Logical and Arithmetic Shifting

Prof. James L. Frankel
Harvard University

Version of 10:36 AM 2-Dec-2021

Logical Shifting

- Logical shifting moves the bits in data either left or right with the incoming bits always equal to zero
- Logical shift right – distance is one bit
  - Before:
  - After:

Before:	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
After:	0	b_7	b_6	b_5	b_4	b_3	b_2	b_1

- Logical shift left – distance is one bit
  - Before:
  - After:

Before:	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
After:	b_6	b_5	b_4	b_3	b_2	b_1	b_0	0

Arithmetic Shifting

- Arithmetic shifting moves the bits in data right with the incoming bits replicated from the sign bit
- Arithmetic shift right – distance is one bit
  - Before:
  - After:

Before:	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
After:	b_7	b_7	b_6	b_5	b_4	b_3	b_2	b_1

- Arithmetic shift right – distance is two bits
  - Before:
  - After:

Before:	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
After:	b_7	b_7	b_7	b_6	b_5	b_4	b_3	b_2

Why is Arithmetic Shifting Useful?

- Two's-complement signed numbers
  - Positive
    - Equivalent for logical and arithmetic right shift
  - Negative
    - Logical and arithmetic have different results
      - Logical would shift in zero bits
      - Arithmetic would shift in one bits (for negative values)

Example of Negative Number with Logical Shift

- Using an 8-bit representation

- The value -2_{10} would be represented in binary as
  - $2_{10} = 0000\ 0010_2$
  - $\sim 2_{10} = 1111\ 1101_2$
  - $\sim 2_{10} + 1 = 1111\ 1110_2$

- -2_{10} logical shift right one-bit position would be $0111\ 1111_2$ or 127_{10}

Example of Negative Number with Arithmetic Shift

- Using an 8-bit representation

- The value -2_{10} would be represented in binary as
  - $2_{10} = 0000\ 0010_2$
  - $\sim 2_{10} = 1111\ 1101_2$
  - $\sim 2_{10} + 1 = 1111\ 1110_2$

- -2_{10} arithmetic shift right one-bit position would be $1111\ 1111_2$ or -1_{10}

Another Example of Negative Number with Arithmetic Shift

- Using an 8-bit representation

- The value -3_{10} would be represented in binary as
  - $3_{10} = 0000\ 0011_2$
  - $\sim 3_{10} = 1111\ 1100_2$
  - $\sim 3_{10} + 1 = 1111\ 1101_2$

- -3_{10} arithmetic shift right one-bit position would be $1111\ 1110_2$ or -2_{10}

How Does Integer Division Round Results?

- This is interesting when either the numerator or the denominator is negative

How Does Integer Division Round Results?

- This is interesting when either the numerator or the denominator is negative
- Usually toward zero, but sometimes toward negative infinity (*i.e.,* floor)

- In C89, either is acceptable
  - That is, the result is implementation-defined
- In C99, the result is rounded toward zero
- In Python & Ruby, the result is rounded toward negative infinity
  - You can read on-line about the reasons for these decisions