ECE 177 – Programming I: From C Foundations to Hardware Interaction Lecture 10

Vince Weaver

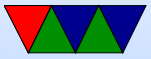
https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

13 February 2026

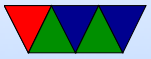
Announcements

- Reminder: no class/labs on Monday
- HW#2 was posted! Due Wednesday night!
- Sorry for mixup on HW#1 due date
- HW#1 notes, most people that were stuck seem to be either not using L on long constants, or ones it wanted you to use double over float.
- Note there is a button that says "request professor help" but I don't always know how to find who sent it, so e-mailing me might be better for now



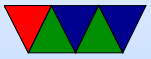
Detecting Odd of Even Numbers

- Think of how you'd do it
- We'll come back to it later in lecture



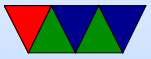
Lab3 Preview

- We will be wiring up breadboards with parts
- The Pi Pico will control these parts
- We will eventually write some C code to run on the Pico



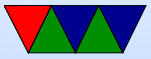
Cross-Compiling

- You might ask, what's the difference between Lab#2 coding and VS/pico coding?
- For Lab#2 we write the code locally, run the compiler on it, and run it on your laptop
- For Labs on the Pico, you write the code locally, but you "cross-compile" 32-bit ARM code (that won't run on your laptop), then copy it over to the board
- This is a common thing to do on embedded systems



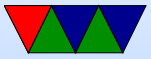
Using Breadboards

- If you had ECE101 probably seen this already
- Board with holes into it that you can plug in parts and wires
- There are (hidden) conductive lanes under the plastic. This means wires inserted into the same column end up electrically connected
- This is a fast way of prototyping circuits (easier than soldering, wire-wrap, making PCBs)



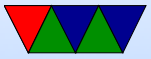
Programming Board to Test

- Two ways to program
  - If using VS-code, pressing "Run" will compile code and upload to board
  - pi-pico also supports USB-storage mode. Sometimes when plug in, appears as a hard-drive on your desktop. If you click "compile" in VS code it will make a .UF2 file, if you copy (or drag) this file to the drive it will program if for you
  - You can force hard-drive mode by holding down the



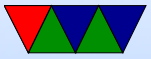
white button on the pico while plugging in the USB connector

- You can do all this manually w/o VS code (especially on Linux) if I have more time I'll post directions on that



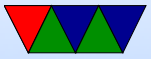
Lab 3 summary

- Wire up your breadboard with parts
- Have TA double-check wiring (you can possibly damage things if mis-wire, especially if ground and VCC shorted or connected to wrong places)
- Program the pico with a provided test .uf2 we give you (no need to compile it using VS code)
- If all goes well -1 on screen, lights blinking, and keypad presses will do things



Bitwise Logic

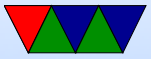
- You probably haven't seen this yet and won't until ECE275?
- In addition to math, computers good at boolean logic
- Your computer's CPU is essentially made up with a huge amount of this type of logic gates made up of transistors
- It is useful C exposes this to you (I've tried to do programming tasks in a version of BASIC without bitwise logic and it can be a pain)
- Things might make more sense if you take ECE275



Bitwise Logic

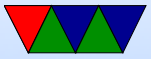
- This takes two numbers in binary and does the operation on each bit offset
- If you bitwise AND 0101 and 1111: bitwise AND input 1 bit 0 with input 2 bit 0 storing result in output bit 0, input 1 bit 1 with input 2 bit 1 storing result in output bit 1, etc

```
    1 0 1 0  0 1 0 1
 &  0 1 1 1  1 1 1 0
---------------------
    0 0 1 0  0 1 0 0
```



Truth Tables

- Simple digital logic operates on two bits input with one bit output
  (though when designing hardware this isn't always true)

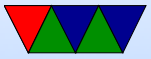


Bitwise And

- Just use &
- Output only true if both inputs true
- `a=b&c;`

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

```
    1 0 1 0  0 1 0 1
 &  0 1 1 1  1 1 1 0
--------------------
    0 0 1 0  0 1 0 0
```



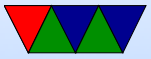
AND Aside

- ANDing against a value like 01110 the bits with 0s will be forced to 0 but with 1 will stay the same
- This is sometimes called masking
- For example, to mask (set to 0) all but the lowest bit do

```
char result=x&0b00000001;
```

- Also note, if you MOD by a power of two (let's call it X) then it's equivalent to AND with X-1

```
x=14%4;      // x will be 2  (14/4 = 3R2)
x=14&(4-1); // x will also be 2 ( 1110 & 0011 = 0010)
```

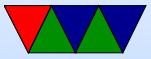


Bitwise Or

- Just use |
- Output true if either input true
- `a=b|c;`

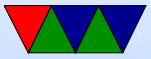
Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	1

```
    1 0 1 0   0 1 0 1
 |  0 0 1 0   1 1 1 0
----------------------
    1 0 1 0   1 1 1 1
```



Or Aside

- ORing against a value like 01110 the bits with 0s stay the same but with 1s it will force it to 1 if not already set
- If you want to force the lowest bit of a number on, you can do something like `char ch=value|0b00000001;`

