操作系统 Operating Systems

L2. 揭开钢琴的盖子

Open the OS!

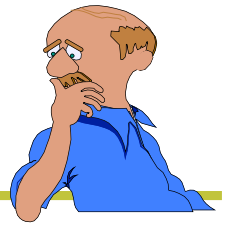
lizhijun_os@hit.edu.cn
授课教师：李治军 综合楼404室

从打开电源开始…

这神秘的黑色背后发生着什么?...

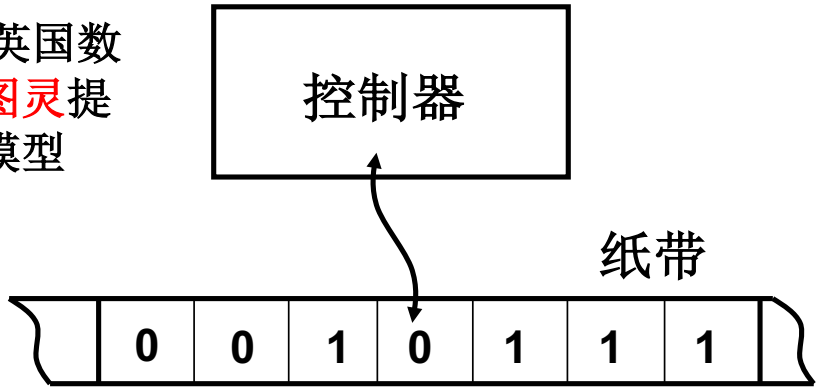
- 不要总等着别人告诉你答案，尽量自己去寻找…
  - 从知识和常识出发进行思索…
  - 打开电源→计算机要开始工作了
  - 计算机怎么工作? 这是我们最最基本，也最最重要的常识…

