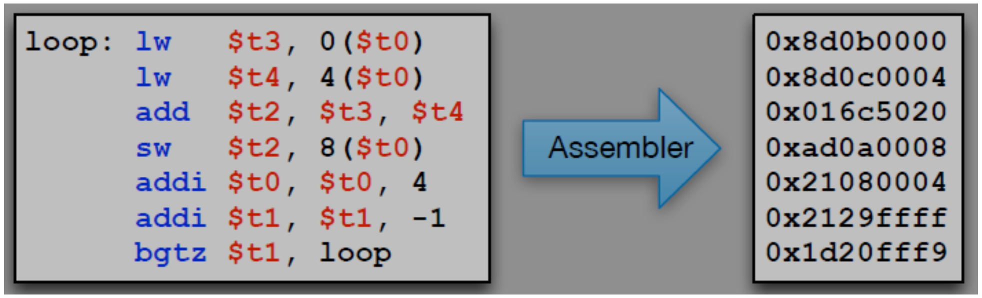
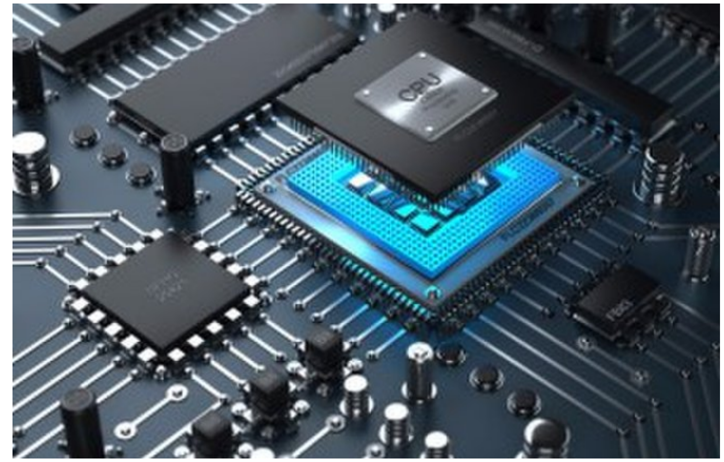




Welcome to “Computer Organization and Design Logic”

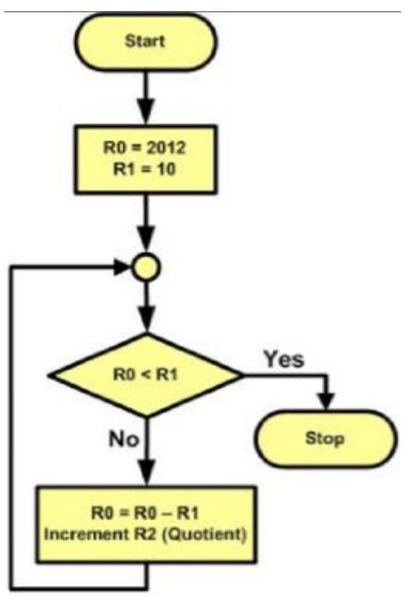
CS 64: Computer Organization and Design Logic

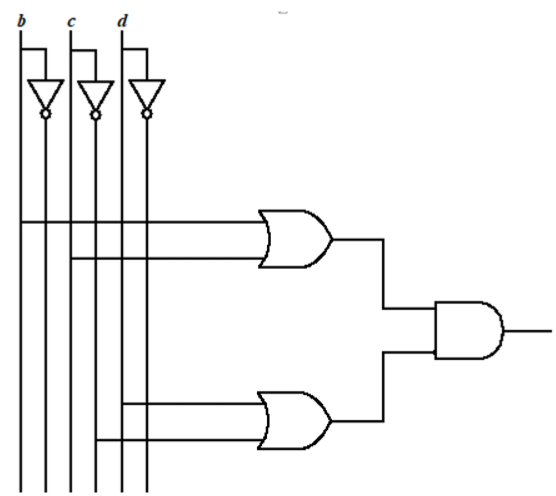
Lecture #1

Winter 2020

Ziad Matni, Ph.D.

