# CSE 2600
# Intro. To Digital Logic & Computer Design

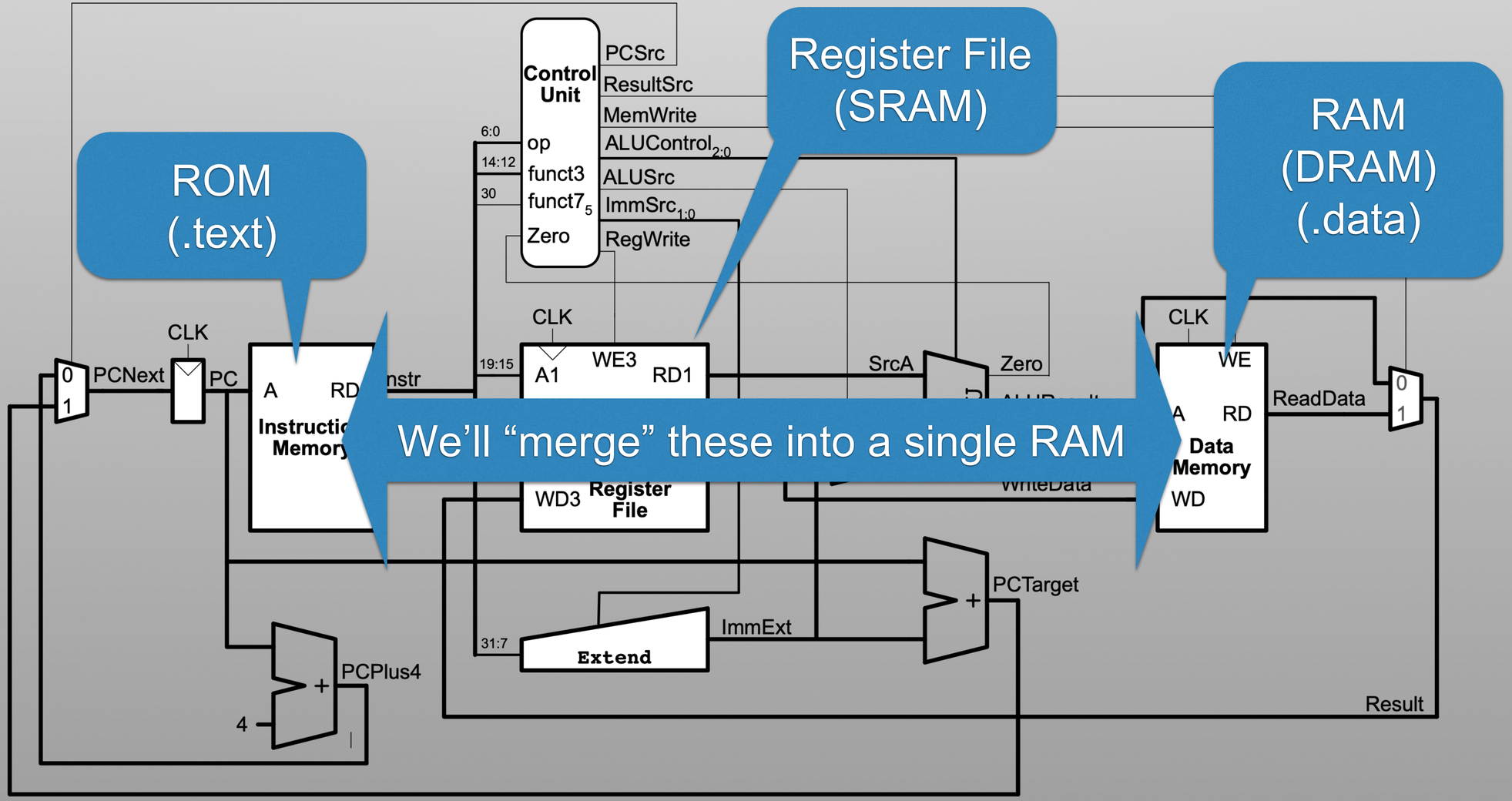
Bill Siever
&
Michael Hall

# This week

- Homework 6B posted - Due Sunday, April 5th by 11:59pm

# Studio 6A

# Foreshadowing: Simple RISC-V Computer



# Basic Model

- Machine is basically 2-3 memories + CPU
  - Registers (small, easy to use; temporary/ephemeral)
    - Ex:  You have 31, 32-bit data registers = 124 ***Bytes***
  - RAM:  Place for most data (Gigabytes!)
  - Program Memory: Possible in RAM or some additional “program memory”

