# ECE 571 – Advanced Microprocessor-Based Design Lecture 3

Vince Weaver

http://www.eece.maine.edu/~vweaver

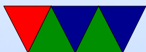
vincent.weaver@maine.edu

26 January 2016

# Announcements

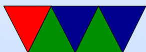
- Homework #1 was posted.
  Was an issue with valgrind section, an extra space in the command line argument, should be fixed now.

- Accounts: Log in to the Haswell machine for homework.
  Make sure you connect to port 2131.
  ece571-1 names are a bit impersonal.
  Use `passwd` to change your password.
  You can use `chfn` to change your name as it appears in `w` if you want.



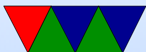
Later in semester I might give accounts on more machines.

They will have the same password as your current password on Haswell. I don't know what you changed it to. How can I do that?

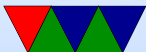


# CPUs – Central Processing Unit

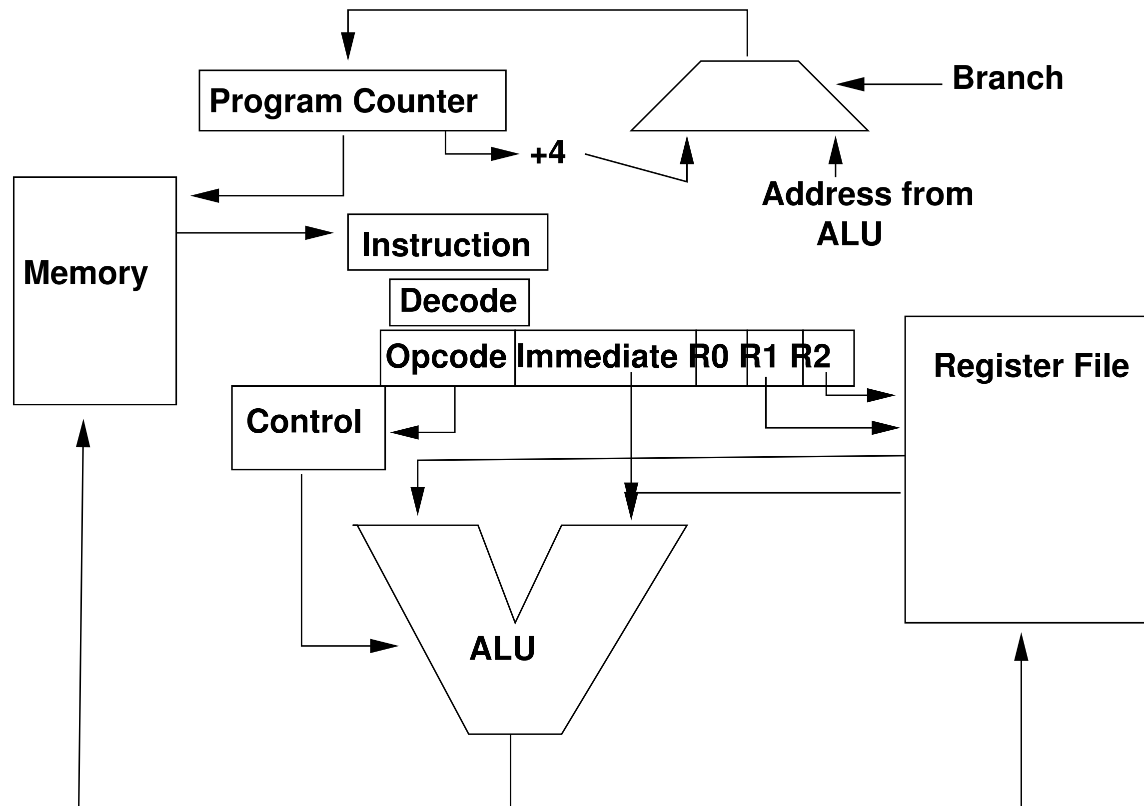
- Do the general purpose calculations in a system
- Originally big, multi-cabinet, multi-board, multi-chip
- The first “microprocessor” was one that could fit on one chip.
  Often regarded as the 4-bit Intel 4004. (history?)
- In the old days you could buy a discrete CPU, plop onto circuit board, hook up some memory and a terminal, and you had a computer.



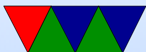
- These days things are a lot more complex.



