ECE 498 – Linux Assembly Language Lecture 9

Vince Weaver

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

vincent.weaver@maine.edu

13 December 2012

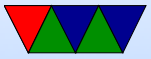
SNES Followup

- The SNES demo now has sprite support working.

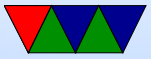
- Why does

```
.word xpos,1
ldx #$0
stx xpos
ldx xpos
```

not have 0 in result? Answer: we are executing out of ROM. Be sure to put variables in `.data`

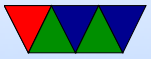


- Linux supports running out of ROM too, “Execute in Place” (XIP).
  ROM/disk is slow, which is why on Linux typically you copy code to RAM before executing. Even on PC’s the BIOS is often “shadowed” before running.



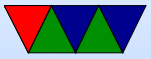
SNES HW#3 Walkthrough

- Code almost enough for a game, (something simple like a breakout clone).
- Reads keyboard input, updates X position of sprite, uses DMA to copy sprite data to memory.
- VBlank... called 30fps (or 60? need to check).



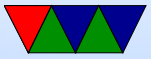
Project Followup

- Bug in project description. Result should look a bit different; pseudo-code updated to mention we have to saturate to 0/255.
- Changes in project description in red in updated pdf on website.

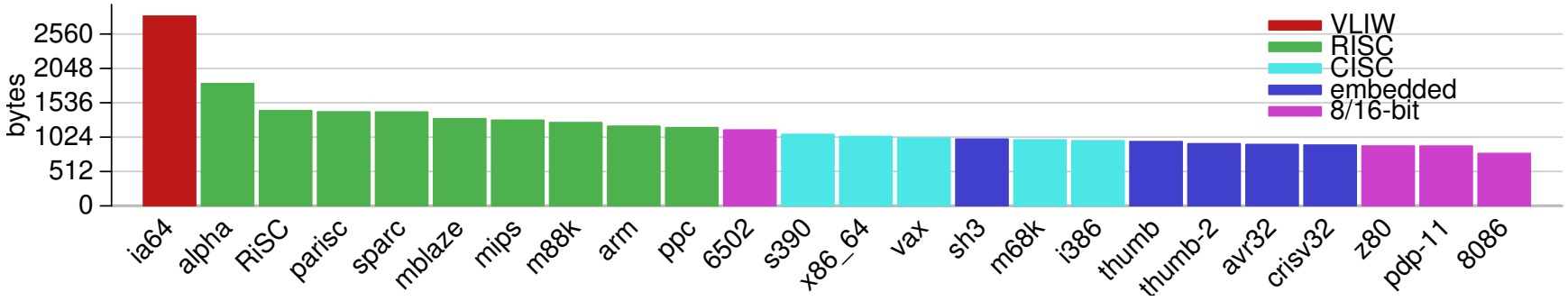


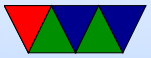
Code Density

- Overview from my `ll` ICCD'09 paper
- Show code density for variety of architectures, recently added Thumb-2 support.
- Shows overall size, though not a fair comparison due to operating system differences on non-Linux machines



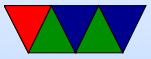
Code Density – overall



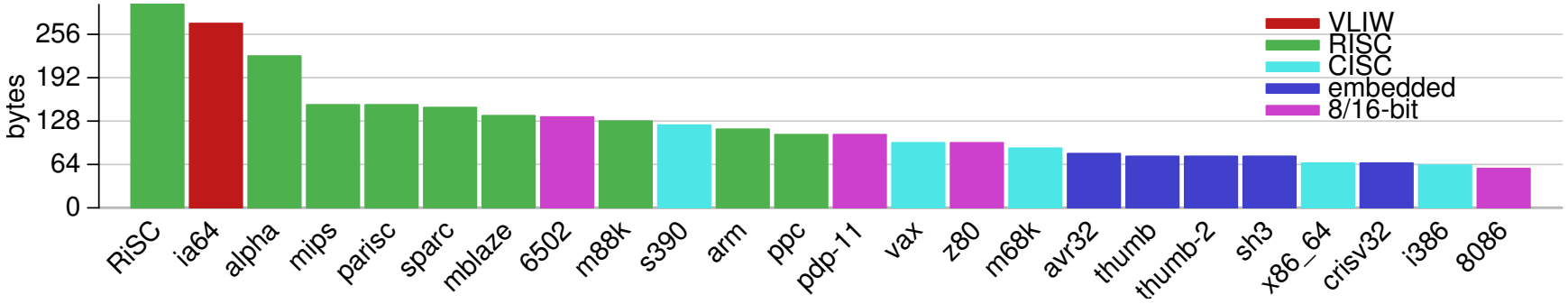


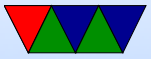
lzss compression

- Printing routine uses lzss compression
- Might be more representative of potential code density



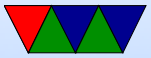
Code Density – lzss





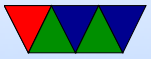
Code Optimization

- When you hit the wall, time to change algorithms.
- Research for better algorithms. Google, papers, *Hacker's Delight* book.

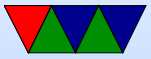


Ways to Divide by 10

- hardware div instruction
- BCD, shift by 8 (most machines don't have BCD in hardware)
- 32x32 with 64-bit result multiply reciprocal
- multiply by fraction
- lots of shifts (useful if no multiply instruction)

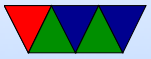


- iterative subtraction (useful if no mul either)



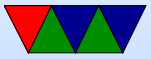
Performance Analysis

- using `time` command. Real time is wallclock, user is how much actual code used (ignores other processes on system), sys is how much kernel used
- can use `perf` command to use hardware performance counters

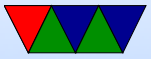


`perf` tool

- `perf stat COMMAND`
  where COMMAND is the name of your program prints summary of various OS and hardware counts
- `perf stat -e instructions,cycles COMMAND`
  prints results of just instructions and cycles
- `perf stat -e instructions:u,cycles:u COMMAND`
  the u indicates only measure user time, not kernel too
- There are many fancier things you can do with perf;

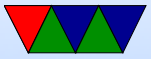


there isn't much good documentation for it



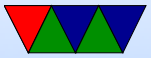
x86 div/mod by 10 analysis

- integer div – smallest code, but slower
- high-mul by (8/10)*2**32 – faster almost by factor of 2 despite twice as many instructions

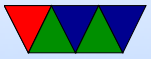


arm div by 10 analysis

- div – (not available on most ARM chips)
- high-mul – fastest, but `umull` instruction not available on THUMB
- shifts – slower than high-mul, (needed on armv4 with no `umull` instruction)



C Compiler – 7 million div/10



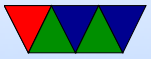
C Compiler – x86

icc	`mul 0xcccccccccd, >>3, lea`	5.00s
gcc -O0	`mul 0xcccccccccd, >>3, shift/add`	5.00s
gcc -O2	`mul 0xcccccccccd, >>3, lea`	5.00s
gcc -Os	`div`	5.10s

1.5% speedup

Overhead comes from printing? Why must I print?

simple way to keep the compiler from optimizing away the divide (if I don't output the result, then it doesn't need to be executed)



C Compiler – arm

gcc -O0	`umull 0xcccccccd, >>3, again, shift`	10.02s
gcc -O2	`umull 0xcccccccd, >>3, shift`	9.90s
gcc -Os	`__aeabi_uidivmod`	11.40s

calls __aeabi_uidiv and subtracts, shift/subtract

15% speedup