

Unit 1

Integer Representation

Skills & Outcomes

- You should master (understand + apply)
 - **Unsigned** binary number to and from decimal
 - **Combinations** that can be made with n bits
 - **2's complement** binary to and from decimal
 - Bit sequences to and from **hexadecimal**
 - Predict the outcome & perform **casting**

0's and 1's

DIGITAL REPRESENTATION

Information Representation

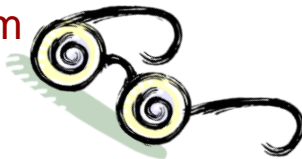
- Information in computer systems is represented as bits
 - bit = (_____) = 0 or 1
- A single bit can only represent 2 values
To represent more options we use sequences of bits
 - Common sequences:
8-bits (aka a "byte"), 16-bit, 32-bit and 64-bits
- Kinds of information
 - Numbers, text, code/instructions, sound, images/videos

Interpreting Binary Strings

- Given a sequence of 1's and 0's, you need to know the **representation system** being used, before you can understand the value of those 1's and 0's.
- Information (value) = _____**

01000001 = ?

Unsigned
Binary system



65 decimal

x86 Assembly
Instruction



`inc %ecx`

(Add 1 to the ecx register)

ASCII
system



'A'_{ASCII}

Binary Representation Systems

- **Integer Systems**

- Unsigned
 - Unsigned binary
- Signed
 - Signed magnitude
 - 2's complement
 - *Excess-N**
 - *1's complement**

- **Floating Point**

- For very large and small (fractional) numbers

- **Codes**

- Text
 - ASCII / Unicode
 $(01000001)_2 = (65)_{10} = \text{"A"}$
- Decimal Codes*
 - BCD (Binary Coded Decimal)
8421 code:
 $(0100)_2(0001)_2 = (4)_{10}(1)_{10} = (41)_{10}$

* = Not covered in this class

Data Representation

- In C/C++ variables can be of different types and sizes
 - Integer Types on 32-bit (64-bit) architectures

C Type (Signed)	C Type (Unsigned)	Bytes	Bits	x86 Name
char	unsigned char	1	8	_____
short	unsigned short	2	16	_____
int / int32_t †	unsigned / uint32_t †	4	32	_____
long	unsigned long	4 (8)	32 (64)	double (quad) word
long long / int64_t †	unsigned long long / uint64_t †	8	64	quad word
char *	-	4 (8)	32 (64)	double (quad) word
int *	-	4 (8)	32 (64)	double (quad) word

- Floating Point Types

† = defined in stdint.h

C Type	Bytes	Bits	x86 Name
float	4	32	_____
double	8	64	_____

Using power-of-2 place values

UNSIGNED BINARY TO DECIMAL

Number Systems

- Unsigned binary follows the rules of **positional number systems**
- A positional number systems consist of
 1. _____
 2. _____
- **Humans:** Decimal (base 10): 0,1,2,3,4,5,6,7,8,9
- **Computers:** Binary (base 2): 0,1
- **Humans working with computers:**
 - Octal (base 8): 0,1,2,3,4,5,6,7
 - Hexadecimal (base 16): 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
(digits A through F encode values **10 through 15**)

Anatomy of a Decimal Number

- A number consists of a string of explicit coefficients (digits).
- Each coefficient is multiplied by an implicit place value r^n
- The value of a decimal number (string of decimal coefficients) is the **sum of each coefficient times its place value**

radix
(base)

$$(934)_{10} = 9 * 10^2 + 3 * 10^1 + 4 * 10^0 = 934$$

Explicit coefficients

Implicit place values

$$(3.52)_{10} = 3 * 10^0 + 5 * 10^{-1} + 2 * 10^{-2} = 3.52$$

Anatomy of an Unsigned Binary Number

- Same as decimal but now the coefficients are 1 and 0 and the place values are the powers of 2

