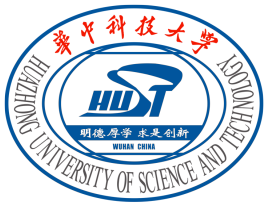

華中科技大學

课 程 实 验 报 告

课程名称：系 统 能 力 培 养

姓　　名：张 丘 洋

学　　号：U201614667

院　　系：计算机科学与技术学院

专业班级：计算机科学与技术 1606 班

指导教师：谭 志 虎

分数	
教师签名	

2019 年 10 月 31 日

目录

1

开发环境

1.1 硬件配置

1.2 软件配置

- gcc 8.3.0
- nvim 0.3.1
- ld 2.32.0
- python 3.6.9

2

硬件平台—NEMU

2.1 前言

编写虚拟机的第一个任务就是去实现其硬件设施。根据冯诺伊曼计算机的思想，一个完整的计算设备需要有运算器、控制器、存储器、输入设备和输出设备，而本章介绍的 NEMU 就是去实现这五大部件。

在今年，南京大学的 NEMU 项目进行了扩展，使 NEMU 提供了三种指令集架构可供选择。一个指令集架构约定了指令的编码方式以及运算器和控制器的解码及执行方式。在 NEMU 中提供了三种指令集架构，分别是：`x86`，`mips32` 和 `riscv32`。在这里我选择的是日常使用中最常见的 `x86` 架构。

