ECE 571 – Advanced Microprocessor-Based Design Lecture 4

Vince Weaver

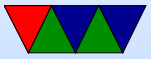
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

11 September 2024

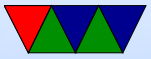
Announcements

- Homework #1 due Friday



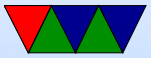
Last Time

- Briefly discussed the bare minimum instructions needed on a RISC-like CPU to have a decent computer



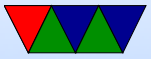
ISA Extensions

- Lots of other stuff *can* be added
- Bit manipulation
- Dedicated stack instructions (push/pop)
- Floating Point
- DSP instructions
- String copy
- Vector instructions (MMX/SSE/AVX/NEON/SVE)
- Crazy polynomial/vector insns



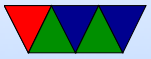
System Manipulation

- Syscall to call into kernel
- Setup of things like IRQ/DMA
- Special system registers, Model Specific Registers (MSRs)
  - CPU detection (cpuid)
  - Hardware performance counters
  - Cache config



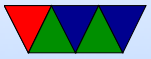
Other ISA Decisions

- How many arguments to opcode? 2 or 3?
- Is there a Flags register? Maybe comparison-result registers?
- Predicated/conditional execution
- Number of registers? 1/3/8/16/32/128/windowed?
- Special registers (Zero Register, Link Register)
- Branch or Memory Delay Slots
- RISC vs CISC
- Big or Little Endian?

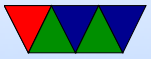


Other ISA Decisions – Bitsize

- Bitsize (4, 8, 16, 32, 64 bit?) (or 18/36 bit?)
- What does that mean? Complex issue.
  - Size of registers?
  - ALU?
  - Memory Address
  - Range?
  - Data bus width?
- Internal vs External (4-but ALU on z80, AMD Zen double-pumped 256 bit for AVX 512, bitslice

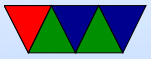


architectures



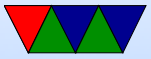
ISA Encoding

- RISC: Fixed-width 4-byte (32-bit) instructions?
- CISC: Completely variable sized instructions (x86 1-15 bytes?)
- VLIW: (3-instructions in a 128-bit package?)
- Embedded (THUMB, THUMB2) mostly 16-bit (Code Density)

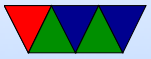


Microcoding

- Implementing complex CISC instructions in verilog/transisto complex
- Instead, have simple specialized RISC-like core that can be programmed
- Microprogramming, instructions unpacked into sets of instructions
  Simple might be 1:1 but complex might end up tens to hundreds
- 8086 has been reverse engineered

