ELE 455/555
Computer System Architecture

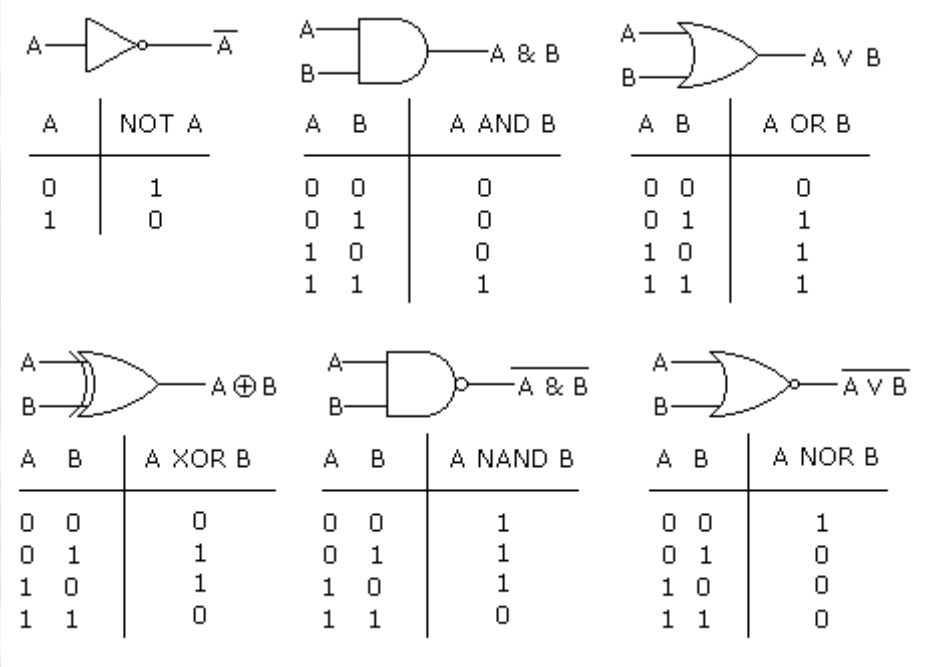
Section 1 – Review and Foundations

Class 2 – Logic and Arithmetic


HOW'D YOU GET THE BLACK EYE?
I WAS PULLING UP MY BLANKET IN BED. MY HAND SLIPPED AND I PUNCHED MYSELF IN THE FACE.
OKAY, LET'S MAKE SOME BILLION-DOLLAR TECHNOLOGY DECISIONS.
Dilbert.com DilbertCartoonist@gmail.com
8-27-13 ©2013 Scott Adams, Inc. /Dist. by Universal Uclick

Logic Review

Simple Gates



Inverter: A → A
A'
A not

OR: A,B → A+B
AvB
A∪B
AorB

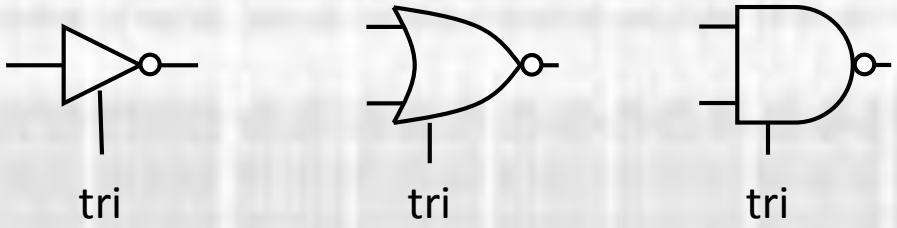
AND: A,B → AB
A^B
A∩B
AandB

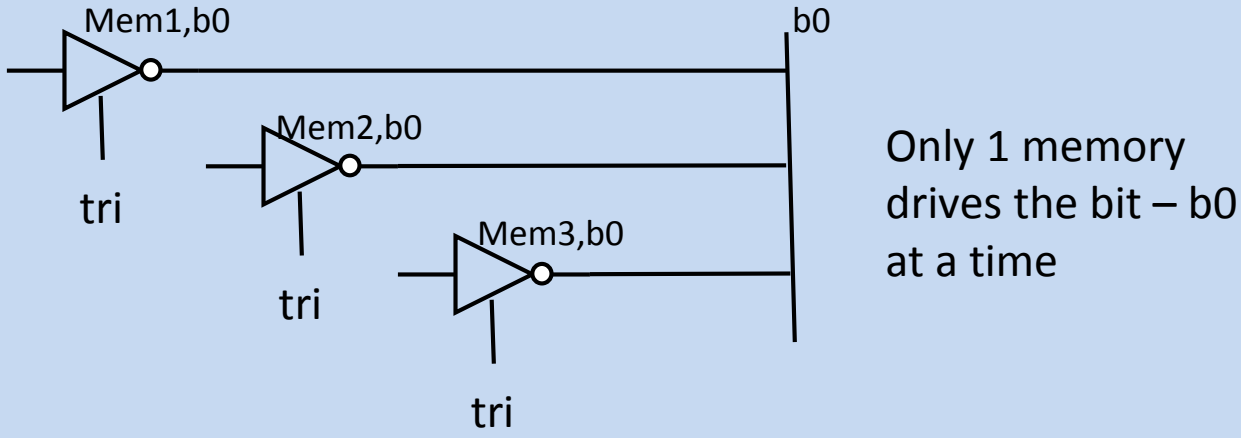
XOR: A,B → A⊗B
AxorB

Logic Review

Simple Gates

- Tri-state gates
  - when the tristate input is high – the outputs are disabled (floating)

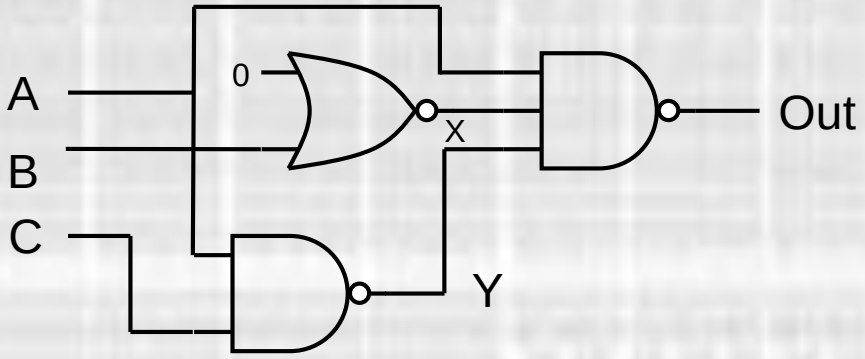




Only 1 memory drives the bit – b0 at a time

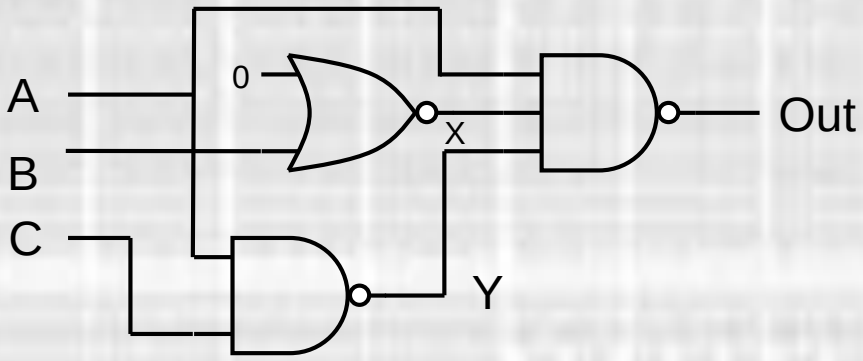
Logic Review

Simple Gates



Logic Review

Simple Gates



A	B	C	X	Y	Out
			(B+0)'	(AC)'	(AXY)'
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	0	1	1
0	1	1	0	1	1
1	0	0	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	0	0	1

Logic Review

Simple Gates

- de Morgan's Laws

$$(A \cup B)' = A' \cap B' \qquad \overline{(A + B)} = \overline{A}\,\overline{B}$$

$$(A \cap B)' = A' \cup B' \qquad \overline{(AB)} = \overline{A} + \overline{B}$$

$$(A')' = A \qquad \overline{(\overline{A})} = A$$

Logic Review

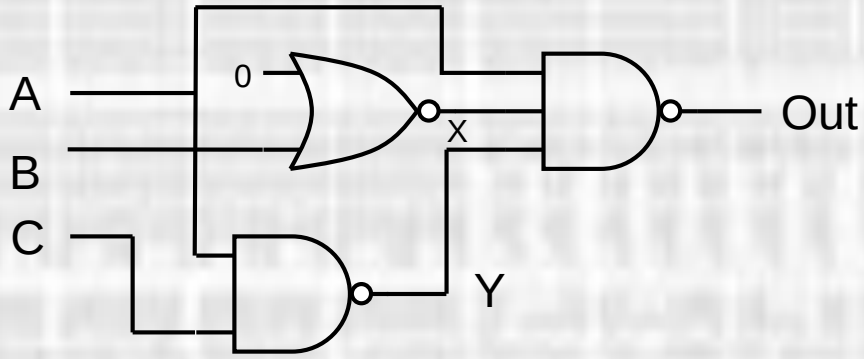
Simple Gates

- de Morgan's Laws

$$(A \cup B)' = A' \cap B' \qquad \overline{(A + B)} = \overline{A}\,\overline{B}$$

$$(A \cap B)' = A' \cup B' \qquad \overline{(AB)} = \overline{A} + \overline{B}$$

$$(A')' = A \qquad \overline{(\overline{A})} = A$$



Logic Review

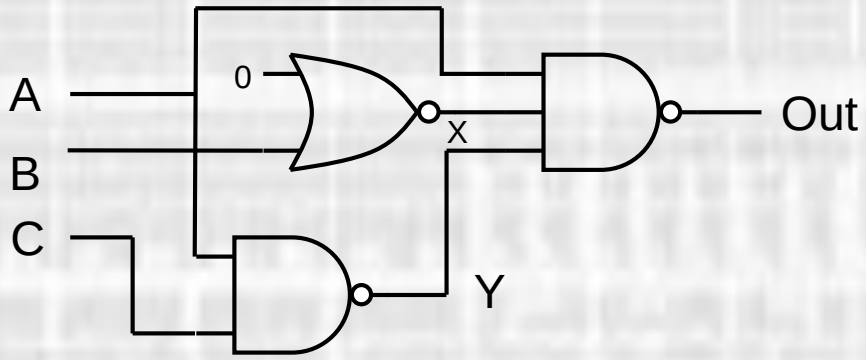
Simple Gates

- de Morgan's Laws

$$(A \cup B)' = A' \cap B' \qquad \overline{(A + B)} = \overline{A}\,\overline{B}$$

$$(A \cap B)' = A' \cup B' \qquad \overline{(AB)} = \overline{A} + \overline{B}$$

$$(A')' = A \qquad \overline{(\overline{A})} = A$$



Out = $\overline{AXY}$

$\overline{A(\overline{B+0})(\overline{AC})}$

$\overline{A} + (B+0) + (AC)$

$\overline{A} + B + AC$

$\overline{A} + B + AC$

Logic Review

Simple Gates

- Sum-of-Products, Product-of-Sums

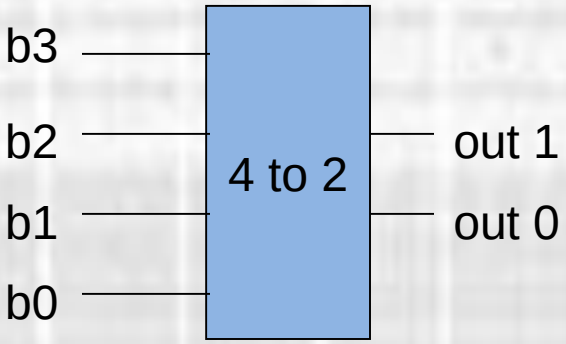
Sum-of-Products	$(A \cap B) \cup (C \cap D) \cup (E \cap F)$	$AB + CD + EF$
Product-of-Sums	$(A \cup B) \cap (C \cup D) \cap (E \cup F)$	$(A + B)(C + D)(E + F)$

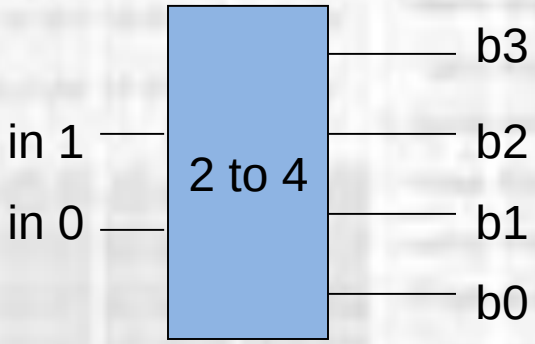
Why ???

