Programming Assignment 2

161220049 黄奕诚

October 8, 2017

I. 实验进度

完成所有内容。通过所有测试、跑分程序，并成功运行打字小游戏与超级玛丽。

II.必答题

1.编译与链接

考虑到 rtl_push 函数在所有程序中都有调用，所以我首先删去了 rtl_push 之前的 inline，发现如下报错：

```
broccoli@ubuntu:~/ics2017/nexus-am/tests/cputest$ make ALL=dummy run
Building dummy [x86-nemu]
Building am [x86-nemu]
make[2]: *** No targets specified and no makefile found.  Stop.
+ CC src/monitor/debug/ui.c
+ CC src/cpu/decode/decode.c
+ CC src/cpu/decode/modrm.c
In file included from ./include/cpu/decode.h:6:0,
                 from ./include/cpu/exec.h:9,
                 from src/cpu/decode/modrm.c:1:
./include/cpu/rtl.h:176:14: error: ‘rtl_push’ defined but not used [-Werror=unus
ed-function]
 static  void rtl_push(const rtlreg_t* src1) {
             ^
cc1: all warnings being treated as errors
Makefile:23: recipe for target 'build/obj/cpu/decode/modrm.o' failed
make[2]: *** [build/obj/cpu/decode/modrm.o] Error 1
/home/broccoli/ics2017/nexus-am/Makefile.app:35: recipe for target 'run' failed
make[1]: *** [run] Error 2
Makefile:12: recipe for target 'Makefile.dummy' failed
make: [Makefile.dummy] Error 2 (ignored)
 dummy
```

错误原因：static 函数作用域为文件内，如果没有该文件其他函数调用它，并且其他文件无法调用它。因此报错情况为：定义了却未使用。

其次我删去了 rtl_push 之前的 static，发现如下报错：

```
broccoli@ubuntu:~/ics2017/nexus-am/tests/cputest$ make ALL=dummy run
Building dummy [x86-nemu]
Building am [x86-nemu]
make[2]: *** No targets specified and no makefile found.  Stop.
+ CC src/cpu/decode/decode.c
In file included from ./include/nemu.h:6:0,
                 from ./include/cpu/exec.h:4,
                 from src/cpu/decode/decode.c:1:
./include/cpu/reg.h:68:31: error: ‘check_reg_index’ is static but used in inline function ‘rtl_push’ which is not static [-Werror]
 #define reg_l(index) (cpu.gpr[check_reg_index(index)]._32)
                               ^
./include/cpu/rtl.h:178:11: note: in expansion of macro ‘reg_l’
   rtl_sm(&reg_l(R_ESP), 4, src1);
           ^
In file included from ./include/cpu/decode.h:6:0,
                 from ./include/cpu/exec.h:9,
                 from src/cpu/decode/decode.c:1:
./include/cpu/rtl.h:178:3: error: ‘rtl_sm’ is static but used in inline function ‘rtl_push’ which is not static [-Werror]
   rtl_sm(&reg_l(R_ESP), 4, src1);
   ^
In file included from ./include/nemu.h:6:0,
                 from ./include/cpu/exec.h:4,
                 from src/cpu/decode/decode.c:1:
./include/cpu/reg.h:68:31: error: ‘check_reg_index’ is static but used in inline function ‘rtl_push’ which is not static [-Werror]
 #define reg_l(index) (cpu.gpr[check_reg_index(index)]._32)
                               ^
./include/cpu/rtl.h:177:28: note: in expansion of macro ‘reg_l’
   rtl_subi(&reg_l(R_ESP), &reg_l(R_ESP), 4);
                            ^
./include/cpu/reg.h:68:31: error: ‘check_reg_index’ is static but used in inline function ‘rtl_push’ which is not static [-Werror]
 #define reg_l(index) (cpu.gpr[check_reg_index(index)]._32)
                               ^
./include/cpu/rtl.h:177:13: note: in expansion of macro ‘reg_l’
   rtl_subi(&reg_l(R_ESP), &reg_l(R_ESP), 4);
           ^
In file included from ./include/cpu/decode.h:6:0,
                 from ./include/cpu/exec.h:9,
                 from src/cpu/decode/decode.c:1:
./include/cpu/rtl.h:177:3: error: ‘rtl_subi’ is static but used in inline function ‘rtl_push’ which is not static [-Werror]
   rtl_subi(&reg_l(R_ESP), &reg_l(R_ESP), 4);
   ^
cc1: all warnings being treated as errors
Makefile:23: recipe for target 'build/obj/cpu/decode/decode.o' failed
make[2]: *** [build/obj/cpu/decode/decode.o] Error 1
/home/broccoli/ics2017/nexus-am/Makefile.app:35: recipe for target 'run' failed
make[1]: *** [run] Error 2
Makefile:12: recipe for target 'Makefile.dummy' failed
make: [Makefile.dummy] Error 2 (ignored)
 dummy
broccoli@ubuntu:~/ics2017/nexus-am/tests/cputest$
```

