

PA2 实验报告

王卫东 221900332

2023 年 11 月 24 日

一 必做题

1 理解 YEMU 如何执行程序

1.1 实例程序状态机

对于 $(PC, r0, r1)$, 有: $(0, y, ?) \rightarrow (1, y, y) \rightarrow (2, x, y) \rightarrow (3, x + y, y) \rightarrow (4, x + y, y)$

1.2 指令执行

程序执行:

- `exec_once` 函数:
- 取指: 从内存中取得由程序计数器 (`pc`) 指向的指令。
- 译码: 通过切换操作码 (`this.rtype.op` 或 `this.mtype.op`) 来解码指令, 确定指令类型。
- 执行: 执行由操作码指定的操作。
- 更新程序计数器: 将程序计数器递增, 指向下一条指令。

主函数:

它运行一个循环, 调用 `exec_once` 直到 `halt` 标志被设置。循环后, 打印计算结果, 该结果存储在内存位置 `M[7]` 中。

示例程序:

内存中的程序从 `M[6]` 和 `M[5]` 加载值, 将它们相加, 并将结果存储在 `M[7]` 中。值 16, 33 和 0 初始化在内存位置 `M[6]`, `M[5]` 和 `M[7]` 中。

2 整理一条指令在 NEMU 中的执行过程

不妨考虑 `auipc` 指令：

1. `auipc` 用于将一个无符号立即数加到当前程序计数器 (PC) 的高 20 位，形成一个 32 位的立即数，然后存储到目标寄存器中。在上述代码中，`auipc` 的匹配模式为
 - `INSTRPAT("??????? ???? ???? ? ???? 00101 11", auipc, U, R(rd) = s->pc + imm);`
2. 在 `decode_operand` 函数中，根据 `TYPE_U`，即 `auipc` 的类型，调用 `immU()` 解析出立即数。然后，在执行阶段，将当前 PC 加上立即数，并将结果存储到目标寄存器 `rd` 中。
3. 这样，`auipc` 指令就完成了将无符号立即数加到当前 PC 的高 20 位的操作。

3 打字小游戏究竟是如何在计算机上运行的？

- 硬件层 (AM)：游戏通过 AM 层来与计算机硬件进行交互。在这个层次上，游戏使用了 AM 的 GPU 模块（图形处理单元）和键盘模块。GPU 负责绘制游戏界面，而键盘模块用于接收键盘输入。
- 指令集架构层 (ISA)：在这个层次上，游戏使用了一些特定的指令来操作寄存器、内存，以及与输入输出相关的操作。例如，`io_read` 和 `io_write` 函数用于读取和写入 I/O 设备（键盘和屏幕）。
- 运行时环境层 (NEMU)：游戏的主逻辑通过调用 `game_logic_update` 和 `render` 函数来进行更新和渲染。NEMU 负责模拟指令的执行、寄存器和内存的管理。
- 应用程序层：打字小游戏是一个应用程序，主要由 C 语言编写。在这个层次上，游戏实现了一些游戏逻辑，比如生成字符、处理键盘输入、更新游戏状态等。通过调用 `check_hit` 函数，游戏检查键盘输入是否命中了正在下落的字符。

游戏的运行过程可以概括为：

游戏逻辑通过 `game_logic_update` 函数进行更新，其中包括生成新的字符、更新字符的位置等。用户的键盘输入通过 `io_read(AM_INPUT_KEYBRD)` 获取，并在 `check_hit` 函数中进行处理。游戏界面通过 `render` 函数进行渲染，包括绘制下落的字符、统计得分等。

4 static inline 开头定义的 inst_fetch() 函数

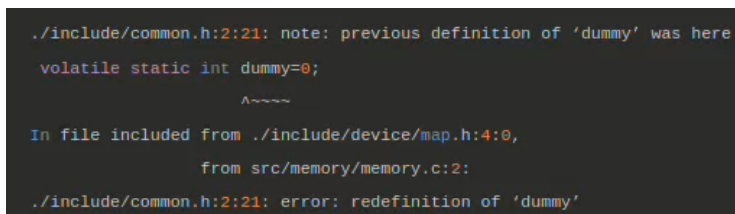
- 去掉两者：出现报错：“`hostcall.c:(.text+0x0): multiple definition of inst_fetch; src/isa/riscv32/inst.o:inst.c:(.text+0xea0): first defined here`”，多个源文件中包含相同的函数的定义（而不是声明）时，如果这些定义都带有 `static` 关键字，就会导致多重定义错误（multiple definition error）。这是因为每个源文件被编译成一个独立的目标文件，而每个目标文件都包含了相同的静态定义，这违反了链接阶段的规则。
- 分别去掉 `static/inline`：没有 `trace` 到明显错误，程序会正常运行（估计调用函数开销增大了）

