)

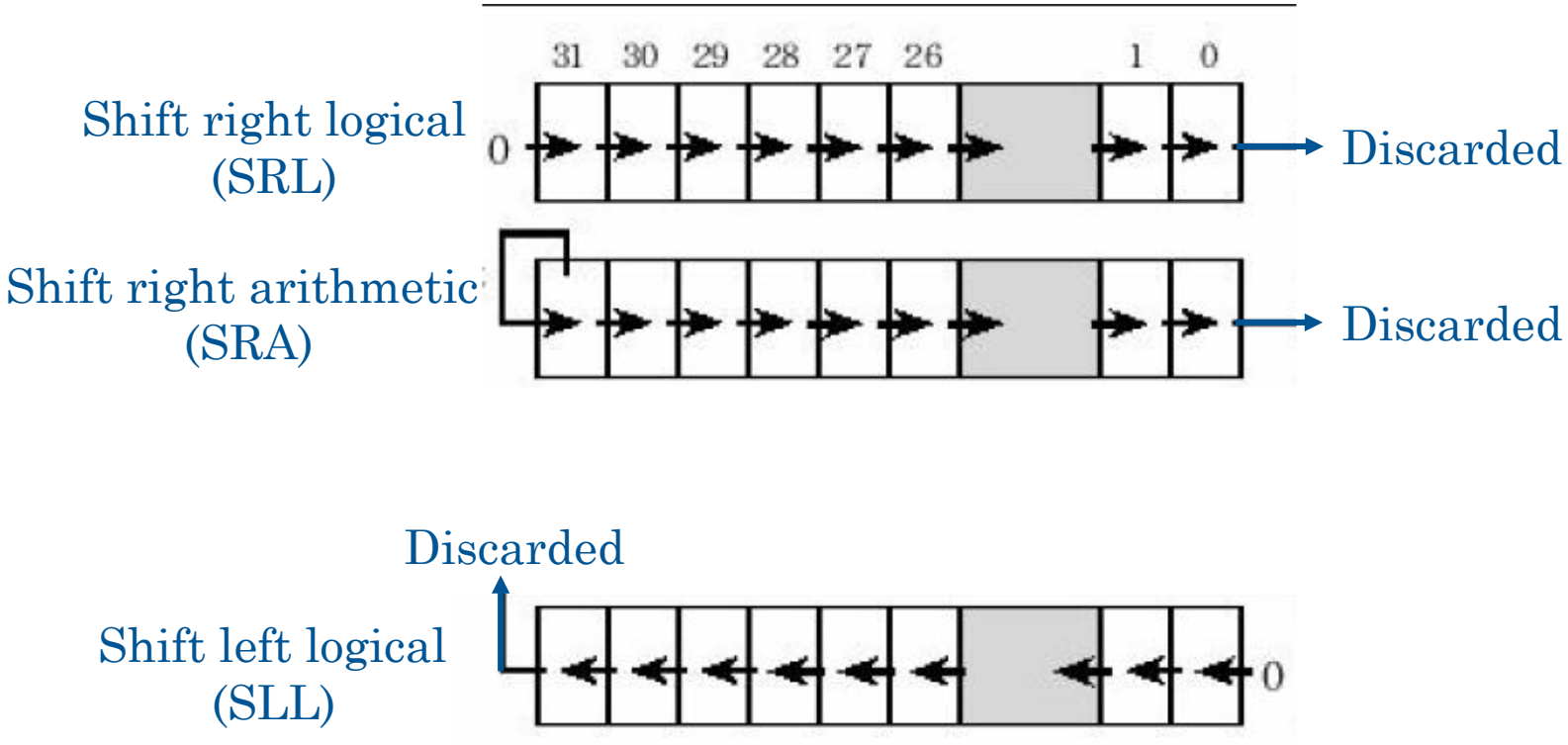
Operation	C	Java	RISC-V
Shift left	<<	<<	sll, slli
Shift right	>>	>>>	srl, srli, sra, srai
Bit-by-bit AND	&	&	and, andi
Bit-by-bit OR	|	|	or, ori
Bit-by-bit XOR	^	^	xor, xori
Bit-by-bit NOT	~	~	not

- srl = shift-right logical. “i” refers to immediate

Arithmetic and logical shifts

Here, we show 32b registers. Same applies to a 64b register also

n = #shifted bits



Equivalent to

Unsigned divide by 2^n

Signed divide by 2^n

Multiply by 2^n (both signed and unsigned)

https://stackoverflow.com/questions/7622/are-the-shift-operators-arithmetic-or-logical-in-c

Examples

- Q1. Shift-left register x10 by the number stored in x11. Store result in x9.
- sll x9, x10, x11
- So, if x11 stores 5, then x10 shifted by 5 will be stored in x9.

