CSE 260M / ESE 260
Intro. To Digital Logic & Computer Design

Bill Siever
&
Jim Feher

This week

- Thursday:
  Studio — Here / Seigle 301
  Bring kits (1 per group)— will be used briefly

- Hw#6 Due Fri.

Chapter 6 & 7

Architectures: RISC-V

- “Architecture”: Programmer’s view of CPU
- Fundamental data size: 32-bit “word”. CPU/ALU designed for 32-bit operations (Multiple 32-bit operations can be done to do larger operations)
- Memories
  - RAM: Big array of numbers; Uses 32-bit addresses
  - Registers: Array with 32, 32-bit values. Special names correspond to intended uses (Ex: a-registers,like a0, are for “arguments” to functions)
  - Instructions: Also 32-bit values; May be stored in RAM or separate memory

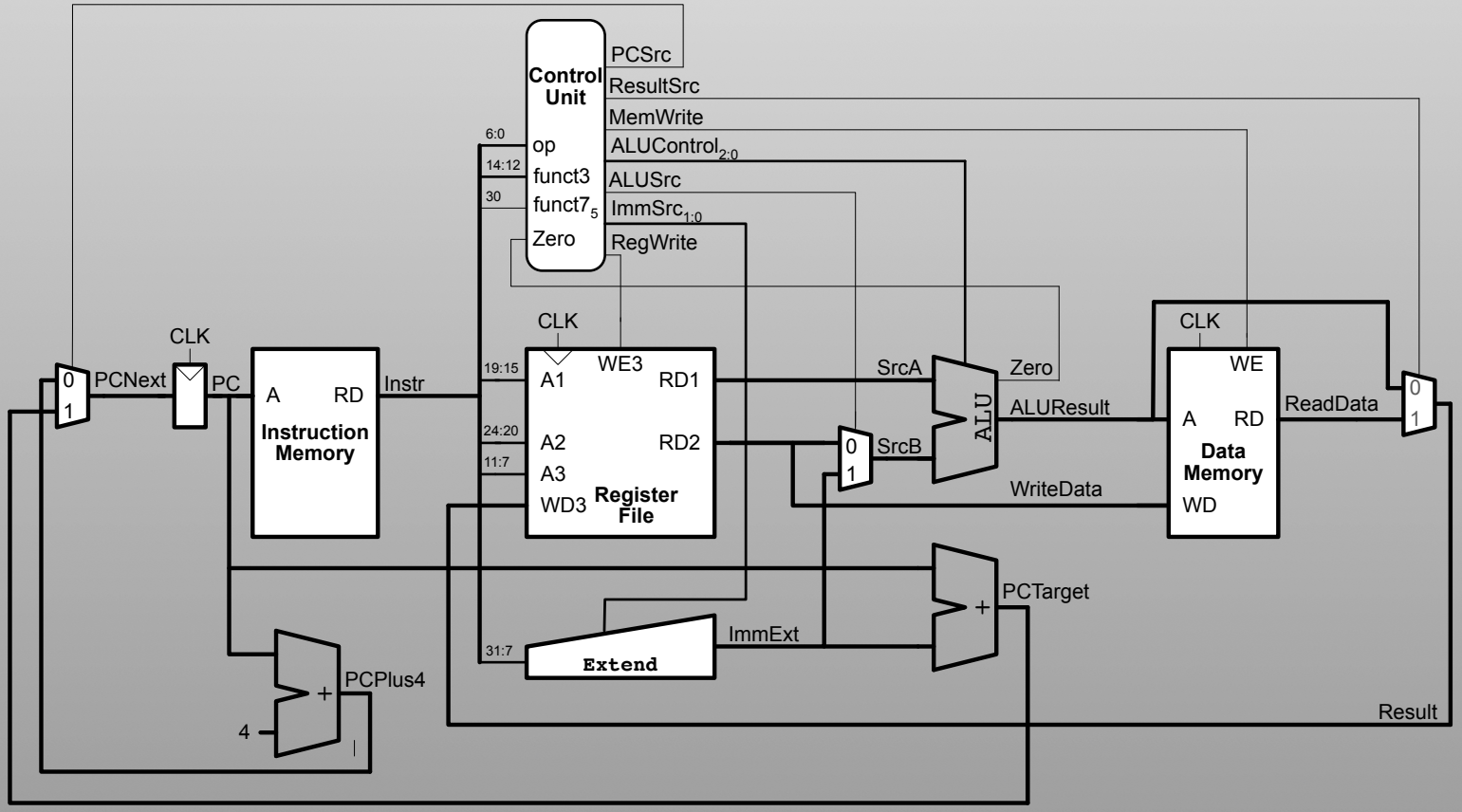
Architectures: RISC-V

- Machine codes
  - “Substitution code”: Numbers represent concepts
  - RISC-V “Instruction Set Architecture” (ISA):
    - Formats are about data locations (“addressing information needed”)

