# Binary Multiplication and Division

Tom Kelliher, CS 220

## 1 Administrivia

### Today's Objectives

1. Perform left and right arithmetic shifts on binary numbers, recognizing that this is equivalent to multiplying by two and dividing by two, respectively, and also recognizing when the result will be incorrect.

2. Perform unsigned binary multiplication, using both the traditional "paper and pencil" method and Booth's algorithm.

3. Perform unsigned restoring and non-restoring division.

4. Implement Booth's algorithm and non-restoring division by modifying C programs implementing "paper and pencil" multiplication and restoring division.

### Next Up

Read 2.6 and 2.7.

1. Explain the structure of IEEE single precision binary numbers.

2. Explain the meanings of overflow and underflow for floating point numbers.

3. Convert decimal numbers in scientific notation to and from IEEE single precision binary numbers.

4. Add, subtract, and multiply IEEE single-precision binary numbers, producing correctly normalized results.

## 2 Warm-Up

1. After performing a left shift of the binary value 10010110, the result is

   (a) 00101100

   (b) 11001011

   (c) 01001011

   (d) 00101101

2. After performing a right shift of the binary value 10010110, the result is

   (a) 00101100

   (b) 11001011

   (c) 01001011

   (d) 00101101

3. The value of v after executing this C code

```
v = 0xC;
v = v << 3;
```

is

(a) 0xC

(b) 3

(c) 0x60

(d) 0xC000

4. The value of v after executing this C code

```
v = 0x4000;
v = v >> 1;
```

is

(a) 0xC000

(b) 0x2000

(c) 0x8000

(d) 0xA000

5. Performing a right shift always gives a result equivalent to dividing by two. (Consider -3 — 11111101.)

   True/False

6. Performing a left shift always gives a result equivalent to multiplying by two.

   True/False

7. Consider a multiplier containing a run of $n$ 1 bits. Booth's algorithm will perform $n$ additions as a result of this run.

   True/False

8. The key observation in the non-restoring division algorithm is

   (a) Shifting the divisor one bit position right (halving it) can be done at any time.

   (b) Addition is much faster than subtraction.

   (c) In any one step, if the partial dividend is negative, we would add the divisor (restore) and then subtract half the current divisor during the next step. This amounts to adding half the current divisor.

   (d) None of the above.

# 3 Problems

1. Modify mult.c to implement Booth's algorithm. Hints: Before entering the for loop, shift the multiplier left one bit position to make available the 0 bit assumed to be to the right of the least significant bit of the multiplier in Booth's algorithm. Then, inside the for loop, test the two least significant bits of the multiplier:

   ```
   if ((multiplier & 0x3) == 0x2)   /* The binary 10 case. */
      ...
   ```

   Note that a mask value of 0x3 will isolate the two least significant bits of the multiplier.

   Compile, run, and test your modifications.

2. Modify div.c to implement non-restoring division. Hints: Note that the example shown in Table 2.6 shows partial dividends and divisors with a bit to the right of the binary point. To implement this bit, shift both the dividend and the divisor left one bit position before initially subtracting the divisor from the dividend. Then, before printing the quotient and remainder, shift the dividend right one bit position.

   Referring to Figure 2.5 on pg. 70, recall that the "Subtract divisor from dividend" step is not part of the loop.

   Compile, run, and test your modifications.