计算机系统基础
Programming Assignment

PA Makefile、Macro和Debugging

2018年9月30日

内容提要

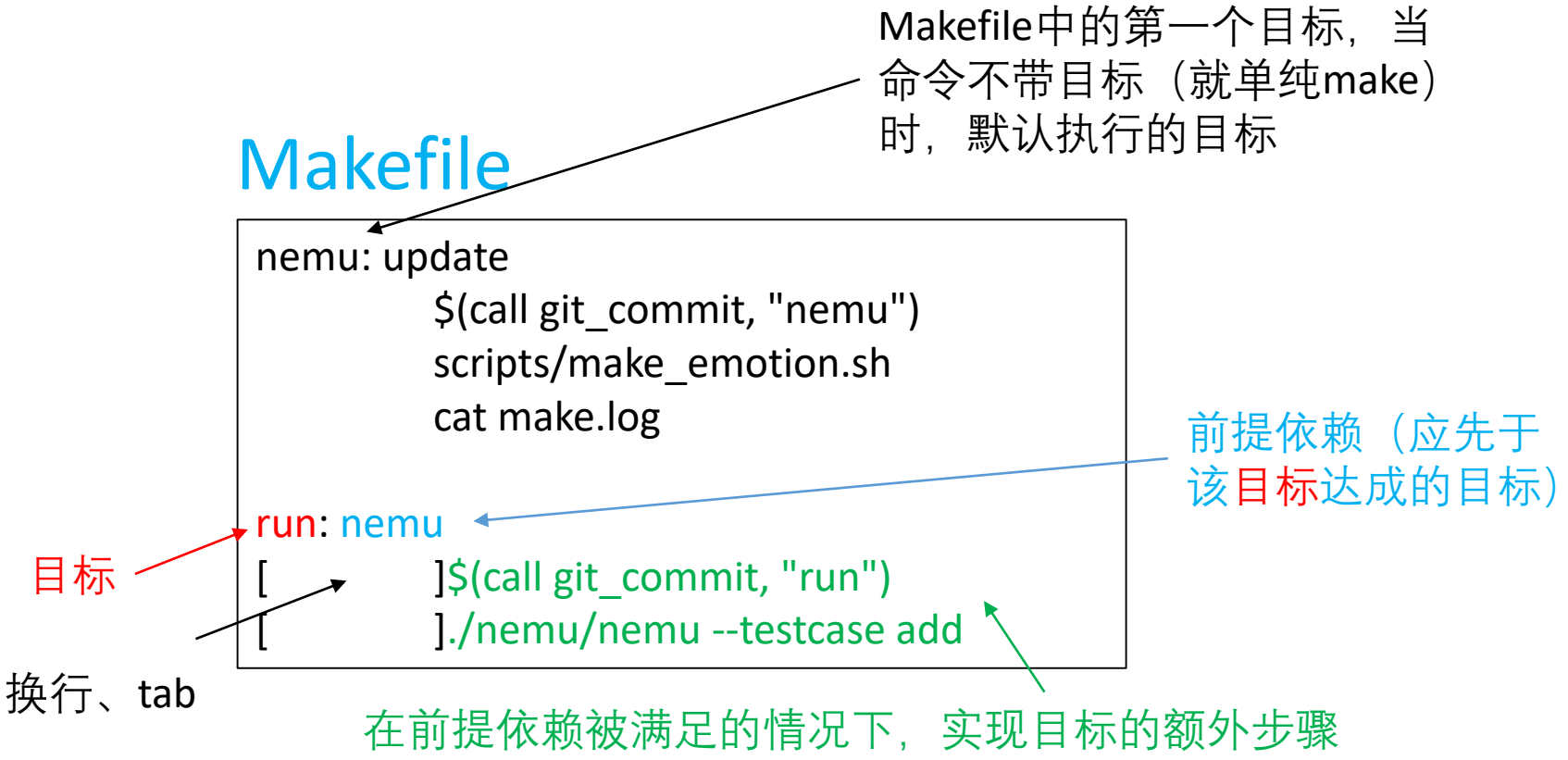
- 从make run讲起
- 框架代码中宏的理解和运用
- 以正确的心态和方法去应对bug

