CS 152, Spring 2012
Section 1

Christopher Celio

University of California, Berkeley

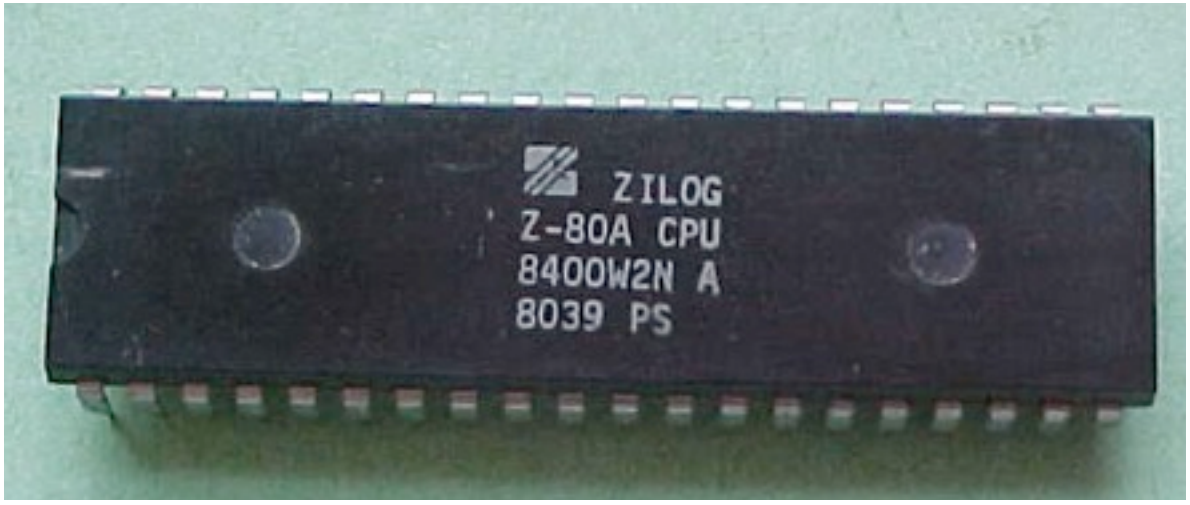
Friday, January 27, 2012

Administrivia

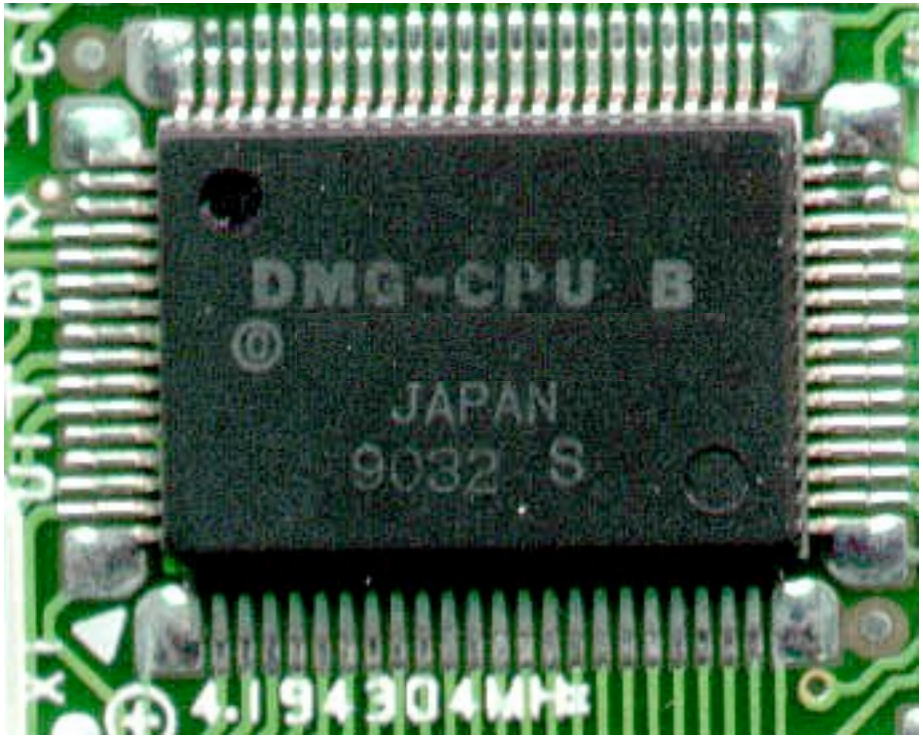
- Account forms
  - For Instructional Accounts
  - Sign sheet

