

计算机系统基础  
Programming Assignment

# PA 4 异常、中断与I/O ——PA 4-2 外设与I/O

2018年12月19日

# 第七、八章作业布置

第七章：

pg. 313. 第2、4、5题

第八章：

pg. 348. 第3、4、5、6、8题

作业截止时间：2019年1月5日24时（6日0时）

PA 4-1截止时间：2019年1月5日24时

PA 4-2截止时间：2019年1月19日24时

PA 4-3截止时间：2019年2月3日24时

# 前情提要



PA 1 ~ PA 3构建的由  
CPU和内存构成的计算  
机，配合Kernel的支持，  
已经拥有了超强的运  
算和管理能力

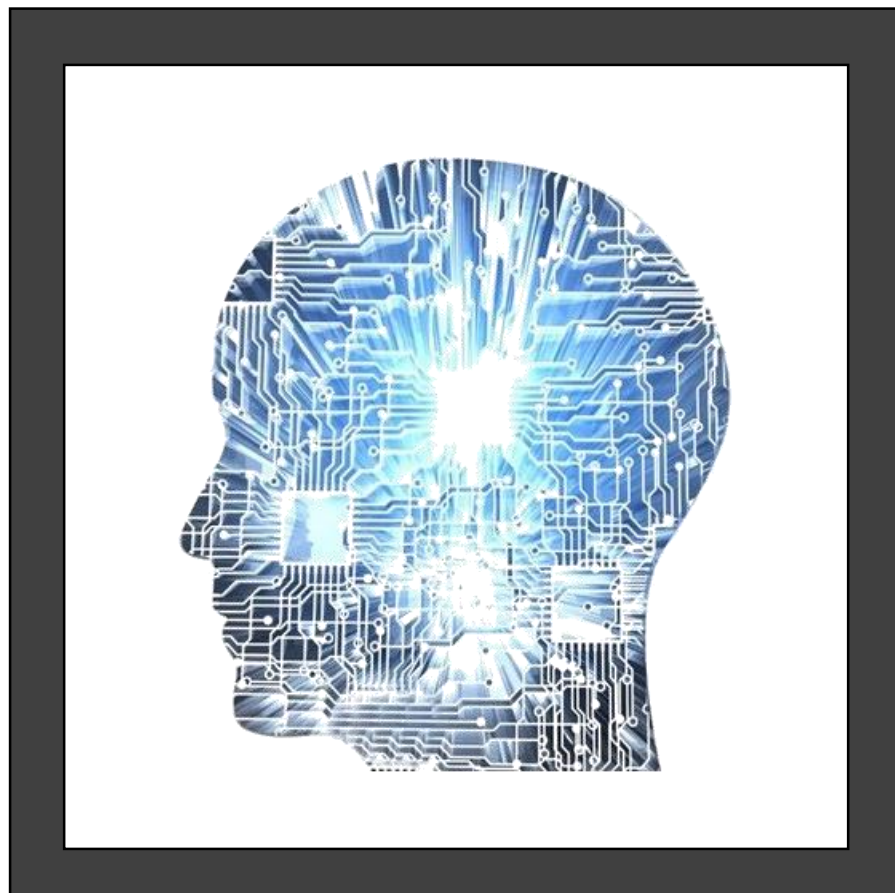


使用PA 4-1构建的“中断  
之棍”戳它还能有反应



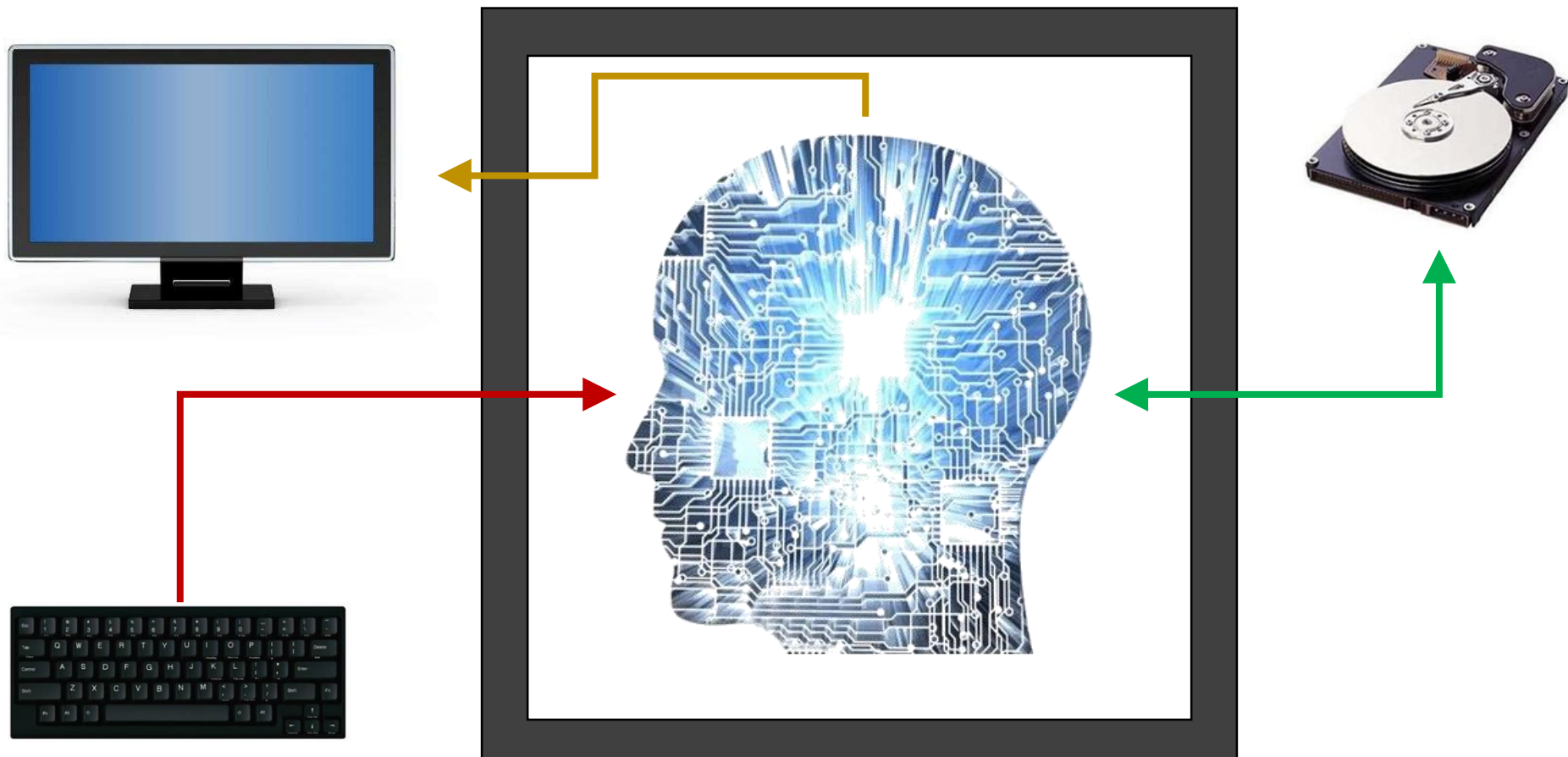
# 前情提要

但是！缺少输入和输出的能力，  
基本上还是封闭在机箱的内部。



# PA 4-2的任务

为它接上眼睛和嘴巴，完成实现一台现代计算机的“最后的拼图”。

