



中山大學
SUN YAT-SEN UNIVERSITY



计算机组成原理

第二章：指令：计算机的语言

中山大学计算机学院
陈刚

2022年秋季

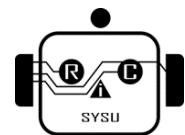
本讲内容

- 什么是计算机语言？

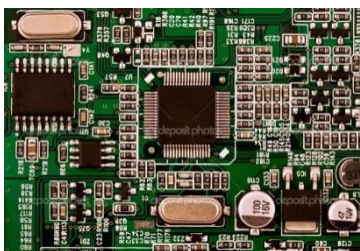
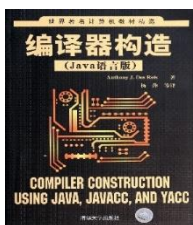
 - 指令集概述 Introduction to Instruction Set

- 指令格式

- 寻址方式



指令系统概述



25482+426858=????????????

```
a=25482;  
b=426858;  
c=a+b;
```

```
STORE a 25482;  
STORE b 25482;  
ADD c,a,b;
```

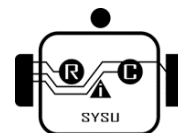
```
010100101001  
011100101010  
100110000010  
....
```

自然语言

高级程序语言

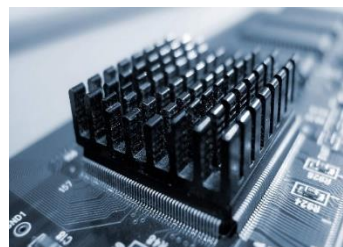
汇编语言

机器语言



指令系统概述

010100101001
011100101010
100110000010
....



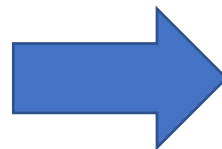
机器“语言”
二进制：0和1
硬件：逻辑电路



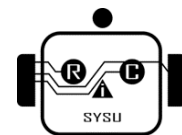
“机器指令”

计算机设计者赋予计算机实现某
种基本操作的命令

指令：0101110010101010
基本操作：load r1 1382



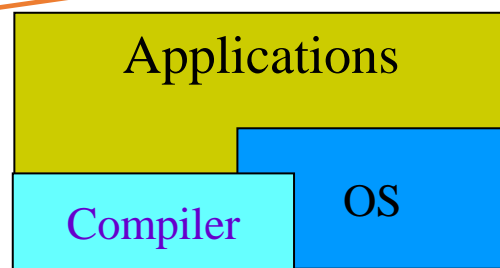
“指令系统”
一台计算机的
所有指令
集合



指令系统概述

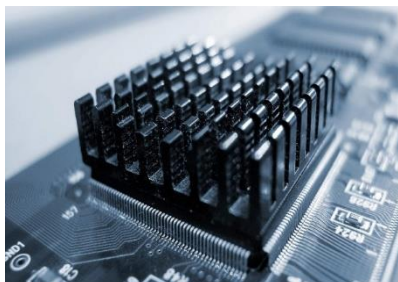


软件层
抽象

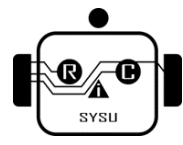
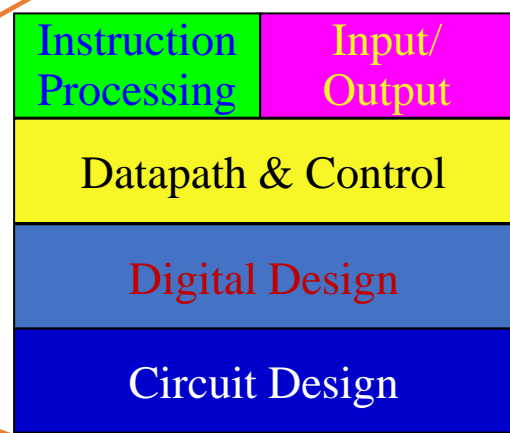


