

Lecture 2: Representing Integers

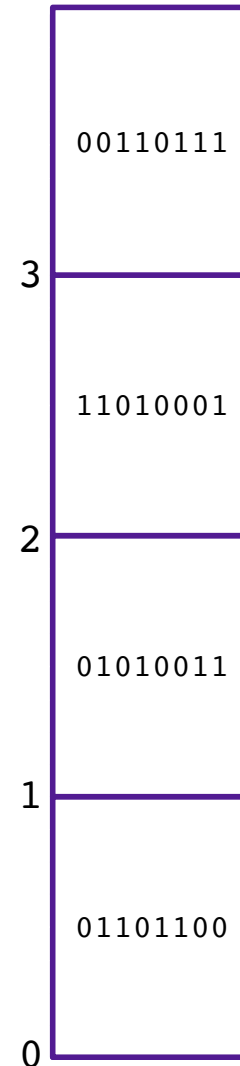
CS 105

Abstraction




Memory: A (very large) array of bytes

- **Memory** is an array of ^{bytes}~~bits~~
- A **byte** is a unit of eight bits
- An index into the array is an **address**, **location**, or **pointer**
 - Often expressed in hexadecimal
- We speak of the *value* in memory at an address
 - The value may be a single byte ...
 - ... or a multi-byte quantity starting at that address

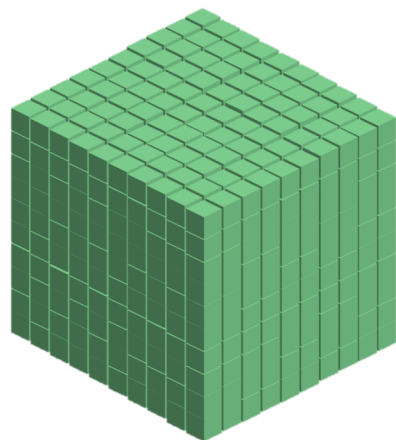


Representing Integers

- Arabic Numerals: 47
- Roman Numerals: XLVII
- Brahmi Numerals:
- Tally Marks: 

Base-10 Integers

1000 (10^3)

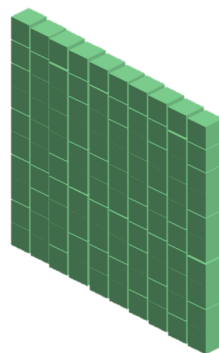


0

0

1

100 (10^2)



0

0

8

10 (10^1)



0

4

8

1 (10^0)



