CS33 Lecture 1: Intro to Computer Organization
Bits, Bytes, and Integers

Sandra Batista

Adapted from Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition
And with permission CS33 F24 and S24 slides from Professors Tony Nowatzki and Glenn Reinman

Objectives for Today

1. Introduction, Course Administration, Course Overview
2. Representing data in binary
3. Boolean operations
4. Activity: Forming Learning Pods

Introduction

Instructor: Sandra Batista, sandra@cs.ucla.edu

Class Meeting: TTh 2-3:50 pm Moore 100

Office Hours: M 3-4 pm and Th 6-7 pm ENGR VI 282 and via personal Zoom Link on BruinLearn homepage

Best to use piazza for course content and assignment questions

Email is better for private concerns

Introduction

1. **What am I most excited or nervous about teaching CS33?**
   I actually find reading and writing assembly code relaxing. I am a little nervous because I think like a complexity theorist!
2. **Please share something that has nothing to do work.**
   I love Nordic Walking, hot yoga, gardening, Heartfulness meditation (there is a group here at UCLA), and all things baby Yoda (and Disney+ Star Wars series).

Please reply to piazza post for introducing yourselves.

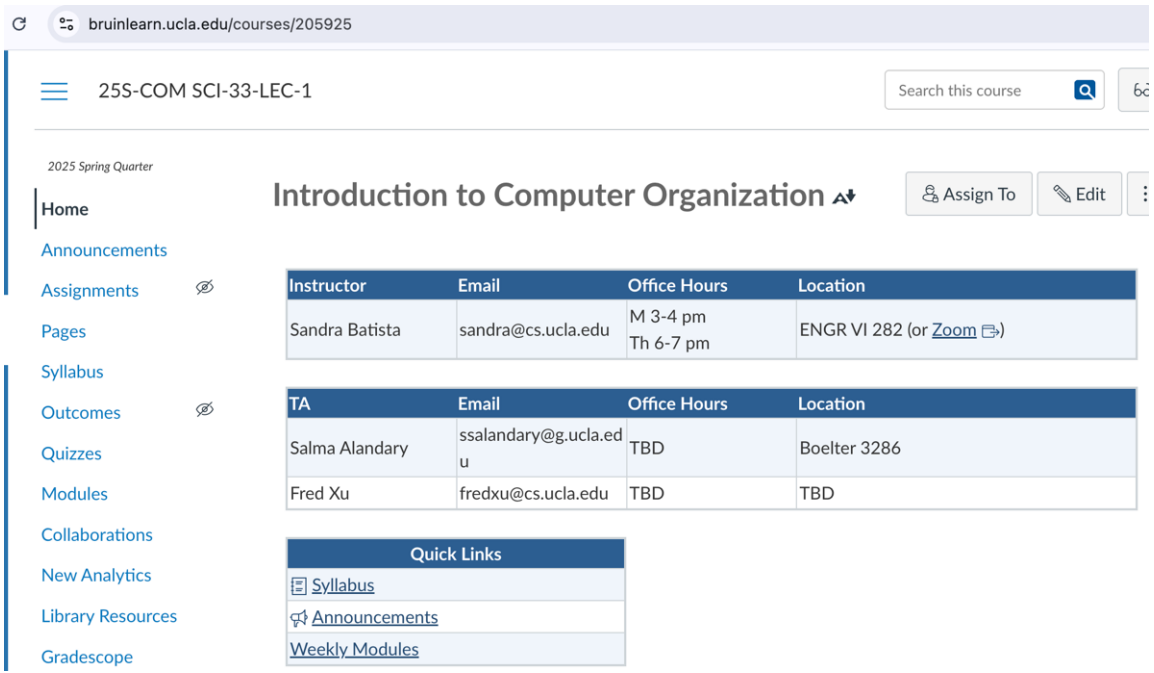
Please ask questions…

Please feel free to ask questions at any time. If you'd prefer anonymity. Here is slido: https://app.sli.do/event/sdKwrCBV3yQxwJATJMXvuf

Introduction

Course Administration

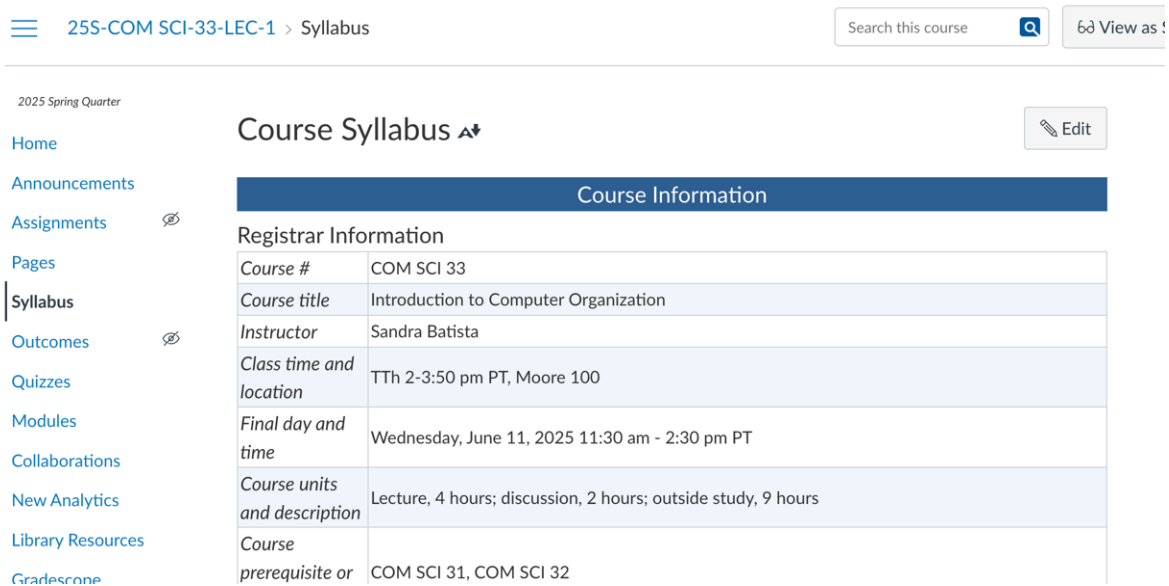
All course content will be on the course BruinLearn page: https://bruinlearn.ucla.edu/courses/205925



Introduction

Course Administration

All course content will be on the course BruinLearn page:
https://bruinlearn.ucla.edu/courses/198908

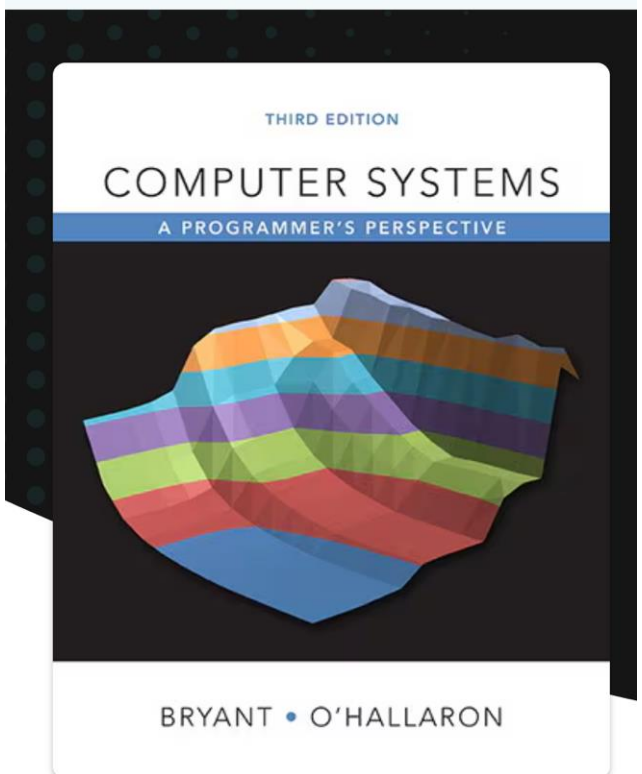


Introduction

Required Textbook

Randal E. Bryant and David R. O'Hallaron. 2015. Computer Systems: A Programmer's Perspective (3rd. ed.). Pearson.

A package deal including e-book is available from the bookstore



Course Components

- **Lectures**
  - Teach Concepts, show examples
- **Discussions**
  - Applied concepts, important tools and skills for labs, clarification of lectures, exam coverage
  - Worksheets handed out in discussion will serve as “homeworks”
- **Labs**
  - The heart of the course
  - Very fun and time-consuming
  - Provide in-depth understanding of an aspect of systems

Introduction

Assessments

Final Weighted Grades

- 35% Labs
- 20% Discussion Work
- 20% Midterm
- 25% Final

You may collaborate, but please submit your own understanding as your solutions and list your collaborators. You may use scholarly sources and must cite them.

Exams are individual efforts and are open course materials.

Our Wonderful S25 CS33 Course Staff

- TAs: Salma, Fred, Lucas, Dylan, and Alexis
- LAs: Jeff, Aiden, Anagh, Edward, Jason, Kenny, Randy, Rithvik, Ridhima

Please attend discussion in which you are enrolled

They will support you during discussion, office hours, and on piazza.