指令系统

计算机硬件和软件的
接口及界面



硬件层
抽象



指令系统概述

□ 指令系统(IS)处在软件和硬件的交界面上

■ 能同时被硬件设计者和系统程序员看到

□ 从硬件设计者角度来看

■ IS为CPU设计提供功能需求

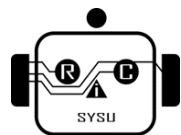
■ IS设计目标：易于硬件逻辑设计

□ 从系统程序员角度来看

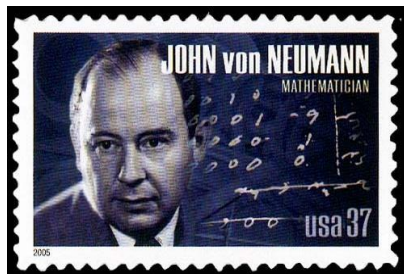
■ 通过IS使用硬件资源

■ IS设计目标：易于编写编译器

□ IS设计的好坏决定了计算机的性能和成本



指令系统概述

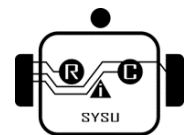


指令系统放在哪儿很简单，
复杂的是**如何设计**指令系统

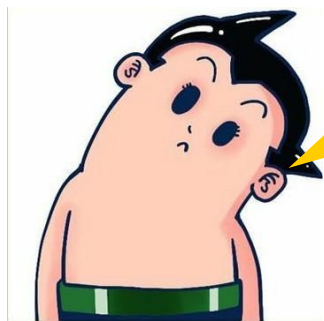
设计原则

完备性
有效性
规整性
兼容性

提供的指令足够解决任何可解的问题
简洁、加速常用操作、没有歧义
对称、匀齐、一致（简单源于规整）
之前/之后的都要能用



指令系统概述

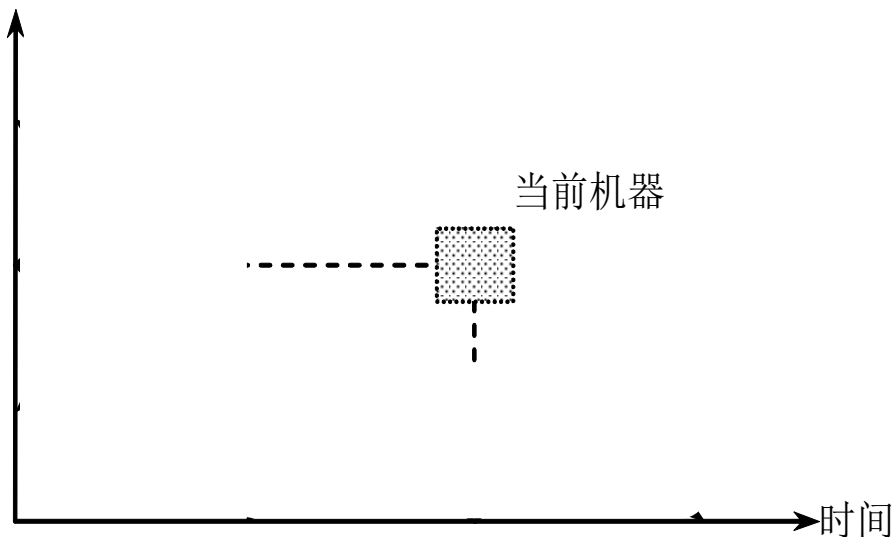


“兼容性” ???
不懂！！

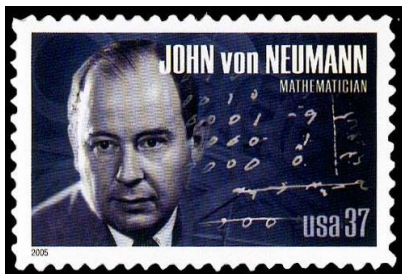
兼容性

- **向上（下）兼容**：按某档机器编制的程序，不加修改的就能运行于比它高（低）档的机器
- **向前（后）兼容**：按某个时期投入市场的某种型号机器编制的程序，不加修改就能运行于在它之前（后）投入市场的机器

