

# Programming Assignment Report

## Stage 2

161180162 许致明

2017 年 10 月 28 日

### 摘要

我完成了本阶段的所有实验内容。在CPU指令方面，可以正确通过cputest、dhtystone、coremark、microbench中的所有测试项目，包括需要rotate（ROL/ROR/RCL/RCR）指令的MD5测试。在I/O指令方面，通过了hello、time-test、keytest、videotest这些测试内容，并能在NEMU的提供的支持下以11-15FPS的帧数（NEMU运行在Docker中）运行打字游戏。apps/litenes下的超级玛丽奥游戏由于NEMU性能太差（不足i7-6700的1/100），操作比较不便。

## 1 Answers to compulsory questions

### 1.1 static/inline关键字对RTL函数的影响

#### 1.1.1 去掉static关键字后的运行结果

在nemu/include/cpu目录下的rtl.h文件中，做以下更改：

```
- - -cpu/rtl.h
+ + +cpu/rtl.h
make_rtl_arith_logic(sltu)
- static inline void rtl_mul
+ inline void rtl_mul
```

编译结果（在nemu目录下执行make run）并未报错，同时程序也可以正确运行，能够通过coremark测试，如图1、图2所示。

#### 1.1.2 去掉inline关键字后的运行结果

在nemu/include/cpu目录下的rtl.h文件中，做以下更改：

```
- - -cpu/rtl.h
+ + +cpu/rtl.h
make_rtl_arith_logic(sltu)
- static inline void rtl_mul
+ static void rtl_mul
```

编译过程中就会产生错误，提示更改过的函数定义了但是没有使用，如图3所示。

图 1: 编译和运行默认镜像结果

```
tmux
+ CC src/cpu/exec/logic.c
+ CC src/cpu/exec/cc.c
+ CC src/cpu/exec/system.c
+ CC src/cpu/exec/arith.c
+ CC src/cpu/exec/exec.c
+ CC src/cpu/exec/data-mov.c
+ CC src/cpu/exec/control.c
+ CC src/cpu/exec/prefix.c
+ CC src/cpu/exec/special.c
+ CC src/cpu/decode/modrm.c
+ CC src/cpu/decode/decode.c
+ CC src/cpu/intr.c
+ LD build/nemu
./build/nemu -l ./build/nemu-log.txt
[src/monitor/monitor.c,47,load_default_img] No image is given. Use the default build-
in image.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 06:09:46, Oct 23 2017
For help, type "help"
(nemu) c
nemu: HIT GOOD TRAP at eip = 0x00100026

(nemu)
[0] 0:vim- 1:..apps/coremark 2:make* "e10a536e9ba0" 02:54 24-10月-17
```

图 2: 运行coremark结果

```
tmux
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 06:09:46, Oct 23 2017
For help, type "help"
(nemu) c
Running CoreMark for 1000 iterations
2K performance run parameters for coremark.
CoreMark Size      : 666
Total time (ms)    : 77043
Iterations         : 1000
Compiler version   : GCC6.3.0 20170516
seedcrc            : 0xe9f5
[0]crclist         : 0xe714
[0]crcmatrix       : 0x1fd7
[0]crcstate        : 0x8e3a
[0]crcfinal        : 0xd340
Finised in 77043 ms.
=====
CoreMark PASS      58 Marks
                   vs. 100000 Marks (i7-6700 @ 3.40GHz)
nemu: HIT GOOD TRAP at eip = 0x0010006e

(nemu) q
Steven@e10a536e9ba0 > ~/ics2017/nexus-am/apps/coremark > pa2 >
[0] 0:vim- 1:..apps/coremark* 2:make "e10a536e9ba0" 02:54 24-10月-17
```

图 3: 去掉inline后的编译结果

```
tmux
[src/monitor/monitor.c,47,load_default_img] No image is given. Use the default build-
in image.
Welcome to NEMU!
[src/monitor/monitor.c,30,welcome] Build time: 06:09:46, Oct 23 2017
For help, type "help"
(nemu) c
nemu: HIT GOOD TRAP at eip = 0x00100026

(nemu) q
Steven@e10a536e9ba0 ~/ics2017/nemu pa2 make run
+ CC src/cpu/exec/logic.c
In file included from ./include/cpu/decode.h:6:0,
                 from ./include/cpu/exec.h:9,
                 from src/cpu/exec/logic.c:1:
./include/cpu/rtl.h:46:24: error: 'rtl_mul' defined but not used [-Werror=unused-func-
tion]
static /*inline*/ void rtl_mul(rtlreg_t* dest_hi, rtlreg_t* dest_lo, const rtlreg_t*
src1, const rtlreg_t* src2) {
                        ^~~~~~
cc1: all warnings being treated as errors
Makefile:23: recipe for target 'build/obj/cpu/exec/logic.o' failed
make: *** [build/obj/cpu/exec/logic.o] Error 1
x Steven@e10a536e9ba0 ~/ics2017/nemu pa2
[0] 0:vim 1:..apps/coremark- 2:~/ics2017/nemu* "e10a536e9ba0" 03:03 24-10月-17
```

