PA2实验报告

实验进度

我完成了所有的必做内容。

遇到的问题及思考

“实现更多的指令”部分

- 实现完指令后测试mul-longlong.c时报错，解决方法：使用sdb逐条执行指令并察看结果，发现RV32中乘法指令实现有误。

实现printf

- 考虑如何实现`%[flags][width][.precision][length]specifier`的格式时想了一会，最后意识到只要按从左到右顺序贪婪匹配即可。

"看看NEMU跑多快"部分

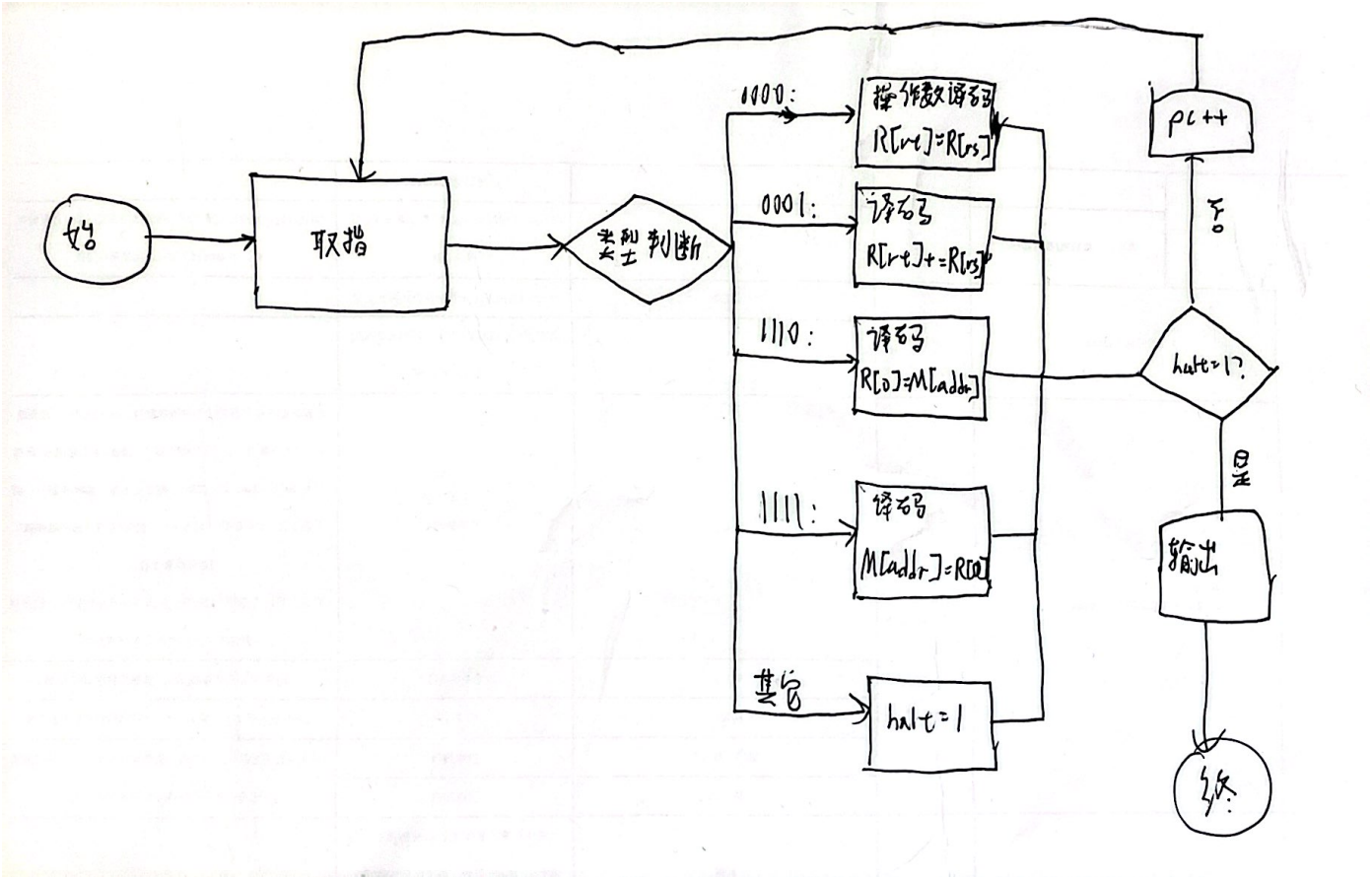
- 由于时钟源原因跑分过慢（更换时钟源后跑分加快几十倍），按照所给链接中内容设置也未解决问题。目前还未找到解决方案。

其他

- 本地所有功能测试OK但是在OJ上只Partially Accepted，察看后发现提示“Wrong Answer (maybe frame error) LiteNES emulator (uses printf).”尝试运行LiteNES emulator时发现长时间加载ROM后调用了不应该调用的函数并退出。尝试ftrace无果后意识到长时间加载ROM可能和klib中mem相关函数实现错误有关，debug时发现错误实现了memcpy（将大于号和小于号打反了），修改后通过。