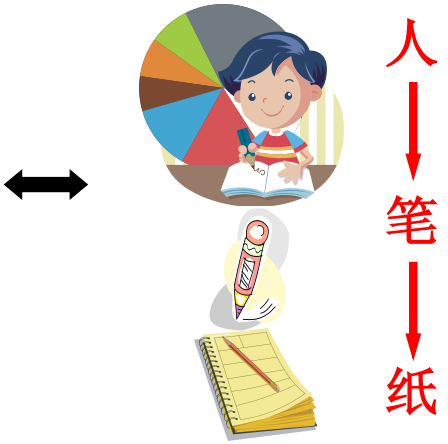
从白纸到图灵机



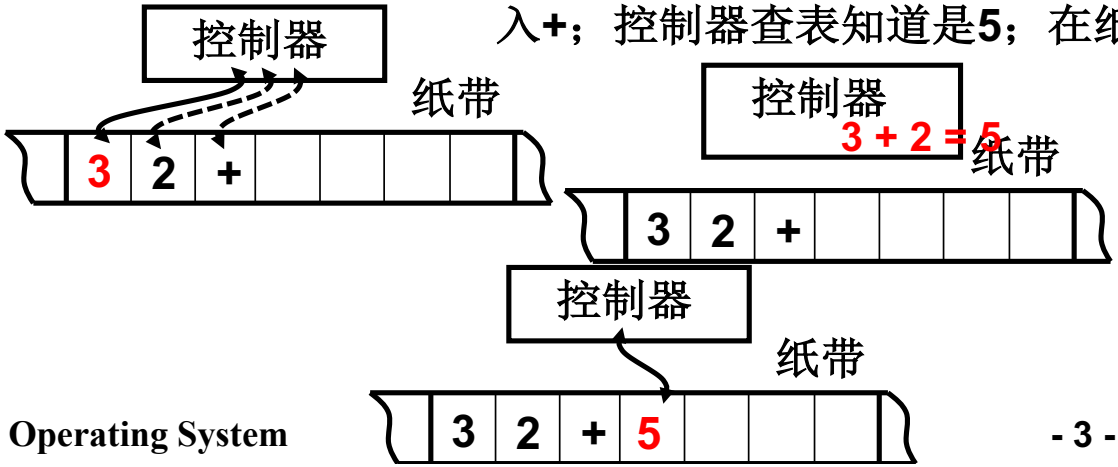
- 计算机怎么工作？说到底就是一个计算模型

1936年，英国数学家A.C.图灵提出了一种模型

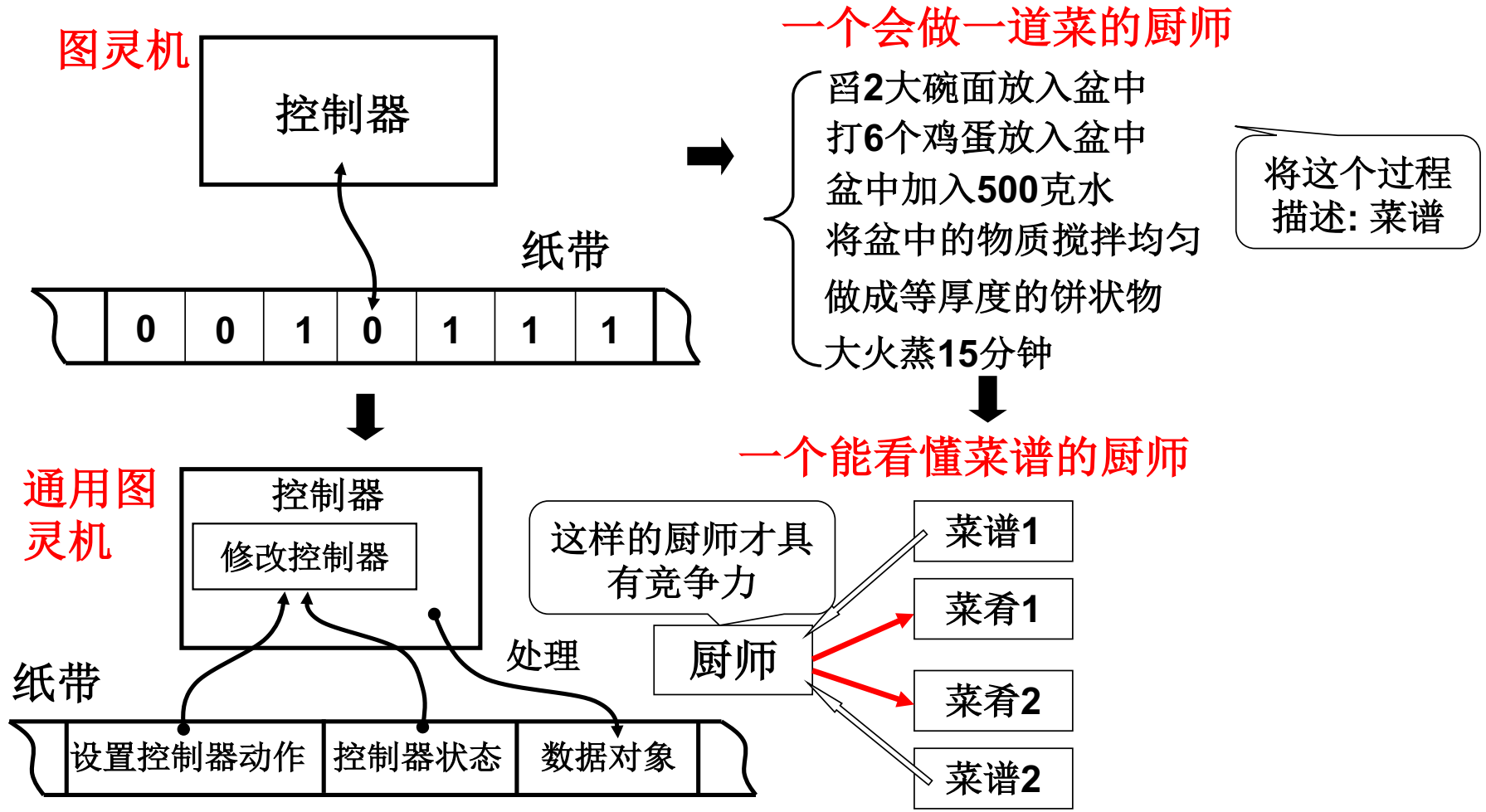




- 在纸带上读入3；在纸带上读入2；在纸带上读入+；控制器查表知道是5；在纸带上写下5



从图灵机到通用图灵机

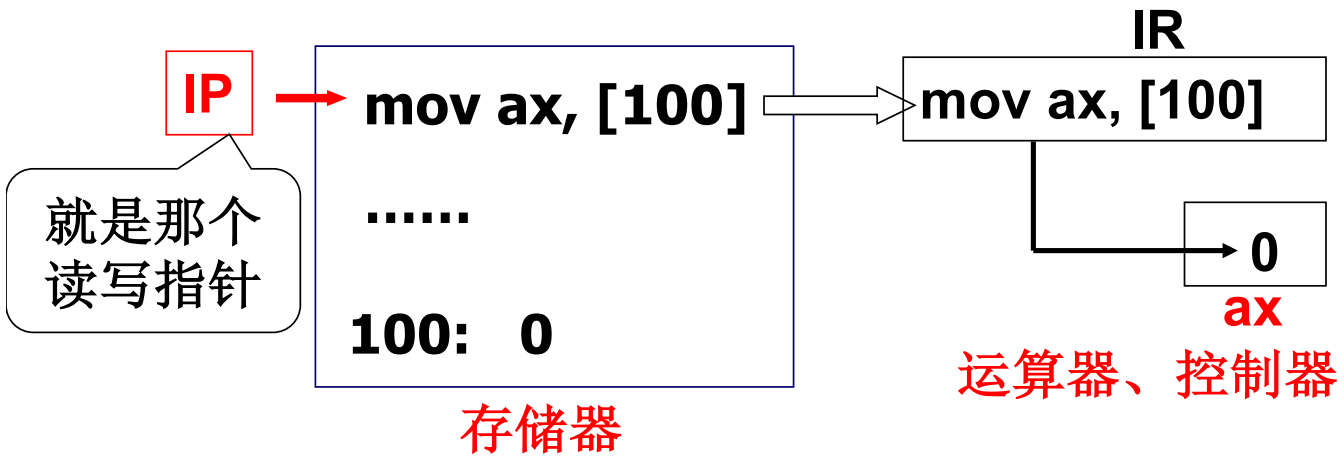

图灵机
控制器
纸带
0 0 1 0 1 1 1
一个会做一道菜的厨师
舀2大碗面放入盆中
打6个鸡蛋放入盆中
盆中加入500克水
将盆中的物质搅拌均匀
做成等厚度的饼状物
大火蒸15分钟
将这个过程描述: 菜谱
一个能看懂菜谱的厨师
通用图灵机
控制器
修改控制器
这样的厨师才具有竞争力
厨师
处理
菜谱1
菜肴1
菜肴2
菜谱2
纸带
设置控制器动作
控制器状态
数据对象

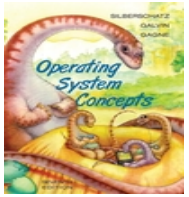

Operating System Concepts

从通用图灵机到计算机

- 伟大想法的工程实现…
  - 又一个伟大的发明: 冯·诺依曼存储程序思想
  - 存储程序的主要思想：将程序和数据存放到计算机内部的存储器中，计算机在程序的控制下一步一步进行处理
