ECE 152 / 496
Introduction to Computer Architecture

Intro and Overview

Benjamin C. Lee

Duke University

Slides from Daniel Sorin (Duke)
and are derived from work by
Amir Roth (Penn) and Alvy Lebeck (Duke)

Spring 2013

Instructor

- Professor: Benjamin C. Lee
    - Office: Hudson Hall 210
    - Email: benjamin.c.lee@duke.edu

      → subject of all emails to me must begin with ECE496
    - Office Hours: TBD
        - If you have class at all of these times, email me to meet

Undergrad Teaching Assistants

- Undergraduate TAs (UTAs):
  - Oliver Fang
  - James Hong
  - Amay Jhaveri
  - Oliver Fang
  - Zachary Michaelov
- Will help with
  - Answering email questions about homeworks and project
  - Holding office hours to help with CAD software
- Will NOT bail you out at 3am when deadline is at 10am

Course Website

- Course Web Page
  http://people.duke.edu/~bcl15/class/class_ece496spr13.html
  - Lecture slides available on web before or shortly after class
  - Print them out and bring them with you to class
  - Value (just reading slides) << Value (attending class)
    - Missing class = missing important course material
- Most important info for course is on website
  - Please check it before emailing me or TAs
- You are required to monitor web page
  - Homework and project assignments will appear on web page
  - I will announce them as well

Sakai and Piazza for ECE496

- I will communicate with you via email
  - You must check your email
- Sakai
  - Assignments are posted online, submitted online
  - Grades are managed online
- Piazza
  - Post all questions to Piazza
  - TA's and I will answer questions on the forum

Textbook

- Text: *Computer Organization & Design* (Patterson & Hennessy)
  - 4th edition of the textbook
  - You are expected to complete the assigned readings

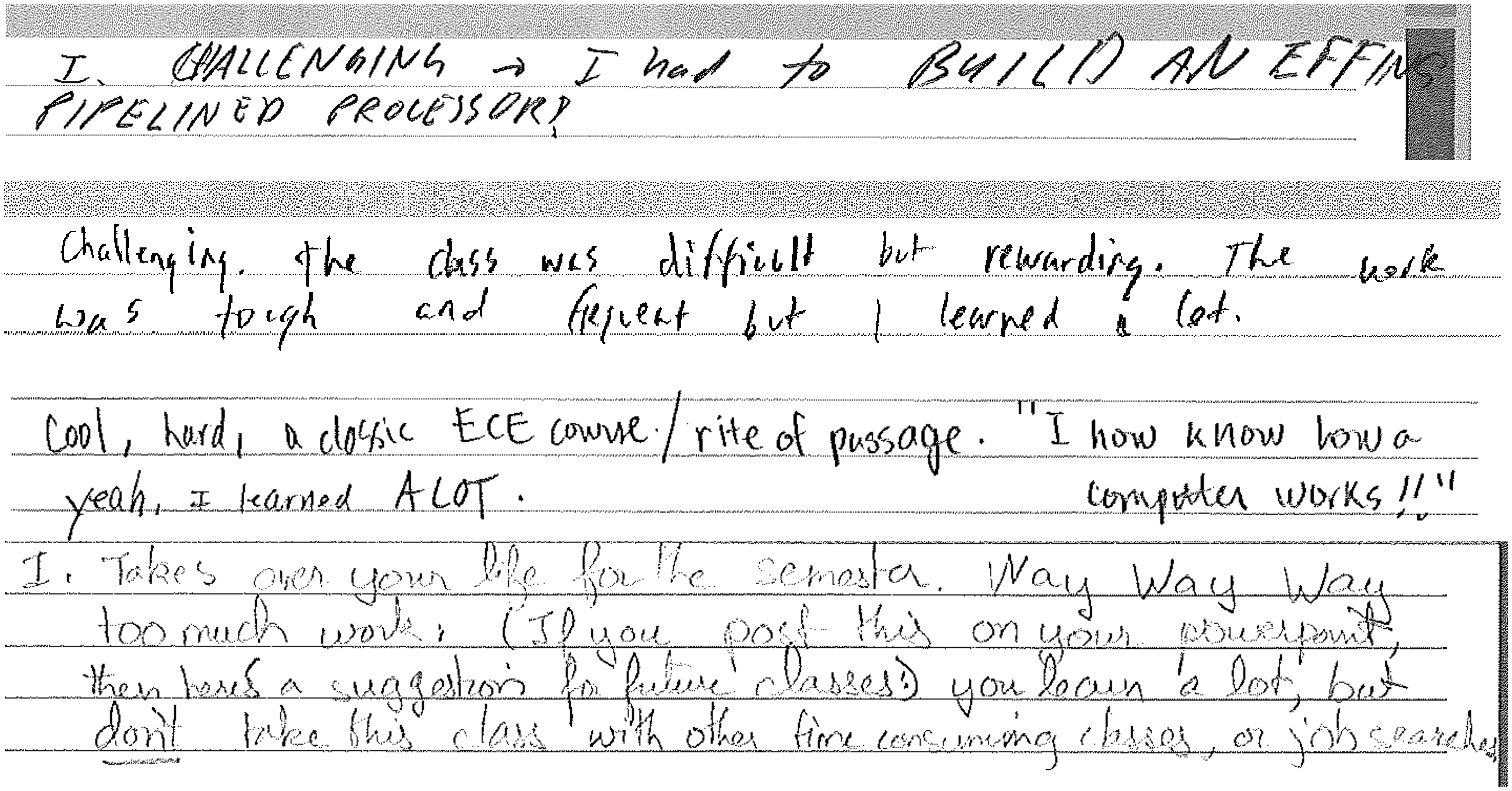
- We will not cover material in the textbook in a strictly linear fashion

Workload

- Readings from textbook
- Homework assignments (performed in groups of 2)
  - Pencil and paper problems
  - Small programming problems
- Project (performed in groups of 2)
  - Building the Duke152-S11 computer in real hardware!
  - Programming the Duke152-S11 computer you built
  - You will choose project partners, and I will ensure that group grading is done fairly by end-of-semester questionnaires
  - Project will be broken up into smaller parts to make it more manage-able
    - Project Part 1: Building a register file, due: Jan 18

This is not an easy class

- In case you don’t believe me, listen to your colleagues



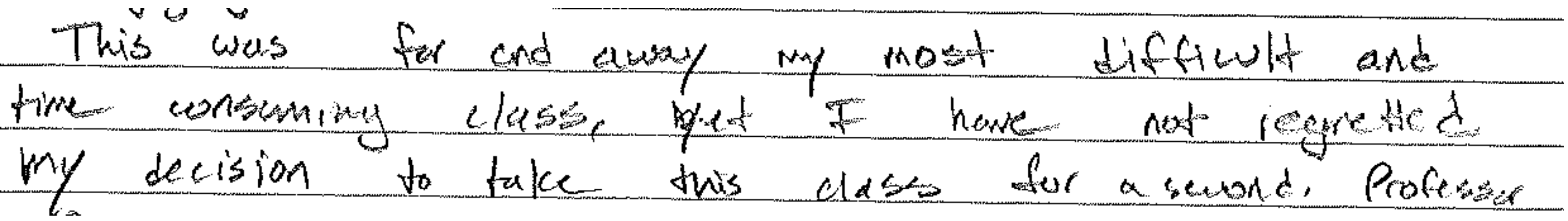
I. CHALLENGING → I had to BUILD AN EFFIN
PIPELINED PROCESSOR!

