University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 120: Introduction to Computing

The Unsigned Representation

We Can Represent Anything with Bits

Recall: All information in a computer is **represented with bits**.

We can represent anything with bits.*

useful examples: integers
real numbers
human language characters
(alphabet, digits, punctuation)

Important: **Computers do not "know" the meaning of the bits!**

* A computer only stores a finite number of bits, of course!

How Do We Decide What to Represent?

Let's think about integer (whole number) representations.

What numbers should we represent?
- Some random set?
- Everyone in our class' favorite number (mine is 42!)?
- A contiguous set starting with 0?

Does the Representation Matter?

We want computers to do arithmetic.

How does a representation affect arithmetic?
- Imagine that we represent numbers in the range **[100, 131]**.
- We need **5 bits** (32 different numbers).
- What happens if we add two numbers?
- Can we represent the sum using the same representation?

Choose a contiguous range including 0.

Human Representations are Good Choices

Let's borrow a human representation, **base 2** from mathematics.

For example,

17_{10} = **10001_2**

42_{10} = **101010_2**

1000_{10} = **1111101000_2**

The subscripts indicate the base.

But computers have no "blank" bits!

The Unsigned Representation: Base 2 with Leading 0s

Use leading 0s to fix the number of bits (to **N**).

Result: the **N-bit unsigned representation**.

Using the 8-bit unsigned representation,

17_{10} = **00010001**

42_{10} = **00101010**

1000_{10} = Cannot be represented!

What Can the Unsigned Representation Represent?

What range of integers can be represented with the ***N*-bit unsigned representation**?

- smallest value… all 0s
- largest value … all 1s

Note that **$100...000_2$** (**N** 0s after a 1) is **2^N**.

The range is thus **$[0, 2^N - 1]$**.

Use a Polynomial to Convert to Decimal

How can we **calculate the decimal number represented by a bit pattern** in an unsigned representation?

Remember the place values.

Let's name the bits of the bit pattern:

$$a_5\ a_4\ a_3\ a_2\ a_1\ a_0$$

Multiply each bit by its place value, then sum:

$$a_5 32 + a_4 16 + a_3 8 + a_2 4 + a_1 2 + a_0 1$$
$$= a_5 2^5 + a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0$$

What about Converting from Decimal?

What about finding the bit pattern that represents a decimal number **D** using an unsigned representation?

Seem harder?

Again, name our bits a_i.

In the unsigned representation, every bit pattern represents a different number.

Thus the a_i that represent **D** are unique.

Use the Same Polynomial to Convert from Decimal

The decimal number is given by

$D = a_5 2^5 + a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0$

All terms in the sum except for the last are even (they are multiples of 2).

So, if **D** is odd, $a_0 = 1$.

And if **D** is even, $a_0 = 0$.

We subtract out a_0, divide by 2, and use the same reasoning until we run out of digits.

Example: the Unsigned Bit Pattern for D = 37.

$37 = a_5 2^5 + a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1 + a_0 2^0$

37 is odd, so $a_0 = 1$.

$(37–1)/2 = (a_5 2^5 + a_4 2^4 + a_3 2^3 + a_2 2^2 + a_1 2^1)/2$

$18 = a_5 2^4 + a_4 2^3 + a_3 2^2 + a_2 2^1 + a_1 2^0$

18 is even, so $a_1 = 0$.

$(18 – 0)/2 = (a_5 2^4 + a_4 2^3 + a_3 2^2 + a_2 2^1)/2$

$9 = a_5 2^3 + a_4 2^2 + a_3 2^1 + a_2 2^0$

Example: the Unsigned Bit Pattern for D = 37.

$9 = a_5 2^3 + a_4 2^2 + a_3 2^1 + a_2 2^0$

9 is odd, so $a_2 = 1$.

$(9 – 1)/2 = (a_5 2^3 + a_4 2^2 + a_3 2^1)/2$

$4 = a_5 2^2 + a_4 2^1 + a_3 2^0$

4 is even, so $a_3 = 0$.

$(4 – 0)/2 = (a_5 2^2 + a_4 2^1)/2$

$2 = a_5 2^1 + a_4 2^0$

Example: the Unsigned Bit Pattern for D = 37.

$2 = a_5 2^1 + a_4 2^0$

2 is even, so **$a_4 = 0$**.

$(2 – 0)/2 = (a_5 2^2)/2$

$1 = a_5 2^0$

Putting the bits together, we obtain

37_{10} = **`100101`**

Note: be sure to put the bits in the right order!

Example: the Unsigned Bit Pattern for D = 137.

We don't need to write the polynomial…

137 (odd)		→ 1
(137 – 1) / 2	= 68	→ 0
(68 – 0) / 2	= 34	→ 0
(34 – 0) / 2	= 17	→ 1
(17 – 1) / 2	= 8	→ 0
(8 – 0) / 2	= 4	→ 0
(4 – 0) / 2	= 2	→ 0
(2 – 0) / 2	= 1	→ 1
(1 – 1) / 2	= 0	(done)

137_{10} = **`10001001`**

Read the bits from bottom to top (and add leading 0s if needed).

