lecture 7

sequential circuits 3:

- integer multiplication & division,
- floating point +, −, *, /

Integer Multiplication

```
     3 5 2
   × 9 6 4
   -------
   1 4 0 8
 2 1 1 2
3 1 6 8
-----------
3 3 9 3 2 8
```

Unsigned Integer Multiplication

(how to do it in binary?)

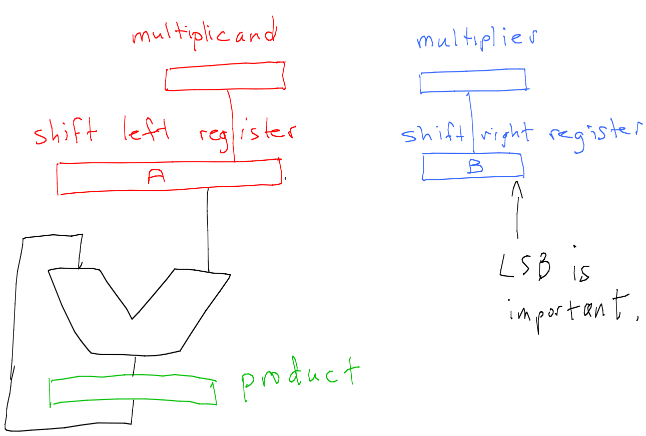
$$\begin{array}{ccccccccc} & & & A_{n-1} \ldots & A_2 & A_1 & A_0 & & \text{multiplicand} \\ & & \times & B_{n-1} \ldots & B_2 & B_1 & B_0 & & \text{multiplier} \\ \hline P_{2n-1} & \ldots & P_n & P_{n-1} & P_2 & P_1 & P_0 & & \text{product} \end{array}$$

Note: $(2^n - 1)(2^n - 1) < 2^{2n} - 1$

```
        0 1 0 0 1 1 0 1        multiplicand
      * 0 0 0 1 0 1 1 1        multiplier
      -----------------
        0 1 0 0 1 1 0 1
      0 1 0 0 1 1 0 1
    0 1 0 0 1 1 0 1
  0 0 0 0 0 0 0 0
0 1 0 0 1 1 0 1
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
---------------------------
0 0 0 1 1 0 1 1 1 0 1 0 1 1    product
```

Alternative approach?

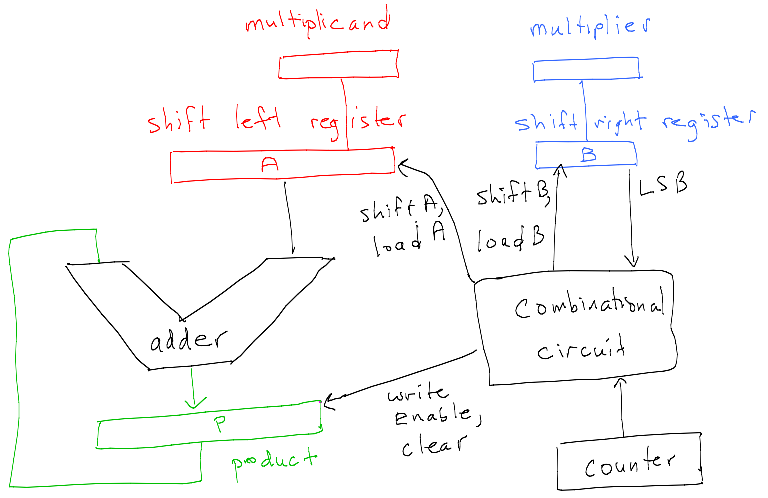
```
  0 1 0 0 1 1 0 1
* 0 0 0 1 0 1 1 1
-----------------
```



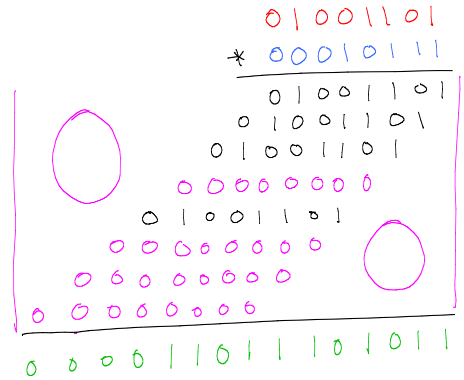
Multiplication Algorithm

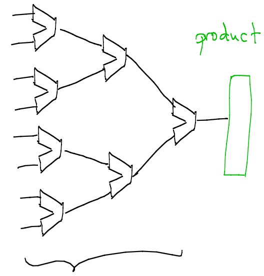
load multiplicand into lower n bits of 2n bit register A
load multiplier into n bit register B
clear 64 bit product register P
for counter = 1 to n {
  if LSB of B is 1
    P = P + A
  Shift A left (one bit)
  Shift B right (one bit)
}

in parallel

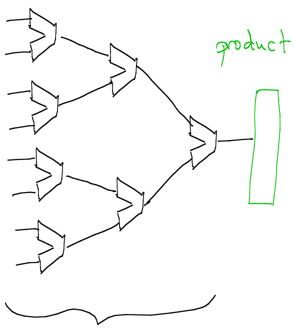


Fast integer multiplication (sketch)

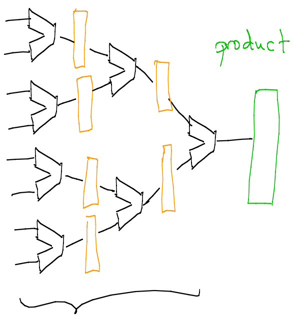




Use big and fast adders.



