实验报告

211830115 郑九铭

实验进度：我完成了所有内容。

必做题 1（程序是个状态机）：（1）在 YEMU 中，执行加法指令时会发生改变的寄存器只有 PC 和两个参与加法的寄存器，即(PC,ADD1,ADD2),则状态机为：(0,a,b)->(1,a+b,b)。
(2)YEMU 执行指令的方式是：依照 PC 的值取出指令，译码，得到所需操作数和操作码，依照对应规则执行，最后将 PC 更新。
(3)两者具有以下联系：都反映出了程序的本质是状态机，其下一状态由输入和当前状态共同决定。

必做题 2：（RTFSC）：在 NEMU 中，一条指令的进行即一次 exec_once()函数的调用过程；此时将 Decode(s)中的 pc 和 snpc 更新为当前 pc，方便译码器调用；接着进入 isa 相关的取指环节，由于我所选指令集是 riscv_32 架构，指令长度为定长，故从当前 pc 所指的内存地址中连续取 4 个字节取出指令；之后进入译码环节，利用 INSPART 进行模式匹配识别当前指令，并将其行为用具体的函数实现。返回到 exec_once()函数时当前 PC 将被更新为 dnpc，之后进行 trace 与 difftest，这样一条指令的执行就完成了。

必做题 3：（程序如何运行）打字小游戏首先通过调用 ioe_init 和 video_init 来检测设备是否实现及初始化设备；之后程序的主体部分是一个 while(1)的循环，在每次循环的过程中，其先根据当前的帧数更新 frames 次游戏逻辑，之后再进入一个 while(1)的内层循环检测当前按键，若 ESC 按键按下则退出程序；若无按键按下则退出内层循环继续更新游戏逻辑；若有按键按下则调用 check_hit 函数处理该按键是 hit/miss/wrong 的哪种情况，并继续。

具体地，当玩家按下一个字母并命中的时候，check_hit 函数在程序内部处理逻辑，使 hit++；同时改变按下的那个字母的 char[m].v；render()函数调用了 AM 的 I/O 接口，将符合当前游戏逻辑的画面写入 NEMU 的 mmio 地址中，以此访问 VGA 设备，播放画面。

必做题 4：（编译与链接）仅去掉 static 或仅去掉 inline 均不报错，而去掉两者会在编译时出现如下报错：

原因：去掉两者时该函数即为普通的函数；由于 ifetch.h 被其它多个文件所 include，而其他文件在调用该函数时 inst_fetch 这一函数符号在整个文件中被多次定义，导致出错；
仅去掉 static，编译器将 inst_fetch 视作内联函数，在其它文件调用它时直接内联，故不会多次定义；
仅去掉 inline，inst_fetch 是静态成员函数，仅在 ifetch.h 中被定义，故不会多次定义；

必做题 5：（编译与链接）
（1）有 35 个。因为在执行 grep -r -c 'dummy'指令后，观察到有 35 个.o 文件中含有 1 个“dummy”

（2）还是 35 个。因为 debug.h 中已经 include 了 common.h，而且由于 dummy 没有初始化，故其并不是强符号，因此上述 35 个.o 文件中依然只有一个“dummy”。

（3）出现如图所示报错：

```
+ CC src/device/device.c
In file included from src/device/device.c:16:
/home/znm/ics2022/nemu/include/common.h:49:21: error: redefinition of ‘dummy’
   49 | volatile static int dummy = 0;
      |                     ^~~~~
In file included from /home/znm/ics2022/nemu/include/common.h:47,
                 from src/device/device.c:16:
/home/znm/ics2022/nemu/include/debug.h:43:21: note: previous definition of ‘dummy’ with type ‘int’
   43 | volatile static int dummy = 0;
      |                     ^~~~~
make: *** [/home/znm/ics2022/nemu/scripts/build.mk:34: /home/znm/ics2022/nemu/build/obj-riscv32-nemu-interpreter/src/device/device.o] Error 1
```

原因是初始化后的 dummy 成为了强符号，故在别的文件中同时 include common.h 和 debug.h 时出现了重复定义强符号的报错。

必做题 6：（了解 Makefile）在 am-kernels/kernels/hello/目录下敲入 make ARCH=$ISA-nemu 后，首先进入的是 hello 文件夹下的 makefile，其引用了$AM-HOME 中的 makefile 文件，并将 hello.c 作为参数 src,hello 作为参数 name 传入；之后在$AM-HOME 中的 makefile 里面根据定义好的编译和链接的规则，以及生成 elf 文件的方式进行 build，并依据 nemu.mk 和 riscv32.mk 两文件中的相关配置提供设备信息和 isa 相关的编译选项，最终完成编译，生成 hello-$ISA-nemu.elf

实验心得

（1）深刻地感受到了指令集体系的重要性，容不得半点马虎，毕竟指令集错了会酿成“牵一发而动全身”的恶果。

（2）深刻地感受到了 PA1 中基础设施的重要性。在完成 PA2-1 时有许多程序“hit bad trap”，PA1 中的单步调试和监视点帮我相当有效率地完成了 debug（至少感觉自己的 PA2-1 完成得还是挺快的，没有受到太大阻碍）

（3）还是吐槽一下 PA2 框架代码里面悄悄挖的坑是真的多，例如在实现时钟时，由于 NEMU 中 timer.c 的坑爹的硬件实现，导致我在实现__am_timer_uptime 时必须要先调用 inl(RTC_ADDR + 4)再调用 inl(RTC_ADDR)，否则在运行跑分程序时会出现浮点数异常（似乎有很多同学都遇到了这个问题）；在我读源码的过程中发现原因是每次调用 inl 时都会调用 invoke_callback，若先调用 inl(RTC_ADDR)，RTC_ADDR + 4 的值会停留在“上一次”而导致出错。

（4）鸣谢姜凯同学，胡德谞同学提供的许多建议和帮助。

（5）

- 按时完成，拒绝拖延
  - 这样你才有时间做到上面几点

可以说体会得还是很深刻。（还好，这次感觉没怎么拖延）