### 1.1.3 去掉两个关键字后的结果

在nemu/include/cpu目录下的rtl.h文件中，做以下更改：

```
- - -cpu/rtl.h
+ + +cpu/rtl.h
make_rtl_arith_logic(sltu)
- static inline void rtl_mul
+ void rtl_mul
```

编译不能通过，错误为函数进行了多重定义，如图4所示。

### 1.1.4 原因分析

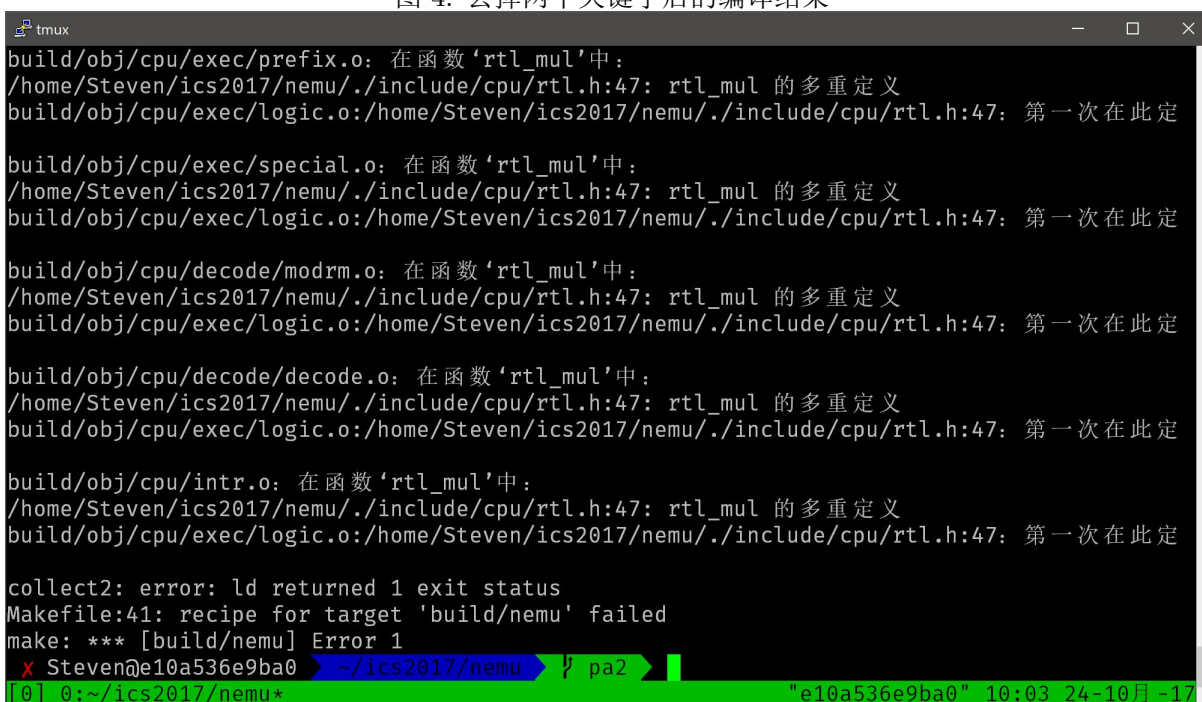
**static**关键字使得函数的生存期为整个程序的运行时间，而在包含它所在文件的文件中均可见。去掉这个关键字对函数影响不大。因为这些rtl函数体都很小，编译器会接受inline选项，将这些调用全部替换为相应的代码段。**inline**关键字我更改的是rtl\_mul函数，在nemu编译时的默认镜像没有使用这个函数，会产生一个“defined but not used”警告。而编译选项中将所有警告都视为错误，导致编译错误。将两个关键字都去掉之后，这个函数不再是全局的，在每个包含它所在头文件的文件中，都进行了一次定义，编译器会将这些重复定义视为错误。

