



# 计算机系统 计算机简化模型

湖南大学

《计算机系统》课程教学组

# 本讲学习目标

描述计算机基本结构



认识简化模型

模拟原型系统机器指令执行

### CPU



0000	00000000
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

## 简化模型

CPU



内存

0000	00000000
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

代码

```
int i=1;
```

```
int j=2;
```

```
int k;
```

```
k=i+j;
```

## 简化模型

### 内存

### 代码



0000	00000001
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

**int i=1; ← PC**

**int j=2;**

**int k;**

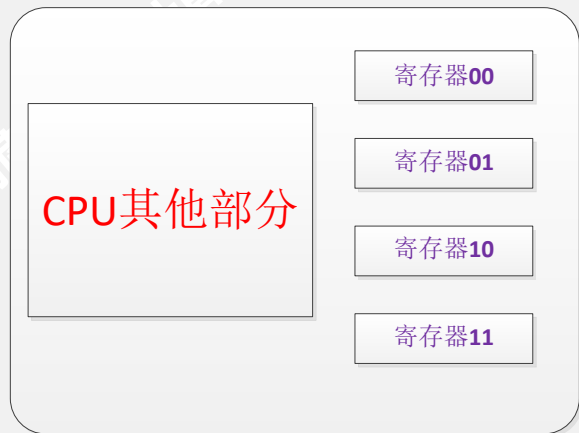
**k=i+j;**

**CPU执行命令“ int i=1” , 在内存中给变量 i 分配一个空间, 并赋值1.**

## 简化模型

### 内存

### 代码



0000	00000001
0001	00000010
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

**int i=1;**

**int j=2; ←**

**int k;**

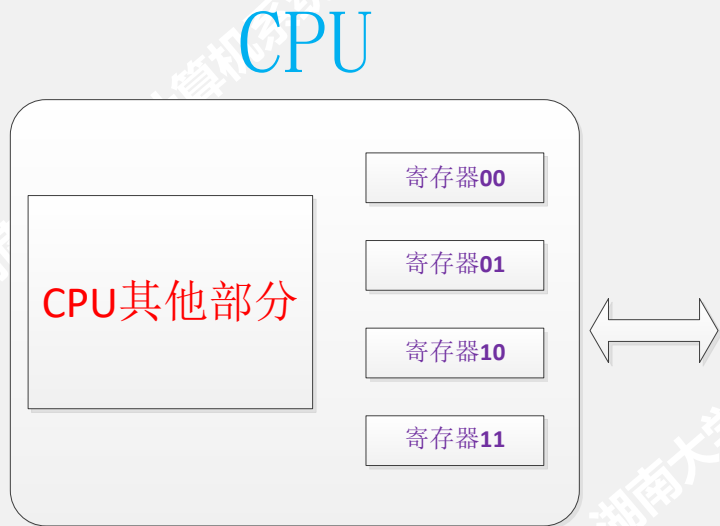
**k=i+j;**

CPU执行命令“ **int j=2** ”，在内存中给变量j分配一个空间，并赋值2.

## 简化模型

### 内存

### 代码



0000	00000001
0001	00000010
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

**int i=1;**

**int j=2;**

**int k; ←**

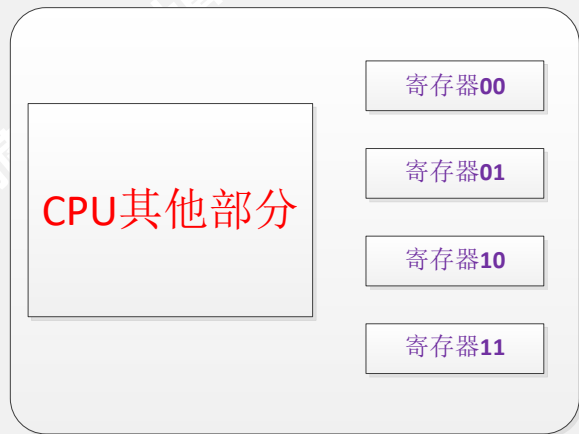
**k=i+j;**

**CPU执行命令“ int k” , 在内存中给变量k分配一个空间, 不赋值.**

## 简化模型

### 内存

### 代码



0000	00000001
0001	00000010
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

