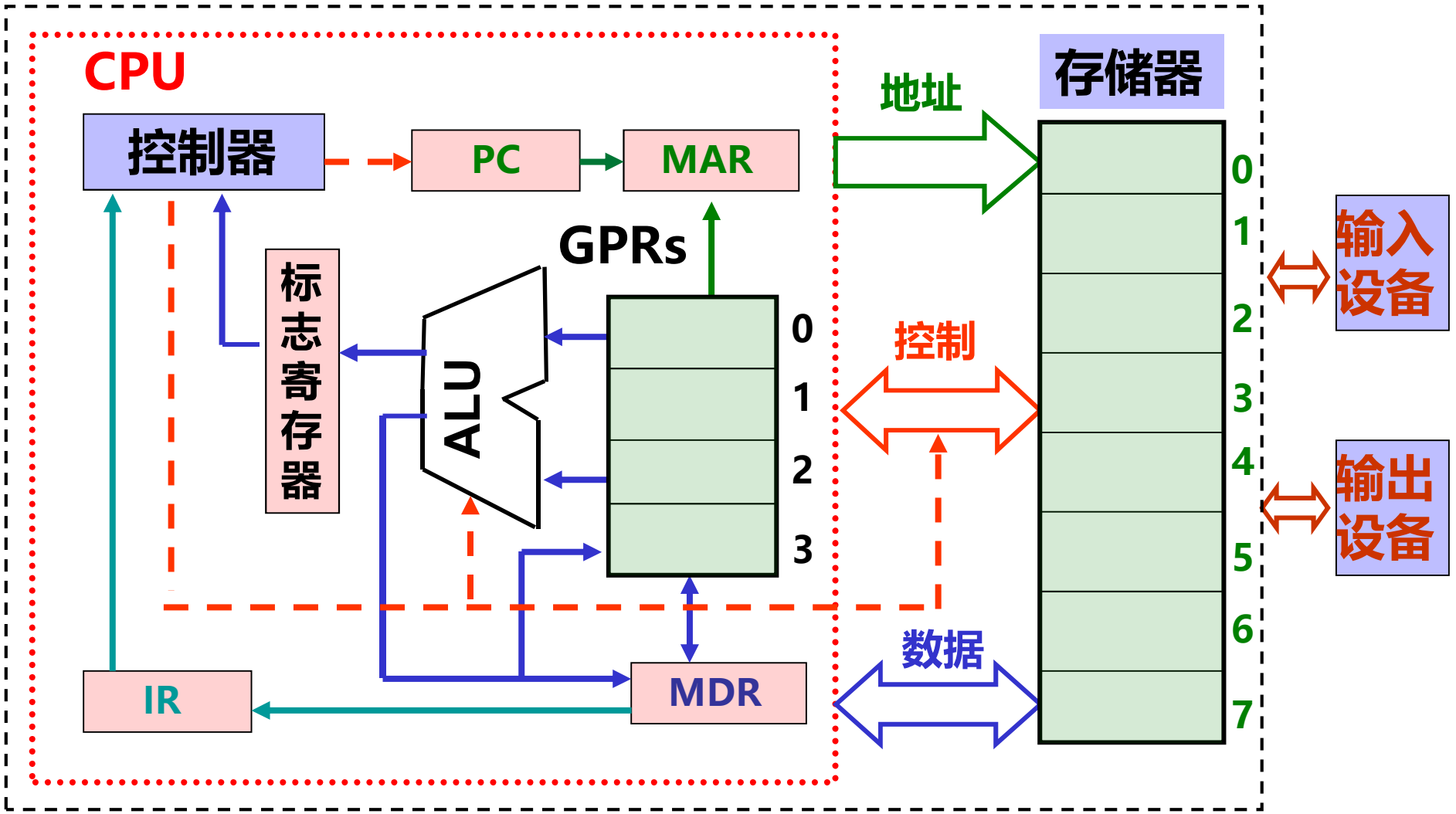
程序转换概述

南京大学
计算机科学与技术系
袁春风
email: cfyuan@nju.edu.cn
2015.6

回顾：计算机硬件基本组成

“存储程序”工作方式！



回顾：计算机是如何工作的？

程序由指令组成（菜单由菜谱组成）

- **程序在执行前**

  数据和指令事先存放在存储器中，每条指令和每个数据都有地址，指令按序存放，指令由OP、ADDR字段组成，程序起始地址置PC

  （原材料和菜谱都放在厨房外的架子上， 每个架子有编号。妈妈从第5个架上指定菜谱开始做）

- **开始执行程序**

  第一步：根据PC取指令（从5号架上取菜谱）

  第二步：指令译码（看菜谱）

  第三步：取操作数（从架上或盘中取原材料）

  第四步：指令执行（洗、切、炒等具体操作）

  第五步：回写结果（装盘或直接送桌）

  第六步：修改PC的值（算出下一菜谱所在架子号6=5+1）

  继续执行下一条指令（继续做下一道菜）

回顾：指令和数据

- 程序启动前，指令和数据都存放在存储器中，形式上没有差别，都是0/1序列
- 采用”存储程序“工作方式：
  - 程序由指令组成，程序被启动后，计算机能自动取出一条一条指令执行，在执行过程中无需人的干预。
- 指令执行过程中，指令和数据被从存储器取到CPU，存放在CPU内的寄存器中，指令在IR中，数据在GPR中。

指令中需给出的信息：

操作性质（操作码）

源操作数1 或/和 源操作数2 （立即数、寄存器编号、存储地址）

目的操作数地址 （寄存器编号、存储地址）

存储地址的描述与操作数的数据结构有关！

回顾：不同层次语言之间的等价转换


```
tmp = a[i];
a[i] = a[i+1];
a[i+1] = tmp;
```

```
lw $8, 0($2)
lw $9, 4($2)
