Data Representation in Memory

CSCI 2400 / ECE 3217: Computer Architecture

Instructor:

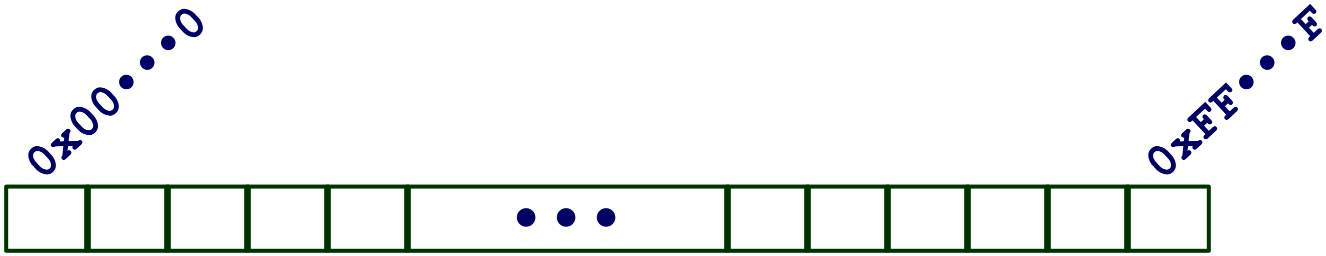
David Ferry

Slides adapted from Bryant & O'Hallaron's slides via Jason Fritts

Data Representation in Memory

- **Basic memory organization**
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- Representing Integers
  - Unsigned integers
  - Signed integers
- Representing Text
- Representing Pointers

Byte-Oriented Memory Organization

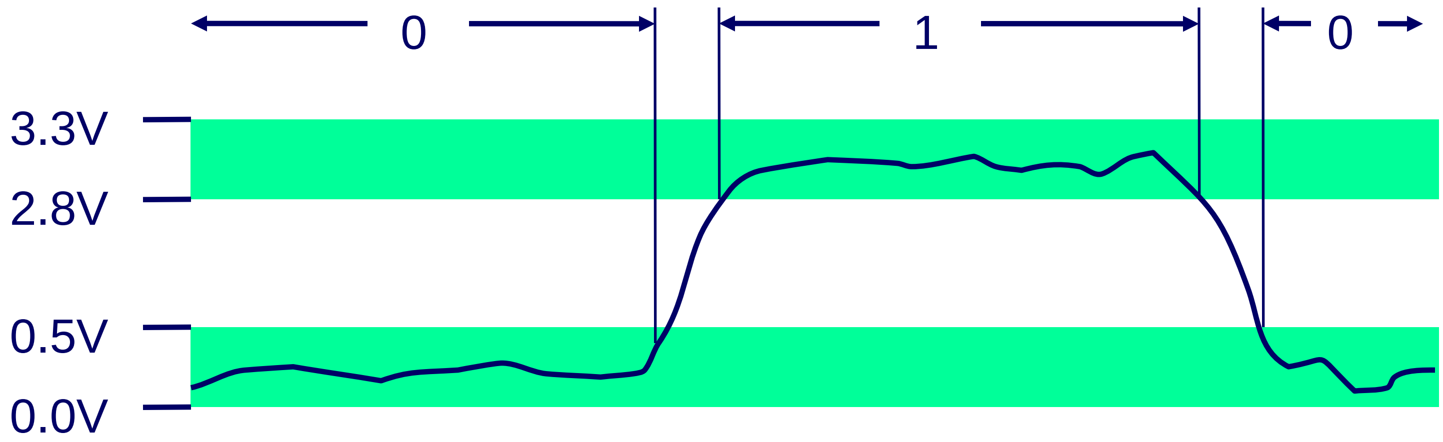


- **Modern processors: Byte-Addressable Memory**
  - Conceptually a very large array of bytes
  - Each byte has a unique address
  - Processor *address space* determines *address range*:
    - 32-bit address space has 2^{32} unique addresses: 4GB max
      - 0x00000000 to 0xffffffff (in decimal: 0 to 4,294,967,295)
    - 64-bit address space has 2^{64} unique addresses: ~ 1.8×10^{19} bytes max
      - 0x0000000000000000 to 0xffffffffffffffff
      - Enough to give everyone on Earth about 2 Gb
  - Address space size is not the same as processor size!
    - E.g.: The original Nintendo was an 8-bit processor with a 16-bit address space

Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- Representing Integers
  - Unsigned integers
  - Signed integers
- Representing Text
- Representing Pointers

Why Use Bits & Binary?