**int i=1;**

**int j=2;**

**int k;**

**k=i+j; ←**

**CPU执行命令“ k=i+j” ， 需要分四步走。**



## 简化模型

### 内存

### 代码



0000	00000001
0001	00000010
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

int i=1;

int j=2;

int k;

k=i+j; ←

CPU执行命令“ $k=i+j$ ”，需要分四步走。

第一步，将i值从内存拷至寄存器00中

## 简化模型

### 内存

### 代码

### CPU



0000	00000001
0001	00000010
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

**int i=1;**

**int j=2;**

**int k;**

**k=i+j; ←**

**CPU执行命令“k=i+j”，需要分四步走。**

**第二步，将j值从内存拷至寄存器01中**

## 简化模型

### 内存

### 代码

### CPU



0000	00000001
0001	00000010
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

int i=1;

int j=2;

int k;

k=i+j; ←

CPU执行命令“k=i+j”，需要分四步走。

第三步，计算寄存器00与01中数值的和，结果放于00中

## 简化模型

### 内存

### 代码



0000	00000001
0001	00000010
0010	00000011
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

int i=1;

int j=2;

int k;

k=i+j; ←

CPU执行命令“ $k=i+j$ ”，需要分四步走。

第四步，将寄存器00中的结果传送到内存变量k处

程序在机器中执行时，其数据在执行过程中的路径是：

A

内存→寄存器→内存

B

内存→寄存器→CPU→寄存器→内存

C

内存→CPU→内存

D

寄存器→CPU→内存→寄存器

提交

## 为什么不直接在内存计算？

如果内存距离CPU稍远一点（**超过5cm**），就会导致数据滞后，而寄存器就在CPU内部，基本上是零距离。



**3.0GHz**的CPU，大部分简单指令的执行只需要一个时钟周期，也就是约**1/3纳秒**。在这个时间里，电只能前进**10厘米**。

CPU的运行频率大大超过内存的运行频率（**GHz vs. MHz**）

# 存算体系

从CPU**高速缓存**中读取信息就好像是拿起桌上的一张草稿纸 (**3秒**) 或从身边的书架上取出一本书 (**约14秒**)

而从**内存**中读取信息则相当于走到办公楼下去买个零食 (**约4分钟**)

