# ECE 271 – Microcomputer Architecture and Applications Lecture 5

Vince Weaver

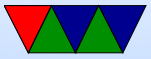
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

5 February 2019

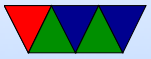
# Announcements

- Read Chapter #3 and #4 of the book.
- We have a grad TA, Colin Leary, who will be having office hours on Wednesday at 2pm.
  If earlier/later/different day might work better for everyone, let me know.
- Reminder: no food or drink in the labs

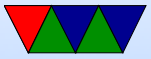


# Gitlab Update

- Hopefully it is working. Not as stable as it should be.
- Be sure you create your own ECE271 project before pushing to it
- Pushing issues if off campus or eduroam?
  For security probably blocking ssh access from off campus
  New eduroam probably not whitelisted yet
- way git works, you have a full repository/versioning locally. the push just syncs things so other people can see it

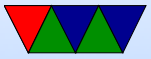


- How do ssh keys work?
  Public Key Cryptography, interesting, but could give a whole lecture on it
  We talk about this in ECE435 (Network Engineering) but that course might not be offered next year.

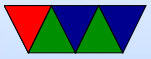


# More Lab Notes

- Don't look at provided character translation code, it's horrible
- What is the deal with uint8_t vs char?
- Something else the code does, copying data and bss segments
- Strings in C, pointers
- Commenting styles, Doxygen
- Using the predefined constants in stm32l476xx.h
- How do you make a delay? For loop? Don't forget the



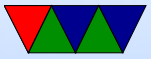
volatile.



# Thumb-2 encoding

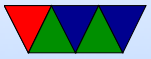
```
ADD{S}<c>.W <Rd>,<Rn>,<Rm>{,<shift>}
```

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | S | Rn | | | | 0 | imm3 | | | Rd | | imm2 | | type | | | Rm | | | | |

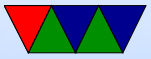


# Registers

- How are registers designed? SRAM (static RAM: flip-flops)
  Aside: main memory on a desktop/laptop is DRAM (dynamic RAM) with one transistor and a capacitor, which drains quickly and has to be constantly refreshed.
- Three ports: two output and one input
- The rules for what goes in what register are part of the ABI (Application Binary Interface)
- ARM32 registers:

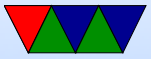


- Has 16 GP registers (more available in supervisor mode)
- r0 - r12 are general purpose
- r11 is sometimes the frame pointer (fp) [iOS uses r7]
- r13 is stack pointer (sp)
- r14 is link register (lr)
- r15 is program counter (pc)
  reading r15 usually gives PC+8
- 1 status register (more in system mode).
  **NZCVQ** (Negative, Zero, Carry, oVerflow, Saturate)



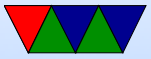
# 3-stage pipeline

- Fetch/decode/execute



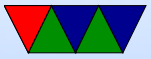
# Assembly Language: What's it good for?

- Understanding your computer at a low-level
- Shown when using a debugger
- It's the eventual target of compilers
- Operating system writers (some things not expressible in C)
- Embedded systems (code density)
- Research. Computer Architecture. Emulators/Simulators.
- Video games (or other perf critical routines, glibc, kernel, etc.)



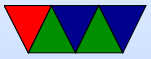
# Let's start with ALU instructions

- a=b+c;
- How is ALU designed? Adder/subtractor/logic?



# Add instruction

- `add r1,r2,r3` – r1=r2+r3
- Gets the values, adds two, stores in third

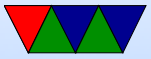


# What does an assembly line look like

- annoyingly this can vary by platform, and even by assembler program on the same platform. (could be worse, intel vs at&t on x86)
- GNU asm style:

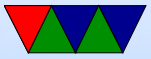
```
label: opcode  dest, src1, src2 ; comment
/* comment */
```

- Keil style:



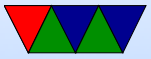
```
label
 opcode  dest, src1, src2 ; comment
```

- Label marks a point in the program. If you reference it the assembler will turn it to an address. You can do things like jump/branch/goto it. You can load from/store to it.
- The opcode or mnemonic says what you want to do. add/sub/eor, etc



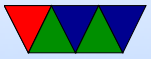
# Assembly Directives: Keil / GNU

- Put this in your code to give the assembler directions
- Things like where to reserve memory, where functions start, etc.
- Slightly different from Keil to GNU (GNU starts with a . )



# Add instruction

- `add r1,r2,r3` – r1=r2+r3
- `add r1,r2,#immediate` r1=r2+constant
  There are limits to constant size. Why?
  The thumb2 constants are exciting, will get to later



# Settings Flags

- `adds r1,r2,r3` – set condition flag
  flags NZCV
  - N = negative (how can you tell if negative?)
  - Z = zero (how can you tell if zero?)
  - C = carry (how can you tell if carry? Why is it useful?)
  - V = overflow (will get to this later), signed overflow
  - Q = saturate