- Digital transistors operate in high and low voltage ranges
- Voltage Range dictates Binary Value on wire
  - high voltage range (e.g. 2.8V to 3.3V) is a logic 1
  - low voltage range (e.g. 0.0V to 0.5V) is a logic 0
  - voltages in between are indefinite values
- Ternary or quaternary systems have practicality problems

Bits & Bytes

- **Computers use bits:**
  - a “bit” is a base-2 digit
  - {L, H} => {0, 1}

- **Single bit offers limited range, so grouped in bytes**
  - 1 byte = 8 bits
  - a single datum may use multiple bytes

- **Data representation 101:**
  - Given N bits, can represent 2^N unique values
    - Letters of the alphabet?
    - Colors?

Encoding Byte Values

- **Processors generally use multiples of Bytes**
  - common sizes: 1, 2, 4, 8, or 16 bytes
  - Intel data names:
    - Byte 1 byte (8 bits) $2^8 = 256$
    - Word 2 bytes (16 bits) $2^{16} = 65{,}536$
    - Double word 4 bytes (32 bits) $2^{32} = 4{,}294{,}967{,}295$
    - Quad word 8 bytes (64 bits) $2^{64} = 18{,}446{,}744{,}073{,}709{,}551{,}616$

Unfortunately, these names are not standard
so we'll often use C data names instead
(but these vary in size too… /sigh)

C Data Types

32-bit *64-bit*

C Data Type	Typical 32-bit	Intel IA32	x86-64
char	1 byte	1	1
short	2	2	2
int	4	4	4
long	4	***4***	***8***
long long	8	8	8
float	4	4	4
double	8	8	8
long double	8	10/12	10/16
pointer (addr)	4	***4***	***8***

key differences

Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- Representing Integers
  - Unsigned integers
  - Signed integers
- Representing Text
- Representing Pointers

Encoding Byte Values

- **1 Byte = 8 bits**
  - Binary: 00000000_2 to 11111111_2

- **A byte value can be interpreted in many ways!**
  - depends upon how it's used

- **For example, consider byte with: 01010101_2**
  - as ASCII text: 'U'
  - as integer: 85_{10}
  - as IA32 instruction: pushl %ebp
  - the 86th byte of memory in a computer
  - a medium gray pixel in a gray-scale image 
  - could be interpreted MANY other ways…

Binary is Hard to Represent!

- **Problem with binary – Cumbersome to use**
  - e.g. approx. how big is: $1010011101010001011101011_2$?
  - Would be nice if the representation was closer to decimal: 21,930,731

- **Let's define a larger base so that**

$$\boldsymbol{R^1 = 2^x}$$

  - for equivalence, *R* and *x* must be integers – then 1 digit in *R* equals *x* bits
  - equivalence allows direct conversion between representations
  - two options closest to decimal:
    - octal: $\mathbf{8^1 = 2^3}$ (base eight)
    - hexadecimal: $\mathbf{16^1 = 2^4}$ (base sixteen)

Representing Binary Efficiently

- **Octal or Hexadecimal?**
  - binary : $101001110101000101110101 1_2$
  - octal: 123521353_8
  - hexadecimal number: $14EA2EB_{16}$
  - decimal: 21930731
- **Octal and Hex are closer in size to decimal, BUT…**
- **How many base-*R* digits per byte?**
  - Octal: 8/3 = 2.67 octal digits per byte -- BAD
  - Hex: 8/4 = 2 hex digits per byte -- GOOD

Hexadecimal wins: 1 hex digit ⇔ 4 bits

Expressing Byte Values

Juliet:
"What's in a name? That which we call a rose
By any other name would smell as sweet."