  - 计算机由五大部件组成:输入设备、输出设备、存储器、运算器、控制器

1946年提出





打开电源，计算机执行的第一句指令什么？

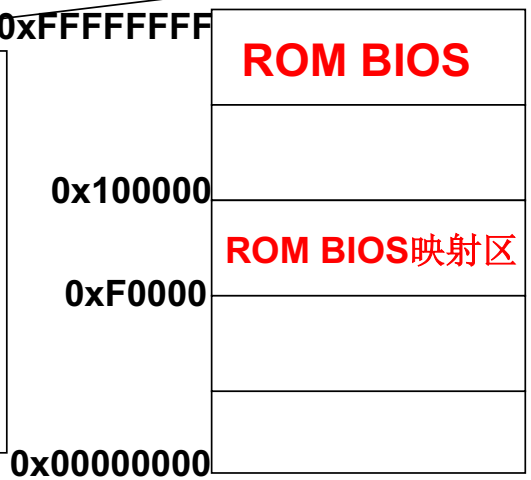
可以打开电源了…

- 计算模型⇒我们要关注指针IP及其指向的内容
  - 计算机刚打开电源时，IP=?
  - 由硬件设计者决定!

和保护模式对应，实模式的寻址CS:IP(CS左移4位+IP)，和保护模式不一样!

看看x86 PC

(1) x86 PC刚开机时CPU处于实模式
(2) 开机时，CS=0xFFFF; IP=0x0000
(3) 寻址0xFFFF0(ROM BIOS映射区)
(4) 检查RAM，键盘，显示器，软硬磁盘
(5) 将磁盘0磁道0扇区读入0x7c00处
(6) 设置cs=0x07c0，ip=0x0000



执行BIOS程序
将引导扇区读入内存
固定位置处，
并跳转到这个地址处执行
0磁道0扇区一共512个字节
读取到0x7c00地址处

0x7c00处存放的代码

- 就是从磁盘引导扇区读入的那512个字节
  - 引导扇区就是启动设备的第一个扇区
  - 启动设备信息被设置在CMOS中…
  - 因此，硬盘的第一个扇区上存放着开机后执行的第一段我们可以控制的程序。
  - 操作系统的故事从这里开始…

开机时按住del键可进入启动设备设置界面，可以设置为光盘启动!