2.2 系统设计与实现

下面我们就正式进入冯诺伊曼机 NEMU 的设计与实现。

2.2.1 NEMU 总体架构

由于 NEMU 模拟器是一个冯诺伊曼机，所以其整体架构遵循冯诺伊曼机。NEMU 的总体架构图如图 2.1 所示。

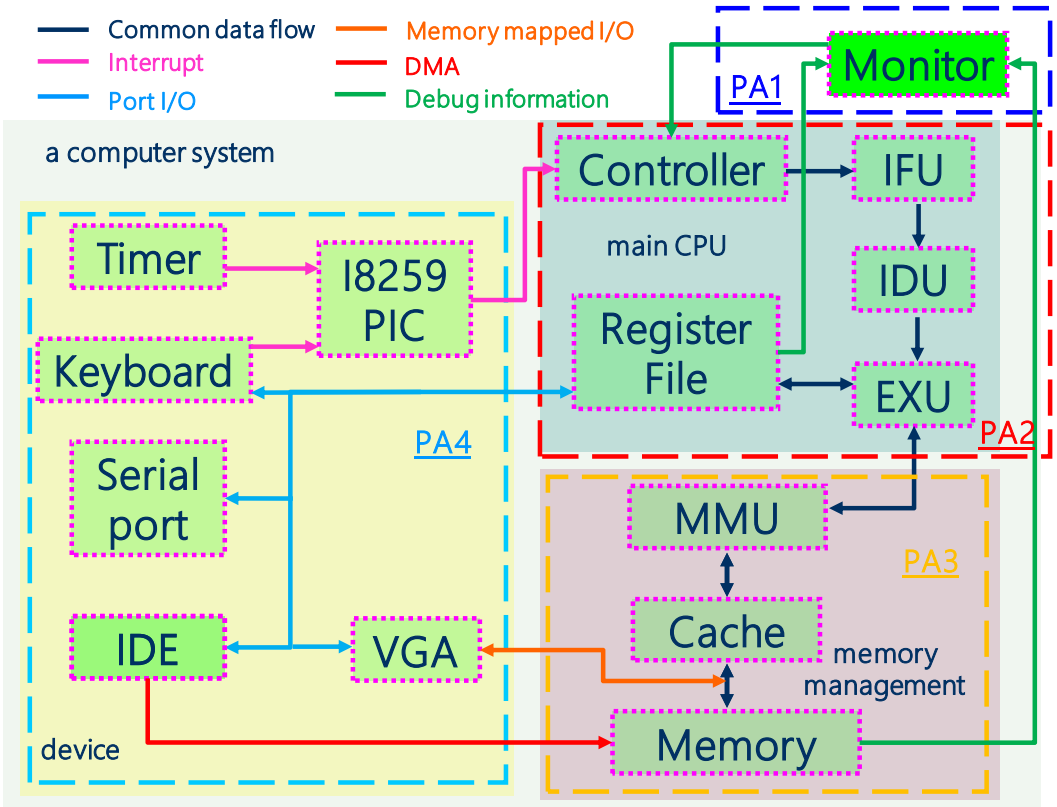

图 2.1: NEMU 总体架构图

2.2.2 框架代码结构

```
nemu/
├── include/
│   ├── cpu/
│   ├── device/
│   ├── memory/ ..... 内存访问有关
│   ├── monitor/ ..... 监视器有关
│   ├── rtl/ ..... 通用 rtl 指令定义
│   ├── common.h ..... 公用头文件
│   ├── debug.h
│   ├── macro.h
│   └── nemu.h
├── src/
│   ├── cpu/ ..... CPU 执行有关
│   ├── device/ ..... IO 设备实现
│   ├── isa/ ..... 指令集架构封装
│   │   ├── mips32/ ..... mips32 指令集
│   │   ├── riscv32/ ..... riscv32 指令集
│   │   └── x86/ ..... x86 指令集
│   ├── memory/ ..... 内存访问实现
│   ├── monitor/
│   │   ├── debug/ ..... 调试器实现
│   │   │   ├── expr/ ..... 表达式解析
│   │   │   │   ├── def.h ..... 表达式解析有关函数定义
│   │   │   │   ├── lex.l ..... 表达式解析词法规则定义
│   │   │   │   └── parser.y ..... 表达式解析语法规则定义
│   │   │   ├── log.c ..... Log 信息输出
│   │   │   ├── ui.c ..... 监视器交互命令实现
│   │   │   └── watchpoint.c ..... 监视点实现
│   │   ├── diff-test/
│   │   ├── cpu-exec.c
│   │   └── monitor.c
│   └── main.c
├── tools/ ..... 测试及调试用工具
└── Makefile
```

```
├── Makefile.git ..... git 版本控制相关
└── runall.sh ..... 一键测试脚本
```

2.2.3 NEMU 执行流

为了能够了解 NEMU 的工作方式，我们来看看 NEMU 整体的一个执行流程。

进入 nemu/src/main.c 文件，能够看到里面定义了 main() 函数。在 main() 函数中只有两行，第一行调用 init_monitor() 函数对 NEMU 进行各项初始化，并根据调用参数来判断本次程序运行是否是批处理模式。第二行调用 ui_mainloop() 函数。ui_mainloop() 函数在 nemu/src/monitor/debug/ui.c 中定义，该函数是监视器与用户进行 IO 交互的主函数。在该函数中首先判断程序是否是批处理模式，若是批处理模式则直接运行在命令行中指定的程序，不会出现与用户的交互界面。若不是批处理模式则会进行循环，在循环体中首先等待用户的命令，之后根据用户所输入的命令调用相应的处理函数。

由此我们可以得到 NEMU 的总体流程图，如图 2.2 所示。

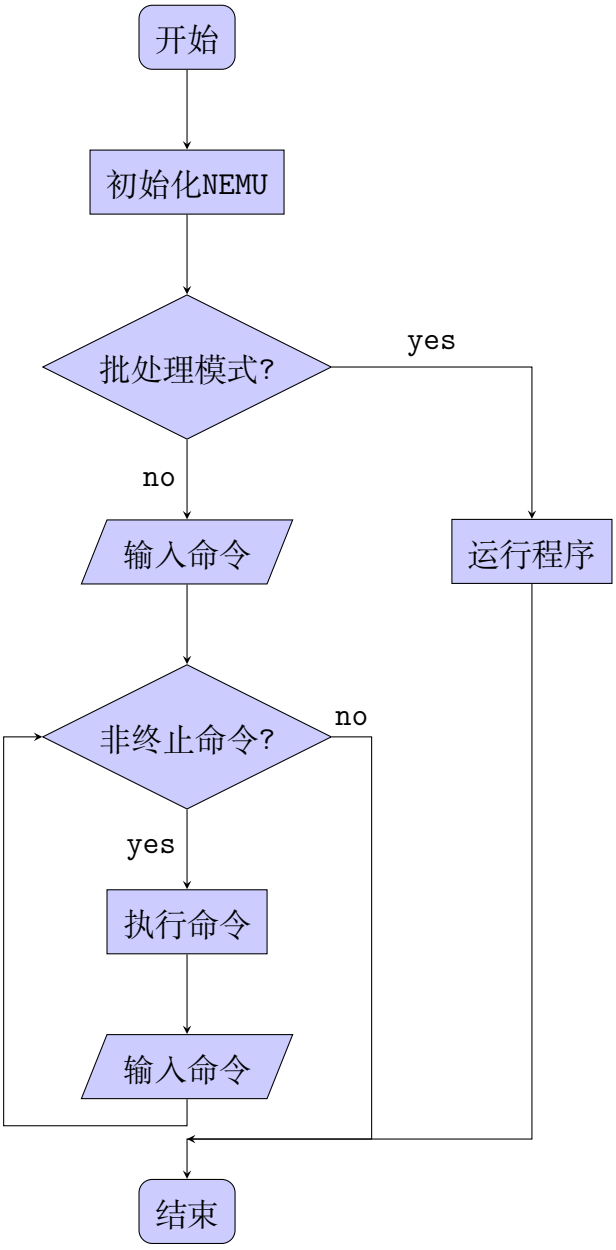

图 2.2: NEMU 整体流程图

3

运行时环境—AM

4

操作系统—nanos-lite

程序清单 4.1: test C

```c
#include <stdio.h>

int main(void) {
  printf("hello, world!\n");
  return 0;
}
```

nemu/src/monitor/debug/expr/expr_def.h

解码帮助函数 decode_op_SI（）没写对，导致后面 and 指令取操作数时 8 位立即数没有扩展，写错的原因是没有认清 rtl_sext（）函数的 width 参数指的是要扩展的数的原始位数。