内容提要

- 从make run讲起
- 框架代码中宏的理解和运用
- 以正确的心态和方法去应对bug

从make run讲起

- 当在pa2018_fall/中输入make run的时候，发生了什么?
  1. make程序在当前目录下寻找Makefile
  2. 若找到，执行指定的目标（如run）或第一个目标（命令中没有目标只有make的情形）



从make run讲起

- 当在pa2018_fall/中输入make run的时候，发生了什么?
  1. make程序在当前目录下寻找Makefile
  2. 若找到，执行指定的目标（如run）或第一个目标（命令中没有目标只有make的情形）

Makefile

```
nemu: update
        $(call git_commit, "nemu")
        scripts/make_emotion.sh
        cat make.log

run: nemu
        $(call git_commit, "run")
        ./nemu/nemu --testcase add
```

编译项目，调查问卷

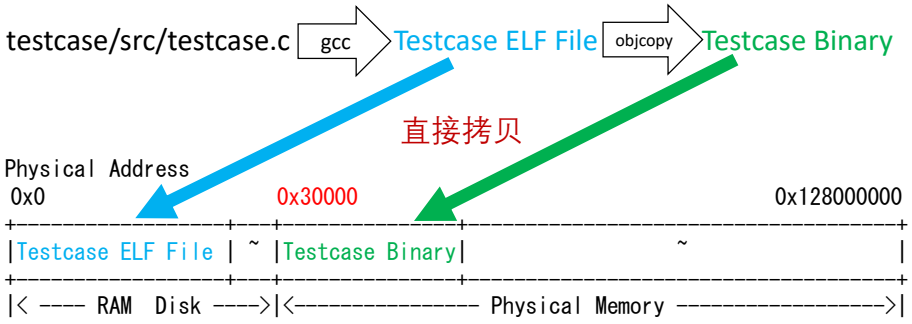
执行测试用例

./nemu/nemu --testcase add

```
int main(int argc, char* argv[]) {

        if(!parse_args(argc, argv)) { // parse the arguments
                printf("NEMU exit with argment parse error\n");
                return -1;
        }
        // …
        if(flag_reg_alu_fpu) {
                // 执行reg，alu，fpu对应的测试
        } else { // 执行测试用例
                // either testcase or game, the img and elf paths should be properly set
                if(flag_score) {
                        single_run(image_path_score, elf_path_score);
                } else {
                        single_run(image_path, elf_path);
                }
                //…
        }
        return 0;
}
```

nemu/src/main.c

single_run(image_path, elf_path)



nemu/src/main.c

```
static void single_run(const char * img_file_path, const char * elf_file_path) {
        restart(INIT_EIP); // restart the machine, do some initializations
        printf(…); // 输出路径
        // Load the memory image of executable
        load_exec(img_file_path, LOAD_OFF);
#ifdef HAS_DEVICE_IDE
        init_ide(elf_file_path); // Initialize hard drive
#else
        load_exec(elf_file_path, 0); // Load ELF file
#endif
        load_elf_tables(elf_file_path);
        nemu_state = NEMU_READY; // Set the state of the machine to NEMU_READY
        ui_mainloop(flag_autorun); // Enter UI mainloop to accept user commands
}
```

ui_mainloop(flag_autorun)

nemu/src/monitor/ui.c

```
// the main loop of accepting user commands
void ui_mainloop(bool autorun) {

        if(autorun) { // 如果执行时有--autorun标志，则直接执行
                cmd_c("");
                if(nemu_state == NEMU_STOP) {
                        return;
                }
        }

        while(true) {
                // 扫描用户命令并执行
        }
}
```

```
cmd_handler(cmd_c) {
        // execute the program
        exec(-1);
        return 0;
}
```

你懂的

```
(nemu) c
```

exec(-1)

nemu/src/cpu/cpu.c

```
void exec(uint32_t n) {
        ...
        while( n > 0 && nemu_state == NEMU_RUN) {
                ...
                instr_len = exec_inst();
                cpu.eip += instr_len;
                n--;
                ...
        }
        ...
}

int exec_inst() {
        uint8_t opcode = 0;
        // get the opcode，  取操作数
        opcode = instr_fetch(cpu.eip, 1);
        // instruction decode and execution，执行这条指令
        int len = opcode_entry[opcode](cpu.eip, opcode);
        return len;  // 返回指令长度
}
```

你懂的

那么，PA 2-1要怎么开始做呢?

