Lab 0: Environment Setup

Contents

Introduction

In Lab 0, you need to prepare the environment for future development. You should install the target toolchain, and use them to build a bootable image for rpi3.

Goals of this lab

- Set up the development environment.
- Understand what's cross-platform development.
- Test your rpi3.

Important

This lab is an introductory lab. It won't be part of your final grade, but you still need to finish all `todo` parts, or you'll be in trouble in the next lab.

Cross-Platform Development

Cross Compiler

Rpi3 uses ARM Cortex-A53 CPU. To compile your source code to 64-bit ARM machine code, you need a cross compiler if you develop on a non-ARM64 environment.

Todo

Install a cross compiler on your host computer.

Linker

You might not notice the existence of linkers before. It's because the compiler uses the default linker script for you. (`ld --verbose` to check the content) In bare-metal programming, you should set the memory layout yourself.

This is an incomplete linker script for you. You should extend it in the following lab.

```
SECTIONS
{
  . = 0x80000;
  .text : { *(.text) }
}
```

QEMU

In cross-platform development, it's easier to validate your code on an emulator first. You can use QEMU to test your code first before validating them on a real rpi3.

Warning

Although QEMU provides a machine option for rpi3, it doesn't behave the same as a real rpi3. You should validate your code on your rpi3, too.

Todo

Install `qemu-system-aarch64`.

From Source Code to Kernel Image

You have the basic knowledge of the toolchain for cross-platform development. Now, it’s time to practice them.

From Source Code to Object Files

Source code is converted to object files by a cross compiler. After saving the following assembly as `a.S`, you can convert it to an object file by `aarch64-linux-gnu-gcc -c a.S`. Or if you would like to, you can also try llvm’s linker `clang -mcpu=cortex-a53 --target=aarch64-rpi3-elf -c a.S`, especially if you are trying to develop on macOS.

```
.section ".text"
_start:
  wfe
  b _start
```

From Object Files to ELF

A linker links object files to an ELF file. An ELF file can be loaded and executed by program loaders. Program loaders are usually provided by the operating system in a regular development environment. In bare-metal programming, ELF can be loaded by some bootloaders.

To convert the object file from the previous step to an ELF file, you can save the provided linker script as `linker.ld`, and run the following command.

```
# On GNU LD
aarch64-linux-gnu-ld -T linker.ld -o kernel8.elf a.o
# On LLVM
ld.lld -m aarch64elf -T linker.ld -o kernel8.elf a.o
```

From ELF to Kernel Image

Rpi3's bootloader can't load ELF files. Hence, you need to convert the ELF file to a raw binary image. You can use `objcopy` to convert ELF files to raw binary.

```
aarch64-linux-gnu-objcopy -O binary kernel8.elf kernel8.img
# Or
llvm-objcopy --output-target=aarch64-rpi3-elf -O binary kernel8.elf kernle8.img
```

Check on QEMU

After building, you can use QEMU to see the dumped assembly.

```
qemu-system-aarch64 -M raspi3b -kernel kernel8.img -display none -d in_asm
```

Todo

Build your first kernel image, and check it on QEMU.

Deploy to REAL Rpi3

Flash Bootable Image to SD Card

To prepare a bootable image for rpi3, you have to prepare at least the following stuff.

- An FAT16/32 partition contains
  - Firmware for GPU.
  - Kernel image.(kernel8.img)

There are two ways to do it.

1. We already prepared a bootable image.

   You can use the following command to flash it to your SD card.

```
dd if=nycuos.img of=/dev/sdb
```

Warning

/dev/sdb should be replaced by your SD card device. You can check it by *lsblk*

It's already partition and contains a FAT32 filesystem with firmware inside. You can mount the partition to check.

2. Partition the disk and prepare the booting firmware yourself. You can download the firmware from raspberrypi/firmware

   bootcode.bin, fixup.dat and start.elf are essentials. More information about pi3's booting could be checked on the official website
   https://www.raspberrypi.org/documentation/configuration/boot_folder.md
   https://www.raspberrypi.org/documentation/hardware/raspberrypi/bootmodes/README.md

   Finally, put the firmware and your kernel image into the FAT partition.

   Important

   Besides using `mkfs.fat -F 32` to create a FAT32 filesystem, you should also set the partition type to FAT.

Todo

Use either one of the methods to set up your SD card.

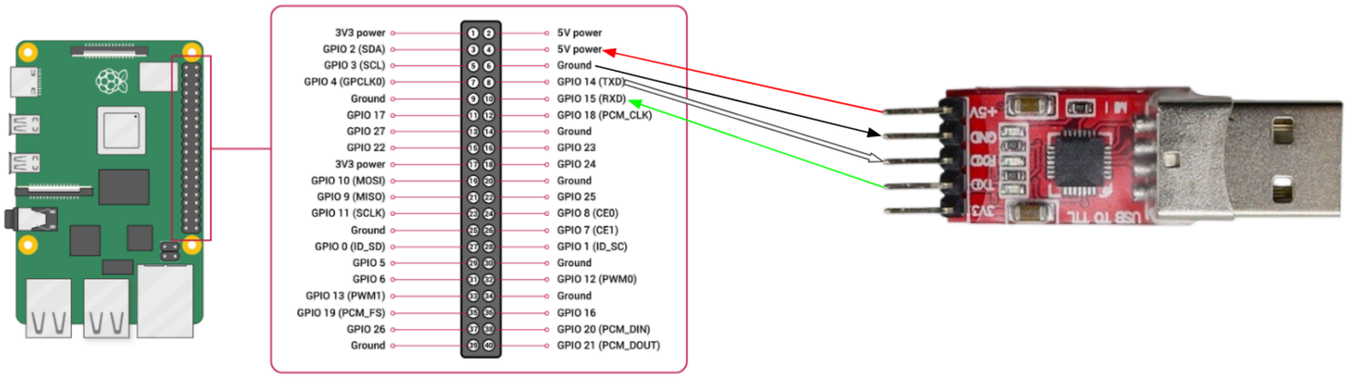
Interact with Rpi3

In our provided bootable image, it contains a kernel image that can echoes what you type through UART. You can use it to test if your Lab kits function well.

1. If you use method 2 to set up your bootable image, you should download kernel8.img , and put it into your boot partition. It's identical to the one in the provided bootable image.

2. Plug in the UART to USB converter to your host machine, and open it through a serial console such as screen or putty with the correct baud rate.
3. Connect TX, RX, GND to the corresponding pins on rpi3, and turn on your rpi3. You can follow the picture below to set up your UART.
4. After your rpi3 powers on, you can type some letters, and your serial console should print what you just typed.

```
screen /dev/ttyUSB0 115200
```



Debugging

Debug on QEMU

Debugging on QEMU is a relatively easier way to validate your code. QEMU could dump memory, registers, and expose them to a debugger. You can use the following command waiting for gdb connection.

```
qemu-system-aarch64 -M raspi3b -kernel kernel8.img -display none -S -s
```

Then you can use the following command in gdb to load debugging information and connect to QEMU.

```
file kernel8.elf
target remote :1234
```

Important

Your gdb should also be cross-platform gdb.

Debug on Real Rpi3

You could either use print log or JTAG to debug on a real rpi3. We don’t provide JTAG in this course, you can try it if you have one. https://metebalci.com/blog/bare-metal-raspberry-pi-3b-jtag/

Previous
Overview

Next
Lab 1: Hello World