## 1.2 头文件中增加变量

### 1.2.1 include/common.h中添加

编译后使用grep命令查找dummy，结果如图5所示。可以看到除了include/common.h中声明的一次之外，

图 4: 去掉两个关键字后的编译结果



```
tmux
build/obj/cpu/exec/prefix.o: 在函数 'rtl_mul' 中:
/home/Steven/ics2017/nemu/./include/cpu/rtl.h:47: rtl_mul 的多重定义
build/obj/cpu/exec/logic.o:/home/Steven/ics2017/nemu/./include/cpu/rtl.h:47: 第一次在此定

build/obj/cpu/exec/special.o: 在函数 'rtl_mul' 中:
/home/Steven/ics2017/nemu/./include/cpu/rtl.h:47: rtl_mul 的多重定义
build/obj/cpu/exec/logic.o:/home/Steven/ics2017/nemu/./include/cpu/rtl.h:47: 第一次在此定

build/obj/cpu/decode/modrm.o: 在函数 'rtl_mul' 中:
/home/Steven/ics2017/nemu/./include/cpu/rtl.h:47: rtl_mul 的多重定义
build/obj/cpu/exec/logic.o:/home/Steven/ics2017/nemu/./include/cpu/rtl.h:47: 第一次在此定

build/obj/cpu/decode/decode.o: 在函数 'rtl_mul' 中:
/home/Steven/ics2017/nemu/./include/cpu/rtl.h:47: rtl_mul 的多重定义
build/obj/cpu/exec/logic.o:/home/Steven/ics2017/nemu/./include/cpu/rtl.h:47: 第一次在此定

build/obj/cpu/intr.o: 在函数 'rtl_mul' 中:
/home/Steven/ics2017/nemu/./include/cpu/rtl.h:47: rtl_mul 的多重定义
build/obj/cpu/exec/logic.o:/home/Steven/ics2017/nemu/./include/cpu/rtl.h:47: 第一次在此定

collect2: error: ld returned 1 exit status
Makefile:41: recipe for target 'build/nemu' failed
make: *** [build/nemu] Error 1
x Steven@e10a536e9ba0 ~/ics2017/nemu pa2
[0] 0:~/ics2017/nemu+ "e10a536e9ba0" 10:03 24-10月 -17
```

在其他28个可重定位目标文件中均出现了dummy符号。

### 1.2.2 include/debug.h中添加

编译后继续使用grep命令查找，结果如图6所示。除了之前那些二进制文件外，build/nemu这个整体中也出现了一个dummy变量。

### 1.2.3 对这两处定义进行初始化

编译时，出现了重复定义的错误。原因是在未初始化时，这两个全局变量都只是弱符号，包含在.bss节。一旦进行了初始化，就会变为强符号，被保存在.data节。对强符号的重复定义是不被允许的，而在本例中对dummy进行了两次初始化，所以编译出错。