sw $9, 0($2)
sw $8, 4($2)
```

每条指令由操作码和若干地址码组成

```
100011 00010 01000 0000000000000000
100011 00010 01001 0000000000000100
101011 00010 01001 0000000000000000
101011 00010 01000 0000000000000100
```

…, EXTop=1,ALUSelA=1,ALUSelB=11,ALUop=add,
IorD=1,Read,MemtoReg=1,RegWr=1, …

任何高级语言程序最终通过执行若干条指令来完成！

“指令”的概念

- 计算机中的指令有微指令、机器指令和伪（宏）指令之分
- 机器指令处于硬件和软件的交界面

  相当于一个菜谱指定的一个完整做菜过程

  – 本章中提及的指令都指机器指令
- 微指令是微程序级命令，属于硬件范畴

  相当于洗、切、煮、炒等做菜“微过程“
- 伪指令是由若干机器指令组成的指令序列，属于软件范畴

  相当于由多个菜谱合成一个”大菜“的过程
- 汇编指令是机器指令的汇编表示形式，即符号表示
- 机器指令和汇编指令一一对应，它们都与具体机器结构有关，都属于机器级指令

机器级指令

- 机器指令和汇编指令一一对应，都是机器级指令
- 机器指令是一个0/1序列，由若干字段组成

补码11111010
的真值为多少？

100010 DW	mod	reg	r/m	disp8
100010 0 0	01	001	001	11111010

操作码　寻址方式　寄存器编号　立即数(位移量)

- 汇编指令是机器指令的符号表示（可有不同格式）

长度后缀

mov [bx+di-6], cl　或　movb %cl, -6(%bx,%di)

Intel格式　AT&T 格式

指令的功能为：M[R[bx]+R[di]-6]←R[cl]