About Me

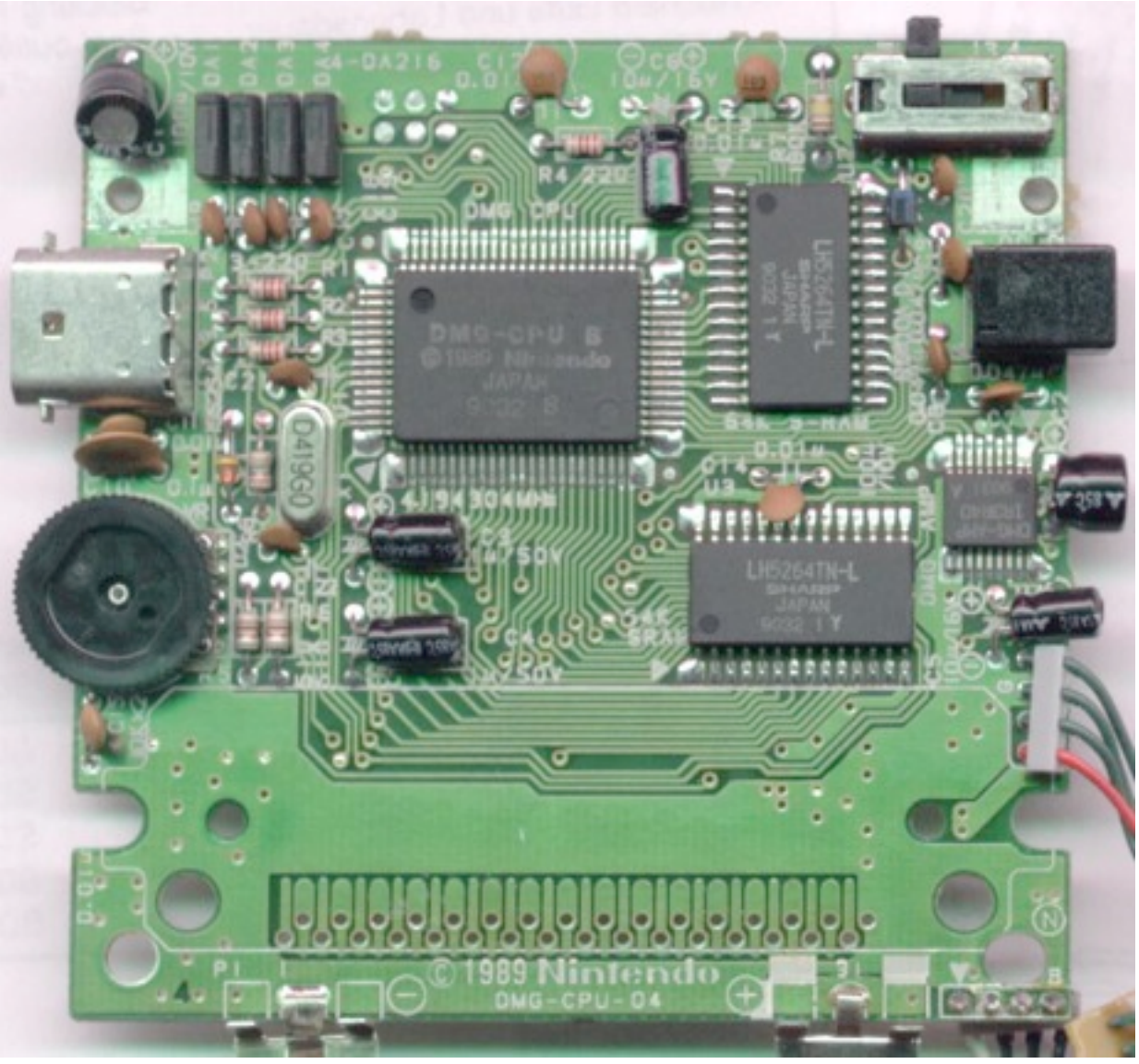
- Christopher Celio
- celio @ eecs



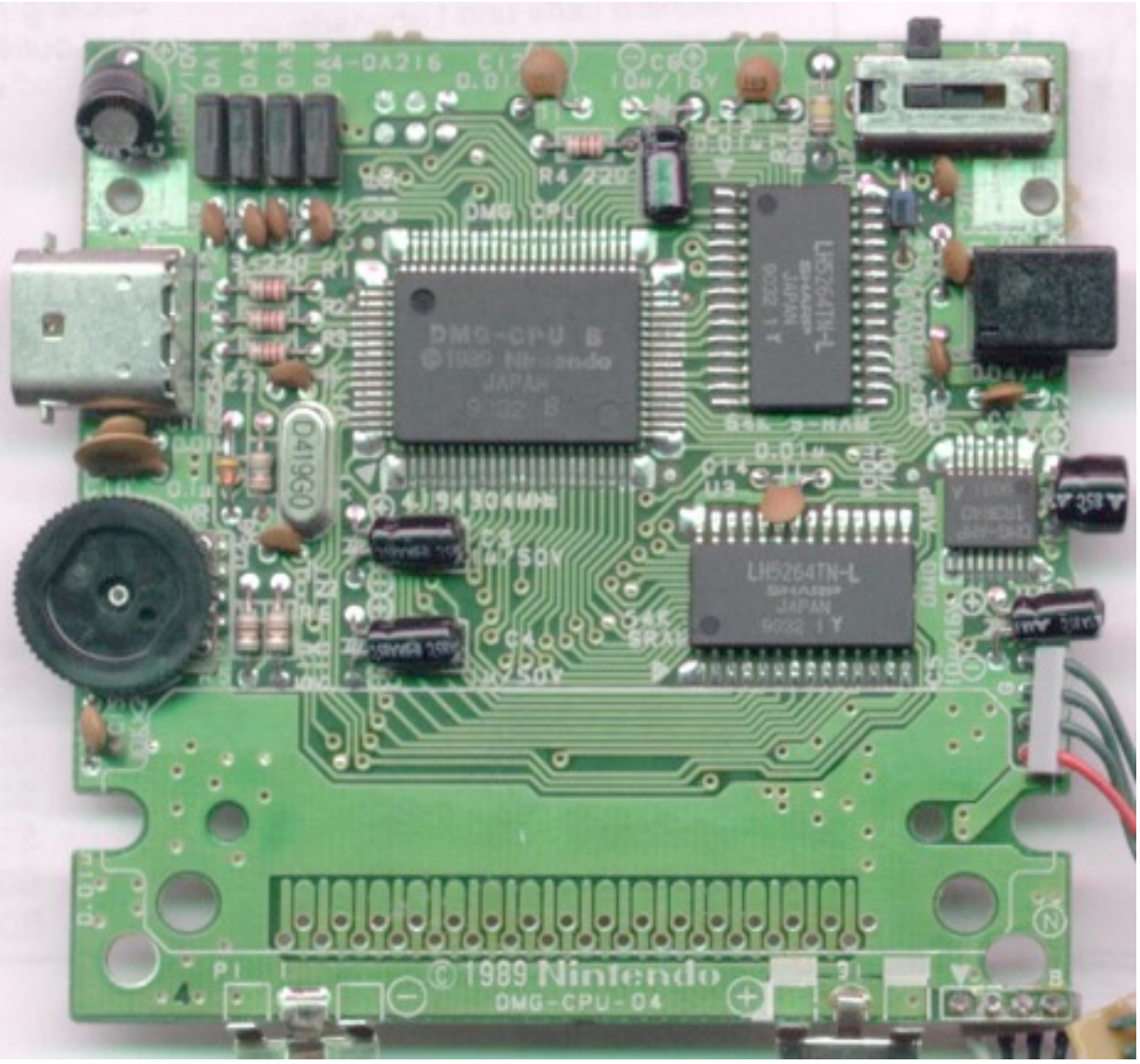
Mystery CPU



Recognize it now?



Recognize it now?