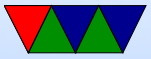


- Modern x86 has uops

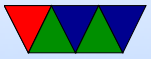


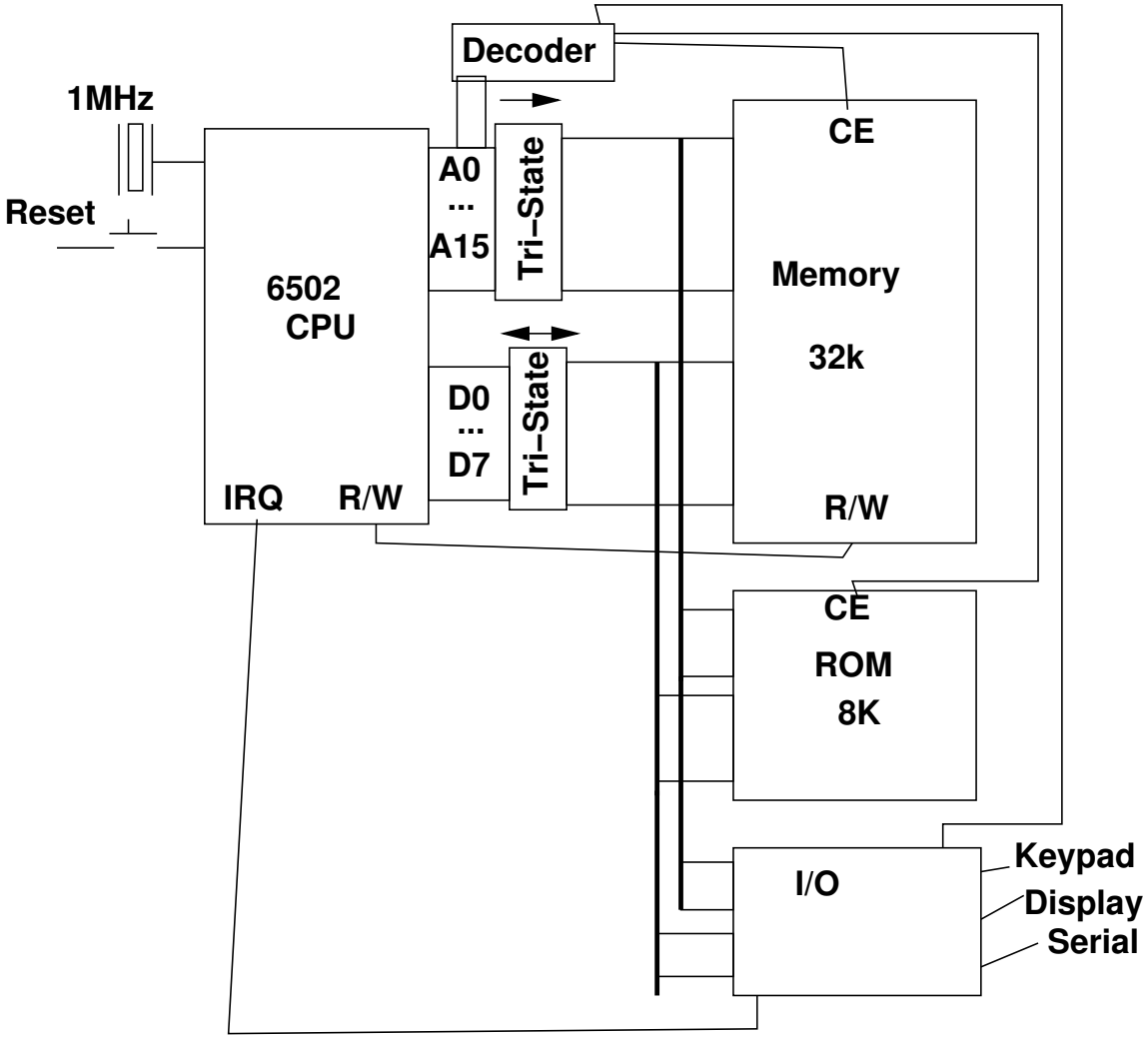
Other Design Constraints

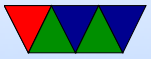
- Number of pins
  - DIP (dual inline package): 4004 = 16, z80/6502/8080/8086 = 40
  - PGA (pin grid array): Pentium = 273
  - LGA (land grid array) pins on socket not chip): Sandybridge = 1000+
- Power
  - Original ARM 1W so could use cheaper plastic vs ceramic case (ended up 0.1W)



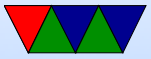
Simple Computer Redux



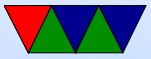

Decoder
1MHz
Reset
6502
CPU
IRQ
R/W
A0
...
A15
Tri–State
D0
...
D7
Tri–State
CE
Memory
32k
R/W
CE
ROM
8K
I/O
Keypad
Display
Serial



- 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)



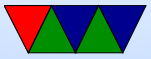
- To read memory: CPU puts address on address bus, says want to read. Decoder logic enables proper device. Device decodes address, finds 8-bit value, puts it on data bus. CPU latches the result and does whatever with it (puts in instruction buffer, puts in register)
- To write memory: CPU puts data on 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.

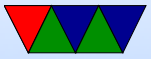
If I/O wants CPU attention it can pull an IRQ line to request interrupt. Otherwise CPU must poll.

- 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).



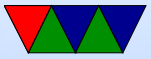
How Are Modern Systems Different?

- A lot of the I/O and memory controller pushed onto chip
- No Address/Data busses anymore.
- Memory is almost like a network/packet thing where addresses and data sent out serially
- Same with expansion, like USB or PCIe

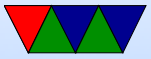


More Complex Early computers

- 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

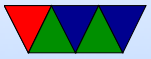


- 8086 was full 16-bit CPU. PC uses 8088 which had only 8-bit data bus (but same ISA!). Also 24-bit address bus, played games to address properly.



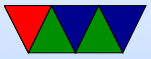
Modern Systems Even More Complex

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



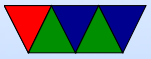
On simple processors often take multiple cycles to finish

- 6502, 1MHz, instructions generally 1 cycle per mem access
  - $69,$0A - adc #10 - 2 cycles (add immediate)
  - $65,$10 - adc $10 - 3 cycles (add zero page)
  - $70,$34,$12 - adc $1234,X - 4+ cycles (add absolute indexed)
    can take extra cycle if wraps page (carry from add)



IPC Metric

- Instructions per Cycle
- Higher is better
- Inverse of CPI (cycles per instruction)



How can we increase IPC?