They'll guide you through the check-in process during the first discussion *this Friday*

Introduction

Additional Course Policies

Discussion Work:

Attendance and participation at discussion is required.

1. You will work in **learning pods.**

At 2 least people from your pod must attend each entire discussion section to present group work at check-ins.

Check-ins are how LAs and TAs will track participation.

Everyone must attend at least 5 discussion sections.

Introduction

Additional Course Policies

What to submit for Weekly Discussion Work:

1. Your pod will have a check-in during discussion and submit to Gradescope work as proof of that check-in. This is a group assignment for your learning pod.
2. You must submit your attempts at writing up your solutions to the discussion worksheets. This is an individual assignment.

Introduction

Additional Course Policies

Late Policy: No late work accepted.

Wellness matters! You come first. Please let me know if you need extra support for course materials or extenuating circumstances.

Introduction

Additional Course Policies

Midterm Exam during class on 5/6

Final exam on 6/11 11:30 am-2:30 pm

Exams are independent effort and open course materials.
You may only discuss questions with the course staff.

Introduction

Additional Course Policies

Warmup Lab 3%
Data Lab 8%
Bomb Lab 8%
Attack Lab 8%
Parallel Lab 8%

Labs are independent effort but you may collaborate at a high level (not code). You must list collaborators and scholarly sources.

Academic Community Expectations

1. **Please only submit your own efforts for your assignments.**
2. Authorized course materials are only released by the course staff this quarter. Using exams and assignments from previous quarters is considered using unauthorized materials. Using Chegg or other solution repositories is also considered using unauthorized materials.
3. Sharing your code from lab solutions or other individual assessments is not permitted during this quarter or in the future.
4. Since discussing course content, exercises, and lab assignments at high-level (i.e. not at code level) is permitted, it is necessary for you to list your collaborators on your assignments. You must also give citations for any additional resources such as textbooks or websites that you use.
5. The use of GenAI is not permitted on your assignments.

Lab Facilities

- **SEAS Administered Linux Machine**
  - Remote access only
    - Use ssh to log in with your SEAS account
    - Machine: cs33.seas.ucla.edu
    - Some labs require a specific machine to work
    - Off campus it may be necessary to use VPN to connect
  - Please direct any account issues to the SEAS help desk as they are the only ones with root access on this machine

How to write code?

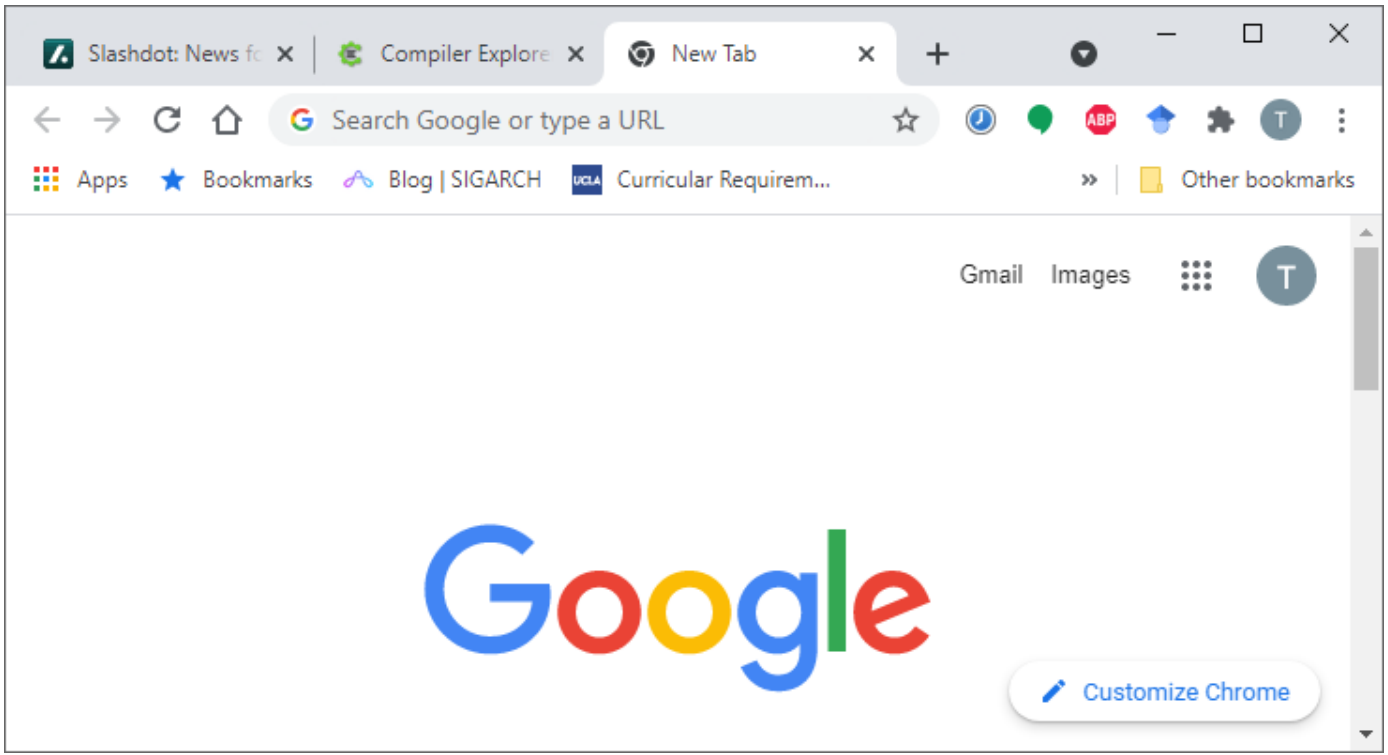
- **Please test code on seas machines**
  (that's where we grade them)

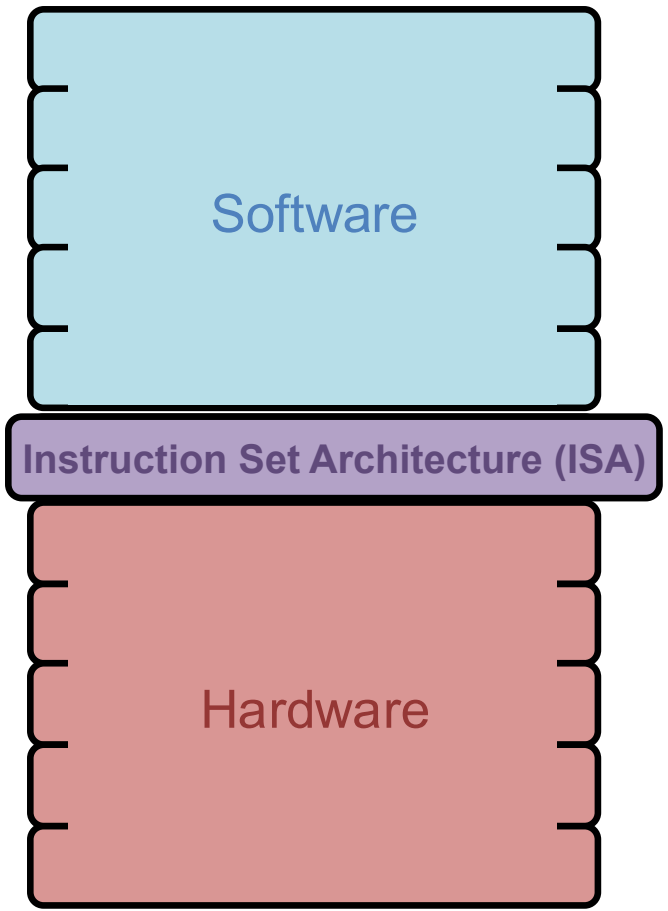
- **3 ways to edit files:**
  1. Log in, edit remotely (vim emacs)
  2. Edit locally, transfer files
     (winscp,cyberduck,sshfs)
     - Files opened from within
       these tools will be automatically transferred to the server when saved
  3. Use IDE with built in remote file editing
     - E.g. VSCode with remote ssh extension

  - **TA's should discuss all three options on Friday**
  - **Please bring your Laptop to discussion!**

Course Overview

Course in a nutshell:
How software and hardware work together…





Early Computers: Had No “Architecture”!