Z80-like

- Custom 8-bit Sharp LR35902
  - @ 4.19 MHz
  - A few missing registers from the Z80, different I/O and interrupt behavior
  - also very similiar to the 8080
- RAM 8 kB
- Video RAM 8 kB
- ROMs from 32kB to 8Mb
- 0.7 W (~15 hours from 4 AA)

Z80

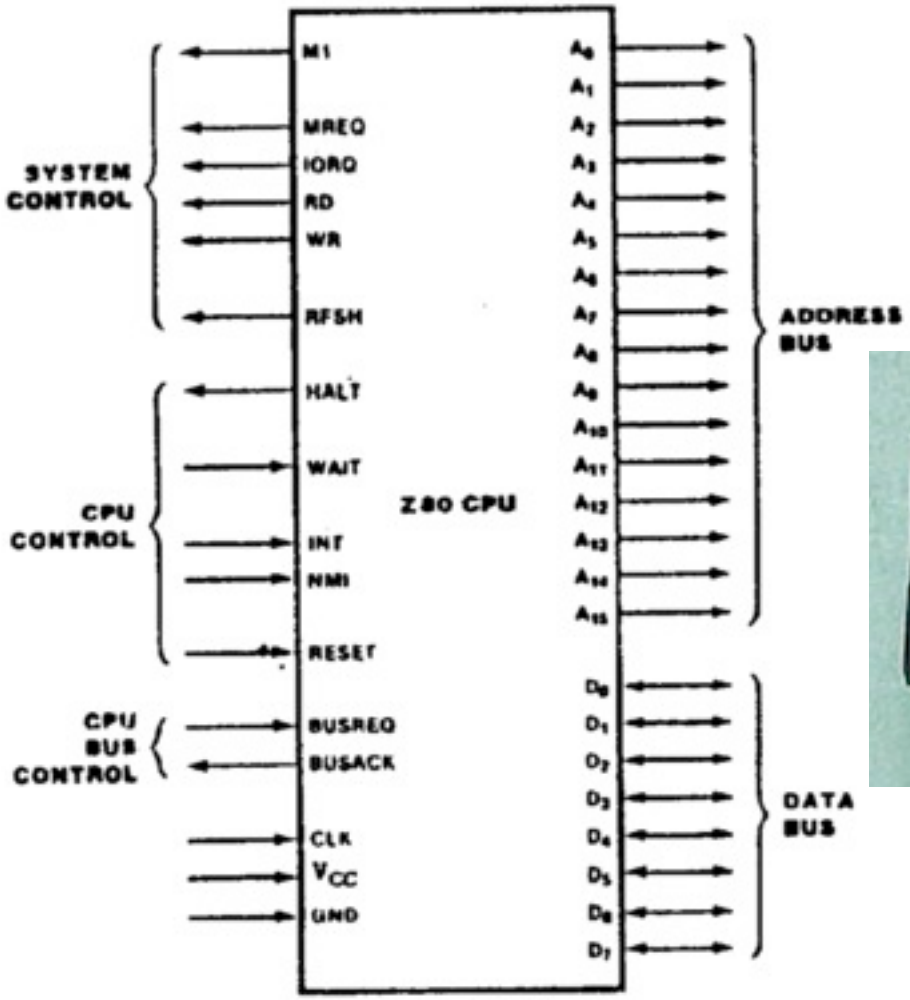
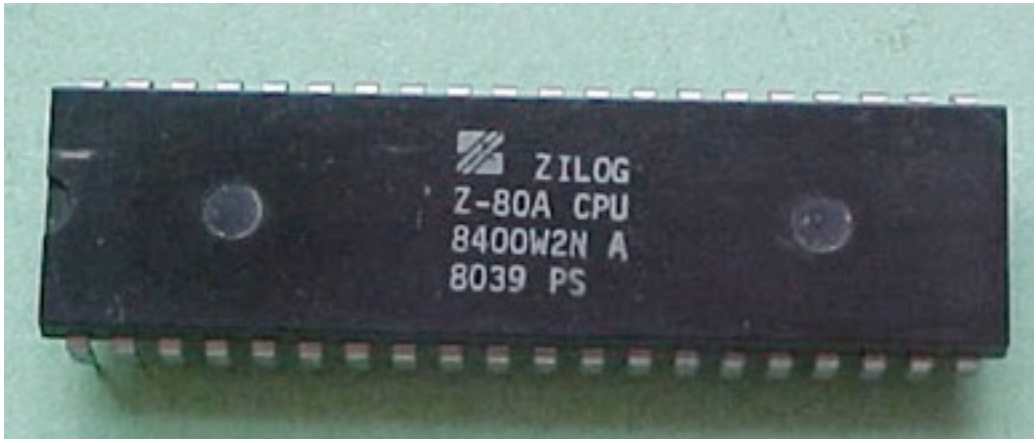
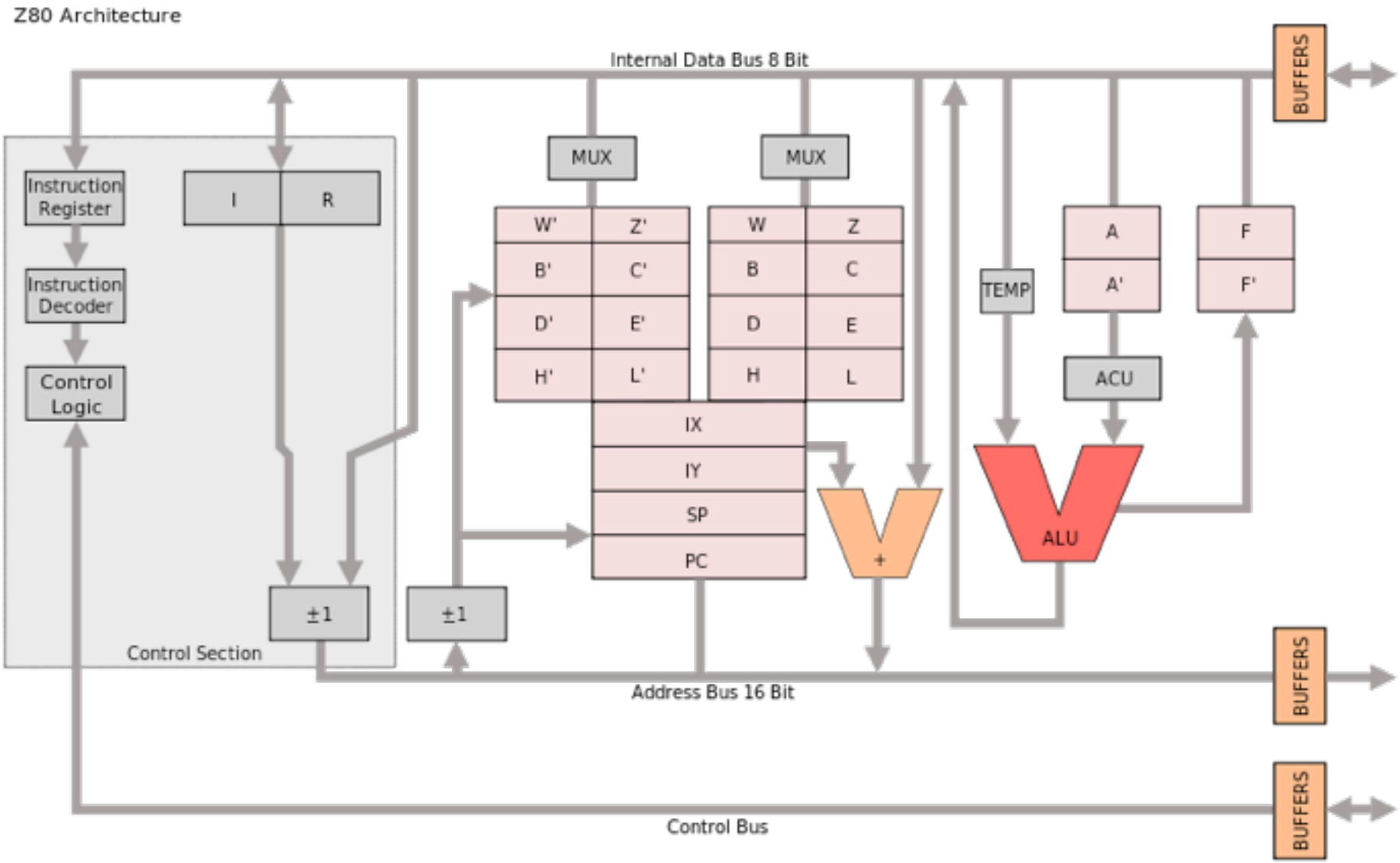