Dept. of Computer Science, UCSB





A Word About Registration for CS64

FOR THOSE OF YOU NOT YET REGISTERED:

- This class is **FULL** ***and*** there is a **WAITLIST**

```
if (want2add) && (on_waitlist)
{
    SeeMeAfterLecture(True);
}
else
{
    YoureGonnaHaveABadTime(True);
}
```

Your Instructor

Your instructor: **Ziad Matni, Ph.D.** *(zee-ahd mat-knee)*

Email: ***zmatni@ucsb.edu***

(please put CS64 at the start of the subject header!!)

My office hours:

Mondays 10:00 AM – 11:30 AM, at **SMSS 4409**

(or by appointment)

Your TAs

All labs will take place in **PHELPS 3525**
All TA office hours will take place in **Trailer 936**

Teaching Assistant	**Office Hours**
Kunlong Liu	tbd
Michael Christensen	tbd
Shu Yang (Reader)	N/A (none)

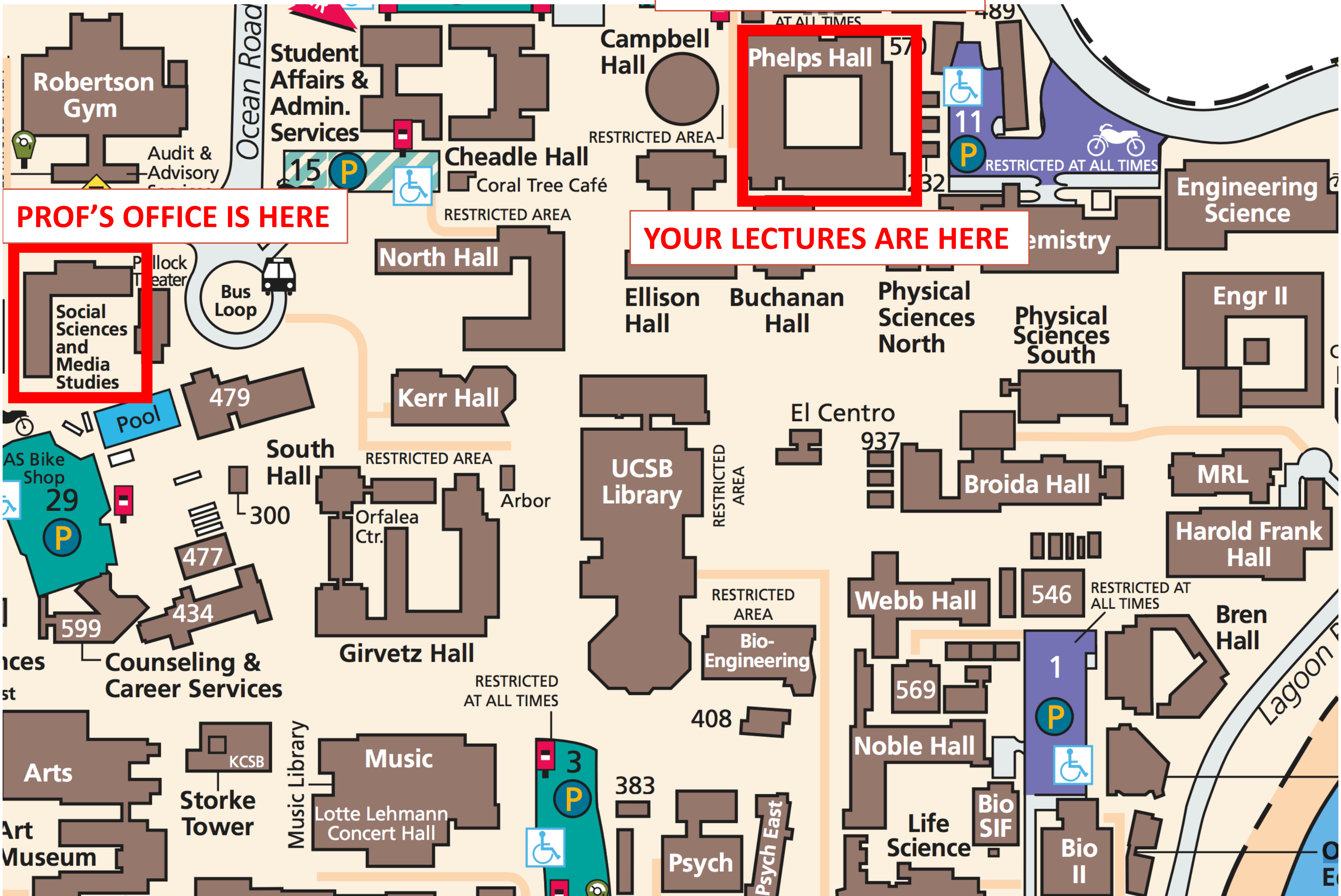
Your FIRST lab is THIS THURSDAY! (posted on Wednesday)

