ECE 471 – Embedded Systems
Lecture 7

Vince Weaver

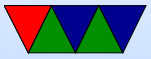
http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

24 September 2013

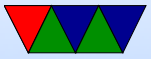
Announcements

- HW#2 is delayed, working on hardware
- Read Chapter 11 in Textbook

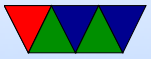


HW Review

- The "state of the art" space-rated processor is the Rad750, a 32-bit PowerPC running at 133MHz. It uses 10 watts of power and costs $200,000.
- Great things done with 16-bits and less. Voyager probes.
- Apollo Computer. Interesting book on their Block-1 computer. The story goes the ROM was hand-sewn by old ladies.
- C-compilers may produce bloated executables now, but

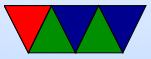


back in 1970 C was designed to run on machines we'd consider small 16-bit embedded systems today.



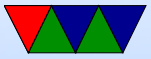
ARM Instruction Set Encodings

- ARM – 32 bit encoding
- THUMB – 16 bit encoding
- THUMB-2 – THUMB extended with 32-bit instructions
- THUMB-EE – some extensions for running in JIT runtime
- AARCH64 – 64 bit. Just released

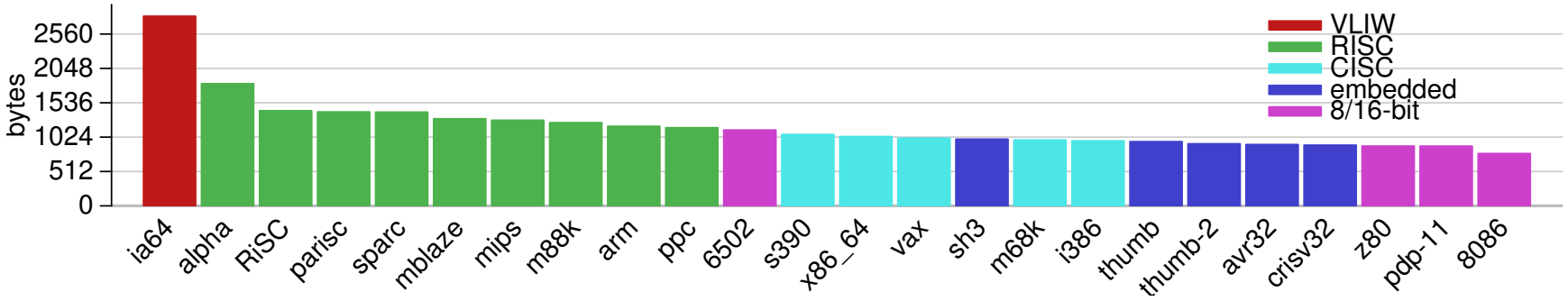


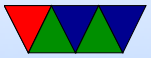
Code Density

- 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



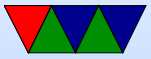
Code Density – overall



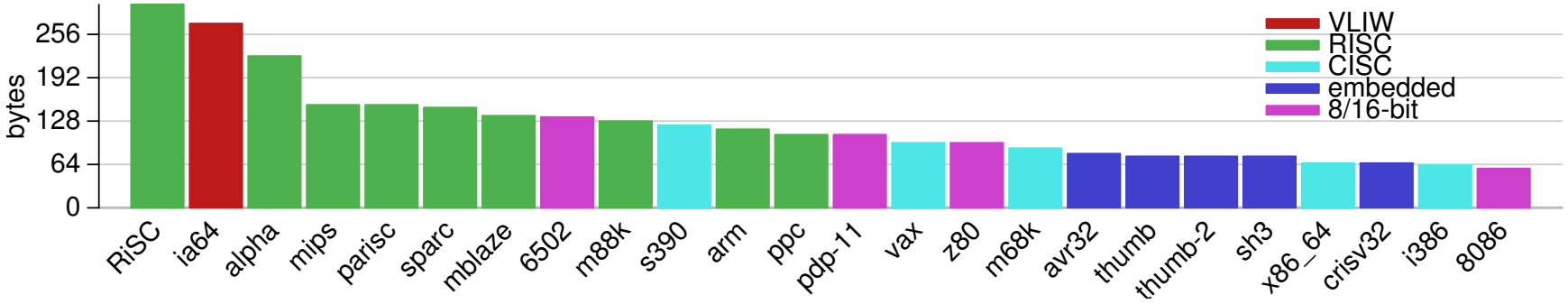


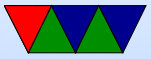
lzss compression

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



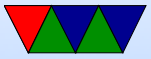
Code Density – lzss





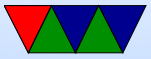
THUMB

- Most instructions length 16-bit (a few 32-bit)
- Only r0-r7 accessible normally
  `add`, `cmp`, `mov` can access high regs
- Some operands (sp, lr, pc) implicit
  Can't always update sp or pc anymore.
- No prefix/conditional execution
- Only two arguments to opcodes