上一回的ppt，应该看过了，快速过一遍

NEMU模拟指令译码和执行

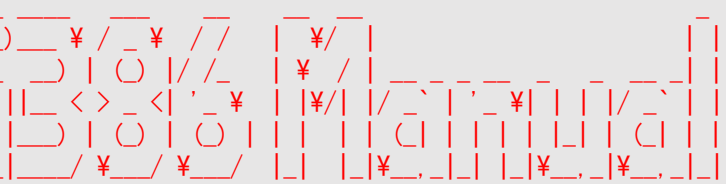
./nemu/nemu --testcase add
也可以 ./nemu/nemu --autorun --testcase add

- 所以PA 2-1要做的任务：执行make run或make test_pa-2-1

invalid opcode(eip = 0x00030033): 83 f8 01 66 c7 05 34 12 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x00030033 is not implemented.
2. Something is implemented incorrectly.
Find this eip value(0x00030033) in the disassembling result to distinguish which case it is.

If it is the first case, see



for more details.

If it is the second case, remember:
* The machine is always right!
* Every line of untested code is always wrong!

不加--autorun的话会进入交互调试界面，相应的命令看guide pg. 28-29的表格

1. 查i386手册得知这是一条什么指令
   a) 先查appendix A得知指令的类型和格式
   b) 必要的话查section 17.2.1译码ModR/M和SIB字节
   c) 必要的话查section 17.2.2.11查看指令的具体含义和细节

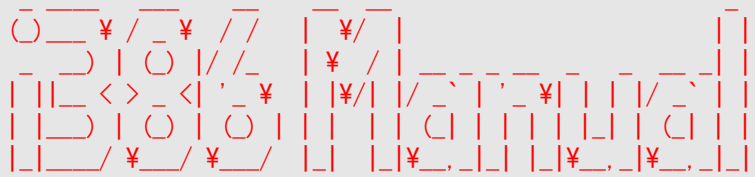
NEMU模拟指令译码和执行

- 所以PA 2-1要做的任务：执行make run或make test_pa-2-1

invalid opcode(eip = 0x00030033): 83 f8 01 66 c7 05 34 12 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x00030033 is not implemented.
2. Something is implemented incorrectly.
Find this eip value(0x00030033) in the disassembling result to distinguish which case it is.

If it is the first case, see



for more details.

If it is the second case, remember:
* The machine is always right!
* Every line of untested code is always wrong!

1. 查i386手册得知这是一条什么指令
2. 写该操作码对应的instr_func
   a) 例如：make_instr_func(mov_i2rm_v)

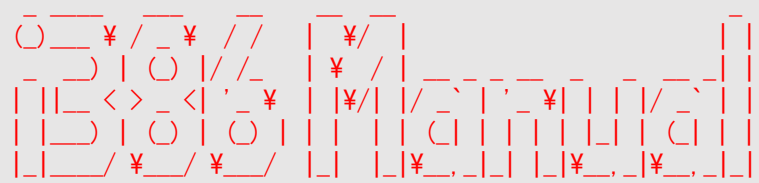
NEMU模拟指令译码和执行

- 所以PA 2-1要做的任务：执行make run或make test_pa-2-1

invalid opcode(eip = 0x00030033): 83 f8 01 66 c7 05 34 12 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x00030033 is not implemented.
2. Something is implemented incorrectly.
Find this eip value(0x00030033) in the disassembling result to distinguish which case it is.

If it is the first case, see



for more details.

If it is the second case, remember:
* The machine is always right!
* Every line of untested code is always wrong!