机器档次



指令系统概述



指令系统放在哪儿很简单，
复杂的是**如何设计**指令系统

设计原则

完备性
有效性
规整性
兼容性

该有的都要有
简洁、加速常用操作、没有歧义
对称、匀齐、一致
之前/之后的都要能用

一个较完善的指令系统应该包括

数据传送指令

Load/Store指令

输入输出指令

In/Out指令

算术运算指令

Add等指令

逻辑运算指令

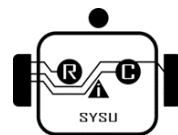
And等指令

系统控制指令

中断等指令

程序控制指令

Jump等指令

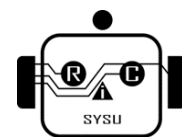


什么是计算机的语言？

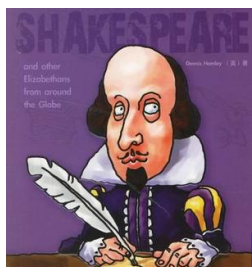
2.1.1 指令系统概述

2.1.2 两种类型指令系统计算机：CISC与RISC

WALL·E

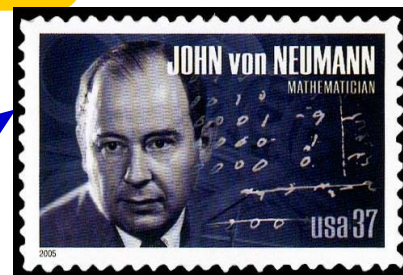


两种类型指令系统计算机CISC与RISC



There are a thousand Hamlets in a thousand people's eyes.

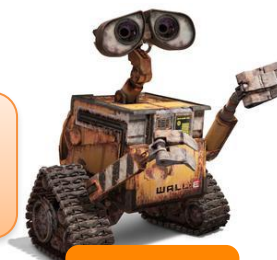
但是，指令系统的设计大体上**只有两种**，
不是1000种



RISC

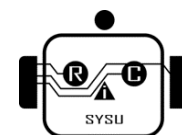


复杂指令集计算机
Complex Instruction Set Computer



CISC

精简指令集计算机
Reduced Instruction Set Computer



两种类型指令系统计算机CISC与RISC



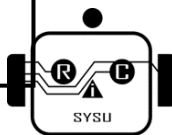
大家好，我
是CISC

- (1) 指令系统复杂
- (2) 指令周期长
- (3) 各种指令都能访问存储器
- (4) 有专用寄存器
- (5) 采用微程序控制
- (6) 难以进行编译优化生成高效目标代码

出现较早，大而全

例：VAX-11/780小型机

- 16种寻址方式
- 9种数据格式
- 303条指令
- 一条指令包括1 ~ 2个字节的操作码和下续N个操作数说明符
- 一个说明符的长度达1 ~ 10个字节
- 除专门的存储器读写指令外，运算指令也能访问存储器



两种类型指令系统计算机CISC与RISC



大家好，我是
CISC

- (1) 采用微程序控制
- (2) 有专用寄存器
- (3) 各种指令都能访问存储器
- (4) 指令系统复杂
- (5) 指令周期长
- (6) 难以进行编译优化生成高效目标代码

出现较早，大而全

CISC存在的问题

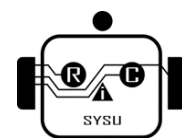
- 研制周期长
- 难以保证设计的正确性，难以调试和维护
- 机器的时钟周期长，降低了系统性能
- 效率低下 (*IBM*的测试发现)
 - 指令系统中约占20%的简单指令，在程序中的比例约为80%
 - 在程序中比例20%的一些复杂指令，占用了控制存储器容量的80%

Top 10 80x86 Instructions

° Rank	instruction	Integer Average Percent total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