# Program -> Assembly Language -> Machine Code -> Memory

# Problem: Find x such that $2^x = 128$

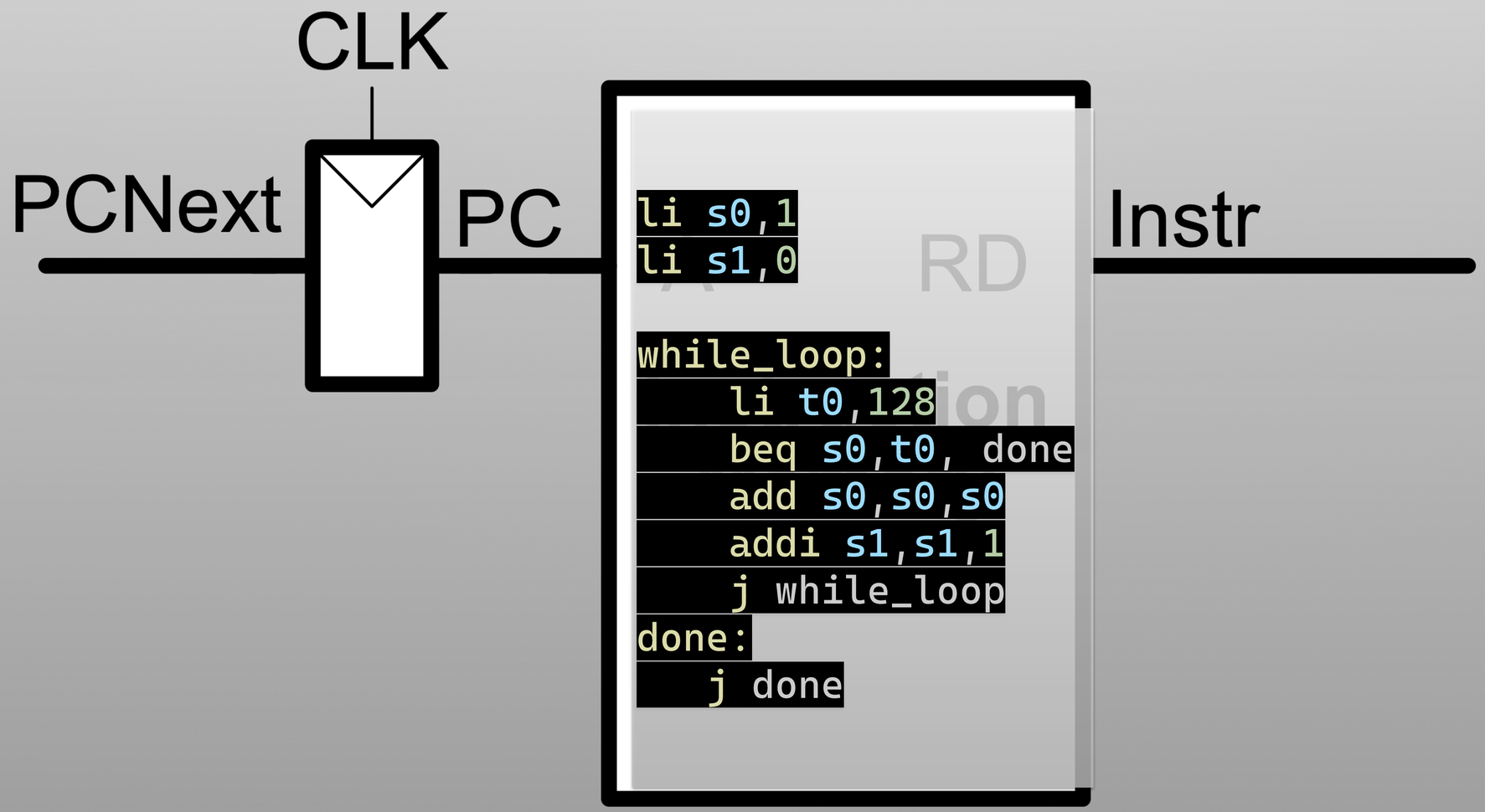
```
// determines the power
// of x such that 2^x = 128
// pow is 2^x; Start at 2^0 = 1
int pow = 1;
int x = 0;

while (pow ≠ 128) {
   pow = pow * 2;
   x = x + 1;
}
```