1. 查i386手册得知这是一条什么指令
2. 写该操作码对应的instr_func
3. 把这个函数在nemu/include/cpu/instr.h中声明一下
4. 在opcode_entry对应该操作码的地方把这个函数的函数名填进去替代原来的inv

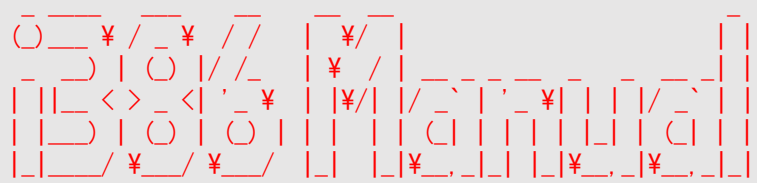
NEMU模拟指令译码和执行

- 所以PA 2-1要做的任务：执行make run或make test_pa-2-1

invalid opcode(eip = 0x00030033): 83 f8 01 66 c7 05 34 12 ...

There are two cases which will trigger this unexpected exception:
1. The instruction at eip = 0x00030033 is not implemented.
2. Something is implemented incorrectly.
Find this eip value(0x00030033) in the disassembling result to distinguish which case it is.

If it is the first case, see



for more details.

If it is the second case, remember:
* The machine is always right!
* Every line of untested code is always wrong!

1. 查i386手册得知这是一条什么指令
2. 写该操作码对应的instr_func
3. 把这个函数在nemu/include/cpu/instr.h中声明一下
4. 在opcode_entry对应该操作码的地方把这个函数的函数名填进去替代原来的inv
5. 重复上述过程直至完成所有需要模拟的指令

内容提要

- 从make run讲起
- 框架代码中宏的理解和运用
- 以正确的心态和方法去应对bug

怎么从左边变到右边的?

```
make_instr_func(mov_r2rm_v) {
    OPERAND r, rm;
    // 指定操作数长度
    rm.data_size = r.data_size = data_size;
    int len = 1;
    // 操作数寻址
    len += modrm_r_rm(eip + 1, &r, &rm);
    // 执行mov操作
    operand_read(&r);
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

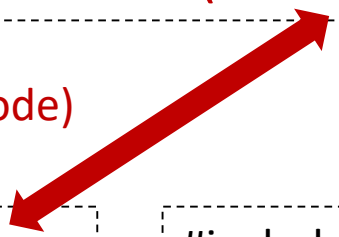
```
#include "cpu/instr.h"

static void instr_execute_2op() {
        operand_read(&opr_src);
        opr_dest.val = opr_src.val;
        operand_write(&opr_dest);
}

make_instr_impl_2op(mov, r, rm, v)
```

nemu/src/cpu/instr/mov.c

nemu/include/cpu/instr_helper.h

```
// macro for making an instruction entry
#define make_instr_func(name) int name(uint32_t eip, uint8_t opcode)
```

int mov_r2rm_v (uint32_t eip, uint8_t opcode)



```
make_instr_func(mov_r2rm_v) {
    OPERAND r, rm;
    // 指定操作数长度
    rm.data_size = r.data_size = data_size;
    int len = 1;
    // 操作数寻址
    len += modrm_r_rm(eip + 1, &r, &rm);
    // 执行mov操作
    operand_read(&r);
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

```
#include "cpu/instr.h"

static void instr_execute_2op() {
        operand_read(&opr_src);
        opr_dest.val = opr_src.val;
        operand_write(&opr_dest);
}

make_instr_impl_2op(mov, r, rm, v)
```

nemu/src/cpu/instr/mov.c

```
// macro for generating the implementation of an instruction with two operands
#define make_instr_impl_2op(inst_name, src_type, dest_type, suffix) \
    // 等于 make_instr_impl_2op(mov, r, rm, v)
    make_instr_func(concat7(inst_name, _, src_type, 2, dest_type, _, suffix)) {\
     // 宏展开等于 make_instr_func(mov_ r2rm_v) {
            int len = 1; \
            concat(decode_data_size_, suffix) \
            concat3(decode_operand, _, concat3(src_type, 2, dest_type)) \
            print_asm_2(...); \
            instr_execute_2op(); \
            return len; \
    }
```

nemu/include/cpu/instr_helper.h

```
    operand_write(&opr_dest);
}
```

make_instr_impl_2op(mov, r, rm, v)

nemu/src/cpu/instr/mov.c

```
    // 操作数寻址
    len += modrm_r_rm(eip + 1, &r, &rm);
    // 执行mov操作
    operand_read(&r);
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