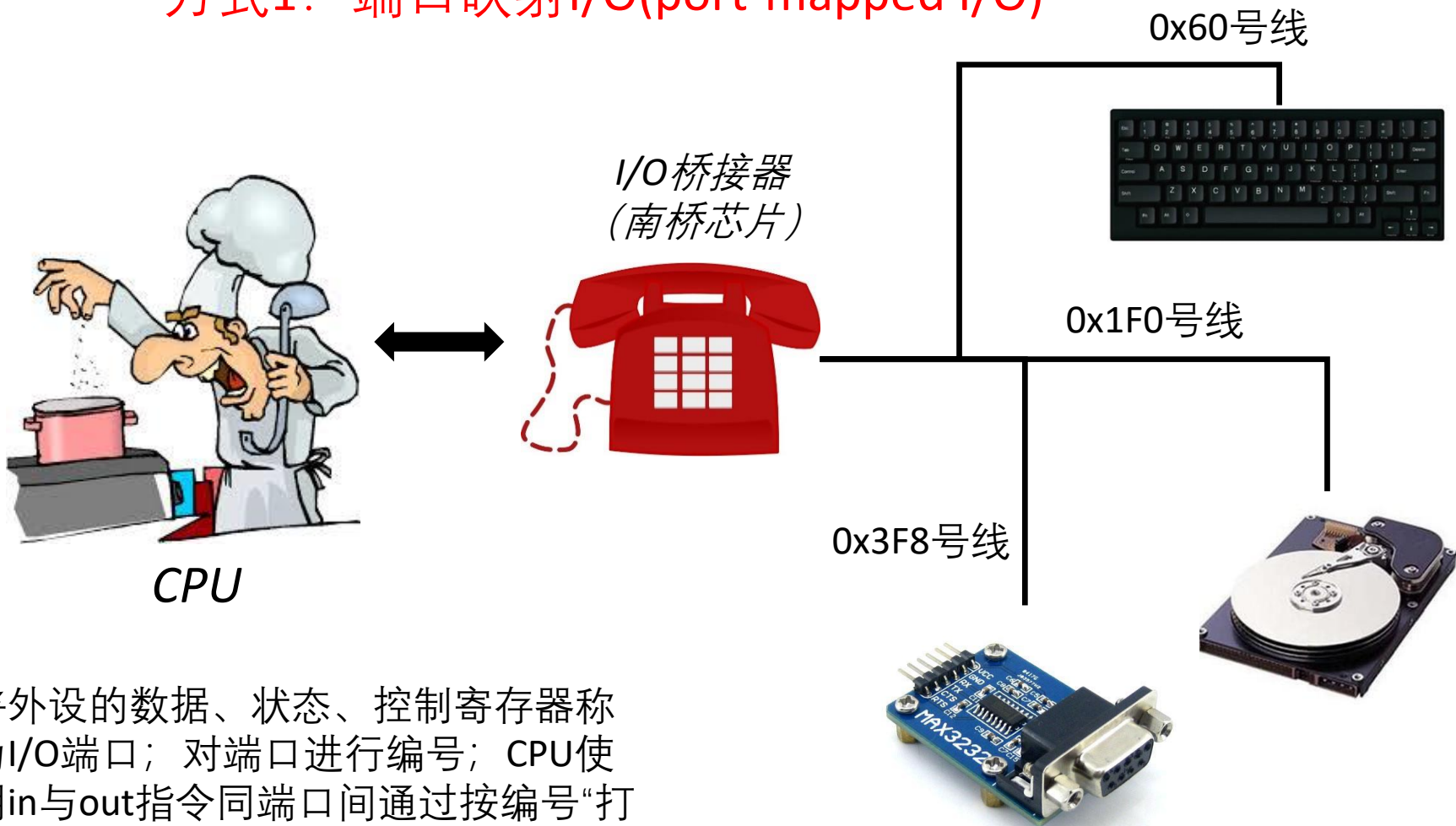


# 外设与I/O

- CPU完成与外设通信的几种方式
  - 方式1： 端口映射I/O(port-mapped I/O)
  - 方式2： 内存映射I/O (Memory Mapped I/O, mmio)

# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式1：端口映射I/O(port-mapped I/O)



