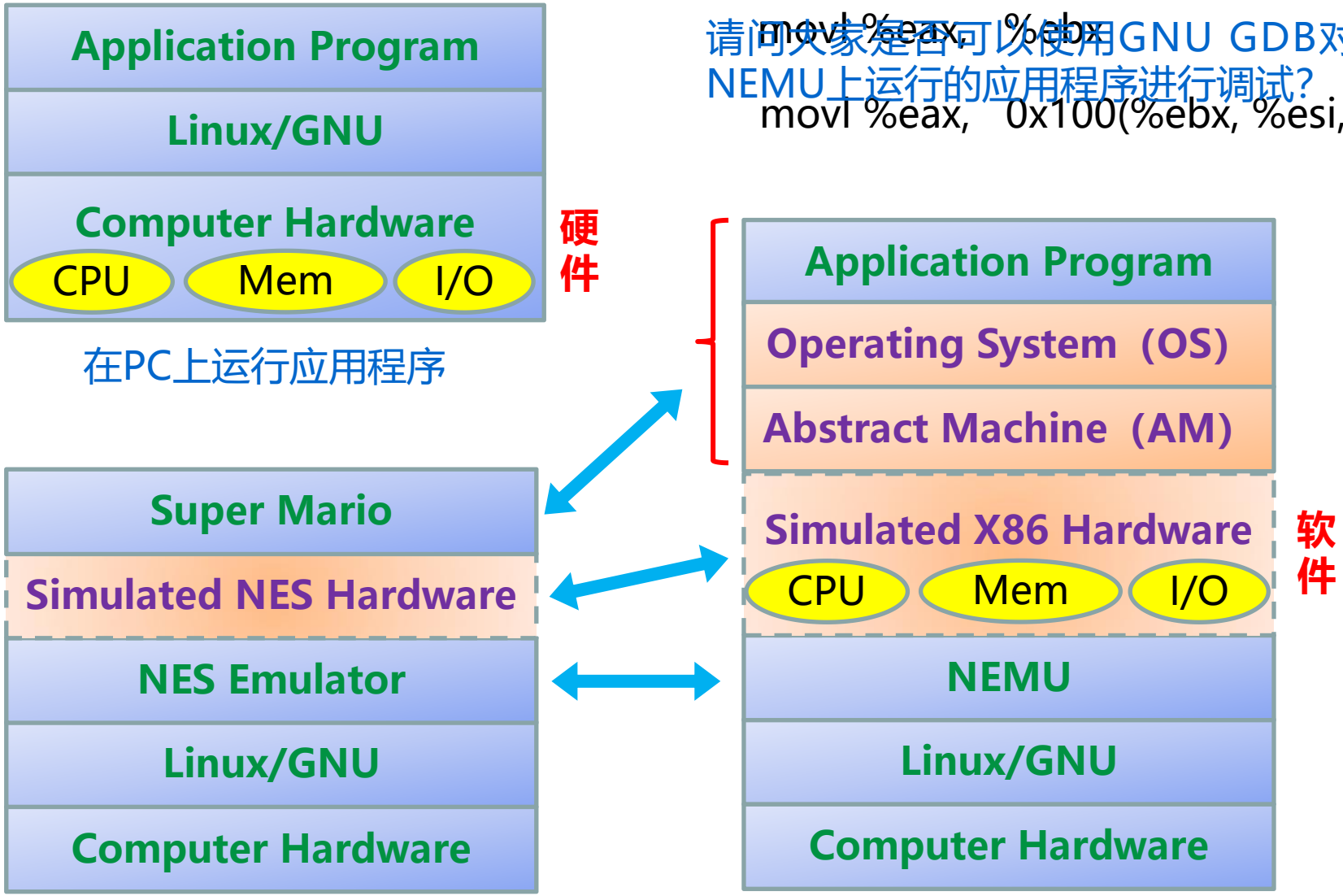
PA1–开天辟地的篇章: 最简单的计算机

PA能做什么？什么是NEMU？



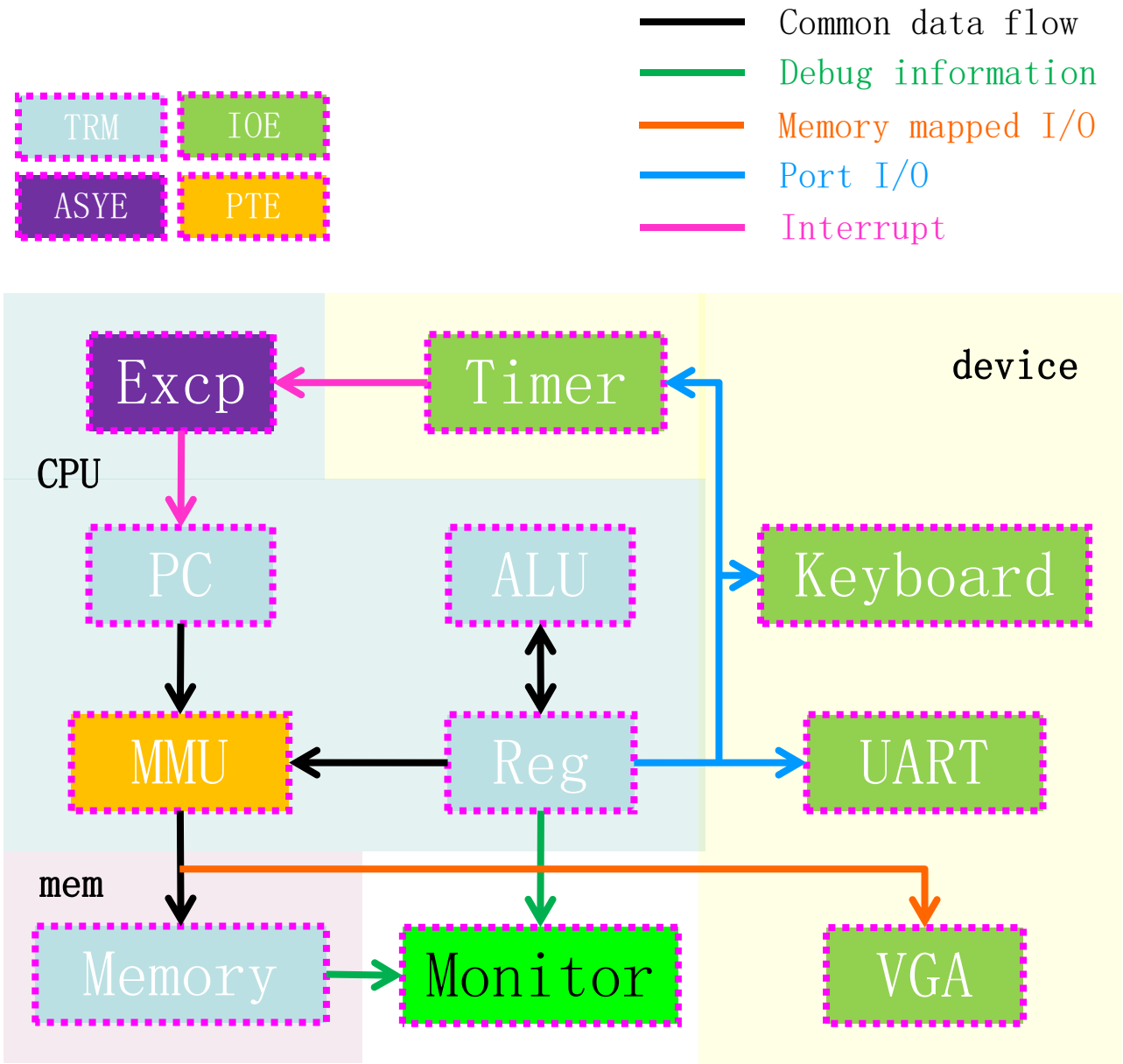
主要内容

- NEMU的主体代码框架（最简单的计算机）
- 简易调试器

PA的文件结构

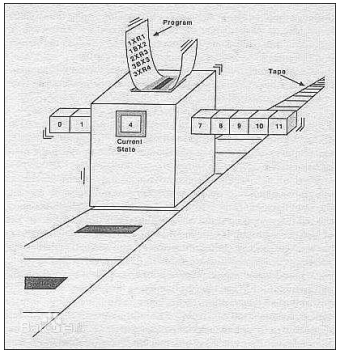
```
ics2017
├── nanos-lite      # 微型操作系统内核
├── navy-apps       # 应用程序集
├── nemu            # NEMU
└── nexus-am        # 抽象计算机
```