Architectures: RISC-V

- Three major kinds of things instructions do:
  - Computation (use the ALU; do basic math/logic): `add`, `addi`, `or`, ...
  - Move data (between registers or registers to/from memory): `mv*`, `lw`, `sw`, ...
  - Control program flow (which instruction happens next): `beq`, `blt`, `cal`, ...

Simple RISC-V Computer



The Problem

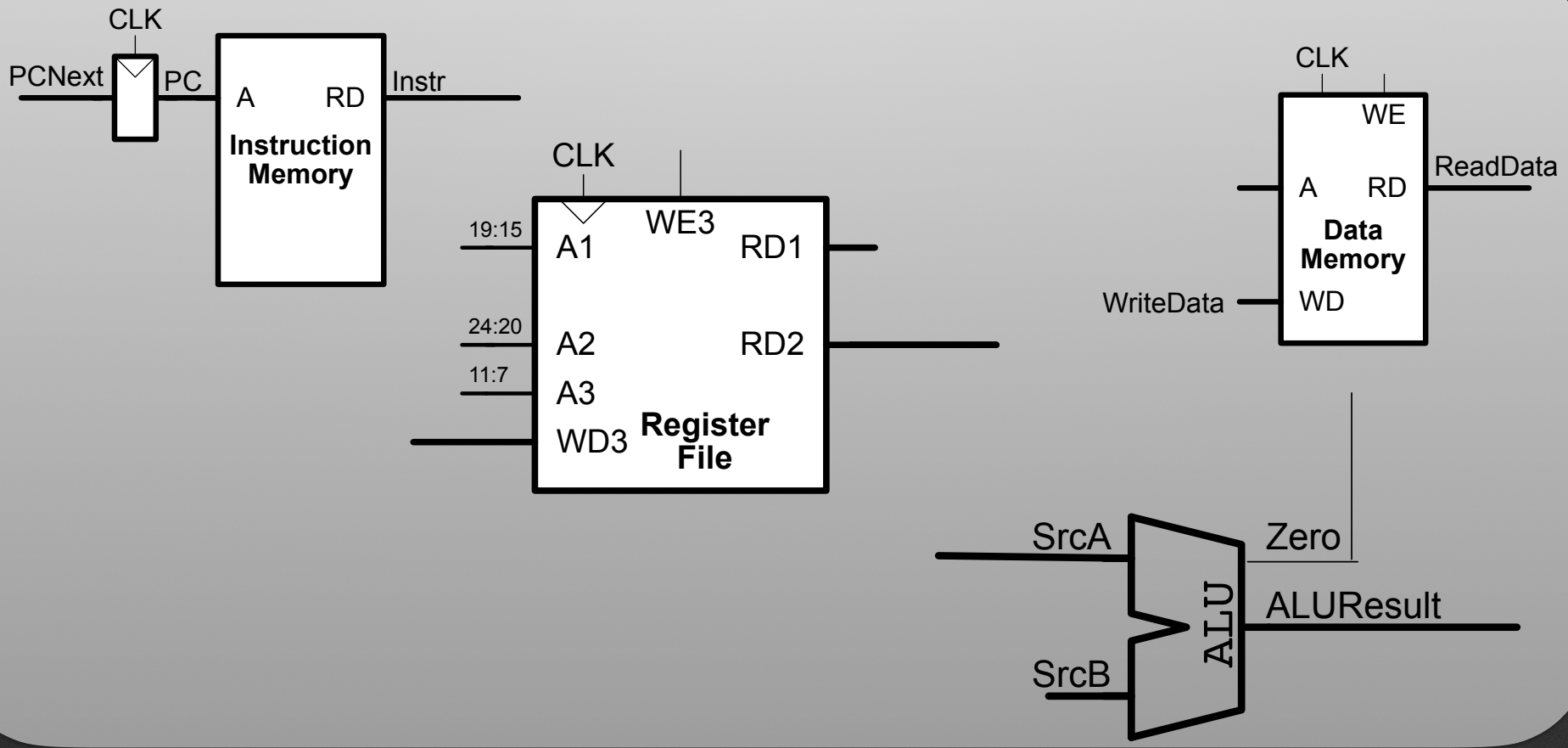
Find x such that $2^x = 128$