Figure 1. Logic Functions



Z80


Z80 Architecture
Internal Data Bus 8 Bit
BUFFERS
Instruction Register
Instruction Decoder
Control Logic
I
R
MUX
MUX
W'
Z'
B'
C'
D'
E'
H'
L'
W
Z
B
C
D
E
H
L
IX
IY
SP
PC
TEMP
A
A'
F
F'
ACU
ALU
+
±1
±1
Control Section
Address Bus 16 Bit
BUFFERS
BUFFERS
Control Bus

Agenda

- Lab 1: What are we asking you to do?
- RISC-V Infrastructure
  - What is available?
  - How to use it?
- Chisel
  - What is it?
  - Some basic Chisel Examples

Familiarity Survey

- C
- Python
- gdb
- emacs/vim
- Unix/Linux
- makefiles

Administrivia

- Join Piazza!
  - All notifications go out on it
  - Room changes, lab errata, etc.
- Handouts
  - http://www-inst.eecs.berkeley.edu/~cs152/sp12/
    - PS #1
    - Lab #1
    - Micro-coded RISC-V Handout (~cs152/sp12/handouts.html)
    - Chisel Tutorial
    - RISC-V ISA Manual

PSet 1

- Is out now!
- Great preparation for the test
- Each question is graded on 0,1,2 in terms of effort shown
- 15% of your final grade

Lab 1

- Is out now!
- Caveat: Brand new!

Lab 1: In The Past

- Simics
  - Software Simulator
    - needed additional packages to fake cycle-accurate
  - No way to see how the processor works, or change it

Lab 1

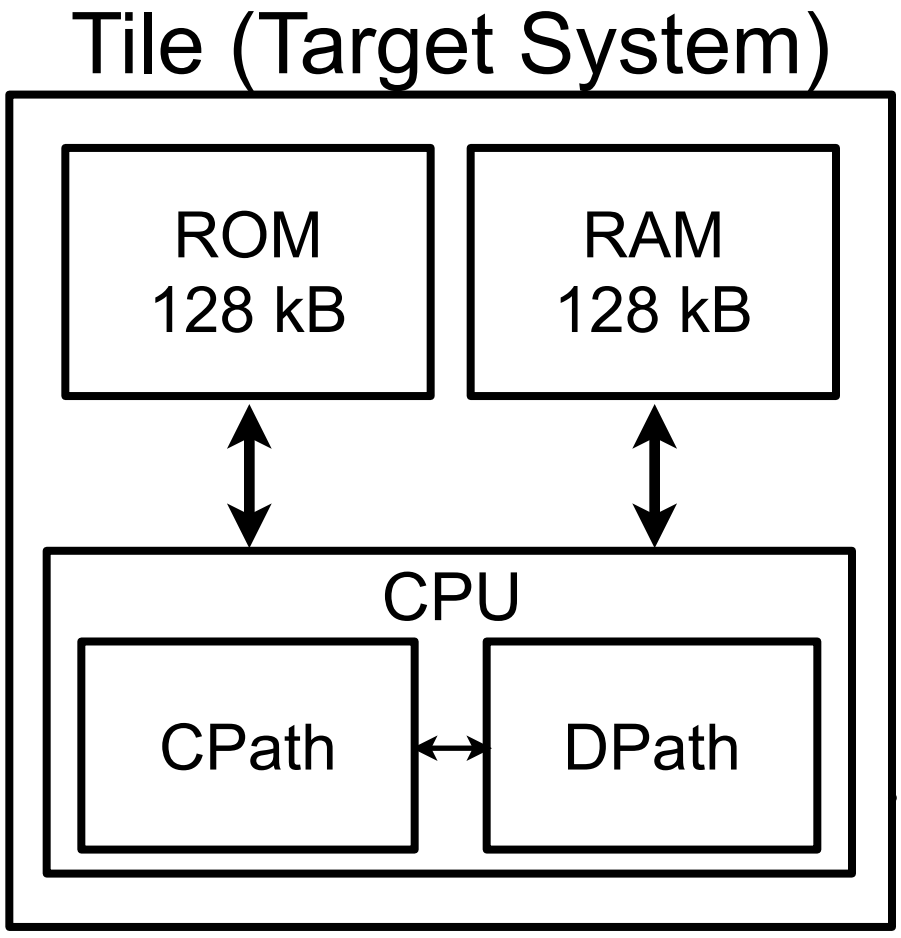
- Provided RISC-V Processors
  - implemented in Chisel
- Build C++ simulators
- Run benchmarks on simulators, gather numbers
  - CPI, instruction mix
- Answer, make recommendations, propose new designs

Lab 1

- Provided RISC-V 32-bit Processors
  - 1-stage
  - 2-stage
  - 5-stage
    - fully bypassed
    - fully interlocked (stalls to resolve all hazards)
  - micro-coded
- Only use 1-stage, 5-stage in Lab
  - 2-stage, micro-code are there for your own edification (or open-ended portion, if you choose)
- Processors use 1 cycle access memory
  - 128 kB of memory (nothing else to back it)

