ECE 471 – Embedded Systems
Lecture 4

Vince Weaver

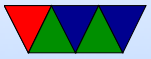
http://www.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

10 September 2015

Announcements

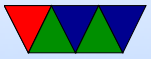
- Any questions on HW#1?
- HW#2 will be posted today



Homework #2 background

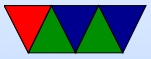
- It's mostly about getting your pi up and running, a small C coding assignment, and some short-answer questions.

- The directions will have you copy a file to your pi. It's a .tar.gz file. What is that?
  Sort of the Linux equivalent of a zip file
  tar = tape archive (ancient history) that runs lots of files together
  gz = gzip, which compresses it (makes it smaller)
  you may see other (Z, bz2, xz). What are the differences?



Mostly in compressed size vs compress/uncompress resources
gzip good enough for what we are doing.

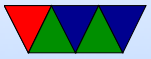
- Coding homework is to take some existing C code and modify it. Can use the editor of your choice. Many on Linux. “nano” is easy. “vim” if you are serious about Linux. Also various graphical ones, and if you want you can even code it up on your desktop/laptop, but you probably want to copy it over to test before submitting.



C compiling, Makefiles

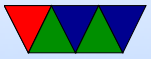
- C compiling on Linux
  We will use gcc (what others exist. clang?)
  Typical command line is something like:
  `gcc -O2 -Wall -o hello_world hello_world.c`
  `-O2` is optimization, `-Wall` is show all warnings
  A lot more options, see man page

- We use a Makefile to automate the process. What is make?
  You give it a list of dependencies, then it automatically



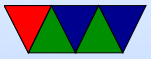
sees what files have changed and then runs commands to build things

Feel free to play with it, but a warning, tabs are significant so weird errors if you use spaces instead.



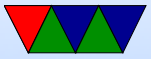
Cross compiling

- Can compile for a different architecture, for example x86 to ARM
- Why do it? Faster. Target doesn't have enough resources. Want to target multiple devices.
- To test would need an emulator (like qemu)



Comment your Code!

- Comment your code!!!!!
  Why?
  I will take points off it you don't.
  Also helps other people looking at your code figure out what's going on. Including me the graded. Including you trying to re-use some code a year from now.
  Having your name and a description of what the overall file and each function does doesn't hurt.
  Even fancier commenting conventions companies will



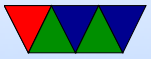
have for automated tools.

Mostly comment non-obvious stuff.

So `for(i=0;i<10;i++)` not so much.

But something like `i=4.3+10*j;` yes.

You can't really over-comment (well you can, but it's harder to over-comment than under-comment)



C Review

- Loops in C

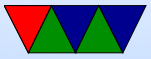
  ```
  for(i=0;i<10;i++) {}
  while(i<10) { i++}
  do {} while(i<10);
  ```

- printf

  See the man page

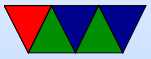
  How print an integer? `printf("%d");`. Character? String? floating point? More advanced formatting stuff

  Escape characters like percent and quotes.

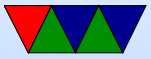


- Something cool

- ANSI escape color/art
  Back in the day, how could you do fancy colors on screen? Over serial port, so couldn't directly write to video card remotely?
  Escape sequences. A pattern you wouldn't normally type by accident.
  Default ended up being ESCAPE (ASCII 27) followed by [then some pattern of characters. Can move cursor, change colors, etc.
  Back in the day I used to make a lot of ANSI/ASCII art.

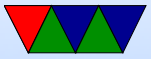


Sort of useless skill now.



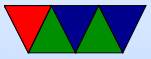
Assembly Language: What's it good for?

- 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.)



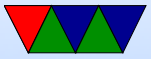
How Code Works

- Compiler generates ASM (Cross-compiler)
- Assembler generates machine language objects
- Linker creates Executable (out of objects)



Tools

- compiler: takes code, usually (but not always) generates assembly
- assembler: GNU Assembler as (others: tasm, nasm, masm, etc.)
  creates object files
- linker: ld
  creates executable files. resolves addresses of symbols. shared libraries.

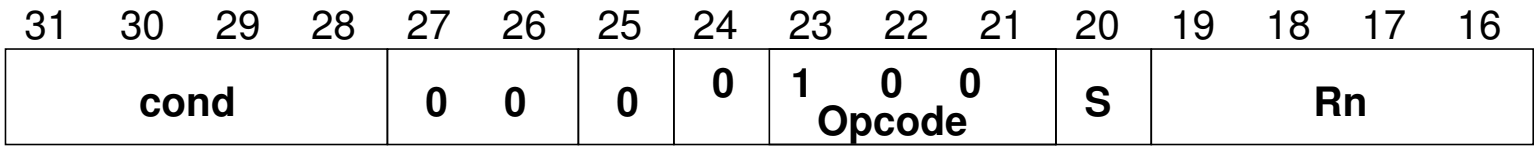


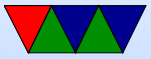
Converting Assembly to Machine Language

Thankfully the assembler does this for you.

ARM32 ADD instruction – `0xe0803080` == `add r3, r0, r0, lsl #1`

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





Executable Format

- ELF (Executable and Linkable Format, Extensible Linking Format)
  Default for Linux and some other similar OSes
  header, then header table describing chunks and where they go

- Other executable formats: a.out, COFF, binary blob