02

04

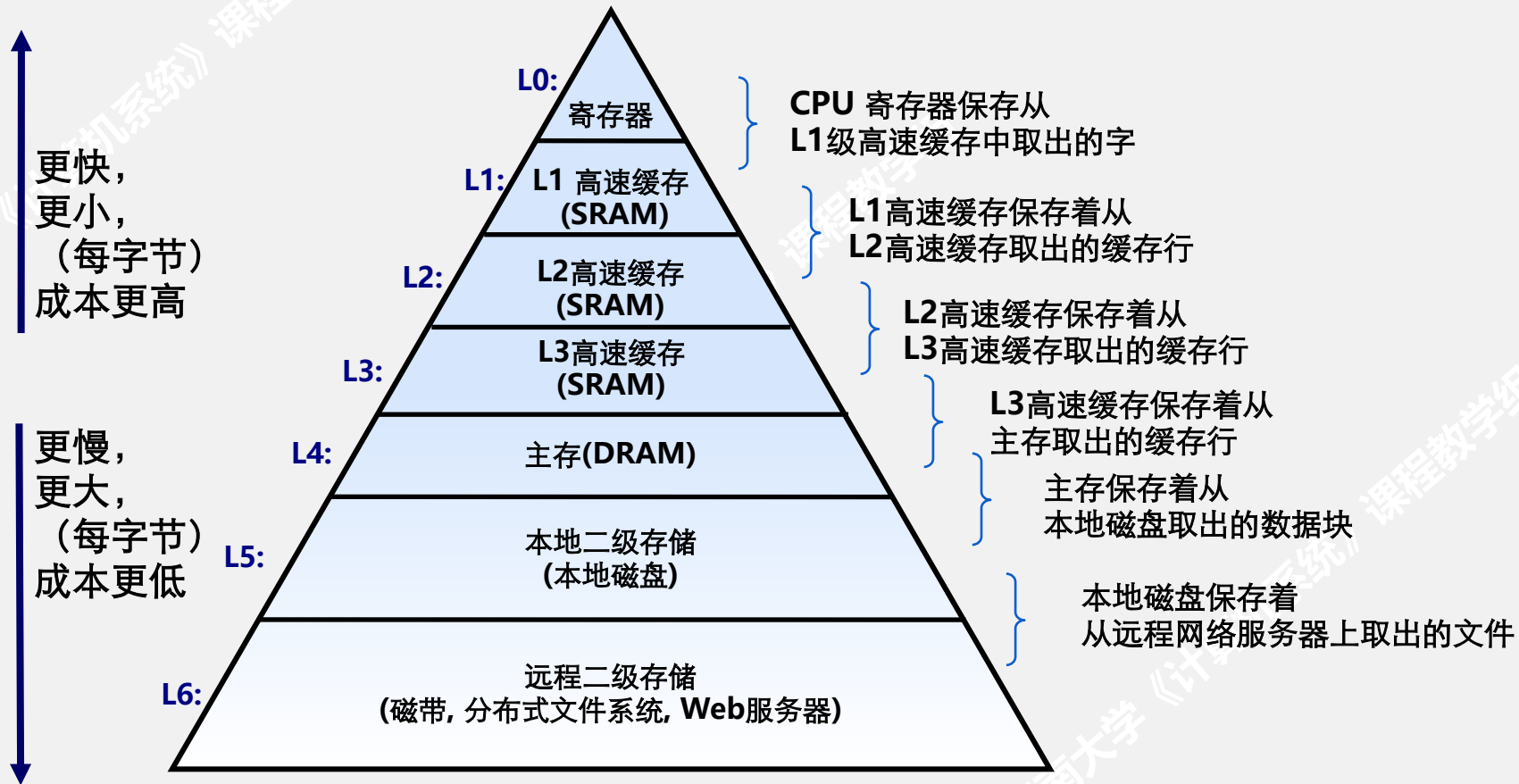
01

那么, 从**寄存器**中取数据**小于1秒**  
(但寄存器非常昂贵)

