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
```

Decoders

A circuit that changes a code into a set of signals and does the opposite job of an *encoder*.

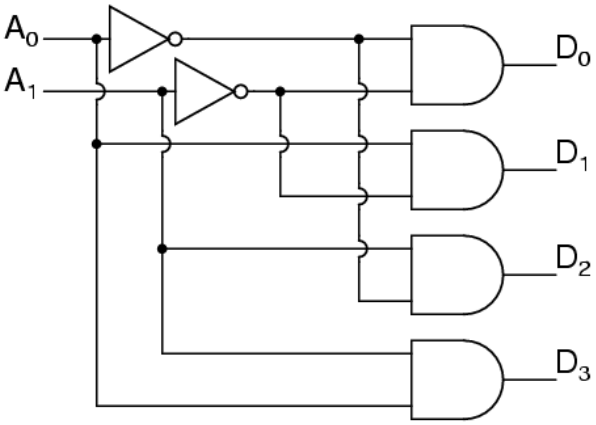


Figure 1: Decoder Example

Contents

Flip Flops **1**
Clocks 1
Caveat 1
D Flip Flop 2
D Flip Flop (Clocked) 3
SR Flip Flop 4
JK Flip Flop 5
T Flip Flop 6

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

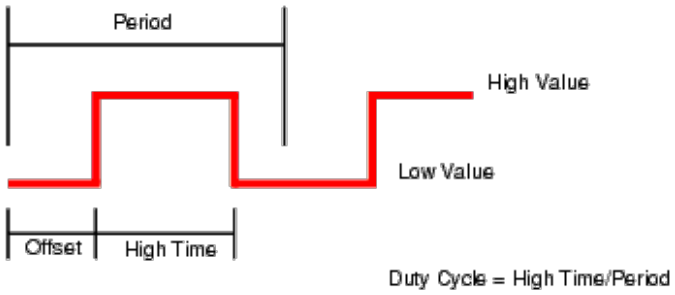


Figure 1: Clock Diagram

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

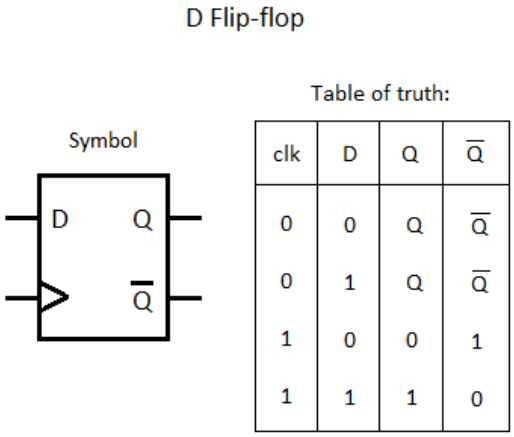


Figure 2: D Flip Flop with Truth Table

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

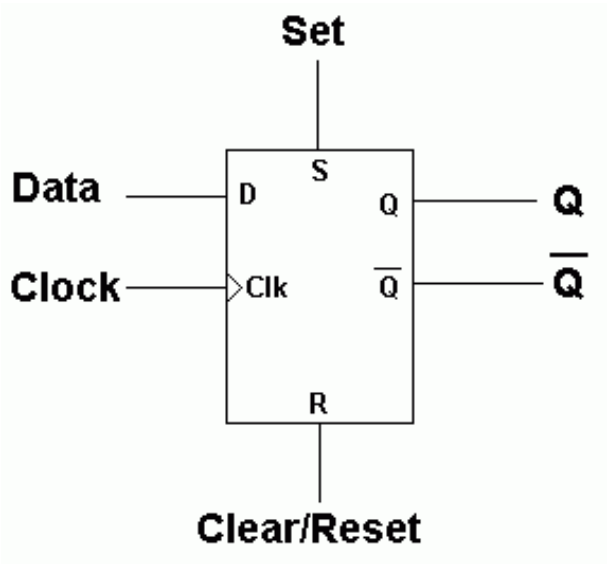


Figure 3: D Latch Diagram

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

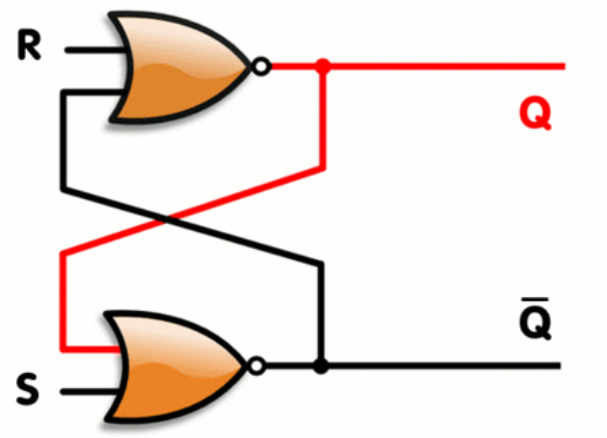