- **Each Computer Required Tremendous Effort**

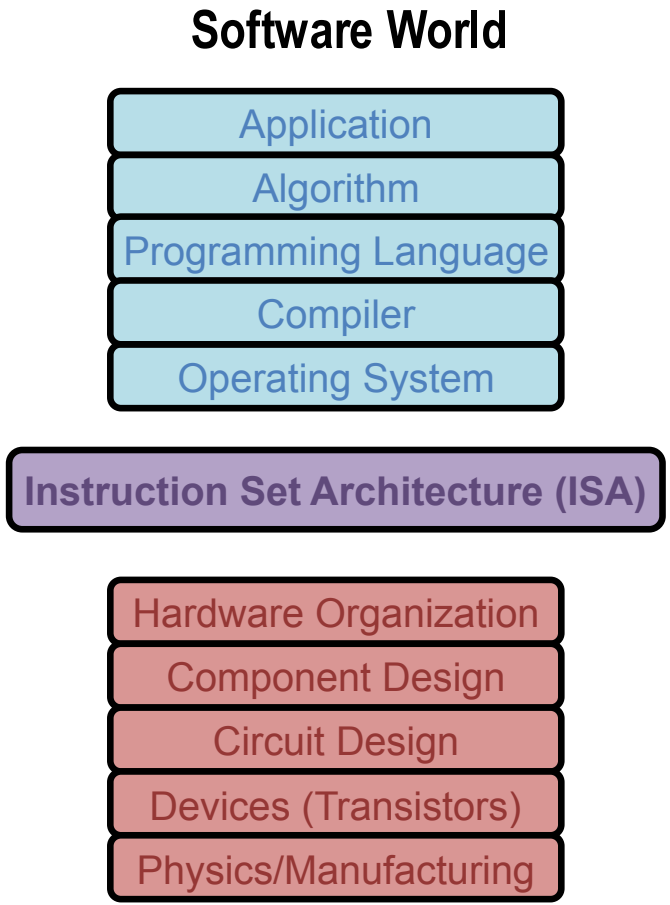
- **Without clear Boundary between hardware and software**
  - No standard interface to specify programs!
  - New programs meant changing hardware
  - New hardware required changing programs!
  - Software Lagged Hardware

- **Today:** Separation between hardware and software
  - Separation of HW interface from implementation

Betty Jean Jennings and Fran Bilas

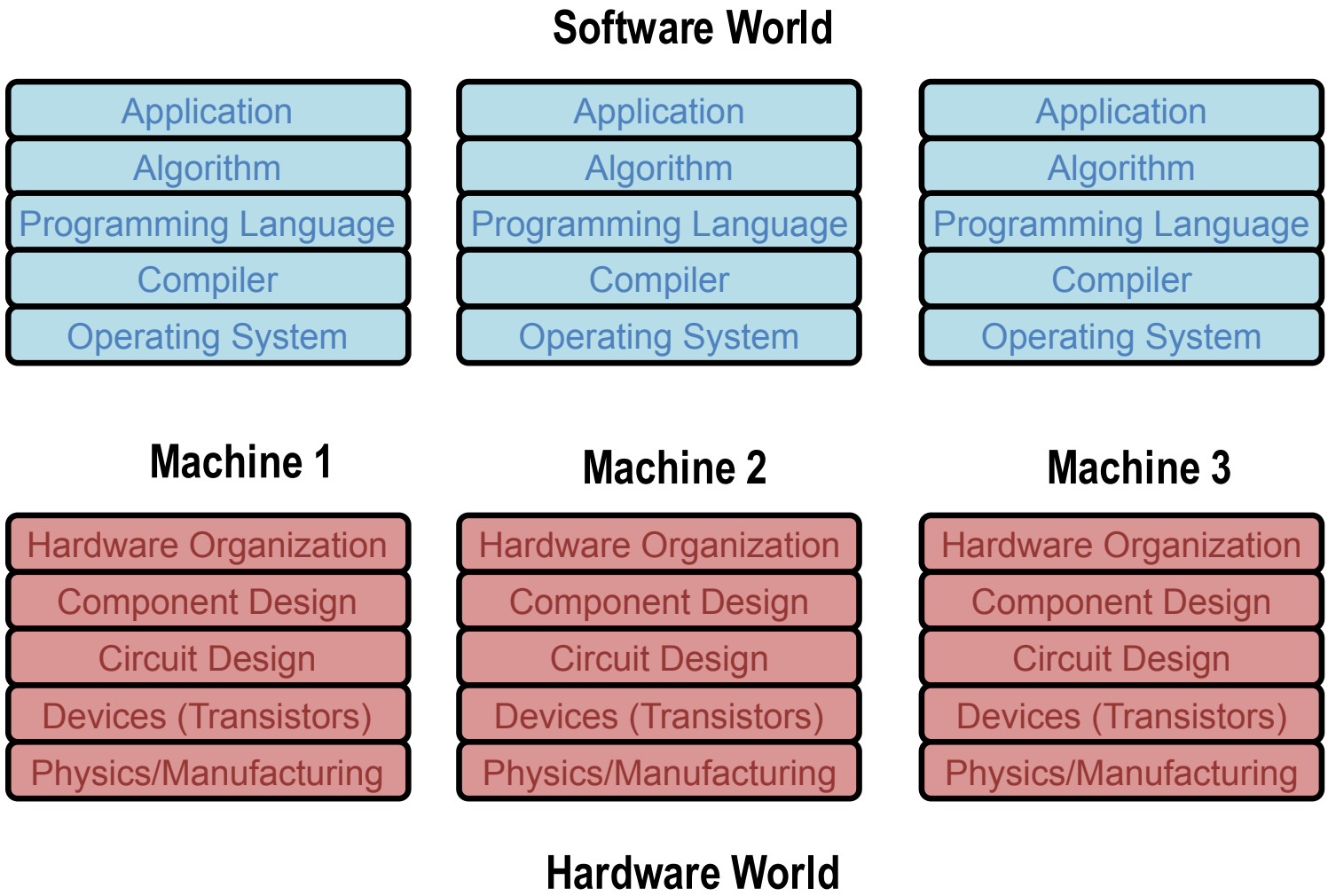
Creators: John Mauchly, J. Presper Eckert

John Von Neumann -> popularized the concept of ISA

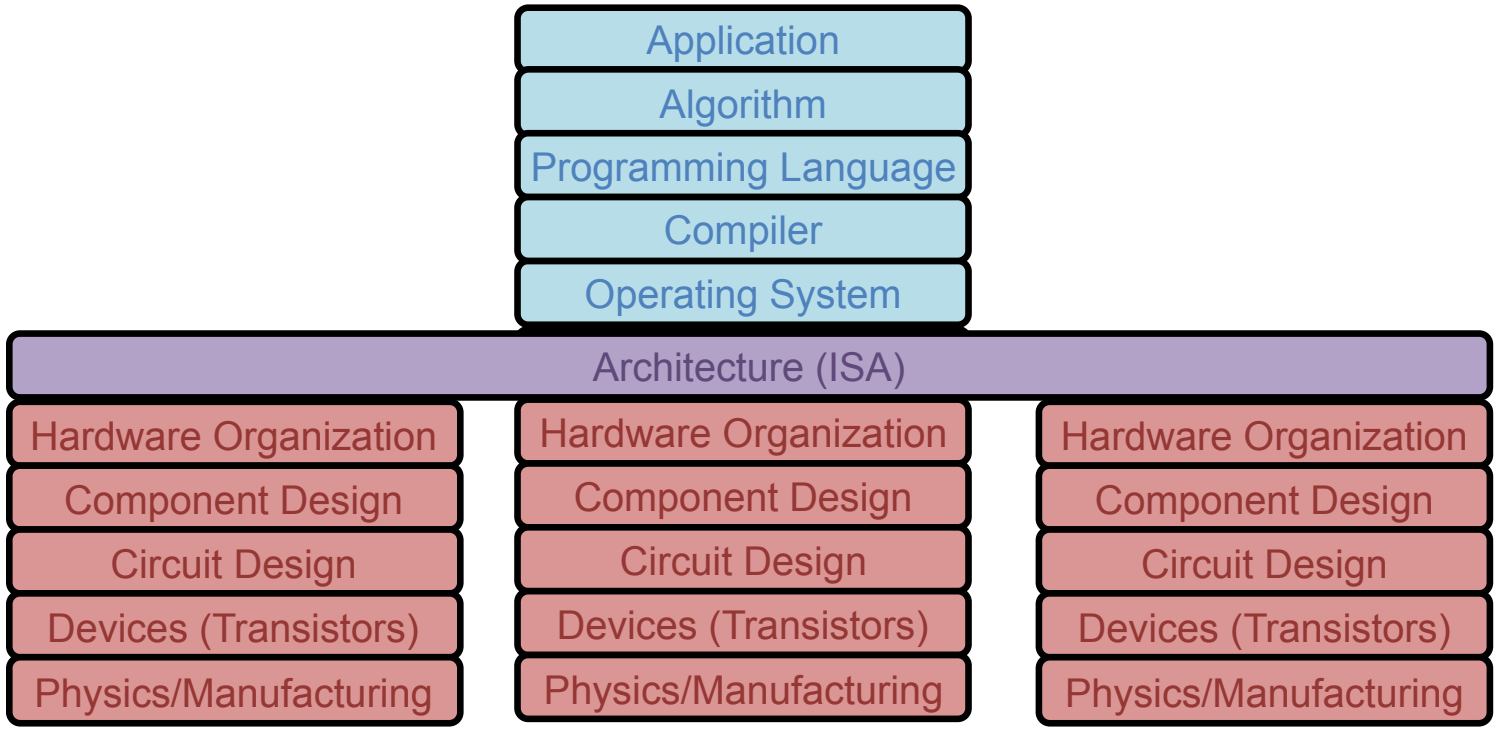


Instruction Set Architecture (ISA): ”defines how the CPU is controlled by the software” and “acts as an interface between the hardware and the software, specifying both what the processor is capable of doing as well as how it gets done”

Citation: https://www.arm.com/glossary


Software World
Application
Algorithm
Programming Language
Compiler
Operating System
Application
Algorithm
Programming Language
Compiler
Operating System
Application
Algorithm
Programming Language
Compiler
Operating System
Machine 1
Machine 2
Machine 3
Hardware Organization
Component Design
Circuit Design
Devices (Transistors)
Physics/Manufacturing
Hardware Organization
Component Design
Circuit Design
Devices (Transistors)
Physics/Manufacturing
Hardware Organization
Component Design
Circuit Design
Devices (Transistors)
Physics/Manufacturing
Hardware World

Architecture



Machine 1

Better for server workloads

Machine 2

Better for desktop/gaming

Machine 3

Better for IoT/mobile applications

What should be in an ISA?

