Introduction & Bits and Bytes

CS 105: Computer Systems
Lecture 01

Prof Melissa O'Neill

January 21, 2026

Course Theme:
Abstraction is good, *but don't forget reality*

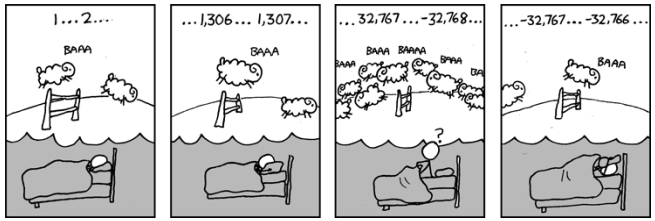
- **Most CS courses emphasize abstraction**
  - Abstract data types
  - Asymptotic analysis
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations

Lecture 01 Learning Goals

- **Understand the goals of the course**
- **Describe a basic computer architecture and explain how its components interact to run a process**
- **Understand how bits are organized in computer memory**
  - Important for this course: binary and hexadecimal representations
- **Apply and predict the results of bit operations including &, |, ~, ^, <<, >> as well as logical operators**

Reality Example #1
Ints are not Integers, Floats are not Reals

- **Is $x^2 \geq 0$?**
- **Is (x+y)+z = x + (y+z)?**



Reality Example #2

There's more to performance than asymptotic complexity

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

2.0 GHz Intel Core i7 Haswell

- **Performance depends on access patterns**
  - Including how you step through multi-dimensional array

Course: Useful Outcomes

- **Understand how computers work**
- **Become more effective programmers**
  - Able to find and eliminate bugs efficiently
  - Able to understand and tune for program performance
- **Prepare for later "systems" classes in CS**
  - Compilers, Operating Systems, Networks, File Systems, Computer Architecture, Robotics, etc.

Logistics Overview

- **Course website**
  - https://www.cs.hmc.edu/cs105
  - Schedule on website gives due dates, quiz/exam times, readings, practice problems from text
  - Grades

Do these!!

- **Piazza for Q&A**
  - https://piazza.com/hmc/spring2025/hmccs105sp25

Participation counts!

- **Gradescope for quiz and exam feedback**
- **Labs:** Fridays in McGregor 203-204
- **Quizzes:** about every two weeks, closed notes
- **Exams:** midterm and final, open notes
- See course overview for more (grade break down; late days)

Textbooks

- **Randal E. Bryant and David R. O'Hallaron,**

  "Computer Systems: A Programmer's Perspective", 3rd Edition, Prentice Hall, 2015.

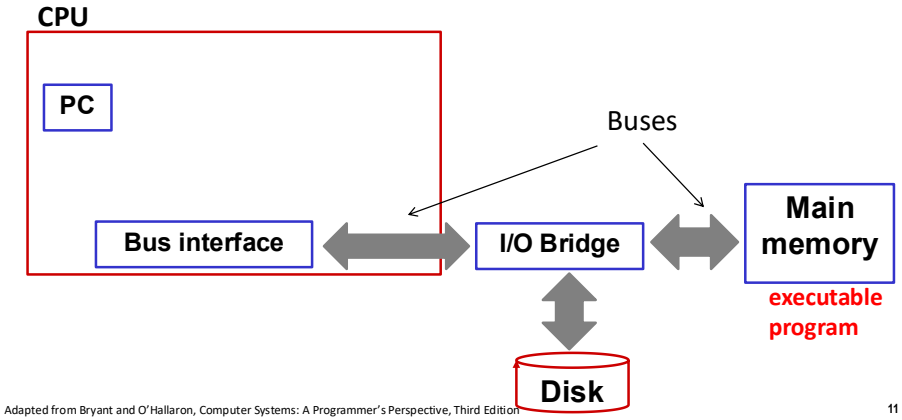
- **Optional**
  - Brian Kernighan and Dennis Ritchie,

    "The C Programming Language, Second Edition", Prentice Hall, 1988
  - Larry Miller and Alex Quilici

    The Joy of C, Wiley, 1997

Running a Process: Hardware Organization Perspective

- **Definition: A *process* is an instance of a running program**
  - Conceptually, CPU executes logical sequence of *instructions,* driven by clock *ticks* (aka cycles)



Counting things and number representation

- **Suppose we have one tally mark on board for each student**
  - How many students are there?

- **How many unique ID numbers could I create with two digits in decimal representation?**

Exercise

In the *octal* number system (i.e., base 8), each digit can have one of eight values: 0 – 7.

1. **How many unique IDs could you create using only 2 octal digits?**

2. **What is the octal equivalent of the decimal number 53_{10}?**

Exercise (cntd)

In the *binary* number system (i.e., base 2), each digit can have one of two values: 0 – 1.

