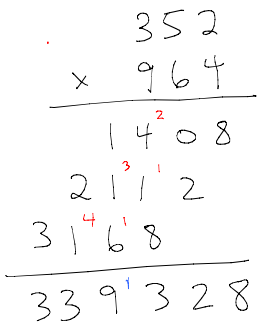
# lecture 7

## Sequential circuits 3

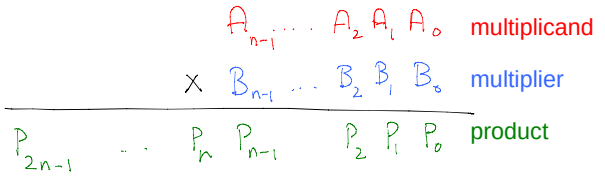
- integer multiplication and division
- floating point arithmetic
- finite state machines

February 1, 2016

## Integer multiplication (grade school)

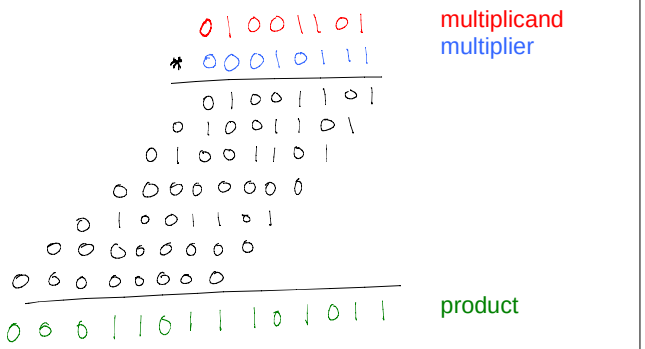


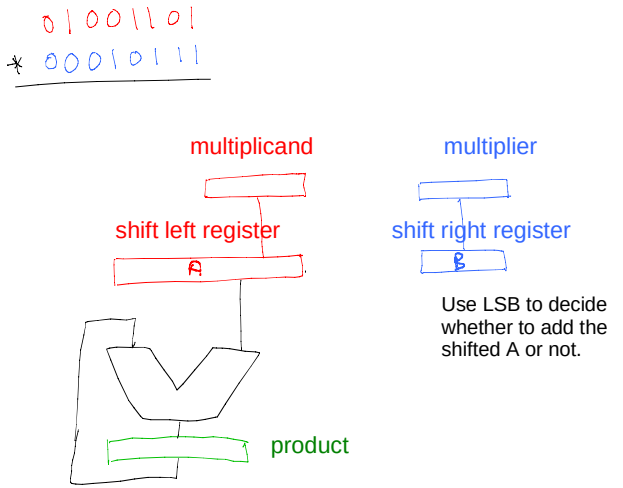

## How to do (unsigned) integer multiplication in binary ?




Why does the product have 2^ bits ?

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




## Algorithm: integer multiplication

// three instructions below done in parallel

extend multiplicand to 2n bits and load into left shift register A
load multiplier into right shift register B
clear product register

// three instruction below done in parallel

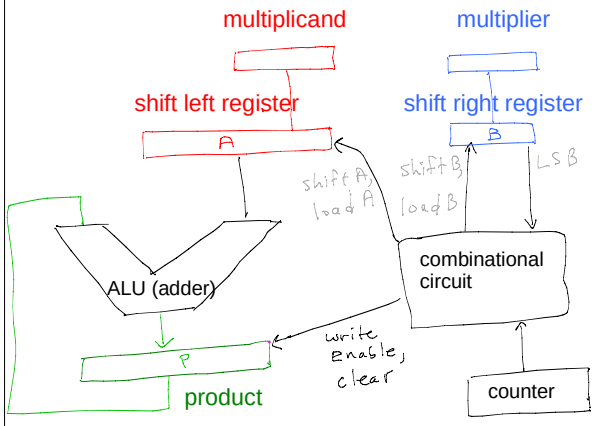
for counter = 1 to n {

    if LSB of B is 1`
        P = P + A

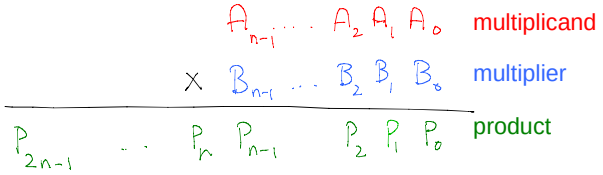
    shift A left by 1 bit
    shift B right by 1 bit

}

## Assignment 1 (Logisim) posted today




## Alternative approaches ?

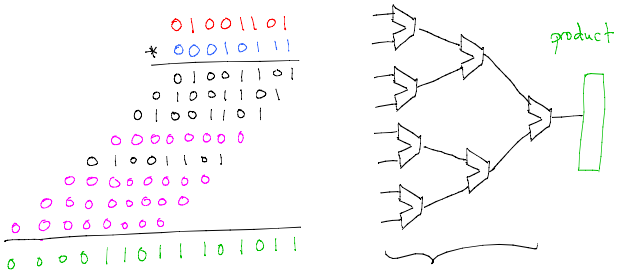



Use a combinational circuit only.

Advantages ?

Disadvantages ?

## Faster (good) and bigger (bad) ?



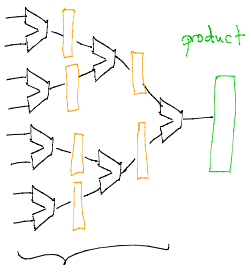

Requires **big and fast** adders !