```
// macro for generating the implementation of an instruction with two operands
#define make_instr_impl_2op(inst_name, src_type, dest_type, suffix) \
        // 等于 make_instr_impl_2op(mov, r, rm, v)
        make_instr_func(concat7(inst_name, _, src_type, 2, dest_type, _, suffix)) {\
         // 宏展开等于 make_instr_func(mov_ r2rm_v) {
                int len = 1; \ // 不变
                concat(decode_data_size_, suffix) \
                concat3(decode_operand, _, concat3(src_type, 2, dest_type)) \
                print_asm_2(...); \
                instr_execute_2op(); \
                return len; \
        }
```

nemu/include/cpu/instr_helper.h

```
    // 操作数寻址
    len += modrm_r_rm(eip + 1, &r, &rm);
    // 执行mov操作
    operand_read(&r);
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

```
    operand_write(&opr_dest);
}

make_instr_impl_2op(mov, r, rm, v)
```

nemu/src/cpu/instr/mov.c

```
// macro for generating the implementation of an instruction with two operands
#define make_instr_impl_2op(inst_name, src_type, dest_type, suffix) \
        // 等于 make_instr_impl_2op(mov, r, rm, v)
        make_instr_func(concat7(inst_name, _, src_type, 2, dest_type, _, suffix)) {\
         // 宏展开等于 make_instr_func(mov_ r2rm_v) {
                int len = 1; \ // 不变
                concat(decode_data_size_, suffix) \
// 宏展开等于 decode_data_size_v
// 下方宏定义 #define decode_data_size_v opr_src.data_size = opr_dest.data_size = data_size;
                concat3(decode_operand, _, concat3(src_type, 2, dest_type)) \
                print_asm_2(...); \
                instr_execute_2op(); \
                return len; \
        }
```

nemu/include/cpu/instr_helper.h

```
    // 执行mov操作
    operand_read(&r);
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

```
make_instr_impl_2op(mov, r, rm, v)
```

nemu/src/cpu/instr/mov.c

```
// macro for generating the implementation of an instruction with two operands
#define make_instr_impl_2op(inst_name, src_type, dest_type, suffix) \
        // 等于 make_instr_impl_2op(mov, r, rm, v)
        make_instr_func(concat7(inst_name, _, src_type, 2, dest_type, _, suffix)) {\
         // 宏展开等于 make_instr_func(mov_ r2rm_v) {
                int len = 1; \ // 不变
                concat(decode_data_size_, suffix) \
// 宏展开等于 decode_data_size_v
// 下方宏定义 #define decode_data_size_v opr_src.data_size = opr_dest.data_size = data_size;
                concat3(decode_operand, _, concat3(src_type, 2, dest_type)) \
// 宏展开等于 decode_operand_r2rm
// 下方宏定义 #define decode_operand_r2rm \
//                 len += modrm_r_rm(eip + 1, &opr_src, &opr_dest);
                print_asm_2(...); \
                instr_execute_2op(); \
                return len; \
        }
```

nemu/include/cpu/instr_helper.h

```
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

```
// macro for generating the implementation of an instruction with two operands
#define make_instr_impl_2op(inst_name, src_type, dest_type, suffix) \
        // 等于 make_instr_impl_2op(mov, r, rm, v)
        make_instr_func(concat7(inst_name, _, src_type, 2, dest_type, _, suffix)) {\
         // 宏展开等于 make_instr_func(mov_ r2rm_v) {
                int len = 1; \ // 不变
                concat(decode_data_size_, suffix) \
// 宏展开等于 decode_data_size_v
// 下方宏定义 #define decode_data_size_v opr_src.data_size = opr_dest.data_size = data_size;
                concat3(decode_operand, _, concat3(src_type, 2, dest_type)) \
// 宏展开等于 decode_operand_r2rm
// 下方宏定义 #define decode_operand_r2rm \
//                  len += modrm_r_rm(eip + 1, &opr_src, &opr_dest);
                print_asm_2(...); \ // 单步执行打印调试信息，不变
                instr_execute_2op(); \
                return len; \
        }
```

nemu/include/cpu/instr_helper.h

```
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