# Problem: Find x such that $2^x = 128$

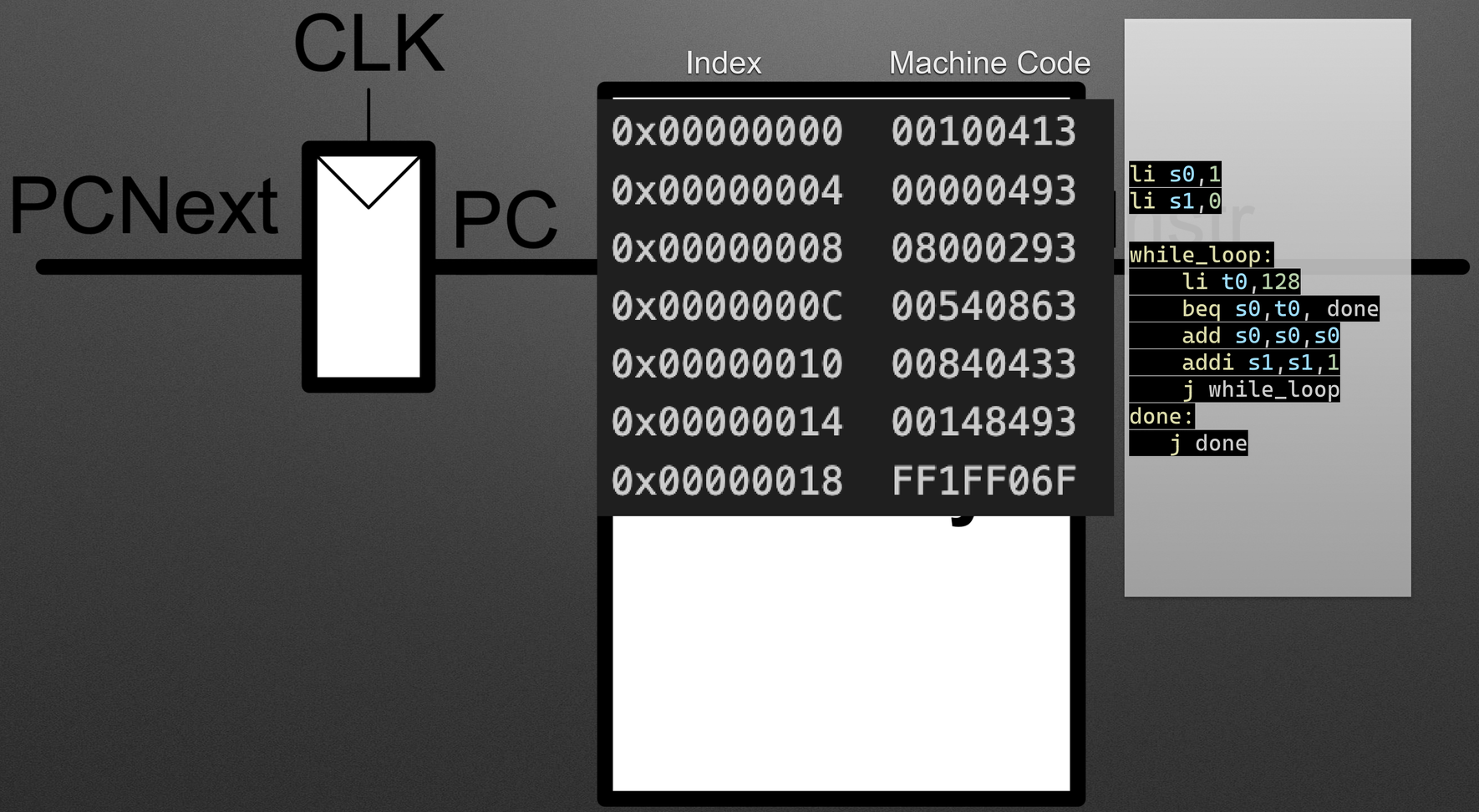
```
// determines the power of x such that 2x = 128
// pow is 2^x

int pow = 1;
int x = 0;

while (pow ≠ 128) {

   pow = pow * 2;
   x = x + 1;
}
```

```
// .text
// main:
//    li s0, 1
//    li s1, 0
// while_loop:
//    li t0, 128
//    beq s0, t0, done
//    add s0, s0, s0
//    addi s1, s1, 1
//    j while_loop
// done:
```

# Behavior: Parts of CPU Model



# Behavior: Parts of CPU Model



# Chapter 6

# Instruction Sets: Basic Categories

- Machine has small primitive set of “commands” in a few rough categories:
  - Data Manipulation: “Computation” (typically uses an ALU)

    ```
    add t0,t1,t2
    ```

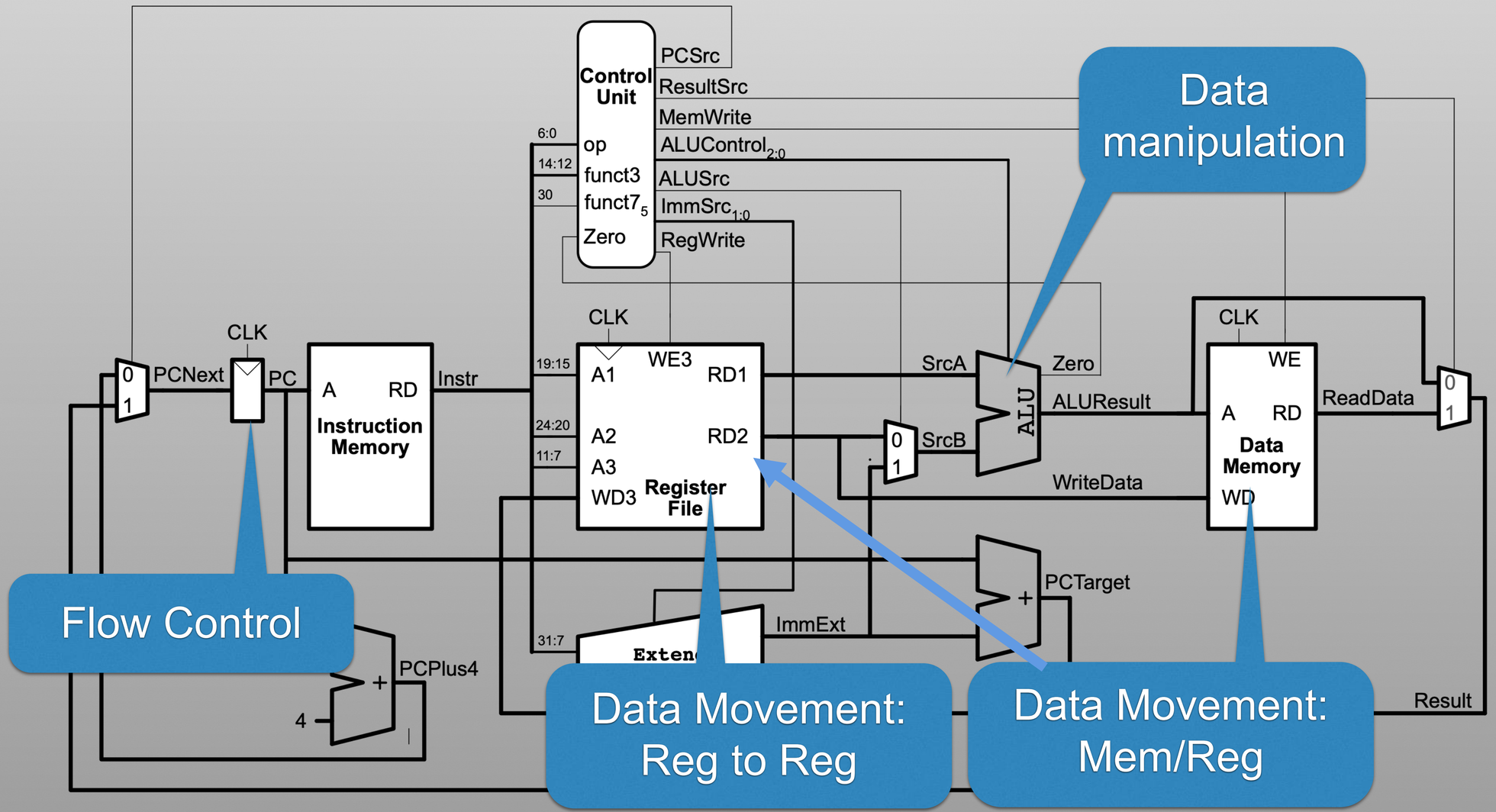
  - Data Movement: Move data between registers and RAM or initializing values

