National Yang Ming Chiao Tung University
Computer Architecture & System Lab

Operating System Capstone

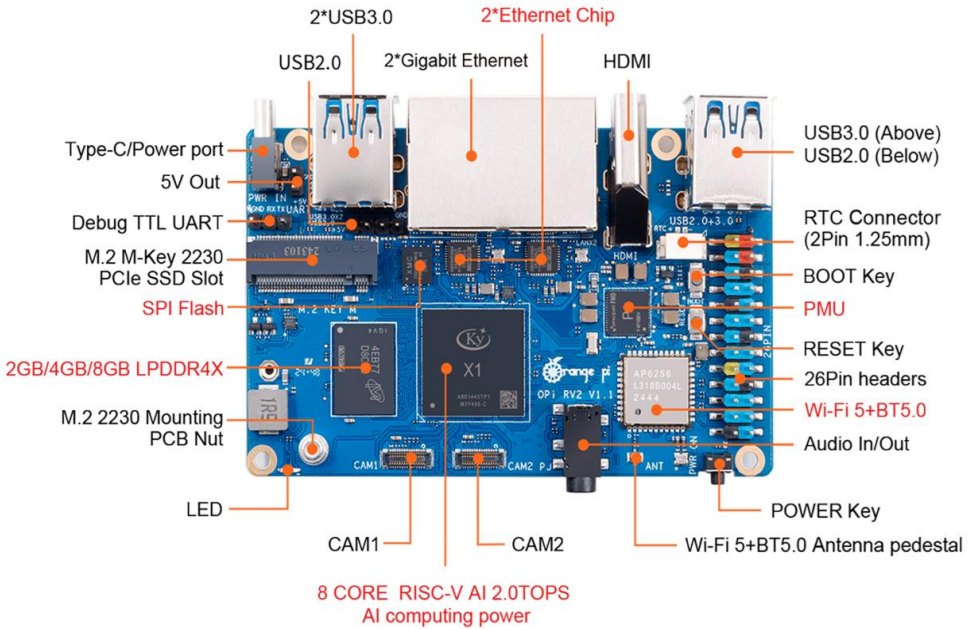
Lab0: Environment Setup

Overview

- OrangePi RV2
- Cross Compiler
- Linker
- Deploy to OrangePi RV2
- From Source code to Kernel Image
- Debug

OrangePi RV2

- Architecture: 64-bit RISC-V



Cross Compiler

What is a Cross Compiler?

A tool used to compile 64-bit RISC-V machine code on a non-RISC-V host environment (e.g., x86 Linux).

Installation

Tool name: **gcc-riscv64-unknown-elf**

```
$ sudo apt update
$ sudo apt install gcc-riscv64-unknown-elf
```

Linker Script

Role of the Linker

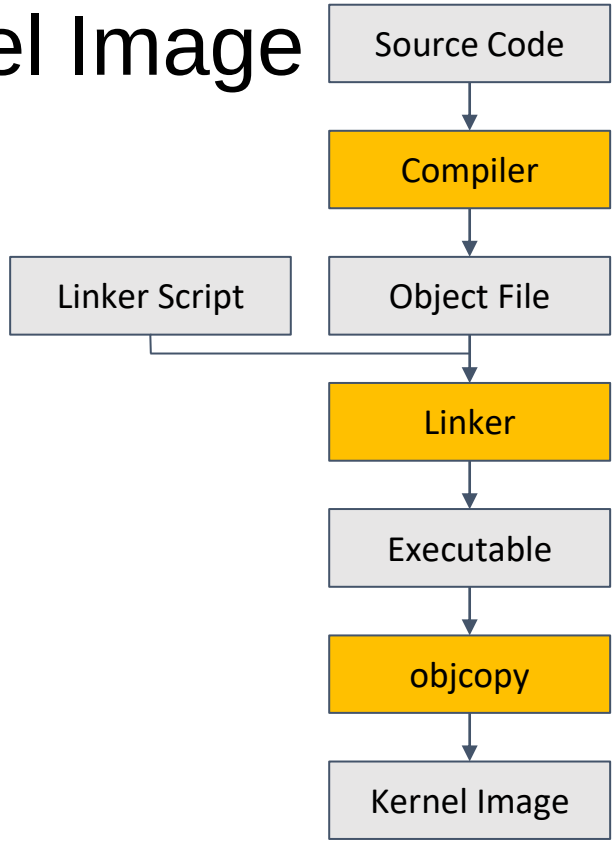
In bare-metal programming, you are responsible for defining the memory layout manually.

The linker script specifies where different sections of your code (text, data, bss) will reside in memory.