Logic Review

Simple Gates

- Coder / Decoder





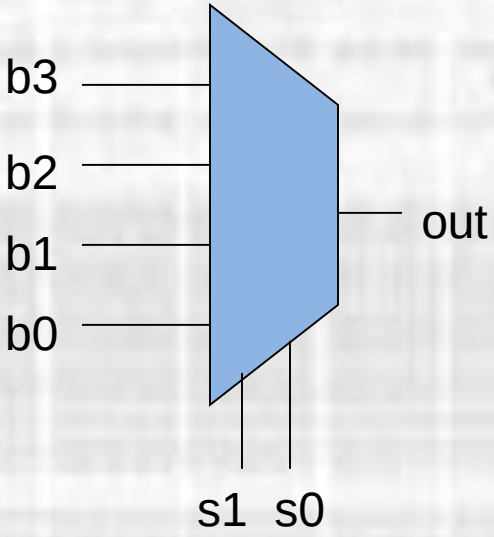
b3	b2	b1	b0	out 1	out 0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

In 1	In 2	b3	b2	b1	b0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

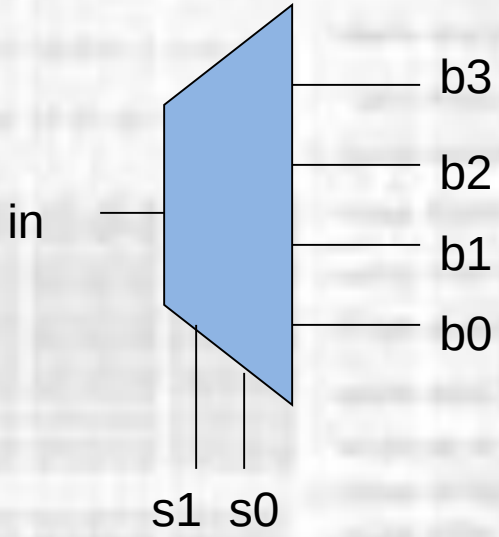
Logic Review

Simple Gates

- Multiplexor / Demultiplexor



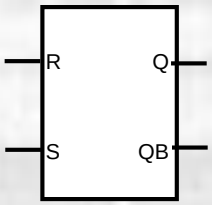
s1	s0	out
0	0	b0
0	1	b1
1	0	b2
1	1	b3

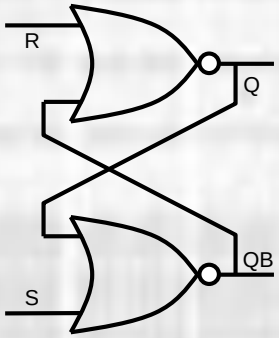


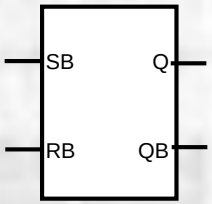
s1	s0	b3	b2	b1	b0
0	0	0	0	0	in
0	1	0	0	in	0
1	0	0	in	0	0
1	1	in	0	0	0

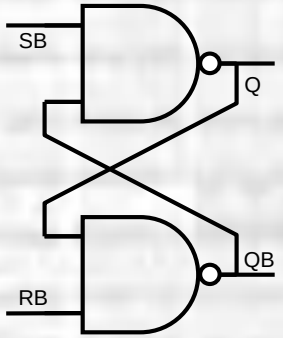
Logic Review
Complex Gates

- Latches



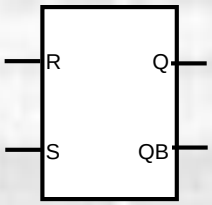


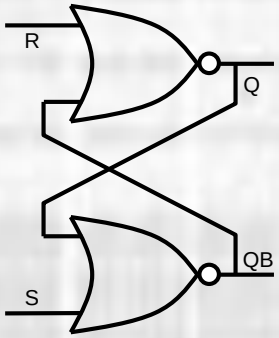


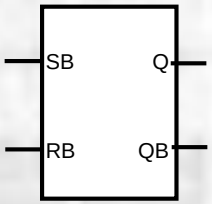


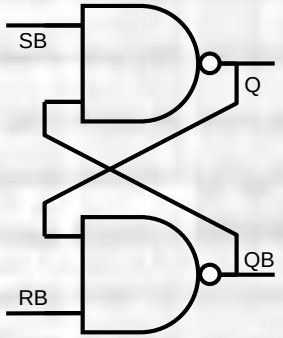
Logic Review
Complex Gates

- Latches







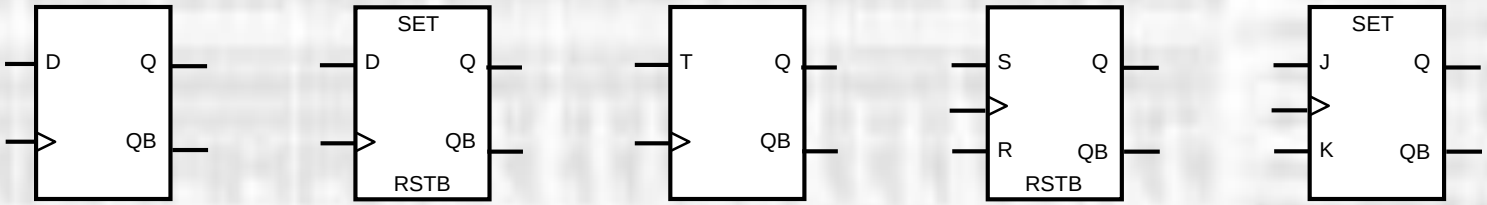


S	R	Q	QB
0	0	No change	No change
0	1	0	1
1	0	1	0
1	1	**0**	**0**

SB	RB	Q	QB
0	0	**1**	**1**
0	1	1	0
1	0	0	1
1	1	No change	No change

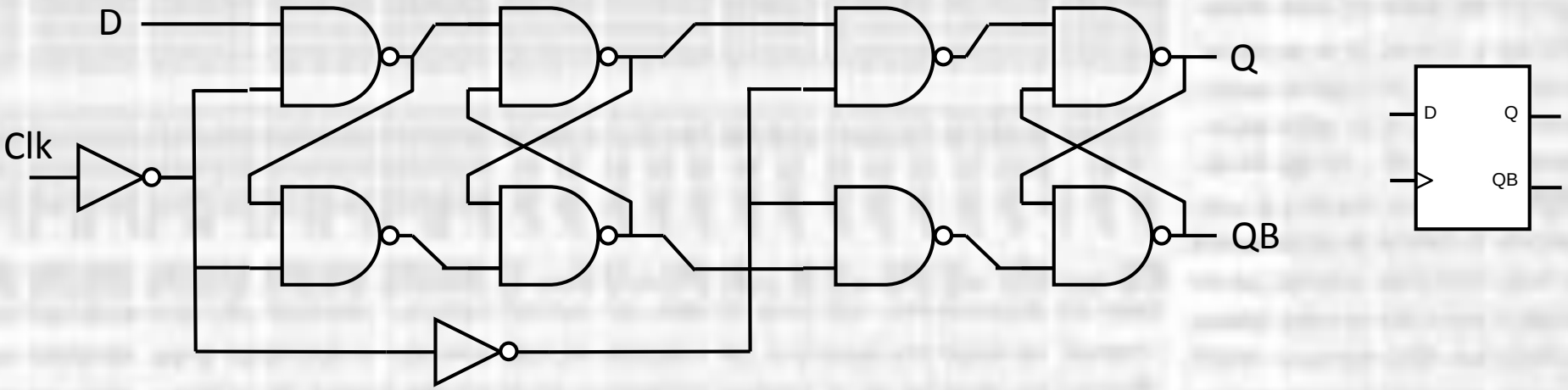
Logic Review
Complex Gates

- Flip-Flops
  - Edge Triggered
    - Outputs only change on a rising or falling clock edge (synchronous)
    - Outputs depend on the state of the inputs at the clock edge
  - Some have asynchronous set or reset inputs



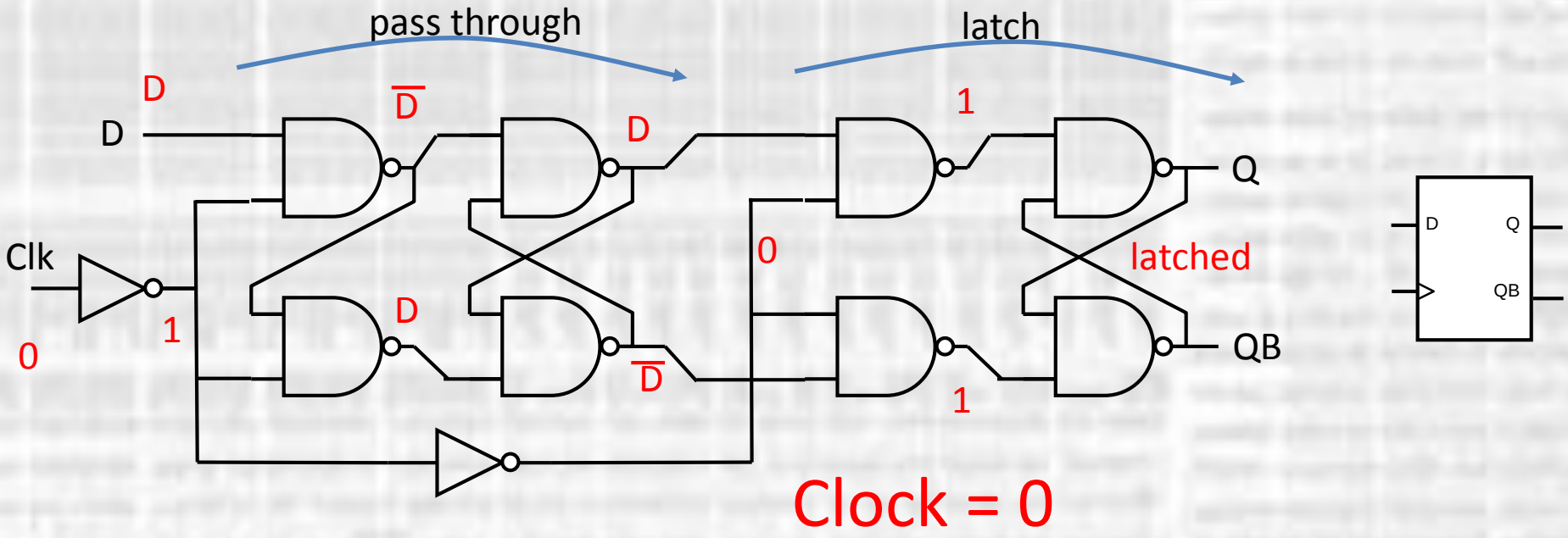
Logic Review
Complex Gates

- Flip-Flops
  - D - Type
    - Data Flip-Flop
    - On the rising clock edge, the D-input value is transferred to the Q output



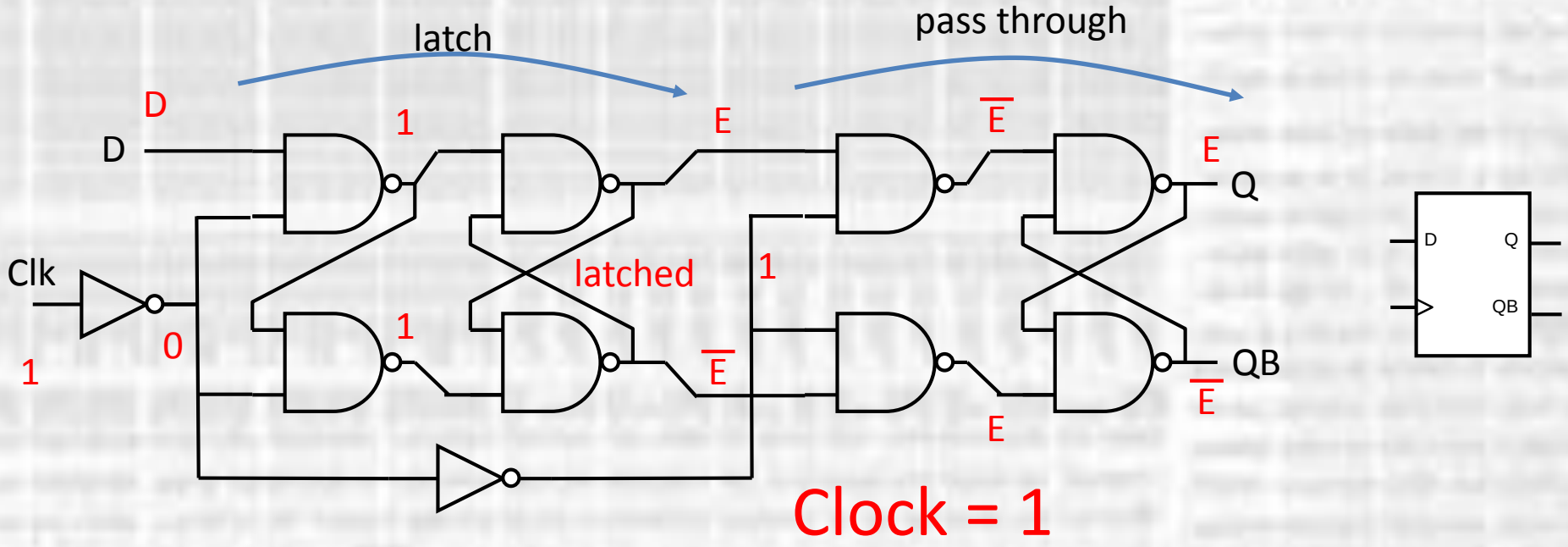
Logic Review
Complex Gates

- Flip-Flops
  - D - Type
    - Data Flip-Flop
    - On the rising clock edge, the D-input value is transferred to the Q output



