CS/COE 0447

Course Goals and Introduction to Assembly

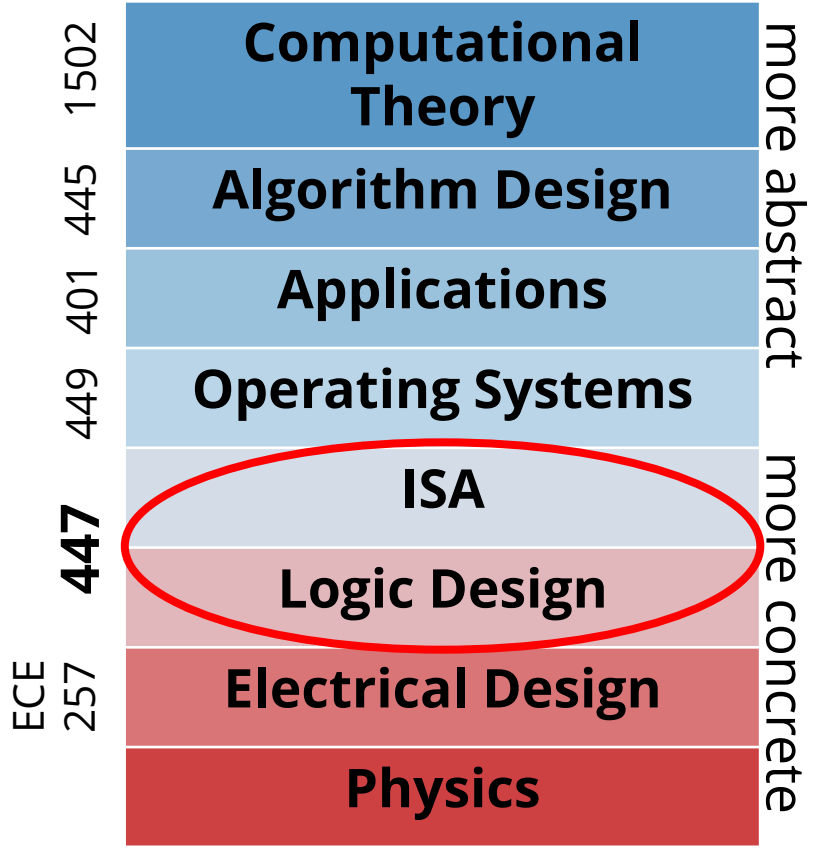
wilkie (with content borrowed from:
Jarrett Billingsley
Dr. Bruce Childers)

Goals

- The role of any introductory CS course is to *demystify* computation.
- In the case of CS0447, we want to examine *the processor* and how it runs programs. (CS0449 then looks at *the programs*.)
- Topics:
  - Data representation
  - Memory organization
  - How programs are executed
  - Program conventions (call stack, function calls, system calls)
- Skills:
  - Understanding common bases (binary, hex, etc)
  - Logical operations
  - Reading logic diagrams
  - Assembly language (reading, writing, debugging!)

Where does CS0447 fit?

- The “hardware-software” interface.
- Where CS and EE overlap.
- Each layer is affected by the layer above and below.
- Often the higher layers are *abstractions* of the lower.
- ISA: **Instruction Set Architecture**
  - A programmer’s interface to the computer hardware.
- Logic Design: (CPU design, etc)
  - “how we make rocks do stuff”
  - Processors: do lots of things with math.



Goals

- Learning Assembly!