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

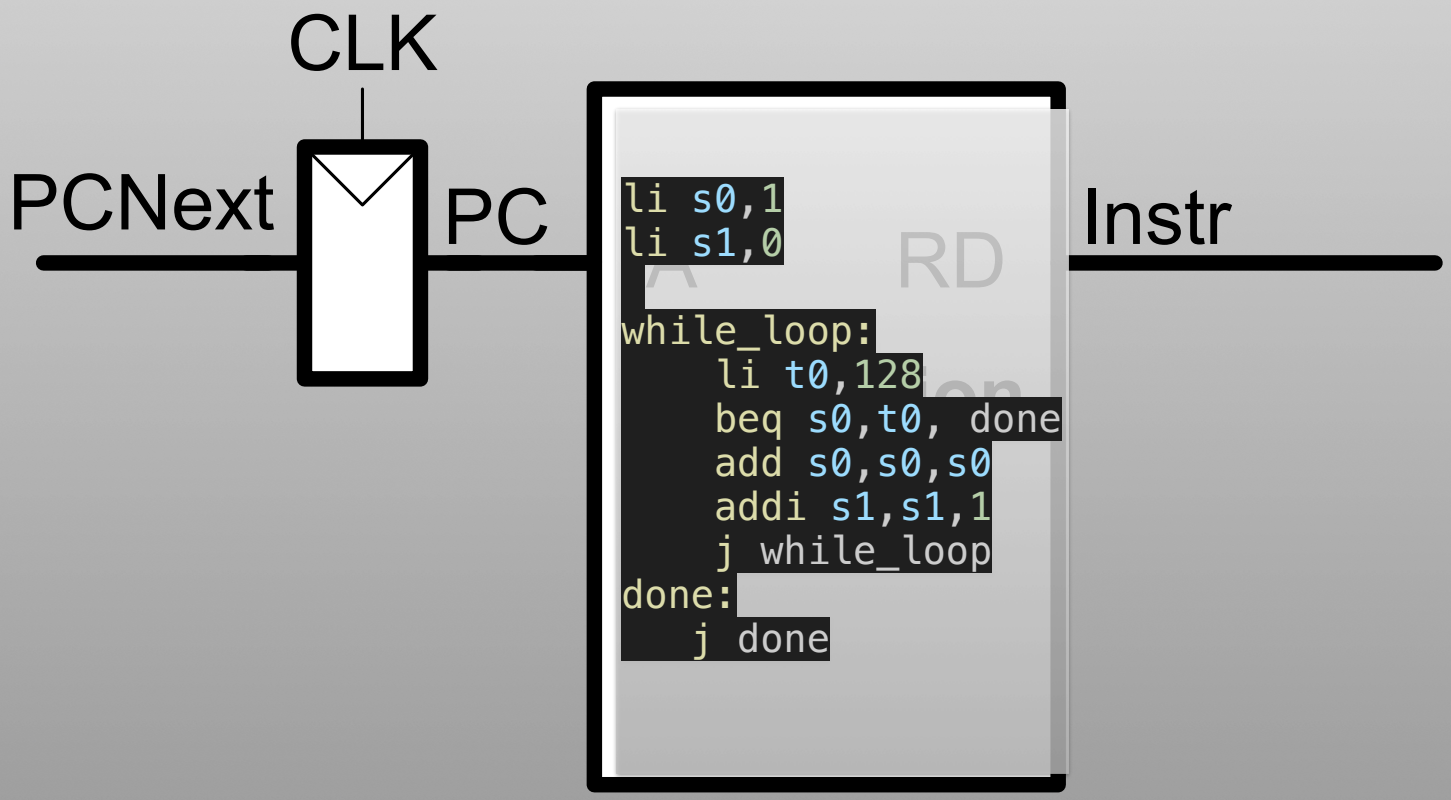
```
// determines the power
// of x such that 2x = 128
int pow = 1;
int x = 0;