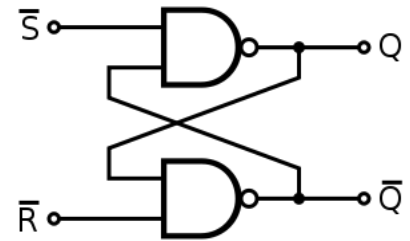
5

7

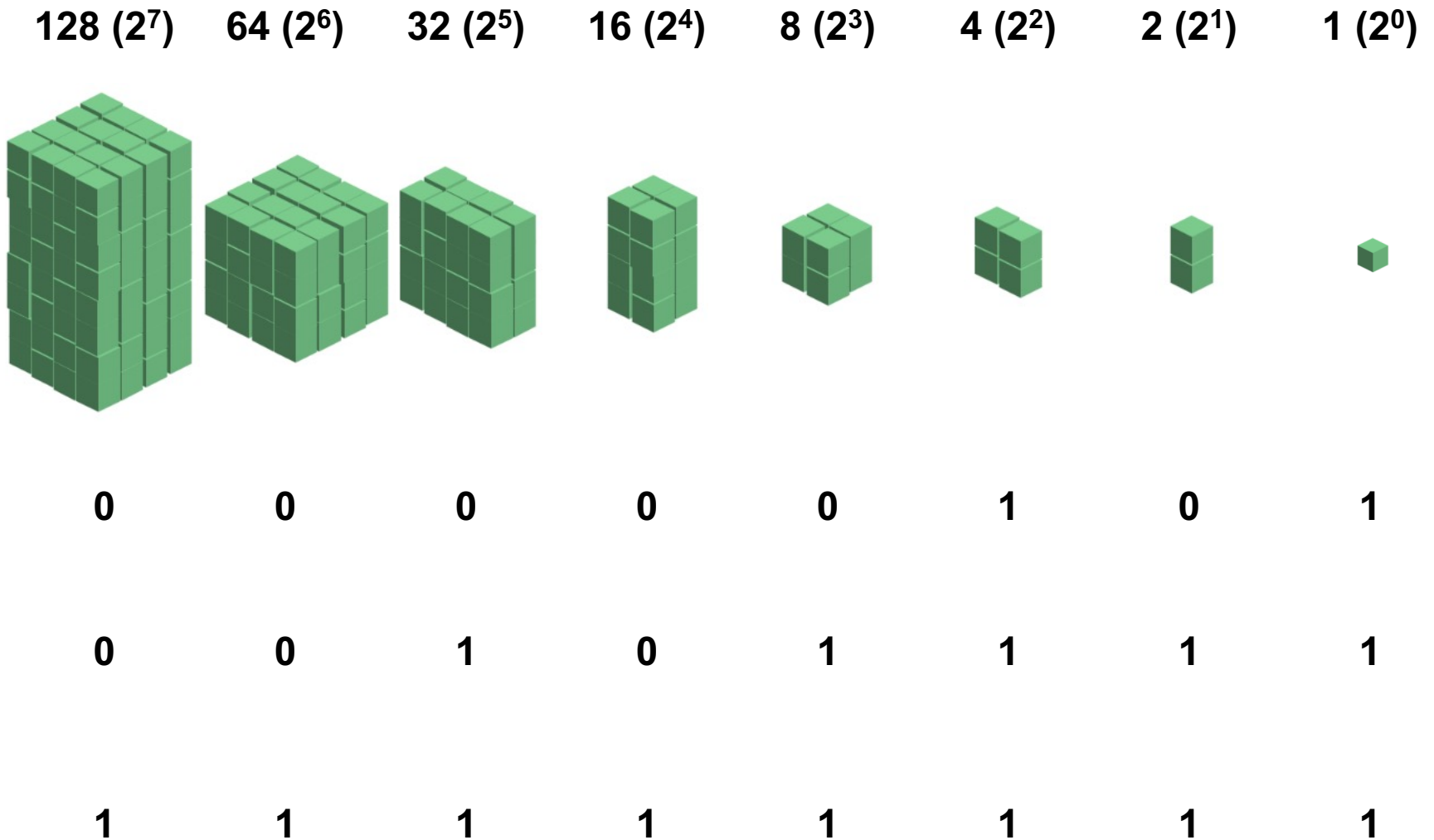
7

Storing bits

- Static random access memory (SRAM): stores each bit of data in a flip-flop, a circuit with two stable states
- Dynamic Memory (DRAM): stores each bit of data in a capacitor, which stores energy in an electric field (or not)
- Magnetic Disk: regions of the platter are magnetized with either N-S polarity or S-N polarity
- Optical Disk: stores bits as tiny indentations (pits) or not (lands) that reflect light differently
- Flash Disk: electrons are stored in one of two gates separated by oxide layers



Base-2 Integers (aka Binary Numbers)



Binary Numbers

- Decimal (Base-10):

4211

$$\begin{aligned} &= 4 \cdot 10^3 + 2 \cdot 10^2 + 1 \cdot 10^1 + 1 \cdot 10^0 \\ &= 4211 \end{aligned}$$

- Binary (Base-2):

1011

$$\begin{aligned} &= 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 11 \end{aligned}$$

Exercise 1: Binary Numbers

- Consider the following four-bit binary values. What is the (base-10) integer interpretation of these values?

1. 0001 $= 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 1$

2. 1010 $= 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 2 = 10$

3. 0111 $= 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 4 + 2 + 1 = 7$

4. 1111 $= 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 2 + 1 = 15$

Exercise 2: Binary Number Range

- What are the max number and min number that can be represented by a w-bit binary number?

$$1. \quad w = 3 \quad \min = 000_2 = 0_{10} \quad \max = 111_2 = 2^2 + 2^1 + 2^0 = 7_{10}$$

$$2. \quad w = 4 \quad \min = 0000_2 = 0_{10} \quad \max = 1111_2 = 2^3 + 2^2 + 2^1 + 2^0 = 15_{10}$$

$$3. \quad w = 8 \quad \min = 00000000_2 = 0_{10} \quad \max = 11111111_2 = 2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 255_{10}$$

- What is the general equation?