Most Significant Digit (MSB) Least Significant Bit (LSB)

$$(1011)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$

radix (base) coefficients place values = powers of 2

Binary Examples

$$\begin{array}{cccccc} \underline{1} & \underline{0} & \underline{0} & \underline{1} & \underline{.} & \underline{1} \\ \hline & 8 & 4 & 2 & 1 & .5 \end{array} \quad =$$

$$\begin{array}{ccccccc} \underline{1} & \underline{0} & \underline{1} & \underline{1} & \underline{0} & \underline{0} & \underline{0} & \underline{1} \\ \hline & 128 & 32 & 16 & & & & 1 \end{array} \quad =$$

Powers of 2 (memorize these!)

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4$$

$$2^3 = 8$$

$$2^4 = 16$$

$$2^5 = 32$$

$$2^6 = 64$$

$$2^7 = 128$$

$$2^8 = 256$$

$$2^9 = 512$$

$$2^{10} = 1024$$

1024 512 256 128 64 32 16 8 4 2 1

General conversion from unsigned base- r to decimal

- An unsigned number in base r has place values/weights that are the powers of r
- Denote the coefficients as: a_i

$$(a_3 a_2 a_1 a_0 . a_{-1} a_{-2})_r = a_3 * r^3 + a_2 * r^2 + a_1 * r^1 + a_0 * r^0 + a_{-1} * r^{-1} + a_{-2} * r^{-2}$$

Left-most digit =
Most Significant
Digit (MSD)

Right-most digit =
Least Significant
Digit (LSD)

$$N_r \Rightarrow \sum_i (a_i * r^i) \Rightarrow D_{10}$$

Number in base r

Decimal Equivalent

Examples base-8 and base-16

$$(746)_8 =$$

$$(1A5)_{16} =$$

$$(AD2)_{16} =$$

Subtracting powers of 2

UNSIGNED DECIMAL TO BINARY

Decimal to Unsigned Binary

- To convert a decimal number, x , to binary:
 - Only coefficients are 0 and 1. Simply find place values that **add up to the desired values**, starting with larger place values and proceeding to smaller values
 - Place a 1 in those place values and 0 in all others

$$25_{10} = \begin{array}{cccccc} \hline & & & & & \\ \hline & 32 & 16 & 8 & 4 & 2 & 1 \\ \hline \end{array}$$

For 25_{10} the place value 32 is **too large** to include so we include 16. Including 16 means we have to make 9 left over. Include 8 and 1.

Decimal to Unsigned Binary

$$73_{10} = \begin{array}{cccccccc} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ \hline & 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \end{array}$$

$$87_{10} = \begin{array}{cccccccc} \hline & & & & & & & & \end{array}$$

$$145_{10} = \begin{array}{cccccccc} \hline & & & & & & & & \end{array}$$

$$0.625_{10} = \begin{array}{cccccc} . & \hline & & & & & \\ & .5 & .25 & .125 & .0625 & .03125 \end{array}$$

General Algorithm: Integer Part

Binary digits **right-to-left**: divide by 2, take rest

$$87 / 2 = 43 \text{ r } \mathbf{1}$$

$$43 / 2 = 21 \text{ r } \mathbf{1}$$

$$21 / 2 = 10 \text{ r } \mathbf{1}$$

$$10 / 2 = 5 \text{ r } \mathbf{0}$$

$$5 / 2 = 2 \text{ r } \mathbf{1}$$

$$2 / 2 = 1 \text{ r } \mathbf{0}$$

$$1 / 2 = 0 \text{ r } \mathbf{1}$$

$$\text{So, } (87)_{10} = (\mathbf{1010111})_2$$

General Algorithm: Fraction

Digits **left-to-right**: multiply by 2, take integer part

$$0.1 * 2 = 0.2$$

$$0.2 * 2 = 0.4$$

$$0.4 * 2 = 0.8$$

$$0.8 * 2 = 1.6$$

$$0.6 * 2 = 1.2$$



So, $(0.1)_{10} = (0.0\ 0011\ 0011\ \dots)_2 = 0.00011$

Decimal to Another Base

- To convert a decimal number, x , to base r :
 - Use the place values of base r (powers of r). Starting with largest place values, fill in **coefficients** that, multiplied by the implicit place value, sum up to desired decimal value.

