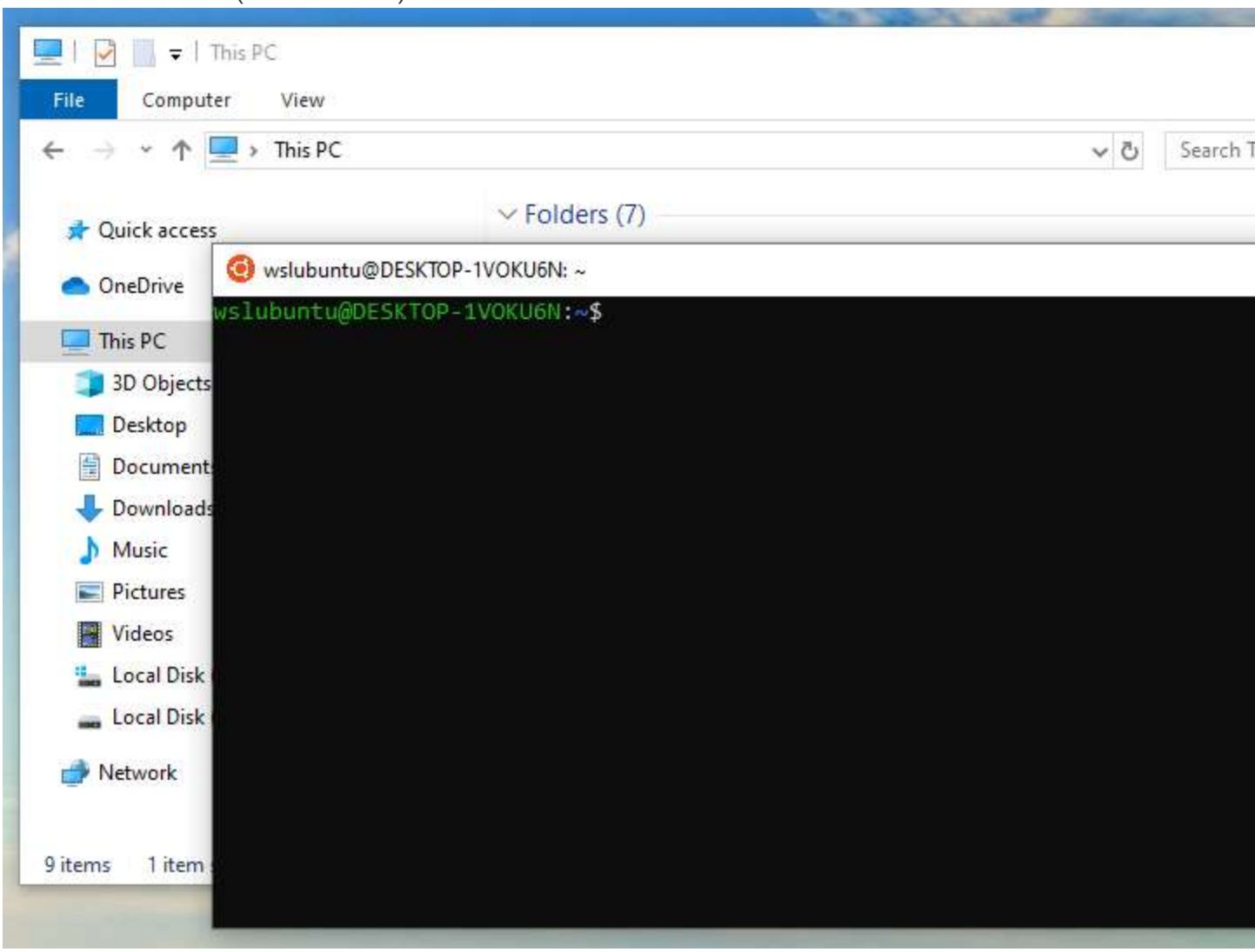
# Use faster WSL instead of VirtualBox

VirtualBox has been slow and sometimes buggy for some of you. Windows 10 now includes a more integrated Linux environment called WSL (Windows Subsystem for Linux). You can pretty now much run console-mode Ubuntu (or other distros) within windows without the overhead of a virtual machine!




