

Recitation 3

Computer Architecture (section 1)

Number Systems

- A standard way to represent values.
 - You are all familiar with the *decimal number system*.
 - More importantly, we will discuss *positional number systems*.
- *Base* of a number system refers to the number of symbols.

2,839,239,794₁₀

110100010111₂

0123456789ABCDEF₁₆

1111111₁

The Decimal Number System

3458_{10}

The Decimal Number System

3×10^3

3458_{10}

The Decimal Number System

The diagram illustrates the decomposition of the decimal number 3458 into its thousands and hundreds place values. Two red lines originate from the '3' and '4' digits of 3458 and point to the terms 3×10^3 and 4×10^2 respectively. The number 3458 is followed by a subscript 10, indicating it is in base 10.

$$3 \times 10^3 + 4 \times 10^2$$

3458₁₀

The Decimal Number System

The diagram illustrates the expansion of the decimal number 3458 into its place value components. At the top, the number 3458 is shown with a subscript 10. Three red lines branch downwards from the digits 3, 4, and 5 to their respective place value terms: 3×10^3 , 4×10^2 , and 5×10^1 . The digit 8 is not expanded in this diagram.

$$3458_{10}$$
$$3 \times 10^3 + 4 \times 10^2 + 5 \times 10^1$$

The Decimal Number System

The diagram illustrates the expansion of the decimal number 3458 into its positional components. The number 3458 is shown at the top, with a subscript 10 indicating its base. Four red lines branch out from the digits 3, 4, 5, and 8 to their respective terms in the sum below: 3×10^3 , 4×10^2 , 5×10^1 , and 8×10^0 . The terms are separated by plus signs, showing that the number 3458 is equal to the sum of these four terms.

$$3458_{10} = 3 \times 10^3 + 4 \times 10^2 + 5 \times 10^1 + 8 \times 10^0$$

The Decimal Number System

The diagram illustrates the expansion of the decimal number 3458 into its place value components. Red lines connect the digits of 3458 to their respective terms in the expansion below.

$$3458_{10}$$
$$3 \times 10^3 + 4 \times 10^2 + 5 \times 10^1 + 8 \times 10^0$$
$$3000 + 400 + 50 + 8$$

Decimal to Binary Example

$$91_{10}$$

Decimal to Binary Example

Just keep dividing by 2

$$91_{10}$$

Decimal to Binary Example

Just keep dividing by 2

$$\begin{array}{r} 91_{10} \\ 45 \text{ R } 1 \end{array}$$

Decimal to Binary Example

Just keep dividing by 2

$$\begin{array}{r} 91_{10} \\ 45 \text{ R } 1 \\ 22 \text{ R } 1 \end{array}$$

Decimal to Binary Example

Just keep dividing by 2

$$\begin{array}{r} 91_{10} \\ 45 \text{ R } 1 \\ 22 \text{ R } 1 \\ 11 \text{ R } 0 \end{array}$$

Decimal to Binary Example

Just keep dividing by 2

$$\begin{array}{r} 91_{10} \\ 45 \text{ R } 1 \\ 22 \text{ R } 1 \\ 11 \text{ R } 0 \\ 5 \text{ R } 1 \end{array}$$