Enough to run any program in any language…

C Code

```c
#include <stdio.h>

int a[100], b[100];

int main(int argc, char** argv) {
  int c;

  for(int i = 0; i < 100; ++i) {
    c = c + a[i] * b[i];
  }

  if(c > 5) {
    printf("yay!\n");
  }

}
```

Python

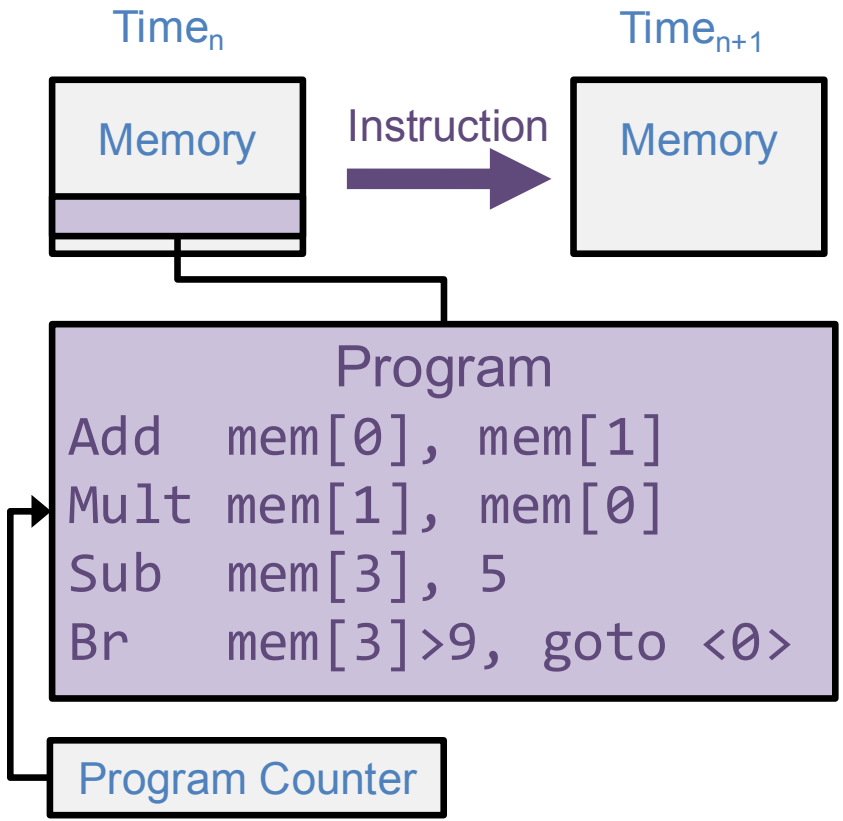
```python
import fnmatch
import os

images = ['*.jpg', '*.jpeg', '*.png', '*.tif',
'*.tiff']
matches = []

for root, dirnames, filenames in os.walk("C:\"):
    for extensions in images:
        for filename in fnmatch.filter(filenames,
extensions):
            matches.append(os.path.join(root,
filename))
```

Javascript

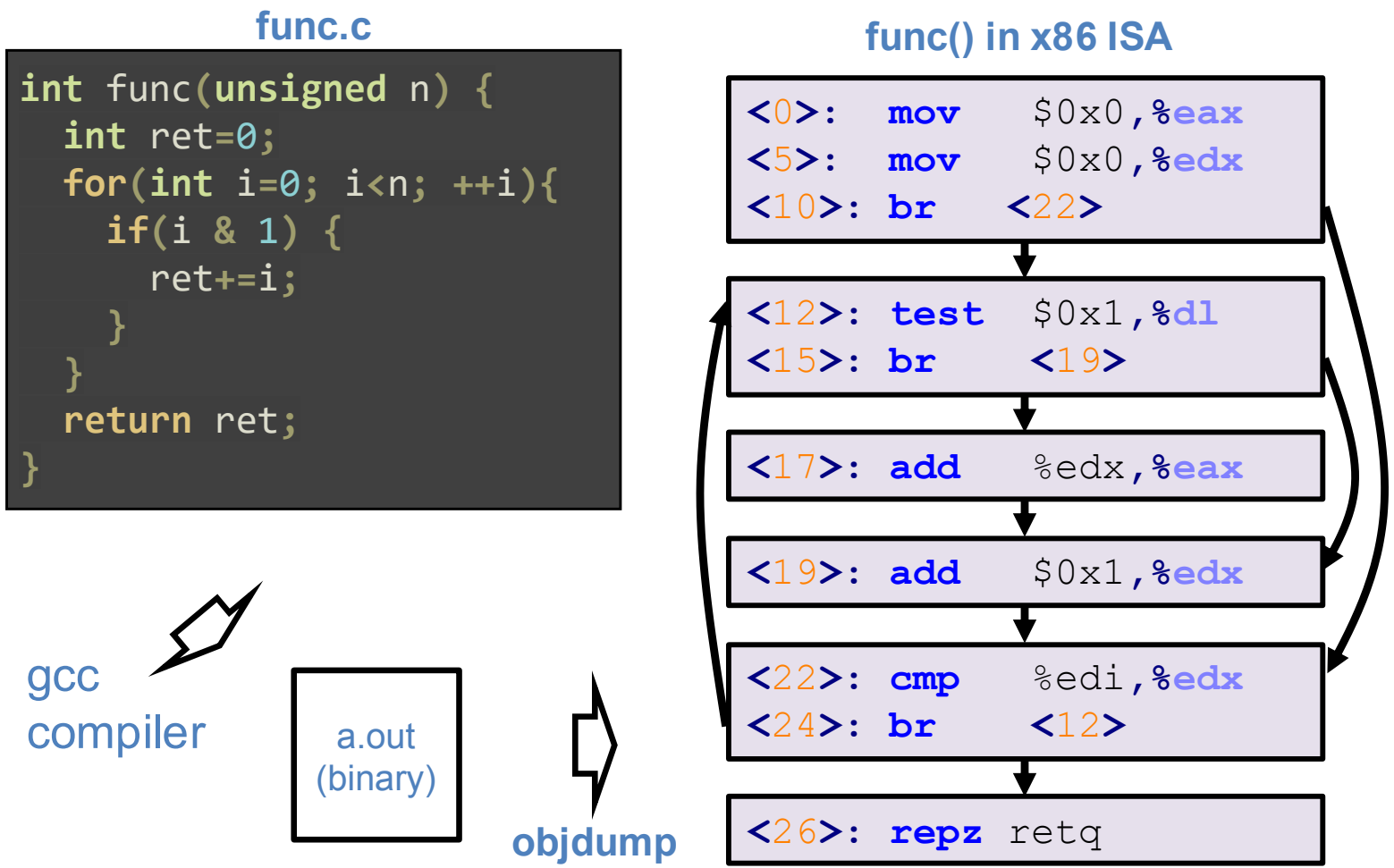
```javascript
<SCRIPT LANGUAGE="JavaScript">
var now = new Date();

var days = new
Array('Sunday','Monday','Tuesday','Wednesday','Thursday','Fri
day','Saturday');

var months = new
Array('January','February','March','April','May','June','July
','August','September','October','November','December');

var date = ((now.getDate()<10) ? "0" : "")+ now.getDate();

function fourdigits(number)          {
            return (number < 1000) ? number + 1900 : number;
```

What should be in an ISA (Instruction Set Architecture)?

- **Ingredients:**
  - **Memory:** a place to put values (state, variables, etc.)
  - **Instructions:** moves from one state to the next
  - **Program:** set of instructions (lets put it in memory)
  - **Execution model:** When do we execute each instruction?
- **Von Neuman Execution:**
  - Most common model today
  - Instructions are executed sequentially, **defined by a "program counter"**
  - Branch instruction (br)



How does software use the ISA?