    ```
    lw t0, 8(sp)
    li t1,5
    ```

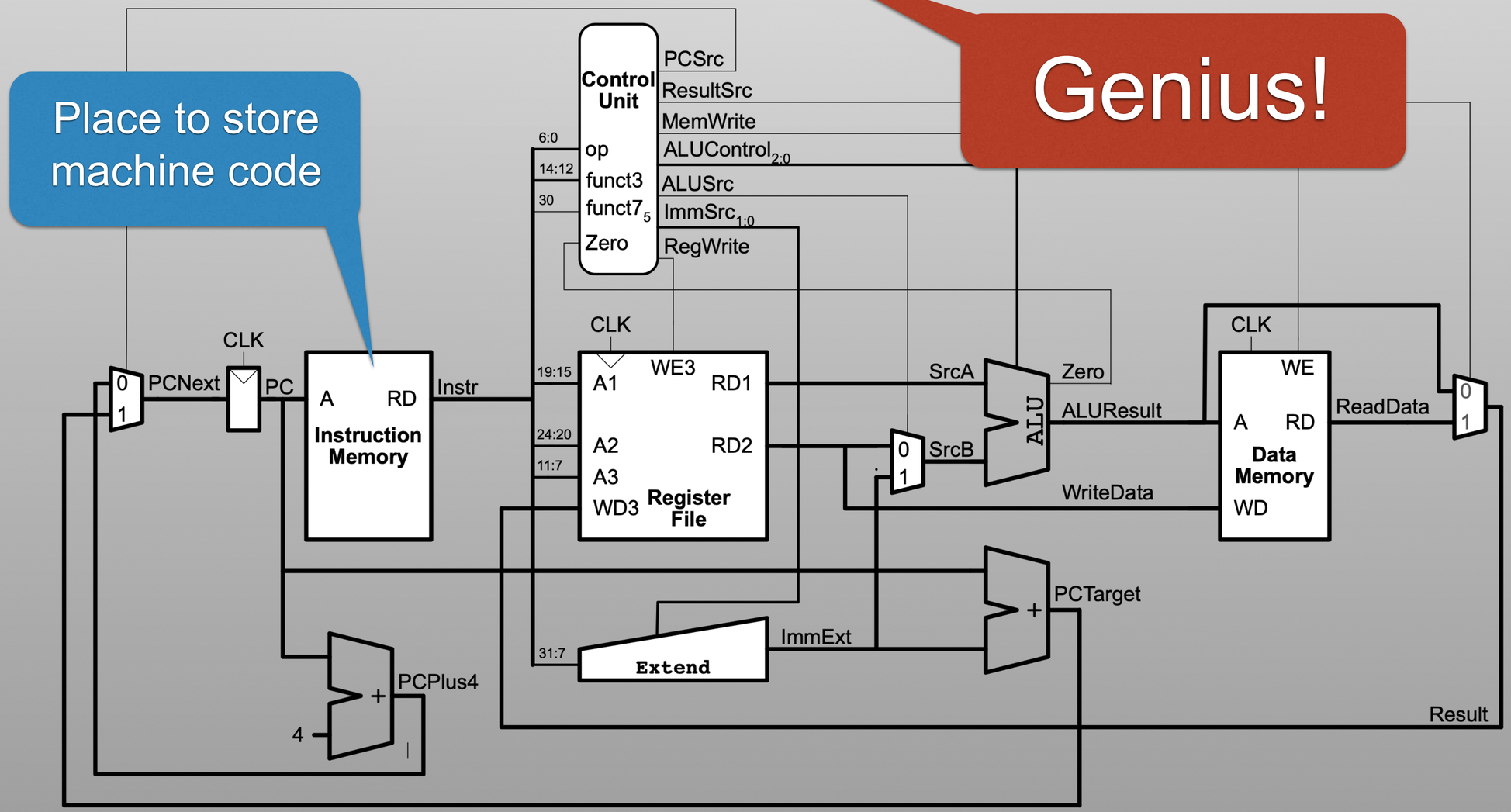
  - Flow Control: Controlling what instruction happens next (loops, if/else, functions)