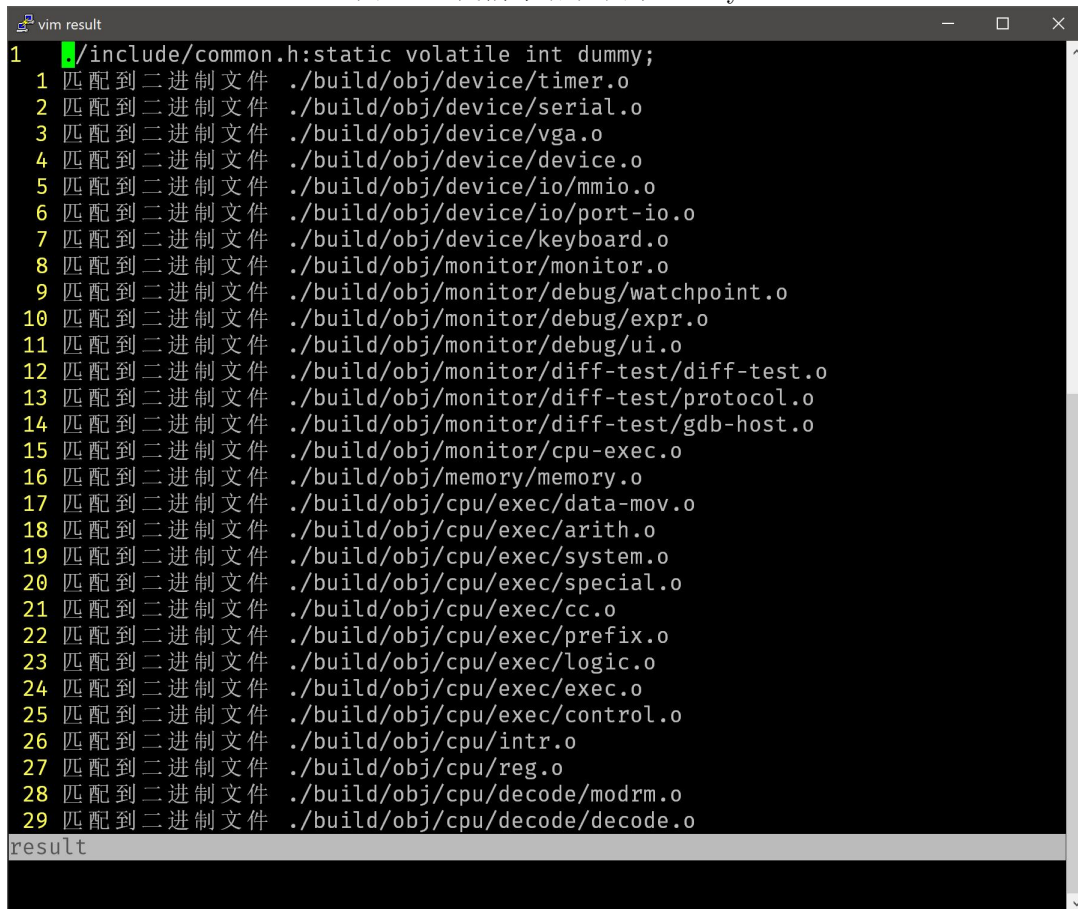
## 1.3 GNU make

### 1.3.1 make的工作方式

在shell中输入make命令之后，GNU make会进行以下工作：

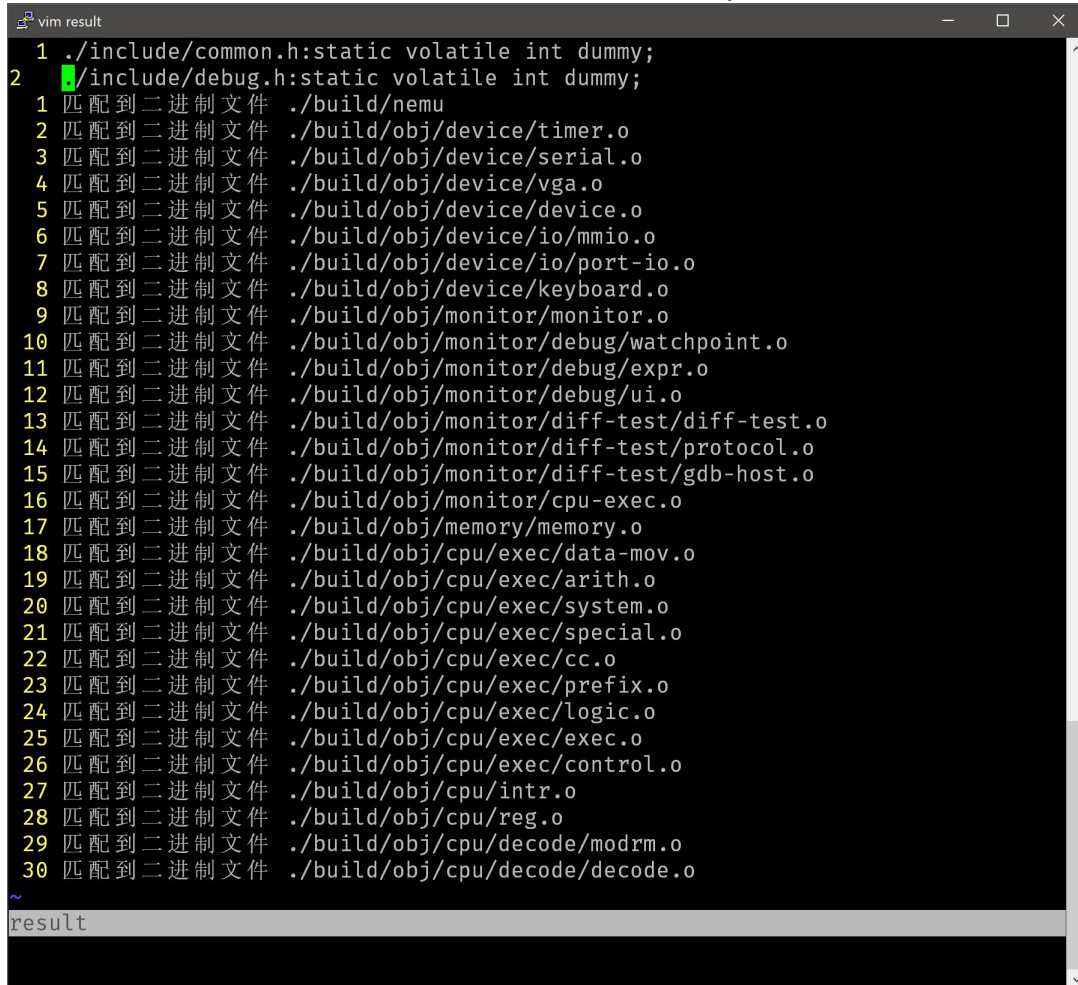
- 1.make在当前目录下寻找一个叫做“Makefile”或“makefile”的文件；
- 2.如果找到，make会寻找这个文件中的第一个目标文件，在nemu/Makefile中即为nemu，这个文件被作为最终的目标文件；
- 3.如果这个目标文件还不存在，或者它依赖的目标文件比它新，make就会执行后面定义的命令来生成的新的nemu文件；
- 4.递归查找nemu依赖的目标文件依赖的文件，然后按自底向上生成这些目标文件；

