
MAN, I SUCK AT THIS GAME. CAN YOU GIVE ME A FEW POINTERS?
0x3A28213A
0x6339392C,
0x7363682E.
I HATE YOU.

The C Programming Language

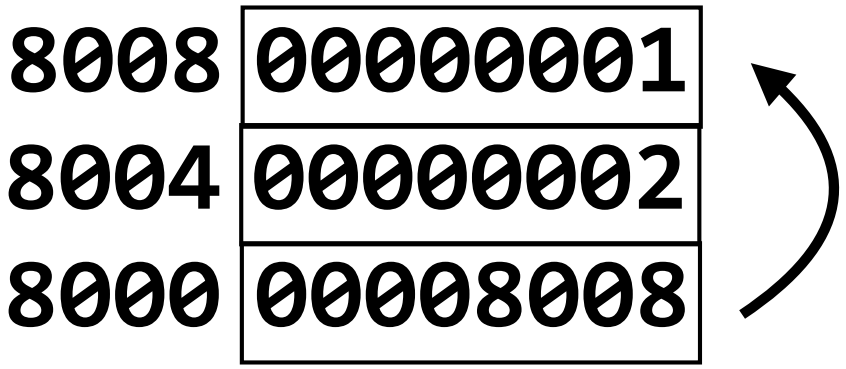
Part 2: Pointers and Arrays

```
// on.c

#define FSEL2 0x20200008
#define SET0  0x2020001C

void main()
{
   *(volatile int *)FSEL2 = 1;

   *(volatile int *)SET0 = 1<<20;
}
```

```
// on.list

8000: e59f3010  ldr r3, [pc, #0x10]
8004: e3a02001  mov r2, #1
8008: e5832008  str r2, [r3, #8]
800c: e3a02601  mov r2, #0x100000
8010: e583201c  str r2, [r3, #28]
8014: eafffffe  b   8014
8018: 20200000  .word   0x20200000
```

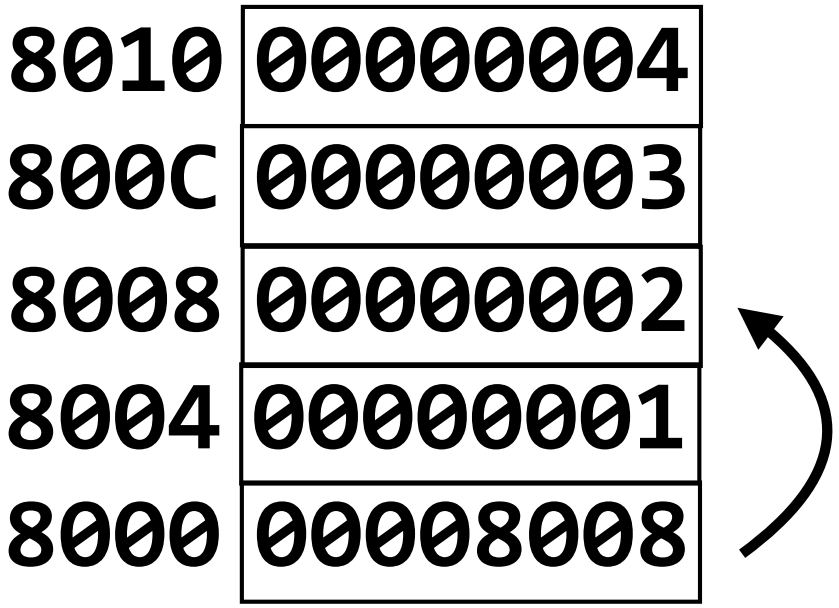
Data in Memory

int/



Memory

array/


8010 00000004
800C 00000003
8008 00000002
8004 00000001
8000 00008008
Memory

intp/ & arrayp/

Types and Pointers

`int` is an integer (4 bytes on the arm)

`int*` is a pointer to an `int` (`int *` is also 4 bytes on the arm)