Software

Interface

Hardware

C Program

```
void main() {
  while(1) {…}
}
```

Compiler

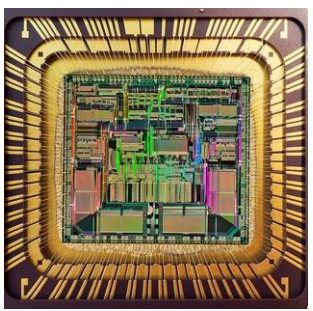
binary

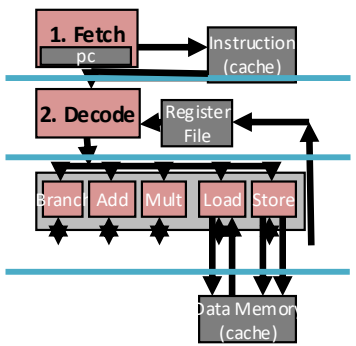
```
push %rbp
mov  %rsp,%rbp
jmp  0x4004bc
```

process

Operating System

Instruction Set Architecture (ISA): X86, ARM, RISCV…





"microarchitecture"

Course Outline

Tentative Schedule

Week	Day	Topic	Readings	Due
1	4/1 4/3 4/4	Intro+Bits and Bytes Integers	Ch. 1, 2.1, 2.2 Ch. 2.1-2.3	 DW 1 & Pod contract
2	4/8 4/10 4/11	Assembly I: Basics Assembly I: Basics	Ch 3.1-3.5	 DW 2 & Warmup Lab
3	4/15 4/17 4/18	Assembly II: Control Assembly III: Procedures	Ch 3.6 Ch 3.7	 DW 3 & Data Lab
4	4/22 4/24 4/25	Assemby IV: Data Assembly V: Advanced	Ch 3.8-3.9 Ch. 3.10	 DW 4
5	4/29 4/30 5/1 5/2	Floating Point Midterm Review	Ch. 2.4	 Bomb Lab DW5
6	5/6 5/8 5/9	**Midterm Exam** Program Optimization	Ch. 5	 DW 6
7	5/13 5/15 5/16	Memory Hierarchy Caches	Ch 6.1-6.4 Ch. 6.5-6.7	 DW 7
8	5/20 5/22 5/23	Multi-threading Thread Synchronization	Ch 12.1-12.3 Ch 12.4-12.7	 DW 8 & Attack Lab
9	5/27 5/29 5/30	Virtual Memory Exceptions, Processes Linking	Ch. 9.1-9.8 Ch. 8.1-8.3 Ch. 7.1-7.9	 DW 9
10	6/3 6/5 6/6	MIPS Final Exam Review		 DW 10 & Parallel Lab
	6/11	**Final exam 11:30 am - 2:30 pm**		

Data and Environment

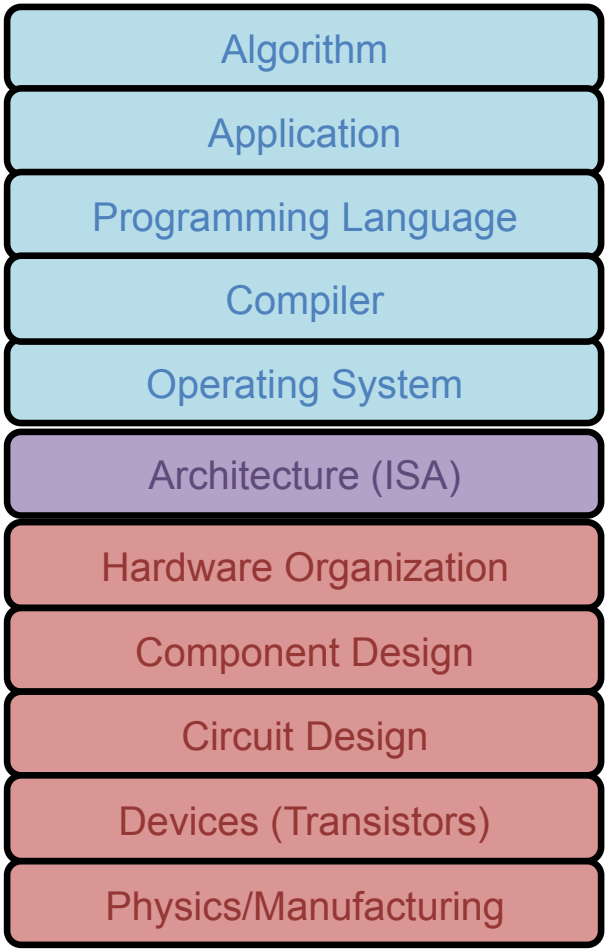
Please make sure you have an Active SEAS account.

Instructions for how to obtain announced on BruinLearn

Course Outline

Tentative Schedule

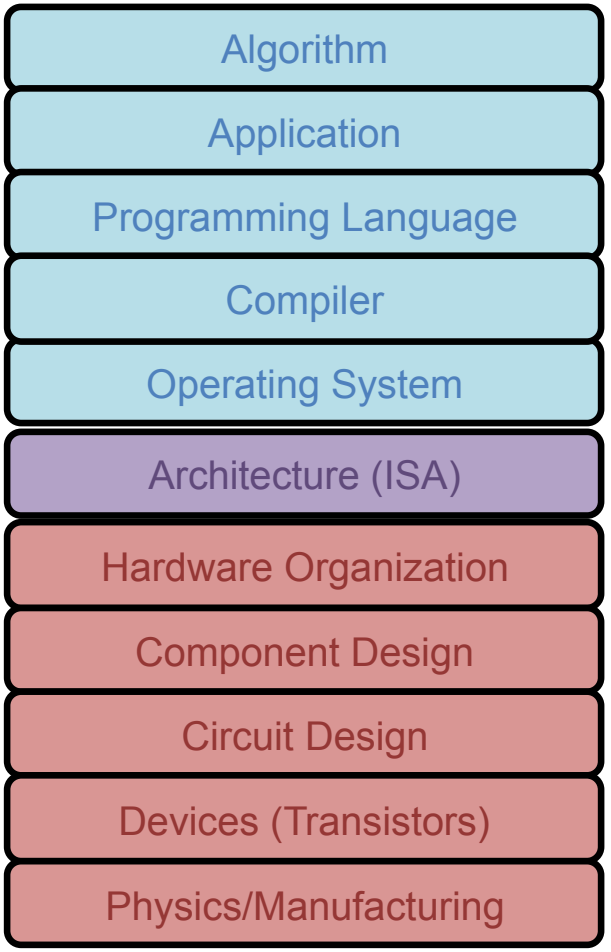
Week	Day	Topic	Readings	Due
1	4/1	Intro+Bits and Bytes	Ch. 1, 2.1, 2.2	
	4/3	Integers	Ch. 2.1-2.3	
	4/4			DW 1 & Pod contract
2	4/8	Assembly I: Basics	Ch 3.1-3.5	
	4/10	Assembly I: Basics		
	4/11			DW 2 & Warmup Lab
3	4/15	Assembly II: Control	Ch 3.6	
	4/17	Assembly III: Procedures	Ch 3.7	
	4/18			DW 3 & Data Lab
4	4/22	Assemby IV: Data	Ch 3.8-3.9	
	4/24	Assembly V: Advanced	Ch. 3.10	
	4/25			DW 4
5	4/29	Floating Point	Ch. 2.4	
	4/30			Bomb Lab
	5/1	Midterm Review		
	5/2			DW5
6	5/6	**Midterm Exam**		
	5/8	Program Optimization	Ch. 5	
	5/9			DW 6
7	5/13	Memory Hierarchy	Ch 6.1-6.4	
	5/15	Caches	Ch. 6.5-6.7	
	5/16			DW 7
8	5/20	Multi-threading	Ch 12.1-12.3	
	5/22	Thread Synchronization	Ch 12.4-12.7	
	5/23			DW 8 & Attack Lab
9	5/27	Virtual Memory	Ch. 9.1-9.8	
	5/29	Exceptions, Processes	Ch. 8.1-8.3	
		Linking	Ch. 7.1-7.9	
	5/30			DW 9
10	6/3	MIPS		
	6/5	Final Exam Review		
	6/6			DW 10 & Parallel Lab
	6/11	**Final exam 11:30 am - 2:30 pm**		



Course Outline

Tentative Schedule

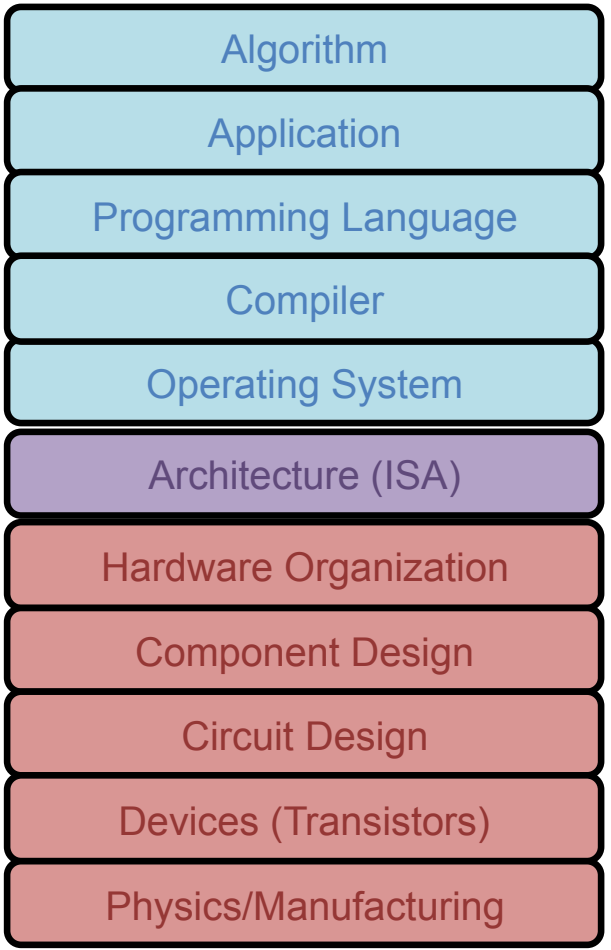
Week	Day	Topic	Readings	Due
1	4/1 4/3 4/4	Intro+Bits and Bytes Integers	Ch. 1, 2.1, 2.2 Ch. 2.1-2.3	 DW 1 & Pod contract
2	4/8 4/10 4/11	Assembly I: Basics Assembly I: Basics	Ch 3.1-3.5	 DW 2 & Warmup Lab
3	4/15 4/17 4/18	Assembly II: Control Assembly III: Procedures	Ch 3.6 Ch 3.7	 DW 3 & Data Lab
4	4/22 4/24 4/25	Assemby IV: Data Assembly V: Advanced	Ch 3.8-3.9 Ch. 3.10	 DW 4
5	4/29 4/30 5/1 5/2	Floating Point Midterm Review	Ch. 2.4	 Bomb Lab DW5
6	5/6 5/8 5/9	Midterm Exam Program Optimization	Ch. 5	 DW 6
7	5/13 5/15 5/16	Memory Hierarchy Caches	Ch 6.1-6.4 Ch. 6.5-6.7	 DW 7
8	5/20 5/22 5/23	Multi-threading Thread Synchronization	Ch 12.1-12.3 Ch 12.4-12.7	 DW 8 & Attack Lab
9	5/27 5/29 5/30	Virtual Memory Exceptions, Processes Linking	Ch. 9.1-9.8 Ch. 8.1-8.3 Ch. 7.1-7.9	 DW 9
10	6/3 6/5 6/6	MIPS Final Exam Review		 DW 10 & Parallel Lab
	6/11	**Final exam 11:30 am - 2:30 pm**		



