

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

2020年12月31日 / 2021年1月1日
南京大学《计算机系统基础》课程组

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，已经拥有了很强的运算能力



中断

使用PA 4-1模拟的中断机制能够让机器对外部中断产生响应

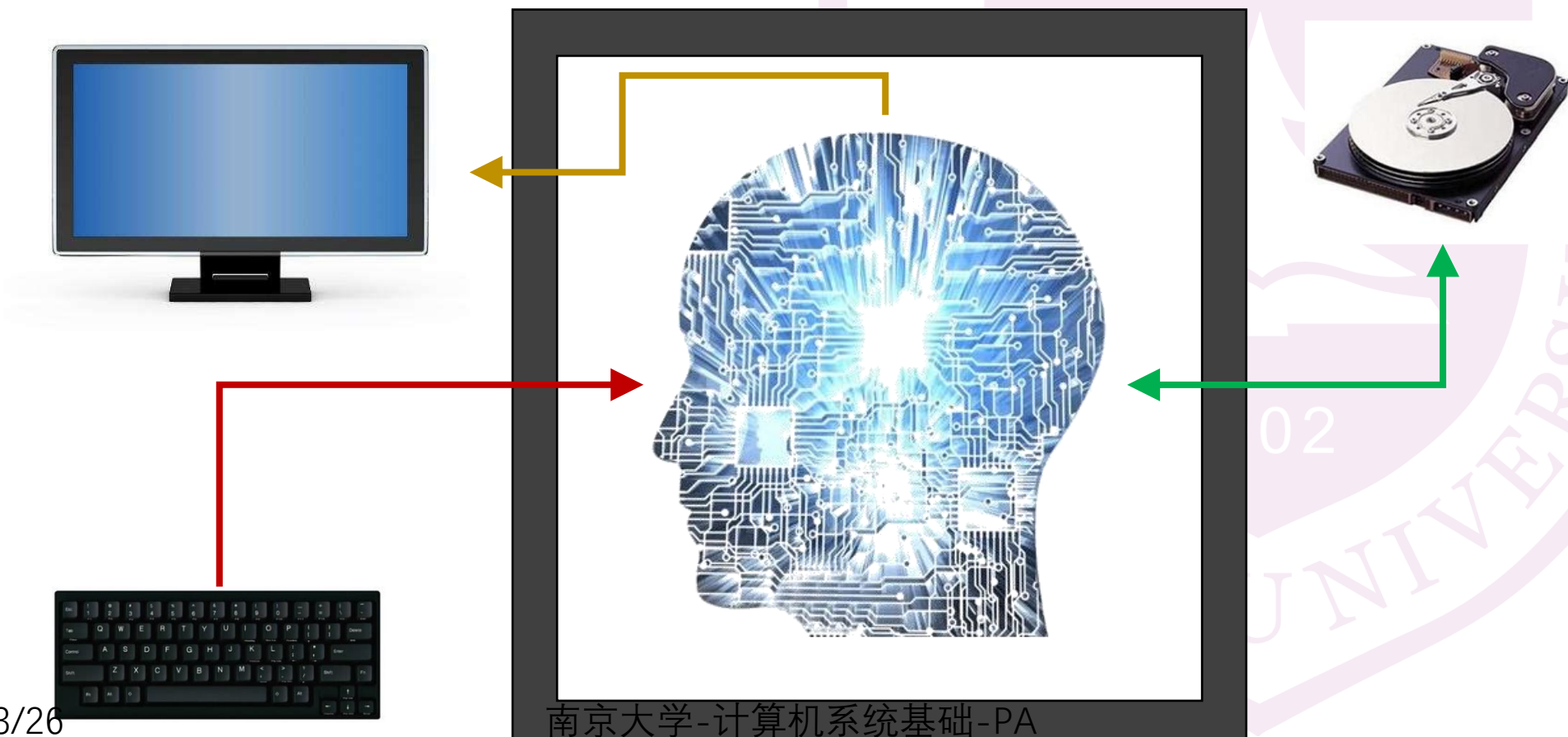
前情提要

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”。



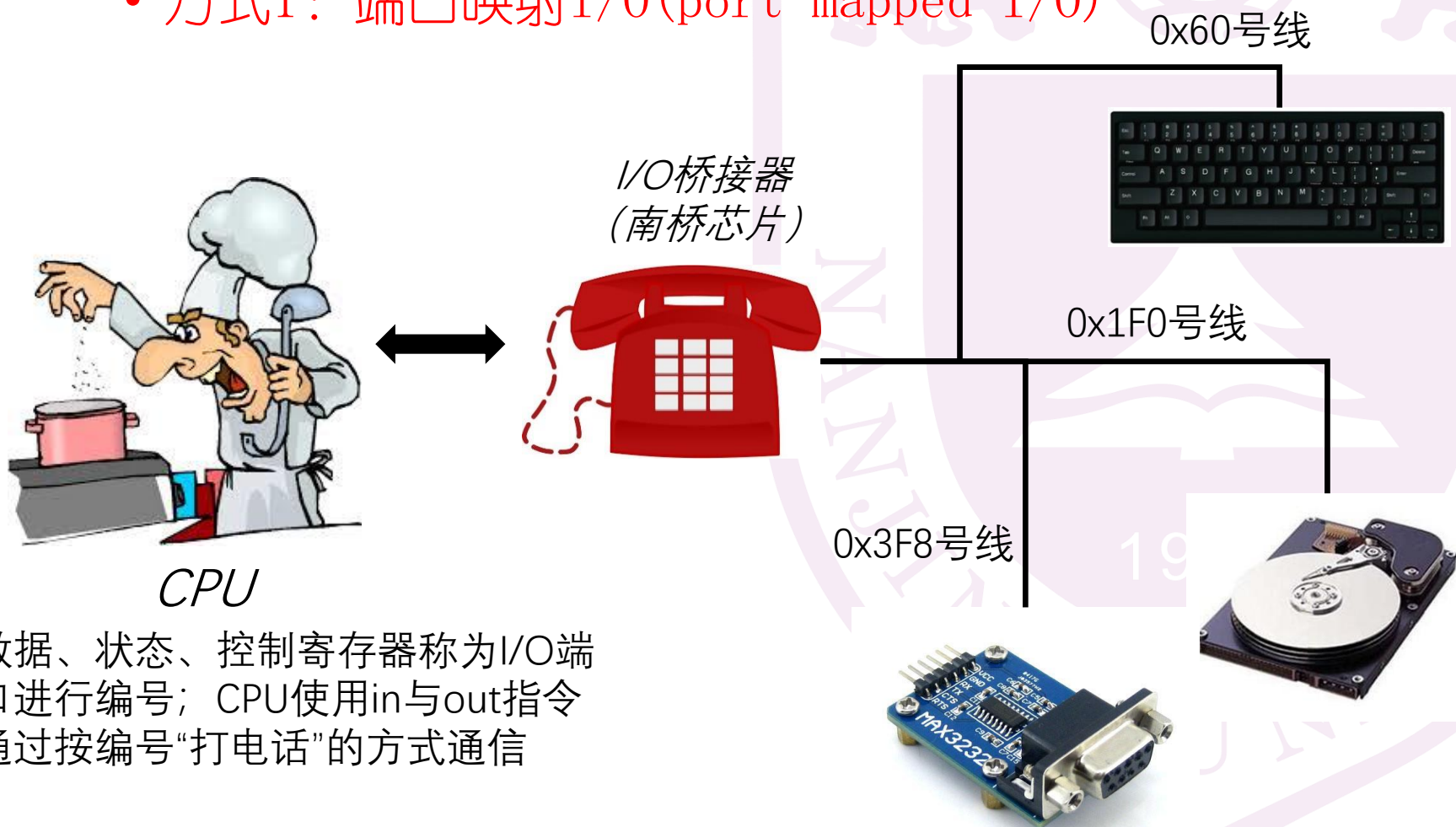
外设与I/O

- 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)

