

lecture 7

Sequential circuits 3

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

February 1, 2016

Integer multiplication (grade school)

$$\begin{array}{r}
 352 \\
 \times 964 \\
 \hline
 1408 \\
 2112 \\
 3168 \\
 \hline
 339328
 \end{array}$$

How to do (unsigned) integer multiplication in binary ?

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

Why does the product have 2^n bits ?

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

Handwritten binary multiplication showing the multiplicand (01001101), multiplier (00010111), and the resulting product (00011011101011).

Diagram of a shift register multiplier circuit. It shows a multiplicand register (A) that shifts left and a multiplier register (B) that shifts right. A combinational circuit uses the LSB of B to decide whether to add the shifted A to the product register (P).

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

Circuit diagram for a sequential multiplier. It includes a counter, a combinational circuit that controls shift registers A and B, an ALU (adder) that adds A to P, and a product register P.

Alternative approaches ?

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

Use a combinational circuit only.

Advantages ?

Disadvantages ?

Faster (good) and bigger (bad) ?

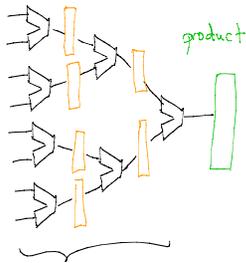
Diagram of a combinational multiplier circuit. It shows the multiplicand and multiplier registers, followed by a series of AND gates that generate partial products, which are then summed using a tree of adders to produce the final product.

Requires **big and fast** adders !
 (one clock cycle... details omitted)

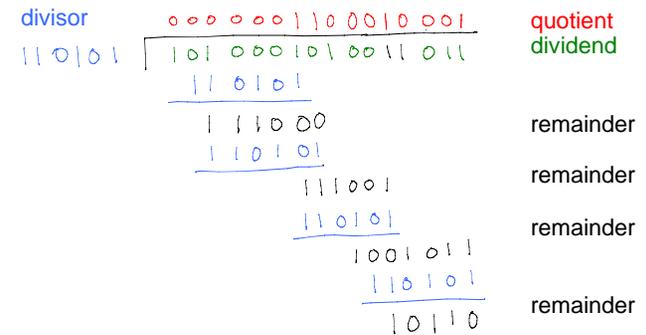
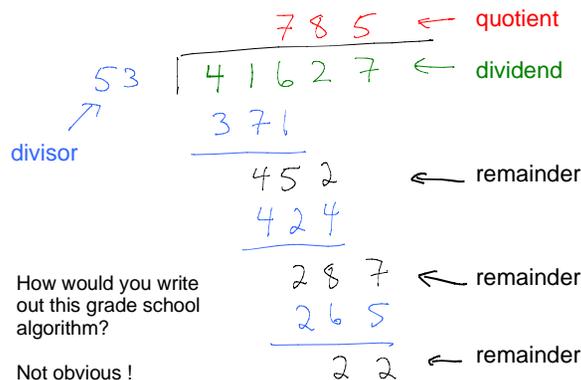
Use registers.

Take several clock cycles.

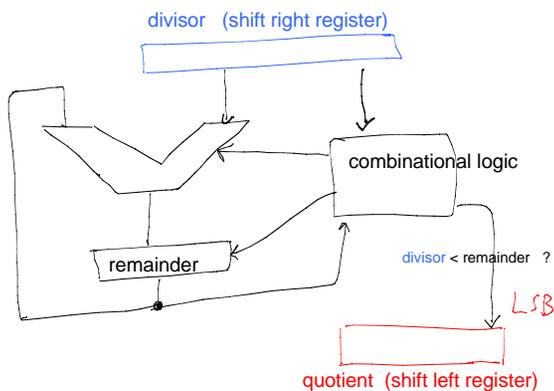
In terms of speed and size of circuit, this one falls between the original approach (Assignment 1) and the approach on the previous slide.



Long Division



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

$$\begin{array}{r}
 1. \text{---} \times 2^{e_x} \\
 * 1. \text{---} \times 2^{e_y} \\
 \hline
 1. \text{---} \times 2^{e_x + e_y}
 \end{array}$$

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

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

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 current state and/or input (i)

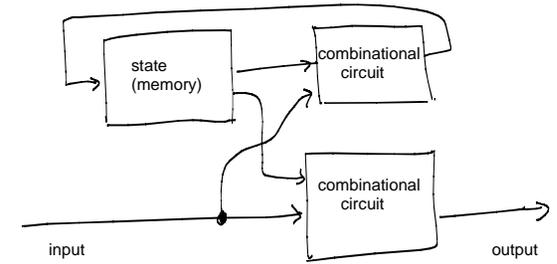
Finite State Machine

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