从**硬盘**中取得一个数据时间相当于离开家并开始长达**一年零三个月**的环球旅行

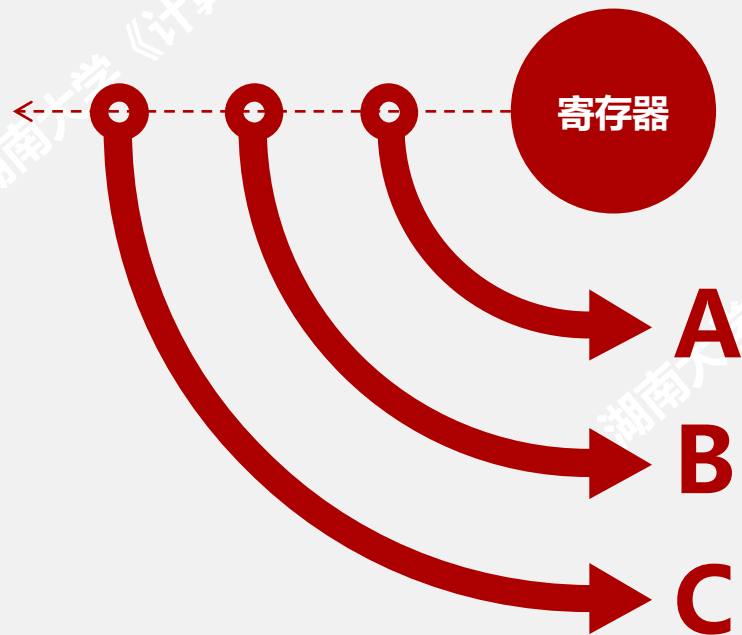
如果把CPU的一个时钟周期看作一秒

# 存储器层次结构





# 寄存器



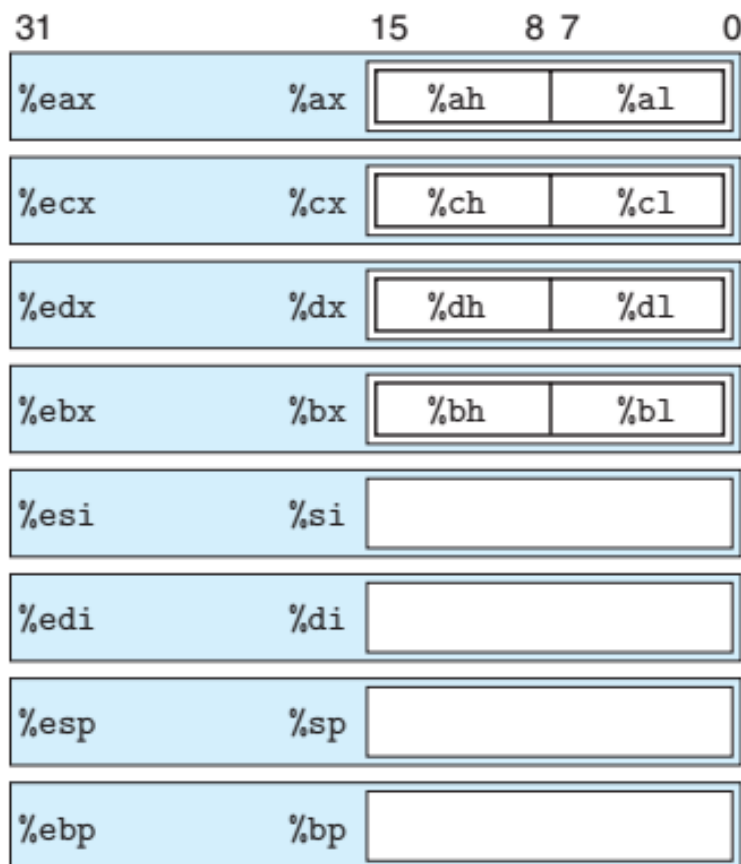
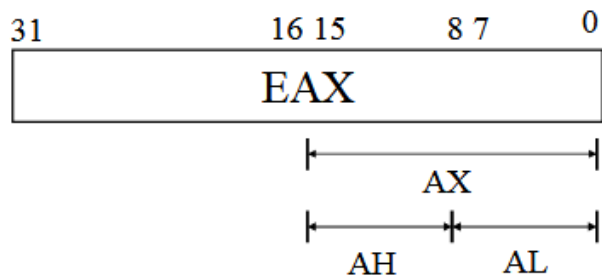
Assembly Name	Reg #
EAX	000
EBX	001
ECX	010
EDX	011
ESP	100
EBP	101
ESI	110
EDI	111

(X86) 32位CPU中包含一组8个32bit的通用寄存器

寄存器用以存储**整型数据**和**指针**

现在的CPU中寄存器数目可能过百

## 寄存器



下列寄存器单元存储的字节数依次是：  
%EAX, %DL, %BX, %CH, %ESP

A 4, 2, 1, 2, 4

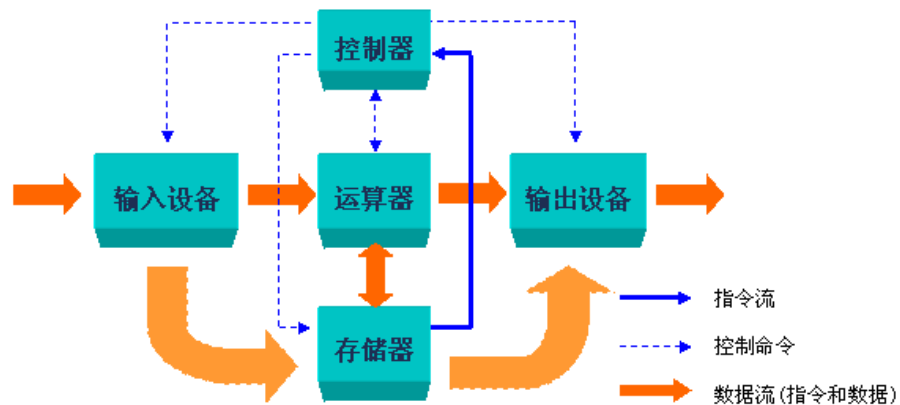
B 4, 1, 2, 1, 4

C 2, 1, 4, 1, 2

D 2, 4, 1, 4, 2

提交

## 冯诺依曼结构



从最初的EDVAC到当前最先进的计算机都采用的是冯·诺依曼体系结构

计算机硬件由运算器、控制器、存储器、输入设备和输出设备五大部分组成

其它体系结构:

- 哈佛体系结构
- 忆阻体结构

# 计算机原型系统

## CPU



输入

## 内存

0000	00000000
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000
1000	00000000
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000

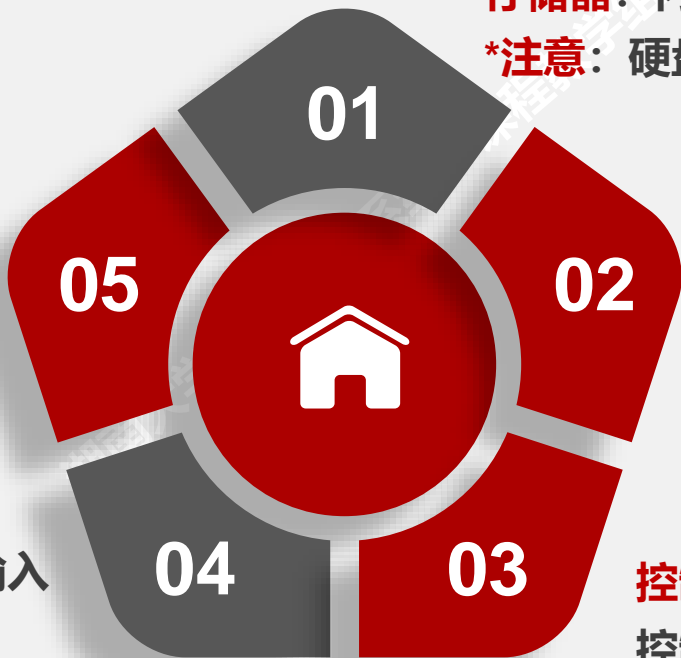
输出



# 原型系统

**输出设备：**一个四位的数码显示管，其数据来自内存1111地址的数据（显存）

**输入设备：**通过数字小键盘输入一个数字，存储到R0寄存器



**存储器：**内存

**\*注意：**硬盘、软驱、光驱都属于外部设备！

**运算器：**在CPU内部，用于计算，假设本系统的运算器能做加法、减法、乘法

**控制器：**通过向其他设备发出控制信号来控制整个机器运行

控制器：运行**指令**，支持的指令有：

- **输入**：等待数字小键盘输入1-127的数字，存储到**R0**寄存器
- **做加法 Ra, Rb**：将**Rb**和**Ra**两个寄存器的值相加，结果放在**Rb**中。
- **做减法 Ra, Rb**：将**Rb**减去**Ra**中的值，结果放在**Rb**中（**这两个寄存器都不能是R3**），并且根据结果来设置**R3**的值
- **寄存器直接赋值 v, Ra**：将寄存器**Ra**的值设置为**v**，**v**为常数
- **数据拷贝A,B**：将数据从**A**处拷到**B**处，其中**A**，**B**为**寄存器名称或内存地址**
- **判断跳转 v**：如果**R3**的值等于**1**，那么就跳转到当前指令地址+**v**处的指令去执行，如果不等于1，则执行下一条指令
- **直接跳转 v**：跳转到当前指令地址+**v**处的指令去执行
- **停机**：原型机停止运行指令

