ECE 471 – Embedded Systems
Lecture 6

Vince Weaver

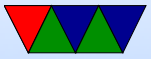
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

11 September 2017

Announcements

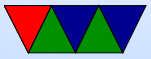
- HW#2 was posted, it is due Friday



Homework #1 Review

- Characteristics of embedded system
  - embedded inside, resource constrained, dedicated purpose, real-time
  - Toothbrush is actual specs I came across
  - Real-Time Confusion: we will discuss this more in future.

    Toothbrush: Just turning off the motor, and it takes an extra 1/2s is not really considered a real time thing. No one dies, no hardware destroyed, just mild annoyance

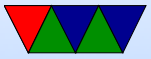


if noticed at all. Now if somehow it had to keep the waveform to H-bridge exact within 1ms or the motor would overheat and catch on fire, that could be a real-time issue.

Microwave: having a clock doesn't make it real time. Hopefully the door control has a physical interlock, but you never know. Usually when cooking food second granularity and some jitter not matter much.

- Limited Hardware

  bitness of processor: while 8 or 16 bit probably embedded these days, 32 vs 64 bit not necessarily



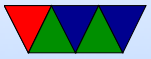
a sure sign.

Cost is an interesting one. Something like a desktop might be optimized for cost extremely, while a one-off embedded system might not, and in fact might be over-engineered (like a spaceprobe) because has to operate in tough conditions.

- Operating system?

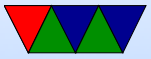
  Can have an OS and still be considered embedded.
- Be strong in your convictions!

- ASIC



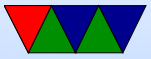
- cost/power. Depends a lot on numbers made, process, and how well designed it is.
- Extra hardware overhead? ASIC mostly just flip flops and gates. SoC internally a lot more, but these days not much else is needed.

- ARM1176JZF-S: Java, TrustZone, Vector Floating, Synthesizable Jazelle = Java acceleration
  This was in the class notes (which I post), and in ARMv6 documentation.



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



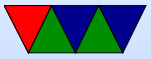
have for automated tools.

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)



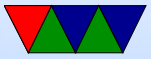
C Review

In past years sometimes the reason a HW assignment didn't work was due to using C poorly rather than misunderstandings of the desired algorithm.

- Loops in C

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

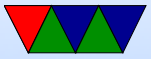
- printf



See the man page

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

Escape characters like percent and quotes.



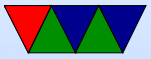
Common C Pitfalls

- Out of bounds in memory (see the `a[5]` example earlier. Also a problem with malloc() memory, Valgrind can help with that.

- Missing braces

```
if  (a==0)
        b=2;

if  (a==0)
```

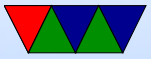


```
b=2;
c=3;
```

- = vs ==

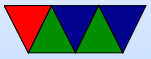
```
if (a=0) do_something_important()
```

- Never ignore warnings from the compiler!



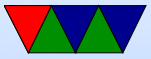
Debugging

- printf
- gdb



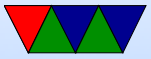
How Code Works

- 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.

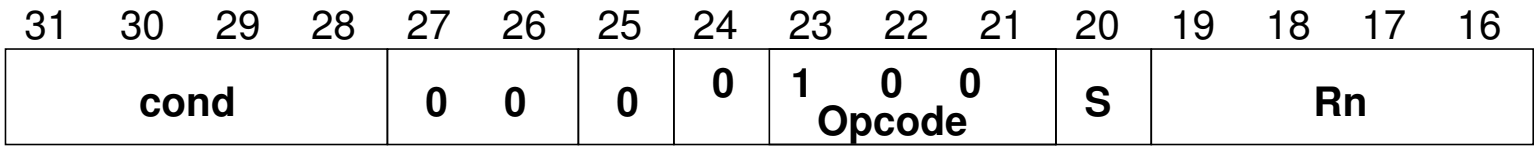


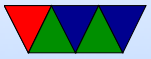
Converting Assembly to Machine Language

Thankfully the assembler does this for you.

