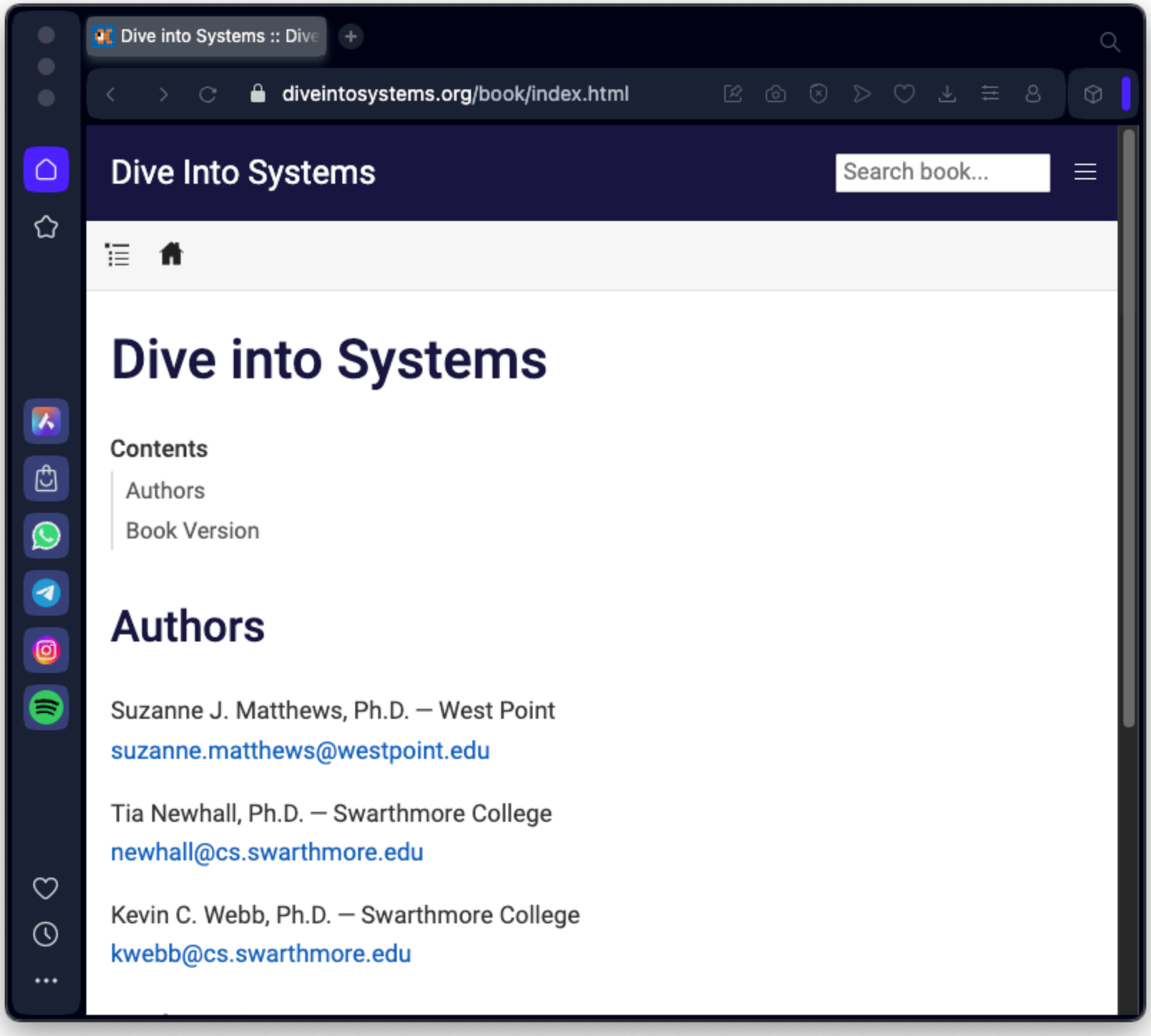
4. Binary and Data Representation

For COMSC 142

Sam Bowne

Jan 21, 2025

Free online textbook



- https://diveintosystems.org/book/index.html

Topics

4.1. Number Bases and Unsigned Integers

4.2. Converting Between Bases

4.3. Signed Binary Integers
4.4. Binary Integer Arithmetic

4.5. Overflow

4.6. Bitwise Operators

4.7. Integer Byte Order

4.8. Real Numbers in Binary

4.1. Number Bases and Unsigned Integers
4.2. Converting Between Bases

C 101: Binary Games (25 pts + 15 extra)