NEMU的组成结构


Common data flow
Debug information
Memory mapped I/O
Port I/O
Interrupt
TRM
IOE
ASYE
PTE
device
Excp
Timer
CPU
PC
ALU
Keyboard
MMU
Reg
UART
mem
Memory
Monitor
VGA

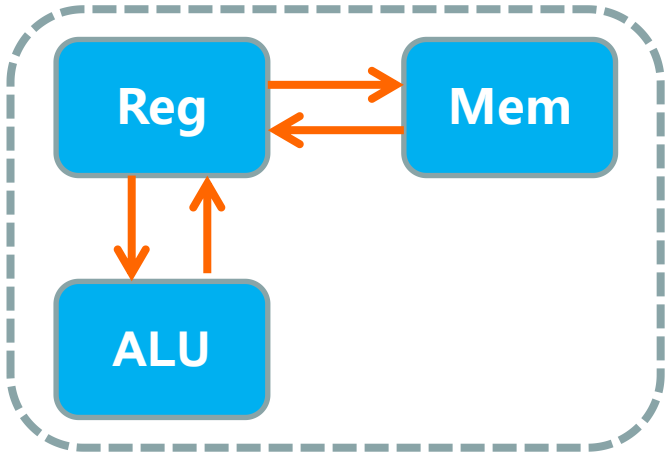
最简单的计算机（图灵机TRM）

最简单的计算机的结构?



- 程序需要有地方放置 -> **存储器**
- 程序需要处理数据 -> **加法器（运算器ALU）**
- 需要高效地暂存处理的中间结果 -> **寄存器**

最简单的计算机的工作方式?



1. 根据PC值从存储器中取出指令
2. 执行指令
3. 更新PC值

图灵机（TRM）：存储程序

NEMU的主体代码框架

ics2017/nemu/src/main.c（注：“~”表示“ics2017/nemu/src”）

~/monitor/monitor.c ~/cpu/reg.c ~/monitor/monitor.c ~/monitor/monitor.c

init_monitor() → **reg_test()** → **load_img()** → **restart()**

测试寄存器结构　加载客户程序　启动计算机运行

设置eip寄存器

设置cr0寄存器

设置EFLAGS

ui_mainloop()

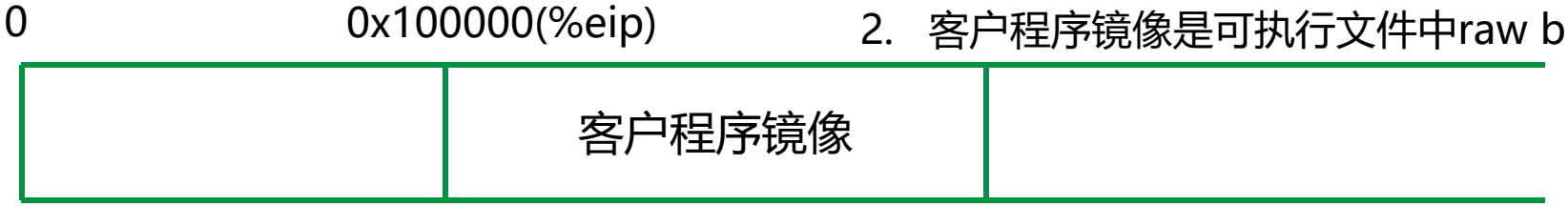
~/monitor/debug/ui.c

模拟器主循环，类似GDB，等待用户输入命令。

输入“c”，进入指令执行主循环“cpu_exec()”