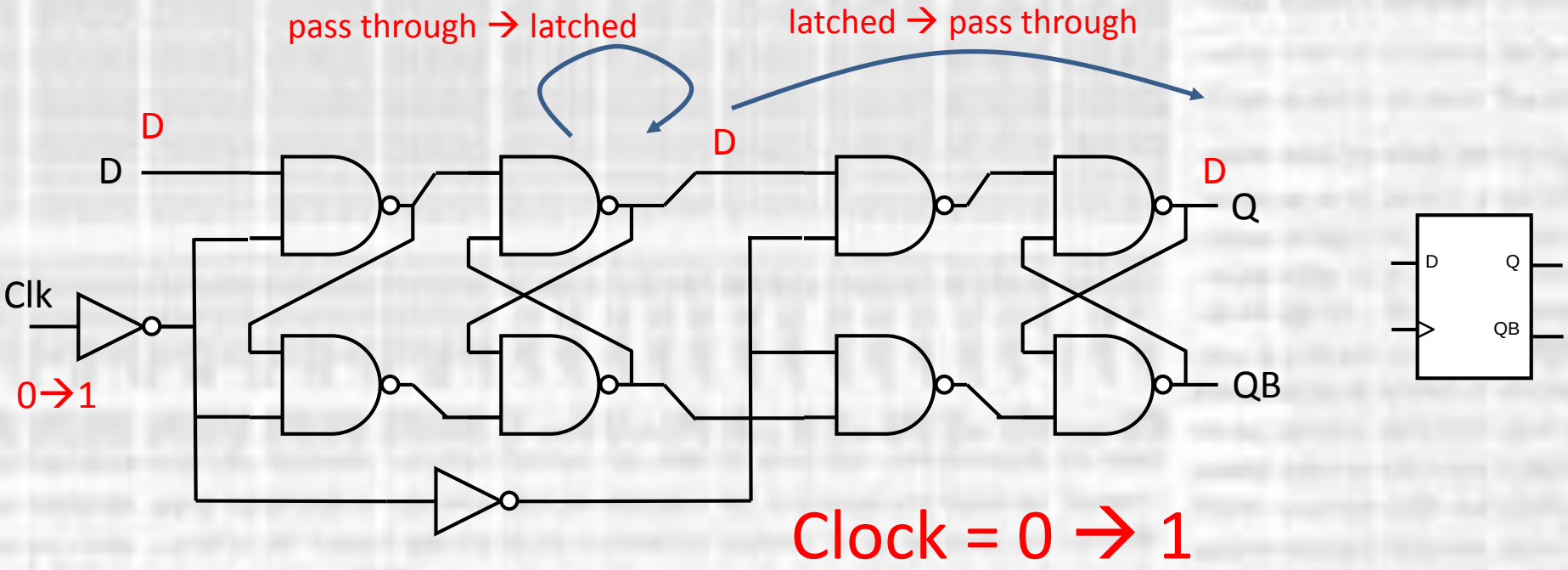
Logic Review
Complex Gates

- Flip-Flops
  - D - Type
    - Data Flip-Flop
    - On the rising clock edge, the D-input value is transferred to the Q output



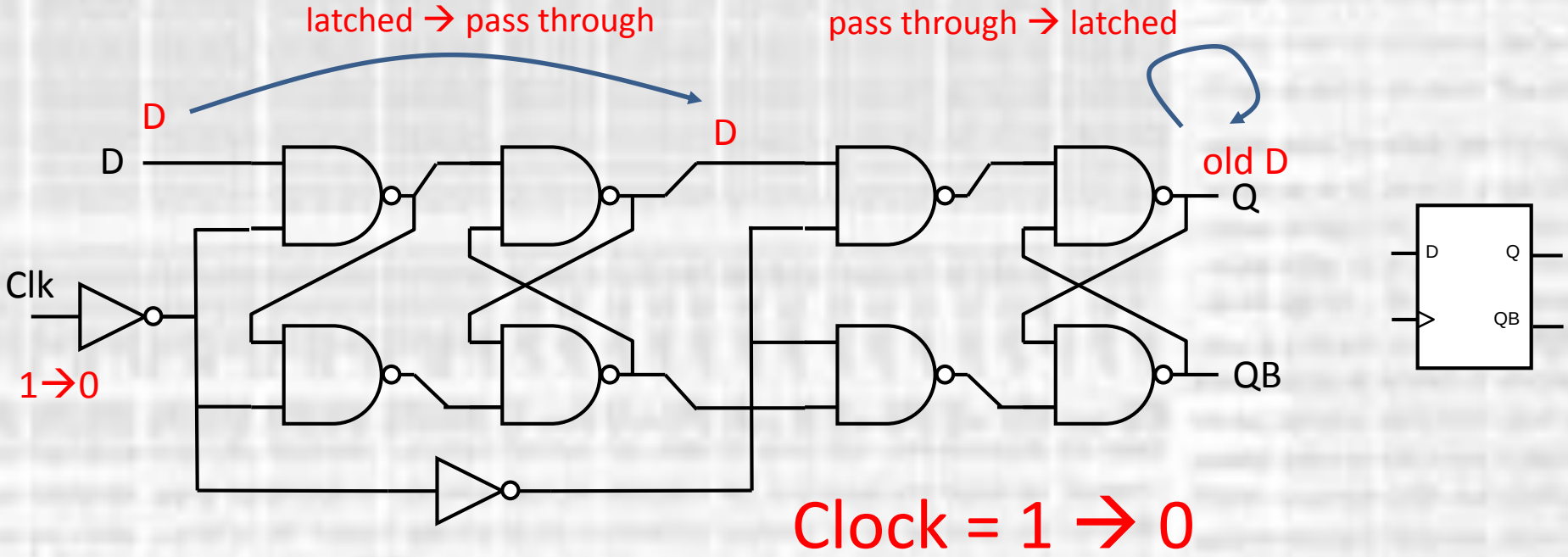
Logic Review
Complex Gates

- Flip-Flops
  - D - Type
    - Data Flip-Flop
    - On the rising clock edge, the D-input value is transferred to the Q output



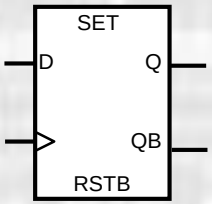
Logic Review
Complex Gates

- Flip-Flops
  - D - Type
    - Data Flip-Flop
    - On the rising clock edge, the D-input value is transferred to the Q output



Logic Review
Complex Gates

- Flip-Flops
  - Edge Triggered D Flip-Flop
    - Outputs depend on the state of the inputs at the rising clock edge
  - Some have asynchronous set or reset inputs

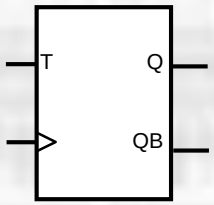


Set	Reset Bar	D		Q	QB
0	1	**0**		0	1
0	1	**1**		1	0
1	1	x		1	0
0	**0**	x		0	1

at the rising clock edge

Logic Review
Complex Gates

- Flip-Flops
  - Edge Triggered T Flip-Flop (Toggle)
    - Outputs depend on the state of the inputs at the rising clock edge
  - Some have asynchronous set or reset inputs



T	Q	QB
0	Q_{OLD}	QB_{OLD}
1	QB_{OLD}	Q_{OLD}

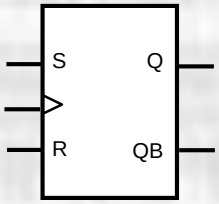
T at the rising clock edge

T	Q	QB
0	No Change	No Change
1	Toggle	Toggle

T at the rising clock edge

Logic Review
Complex Gates

- Flip-Flops
  - Edge Triggered SR Flip-Flop (Set/Reset)
    - Outputs depend on the state of the inputs at the rising clock edge
  - Some have asynchronous set or reset inputs

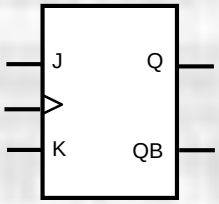


R	S		Q	QB
0	0		Q_{OLD}	QB_{OLD}
0	1		1	0
1	0		0	1
1	1		N/A	N/A

S,R at the rising clock edge

Logic Review
Complex Gates

- Flip-Flops
  - Edge Triggered JK Flip-Flop
    - Outputs depend on the state of the inputs at the rising clock edge
  - Some have asynchronous set or reset inputs



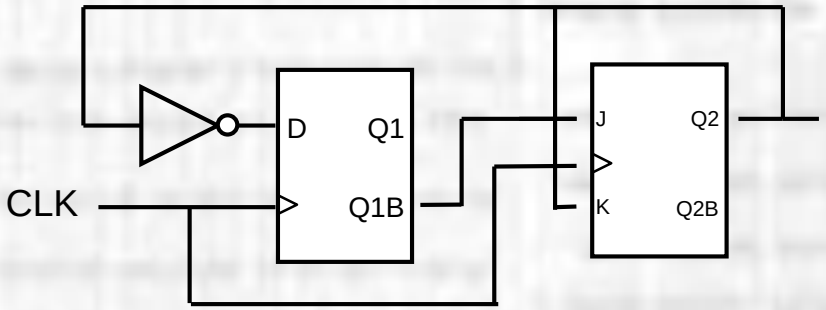
J	K		Q	QB
0	**0**		Q_{OLD}	QB_{OLD}
0	**1**		0	1
1	**0**		1	0
1	**1**		Toggle	Toggle

at the rising clock edge

Logic Review

Complex Gates

- Flip-Flops - Example



Initial State

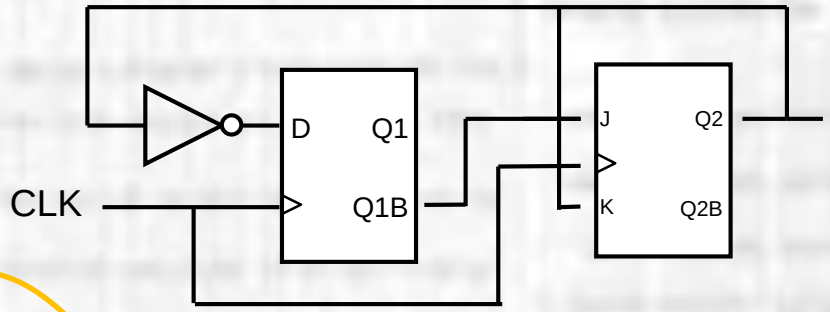
	D	J	K	Q1	Q1B	Q2	Q2B
Initial							
				1			0
After clk1							
After clk2							
After clk3							

