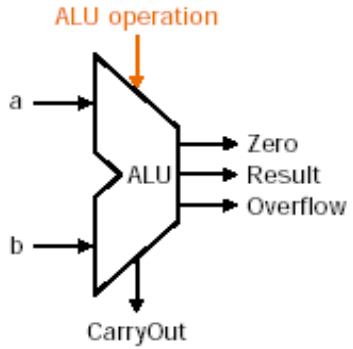
Combinational circuits: ALU

ALU is a combinational circuit

- **outputs depend only on inputs**
- **operations performed**
  - **AND**
  - **OR**
  - **ADD**
  - **SUB**
  - **SLT**
  - **`Zero (a == b)`**



This is an ARITHMETIC/logic unit **(Fig. 4.21)**

- **What about multiplication?**
  - **Result depends only on inputs**

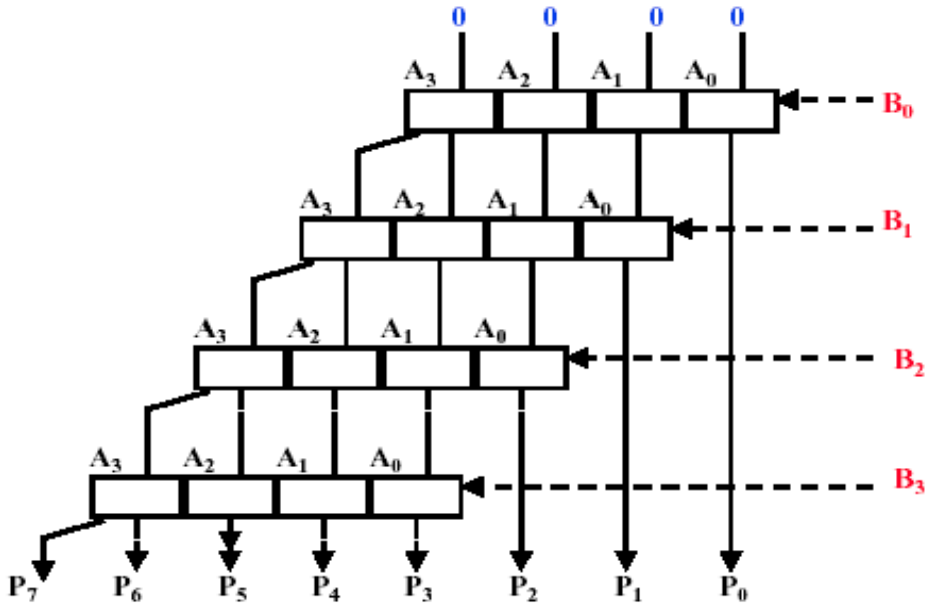
1	1	1	0	0		Carry
			1	1	1	Multiplicand
			1	0	1	Multiplier
			1	1	1	Partial products
		0	0	0		
	1	1	1			
1	0	0	0	1	1	Product

Can this be done with a combinational circuit?

Sure, but

- **How big is this truth table?**
- **What would the circuit have to look like?**

Unsigned Combinational Multiplier



This gets rather large even for 4 bits.

Can we split up the problem in groups of 4?

