ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 20

Vince Weaver

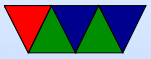
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

23 March 2026

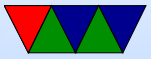
Announcements

- HW#5 was posted (due Friday)
- Midterm was graded but has not been handed back yet
- Lab#6 this week (the snow tried its best but classes still happening)



HW5 Extra Info

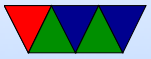
- HW#5 was posted to codelab
- What follows are some topics that come up in the homework that maybe weren't covered fully when we discussed the topics originally



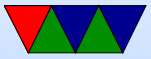
HW5 – Structures

- Hopefully remember how to do those
- The “type” of the structure that goes first, codelab calls the “tag name”

```
struct emp_type { // codelab calls this the "tag name"
    double q;    // double
    char *ptr;   // character pointer
    int a[10];   // array
};
struct emp_type emp[50]; // declare array of 50 of the
                         //    struct emp_type
                         // you can also do this directly
                         // as part of the definition
```

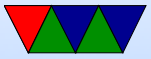


```
emp[0].q=45.5; // set value of the q element in the first
               // emp_type
emp[20].a[5]=5; // can have arrays inside of arrays
emp[49].ptr="Hello"; // can point char pointers to
                     // strings, note you cannot
                     //    strcpy(emp[49].ptr,"Hello");
                     // as it doesn't point to anything
                     // yet so there's nowhere to copy
                     //  the data to
```



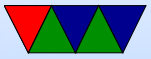
HW5 – scanf()

- To get an integer `scanf("%d",&x);`
- To get a double `scanf("%lf",&f);`
- The return value is how many values were successfully read. So in the above case if you check the return value of `scanf()` it should be 1 if it worked, 0 or maybe `EOF` (-1) if it failed
- More info on `scanf`? Run `man scanf` on Linux



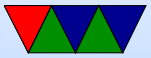
HW5 – string compare

- How to compare strings? `strcmp(string1, string2)`
- Codelab asks some odd things, like if a string is bigger than another?
- Can look up documentation (manpage), `strcmp()` works by subtracting each byte from the other
  - If result is 0, means exact match
  - if result is positive, means string1 was "bigger"
  - If result is negative, means string2 was "bigger"
- So to check if string1 is "bigger" than string2



```
is (strcmp(string1,string2)>0) ...
```

- Why is this useful? Sorting
  Used when arranging strings in Alphabetical Order
- Sorting strings seems like it might be straightforward, but there are hundreds of algorithms each with its own strong and weak points

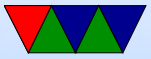


HW5 – string assignment

- A C string is a char array `char a[10];`
- You can also treat a pointer to a character array as a string `char *b;` but it starts out with no memory backing it so you can't use it until you point it to an existing chunk of memory (ideally another string)
- You can't assign a string to an array

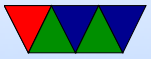
  ```
  char a[10]; a="Hi"; //gives an error
  ```

  (You would use `memcpy()` or `strcpy()` for this)
- You can however point to string `char *a; a="Hi";`



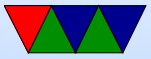
- Homework has you doing a lot of that
- Still hard to think of pointers

