PA2实验报告

进度

已完成PA2.3阶段及之前所有内容

吐槽

```
强大的宏
如果你知道C++的"模板"功能，你可能会建议使用它，但事实上在这里做不到．我们知道宏是在编译预处理阶段进行处理的，这意味着宏的功能不受编译阶段的约束(
```

这段话是什么鬼？"靠宏生成简洁的代码"真是呵呵了。。。
为了实现函数重载，靠一堆莫名其妙的后缀拼接，每次实现一条新指令就要大量复制一堆莫名其妙的宏（什么make_instr_helper，make_helper_v），这些宏又依赖不知哪里定义的宏和变量，真是c语言滥用宏的典范，这可读性简直....I'm angry. 靠这堆乱七八糟的宏实现，想没bug都难，连作者自己在框架代码里面的实现的指令都有BUG的...
作者还是要学习一个C++模板元编程和函数式编程，提高自己的姿势水平，这些比宏高到不知哪里去。不能生成新的token不等于不能做所有宏能做的事，是因为根本不需要拼接token就能实现，图灵完备的lambda calculus就摆在这，说比宏弱这让Church情何以堪...模板参数化就能做的事情用拼接token的方法做就是蛋疼...

自动测试脚本

为了随时测试testcase/src/*.c和检查剩余未实现opcode，写了个脚本run_all_test.sh，用pcregrep,objdump等把未实现opcode对应的反汇编、地址，以及测试结果整理汇总，脚本如下：

```
#!/bin/bash

for i in $(ls testcase/src/*.c | sed -- 's/.*\///g' | sed -- 's/\.c//g'); do
  export BUILD_OBJ=./obj/testcase/$i;
  (echo -ne 'c\nq\n' | make run > $i-log.txt) || (echo "Aborted" >> $i-log.txt);
  echo "$i done.";
done;

echo "TODO: ";
IFS='
';
set -f;
res=$(find ./ -type f -name *-log.txt -exec grep invalid {} +);
for i in $res; do
  addr=$(echo "$i" | sed 's/^.*eip\s=\s0x00\([0-9a-z]*\).*$/\1/g');
  filepath=$(echo "$i" | sed 's|^\./\(.*\)-log.txt:.*$|\1|g');
  echo "$addr $filepath";
  objdump -d "./obj/testcase/$filepath" | grep "$addr" --color;
done;
set +f;
unset IFS;

#grep invalid ./*-log.txt --color;
echo "Result: ";
grep GOOD ./*-log.txt --color;
grep BAD ./*-log.txt --color;
pcregrep -M 'c\nAborted' ./*-log.txt;
```

使用前需修改Makefile如下:

```
#USERPROG := obj/testcase/add
USERPROG := $(BUILD_OBJ)
```