将外设的数据、状态、控制寄存器称为I/O端口；对端口进行编号；CPU使用in与out指令同端口间通过按编号“打电话”的方式通信

# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式1： 端口映射I/O(port-mapped I/O)

*nemu/src/device/io/port\_io.c*

```
#define IO_PORT_SPACE 65536
static uint8_t io_port[IO_PORT_SPACE]; // 65535个8位的I/O端口

static struct pio_handler_map {
    uint16_t port;
    pio_handler handler;
} pio_handler_table [] = { // 端口映射表
    // 格式 {port, handler}
};

// called by the out instruction, 写端口
void pio_write(uint16_t port, size_t len, uint32_t data) {...}

// called by the in instruction, 读端口
uint32_t pio_read(uint16_t port, size_t len) {...}
```



# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式1： 端口映射I/O(port-mapped I/O)

*nemu/src/device/io/port\_io.c*

```
#define IO_PORT_SPACE 65536
static uint8_t io_port[IO_PORT_SPACE]; // 65535个8位的I/O端口

static struct pio_handler_map {
    uint16_t port;
    pio_handler handler;
} pio_handler_table [] = { // 端口映射表
    // 格式 {port, handler}
};

// called by the out instruction, 写端口
void pio_write(uint16_t port, size_t len, uint32_t data) {...}

// called by the in instruction, 读端口
uint32_t pio_read(uint16_t port, size_t len) {...}
```

*nemu/src/device/dev/xxx.c*

```
make_pio_handler(handler_xxx) {
    ...
}
```

访问（读/写）这个端口，  
引起对handler的调用

# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式1：端口映射I/O(port-mapped I/O)

设备制造商和OS可以约定，  
如：我占用哪几个端口，  
控制端口写0我就读，端口  
写1我就写.....

*nemu/src/device/io/port\_io.c*

```
#define IO_PORT_SPACE 65536
static uint8_t io_port[IO_PORT_SPACE]; // 65535个8位的I/O端口

static struct pio_handler_map {
    uint16_t port;
    pio_handler handler;
} pio_handler_table [] = { // 端口映射表
    // 格式 {port, handler}
};

// called by the out instruction, 写端口
void pio_write(uint16_t port, size_t len, uint32_t data) {...}

// called by the in instruction, 读端口
uint32_t pio_read(uint16_t port, size_t len) {...}
```

*nemu/src/device/dev/xxx.c*

```
make_pio_handler(handler_xxx) {
    ...
}
```

访问（读/写）这个端口，  
引起对handler的调用

# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式1：端口映射I/O(port-mapped I/O)

设备制造商和OS可以约定，  
如：我占用哪几个端口，  
控制端口写0我就读，端口  
写1我就写.....

*nemu/src/device/io/port\_io.c*

```
#define IO_PORT_SPACE 65536
static uint8_t io_port[IO_PORT_SPACE]; // 65535个8位的I/O端口
```

```
static struct pio_handler_map {
    uint16_t port;
    pio_handler handler;
} pio_handler_table [] = { // 端口映射表
    // 格式 {port, handler}
};
```

```
// called by the out instruction, 写端口
```

```
void pio_write(uint16_t port, size_t len, uint32_t data) {...}
```

```
// called by the in instruction, 读端口
```

```
uint32_t pio_read(uint16_t port, size_t len) {...}
```

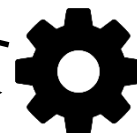
*nemu/src/device/dev/xxx.c*

```
make_pio_handler(handler_xxx) {
    ...
}
```

OS中包含的驱动程序熟知  
这些约定，便可通过in和  
out指令完成对设备的控制  
和数据读写（直接控制法）

out

in



# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式1：端口映射I/O(port-mapped I/O)
  - NEMU中典型的端口映射I/O设备

- 串口 (Serial)

- 端口映射: [nemu/src/device/io/port\\_io.c](#)

```
{SERIAL_PORT + [0-7], handler_serial}
```