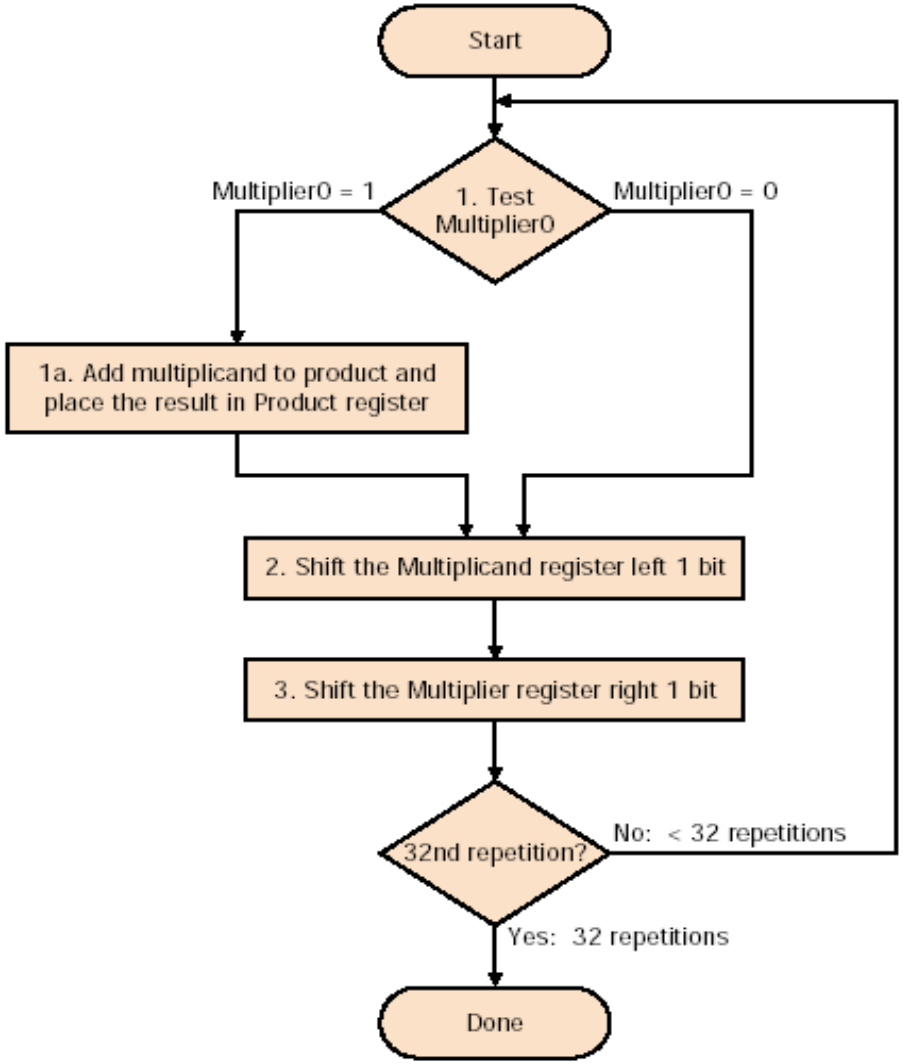
Reference:

http://www-inst.eecs.berkeley.edu/~cs152/

Multiplier

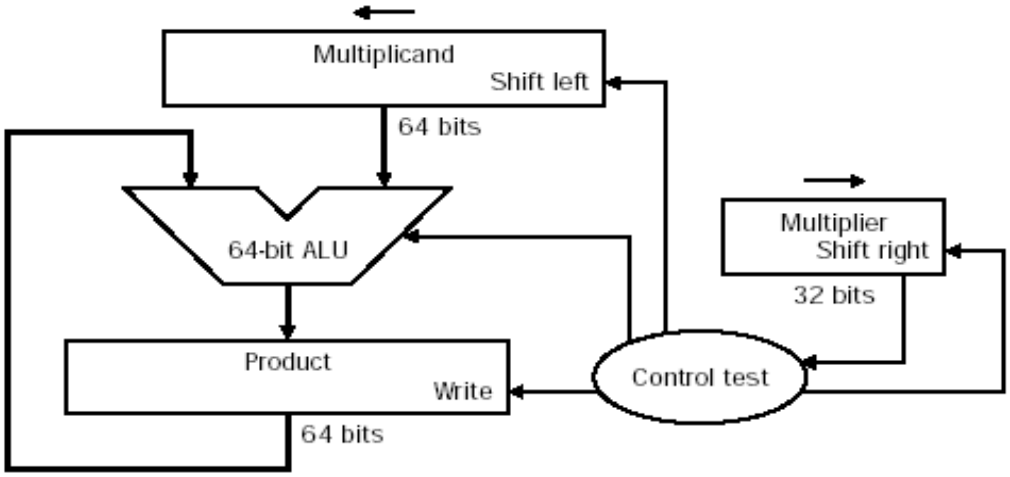
Better solution: iterate
(Fig. 4.26)