g

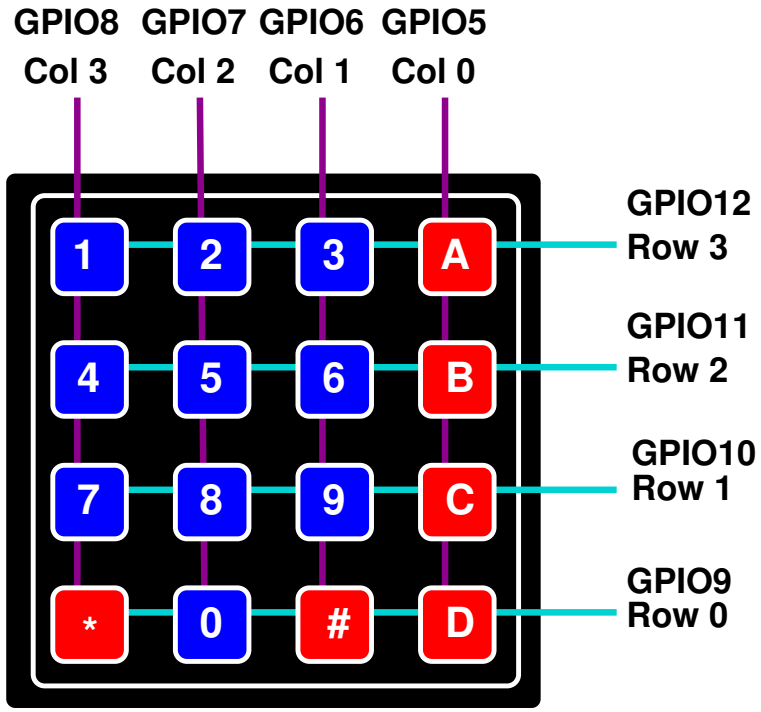


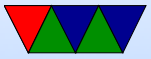
Lab6 – Keypad

- Implement the keypad
- NOTE! '*'+'D' won't work for this lab, as we are reimplementing keypad support ourself
- The lab handout has lots of useful info, don't immediately skip to the code comments



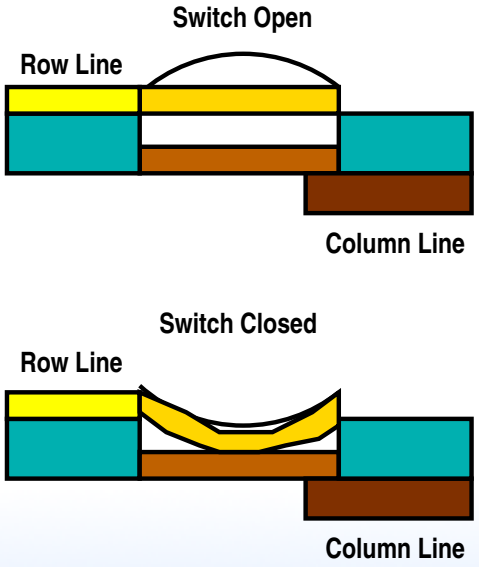
Lab6 – Keypad

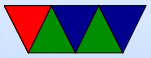

GPIO8
Col 3
GPIO7
Col 2
GPIO6
Col 1
GPIO5
Col 0
1
2
3
A
4
5
6
B
7
8
9
C
*
0
#
D
GPIO12
Row 3
GPIO11
Row 2
GPIO10
Row 1
GPIO9
Row 0



Lab6 – Keypad

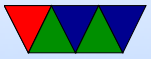
- 4x4 keypad
- Eight GPIOs, 4 for row, 4 for column
- When you press a button it connects the row to column