注：

1. 区别于实际的计算机启动流程
2. 客户程序镜像是可执行文件中raw binary



主存空间

实现正确的寄存器结构体

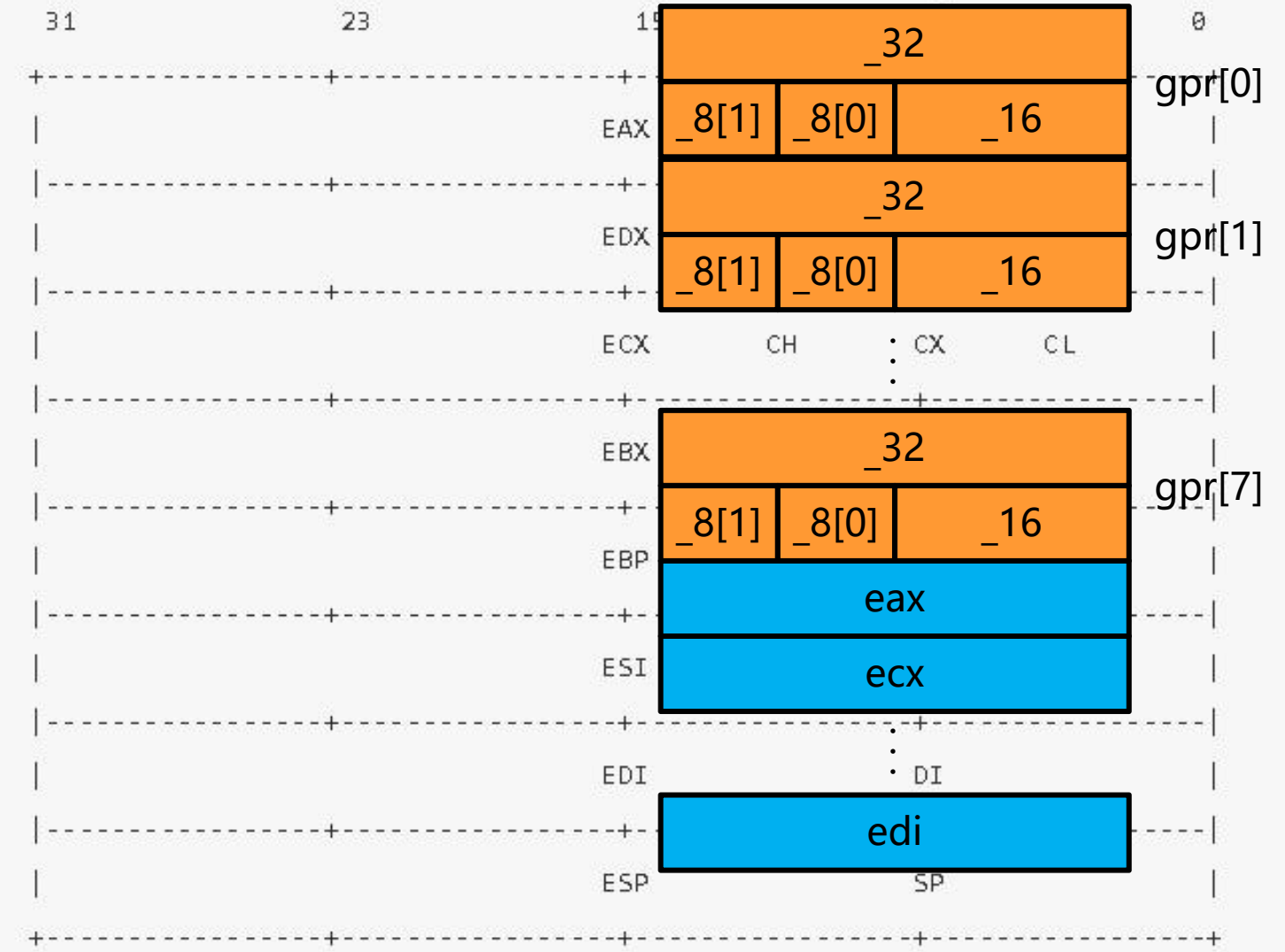
```
nemu/include/

typedef struct {
    struct {
        uint32_t _
        uint16_t _
        uint8_t _8[
    } gpr[8];


    uint32_t eax,

    vaddr_t eip;
    . . . . . .

} CPU_state;
```