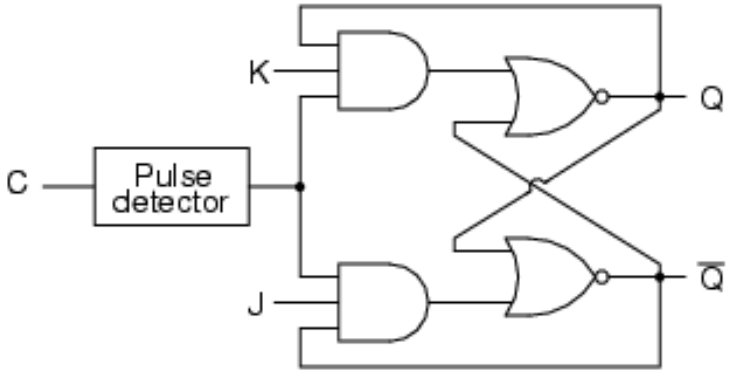
Figure 4: SR Latch

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

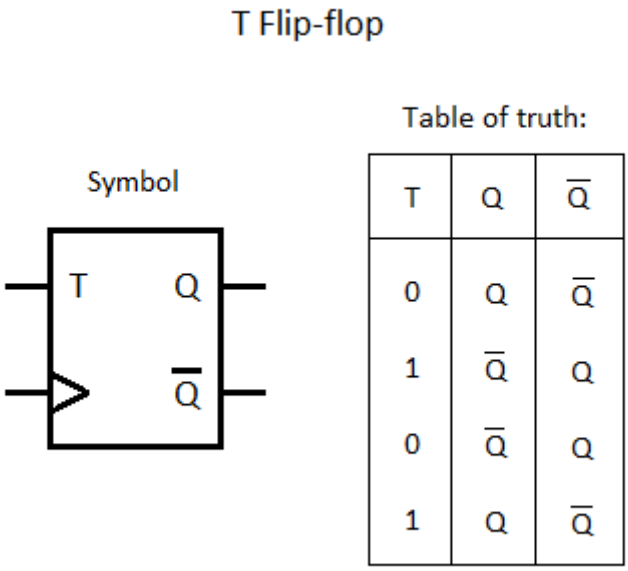


Figure 7: T Flip Flop Diagram

Contents

Introduction to Interrupts **1**

Software Interrupts 2

Hardware Interrupts 2

More On Interrupts 2

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

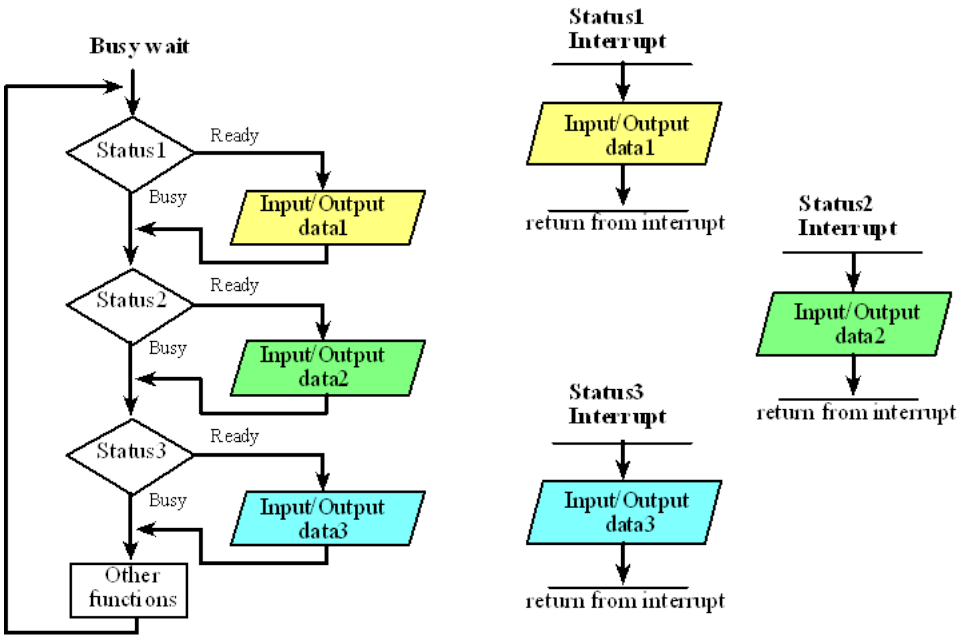


Figure 1: Interrupt Diagram

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

Parallel Processing

External Links

Multiprocessing vs Threading

Worksheet 5 - Logic Circuits

1. AND, OR, XOR, NOT
2. AND

INPUT	INPUT	OUTPUT
0	0	0
0	1	0
1	0	0
1	1	1

OR

INPUT	INPUT	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	0

XOR

INPUT	INPUT	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	0

NOT

True → False False → True

AB	CD	Output
0	0	0
0	1	1
1	0	1
1	1	1

3.