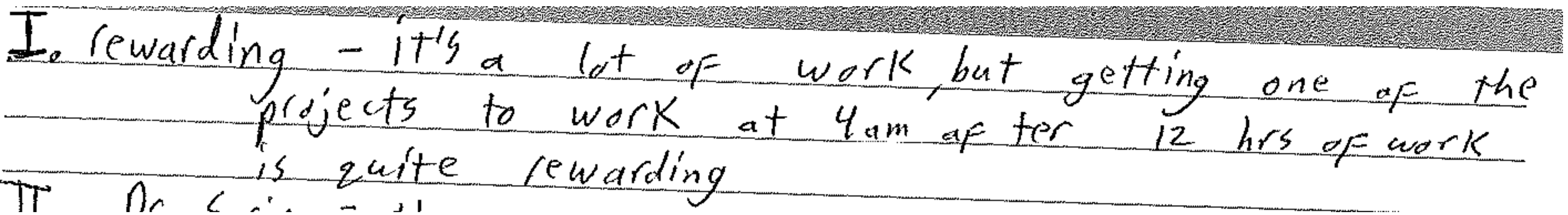
Challenging. The class was difficult but rewarding. The work
was tough and frequent but I learned a lot.

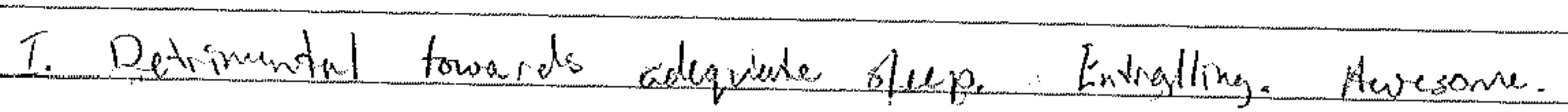
Cool, hard, a classic ECE course / rite of passage. "I now know how a
yeah, I learned ALOT. computer works!!"

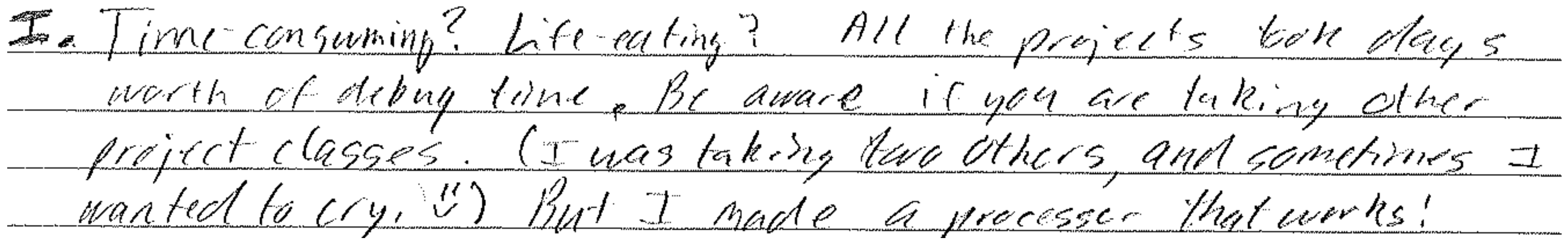
I. Takes over your life for the semester. Way Way Way
too much work. (If you post this on your powerpoint,
then here's a suggestion for future classes:) you learn 'a lot', but
don't take this class with other time consuming classes, or job searching

Seriously, this is not an easy class


This was far and away my most difficult and time consuming class, yet I have not regretted my decision to take this class for a second. Professor


I. rewarding - it's a lot of work, but getting one of the projects to work at 4am after 12 hrs of work is quite rewarding


I. Detrimental towards adequate sleep. Enthralling. Awesome.


I. Time-consuming? Life-eating? All the projects took days worth of debug time. Be aware if you are taking other project classes. (I was taking two others, and sometimes I wanted to cry. ☹) But I made a processor that works!

Grading

- Grade breakdown

Homework	20%	
Project	35%	(graded as a group, but fairly)
Midterm Exam	20%	
Final Exam	25%	

- I strongly believe in partial credit
  - Please explain your answers to get as much credit as possible
- Late homework policy
  - 10% reduction for each day late
  - No credit after the homework is graded and handed back
- Assignments take a lot of time, so start them early
  - Yes, this means you! And you and you and especially you.

Academic Misconduct

- Academic Misconduct
  - Refer to Duke Honor Code
  - Studying together in groups is encouraged
  - Homework and project must be in groups of 2
    - You may choose a different partner for each homework
    - You will choose one partner for all parts of project
  - Common examples of cheating:
    - Running out of time and using someone else's output
    - Borrowing code from someone who took course before
    - Using solutions found on the Web
- I will not tolerate any academic misconduct!
  - Historically, this course has "led the league" in cases of academic misconduct that have led to suspensions and expulsions
  - Software for detecting cheating is very, very good … and I use it

Goals of This Course

- By end of semester:
  - You will know how computers work
    - What's inside a computer?
    - How do computers run programs written in C, C++, Java, Matlab, etc.?
  - You will design your own computer, build it in real hardware, and program it!
  - You will understand the engineering tradeoffs to be made in the design and implementation of different types of computers
  - You may, like me, decide to become an architect. ☺
- If, at any point, it's not clear why I'm talking about some topic, please ask!

Outline of Introduction

- Administrivia
- What is a computer?
- What is computer architecture?
- Why are there different types of computers?
- How do we tell computers what to do?

Reading Assignment

- Patterson & Hennessy
  - Chapter 1
  - This is a short and relatively easy-to-read chapter
  - Please read it such that afterwards you'd feel comfortable teaching the material to an ECE 52 student
- For those of you who haven't done digital logic design in a while, you may want to go back to your ECE 52 textbook
  - Digital logic design is a pre-requisite for this course, but I understand if some of you haven't done this in a while
  - You should be able to design combination logic and finite state machines – otherwise, you're going to have a very rough semester

What is a Computer?

- A computer is just a digital system
  - Consists of combinational and sequential logic
  - One big, honking finite state machine
  - A computer would be a very exciting ECE 52 project
- Seriously, it's just a digital system
- Yes, but what does this digital system do?
  - Whatever you tell it to do! No more, no less
- A computer just does what software tells it to do
  - Software is a series of **instructions**
- ICQ (In-Class Question): What instructions does a computer need?