必答题

- 程序是个状态机



状态机如图。在YEMU中执行一条指令先用`this.inst = M[pc];`取值，再根据指令类型译码操作码和操作数，之后根据操作码对操作数进行相应操作，最后更新pc。若未找到指令对应的操作码则将halt赋为1并退出。

- RTFSC

在NEMU中执行一条指令先在`cpu-exec.c`中的`exec_once`中取指令，再在`inst.c`中通过`decode_exec`函数译码操作码和操作数，之后根据操作码对操作数进行相应操作（`INSTPAT`函数在`decode.h`中定义），最后在`cpu-exec.c`中的`exec_once`中将`pc`更新为下一条指令地址。

- 程序如何运行
- 打字小游戏初始时获取屏幕宽高，提前求出全屏空白和26个字母3种颜色的显示信息并存放。然后程序进入死循环，循环中不停获取时间并算出当前游戏进行到第几帧，若帧数有更新则运行`game_logic_update`。
  - 在`game_logic_update`中，先检查帧数判断是否应该在屏幕上添加一个新的字符，再将每个字符的状态进行更新。
- 之后，循环检查是否有键盘输入，若有键按下则运行`check_hit`察看是否命中以更新hit/wrong。
- 最后察看当前帧是否被渲染，若未被渲染则运行`render()`并更新已渲染帧。
  - 渲染时先将上一帧渲染的所有字母位置清空，然后检查存储字母的结构体`chars`，并根据每个字母的状态选择颜色并渲染，最后打印提示信息。

- 编译与链接

分别去掉`static`和`inline`编译时不会发生错误，但是同时去掉会报错`multiple definitions of inst_fetch`，这是因为在`inst.c`中也定义了名为`inst_fetch`的（不同功能的）函数，链接时产生冲突。加上`static`后不产生冲突的原因是这样做使`ifetch.h`中的`inst_fetch`仅在包括该头文件中的`.c`文件中可见，从而不与`inst.c`中的`inst_fetch`调用产生冲突。加上`inline`后不产生冲突的原因是这样做使编译器将函数内联，即通过插入函数内的语句"模仿"函数调用，也即链接后包括`ifetch.h`的`.c`文件中仅包含`inst_fetch`的代码，链接过程中符号表中没有`inst_fetch`的符号。

猜测可通过检查链接中包含`inst_fetch`符号的部分检验`static`的作用，通过确认包括`ifetch.h`的`.c`文件编译时不存在符号`inst_fetch`检验`inline`的作用。

- 编译与链接

1. 在`nemu/build/obj-riscv32-nemu-intepreter`中运行`grep -lnr dummy | wc -l`可得答案为33。
2. 在`nemu/build/obj-riscv32-nemu-intepreter`中运行`grep -lnr dummy | wc -l`可得答案为33，与1中数量相同，这是因为`debug.h`中包含了`common.h`，因此包含`debug.h`的文件包含`common.h`中对`dummy`的定义，所以在`debug.h`中增加对`dummy`的定义不会增加`dummy`的数量。
3. 只申明变量而不初始化是弱定义，因此链接时链接器会从其中二选一而不会报错；都初始化后存在两个强定义的`dummy`，链接时会出错。

- 了解Makefile

通过在`~/ics2023/am-kernels/kernels/hello`中执行`make -n ARCH=$ISA-nemu`并加以分析可知Makefile生成`am-kernels/kernels/hello/build/hello-$ISA-nemu.elf`的过程如下：

- `am-kernels/kernels/hello/Makefile`声明了NAME和SRCS参数，并使用了`$(AM_HOME)/Makefile`中的内容，可看作是生成流程跳转到了`$(AM_HOME)/Makefile`。
- 在`$(AM_HOME)/Makefile`中，
  - 运行一些对生成目标和目录的检验
  - 设置更多生成相关的参数，如工作目录，目标生成目录，目标生成文件，`.o`文件，库文件等等。
  - 设置编译时的flag文件等，主要设置了编译器，架构头文件，以及编译`.c,.cpp,.a`等文件时用到的`flags`。
  - 将架构相关的`.mk`文件内容粘贴进来，该文件内进一步定义了`CFLAGS,COMMON_CFLAGS,LDFLAGS`中的内容，以及增加了一些架构相关的源文件。
  - 编译源文件。
    - 将源`.c`文件编译为`.o`文件。具体到本例而言，`hello.c->hello.o, trm.c->trm.o`
    - 将多个`.o`文件编译为`.a`文件。具体到本例而言，将包括`trm.o`以及设备相关`.o`文件编译为`am-riscv32-nemu.a`。不包括`hello.o`。
    - 将所有的`*.o`以及`*.a`文件链接为`.ELF`文件。具体到本例而言，将`hello.o`与`am-riscv32-nemu.a`通过`linker.ld`中的规则编译成`hello-riscv32-nemu.elf`文件