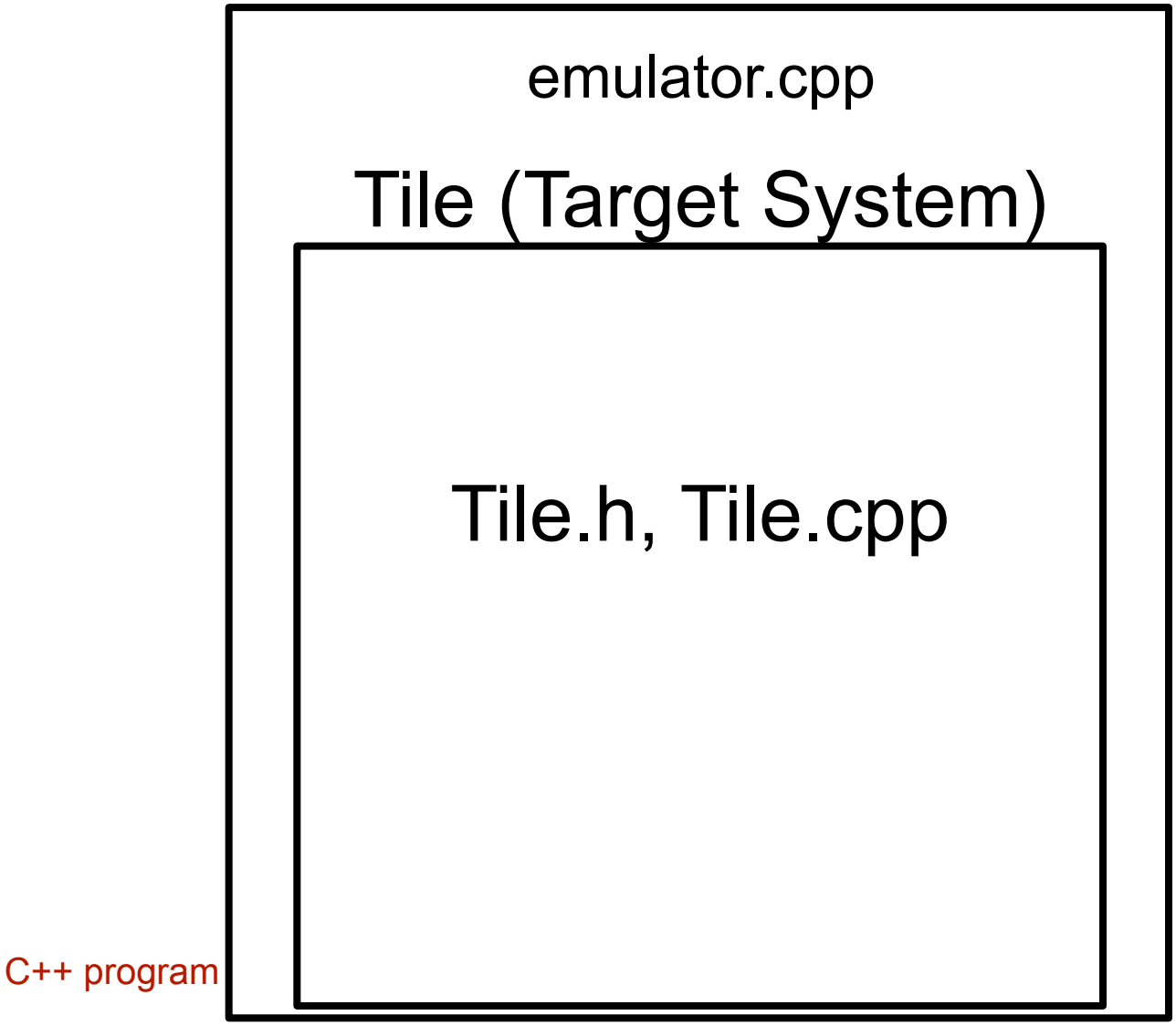
A Lab 1 Chisel Processor

- Entire "Tile" is described in Chisel code



Except micro-coded processor, which has only one memory

A Lab 1 Chisel Processor



Demo time!

- Add tools to your path

  $ source ~cs152/tools/cs152.bashrc

- Copy Lab Files

  $ cp -R ~cs152/Lab1 ./Lab1

- Build a Chisel Processor, Compile Simulator, Run all Tests & Benchmarks

  $ cd ./Lab1/chisel

  $make run-emulator

Build Another Processor

```
$ cd ./Lab1/chisel
$ export MK_TARGET_PROC=rv32_ucode
$make run-emulator
```

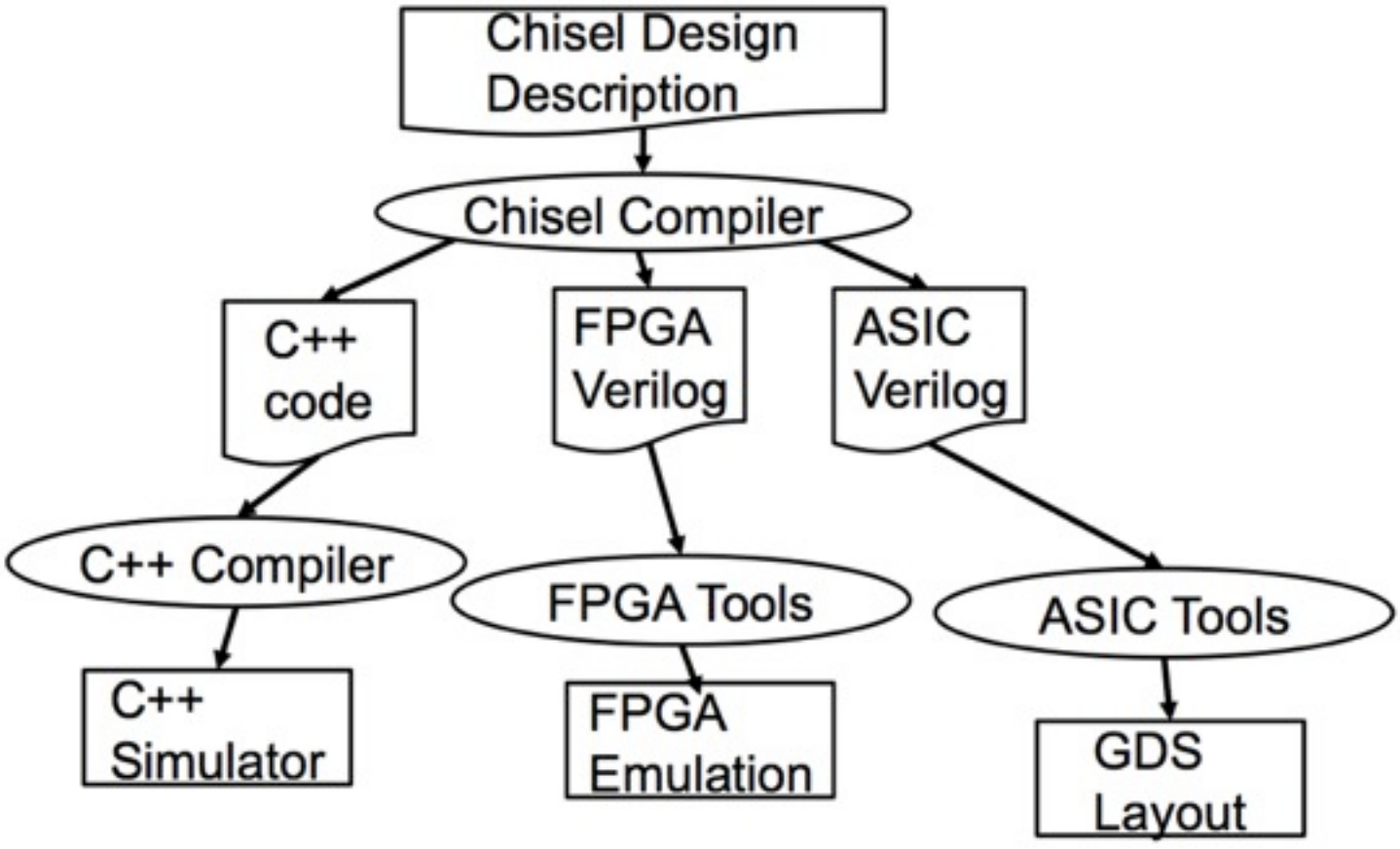
(other options)

```
$ export MK_TARGET_PROC=rv32_2stage
$ export MK_TARGET_PROC=rv32_5stage
```

