ECE 2049 LECTURE 1

E21 0

TODAY

- INTRO TO NUMBER REPRESENTATIONS

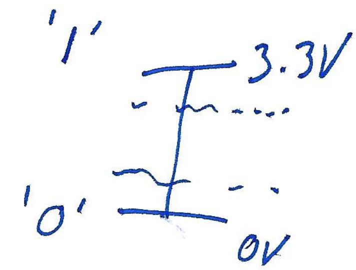
ADMINISTRIVIA

- HW1: ONLINE AFTER CLASS, DUE NEXT TUES
- LAB Ø: STARTS THURSDAY
    - ORDER YOUR PARTS IF YOU HAVE NOT DONE SO ALREADY
    - PLEASE FILL OUT THE PARTS SURVEY TO LET US KNOW YOUR STATUS
- PLEASE FILL OUT THE "BACKGROUND SURVEY" ON THE COURSE WEBSITE

Module 1. Intro to Number Representations

Topics

- How do we store (or "encode") information in digital systems?
- Specifically: how do we store numbers?



First things first: Remembering Digital Logic

Before we can talk about how computing systems are built, we first need to talk about their basic building block: digital logic. In digital logic, information is represented in binary *bits*.

1 BIT = VALUE OF 0 OR 1

Digital logic defines how we can process information using bits:

- LOGICAL (AND, OR, NOT...)
- STORAGE
- ARITHMETIC (+, -, /, ...)

=> COMBINE THESE ELEMENTS TO BUILD MORE COMPLEX COMPONENTS

First things first: *n* bits differentiate among 2^n things.

N BITS $= 2^N$ UNIQUE "CODES"

EX 2 BITS $= 2^2 = 4$ "CODES"
00, 01, 10, 11

Terminology: 1 byte = 8 binary digits = 8 bits (e.g. 10010011)

½ byte = 1 nibble = 4 bits

1 word = 2 (or more) bytes --> MSP430 word = 2 bytes (16 BITS)

1 double word = 2 words (4 bytes on MSP430) (32 BITS)

In computers, ***information*** and ***memory space*** is organized in to multiples of bytes.
But what do the bytes mean?

The meaning of bits and bytes <u>assigned by convention</u>!

>> Under a given coding convention, a byte can represent up to $2^8 = 256$ things

For example, 1 byte (8 bits) could encode:

- A letter in an alphabet

  'D' ASCII

- One or more decimal numbers

  42, -42, -42.5

- The state of eight individual things (one per bit)

  "BIT VECTOR"

  ~~BIT~~ BIT6 = 1

  BIT3 = 0

  MOST SIGNIFICANT BIT (MSB) LEAST SIGNIFICANT BIT (LSB)

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

- An *instruction* that tells the CPU to do something:

  1100 0011 = C3h

  => RET ← CPU INSTRUCTION TO RETURN FROM A FUNCTION

We call these conventions **encoding formats**. They represent a kind of contract on how data will be stored and used. As programmers, it is up to us to assign meaning to those bits—which defines what operations we perform on them.

Conversion between Bases and Formats: Binary

Positional Number Systems

We write numbers in a positional system, which can be defined as:

RADIX (BASE)

DECIMAL BASE 10

$$1734 \Rightarrow 1 \times 10^3 + 7 \times 10^2 + 3 \times 10^1 + 4 \times 10^0$$

ddd.ddd

$$D = \sum_{i=-N}^{P-1} d_i r^i$$ WHERE

r : RADIX
d_i : i'TH DIGIT
N : DIGITS LEFT OF .
P : DIGITS RIGHT OF .