```
// macro for generating the implementation of an instruction with two operands
#define make_instr_impl_2op(inst_name, src_type, dest_type, suffix) \
        // 等于 make_instr_impl_2op(mov, r, rm, v)
        make_instr_func(concat7(inst_name, _, src_type, 2, dest_type, _, suffix)) {\
         // 宏展开等于 make_instr_func(mov_ r2rm_v) {
                int len = 1; \ // 不变
                concat(decode_data_size_, suffix) \
// 宏展开等于 decode_data_size_v
// 下方宏定义 #define decode_data_size_v opr_src.data_size = opr_dest.data_size = data_size;
                concat3(decode_operand, _, concat3(src_type, 2, dest_type)) \
// 宏展开等于 decode_operand_r2rm
// 下方宏定义 #define decode_operand_r2rm \
//                  len += modrm_r_rm(eip + 1, &opr_src, &opr_dest);
                print_asm_2(...); \ // 单步执行打印调试信息，不变
                instr_execute_2op(); \ //调用执行函数
                return len; \
        }
```

nemu/include/cpu/instr_helper.h

```
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

```
// macro for generating the implementation of an instruction with two operands
#define make_instr_impl_2op(inst_name, src_type, dest_type, suffix) \
        // 等于 make_instr_impl_2op(mov, r, rm, v)
                make_instr_func(concat7(inst_name, _, src_type, 2, dest_type, _, suffix)) {\
                 // 宏展开等于 make_instr_func(mov_ r2rm_v) {
                                int len = 1; \ // 不变
                                concat(decode_data_size_, suffix) \
// 宏展开等于 decode_data_size_v
// 下方宏定义 #define decode_data_size_v opr_src.data_size = opr_dest.data_size = data_size;
                                concat3(decode_operand, _, concat3(src_type, 2,
dest_type)) \
// 宏展开等于 decode_operand_r2rm
// 下方宏定义 #define decode_operand_r2rm \
//                 len += modrm_r_rm(eip + 1, &opr_src, &opr_dest);
                                print_asm_2(...); \ // 单步执行打印调试信息，不变
                                instr_execute_2op(); \ //调用执行函数
                                return len; \
                }
```

nemu/include/cpu/instr_helper.h

```
        int len = 1;
        // 操作数寻址
        len += modrm_r_rm(eip + 1, &r, &rm);
        // 执行mov操作
        operand_read(&r);
        rm.val = r.val;
        operand_write(&rm);
        // 返回操作数长度
        return len;
    }
```

Static关键字很关键!

```
#include "cpu/instr.h"

static void instr_execute_2op() {
        operand_read(&opr_src);
        opr_dest.val = opr_src.val;
        operand_write(&opr_dest);
}

make_instr_impl_2op(mov, r, rm, v)
```

nemu/src/cpu/instr/mov.c

```
// macro for generating the implementation of an instruction with two operands
#define make_instr_impl_2op(inst_name, src_type, dest_type, suffix) \
        // 等于 make_instr_impl_2op(mov, r, rm, v)
        make_instr_func(concat7(inst_name, _, src_type, 2, dest_type, _, suffix)) {\
         // 宏展开等于 make_instr_func(mov_ r2rm_v) {
                int len = 1; \ // 不变
                concat(decode_data_size_, suffix) \
// 宏展开等于 decode_data_size_v
// 下方宏定义 #define decode_data_size_v opr_src.data_size = opr_dest.data_size = data_size;
                concat3(decode_operand, _, concat3(src_type, 2, dest_type)) \
// 宏展开等于 decode_operand_r2rm
// 下方宏定义 #define decode_operand_r2rm \
//                  len += modrm_r_rm(eip + 1, &opr_src, &opr_dest);
                print_asm_2(...); \ // 单步执行打印调试信息，不变
                instr_execute_2op(); \ //调用执行函数
                return len; \  // 返回指令长度
        }
```

nemu/include/cpu/instr_helper.h

```
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

怎么从左边变到右边的?