$$75_{10} = \frac{\quad}{256} \frac{\quad}{16} \frac{\quad}{1} \quad \text{hex}$$

The 2^n rule

UNIQUE COMBINATIONS

Unique Combinations

- Given n digits of base r , how many unique numbers can be formed? _____
 - What is the range? **[0 to r^n-1]**

2-digit, decimal numbers ($r=10, n=2$)

 0-9 0-9

100 combinations:
 00-99

3-digit, decimal numbers ($r=10, n=3$)

1000 combinations:
 000-999

4-bit, binary numbers ($r=2, n=4$)

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

16 combinations:
 0000-1111

6-bit, binary numbers
 ($r=2, n=6$)

64 combinations:
 000000-111111

Main Point: Given n digits of base r , _____ unique numbers can be made with the range **0 to r^n-1**

Corollary: Unsigned with all 1's

- What is the decimal value of an unsigned **binary string of all 1's**?
 - “111” is $2^3-1 = 7$ (4 + 2 + 1)
 - “1111” is $2^4-1 = 15$ (8 + 4 + 2 + 1)
 - “11111111” is $2^8-1 = 255$
 - “11111111 11111111” is $2^{16}-1 = 65535$
- What is the decimal value of an unsigned **hex string of all F's**?
 - “FF” is $16^2-1 = (2^4)^2 - 1 = 2^8 - 1 = 255$
 - “FFFF” is $16^4-1 = (2^4)^4 - 1 = 2^{16} - 1 = 65535$

Range of C data types

- For an unsigned integer data type: $[0, 2^n-1]$
- For a signed integer data type: $[-2^{n-1}, 2^{n-1}-1]$
 - We use half combinations for negative and half for positive
 (0 is considered a positive number by common integer convention)

Bytes	Bits n	Type	Unsigned Range	Signed Range
1	8	[unsigned] char	0 to 255	-128 to +127
2	16	[unsigned] short	0 to 65535	-32768 to +32767
4	32	[unsigned] int	0 to 4,294,967,295 (0 to +4 billion)	-2,147,483,648 to +2,147,483,647 (-2 billion to +2 billion)
8	64	[unsigned] long	0 to 18,446,744,073,709,551,615 (0 to +18 billion billion)	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (-9 to +9 billion billion)
4 (8)	32 (64)	char *	0 to 4,294,967,295 0 to 18,446,744,073,709,551,615	

- How will I ever remember those ranges?

Approximating Large Powers of 2

- We often need to find the decimal approximation of a large power of 2 like 2^{16} , 2^{32} , etc.
- Use following approximations:
 - 1 kilo- is _____
 - 1 Mega- is _____
 - 1 Giga- is _____
 - 1 Tera- is _____
- For other powers of 2, decompose into product of 2^{10} or 2^{20} or 2^{30} and a power of 2 that is less than 2^{10}
 - 16-bit word: 64k numbers
 - 32-bit dword: 4G numbers
 - 64-bit qword: 16 million trillion numbers
(or 16 billion billion numbers)