# Simple CPU

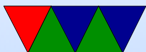



- Program Counter / Instruction Pointer points to next

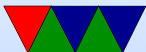


instruction
Increments each clock and loads next instruction from memory. If a branch instead loads new address if branch taken.

- Instruction is decoded. Opcode (says what type of instruction), Registers to use, possibly an immediate value.
- Opcode goes to control (usually a PLA) that splits up signals to all the functional units and tells them what to do (what kind of operation, whether to read or write, to branch or not, etc)

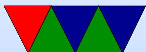


- Source register values are read from register file and fed to ALU
- ALU does math/logic based on control
- Result written back to destination register on register file.
- If load or store instruction, then address calculated (often by ALU) and sent to memory. If load, value written to reg file, if store value to write sent out
- Once instruction is done, advance to next instruction.



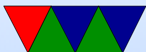
# CPU – Design decisions

- ISA (Instruction Set Architecture) – what insns do you need?
  ALU: add/sub/and/or/xor (mul and divide? many do not have)
  memory: load/store
  branch: branch if zero/not zero
  shift?
  nop? often a pseudo-op
  syscall?



compare (can be implemented with subtract)
Lots of other stuff *can* be added. Floating Point. String copy. Predicated/conditional execution. Crazy polynomial/vector insns.

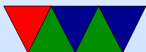
- Other decisions: how many arguments to opcode? 2 or 3?
- Flags register?
- Number of registers? 1/3/8/16/32/128/windowed?
- Decode logic: Fixed-width 4-byte (32-bit) instructions? Completely variable sized instructions (x86 1-15 bytes?) VLIW (3-instructions in a 128-bit package?)



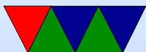
Embedded (THUMB, THUMB2) mostly 16-bit (Code Density)

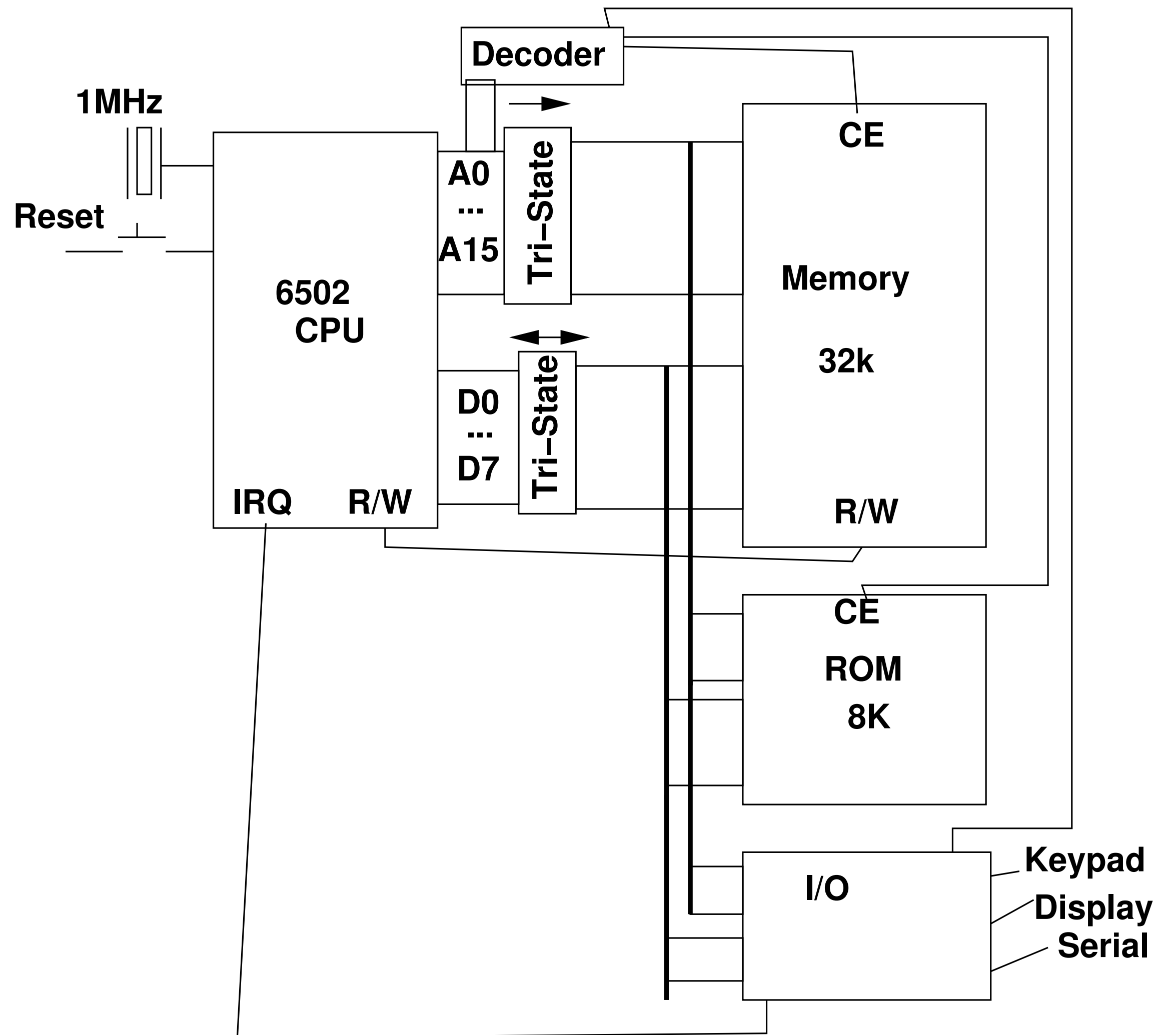
- RISC vs CISC
- Big or Little Endian?
- Bitsize (4, 8, 16, 32, 64 bit?)

