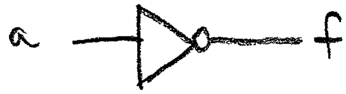
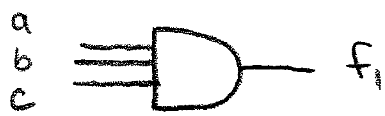
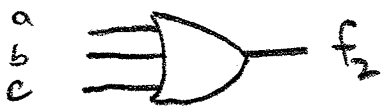
Digital Logic (256 Review)

Basic logic gates



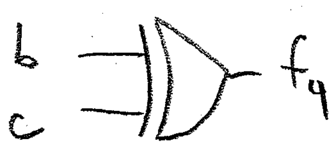
a	f
0	1
1	0



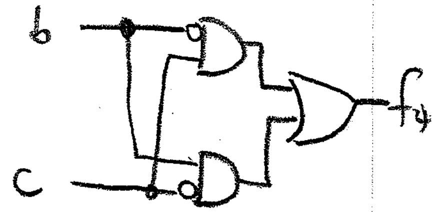


a	b	c	f_1	f_2	f_3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	0	1	1
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	1	1	1

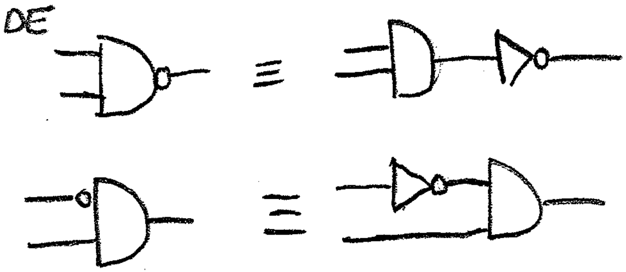
f_4



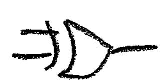
$$f_4 = \bar{b}c + b\bar{c}$$



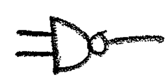
ASIDE



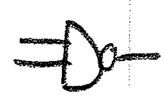
• can also build

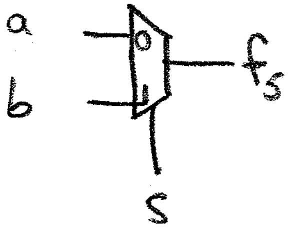


using just

4 x 

In fact,  is a universal logic gate.

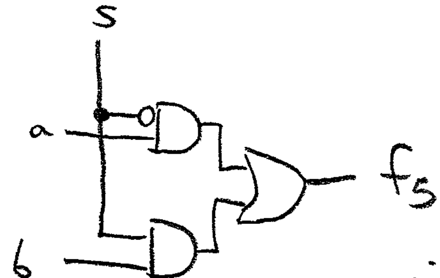
You can build <u>any</u> digital system using just 



s	a	b	f_5
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

if $s = 1$
  $f_5 = b$
else
  $f_5 = a$

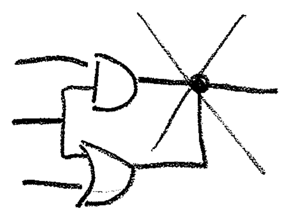
$$f_5 = \bar{s}a + sb$$



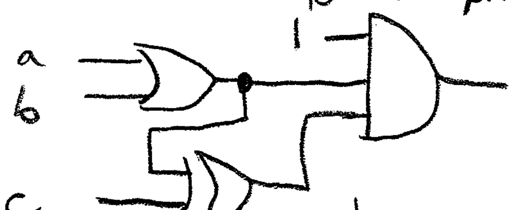
very similar to XOR!

<u>Some Logic Rules</u>: inputs + outputs are <u>directional</u> + have rules

① never connect 2 outputs together

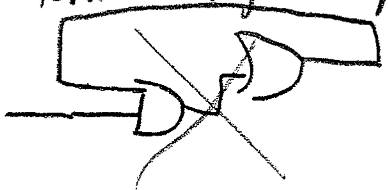


② make sure every input to a logic gate is connected
- to a 0 or 1 (constant)
- to the output of just 1 other gate
- to a "primary input" from outside world



note: an output can connect to multiple inputs

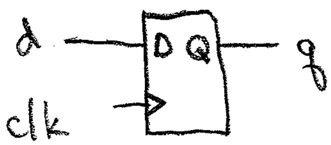
③ never form a logic loop



④ never use a mux backwards

Digital Logic Review (cont'd)

D flip flop - positive edge triggered

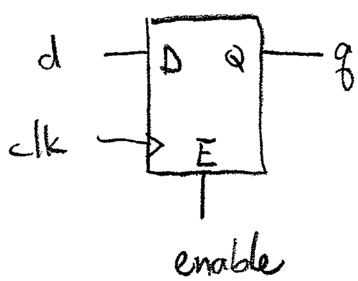


clk	d	q_{n+1}
0	x	q_n
1	x	q_n
↑	0	0
↑	1	1
↓	x	q_n

↑ 0 → 0, ↑ 1 → 1: captures d

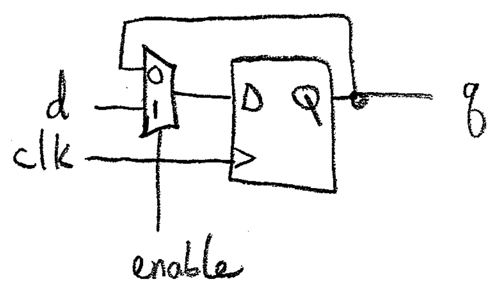
$$q_{n+1} = \begin{cases} d & \text{on } \uparrow clk \\ q_n & \text{otherwise} \end{cases}$$