Computers Execute Instructions

- What kinds of instructions are there?
  - Arithmetic: add, subtract, multiply, divide, etc.
  - Access memory: read, write
  - Conditional: if condition, then jump to other part of program
  - What other kinds of instructions might be useful?
- How do we represent instructions?
  - Digitally! With strings of zeros and ones
  - Remember: a computer is just a digital system!
- So how do computers run programs in Java or C/C++ or Matlab or whatever whippersnappers use these days?
  - None of us write programs in binary (zeros and ones) …
  - We' ll get to this in a few minutes

Instruction Sets

- Computers can only execute instructions that are in their specific machine language
- Every type of computer has a different instruction set that it understands
  - Intel (and AMD) IA-32 (x86): Pentium, Core i7, AMD Opteron "Magny Cours", etc.
  - Intel IA-64: Itanium, Itanium 2
  - PowerPC: In Cell Processor and old Apple Macs
  - SPARC: In computers from Sun Microsystems/Oracle
  - ARM: In many embedded processors
  - MIPS: MIPS R10000 → this is the example used in the textbook
- Note: no computer executes Java or C++
  - Not even Matlab

Outline of Introduction

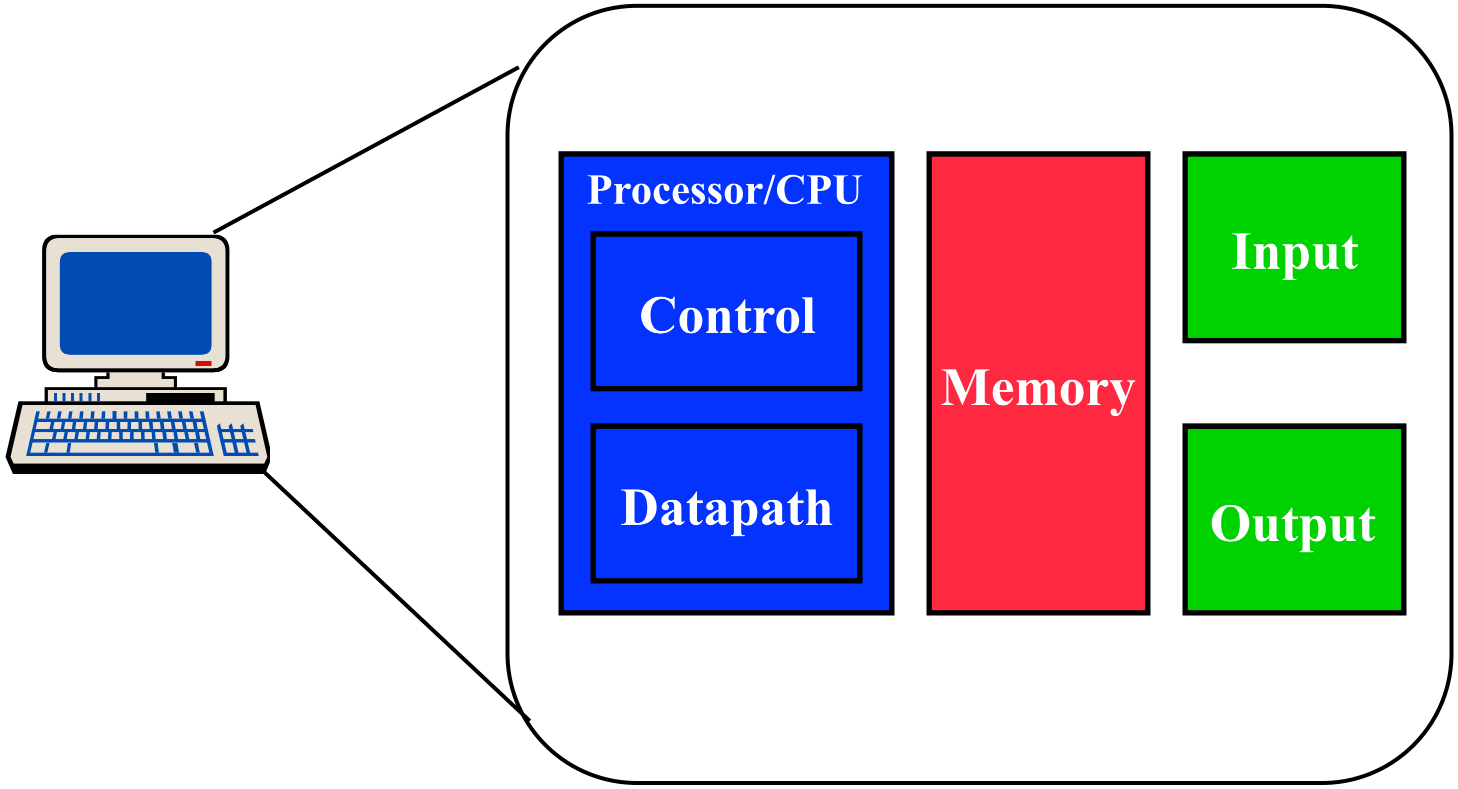
- Administrivia
- What is a computer?
- What is computer architecture?
- Why are there different types of computers?
- How do we tell computers what to do?

Hint: It Doesn't Involve Skyscrapers …

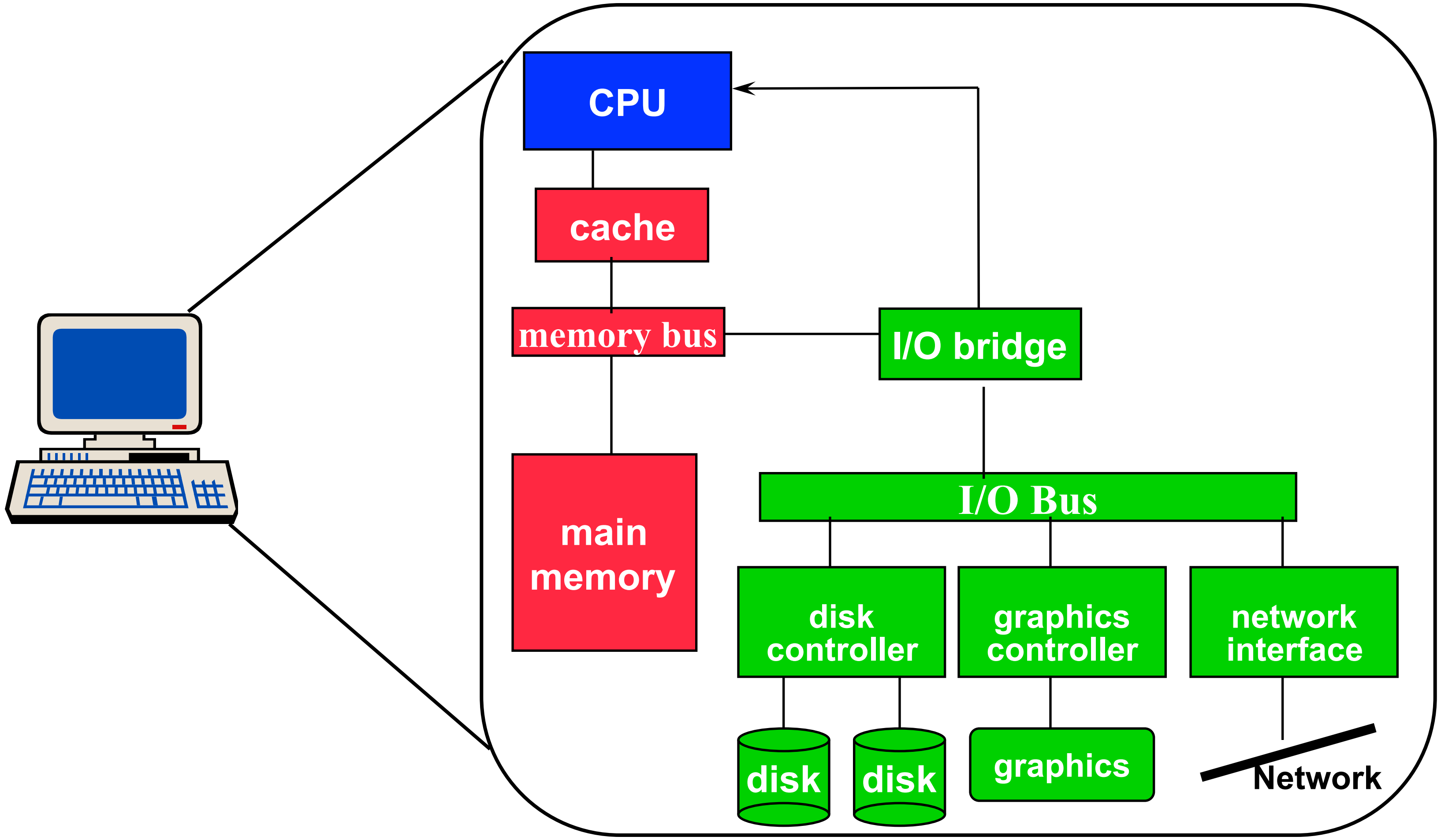
- Strictly speaking, a **computer architecture** specifies what the hardware looks like (its interface), so that we can write software to run on it
  - Exactly what instructions does it have
  - Number of register storage locations it has
  - And more that we'll learn about later in semester
- **Important point:** there are many, many different ways to build digital systems that provide the same interface to software
  - There are many **microarchitectures** that conform to same architecture
  - Some are better than others! If you don't believe me, I'll trade you my original Intel Pentium for your Intel Core i7
- ICQ: So what's inside one of these digital systems?