    ```
    beq t0,t1, done
    ```

# Foreshadowing: Simple RISC-V Computer



# “Stored Program” Concept



# 2600 Prereq: Intro to Programming

# Programming Languages

- Prereq: Intro to Programming
  - Consider large programs and how you manage info.
  - Ex: A program/function that computes an average of three integers
    - Java vs. Python vs. C
  - Examples
  - “Types”

# Programming Langs: History & Motivation

1. Efficiency: Allow more people to create programs

   Compilers “compile” a high-level language (HLL) representation to a list of simpler assembly language instructions

   (Compilers are used obviously/explicitly in many languages, like Java & C; Often behind-the-scenes in others, like Python)

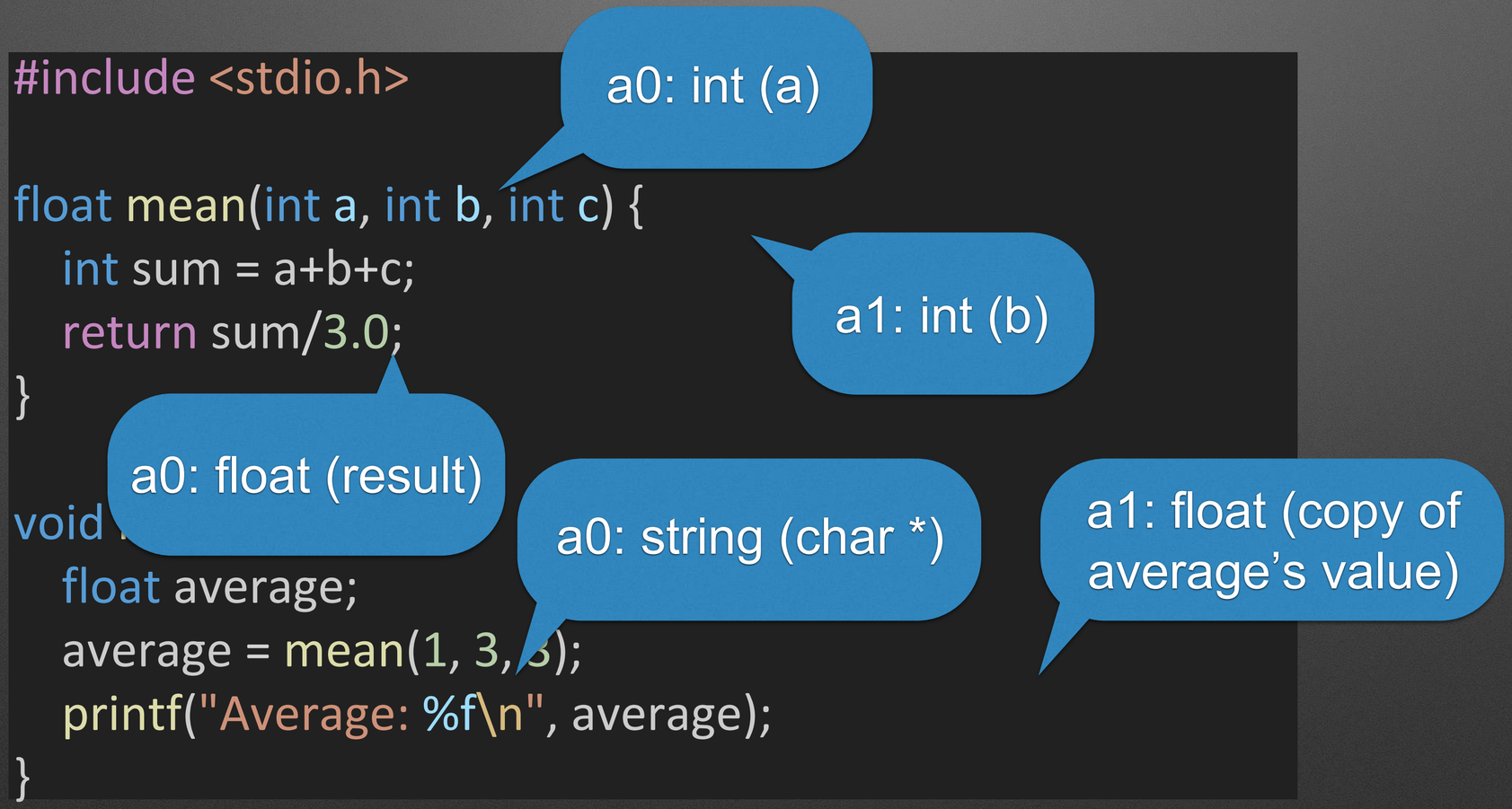
2. Manage complexity / avoid problems (increasing focus)