Labs are due on TUESDAYS!

YOUR LABS ARE HERE

PROF'S OFFICE IS HERE

YOUR LECTURES ARE HERE



You!

With a show of hands, tell me… how many of you…

A. Are Freshmen? Sophomores? Juniors? Seniors?
B. Are CS majors? Other?
C. Know: scripting language (PERL, csh, bash) programming?
D. Have NOT used a Linux or UNIX system before?
E. Have *seen* actual "assembly code" before?
F. Programmed in assembly before?
G. Written/seen code for *firmware*?
H. Understand basic binary logic (i.e. OR, AND, NOT)?
I. Designed any digital circuit before?

This Class

- This is an **introductory** course in **low-level programming** and **computer hardware**.
  - Two separate but very intertwined areas

- What happens between your C/C++/Java/Python command:

  int a = 3, b = 4, c = a+b;

  and the actual ***"digital mechanisms***" in the CPU
  that process these "simple" (and other "no-so-simple")
  commands?

- This class can sometimes move *fast* – so please prepare accordingly.

Lecture Etiquette!

- I need you to be INVOLVED and ACTIVE!

- **Phones OFF!** and laptops/tablets are for **NOTES** only
  - No social media use, please

- To succeed in this class, take <u>thorough</u> notes
  - I'll provide my slides, but not class notes
  - Studies show that ***written*** notes are ***superior to*** typed ones!

Main Class Website

Main Website:

https://ucsb-cs64.github.io/w20/

On there, I will keep:

- Latest syllabus
- Class assignments
- Lecture slides (after I've given them)
- Exam prep material
- Important handouts and articles

Other Class Websites/Tools

Piazza

https://piazza.com/ucsb/winter2020/cs64

On there, we will:

- Engage in Q & A and online discussions
- Make important announcements
- Have (maybe) Interesting handouts and articles

Register Today!

Gradescope

https://www.gradescope.com

On there:

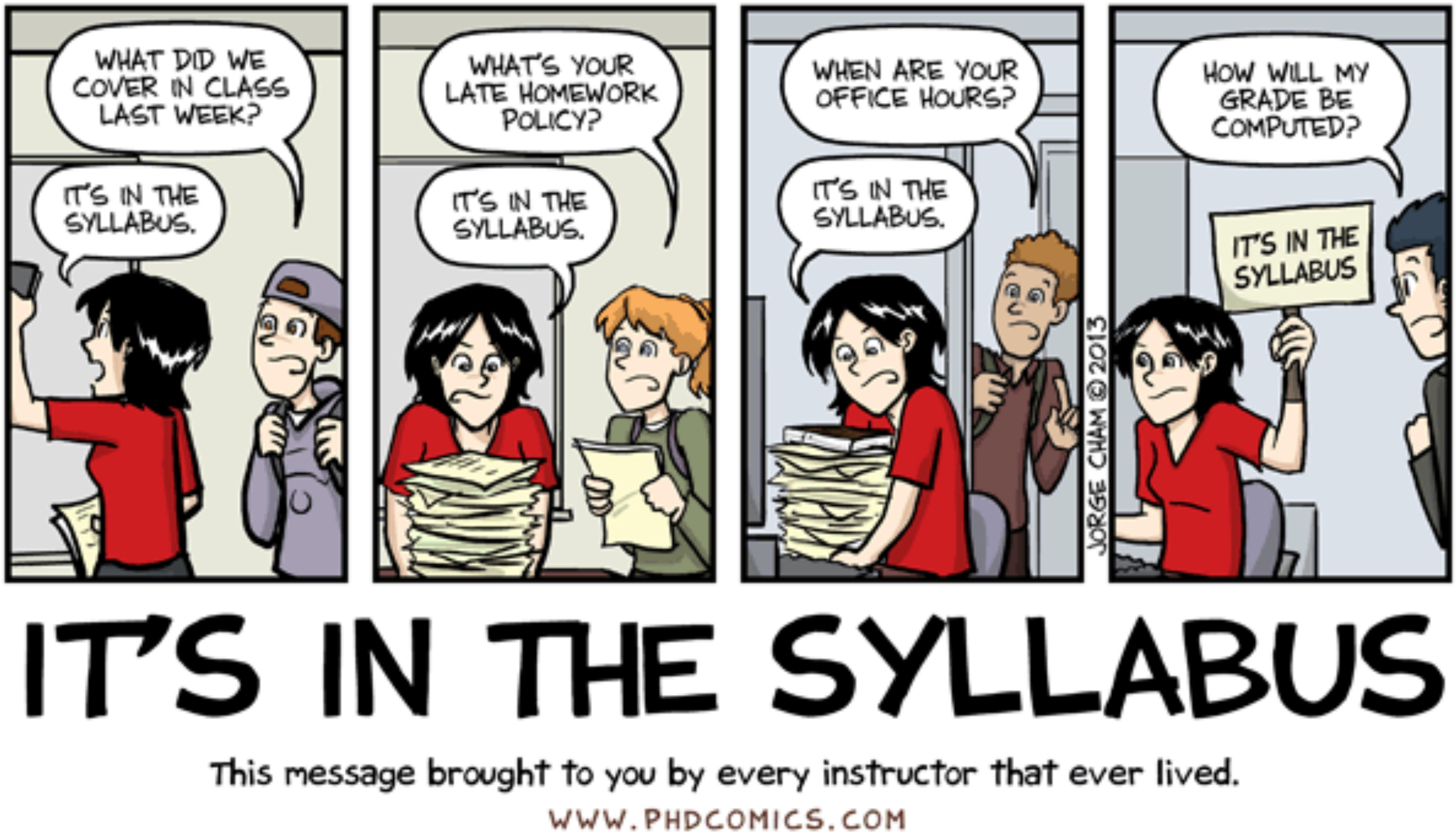
- You will submit all your assignments, typically as **PDF**s
- We will post your assignment grades

GauchoSpace

https://gauchospace.ucsb.edu

- This is where we will post your other grades

Just in Case…


WHAT DID WE COVER IN CLASS LAST WEEK?
IT'S IN THE SYLLABUS.
WHAT'S YOUR LATE HOMEWORK POLICY?
IT'S IN THE SYLLABUS.
WHEN ARE YOUR OFFICE HOURS?
IT'S IN THE SYLLABUS.
JORGE CHAM ©2013
HOW WILL MY GRADE BE COMPUTED?
IT'S IN THE SYLLABUS
IT'S IN THE SYLLABUS
This message brought to you by every instructor that ever lived.
WWW.PHDCOMICS.COM

So… let’s take a look at that syllabus…

Electronic version found on Main Website *or* at:

http://cs.ucsb.edu/~zmatni/syllabi/CS64W20_syllabus.pdf

- Instructor & T.A.s’ vital info
- Class websites’ info
- Textbook
- Class organization and expected conduct
- Grading info
- Lectures, quizzes & participation
- Labs & assignments
- My policies (absences, make ups, my copyrights, academic integrity)
- Class schedule