Add and shift:
Look at current bit position
If multiplier bit is 1
add multiplicand
else
add 0
Shift multiplicand



Multiplier

Multiplier circuit
(Fig. 4.26)



The efficiency of this can be improved, but there is a more fundamental problem:

How to implement the notions of:

state (registers)

update (clock)

Sequential circuits: state

Mathematical functions have no state: inputs combined to produce output, no memory

C/C++ functions may have state: store data from 1 call to another

- static variables
- objects store values in data members

State of a running program

- Values of variables
- Values of registers
- Contents of stack
- Address of current instruction

Hibernate laptop

- Save state of entire machine, including all programs

Sequential logic circuits

- Previous history is used together with inputs to produce output
- We don't care HOW previous value was obtained
- State encoded in bits
- Finite number of bits, so finite number of states

Sequential circuits vs. combinational

Comparison to combinational

- **Combinational circuits implement Boolean functions**
  - **Input ---> output, no memory**
  - **Only data inputs and control inputs determine the output**
- **Example: Coke (um, Pepsi) machine**
  - **Assume: price 75 cents, machine only accepts quarters**
  - **Action:**
    - **deposit quarter ---> no output**
    - **deposit quarter ---> no output**
    - **deposit quarter ---> drink delivered**
  - **Notice that the output was not always the same for the same input**
  - **This is not a Boolean function (combinational circuit)**
    - **Memory was used to determine output along with input**
- **Example: linked list object**
  - **call list.size() method**
  - **What is input? empty**
  - **Does it always return same value?**
    - **No, its output is the current length of the linked list**
- **Sequential circuit**
  - **Inputs: values x, labelled with subscripts**
  - **Outputs: values z, labelled with subscripts**

Uses clock, unlike combinational circuit

State

- **made up of devices called flip-flops**
- **k flip-flops store a k-bit number representing the current state**
- **values denoted by q with subscripts**

Output z computed from

- **inputs x**
- **state q**

Needed:

- **store current state**
- **update to new state**

Circuit elements

- **Combinational logic**
- **Clock**
- **State storage**

Example

state	input	output	new state
00	1	10	11

Sequential circuits: clock

"Without time, everything would happen at once."

- Anonymous

Clock

Outside world: way to tell time

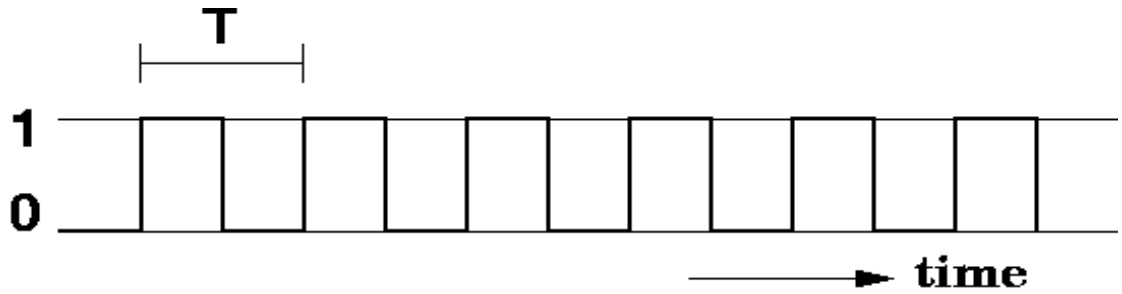
Computer: think of a metronome (number of beats, or cycles, per minute)

Measured in MHz or Ghz (millions or billions of cycles per second)