错误原因：inline 关键字是“建议内联而不强制内联”，gcc 编译器中它独自不会内联，仍然要产生函数实体。若没有 static，编译器会认为它是全局函数，所以与普通函数无异。因此报错中提到了 rtl_push 并不是静态函数，而中间却包含静态内联函数（如 rtl_subi 等）

再者，我删去了 static inline，发现如下情况：

```
build/obj/cpu/decode/modrm.o: In function `rtl_subi':
/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: multiple definition of `rtl_push'
build/obj/cpu/decode/decode.o:/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: first defined here
build/obj/cpu/exec/control.o: In function `rtl_subi':
/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: multiple definition of `rtl_push'
build/obj/cpu/decode/decode.o:/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: first defined here
build/obj/cpu/exec/data-mov.o: In function `rtl_subi':
/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: multiple definition of `rtl_push'
build/obj/cpu/decode/decode.o:/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: first defined here
build/obj/cpu/exec/arith.o: In function `rtl_subi':
/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: multiple definition of `rtl_push'
build/obj/cpu/decode/decode.o:/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: first defined here
build/obj/cpu/exec/prefix.o: In function `rtl_subi':
/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: multiple definition of `rtl_push'
build/obj/cpu/decode/decode.o:/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: first defined here
build/obj/cpu/exec/cc.o: In function `rtl_subi':
/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: multiple definition of `rtl_push'
build/obj/cpu/decode/decode.o:/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: first defined here
build/obj/cpu/exec/logic.o: In function `rtl_subi':
/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: multiple definition of `rtl_push'
build/obj/cpu/decode/decode.o:/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: first defined here
build/obj/cpu/exec/special.o: In function `rtl_subi':
/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: multiple definition of `rtl_push'
build/obj/cpu/decode/decode.o:/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: first defined here
build/obj/cpu/exec/exec.o: In function `rtl_subi':
/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: multiple definition of `rtl_push'
build/obj/cpu/decode/decode.o:/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: first defined here
build/obj/cpu/exec/system.o: In function `rtl_subi':
/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: multiple definition of `rtl_push'
build/obj/cpu/decode/decode.o:/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: first defined here
build/obj/cpu/intr.o: In function `rtl_subi':
/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: multiple definition of `rtl_push'
build/obj/cpu/decode/decode.o:/home/broccoli/ics2017/nemu/./include/cpu/rtl.h:38: first defined here
collect2: error: ld returned 1 exit status
Makefile:41: recipe for target 'build/nemu' failed
make[2]: *** [build/nemu] Error 1
/home/broccoli/ics2017/nexus-am/Makefile.app:35: recipe for target 'run' failed
make[1]: *** [run] Error 2
Makefile:12: recipe for target 'Makefile.dummy' failed
make: [Makefile.dummy] Error 2 (ignored)
 dummy
```

错误原因：当全删去 static 和 inline 之后，会报出在 rtl_subi 中重复定义了 rtl_push，而这些重复定义分别发生在许多译码、执行文件的可执行文件中。因为此时 rtl_push 与普通函数无异，在链接时不会被当成静态内联函数处理，于是在链接各个可重定位目标文件的过程中，由于在许多文件中多次出现，继而引发多重定义，因此报错。

证明想法时，可以观察删去前后的汇编文件差异，这一步可以发现编译过程的差异（因为编译过程能够生成汇编文件，而链接过程可以由汇编代码反映；

2.编译与链接

1.重新编译后的 nemu 中有 29 个 dummy 变量的实体。我在 nemu/build 目录下，通过指令 readelf -a nemu > nemu.elf 获得 elf 头表，然后 grep dummy nemu.elf 中寻找名称为 dummy 的个数即可；

2.此时重新编译后的 nemu 中仍然有 29 个 dummy 变量的实体，与上题相同。因为 common.h 与 debug.h 中的 dummy 变量作用域都是本地，进行链接时，名称为 dummy 的变量都为弱符号，取其中一个即可。