- 设备模拟: [nemu/src/device/dev/serial.c](#)

```
make_pio_handler(handler_serial) {...} // 响应端口读写
```

- 驱动程序: [kernel/src/lib/serial.c](#)

```
void serial_printc(char ch) { // 请你实现  
    while (!serial_idle()); // wait until serial is idle  
}
```

# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式1：端口映射I/O(port-mapped I/O)
  - NEMU中典型的端口映射I/O设备
    - 串口 (Serial)
      - 端口映射： `nemu/src/device/io/port_io.c`

```
{SERIAL_PORT + [0-7], handler_serial}
```

- 设备模拟： `nemu/src/device/dev/serial.c`

```
make_pio_handler(handler_serial) {...} // 响应端口读写
```

- 驱动程序： `kernel/src/lib/serial.c`

```
void serial_printc(char ch) { // 请你实现  
    while (!serial_idle()); // wait until serial is idle  
}
```

任务



§4-2.3.1 完成串口的模拟

1. 在 `include/config.h` 中定义宏 `HAS_DEVICE_SERIAL` 并 `make clean`;
2. 实现in和out指令;
3. 实现`serial_printc()`函数;
4. 运行`hello-inline`测试用例，对比实现串口前后的输出内容的区别。

# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式1：端口映射I/O(port-mapped I/O)
  - NEMU中典型的端口映射I/O设备
    - 硬盘 (IDE)
      - 端口映射： [nemu/src/device/io/port\\_io.c](#)

```
{IDE_PORT_BASE + [0-7], handler_id}
```

- 设备模拟： [nemu/src/device/dev/ide.c](#)

```
make_pio_handler(handler_id) {...} // 响应端口读写
```

- 驱动程序： [kernel/src/driver/disk.c](#) // 底层驱动  
[kernel/src/driver/ide.c](#) // 上层磁盘读写接口

```
void ide_read(uint8_t *buf, uint32_t offset, uint32_t len);  
void ide_write(uint8_t *buf, uint32_t offset, uint32_t len);
```

# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式1：端口映射I/O(port-mapped I/O)
  - NEMU中典型的端口映射I/O设备
    - 硬盘 (IDE)
      - 端口映射： `nemu/src/device/io/port_io.c`

```
{IDE_PORT_BASE + [0-7], handler_id}
```

- 设备模拟： `nemu/src/device/dev/ide.c`

```
make_pio_handler(handler_id) {...} // 响应端口读写
```

任务  
↓

- §4-2.3.2 通过硬盘加载程序
- 驱动程序： `kernel/src/driver/disk.c` // 底层驱动  
`kernel/src/driver/ide.c` // 上层磁盘读写接口