- $2^w - 1$

Unsigned Integers in C

C Data Type	Size (bytes)
unsigned char	1
unsigned short	2
unsigned int	4
unsigned long	8

For x86_64

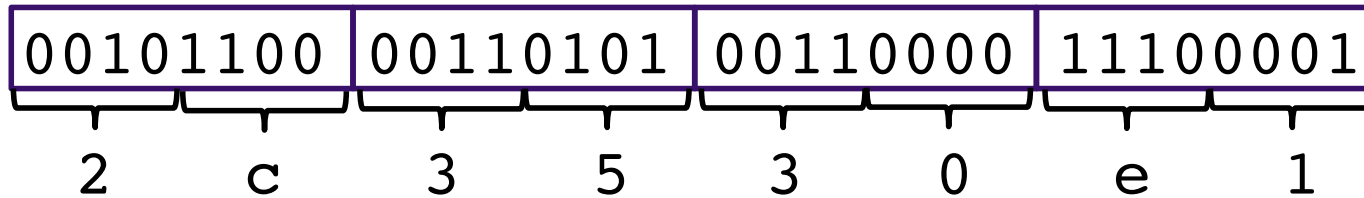
Unsigned -> Cannot represent negative numbers.

ASCII characters

<https://www.ascii-code.com/>

Char	Dec	Binary	Char	Dec	Binary	Char	Dec	Binary	Char	Dec	Binary	Char	Dec	Bin
!	33	00100001	1	49	00110001	A	65	01000001	Q	81	01010001	a	97	0110
"	34	00100010	2	50	00110010	B	66	01000010	R	82	01010010	b	98	0110
#	35	00100011	3	51	00110011	C	67	01000011	S	83	01010011	c	99	0110
\$	36	00100100	4	52	00110100	D	68	01000100	T	84	01010100	d	100	0110
%	37	00100101	5	53	00110101	E	69	01000101	U	85	01010101	e	101	0110
&	38	00100110	6	54	00110110	F	70	01000110	V	86	01010110	f	102	0110
'	39	00100111	7	55	00110111	G	71	01000111	W	87	01010111	g	103	0110
(40	00101000	8	56	00111000	H	72	01001000	X	88	01011000	h	104	0110
)	41	00101001	9	57	00111001	I	73	01001001	Y	89	01011001	i	105	0110
*	42	00101010	:	58	00111010	J	74	01001010	Z	90	01011010	j	106	0110
+	43	00101011	;	59	00111011	K	75	01001011	[91	01011011	k	107	0110
,	44	00101100	<	60	00111100	L	76	01001100	\	92	01011100	l	108	0110
-	45	00101101	=	61	00111101	M	77	01001101]	93	01011101	m	109	0110
.	46	00101110	>	62	00111110	N	78	01001110	^	94	01011110	n	110	0110
/	47	00101111	?	63	00111111	O	79	01001111	_	95	01011111	o	111	0110
0	48	00110000	@	64	01000000	P	80	01010000	`	96	01100000	p	112	0111

Hexadecimal Numbers (Base 16)



0x2c3530e1

How many binary digits can you represent with a single hexadecimal (base 16) digit?

Dec	Hex
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	a
11	b
12	c
13	d
14	e
15	f

Exercise 3: Hexadecimal Numbers

- Consider the following hexadecimal values. What is the representation of each value in (1) binary and (2) decimal?

1. $0x0a = 00001010_2 = 10_{10}$

2. $0x11 = 00010001_2 = 17_{10}$

3. $0x2f = 00101111_2 = 47_{10}$

Endianness

- **Big Endian:** low-order bits go on the right (47)
 - I tend to think in big endian numbers, so examples in class will generally use this representation
 - Networks generally use big endian (aka network byte order)
- **Little Endian:** low-order bits go on the left (74)
 - Most modern machines use this representation
- I will try to always be clear about whether I'm using a big endian or little endian representation
- When in doubt, ask!

Arithmetic Logic Unit (ALU)