$$2^{16} =$$

$$2^{24} =$$

$$2^{28} = 2^8 * 2^{20} \\ \approx 256 * 10^6 = 256,000,000$$

$$2^{32} =$$

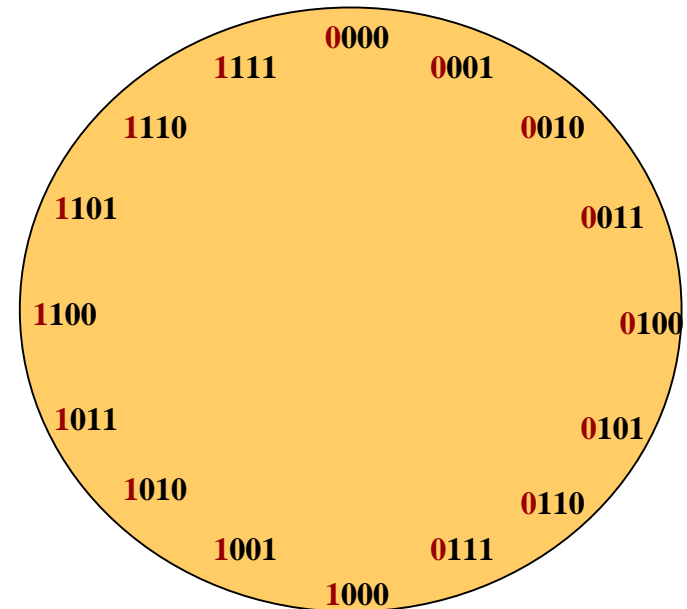
$$2^{64} = 2^4 * 2^{60} \\ = 2^4 * 2^{20} * 2^{40} \approx 16 * 10^6 * 10^{12} \\ = 2^4 * 2^{30} * 2^{30} \approx 16 * 10^9 * 10^9$$

2's complement

SIGNED NUMBERS TO DECIMAL

Signed numbers

- Systems used to represent signed numbers split binary combinations in half
 - half for positive, incl. zero
 - half for negative
- Generally, positive and negative numbers are distinguished by the _____
 - _____ means negative
 - _____ means positive

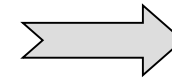


2's Complement System

- Only change to place values: negative value of MSB
 - MSB of 1 has value of _____

**4-bit
Unsigned**

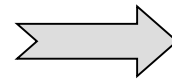
Bit 3	Bit 2	Bit 1	Bit 0
8	4	2	1



**0 to 15
(0000 to 1111)**

**4-bit
2's complement**

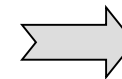
Bit 3	Bit 2	Bit 1	Bit 0
—	4	2	1



**-8 to +7
(1000 to 0111)**

**8-bit
2's complement**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
—	64	32	16	8	4	2	1



**-128 to +127
(10000000 to 01111111)**

2's Complement Examples

**4-bit
2's complement**

1	0	1	1	= -5
-8	4	2	1	
0	0	1	1	= +3
-8	4	2	1	
1	1	1	1	= -1
-8	4	2	1	

Notice that +3 in 2's comp.
is the **same** as in the
unsigned system

**8-bit
2's complement**

1	0	0	0	0	0	0	1	= -127
-128	64	32	16	8	4	2	1	
0	0	0	1	1	0	0	1	= +25
-128	64	32	16	8	4	2	1	

Important: Positive numbers have the same representation in 2's complement as in normal unsigned binary