For binary numbers, we can write this definition as:

BASE 2: 0, 1

$$B = \sum_{i=-N}^{P-1} b_i (2^i)$$ WHERE $b_i \in \{0,1\}$

Unsigned integers = All bits used to convey magnitude (whole numbers ≥ 0)

Ex. $1001\ 0001_b$ (7 ← MSB, 0 ← LSB) $\Rightarrow 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

$$= 128 + 16 + 1 = \boxed{145d}$$

Decimal to Binary Conversion – Successive Division

↳ DIVIDE BY 2, LOOK AT REMAINDER

Ex. 44

LSB

44/2 = 22 R 0
22/2 = 11 R 0
11/2 = 5 R 1
5/2 = 2 R 1
2/2 = 1 R 0
1/2 = 0 R 1

MSB

$\boxed{101100}b$

Note: To differentiate numbers in different formats, we use notation to denote the radix used wo write it. For binary: 1010_2 or 1010b; decimal: 1010_{10} or 1010d (or just 1010)

Hexadecimal: A common way to write binary numbers

Since working in binary can be cumbersome, we often write numbers in *hexadecimal*, which is base 16.

Simple rule for conversion: ONE HEX "DIGIT" = 4 BITS $2^4 = 16$

--> 1 Hex character represents values from 0 to 15d using digits 0 – Fh

DEC	BIN	HEX	DEC	BIN	HEX
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Ex. 158d =

158/16 = 9 R 14

9/16 = 0 R 9

$\boxed{9Eh}$ = 0x9E

If you memorize anything in this class, memorize these!

Notation: Numbers in hex are written as 1010h or 0x1010

Conversion between hex and binary is piece of cake! Just convert each hex digit to a binary nibble...

1001 1110b = 158d

9 E

1001 = $2^3 + 1 \times 2^0 = 8 + 1$

2^3 2^2 2^1 2^0

8 4 2 1

Or vice versa:

8AC4h = 8 A C 4

$\boxed{1000 \quad 1010 \quad 1100 \quad 0100\ b}$

Note: A modern computer always stores information in binary form. Writing in hex is just a faster way for us to read and write these numbers—the machine's representation is still binary.

How do we store negative numbers? -6

One way: Sign Magnitude integers = n-1 bits used to convey magnitude with "most significant bit" or MSB used for sign. Convention: 0 = +, 1 = -

FOR 8 BITS SMMM MMMM

0|010 0001 = +33
1|010 0001 = -33

0|000 0000 = 0
1|000 0000 = -0 ???

Note: This format has 2 representations of 0 = +0 and -0 !

ABIGUOUS
INEFFICIENT

Another way: Two's Complement integers = More common format for signed integers. For n bits, values range from $-2^{(n-1)}$ to $2^{(n-1)}-1$

How it works:

Positive numbers: Follow same format as unsigned numbers

1026 = 0000 0100 0000 0010b = 0402h

$2^{10} + 2^{1} =$ 1024 + 2 = 1026 ✓

ASSUME 8 BIT NUMBERS

$0+0=0$
$0+1=1$
$1+1=10$

Negative numbers: *Write* magnitude, *Complement* each bit, *Add* 1

-15 =

ONLY IF NEGATIVE

```
  0000 1111     MAGNITUDE
  1111 0000     COMPLEMENT (FLIP ALL BITS)
  0000 0001     ADD 1
  ---------
  1111 0001 b
```

SIGN BIT IS SET

Ex. -1

```
  0000 0001
  1111 1110
  0000 0001
  ---------
  1111 1111 => -1
```

Range of Values for 2's Complement

FOR 16 BITS = (-2^{15}) TO $(2^{15}-1)$ = -32768 TO +32767

```
0111 1111 1111 1111b = +32767
0111 1111 1111 1110b = +32766
       . . .
0000 0000 0000 0001 = 1
0000 0000 0000 0000b = 0
       . . .
1000 0000 0000 0001b
1000 0000 0000 0000b = -32768
```

Ex: Find the 8-bit two's complement representation of 104 and -80

Ex. 104 POSITIVE, TREAT IT LIKE UNSIGNED

~~100~~
104/2 = 52 R 0
52/2 = 26 R 0
⋮

$\boxed{0110\ 1000b}$
$= 2^6 + 2^5 + 2^3 = 104$

Ex. -80 => NEGATIVE, SO NEED 2'S COMP PROCEDURE

80 => 0101, 0000
1010 1111 COMP
+ 0000 0001
$\boxed{1011\ 0000} = -80$

Ex: What are the decimal equivalent values of these 2's complement values

0010 0011b

POSITIVE

$2^5 + 2^1 + 2^0 = 32 + 2 + 1 = \boxed{35d}$

1000 0011b

NEGATIVE, NEED 2'S COMP PROCEDURE

1000 0011
0111 1100
1
0111 1101 = $\boxed{-125}$

$2^8 + 2^4 + \ldots$

Ex: What decimal value does 8008h represent as an…
(a) unsigned integer (b) 2's comp integer (c) sign-magnitude integer