I got it all working on my system, including building GCC and related tools from the latest source, configured for the RISC-V ISA. Here's the ready-to-go package that you can just extract and use quickly. It includes the precompiled toolchain, the precompiled latest update to my programmer utility, the otter-tools package from Canvas with some new tweaks to the Makefile such as programmer integration, and some handy scripts:

https://drive.google.com/open?id=1_Bb2TqdxsyVu4neeWUekvD-xLif6Ta9B

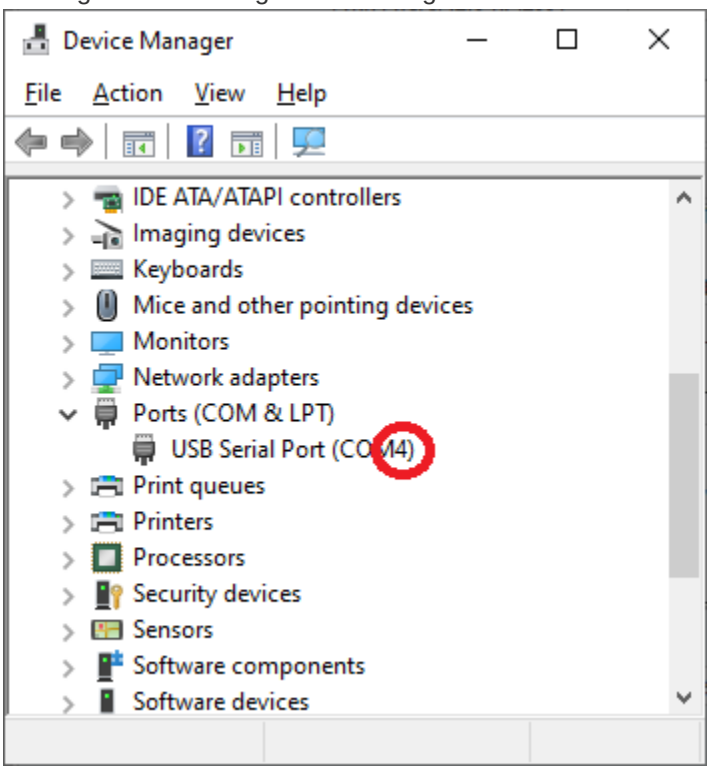
Follow the instructions here to enable WSL and install Ubuntu in it, including the initialization steps linked near the bottom:

https://docs.microsoft.com/en-us/windows/wsl/install-win10

Now that WSL-Ubuntu is installed with the standard set of packages, run its terminal by start-searching for "ubuntu", then install one more package:

```
sudo apt install build-essential
```

Going forward, to access files between the systems:
* In WSL-Ubuntu, you can access Windows files under /mnt/c/...
* In Windows, you can access WSL-Ubuntu files under \\wsl$\Ubuntu\...
In Windows, with your board connected and powered, determine the COM port number assigned to it, by running Device Manager and looking under Ports:




In WSL-Ubuntu, install my otter tools/scripts package from the top link above (change the tgz path as needed):

```
tar zxf /mnt/c/users/keefe/downloads/otter_tools_wsl.tgz
source otter_tools/scripts/setup_env.sh
```

Update the port setting that the programmer uses (do this step again in the future if the number changes, e.g. different board or usb port):