Sample Script (linker.ld)

```
SECTIONS
{
 . = 0x80200000;
 .text : { *(.text) }
 }
```

Build Flow: Source to Kernel Image



1.Compilation

Convert assembly (.S) or C source files into object

```
$ riscv64-unknown-elf-gcc -c a.S -o a.o
```

2. Linking

Link object files into an Executable and Linkable

```
$ riscv64-unknown-elf-ld -T linker.ld -o kernel.elf a.o
```

3. **Binary Conversion**

Extract the raw binary image from the ELF file for

```
$ riscv64-unknown-elf-objcopy -O binary kernel.elf kernel.bin
```

Deploy to OrangePi RV2: FIT Image

Flattened Image Tree (FIT)

The OrangePi bootloader expects the kernel in FIT format.

FIT packages the kernel binary and the Device Tree Blob (DTB) together.

Packaging Tool

Use **mkimage** from the **u-boot-tools** package.

```
$ sudo apt-get install u-boot-tools
$ mkimage -f src/kernel.its kernel.fit
```

```
/dts-v1/;
/ {
    description = "U-boot FIT Image for Orange Pi RV2";
    #address-cells = <2>;
    images {
        kernel {
            description = "Kernel Image";
            data = /incbin/("kernel.bin");
            type = "kernel";
            arch = "riscv";
            os = "linux";
            compression = "none";
            load = <0x0 0x00200000>;
            entry = <0x0 0x00200000>;
        };
        fdt {
            description = "Flat Device Tree";
            data = /incbin/("x1_orangepi-rv2.dtb");
            type = "flat_dt";
            arch = "riscv";
            compression = "none";
            load = <0x0 0x31000000>;
        };
    };
    configurations {
        default = "config-1";
        config-1 {
            description = "NYCU OSC RISC-V KERNEL";
            kernel = "kernel";
            fdt = "fdt";
        };
    };
};
```

src/kernel.its

Deploy to OrangePi RV2: FIT Image (Step 2: SD Card)

Flashing the prebuilt Bootable Image

Use the **dd** command to write the bootable image to your SD card:

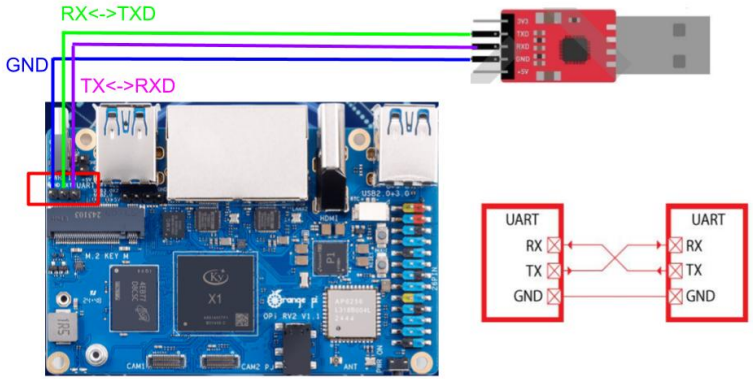
```
$ sudo dd if=image.img of=/dev/sdX bs=4M status=progress conv=fsync
```

Replace /dev/sdX with your actual device name (check the device name using lsblk)

Interact with OrangePi RV2

- After setting up your bootable image, connect the board through a UART-to-USB adapter to your host machine.
- Screen the serial port using a serial console program

```
$ sudo screen /dev/ttyUSB0 115200
```



```
[    6.619]       Load Address: 0x31000000
[    6.623]    Verifying Hash Integrity ... OK
[    6.627]    Loading fdt from 0x110002cc to 0x31000000
[    6.632]    Booting using the fdt blob at 0x31000000
[    6.737]    Loading Kernel Image
[    6.745]    Loading Device Tree to 000000007dd71000, end 000000007dd8f8aa ...
OK

Starting kernel ...


NYCU OSC RISC-V KERNEL
```

QEMU Emulator

- A software Emulator that helps you test code on host

```
$ qemu-system-riscv64 -M virt -kernel kernel.bin -display none -d in_asm
```

-M virt
Selects the "virt" virtual machine model
-kernel kernel.bin
Loads kernel.bin directly into guest memory
-display none
Disables all graphical display output
-d in_asm
Enables QEMU instruction-level tracing

```
----------------
IN:
Priv: 3; Virt: 0
0x8000aedc:  b7e5            j                 -24               # 0x8000aec4

----------------
IN:
Priv: 3; Virt: 0
0x8000aec4:  8552            mv                a0,s4
0x8000aec6:  85d6            mv                a1,s5
0x8000aec8:  30200073        mret

----------------
IN:
Priv: 1; Virt: 0
0x80200000:  10500073        wfi
0x80200004:  bff5            j                 -4                # 0x80200000
```

Debugging Techniques

- Install **gdb-multiarch** (optionally GEF)
- Start QEMU paused and listen at port:1234 for incoming

```
$ qemu-system-riscv64 … -S -s
```

-
```
$ gdb-multiarch
(gdb) file kernel.elf
(gdb) target remote:1234
```

GDB Usage