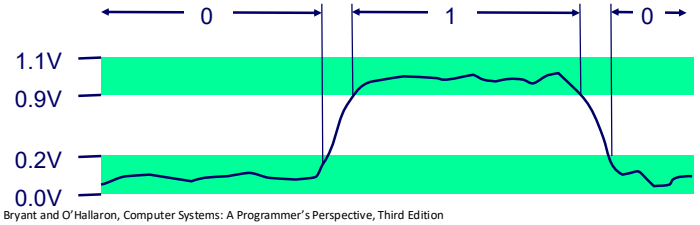
3. **How many unique IDs could you create using 4 binary digits? What about 6 binary digits?**

4. **What is the binary equivalent of the decimal number 53_{10}?**

5. **Suppose in some number system each digit can have *b* distinct values. How many unique IDs could you create with 4 digits?**

Everything is bits – *binary digits*

- **Each bit is 0 or 1 (base 2)**
- **Bits encode everything:**
  - Program instructions, program data (numbers, characters, strings, etc.), addresses in memory, etc.
- **Why bits? Electronic Implementation**
  - Easy to store with bistable elements (hardware circuits)
  - Reliably transmitted on noisy and inaccurate wires



Hexadecimal: base 16

- **Hexadecimal**
  - Base 16 number representation
  - Use characters '0' to '9' and 'A' to 'F'
- **Each digit of hex is 4 binary digits**
  - Write $FA1D37B_{16}$ in C as
    - 0xFA1D37B
    - 0xfa1d37b

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

Encoding Byte Values

- **Byte = 8 bits**
  - Binary 00000000_2 to 11111111_2
  - Decimal: 0_{10} to 255_{10}
- **How many hex digits encode one byte?**

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

Bytes for data types and architectures

C Data Type	x86-64
char	1
short	2
int	4
long	8
float	4
double	8
pointer	8

"character" set encoding that used 1 byte to represent characters; namely ASCII encoding

bytes in pointer influences how many things you can point to!

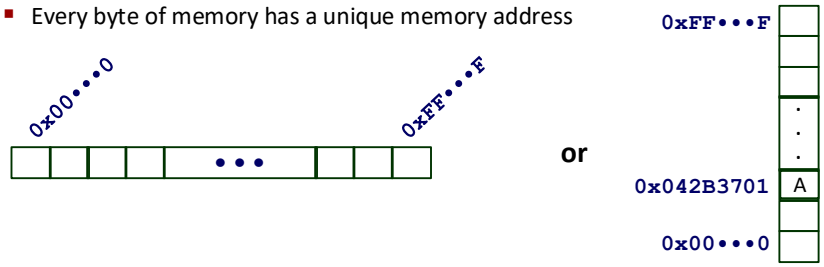
Word size of system

Machine Words

- **Any given computer has a "Word Size"**
  - Nominal size of integer-valued data and of **memory addresses**
    - Registers typically hold 64 bits
    - The ALU accepts 64 bit inputs
    - All buses (which transport data) can transport 64 bits in parallel
  - Until recently, most machines used 32 bits (4 bytes) as word size
    - Limits addresses to 4GB (2^{32} bytes)
- Increasingly, machines have 64-bit word size
  - Potentially, could have 18 EB (exabytes) of addressable memory
  - That's 18.4 X 10^{18}

Main memory is byte-addressable

- Every byte of memory has a unique memory address

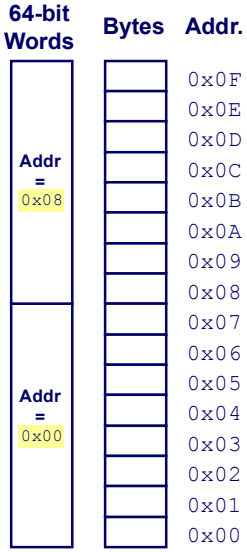


- Programs refer to data in memory by its starting address
- A pointer is just an address in memory

```
char c = 'A';        /* c is type char, 1 byte */
char * ptr = &c;     /* ptr is type char *,
                        8 bytes on x86-64 */
```

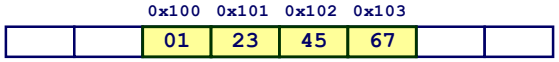

Word-Oriented Memory Organization

- **Multi-byte words have address that is address of the first byte in the word**
- **E.g., storing 8-byte word**
  - Addresses of successive words differ by 8 bytes (64-bits)



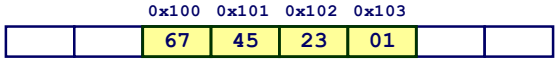
Byte Ordering: little endian vs. big endian

- **How are bytes within a multi-byte word ordered in memory?**
- **Two Conventions: *little* endian and *big* endian**
- **Example:**
  - Variable `x` has 4-byte value of 0x01234567
  - Address given by `&x` is 0x100
  - **Big Endian**: Sun, PPC Mac, Internet
    - Least significant byte has *highest address*



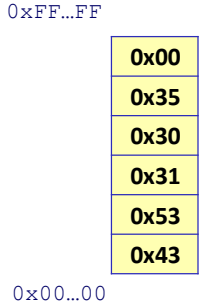
  - **Little Endian**: x86, ARM processors running Android, iOS, and Windows
    - Least significant byte has *lowest address*