```
make_instr_func(mov_r2rm_v) {
    OPERAND r, rm;
    // 指定操作数长度
    rm.data_size = r.data_size = data_size;
    int len = 1;
    // 操作数寻址
    len += modrm_r_rm(eip + 1, &r, &rm);
    // 执行mov操作
    operand_read(&r);
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

```
#include "cpu/instr.h"

static void instr_execute_2op() {
        operand_read(&opr_src);
        opr_dest.val = opr_src.val;
        operand_write(&opr_dest);
}

make_instr_impl_2op(mov, r, rm, v)
// 将其进行宏展开后，变为。。。
```

nemu/src/cpu/instr/mov.c

怎么从左边变到右边的?

nemu/src/cpu/instr/mov.c

```
make_instr_func(mov_r2rm_v) {
    OPERAND r, rm;
    // 指定操作数长度
    rm.data_size = r.data_size = data_size;
    int len = 1;
    // 操作数寻址
    len += modrm_r_rm(eip + 1, &r, &rm);
    // 执行mov操作
    operand_read(&r);
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

等价

```
#include "cpu/instr.h"

static void instr_execute_2op() {
        operand_read(&opr_src);
        opr_dest.val = opr_src.val;
        operand_write(&opr_dest);
}

make_instr_impl_2op(mov, r, rm, v)
// 将其进行宏展开后，变为。。。
make_instr_func(mov_ r2rm_v) {
        int len = 1;
        opr_src.data_size = opr_dest.data_size = data_size;
        len += modrm_r_rm(eip + 1, &opr_src, &opr_dest);
        print_asm_2(…);
        instr_execute_2op();
        return len;
}
```

怎么从左边变到右边的?

nemu/src/cpu/instr/mov.c

```
make_instr_func(mov_r2rm_v) {
    OPERAND r, rm;
    // 指定操作数长度
    rm.data_size = r.data_size = data_size;
    int len = 1;
    // 操作数寻址
    len += modrm_r_rm(eip + 1, &r, &rm);
    // 执行mov操作
    operand_read(&r);
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

等价

```
#include "cpu/instr.h"

static void instr_execute_2op() {
        operand_read(&opr_src);
        opr_dest.val = opr_src.val;
        operand_write(&opr_dest);
}

make_instr_impl_2op(mov, r, rm, v)
// 将其进行宏展开后，变为。。。
make_instr_func(mov_ r2rm_v) {
        int len = 1;
        opr_src.data_size = opr_dest.data_size = data_size;
        len += modrm_r_rm(eip + 1, &opr_src, &opr_dest);
        print_asm_2(...);
        instr_execute_2op();
        return len;
}
```

opr_src和opr_dest是定义在operand.c中的两个全局变量

怎么从左边变到右边的?

nemu/src/cpu/instr/mov.c

```
make_instr_func(mov_r2rm_v) {
    OPERAND r, rm;
    // 指定操作数长度
    rm.data_size = r.data_size = data_size;
    int len = 1;
    // 操作数寻址
    len += modrm_r_rm(eip + 1, &r, &rm);
    // 执行mov操作
    operand_read(&r);
    rm.val = r.val;
    operand_write(&rm);
    // 返回操作数长度
    return len;
}
```

等价

```
#include "cpu/instr.h"

static void instr_execute_2op() {
        operand_read(&opr_src);
        opr_dest.val = opr_src.val;
        operand_write(&opr_dest);
}

make_instr_impl_2op(mov, r, rm, v)
// 将其进行宏展开后，变为。。。
make_instr_func(mov_ r2rm_v) {
        int len = 1;
        opr_src.data_size = opr_dest.data_size = data_size;
        len += modrm_r_rm(eip + 1, &opr_src, &opr_dest);
        print_asm_2(…);
        instr_execute_2op();
        return len;
}
```

modrm系列函数看Guide的描述

内容提要

- 从make run讲起
- 框架代码中宏的理解和运用
- 以正确的心态和方法去应对bug

Bug总是会有的

- 在实现的过程中出现了许多稀奇古怪的bug
- 基本的心理发展过程
  - 第一阶段：**不可能是我的错！**一定是框架代码、编译器、操作系统、虚拟机、CPU……里有bug!
  - 第二阶段：嗯……似乎这里有一点小问题，但是**不至于**吧~
  - 第三阶段：**当初这代码怎么能跑起来的！！！！？？？？**
- 调试公理
  - 机器永远是对的
  - 未测试代码永远是错的

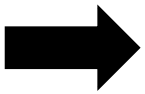
从Fault到Failure

- 一个Bug是怎样最终导致程序出错的?