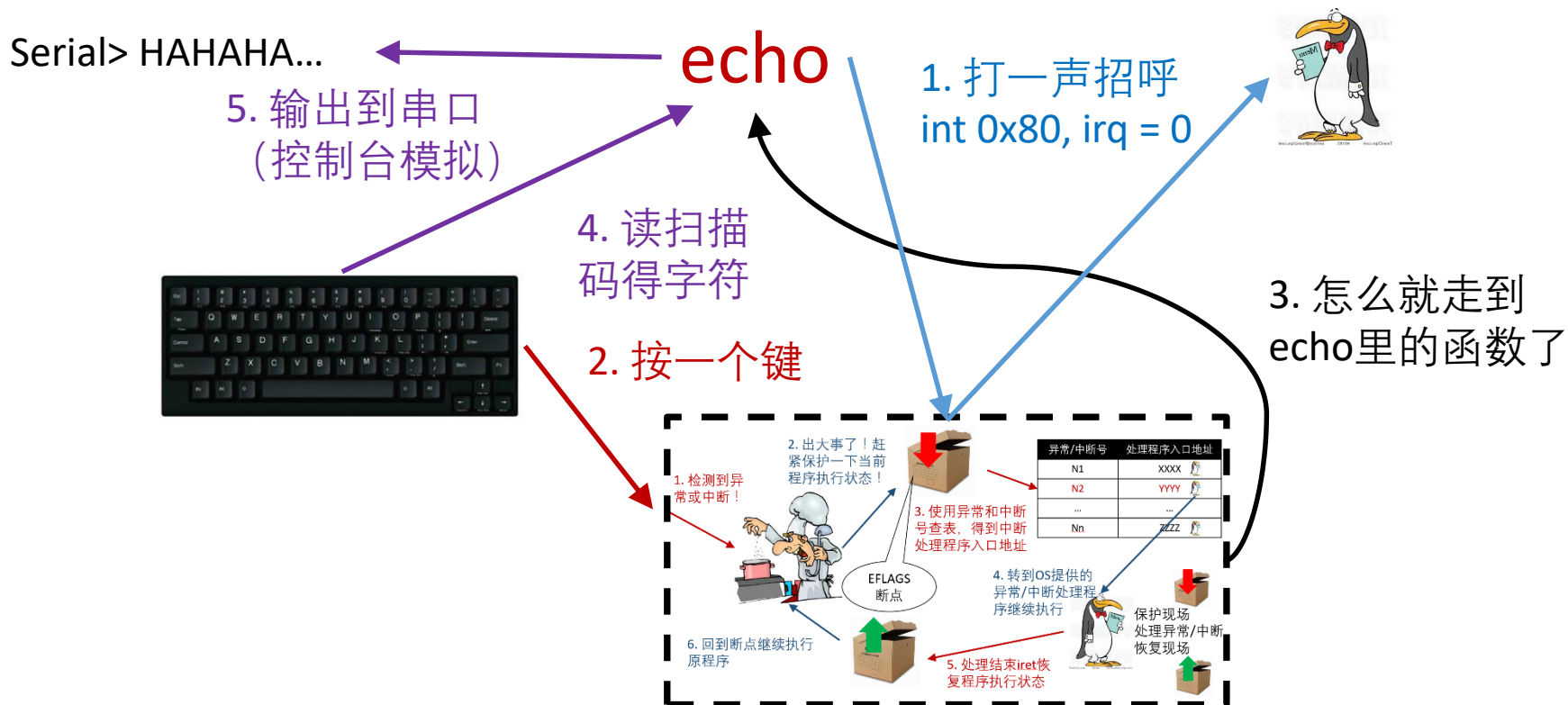
```
void ide_read(uint8_t *buf, uint32_t offset, uint32_t len);  
void ide_write(uint8_t *buf, uint32_t offset, uint32_t len);
```

1. 在 `include/config.h` 中定义宏 `HAS_DEVICE_IDE` 并 `make clean`;
2. 修改Kernel中的 `loader()`，使其通过 `ide_read()` 和 `ide_write()` 接口实现从模拟硬盘加载用户程序；

3. 通过 `make test_pa-4-2` 执行测试用例，验证加载过程是否正确。提示：有些接口这里用不到咱就不用

# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式1：端口映射I/O(port-mapped I/O)
  - NEMU中典型的端口映射I/O设备
    - 键盘（Keyboard）：结合echo程序彻底理解流程，这里给点提示





# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式2: 内存映射I/O (Memory Mapped I/O, mmio)
  - NEMU中典型的内存映射I/O设备
    - VGA



我在冰箱这一块地方放的小纸条就是给你看的



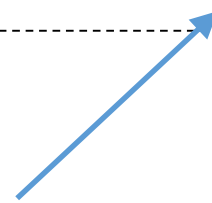
哦

我们约定内存从物理地址0xa0000开始，长度为  $320 \times 200$  字节的区间为显存区间

# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式2：内存映射I/O (Memory Mapped I/O, mmio)
  - NEMU中典型的内存映射I/O设备
    - VGA

```
paddr_read/write() {  
    if(is_mmio(paddr) == -1) {  
        ...  
    } else {  
        mmio_read/write()  
    }  
}
```