简单指令占主要部分
使用频率高！

- ° Simple instructions dominate instruction frequency



两种类型指令系统计算机CISC与RISC

John cock & 小而精

- (1) 简化的指令系统
- (2) 以寄存器-寄存器方式工作
- (3) 指令周期短
- (4) 采用大量通用寄存器，以减少访存次数
- (5) 采用组合逻辑电路控制，不用或少用微程序控制
- (6) 采用优化的编译系统，力求有效地支持高级语言程序

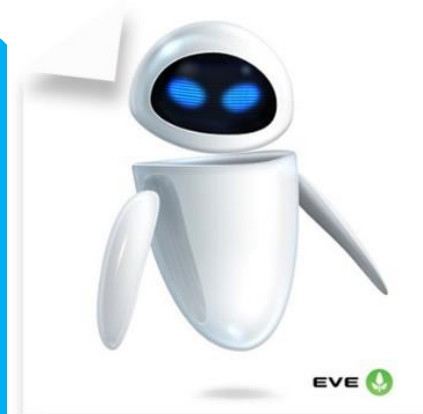


2.1.2 两种类型指令系统计算机CISC与RISC

Hi, 我是RISC

John cock & 小而精

- (1) **简化**的指令系统
- (2) 以**寄存器-寄存器**方式工作
- (3) **指令周期短**
- (4) 采用大量**通用寄存器**，以减少访存次数
- (5) 采用**组合逻辑电路**控制，不用或少用微程序控制
- (6) 采用**优化**的编译系统，力求有效地支持高级语言程序

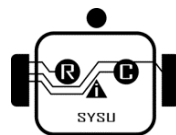


出现较早，大而全

- (1) 指令系统**复杂**
- (2) **各种指令**都能访问存储器
- (3) **指令周期长**
- (4) 有**专用寄存器**
- (5) 采用**微程序**控制
- (6) **难以**进行**编译优化**生成高效目标代码

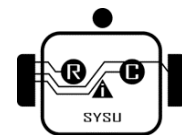


我是CISC



典型体系结构的通用计算器的数目

Machine	Number of general-purpose registers	Architectural style	Year
EDSAC	1	Accumulator	1949
IBM 701	1	Accumulator	1953
CDC 6600	8	Load-store	1963
IBM 360	16	Register-memory	1964
DEC PDP-8	1	Accumulator	1965
DEC PDP-11	8	Register-memory	1970
Intel 8008	1	Accumulator	1972
Motorola 6800	2	Accumulator	1974
DEC VAX	16	Register-memory, memory-memory	1977
Intel 8086	1	Extended accumulator	1978
Motorola 68000	16	Register-memory	1980
Intel 80386	8	Register-memory	1985
ARM	16	Load-store	1985
MIPS	32	Load-store	1985
HP PA-RISC	32	Load-store	1986
SPARC	32	Load-store	1987
PowerPC	32	Load-store	1992
DEC Alpha	32	Load-store	1992
HP/Intel IA-64	128	Load-store	2001
AMD64 (EMT64)	16	Register-memory	2003



2.1.2 两种类型指令系统计算机CISC与RISC

John cock & 小而精

例：第一代RISC机

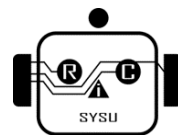
- 加州伯克利大学的RISC I
- 斯坦福大学的MIPS
- IBM公司的IBM801



CISC与RISC之争

- 现代处理器大多采用RISC体系结构
- Intel x86为“兼容”需要，保留CISC风格，同时借鉴了RISC思想

CISC与RISC的逐步融合



练习1

以下指令集架构属于复杂指令集架构的是？

ARM

MIPS

SPARC

以上皆不是

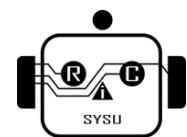
两种类型指令系统计算机CISC与RISC

常用的RISC指令集

- ARM
- MIPS
- RISC-V
- DECAIpha
- PowerArchitecture (包括PowerPC)
- SPARC



ARM: Advanced RISC Machine



两种类型指令系统计算机CISC与RISC

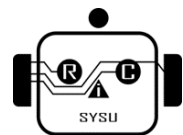
常用的RISC指令集

- ARM
- MIPS
- RISC-V
- DECAIpha
- PowerArchitecture (包括PowerPC)
- SPARC



国产CPU

- ARM系——飞腾、华为海思、展讯和华芯通
- MIPS系——龙芯和君正
- x86系——北大众志、兆芯和海光
- Alpha系——申威
- Power系——中晟宏芯