   Variables and Data Type rules are a large part of that

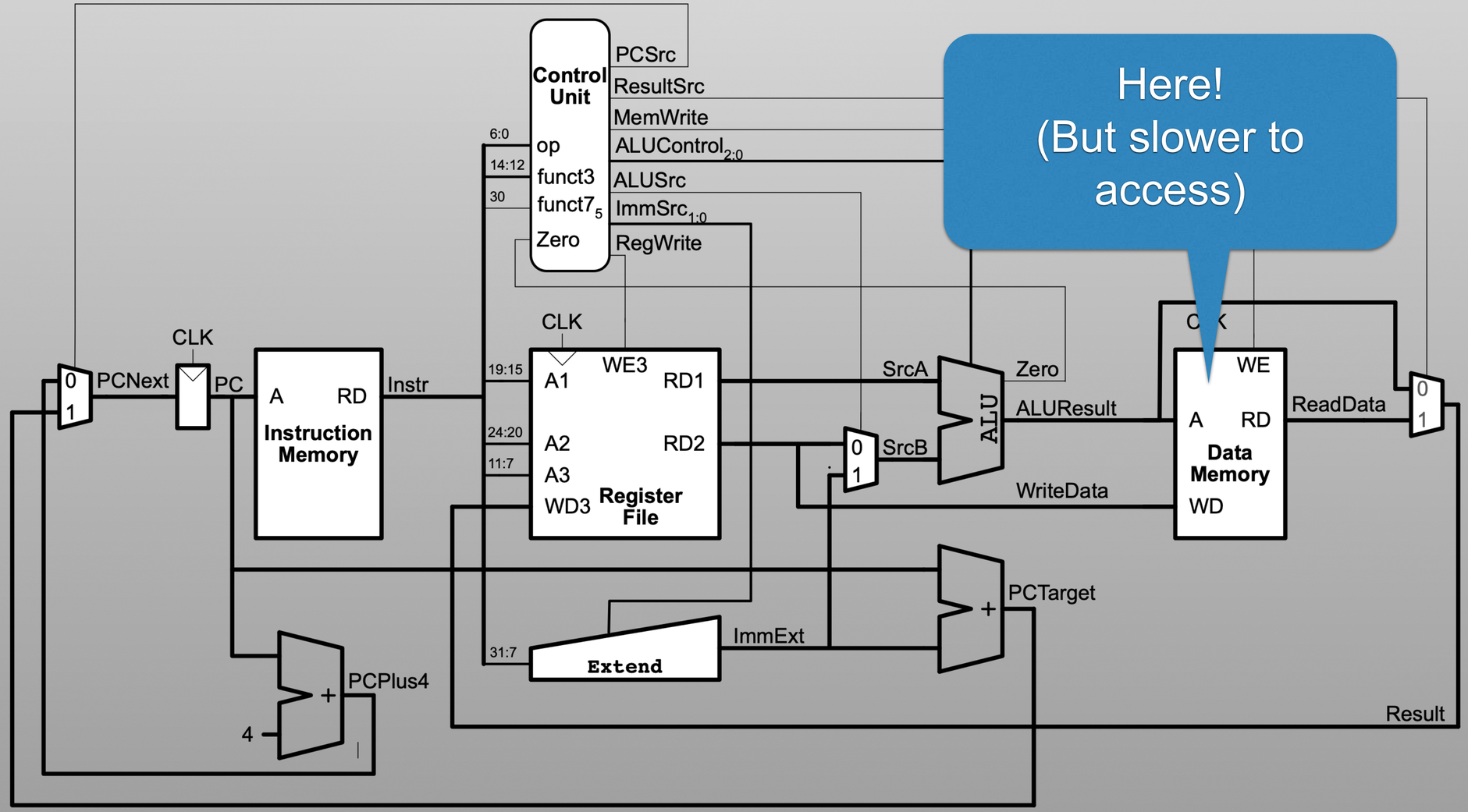
# Registers

- Just 31, 32-bit registers (124 bytes)
  - Used for all data operations!
  - Very very different than HLL
- **No types**
- **Meaning may change with time**

| Name | Number | Usage |
|---|---|---|
| `zero` | x0 | Constant value 0 |
| `ra` | x1 | Return address |
| `sp` | x2 | Stack pointer |
| `gp` | x3 | Global pointer |
| `tp` | x4 | Thread pointer |
| `t0-2` | x5-7 | Temporaries |
| `s0/fp` | x8 | Saved register / Frame pointer |
| `s1` | x9 | Saved register |
| `a0-1` | x10-11 | Function arguments / return values |
| `a2-7` | x12-17 | Function arguments |
| `s2-11` | x18-27 | Saved registers |
| `t3-6` | x28-31 | Temporaries |