A pointer is a typed address

The type of `int` is not the same as `int*`

The operations on `int` are different than the operations on `int*`

Adding an `int i` to an `int* p`, increments the pointer by `p += i*sizeof(int)`

```
// Which of these statements generate warnings, errors?
// Hint: What operations make sense on addresses

   int *p, *q, n;

   n = 0x20200000;
   p = n;
   q = p + 1;
   q = 2 * p;
   n = q - p;
   if( p == q ) ;
   if( p != q ) ;
   if( p < q )  ;
   if( p == 0 ) ;
   if( p == 1 ) ;
   q = p & (~3);
   q = p | q;
   n = q;
   p = &(n+1);
   n = *&n;
   p = &*p
   p = &3;
```

C Types

```
// C Types

char c;
unsigned char uc;

short s;
unsigned short us;

int i;
unsigned int ui;

long long l;
unsigned long long ul;
```

From the C specification:

The three types `char`, `signed char`, **and** `unsigned char` **are collectively called the character types. The implementation shall define** `char` **to have the same range, representation, and behavior as either** `signed char` ***or*** `unsigned char`.

OMG: It is not defined!

In arm `gcc`, `char` **is** `unsigned`

```
#include <stdint.h>

int8_t i8;
uint8_t u8;

int16_t i16;
uint16_t u16;

int32_t i32;
uint32_t u32;

int64_t i64;
uint64_t u64;
```

```
// Character constants are enclosed in ' '

// Strings are arrays of characters,
// ending with a '\0' (NUL)

char s1[] = {'c','s', 1','0','7','e','\0'};
```

```
// String constants are enclosed in " "

char s2[] = "cs107e";
```

```
// Strings

char s1[] = {'c','s', 1','0','7','e','\0'};

char s2[] = "cs107e";

// What is the difference between s1 and s2?
```

```
// Strings

char s1[] = {'c','s', 1','0','7','e','\0'};

char s2[] = "cs107e;

char *s3 = "cs107e";

// What is the difference between s2 and s3?
```

```
// strings.c

char cs107e[] = "cs107e";

for( char *s=cs107e; *s != '\0'; s++ )
    putchar(*s);

putchar('\n');

// for pointers, s++ ≡ s+sizeof(*s)
```

Read:

The most expensive 1-byte mistake,

Did Ken, Dennis, and Brian choose wrong with 0-terminated text strings?

Poul-Henning Kamp

http://queue.acm.org/detail.cfm?id=2010365

Why Pointers?

Access particular memory locations like FSEL2

Pointers can be used to reference elements of an array

Pointers are often used to pass references to large objects to functions (coming)

Pointers are used in data structures to reference other data structures (coming)

Assignment 2

`blink.c`

```
#include "gpio.h"
#include "timer.h"

const int pin = GPIO_PIN20;

void main(void)
{
    gpio_init();
    timer_init();

    gpio_set_function(pin, GPIO_FUNC_OUTPUT);
    while (1) {
        gpio_write(pin, 1);
        delay(1);
        gpio_write(pin, 0);
        delay(1);
    }
}
```

Testing

We grade your software by testing it

```
// Assignment 2 implement functions "gpio.h"

enum {
    GPIO_FUNC_INPUT   = 0,
    GPIO_FUNC_OUTPUT  = 1,
    GPIO_FUNC_ALT0    = 4,
    GPIO_FUNC_ALT1    = 5,
    GPIO_FUNC_ALT2    = 6,
    GPIO_FUNC_ALT3    = 7,
    GPIO_FUNC_ALT4    = 3,
    GPIO_FUNC_ALT5    = 2,
    GPIO_FUNC_INVALID = 8
      // Only returned for invalid pins
};

void gpio_set_function(unsigned pin, unsigned func);
unsigned gpio_get_function(unsigned pin);
```