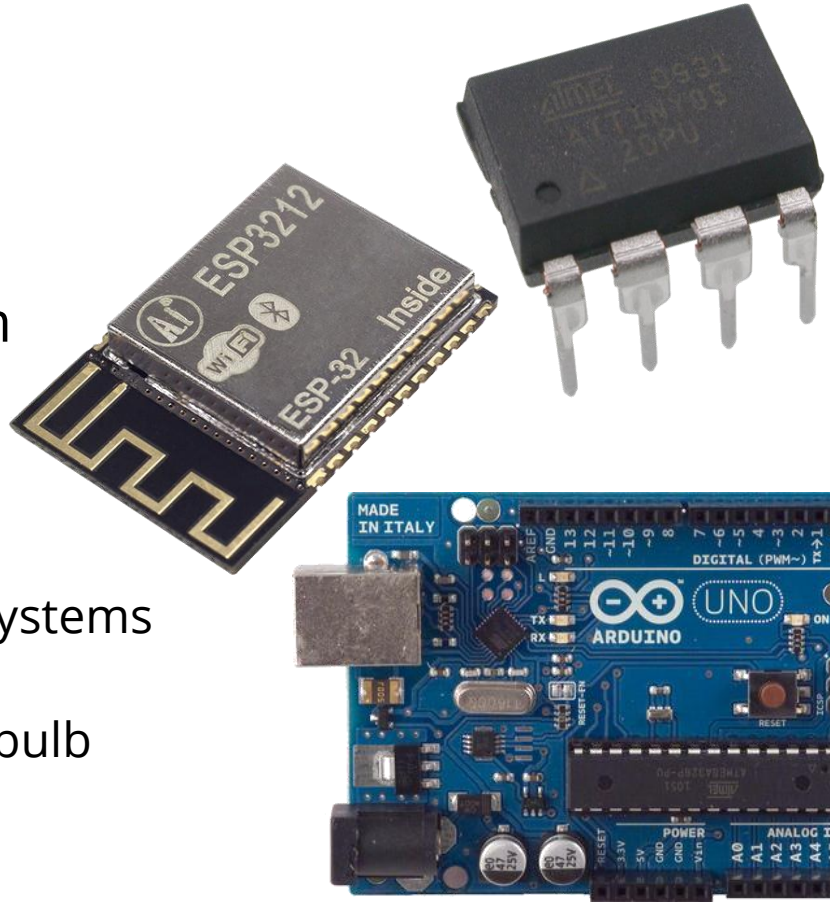
- “Unlearning” Java
  - Java is at higher a level, and thus abstracts a lot.
  - We now get to learn what we’ve taken for granted!

- “Learning assembly is like a car mechanic learning how an engine runs”
  - A “normal” person doesn’t need to know this…
  - But we’re not normal, are we…

- It’s fun to take things apart sometimes… so, let’s take apart a computer.
  - And then put it back together again.

Classes of Computer: Embedded

- "**Microcontrollers**"
- 8-bit and 16-bit architectures
  - Still VERY common
- As little as *32 bytes* of memory
- Generally runs a single program
- Low-power, low-cost, small
  - Mass-produced
- More common every day
  - Most computers are embedded systems
  - Internet-of-things
  - A person can now hack your lightbulb
    - "What a stupid future"
    - Unless we get it right, I guess.



Classes of Computer: Consumer

- "**Desktop**" but also "**Mobile**"
- 1-8 cores, 32 or 64-bit architectures
- GB of memory with GB/TB of storage
- **Multitasking** operating systems
- Focused on real-time interactivity (browsers, games PowerPoint, etc)

- Do we live in a "Post-PC" world?