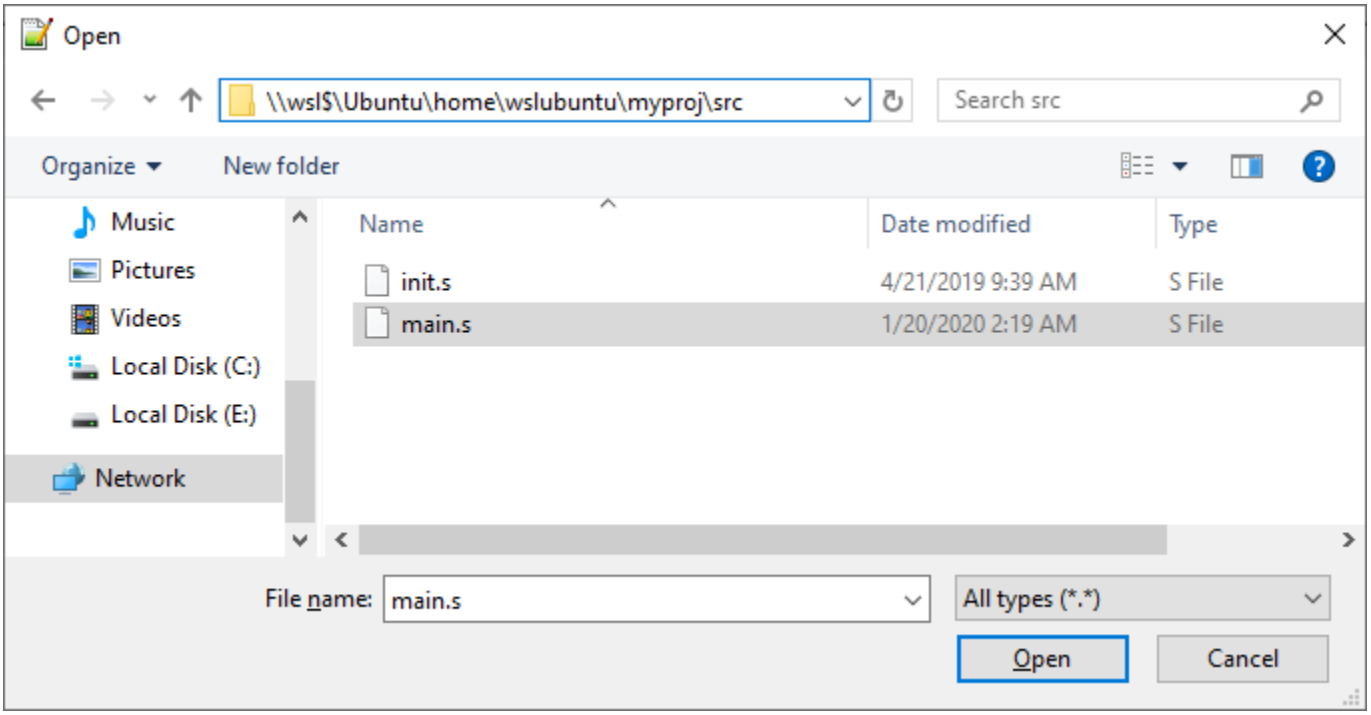
```
set_com 4
```

Now you can start a new project:

```
create_project_asm myproj
cd myproj
```

(You can also use create_project_c instead, with the only difference being it starts you out with the empty main function coded in C instead of assembly. You can always mix C and assembly source files in the same project regardless of how you started. You just can't have two main's, and you can't have two source files with the same name but just different extensions. One more catch though: if you use any C code, you should edit src/init.s to call _start instead of main, as the C compiler assumes certain stuff is initialized before main, in particular the global pointer register gp. We'll figure out a cleaner way of automating this distinction later.)