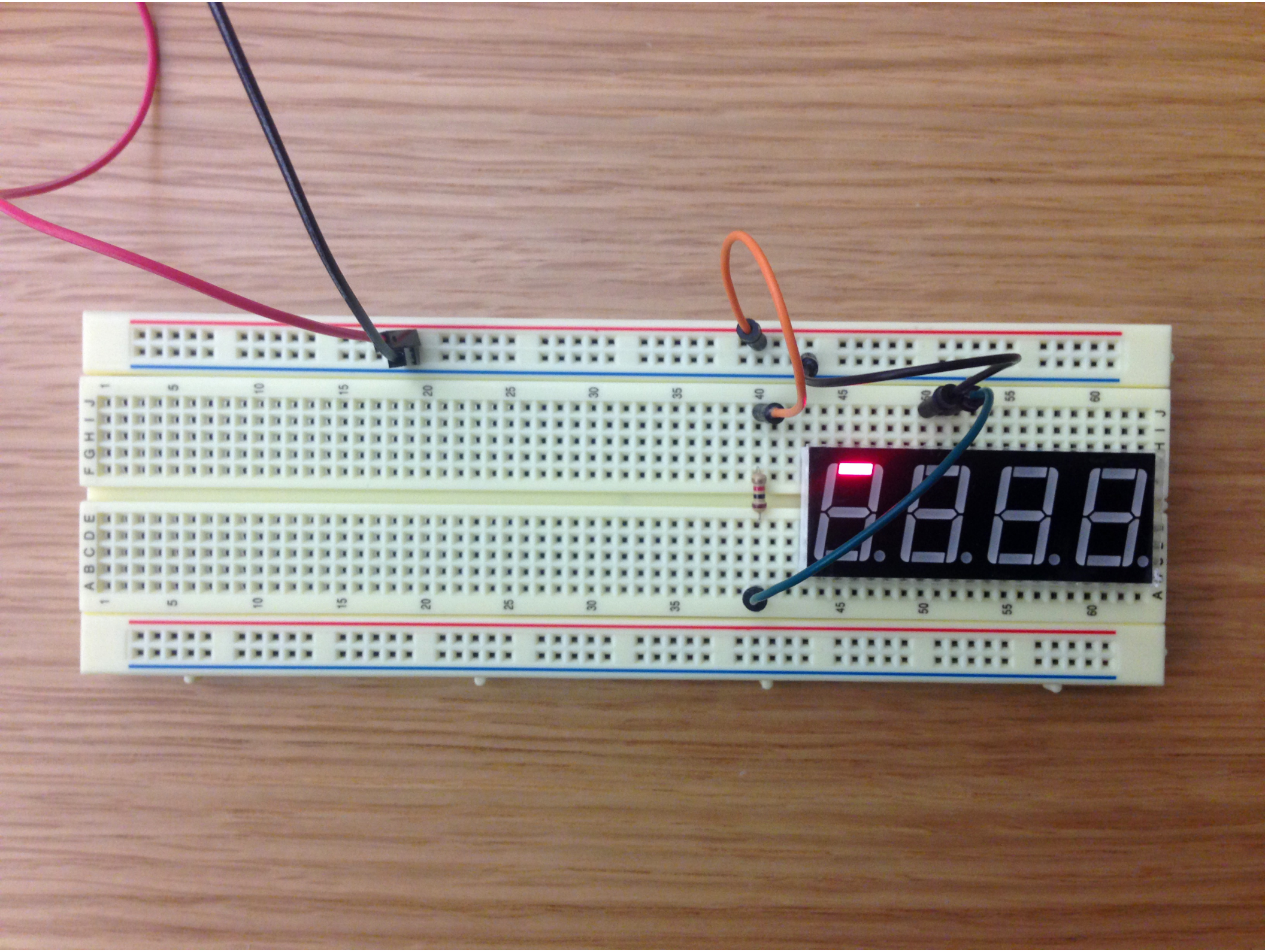
unittest/

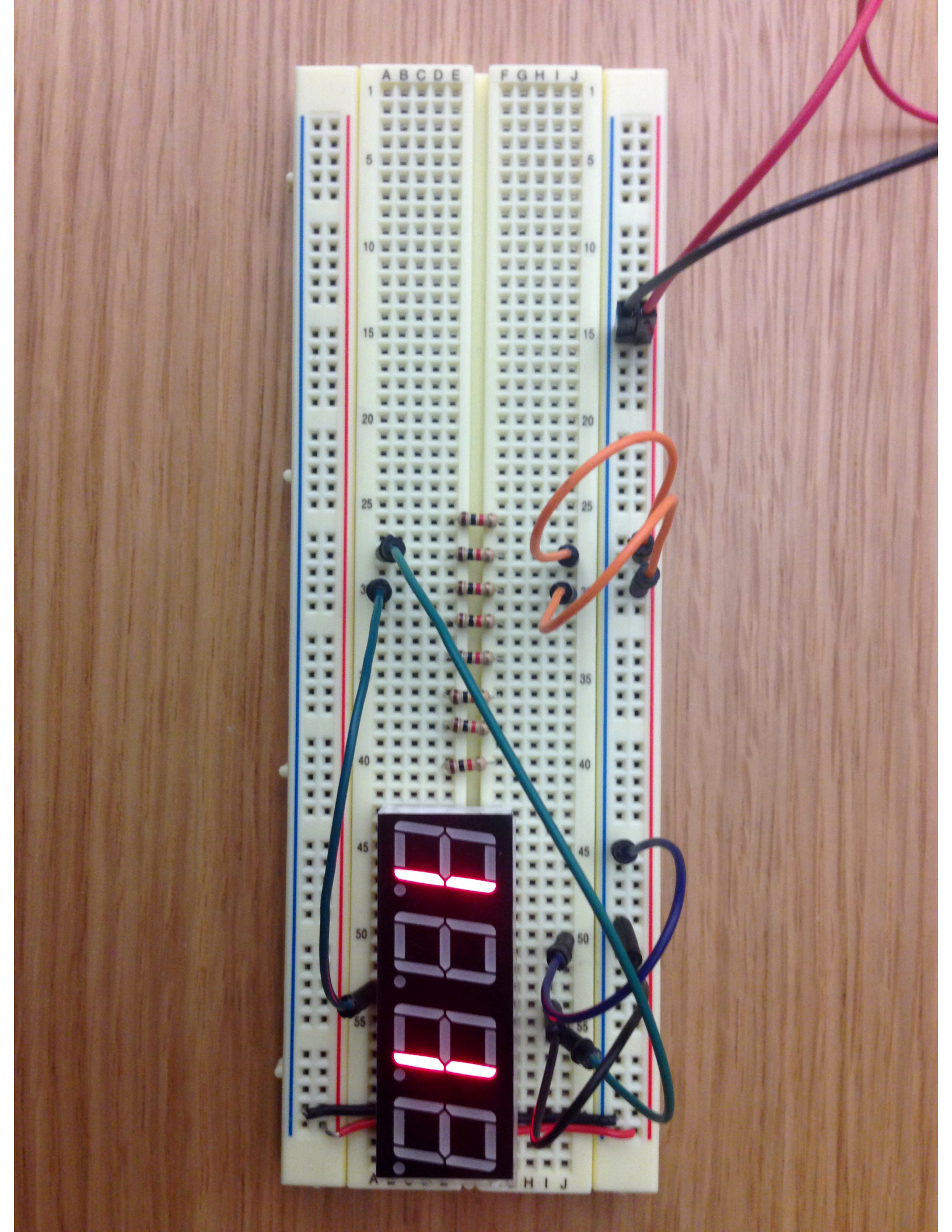