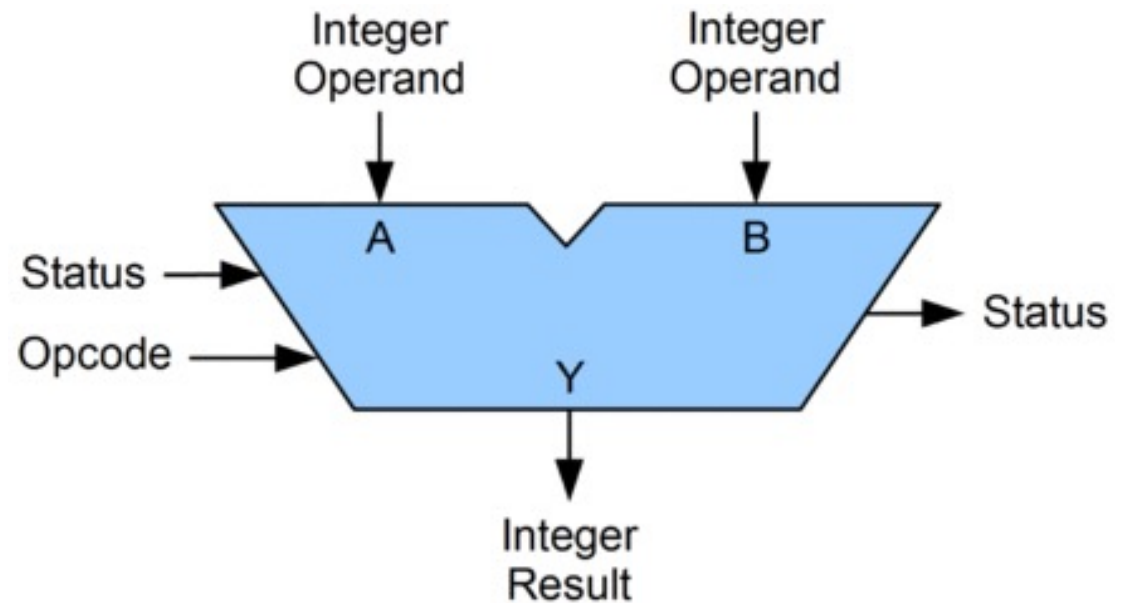
- A circuit that performs bitwise operations and arithmetic on integer binary types

Status examples:

- Carry-out
- Zero
- Negative
- Overflow
- parity

Opcode examples:

- Add, Subtract
- Increment, Decrement
- AND, OR, XOR
- Shift, Rotate



Bitwise vs Logical Operations in C

- Bitwise Operators `&`, `|`, `~`, `^`
 - View arguments as bit vectors
 - operations applied bit-wise in parallel
- Logical Operators `&&`, `||`, `!`
 - View 0 as “False”
 - View anything nonzero as “True”
 - Always return 0 or 1
 - **Short-circuit termination**
- Shift operators `<<`, `>>`
 - Left shift fills with zeros
 - For unsigned integers, right shift is logical (fills with zeros)

Exercise 4: Bitwise vs Logical Operations

Assume unsigned char data type (one byte). What do each of the following expressions evaluate to (interpreted as unsigned integers and expressed base-10)?

$$1. \sim 226 = \sim 11100010 = 00011101 = 29$$

$$2. !226 = !11100010 = 00000000 = 0$$

$$3. 120 \& 85 = 01111000 \& 01010101 = 01010000 = 80$$

$$4. 120 | 85 = 01111000 | 01010101 = 01111101 = 125$$

$$5. 120 \&\& 85 = 01111000 \&\& 01010101 = 00000001 = 1$$

$$6. 120 || 85 = 01111000 || 01010101 = 00000001 = 1$$

$$7. 81 \ll 4 = 01010001 \ll 4 = 00010000 = 16$$

$$8. 81 \ll 2 = 01010001 \ll 2 = 01000100 = 68$$

$$9. 81 \gg 4 = 01010001 \gg 4 = 00000101 = 5$$

$$10. 81 \gg 2 = 01010001 \gg 2 = 00010100 = 20$$

Example: Using Bitwise Operations

What do these operations do?

$x \ll 2$

- "multiply by 4"

$x \& 1$

- "x is odd"

$(x + 7) \& 0xFFFFFFFF8$

- "round up to a multiple of 8"

Addition Example

- Compute 5 + 6 assuming all ints are stored as **eight-bit** (1 byte) unsigned values

$$\begin{array}{r}
 1 \\
 00000101 \\
 + 00000110 \\
 \hline
 00001011 = 11 \text{ (Base-10)}
 \end{array}$$

Like you learned in grade school, only binary!
 ... and with a finite number of digits

Addition Example with Overflow

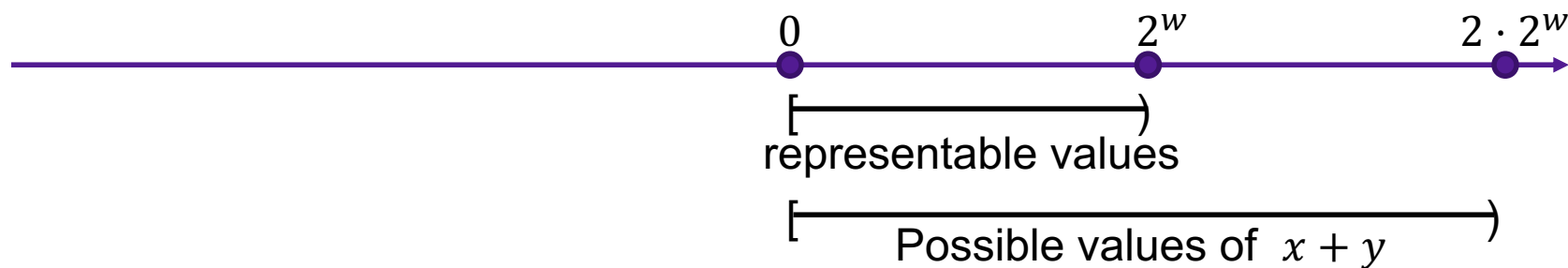
- Compute 200 + 100 assuming all ints are stored as **eight-bit** (1 byte) unsigned values