You must know these to analyze the circuit

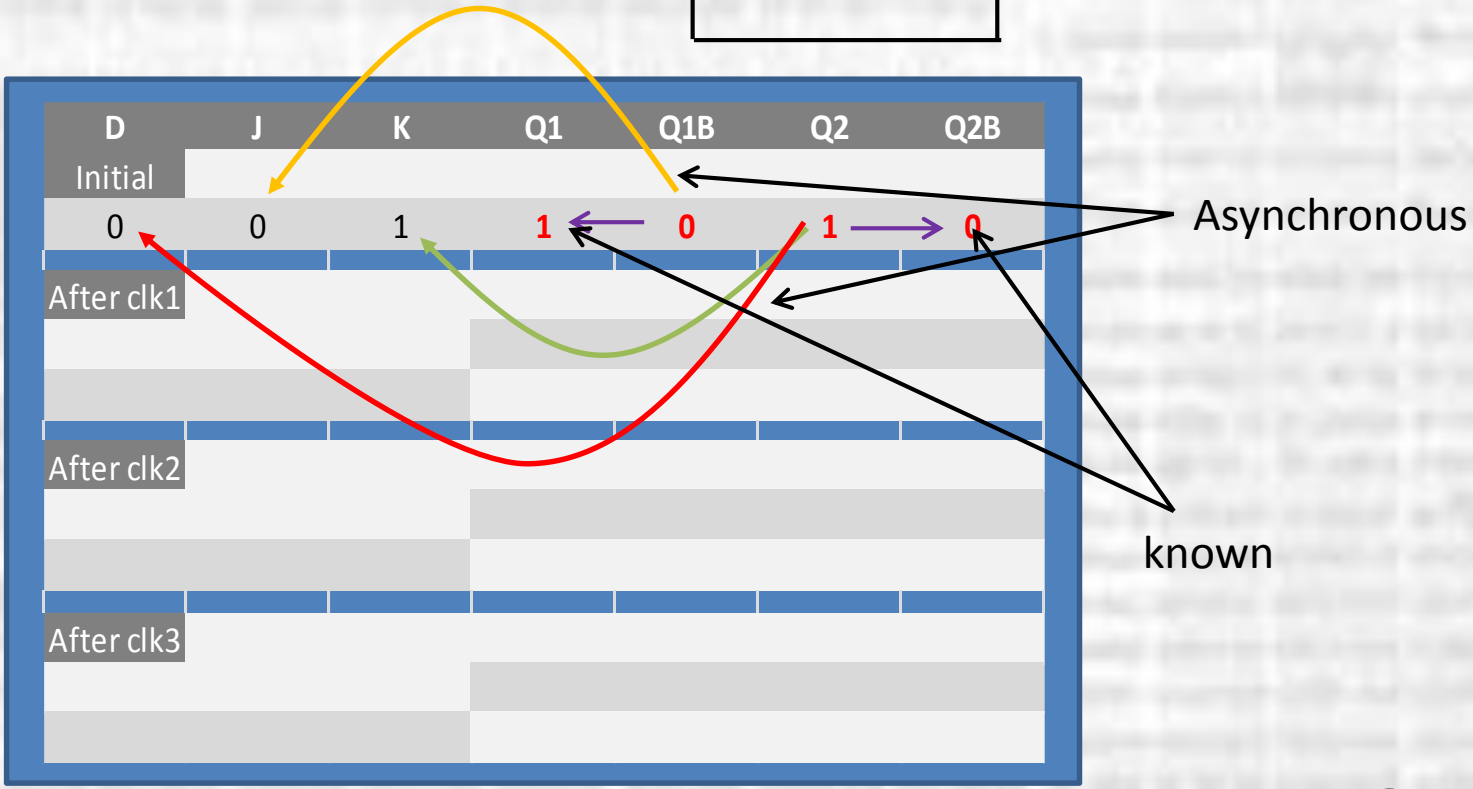
Logic Review

Complex Gates

- Flip-Flops - Example



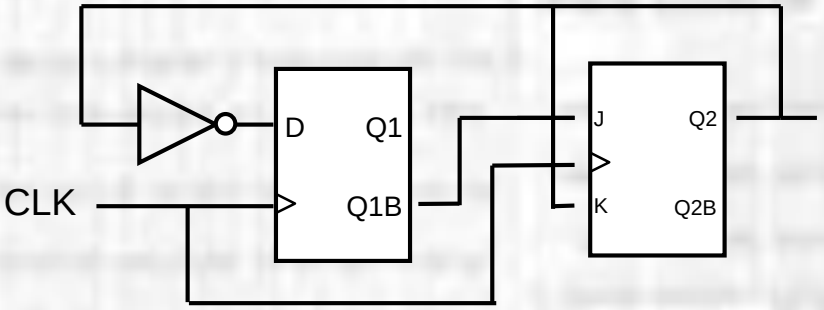
Initial State



Logic Review

Complex Gates

- Flip-Flops - Example



Clk

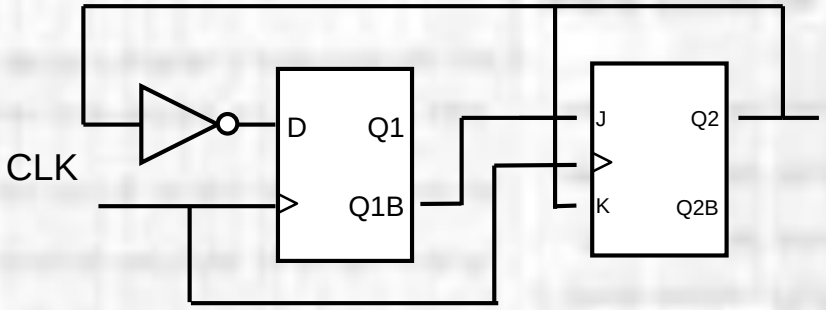
Synchronous Propagation

D	J	K	Q1	Q1B	Q2	Q2B
Initial						
0	0	1	1	0	1	0
After clk1						
			0	1	0	1
After clk2						
After clk3						

Logic Review

Complex Gates

- Flip-Flops - Example



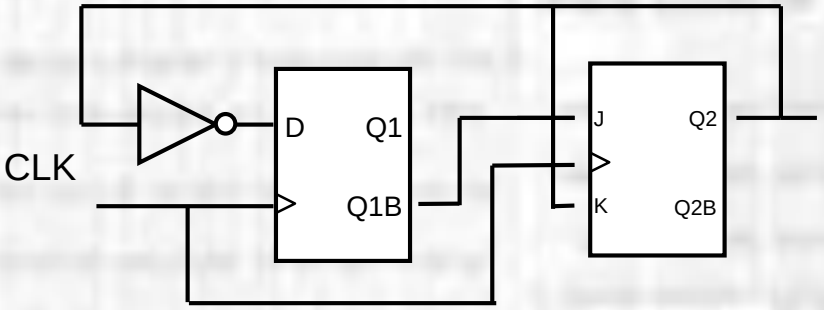
Asynchronous Propagation

D	J	K	Q1	Q1B	Q2	Q2B
Initial						
0	0	1	1	0	1	0
After clk1						
			0	1	0	1
1	1	0				
After clk2						
After clk3						

Logic Review

Complex Gates

- Flip-Flops - Example



Clk

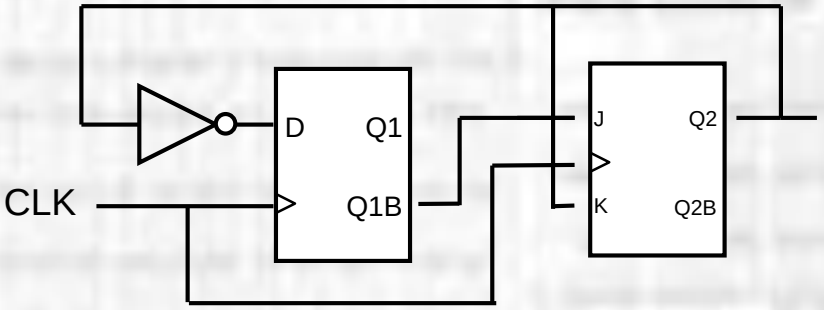
Synchronous Propagation

D	J	K	Q1	Q1B	Q2	Q2B
Initial						
0	0	1	1	0	1	0
After clk1						
			0	1	0	1
1	1	0				
After clk2						
			1	0	1	0
After clk3						

Logic Review

Complex Gates

- Flip-Flops - Example



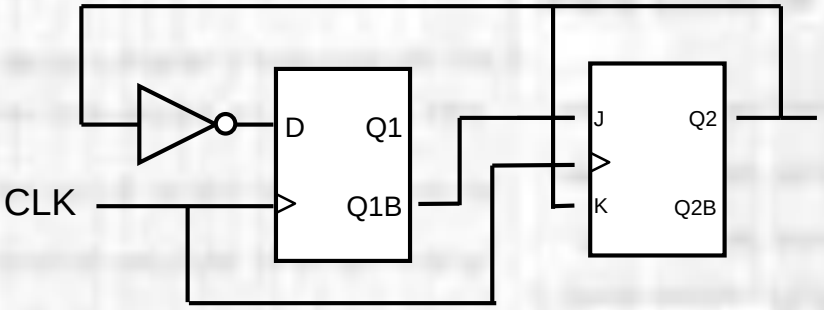
Asynchronous Propagation

D	J	K	Q1	Q1B	Q2	Q2B
Initial						
0	0	1	1	0	1	0
After clk1						
			0	1	0	1
1	1	0				
After clk2						
			1	0	1	0
0	0	1				
After clk3						

Logic Review

Complex Gates

- Flip-Flops - Example



Clk

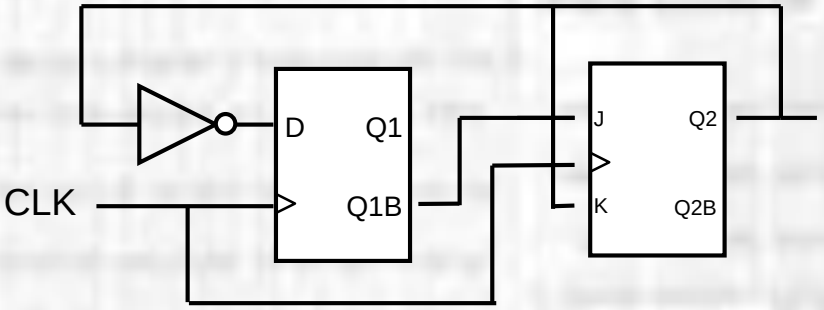
Synchronous Propagation

D	J	K	Q1	Q1B	Q2	Q2B
Initial						
0	0	1	1	0	1	0
After clk1						
			0	1	0	1
1	1	0				
After clk2						
			1	0	1	0
0	0	1				
After clk3						
			0	1	0	1

Logic Review

Complex Gates

- Flip-Flops - Example



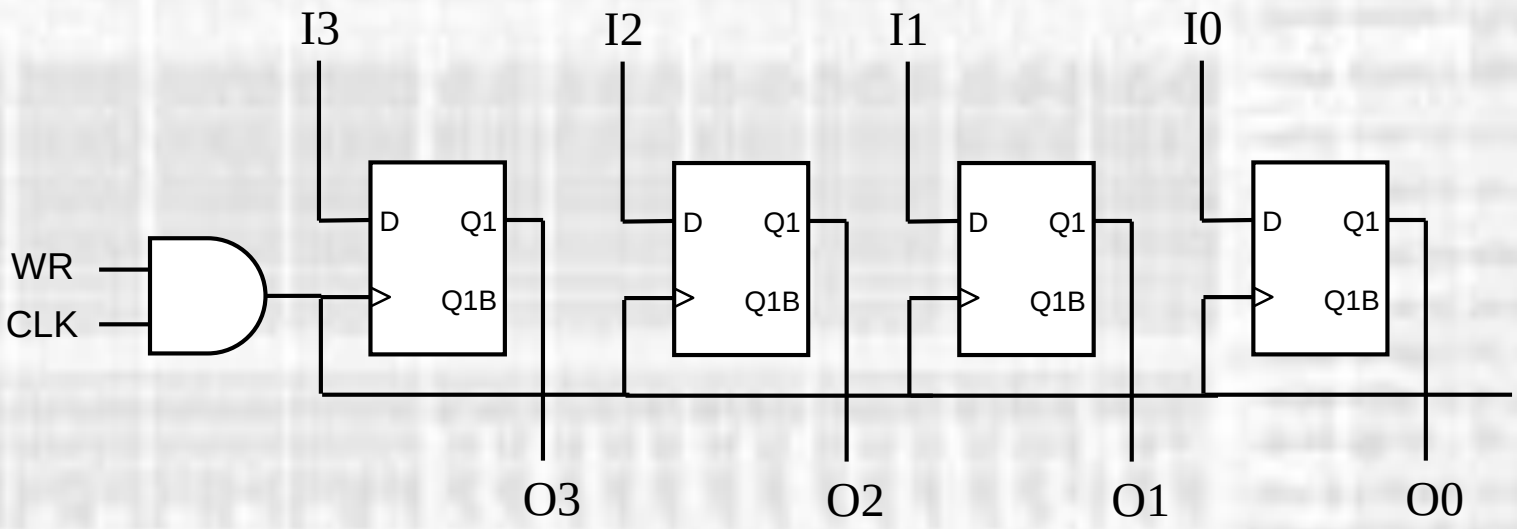
Asynchronous Propagation

D	J	K	Q1	Q1B	Q2	Q2B
Initial						
0	0	1	1	0	1	0
After clk1						
			0	1	0	1
1	1	0				
After clk2						
			1	0	1	0
0	0	1				
After clk3						
			0	1	0	1
1	1	0				