The Inside of a Computer

- The Five Classic Components of a Computer

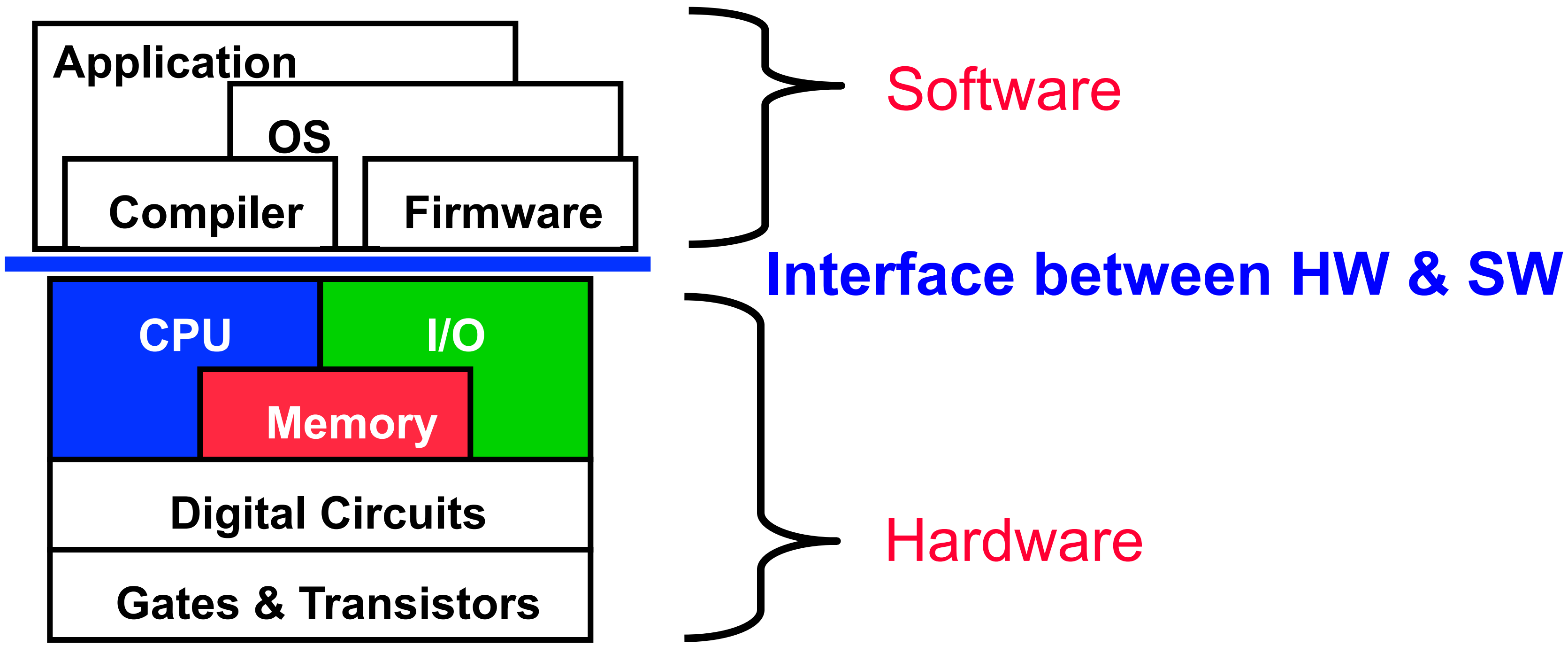


System Organization



What Is ECE 152 All About?

- Architecture = interface between hardware and software



- ECE 152 = design of CPU, memory, and I/O

Outline of Introduction

- Administrivia
- What is a computer?
- What is computer architecture?
- Why are there different types of computers?
- How do we tell computers what to do?

Differences Between Computers

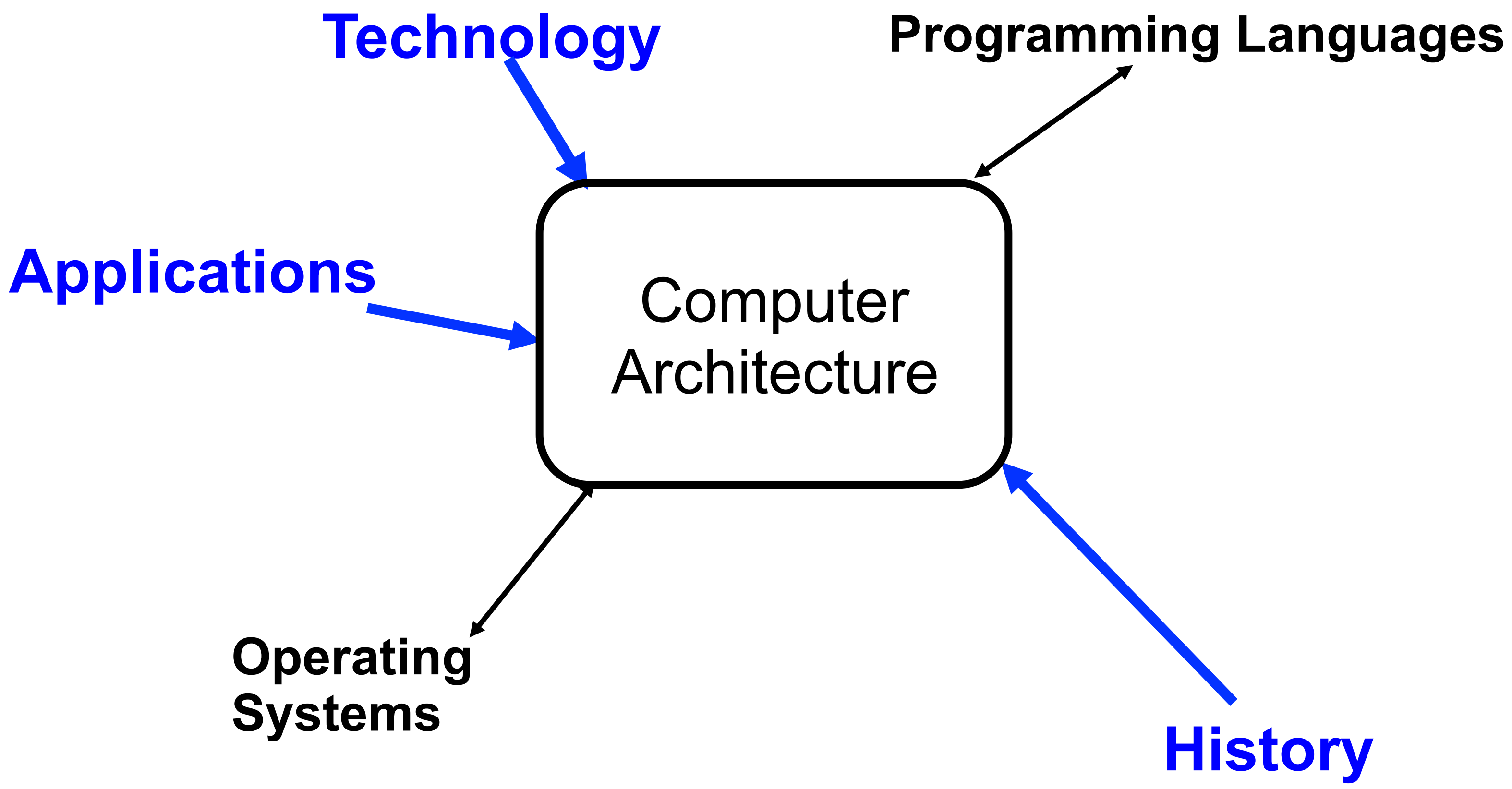
- We have different computers for different purposes
- Some can achieve performance needed for high-performance gaming
  - E.g., Cell Processor in PlayStation 3
- Others can achieve decent enough performance for laptop without using too much power
  - E.g., Intel Pentium M (for Mobile)
- And yet others can function reliably enough to be trusted with the control of your car' s brakes