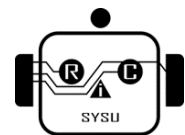
本讲内容

- 什么是计算机语言？

 - 指令集概述 Introduction to Instruction Set

- 指令格式

- 寻址方式



指令格式

指令含义

指令是指挥计算机实现**某个基本操作**的命令

决定

指令格式

什么操作?

操作码

操作的对象?

操作数

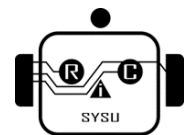
如何找到操作对象?

寻址方式

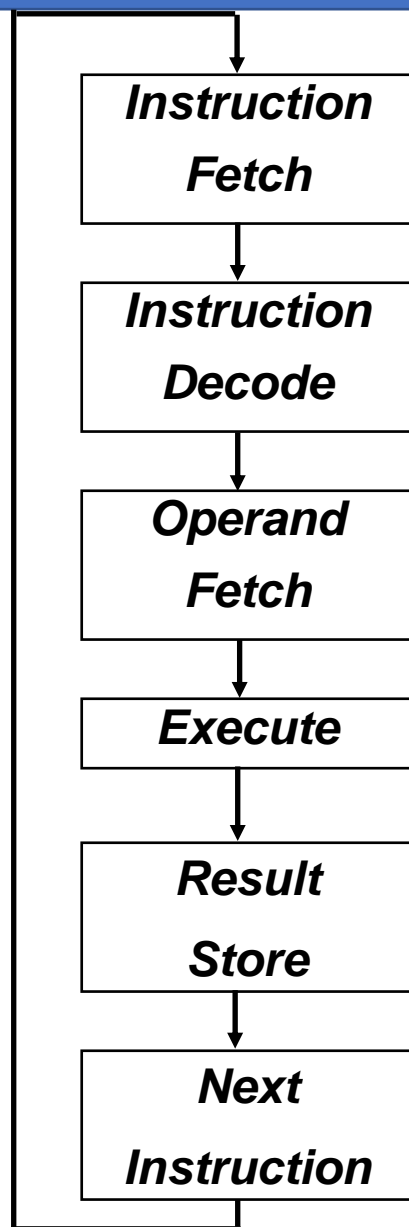
操作码

地址码

指令长度



从指令执行过程看指令设计涉及的问题



Obtain instruction from program storage (取指)
指令地址、指令长度(定长/变长)

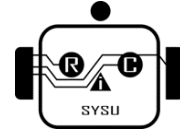
Determine required actions (译码)
指令格式、操作码编码、操作数类型

Locate and obtain operand data (取操作数)
地址码格式、寻址方式、操作数格式和存放方式

Compute result value or status (执行)
操作类型、标志或条件码

Deposit results in storage for later use (写结果)
结果数据位置

Determine successor instruction (下条指令地址)
下条指令地址(顺序 / 转移)



指令格式

□与指令设计相关的问题

□操作码组成：操作码个数/种类/复杂度

如：Load/Store/INC/Branch 四种指令已足够编制任何可计算程序，但编写的程序会很长

□数据类型：多种数据类型可执行操作

如：字节、半字、字等

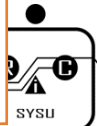
□指令格式：指令长度/地址码个数/各字段长度

□通用寄存器：个数/功能/长度

□寻址方式：操作数地址的指定/计算方式

□下条指令地址的确定：顺序(PC+4)、转移(目的地址)