Logic Review

Basic Blocks

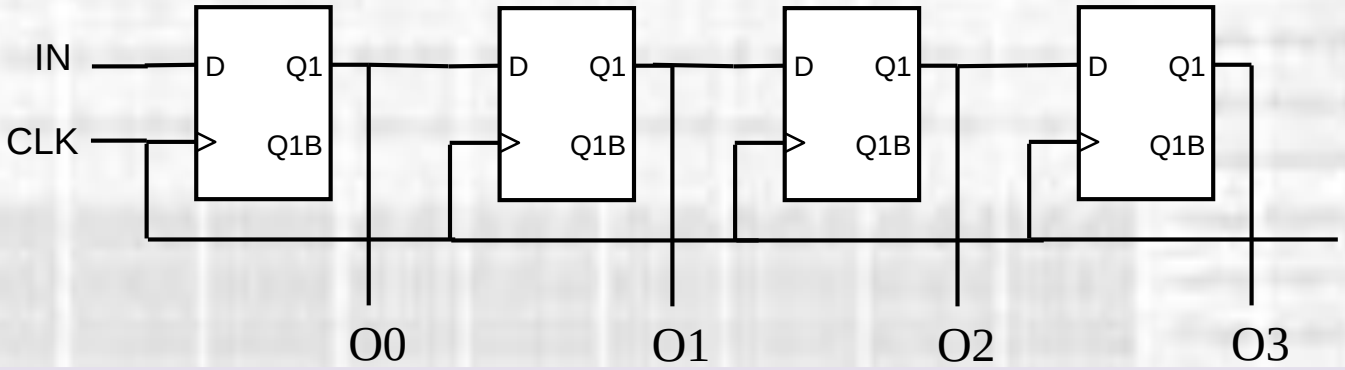
- Parallel Data Register

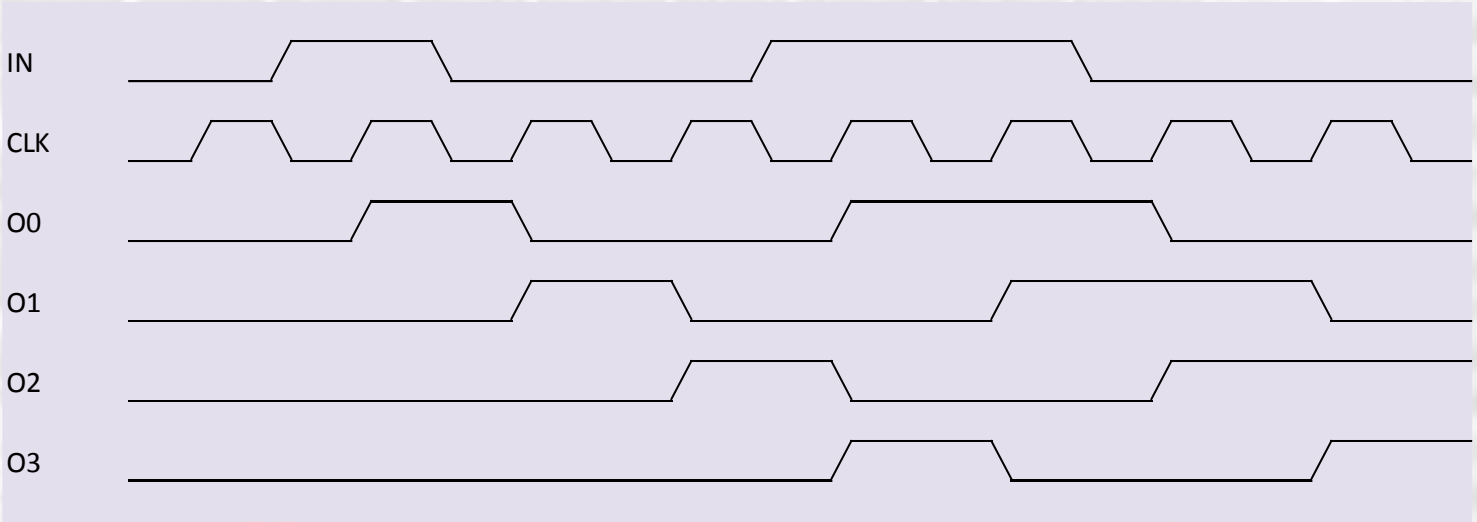


Logic Review

Basic Blocks

- Shift Register

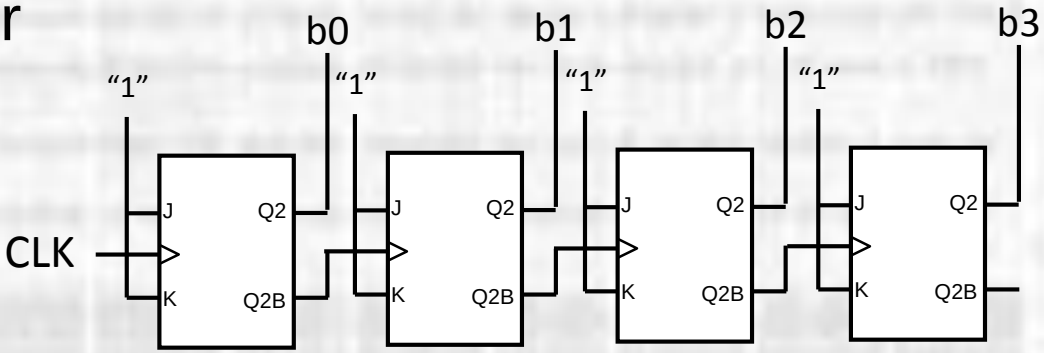


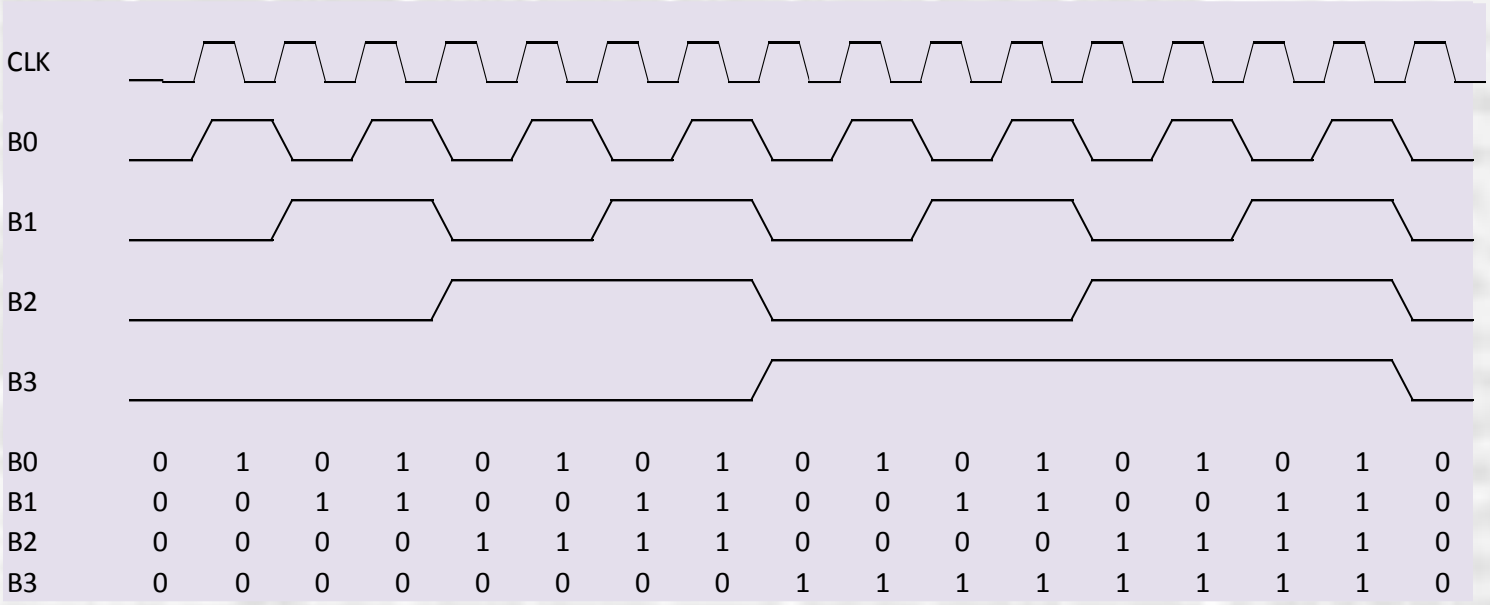


Logic Review

Basic Blocks

- Counter



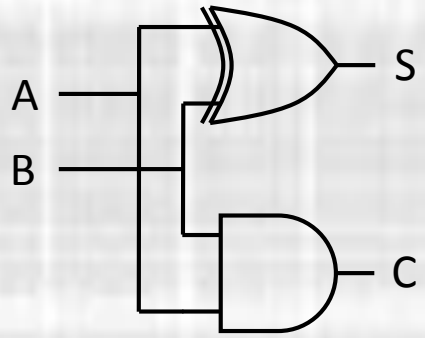


B0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
B1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
B2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0
B3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0

Logic Review

Basic Blocks

- Half Adder

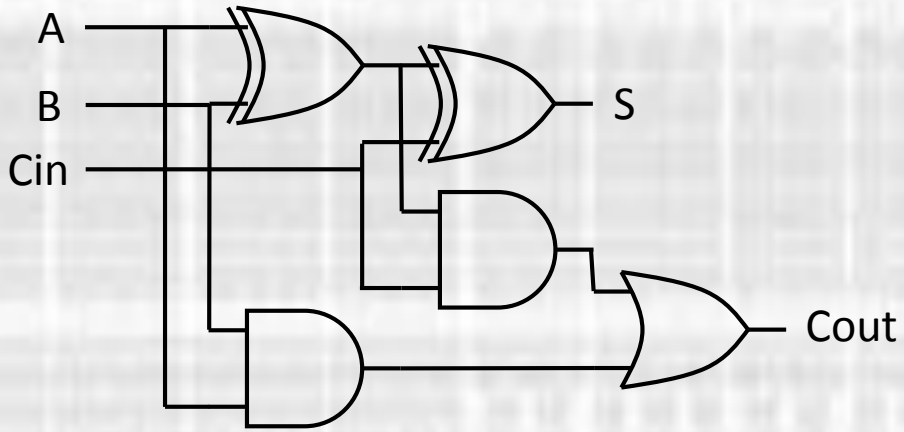


		Sum	Carry
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Logic Review

Basic Blocks

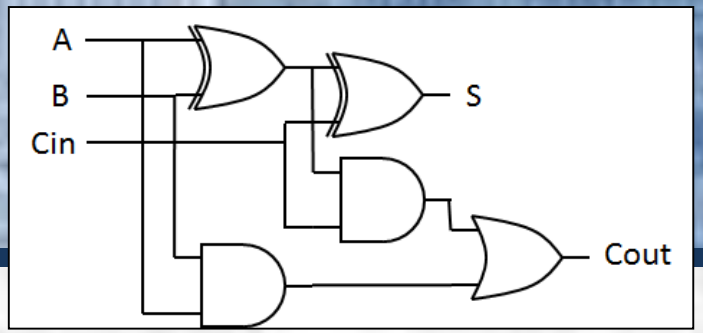
- Full Adder



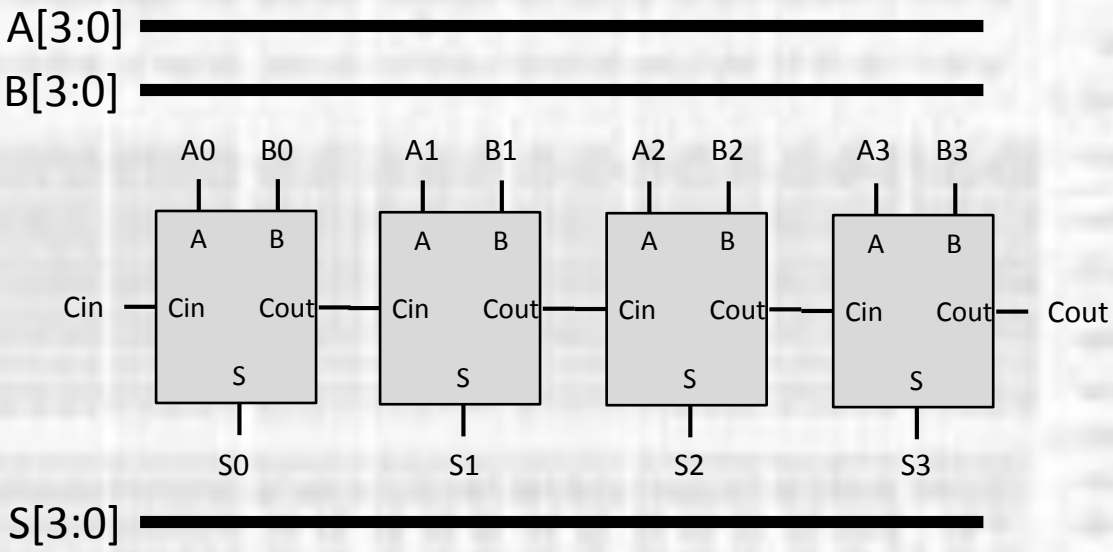
			Sum	Carry
Cin	**A**	**B**	**S**	**Cout**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Logic Review