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

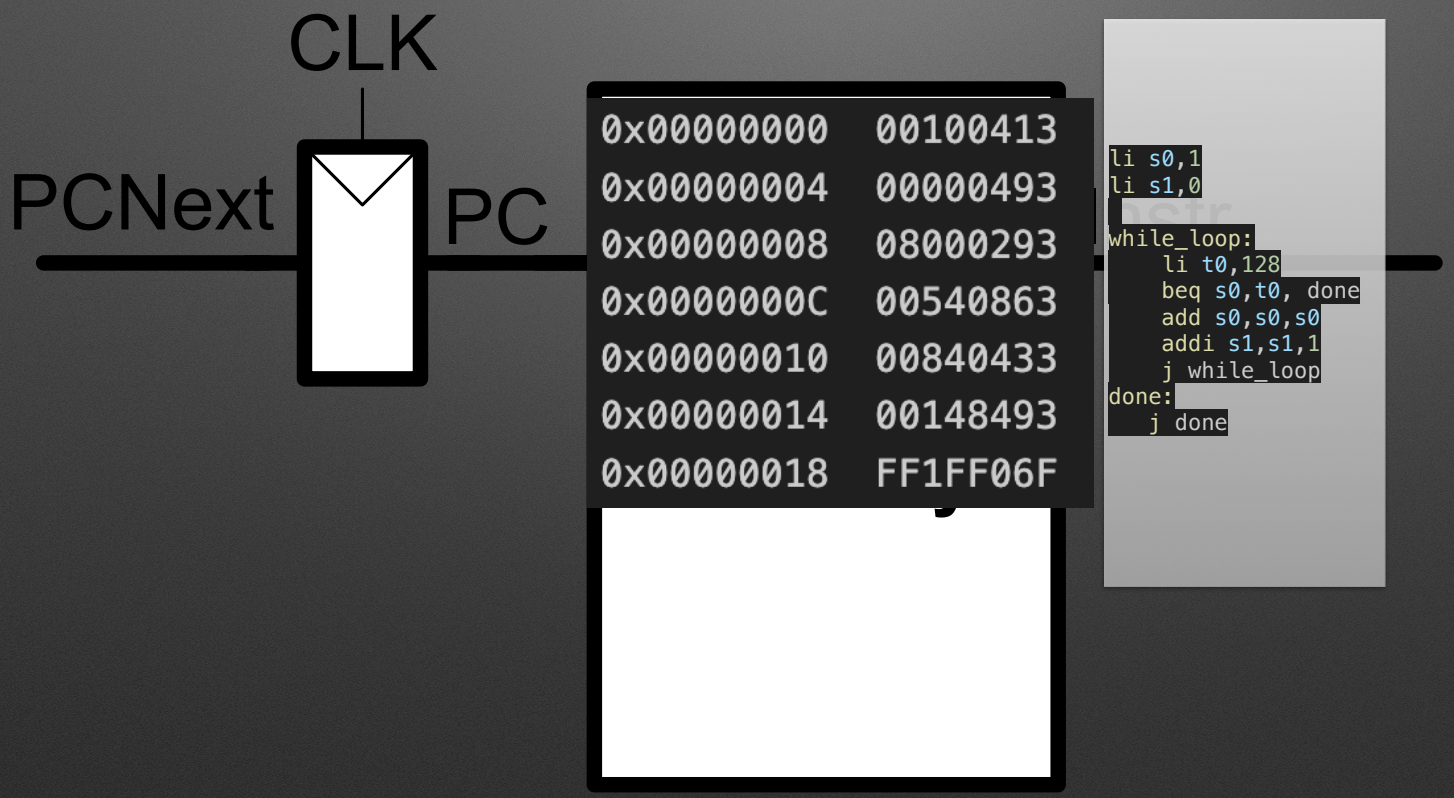
Behavior: Parts of CPU Model



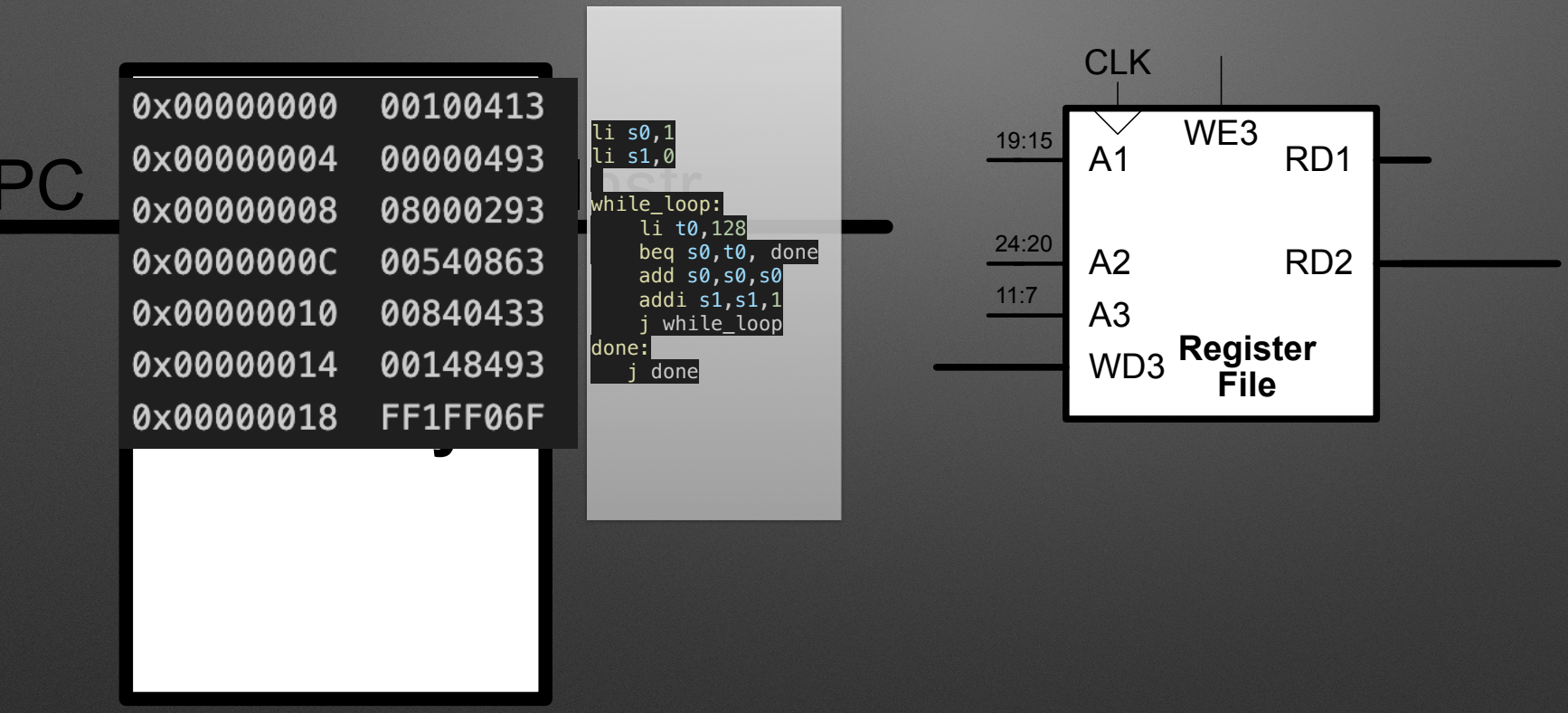
Behavior: Parts of CPU Model



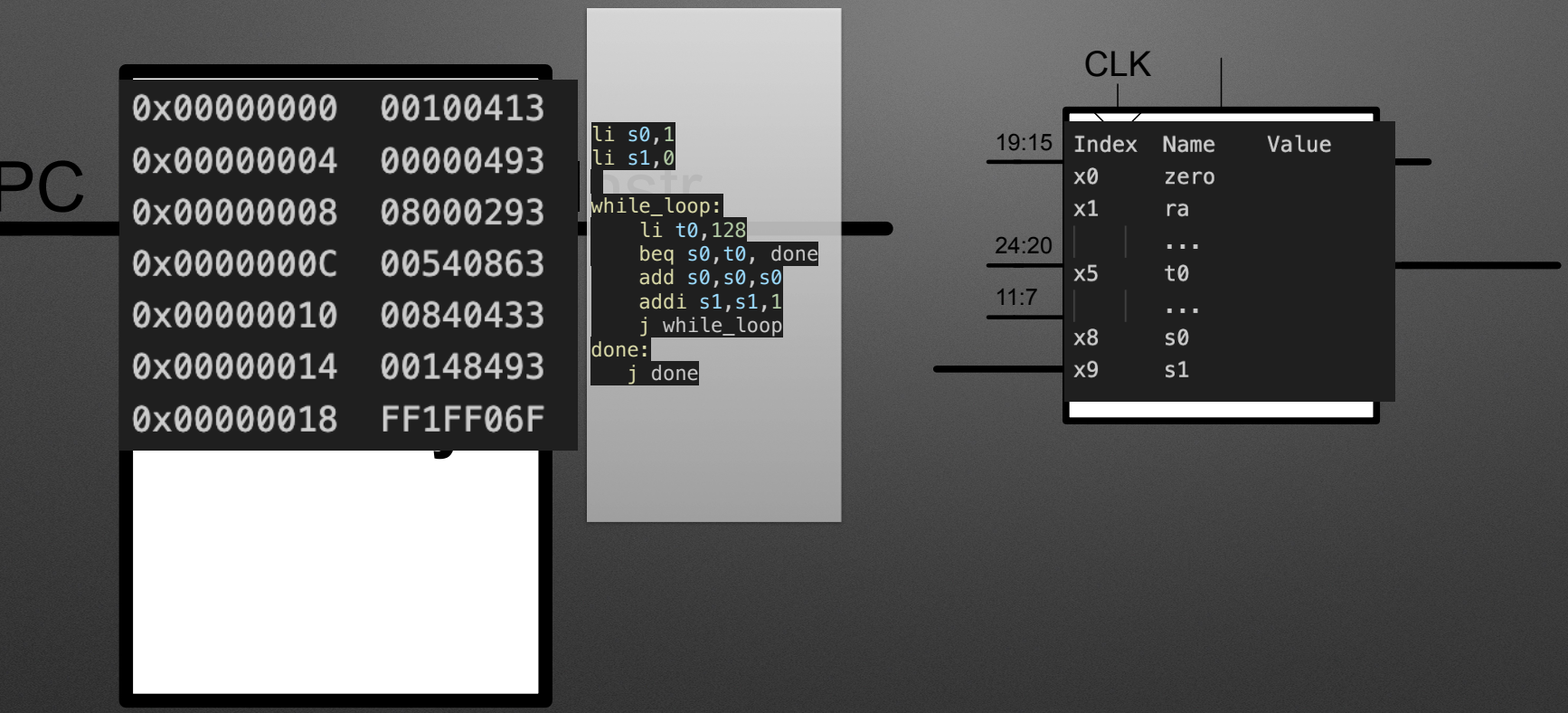