- **Common ways of expressing a byte**
  - Binary: 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
  - Hexadecimal: 00_{16} to FF_{16}
    - Base-16 number representation
    - Use characters '0' to '9' and 'A' to 'F'
    - in C/C++ programming languages, $D3_{16}$ written as either
      - 0xD3
      - 0xd3

Decimal vs Binary vs Hexadecimal

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
16	10000	10
17	10001	11
18	10010	12

Convert Between Binary and Hex

- **Convert Hexadecimal to Binary**
  - Simply replace each hex digit with its equivalent 4-bit binary sequence
  - Example: **6 D 1 9 F 3 C_{16}**

0110 1101 0001 1001 1111 0011 1100_2

- **Convert Binary to Hexadecimal**
  - Starting from the radix point, replace each sequence of 4 bits with the equivalent hexadecimal digit
  - Example: $1011001000110101110101100010100011_2$

1 6 4 6 B A C 5 3_{16}

Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- **Representing Integers**
  - **Unsigned integers**
  - Signed integers
- Representing Text
- Representing Pointers

Unsigned Integers – Binary

- **Computers store Unsigned Integer numbers in Binary (base-2)**
  - Binary numbers use place valuation notation, just like decimal
  - Decimal value of *n*-bit unsigned binary number:

$$value_{10} = \sum_{i=0}^{n-1} a_i * 2^i$$

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

$$\begin{aligned} value_{10} &= 0 * 2^7 + 1 * 2^6 + 1 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 \\ &= 2^6 + 2^5 + 2^4 + 2^2 + 2^0 \\ &= 64 + 32 + 16 + 4 + 1 \ \boxed{= 117_{10}} \end{aligned}$$

Unsigned Integers – Base-R

- **Convert Base-R to Decimal**
  - Place value notation can similarly determine decimal value of any base, *R*
  - Decimal value of *n*-digit base *r* number:

$$value_{10} = \sum_{i=0}^{n-1} a_i * r^i$$

  - Example: $317_8 = ?_{10}$

$$value_{10} = 3 * 8^2 + 1 * 8^1 + 7 * 8^0$$
$$= 3 * 64 + 1 * 8 + 7 * 1$$
$$= 192 + 8 + 7 \boxed{= 207_{10}}$$

Unsigned Integers – Hexadecimal

- **Commonly used for converting hexadecimal numbers**
  - Hexadecimal number is an "equivalent" representation to binary, so often need to determine decimal value of a hex number
  - Decimal value for *n*-digit hexadecimal (base 16*)* number:

$$value_{10} = \sum_{i=0}^{n-1} a_i * 16^i$$

  - Example: $9E4_{16} = ?_{10}$

$$value_{10} = 9 * 16^2 + 14 * 16^1 + 4 * 16^0$$
$$= 9 * 256 + 14 * 16 + 4 * 1$$
$$= 2304 + 224 + 4 \boxed{= 2532_{10}}$$

Unsigned Integers – Convert Decimal to Base-R

- **Also need to convert decimal numbers to desired base**
- **Algorithm for converting unsigned Decimal to Base-R**
  a) Assign decimal number to *NUM*
  b) Divide *NUM* by *R*
    - Save remainder *REM* as next least significant digit
    - Assign quotient *Q* as new *NUM*
  c) Repeat step b) until quotient *Q* is zero
- **Example:** $\mathbf{83_{10} = ?_7}$

NUM		*R*		*Q*		*REM*
$\mathbf{83}$	/	$\mathbf{7}$	$\rightarrow$	$\mathbf{11}$	$\boldsymbol{r}$	$\mathbf{6}$
$\mathbf{11}$	/	$\mathbf{7}$	$\rightarrow$	$\mathbf{1}$	$\boldsymbol{r}$	$\mathbf{4}$
$\mathbf{1}$	/	$\mathbf{7}$	$\rightarrow$	$\mathbf{0}$	$\boldsymbol{r}$	$\mathbf{1}$

