CMPT 295

Chapter 2 of our textbook

Unit - Data Representation

Lab 0 – Review of the Binary numeral system,

the Hexadecimal numeral system

and the Binary ⇔ Hexadecimal Conversion

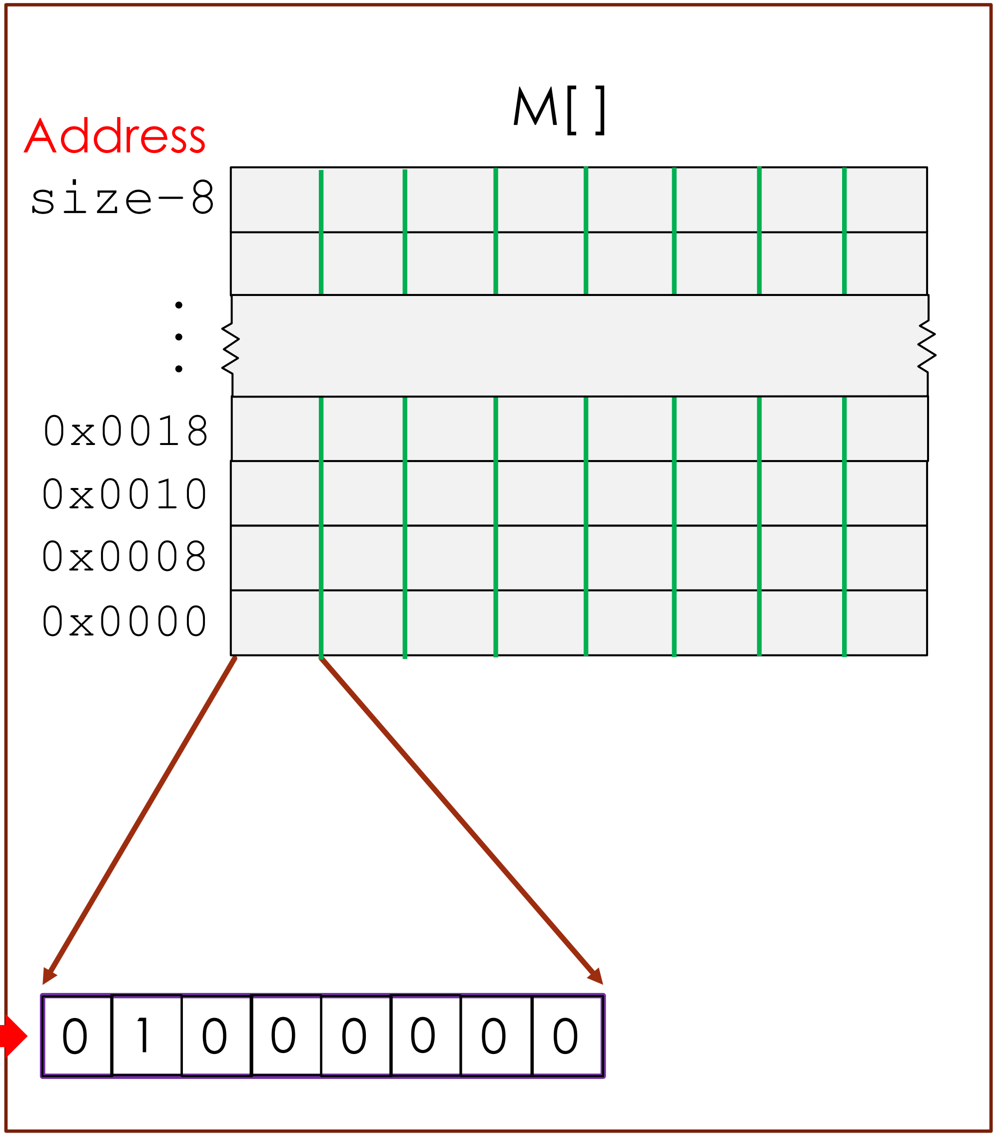
Lab 0 - Objectives and Instructions

- In this lab, we shall review
  - The binary numeral system
  - The hexadecimal numeral system
  - How to represent memory content, i.e., series of bits, using each of the above two numeral systems
  - How to convert from one numeral system to the other: binary ⬄ hexadecimal

- Instructions:
  - Read each slide and answer the questions
  - Solution are located on the last slides of this lab