#include <stdio.h>
float mean(int a, int b, int c) {
int sum = a+b+c;
return sum/3.0;
}
void
float average;
average = mean(1, 3,
printf("Average: %f\n", average);
}
a0: int (a)
a1: int (b)
a0: float (result)
a0: string (char *)
a1: float (copy of average's value)

# More Data!

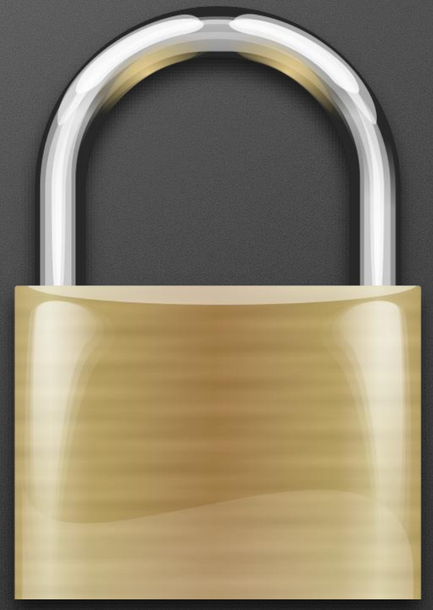

Here!
(But slower to access)
Control Unit
op
funct3
funct7$_5$
Zero
PCSrc
ResultSrc
MemWrite
ALUControl$_{2:0}$
ALUSrc
ImmSrc$_{1:0}$
RegWrite
6:0
14:12
30
CLK
PCNext
PC
Instruction Memory
A
RD
Instr
19:15
24:20
11:7
WE3
A1
A2
A3
WD3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
Data Memory
WD
ReadData
0
1
PCTarget
ImmExt
31:7
Extend
PCPlus4
4
+
Result

# Sharing Resources

- Examples:
  - **Kitchen** – Can use dishes, utensils, and ingredients, but must clean up and put things back where they belong
  - **Books in a library** – Can borrow books but must return them to the same place so that others can find them later
  - **Lab Instrument** – Can borrow an oscilloscope or FPGA board, but return it in the same condition, settings, and cable connections as before

# The Rules

- **Sharing Registers / Memory:** You can do almost anything as long as you put things back how you found them before finishing (but how?)



# The Rules

- Register Conventions:
  - “Convention” (agreement about use) for how registers will be shared
- & Beyond — Memory & the “Stack Discipline”
  - Rules for how to use memory for additional info
  - And to use as the “copy” to restore to original condition

# The Rules

- Memory
  - Use part for the “run time stack”
  - Create a “stack”: Like stack data structure (CSE 247 / 2407)
    - Last-in-first-out

# LIFO


# Why a stack

- main()
  - run_game()
    - update()
      - update_character()
        - update_life()

# Why a stack

- main()
  - run_game()
    - update()
      - update_character()

# Why a stack

- main()
  - run_game()
    - update()

# Why a stack

- main()
  - run_game()

# Why a stack

- main()

# LIFO with an array

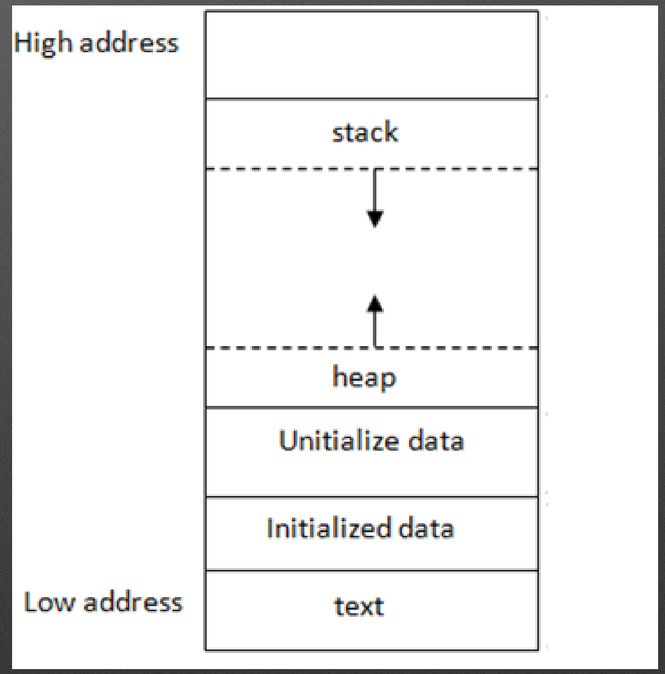
- Use an array
  - Keep track of index of “last item”
  - Add / (push)
  - Remove (pop)