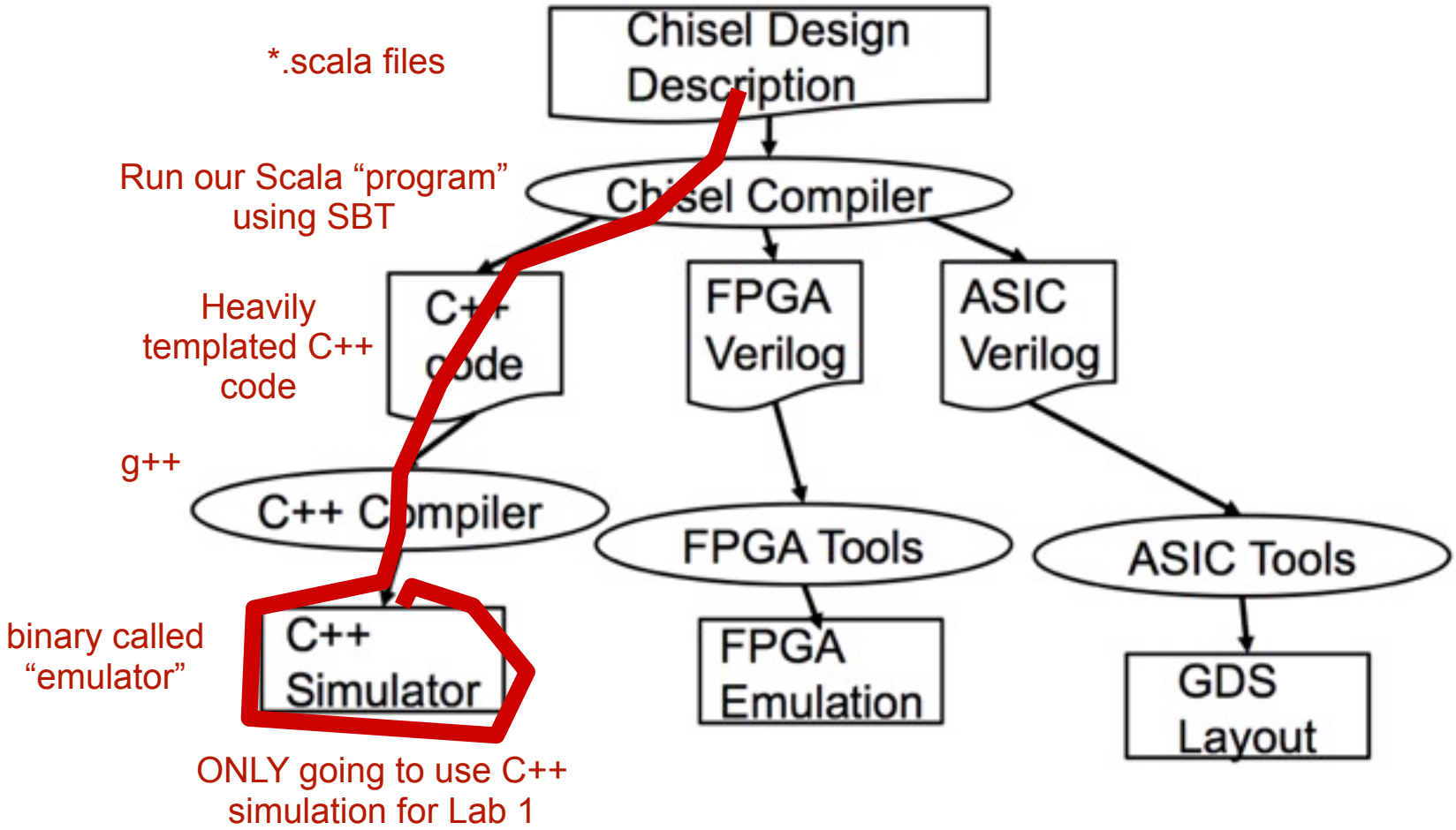
Chisel

- A new HCL language
  - designed for GENERATING hardware, not simulating it!
  - Verilog , others, were designed to simulate hardware, NOT to build hardware.
    - Thus Verilog can be very difficult to use in creating hardware
- Chisel is "embedded in Scala"
  - A Chisel processor is actually a legal Scala program
    - output of Chisel "program" is either Verilog or C++ code, that describe the processor at the cycle-accurate level

Chisel


Chisel Design Description
Chisel Compiler
C++ code
FPGA Verilog
ASIC Verilog
C++ Compiler
FPGA Tools
ASIC Tools
C++ Simulator
FPGA Emulation
GDS Layout