least significant digit

$$\boxed{= \mathbf{146_7}}$$

most significant digit

Unsigned Integers – Convert Decimal to Binary

- **Example with Unsigned Binary:** $52_{10} = ?_2$

least significant digit

NUM	R		Q		REM
52	$/\ 2$	$\rightarrow$	26	r	0
26	$/\ 2$	$\rightarrow$	13	r	0
13	$/\ 2$	$\rightarrow$	6	r	1
6	$/\ 2$	$\rightarrow$	3	r	0
3	$/\ 2$	$\rightarrow$	1	r	1
1	$/\ 2$	$\rightarrow$	0	r	1

$$\boxed{= 110100_2}$$

most significant digit

Unsigned Integers – Convert Decimal to Hexadecimal

- **Example with Unsigned Hexadecimal:** $437_{10} = ?_{16}$

least significant digit

NUM	*R*		*Q*		*REM*
437	$/\ 16$	$\rightarrow$	27	r	5
27	$/\ 16$	$\rightarrow$	1	r	11
1	$/\ 16$	$\rightarrow$	0	r	1

$$\boxed{= 1B5_{16}}$$

most significant digit

Unsigned Integers – Ranges

- **Range of Unsigned binary numbers based on number of bits**
  - Given representation with *n* bits, min value is always sequence
    - 0....0000 = 0
  - Given representation with *n* bits, max value is always sequence
    - 1....1111 = $2^n - 1$
  - So, ranges are:
    - unsigned char: $\mathbf{0 \rightarrow 255 \quad (2^8 - 1)}$
    - unsigned short: $\mathbf{0 \rightarrow 65,535 \quad (2^{16} - 1)}$
    - unsigned int: $\mathbf{0 \rightarrow 4,294,967,295 \quad (2^{32} - 1)}$

1	*1*	. . .	*1*	*1*	*1*	*1*
2^{n-1}	2^{n-2}		2^3	2^2	2^1	2^0

$$= \sum_{i=0}^{n-1} 2^i = 2^n - 1$$

Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- **Representing Integers**
  - Unsigned integers
  - **Signed integers**
- Representing Text
- Representing Pointers

Signed Integers – Binary

- **Signed Binary Integers converts half of range as negative**
- **Signed representation identical, except for most significant bit**
  - For signed binary, most significant bit indicates sign
    - 0 for nonnegative
    - 1 for negative
  - <u>Must know number of bits</u> for signed representation

***Unsigned* Integer representation:**

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

Place value of most significant bit is <u>negative</u> for signed binary

***Signed* Integer representation:**

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

Signed Integers – Binary

- Decimal value of *n*-bit signed binary number:

$$value_{10} = -a_{n-1} * 2^{n-1} + \sum_{i=0}^{n-2} a_i * 2^i$$

- Positive (in-range) numbers have same representation:

Unsigned Integer representation:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	1	1	0	1	0	0	1	$= 105_{10}$

Signed Integer representation:

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
0	1	1	0	1	0	0	1	$= 105_{10}$

Signed Integers – Binary

- Only when most significant bit set does value change

- Difference between unsigned and signed integer values is 2^N

Unsigned Integer representation:

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

$$= 105$$
$$\boxed{+\,128}10$$
$$= 233_{10}$$

Signed Integer representation:

~~0~~ 1	1	1	0	1	0	0	1
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

$$= 105$$
$$\boxed{-\,128}10$$
$$= -23_{10}$$

Quick Check:

For an 8-bit representation:

- What bit pattern has the minimum value?
- What bit pattern has the maximum value?
- What bit pattern represents 0?
- What bit pattern represents -1?

