ECE 271 – Microcomputer Architecture and Applications Lecture 14

Vince Weaver

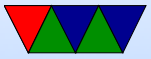
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

3 March 2022

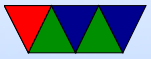
Announcements

- Read Chapters 8 + 9
- Midterm, Tuesday, 8 March



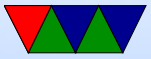
Midterm Review

- Closed book/notes.
- Short answer.
- Will be on mostly C and assembly language
- I expect you to know at least basic C
- This includes being able to set/clear bits using the bitwise logic operations
- Assembly language, I will provide a table of THUMB2 instructions so no need to memorize.
- No need to memorize all of the MODER register fields,

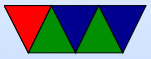


etc.

- Things like, what does this code do? Or, add comments to this code, or, what is wrong with this code.
- Know high-level things from lab, GPIO, LCD, Scanning, Stepper
- Basic understanding of what the hardware is doing, but not super detailed
- Questions may be similar to those on pre/post-lab
- There will be a question on setting the N/Z/C/V flags in assembly language. A sample problem of this type was sent out.

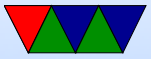


- Also stack. Where do local vars go?
- ABI, know why we have one. Where args go.



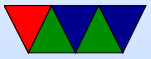
N/Z/C/V review

- We went over some examples from the sample problem handout
- Remember, ARM only updates the flags if "S" is specified in instruction (e.g. `adds`). Also `cmp` always sets flags even if no S
- N (negative) flag is set if result is negative (most-significant bit (MSB) is 1)
- Z (zero) flag is set if result is zero
- C (carry) flag is set if carry out of MSB when



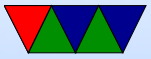
addition/subtraction. Also gets set to shifted out bit in shift and rotate instructions

- V (overflow) flag is set if carry into the MSB is not the same as the carry out of the MSB



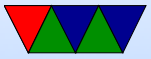
Lab #6 Information

- Re-do the stepper motor lab (lab#5), but in assembly
- Mostly learning to write functions in assembly



Lab #6 – The Delay Code (C)

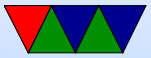
```
for(i=0;i<6000;i++) ;
```



Lab #6 – The Delay Code (asm)

A quick straightforward translation

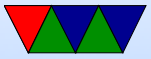
```
      mov     r5,#0          // load 0 into iterator
delay_loop
      // loop body usually goes here, which is empty
      add     r5,r5,#1       // increment iterator
      cmp     r5,#6000       // compare to 6000
      bne     delay_loop     // if we aren't there yet, loop
```



Lab #6 – The Delay Code (optimized)

A somewhat shorter version. Note that optimizing delay code can be ill-advised because making it faster changes the timing.

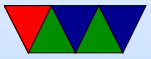
```
     mov     r5,#6000        // change to count down to 0
delay_loop
     subs    r5,r5,#1        // subtract and update flags
     bne     delay_loop      // branch if Zero flag not set
```



Lab #6 – The Delay Code (gcc Compiler)

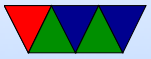
Note this is with no optimizations, so it has to save out i to memory

```
      mov     r3,#0           // load 0 into i
      str     r3,[fp,#-8]     // i is local var on stack
                              // indexed off of frame pointer
      b       next            // skip ahead
loop
      ldr     r3,[fp,#-8]     // load i from local var area
      add     r3,r3,#1        // increment i
      str     r3,[fp,#-8]     // store i back to local var
next
      ldr     r2,=5999        // load 5999
      cmp     r3,r2           // see if we've reached it
      ble     loop            // if not, loop
```



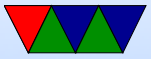
Lab #6 – The Delay Code (gcc Compiler, Optimized)

- With -O2 optimization the compiler realized that the loop didn't do anything and just skipped it all together
- How can you avoid this happening?
  - Declare i to be volatile. This works, but it forces the unnecessary loads/stores to memory
  - Could try putting an inline assembly call to a nop



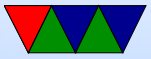
Lab #6 – Converting to a function

- Add a label for the entry point
- Grab the arguments from r0...r3
- Save any callee saved registers you use (r4...r12) on stack
- Save link register (lr) if you're not a leaf function
- Advanced: allocate any local vars if needed
- At end, restore values off stack
- Put return value in r0 if there is one
- Return from the function



Lab #6 – Delay converted to Function

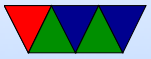
```
    // Delay, with amount in r0
delay   PROC                    // entry point
     push    {r5}               // save r5 (callee saved register)
                                //  we are a leaf function so
                                //  we don't need to save lr
     mov     r5,#0              // set loop iterator to 0
delay_loop
     add     r5,r5,#1           // increment iterator
     cmp     r5,r0              // compare to the passed in argument
     bne     delay_loop         // if we aren't there yet, loop
     pop     {r5}               // restore r5 from stack
     bx      lr                 // return to callsite
     ENDP                       // marks end of procedure (for Keil)
```



Lab #6 – Calling a function (asm)

- If for whatever reason you need caller-saved registers (r0..r3) to be preserved, save them on the stack first
- Next, set up the arguments
- Then branch and link to the function

```
mov     r0, #6000   // load 6000 as first argument
bl      delay       // branch-and-link to delay
                    //   stores value of next instruction
                    //   link-register so we can return
                    //   to right place
```

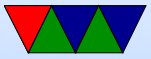


lab #6 – Using Arrays in Assembly

Code for doing a step sequence might look like this

```
int steps[4]={0x00480084,0x00880044,0x00840048,0x00440088};
int current_step,i;

for(i=0;i<4;i++) {
   current_step=steps[i];
   GPIOB->BSRR=current_step;
   delay(6000);
}
```



lab #6 – Using Arrays in Assembly

```
      mov     r5,=GPIOB             // point to GPIOB region
      mov     r6,#0                 // setup loop iterator
loop
      ldr     r1,=steps             // point to steps array
      ldr     r2,[r1,r6, LSL 2]     // index into steps array with iterator
                                    //   LSL 2 multiplies by 4 as we're an int
                                    //   (4-byte) value
      str     r2,[r5,#BSRR]         // store out to GPIOB->BSRR
      mov     r0,#6000              // load value to delay
      bl      delay                 // call delay function
      add     r6,r6,#1              // increment iterator
      cmp     r6,#4                 // see if done
      bne     loop                  // if not, loop

steps                               // define an array of four 32-bit values
      DCD     0x00480084,0x00880044,0x00840048,0x00440088
```