## §4-2.3.4 实现VGA的MMIO

1. 在include/config.h中定义宏HAS\_DEVICE\_VGA;
2. 在nemu/src/memory/memory.c中添加mm\_io判断和对应的读写操作;
3. 在kernel/src/memory/vmem.c中完成显存的恒等映射;
4. 通过make test\_pa-4-2执行测试用例，观察输出测试颜色信息，并通过video\_mapping\_read\_test()。

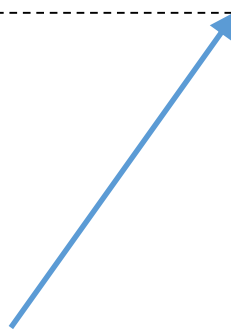
# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式2：内存映射I/O (Memory Mapped I/O, mmio)
  - NEMU中典型的内存映射I/O设备
    - VGA

## §4-2.3.4 实现VGA的MMIO

1. 在include/config.h中定义宏HAS\_DEVICE\_VGA;
2. 在nemu/src/memory/memory.c中添加mm\_io判断和对应的读写操作;
3. 在kernel/src/memory/vmem.c中完成显存的恒等映射;
4. 通过make test\_pa-4-2执行测试用例，观察输出测试颜色信息，并通过video\_mapping\_read\_test()。

```
create_video_mapping() {  
    // 0xa0000    ->    0xa0000  
    //   + SCR_SIZE ->    + SCR_SIZE  
    // 虚拟地址      物理地址  
}
```

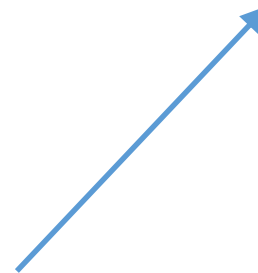
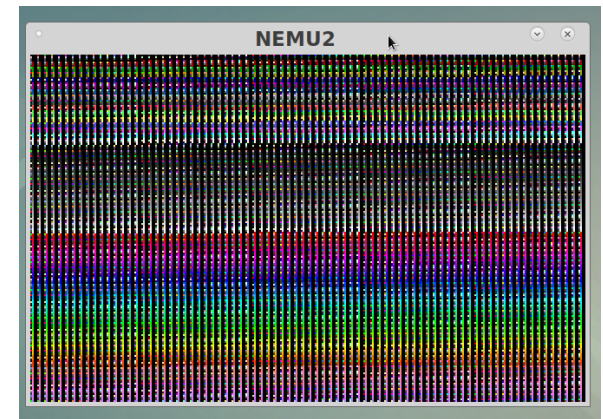


# 外设与I/O

- CPU如何向外设进行输入输出？
  - 方式2：内存映射I/O (Memory Mapped I/O, mmio)
  - NEMU中典型的内存映射I/O设备
    - VGA

## §4-2.3.4 实现VGA的MMIO

1. 在include/config.h中定义宏HAS\_DEVICE\_VGA;
2. 在nemu/src/memory/memory.c中添加mm\_io判断和对应的读写操作;
3. 在kernel/src/memory/vmem.c中完成显存的恒等映射;
4. 通过make test\_pa-4-2执行测试用例，观察输出测试颜色信息，并通过video\_mapping\_read\_test()。



# I/O的控制方式

- 基本方式
  - 直接控制法
  - 中断控制法
  - DMA控制法
- 在PA的实现中，大多数设备采用直接控制法，Audio的实现采用了DMA控制法，有兴趣的同学可以去阅读相应代码