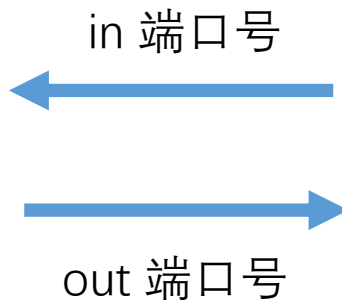
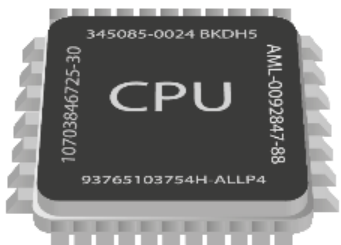
- NEMU中CPU完成与外设通信的几种方式
 - 方式1：端口映射I/O (port-mapped I/O)
 - 串口 (Serial)、键盘 (Keyboard)、硬盘 (IDE)
 - 方式2：内存映射I/O (Memory Mapped I/O, mmio)
 - 显卡 (VGA)
 - 其它只需要理解：
 - 声卡 (Audio) 实验性质
 - 时钟 (Timer) 只产生时钟中断PA 4-1

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



65536个I/O
端口，可以
由16位无符
号整型编码

I/O端口地址（编号）

0	字节
1	字节
2	字节
3	字节
4	字节
5	字节
...	...
65534	字节
65535	字节

I/O端口
(设备控制器中的数据、
状态、控制寄存器)

0x60

键盘

0x1F0 ~ 0x1F7

硬盘

```
movb $'A', %al
movw $0x3F8, %dx
outb %al, %dx
```

0x3F8 ~ 0x3FF

串口

NEMU中I/O端口和
设备间的对应关系

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) {...}
```

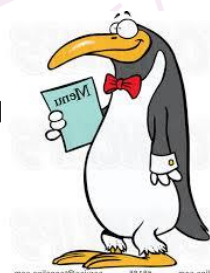
nemu/src/device/dev/xxx.c

```
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



看代码

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  
}
```

看代码

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);
```

看代码

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)

任务



§4-2.3.2 通过硬盘加载程序

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

- 端口映射：`nemu/src/device/io/port_io.c`

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

- 设备模拟：`nemu/src/device/dev/ide.c`

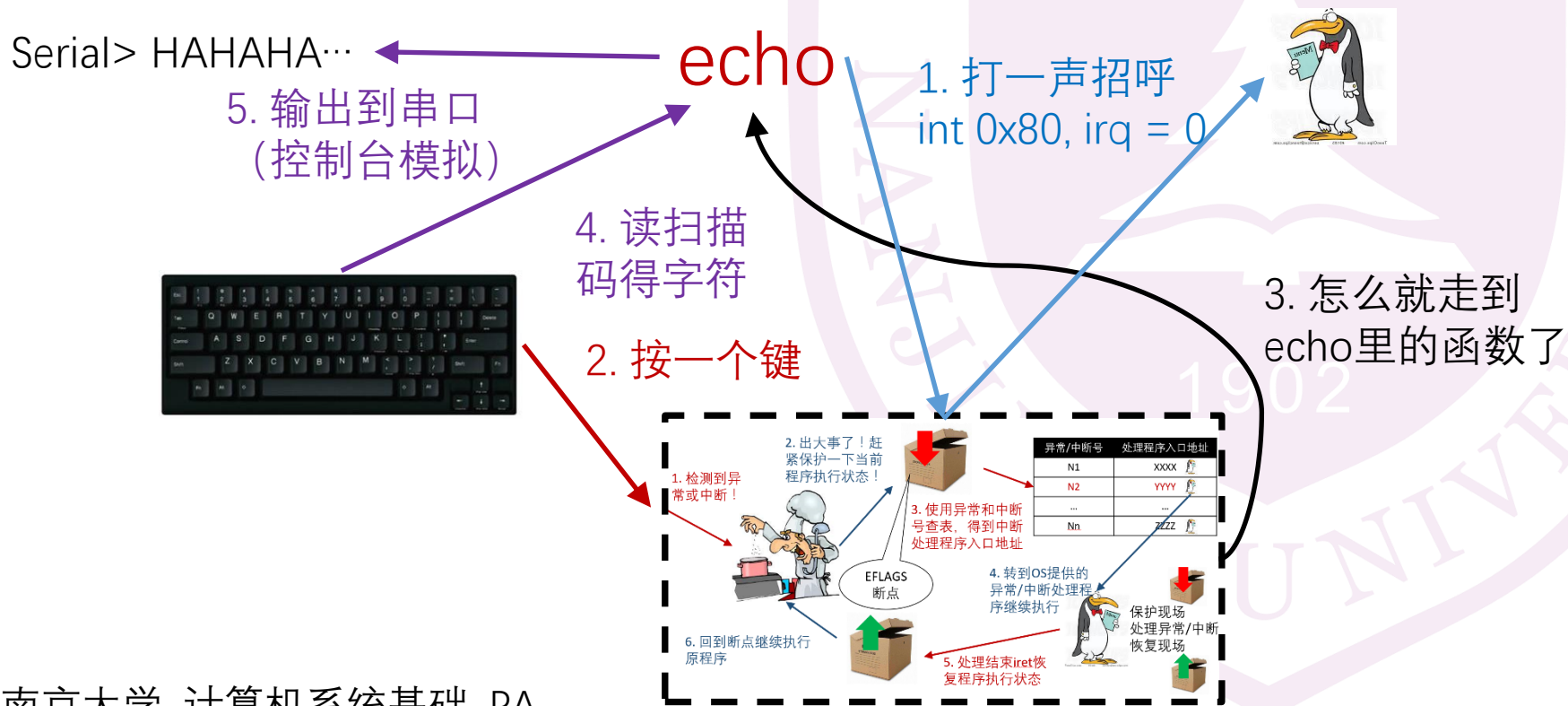
```
make_pio_handler(handler_ide) {...} // 响应端口读写
```