Timing diagram

x-axis: time

y-axis: voltage



values 0 and 1 are represented by low and high voltage

A clock is a device which alternates values between 0 and 1

period: time T

cycle: single alternation between 0 and 1

frequency: f = 1/T, units of Hz (cycles per second)

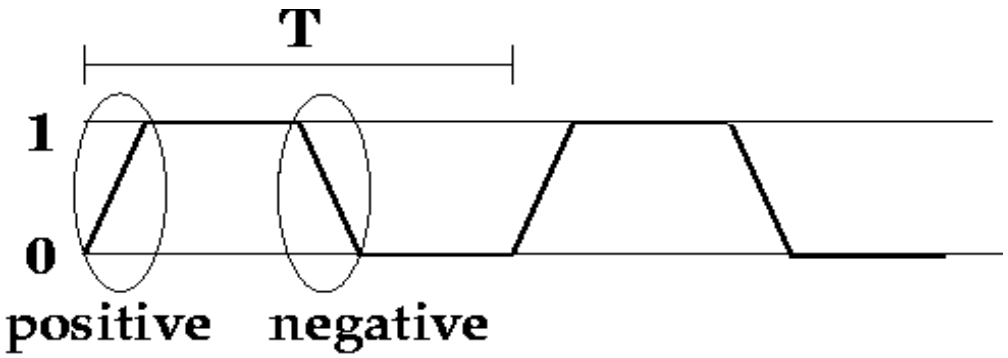
1 GHz means 10^9 cycles per second

(period is 10^{-9} seconds, which is 1 nanosecond)

Sequential circuits: clock

Clock edge

- **In each cycle, the clock transitions:**
  - **0 to 1: positive edge**
  - **1 to 0: negative edge**



- **This does not happen instantaneously**
  - **rise time: positive edge**
  - **fall time: negative edge**

Timed devices can only change state on an edge

- **positive triggered**
- **negative triggered**

Otherwise, they hold their value

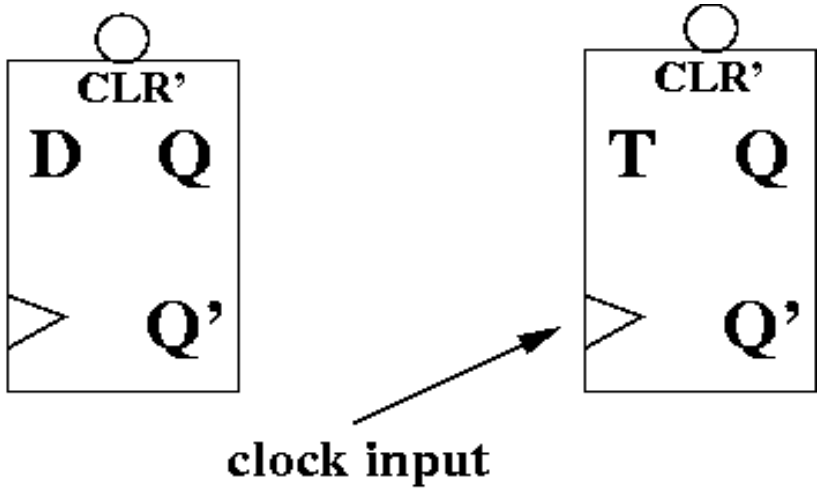
Flip-flops

Basic building blocks

- **Combinational circuits: gates**
- **Sequential circuits: flip-flops**

Examples of flip-flops

- **State: flip-flop stores 1 bit**
- **Inputs: control (D or T)**
  clock (positive edge)
- **Output: Q and Q' (negation)**
  Q is current state



Why both Q and Q'?

- **Because we can!**
- **Flip-flops can be built with NOR and NAND gates, and the negated output is essentially free**

Additional input (rarely drawn)

- **CLR': asynchronous clear**
  - **When set to 0, Q is immediately (asynchronously) set to 0**
  - **This is called active low (consumes less power)**

Flip-flops: latch

How to store a value with a circuit?