(one clock cycle... details omitted)

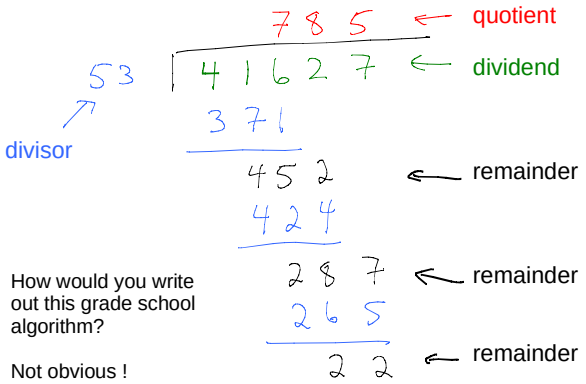
Use registers.

Take several clock cycles.

In terms of speed and size of circuit, this one falls between the original approach (Assignment 1) and the approach on the previous slide.

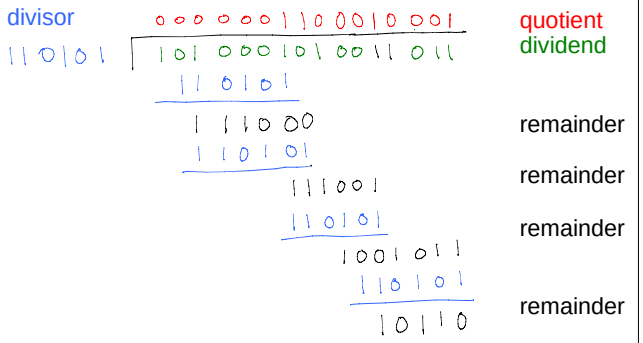



## Long Division

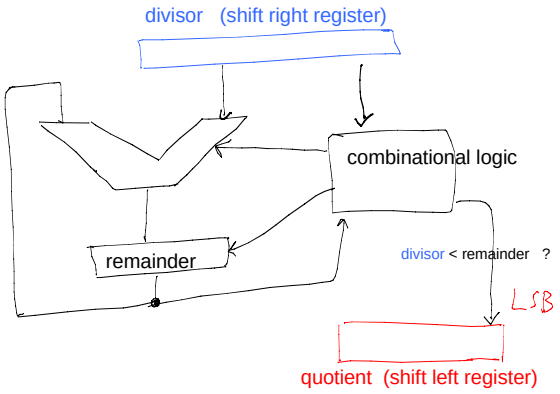



How would you write out this grade school algorithm?

Not obvious !




## Sketch only (ignore register initialization)




## Algorithm (time permitting)

```
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
}
```

## Recall Floating Point Addition (lecture 2)

x = 1.00100100010000010100001 * 2^ 2

y = 1.10101000000000000101010 * 2^ {-3}

x + y = ?

x = 1.00100100010000010100001 00000 * 2^2

y = .00001101010000000000001 01010 * 2^2

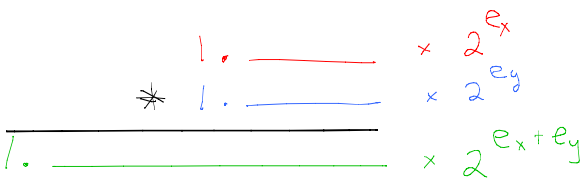
but the result x+y has more than 23 bits of significand

To do floating point addition (or subtraction), we need to:

- compare exponents
- shift right (x or y with smaller exponent)
- use a big adder
  (recall we can use two's complement to do subtraction or to add with negative numbers)
- normalize / shift
  (or treat specially if the result is non-normalized)
- round off

*Details omitted. I just wanted to mention the basic idea.*

## Floating Point Multiplication




Similar to integer multiplication, but ...

- significand must be approximated
- we must take care of exponents too, including handling overflow and underflow.
  (Underflow means getting a non-normalized number.)

## Floating Point Division

$$\frac{x}{y} = \frac{5146.8954}{26.721} = \frac{51468954}{26721} \times \frac{10^{-4}}{10^{-3}}$$

Similar to integer division, but we don't stop when remainder is less than divisor.

# Finite State Machines

Defined by:

- clock (discrete time)
- memory ("state" may change with each time step)
- input and output values
  - next state (i+1) depends on current state and/or input (i)
  - output (i) may depend on currentState and/or input (i)

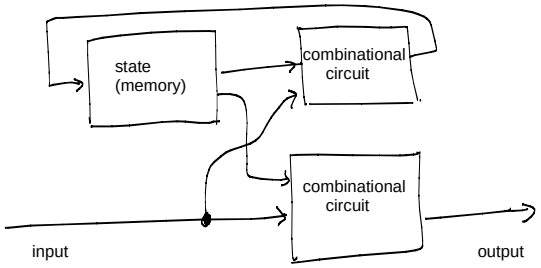
# Finite State Machine

called "finite automata" in COMP 330

| input i | current state i (memory) | output i | nextState i+1 (memory) |
|---|---|---|---|
| | | | |

The table has variables with values that can be written in binary.

However, you cannot implement it with a combinational circuit only. You need sequential circuits too (and a clock to control time).




The behavior of the machine is determined by an initial state and a sequence of inputs.

# e.g. turnstile

# e.g. turnstile

| input 0=coin 1=push | current state 0=locked 1=unlocked | output 0= ~turn 1= turn | next state 0=locked 1=unlocked |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |

# Announcements

- A1 posted today (due Wed. Feb. 10 at 23:59)

  TA (Josh and Noor) office hours will be posted soon.

- Quiz 2 next Monday covers lectures 3-6

  lastname A-H write in ARTS 145