CMOS: 互补金属氧化物半导体(64B-128B)。用来存储实时钟和硬件配置信息。

引导扇区代码：bootsect.s

利用汇编可以完全对应机器指令，
写的是放在哪个地址
就是哪个地址

```
.globl begtext,begdata,begbss,endtext,enddata,endbss
.text   //文本段
begtext:
.data   //数据段
begdata:
.bss    //未初始化数据段
begbss:
entry start //关键字entry告诉链接器“程序入口”
start:
    mov ax, #BOOTSEG     mov ds, ax
    mov ax, #INITSEG     mov es, ax
    mov cx, #256
    sub si, si           sub di,di
    rep    movw
    jmpi   go, INITSEG
```

.text等是伪操作符，告诉编译器产生文本段，.text用于标识文本段的开始位置。
此处的.text、.data、.bss表明这3个段重叠，不分段!

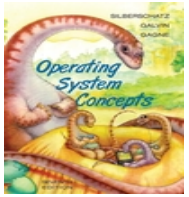
此条语句就是0x7c00处存放的语句!

```
BOOTSEG = 0x07c0
INITSEG = 0x9000
SETUPSEG = 0x9020
```

将ds:si处的256个字，也就是512个字节搬到es:di地址处。
也就是将引导扇区的代码搬了过去

将0x07c0:0x0000处的256个字移动到0x9000:0x0000处

ds变为7c0
es 变为9000
ds和si构成一个地址
es和di构成一个地址



jmpi 指令是，将go赋给ip，将INITSEG赋给CS，并跳转到对应的地址。这里go标号汇编结束存储的是距离开始处的偏移，所以这一调指令就相当于跳转到了go处的指令去

jmpi go, INITSEG

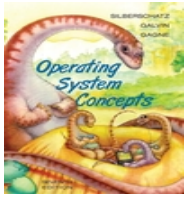
- jmpi (jump intersegment段间跳转): cs=INITSEG, ip=go

```
go: mov ax,cs //cs=0x9000
    mov ds,ax  mov es,ax  mov ss,ax  mov sp,#0xff00
load_setup:  //载入setup模块
    mov dx,#0x0000   mov cx,#0x0002  mov bx,#0x0200
    mov ax,#0x0200+SETUPLEN  int 0x13 //BIOS中断
    jnc ok_load_setup
    mov dx,#0x0000
    mov ax,#0x0000 //复位
    int 0x13
    j   load_setup //重读
```

为call做准备!

0x13是BIOS读磁盘扇区的中断: ah=0x02-读磁盘，al=扇区数量(SETUPLEN=4)，ch=柱面号，cl=开始扇区，dh=磁头号，dl=驱动器号，es:bx=内存地址

将启动扇区后面的四个扇区(setup)读到启动扇区上面紧邻的地址处。



读入setup模块后: ok_load_setup

表示地址之后的多少个字节是字符串

```
Ok_load_setup:   //载入setup模块
    mov dl,#0x00   mov ax,#0x0800  //ah=8获得磁盘参数
    int 0x13       mov ch,#0x00    mov sectors,cx
    mov ah,#0x03   xor bh,bh       int 0x10 //读光标
    mov cx,#24     mov bx,#0x0007
    mov bp,#msg1   mov ax,#1301    int 0x10 //显示字符
    mov ax,#SYSSEG //SYSSEG=0x1000
    mov es,ax
    call read_it //读入system模块
    jmpi 0,SETUPSEG
```

显示这24个字符将是大家的第一个“创举”！

7是显示属性!

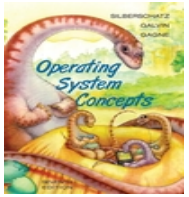
bp表示要显示的字符串的地址

将操作系统后面的代码读入

转入0x9020:0x0000
执行setup.s

```
bootsect.s中的数据 //在文件末尾
sectors: .word 0 //磁道扇区数
msg1:.byte 13,10
     .ascii “Loading system...”
     .byte 13,10,13,10
```

- boot工作:读setup，读system...



read_it //读入system模块

- 为什么读入system模块还需要定义一个函数?

system模块可能很大，要跨越磁道!

```
read_it:   mov ax,es     cmp ax,#ENDSEG    jb ok1_read
           ret
ok1_read:
   mov ax,sectors
   sub ax,sread //sread是当前磁道已读扇区数,ax未读扇区数
   call read_track //读磁道...
```

ENDSEG=SYSSEG+SYSSIZE
SYSSIZE=0x8000 //该变量可根据Image大小设定(编译操作系统时)

- 引导扇区的末尾 //BIOS用以识别引导扇区

否则会打出非引导设备

```
.org 510
   .word 0xAA55 //扇区的最后两个字节
```

- 可以转入setup执行了，jmpi 0, SETUPSEG

跳转到90200地址处，
也就是setup代码被放到的地址处