最终输出：

```
garzon@sixstars-XPS-8300:~/pa$ ./run_all_test.sh
add done.
```

```
add-longlong done.
bit done.
..bubble-sort done.
fact done.
fib done.
gotbaha done.
nemu: nemu/src/cpu/exec/special/special.c:48: inv2: Assertion `0' failed.
make: *** [run] Aborted (core dumped)
hello done.
nemu: nemu/src/cpu/exec/special/special.c:24: inv: Assertion `0' failed.
make: *** [run] Aborted (core dumped)
hello-inline-asm done.
nemu: nemu/src/cpu/exec/special/special.c:24: inv: Assertion `0' failed.
make: *** [run] Aborted (core dumped)
hello-str done.
if-else done.
integral done.
leap-year done.
......................................................................................................................................
matrix-mul-small done.
max done.
min3 done.
mov-c done.
movsx done.
mul-longlong done.
pascal done.
prime done.
quadratic-eq done.
.quick-sort done.
.select-sort done.
.shuixianhua done.
string done.
struct done.
sub-longlong done.
sum done.
switch done.
to-lower-case done.
...................wanshu done.
TODO:
100029 hello-inline-asm
  100029:   cd 80                   int    $0x80
10029f hello-str
  10029f:   d9 ee                   fldz
104778 hello
  104778:   0f 44 c1                cmove  %ecx,%eax
Result:
./add-longlong-log.txt:nemu: HIT GOOD TRAP at eip = 0x00100105
./bit-log.txt:nemu: HIT GOOD TRAP at eip = 0x00100283
./bubble-sort-log.txt:nemu: HIT GOOD TRAP at eip = 0x001000fb
./fact-log.txt:nemu: HIT GOOD TRAP at eip = 0x0010009a
./fib-log.txt:nemu: HIT GOOD TRAP at eip = 0x00100074
./gotbaha-log.txt:nemu: HIT GOOD TRAP at eip = 0x001000d7
./if-else-log.txt:nemu: HIT GOOD TRAP at eip = 0x001000bb
./integral-log.txt:nemu: HIT GOOD TRAP at eip = 0x001006d5
./leap-year-log.txt:nemu: HIT GOOD TRAP at eip = 0x001000b8
./matrix-mul-log.txt:nemu: HIT GOOD TRAP at eip = 0x001000f6
./matrix-mul-small-log.txt:nemu: HIT GOOD TRAP at eip = 0x00100120
./max-log.txt:nemu: HIT GOOD TRAP at eip = 0x001000ab
./min3-log.txt:nemu: HIT GOOD TRAP at eip = 0x001000dc
./mov-c-log.txt:nemu: HIT GOOD TRAP at eip = 0x001000cf
./movsx-log.txt:nemu: HIT GOOD TRAP at eip = 0x0010016e
./mul-longlong-log.txt:nemu: HIT GOOD TRAP at eip = 0x0010010b
./pascal-log.txt:nemu: HIT GOOD TRAP at eip = 0x0010010e
./prime-log.txt:nemu: HIT GOOD TRAP at eip = 0x00100097
./quadratic-eq-log.txt:nemu: HIT GOOD TRAP at eip = 0x0010018c
./quick-sort-log.txt:nemu: HIT GOOD TRAP at eip = 0x001001f4
./select-sort-log.txt:nemu: HIT GOOD TRAP at eip = 0x001000f9
./shuixianhua-log.txt:nemu: HIT GOOD TRAP at eip = 0x00100127
./string-log.txt:nemu: HIT GOOD TRAP at eip = 0x0010015c
./struct-log.txt:nemu: HIT GOOD TRAP at eip = 0x001000f5
./sub-longlong-log.txt:nemu: HIT GOOD TRAP at eip = 0x00100105
./sum-log.txt:nemu: HIT GOOD TRAP at eip = 0x0010004c
./switch-log.txt:nemu: HIT GOOD TRAP at eip = 0x001000b0
./to-lower-case-log.txt:nemu: HIT GOOD TRAP at eip = 0x0010007b
./wanshu-log.txt:nemu: HIT GOOD TRAP at eip = 0x00100096
```

必答题

编译与链接

- 在 nemu/include/cpu/helper.h 中, 你会看到由 static inline 开头定义的 instr_fetch() 函数和 idex() 函数. 选择其中一个函数, 分别尝试去掉 static , 去掉 inline 或去掉两者, 然后重新进行编译, 你会看到发生错误. 请分别解释为什么会发生这些错误? 你有办法证明你的想法吗?