Exercise

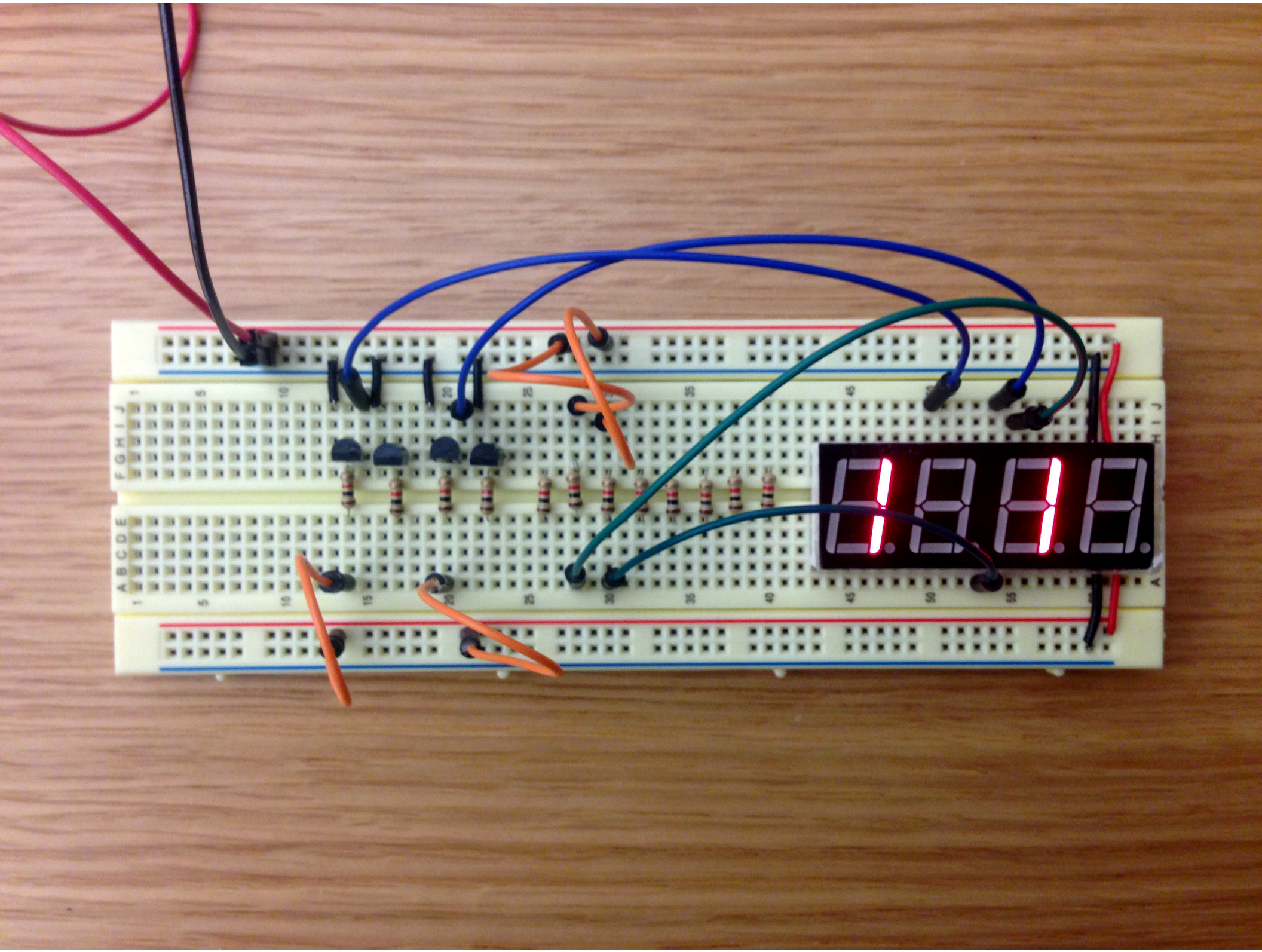
Based on your experience implementing the Larson scanner in Assignment 1, what can go wrong? That is, what bugs did you have to find?