5 重新编译后的 NEMU 含有多少个 dummy 变量的实体？debug.h 中添加一行同样命令，进行比较；为两处 dummy 变量进行初始化，你发现了什么问题？为什么之前没有出现这样的问题？

在 `build/obj` 中利用 `grep -r -c 'dummy' ./ * | grep '0:[1-9]' | wc -l` 命令得出共有 33 个。

仍然是 33 个。

两个初始化后会出现多次定义的错误：



```
./include/common.h:2:21: note: previous definition of 'dummy' was here
volatile static int dummy=0;

~~~~~

In file included from ./include/device/map.h:4:0,
                 from src/memory/memory.c:2:
./include/common.h:2:21: error: redefinition of 'dummy'
```

图 1: 报错

原因是强弱定义的问题，当两个 dummy 都没有初始化的时候 dummy 是一个弱符号，编译器不会报错，编译器会选择占用内存最大的那个弱符号，当把两个 dummy 都初始化后，两个 dummy 就变成强符号了，链接器不允许强符号被多次定义，如果一个是强符号一个是弱符号，那么弱符号会被强符号覆盖 (当然这个弱符号的占用内存大小不能大于强符号，否则会报错)。

6 make 程序如何组织.c 和.h 文件，最终生成可执行文件 am-kernels/kernels/hello/build/hello-\$ISA-nemu.elf

首先来看 am-kernels/kernels/hello/ 目录下的 Makefile：

```
NAME = hello
SRCS = hello.c
include $(AM_HOME)/Makefile
```

这里包含的 Makefile 在 abstract-machine 目录下。

执行来观察一下：

```
# Building hello-image [riscv32-nemu]
+ CC hello.c
# Building am-archive [riscv32-nemu]
+ CC src/platform/nemu/trm.c
+ CC src/platform/nemu/ioe/ioe.c
+ CC src/platform/nemu/ioe/timer.c
+ CC src/platform/nemu/ioe/input.c
+ CC src/platform/nemu/ioe/gpu.c
+ CC src/platform/nemu/ioe/audio.c
+ CC src/platform/nemu/ioe/disk.c
+ CC src/platform/nemu/mpe.c
+ AS src/riscv/nemu/start.S
+ CC src/riscv/nemu/cte.c
+ AS src/riscv/nemu/trap.S
+ CC src/riscv/nemu/vme.c
+ AR -> build/am-riscv32-nemu.a
# Building klib-archive [riscv32-nemu]
+ CC src/cpp.c
+ CC src/string.c
+ CC src/stdlib.c
+ CC src/stdio.c
+ CC src/int64.c
+ AR -> build/klib-riscv32-nemu.a
+ LD -> build/hello-riscv32-nemu.elf
# Creating image [riscv32-nemu]
+ OBJCOPY -> build/hello-riscv32-nemu.bin
```

大概过程如下：

1. 构建 hello 镜像，生成.o 文件
2. 构建 am 库，通过生成的.o 文件生成.a 文件
3. 构建 klib 库，通过生成的.o 文件生成.a 文件
4. 链接，通过之前生成的 hello 镜像和两个库，生成 elf 文件
5. 反汇编 elf 文件得到 txt 文件
6. 通过 objcopy 将 elf 文件转换为 bin 文件（NEMU 可执行的镜像文件）

二 实验心得

一是在处理子问题时不要将问题复杂化。框架代码本身已经给出了非常明确的提示，只要理解每个知识点背后的原理，代码还是很容易实现的。

二是要学会编写工具来提高调试效率，在实现【sdb】、【trace】和【difftest】等工具的过程中，对于程序行为更加了解，更能够解决相应的问题。

三是全面地阅读框架代码，了解整个框架的运作（这个目前还没完成，还需要花这个周末再 review 一下，同时完成比较完整的 printf 代码）。

总体来说，pa2 要更考验耐心和细节，要认真地完成各个选做题以及每个可能出错的小问题。（RTFSC 占据巨量的时间 ~）