ECE 271 – Microcomputer Architecture and Applications Lecture 12

Vince Weaver

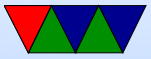
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

28 February 2019

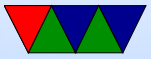
Announcements

- Read Chapters 7 and 8
- Midterm, Tuesday, 12 March
  more info on that as it gets closer

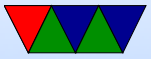


ECE471 Embedded Systems Preview

- Offered in fall.
- Similar to 271 but we use Raspberry Pi and run Linux. The operating system makes for easier coding.
- Use higher-level busses to control displays and sensors (i2c, SPI, 1-wire)
- Mostly in C, one assembly language homework
- Open-ended final project. People choose lots of fun topics.
- It is true that computer engineers seem to enjoy the



class more than electrical engineers (mostly comes down to if you like programming or not)

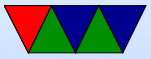


Lab #5 Update

- Many errors are just obscure C coding issues
- Index variable scoping

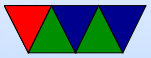
```
int j;
for(j=0;j<100;j++) {
    ...

    for(j=0;j<1000;j++) ; // delay

}
\item Operator precedence
    \begin{lstlisting}
if (x & 0x10==0) {
// actually is
if (x & (0x10==0))
```

A safe bet is to just use extraneous parenthesis



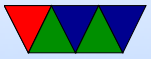
Coding Style

- Some people were asking if we were enforcing coding style in this class?
- Coding style is one of those things that varies from person to person and project to project and is hard to quantify.
- People can be very opinionated.
- Some examples of coding style:
  - Indentation: tabs vs spaces (and how many)
  - Width of screen/wrapping: 80 col or more?



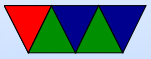
- Variable names: `new_x_size`, `NewXSize` (camel case), Hungarian Notation (strName) with type info
- Length of identifiers (old compilers ignored any past 6)
- Curly bracket on same line or next
- Header files include bare minimum, or inclulde all
- Header files alphabetical, at end, christmas tree (for git collision reasons)
-

```
if (x==0)
if (0==x)
```



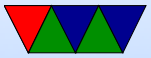
Lab #6

- Going to be stepper motor, but in assembly
- I know you hate assembly
- Try out writing functions in assembly

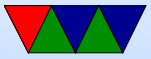


Go over global vs local vars

- Memory layout diagram again
- Code/text (usually read only)
- Variable data (globals)
- BSS (uninitialized / zeroed global variables)
  usually the OS clears these out, on our system we include startup code that does this
- Heap (dynamically allocated: `malloc` or `new`, grows "up"
- Stack, typically toward the top of memory, grows down.

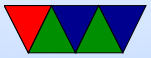


Temporary variables and local variables



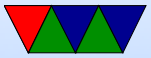
The Stack – Review

- Chunk of memory, LIFO.
- On ARM by default grows down, "Full"
  (full means points to last value pushed, empty would point to next)
- Why does it grow down? Can it grow up?



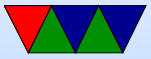
Local Allocation

- Global vs Local variables
- array of size 100.
- sub sp,sp,#400
- r0=sp
- What is it initialized to? Security?
- What must you do before returning?



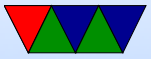
The Stack – Review

- Push – SP=SP-4. [SP]=Rd
- Pop – Rd=[SP]. SP=SP+4.
- Push/Pull multiple. Cheat and restore LR to PC to return.



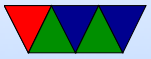
Examples

-



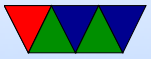
Frame Pointer (r11)

-



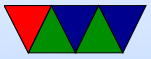
Leaf Function

- No need to save/restore LR



Typical function prolog/epilog

- Save the calleeee saved registers you use
- Allocate any local vars. Use the frame pointer?
- De-allocate local vars
- Restore the callee saved registers
- Return



Using the stack

- Writing assem code, run out of regs. Where can you store value?
- Could store to memory
- Might be easier to just temporarily push/pull on stack
- Less common on machines with less memory pressure
- 6502/x86 do it all the time, less so on ARM