3.编译时发现：

```
broccoli@ubuntu:~/ics2017/nemu$ make
+ CC src/memory/memory.c
In file included from ./include/nemu.h:4:0,
                 from src/memory/memory.c:1:
./include/common.h:26:21: error: redefinition of ‘dummy’
 volatile static int dummy = 0;
                     ^
In file included from ./include/common.h:10:0,
                 from ./include/nemu.h:4,
                 from src/memory/memory.c:1:
./include/debug.h:6:21: note: previous definition of ‘dummy’ was here
 volatile static int dummy = 0;
                     ^
Makefile:23: recipe for target 'build/obj/memory/memory.o' failed
make: *** [build/obj/memory/memory.o] Error 1
```

此时 debug.h 与 common.h 中 dummy 变量都是强符号，由于规定强符号至多只能有一个，所以会发生编译错误。

3.了解 Makefile

```
# Command to execute NEMU
NEMU_EXEC := $(BINARY) $(ARGS)

$(BINARY): $(OBJS)
    $(call git_commit, "compile")
    @echo + LD $@
    @$(LD) -O2 -o $@ $^ -lSDL2 -lreadline

run: $(BINARY)
    $(call git_commit, "run")
    $(NEMU_EXEC)

gdb: $(BINARY)
    $(call git_commit, "gdb")
    gdb -s $(BINARY) --args $(NEMU_EXEC)

clean:
    rm -rf $(BUILD_DIR)
count-c:
    find . -name *.c | xargs wc -l
count-h:
    find . -name *.h | xargs wc -l
count:
    find . -name *.[ch] | xargs wc -l
```

首先，当输入 make 时，首先 Makefile 会检索左侧的部分，若输入诸如 make run, make gdb, make count 的指令，便会对应地进行执行；倘若只输入了 make，而会执行$(BINARY): $(OBJS)部分；

其次，执行 git_commit，这里可以参考 Makefile.git 中的代码，即在编译之前将代码更新到远程库中；

接着，“compile”表示进行的是编译操作，echo+LD 如下，表明使用的是 gcc 编译器，而 include 的文件

```
# Compilation flags
CC = gcc
LD = gcc
INCLUDES  = $(addprefix -I, $(INC_DIR))
CFLAGS   += -O2 -MMD -Wall -Werror -ggdb $(INCLUDES)
```

```
# Files to be compiled
SRCS = $(shell find src/ -name "*.c")
OBJS = $(SRCS:src/%.c=$(OBJ_DIR)/%.o)
```

则是指定包含的文件，如上图所示，在/nemu/src 中的所有 c 文件都被包括在编译中，并进行汇编、链接，由此生存可执行文件。而第三行-O2 表明编译时进行二级优化。

具体的编译链接过程：首先/src 目录下所有的 c 文件先预处理成.i 文件，再通过 gcc 二级优化编译为.s 汇编代码，然后在汇编过程中生成可重定位目标文件，最后链接生成可执行文件。

III.问题&思索&心路历程

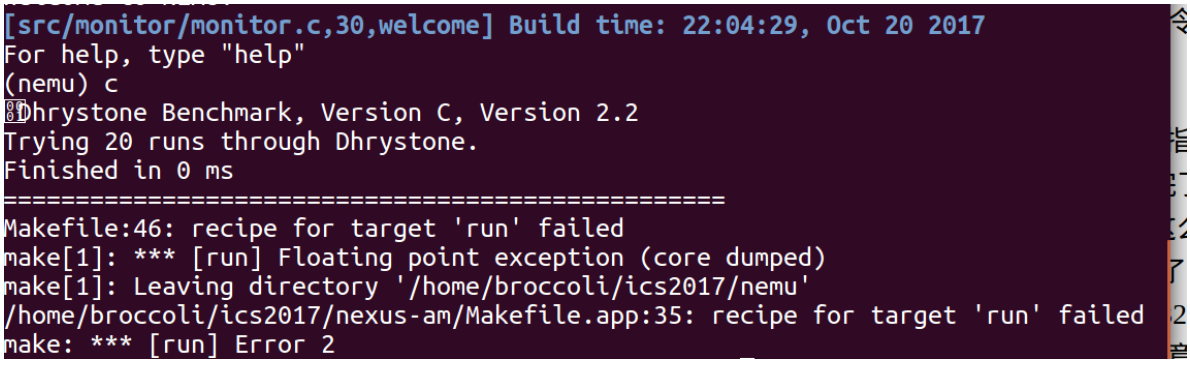
pa 2-1：首先在知道要做什么之前，我花了两三个小时 RTFSC 和阅读讲义，然后开始了 pa2 的征程；这六个指令实现得相当顺利，除了 sub 的执行函数中设置 eflags 费了点时间（后来我发现已实现的 sbb 操作可以拿来模仿），事实证明这时候我的 pop 指令写错了，不过这时我还没发现……