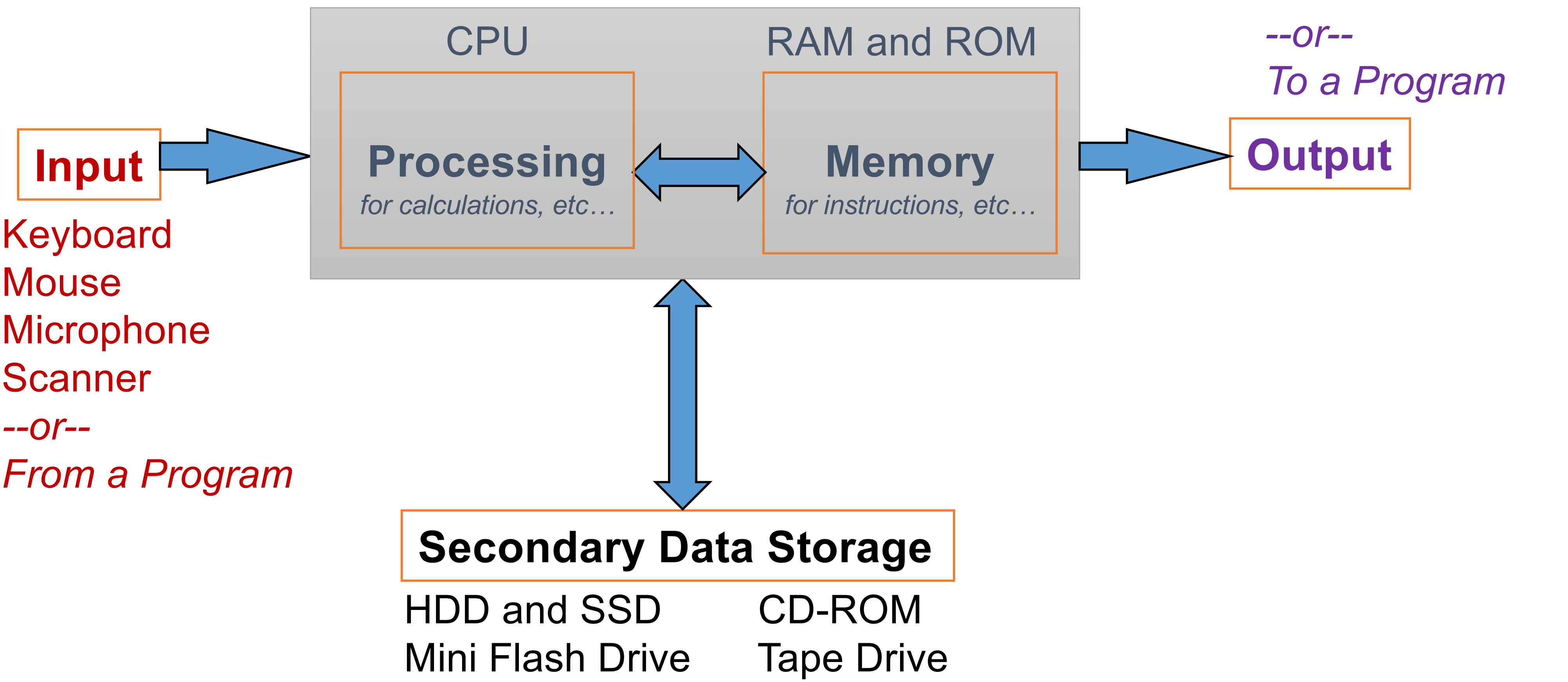
You are responsible for reading it
(yes, the whole thing!)

A Simplified View of Modern Computer Architecture

a.k.a von Neumann Architecture

The 5 Main Components of a Computer:

1. Processor
2. Memory
3. Input
4. Output
5. Secondary Data Storage



Computer Memory

- Usually organized in two parts:
  - Address: ***Where*** can I find my data?
  - Data (payload): ***What*** is my data?

- The smallest representation of the data
  - A binary *bit* ("0"s and "1"s)
  - A common collection of bits is a *byte*
    - 8 bits = 1 byte
  - What is a *nibble*?
    - 4 bits = 1 nibble – not used as often…
  - **What is the minimum number of bits needed to convey an alphanumeric character? And WHY?**

What is the Most Basic Form of Computer Language?