# Data / RAM

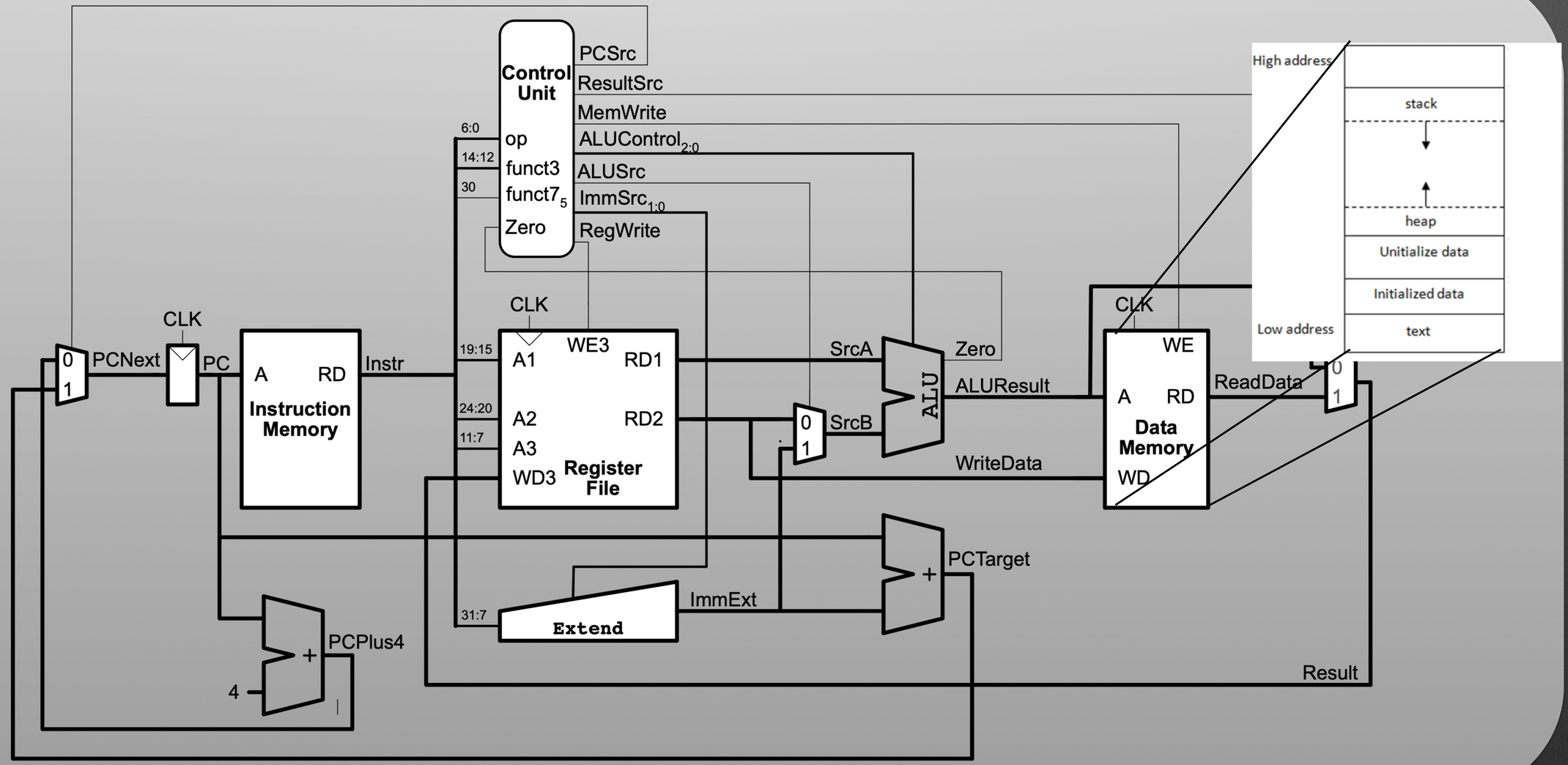
- Arrays (in programming languages) are just a representation of a segment of RAM
  - So, RAM works like arrays — index based
  - There’s a “base”: The index that it starts at
  - However, RAM is an array of BYTES
    - Data types like an `int` are 4 bytes

# Data / RAM

- Split it into regions to serve different purposes
  - Stack
    - Of records for all the currently running functions
  - Heap: A giant heap ‘o memory
    - For *dynamic* memory (new / malloc).
      like in Java: `Thing newThing = new Thing()`



# More Data!


Control Unit
op
funct3
funct7$_5$
Zero
PCSrc
ResultSrc
MemWrite
ALUControl$_{2:0}$
ALUSrc
ImmSrc$_{1:0}$
RegWrite
6:0
14:12
30
CLK
PCNext
PC
Instruction Memory
A
RD
Instr
19:15
24:20
11:7
WE3
A1
A2
A3
WD3
RD1
RD2
Register File
SrcA
SrcB
ALU
Zero
ALUResult
WriteData
WE
ReadData
Data Memory
WD
PCTarget
ImmExt
31:7
Extend
PCPlus4
4
Result
High address
stack
heap
Unitialize data
Initialized data
text
Low address

# Studio 6A, Function Fun, & RA

# Questions

- What are the main advantages of RISC-V compared to x86?  Is there anything that one can do that the other cannot?
- How does the processor decide which exception to handle first if multiple exceptions happen at the same time?
- I am still a little confused about how the memory addressing works, especially how addresses relate to data size (byte vs. word).
- I don’t really understand what ‘machine code’ really is.
- Instead of having a separate assembler and linker, why don’t we have a single tool that looks at all the files simultaneously and produces the final machine code?

# Arrays

```
int i;                  // use s1
int scores[200];  // use s0 for the base of scores
for (i = 0; i < 200; i = i + 1)
   scores[i] = scores[i] + 10;
```

# Next Time

- Studio