ICQ: What computers do you use?
ICQ: Which of those computers do you own?

Kinds of Computers

- “Traditional” personal computers
  - Laptop, desktop, netbook
- Less-traditional personal computers
  - iPad, iPhone, Blackberry, iPod, Xbox, Wii, etc.
- Hidden “big” computers
  - Mainframes and servers for business, science, government
    - E.g., the unreliable servers that run Duke email
  - Google has tens of thousands of computers (that you don’t see)
- Hidden embedded computers
  - Controllers for cars, airplanes, ATMs, toasters, DVD players, etc.
  - Far and away the largest market for computers!
- Other kinds of computers??

Forces on Computer Architecture



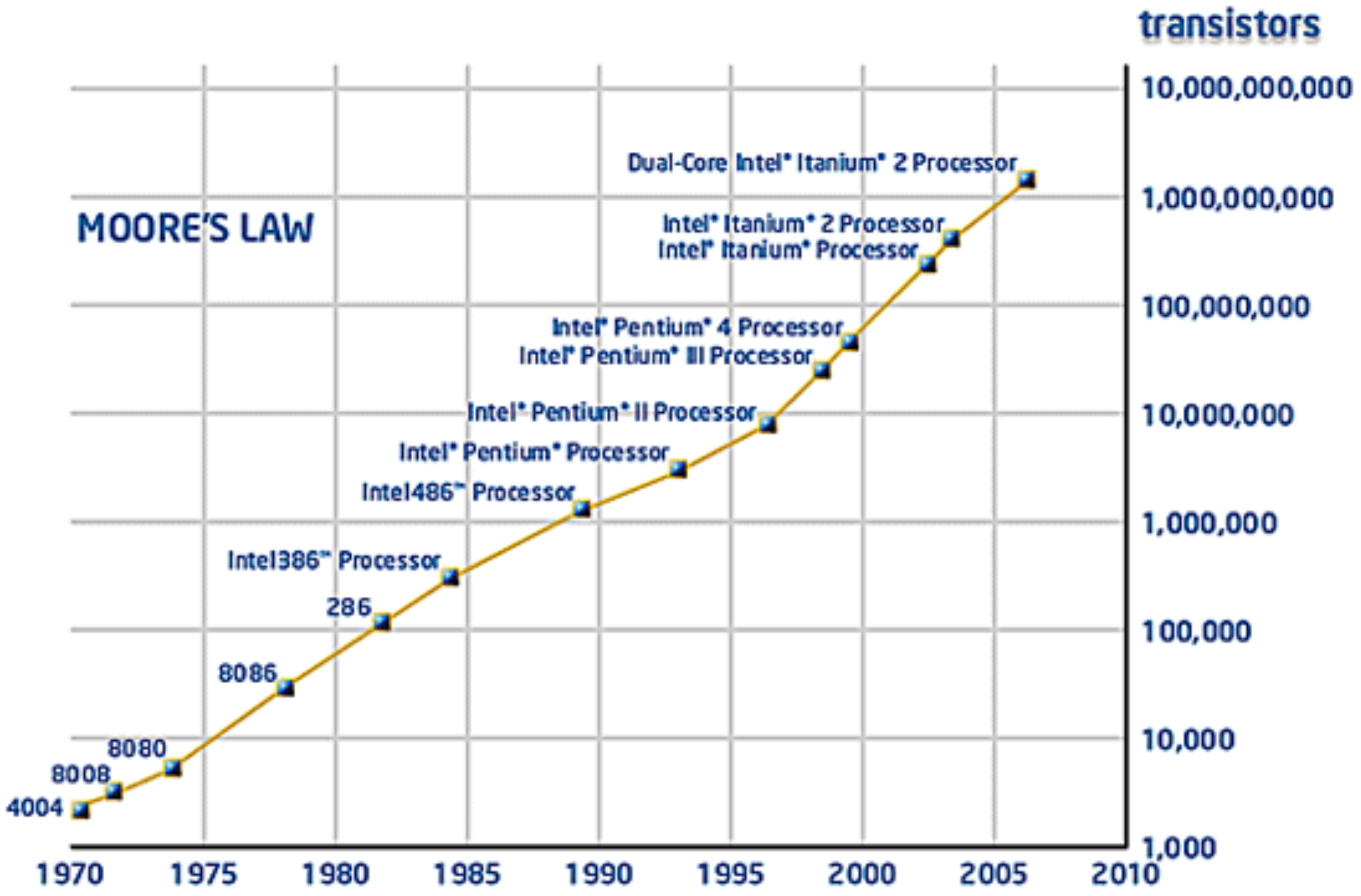
A Very Brief History of Computing

- 1645 Blaise Pascal's Calculating Machine
- 1822 Charles Babbage
  - Difference Engine
  - Analytic Engine: Augusta Ada King first programmer
- < 1946 Eckert & Mauchly
  - ENIAC (Electronic Numerical Integrator and Calculator)
- 1947 John von Neumannn
  - Proposed the Stored Program Computer
  - Virtually all current computers are "von Neumann" machines
- 1949 Maurice Wilkes
  - EDSAC (Electronic Delay Storage Automatic Calculator)

Some Commercial Computers

Year	Name	Size (cu. ft.)	Adds/sec	Price
1951	UNIVAC I	1000	1,900	$1,000,000
1964	IBM S/360 Model 50	60	500,000	$1,000,000
1965	PDP-8	8	330,000	$16,000
1976	Cray-1	58	166 million	$4,000,000
1981	IBM PC	desktop	240,000	$3,000
1991	HP 9000 / model 750	desktop	50 million	$7,400
1996	PC with Intel PentiumPro	desktop	400 million	$4,400
2002	PC with Intel Pentium4	desktop/laptop/ rack	4 billion	$1-2K
2002	PC with Intel Itanium2	desktop/rack	~10 billion	$1-2K
2008	Cell processor	PlayStation3	~200 billion	~$350 (eBay)

Microprocessor Trends (for Intel CPUs)



What Do Computer Architects Do?