2's Complement Range

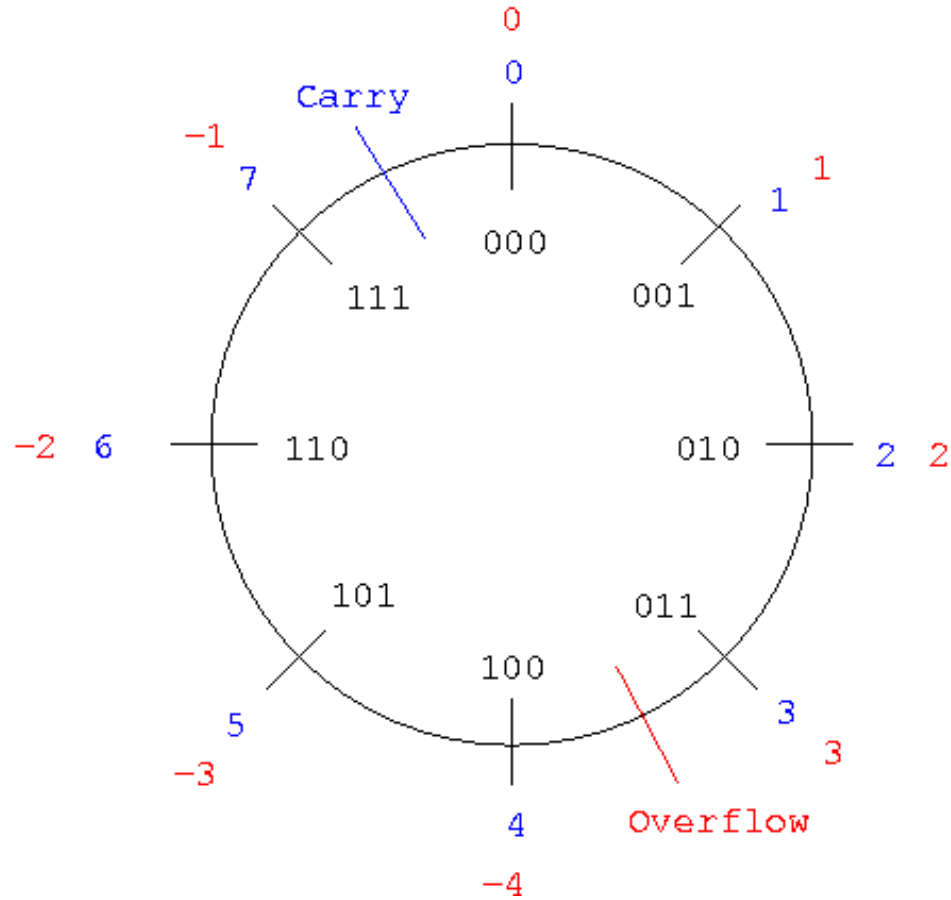
- Given n bits...
 - Max positive value = 011...11
 - Includes all n-1 positive place values
 - Max negative value = 100...00
 - Includes only the negative MSB place value

Range with n bits of 2's complement

$$[-2^{n-1} \text{ to } 2^{n-1}-1]$$

- $(0)_{10}$ always encoded as 000...00
- $(\underline{\quad})_{10}$ always encoded as _____
- $(1)_{10}$ always encoded as 000...01

Wheel of Integers



Unsigned and Signed Variables

- In C, unsigned variables use unsigned binary (pos. power-of-2 place values) to represent numbers

$$\begin{array}{cccccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & = +147 \\ \hline 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 & \end{array}$$

- In C, signed variables use the 2's complement system (neg. MSB weight) to represent numbers

$$\begin{array}{cccccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & = -109 \\ \hline -128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 & \end{array}$$

IMPORTANT NOTE

- All computer systems use the 2's complement system to represent signed integers!
- So from now on, if we say an integer is signed, we are actually saying it uses the 2's complement system unless otherwise specified
 - Other systems like "signed magnitude" or "1's complement" exist but will not be used (signed magnitude: sign bit + value, so -1 is 100..01)

Zero and Sign Extension

- Extension is the process of increasing the number of bits used to represent a number **without changing its value**

Unsigned = Zero Extension (**always add leading 0's**):

$$111011 = 00111011$$



Increase a 6-bit number to 8-bit number by zero extending

2's complement = Sign Extension (**replicate sign bit**):

positive $011010 = 00011010$

Sign bit is just repeated as many times as necessary

negative $110011 = 11110011$

Why does it work?

$$111\dots = -128 + 64 + 32 = -32 \text{ and } 1\dots = -32$$

Zero and Sign Truncation

- Truncation is the process of decreasing the number of bits used to represent a number **without changing its value**

Unsigned = Zero Truncation (Remove leading 0's):

$$\text{~~00~~111011 = 111011}$$

Decrease an 8-bit number to 6-bit number by truncating 0's. Can't remove a '1' because value is changed