Play each game till you have 10 correct. Then the flag will appear.

Nybbles

Lesson (pdf) (ppt)
C 101.1: Nybbles (5 pts)

Bytes

Lesson (pdf) (ppt)
C 101.2: Bytes (5 pts) C 101.3: Bytes (5 pts)

Hexadecimal

Lesson (pdf) (ppt)
C 101.4: Hexadecimal (5 pts)

Modular Arithmetic 1

Lesson (pdf)
C 101.5: Modular Arithmetic 1 (5 pts)

XOR

Lesson (pdf)
C 101.6: XOR (5 pts extra)

Modular Arithmetic 2

Lesson (pdf)
C 101.7: Modular Arithmetic 2 (10 pts extra)

4.3. Signed Binary Integers

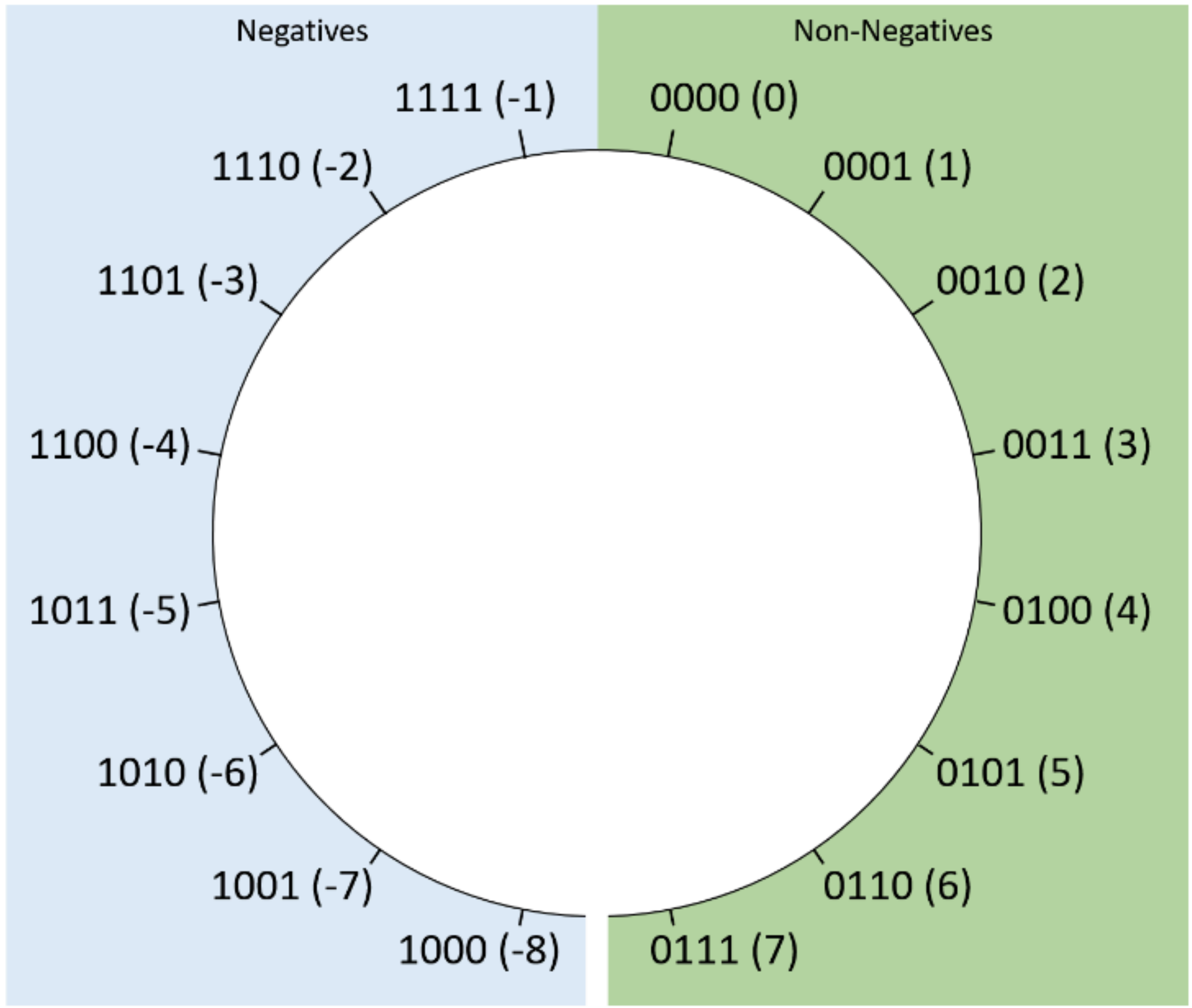
4.3.2. Two's Complement

- Leftmost bit treated as -1 or 0
- All other bits treated as 1 or 0

$$-(d_{N-1} \times 2^{N-1}) \quad + \quad (d_{N-2} \times 2^{N-2}) \quad + \quad \ldots \quad + \quad (d_2 \times 2^2) \quad + \quad (d_1 \times 2^1) \quad + \quad (d_0 \times 2^0)$$

^ note the leading negative sign for just the first term!

4.3.2. Two's Complement



Negation

- To find the negative of a number X
- Flip all the bits and add one
- Example: 13

```
00001101  (decimal 13)
```

Next, "flip the bits" (change all zeros to ones, and vice versa):

```
11110010
```

Finally, adding one yields 0b11110011. Sure enough, applying the formula for interpreting a two’s complement bit sequence shows that the value is -13:

$$-(1 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$

$$= -128 + 64 + 32 + 16 + 0 + 0 + 2 + 1 = -13$$

C Programming With Signed versus Unsigned Integers

- **int** is a signed integer
- **unsigned int** is unsigned

```c
#include <stdio.h>

int main(void) {
    int example = -100;

    /* Print example int using both signed and unsigned placeholders.
    printf("%d  %u\n", example, example);

    return 0;
}
```

Even though this code passes `printf` the same variable (`example`) twice, it prints `-100 4294967196` . Be careful to interpret your values correctly!

4.4. Binary Integer Arithmetic

Addition

Problem Setup **Worked Example**

```
  0010
+ 1011
```

```
         1     <- Carry the 1 from digit 1 into
digit 2
        0010
      + 1011

Result: 1101
```

Subtraction

Problem Setup	Converted to Addition	Worked Example
0111 - 0011	1 (carry in) 0111 + 1100 (bits flipped)	1 (carry in) 0111 + 1100 (bits flipped) Result: 0100 Carry out: 1

Subtraction

Problem Setup	Converted to Addition	Worked Example

`  0111`
`- 1101`
	`     1 (carry in)`
	`  0111`
	`+ 0010 (bits flipped)`
		`       1 (carry in)`
		`    0111`
		`  + 0010 (bits flipped)`
		