Behavior: Parts of CPU Model



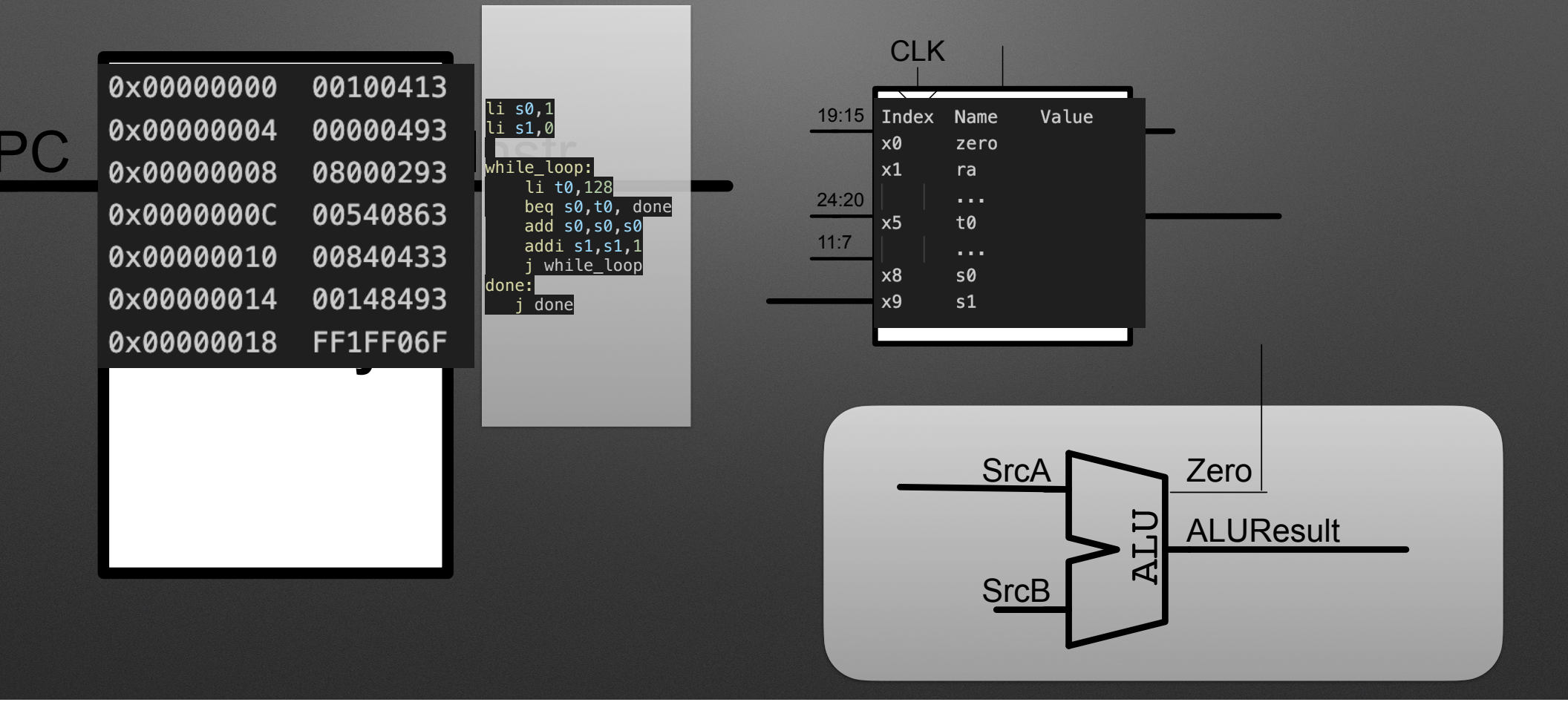
Behavior: Parts of CPU Model



Behavior: Parts of CPU Model



Behavior: Parts of CPU Model



Parts of CPU Model