去掉static后，每个.o文件中都有一个instr_fetch所以

```
/home/garzon/pa/nemu/include/cpu/helper.h:11: multiple definition of `instr_fetch'
obj/nemu/cpu/exec/exec.o:/home/garzon/pa/nemu/include/cpu/helper.h:11: first defined here
obj/nemu/cpu/decode/decode.o: In function `instr_fetch':
/home/garzon/pa/nemu/include/cpu/helper.h:11: multiple definition of `instr_fetch'
obj/nemu/cpu/exec/exec.o:/home/garzon/pa/nemu/include/cpu/helper.h:11: first defined here
obj/nemu/monitor/cpu-exec.o: In function `instr_fetch':
/home/garzon/pa/nemu/include/cpu/helper.h:11: multiple definition of `instr_fetch'
obj/nemu/cpu/exec/exec.o:/home/garzon/pa/nemu/include/cpu/helper.h:11: first defined here
collect2: error: ld returned 1 exit status
```

如此可证：

```
garzon@sixstars-XPS-8300:~/pa$ objdump -d obj/nemu/monitor/cpu-exec.o

obj/nemu/monitor/cpu-exec.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <instr_fetch>:
   0:   e9 00 00 00 00          jmpq   5 <instr_fetch+0x5>
   5:   66 66 2e 0f 1f 84 00    data32 nopw %cs:0x0(%rax,%rax,1)
   c:   00 00 00 00
```

没有inline的话，编译器检测到了instr_fetch在本文件没被使用，于是编译时就出错：

```
garzon@sixstars-XPS-8300:~/pa$ make run
+ cc nemu/src/cpu/exec/exec.c
+ cc nemu/src/cpu/exec/misc/misc.c
+ cc nemu/src/cpu/exec/arith/neg.c
In file included from nemu/include/cpu/exec/helper.h:4:0,
                 from nemu/src/cpu/exec/arith/neg.c:1:
nemu/include/cpu/helper.h:10:17: error: ‘instr_fetch’ defined but not used [-Werror=unused-function]
 static uint32_t instr_fetch(swaddr_t addr, size_t len) {
                 ^
cc1: all warnings being treated as errors
make: *** [obj/nemu/cpu/exec/arith/neg.o] Error 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?

如下所示：