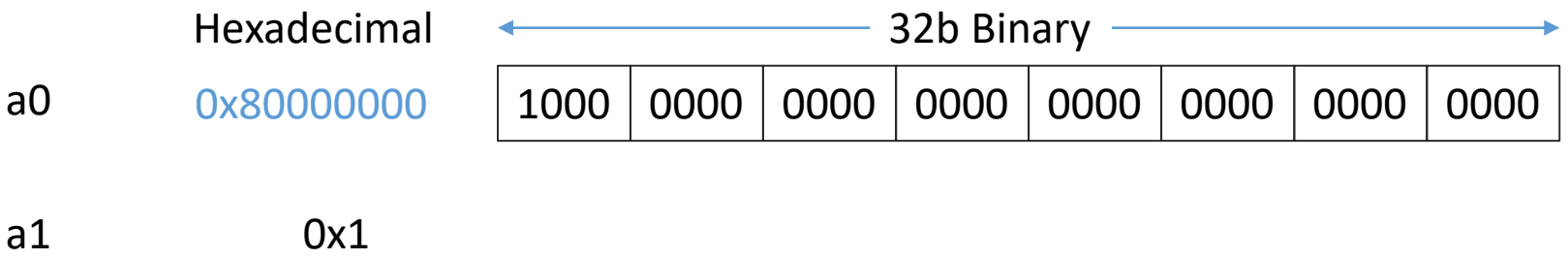
- Q2. Shift-left register x10 by 3 times. Store result in x9.
- slli x9, x10, 3

Shift Instructions (SLL, SRL, SRA)

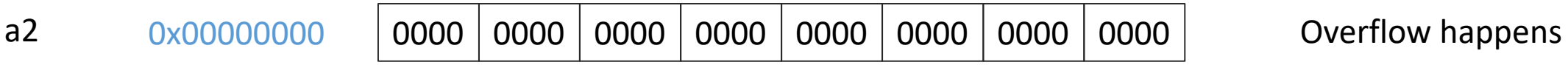
```
li a0, 0x80000000          # a0 ← 0x80000000
li a1, 1                   # a1 ← 1 (We want to shift by 1 position)
sll a2, a0, a1
# a2 ← 0 (The MSB bit is discarded, so the number becomes zero)

srl a3, a0, a1             # a3 ← 0x40000000. New bit filled =0
sra a4, a0, a1             # a4 ← 0xC0000000. New bit filled =1
```

Visual explanation is shown on the next slide.

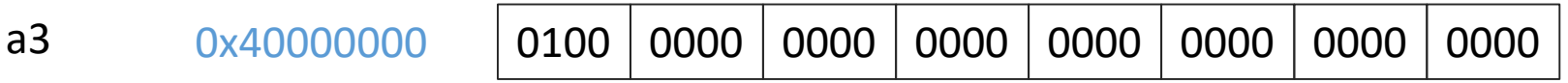
Visual explanation



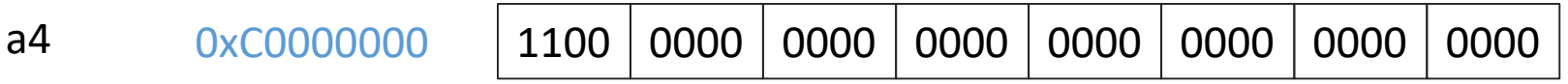
On doing, "**sll a2, a0, a1**", we are shifting a0 left by 1. So now, the value of a2 is:



On doing, "**srl a3, a0, a1**", we are shifting a0 right by 1. It's logical shift, so new bit filled will be 0. a3 will be



On doing, "**sra a4, a0, a1**", we are shifting a0 right by 1. It's arithmetic shift, so new bit filled will be the same as the sign bit, which is 1. a4 will be



Corner case

- **NOTE: div** always rounds towards zero, whereas **srai** rounds towards negative infinity. Hence, srai is not exactly equivalent to division.
- Srai fails for Odd Negative Numbers.