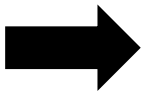
从程序中的Fault到Failure的传播过程

账户余额为负

有一个Fault
也就是Bug



运行时导致
了一个Error



最后导致了
程序的Failure

```
int deposite(int rmb) {
    return balance + rmb;
}
```

```
balance = 2^31 - 1
// 可能时忘记初始化或多次累加导致的
rmb =  10

newBalance = deposite(rmb);
// 计算利息、收费等一系列操作
……
```

防止出错的方法

- 启用严格的静态检查

```
CFLAGS := -ggdb3 -MMD -MP -Wall -Werror -O2 -I./include -I../include
```

现在能理解为何在./nemu/Makefile的编译选项中，加入-Wall和-Werror选项?

- 在尽早让Error变成Failure
  - 设置一系列的检查点，assert，if判断，……

```
int deposite(int rmb) {
    assert(rmb >= 0);
    assert(balance >= 0);
    int  newBalance = balance + rmb;
    assert(newBalance >=0);
    return newBalance;
}
```

防止出错的方法

- 降低程序对预设条件的依赖（永远假设人只要有机会，就一定会犯错）

```
void set_CF_add(uint32_t result, uint32_t src, size_t data_size) {
        result = sign_ext(result & (0xFFFFFFFF >> (32 - data_size)), data_size);
        src = sign_ext(src & (0xFFFFFFFF >> (32 - data_size)), data_size);
        cpu.eflags.CF = result < src;
}
```

- 保持随手测试的习惯
- 不写理解困难的，可能会引起歧义的代码
  - 如后页的代码

内容2.4：无符号和带符号整数的转换

- 课堂习题2.4.2

不要写这种代码！！！！

```
us = -32768;
i = us;
printf("i = %d, 0x%08x\n"

s = -32768;
ui = s;
printf("ui = %d, 0x%08x\n", ui, ui);

us = -32768;
ui = us;
printf("ui = %d, 0x%08x\n", ui, ui);
```

Relative order of conversion between data size and signed/unsigned:
First change the size and then convert between signed/unsigned

ui = -32768, 0xffff8000

经验：同时变长度和无符号/带符号的类型转换的赋值操作别做！宁肯分两步走！

Debugging是需要锻炼的

- 写出不易出错的高质量代码的同时
- Debugging是码农们一生都要面对的问题
  - 基本过程
    - 重现错误（成功一半）：再跑一次、构造新的有针对性的测试用例……
    - 分离和定位root-cause：单步执行、断点……
    - 查看和分析：assert、printf……
    - 总结：不容易犯错的编码方式、构造对测试友好的代码……
  - 踩遍所有的坑，成就伟大程序员

框架代码也是有bug的

- 欢迎大家贡献建议和patch，包括但不限于
  - 逻辑bug
  - 代码风格
  - 交互方式
  - 新的测试用例
  - 新的功能模块
  - 讲课、答疑的内容和方式
- 如果建议最终被采纳
  - 你将作为Contributor被记录在PA最后的彩蛋中

作为学生，你还有老师和助教

- 提问的艺术
  - https://github.com/ryanhanwu/How-To-Ask-Questions-The-Smart-Way/blob/master/README-zh_CN.md
- 我们不愿意或无法回答的问题
  - 没有经过思考或尝试就来问问题
  - 把我们当作human debugger或者试探参考实现
  - 表述不清的问题
    - 如，没有上下文直接问：“我为什么hit bad trap？”
  - 资料里已经讲清楚但没有看资料
- 我们愿意回答的问题
  - 经过思考和努力后，在能够清晰表述问题，并提供一定分析和尝试结果的基础上所提出来的问题
  - 任何其他有趣的问题

作为学生，你还有老师和助教

- 但你也无需太担心
  - 如果问的问题不够友好，我们会回复：“这个问题我无法回答”，并给出简短的理由，并且不存在小本本来记录
  - 如果实在讲不清楚，我们也不会抛弃任何人
- 最关键的原则

You have tried HARD before asking

Happy Hacking O_o

节日快乐