2's complement = Sign Truncation (Remove copies of sign bit):

positive $\text{~~00~~011010 = 011010}$

negative $\text{~~111~~10011 = 10011}$

Any **copies** of the MSB can be removed without changing the numbers value. **Be careful not to change the sign by cutting off ALL the sign bits.**

Shortcuts for Converting Binary to Hexadecimal

SHORTHAND FOR BINARY

Binary and Hexadecimal

- Hex is base 16 which is 2^4
- 1 Hex digit (?)₁₆ can represent: _____
- 4 bits of binary (? ? ? ?)₂ can represent:

- Conclusion...
1 hex digit = _____ bits

Binary to Hex

- Make groups of 4 bits starting from radix point and working outward
- Add leading or trailing 0's where necessary
- Convert each group of 4 to an hex digit

000101001110.1100
1 4 E C

= 14E.C₁₆

01101011.1010
6 B A

= 6B.A₁₆

Hex to Binary

- Expand each hex digit to a group of 4 bits

$14E.C_{16}$

$\overbrace{000} \overbrace{10100} \overbrace{1110} \overbrace{.1100}$

=
 101001110.11_2

$D93.8_{16}$

$\overbrace{1101} \overbrace{1001} \overbrace{0011} \overbrace{.1000}_2$

=
 110110010011.1_2

(leading or trailing 0's can be discarded)

Use of Hex Representation

- Values in modern computers use many bits!
- We use hexadecimal as a shorthand notation (4 bits = 1 hex digit)
 - 1101 0010 = D2 hex or **0xD2** if you write it in C/C++
 - 0111 0110 1100 1011 = 76CB hex or **0x76CB** in C/C++

Interpreting Hex Strings

- What does the following hexadecimal represent?
- Just like binary, you must know the underlying *representation system* being used before you can interpret a hex value
- Information (value) = Hex + Context (System)

0x41 = ?

Unsigned
Binary system



65 decimal

x86 Assembly
Instruction



`inc %ecx`

(Add 1 to the ecx register)

ASCII
system



'A'_{ASCII}

Hexadecimal & Sign

- If a number is represented in 2's complement (e.g. 10010110) then the MSB of its binary representation would correspond to:
 - 0 for positive numbers (incl. 0)
 - 1 for negative numbers
- If that same 2's complement number is viewed as hex (e.g. 0x96), how can we tell if its value is positive or negative?
 - MSD of _____ = _____
 - MSD of _____ = _____

Hex	Binary	Sign
0	0000	Pos
1	0001	Pos
2	0010	Pos
3	0011	Pos
4	0100	Pos
5	0101	Pos
6	0110	Pos
7	0111	Pos
8	1000	Neg
9	1001	Neg
A	1010	Neg
B	1011	Neg
C	1100	Neg
D	1101	Neg
E	1110	Neg
F	1111	Neg

Implicit and Explicit

APPLICATION: CASTING

Implicit and Explicit Casting

- Use your understanding of zero and sign extension to predict the output

```
int main()
{
    short int v = 0xcfc7;          /* -12345 */
    int vi = v;                   /* ?????? */
    unsigned short uv = 0xcfc7;   /* 53191 */
    unsigned uvi = uv;           /* ?????? */
    printf("vi = %x, uvi = %x\n", vi, uvi);
    return 0;
}
```

Expected Output:

```
vi = ffff cfc7, uvi = _____
```

```
int main()
{
    int x = 53191;                /* 0xcfc7 */
    short sx = x;
    int y = sx;
    char z = x;

    printf("sx = %d, y = %d ", sx, y);
    printf("z = %d\n", z);
    return 0;
}
```

Expected Output:

```
sx = -12345, y = -12345, z = _____
```

Advice

- Casting can be done implicitly and explicitly
- Casting from one system to another applies a new "interpretation" (pair of glasses) on the same bits
- Casting from one size to another will perform extension or truncation (based on the system)