- Full disclosure: I’m a computer architect
- Design new microarchitectures
  - Very occasionally, we design new architectures
- Design computers that meet ever-changing needs and challenges
  - Tailored to new applications (e.g., image/video processing)
  - Amenable to new technologies (e.g., faster and more plentiful transistors)
  - More reliable, more secure, use less power, etc.
- Computer architecture is engineering, not science
  - There is no one right way to design a computer → this is why there isn’t just one type of computer in the world
  - This does not mean, though, that all computers are equally good

Outline of Introduction

- Administrivia
- What is a computer?
- What is computer architecture?
- Why are there different types of computers?
- How do we tell computers what to do?

We Use High Level Languages

High Level Language Program

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

- There are many high level languages (HLLs)
  - Java, C, C++, C#, Fortran, Basic, Pascal, Lisp, Ada, Matlab, etc.
- HLLs tend to be English-like languages that are “easy” for programmers to understand
- In this class, we’ll focus on C/C++ as our running example for HLL code. Why?
  - C/C++ has pointers
  - C/C++ has explicit memory allocation/deallocation
  - Java hides these issues (don’t get me started on Matlab)

HLL → Assembly Language

$$\begin{tikzcd}
\boxed{\text{High Level Language Program}} \arrow[d, "\text{Compiler}"] \\
\boxed{\text{Assembly Language Program}}
\end{tikzcd}$$

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw      $15,   0($2)
lw      $16,   4($2)
sw      $16,   0($2)
sw      $15,   4($2)
```

- Every computer architecture has its own assembly language
- Assembly languages tend to be pretty low-level, yet some actual humans still write code in assembly
- But most code is written in HLLs and compiled
  - Compiler is a program that automatically converts HLL to assembly

Assembly Language → Machine Language

$$\begin{tikzcd}
\boxed{\text{High Level Language Program}} \arrow[d, "\text{Compiler}"] \\
\boxed{\text{Assembly Language Program}} \arrow[d, "\text{Assembler}"] \\
\boxed{\text{Machine Language Program}}
\end{tikzcd}$$

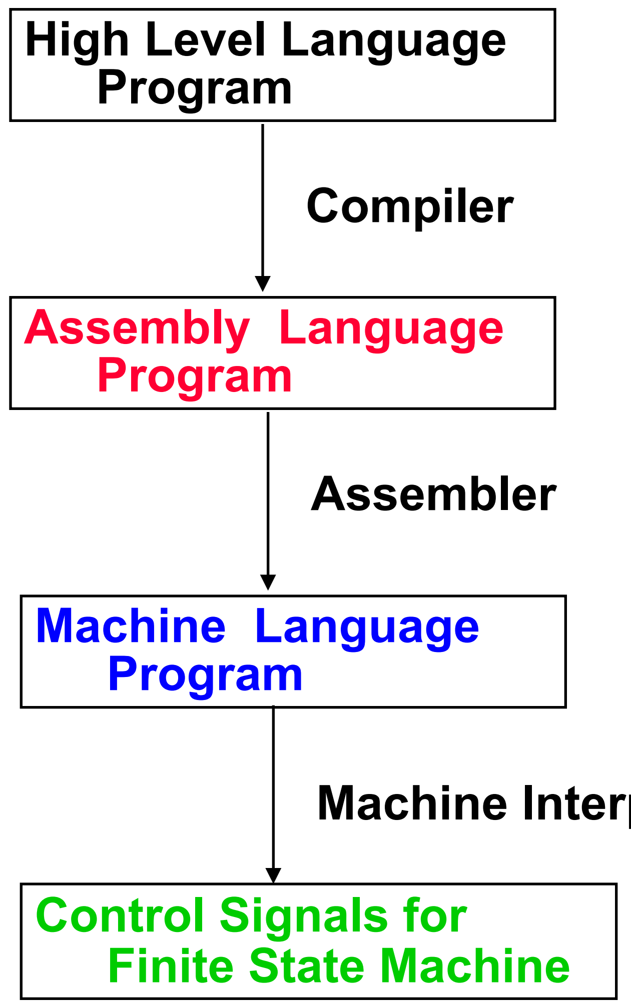
```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw      $15,    0($2)
lw      $16,    4($2)
sw      $16,    0($2)
sw      $15,    4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

- Assembler program automatically converts assembly code into the binary machine language (zeros and ones) that the computer actually executes

Machine Language → Inputs to Digital System


High Level Language Program
Compiler
Assembly Language Program
Assembler
Machine Language Program
Machine Interpretation
Control Signals for Finite State Machine

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw     $15,   0($2)
lw     $16,   4($2)
sw     $16,   0($2)
sw     $15,   4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Transistors turning on and off

Representing High Level Things in Binary

- Computers represent everything in binary
- Instructions are specified in binary
- Instructions must be able to describe
  - Data objects (integers, decimals, characters, etc.)
  - Memory locations
  - Operation types (add, subtract, shift, etc.)

Basic Data Types

Bit: 0, 1

Bit String: sequence of bits of a particular length

- 4 bits is a nibble
- 8 bits is a byte
- 16 bits is a half-word
- 32 bits is a word
- 64 bits is a double-word
- 128 bits is a quad-word

Character:
ASCII 7-bit code

Integers:
2's Complement (32-bit or 64-bit representation)

Floating Point:
Single Precision (32-bit representation)
Double Precision (64-bit representation)
Extended Precision (128-bit representation)

Issues for Binary Representation of Numbers

- There are many ways to represent numbers in binary
  - Binary representations are encodings → many encodings possible
  - What are the issues that we must address?
- Issue #1: Complexity of arithmetic operations
- Issue #2: Negative numbers
- Issue #3: Maximum representable number
- Choose representation that makes these issues easy for machine, even if it's not easy for humans (i.e., ECE 152 students)
  - Why? Machine has to do all the work!

Review from ECE 52: 2's Complement Integers

- Use large positives to represent negatives
- $(-x) = 2^n - x$
- This is 1's complement + 1
- $(-x) = 2^n - 1 - x + 1$
- So, just invert bits and add 1

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

6-bit examples:

$010110_2 = 22_{10}$; $101010_2 = -22_{10}$

$1_{10} = 000001_2$; $-1_{10} = 111111_2$

$0_{10} = 000000_2$; $-0_{10} = 000000_2$ → good!

Pros and Cons of 2's Complement

- Advantages:
  - Only one representation for 0 (unlike 1's comp): 0 = 000000
  - Addition algorithm is much easier than with sign and magnitude
    - Independent of sign bits
- Disadvantage:
  - One more negative number than positive
  - Example: 6-bit 2's complement number
    100000_2 = -32_{10}; but 32_{10} could not be represented

All modern computers use 2's complement for integers

2's Complement Precision Extension

- Most computers today support 32-bit (int) or 64-bit integers
  - Specify 64-bit using gcc C compiler with long long
- To extend precision, use sign bit extension
  - Integer precision is number of bits used to represent a number

Examples

14_{10} = 001110_2 in 6-bit representation.