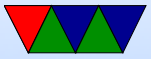
Lab6 – Reading Input with GPIOs

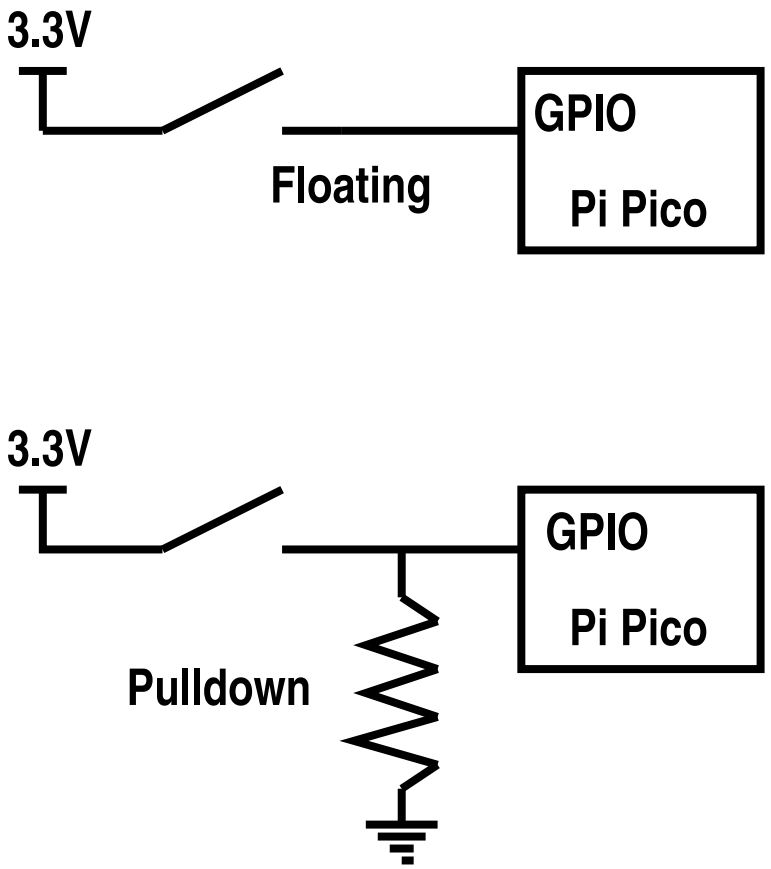
- You can configure GPIOs as inputs
- If it's closer to Vdd (3.3V on Pico) it reads as a 1
- If it's closer to Gnd (0V) it reads as a 0
- The input pins are in a High-Z (high impedance) state

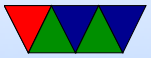


Lab6 – Pull-down resistors

- You might wonder why there are resistors on the pins taking the GPIOs to ground
- If a line is left "floating" (not tied to Vdd or gnd) the voltage on it can float, often to a previous value
- You can add resistors to ground that pull the voltage to ground when floating.
- If you then drive it high, it overpowers the pulldown and goes high with only a small amount of drop

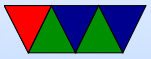


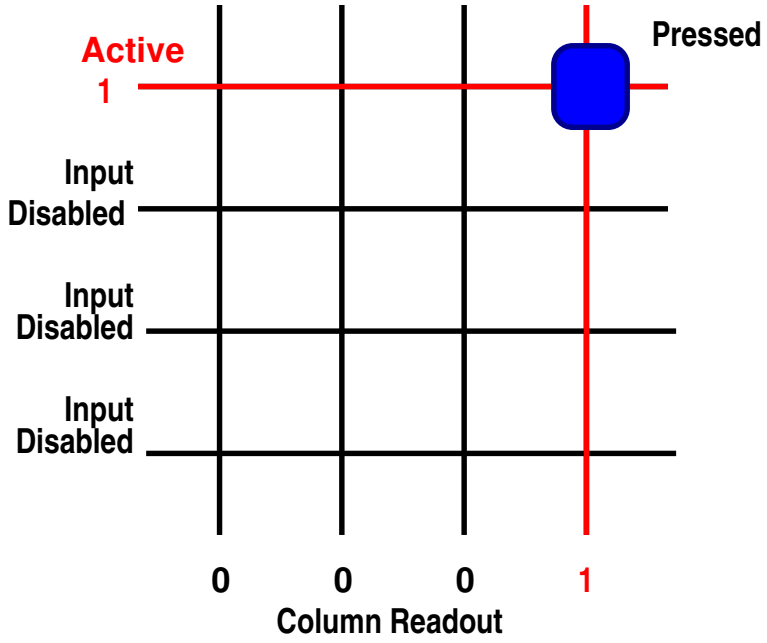

3.3V
GPIO
Floating
Pi Pico
3.3V
GPIO
Pi Pico
Pulldown

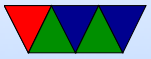


Lab6 – Scanning

- With 16 buttons you'd need 16 GPIOs which is a lot
- Instead you can arrange them on a grid and scan one row at a time
- There's a switch that shorts between a row and column line
- To scan, set one row high, then read the columns to see if any of the keys pressed
- Scan it frequently