Signed Integers – Ranges

- **Range of Signed binary numbers:**
  - Given representation with *n* bits, min value is always sequence
    - 100....0000 = -2^{n-1}
  - Given representation with *n* bits, max value is always sequence
    - 011....1111 = $2^{n-1}-1$
  - So, ranges are:

C data type	# bits	Unsigned range	Signed range
char	8	0 → 255	-128 → 127
short	16	0 → 65,535	-32,768 → 32,767
int	32	0 → 4,294,967,295	-2,147,483,648 → 2,147,483,647

Signed Integers – Convert to/from Decimal

- **Convert Signed Binary Integer to Decimal**
  - Easy – just use place value notation
    - two examples given on last two slides
- **Convert Decimal to Signed Binary Integer**
  - MUST know <u>number of bits</u> in signed representation
  - ***Algorithm***:
    a) Convert magnitude (abs val) of decimal number to unsigned binary
    b) Decimal number originally negative?
      – If positive, conversion is <u>done</u>
      – If negative, perform negation on answer from part a)
        » zero extend answer from a) to N bits (size of signed repr)
        » negate: flip bits and add 1

Signed Integers – Convert Decimal to Base-R

- **Example:** $-37_{10} = ?_{8-bit\ signed}$
  - A) $|-37_{10}| = ?_2$

least significant bit

NUM	*R*		*Q*		*REM*
37	$/\ 2$	$\rightarrow$	18	r	1
18	$/\ 2$	$\rightarrow$	9	r	0
9	$/\ 2$	$\rightarrow$	4	r	1
4	$/\ 2$	$\rightarrow$	2	r	0
2	$/\ 2$	$\rightarrow$	1	r	0
1	$/\ 2$	$\rightarrow$	0	r	1

$= 100101_2$

most significant bit

Signed Integers – Convert Decimal to Base-R

- **Example:** $-37_{10} = ?_{8-bit\ signed}$
  - B) -37_{10} was negative, so perform *negation*
    - zero extend 100101 to 8 bits

$$100101_2 \rightarrow \underline{00}100101_2$$

    - negation
      - flip bits:

$$00100101_2$$
$$\downarrow$$
$$11011010_2$$

      - add 1:

$$\begin{array}{r} 11011010_2 \\ + \quad\quad\quad 1_2 \\ \hline 11011011_2 \end{array}$$

$$\boxed{= 11011011_2}$$

Can validate answer using place value notation

Quick check:

For an 8-bit representation:

- Convert 67_{10} into a signed integer

Signed Integers – Convert Decimal to Base-R

- **Example:** $67_{10} = ?_{8-bit\ signed}$
  - A) $|67_{10}| = ?_2$

NUM		R		Q		REM
67	/	2	$\rightarrow$	33	r	1
33	/	2	$\rightarrow$	16	r	1
16	/	2	$\rightarrow$	8	r	0
8	/	2	$\rightarrow$	4	r	0
4	/	2	$\rightarrow$	2	r	0
2	/	2	$\rightarrow$	1	r	0
1	/	2	$\rightarrow$	0	r	1

least significant bit

$$\boxed{= 1000011_2}$$

most significant bit

Signed Integers – Convert Decimal to Base-R

- **Example:** $\mathbf{67_{10} = ?_{8-bit\ signed}}$
  - B) 67_{10} was positive, so done

$$\boxed{= \mathbf{1000011_2}}$$

Can validate answer using place value notation

Quick check:

For an 8-bit representation:

- Convert -100_{10} into a signed integer

Signed Integers – Convert Decimal to Base-R

- **Example:** $-100_{10} = ?_{8-bit\ signed}$
  - A) $|-100_{10}| = ?_2$

least significant bit

NUM	R		Q		REM
100	$/\ 2$	$\rightarrow$	50	r	0
50	$/\ 2$	$\rightarrow$	25	r	0
25	$/\ 2$	$\rightarrow$	12	r	1
12	$/\ 2$	$\rightarrow$	6	r	0
6	$/\ 2$	$\rightarrow$	3	r	0
3	$/\ 2$	$\rightarrow$	1	r	1
1	$/\ 2$	$\rightarrow$	0	r	1