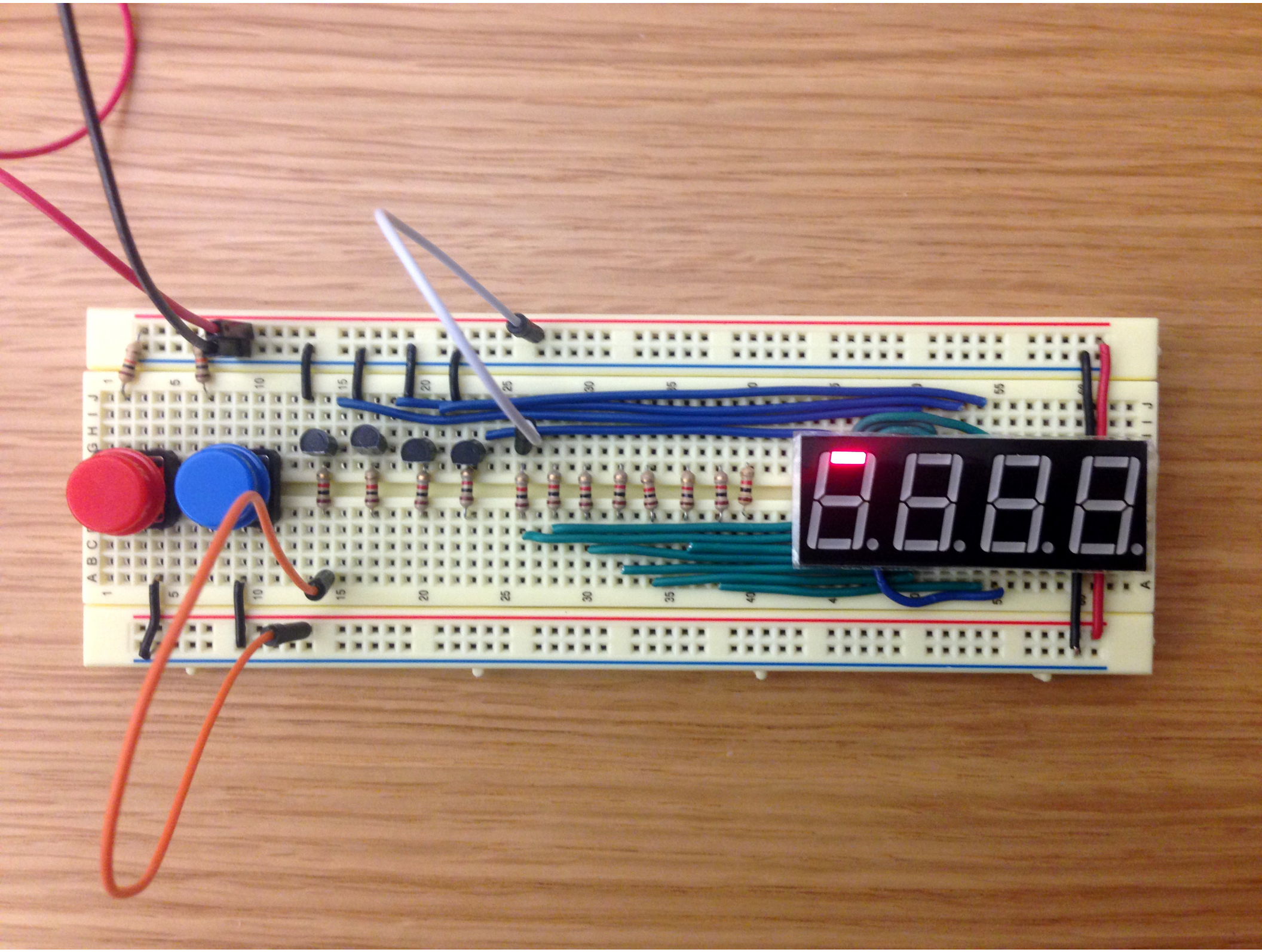
Develop a test that would fail for one of those bugs (using `assert`).

Test as You Go!









Test as You Go

Implement the simplest possible thing first, then test it. A simple thing is much more likely to work than a complex thing.

Next add more functionality, and test it before moving on. Take short, baby steps.

Rerun all your tests at each step. You may break something as you go. This is particularly when you are improving the style of the code, since it should still work.

Tests are Your Friend

Think of the tests as a specification of what it should do. Assertions will clarify your understanding of how it should work.

Part of the skill of software development is anticipating what can go wrong. We will give you some starter tests, but you are responsible for more thoroughly testing your software.

Questions

Pointers: the fault in our *s

Pointers are ubiquitous in C, and inherently dangerous. Watch out!

Q. For what reasons might a pointer be invalid?

Q. What is consequence of using an invalid pointer?

```
main()
{
    int *mem;
    int addr;

    mem = 0;
    for(addr=0; addr<0x400000; addr++)
       mem[addr] = 0;
}

// What's this program do?
```