Decimal to Binary Example

Just keep dividing by 2

91_{10}
45 R 1
22 R 1
11 R 0
5 R 1
2 R 1

Decimal to Binary Example

Just keep dividing by 2

91_{10}
45 R 1
22 R 1
11 R 0
5 R 1
2 R 1
1 R 0

Decimal to Binary Example

Just keep dividing by 2

91_{10}
45 R 1
22 R 1
11 R 0
5 R 1
2 R 1
1 R 0
0 R 1

Decimal to Binary Example

Just keep dividing by 2

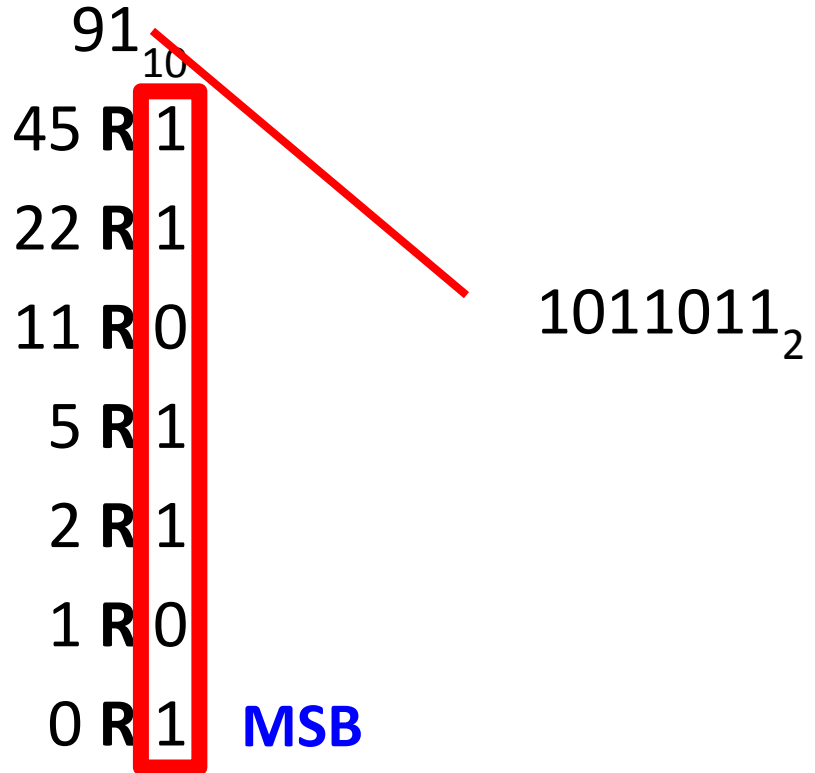
		91	₁₀
45	R	1	
22	R	1	
11	R	0	
5	R	1	
2	R	1	
1	R	0	
0	R	1	MSB

Decimal to Binary Example

Just keep dividing by 2

91			
	10		
45	R	1	
22	R	1	
11	R	0	
5	R	1	
2	R	1	
1	R	0	
0	R	1	MSB

1011011₂



Binary to Decimal Example

10011_2

Binary to Decimal Example

10011_2

1×2^4

Binary to Decimal Example

$$10011_2$$

$1 \times 2^4 + 0 \times 2^3$

Binary to Decimal Example

$$10011_2$$

1 x 2⁴ + 0 x 2³ + 0 x 2²

Binary to Decimal Example

$$10011_2$$

The diagram illustrates the expansion of the binary number 10011_2 into its decimal components. Red lines connect the digits of the binary number to their respective terms in the sum:

- The leftmost '1' is connected to 1×2^4 .
- The first '0' is connected to 0×2^3 .
- The second '0' is connected to 0×2^2 .
- The rightmost '1' is connected to 1×2^1 .

$$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1$$

Binary to Decimal Example

$$10011_2$$
$$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

Binary to Decimal Example

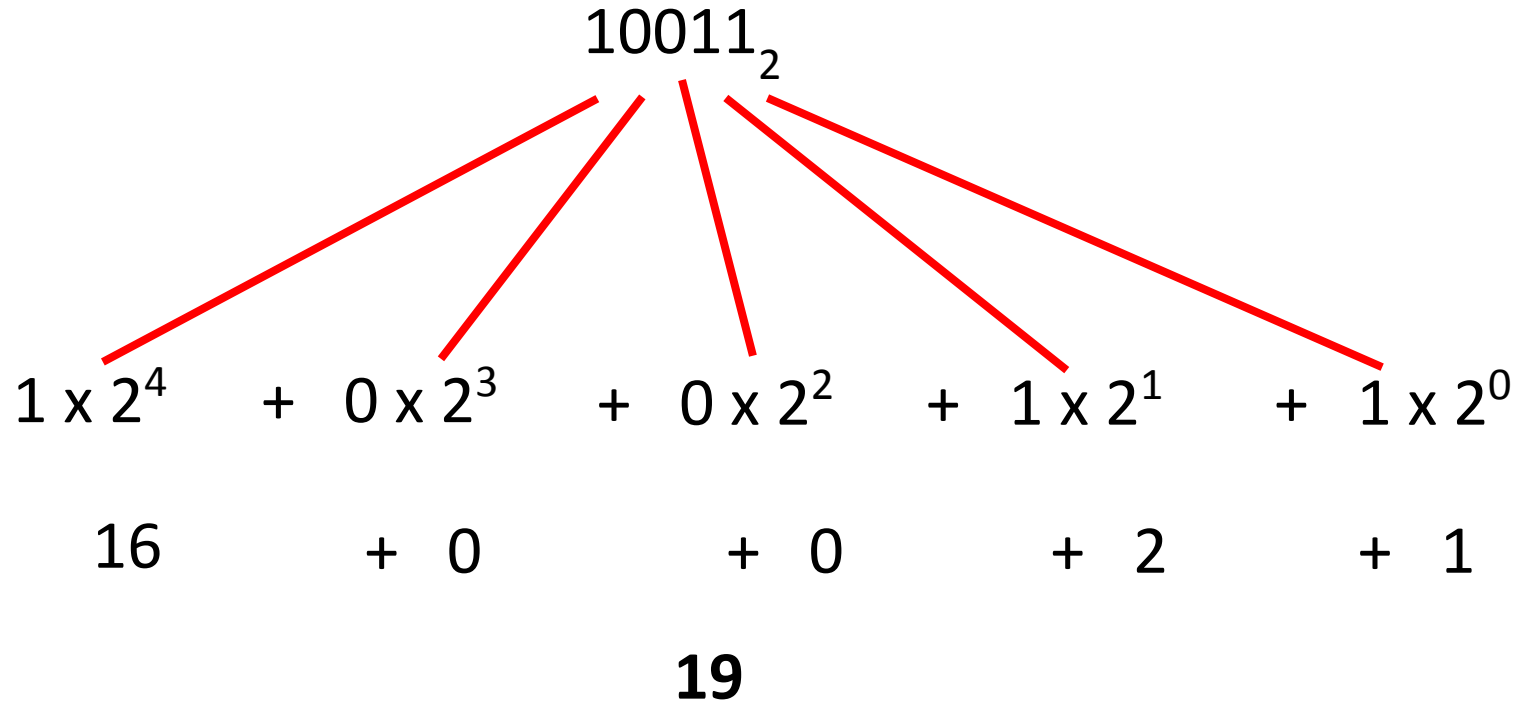
Diagram illustrating the conversion of the binary number 10011_2 to its decimal equivalent. The binary number is expanded into a sum of products, where each bit is multiplied by its corresponding power of 2:

$$1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

The corresponding decimal values are shown below the terms:

$$16 + 0 + 0 + 2 + 1$$

Binary to Decimal Example



Binary to Decimal Example

Use this method for
any base b to decimal

$$\begin{array}{cccccc} & & 10011_2 & & & \\ & / & | & | & \backslash & \\ 1 \times 2^4 & + & 0 \times 2^3 & + & 0 \times 2^2 & + & 1 \times 2^1 & + & 1 \times 2^0 \\ 16 & + & 0 & + & 0 & + & 2 & + & 1 \\ & & & & & & & & \mathbf{19} \end{array}$$

Hexadecimal to Decimal Example

$$2DF_{16}$$

Hexadecimal to Decimal Example

 $2DF_{16}$

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

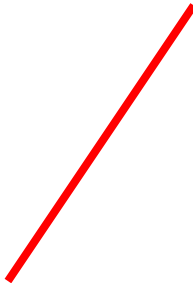
8	9	10	11	12	13	14	15
8	9	A	B	C	D	E	F

Hexadecimal to Decimal Example

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

8	9	10	11	12	13	14	15
8	9	A	B	C	D	E	F

$2DF_{16}$



2×16^2

Hexadecimal to Decimal Example

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

8	9	10	11	12	13	14	15
8	9	A	B	C	D	E	F

$$2DF_{16} = 2 \times 16^2 + 13 \times 16^1$$

Hexadecimal to Decimal Example

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

8	9	10	11	12	13	14	15
8	9	A	B	C	D	E	F

2DF₁₆

$$2 \times 16^2 + 13 \times 16^1 + 15 \times 16^0$$

Hexadecimal to Decimal Example

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

8	9	10	11	12	13	14	15
8	9	A	B	C	D	E	F

2DF₁₆

$$2 \times 16^2 + 13 \times 16^1 + 15 \times 16^0$$
$$512 + 208 + 15$$

Hexadecimal to Decimal Example

0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

8	9	10	11	12	13	14	15
8	9	A	B	C	D	E	F

$$\begin{array}{r} 2DF_{16} \\ \swarrow \quad \downarrow \quad \searrow \\ 2 \times 16^2 \quad + \quad 13 \times 16^1 \quad + \quad 15 \times 16^0 \\ 512 \quad \quad \quad + \quad 208 \quad \quad \quad + \quad 15 \\ \hline 735 \end{array}$$