R：寄存器内容
M：存储单元内容

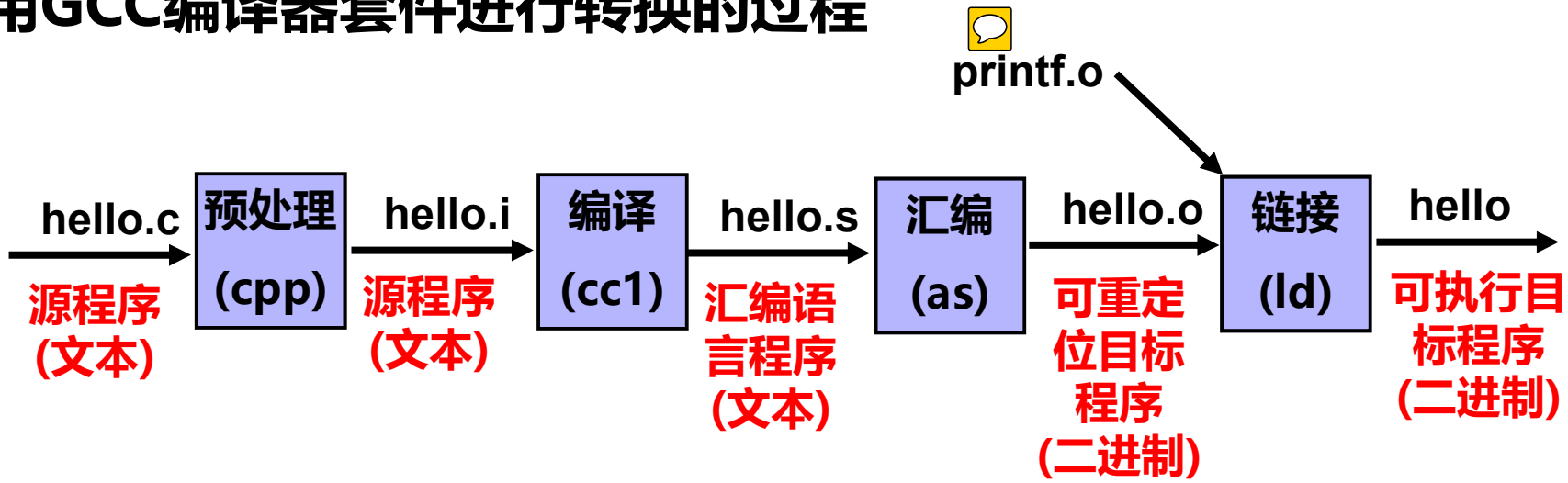
寄存器传送语言 RTL（Register Transfer Language）

mov、bx、movb、%bx等都是助记符

注：也可用(x)表示
地址x中的内容

回顾：高级语言程序转换为机器代码的过程

用GCC编译器套件进行转换的过程



预处理：在高级语言源程序中插入所有用#include命令指定的文件和用#define声明指定的宏。

编译：将预处理后的源程序文件编译生成相应的汇编语言程序。

汇编：由汇编程序将汇编语言源程序文件转换为可重定位的机器语言目标代码文件。

链接：由链接器将多个可重定位的机器语言目标文件以及库例程（如printf()库函数）链接起来，生成最终的可执行目标文件。

GCC使用举例

```
1    // test.c
2  int add(int i, int j )
3  {
4        int x = i +j;
5        return x;
6  }
```

- 两个源程序文件main.c和test.c，最终生成可执行文件为test

  gcc -O1 main.c test.c -o test

- 选项-O1表示一级优化，-O2为二级优化，选项-o指出输出文件名

- 目标文件可用“objdump -d test.o”反汇编为汇编语言程序

gcc -E test.c -o test.i
gcc -S test.i -o test.s

test.s ← gcc –S test.c –o test.s

```
add:
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
movl   12(%ebp), %eax
movl   8(%ebp), %edx
leal    (%edx, %eax), %eax
movl   %eax, -4(%ebp)
movl   -4(%ebp), %eax
leave
ret
```

gcc –c test.s –o test.o

```
00000000 <add>:
 0:  55           push   %ebp
 1:  89 e5        mov    %esp, %ebp
 3:  83 ec 10     sub    $0x10, %esp
 6:  8b 45 0c     mov    0xc(%ebp), %eax
 9:  8b 55 08     mov    0x8(%ebp), %edx
 c:  8d 04 02     lea    (%edx,%eax,1), %eax
 f:  89 45 fc     mov    %eax, -0x4(%ebp)
12:  8b 45 fc     mov    -0x4(%ebp), %eax
15:  c9           leave
16:  c3           ret
```

位移量　机器指令　汇编指令

编译得到的与反汇编得到的汇编指令形式稍有差异

两种目标文件

```
1    // test.c
2  int add(int i, int j )
3  {
4        int x = i +j;
5        return x;
6  }
```

test.o：可重定位目标文件

test：可执行目标文件

“objdump -d test.o” 结果

00000000 <add>:

0:	55	push	%ebp
1:	89 e5	mov	%esp, %ebp
3:	83 ec 10	sub	$0x10, %esp
6:	8b 45 0c	mov	0xc(%ebp), %eax
9:	8b 55 08	mov	0x8(%ebp), %edx
c:	8d 04 02	lea	(%edx,%eax,1), %eax
f:	89 45 fc	mov	%eax, -0x4(%ebp)
12:	8b 45 fc	mov	-0x4(%ebp), %eax
15:	c9	leave	
16:	c3	ret	