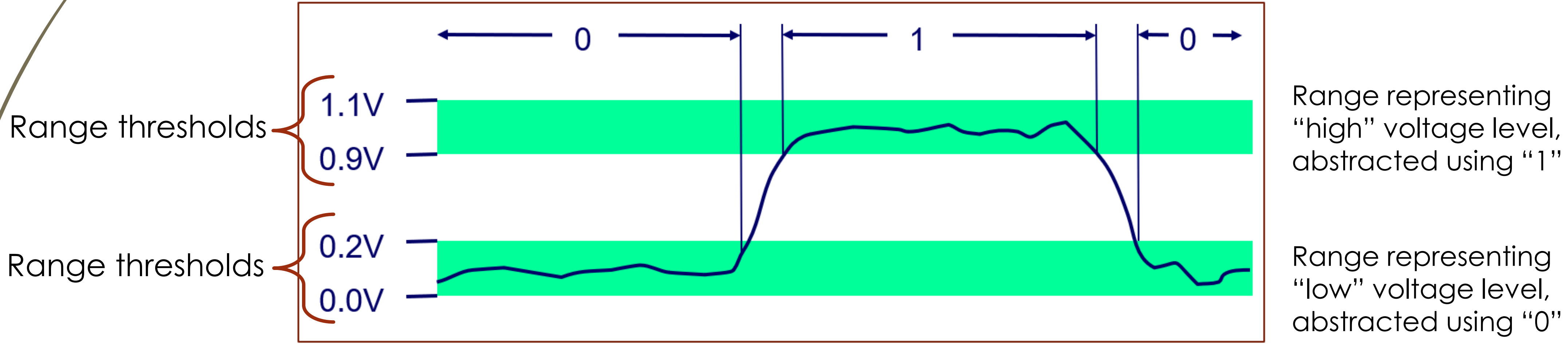
In Lecture 2, we saw …

- … that, typically, in a diagram representing memory,
we represent the content of a byte of memory as a series of memory "***cells***" in which one of two possible values is stored
- … and that these two possible values are represented using '**0**' and '**1**'



Why can only two possible values be stored in a memory "***cell***"?

- Because, as electronic machines, computers operate using two voltage levels, more specifically, two ranges of voltage levels
  - Why ***ranges***? Because voltage levels are transmitted on ***noisy*** wires and this noise creates fluctuations in the value of the two voltage levels, hence a ***range*** of values as opposed to a ***precise*** voltage level value
- These two ranges are abstracted using "**0**" and "**1**":



- However, computers have not always use two values …

Indeed, the ENIAC used ten …

A bit of history

ENIAC: Electronic Numerical Integrator And Calculator

- U. Penn by Eckert + Mauchly (1946)
- Data: 20 × **10-digit** registers
  + ~18,000 vacuum tubes
- To code: manually set switches
  and plug cables
  - Debugging was manual
  - No method to save program for later use
  - Separated code from the data

Source: https://en.wikipedia.org/wiki/ENIAC#/media/File:ENIAC_Penn1.jpg

Review

Why do we abstract these two ranges using ‘**0**’ and ‘**1**’?

We call “memory cell” a bit.

- Because we can use the **binary numeral system**, for which there is already a well-established body of algebra.
- Base: **2**
- Bit (**bi**nary digi**t**) values: **0** and **1**
- Possible content of a byte (8 memory cells/bits):
  - 00000000_2, 00000001_2, 00000010_2 … to 11111111_2 => 2^8 (256) **bit patterns**

Also called ***bit vectors***

- *Drawback of using ‘0’ and ‘1’* (binary numbers) to represent memory content:
  - What number is this -> $10011001100100101000101010001000_2$?
    - As you can see, the drawback is that …
      - *Such bit patterns (binary numbers) are difficult to read*
      - *Such bit patterns (binary numbers) are lengthy to write -> not very compact*

Both very error prone!

A solution: hexadecimal numbers

- Base: 16
- Values: 0, 1, 2, …, 9, A (or a), B (or b), C (or c), D (or d), E (or e), F (or f)
- Possible patterns stored in a byte (8 bits):
  - 00_{16} , 01_{16}, 02_{16} … FF_{16} => 256 bit patterns

Exercise 1. Conversion exercise: binary -> hex

- Convert 1001100 11001001 01000101 01001000_2 (in **C**: 0b1001100110010010100010101001000) to hex:

____

Exercise 2. Conversion exercise: hex -> binary

- Convert $3D5F_{16}$ (in **C**: 0x3D5F) to binary (**word size = 16**):

____

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

Exercise 3.

Do you know another numeral system?

If so, complete the following:

1. Name of this numeral system:

____

2. Base of this numeral system:

____

3. Digits values of this numeral system:

____

Endian – Exercise

Exercise 4. <u>Question</u>: How would the following 4-byte bit pattern be stored in memory starting at address `0x0000` …

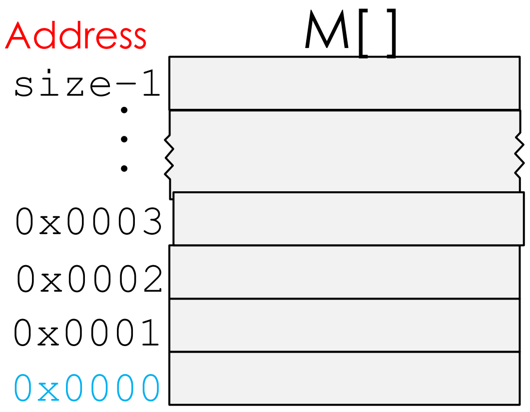
$10100011\ 10110100\ 00110101\ 11101001_2$

if the microprocessor uses *little endian*?