- 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);
```

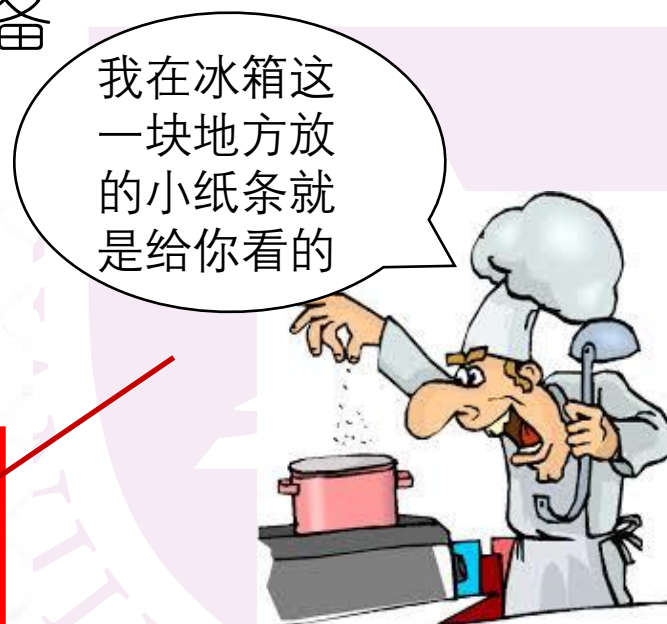
外设与I/O

- 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
 - 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



我们约定内存从物理地址0xa0000开始，长度为 320×200 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

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

§4-2.3.4 实现VGA的MMIO

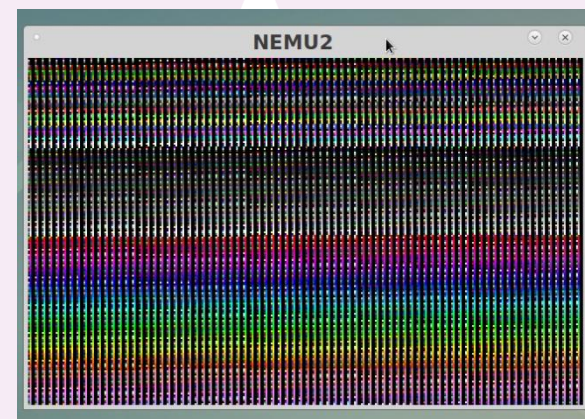
1. 在include/config.h中定义宏HAS_DEVICE_VGA;
