Binary Addition, Subtraction, and Multiplication

Tom Kelliher, CS 220

Oct. 21, 2005

1 Administrivia

Announcements

Assignment

Read 3.6–3.7.

From Last Time

Integer representation schemes.

Outline

1. Binary Addition.
2. Binary Subtraction.
3. Overflow.
4. Binary Multiplication.

Coming Up

Floating point.

2 Binary Addition

1. Similar to decimal addition.
2. Nomenclature: Augend + Addend = Sum
3. Eight bit example: 0x15 + 0x4C.

3 Binary Subtraction

1. Nomenclature: Minuend − Subtrahend = Difference
2. To subtract, we add. How?
3. Eight bit example: 0x15 - 0x4C.

4 Overflow

1. In eight bits, consider 0x7F + 0x1. Augend and Addend are positive, but the sum is negative.

   How can this be?

   Would we have this result if we were working in 16 bits?

2. Overflow: when the sign of the result is wrong.
3. If we are adding two numbers of different signs or subtracting two numbers of the same sign, we will not have overflow. Why?
4. Overflow cases:

Operation	A	B	Overflow Result
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

5. Addition logic:

```
if the signs of A and B are the same
   if the signs of A and Sum are different
      overflow has occured.
```

   Subtraction is similar.

   A compact way of writing this in C:

```
if (a ^ b >= 0 && a ^ sum < 0)
   overflow();
```

6. Overflow and programming languages:

   (a) C: no overflow exceptions. Uses `addu`, etc.

   (b) FORTRAN: overflow exceptions. Uses `add`, etc.

   What's an exception?

5 Binary Multiplication

1. Consider paper and pencil binary unsigned multiplication:

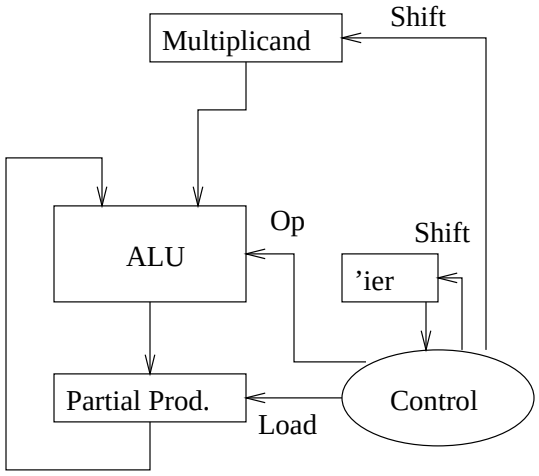
   (a) Shift multiplicand left one bit position after each add/hold cycle.

   (b) Add/hold depending upon current multiplier bit (examine lsb; shift right).

   (c) Example: 1101×0110.

(d) Multiplying two n-bit numbers results in a $2n$-bit result.

2. Naive 32-bit shift-and-multiply hardware:



How many bits in the datapath? *Demonstrate on board*.

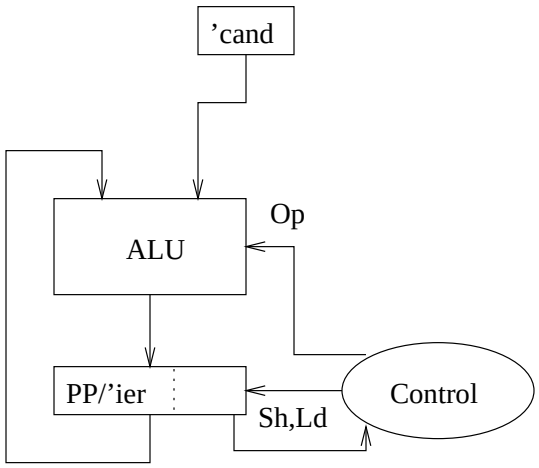
Running time?

3. Observation: Shifting multiplicand left and keeping partial product stationary is equivalent to keeping multiplicand stationary and shifting partial product right.

   (a) Bonus 1: only 32-bits are added at any one time.

   (b) Bonus 2: the multiplier can be stored in the unused part of the partial product register.

More sophisticated 32-bit shift-and-multiply hardware:



Demonstrate on board.

4. Optimal running time: $O(\log n)$, achieved with tree of carry-save adders with CLA adder at root.