# 打字小游戏与仙剑（选做任务）

这一部分的代码和教程都相对比较老了，属于对老版本致敬的部分，会有一些不一致，估计也最多再使用一个学期，希望有志之士参与重构，成为核心开发者。

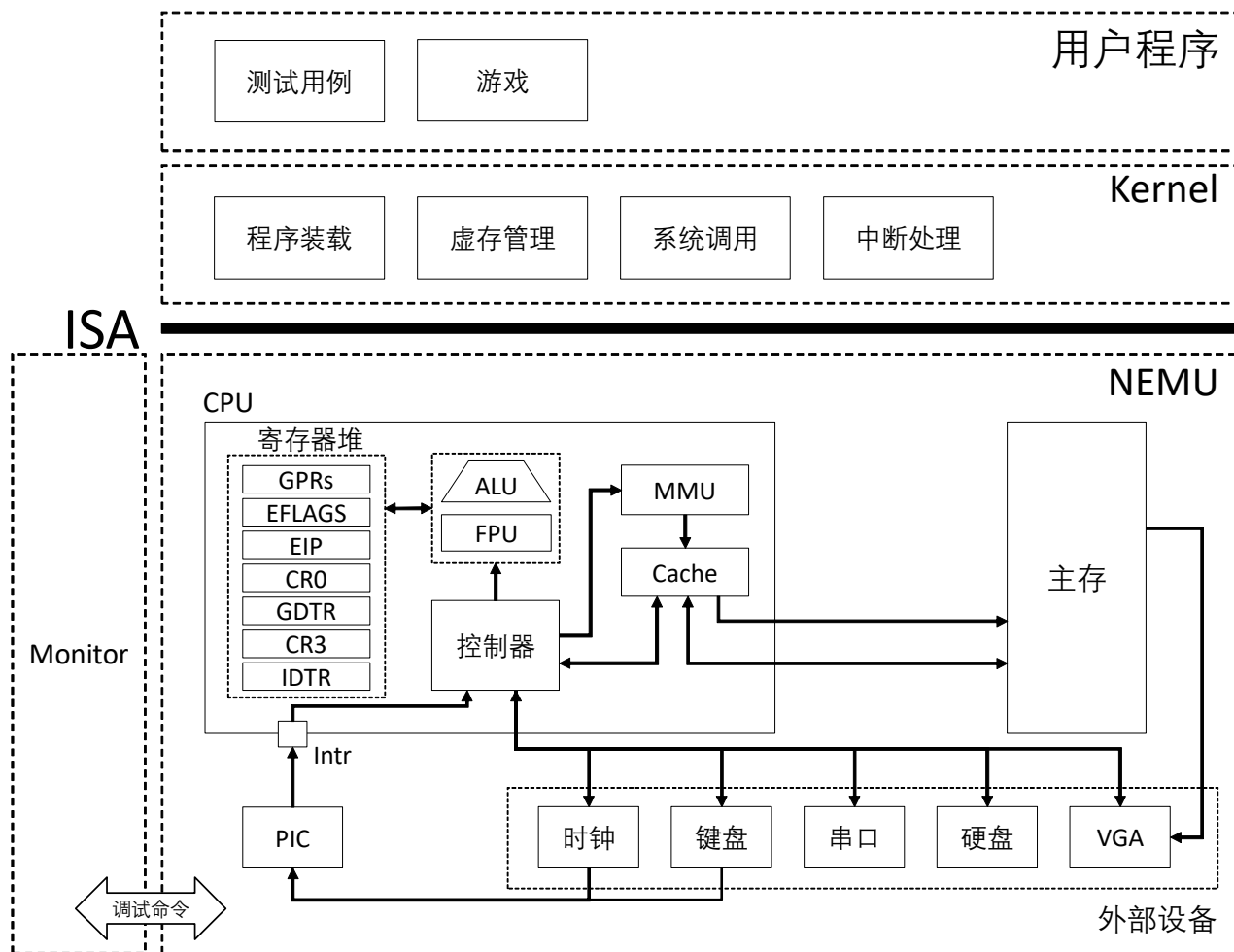
作为最后的挑战任务，以理解框架代码和debug为主。加油了

建议关调cache

所有数据文件已经存放到game/data/文件夹下了，不需要再下载

执行命令：make test\_pa-4-3

# PA的构成 – 路线图 回顾



PA 到此结束

祝大家学习快乐，身心健康！