There is no SLA operation. Why?
Answer: Two's complement ensures that SLA and SLL lead to same effect.

- Right shift requires both logical and arithmetic modes
  - Assuming 4 bits
  - $(4_{10})>>1 = (0100_2)>>1 = 0010_2 = 2_{10}$ Correct!
  - $(-4_{10})>>_{logical}1 = (1100_2)>>_{logical}1 = 0110_2 = 6_{10}$ For signed values, Wrong!
  - $(-4_{10})>>_{arithmetic}1 = (1100_2)>>_{arithmetic}1 = 1110_2 = -2_{10}$ Correct!
  - Arithmetic shift replicates sign bits at MSB
- Left shift is the same for logical and arithmetic
  - Assuming 4 bits
  - $(2_{10})<<1 = (0010_2)<<1 = 0100_2 = 4_{10}$ Correct!
  - $(-2_{10})<<_{logical}1 = (1110_2)<<_{logical}1 = 1100_2 = -4_{10}$ Correct!

Understanding left-shift

- Consider a 4-bit number -6, which is 1010.
- On left-shifting, we should get -12. But we get 0100, which is just 4. Is this an error?
- **Explanation:** With 4 bits, we can represent only -8 to 7. The number -12 is outside the range, so overflow happens.
- We have to use 5-bit number system. Here, -6 is 11010.
- On left-shifting, we get 10100, which is -12, the correct answer.

Solved example

	C code	RISC-V code

Left-shift by 2
=
Multiplication by 4
	a=64;
	b=2;
	c = a<<b;
	/*Final value of c becomes 256 */
		li a0, 64
		li a1, 2
		sll a2, a0, a1

Right-shift by 2
=
Division by 4
	a=64;
	b=2;
	c = a>>b;
	/*Final value of c becomes 16 */
		li a0, 64
		li a1, 2
		sra a2, a0, a1

Right-shift by 2
=
Division by 4.
 	
Floor(50/2) = 12
	a=50;
	b=2;
	c = a>>b;
	/*Final value of c becomes 12 */
		li a0, 50
		li a1, 2
		sra a2, a0, a1

Solved Exercise

- Program to (unsigned) divide register a0 by 8. Result in a1
- Srli a1 a0 3

- Program to (signed) divide register a0 by 8. Result in a1
- Srai a1 a0 3

- Program to multiply register a0 by 8. Result in a1
- Slli a1 a0 3

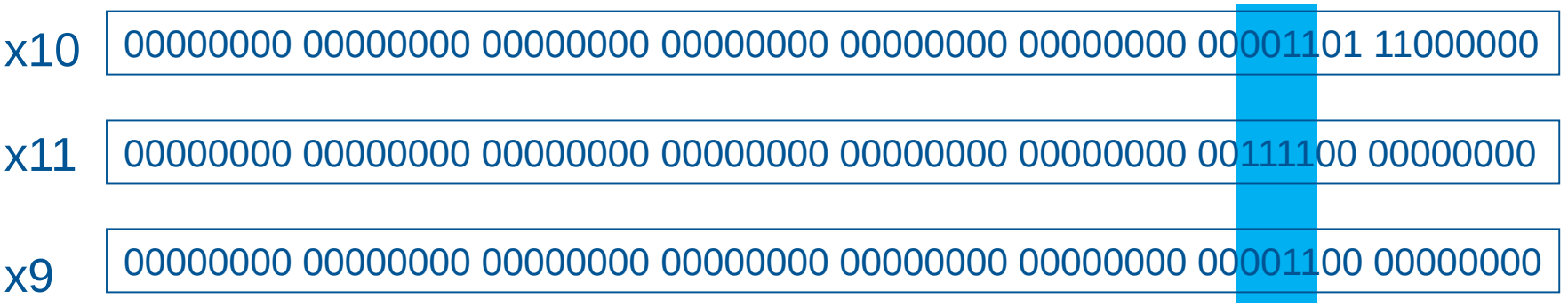
Solved Exercise

- Program to multiply register a0 by 10. Result in a3
- Slli a1 a0 3 # y = x*8
- Slli a2 a0 1 # z = x*2
- Add a3 a1 a2 #v = y+z = x*10