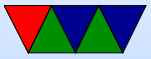



Active
1
Pressed
Input
Disabled
Input
Disabled
Input
Disabled
0
0
0
1
Column Readout



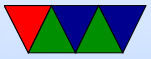
Lab6 – Keyboard Background

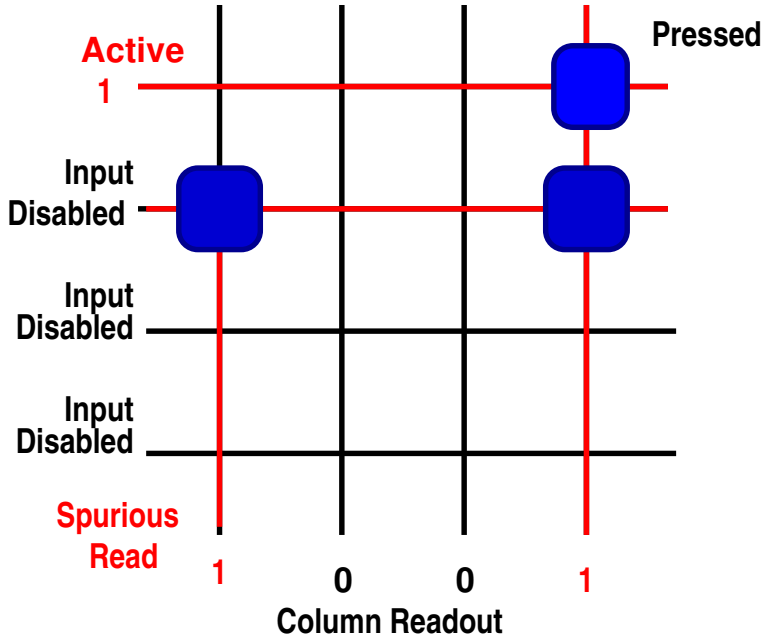
- All keyboards basically work like this
- It would be a waste for your computer to constantly be scanning the keyboard
- Usually there is some sort of micro-controller that does this, and when it sees something pressed it "interrupts" your CPU to let it know
- Getting direct raw keyboard is possible (difficult under Linux, it's more complex than just `scanf()`
- Modern keyboards are USB which complicates things



Lab6 – Keyboard Rollover

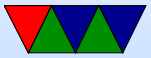
- You've maybe encountered this when gaming
- N-key rollover (can you press an arbitrary number of keys and have them all recognized)
- If you press enough a path can happen that gives false keyreads
- The fix for this is diodes but this complicates/makes more expensive




Active
1
Pressed
Input
Disabled
Input
Disabled
Input
Disabled
Spurious
Read
1
0
0
1
Column Readout

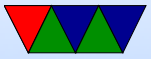
Lab6 – Phone Aside

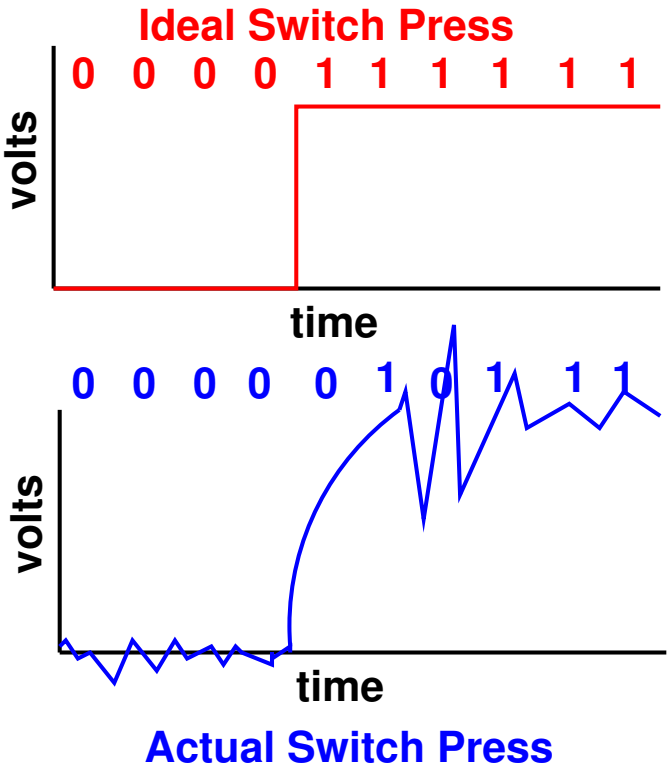
- Aside on Touch-tone phones
- Different frequencies for rows/columns
- Might say only 3x4, but secret 4th row