		`   Result:   1010`
		`Carry out:  0`

Addition

Problem Setup | Worked Example

```
  0010
+ 1011
```

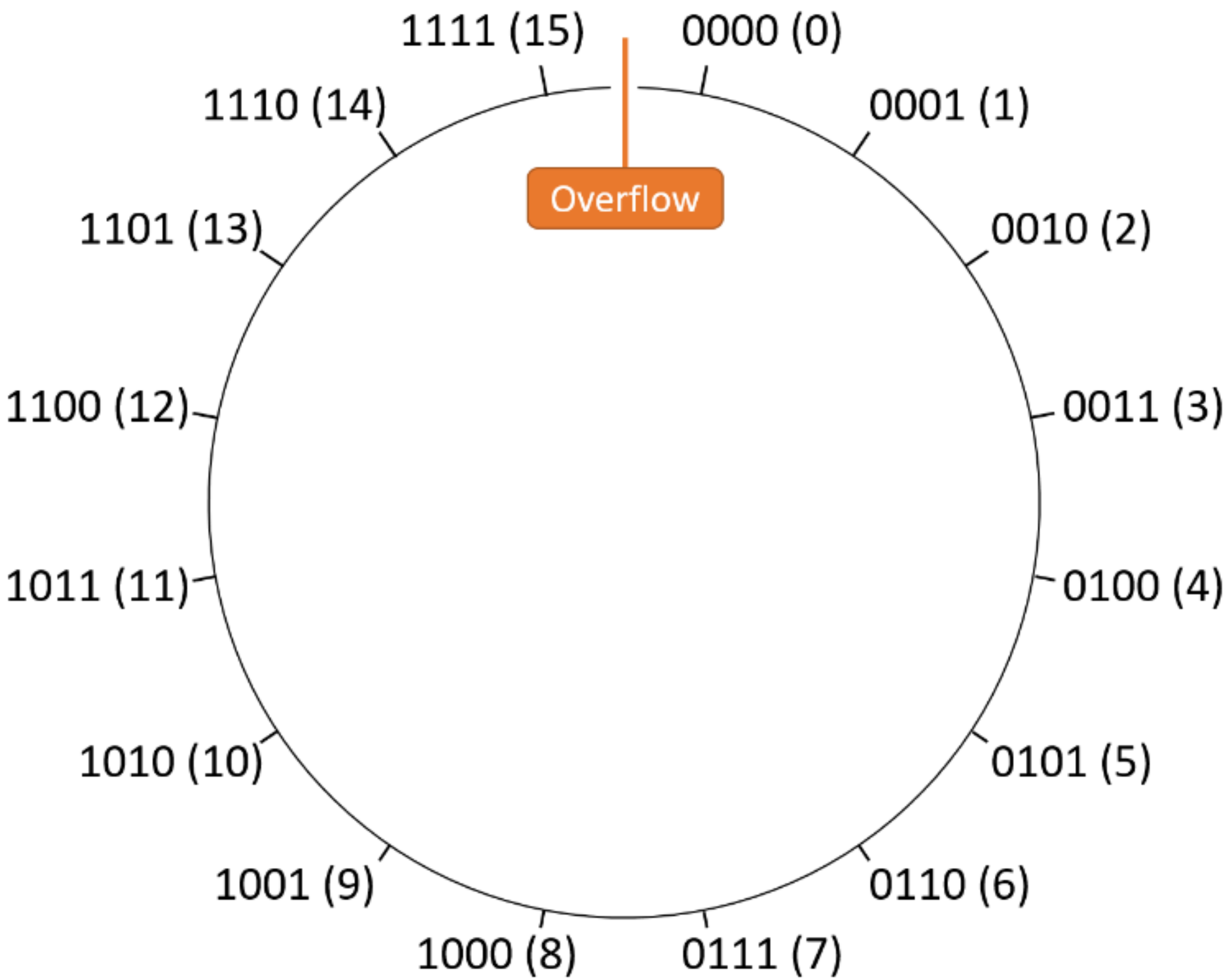
```
          1    <- Carry the 1 from digit 1 into
digit 2
         0010
       + 1011

Result: 1101
```

Kahoot!