Basic Blocks



- Adder



Number Systems

Base

- Base 10 (decimal)
  - The most familiar base for most people
    - ones, tens, hundreds, thousands
    - tenths, hundredths, thousandths
  - Base 10 → 10 individual digits
    - Range of individual digit: 0 → 9
  - Each position to the left of the decimal point is 10X the previous position
  - Each position to the right of the decimal point is 1/10th the previous position

1	2	3	4	.	5	6	7
Thousands	Hundreds	Tens	Ones	decimal point	tenths	hundredths	thousandths

1	2	3	4	.	5	6	7
digit x 10^3	digit x 10^2	digit x 10^1	digit x 10^0	decimal point	digit x 10^{-1}	digit x 10^{-2}	digit x 10^{-3}

Number Systems

Base

- Base 2 (binary)
  - The most common base for digital electronics
    - ones, twos, fours, eights
    - halves, quarters, eighths
  - Base 2 → 2 individual digits
    - Range of individual digit: 0 → 1
  - Each position to the left of the decimal point is 2X the previous position
  - Each position to the right of the decimal point is 1/2 the previous position

1	1	0	1	.	1	0	1
Eights	Fours	Twos	Ones	binary point	Halves	Quarters	Eighths

1	1	0	1	.	1	0	1
digit x 2^3	digit x 2^2	digit x 2^1	digit x 2^0	binary point	digit x 2^{-1}	digit x 2^{-2}	digit x 2^{-3}

Number Systems

Base

- Base 16 (hexadecimal)
  - Used as a short hand for binary
    - ones, 16s, 256s, 4096s
    - 16ths, 256ths
  - Base 16 → 16 individual digits
    - Range of individual digit: 0 → 9,A→F
    - 10=A, 11=B, 12=C, 13=D, 14=E, 15=F
  - Each position to the left of the decimal point is 16X the previous position
  - Each position to the right of the decimal point is 1/16 the previous position

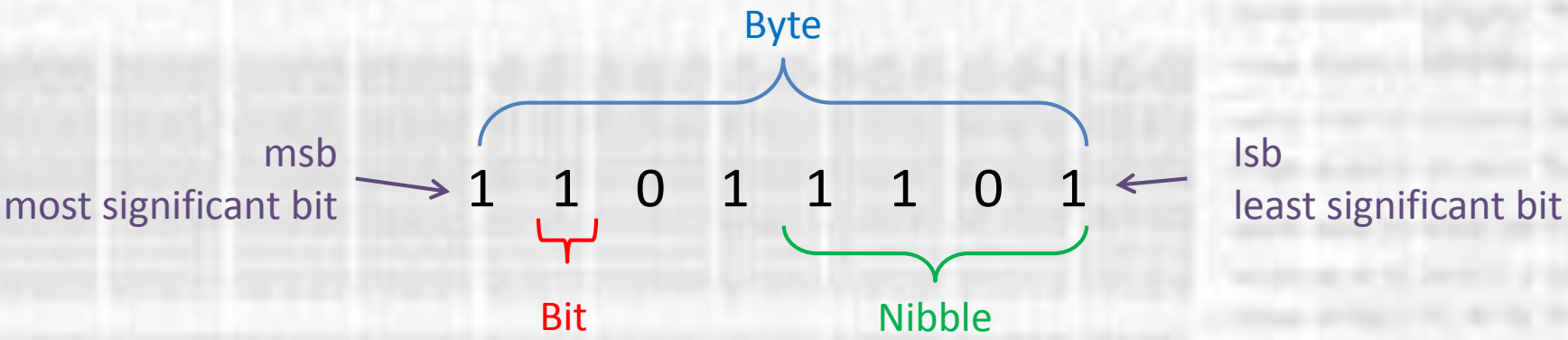
1	1	0	1	.	1	0	1
4096s	256s	16s	Ones	hexadecimal point	16ths	256ths	4096ths

1	1	0	1	.	1	0	1
digit x 16^3	digit x 16^2	digit x 16^1	digit x 16^0	hexadecimal point	digit x 16^{-1}	digit x 16^{-2}	digit x 16^{-3}

Number Systems

Binary

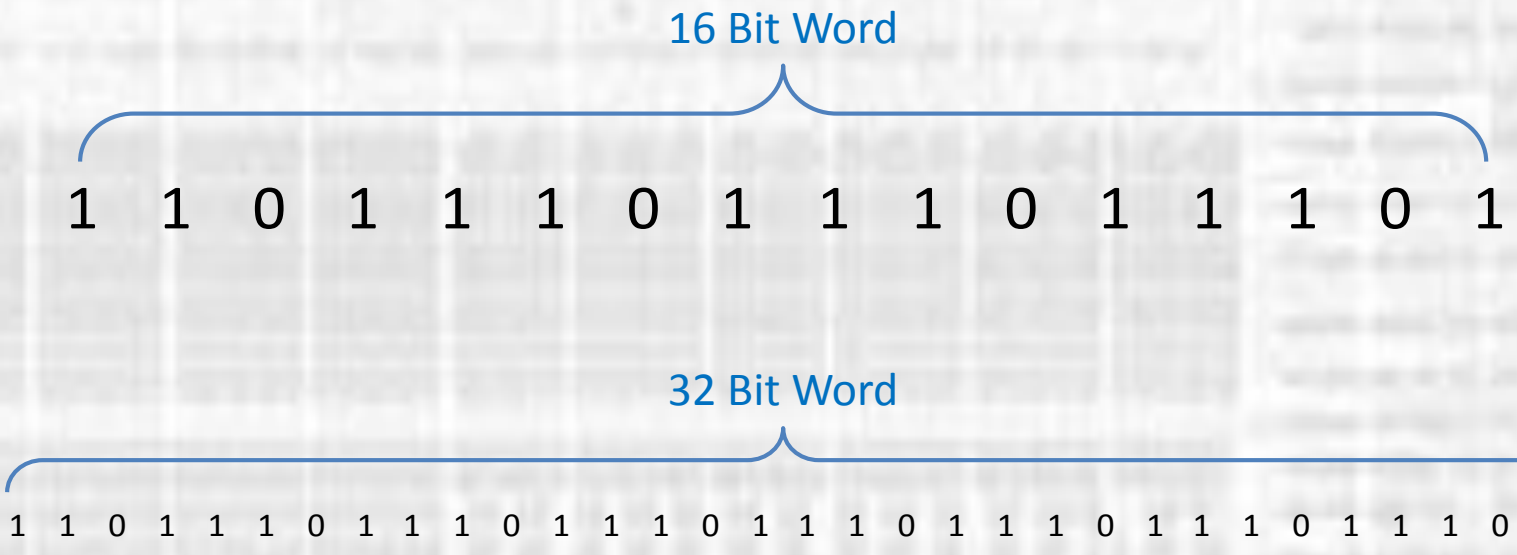
- Terminology



Number Systems

Binary

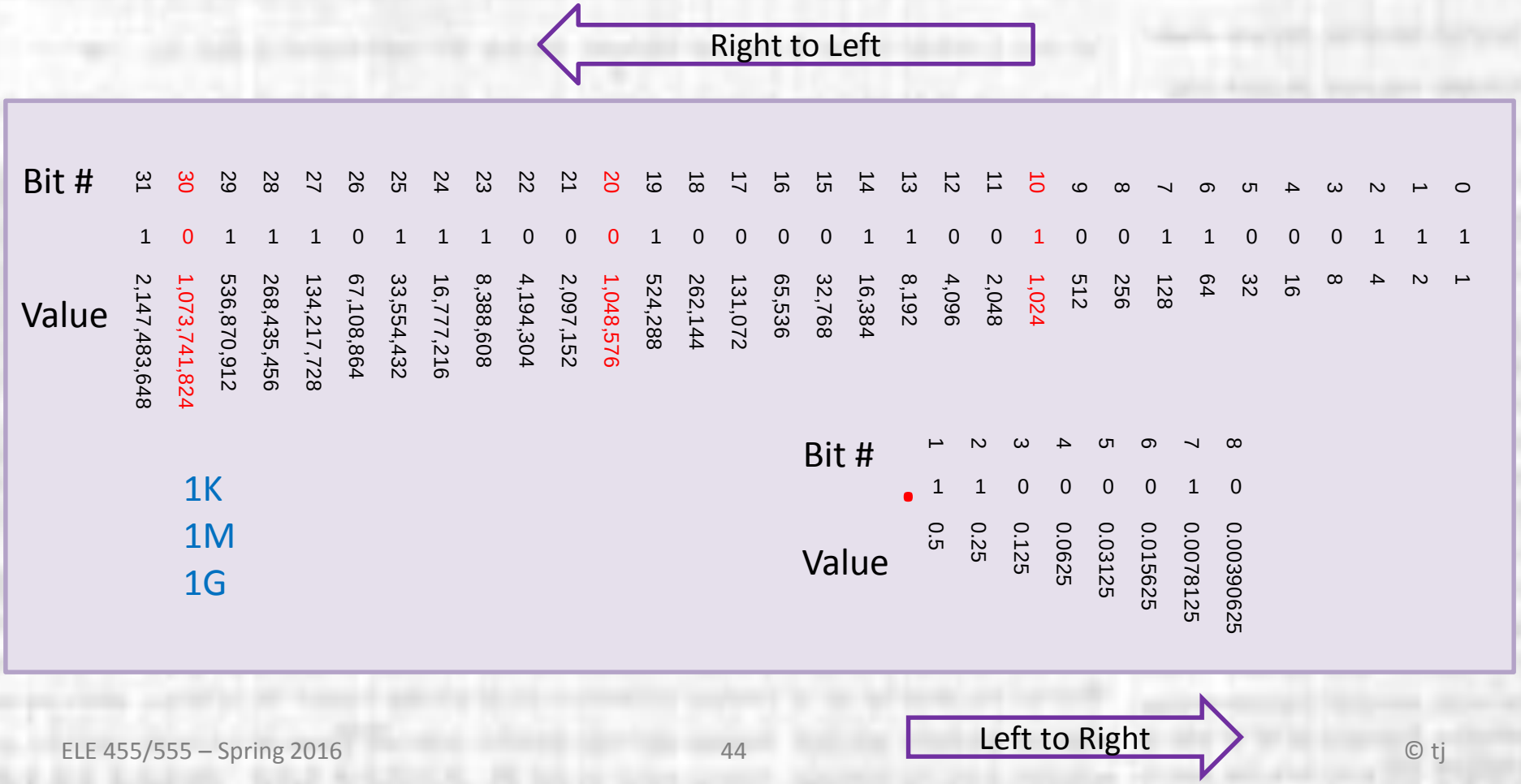
- Terminology



Number Systems

Binary

- Bit Values



Number Systems

Binary

- More Terminology
  - Assume S is an 8 bit binary number

    S = 10010110

    - S[7:0] = 10010110 — 10010110
    - S[3:0] = 1001 — 10010110
    - S[7:6] = 10 — 10010110
    - S[5] = 0 — 10010110
    - S[6,3] = 00 — 10010110
    - S[1] = 1 — 10010110
    - S[0] = 0 — 10010110

Number Systems

Binary

- Unsigned Binary (Binary)
  - All *n* bits used to represent the magnitude of the value
  - No negative values
  - Often used as absolute memory addresses

4	→	00000100
32	→	00100000
16	→	00010000
50	→	?
10010110_b	→	?
0.625	→	?