Binary to Hexadecimal Example

11010010₂

Binary to Hexadecimal Example

**1 hex symbol can
represent 4 bits (2^4)**

11010010₂

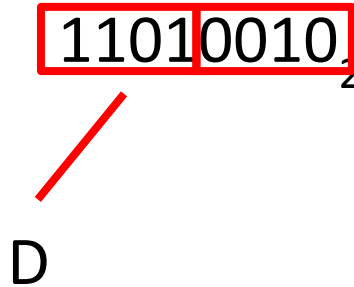
Binary to Hexadecimal Example

**1 hex symbol can
represent 4 bits (2^4)**

11010010₂

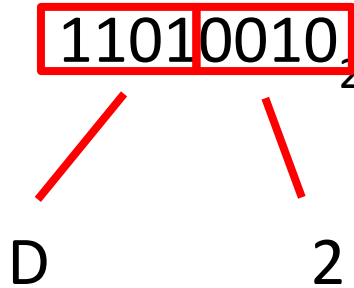
Binary to Hexadecimal Example

1 hex symbol can
represent 4 bits (2^4)



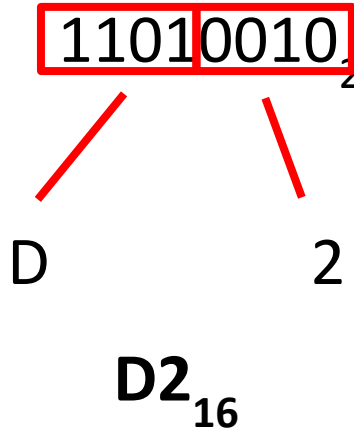
Binary to Hexadecimal Example

1 hex symbol can
represent 4 bits (2^4)



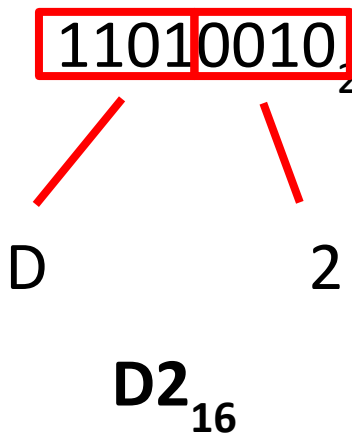
Binary to Hexadecimal Example

1 hex symbol can
represent 4 bits (2^4)



Binary to Hexadecimal Example

1 hex symbol can
represent 4 bits (2^4)



Hex to binary is no
different

Properties of Binary Representation

- Even or odd?
 - Look at the LSB.

1001011**1**₂ **Odd: 151**

Properties of Binary Representation

- Even or odd?

- Look at the LSB.

10010111_2 **Odd: 151**

- Bitwise shifts

- Left = Multiply by 2.
- Right = Divide by 2.

100101110_2 **302**

001001011_2 **75**

Properties of Binary Representation

- Even or odd?
 - Look at the LSB.
$$10010111_2 \quad \text{Odd: } 151$$
- Bitwise shifts
 - Left = Multiply by 2.
 - Right = Divide by 2.
$$100101110_2 \quad 302$$
$$001001011_2 \quad 75$$
- Multiplying any two n bit binary values requires to most $2n$ bits.
 - Adding requires at most $n+1$ bits.

Computing in Binary

- The smallest addressable unit of data on computer is a byte.
 - This is precisely 8 bits.
 - We can represent 2^8 (256) values.
- If we want to talk bits, we need to be a *bit* more clever.

```
// sum primes up to limit by sieving
long long sum_primes(long long limit){
    long long sum = 0;
    bool* marks = malloc(sizeof(bool)*(limit+1));
    if(marks == NULL)
    {
        perror("Malloc failed!");
        abort();
    }
    for(long long i=0; i < limit+1; i++){
        marks[i] = false;
    }
    for(long long i = 2; i <= limit; i++){

        if(!marks[i]){
            sum+=i;

            for(long long m=2*i; m <= limit; m += i)
            {
                marks[m] = true;
            }
        }
    }
    free(marks);
    return sum;
}
```

Computing in Binary

- The smallest addressable unit of data on computer is a byte.
 - This is precisely 8 bits.
 - We can represent 2^8 (256) values.
- If we want to talk bits, we need to be a *bit* more clever.

```
// sum primes up to limit by sieving
long long sum_primes(long long limit){
    long long sum = 0;
    bool* marks = malloc(sizeof(bool)*(limit+1));
    if(marks == NULL)
    {
        perror("Malloc failed!");
        abort();
    }
    for(long long i=0; i < limit+1; i++){
        marks[i] = false;
    }
    for(long long i = 2; i <= limit; i++){

        if(!marks[i]){
            sum+=i;

            for(long long m=2*i; m <= limit; m += i)
            {
                marks[m] = true;
            }
        }
    }
    free(marks);
    return sum;
}
```

**7 bits are
unused...**

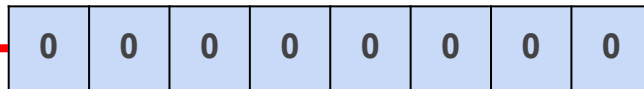
The Bitmap

Idea: Each bit of a byte represents a true or false value.