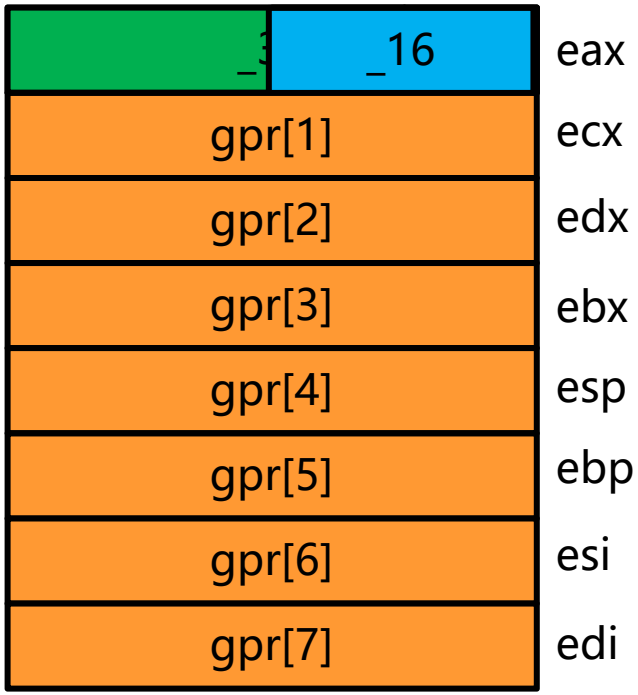


实现正确的寄存器结构体---答案

```
typedef struct {
    union {
        union {
            uint32_t _32;
            uint16_t _16;
            uint8_t _8[2];
        } gpr[8];


        struct {
            rtlreg_t eax, ecx, edx, ebx, esp, ebp, esi, edi;
        };
    };

    vaddr_t eip;
    // ...
} CPU_state;
```



指令执行主循环 — — cpu_exec()

所在文件：nemu/src/monitor/cpu-exec.c

```
void cpu_exec(uint32_t n) {
        if(nemu_state == END) {
                printf(  “Program execution has ended. To restart ….\n");
                return;
        }
        nemu_state = RUNNING;
        . . . . .

        for(; n > 0; n --) {        //每次执行%eip指向的一条指令

                exec_wrapper(print_flag);

                ……..

                if(nemu_state != RUNNING) { return; }
        }

        if(nemu_state == RUNNING) { nemu_state = STOP; }
}
```

指令执行循环退出条件：
1. 达到循环次数
2. 执行了nemu_trap指令

NEMU中的“图灵机TRM”

- 程序需要有地方放置 -> **存储器**
  - 数组pmem（128MB，位于" nemu/src/memory/[illegible]ory.c "）
- 程序需要处理数据 -> **加法器（[illegible]器ALU）**
- 需要高效地暂存[illegible]的中间结果 -> **寄存器**
  - 结构[illegible]_state（位于" nemu/include/cpu/reg.h "）
- TRM的工作方式
  - 函数cpu_exec() （位于" nemu/src/monitor/cpu-exec.c"）

使用软件的数据结构模拟数字电路（虚拟计算机）

主要内容

- NEMU的主体代码框架
- 简易调试器

简易调试器支持的功能

已实现

正则表达式
递归

链表

命令	格式	使用举例	说明
帮助(1)	`help`	`help`	打印命令的帮助信息
继续运行(1)	`c`	`c`	继续运行被暂停的程序
退出(1)	`q`	`q`	退出NEMU
单步执行	`si [N]`	`si 10`	让程序单步执行 `N` 条指令后暂停执行，当 `N` 没有给出时，缺省为 `1`
打印程序状态	`info SUBCMD`	`info r` `info w`	打印寄存器状态 打印监视点信息
表达式求值	`p EXPR`	`p $eax + 1`	求出表达式 `EXPR` 的值，`EXPR` 支持的运算请见调试中的表达式求值小节
扫描内存(2)	`x N EXPR`	`x 10 $esp`	求出表达式 `EXPR` 的值，将结果作为起始内存地址，以十六进制形式输出连续的 `N` 个4字节
设置监视点	`w EXPR`	`w *0x2000`	当表达式 `EXPR` 的值发生变化时，暂停程序执行
删除监视点	`d N`	`d 2`	删除序号为 `N` 的监视点
打印栈帧链(3)	`bt`	`bt`	打印栈帧链

简易调试器支持的功能

- 简易调试器：nemu/src/monitor/debug/ui.c
- 表达式求值： nemu/src/monitor/debug/expr.c
- 监视点： nemu/src/monitor/debug/watchpoint.c

请各位同学阅读“ui.c”中关于单步、打印寄存器状态和扫描内存3个调试功能的代码

总结

- nemu/src/main.c: main()
  - NEMU入口
- nemu/src/monitor/debug/ui.c: ui_mainloop()
  - 用户接口主循环
- nemu/src/monitor/cpu-exec.c: cpu_exec()
  - CPU执行主循环

PA1到此结束

Q&A?