Shifter and Rotator

Sparsh Mittal

Understanding sign/zero extension

Extension

- Extension is required when we want to preserve the numeric value, while we represent a number using more bits; or store it in a register with more number of bits.

- There are two possibilities: sign-extension and zero-extension
- Sign-extension: replicate the sign bit to the left
- Zero-extension: replicate zero bit to the left.

Understanding extension

Case 1: Assume our 8b data is 1101 0011. We need to store it in a 16b register.

Sign- extension

1111 1111 1101 0011

Zero- extension

0000 0000 1101 0011

Case 2: Assume our 8b data is 0101 0011. We need to store it in a 16b register.

Sign- extension

0000 0000 0101 0011

Zero- extension

0000 0000 0101 0011

Notice that in case 2, sign- and zero-extension have the same impact.

Sign extension example.

- Write the value of -7 in 32 bits.

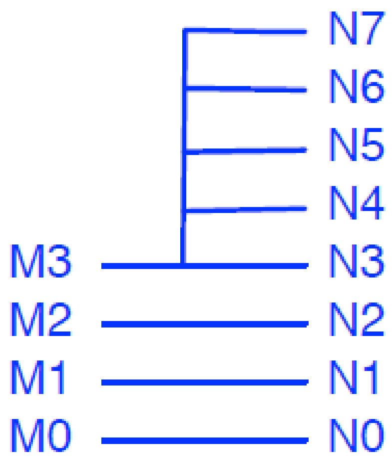
$(-7)_{10} = (1001)_2$ # 2's complement representation.

- Sign extension in 32-bit representation:

$(-7)_{10} = \mathbf{(1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1001)}_2$

Question

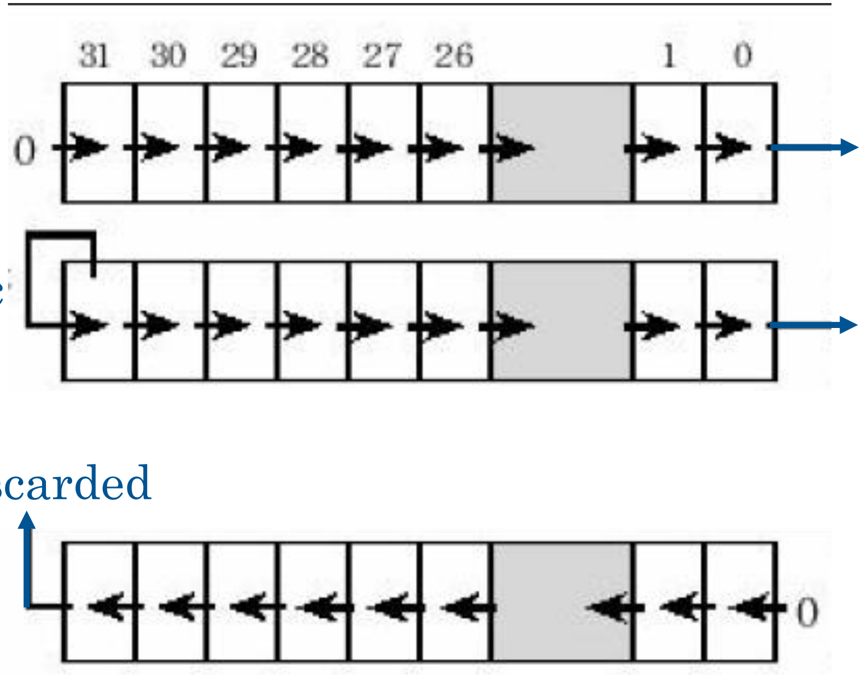
- Draw a circuit for a sign extension unit with a 4-bit input and an 8-bit output



Arithmetic and logical shifts

n = #shifted bits

Equivalent to



Shift right logical (SRL)		Discarded	Unsigned divide by 2^n
Shift right arithmetic (SRA)		Discarded	Signed divide by 2^n
Shift left logical (SLL)	Discarded		Multiply by 2^n (both signed and unsigned)

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

Visual explanation

	Hexadecimal	32b Binary							
a0	0x80000000	1000	0000	0000	0000	0000	0000	0000	0000
a1	0x1								

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

a2	0x00000000	0000	0000	0000	0000	0000	0000	0000	0000	Overflow happens

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

a3	0x40000000	0100	0000	0000	0000	0000	0000	0000	0000

On doing, "**a4 = a0 sra 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

a4	0xC0000000	1100	0000	0000	0000	0000	0000	0000	0000

- There is no shift-left arithmetic (SLA) operation. Why?
- Answer: Two's complement ensures that SLA and SLL lead to same effect.