图 5: 查找编译结果中的dummy



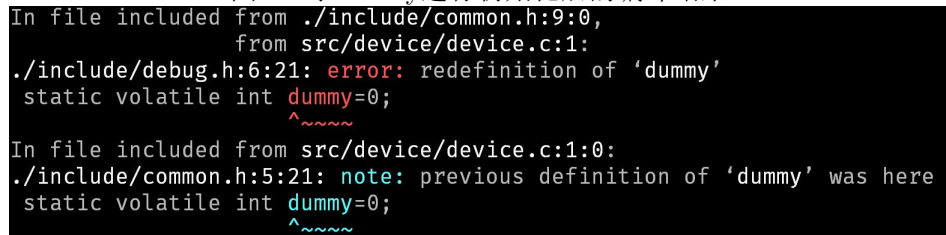
```
vim result
1 /include/common.h:static volatile int dummy;
2 匹配到二进制文件 ./build/obj/device/timer.o
3 匹配到二进制文件 ./build/obj/device/serial.o
4 匹配到二进制文件 ./build/obj/device/vga.o
5 匹配到二进制文件 ./build/obj/device/device.o
6 匹配到二进制文件 ./build/obj/device/io/mmio.o
7 匹配到二进制文件 ./build/obj/device/io/port-io.o
8 匹配到二进制文件 ./build/obj/device/keyboard.o
9 匹配到二进制文件 ./build/obj/monitor/monitor.o
10 匹配到二进制文件 ./build/obj/monitor/debug/watchpoint.o
11 匹配到二进制文件 ./build/obj/monitor/debug/expr.o
12 匹配到二进制文件 ./build/obj/monitor/debug/ui.o
13 匹配到二进制文件 ./build/obj/monitor/diff-test/diff-test.o
14 匹配到二进制文件 ./build/obj/monitor/diff-test/protocol.o
15 匹配到二进制文件 ./build/obj/monitor/diff-test/gdb-host.o
16 匹配到二进制文件 ./build/obj/monitor/cpu-exec.o
17 匹配到二进制文件 ./build/obj/memory/memory.o
18 匹配到二进制文件 ./build/obj/cpu/exec/data-mov.o
19 匹配到二进制文件 ./build/obj/cpu/exec/arith.o
20 匹配到二进制文件 ./build/obj/cpu/exec/system.o
21 匹配到二进制文件 ./build/obj/cpu/exec/special.o
22 匹配到二进制文件 ./build/obj/cpu/exec/cc.o
23 匹配到二进制文件 ./build/obj/cpu/exec/prefix.o
24 匹配到二进制文件 ./build/obj/cpu/exec/logic.o
25 匹配到二进制文件 ./build/obj/cpu/exec/exec.o
26 匹配到二进制文件 ./build/obj/cpu/exec/control.o
27 匹配到二进制文件 ./build/obj/cpu/intr.o
28 匹配到二进制文件 ./build/obj/cpu/reg.o
29 匹配到二进制文件 ./build/obj/cpu/decode/modrm.o
30 匹配到二进制文件 ./build/obj/cpu/decode/decode.o
result
```

图 6: 查找编译结果中的dummy



```
vim result
1 ./include/common.h:static volatile int dummy;
2 ./include/debug.h:static volatile int dummy;
1 匹配到二进制文件 ./build/nemu
2 匹配到二进制文件 ./build/obj/device/timer.o
3 匹配到二进制文件 ./build/obj/device/serial.o
4 匹配到二进制文件 ./build/obj/device/vga.o
5 匹配到二进制文件 ./build/obj/device/device.o
6 匹配到二进制文件 ./build/obj/device/io/mmio.o
7 匹配到二进制文件 ./build/obj/device/io/port-io.o
8 匹配到二进制文件 ./build/obj/device/keyboard.o
9 匹配到二进制文件 ./build/obj/monitor/monitor.o
10 匹配到二进制文件 ./build/obj/monitor/debug/watchpoint.o
11 匹配到二进制文件 ./build/obj/monitor/debug/expr.o
12 匹配到二进制文件 ./build/obj/monitor/debug/ui.o
13 匹配到二进制文件 ./build/obj/monitor/diff-test/diff-test.o
14 匹配到二进制文件 ./build/obj/monitor/diff-test/protocol.o
15 匹配到二进制文件 ./build/obj/monitor/diff-test/gdb-host.o
16 匹配到二进制文件 ./build/obj/monitor/cpu-exec.o
17 匹配到二进制文件 ./build/obj/memory/memory.o
18 匹配到二进制文件 ./build/obj/cpu/exec/data-mov.o
19 匹配到二进制文件 ./build/obj/cpu/exec/arith.o
20 匹配到二进制文件 ./build/obj/cpu/exec/system.o
21 匹配到二进制文件 ./build/obj/cpu/exec/special.o
22 匹配到二进制文件 ./build/obj/cpu/exec/cc.o
23 匹配到二进制文件 ./build/obj/cpu/exec/prefix.o
24 匹配到二进制文件 ./build/obj/cpu/exec/logic.o
25 匹配到二进制文件 ./build/obj/cpu/exec/exec.o
26 匹配到二进制文件 ./build/obj/cpu/exec/control.o
27 匹配到二进制文件 ./build/obj/cpu/intr.o
28 匹配到二进制文件 ./build/obj/cpu/reg.o
29 匹配到二进制文件 ./build/obj/cpu/decode/modrm.o
30 匹配到二进制文件 ./build/obj/cpu/decode/decode.o
~
result
```