“objdump -d test” 结果

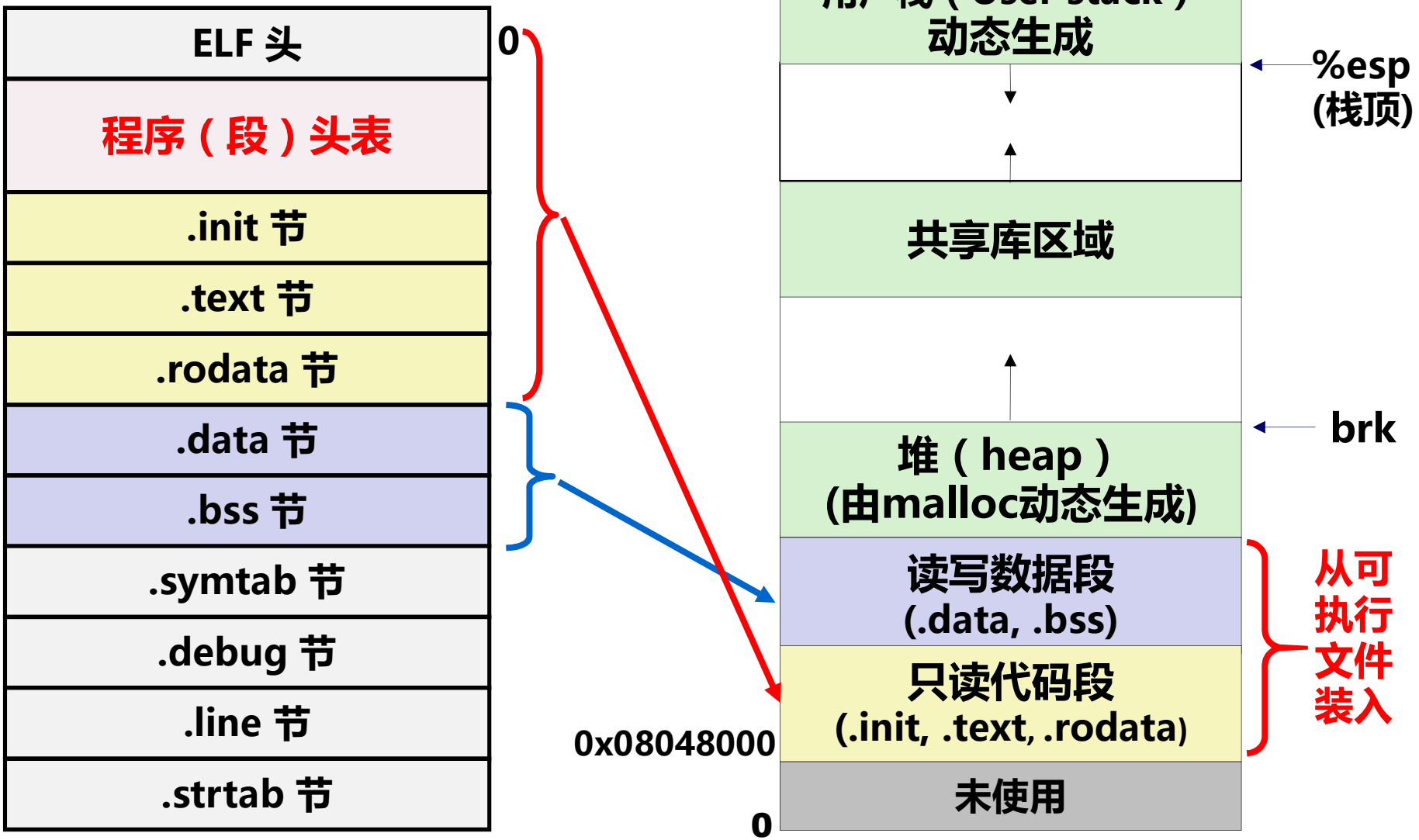
080483d4 <add>:

80483d4:	55	push ...
80483d5:	89 e5	…
80483d7:	83 ec 10	…
80483da:	8b 45 0c	…
80483dd:	8b 55 08	…
80483e0:	8d 04 02	…
80483e3:	89 45 fc	…
80483e6:	8b 45 fc	…
80483e9:	c9	…
80483ea:	c3	ret

