L5 - Bit Shifting

Luke Sweeney

CSE 2312

Dr. Losh

UT Arlington

Feb 2, 2021

Bit Shifting

Right bit shifting by a number `k` will divide the number by `2^k`

```
   1 0 0 0 0 0 0 0    (unsigned 128)
>> 4
-----------------
   0 0 0 0 1 0 0 0   (unsigned 8)

   (128 / 2^4 = 128 / 16 = 8)



          0 0 1 0 0 0 0 0   (unsigned 32)
32 >> 1 = 0 0 0 1 0 0 0 0
32 >> 2 = 0 0 0 0 1 0 0 0
32 >> 3 = 0 0 0 0 0 1 0 0
32 >> 4 = 0 0 0 0 0 0 0 1
32 >> 5 = 0 0 0 0 0 0 0 0  (eventually goes to 0)
```

Left bit shifting by `k` is like multiplying by `2^k`

```
         0 0 0 1   (unsigned 1)
1 << 1 = 0 0 1 0
1 << 2 = 0 1 0 0
1 << 3 = 1 0 0 0  (1 * 2^3 = 1 * 8 = 8)
```

Logical Shifts

Unsigned shifts are called "logical shifts"

```
('-' represents a bit, 0 or 1)

For right shifting, you insert a 0 on the left and throw away the bit out the right side

            |====================|
       0 -> | - - - - >> - - - - | -> (trash)
            |====================|

Left bit shifting is the same
```

```
            |====================|
 (trash) <- | - - - - << - - - - | <- 0
            |====================|
```

Arithmetic Shifts

For shifting signed integers, we use "arithmetic shifts".

```
int8_t x = -2;
```

```
('-' represents a bit, 0 or 1)

Arithmetic shift left (ASL)

            |====================|
 (trash) <- | - - - - << - - - - | <- 0
            |====================|

Arithmetic shift right (ASR)

            |====================|
      x ->  | x - - - >> - - - - | -> (trash)
            |====================|
```

Remember we're working with signed numbers. Arithmetic shift left (ASL) is pretty much the same as Logical shift left (LSL). This is like multiplying by 2^k where k is the number of bits shifted.

Arithmetic shift right is a little different. For For ASR, we maintain the sign bit: the leftmost bit shouldn't change.

```
            1 0 0 0 0 0 0 0    (signed -128)
-128 >> 1 = 1 1 0 0 0 0 0 0    (signed -64)

If we hadn't have copied the sign bit, we would have
            0 1 0 0 0 0 0 0    (signed +64)
Which is not correct.
```

Operation	Name	Operator	Number Type	How
LSL	Logical Shift Left	`<<`	`unsigned`	Insert 0 on right, push bits off left
LSR	Logical Shift Right	`>>`	`unsigned`	Insert 0 on the left, push bits off right
ASL	Arithmetic Shift Left	`<<`	`signed`	Insert 0 on right, push bits off left
ASR	Arithmetic Shift Right	`>>`	`signed`	Shift bits right, maintain sign bit, push off right bit

If you shift many times, `ASL` , `LSL` , and `LSR` will `-> 0` .

`ASR` will `-> 0` if the number is positive, `-> -1` if the number is negative.

Error Cases

say you have a signed 8 bit integer 32, which you ASL by 2

```
          0 0 1 0 0 0 0 0     (signed +32)
32 << 1 = 0 1 0 0 0 0 0 0     (signed +64)
32 << 2 = 1 0 0 0 0 0 0 0     (signed -128)
```

This is not mathematically correct. You have to be careful when working with signed ints.

- Use `ASL` and `ASR` with `int`
- Use `LSR` and `LSL` on `uint`
- Be sure you don't shift too far and end up with 0
- The easiest way to prevent errors is to write out every bit instead of working with decimals.