- Binary *a.k.a* Base-2
- Expressing data AND instructions in either “1” or “0”
  - So,

01010101 01000011 01010011 01000010 00100001 00100001

could mean a *CPU instruction* to “calculate 2 + 3”

Or it could mean an *integer number* (856,783,663,333)

Or it could mean a *string of 6 ASCII characters* (“UCSB!!”)

Or other things…!?!

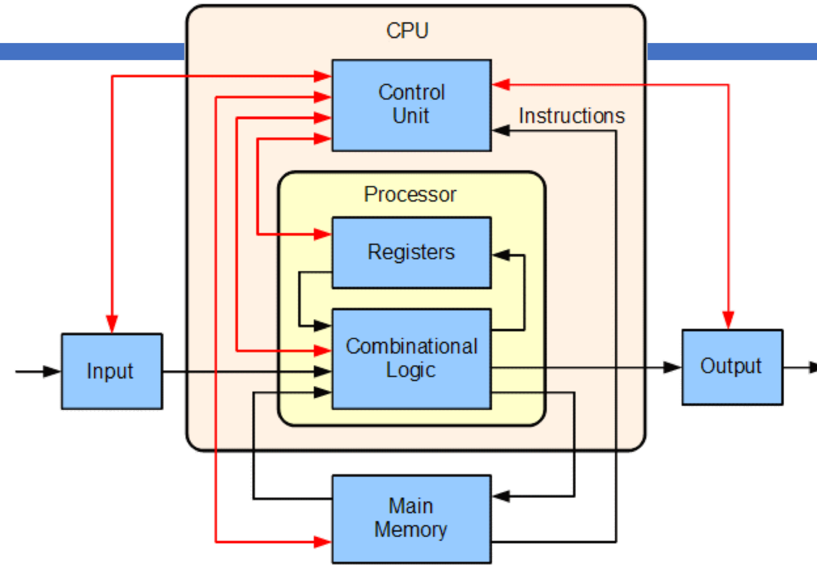
So… Like…
What Processes Stuff In A Computer?

- The Central Processing Unit (CPU)
  - Executes program instructions

- Typical capabilities of CPU include:
  - Add
  - Subtract
  - Multiply
  - Divide
  - Move data from location to location

You can do just about anything with a computer with just these simple instructions!

Parts of the CPU



The CPU is made up of 2 main parts:

- The Arithmetic Logic Unit (**ALU**)
  and other related blocks, all together called the Datapath
- The Control Unit (**CU**)
- The ALU does the calculations in binary using "registers" (small RAM) and logic circuits
- The CU handles breaking down instructions into control codes for the ALU and memory

Image from wikimedia.org

The CPU’s Fetch-Execute Cycle

- **Fetch** the next instruction
- **Decode** the instruction
- **Get data** if needed
- **Execute** the instruction
- ***Why is it a cycle???***

This is what happens inside a computer interacting with a program at the “lowest” level

Pipelining (Parallelism) in CPUs

- Pipelining is a fundamental design in CPUs
- Allows multiple instructions to go on at once
  - a.k.a instruction-level parallelism

Basic five-stage pipeline

Instr. No. \ Clock cycle	1	2	3	4	5	6	7
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX

(IF = Instruction Fetch, ID = Instruction Decode, EX = Execute, MEM = Memory access, WB = Register write back).

Computer Languages and the F-E Cycle

- Instructions get executed in the CPU in machine language (i.e. all in “1”s and “0”s)

- Even *small* instructions, like

  “add 2 to 3 then multiply by 4”,

  need *multiple* cycles of the CPU to get fully executed

- But **THAT’S OK!** Because, typically, CPUs can run *many millions* of instructions per second

Computer Languages and the F-E Cycle

- But **THAT'S OK!** Because, typically,
  CPUs can run *many millions* of instructions per second

- In *low-level languages* (like assembly or machine lang.) you need to spell those parts of the cycles one at a time

- In *high-level languages* (like C, Python, Java, etc…) you don't
  - 1 HLL statement, like "*x = c*(a + b)*" is enough to get the job done
  - This would translate into multiple statements in LLLs
  - **What translates HLL to LLL?**