```
garzon@sixstars-XPS-8300:~/pa$ readelf -a obj/nemu/nemu  | grep dummy
    40: 000000000061e474     4 OBJECT  LOCAL  DEFAULT   25 dummy
    46: 000000000061e4a0     4 OBJECT  LOCAL  DEFAULT   25 dummy
    59: 000000000061e4a4     4 OBJECT  LOCAL  DEFAULT   25 dummy
    68: 0000000000401350     0 FUNC    LOCAL  DEFAULT   13 frame_dummy
    69: 000000000061ce00     0 OBJECT  LOCAL  DEFAULT   18 __frame_dummy_init_array_
    73: 000000000061e3f4     4 OBJECT  LOCAL  DEFAULT   25 dummy
   106: 000000000061e3f8     4 OBJECT  LOCAL  DEFAULT   25 dummy
   109: 000000000061e3fc     4 OBJECT  LOCAL  DEFAULT   25 dummy
   114: 000000000061e400     4 OBJECT  LOCAL  DEFAULT   25 dummy
   122: 000000000061e404     4 OBJECT  LOCAL  DEFAULT   25 dummy
   129: 000000000061e408     4 OBJECT  LOCAL  DEFAULT   25 dummy
   137: 000000000061e40c     4 OBJECT  LOCAL  DEFAULT   25 dummy
   145: 000000000061e410     4 OBJECT  LOCAL  DEFAULT   25 dummy
   153: 000000000061e414     4 OBJECT  LOCAL  DEFAULT   25 dummy
   160: 000000000061e418     4 OBJECT  LOCAL  DEFAULT   25 dummy
   165: 000000000061e41c     4 OBJECT  LOCAL  DEFAULT   25 dummy
   174: 000000000061e420     4 OBJECT  LOCAL  DEFAULT   25 dummy
   179: 000000000061e424     4 OBJECT  LOCAL  DEFAULT   25 dummy
   184: 000000000061e428     4 OBJECT  LOCAL  DEFAULT   25 dummy
   186: 000000000061e42c     4 OBJECT  LOCAL  DEFAULT   25 dummy
   189: 000000000061e430     4 OBJECT  LOCAL  DEFAULT   25 dummy
   191: 000000000061e434     4 OBJECT  LOCAL  DEFAULT   25 dummy
   193: 000000000061e438     4 OBJECT  LOCAL  DEFAULT   25 dummy
   198: 000000000061e43c     4 OBJECT  LOCAL  DEFAULT   25 dummy
   200: 000000000061e440     4 OBJECT  LOCAL  DEFAULT   25 dummy
   205: 000000000061e444     4 OBJECT  LOCAL  DEFAULT   25 dummy
   207: 000000000061e448     4 OBJECT  LOCAL  DEFAULT   25 dummy
   214: 000000000061e44c     4 OBJECT  LOCAL  DEFAULT   25 dummy
   220: 000000000061e450     4 OBJECT  LOCAL  DEFAULT   25 dummy
   227: 000000000061e454     4 OBJECT  LOCAL  DEFAULT   25 dummy
   232: 000000000061e458     4 OBJECT  LOCAL  DEFAULT   25 dummy
   246: 000000000061e45c     4 OBJECT  LOCAL  DEFAULT   25 dummy
   248: 000000000061e460     4 OBJECT  LOCAL  DEFAULT   25 dummy
   256: 000000000061e464     4 OBJECT  LOCAL  DEFAULT   25 dummy
   264: 000000000061e468     4 OBJECT  LOCAL  DEFAULT   25 dummy
   272: 000000000061e46c     4 OBJECT  LOCAL  DEFAULT   25 dummy
   275: 000000000061e470     4 OBJECT  LOCAL  DEFAULT   25 dummy
   283: 000000000061e478     4 OBJECT  LOCAL  DEFAULT   25 dummy
   288: 000000000061e47c     4 OBJECT  LOCAL  DEFAULT   25 dummy
   296: 000000000061e480     4 OBJECT  LOCAL  DEFAULT   25 dummy
   304: 000000000061e484     4 OBJECT  LOCAL  DEFAULT   25 dummy
   310: 000000000061e488     4 OBJECT  LOCAL  DEFAULT   25 dummy
   315: 000000000061e48c     4 OBJECT  LOCAL  DEFAULT   25 dummy
   323: 000000000061e490     4 OBJECT  LOCAL  DEFAULT   25 dummy
   331: 000000000061e494     4 OBJECT  LOCAL  DEFAULT   25 dummy
   339: 000000000061e498     4 OBJECT  LOCAL  DEFAULT   25 dummy
   342: 000000000061e49c     4 OBJECT  LOCAL  DEFAULT   25 dummy
   349: 000000000061e578     4 OBJECT  LOCAL  DEFAULT   25 dummy
   355: 000000000061e9a0     4 OBJECT  LOCAL  DEFAULT   25 dummy
   371: 000000000061e9d8     4 OBJECT  LOCAL  DEFAULT   25 dummy
   377: 000000000061f900     4 OBJECT  LOCAL  DEFAULT   25 dummy
   382: 000000000061f910     4 OBJECT  LOCAL  DEFAULT   25 dummy
   384: 000000000061f918     4 OBJECT  LOCAL  DEFAULT   25 dummy
   393: 000000000065fa20     4 OBJECT  LOCAL  DEFAULT   25 dummy
   402: 000000000066fae4     4 OBJECT  LOCAL  DEFAULT   25 dummy
   404: 000000000066fae8     4 OBJECT  LOCAL  DEFAULT   25 dummy
   406: 000000000066faec     4 OBJECT  LOCAL  DEFAULT   25 dummy
   410: 000000000066fb00     4 OBJECT  LOCAL  DEFAULT   25 dummy
   412: 000000000066fb04     4 OBJECT  LOCAL  DEFAULT   25 dummy
   414: 000000000066fb08     4 OBJECT  LOCAL  DEFAULT   25 dummy
   426: 000000000066fb38     4 OBJECT  LOCAL  DEFAULT   25 dummy
   435: 000000000066fb48     4 OBJECT  LOCAL  DEFAULT   25 dummy
   439: 000000000066fb58     4 OBJECT  LOCAL  DEFAULT   25 dummy
   441: 000000000066fb5c     4 OBJECT  LOCAL  DEFAULT   25 dummy
   445: 000000000066fb60     4 OBJECT  LOCAL  DEFAULT   25 dummy
   451: 000000000066fb64     4 OBJECT  LOCAL  DEFAULT   25 dummy
garzon@sixstars-XPS-8300:~/pa$ readelf -a obj/nemu/nemu  | grep dummy | awk "{print NR}" | tail -n1
66
```

减去两个多出来的，有64个dummy

- 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.