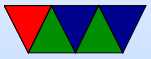


Lab6 – Debouncing

- When you press a key, do you get a nice clean signal?
- Often contact is messy, so you get more of a transient, on and off a few times before settling
- You might have to handle this yourself in code to "debounce" it
- For this lab it doesn't seem to be necessary

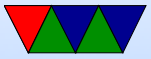


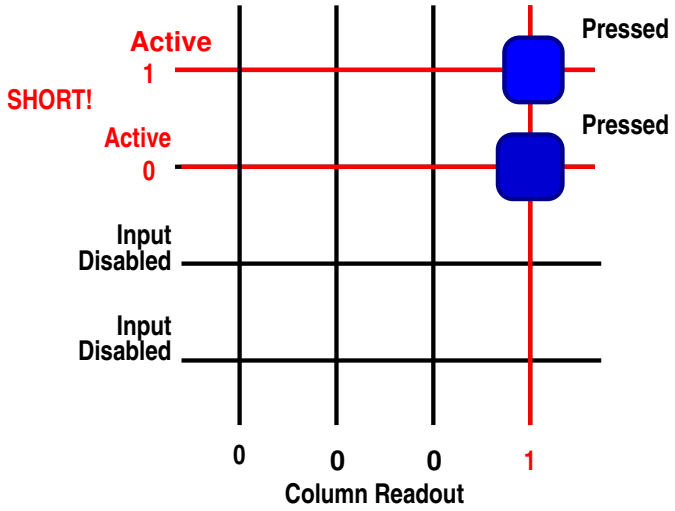

Ideal Switch Press
0 0 0 0 1 1 1 1 1 1
volts
time
0 0 0 0 0 1 0 1 1 1
volts
time
Actual Switch Press

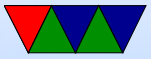


Lab6 – Shorting

- Need to be careful not to short a GPIO pin driving 0 to one driving 1.
- Directly connect those and a lot of current can flow and it potentially can damage the Pico
- There's a limit to how much current a GPIO can handle
- This could happen if multiple buttons pressed
- We can avoid this by only ever having 1 row pin active at a time and having rest in High-Z "input" mode




Active
1
SHORT!
Active
0
Input
Disabled
Input
Disabled
Pressed
Pressed
0
0
0
1
Column Readout

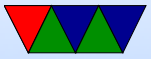


Lab6 Code – Init

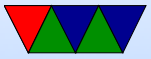
- First we have you initialize all the GPIOs.
  - First the LED ones again (like from last-lab) (why?)
  - Then the 8 additional ones for the keypad
  - Set GPIO5 to 9 to 0, GPIO10 to 13 to 1? (why?) this is what previous lab has you do?
- Have you use the

```
gpio_init(gpio_number);
gpio_put(gpio_number,0);
gpio_set_dir(gpio_number,GPIO_OUT);
```

  interface for this rather than MMIO. Why? Dunno.