Number Systems

Binary

- Unsigned Binary (Binary)

convert 50 decimal to 8 bit unsigned binary

8 bits → bit values of 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1

greatest bit value ≤ 50 = 32 — 0 0 1
50 – 32 = 18

greatest bit value ≤ 18 = 16 — 0 0 1 1
18 – 16 = 2

greatest bit value ≤ 2 = 2 — 0 0 1 1 0 0 1
2 – 2 = 0

no more left — 0 0 1 1 0 0 1 0

Number Systems

Binary

- Unsigned Binary (Binary)

convert 10010110 unsigned binary to decimal

8 bits → bit values of 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1

$1*128 + 0*64 + 0*32 + 1*16 + 0*8 + 1*4 + 1*2 + 0*1$
$128 + 16 + 4 + 2 = 150$

10010110_b → 150

Number Systems

Binary

- Unsigned Binary (Binary)

convert 0.625 decimal to unsigned binary

first few fractional bits → bit values of 1/2 | 1/4 | 1/8 | 1/16

greatest bit value ≤ 0.625 = 1/2 . 1
0.625 - 0.5= 0.125

greatest bit value ≤ 0.125 = 1/8 . 1 0 1
0.125 – 0.125 = 0

no more left . 1 0 1 0 or 0.101

Number Systems

Binary

- Unsigned Binary (Binary)
  - Maximum values: (non fractional)
    - 4 bits (1111) = 15
    - 8 bits (1111 1111) = 255
    - 16 bits (1111 1111 1111 1111) = 65,535
    - 32 bits (1111 1111 1111 1111 1111 1111 1111 1111) = 4,294,967,295
  - Wait! 4 bits → 2^4 = 16, why is the maximum value 15

    8 bits → 2^8 = 256, why is the maximum value 255

    …

Number Systems

Binary

- Unsigned Binary (Binary)
  - Wait! 4 bits → $2^4 = 16$, why is the maximum value 15

    8 bits → $2^8 = 256$, why is the maximum value 255

    …
  - Zero is one of our values, that only leaves 15 more …

decimal	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
unsigned binary	1111	1110	1101	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001	0000

Number Systems

Binary

- Unsigned Binary (Binary)
  - Maximum values:
    - 4 bits (1111) = 15 = 2^4-1
    - 8 bits (1111 1111) = 255 = 2^8-1
    - 16 bits (1111 1111 1111 1111) = 65,535 = $2^{16}-1$
    - 32 bits (1111 1111 1111 1111 1111 1111 1111 1111) = 4,294,967,295 = $2^{32}-1$

Number Systems

Binary

- Signed Magnitude
  - MSB used to represent the sign of the value
    - MSB = 0 → positive
    - MSB = 1 → negative
  - Remaining bits represent the magnitude of the value
  - Used in most floating point number representations

50 → 0011 0010

-50 → 1011 0110

-37 →

10010110_b signed magnitude →

Number Systems

Binary

- Signed Magnitude

convert -37 decimal to 8 bit signed magnitude

8 bits → bit values of s | 64 | 32 | 16 | 8 | 4 | 2 | 1

s = negative 1
|-37| = 37

greatest bit value ≤ 37 = 32 1 0 1
37 – 32 = 5

greatest bit value ≤ 5 = 4 1 0 1 0 0 1
5 – 4 = 1

greatest bit value ≤ 1 = 1 1 0 1 0 0 1 0 1
1 – 1 = 0

Number Systems

Binary

- Signed Magnitude

convert 10010110 signed magnitude to decimal

8 bits → bit values of s | 64 | 32 | 16 | 8 | 4 | 2 | 1

0∗64 + 0∗32 + 1∗16 + 0∗8 + 1∗4 + 1∗2 + 0∗1
16 + 4 + 2 = 22

sign = 1 = negative → -22

10010110_b signed magnitude → -22

Number Systems

Binary

- Signed Magnitude
  - Maximum values: (non fractional)
    - 4 bits (s111) = ±7 = 2^3-1
    - 8 bits (s111 1111) = ± 127 = 2^7-1
    - 16 bits (s111 1111 1111 1111) = ± 32,767 = 2^{15}-1

7	6	5	4	3	2	1	0	0	-1	-2	-3	-4	-5	-6	-7
0111	0110	0101	0100	0011	0010	0001	0000	1000	1001	1010	1011	1100	1101	1110	1111

Number Systems

Binary

- Signed Magnitude
  - Issues
    - Binary math is difficult with sign magnitude representation
    - 2 zeros really causes a lot of problems

Number Systems

Binary

- One's Complement
  - Negative numbers are formed by flipping all bits
  - Most Significant Bit (MSB) is the sign bit
    - MSB = 0 → positive
    - MSB = 1 → negative
  - All bits are used to represent the magnitude of the value
  - Not widely used anymore – but a stepping stone to 2's complement

50 → 0011 0010
-50 → 1100 1101
-37 →
10010110_b 1's comp →

Number Systems

Binary

- One's Complement

convert -37 decimal to one's complement

8 bits → positive bit values of s | 64 | 32 | 16 | 8 | 4 | 2 | 1

s = negative → flip all bits at end
|-37| = 37

greatest bit value ≤ 37 = 32 0 0 1
37 – 32 = 5

greatest bit value ≤ 5 = 4 0 0 1 0 0 1
5 – 4 = 1

greatest bit value ≤ 1 = 1 0 0 1 0 0 1 0 1
1 – 1 = 0

Continued

Number Systems

Binary

- One's Complement

convert -37 decimal to one's complement – cont'd

s = negative → flip all bits at end

0 0 1 0 0 1 0 1 → 1 1 0 1 1 0 1 0

-37 → 1 1 0 1 1 0 1 0 one's complement

Number Systems

Binary

- One's Complement

convert 10010110 one's complement to decimal

sign is negative → remember this for the end
→ flip the bits

10010110 → 01101001

8 bits → positive bit values of s | 64 | 32 | 16 | 8 | 4 | 2 | 1

1∗64 + 1∗32 + 0∗16 + 1∗8 + 0∗4 + 0∗2 + 1∗1
64 + 32 + 8 + 1 = 105

sign = 1 = negative → -105

10010110_b 1's comp → -105

Number Systems

Binary

- One's Complement
  - Maximum values:
    - 4 bits = ±7 = 2^3-1
    - 8 bits = ± 127 = 2^7-1
    - 16 bits = ± 32,767 = $2^{15}-1$

7	6	5	4	3	2	1	0	0	-1	-2	-3	-4	-5	-6	-7
0111	0110	0101	0100	0011	0010	0001	0000	1111	1110	1101	1100	1011	1010	1001	1000

Number Systems

Binary

- One's Complement
  - Issues
    - 2 zeros really causes a lot of problems

Number Systems

Binary

- Two's Complement
  - Negative numbers are formed by flipping all bits and adding 1
  - Most Significant Bit (MSB) is the sign bit
    - MSB = 0 → positive
    - MSB = 1 → negative
  - All bits are used to represent the magnitude of the value
  - The dominant representation for binary arithmetic

50 → 0011 0010
-50 → 1100 1110

-37 →
10010110_b 2's comp →

Number Systems

Binary

- Two's Complement

convert -37 decimal to two's complement

8 bits → positive bit values of s | 64 | 32 | 16 | 8 | 4 | 2 | 1

s = negative → flip all bits and add 1 at end
|-37| = 37

greatest bit value ≤ 37 = 32 0 0 1
37 – 32 = 5

greatest bit value ≤ 5 = 4 0 0 1 0 0 1
5 – 4 = 1

greatest bit value ≤ 1 = 1 0 0 1 0 0 1 0 1
1 – 1 = 0

Number Systems

Binary

- Two's Complement

convert -37 decimal to two's complement – cont'd

s = negative → flip all bits and add 1 at end

flip +1

0 0 1 0 0 1 0 1 → 1 1 0 1 1 0 1 0 → 1 1 0 1 1 0 1 1

-37 → 1 1 0 1 1 0 1 1 two's complement

Number Systems

Binary

- Two's Complement

convert 10010110 two's complement to decimal

sign is negative → remember this for the end
→ flip the bits and add 1 (works both directions)

10010110 $\xrightarrow{\text{flip}}$ 01101001 $\xrightarrow{+1}$ 01101010

8 bits → positive bit values of s | 64 | 32 | 16 | 8 | 4 | 2 | 1

1∗64 + 1∗32 + 0∗16 + 1∗8 + 0∗4 + 1∗2 + 0∗1
64 + 32 + 8 + 2 = 106

sign = 1 = negative → -106

10010110 2's comp → -106

Number Systems

Binary

- Two's Complement
  - Maximum values:
    - 4 bits = +7, -8 = $2^3-1, -2^4$
    - 8 bits = + 127, -128 = $2^7-1, -2^8$
    - 16 bits = + 32,767, -32,768 = $2^{15}-1, -2^{16}$

7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8
0111	0110	0101	0100	0011	0010	0001	0000	1111	1110	1101	1100	1011	1010	1001	1000

Number Systems

Binary

- Two's Complement
  - Advantages
    - Addition is done the same way as unsigned numbers – same adder circuit
    - ONLY 1 ZERO !
    - Simple word length extension
  - Disadvantages
    - Asymmetric range
    - Harder to do comparisons
    - Not intuitive

7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8
0111	0110	0101	0100	0011	0010	0001	0000	1111	1110	1101	1100	1011	1010	1001	1000

Number Systems

Binary

- Two's Complement
  - Sign Extension
    - When extending to larger word sizes, extend the sign bit to the left

4 bit 8 bit 16 bit

0110 → 00000110 → 0000000000000110

1001 → 11111001 → 1111111111111001

this works for 1's complement also
not the same for signed magnitude: -1 = 1001 → 10000001 = -1

Number Systems

Binary

- Two's Complement
  - Fast way to do 2's complement conversions

FOR NEGATIVE NUMBERS

- working from the right
  find the first 1 and leave it and all preceding 0's the same
  flip all remaining bits to the left

10010110 2's complement
10 - first 1 from the right
01101010 - all remaining bits flipped
106
- 106 - since we started with a negative sign bit

Number Systems

Binary

- Binary Coded Decimal (BCD)
  - Encode base 10 digits into 4 bit nibbles
  - No negative representation
  - Used in some financial applications

50 → 0101 0000
79 → 0111 1001

37 → BCD
10010110_{BCD} → decimal

Number Systems

Binary

- Binary Coded Decimal

convert 37 decimal to BCD

4 bits → bit values of 8 | 4 | 2 | 1

3 → 0011 0 0 1 1

7 → 0111 0 0 1 1 0 1 1 1

37 → 0011 0111 BCD

Number Systems

Binary

- Binary Coded Decimal

convert 10010110 BCD to decimal

Break into 4 bit nibbles

10010110 → 1001 0110

1001 → 9
0110 → 6

10010110 BCD → 96

Number Systems

Binary

- Binary Coded Decimal
  - Maximum values:
    - 4 bits = 9
    - 8 bits = 99
    - 16 bits = 9999

9	8	7	6	5	4	3	2	1	0
1001	1000	0111	0110	0101	0100	0011	0010	0001	0000

Number Systems

Binary

- Binary Coded Decimal
  - Issues
    - No negative values
    - Not efficient – limited range

9	8	7	6	5	4	3	2	1	0
1001	1000	0111	0110	0101	0100	0011	0010	0001	0000

Number Systems

Binary

- Representation Summary

	Unsigned Binary	Signed Magnitude	1's Complement	2's Complement	BCD
50	0011 0010	0011 0010	0011 0010	0011 0010	0101 0000
-50	N/A	1011 0010	1100 1101	1100 1110	N/A

0110 1001 Unsigned Binary	0110 1001 Signed Magnitude	0110 1001 1's Complement	0110 1001 2's Complement	0110 1001 BCD
105	105	105	105	69

1001 0110 Unsigned Binary	1001 0110 Signed Magnitude	1001 0110 1's Complement	1001 0110 2's Complement	1001 0110 BCD
150	-22	-105	-106	96

Number Systems

Hexadecimal

- Use hexadecimal (hex) as a shorthand for binary
  - Group sets of 4 binary bits and represent them with the hexadecimal equivalent
    - 1011 → B 0110 → 6 1110 → E
    - 10110110 → B6 01101110 → 6E
    - 1011011001101110 → B66E
  - Often it is easier if a space is inserted when writing these
    - 1011 0110 0110 1110 → B66E
  - When it is not obvious from the context you need to indicate the binary representation that the hex represents
    - Address = B66E → binary equivalent is unsigned binary → 46,702
    - Data value = B66E → binary equivalent is 2's complement → -18,834