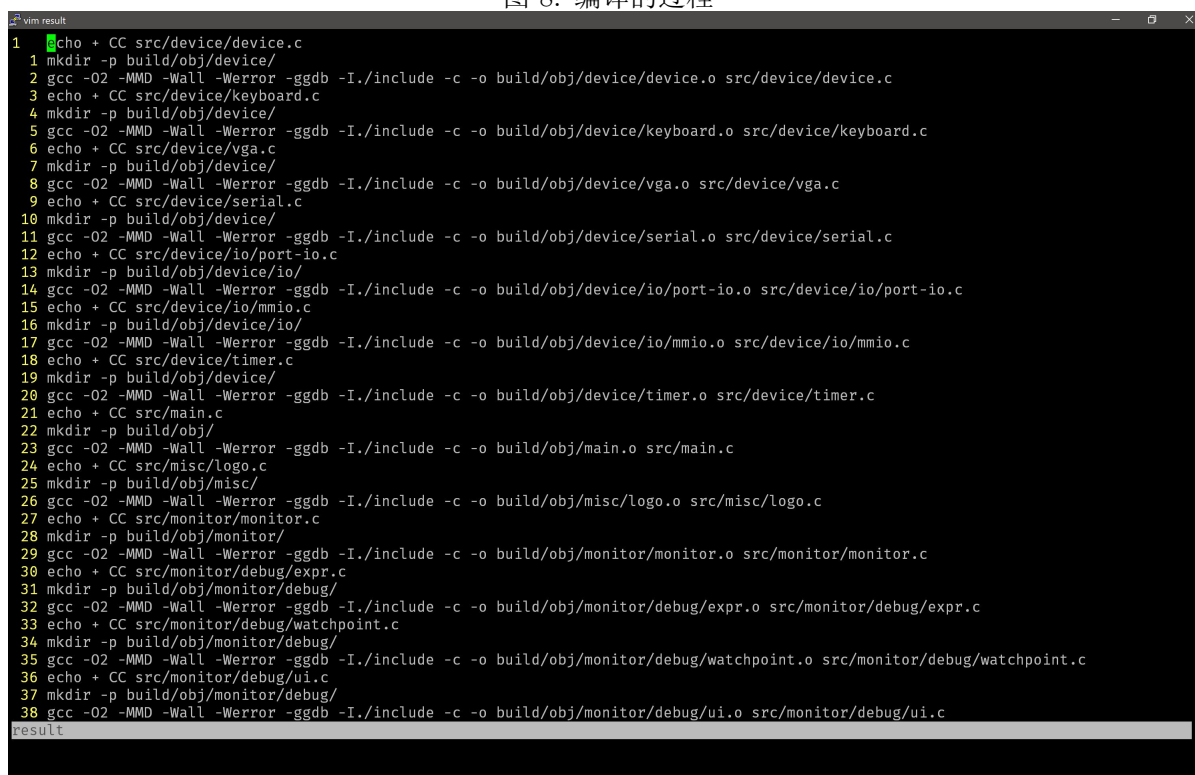
图 7: 对dummy进行初始化后的编译结果



```
In file included from ./include/common.h:9:0,
                 from src/device/device.c:1:
./include/debug.h:6:21: error: redefinition of 'dummy'
static volatile int dummy=0;
                   ^~~~~
In file included from src/device/device.c:1:0:
./include/common.h:5:21: note: previous definition of 'dummy' was here
static volatile int dummy=0;
                   ^~~~~
```