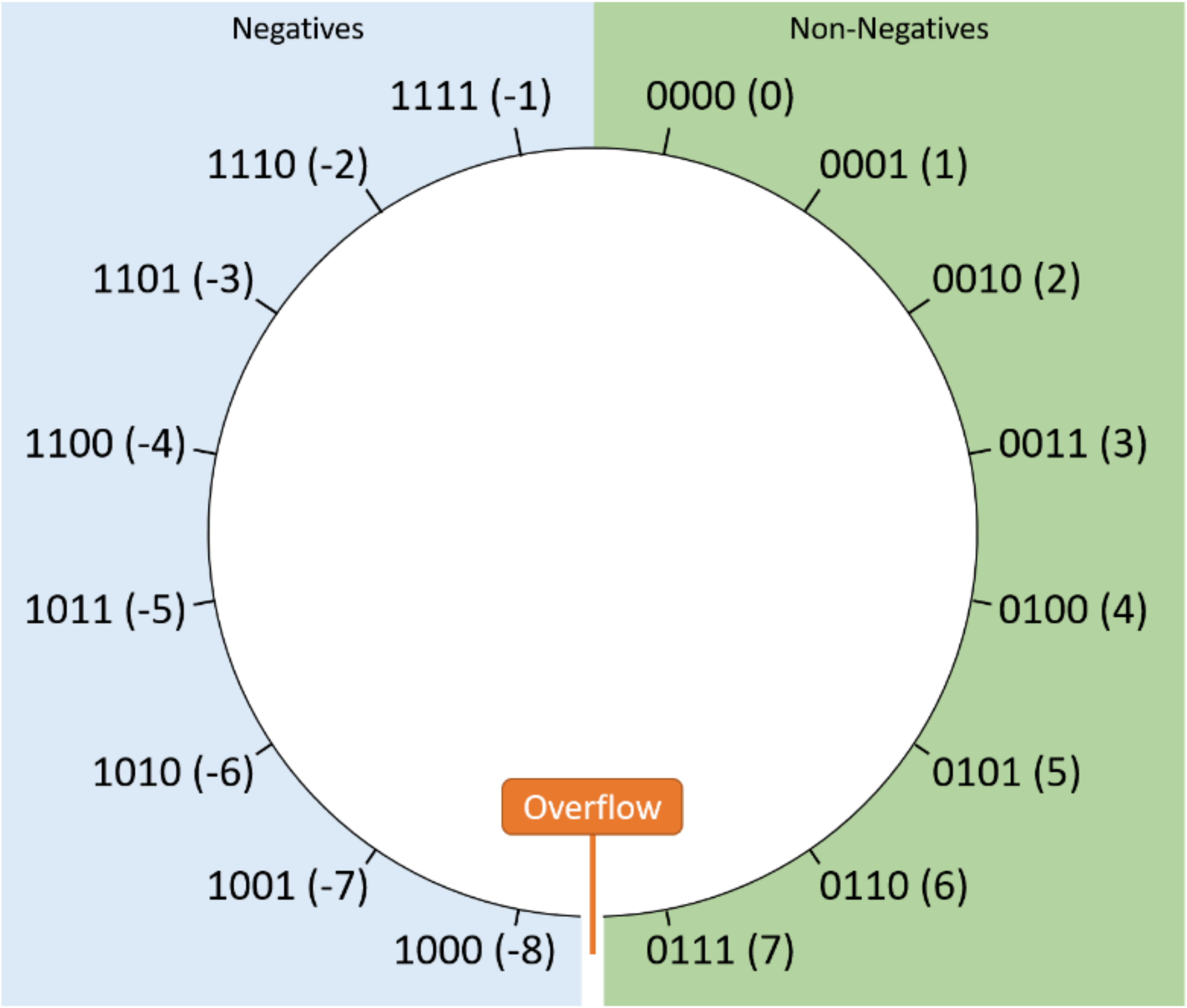
Ch 4a

4.5. Overflow

Unsigned Overflow



Signed Overflow



4.6. Bitwise Operators

4.6.1. Bitwise AND

Table 1. The Results of Bitwise ANDing Two Values (A AND B)

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

4.6.2. Bitwise OR

Table 2. The Results of Bitwise ORing Two Values (A OR B)

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

4.6.3. Bitwise XOR (Exclusive OR)

Table 3. The Results of Bitwise XORing Two Values (A XOR B)

