ECE 471 – Embedded Systems
Lecture 1

Vince Weaver

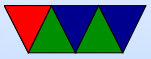
http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

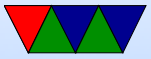
3 September 2013

Introduction

- Distribute and go over syllabus



Embedded Systems

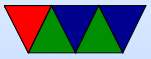


What is an embedded system?

- Embedded. Inside of something.
  Traditionally fixed-purpose.
  Why? You can optimize. For cost, for power, for size, for reliability, for performance.

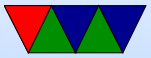
- Resource constrained. Small CPU, Memory, I/O, Bandwidth

- Often real-time constraints.



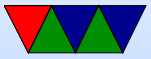
What are some embedded systems?

- Cellphone (though lines blurring, general purpose)
- Vehicles (Cars/Airplanes)
- Appliances (TVs, Washers), Medical Equipment
- Space Probes

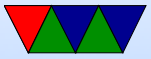


What Size CPU/Memory?

- Anything from 8-bit/tiny RAM to 32-bit 1GHz 1GB
- Performance has greatly improved over the years. ARM Cortex A9 in an iPad2 scores same on Linpack as an early Cray supercomputer



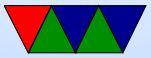
Pushing the Limits



What Processors Commonly Used?

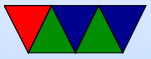
As reported by IDC at the SMART Technology conference in San Francisco for 2011

- ARM 71%
- MIPS 11%
- Other 9%
- x86 8% (at least Intel's desperately trying)
- Power 2%



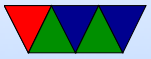
We'll Mostly Use ARM in this Class

- Widely used
- You'll see if it you move to industry
- Other classes in ECE are moving to it

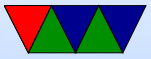


Brief Computer Architecture Review

- Embedded processors used to use simple processors
- Over the year, due to Moore's Law, more complex processors have entered the embedded space
- This involves many tradeoffs. A lot of processors trade complexity for speed, and often become non-deterministic. This impacts real-time operation, as you cannot predict how long an operation will take.
- Embedded system designers thus have to learn more



about the underlying hardware.

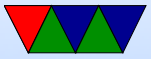


Instruction Set Architecture

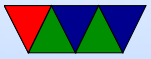
- RISC: Reduced Instruction Set Computer
  Small set of instructions to make processor design simpler. Usually fixed-length instructions, load/store

- CISC: Complex Instruction Set Computer
  Wide ranging complicated instructions; have complicated CPU decode circuitry. Often variable length instructions. Often allow operating on memory directly.

- VLIW: Very Long Instruction Word



Instructions come in long “bundles”, often 3 at a time. Cannot have dependencies; may have to fill with “nops”. Allows compiler to exploit inherit parallelism in code (most modern CPUs do this in hardware instead, VLIW puts this complexity in software).

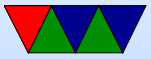


CISC/RISC/VLIW Examples

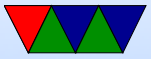
- MIPS is RISC: roughly only 40 integer instructions , (more if you include FP)

- x86 is CISC: hundreds of complicated instructions, including ones that access memory, auto-increment registers, have complex shift/add address modes

- Hybrid: ARM or Power started out RISC but have accumulated more complicated instructions over time

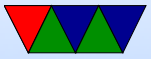


- x86, while CISC externally, internally decodes to a RISC-like code before executing



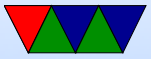
ISA / Code Density

- ISA affects code density, which can be important in embedded systems
- Dense code takes up less space in caches (good for small systems), less space in storage (smaller/cheaper disk or flash)
- Dense code often also is more complex to decode, so may lead to larger/hotter/power-drawing processors.



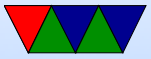
Bit-size

- 8-bit, 16-bit, 32-bit, 64-bit
- Generally refers to register size, but also is tied to memory addressing size
- Endianess can be an issue



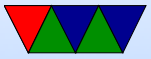
Multiprocessor

- Moore's law no longer fights for frequency, instead you get more cores per CPU
- Multi-core systems are starting to appear in embedded systems
- SMP/CMP/SMT



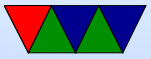
Processor Type

- In-order Processors – Old 8-bits
- Super-scalar – multiple instructions "in-flight" at once.
- Multi-issue – duplicated functional units and multiple instructions can execute simultaneously
  Original Pentium
- Out-of-order – instructions can execute when read, even out of order, with correct program behavior guaranteed



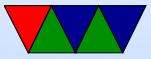
at retirement

Pentium Pro and newer, Arm Cortex A8 and newer



Caches

- Fast memory close to the processor. If your data/code is in cache it will execute up to 100x faster. If it doesn't fit in cache, goes slowly.
- Exploits temporal (nearby in time of access) and spatial (nearby in memory location) locality
- Usually values are automatically moved into cache, you don't have to do anything manually.
- Memory Wall – Memory speeds have not kept up with



CPU speeds over the years.

- L1/L2/L3
- Instruction/Data/Unified
- Coherency