ECE 471 – Embedded Systems
Lecture 12

Vince Weaver

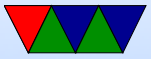
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

30 September 2019

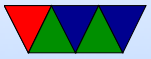
Announcements

- HW#4 was posted.
- Permissions!
  Unless your user is configured to have gpio permissions you'll have to run as root or use sudo. raspbian there's a "gpio" group which has permissions `sudo addgroup vince gpio`
  `udev` is responsible for updating permissions as the files are created and it can take a fraction of a second to detect and update.



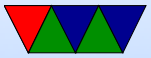
This might not work if you have an older version of Raspbian

- What should your code do if permission is denied? Not crash, certainly.



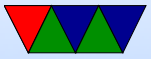
Homework 3 – General

- If you are messing with the function call parameters, be sure you fix all calling locations!
- Don't use the link register LR unless you save/restore it
- Be sure to put your name in the README!
- Should do HW, even if you only do the short-answer part. Good practice for midterm
- Comment code!
  Especially important in assembly language. Make sure comments makes sense



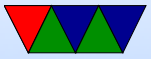
Also high level comments are best:

```
add r1,r2,#0x30 @ add 48 to r2 and store in r1
vs @ convert integer result to ASCII
```



Homework 3 – Exit

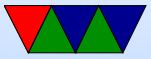
- Exit – value is an integer which goes into r0
- Note it is an integer, not ASCII



Homework 3 – Print Number

- `print_ number()` code
  - No conversion to binary, number is in binary in register.
  - The divide by 10 code is almost more interesting.
  - Good to be able to look at code and see what doing. Reverse engineering, but also debugging code you don't have the source to.

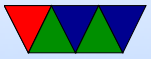
```
print_number:
        push    {r10,LR}        // Save registers
        ldr     r10,=buffer     // what does = mean?  where is buffer?
        add     r10,r10,#10     // why 10 bytes?
```



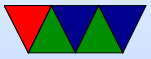
```
divide:
        bl      divide_by_10    // why no div instruction?
        add     r8,r8,#0x30     // why add 0x30?
        strb    r8,[r10],#-1    // why moving backwards?
        adds    r0,r7,#0        //
        bne     divide          //

write_out:
        add     r1,r10,#1       // why adjust pointer?

        bl      print_string    //

        pop     {r10,LR}        //

        mov     pc,lr           //
```

how would you convert to hex? Why 10 chars reserved?

```
divide_by_10:
        ldr     r4,=429496730           @ 1/10 * 2^32
        sub     r5,r0,r0,lsr #30
        umull   r8,r7,r4,r5             @ {r8,r7}=r4*r5
```



```
mov     r4,#10                  @ calculate remainder
mul     r8,r7,r4
sub     r8,r0,r8
mov     pc,lr
```



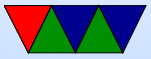
Homework 3 – Print String / Strlen

- strlen code example, many ways to do this

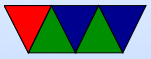
```
        mov     r2,#0           @ set length to zero
print_loop:
        ldrb    r0,[r1,r2]      @ load byte at r1+r2
        add     r2,r2,#1        @ increment length
        cmp     r0,#0           @ see if NULL
        bne     print_loop      @ loop if not
```

- Many were trying to do it without fancy addressing. Which is fine, but at the end make sure r1 still points to string and r2 has the length

```
        mov     r2,#0           @ set length to zero
```

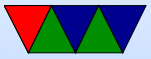


```
        push    {r1}            @ save r1
print_loop:
        ldrb    r0,[r1]         @ load byte
        add     r2,r2,#1        @ increment length
        add     r1,r1,#1        @ increment ptr
        cmp     r0,#0           @ check if NUL
        bne     print_loop      @ if not, loop
        pop     {r1}            @ restore r1
```



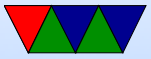
Homework 3 – THUMB/THUMB2

- Mostly a matter of removing R10 to be smaller than R8
- Illegal instruction error usually because there are *two* calls to print string, need to make sure both are blx
- Can see why THUMB2 is nicer than THUMB (assembler does most of work)
- THUMB code should have been less.



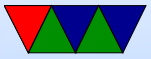
Homework 3 – Code Density

- You need to run `strip` on this to see it. Why?
  Debug info, including extra thumb debug as well as the longer filename.
- You can use `readelf -a` and `readelf -s` to see the space the various segments take up.
  Look at executables, *not* the C source code.
- Sizes



arch	unstripped	stripped
arm32	1424	620
thumb	1444	600
thumb2	1420	596
C	8156	5608
C/thumb2	8144	5612
C static	569288	484620

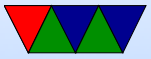
- You would think THUMB2 would be much smaller, but the assembler makes some poor decisions about wide/narrow instructions.



- Reference my LL work
- C code is larger, but also remember to include the C library:

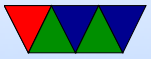
```
ls -lart /lib/arm-linux-gnueabihf/libc-2.24.so
-rwxr-xr-x 1 root root 1234700 Jan 14  2018 /lib/arm-linux-gnueabihf/libc-2.24.so
```

- There are embedded C libraries, musl, newlib, uclibc, which are much smaller and often used in embedded systems.



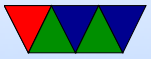
Homework 3 – Something Cool

- Mostly backward count
- How would you convert print_number to hexadecimal?



Homework 3 – Linux Tools

- cal missing days
- Julian to Gregorian calendar.
- People sad who paid weekly but paid rent monthly.
- George Washington's birthday
- Hunt for Red October
- Beware believing any page you google. Some urban legends / joke sites about this. If it were some sort of programmer bug it would have been fixed years ago.

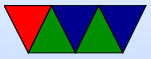


Buffered "Stream" I/O

- Slightly higher-level I/O routines in C library
- Buffered I/O
- Still use open/close/read/write underneath
  Can find file descriptor with `fileno()`
-

```
FILE *f;
f=fopen("filename","r");
if (f==NULL) fprintf(stderr,"Error!\n");
fwrite(buffer,size,members,f);
fclose(f);
```

- Buffered I/O (saves overhead, fewer syscalls, maybe makes I/O faster, but also adds potential delay)



- Use `fflush()` to force buffer flush
- Use `rewind()` to rewind to beginning of file