14_{10} = 000000001110_2 in 12-bit representation

-14_{10} = 110010_2 in 6-bit representation

-14_{10} = 111111110010_2 in 12-bit representation.

What About Non-integer Numbers?

- There are infinitely many real numbers between two integers
- Many important numbers are real
  - Speed of light ~= 3×10^8
  - Pi = 3.1415…
- Fixed number of bits limits range of integers
  - Can't represent some important numbers
- Humans use Scientific Notation
  - 1.3×10^4
- We'll revisit how computers represent floating point numbers later in the semester

What About Strings?

- Many important things stored as strings…
  - E.g., your name
- How should we store strings?

ASCII Character Representation

Oct.	Char														
000	nul	001	soh	002	stx	003	etx	004	eot	005	enq	006	ack	007	bel
010	bs	011	ht	012	nl	013	vt	014	np	015	cr	016	so	017	si
020	dle	021	dc1	022	dc2	023	dc3	024	dc4	025	nak	026	syn	027	etb
030	can	031	em	032	sub	033	esc	034	fs	035	gs	036	rs	037	us
040	sp	041	!	042	"	043	#	044	$	045	%	046	&	047	'
050	(	051	)	052	*	053	+	054	,	055	-	056	.	057	/
060	0	061	1	062	2	063	3	064	4	065	5	066	6	067	7
070	8	071	9	072	:	073	;	074	<	075	=	076	>	077	?
100	@	101	A	102	B	103	C	104	D	105	E	106	F	107	G
110	H	111	I	112	J	113	K	114	L	115	M	116	N	117	O
120	P	121	Q	122	R	123	S	124	T	125	U	126	V	127	W
130	X	131	Y	132	Z	133	[	134	\	135	]	136	^	137	_
140	`	141	a	142	b	143	c	144	d	145	e	146	f	147	g
150	h	151	i	152	j	153	k	154	l	155	m	156	n	157	o
160	p	161	q	162	r	163	s	164	t	165	u	166	v	167	w
170	x	171	y	172	z	173	{	174	|	175	}	176	~	177	del

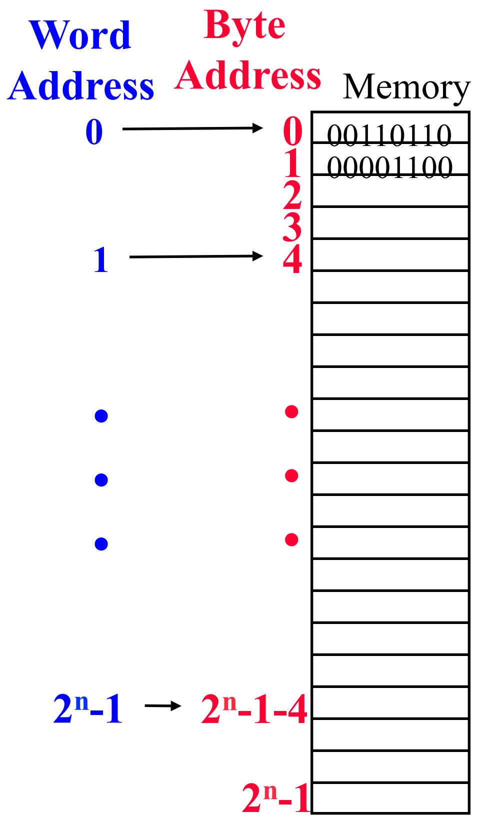
- Each character represented by 7-bit ASCII code.
  - Packed into 8-bits

Computer Memory

- What is computer memory?
- What does it “look like” to the program?
- How do we find things in computer memory?

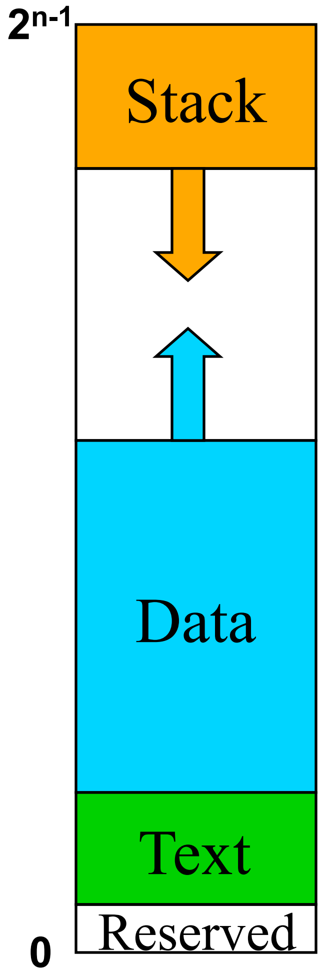
A Program's View of Memory

- What is memory? a bunch of bits
- Looks like a large linear array
- Find things by indexing into array
  - Index is unsigned integer
- Most computers support byte (8-bit) addressing
  - Each byte has a unique address (location)
  - Byte of data at address 0x100 and 0x101
  - Word of data at address 0x100 and 0x104
- 32-bit v.s. 64-bit addresses
  - We will assume 32-bit for rest of course, unless otherwise stated
  - How many bytes can we address with 32 bits? With 64 bits?



Memory Partitions

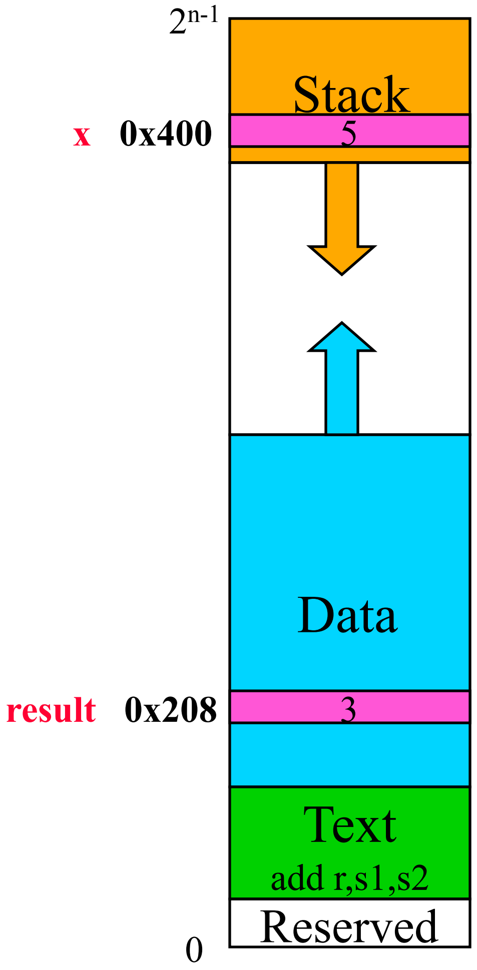
- Text for instructions
  - add dest, src1, src2
  - mem[dest] = mem[src1] + mem[src2]
- Data
  - Static (constants, globals)
  - Dynamic (heap, new allocated)
  - Grows upward
- Stack
  - Local variables
  - Grows down from top of memory
- Variables are names for memory locations
  - int x; // x is a location in memory



Typical Address Space

A Simple Program's Memory Layout

```
...
int result; // global variable
main()
{
   int x; // allocated on stack
   ...
   result = x + result;
   ...
}
```



Pointers

- A pointer is a memory location that contains the address of another memory location
- “address of” operator & in C/C++
    - Don’t confuse with bitwise AND operator (which is && operator)