4. False. It uses a hardware description language or as a schematic.
5. True.
6. False. We have studied SR, D, JK, and T flip flops.
7. True.
8. True. *add this to our notes of flip flops*
9. True.
10. False. The inputs for a SR flip flop are S and R
11. True,
12. False. There is only one input and the clock
13. True.
14. False. It will go high.
15. True.

External Links

Conversion of Flip Flops

Contents

Worksheet 6 **1**

Worksheet 6

1. SIGFAULT: a type of page fault but usually happens when you have a bad pointer. It is also a software interrupt and is an exception.
2. divide by zero, which is an example of a program exception.
3. Typing on the keyboard, moving the mouse pointer, pressing the track pad. Anything that takes away the attention of the processor's main tasks
4. Operating system code runs at 0, which in Linux is considered to be root and is stored in $EUID
5. ISR: Interrupt Service Routine
6. The processor saves the current RIP register, the loads the appropriate ISR address into the RIP register to begin executing the ISR
7. Context
8. Difference:
    - FILH (first level interrupt handler): saves the context of the processor (state of things happening in the processor when the interrupt is issued)
    - SILH (second level interrupt handler): finds what type of interrupt, such as a disk drive interrupt needing to schedule another type of request (I/O). Consults the IDT.
9. Polling: when the CPU is waiting and checking for all the hardware of the availability of any request. Spends wasted time waiting/checking for something to happen.
10. System Timer: provides a constant "tick" interrupt periodically
11. It would be very taxing on the CPU if there are many interrupts a second, it would be better to toggle between using polling and other methods rather than constantly interrupting. Network devices use polling to handle all the overhead of many packets transmitted to the computer.

Contents

Worksheet 7 : Parallel Processing **1**

External Links **3**

Worksheet 7 : Parallel Processing

> Jared Dyreson

> CPSC-240 09 @ 11:30 - 13:20 (TR)

> May 2, 2019

1. Mutex: a mutual exclusion object is a program object that is created so that multiple program threads can take turns sharing the same resource, such as access to a file
2. Deadlock: a situation where two programs sharing the same resource are effectively preventing each other from accessing the resource, resulting in both programs to cease functionality
3. Race Condition: the system's substantive behavior is dependent on the sequence or timing of other uncontrollable events
4. Lock contention: a condition where one thread is waiting for lock/object that is currently being held by another thread. Therefore, this waiting thread cannot use that object until the other thread has unlocked that particular object
5. Moore's Law: the principle that the speed and capability of computers can be expected to double every two years, because of the increase of transistors on a given chip
6. Two approaches to parallel processing
    - Multiple processors working in tandem with each other to conquer a task
    - Lock and Unlocking processes through the use of mutexes and semaphores
7. All the threads will have their own stack as it would would possibly create a race condition and break program flow. However, all the threads share memory with the heap, as it is meant to be dynamically allocated.

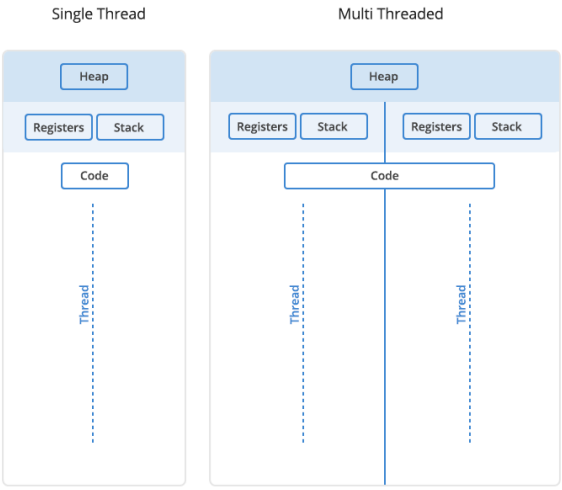


Figure 1: Thread memory

8. The mutex/semaphore protect the current process running so another process does not interrupt it prematurely
9. Atomic Operation: program operations that run completely independently of any other processes
10. Since a thread will get its own set of general purpose registers, it is accessing them through the CPU cache which contains “second class” registers and are blazing fast.

External Links

How threads get their own registers