One clock cycle.



Use registers.

Take several clock cycles.

Why?

Long Division

```
                 785   ← quotient
divisor  53 | 41627    ← dividend
              371
               452
               424
                287
                265
                 22    ← remainder
```

How would you write out the algorithm?

```
         700              80               5
                         700              80
                                         700
53 | 41627       53 | 41627       53 | 41627
     37100            37100            37100
      4527             4527             4527
                       4240             4240
                        287              287
                                         265
                                          22
```

Revisit this grade school algorithm on your own and see if you really understand it.

```
                                 785
divisor       000000110001 0001          quotient
110101 | 101000101001 1011              dividend
         110101
          111000
          110101
             111001
             110101
                1001011
                 110101
                  10110
                   22
```

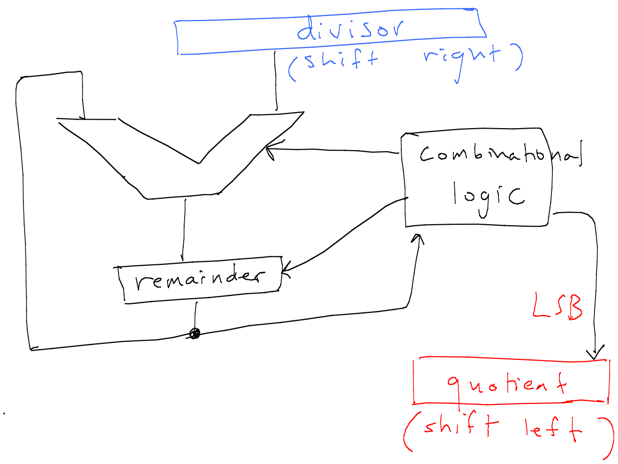
To perform subtractions, either use twos complement or use grade school subtraction (base 2).

CAREFUL!

Algorithm for Long Division
(note: divisor < dividend)

```
divisor = divisor * 2^n
quotient = 0
remainder = dividend
for i = 1 to n {
    shift quotient left by one bit
    if ( divisor ≤ remainder) {
        set LSB of quotient to 1
        remainder = remainder - divisor
    }
    shift divisor right
}
```

Sketch (ignore register initialization)



Fast integer division ?

RST algorithm (1950s)

gives <u>some</u> speedup

(details omitted – its complicated)

but its still slower than multiplication

Floating Point Addition
(assuming positive numbers)

$x = 1.10101000000000000101010 \times 2^{-3}$

$y = 1.00100100010000010100001 \times 2^{2}$

$x + y$?

$x = .000011010100000000000101010 \times 2^{2}$

$x + y = $?

$x = 0.000011010100000000000101010 \times 2^2$

$y = 1.00100100010000010100001 \times 2^2$

$1.0011000110000001010000101010 \times 2^2$

28 bits significand

How to accomplish this ? (Sketch only.)

We need :

- compare exponents
- shift significand right (number with smaller exponent)
- big adder
- normalize (shift)
- round off

Floating point addition $(x > 0,\ y < 0)$

Represent negative non-integer using two's complement as follows:

$$y = 0001.01001 \times 2^{e}$$

$$-y = 1110.10111 \times 2^{e}$$

eg. $x = 26.5$

$y = -8.375$

$$\begin{array}{rr} & 1.1010100 \times 2^{4} \\ - & 0.1000011 \times 2^{4} \\ \hline \end{array}$$

write using twos complement

$$\begin{array}{rr} & 001.1010100 \times 2^{4} \\ + & 111.0111101 \times 2^{4} \\ \hline & 001.0010001 \times 2^{4} \end{array}$$

$\Longrightarrow x + y = 10010.001 = 18.125$

Floating point subtraction?

$$x - y$$

$$= x + (-y)$$

Floating point multiplication

$$\begin{array}{rr} & 1.\underline{\qquad\qquad} \times 2^{e_x} \\ * & 1.\underline{\qquad\qquad} \times 2^{e_y} \\ \hline & 1.\underline{\qquad\qquad\qquad} \times 2^{e_x + e_y} \end{array}$$

Similar to integer multiplication (but must take care of exponents too including handling overflow and underflow)

Floating point division

$$\frac{x}{y} = \frac{5146.8954}{26.721}$$

$$= \frac{51468954}{26721} \times \frac{10^{-4}}{10^{-3}}$$

$$26721 \overline{)\ 51468954.0000\ldots}$$

Similar to integer division