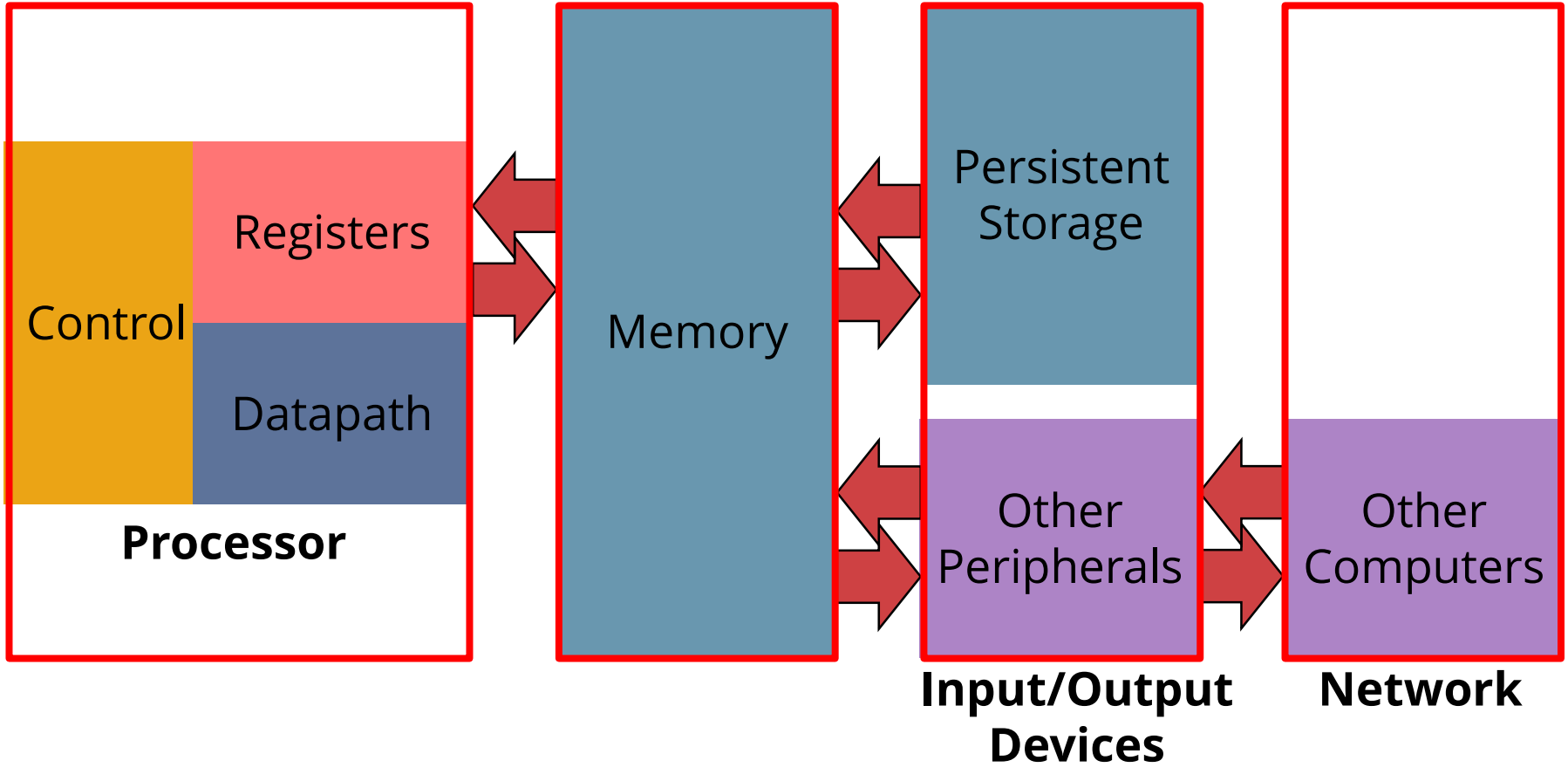
Classes of Computer: Server

- "**Servers**", "**Mainframes**", "**Racks**"
- 12+ cores, multiprocessors
- 128+GB of memory, storage systems
- Very fast networking.
- Focused on real-time data delivery (websites, video game servers, banking…)
- Design goals: redundancy (reliability), hot-swappability, availability.
  - Power and cooling become *huge* concerns
- Goal: 100% uptime!
  - Is that possible?

Classes of Computer: Supercomputer

- Clusters of thousands of CPUs
- Focused on consuming ENORMOUS datasets (non-interactively)
- Science, research, design, and simulation.
  - Stock trading, financial speculation, “cryptocurrencies” …

Computer Organization



What is “Assembly”

- **Assembly**: Human-readable representation of machine code.
- **Machine code**: what a computer *actually* runs.
- The “atoms” that make up a program.
  - CPUs are actually fairly simple in concept.
  - (Yet we have an *entire semester* to fill, hmm)
- Each CPU chooses its own machine code (and therefore its own style of assembly language)
- We are using MIPS
  - Not that common, unfortunately.
  - Yet, often found in surprising places. (Nintendo 64, PS1/2, FPGAs)
  - But, very influential, and most common assembly looks like it.
    - ARM is a similar-ish ISA which is seeing much more usage

What is “Assembly”

- Involves very simple commands.
- This command copies data from one place to another.
  - In spite of being called “move”, ugh!
- Surprise! It’s actually shorthand for a different set of instructions.
  - The processor can be made simpler.
- This command gets transformed into a numerical representation.
  - More on that later!
- The processor then interprets the binary representation.
  - That’s essentially all a computer does!
  - We will design such a processor!

```
mov a0, t0
```

```
-> add a0, t0, zero
```

-> 00000001000000000010000000100000

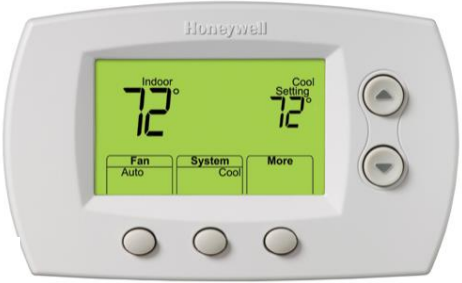
Compute t0+0
Put into “a0”

Is Assembly Useful?