Course Outline

Tentative Schedule

Week	Day	Topic	Readings	Due
1	4/1	Intro+Bits and Bytes	Ch. 1, 2.1, 2.2	
	4/3	Integers	Ch. 2.1-2.3	
	4/4			DW 1 & Pod contract
2	4/8	Assembly I: Basics	Ch 3.1-3.5	
	4/10	Assembly I: Basics		
	4/11			DW 2 & Warmup Lab
3	4/15	Assembly II: Control	Ch 3.6	
	4/17	Assembly III: Procedures	Ch 3.7	
	4/18			DW 3 & Data Lab
4	4/22	Assemby IV: Data	Ch 3.8-3.9	
	4/24	Assembly V: Advanced	Ch. 3.10	
	4/25			DW 4
5	4/29	Floating Point	Ch. 2.4	
	4/30			Bomb Lab
	5/1	Midterm Review		
	5/2			DW5
6	5/6	**Midterm Exam**		
	5/8	Program Optimization	Ch. 5	
	5/9			DW 6
7	5/13	Memory Hierarchy	Ch 6.1-6.4	
	5/15	Caches	Ch. 6.5-6.7	
	5/16			DW 7
8	5/20	Multi-threading	Ch 12.1-12.3	
	5/22	Thread Synchronization	Ch 12.4-12.7	
	5/23			DW 8 & Attack Lab
9	5/27	Virtual Memory	Ch. 9.1-9.8	
	5/29	Exceptions, Processes	Ch. 8.1-8.3	
		Linking	Ch. 7.1-7.9	
	5/30			DW 9
10	6/3	MIPS		
	6/5	Final Exam Review		
	6/6			DW 10 & Parallel Lab
	6/11	**Final exam 11:30 am - 2:30 pm**		



Main Course Theme: Abstraction Is Good, but …

- **Abstractions have limits!**
  - Important for correct programs
    - Abstractions don't necessarily conform to expectations
  - Important for Security
    - Abstractions can often be circumvented
  - Important for Performance
    - Abstractions tend to hide performance implications

Example 1: `int`'s are not Integers, `float`'s are not Reals

- **Example 1: Is $x^2 \geq 0$?**
  - Float's: Yes!
  - Int's:
    - 40000 * 40000 --> 1600000000
    - 50000 * 50000 --> ?? overflow

Example 1: `int`'s are not Integers, `float`'s are not Reals

```
 8
 9  #include <stdio.h>
10
11  int main()
12  {
13      int test = 50000;
14      test *= test;
15      printf("Big number:%d",test );
16
17      return 0;
18  }
```

input

```
Big number:-1794967296
```

Example 1: `int`'s are not Integers, `float`'s are not Reals

- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Example 1: int's are not Integers, float's are not Reals

```
/*****************************************************************************/

#include <stdio.h>

int main()
{
    float test = (1e20 + -1e20) + 3;
    printf("Associativity test 1: %f \n",test );
    test = 1e20 + (-1e20 + 3);
    printf("Associativity test 2: %f \n",test );

    return 0;
}
```

input

```
Associativity test 1: 3.000000
Associativity test 2: 0.000000
```

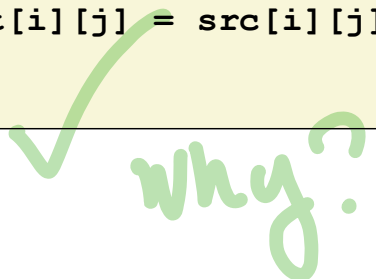
Example 2: Memory System Performance Example

let n = 2048

```
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];
}
```

$O(n^2)$



Example 2: Memory System Performance Example

```
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];
}
```

4.3ms

```
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];
}
```

81.8ms

- **Performance depends on access patterns**
  - Here: systems are optimized for “contiguous” access
- **There’s more to performance than asymptotic complexity!**
  - Many levels: Algorithm, data type/layout, schedule/order, parallelism’
  - Must understand system to optimize well (compiler, OS, hardware, etc.)

Highlights in Computer systems

Finite Number Representations

Von Neuman ISAs: Virtualize H/W

Processes + OS: Illusion of infinite cores

Compiler Transformations

Locality: proximity in time & space

Virtual Memory: Address Space Illusion

Processor's Role: Instruction Level Parallelism

Caches: bridge huge memories and fast memories

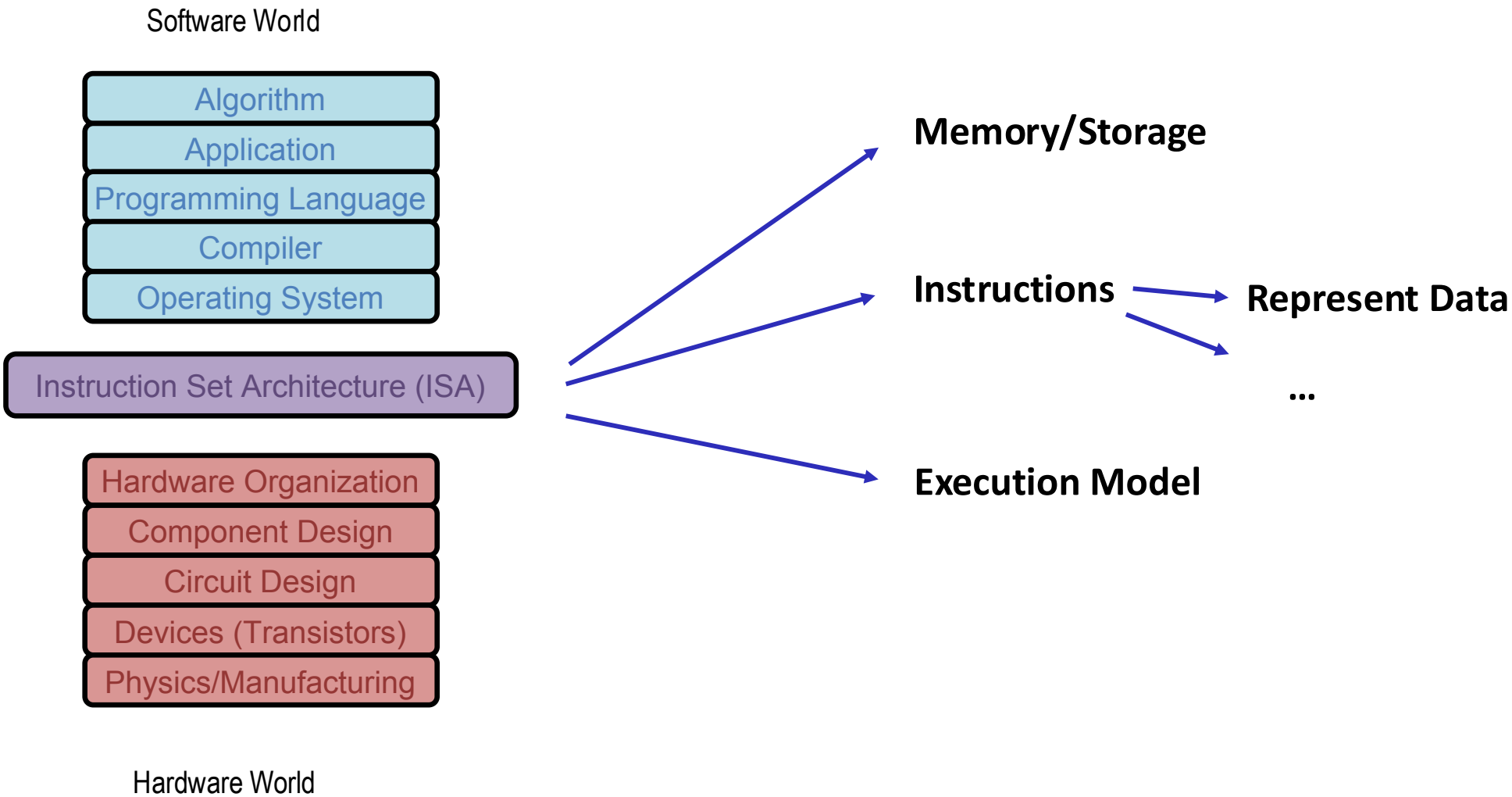
Multithreading: Free-for-all parallelism