2. 在nemu/src/memory/memory.c中添加mm_io判断和对应的读写操作;
3. 在kernel/src/memory/vmem.c中完成显存的恒等映射;
4. 通过make test_pa-4-2执行测试用例，观察输出测试颜色信息，并通过video_mapping_read_test()。

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()。



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，希望有志之士参与重构，成为核心开发者。

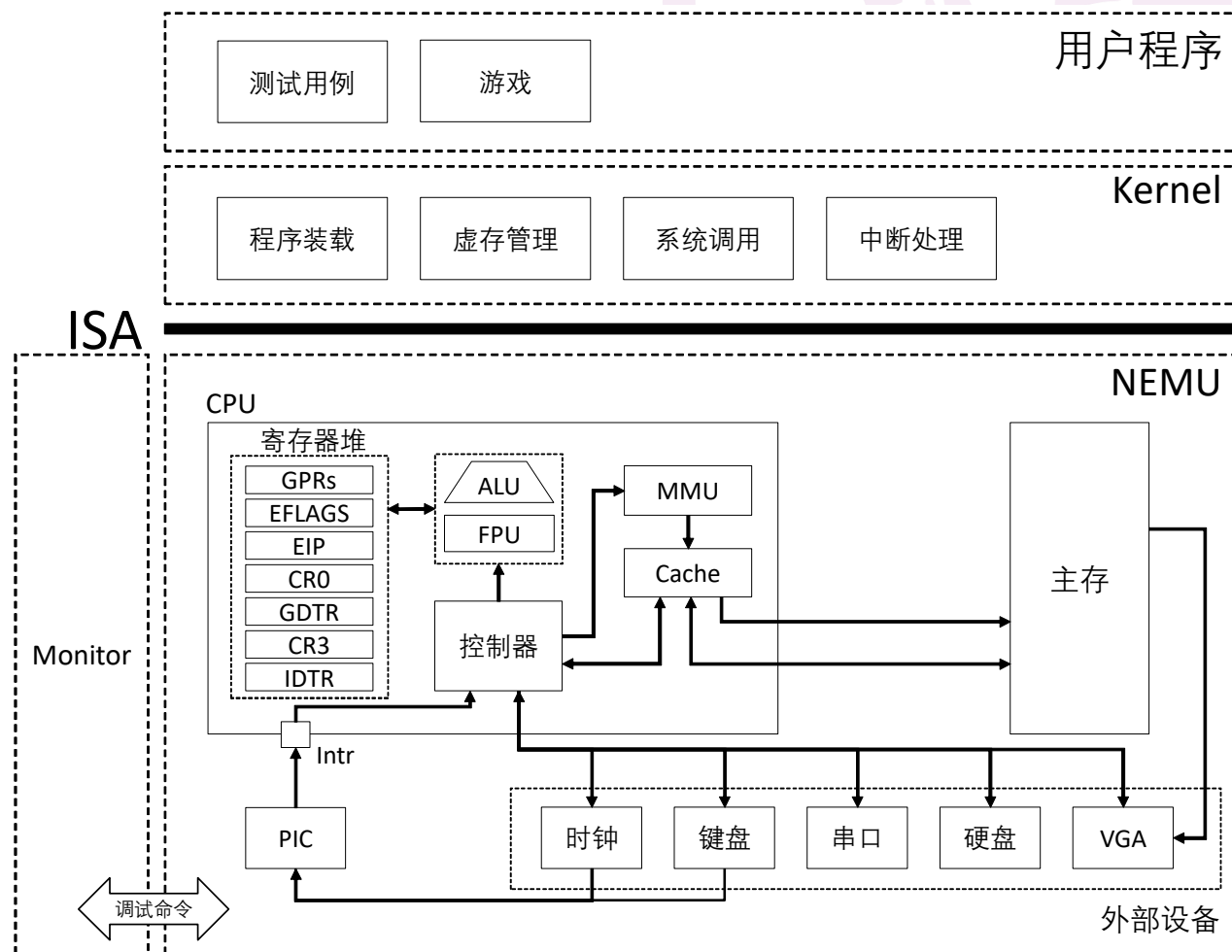
作为最后的挑战任务，以理解框架代码和debug为主。加油了

建议关调cache

执行命令：make test_pa-4-3 // 最后submit执行的是credits小游戏

最终的截止时间会另行公布

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！

感谢同学们一个学期的努力 :-)