A. $8008h = 1000\ 0000\ 0000\ 1000$

$= 2^{15} + 2^3 = \boxed{32776}$

B. 2's COMPLEMENT

NEGATIVE => 2's COMP PROCEDURE

$$\begin{array}{rl} & 1000\ 0000\ 0000\ 1000 \\ & 0111\ 1111\ 1111\ 0111 \quad \text{COMP} \\ + & 0000\ 0000\ 0000\ 0001 \\ \hline & 0111\ 1111\ 1111\ 1000 \end{array}$$

$= 2^{14} + 2^{13} + \ldots + 2^3 = \boxed{-32760}$

C. SIGN MAGNITUDE

8008h

SIGN = NEGATIVE

MAG = 8

$\Rightarrow \boxed{-8}$

What about things that aren't integers?

Characters

To handle letters and other displayable characters, we need an encoding format to describe how we can represent these values in binary. One very common format for this is ASCII (American Standard Code for Information ~~Exchange~~ INTERCHANGE), which defines a table of binary codes that represent various characters.

ASCII = 7 OR 8 BIT ENCODING

1 ASCII CHARACTER = 1 BYTE

'D' = 44h = 0100 0100b
= 68d

Note: Other formats exists for representing different types of characters (alphabets and character sets for all human languages, emoji, etc.). For information on this, see "Unicode".

IN UNICODE, ONE CHARACTER = ≥1 BYTE

Unicode Examples

Unicode Name	Bit representation	Character
U+00FC LATIN SMALL LETTER U WITH DIAERESIS	C3 BC	ü
U+1F602 FACE WITH TEARS OF JOY	F0 9f 98 82	😂
U+1F363 SUSHI	F0 9F 8D 83	🍣

Non-integer Numbers

In future lectures we will talk about representing non-integer data. These are called fixed-point and floating-point data types, which we will cover soon!

Preview: How is data actually stored in a program?

C defines a set of standard data types to store information. Each datatype has a specific representation, which depends on the compiler and the CPU architecture.

For the MSP430 architecture, the standard datatypes are defined as follows:

DEPENDS ON ARCHITECTURE

```
int a;              // 16-bit two's-complement signed (2 bytes)
unsigned int b;     // 16-bit unsigned integer (2 bytes)
long int c;         // 32-bit signed integer (two's complement) (4 bytes)
char d;             // 8-bit unsigned integer (1 byte)

float e;            // 32-bit IEEE754 single-precision floating point value (4 bytes)
double f;           // 64-bit IEEE754 double-precision floating point value (8 bytes)
```

STANDARD SIZES ON ALL SYSTEMS

Note that the types `char`, `float`, and `double` have the same size on all architectures–these are part of the C standard.

We can use these standard datatypes to hold different kinds of information (signed/unsigned numbers, characters, decimal values), or compose more complex types (like arrays or structs).

Important: The size and type of a variable define the range of values they can represent!

- The value of a variable CANNOT exceed the fixed size of the variable
- Variables will "overflow" or "roll over" if the value exceeds the variable size!

INT $= 2^{16}$ POSSIBLE STATES

FOR UNSIGNED: RANGE $= 0 - (2^{16}-1) = 0 - 65535$

CHAR 2^8 " " $0 - (2^8-1) = 0 - 255$

Ex. CHAR C = 253;

C = C + 1; // 254 $\boxed{1111\ 1110}$

C = C + 1 // 255 $\boxed{1111\ 1111}$

C = C + 1 // 0

$\Rightarrow$ OVERFLOW

$$\begin{array}{r} 1111\ 1111 \\ +\quad\quad\quad 1 \\ \hline 1\ \boxed{0000\ 0000} \end{array}$$