图 8: 编译的过程



```
1 echo + CC src/device/device.c
2 mkdir -p build/obj/device/
3 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/device/device.o src/device/device.c
4 echo + CC src/device/keyboard.c
5 mkdir -p build/obj/device/
6 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/device/keyboard.o src/device/keyboard.c
7 echo + CC src/device/vga.c
8 mkdir -p build/obj/device/
9 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/device/vga.o src/device/vga.c
10 echo + CC src/device/serial.c
11 mkdir -p build/obj/device/
12 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/device/serial.o src/device/serial.c
13 echo + CC src/device/io/port-io.c
14 mkdir -p build/obj/device/io/
15 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/device/io/port-io.o src/device/io/port-io.c
16 echo + CC src/device/io/mmio.c
17 mkdir -p build/obj/device/io/
18 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/device/io/mmio.o src/device/io/mmio.c
19 echo + CC src/device/timer.c
20 mkdir -p build/obj/device/
21 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/device/timer.o src/device/timer.c
22 echo + CC src/main.c
23 mkdir -p build/obj/
24 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/main.o src/main.c
25 echo + CC src/misc/logo.c
26 mkdir -p build/obj/misc/
27 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/misc/logo.o src/misc/logo.c
28 echo + CC src/monitor/monitor.c
29 mkdir -p build/obj/monitor/
30 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/monitor/monitor.o src/monitor/monitor.c
31 echo + CC src/monitor/debug/expr.c
32 mkdir -p build/obj/monitor/debug/
33 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/monitor/debug/expr.o src/monitor/debug/expr.c
34 echo + CC src/monitor/debug/watchpoint.c
35 mkdir -p build/obj/monitor/debug/
36 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/monitor/debug/watchpoint.o src/monitor/debug/watchpoint.c
37 echo + CC src/monitor/debug/ui.c
38 mkdir -p build/obj/monitor/debug/
39 gcc -O2 -MMD -Wall -Werror -ggdb -I./include -c -o build/obj/monitor/debug/ui.o src/monitor/debug/ui.c
result
```

5.这些文件都能编译的话，就会链接起来，生成nemu。

对于nemu的Makefile，它会寻找nemu/src目录下的所有.c文件和uild/obj目录下名称相对应的.o文件。然后将.c文件使用gcc（CC）的-O2 -MMD -Wall -Werror -ggdb（CFLAGS）编译成同名.o文件。最后使用gcc（LD）-O2 -lSDL2 -lreadline链接成可执行目标文件。

### 1.3.2 编译链接的过程

make的-n选项可以将打印出将要执行的命令，但不执行这些命令。由“make -n > result”，可以看到make应当执行以下命令（局部）：看到make在build/obj目录下创建了一系列目录，存放相关源文件的编译出的同名可重定位目标文件（\*.o）。编译完成这些模块文件后，进行git add & git commit。最后将这些.o文件链接起来生成build目录下的nemu文件。

## 2 Answers to some additional questions

### 2.1 捕捉死循环

根据我在网上查找的资料和问题求解课程的相关内容，我认为有以下两种方法，对应于两种不同的死循环形式。对于任意的死循环，我认为这是一种“停机问题”，不存在检测的可能性。

#### 2.1.1 每个循环的状态一致

如果这些循环可以被归为一个状态，例如，所有变量取值一样，或者一组状态的固定循环，例如，所有

变量的值按同一周期形成周期数列。那么，可以通过保存每个状态，并对这些状态进行对比来判断是否进入了死循环。若存储的较多状态形成了周期序列，则认为这是infinite loop.

### 2.1.2 状态不一致，但整体状态数有限

这种情况可以看作上述情况的一种特例。思路如下：若已知这个程序中最多出现 $n$ 个状态，则若这些状态全部出现过之后，程序仍然没有结束，就认为程序进入了死循环。这种判断方法思路来源于线性规划的SIMPLEX算法判断PIVOT是否已经做完。在这个算法中，最多有 $\binom{n+m}{2}$ 种PIVOT方式，若进行了这么多次循环后，程序依然没有停止，就可以认为参与entering和leaving的元素只有固定的两个，只是在不断互换位置，即进入了死循环。此时，SIMPLEX算法已经做完了所有PIVOT，只需退出循环，即可得到线性规划的正确解。

## 2.2 三条指令的整数加法

```
1  ADDITION(a, b)// Addition with only 3 instructions:  $a \leftarrow a + b$ 
2  label:
3  INC(a);
4  DEC(b);
5  JNE(b, 0);// Jump to label if not equal
6  return a;
```

## 2.3 大端机和小端机的立即数转换

我认为可以加入一条伪指令：RTL\_REVERSE(*dest*,*src*).此外，还可以在decoding中加入一个指示big/little endian的标志位，当客户程序和NEMU存储方式不一致，且操作数是立即数时，就执行这条伪指令。

// 这两个过程的参数都是指针

```
1  IMMEDIATE-COMPATIBILITY(decoding, os)
2  if decoding.endian != os.endian && width != 1
3      RTL_REVERSE(t1, decoding -> src, decoding -> width)
4  return
```

```
1  void RTL_REVERSE(int* dest, int* src, int* width)
2  {
3      switch(width)
4      {
5          case 2:
6              short tmp;
7              int i;
8              for(int i=0; i<2; ++i)
9                  {
10                     tmp=(src>>( i*8))&0xff;
11                     dest+=tmp<<((1-i)*8);
```



```

12         }
13         break;
14     case 4:
15         int tmp;
16         dest=0;
17         for (int i=0;i<4;++i)
18         {
19             tmp=(src>>(i*8))&0xff;
20             dest+=tmp<<((3-i)*8);
21         }
22         break;
23     default:
24         printf("Invalid _width!\n");
25         assert(0);
26         break;
27     }
28 }