AND Operations

- Useful to mask bits in a word
  - Select some bits, clear others to 0

```
and  x9,x10,x11
```

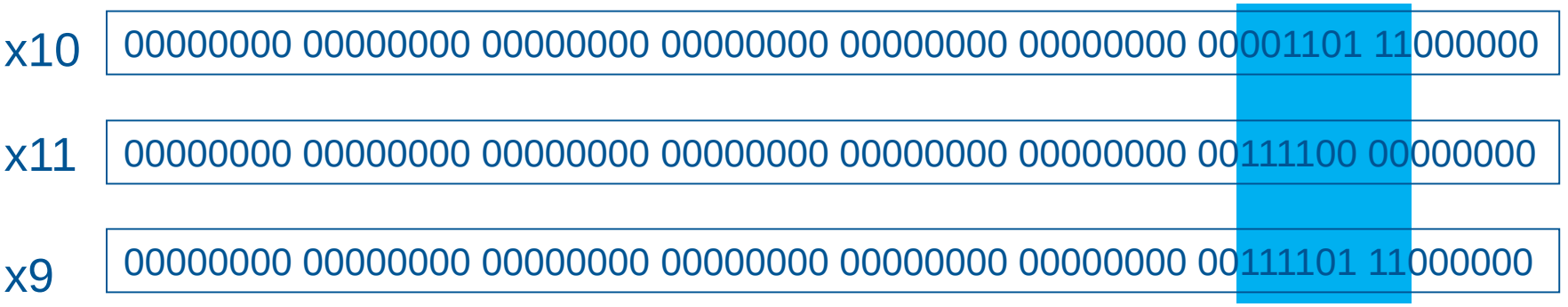


- Clear a register:
- andi t0, t0, 0 # Clear register t0

OR Operations

- Useful to include bits in a word
  - Set some bits to 1, leave others unchanged

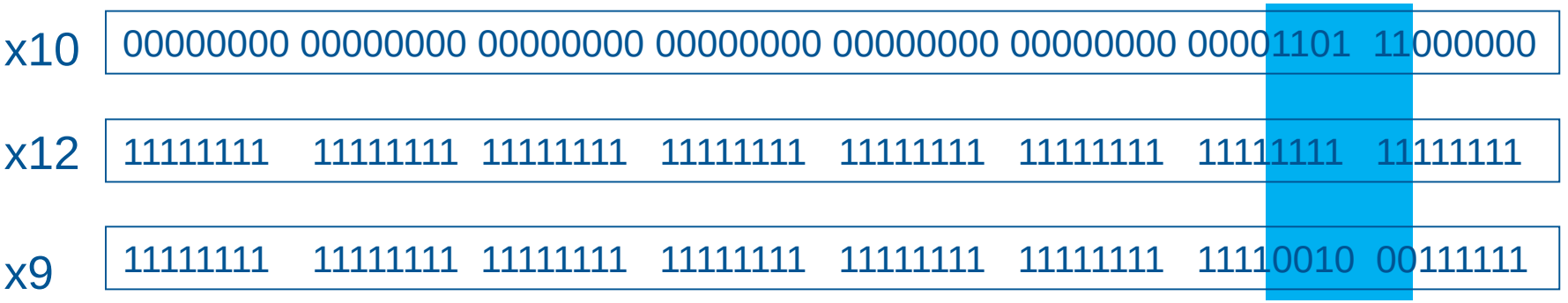
```
or x9,x10,x11
```



XOR Operations

- Differencing operation
  - Set some bits to 1, leave others unchanged

```
xor x9,x10,x12
```



Negate

Negate (NEG) instruction computes two's complement of a value.

```
neg x6, x5        # x6 ← −x5
```

If value of x5 was 1, then above instruction would store -1 in x6.