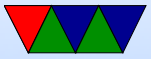


(some exceptions for small constants: `add r0,r1,#1`)

- 8-bit constants rather than 12-bit
- Limited addressing modes: [rn,rm], [rn,#imm], [pc|sp,#imm]
- No shift parameter ALU instructions
- Makes assumptions about "S" setting flags
  (gas doesn't let you superfluously set it, causing problems if you naively move code to THUMB-2)

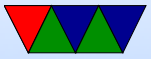


- new push/pop instructions (subset of ldm/stm), neg (to negate), asr,lsl,lsr,ror, bic (logic bit clear)

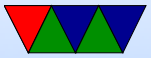


THUMB/ARM interworking

- See `print_string_armthumb.s`
- BX/BLX instruction to switch mode.
  If target is a label, *always* switchmode
  If target is a register, low bit of 1 means THUMB, 0 means ARM
- Can also switch modes with `ldrm`, `ldm`, or `pop` with PC as a destination
  (on armv7 can enter with ALU op with PC destination)

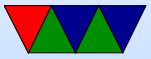


- Can use `.thumb` directive, `.arm` for 32-bit.



THUMB-2

- Extension of THUMB to have both 16-bit and 32-bit instructions
- 32-bit instructions *not* standard 32-bit ARM instructions. It's a new encoding that allows an instruction to be 32-bit if needed.
- Most 32-bit ARM instructions have 32-bit THUMB-2 equivalents *except* ones that use conditional execution. The `it` instruction was added to handle this.

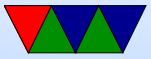


- `rsc` (reverse subtract with carry) removed

- Shifts in ALU instructions are by constant, cannot shift by register like in arm32

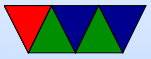
- THUMB-2 code can assemble to either ARM-32 or THUMB2
  The assembly language is compatible.
  Common code can be written and output changed at time of assembly.

- Instructions have “wide” and “narrow” encoding.



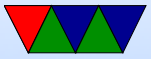
Can force this (`add.w` vs `add.n`).

- Need to properly indicate “s” (set flags).
  Regular THUMB this is assumed.



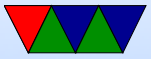
THUMB-2 Coding

- See `test_thumb2.s`
- Use `.syntax unified` at beginning of code
- Use `.arm` or `.thumb` to specify mode



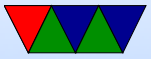
New THUMB-2 Instructions

- BFI – bit field insert
- RBIT – reverse bits
- movw/movt – 16 bit immediate loads
- TB – table branch
- IT (if/then)
- cbz – compare and branch if zero; only jumps forward



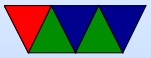
Thumb-2 12-bit immediates

```
top 4 bits 0000 -- 00000000 00000000 00000000 abcdefgh
           0001 -- 00000000 abcdefgh 00000000 abcdefgh
           0010 -- abcdefgh 00000000 abcdefgh 00000000
           0011 -- abcdefgh abcdefgh abcdefgh abcdefgh
           0100 -- 1bcdedfh 00000000 00000000 00000000
            ...
           1111 -- 00000000 00000000 00000001 bcdefgh0
```



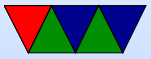
Compiler

- `gcc -S hello_world.c`
  On gumstix board creates arm32
- `gcc -S -march=armv5t -mthumb hello_world.c`
  Creates THUMB
- `-mthumb -march=armv7-a` Creates THUMB2



IT (If/Then) Instruction

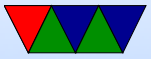
- Allows limited conditional execution in THUMB-2 mode.
- The directive is optional (and ignored in ARM32)
  the assembler can (in-theory) auto-generate the IT instruction
- Limit of 4 instructions



Example Code

```
it cc
addcc r1,r2

itete cc
addcc r1,r2
addcs r1,r2
addcc r1,r2
addcs r1,r2
```



`ll` Example Code

```
        ittt cs @ If CS Then Next plus CS for next 3
discrete_char:
        ldrbcs  r4,[r3]       @ load a byte
        addcs   r3,#1         @ increment pointer
        movcs   r6,#1         @ we set r6 to one so byte
        bcs.n   store_byte    @ and store it
offset_length:
```