Number Systems

Hexadecimal

- Use hexadecimal (hex) as a shorthand for binary
  - Multiple ways to indicate a hex value
    - 12CDh h at end
    - h12CD h at beginning
    - $12CD $ at beginning
    - 0x12CD 0x at beginning
  - Different processors use different shorthand

Number Systems

Hexadecimal

- Use hexadecimal (hex) as a shorthand for binary

	Unsigned Binary	Signed Magnitude	1's Complement	2's Complement	BCD
50	0011 0010	0011 0010	0011 0010	0011 0010	0101 0000
	h32	32h	$32	0x32	32h
-50	N/A	1011 0010	1100 1101	1100 1110	N/A
		B2h	$CD	0xCE	

h96	96h	$96	0x96	96
Unsigned Binary	Signed Magnitude	1's Complement	2's Complement	BCD
150	-22	-105	-106	96

Number Systems

Floating Point

- Scientific Number Representation

  - $1.60217657 \times 10^{-19}$ coulombs
  - $6.0221413 \times 10^{+23}$

  - Normalized to have only 1 digit (non-zero) to the left of the decimal point
  - multiplied by a power of 10

  - $5692.3456 \rightarrow 5.6923456 \times 10^{+3}$
  - $.00023456 \rightarrow 2.3456 \times 10^{-4}$

  - format is: $\text{mantissa} \times 10^{\text{exponent}}$

Number Systems

Floating Point

- Binary Floating Point Number Representation
  - Normalized to have only 1 digit to the left of the decimal point
    - this must be a 1 since our choices are only 0 and 1 and we don't use 0
  - multiplied by a power of 2

  - 1011.1101 → $1.0111101 \times 2^{+3}$
  - .00011001 → 1.1001×2^{-4}

  - format is: mantissa × 2^{exponent}

  BUT

  - since the mantissa always starts with "1." we can use 1.fraction × 2^{exponent}

Number Systems

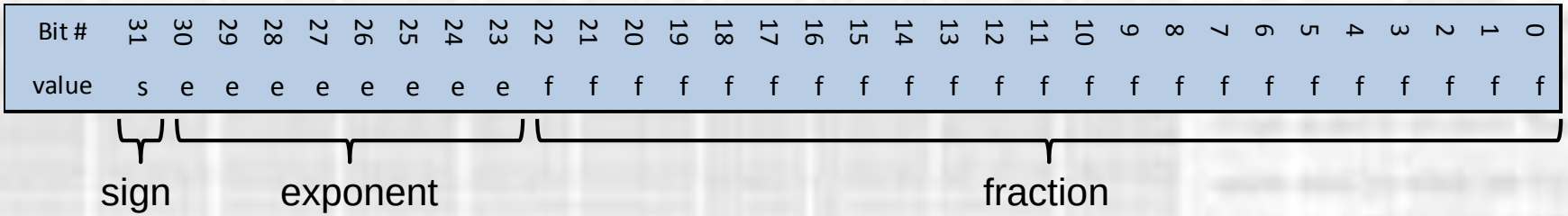
Floating Point

- Binary Floating Point Number Representation
  - It is simpler to work with only positive exponents
  - Bias the exponent
    - With an 8 bit exponent the range is:
      +127 to -127 using signed magnitude notation
    - Add 127 to the desired exponent value (for use in the representation)
      actual range is still +127 to -127
      representation range is 254 to 0
    - called an exponent with +127 bias
  - format is now: value = 1.fraction × $2^{(\text{exponent} - 127)}$
    desired value representation

Number Systems

Floating Point

- Binary Floating Point Number Representation
  - IEEE Standard
    - value = (-1 x sign) x 1.fraction × $2^{(exponent - 127)}$
    - 32 bit format



    - Special cases
      - If E = 255, and F is non-zero, then the value is NaN (Not a Number)
      - If E = 255, F = 0 and S = 1, then the value is -infinity
      - If E = 255, F = 0, and S = 0, then the value is +infinity
      - If E = 0, and F = 0, then the value is 0
    - Range
      - $1.11111111111111111111111_2 \times 2^{+127} = 3.4028 \times 10^{38}$
      - $1.00000000000000000000001_2 \times 2^{-127} = 1.1754 \times 10^{-38}$
      - 24 bit fractional precision ↔ 6 to 7 decimal digits

Number Systems

Floating Point

- Example

use IEEE standard floating point to represent: 2,345,678.7109375

2,345,678 = 0010 0011 1100 1010 1100 1110 = 0x23CACE
0.7109375 = 0.10110110 = 0x0.B6

2,345,678.7109375 = 0010 0011 1100 1010 1100 1110 . 1011 0110
= 1. 0 0011 1100 1010 1100 1110 1011 0110 × 2^{21}

fraction = 0001 1110 0101 0110 0111 0101 1011 0
exponent = 21 + 127 = 148 = 1001 0100
sign = 0

will not fit in fraction part of the notation

0 10010100 0001 1110 0101 0110 0111 010

Number Systems

Floating Point

- Example

convert the IEEE floating point number
0 10010100 0001 1110 0101 0110 0111 010 to decimal

sign = 0
exponent = 1001 0100 = 148 → $2^{148-127} = 2^{21}$
fraction = 0001 1110 0101 0110 0111 010

+ $1.0001\ 1110\ 0101\ 0110\ 0111\ 010 \times 2^{21}$
= 1 0001 1110 0101 0110 01110 . 10
= 2345678.5

error = (0.5 – 0.7109375)/2345678.5 = - 9×10^{-8}
~7 decimal digits of precision

Number Systems

Binary Arithmetic (2's complement)

- Addition
  - Add bit place by bit place
  - overflow indicates the new answer does not fit in our # of bits
    - carry-in of the msb ≠ carry-out of the msb

0		1	1	1	1	1			
	0	1	0	0	0	1	1	1	71
+	0	0	0	1	1	1	0	1	+ 29
	0	1	1	0	0	1	0	0	100

0	1	1	1	1	1	1	1		
	0	1	1	0	0	1	1	1	103
+	0	0	1	1	1	1	0	1	+ 61
	1	1	1	0	0	1	0	0	164

overflow -28

1	1			1	1	1			
	1	1	0	0	0	1	1	1	-57
+	1	1	1	1	0	1	0	1	+ -11
	1	0	1	1	1	1	0	0	-68

1				1	1	1			
	1	1	0	0	0	1	1	1	-57
+	1	0	0	0	0	1	0	1	+ -123
	0	1	0	0	0	1	0	0	-180

overflow -28

Number Systems

Binary Arithmetic (2's complement)

- Subtraction
  - Negate the subtrahend and add

	0	1	0	0	0	1	1	1
-	0	0	0	1	1	1	0	1

71 - 29

1	1				1	1	1	
	0	1	0	0	0	1	1	1
+	1	1	1	0	0	0	1	1
	0	0	1	0	1	0	1	0

71 + (-29) = 42

	0	0	0	0	0	1	1	1
-	1	0	0	1	1	1	0	1

7 - (-99)

0					1	1	1	
	0	0	0	0	0	1	1	1
+	0	1	1	0	0	0	1	1
	0	1	1	0	1	0	1	0

7 + 99 = 116

Number Systems

Binary Arithmetic (2's complement)

- Subtraction
  - Negate the subtrahend and add

	1	0	1	1	1	0	0	1
-	0	0	0	1	1	1	0	1

-71 - 29

1	1	1				1	1		
	1	0	1	1	1	0	0	1	
+	1	1	1	0	0	0	1	1	
	1	0	0	1	1	1	0	0	

-71 + (-29) = -100

	1	1	1	1	1	0	0	1
-	1	0	0	1	1	1	0	1

-7 - (-99)

1	1	1				1	1		
	1	1	1	1	1	0	0	1	
+	0	1	1	0	0	0	1	1	
	0	1	0	1	1	1	0	0	

7 + 99 = 92

Number Systems

Binary Arithmetic (2's complement)

- Multiplication
  - the product requires twice as many bits as the multiplier/multiplicand
    -7 x 6 = -42 1001 x 0110 = 1101 0110
  - In 2's complement you must sign extend to the product bit width

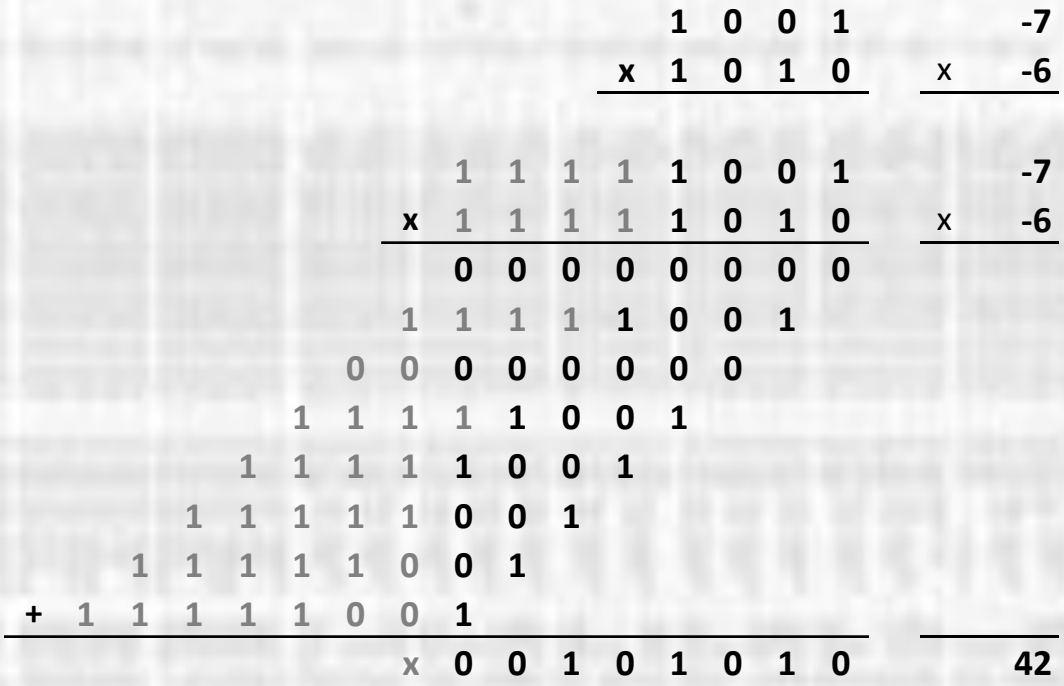
```
                    0 1 1 1        7
                x   0 1 1 0    x   6
                ---------------------

                0 0 0 0 0 1 1 1    7
            x   0 0 0 0 0 1 1 0    x   6
            -------------------    -----
                0 0 0 0 0 0 0 0
              0 0 0 0 0 1 1 1
            0 0 0 0 0 1 1 1
+ 0 0 0 0 0 0 0 0
  ---------------------------------
  x x x 0 0 1 0 1 0 1 0          42
```

Number Systems

Binary Arithmetic (2's complement)

- Multiplication



Number Systems

Binary Arithmetic (2's complement)

- Multiplication
  - Where possible multiply by the positive value

```
          0 1 1 1      7
        x 1 0 1 0    x -6
      ---------------------
          0 0 0 0
        0 1 1 1
      0 0 0 0 0 0
    1 0 0 0 0 1 1 1
  0 0 0 0 0 0 0 0
1 0 0 0 0 1 1 1
1 0 0 0 0 1 1 1
1 0 0 0 0 1 1 1
+ 0 0 0 1 0 1 1 1
      ---------------------
          x 1 1 0 1 0 1 1 0    -42
```

```
                  1 0 0 1     -7
                x 0 1 1 0   x  6
      ---------------------
                  0 0 0 0
                1 0 0 1
              0 0 0 0 0 0
            1 0 0 1 0 0 1 0
          0 0 0 0 0 0 0 0
        1 0 0 1 0 0 1 0
      + 0 0 0 0 0 0 0 0
      ---------------------
          x x 1 1 0 1 0 1 1 0    -42
```

```
          1 1 1 1 1 0 0 1     -7
        x 0 0 0 0 0 1 1 0   x  6
      ---------------------
          0 0 0 0 0 0 0 0
        1 1 1 1 1 0 0 1
      1 1 1 1 1 0 0 1
    1 1 1 1 1 0 0 1
  + 0 0 0 0 0 0 0 0
      ---------------------
          x 1 1 0 1 0 1 1 0    -42
```