Machine vs. Assembly Language

- **Machine language (ML)** is the actual 1s and 0s

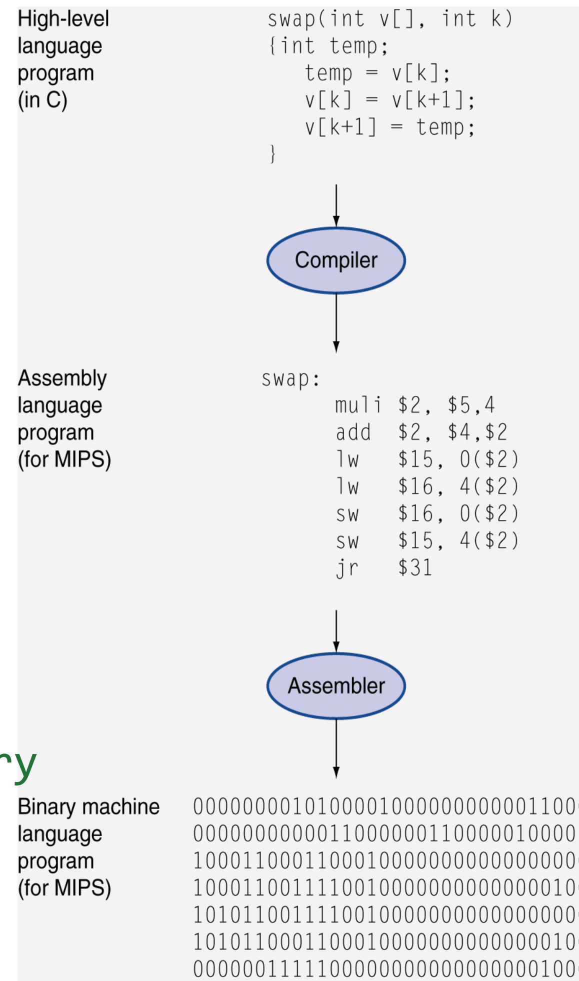
Example:

```
10111101110111100000101010101000
```

- **Assembly language** is one step above ML
  - Instructions are given <u>**mnemonic codes**</u> but still displayed one step at a time
  - Advantage? Better human readability

Example:

```
lw   $t0, 4($sp)    # fetch N from someplace in memory
add  $t0, $t0, $t0   # add N to itself
                     # and store the result in N
```



Why Can Programs Sometimes be Slow?

- Easy answer: they're processing a lot of stuff…
- But, isn't just as "simple" as

  1. getting an instruction,
  2. finding the value in memory,
  3. and doing stuff to it???