test.o中的代码从地址0开始，test中的代码从80483d4开始！

可执行文件的存储器映像

程序(段)头表描述如何映射



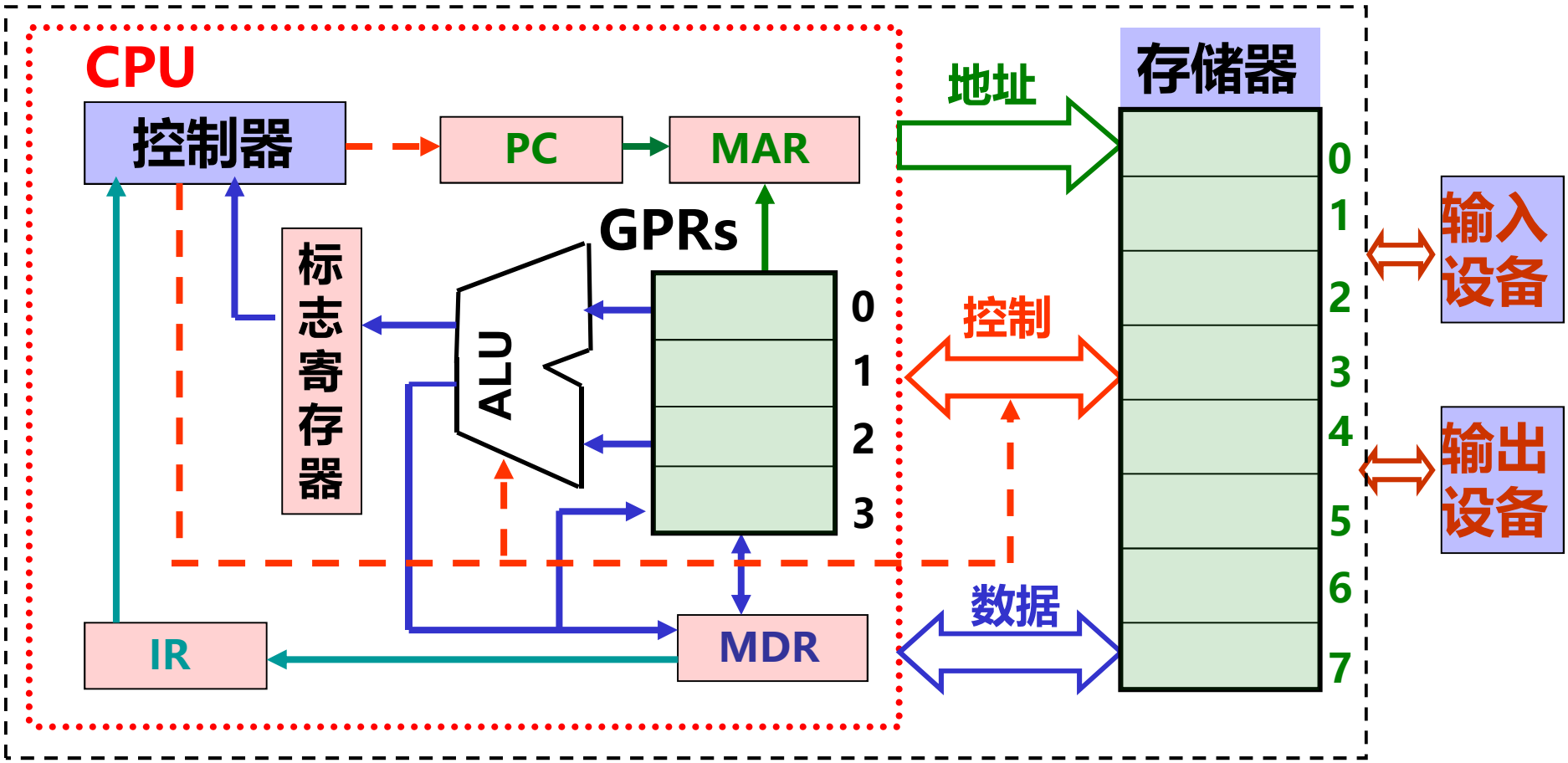
回顾：指令集体系结构（ISA）

- ISA指Instruction Set Architecture，即指令集体系结构
- ISA是一种规约（Specification），它规定了如何使用硬件
  - 可执行的指令的集合，包括指令格式、操作种类以及每种操作对应的操作数的相应规定；
  - 指令可以接受的操作数的类型；
  - 操作数所能存放的寄存器组的结构，包括每个寄存器的名称、编号、长度和用途；
  - 操作数所能存放的存储空间的大小和编址方式；
  - 操作数在存储空间存放时按照大端还是小端方式存放；
  - 指令获取操作数的方式，即寻址方式；
  - 指令执行过程的控制方式，包括程序计数器、条件码定义等。
- ISA在计算机系统中是必不可少的一个抽象层，Why？
  - 没有它，软件无法使用计算机硬件！
  - 没有它，一台计算机不能称为“通用计算机”