```
0x00000000   00100413
0x00000004   00000493
0x00000008   08000293
0x0000000C   00540863
0x00000010   00840433
0x00000014   00148493
0x00000018   FF1FF06F
```

```
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: j done
```

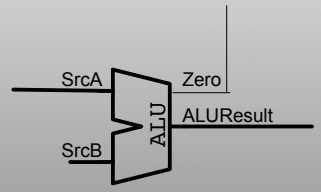
CLK

19:15

24:20

11:7

Index	Name	Value
x0	zero	
x1	ra	
	...	
x5	t0	
	...	
x8	s0	
x9	s1	



Big Picture

- Operations can be represented as numbers
- Operations manipulate memory, registers, and the “Program Counter”
  - Program Counter mostly “counts” (by 4s here, since each instruction is 4 bytes)
- Labels are addresses
  - An address is also an index into something, like RAM

RAM

- RISC-V: “Load-Store Architecture”
  - Specific instructions to
    - Transfer from RAM to register (load data)
    - Transfer from register to RAM (store data)
  - Alternative architectures (x86) able to directly combine data from memory with register data

Reading (from Men to Reg): Load

- Mnemonic: load word (`lw`)
- Format: `lw t1, 5(s0)`
  `lw` destination, offset(base)
- Address calculation: RAM index = add base address (`s0`) to the offset (`5`) = (`s0 + 5`)
- Result: `t1` holds the data value at address (`s0 + 5`)
  I.e. `t1 = RAM[s0+5]`
- In terms of "register" array:

```
REG[t1] = RAM[REG[s0]+5]]
REG[6] = RAM[REG[8]+5]]
```

Reading (from Reg to Men): Store

- Mnemonic: store word (`sw`)
- Format: `sw t1, 5(s0)`
  `sw` destination, offset(base)
- Address calculation: RAM index = add base address (`s0`) to the offset (`5`) = (`s0 + 5`)
- Result: `t1` holds the data value at address (`s0 + 5`)
  I.e. `RAM[s0+5] = t1`
- In terms of "register" array:

```
RAM[REG[s0]+5]] = REG[t1]
RAM[REG[8]+5]] = REG[6]
```

- ***NOTE: Stores are the one case where the "thing on the left" is not the destination!***

Problem

- Assume:
  a0 contains a “base address” of an array of words
  a1 is the number of words (length of the array)

- Draw a picture of this in terms of the RAM

- Write assembly language code to: Initialize all the words in the array to 5.

Next Time

- Studio