Chisel

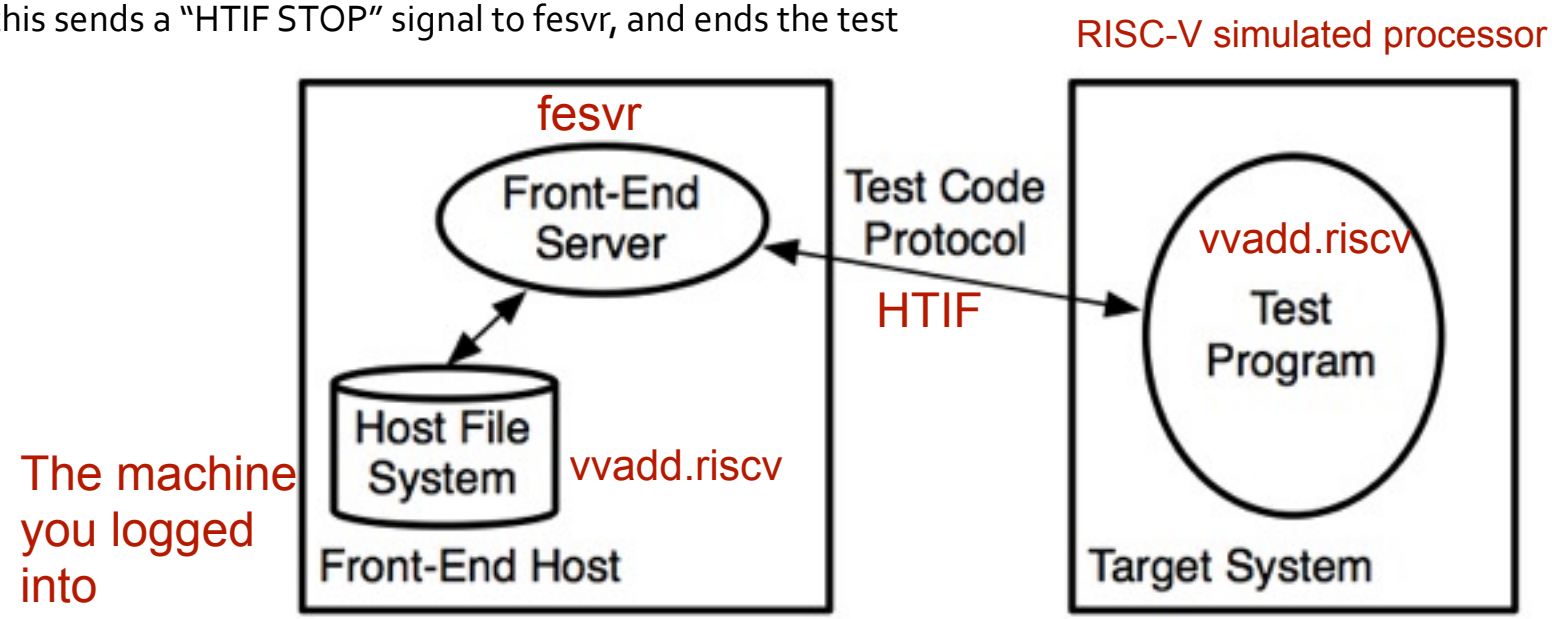

*.scala files
Chisel Design Description
Run our Scala "program" using SBT
Chisel Compiler
Heavily templated C++ code
C++ code
FPGA Verilog
ASIC Verilog
g++
C++ Compiler
FPGA Tools
ASIC Tools
binary called "emulator"
C++ Simulator
FPGA Emulation
GDS Layout
ONLY going to use C++ simulation for Lab 1

RISC-V Tool-chain

- riscv-gcc
  - cross-compiles binaries to RISC-V
  - C and assembly code
- riscv-objdump
  - disassembles binaries into assembly code
  - gives you program address information, register allocations, etc.!
- fesvr
  - front-end server, call this to run RISC-V binaries
  - fesvr -testrun ~cs152/install/riscv-bmarks/vvadd.riscv
    - runs using the riscv-isa-run ISA simulator (NOT cycle-accurate)
    - -d option to give instruction-by-instruction trace
  - fesvr -c -testrun ~cs152/install/riscv-bmarks/vvadd.riscv
    - executes using your Chisel simulator (cycle-accurate)
      - Actually executes a binary called "emulator" in your current directory

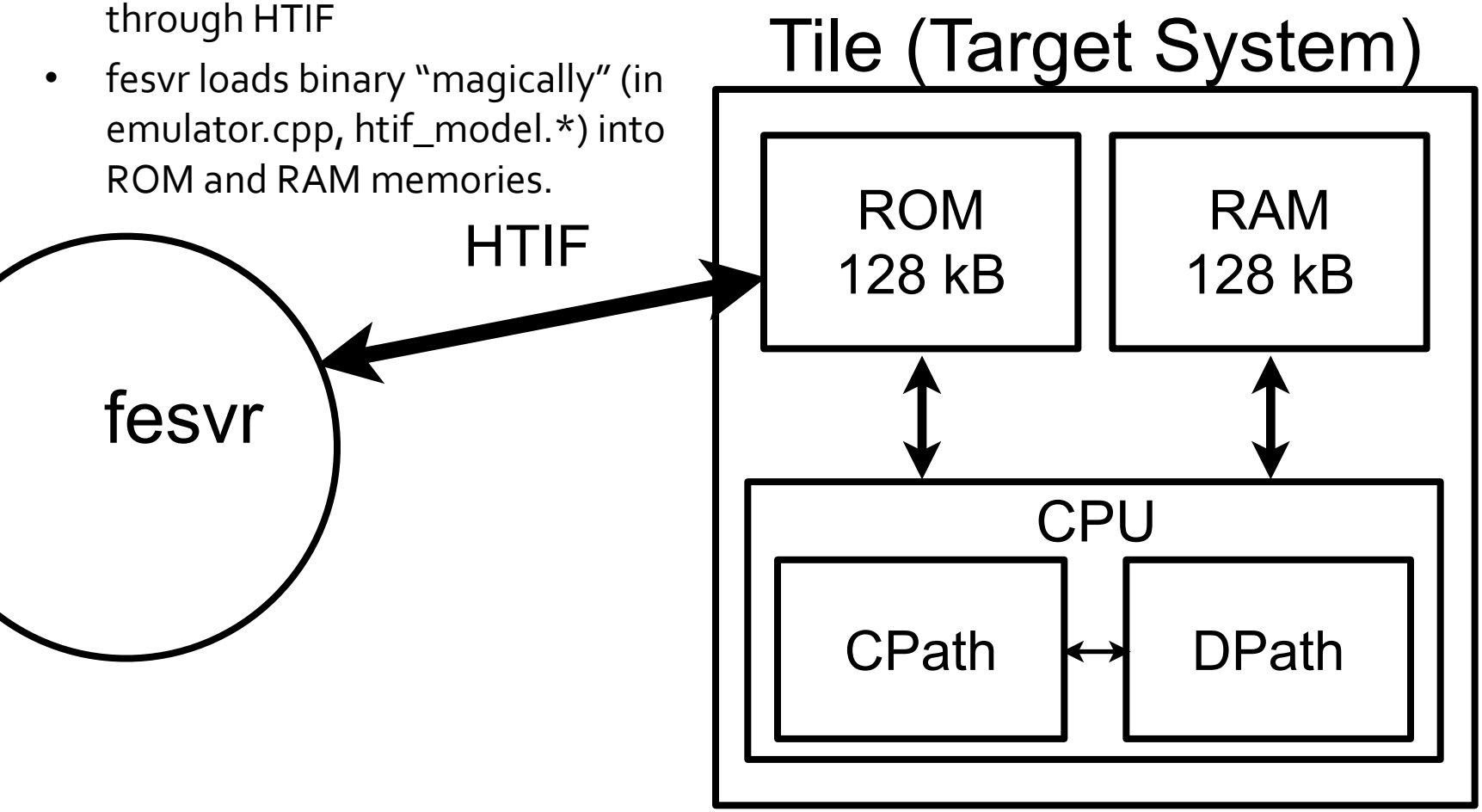
RISC-V "fesvr" (Front-End Server)