A	B	A ^ B
0	0	0
0	1	1
1	0	1
1	1	0

4.6.4. Bitwise NOT

Table 4. The Results of Bitwise NOTing a Value (A)

A	~ A
0	1
1	0

4.6.5. Bit Shifting

- **Left shift**

```
int x = 13;  // 13 is 0b00001101

printf("Result: %d\n", x << 3);  // Prints 104 (0b01101000)
```

4.6.5. Bit Shifting

- **Logical right shift**
  - Prepends zeroes to the higher-order bits
  - Logically shifting 0b**10110011** to the right yields 0b**00101100**
- **Arithmetic right shift**
  - Prepends a copy of the most significant bit to preserve the signedness of the higher-order bits
  - Arithmetically shifting 0b**10110011** to the right yields 0b**11101100**

C Right-Shifting

- Does logical right-shift for **unsigned int**
- Arithmetic right-shift for signed **int**

```
/* Unsigned integer value: u_val. */
unsigned int u_val = 0xFF000000;

/* Signed integer value: s_val. */
int s_val = 0xFF000000;

printf("%08X\n", u_val >> 12);  // logical right shift
printf("%08X\n", s_val >> 12);  // arithmetic right shift
```

```
$ ./a.out
000FF000
FFFFF000
```

C 101: Binary Games

Not Secure bowneconsultingcontent.c

C 101: Binary Games (25 pts + 15 extra)

Play each game till you have 10 correct. Then the flag will appear.

Nybbles

Lesson (pdf) (ppt)
C 101.1: Nybbles (5 pts)

Bytes

Lesson (pdf) (ppt)
C 101.2: Bytes (5 pts) C 101.3: Bytes (5 pts)

Hexadecimal

Lesson (pdf) (ppt)
C 101.4: Hexadecimal (5 pts)

Modular Arithmetic 1

Lesson (pdf)
C 101.5: Modular Arithmetic 1 (5 pts)

XOR

Lesson (pdf)
C 101.6: XOR (5 pts extra)