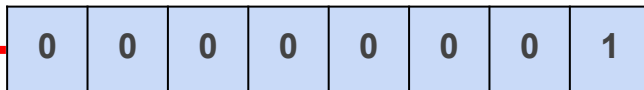
The Bitmap

Idea: Each bit of a byte represents a true or false value.

```
bool b = false;
```

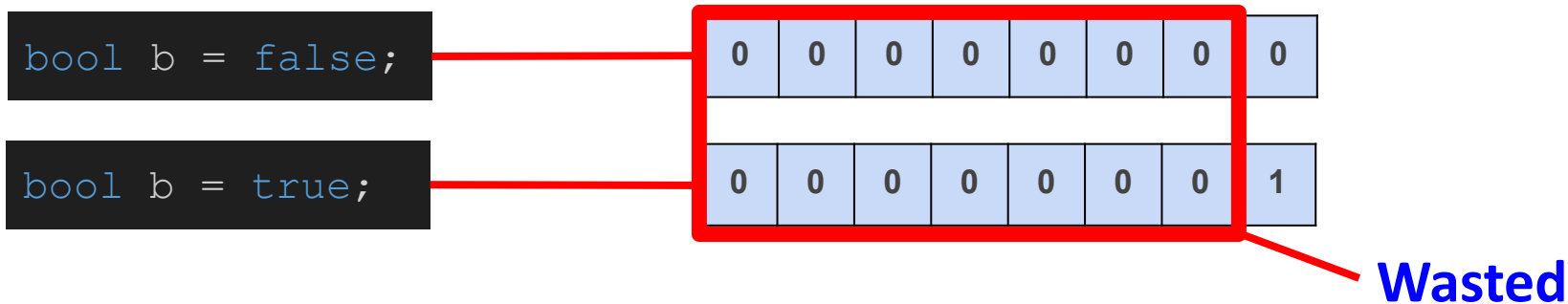


```
bool b = true;
```



The Bitmap

Idea: Each bit of a byte represents a true or false value.



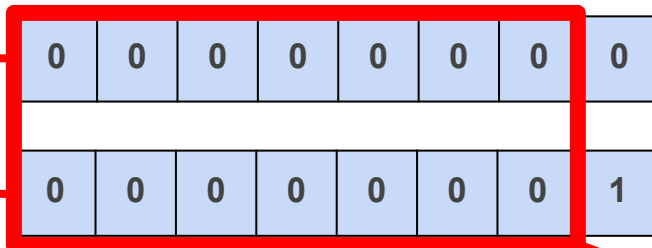
The Bitmap

Idea: Each bit of a byte represents a true or false value.

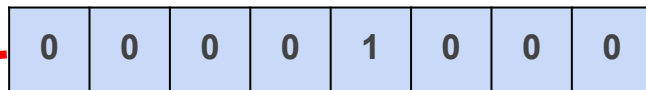
```
bool b = false;
```

```
bool b = true;
```

```
char c;  
set(&c, 3, true);  
bool b = get(c, 3)
```



Wasted



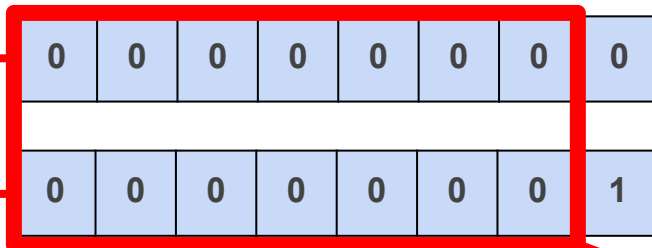
The Bitmap

Idea: Each bit of a byte represents a true or false value.

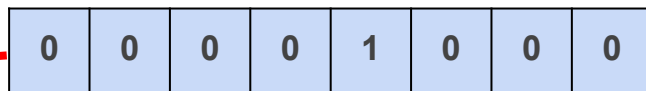
```
bool b = false;
```

```
bool b = true;
```

```
char c;  
set(&c, 3, true);  
bool b = get(c, 3)
```



Wasted



Now each byte can hold 8 true/false values