Examples

- 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!

Shifter

- An N-bit shifter can be built from N N:1 multiplexers.
- The input is shifted by 0 to N-1 bits, depending on the value of the $\log_2 N$-bit select lines.

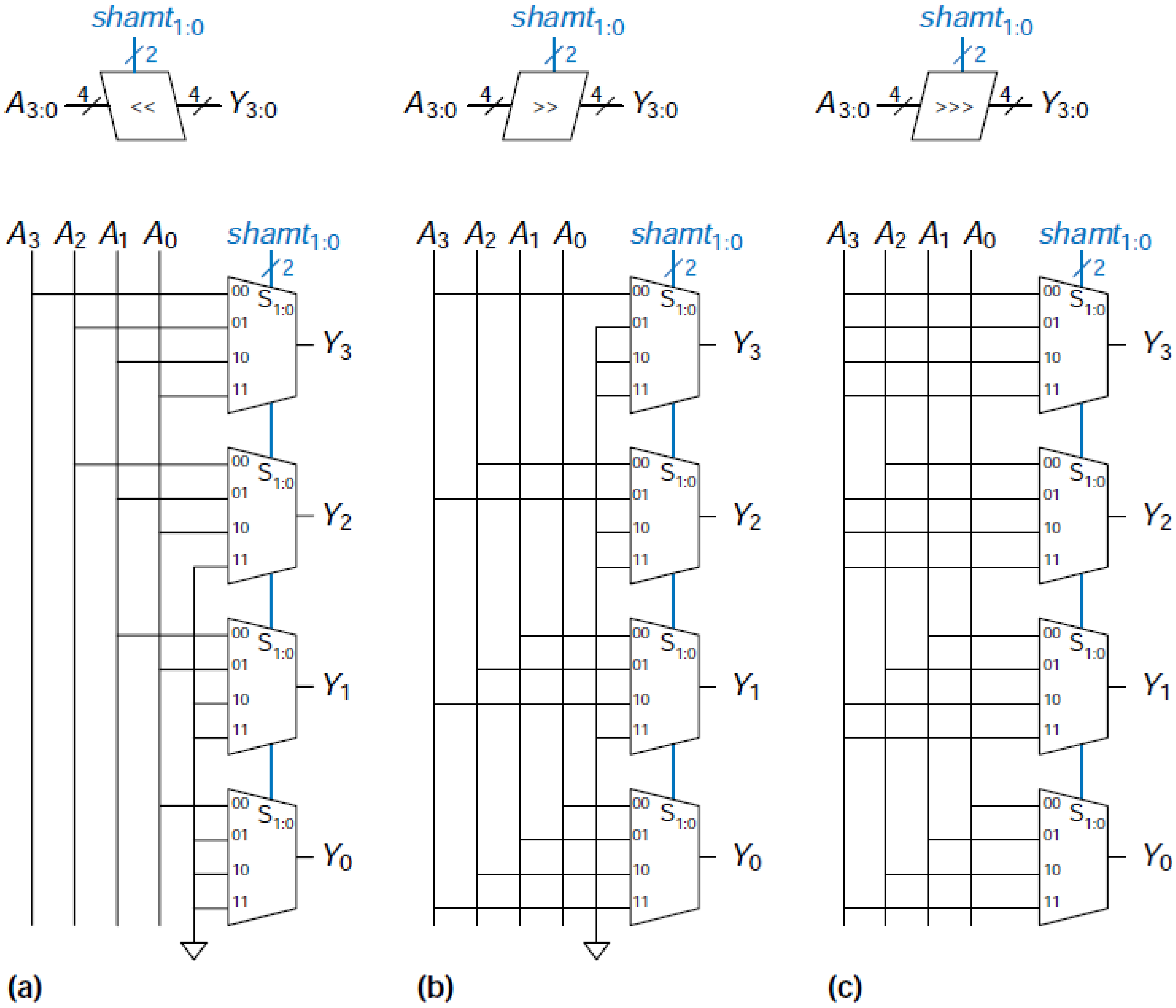


Figure 5.16 4-bit shifters: (a) shift left, (b) logical shift right, (c) arithmetic shift right

Rotator

- **Rotator**—rotates number in circle such that empty spots are filled with bits shifted off the other end.
- 11001 ROR 2 = 01110;
- 11001 ROL 2 = 00111