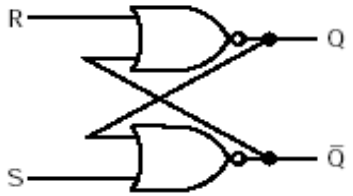
Simplest form: unclocked latch

Set-reset (S-R) latch

2 NOR gates with each output fed back

Note that this is not a valid combinational circuit

(contains a loop)



(Fig. B.12)

Assume:

R = 0

S = 0

Q = 1

Then output Q is fed back as input to the second NOR gate and negated,

producing the negation Q' = 0

Likewise, the output Q' is fed back to the first NOR gate and negated,

producing the output Q = 1

The output can be changed by asserting the value R or S:

If S = 1, then the second NOR gate produces 0, so Q continues to be 1

But, if R = 1, then the first NOR gate produces 0,

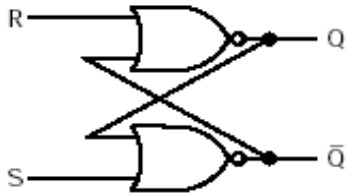
which becomes the new value of Q.

Flip-flops: latch

Characteristic table (corresponds to truth table):

Q^+ represents the NEW value of Q

R	S	Q^+
0	0	Q
0	1	1
1	0	0
1	1	undefined



(Fig. B.12)

When both set and reset are 0, Q is unchanged.
When only S is 1, Q becomes 1.
When only R is 1, Q becomes 0.
When both R and S are 1, the behavior is undefined.

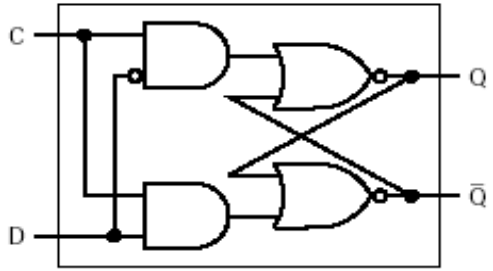
Flip-flops: D (delay)

By adding a clock input C and a control D,

we can set or reset the latch on a clock signal

Characteristic table for D (delay) flip-flop:

D	Q	Q^+	Operation
0	0	0	reset
0	1	0	reset
1	0	1	set
1	1	1	set



(Fig. B.13)

Note that the second column is actually output, but it is used to generate the next state.

The third column is the output at a later time.

When the clock value C is 0, then the output of both AND gates is 0, so there is no change.

(R and S inputs to NOR gates are 0)

When the clock value C is 1, then the output of the AND gates is:

D' for the first AND gate, and D for the second

This means that D acts as a set when it is 1 and a reset when it is 0.

Notice that this arrangement eliminates the possibility that both set and reset will be

1 at the same time.

Flip-flops: Toggle (T)

T (toggle) flip-flop holds the value of Q when input T is 0, and toggles when T is 1
Characteristic table for T (toggle) flip-flop:

T	Q	Q^+	Operation
0	0	0	hold
0	1	1	hold
1	0	1	toggle
1	1	0	toggle

Is this really XOR in disguise?

- **Inputs of XOR**
  - **2 operands**
- **Inputs of T flip-flop**
  - **toggle (T)**
  - **clock**
  - **current state (Q)**
- **Input Q and output Q^+ are really the same output, but at different times**
- **When output changes**
  - **Flip-flop can change only at positive clock edge**
  - **XOR can change whenever inputs change**

Flip-flops: JK

JK flip-flop has 2 control inputs

Characteristic table for T (toggle) flip-flop:

J	K	Q	Q^+	Operation
0	0	0	0	hold
0	0	1	1	hold
0	1	0	0	reset
0	1	1	0	reset
1	0	0	1	set
1	0	1	1	set
1	1	0	1	toggle
1	1	1	0	toggle

JK	operation
0	hold
1	reset
2	set
3	toggle

JK combines functions of D (reset/set) and T (hold/toggle)