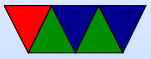


I have to assume these helper functions use MMIO themselves but haven't checked



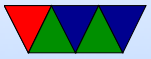
Lab6 Code – Scan

- GPIO9 to GPIO12 are rows, GPIO5 to GPIO8 columns
- Set only one row at a time as output (set to 1), all other rows to 0
- Read the column GPIOs to see if any of them are set. If it is set, it means the key at the intersection of row vs column is pressed
- Use printf to print which row/column active



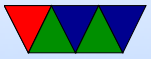
Lab6 Code – MMIO interface

- There are three MMIO struct members we use for this one (allow setting "atomically")
  - `sio_hw->gpio_out` is a 32-bit value, set all 32 GPIO values with one write. You already used this in Lab 5.
  - `sio_hw->gpio_in` is a 32-bit value you can read to get the current state of GPIOs
  - `sio_hw->gpio_oe` is used to pick whether a GPIO is in input our output mode. A '0' represents input and '1' represents an output in each bit position.

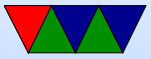


Lab6 Code – Scan Loop

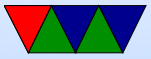
- Create an infinite loop; In the body of the never ending loop, do the following steps
- Create a for loop that iterates over the four rows
- Based on the loop iterator we want to turn on only one of GPIO9, GPIO10, GPIO11, or GPIO12.
  - First clear bits 9, 10, 11, and 12 of `sio_hw->gpio_out` to zero without changing any of the other GPIO pins. As a reminder, you do this with bitwise AND and a proper mask.



- Next clear the output enable bits 9, 10, 11, and 12 of `sio_hw->gpio_oe` to 0 as well. This will set GPIO9..12 to be inputs.
- Next set the output enable bit of the row you want to be '1' to an output via `sio_hw->gpio_oe`. Remember you can use bitwise OR to set a bit, and you can use something like `(1<<9)` to create an integer with the 9th bit set.
- Finally set the output to '1' in `sio_hw->gpio_out` for the proper bit (9 ... 12) using bitwise OR. BE SURE TO ONLY HAVE ONE '1' SET in this range.



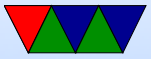
- Delay 10 microseconds (`sleep_us()`) to allow the pin to become an output.
- Create another for loop that iterates over the columns GPIO5, GPIO6, GPIO7, GPIO8 to see if any of the input bits are set
  - Check each column GPIO via `sio_hw->gpio_in` to see if bit 5, 6, 7, or 8 is set.
  - Remember: to see if a bit is set you can use bitwise AND with a 1 in the proper place, and if it's nonzero it is set, otherwise it isn't.
  - If the value is non-zero, print the row and columns



number of the pressed key using `printf()` and it should appear on the LCD panel.

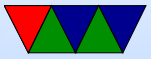
Something like:

```
Row 3, Col 2
Row 0, Col 1
```



Lab6 Code – Bit-manipulation/Masking Reminders

- To set a single bit in an integer you can use left shift, for example to set bit i to be on `(1<<i)`
- To create a mask with all 1s but 0s in a certain range, it's often easier to create the opposite and then invert it with bitwise not `~(1<<i)`
- To set one bit without touching others, use bitwise OR, something like `x=x|(1<<5);`
- To clear one bit without touching others, use bitwise



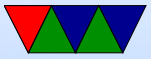
AND with a mask, something like `x=x&(~(1<<5));`

- If you want to clear a bunch you can have something complicated like

```
x=x&(~((1<<9)|(1<<10)|(1<<11)|(1<<12)));
```

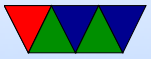
or instead it might be easier to do

```
x=x&(~(0xf<<9));
```



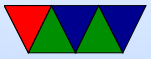
Lab6 Code – Aside

- Can I do this other ways?
- Clear the actual active line at end instead of mass clearing all of them
- Could I use xor?
- Could I read out all the columns at once with a read rather than a loop to find them?



Lab6 Code – Debugging

- What it not working?
- Can debug with `printf()`
- If you want to know if the code reached somewhere? Drop in a `printf()`
- Want to see if the value of a variable matches what you think it should? `printf()`
- Want to quickly check what the GPIO values are? `printf()`



Apple II Floating Bus Aside (if there's time)

- split.dsk
- midline.dsk
- lores_escape.dsk
- megademo.dsk