  - Yes… except for the "simple" part…

- ***Ordering*** the instructions <u>matters</u>

  Where in memory the value is <u>matters</u>

  How instructions get "broken down" <u>matters</u>

  What order these get "pipelined" <u>matters</u>

The Point…

- If you really want ***performance***, you need to know how the "magic" works

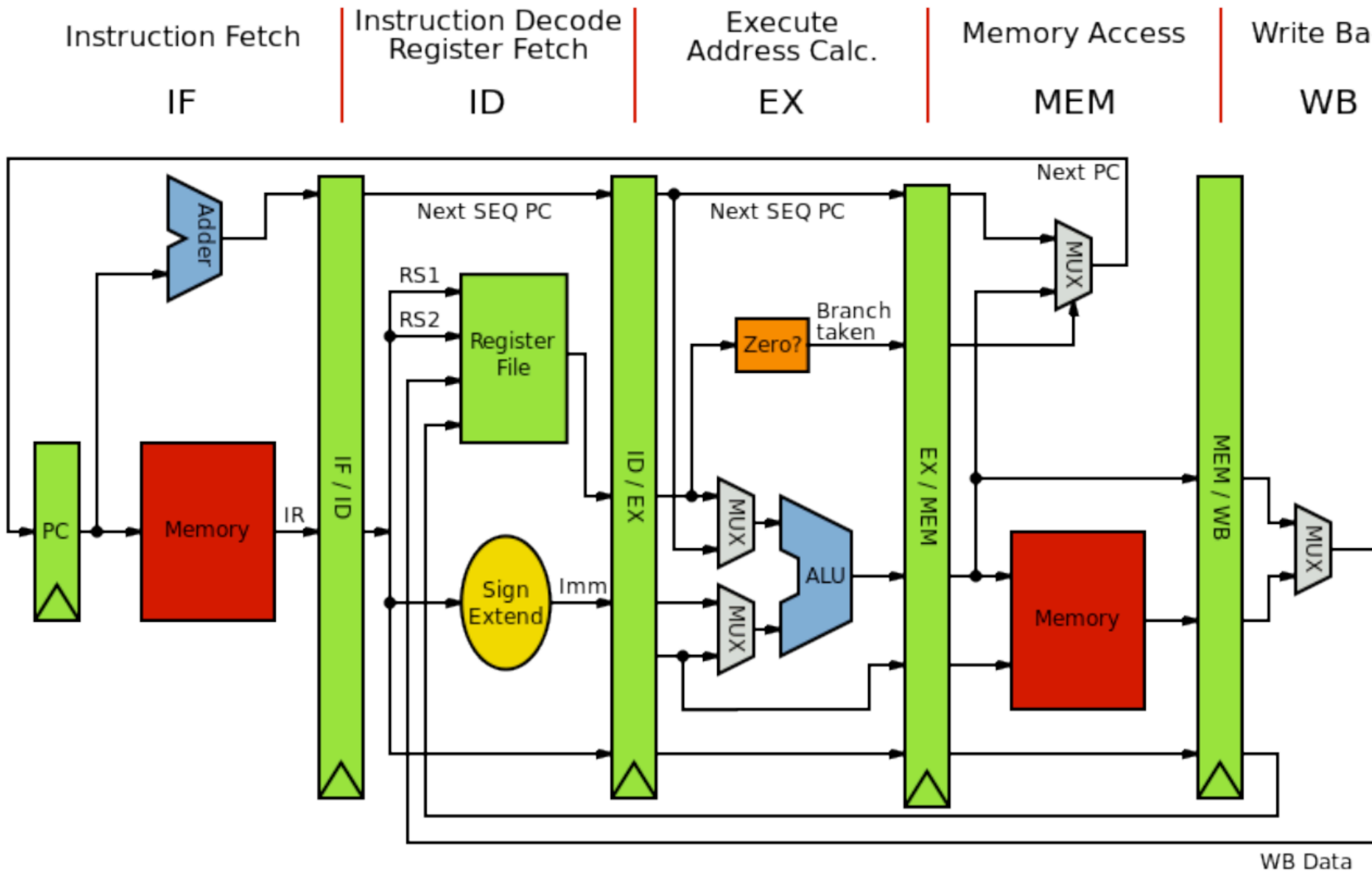
- If you want to write a *naive compiler* (CS 160), you need to know some low-level details of how the CPU does stuff

- If you want to write a *fast compiler*, you need to know tons of low-level details

So Why Digital Design?

- Because that's where the "magic" happens
- Logical decisions are made with 1s and 0s
- Physically (*engineering-ly?*), this comes from electrical currents switching one way or the other & also how semiconducting material work, etc…
- But we don't have to worry about the physics part in this class…

Digital Design of a CPU (Showing Pipelining)



Digital Design in this Course

- We will not go into “deep” dives with digital design in this course
  - For that, check out CS 154 (Computer Architecture) and also courses in ECE

- We will, however, delve deep enough to understand the ***fundamental*** workings of digital circuits and how they are used for ***computing purposes***.

YOUR TO-DOs

- Get accounts on Piazza and Gradescope
- Do your reading for next class
  - Check the syllabus
- Start on Assignment #1 for lab
  - I'll put it up on our main website this Wednesday
  - Meet up in the lab this Thursday
  - Do the lab assignment: setting up CSIL + exercises
  - You have to submit it as a **PDF** using ***Gradescope***
  - Due on **Tuesday, 1/14, by 11:59:59 PM**

</LECTURE>