$= 1100100_2$

most significant bit

Signed Integers – Convert Decimal to Base-R

- **Example:** $-100_{10} = ?_{8-bit\ signed}$
  - B) -100_{10} was negative, so perform *negation*
    - zero extend 100101 to 8 bits

$$1100100_2 \rightarrow \underline{\ }01100100$$

    - negation
      - flip bits:

$$01100100_2$$
$$\downarrow$$
$$10011011_2$$

      - add 1:

$$+ \qquad 1_2$$
$$10011100_2$$

$$\boxed{= 10011100_2}$$

Can validate answer using place value notation

Signed Integers – Convert Decimal to Base-R

- **Be careful of range!**
- **Example:** $-183_{10} = ?_{8-bit\ signed}$
  - A) $|-183_{10}| = ?_2$ $= 10110111_2$
  - B) -183_{10} was negative, so perform *negation*
    - zero extend 10110111 to 8 bits // already done
    - negation
      - flip bits: $10110111_2 \downarrow 01001000_2$
      - add 1: $+ \quad 1_2$

$$01001001_2 = 73_{10}$$

***not* -183$_{10}$… WRONG!**

-183$_{10}$ is not in valid range for 8-bit signed

Representation of Signed Integers

- **Multiple possible ways:**
  - Sign magnitude
  - Ones' Complement
  - Two's Complement (what has been presented)

- **Two's Complement greatly simplifies addition & subtraction in hardware**
  - We'll see why when we cover operations
  - Generally the only method still used

Representation of Signed Integers

- **Why the name Two's Complement?**
  - For a *w*-bit signed representation, we represent -x as 2^w – x
  - E.g.: consider the 8-bit representation of $-\mathbf{37_{10}}$

$$\mathbf{2^8 = 256_{10}}$$

$$\mathbf{2^8 - 371_0 = 219_{10}}$$

$$\mathbf{219_{10} = 11011011_2} \text{ (unsigned)}$$

$$\mathbf{-371_0 = 11011011_2} \text{ (signed)}$$

Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- Representing Integers
  - Unsigned integers
  - Signed integers
- **Representing Text**
- Representing Pointers

Representing Strings

```
char S[6] = "18243";
```

- **Strings in C**
  - Represented by array of characters
  - Each character encoded in **ASCII format**
    - Standard 7-bit encoding of character set
    - Character “0” has code 0x30
  - String should be null-terminated
    - Final character = 0
  - ASCII characters organized such that:
    - Numeric characters sequentially increase from 0x30
      - Digit *i* has code 0x30+*i*
    - Alphabetic characters sequentially increase in order
      - Uppercase chars ‘A’ to ‘Z’ are 0x41 to 0x5A
      - Lowercase chars ‘A’ to ‘Z’ are 0x61 to 0x7A
    - Control characters, like <RET>, <TAB>, <BKSPC>, are 0x00 to 0x1A

Intel / Linux

0x31	‘1’
0x38	‘8’
0x32	‘2’
0x34	‘4’
0x33	‘3’
0x00	null term

Representing Strings

- **Limitations of ASCII**
  - 7-bit encoding limits set of characters to $2^7 = 128$
  - 8-bit extended ASCII exists, but still only $2^8 = 256$ chars
  - Unable to represent most other languages in ASCII
- **Answer: *Unicode***
  - first 128 characters are ASCII
    - i.e. 2-byte Unicode for ‘4’: 0x34 -> 0x0034
    - i.e. 4-byte Unicode for ‘T’: 0x54 -> 0x00000054
  - UTF-8: 1-byte version // commonly used
  - UTF-16: 2-byte version // commonly used
    - allows 2^{16} = 65,536 unique chars
  - UTF-32: 4-byte version
    - allows 2^{32} = ~4 billion unique characters
  - Unicode used in many more recent languages, like Java and Python

UTF-16 on Intel

0x31	‘1’
0x00	
0x38	‘8’
0x00	
0x32	‘2’
0x00	
0x34	‘4’
0x00	
0x33	‘3’
0x00	
0x00	null term
0x00	

Data Representation in Memory

- Basic memory organization
- Bits & Bytes – basic units of Storage in computers
- Representing information in binary and hexadecimal
- Representing Integers
  - Unsigned integers
  - Signed integers
- Representing Text
- **Representing Pointers**

Representing Pointers

```
int B = -15213;
int *P = &B;
```

Sun	IA32	x86-64
EF	D4	0C
FF	F8	89
FB	FF	EC
2C	BF	FF
		FF
		7F
		00
		00

Different compilers & machines assign different locations to objects