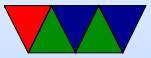


Exclusive Or (XOR)

- Just use ^
- Output true if inputs are different
- `a=b^c;`

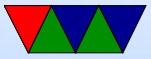
Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

```
   1 0 1 0  0 1 0 1
 ^ 0 0 1 0  1 1 1 0
--------------------
   1 0 0 0  1 0 1 1
```



Exclusive Or (XOR) Aside

- In hardware you can build an adder out of XOR and AND gates
- XORing against a value like 01110 the bits with 0s stay the same but with 1s it will flip the bit
- If you XOR with all 1s it will invert the value
- XORing a number with itself will set it to 0
  Some machines (x86) this is the fastest way to set a value to 0 in assembly language

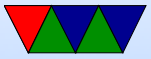


Bitwise Not (invert bits)

- Just use ~
- Flips all bits: 0 to 1, 1 to 0 (one's complement)
- `a=~b;`

Input 1	Output
0	1
1	0

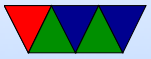
```
 ~   1 0 1 0   0 1 0 1
---------------------
     0 1 0 1   1 0 1 0
```



Shift Left

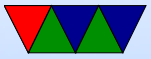
- Just use `<<`
- NOTE: this does *not* print anything (that is weird C++ syntax sugar)
- Shifts all bits to the left. The top bit is thrown away, the bottom bit gets a 0
- `a=a<<b` where b is the times to shift

```
char r,c=0b0011 1111; // 0x3f (63)
r=c<<1; // r will now be be 0111 1110 (0x7e) (126)
```



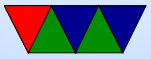
Shift Left aside

- Note: shifting left by 1 is same as multiplying by 2
- UNDEFINED BEHAVIOR: on x86 shifting a 32-bit int left by 32 might leave the number the same, rather than 0 as you might expect



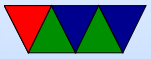
Shift Right

- Just use `>>`
- Shifts all bits to the right
  - For unsigned values it is a logical shift, you shift in a 0 at the top and throw away the bottom bit
  - For signed values this can be *UNDEFINED BEHAVIOR* but usually it is an arithmetic shift (sign-preserving) you preserve the sign bit in the top bit when shifting
- `a=a>>b` where b is the times to shift



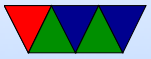
Shift Right Aside

- Note: shifting right by 1 is same as dividing by 2
- MORE UNDEFINED BEHAVIOR: on x86 shifting a 32-bit int right by 32 might leave the number the same, rather than 0 as you might expect



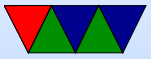
Where do shifted off bits go?

- In old days joke is they ended up in the "bitbucket"
- If your computer crashed, they'd joke the bitbucket was full and you'd have to empty it
- In assembly language shifts often end up in the "Carry Flag" which you can check, and that is handy in certain situations. Unfortunately it is not easy to get the value of that flag from C



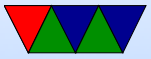
Show off some Lovebyte Examples

- Size-coding demoparty that was usually held around Valentine's Day
- Based in the Netherlands
- Isn't happening in 2026 :(
- Entries are 16, 32, 64, 128, 256, 512 bytes
- Note a simple C "hello world" executable in Linux tends to be tens of *kilobytes* so orders of magnitude bigger than the submissions in this competition

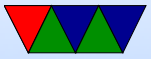


Lovebyte 2022 – Thumbpinksi

- 128 byte Linux executable that prints a purple "Sierpinski Triangle" pattern to the text console
- This pattern is from a bitwise AND of X and Y co-ordinates
- `https://www.pouet.net/prod.php?which=90925`
- It is really hard to make Linux executables this small. It is written in ARM-THUMB assembly language, and does extreme hacks like putting some code in the header.
- An article on the ELF hacking can be found here:

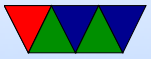


https://tmpout.sh/3/08.html



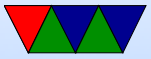
Lovebyte / Bitwise Aside

- bitwise AND of X and Y will get you Sierpinski Triangles
- bitwise XOR of X and Y will get you squares
- You can get lots of complicated mathematical patterns with various combinations of the bitwise operators we learned about



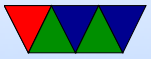
Lovebyte 2022 – Sierzoom

- Sierpinski, but roto-zooming in
- This is Linux/ARM but 256 bytes
- `https://www.pouet.net/prod.php?which=91035`



Lovebyte 2025 – Blackhole

- This is a 64-byte Apple II entry from 2025
- 6502 Assembly Language
- `https://www.pouet.net/prod.php?which=103581`



IOCC2024 – Starpath

- This was an Obfuscated C code entry that didn't win, I don't actually have a good website for it yet
- It's using "sixel" mode where some Linux terminals let you draw bitmap graphics into the terminal with special escape sequences
- It's based on the 64-byte DOS demo "starpath" by Hellmood
  `https://www.pouet.net/prod.php?which=103622`