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

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

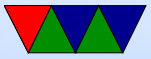




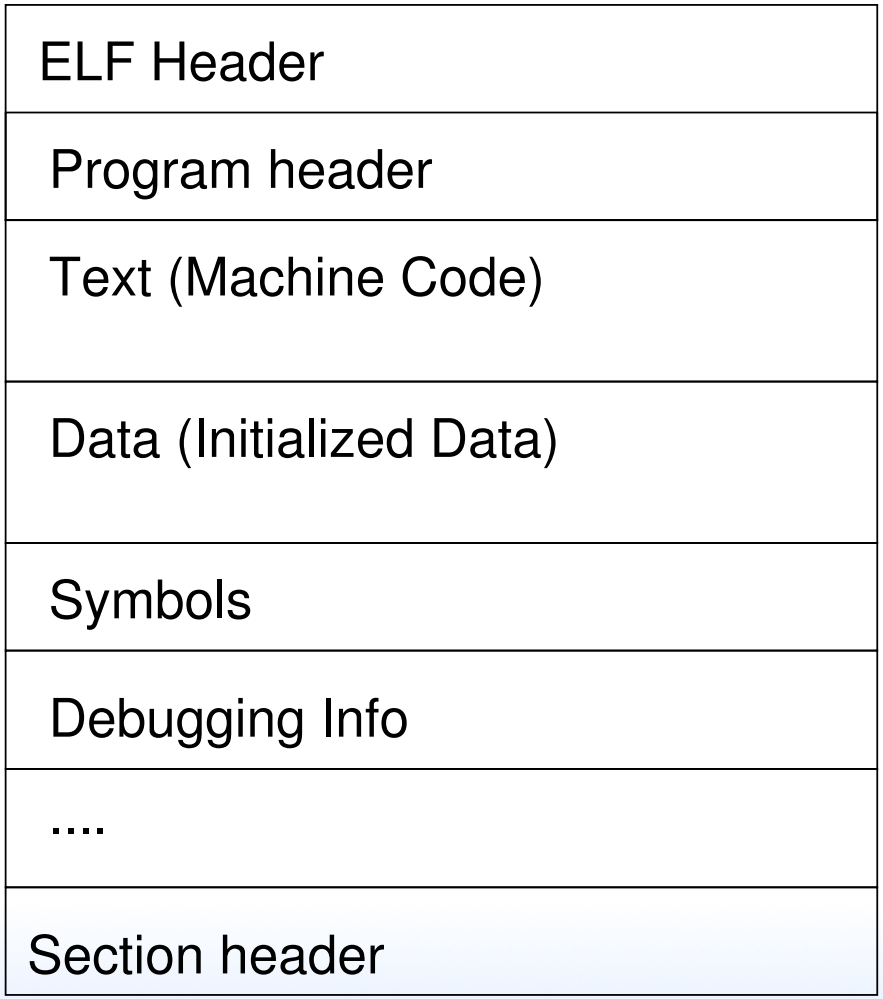
Executable Format

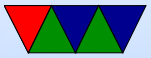
- ELF (Executable and Linkable Format, Extensible Linking Format)
  Default for Linux and some other similar OSes
  header, then header table describing chunks and where they go

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



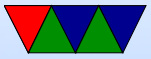
ELF Layout


ELF Header
Program header
Text (Machine Code)
Data (Initialized Data)
Symbols
Debugging Info
....
Section header



ELF Description

- ELF Header includes a "magic number" saying it's 0x7f,ELF, architecture type, OS type, etc. Also location of program header and section header and entry point.
- Program Header, used for execution:
  has info telling the OS what parts to load, how, and where (address, permission, size, alignment)
- Program Data follows, describes data actually loaded into memory: machine code, initialized data



- Other data: things like symbol names, debugging info (DWARF), etc.
  DWARF backronym = "Debugging with Attributed Record Formats"

- Section Header, used when linking:
  has info on the additional segments in code that aren't loaded into memory, such as debugging, symbols, etc.