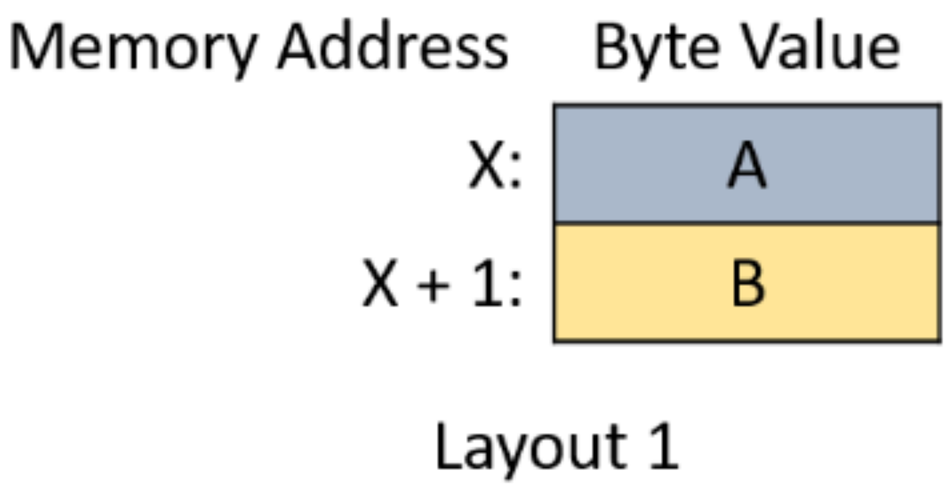
Modular Arithmetic 2

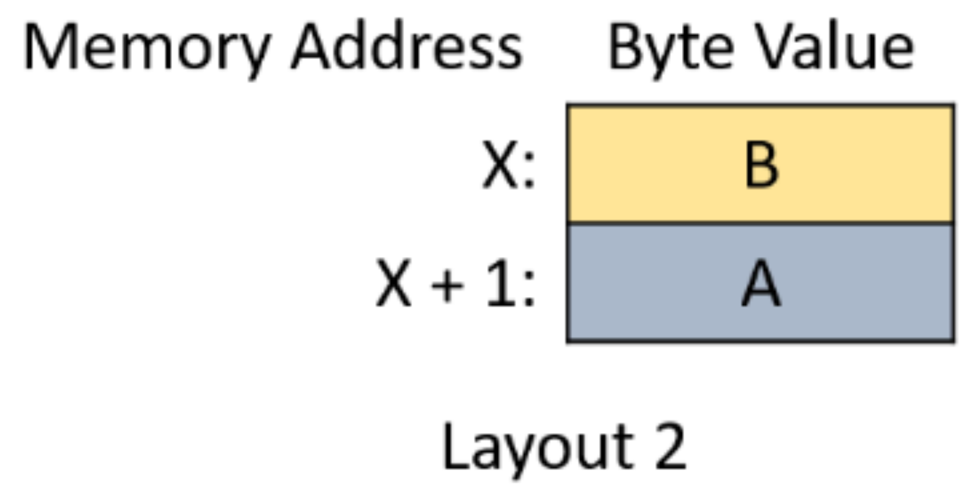
Lesson (pdf)
C 101.7: Modular Arithmetic 2 (10 pts extra)

4.7. Integer Byte Order

Byte Order (Endianness)