Introduce yourselves…

1. **What are you most excited or nervous about taking CS33?**
2. **Please share something that has nothing to do work or school.**

Objectives for Today

1. Introduction, Course Administration, Course Overview
2. Representing data in binary
3. Boolean operations
4. Activity: Forming Learning Pods

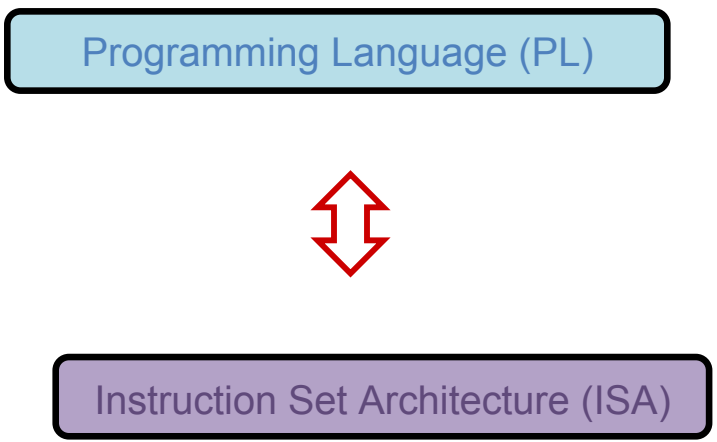

Software World
Algorithm
Application
Programming Language
Compiler
Operating System
Instruction Set Architecture (ISA)
Hardware Organization
Component Design
Circuit Design
Devices (Transistors)
Physics/Manufacturing
Hardware World
Memory/Storage
Instructions
Represent Data
...
Execution Model

Representing information as bits

Datatypes

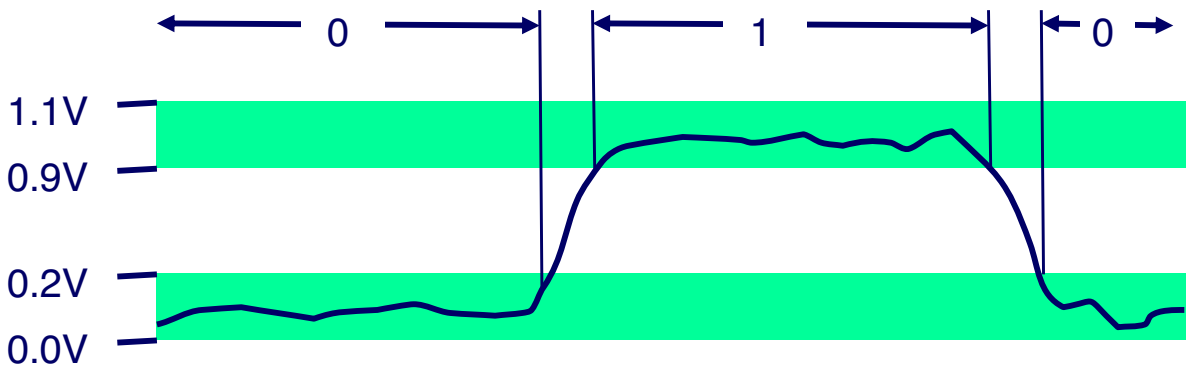
Operations

Relationships from PL<->ISA



Everything is bits

- **Each bit is 0 or 1**
- **By encoding/interpreting sets of bits in various ways**
  - Computers determine what to do (instructions)
  - … and represent and manipulate numbers, sets, strings, etc…
- **Why bits? Electronic Implementation**
  - Easy to store with bistable elements
  - Reliably transmitted on noisy and inaccurate wires



Number Systems

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Decimal		10^3 10^2 10^1 10^0
5142		5 1 4 2

5x1000 + 1x100 + 4x10 + 2x1

Number Systems

0, 1

What is our algorithm to do this?

Binary $2^3\ 2^2\ 2^1\ 2^0$

0110 → **0 1 1 0**

0x8 + 1x4 + 1x2 + 0x1 = 6

Converting between other bases and decimal

$$d_{l-1}d_{l-2}\ldots d_1 d_0 \quad \Longrightarrow \quad num = \sum_{i=0}^{l-1} d_i * b^i$$

- Here d are the **digits** in decimal, l is **length** of the number being converted to decimal and b is the **base** such as 2 for binary

Translation:

num = 0

For each digit from position 0 to l-1:

    Multiply digit by base raised to the position

    Add to the num

Return num

Your turn….

$b = 2$

- **Convert 100001 in binary to decimal**

$l = 6$

$l - 1 = 6 - 1 = 5$

$$\sum_{i=0} d_i * 2^i$$

$$1 \times 2^0 + 0 * 2^1 + 0 * 2^2 + 0 * 2^3 + 0 \times 2^4 + 1 * 2^5 =$$

$$32 = 33$$

1 +

Byte: A group of 8 bits

$2^6 - 1 = 63$

"?"

- **Integer Range**
  - Decimal: 0_{10} to 255_{10}
  - Binary: 00000000_2 to 11111111_2
- **Why do we need the concept of a byte?**
  - It's totally arbitrary
  - But, we do kind of need some smallest unit of data…
    - 8 is a power of 2
    - Only need 8 bits to represent common characters (ASCII)
    - Smallest useful datatype?

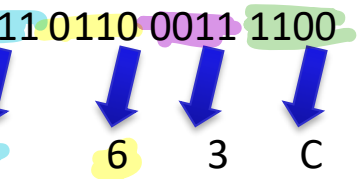
Hexadecimal (base 16)

- Representing Base 16:
  - Use characters ‘0’ to ‘9’ and ‘A’ to ‘F’
  - Write FA1D37B$_{16}$ in C have “0x” prefix:
    - “0xFA1D37B” of “0xfa1d37b”
- Relationship to bytes:
  - Hex represents 4 bits each.
  - A byte in hex is 00$_{16}$ to FF$_{16}$
- Why Hexadecimal?
  - Compact representation that keeps association with binary

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

Binary to Hex

- **0b0111011000111100**
  - Split into 4 bit chunks: 0b 0111 0110 0011 1100
  - Convert each 4 bit chunk to hex: 0x 7 6 3 C
- **0x763C**



- **Hex is a convenient way for us to represent binary values**

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

Your turn....

$1 = 4$ $\quad \sum_{i=0}^{3} d_i * 16^i$

Convert Hexadecimal to Decimal:

3 2 1 0

3DA2

translate d_i to base 10

$2*16^0 + 10*16^1 + 13*16^2 + 3*16^3 =$

15778

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

C Datatypes

- **Datatypes in C have sizes that are multiples of 8**
  - Many reasons, e.g.: makes it easier to calculate addresses of data
- **Most ISAs use similar datatypes**
  - Keeping abstraction's the same makes it easier to write efficient programs

C Data Type	Size in Bytes
`char`	1
`short`	2
`int`	4
`long`	8
`float`	4
`double`	8
`pointer`	8

→ 8 byte addresses for x86

Caveat: The size of numerical datatypes in C depends on the ISA targeted

Objectives for Today

1. Introduction, Course Administration, Course Overview
2. Representing data in binary
3. Boolean operations
4. Activity: Forming Learning Pods

56

Truth table for Logical And

A **truth table** shows the truth values of a compound propositional formula.

A **truth assignment** is an assignment of true (T) or false (F) to each propositional variable.

A truth assignment **satisfies** a formula if it makes the formula true.

p = "7 is prime." q = "25 is divisible by 2."

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

```
int x=5;
int y=7;
p: x<10
q: y>0
```

Truth Table for Logical Inclusive OR

A **truth table** shows the truth values of a compound propositional formula.

A **truth assignment** is an assignment of true (T) or false (F) to each propositional variable.

A truth assignment **satisfies** a formula if it makes the formula true.

p = "7 is prime." q = "25 is divisible by 2."

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Truth Table for Logical Negation

A **truth table** shows the truth values of a compound propositional formula.

A **truth assignment** is an assignment of true (T) or false (F) to each propositional variable.

A truth assignment **satisfies** a formula if it makes the formula true.

p = “7 is prime.”

p	$\neg p$	$\neg\neg p$
T	F	T
F	T	F

Truth Table for Logical Exclusive OR

A **truth table** shows the truth values of a compound propositional formula.

A **truth assignment** is an assignment of true (T) or false (F) to each propositional variable.