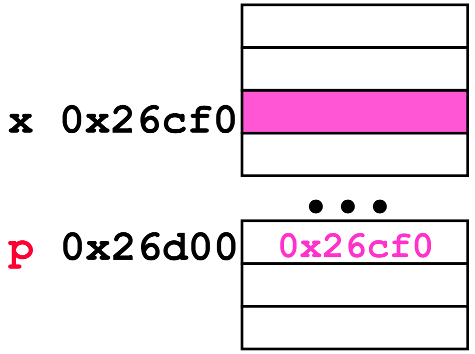
Given

```
   int x; int* p;
   p = &x;  // p points to x (i.e., p is the address of x)
```

Then

```
   *p = 2;  and x = 2; produce the same result
```

On 32-bit machine, p is 32-bits



Arrays

- In C++: allocate using array form of new

  ```
  int* a = new int[100];
  double* b = new double[300];
  ```

- `new[]` returns a pointer to a block of memory
  - How big? Where?
- Size of chunk can be set at runtime
- `delete[] a;` // storage returned
- In C:

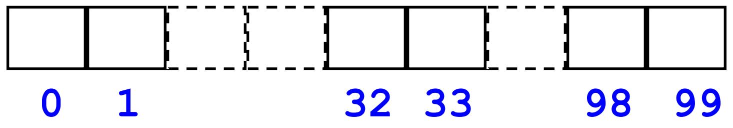
  ```
  int* ptr = malloc(nbytes);
  free(ptr);
  ```

Address Calculation

- If x is a pointer, what is x+33?
- A pointer, but where?
  - What does calculation depend on?
- Result of adding an int to a pointer depends on size of object pointed to
- Result of subtracting two pointers is an int

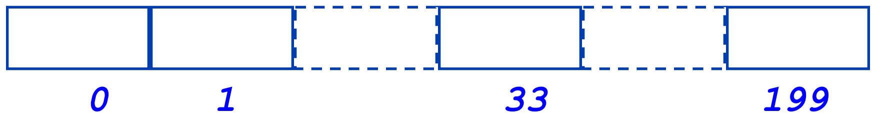
```
(d + 3) - d = _______
```

```
int *a = new int[100]
```



```
a[33] is the same as *(a+33)
if a is 0x00a0, then a+1 is
0x00a4, a+2 is 0x00a8
(decimal 160, 164, 168)
```

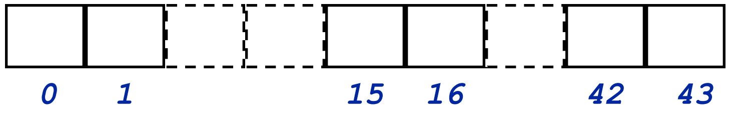
```
double * d = new double[200];
```



```
*(d+33) is the same as d[33]
if d is 0x00b0, then d+1 is
0x00b8, d+2 is 0x00c0
(decimal 176, 184, 192)
```

More Pointer Arithmetic

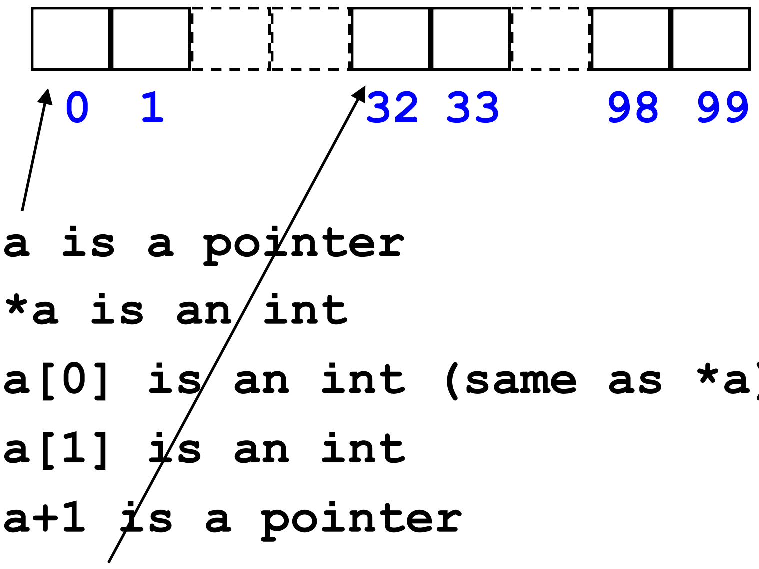
- address one past the end of an array is ok for pointer comparison only
- what's at `*(begin+44)` ?
- what does `begin++` mean?
- how are pointers compared using < and using == ?
- what is value of `end - begin`?



```
char* a = new char[44];
char* begin = a;
char* end = a + 44;

while (begin < end)
{
   *begin = 'z';
   begin++;
}
```

More Pointers & Arrays

```
int* a = new int[100];
```



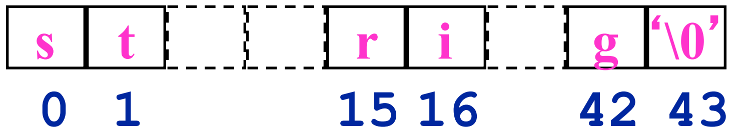
```
a is a pointer
*a is an int
a[0] is an int (same as *a)
a[1] is an int
a+1 is a pointer
a+32 is a pointer
*(a+1) is an int (same as a[1])
*(a+99) is an int
*(a+100) is trouble
```

Array Example

```
main()
{
   int* a = new int[100];
   int* p = a;
   int k;

   for (k = 0; k < 100; k++)
     {
       *p = k;
       p++;
     }

   cout << "entry 3 = " << a[3] << endl;

}
```

Strings as Arrays

- A string is an array of characters with ‘\0’ at the end
- Each element is one byte (in ASCII code)
- ‘\0’ is null (ASCII code 0)



Summary: Representing High Level in Computer

- Everything must be represented in binary!
- Computer memory is linear array of bytes
- Pointer is memory location that contains address of another memory location
- We'll visit these topics again throughout semester

Outline of Introduction

- Administrivia
- What is a computer?
- What is computer architecture?
- Why are there different types of computers?
- How do we tell computers what to do?

What You Will Learn In This Course

- The basic operation of a computer
  - Primitive operations (instructions)
  - Computer arithmetic
  - Instruction sequencing and processing
  - Memory
  - Input/output
  - Doing all of the above, just faster!
- Understand the relationship between abstractions
  - Interface design
  - High-level program to control signals (SW → HW)

Course Outline

- Introduction to Computer Architecture
- Instruction Sets & Assembly Programming (next!)
- Central Processing Unit (CPU)
- Pipelined Processors
- Memory Hierarchy
- I/O Devices and Networks
- Multicore Processors
- Performance Analysis & Advanced Topics (if time permits)

The Even Bigger Picture

- ECE 52: Digital systems
- ECE 152: Basic computers
    - Finish 1 instruction every 1 very-long clock cycle
    - Finish 1 instruction every 1 short cycle (using pipelining)
- ECE 552: High-performance computers (plus more!)
    - Finish ~3-6 instructions every very-short cycle
    - Multiple cores each finish ~3-6 instructions every very-short cycle
    - Out-of-order instruction execution, power-efficiency, reliability, security, etc.
- ECE 559: Highly parallel computers, advanced topics
- ECE 554: Fault tolerant computers
- ECE 590: Energy-efficient computers