

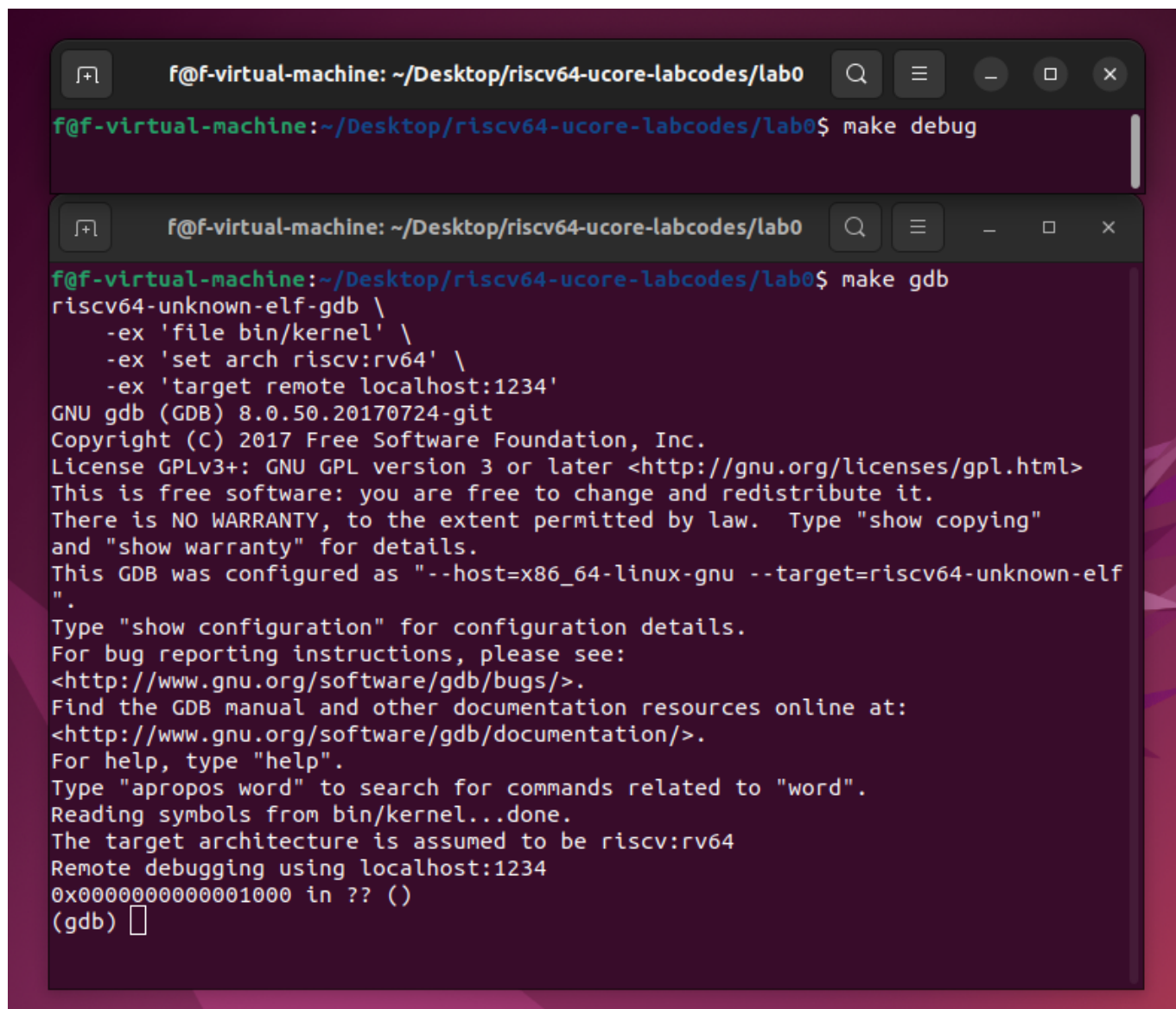
lab0.5: 比麻雀更小的麻雀 (最小可执行内核)

练习 1: 使用 GDB 验证启动流程

为了熟悉使用 qemu 和 gdb 进行调试工作, 使用 gdb 调试 QEMU 模拟的 RISC-V 计算机加电开始运行到执行应用程序的第一条指令 (即跳转到 `0x80200000`) 这个阶段的执行过程, 说明 RISC-V 硬件加电后的几条指令在哪里? 完成了哪些功能?