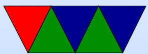
  What does that mean? Size of registers? ALU? Memory Address Range? Data bus width? Complex issue.



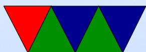
# Simple Computer



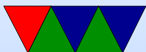

Decoder
1MHz
Reset
6502
CPU
A0
...
A15
Tri–State
D0
...
D7
Tri–State
IRQ
R/W
CE
Memory
32k
R/W
CE
ROM
8K
I/O
Keypad
Display
Serial



- Clock crystal keeps everything in sync (can you run without clock? Yes, asynchronous chips, harder to design)
- Reset button to restart things, start PC at known address
- Address bus, addresses are put out. 16-bit address space, 16 pins, $2^{16}$ (64k) addresses.
  This is used to address instructions *and* data
  Usually tri-state buffers are used to protect CPU pins and also allow multiple devices to drive address bus if needed
- Data bus: bi-directional (read/write)

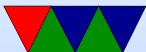


- To read memory: CPU puts address on address bus, says want to read. Decoder logic enables proper device. Device decodes address, finds 8-bit value, puts it on data bus. CPU latches the result and does whatever with it (puts in instruction buffer, puts in register)
- To write memory: CPU puts data on data bus, address on address bus, sets write signal.
- Reading from ROM much like RAM, only you can't write it
- Memory-mapped I/O, the device is enabled by decoder when address matches. Puts data on data bus just like



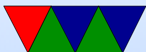
RAM would.
If I/O wants CPU attention it can pull an IRQ line to request interrupt. Otherwise CPU must poll.

- I show a 6502 CPU in example. Simple CPU, found in Apple II, Commodore, NES, many others. Designed in part by UMaine alum Chuck Peddle. Not often used for quick designs like shown because the clock circuitry was quite complex (but better than say the 8080 which needed all kinds of crazy voltages).



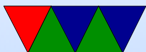
# Design Contraints

- Number of pins
  - DIP (dual inline package): 4004 = 16, z80/6502/8080/8086 = 40
  - PGA (pin grid array): Pentium = 273
  - LGA (land grid array) pins on socket not chip): Sandybridge = 1000+

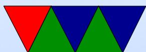


# More Complex Early computers

- Original IBM PC
- Additional helper chips to 8086. Keyboard controller, interrupt controller, DMA controller (did memory refresh, etc), programmable interval timer
- ISA system bus, more or less just exposed CPU address/data bus to slot connectors
- Dynamic memory
- 8086 had separate I/O port space
- Memory too slow, had wait states



- 8086 was full 16-bit CPU. PC uses 8088 which had only 8-bit data bus (but same ISA!). Also 24-bit address bus, played games to address properly.



# Morern Systems Even More Complex

- PCI bus
- North/South Bridges
- Everything on SoC
- Fast memory much more complex
- Everything else we are going to learn about in this class.