ECE 271 – Microcomputer Architecture and Applications Lecture 3

Vince Weaver

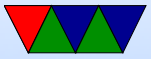
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

29 January 2019

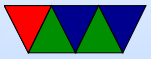
Announcements

- Read Chapter #1 of the book.
- Labs have started! Don't forget to do the pre-lab
- We'll have a grad TA with office hours, probably on Wednesday.



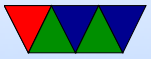
Lab Update

- There was a typo (now fixed) on the pre-lab with joystick, it's GPIOA not GPIOB
- Also you might need to use st-link to erase your boards for the first time before Keil will be able to program them.
- Notes on how the labs work:
  - Do the pre-lab before coming to lab.
    This will be checked off by the TA at the start of the lab.

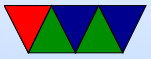


I will try to post prelab by Thursday

- Do lab during lab time.
  If you don't finish that's OK, you have until your next lab session to finish it. So you can work on it at home, just get it checked off by the next lab session.
- Be sure to comment your code! Also put a header at the top of your code with your name, date, class, and a short summary of what the code does.
- Once you have completed the lab, demo it to the TA. You will turn in the code (eventually via git).
- There will be a post-lab with a few questions. Answer

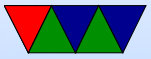


these in the README file (or maybe README.md) and include that with your assignment code.



Lab Grading Rubric

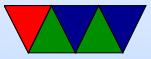
- prelab 2-points: show up for lab, complete pre-lab
- document 5-points: comment code, readme, header, git commits, C style
- functional code 5-points: code works, no warnings or errors (or at least no relevant warnings)
- lab demonstration 5-points: demo it, plus post-lab
- something cool 3-points



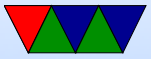
Operating Systems for the labs

- By default just use Keil on Windows
- If you are feeling adventuresome I also posted templates for Linux which you can find on the bottom of the webpage
- If you are using MacOSX you have a few options. You can try using the Linux files via homebrew. Or you can dual-boot your machine into Windows, or run Windows in a VM.

  All of these are sort of advanced and neither the TAs

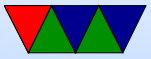


nor I can necessarily be much help if it goes wrong.



Other Lab Notes

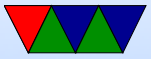
- The lab says you should put an infinite loop at end of code. Why? (to stop your program from running off the end)
  How do you make one?



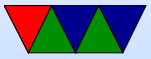
Helper Code in the Template

- There's some code that turns on the 80MHz clock to the whole chip
- There's a linker script which tells the compiler how to set up the code for the flash-image to be burned
- The beginning of the flash image is the initial stack-pointer value, which loads into the stack pointer (SP) at boot.

  What should this point to? (The *end* of memory, stacks grown down).



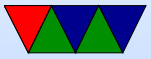
- The next set of 32-bit values in the image are interrupt vectors. One of them is the reset vector; that address is jumped to at reset time. The provided code points this to your main() function.



Coding

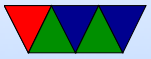
- Comment your code!
- How to comment. Which is useful?

```
i=2;  /* set i equal to 2 */
i=2;  /* set i to the LED to enable */
```

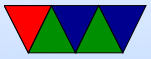


git

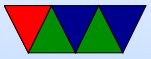
- SCM – source code management
- Digression on how Linus invented it
  CVS not an option
  bitkeeper + LM
  Never make Linus mad
- How it works. Never ask. Lots of hashes and lines and diagrams. xkcd comic?
- Create a repository. git init (or clone existing)
- Add files. Also git add if you modify



- When code is in a good stopping point, commit
- Good commit messages
  Not just "blah" or "fixed bug" or "I hate coding"
  Linux have detailed first line, then often pages of explanation
  Maybe not need that for this class.
  Should help someone auditing/maintaining code understand what code change is doing.
- Push out to server if you have one (github blah MS or gitlab)
- Benefits:

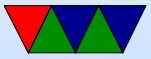


Can roll back to former version of code
When pushed out to other locations full backup
Can work together (though watch for merge)
git bisect
Try to find out why a change was made (git blame), hope for good commit message.

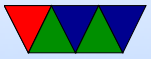


CPU clocks

- A modern CPU has lots of clocks going on
- We have code that turns on the main 80MHz clock
- There's a diagram in the manual showing all the various speeds and sub-clocks you can enable.
- On a desktop/laptop/phone someone does this for you, but here on bare-metal we have to worry about it.

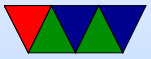


Intro to Computer Architecture



Parameters in an Architecture

- CISC vs RISC
- 8/16/32/64 bits
  How do you determine this (these days integer register/pointer size?)
- Endianess
  Write a 32 bit value 0xdeadbeef to address 0.
  What 8-bit value is in address 0?
  big-endian: byte 0 has 0xde
  little-endian: byte 0 has 0xef



- Instruction Size (fixed/variable)
  RISC traditionally has 4-byte fixed-size (easy to decode)
  x86 (CISC) is variable 1-15 bytes
  Cortex-M4 is Thumb2 so 2-4 bytes
- Load/Store
- Number of commands to opcode
- Weird: Branch Delay slot / Zero register
- Number of registers, special registers
- Flags