- fesvr
  - runs on the HOST machine (i.e., the machine you logged into)
  - loads RISC-V binary off the HOST filesystem
  - opens your C++ simulator in another process, uses sockets to communicate
  - sends binary to Target System using "HTIF" (Host-Target Interface)
  - Once finished, sends the "HTIF START" signal.
  - Your Chisel processor (the Target) sets TOHOST register to non-zero when finished
    - this sends a "HTIF STOP" signal to fesvr, and ends the test



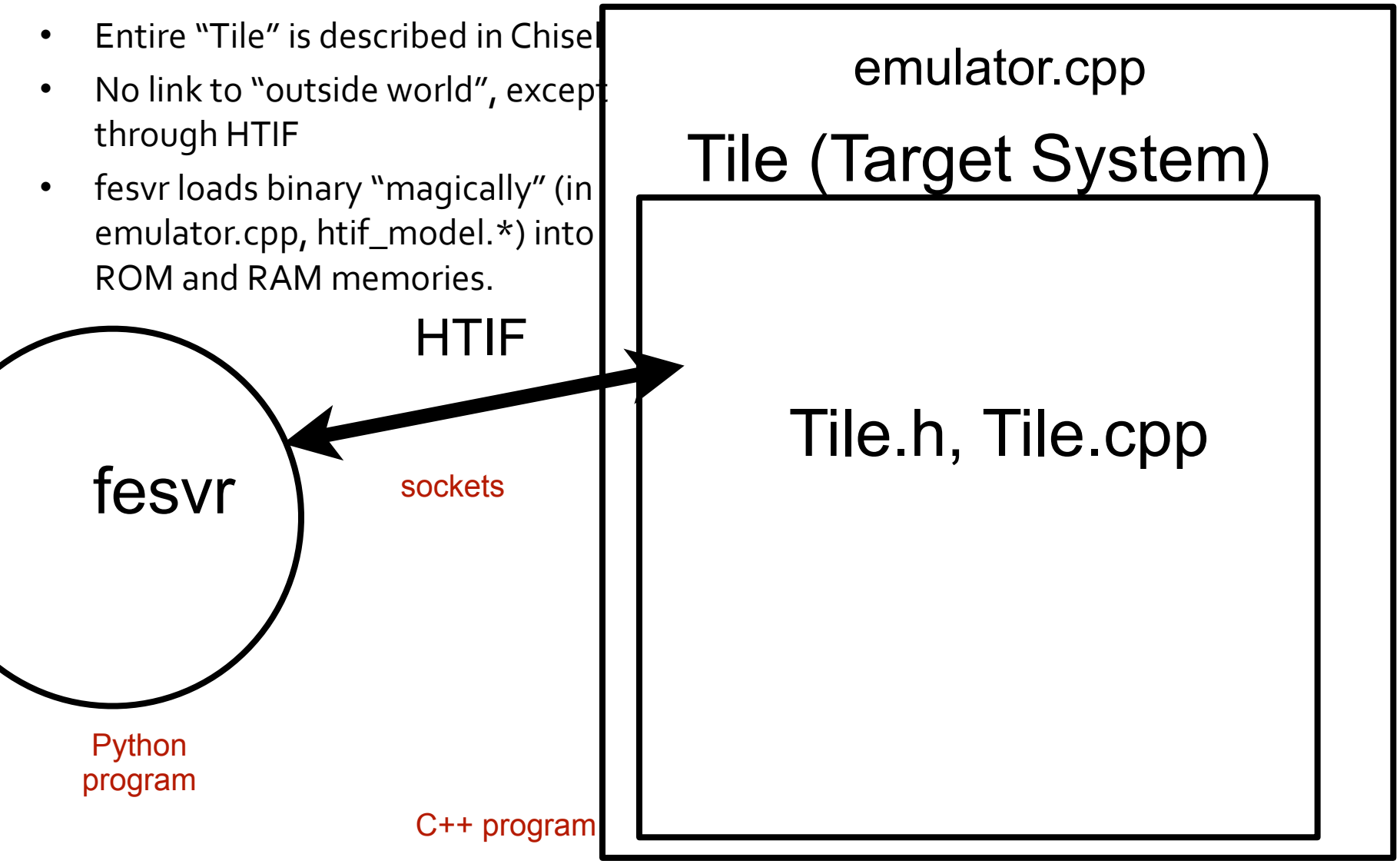
A Lab 1 Chisel Processor

- Entire "Tile" is described in Chisel
- No link to "outside world", except through HTIF
- fesvr loads binary "magically" (in emulator.cpp, htif_model.*) into ROM and RAM memories.



A Lab 1 Chisel Processor

- Entire "Tile" is described in Chisel
- No link to "outside world", except through HTIF
- fesvr loads binary "magically" (in emulator.cpp, htif_model.*) into ROM and RAM memories.



Questions?

- HW
- Lab
- Class Logistics