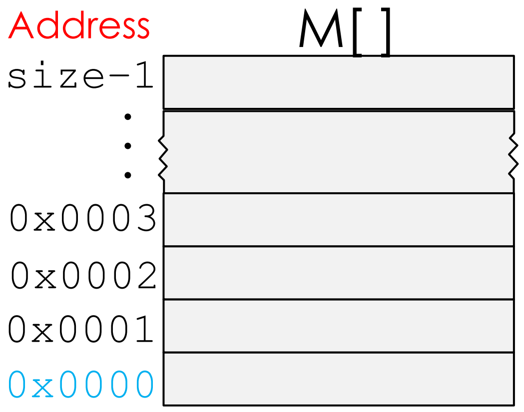
if the microprocessor uses *big endian*?

Complete the following two diagrams with your answer:

1. Little endian



2. Big endian



Solution

A solution: hexadecimal numbers

1. Conversion exercise: binary -> hex
   - Convert 1001100 11001001 01000101 01001000_2 (in **C**: 0b1001100110010010100010101001000) to hex:

Solution to Exercise 1.:

Step 1. Starting from the LSbit (least significant bit, i.e., the rightmost bit), divide each group of 4 bits (padding with zeros if the leftmost group of bits does not have 4 bits):

0100 | 1100 | 1100 | 1001 | 0100 | 0101 | 0100 | 1000_2

pad here, if needed

start here

Step 2. Using the table on the right (soon, we shall know this table by heart ☺), translate each binary number (of 4 bits) into its hexadecimal equivalent:

0100 | 1100 | 1100 | 1001 | 0100 | 0101 | 0100 | 1000_2

4 C C 9 4 5 4 8 => **0x**4CC94548

where **0x** is used to indicate a hexadecimal number

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

A solution: hexadecimal numbers

2. Conversion exercise: hex -> binary

- Convert $3D5F_{16}$ (in **C**: 0x3D5F) to binary (**word size = 16**):

Solution to Exercise 2.: $0011\ 1101\ 0101\ 1111_2$

Again, using the table on the right (I did tell you that soon, we shall know this table by heart ☺), translate each hexadecimal number into its binary equivalent.

To indicate a binary number, you can either use the subscript **2** as illustrated above, or **0b -> 0b**0011 1101 0101 1111

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

Why do we pad (adding zeros) to the left of a binary number as opposed to its right?

To answer this question, let's pad this binary number: $11\ 1101\ 0101\ 1111_2$
to the left: $0011\ 1101\ 0101\ 1111_2$
and to right: $1111\ 0101\ 0111\ 1100_2$

Can you see that $11\ 1101\ 0101\ 1111_2 == 0011\ 1101\ 0101\ 1111_2$ but $!= 1111\ 0101\ 0111\ 1100_2$
Indeed, $1111\ 0101\ 0111\ 1100_2$ converted back to hexadecimal number gives us $F57C_{16}$

To conclude: Adding zeros to the left of any numbers (binary, decimal or hexadecimal) never changes the value of the number. However, adding zeros to its right does change its value.

Solution to Exercise 3.:

Do you know another numeral system?

If so, complete the following:

1. Name of this numeral system: Decimal Numeral System

2. Base of this numeral system: 10

3. Digits values of this numeral system: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Solution to Exercise 3.:

Do you know another numeral system?

If so, complete the following:

1. Name of this numeral system: Octal Numeral System

2. Base of this numeral system: 8

3. Digits values of this numeral system: 0, 1, 2, 3, 4, 5, 6, 7

Endian – Exercise

Solution to

Exercise 4. <u>Question</u>: How would the following 4-byte bit pattern be stored in memory starting at address 0x0000 …

10100011 10110100 00110101 11101001 $_2$

if the microprocessor uses *little endian*?

if the microprocessor uses *big endian*?

Complete the following two diagrams with your answer:

1. Little endian

Address	M[ ]
size-1	
⋮	
0x0003	10100011
0x0002	10110100
0x0001	00110101
0x0000	11101001

2. Big endian

Address	M[ ]
size-1	
⋮	
0x0003	11101001
0x0002	00110101
0x0001	10110100
0x0000	10100011

Trick: LLL -> Little endian: LSB goes into lowest address