## 课程组设计实现的原型机，指令集格式：

指令格式	例子	说明
0	0	停机指令，原型机停止运行
1	1	输入一个整数，这个整数必须大于等于0小于127，输入后，此数值保存在R0 $\text{Input} \Rightarrow \text{R0}$
2 Ra Rb	2 R0 R1	加法指令Ra, Rb, $\text{Rb} + \text{Ra} \Rightarrow \text{Rb}$
3 Ra Rb	3 R2, R1	减法指令Ra, Rb, $\text{Rb} - \text{Ra} \Rightarrow \text{Rb}$ 这两个寄存器不能为R3(执行减法时，R3为专用标志寄存器) 当结果大于0时，R3中赋值为1，当结果小于0时，R3中赋值为-1，当结果等于0时，R3中赋值为0
4 v Ra	4 10 R1	寄存器直接赋值指令。其中Ra为寄存器名称，v为常数，意义是将常数值v直接放到寄存器Ra中 $\text{v} \Rightarrow \text{Ra}$



## 原型系统指令集

### 课程组设计实现的原型机，指令集格式：

5 A B	5 R0 R1 5 R0 0000	传送指令，其中A，B为寄存器编号或内存地址意义是将A处的值传送至B处 $A \Rightarrow B$
6 bias	6 -2 6 3	判断跳转指令。其中 bias为一个整数（可以为负），意义是如果R3的值为1，则跳转当前指令+bias条指令处执行，否则执行下一条指令
7 bias	7 2 7 -3	直接跳转指令。其中bias为一个整数（可以为负），直接跳转当前指令+bias条指令处执行
8 Ra	8 R0	输出指令。将Ra的值发送至显存，并输出 $Ra \Rightarrow \text{Output}$

## 示例 1

编程任务，输入一个大于1的数字a，  
计算 $1+2+\dots+a$ 的值并显示出来。

```
void main()  
{  
    int a;  
    int sum;  
    int i;  
    sum=0;  
    scanf("%d",&a);  
    for(i=a;i>=1;i++)  
    {  
        sum=sum+i;  
    }  
    printf("%d\n",sum);  
}
```

## 机器级表达

编译转换成原型系统能够运行的指令集合，在运行  
为变量a分配内存地址0000，为变量sum分配内存  
地址0001，为变量i分配内存地址0010。所有内存  
地址及寄存器初始值为0

1. 输入
2. 数据拷贝 R0, 0000
3. 寄存器赋值 \$1, R2
4. loop: 做加法 R0, R1
5. 做减法 R2, R0
6. 判断跳转 loop
7. 数据拷贝 R1, 0001
8. 输出 R1
9. 停机

## C语言

程序员编写，原型系统无法执行

```
#include <stdio>

void main()
{
    int a;
    int sum;
    int i;
    sum=0;
    scanf("%d",&a);
    for(i=a;i>=1;i--)
        sum=sum+i;
    printf("%d",sum);
}
```

## 示例1 指令存储

CPU



输入



内存

0000	00000000
0001	00000000
0010	00000000
0011	输入a: 存入R0
0100	传送: R0 → 0000
0101	寄存器赋值: R2=1
0110	加法: R0+R1 → R1
0111	减法: R0-R2 → R0 设R3
1000	判断跳转: 0110
1001	传送: R1 → 0001
1010	输出: R1 → 1111
1011	停机
1100	00000000
1101	00000000
1110	00000000
1111	00000000

数据段  
3字节

代码段  
12字节

显存1字节



## 程序1及原型系统机器指令

### 程序1:

```
#include <stdio>

void main()
{
    int a;
    int sum;
    int i;
    sum=0;
    scanf("%d",&a);
    for(i=a;i>=1;i--)
        sum=sum+i;
    printf("%d",sum);
}
```

编译转换成原型系统能够运行的指令集合，在运行  
为变量a分配内存地址0000，为变量sum分配内存  
地址0001，为变量i分配内存地址0010。所有内存  
地址及寄存器初始值为0

- |                    |              |
|--------------------|--------------|
| 1. 输入              | 1. 1         |
| 2. 数据拷贝 R0, 0000   | 2. 5 R0 0000 |
| 3. 寄存器赋值 \$1,R2    | 3. 4 1 R2    |
| 4. 做加法 R0,R1(loop) | 4. 2 R0 R1   |
| 5. 做减法 R2,R0       | 5. 3 R2 R0   |
| 6. 判断跳转 loop       | 6. 6 -2      |
| 7. 数据拷贝 R1,0001    | 7. 5 R1 0001 |
| 8. 输出 R1           | 8. 8 R1      |
| 9. 停机              | 9. 0         |