DFF + enable input

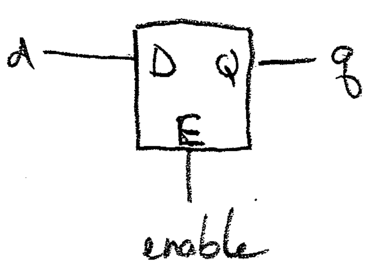


$$q_{n+1} = \begin{cases} d & \text{on } \uparrow clk \ \underline{\text{only if}} \ E=1 \\ q_n & \text{otherwise} \end{cases}$$

≡



Level-Sensitive Latch aka Flow-through Latch



if enable = 1
    $q_{n+1} = d$
else
    $q_{n+1} = q_n$
end if

repeats continuously

captures last d while enable = 1

Binary Numbers

EECE259: Introduction to Microcomputers

Prof. Guy Lemieux

Lecture 2

Binary Numbers

- Humans
  - Powers of 10: 1, 10, 100, 1000, …
  - Example: $2113 = 2113_{10}$
- Computers
  - Powers of 2: 1, 2, 4, 8, …
  - Example: 2113 = 2048+64+1
    - = $1000\ 0100\ 0001_2$
    - = %1000 0100 0001

Binary Numbers

- Why?
  - Humans have 10 fingers
    - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
    - Fingers are called digits!
  - Computer switches have 2 states: on, off
    - 0, 1
    - Called **binary digits**, or “bits” for short

Common Number Bases

- Base 2: binary — 0,1
  - **%0100** or **0b0100** or $\mathbf{0100_2}$
- Base 10: decimal — 0 to 9
  - **2113** or $\mathbf{2113_{10}}$
- Base 8: octal — 0 to 7
  - 8 digits
  - **04101** or **@4101** or $\mathbf{4101_8}$
- Base 16: hexadecimal — 0 to 9, A to F
  - 16 digits, also known as “**hex**”
  - **$841** or **0x841** or $\mathbf{841_{16}}$

Computer Number Storage

- Computers use binary **exclusively**
  - Everything stored as sequence of 0s and 1s
    - 01100101101000011011011110001001 10…
  - Grouped into…

Nibble	4 bits	1 hex digit
Byte	8 bits	2 hex digits
Halfword / short / word	16 bits	4 hex digits
Word / long	32 bits	8 hex digits
Long / long long	64 bits	16 hex digits

Number Format

- In any base, each digit has a **rank**
- Example
  - $2113_{10} = 2*10^3 + 1*10^2 + 1*10^1 + 3*10^0$

Notice the increasing exponents or **ranks**

Important Ranks

- Important Definitions
  - MSB   Most Significant Bit (leftmost digit)
    - Always “bit 7 of byte”, “bit 31 of word”
  - LSB   Least Significant Bit (rightmost digit)
    - Always “bit 0”
  - Example

MSB   LSB

%1001 0111

bit 7   bit 1   bit 0

Base Conversions

- Converting Binary to Decimal is easy
  - Just expand digits with ranks, and add
- Example

1001_2 or %1001 to decimal

%1001 $= 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0$

$= 1*8 + 0*4 + 0*2 + 1*1$

$= 8 + 0 + 0 + 1$

$= 9$

$= 9_{10}$

Base Conversions

- Example

%1001 0111 to decimal

$= 1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 1*2^0$

$= 1*128 + 0*64 + 0*32 + 1*16 + 0*8 + 1*4 + 1*2 + 1*1$

$= 128 + 16 + 4 + 2 + 1$

$= 144 + 7$

$= 151$

Base Conversions

- Leading 0s don’t change value
  - Example

%101 = 4 + 0*2 + 1 = 5

%0101 = 0*8 + 4 + 0*2 + 1 = 5

- We often drop leading 0s in written form
  - Computers cannot!
  - Must store all 8 bits of byte, 32 bits of word

Base Conversions

- Converting Decimal to Binary a bit harder
  - Expand value using powers of 2
- Example

$53_{10} = 32 + 16 + 4 + 1$

$= 32 + 16 + 0*8 + 4 + 0*2 + 1$

= %110101

Base Conversions

- Converting Decimal to Binary
  - Easier way: divide by 2 approach (produces LSB first)

- 53   odd?   Y = 1   ← LSB
- 53/2 = 26   odd?   N = 0
- 26/2 = 13   odd?   Y = 1   = %110101
- 13/2 = 6   odd?   N = 0
- 6/2 = 3   odd?   Y = 1
- 3/2 = 1   odd?   Y = 1   ← MSB
- 1/2 = **0**   **stop**

How do you modify this to convert decimal ➔ hexadecimal ?

Memorize These!

0	**1**
1	**2**
2	**4**
3	**8**
4	**16**
5	**32**
6	**64**
7	**128**

8	**256**
9	512
10	**1024**
11	2048
12	**4096**
13	8192
14	**16384**

16	**65,536**
20	1,048,576 Mega
24	16,771,216 “16 Meg”
30	1,073,741,824 Giga