PA2 必答题

作者：熊浚丞 221240060

程序是个状态机

- 画出在YEMU上执行的加法程序的状态机 加法执行的状态机如下，其余未列出的M均不改变故没有写出。 pc = 0, R = {0,0}, halt = 0, M[7] = 0 -> pc = 1, R = {33,0}, halt = 0, M[7] = 0 -> pc = 2, R = {33,33}, halt = 0, M[7] = 0 -> pc = 3, R = {16,33}, halt = 0, M[7] = 0 -> pc = 4, R = {49,33}, halt = 0, M[7] = 0 -> pc = 5, R = {49,33}, halt = 0, M[7] = 49 -> pc = 5, R = {49,33}, halt = 1, M[7] = 49 终止状态（自环）
- 通过RTFSC理解YEMU如何执行一条指令 首先通过M[pc]取指，然后判断前四位的情况，按照R或M型指令进行提取参数，然后进行操作，并更新pc；若前四位不对应指令，则将halt设为1，停止。

RTFSC 请整理一条指令在NEMU中的执行过程

1. 在exec_once中调用isa_exec_once，准备执行指令
2. 执行inst_fetch，在内存的pc地址处读取，来获取指令内容，同时将静态下条pc设为当前pc+4
3. isa_exec_once再调用decode_exec
4. decode_exec的过程：
    1. 将动态下条pc暂时设为静态下条pc
    2. 按顺序依次匹配指令模式，如果匹配成功，则根据指令类型提取参数到src1,rd等变量中
    3. 根据匹配到的指令执行相应功能
    4. 将0寄存器设为0，从而无须在指令执行时特判
5. 返回到exec_once，将pc设为dnpc

程序如何运行 理解打字小游戏如何运行

- main函数过程：

1. 初始化
2. 进入while循环，即游戏主内容
3. 在while循环中，首先获取到当前经过的时间帧数，然后反复更新游戏逻辑帧，直到逻辑帧达到经过的时间帧数
4. 逐个匹配输入的按键直到没有更多的未处理按键，如果按下退出键则终止游戏
5. 如果逻辑时间大于渲染时间，渲染一次并更新渲染时间

- 游戏逻辑帧更新：每秒设定了30帧，每秒出现5个新字符，因此每6个逻辑帧产生一个新字符。对于所有字符，每帧下降一段距离，如果超出屏幕则删除。
- 处理按键：如果字符在最下方且与按键相等，则匹配上，并将字符速度设为向上的很大速度，这样在y值为负时就会在逻辑帧中删除
- 渲染：先清空，然后将字符全部打印出来，再在终端重新打印分数的信息

编译与链接

- 在nemu/include/cpu/ifetch.h中, 你会看到由static inline开头定义的inst_fetch()函数. 分别尝试去掉static, 去掉inline或去掉两者, 然后重新进行编译, 你可能会看到发生错误. 请分别解释为什么这些错误会发生/不发生? 你有办法证明你的想法吗?

只有都去掉才会发生错误，发生了重复定义错误。static的效果是使函数仅在当前文件生效，加上后即使多个文件include了ifetch.h，也不会冲突；inline的效果是内联展开，相当于把函数内容复制到调用处，编译器不会将它进行冲突判定。两个都去掉后才会冲突。证明：发现该函数被hostcall.c和inst.c调用，而把前者的include删除并去掉相应代码后，就没有冲突了。

编译与链接

1. 在nemu/include/common.h中添加一行volatile static int dummy; 然后重新编译NEMU. 请问重新编译后的NEMU含有多少个dummy变量的实体? 你是如何得到这个结果的?

32个，因为修改后编译时，有32个文件被重新编译

2. 添加上题中的代码后, 再在nemu/include/debug.h中添加一行volatile static int dummy; 然后重新编译NEMU. 请问此时的NEMU含有多少个dummy变量的实体? 与上题中dummy变量实体数目进行比较, 并解释本题的结果.

32个，因为被重新编译的文件都一样，对于这些文件，是两个重复的弱定义，所以不变。

3. 修改添加的代码, 为两处dummy变量进行初始化:volatile static int dummy = 0; 然后重新编译NEMU. 你发现了什么问题? 为什么之前没有出现这样的问题? (回答完本题后可以删除添加的代码.)

报了很多重复声明的错误，因为这是两个重复的强定义，是不允许的。

了解Makefile

- 请描述你在am-kernels/kernels/hello/目录下敲入make ARCH=$ISA-nemu 后, make程序如何组织.c和.h文件, 最终生成可执行文件am-kernels/kernels/hello/build/hello-$ISA-nemu.elf. (这个问题包括两个方面:Makefile的工作方式和编译链接的过程.) 关于Makefile工作方式的提示:
    - Makefile中使用了变量, 包含文件等特性
    - Makefile运用并重写了一些implicit rules
    - 在man make中搜索-n选项, 也许会对你有帮助
    - RTFM

首先将NAME和SRCS都设置为要编译的文件hello.c，然后include了am的makefile（相当于复制）。在该makefile中，有许多变量和规则。首先检查了ARCH是不是可以支持的，不支持则会直接打印报错并终止。然后将ARCH分为ISA和平台。通过SRCS获取OBJS，并收集了需要链接的文件名。同时，从SRCS、ISA等已获取到的变量将各种需要用在c文件中的define放入了CFLAGS等选项中。最后，定义了许多用于编译、链接的规则，这些规则是根据形如$(DST_DIR)/%.o: %.c的命令定义的，它会把所有符合格式的文件都定义一个规则，并将他们加上相应的地址前缀，实现了批量定义规则。

编译链接的过程：

1. 将所有.c,.cc,.cpp和.S文件编译为DST_DIR路径下的.o文件，在编译时调用了相应文件类型的编译选项变量
2. 将.o文件和.a文件进行链接并生成IMAGE.elf文件，并调用了之前定义的编译选项变量