ECE 271 – Microcomputer Architecture and Applications Lecture 3

Vince Weaver

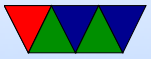
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

25 January 2022

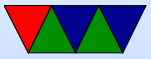
Announcements

- Read Chapter #1 of the book.
- Labs have started! Don't forget to do the pre-lab
- The course website should be visible to the outside world again.

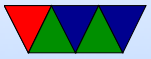


Lab Update

- 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.
  - 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.

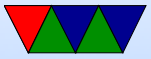


- Once you have completed the lab, demo it to the TA. You will turn in the code (eventually via git).
- There will be a post-lab with a few questions. Answer these in the README file (or maybe README.md) and include that with your assignment code.



Lab Grading Rubric

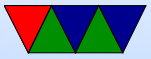
- prelab 2-points: show up for lab, complete pre-lab
- document 5-points: comment code, readme, header, git commits, C style
- functional code 5-points: code works, no warnings or errors (or at least no relevant warnings)
- lab demonstration 5-points: demo it, plus post-lab
- something cool 3-points



Operating Systems for the labs

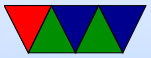
- By default just use Keil on Windows
- If you are feeling adventuresome I also posted templates for Linux which you can find on the webpage
- If you are using MacOSX you have a few options. You can try using the Linux files via homebrew. Or you can dual-boot your machine into Windows, or run Windows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



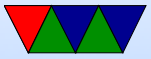
Review Masking

- The prelab has you calculate some values in advance, so you don't have to waste valuable lab time doing it
- You need to set certain bits to a value without changing the surrounding existing bits
- Recall that bitwise OR with 0 leaves the bit a same, OR with 1 sets it to 1. Bitwise AND with 0 clears a bit, AND with 1 leaves the same.
- So what you want to do is first MASK off the value using AND, then you can OR the value in (0 or 1) and



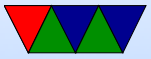
it will get set

- The term MASK is used because you are protecting the other bits from being changed, sort of how you might use a paper mask with an opening in it when painting.



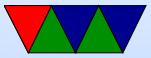
Setting the Right Values

- The hardest thing about this lab is setting the right values
- Embedded systems like this are super unforgiving
- One bit wrong and nothing will work at all
- Double check your values
- How can you tell if the hardware has the right value?



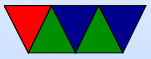
Debugging!

- How do you find out what your code is doing?
- `printf()`, but can we do that on this board?
- Use a debugger!
- Embedded boards support remote debugging, where you can control the processor execution from another machine.
- Keil debugger. gdb is common on Linux



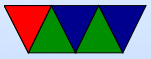
Debuggers

- Set breakpoints to stop when code hits a certain point
- Single-step through code (or assembly language)
- View register state, variable values
- Can view the hardware registers to be sure they're being loaded with proper values



Commenting your Code

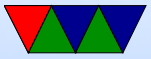
- Comment your code!
- 6 months out could someone else understand your code?
- 6 months out can *you* still understand your code?



Useful Code Comments

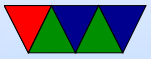
- How to comment. Which is more useful?

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



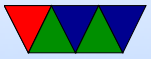
git

- 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?



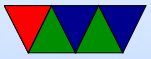
git actions

- Can have GUI or command line
- `git init` creates repository
- `git clone` clones an existing repository
- `git pull` make files up to date
- `git add` queue changes in file to repository
- `git commit` add queued changes to repository
- `git push` push changes to remote repository

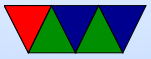


SCM concepts

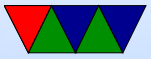
- Good commit messages
  Not just "blah" or "fixed bug" or "asdfghj" or "I hate coding"
  Linux have detailed first line, then often pages of explanation
  Maybe not need that for this class.
  Should help someone auditing/maintaining code understand what code change is doing.
- Push out to server if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



- 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.
- hashes: cryptographic hash (string of hex numbers) used to track changes. Will often see these referenced for changesets.

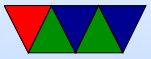


Bare Metal vs Operating System

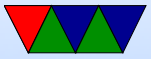


Running code w/ Operating System

- At boot, firmware runs, runs boot loader, this loads Operating System, eventually starts some sort of launcher (shell) that lets you run programs
- Device drivers abstract away talking to hardware
- Program talks to the operating system via "system calls"
- Program is an "executable", which has machine code and data. O/S along with loader/linker will take program, copy to memory, set up registers/stack, then jump to code.

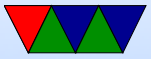


- Draw memory map (Virtual Memory is involved but beyond scope of class)
- When program done, exits, which returns control to Operating System

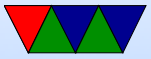


Running code on Bare Metal

- Code loaded into flash
- Power on, start with nothing. Have to do things like enable clocks.
- No device drivers, if want to talk to hardware have to flip bits in config registers manually
- Executable is just a blob, any writable data you have has to be copied to SRAM
- Can't exit your code. If you don't stop execution with a loop you'll run off the end and try to interpret



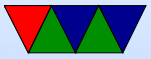
uninitialized RAM as instructions.



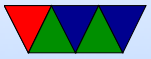
Helper Code in the Template

- 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).

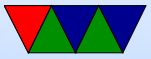


- The next set of 32-bit values in the image are interrupt vectors. One of them is the reset vector; that address is jumped to at reset time. The provided code points this to your main() function.



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.



Switch Debouncing

- What is it
- Why needed
- Hardware
- Software