pa 2-2：在国庆去苏州游玩后回家的当天，当我看到讲义上给定的几排待实现指令上，我直接下手了！（后来才发现自己有多蠢），花了半天时间我终于一口气写完了所有要求的指令（是的……我从来没有测试过……），于是将讲义往后看，原来有这么多测试用例？KISS 法则？顿时发现自己应该对应着测试用例来实现指令……不管了，我先跑一下吧。What？32 个样例才过了 6 个？就这样，我又调了一整天 bug 才把 32 个样例全通过。“好的，这下肯定没 bug 了。”我自语道。这当然是个 flag。（请注意我到这时候还没发现有 diff-test 的存在……

第二天我又把讲义往后面看了。What?还有 diff-test？反正我用例全都通过了，应该影响不大的。开了 diff-test 后，我的 32 个用例出了 29 个红色。顿时心态有点崩。不过我很快接受这个残酷的事实……又调了一天 bug 后，我终于在 diff-test 模式下通过所有样例。那天天气真好。（其中有个 cmp 操作中把 id_dest 和 id_src 写反的事情我竟然看了几个小时都没看出来，被自己蠢哭）

国庆返校的前一天，我心血来潮想直接撸了 pa2-3，让 10 月份过得轻松一点……然后在键盘输入那个地方懵比了一个小时，才发现原来要在弹出的对话框中敲击键盘……有趣的是三个跑分样例+超级玛丽都没跑出来，我判断出这是 pa2-2 的锅……于是在回到学校后，开着 diff-test 调了一晚上 bug 终于把 pa2 所有要跑的用例、程序全跑出来了。那天晚上真的很兴奋！

一个很迷的问题：当我在/nemu/include/common.h 中注释掉 debug 后，跑 dhrystone 20 次会出现这个：



而加上 debug，则能够成功跑出分。若把 20 次改成 500000 次，两种情况下均能跑出分。我并没有找到合理的解释……

IV.蓝框思考题

1.关于 differential-test 能节省多少时间，我在不知道有 diff-test 的时候调了一整天 bug，还留了一些 bug，在知道它的存在后调了一晚上就跑出了超级玛丽和跑分程序，所以节省的时间应该至少为 300%吧（当然，这个没法精确地算，不过知道 diff-test 对 debug 的重要作用是必要的）

2.关于“为了创造出一个缤纷多彩的世界, 你觉得 NEMU 还缺少些什么呢?”，我们现在已经拥有了冯诺依曼计算机的一些基本结构、多数的 x86 指令集以及 IO 设备，离一台更加高能的计算机还差对异常的处理、文件系统、多进程、分页机制等重要功能，所以能做的事还是很有限的。

3.关于“如何检测死循环？”，首先死循环必然发生在循环体内，而一个循环体必然有条件判断语句；我们只需要对每个循环的条件判断语句执行次数进行计数，并设定一个非常大的上限，若计数值超过了这个上限，便判定死循环，输出语句并退出程序。

4.关于“volatile 关键词”，如下图所示，如果保留 volatile，则编译器不会进行优化，即每一步赋

```
void fun() {
    volatile unsigned char *p = (void *)0x8049000;
    *p = 0;
    while(*p != 0xff);
    *p = 0x33;
    *p = 0x34;
    *p = 0x86;
}
```

值都会在汇编代码中显示出来。如果去掉 volatile，则会进行优化，从而省略了中间对*p 的赋值，这对于设备寄存器，则将其中间的寄存过程省略，于是达不到预期的效果。

5.对于“如何检测多个键被同时按下？”，我的猜想是，首先读入其中一个键盘输入，将 status 设定为计数值(int)而不是布尔值，键按下时进行累加，键松开时进行递减，并在这个过程中依次对每个键进行识别、处理，其中需要判断同时按下的键号之间符不符合既定的一种组合方式。