This course



Representing Strings

- **Strings in C**
  - Represented by array of characters
  - First element of array is at lowest memory address
  - Each character encoded in **ASCII** format
    - Standard 7-bit encoding of character set
    - Character "0" has code 0x30
      - Digit *i* has code 0x30+*I*
      - *Capital English letters start at 0x41 for 'A'*
  - String should be null-terminated
    - Final character = 0, aka 0x00




```
char S[6] = "CS105";
```


Bits encode characters – ASCII lookup table

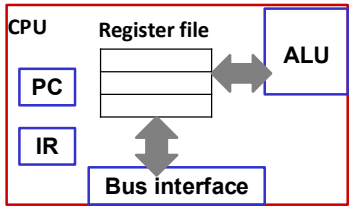
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

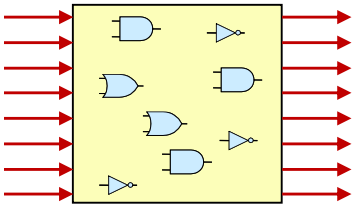
Manipulating bits and program instructions

- **Bits encode instructions**

```
addq %rax, %rbx
```



- **Circuit Logic Gates**
  - Responds to changes on primary inputs
  - Primary outputs become (after a delay) the Boolean functions of primary inputs
  - Can also be designed to "store" bits (using cyclic networks)



Boolean Algebra

- **Developed by George Boole in 19th Century**
  - Algebraic representation of logic
  - Encode "True" as 1 and "False" as 0

And

- **A&B = 1 when both A=1 and B=1**

&	0	1
0	0	0
1	0	1

Or

- **A|B = 1 when either A=1 or B=1**

|	0	1
0	0	1
1	1	1

Not

- **~A = 1 when A=0**

~	
0	1
1	0

Exclusive-Or (Xor)

- **A^B = 1 when either A=1 or B=1, but not both**

^	0	1
0	0	1
1	1	0

Exercise

- **Complete the truth table for ~(A & B)**

~(A&B)	0	1
0		
1		

General Boolean Algebras (supported in C)

- **Operate on Bit Vectors**
  - Operations applied bitwise

```
  01101001     01101001     01101001
& 01010101   | 01010101   ^ 01010101   ~ 01010101
  01000001     01111101     00111100     10101010
```

- **Boolean operations are commutative, associative, and distributive**
  - E.g.
    A & B = B & A
    (A & B) & C = A & (B & C)
    (A | B) & C = (A & C) | (B & C)
    (A & B) | C = (A | C) & (B | C)

Representing & Manipulating Sets

- **Representation**
  - Width *w* bit vector represents membership in some set A
  - Let a_j denote the bit at position *j* in bit vector
    - $a_j = 1$ if the element represented by position j $\in$ the set A
- **Example: suppose each bit position *j* represents a person, w=8**
  - Which people like dogs?
    - 01101001_2 → { 0, 3, 5, 6 } like dogs
  - Which people like Pokémon?
    - 01010101_2 → { 0, 2, 4, 6 } like Pokémon
- **Operations**
  - & Intersection 01000001_2 { 0, 6 }
  - | Union 01111101_2 { 0, 2, 3, 4, 5, 6 }
  - ^ Symmetric difference 00111100_2 { 2, 3, 4, 5 }
  - ~ Complement 10101010_2 { 1, 3, 5, 7 }

  Binary operators; unary

Shift Operations

- **Left Shift: x << y**
  - Shift bit-vector **x** left **y** positions
    - Throw away extra bits on left
    - Fill with 0's on right

Argument x	01100010
<< 3	00010000

- **Right Shift: x >> y**
  - Shift bit-vector **x** right **y** positions
    - Throw away extra bits on right
  - Logical shift
    - Fill with 0's on left
  - Arithmetic shift
    - Replicate most significant bit on left

Argument x	10100010
Log. >> 2	00101000
Arith. >> 2	11101000

Argument x	01100010
Log. >> 2	00011000
Arith. >> 2	00011000

- **Undefined Behavior**
  - Shift amount < 0 or ≥ word size

Exercise

Evaluate each of the expressions below. Recall that 0x is a prefix for hex numbers. Give your answers in binary and hex.

1. **~01000001_2 =**
2. **~0xFF =**
3. **01101001_2 | 01010101_2 =**
4. **(01101001_2 >> 2) & 0x0F =**

Bit-Level Operations in C

- **Operations &, |, ~, ^ Available in C**
  - Apply to any "integral" data type
    - long, int, short, char, unsigned
- **View arguments as bit vectors**
  - Arguments applied bit-wise

Contrast: Logic Operations in C

- **Logical Operators in C**
  - &&, ||, !
    - 0 is "False"
    - Anything nonzero is "True"
    - Always returns 0 or 1
    - Early termination
- **Examples (char data type)**
  - !0x41 ☞0x00
  - !0x00 ☞0x01
  - !!0x41 ☞0x01
  - 0x69 && 0x55 ☞0x01
  - 0x69 || 0x55 ☞0x01

Watch out for && vs. & (and || vs. |)…
one of the more common oopsies in C programming!