微体系结构

ISA和计算机组成（Organization，即MicroArchitecture）是何关系？

ISA和计算机组成（微结构）之间的关系



ISA是计算机组成的抽象

不同ISA规定的指令集不同，如，IA-32、MIPS、ARM等

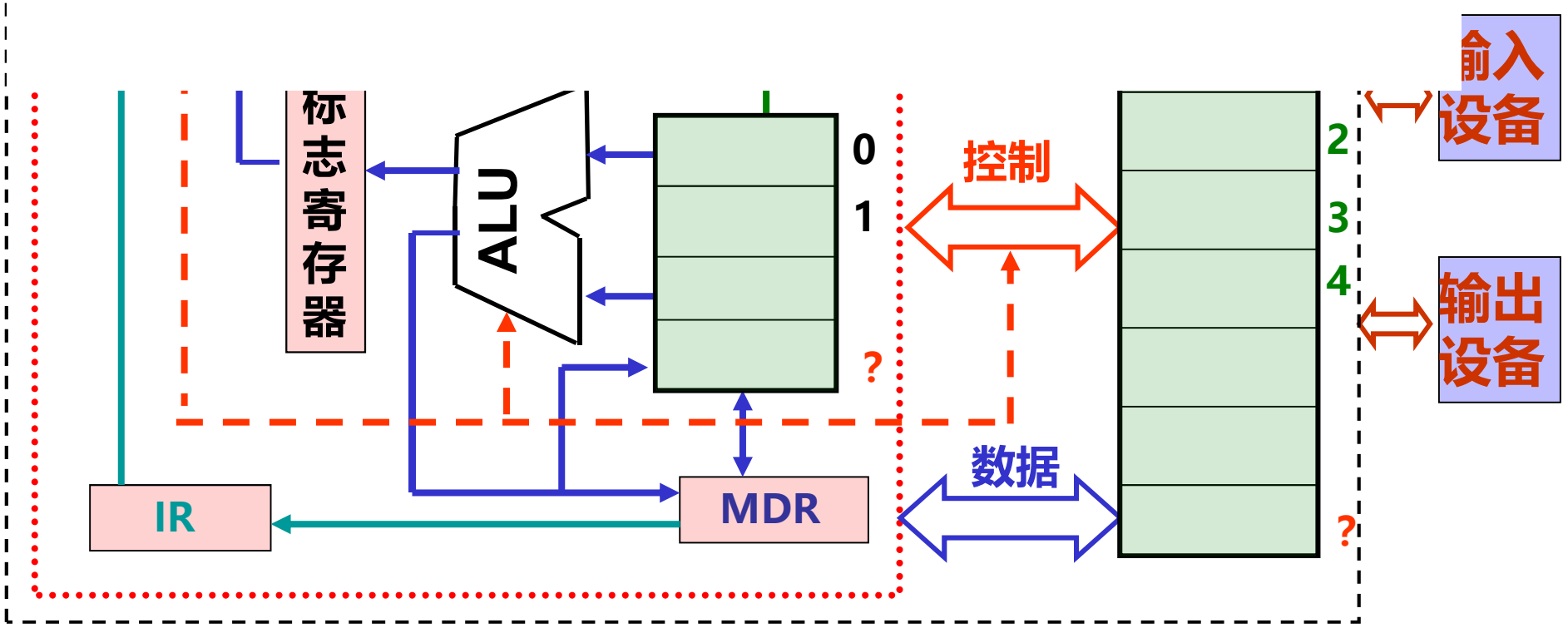
计算机组成必须能够实现ISA规定的功能，如提供GPR、标志、运算电路等

同一种ISA可以有不同的计算机组成，如乘法指令可用ALU或乘法器实现

IA-32的体系结构是怎样的呢？

寄存器个数及各自功能？寄存器宽度？存储空间大小？编址单位？
指令格式？指令条数？指令操作功能？寻址方式？数据类型？
小端/大端？标志寄存器各位含义？PC位数？I/O端口编址方式？……

下一节课开始介绍 IA-32 的指令集体系结构（ISA）



总结

- 高级语言程序总是转换为机器代码才能在机器上执行
- 转换过程：预处理、编译、汇编、链接
- 机器代码是二进制代码，可DUMP为汇编代码表示
- ISA规定了一台机器的指令系统涉及到的所有方面

  例如：
  - 所有指令的指令格式、功能
  - 通用寄存器的个数、位数、编号和功能
  - 存储地址空间大小、编址方式、大/小端
  - 指令寻址方式