- Short answer: ***YES***
- Assembly is “fast”, so we should use it for everything!
  --- **NO!!!** ---
- No type-checking, no control structures, very few abstractions. --- **Fairly impractical for large things** ---
- Tied to a particular CPU.
  - So, large programs have to be rewritten (usually) to work on new things.
- Yet: good for specialized stuff.
  - Critical paths and “boot” code in Kernels / Operating Systems
  - HPC (simulators, supercomputer stuff)
  - Real-time programs (video games; tho increasingly less / abstracted away)
  - And…

Embedded Systems

- You'd be **amazed** at *how many* consumer devices have CPUs.
- Many are programmed largely/entirely in assembly (or C)

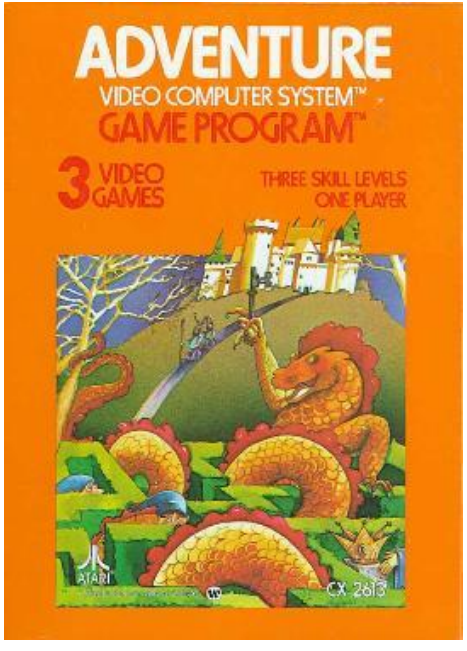


Embedded Systems

Practical Applications of Assembly: Real-time

- Examples of games programmed mostly/entirely in assembly.
  - (Common in the 70s/80s; not at all common today)

Aside: https://all-things-andy-gavin.com/2011/03/12/making-crash-bandicoot-gool-part-9/



1980: 8-bit 1.19 MHz

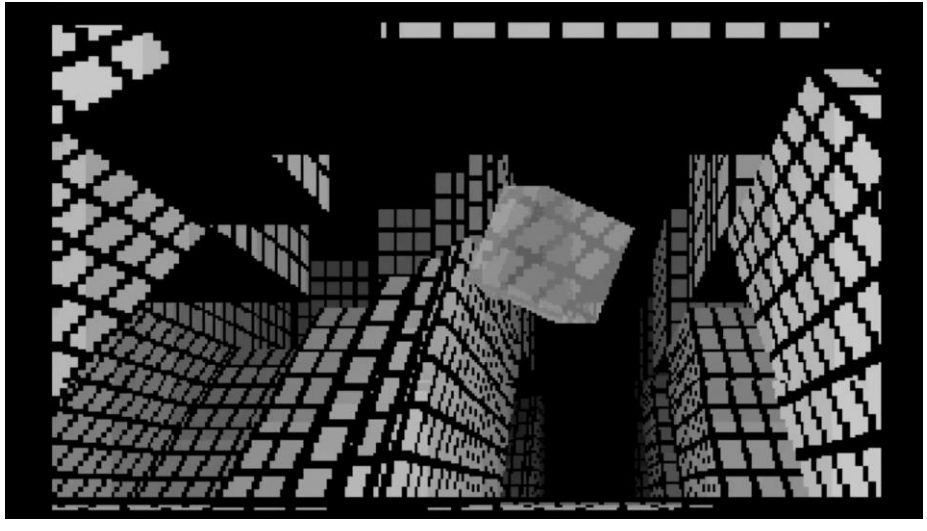
(Generated Assembly … Cheating?)
1994: 32-bit 33 MHz

1999: 32-bit 200 MHz

Practical Applications of Assembly: Art

```
b equ byte
w equ word
d equ dword
  org 100h
  mov al,13h
  int 10h
  mov fs,w[bx]
  mov dx,l
  mov ax,251ch
  int 21h
a:and bp,0ffh
  jnz c
  xor b[cs:l],8
  xor w[f],4a91h
c:mov si,140h
  mov cl,0ffh
e:mov bx,cx
    …
```

