# Floating Point Maths

## Addition

*Assume variable and number are declared double variables*

```
movsd xmm0, [variable]
addsd xmm0, [number]
movsd [answer], xmm0
```

Values are to be stored in the xmm[0-15] register after computation.

## Subtraction

```
movsd xmm0, [variable]
subsd xmm0, [number]
movsd [answer], xmm0
```

## Multiplication

```
; Input in rax register
mov rdx, rax
cvtsi2sd xmm0, rdx
mulsd xmm0, [floating_point]
cvtsd2si rax, xmm0
```

You need to convert the non decimal number into decimal representation. After computation, the number needs to be converted from a decimal representation into a regular integer. This process will loose precision.

## Division

```
; Input in the rax register
movsd xmm0, [answer]
divsd xmm0, [dividing_factor]
movsd [division_output], xmm0
```

## Introduction to Interrupts

- They cause the computer to pause what they are doing
- Can be software or hardware related
- A computer can handle a very large numbers of interrupts in a short amount of time
- Some of these interrupts are bad (segmentation fault)
- INT instruction is the syscall instruction

### Software Interrupts

- Page fault is an example
- **Program Exception:** when a software interrupt is not expected
- **SIGFAULT:** bad pointer

### Hardware Interrupts

- I/O Devices
- **Interval Timers:** provides a constant "tick" interrupt periodically. Another time is use to notify programs after a request interval time has concluded
- Other CPU cores

### More On Interrupts

- There are privilege levels
  - 0: root
  - 3: userland
- Interrupts run in this hierarchy
- **Interrupt Service Routine:** code that runs due to the interrupt
  - First-Level Interrupt Handler (FILH)
    * Saves the context, then handles the hardware requirements (resetting the hardware, saving information that may only be available at the time of the interrupt)
  - Second-Level Interrupt Handler (SLIH)
    * More specific to the interrupt (schedulinng the next I/O request to a storage device)
- **Interrupt Descriptor Table:** a table of ISRs

- When the interrupts happen, the RIP register loads the corresponding ISR address
- After the ISR is done running, the previous state of the processor must be restored to allow the computer to start where it left off
- **Polling:** the CPU keeps checking all the hardware of the availability of any request
  - Waiting for shit to happen
- **Interrupts:** like a doorbell or a notification that a task needs to be completed

## Flip Flops

The purpose of a flip flop is to preserve the data over a duration of time. Ones that are clocked will ignore all of the inputs given until the clock is signaled.

### Clocks

- Rising Edge: the transition from low to high
- Falling Edge: the transition from high to low

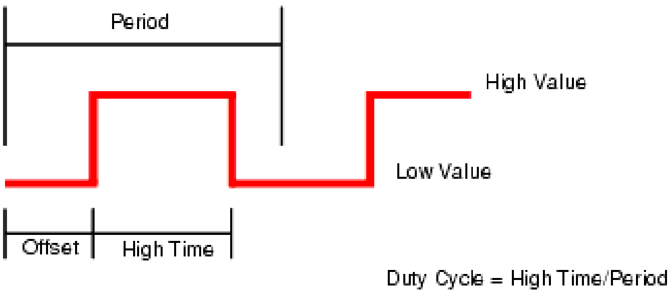



**Figure 1:** Clock Diagram

### Caveat

There is a problem called **metastability** which is when the clock and data line are changed at around the same time, the hardware has a hard time telling which one came first, so there may be undefined behavior that would incur.

## D Flip Flop

- When E is high (1), Q follows D with Q’ the complement of Q
- When E is low (0), the output remains the same (no state change) and D is ignored
- E: Enable
- D: Data
- Q Q’ : Output

D Flip-flop

Symbol

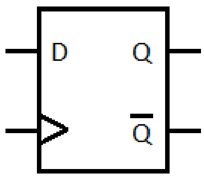



Table of truth:

| clk | D | Q | $\overline{Q}$ |
|---|---|---|---|
| 0 | 0 | Q | $\overline{Q}$ |
| 0 | 1 | Q | $\overline{Q}$ |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

**Figure 2:** D Flip Flop with Truth Table

### D Flip Flop (Clocked)

- Captures the value of the D input at a specific portion of the clock cycle (rising or falling edge)
- **Clock Cycle**: the clock line going high and then low again

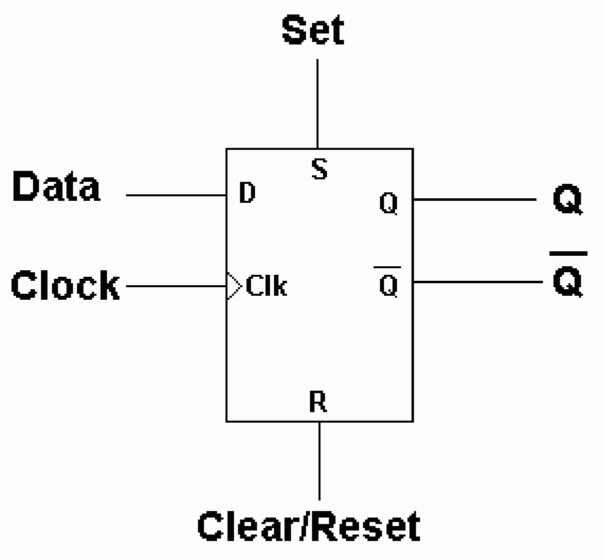



**Figure 3:** D Latch Diagram

### SR Flip Flop

- When both S and R are low, the outputs are in a constant state
- Q and Q’ are complementary; when Q is high, Q’ is low