Logical instructions

```
li a0, 6                # a0 <- 0110 (in binary)
li a1, 10               # a1 <- 1010 (in binary)
and a2, a0, a1          # a2 <- 0010 (in binary) or 2 (in decimal)
or a3, a0, a1           # a3 <- 1110 (in binary) or 14 (in decimal)
not a4, a0              # a4 <- 0xFFFFFFF9 (1's complement of 6 in 32bits)
neg a5, a0              # a5 <- 0xFFFFFFFA (2's complement of 6 in 32bits)
```

Compute 1's complement of 0110 in 32 bits.
Answer: Original number is 0000 0000 0000 0000 0000 0000 0000 0110
1's complement is 1111 1111 1111 1111 1111 1111 1111 1001
In hexa, this is same as F F F F F F F 9 , that means 0xFFFFFFF9

Solved question

- Let x5 = 0x00000000AFAFAFAF
- x6 = 0x1234567812345678
- Then, what is the value of x7 after these instructions?
- `slli x7, x5, 4`
- `or x7, x7, x6`
- Remember: the shift of “4” is at bit-level, whereas the number is specified as a hexadecimal number. Hence, 4-bit shift will lead to shift of one hex symbol.
- Answer:
- Explanation: After slli, x7 becomes 0x0000000AFAFAFAF0
- Finally, x7 is 0x1234567AFAFEFEF8.

Solved Question

- Assume RV32. Register a0 currently stores some non-zero value.

Set its 29th bit to 1.

```
li a1 1
slli a2 a1 29
or a0 a0 a1
```

Reset its 29th bit

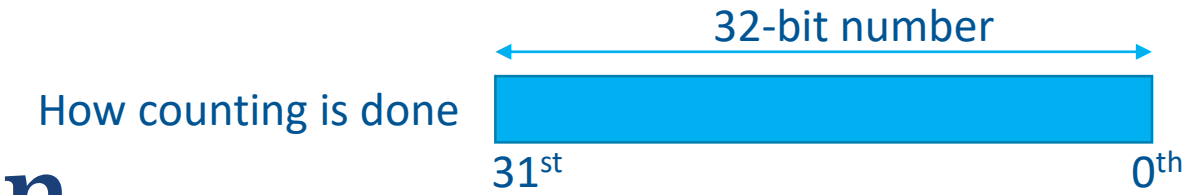
```
li a1 1
slli a2 a1 29
not a3 a2
and a0 a0 a3
```

Flip its 29th bit

```
li a1 1
slli a2 a1 29
xor a0 a0 a2
```

29th bit is actually 30th bit from right hand side. Hence, we are shifting by 29 times.

Solved Question



- Assume RV32. Check whether the lowest 4 bits of register a0 are 1010. If so, put 1 in a5
- li a1 0b1010
- li a2 0b1111
- and a3 a0 a2
- xor a4 a3 a1 # if a3 was 1010, then a4 should be all zeros.
- Seqz a5 a4

Instruction	Type	Example	Meaning
AND	R	`and rd, rs1, rs2`	`R[rd] = R[rs1] & R[rs2]`
OR	R	`or rd, rs1, rs2`	`R[rd] = R[rs1] | R[rs2]`
XOR	R	`xor rd, rs1, rs2`	`R[rd] = R[rs1] ^ R[rs2]`
AND immediate	I	`andi rd, rs1, imm12`	`R[rd] = R[rs1] & SignExt(imm12)`
OR immediate	I	`ori rd, rs1, imm12`	`R[rd] = R[rs1] | SignExt(imm12)`
XOR immediate	I	`xori rd, rs1, imm12`	`R[rd] = R[rs1] ^ SignExt(imm12)`
Shift left logical	R	`sll rd, rs1, rs2`	`R[rd] = R[rs1] << R[rs2]`
Shift right logical	R	`srl rd, rs1, rs2`	`R[rd] = R[rs1] >> R[rs2]` *(logical)*
Shift right arithmetic	R	`sra rd, rs1, rs2`	`R[rd] = R[rs1] >> R[rs2]` *(arithmetic)*
Shift left logical immediate	I	`slli rd, rs1, shamt`	`R[rd] = R[rs1] << shamt`
Shift right logical imm.	I	`srli rd, rs1, shamt`	`R[rd] = R[rs1] >> shamt` *(logical)*
Shift right arithmetic immediate	I	`srai rd, rs1, shamt`	`R[rd] = R[rs1] >> shamt` *(arithmetic)*