```
garzon@sixstars-XPS-8300:~/pa$ readelf -a obj/nemu/nemu  | grep dummy | awk "{print NR}" | tail -n1
66
```

同样为64个，因为声明了static会自动用.o里面的那个同一个实体。

- 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.)

```
garzon@sixstars-XPS-8300:~/pa$ make run
+ cc nemu/src/cpu/reg.c
In file included from nemu/include/nemu.h:4:0,
                 from nemu/src/cpu/reg.c:1:
nemu/include/common.h:26:21: error: redefinition of ‘dummy’
 volatile static int dummy = 0;
```

```
                      ^
In file included from nemu/include/common.h:12:0,
                 from nemu/include/nemu.h:4,
                 from nemu/src/cpu/reg.c:1:
nemu/include/debug.h:9:21: note: previous definition of ‘dummy’ was here
 volatile static int dummy = 0;
                      ^
make: *** [obj/nemu/cpu/reg.o] Error 1
```

因为这两个dummy是同一个，所以不能有两个初始值，就变成了redefinition

了解Makefile

请描述你在工程目录下敲入 make 后, make 程序如何组织.c和.h文件, 最终生成可执行文件 obj/nemu/nemu . (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, 函数, 包含文件等特性
- Makefile 运用并重写了一些implicit rules

Makefile语法大致如下：

```
(要生成的文件/make命令的参数): 该项依赖的文件
    命令
    命令
    ...

临时变量名 := 值

include (Makefile_path)
```

链接大概就是把*.o中的变量连接起来，把代码块拼起来，修改，计算相对地址及符号变量地址构成新的全局符号表，填入原来.o中代码的符号变量的占位符（静态链接），生成可执行文件。

运行结果

bt(add.c)

```
(nemu) c

Hit breakpoint at eip = 0x00100018
(nemu) bt
Now 0x100019 <add+0x7>(0x0, 0x0, 0x0, 0x0)
#1 0x100068 <main+0x3f>(0x0, 0x0, 0x0, 0x0)
(nemu) c

Hit breakpoint at eip = 0x00100018
(nemu) bt
Now 0x100019 <add+0x7>(0x0, 0x1, 0x0, 0x1)
#1 0x100068 <main+0x3f>(0x0, 0x1, 0x0, 0x1)
(nemu) c

Hit breakpoint at eip = 0x00100018
(nemu) bt
Now 0x100019 <add+0x7>(0x0, 0x2, 0x0, 0x2)
#1 0x100068 <main+0x3f>(0x0, 0x2, 0x0, 0x2)
(nemu)
```

p *(GLOBAL_VARIABLE_NAME)

```
garzon@sixstars-XPS-8300:~/pa$ make run
objcopy -S -O binary obj/testcase/add entry
obj/nemu/nemu obj/testcase/add
Welcome to NEMU!
The executable is obj/testcase/add.
For help, type "help"
(nemu) p *(test_str+1)
```

```
eval result: 0x65646362 1701077858
(nemu) p *(test_str+2)
eval result: 0x66656463 1717920867
(nemu) p *(test_str)
eval result: 0x64636261 1684234849
(nemu) p add
eval result: 0x100012 1048594
```