A truth assignment **satisfies** a formula if it makes the formula true.

p = “7 is prime.” q = “25 is divisible by 2.”

p	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Recall: Logic Operations in C

- **Logical Operators**
  - And: &&, Or: ||, Negation: !
    - View 0 as “False”
    - Anything nonzero as “True”
    - Always return 0 or 1
    - Early termination (don’t evaluate what you don’t have to)

Recall: Logic Operations in C

- **Examples (char data type)**
  - !0x41 → 0x00
  - !0x00 → 0x01
  - !!0x41 → !(!0x41) → !(0x00) → 0x01
  - 0x69 && 0x55 → 0x01
  - 0x69 || 0x55 → 0x01
  - Let p be a pointer. Why is the following useful?

p && *p 1st clause p checks if pointer is non-null

2nd check: if value being referenced is non-zero

Boolean Algebra

Developed by George Boole in 19th Century

Algebraic representation of logic: Encode "True" as 1 and "False" as 0

Bit And

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

&	0	1
0	0	0
1	0	1

Boolean Algebra

Developed by George Boole in 19th Century

Algebraic representation of logic: Encode "True" as 1 and "False" as 0

Bit Or

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

|	0	1
0	0	1
1	1	1

Boolean Algebra

Developed by George Boole in 19th Century

Algebraic representation of logic: Encode "True" as 1 and "False" as 0

Bit Negation or Not

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

~	
0	1
1	0

Boolean Algebra

Developed by George Boole in 19th Century

Algebraic representation of logic: Encode "True" as 1 and "False" as 0

Bit Exclusive-Or (Xor)

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

^	0	1
0	0	1
1	1	0

General Boolean Algebras

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

```
  01101001
& 01010101
----------
  01000001
```

- **Or**

```
  01101001
| 01010101
----------
  01111101
```

- **All of the Properties of Boolean Algebra Apply**

General Boolean Algebras

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

```
  01101001
^ 01010101
----------
  00111100
```

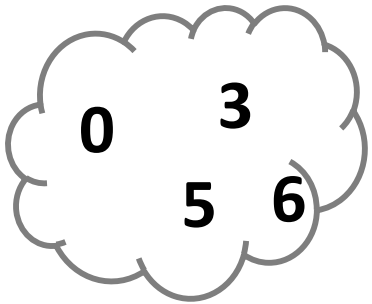
- **Negation**

```
~ 01010101
----------
  10101010
```

- **All of the Properties of Boolean Algebra Apply**

Example: Representing & Manipulating Sets

- **Representation**
  - Width w bit vector represents subsets of {0, …, w–1}
  - $a_j = 1$ if $j \in A$. A = {0,1,2,3,4,5,6,7}



Representing as sequence of bits

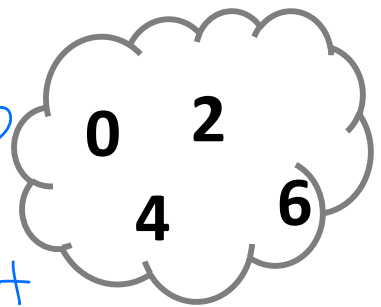
{ 0, 3, 5, 6 }

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

{ 0, 2, 4, 6 }

0 1 0 1 0 1 0 1

Use 8 bits
7 6 5 4 3 2 1 0
0 0 0 0 0 0 0 0
= ∅



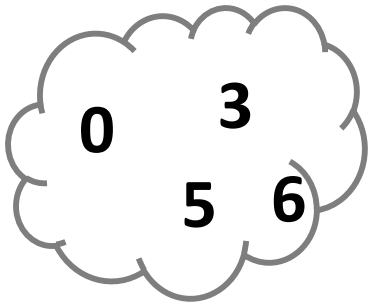
set bit equal to 1 if and only if set contains that element

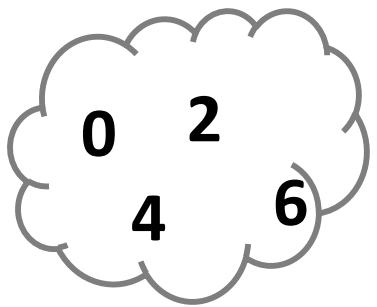
Example: Representing & Manipulating Sets

- **Representation**
  - Width w bit vector represents subsets of {0, …, w–1}
  - $a_j = 1$ if $j \in A$. A = {0,1,2,3,4,5,6,7}

    01000001 = {0, 6}

    &
    - 01101001 { 0, 3, 5, 6 }
    - 01010101 { 0, 2, 4, 6 }
- **Operations**
  - & Intersection

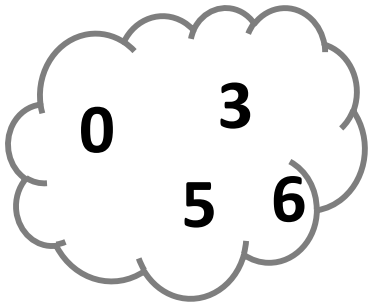


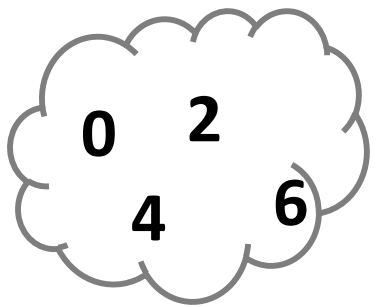


Example: Representing & Manipulating Sets

- **Representation**
  - Width w bit vector represents subsets of {0, …, w–1}
  - $a_j = 1$ if $j \in A$. A = {0,1,2,3,4,5,6,7}
    - 01101001 { 0, 3, 5, 6 }
    - 01010101 { 0, 2, 4, 6 }
- **Operations**
  - | Union

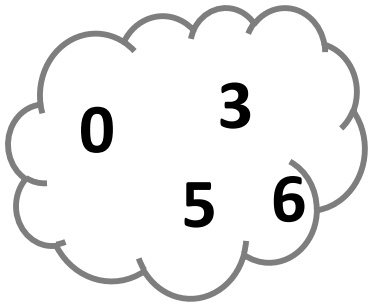
→ 01111101 = {0,2,3,4,5,6}



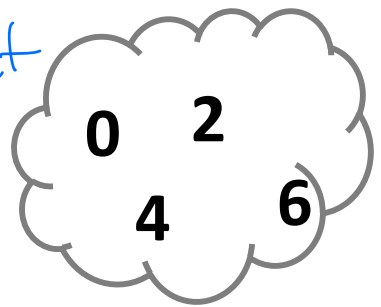


Example: Representing & Manipulating Sets

- **Representation**
  - Width w bit vector represents subsets of {0, …, w–1}
  - $a_j = 1$ if $j \in A$. A = {0,1,2,3,4,5,6,7}
    - 01101001 { 0, 3, 5, 6 }
    - 01010101 { 0, 2, 4, 6 }
- **Operations**
  - ^ Symmetric difference



Symmetric difference is all elements in exactly one set but not both

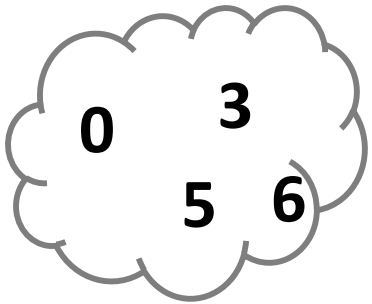


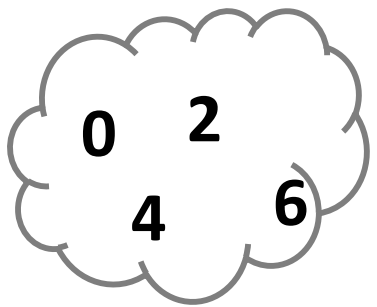
$$\begin{array}{r} 01101001 \\ \wedge\ 01010101 \\ \hline 00111100 \end{array} = \{2,3,4,5\}$$

Example: Representing & Manipulating Sets

- **Representation**
  - Width w bit vector represents subsets of {0, …, w–1}
  - $a_j = 1$ if $j \in A$. A = {0,1,2,3,4,5,6,7}
    - 01101001 { 0, 3, 5, 6 }
    - 01010101 { 0, 2, 4, 6 }
- **Operations**
  - ~ Complement of {0,2,4,6}

~ 01010101 = {7, 5, 3, 1}





Bit-Level Operations in C: Examples

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

- **Examples (Char data type)**
  - ~0x41

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

Bit-Level Operations in C: Examples

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

- **Examples (Char data type)**
  - ~0x00

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

Bit-Level Operations in C: Examples

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

- **Examples (Char data type)**
  - ~0x69 & 0x55

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

Bit-Level Operations in C: Examples

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

- **Examples (Char data type)**
  - ~0x69 | 0x55

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

Bit Operators vs Logic Operations in C

- **C Logical Operators**
  - &&, ||, !
    - View 0 as “False”
    - Anything nonzero as “True”
    - Always return 0 or 1
    - Early termination (don’t evaluate what you don’t have to)
- **Bit Operations: &, |, ~, ^**
  - Apply to any “integral” data type
    - long, int, short, char, unsigned
  - View arguments as bit vectors
  - Operations applied bit-wise

Watch out for:
&& vs. &
|| vs. |

Objectives for Today

1. Introduction, Course Administration, Course Overview
2. Representing data in binary
3. Boolean operations
4. Activity: Forming Learning Pods

79

Forming Learning Pods

1. Please look at the Learning Pod Contract Assignment on Gradescope linked via BruinLearn
2. Take this time to meet people and form a pod within your scheduled discussion section.

Objectives for Today

1. Introduction, Course Administration, Course Overview
2. Representing data in binary
3. Boolean operations
4. Activity: Forming Learning Pods

81

Thank You