## 示例 2

编程任务，输入两个数，保存这两个数，并找出其中的最大值

```
void main()  
{  
    int a,b;  
    int max=0;  
    scanf("%d,%d",&a,&b);  
    max=a;  
    if(b>a)  
        max=b;  
    printf("%d",max);  
}
```

## 程序2及原型系统机器指令

### 程序2:

变量分配内存地址:

a:0000,b:0001,max:0010

```
void main()
{
    int a,b;
    int max=0;
    scanf("%d,%d",&a,&b);
    max=a;
    if(b>a)
        max=b;
    printf("%d",max);
}
```

1. 等待输入; //输入数字a, 并保存在R0中
2. 数据拷贝R0,0000; //a的值保存在内存地址0000中
3. 数据拷贝R0,R1; //a的值保存在寄存器R1中
4. 等待输入; //输入数字b, 并保存在R0中
5. 数据拷贝R0,0001; //b的值保存在内存地址0001中
6. 做减法R1,R0; //做减法, 结果存入R0, 设置R3
7. 判断跳转 L1; //若R0>R1(b>a),跳转到L1,max=b
8. 数据拷贝0000,0010; //若R0<R1, (b<a), max=a
9. 直接跳转L2
10. L1:数据拷贝0001,0010; //输出最大值
11. L2: 数据拷贝0010,1111; //max送至显存输出
12. 停机 //程序执行完成

Copyright © 2012 Pearson Education, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or by any information storage or retrieval system, without prior written permission from Pearson Education, Inc.



## 程序2及原型系统机器指令

### 程序2:

变量分配内存地址:


a:0000,b:0001,max:0010


```
void main()
{
    int a,b;
    int max=0;
    scanf("%d,%d",&a,&b);
    max=a;
    if(b>a)
        max=b;
    printf("%d",max);
}
```

- |                        |                 |
|------------------------|-----------------|
| 1. 等待输入;               | 1. 1            |
| 2. 数据拷贝R0,0000         | 2. 5 R0 0000    |
| 3. 数据拷贝R0,R1;          | 3. 5 R0 R1      |
| 4. 等待输入;               | 4. 1            |
| 5. 数据拷贝R0,0001;        | 5. 5 R0 0001    |
| 6. 做减法R1,R0;           | 6. 3 R1 R0      |
| 7. 判断跳转 L1;            | 7. 6 3          |
| 8. 数据拷贝0000, 0010;     | 8. 5 0000 0010  |
| 9. 直接跳转L2              | 9. 7 2          |
| 10. L1: 数据拷贝0001,0010; | 10. 5 0001 0010 |
| 11. L2: 数据拷贝0010,1111; | 11. 5 0010 R0   |
| 12. 停机                 | 12. 8 R0        |
|                        | 13. 0           |

# 课程组开发的 原型机 hnuvm-0.1

## 将贯穿始终!

泛雅

课程门户

计算机系统

班级活动

课件

教案

章节

资料

通知

讨论

作业

考试

题库

知识点

统计

编辑章节

目录

1 计算机系统漫游

1 1.1 课程介绍

1.2 引论

2 1.2.1 《计算机系统》概论

2 1.2.2 计算机简化模型

4 1.2.3 一个原型系统

1.3 Hello World

3 1.3.1 基础知识

2 1.3.2 程序的存储与编译

1 1.3.3 程序的运行过程

1.4 本章小结


1.5 讨论一：数与字符的编码 & 存储格式

章节详情

一个原型系统

课程视频

文件资源



从最初的ENIAC到当前最先进的计算机都采用的是冯·诺依曼体系结构

计算机硬件由运算器、控制器、寄存器、输入设备和输出设备五大部分组成


其它体系结构：  
· 哈佛体系结构  
· 细胞体系结构

原型系统

CPU 内存

hnuvm-0.1.zip  
9.85 KB

任务点



# 下一节：Hello World!

湖南大学  
《计算机系统》课程教学组