- Consider a 16-bit integer (**short**)
- High-order byte is A, low is B



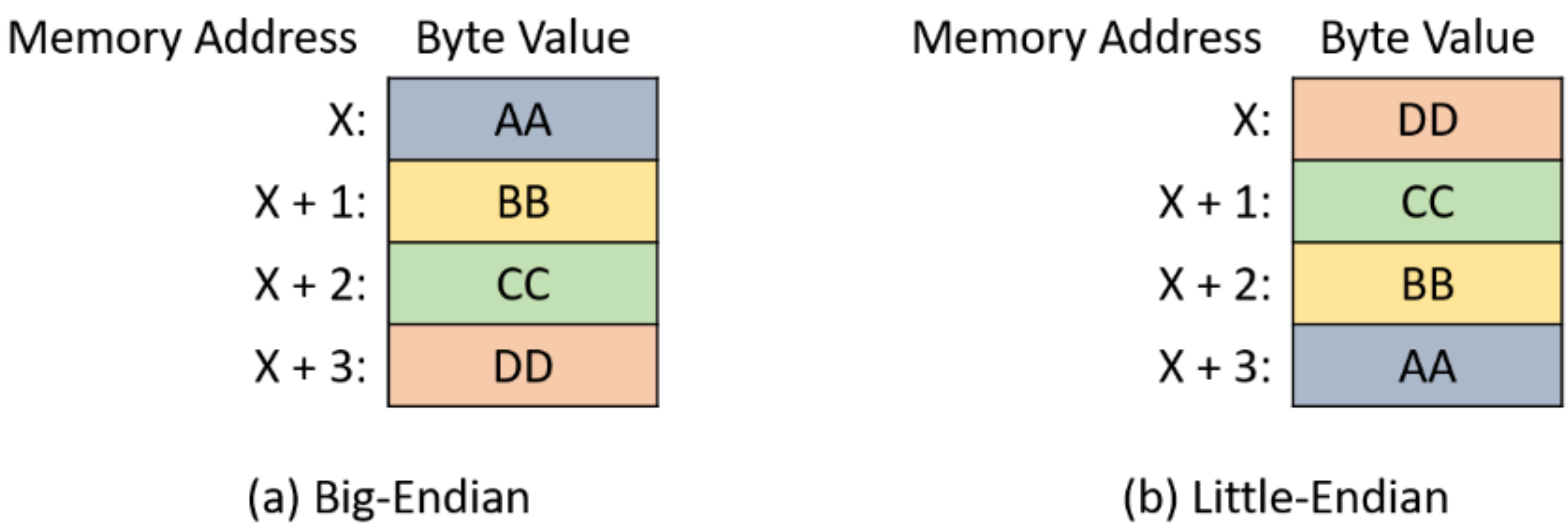


x86 uses Little-Endian

```c
// Initialize a four-byte integer with easily distinguishable byte values
int value = 0xAABBCCDD;

// Initialize a character pointer to the address of the integer.
char *p = (char *) &value;

// For each byte in the integer, print its memory address and value.
int i;
for (i = 0; i < sizeof(value); i++) {
    printf("Address: %p, Value: %02hhX\n", p, *p);
    p += 1;
}
```

```
$ ./a.out
Address: 0x7ffc0a234928, Value: DD
Address: 0x7ffc0a234929, Value: CC
Address: 0x7ffc0a23492a, Value: BB
Address: 0x7ffc0a23492b, Value: AA
```

Byte Order (Endianness)



4.8. Real Numbers in Binary

4.8.1. Fixed-Point Representation

Binary Point

	0	0	0	1	0	1	.	1	0
Digit:	d_5	d_4	d_3	d_2	d_1	d_0		d_{-1}	d_{-2}
Contribution to Value:	32	16	8	4	2	1		½	¼

Figure 1. The value of each digit in an eight-bit number with two bits after the fixed binary point

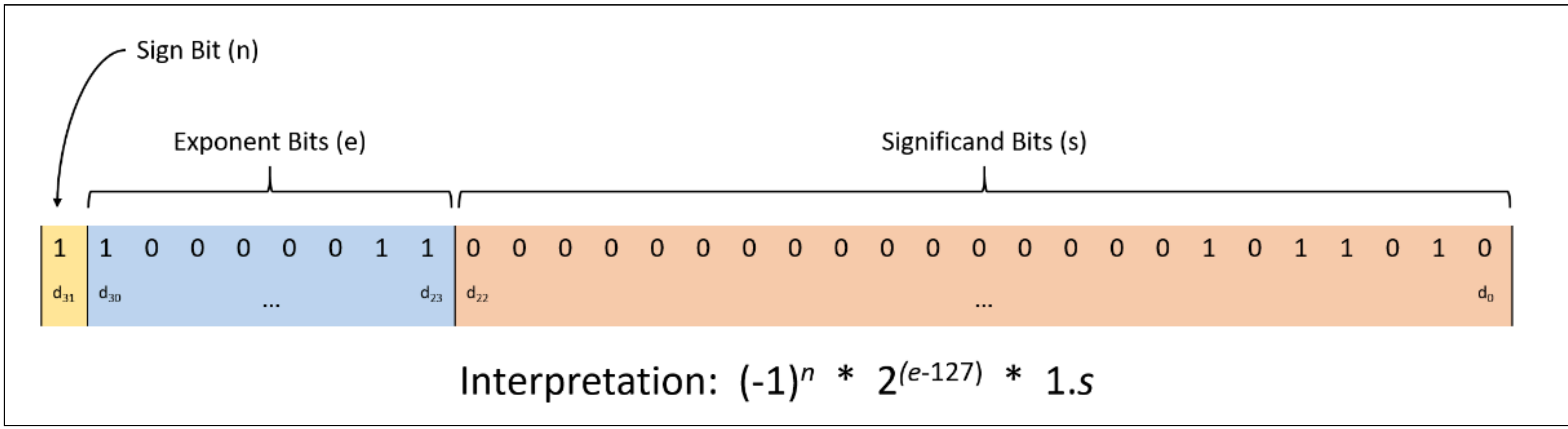
Applying the formula for converting 0b000101.10 to decimal shows:

$$(0 \times 2^5) + (0 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2})$$

$$= 0 + 0 + 0 + 4 + 0 + 1 + 0.5 + 0 = 5.5$$

1. `(0.75 / 2) * 3     =     0.75`
2. `(0.75 * 3) / 2     =     1.00`

4.8.2. Floating-Point Representation



Ch 4b