```

## 2.4 volatile关键字

加上这个关键字是因为：CPU传入到串口的数据，可能会被串口设备修改。而这种修改过程对CPU是不可见的，CPU只能看到修改的结果。如果不加这个关键字，而且编译器进行了优化，会使CPU认为输出的值没有变化。在串口会更改变量的情况下，就会产生错误。

## 3 Reviews on this stage

### 3.1 自己的一些不足

通过这个阶段的实验，我发现了自己在阅读代码、实验手册和动手写代码方面存在的一些偏差，误解和错误。个人认为，PA2的难度相较于PA1还是有一定提升，在这阶段遇到的问题，给了我不少做实验以及在以后写更好代码的经验。总结下来，大致有如下几条：

#### 3.1.1 阅读代码

没有紧跟执行顺序理解代码，经常忽视需要修改文件之外的文件。对于有些函数迟迟没有搞懂，以至于在初期看似正确的结果，随着进度进行，不断暴露出错误。这方面最典型的就是optable数组中的解码函数误用。我最初误解了指定操作数长度的意义，认为有在手册上标注为/r的就应该指定为4位，其他的不需要指定。这种做法初期没有产生问题，直到我开始实现movsx指令时，对于宽度的要求才引起了我的重视，同时我也发现前面的不少实现可能存在错误。只是因为测试样例在接受16/32位宽度的操作数时，通常都给出32位，才勉强通过较多测试。所以在实现指令阶段后期，我浪费了大量精力检查optable的正确性。这一点在以后的实验中要引以为鉴。

### 3.1.2 看上去没有什么用的文件

通常一个目录下最有用的就是.c/.h文件，但是这不代表其他文件就不需要看了。在做I/O画图时，我不是非常明白如何实现\_draw\_rect函数，在工程目录下多次翻找源文件也没有得到想要的结果，直到使用grep搜索函数名，才发现nexus-am下的SPEC.md文件中有较为详细的解释。根据文件中的描述，我很快就做出了正确的实现。受此启发，我又看了一些目录下的README.md文件，发现这些文件都比较完整的概括了这一部分工程的主要内容。这些概括让我对整体框架的印象又深入了一些。在以后阶段，我会仔细阅读这些引导文件。

### 3.1.3 位运算

这一点是我之前一直没有认真对待的。在做标志寄存器赋值判断时，造成了不小的困扰。其实并不是因为我不熟悉位运算的方法，而是我只注意到了我需要的那一位，而忽视了这一位在这个变量中的位置。例如，在取一个32位整数a的符号位时，`sign=a&0x80000000`，确实取到了这个符号。但是我认为此时`sign=1`，这显然是没有考虑到这一位在整个变量中的位置造成的。然而我却迟迟没有发现这个问题，程序运行错误后花了不短的时间才想到。从这里我体会到考虑需要面面俱到，不能因为只关心位运算的结果，就忽视了其他位。

## 3.2 对实验讲义的一些建议

### 3.2.1 内容方面

我认为，实验讲义的内容可以更加简洁、明确。简洁方面，以实现I/O为例，这里使用了大量文字来进行端口介绍，固然非常详细全面，但是缺乏中心，看上去重点不突出。特别是，做出的介绍看似高屋建瓴，实则南辕北辙。以串口为例，我认为做出简单清晰的介绍只需要几行字和/或内存、设备、CPU交互示意图。至于使用的规约、端口使用的发展变化等，可以给出扩展链接作为补充，没有必要长篇大论。明确方面，我认为可以像代码中一样，进行TODO的标注，更加清晰的注明需完成的任务。

### 3.2.2 组织方面

介绍详细固然有合理之处，但是长篇大论确实不合理。我认为，应该在需要大量描述的地方，合理分配小标题，这对于讲义这样以说明为主要目的的文档，是应该而且必要的。不进行合理分段，会使得阅读后完全得不到整体上的印象。这一点，同为说明文档的i386手册可以说是很好的模板，对所有可以单独介绍的内容，都分了小节，并拟定标题。

## References

Intel 80386 Programmer's Reference Manual

ICS2017 Programming Assignment

Can a runtime environment detect an infinite loop?—stackoverflow

Can an operating system restart a process that is stuck in infinite loop?—stackoverflow

## Acknowledgments

在这一阶段的实验中，我要特别感谢凌浩同学。在他的push之下，我加快了因为国庆假期落下的进度。他还帮助我锁定了因不恰当的指令实现导致的nemu框架bug，通过和他的实现效果对比，指出我的代码有较大的优化空间。还要感谢张铭方同学，他帮助我读懂了讲义内容，使我克服了初读代码的恐惧。同时，在和他的讨论中，我也更正了对译码函数的错误认识。此外，还要感谢兰方舟同学，他向我展示了I/O的正确实现，使我搞懂了Putty和Xming接收的键盘输入分别导向什么地方。