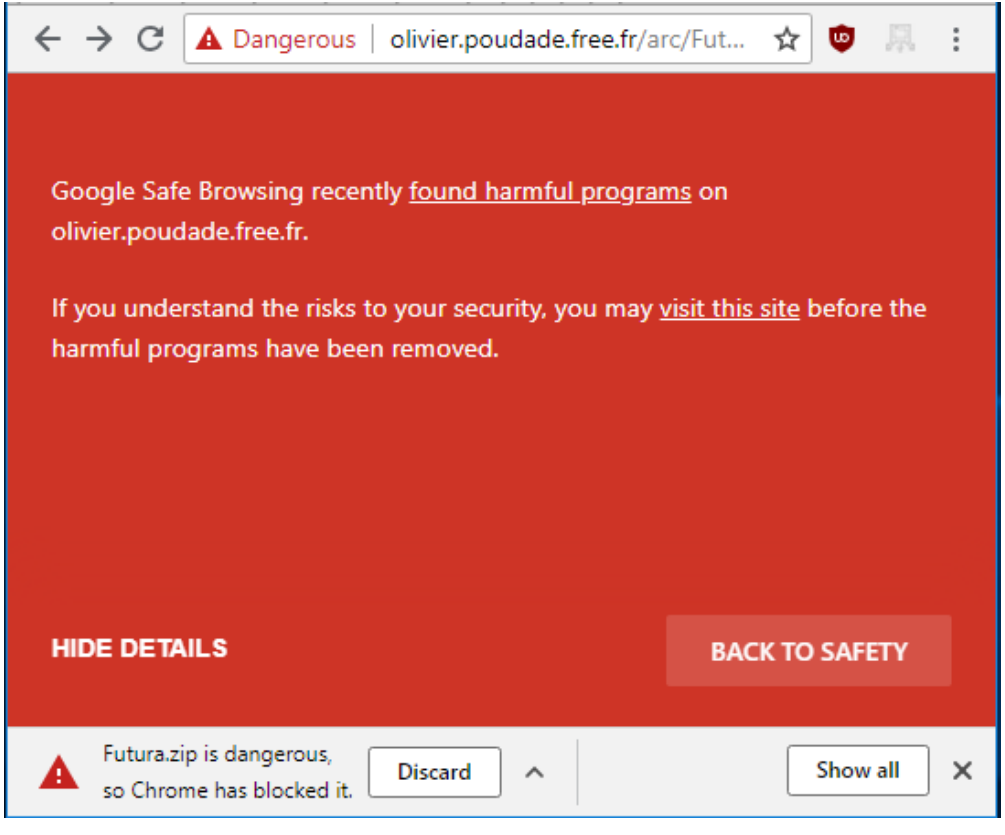
"Futura"
256 Byte Program:



http://www.pouet.net/prod.php?which=66806

Practical Applications of Assembly: Art

- Sometimes, the unusual construction of these programs confuse analysis:



OMG

Google

Chill the freak out...

Practical Applications of Assembly: Modification

- Modifying programs after-the-fact. (Or reverse-engineering them)
- Legal "gray-area," / "confusing-mess" but generally modification/reverse engineering is allowed. Kinda? (Section 1201, US Code 17 § 108, etc)
  - Removing copy protection in order to preserve/backup.
  - **Librarians** and **preservationists** and "**pirates**" alike may all use/view/write assembly for this!
- I patched (the freely distributed) Lost Vikings so it would avoid copy protection and use a different sound configuration (so I could run it in a browser emulator)

```
; patching some bytes
; assembled with: `nasm -fbin -o patch.com patch.asm`

    org  0x100            ; .com files always start 256 by

    mov  ax, 0x00

    mov  dx, msg          ; the address of or message in c
    mov  ah, 9            ; ah=9 - "print string" sub-func
    int  0x21             ; call dos services

    mov  dx, fname        ; open file to patch
        …
```

Practical Applications of Assembly: Debugging

- Programs written in C, etc are generally translated into assembly.
  - And then into machine code.
- Or you can look at the machine code of programs and get an assembly code listing.
  - And step through the program one instruction at a time.
- When programs crash (sometimes programs you don't have the code for) you can look at the assembly code and assess.

- Programs exist to help you (gdb, IDA Pro, radare, etc)

- In CS 449 you will apply this knowledge (using gdb).

Next Time…

- We will look at data representations!

- How do computers store different types of data?
  - Numbers
  - Letters??
  - Color/Images??

- And look at different numeric bases.
  - Decimal (our comfort zone)
  - Binary
  - Octal??
  - Hexadecimal