Answer

一、使用 `make debug` 和 `make gdb` 指令



```
f@f-virtual-machine: ~/Desktop/riscv64-ucore-labcodes/lab0
f@f-virtual-machine:~/Desktop/riscv64-ucore-labcodes/lab0$ make debug

f@f-virtual-machine: ~/Desktop/riscv64-ucore-labcodes/lab0
f@f-virtual-machine:~/Desktop/riscv64-ucore-labcodes/lab0$ make gdb
riscv64-unknown-elf-gdb \
  -ex 'file bin/kernel' \
  -ex 'set arch riscv:rv64' \
  -ex 'target remote localhost:1234'
GNU gdb (GDB) 8.0.50.20170724-git
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=x86_64-linux-gnu --target=riscv64-unknown-elf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
Reading symbols from bin/kernel...done.
The target architecture is assumed to be riscv:rv64
Remote debugging using localhost:1234
0x0000000000000100 in ?? ()
(gdb) □
```

二、加电开始运行后QEMU的执行过程及功能

(1) 上电时

在QEMU源代码中, 可以找到在RISC-V处理器上电时执行的初始指令序列。这些指令执行以下任务:

```
uint32_t reset_vec[10] = {
    0x00000297,          /* 1: auipc t0, %pcrel_hi(fw_dyn) */
    0x02828613,          /*      addi a2, t0, %pcrel_lo(1b) */
    0xf1402573,          /*      csrr a0, mhartid */
#ifdef TARGET_RISCV32
    0x0202a583,          /*      lw a1, 32(t0) */
    0x0182a283,          /*      lw t0, 24(t0) */
#elif defined(TARGET_RISCV64)
    0x0202b583,          /*      ld a1, 32(t0) */
    0x0182b283,          /*      ld t0, 24(t0) */
#endif
    0x00028067,          /*      jr t0 */
    start_addr,          /* start: .dword */
    start_addr_hi32,
    fdt_load_addr,       /* fdt_laddr: .dword */
    0x00000000,          /* fw_dyn: */
};
```

1. 读取当前核心的Hart ID，即核心标识寄存器（CSR）`mhartid`的值，并将其写入寄存器`a0`中。
2. 设置一个暂时未使用的目标寄存器`a1`，用于将来存储Flatten设备树（FDT）在物理内存中的地址，但在这个阶段尚未使用。
3. 执行跳转操作，转移到指定的`start_addr`地址，该地址在本实验中通常是RustSBI的入口点地址。

这些初始指令的目的是在RISC-V处理器上电后，进行一些基本的初始化工作，例如获取核心ID，准备FDT地址（尚未使用），然后跳转到RustSBI的执行入口点以启动后续的系统初始化过程。

(2) 上电后

加电开始运行后，QEMU 便依次执行如下步骤：

- **步骤1：**在RISC-V计算机的启动过程中，首先将程序计数器（PC）初始化为地址`0x1000`。随后，执行一系列引导代码指令，这些指令可能包括初始化一些基本寄存器和硬件设置。然后，通过一系列操作，程序计数器(PC)被设置为地址`0x80000000`，这个地址是RISC-V系统中引导加载程序的入口点。这标志着进入了**步骤2**。
- **步骤2：**在**步骤1**中，程序计数器已经跳转到地址`0x80000000`，这是引导加载程序的起始地址。在这一步，引导加载程序被加载到内存中，通常称为RustSBI。RustSBI负责执行一系列初始化任务，其中包括硬件初始化，如串口等外设的设置。在完成这些初始化任务后，RustSBI通过使用`mret`指令，将程序计数器(PC)设置为内存中操作系统内核的起始地址`0x80200000`。
- **步骤3：**程序计数器(PC)现在指向了内存中操作系统内核的起始地址`0x80200000`。在这个步骤中，操作系统内核镜像从该地址开始加载到内存中。操作系统内核进行进一步的初始化工作，包括但不限于内存管理、进程初始化等。一旦内核完成这些初始化任务，它会使用`sret`指令将程序计数器(PC)设置为应用程序的第一行代码的地址，从而正式启动并执行应用程序。