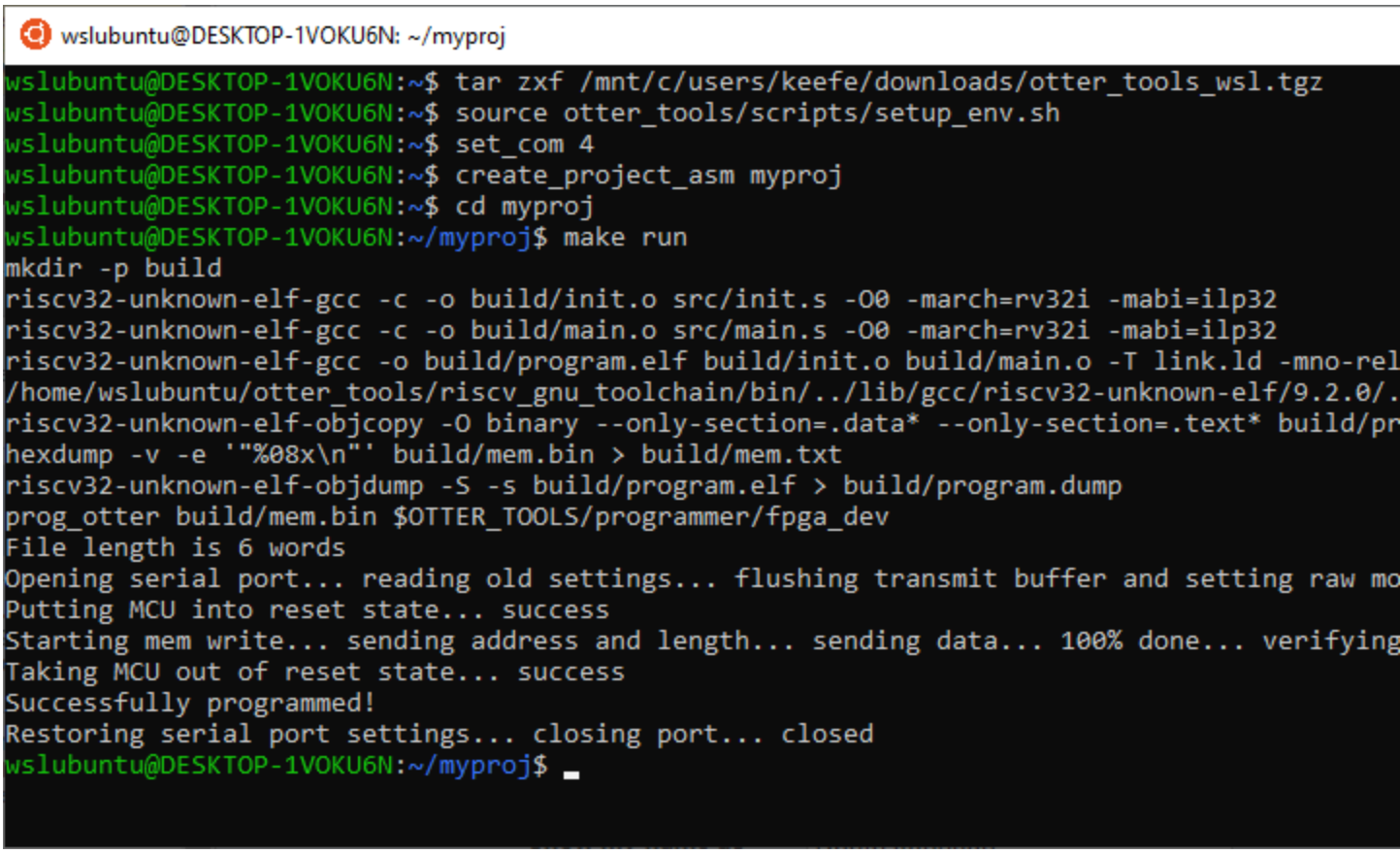
Edit src/main.s as you want (the default main function is blank), either with vim or nano in the terminal, or with your favorite Windows editor:




To run it on the board (this also builds the project if needed):

```
make run
```

This assumes your board is already running a programmable OTTER (see my previous post @13).
In total, everything should look like this (not including the initial setup of Ubuntu, e.g. apt stuff):


wslubuntu@DESKTOP-1VOKU6N: ~/myproj
wslubuntu@DESKTOP-1VOKU6N:~$ tar zxf /mnt/c/users/keefe/downloads/otter_tools_wsl.tgz
wslubuntu@DESKTOP-1VOKU6N:~$ source otter_tools/scripts/setup_env.sh
wslubuntu@DESKTOP-1VOKU6N:~$ set_com 4
wslubuntu@DESKTOP-1VOKU6N:~$ create_project_asm myproj
wslubuntu@DESKTOP-1VOKU6N:~$ cd myproj
wslubuntu@DESKTOP-1VOKU6N:~/myproj$ make run
mkdir -p build
riscv32-unknown-elf-gcc -c -o build/init.o src/init.s -O0 -march=rv32i -mabi=ilp32
riscv32-unknown-elf-gcc -c -o build/main.o src/main.s -O0 -march=rv32i -mabi=ilp32
riscv32-unknown-elf-gcc -o build/program.elf build/init.o build/main.o -T link.ld -mno-rel
/home/wslubuntu/otter_tools/riscv_gnu_toolchain/bin/../lib/gcc/riscv32-unknown-elf/9.2.0/.
riscv32-unknown-elf-objcopy -O binary --only-section=.data* --only-section=.text* build/pr
hexdump -v -e '"%08x\n"' build/mem.bin > build/mem.txt
riscv32-unknown-elf-objdump -S -s build/program.elf > build/program.dump
prog_otter build/mem.bin $OTTER_TOOLS/programmer/fpga_dev
File length is 6 words
Opening serial port... reading old settings... flushing transmit buffer and setting raw mo
Putting MCU into reset state... success
Starting mem write... sending address and length... sending data... 100% done... verifying
Taking MCU out of reset state... success
Successfully programmed!
Restoring serial port settings... closing port... closed
wslubuntu@DESKTOP-1VOKU6N:~/myproj$