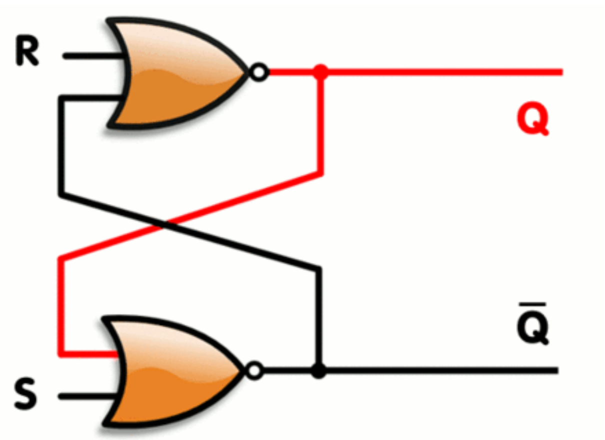


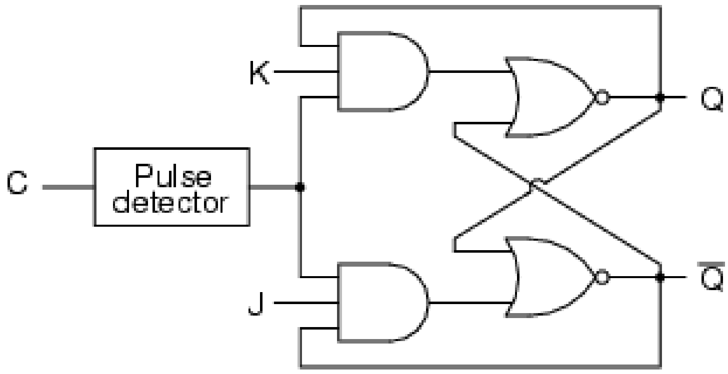

**Figure 4:** SR Latch

| INPUTS | | | OUTPUT | STATE |
|---|---|---|---|---|
| CLK | S | R | Q | |
| X | 0 | 0 | No Change | Previous |
| ↑ | 0 | 1 | 0 | Reset |
| ↑ | 1 | 0 | 1 | Set |
| ↑ | 1 | 1 | - | Forbidden |

**Figure 5:** SR Truth Table

### JK Flip Flop

- All state changes are synced to a clock point
  - When J is 1 and K is 0, on the next clock rising edge, Q will go high
  - When K is 1 and J is 0, on the next clock rising edge Q will go low
  - When J and K are both 0, nothing will happen when the clock is pulsed
  - If J and K are 1, no matter the state of Q, it will change to the opposite state (flipping)

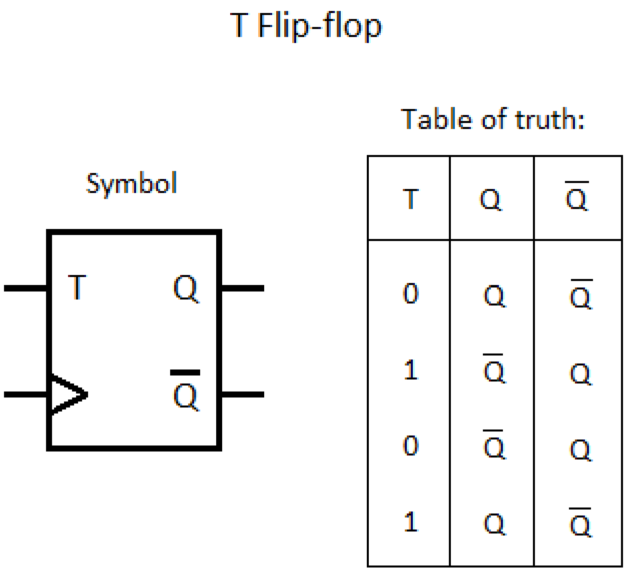



| C | J | K | Q | $\overline{Q}$ |
|---|---|---|---|---|
| ↑ | 0 | 0 | latch | latch |
| ↑ | 0 | 1 | 0 | 1 |
| ↑ | 1 | 0 | 1 | 0 |
| ↑ | 1 | 1 | toggle | toggle |
| x | 0 | 0 | latch | latch |
| x | 0 | 1 | latch | latch |
| x | 1 | 0 | latch | latch |
| x | 1 | 1 | latch | latch |

**Figure 6:** JK Flip Flop Diagram

## T Flip Flop

- When T is high, every clock cycle will toggle the outputs




| T | Q | $\overline{Q}$ |
|---|---|---|
| 0 | Q | $\overline{Q}$ |
| 1 | $\overline{Q}$ | Q |
| 0 | $\overline{Q}$ | Q |
| 1 | Q | $\overline{Q}$ |

**Figure 7:** T Flip Flop Diagram