$$\begin{array}{r}
 11 \\
 11001000 \\
 + 01100100 \\
 \hline
 00101100 = 44 \text{ (Base-10)}
 \end{array}$$

Like you learned in grade school, only binary!
 ... and with a finite number of digits

Error Cases

- Assume w -bit unsigned values



- $$x +_w^u y = \begin{cases} x + y & \text{(normal)} \\ x + y - 2^w & \text{(overflow)} \end{cases}$$
- Overflow occurred if and only if $x +_w^u y < x$

Exercise 5: Binary Addition

- Given the following 5-bit unsigned values, compute their sum and indicate whether an overflow occurred

x	y	x+y	overflow?
00010	00101		
01100	00100		
10100	10001		

Exercise 5: Binary Addition

- Given the following 5-bit unsigned values, compute their sum and indicate whether an overflow occurred

x	y	x+y	overflow?
00010	00101	00111	no
01100	00100	10000	no
10100	10001	00101	yes

Multiplication Example

- Compute 5 x 6 assuming all ints are stored as **eight-bit** (1 byte) unsigned values

$$\begin{array}{r}
 00000101 \\
 \times 00000110 \\
 \hline
 00000000 \\
 000001010 \\
 + 0000010100 \\
 \hline
 00011110 = 30 \text{ (Base-10)}
 \end{array}$$

Like you learned in grade school, only binary!
 ... and with a finite number of digits

Multiplication Example

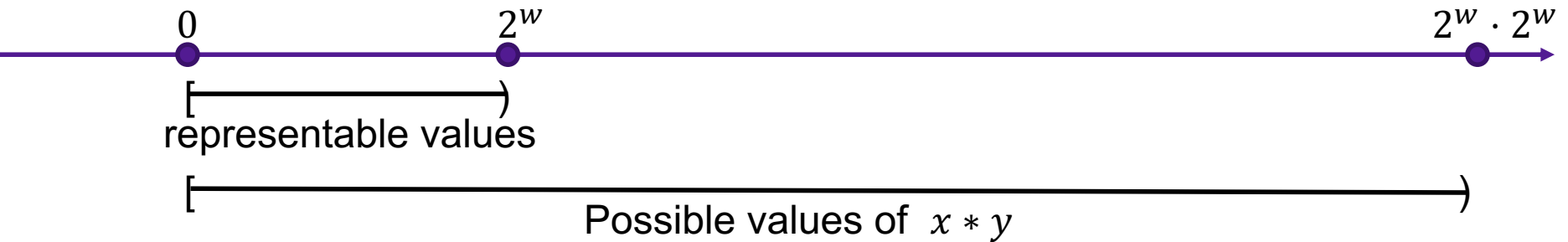
- Compute 200 x 3 assuming all ints are stored as **eight-bit** (1 byte) unsigned values

$$\begin{array}{r}
 11001000 \\
 \times 00000011 \\
 \hline
 11001000 \\
 + 110010000 \\
 \hline
 1001011000 = 88 \text{ (Base-10)}
 \end{array}$$

Like you learned in grade school, only binary!
 ... and with a finite number of digits

Error Cases

- Assume w -bit unsigned values



- $x *_w^u y = (x \cdot y) \bmod 2^w$

Exercise 6: Binary Multiplication

- Given the following 3-bit unsigned values, compute their product and indicate whether an overflow occurred

x	y	x*y	overflow?
100	101		
010	011		
111	010		

Exercise 6: Binary Multiplication

- Given the following 3-bit unsigned values, compute their product and indicate whether an overflow occurred

x	y	x*y	overflow?
100	101	100	yes
010	011	110	no
111	010	110	yes

Multiplying with Shifts

- Multiplication is slow
- Bit shifting is kind of like multiplication, and is often faster
- What is “ $x \ll 3$ ”?
 - $x * 8 = x \ll 3$
- How could you perform “ $x * 10$ ” with shifts and addition?
 - $x * 10 = x \ll 3 + x \ll 1$
- Most compilers will automatically replace multiplications with shifts where possible