MIPS指令系统中，一般通过对操作码的不同编码定义不同的操作。若操作码相同时，再由功能码定义不同的操作！



指令格式

指令格式

操作码

地址码

指令长度的设计

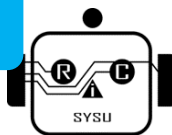
1. 问：每条指令的长度可以是不一样的么？

□ 指令长度

- 一条指令包含的二进制代码位数
- 取决于操作码长度、操作数地址长度和地址个数
 - ❖ 定长指令字：所有指令的长度相同。需向最长指令看齐 RISC
 - ❖ 变长指令字：不同指令的长度不同 CISC

设计原则之一

规整性



指令格式

指令格式

操作码设计

设计原则之一

操作码

地址码

1. 问：每条指令的操作码可以是几个？

1. 答：只能是一个（唯一含义）

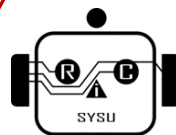
有效性

2. 问：具体的操作是怎么表示的？

2. 答：用一定长度的不同编码表示不同的操作

设计原则之二

完备性(足够的操作码位数)



指令格式

指令格式

操作码

地址码

地址码设计

问：每条指令的地址字段可以是几个？

答：0到多个，看操作码的需要了

□ 一条指令包含1个操作码和多个地址码

➤ 零地址指令

OP

➤ 一地址指令

OP

A1

➤ 二地址指令(最常用)

OP

A1

A2

➤ 三地址指令(RISC风格)

OP

A1

A2

A3

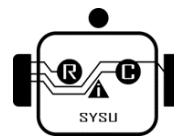
➤ 多地址指令

堆栈

累加器

通用寄存器

地址码个数与性能和实现难度密切相关



操作码设计

指令格式

操作码

地址码

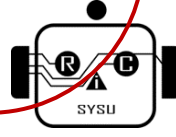
操作码长度

操作码长度问题

- ❑ 操作码的编码方式决定操作码的长度
 - Fixed Length Opcodes (定长操作码法)
 - Expanding Opcodes (变长/扩展操作码法)

问题：是否有定长指令字、变长操作码？
或者是定长操作码、变长指令字呢？

实际上，指令长度是否可变与操作码长度是否可变没有绝对联系，但通常是“定长操作码、不一定是定长指令字”、“变长操作码、一般是变长指令字”。



操作码设计

定长操作码

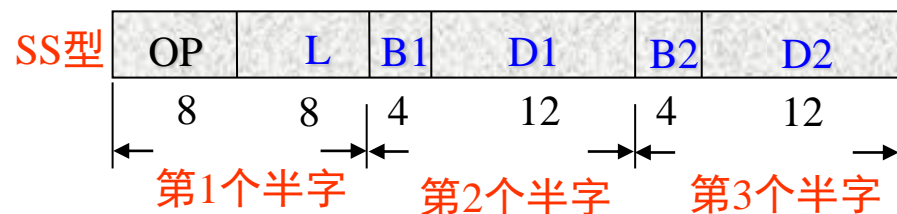
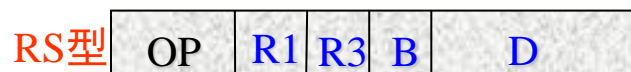
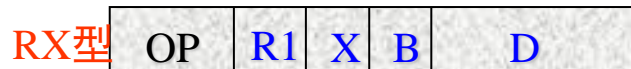
□基本思想:指令操作码部分采用**固定长度**的**编码**

例如: 假设操作码固定为6位, 则系统最多可表示 2^6 种指令

特点

译码简单, 但有信息冗余

例: IBM360/370采用: 8位定长操作码, 最多可有256条指令。只提供了183条指令, 有73种编码为冗余信息。



Ri: 寄存器

X: 变址器

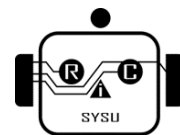
Bi: 基址器

Di: 位移量

I: 立即数

L: 数的长度

IBM370指令格式

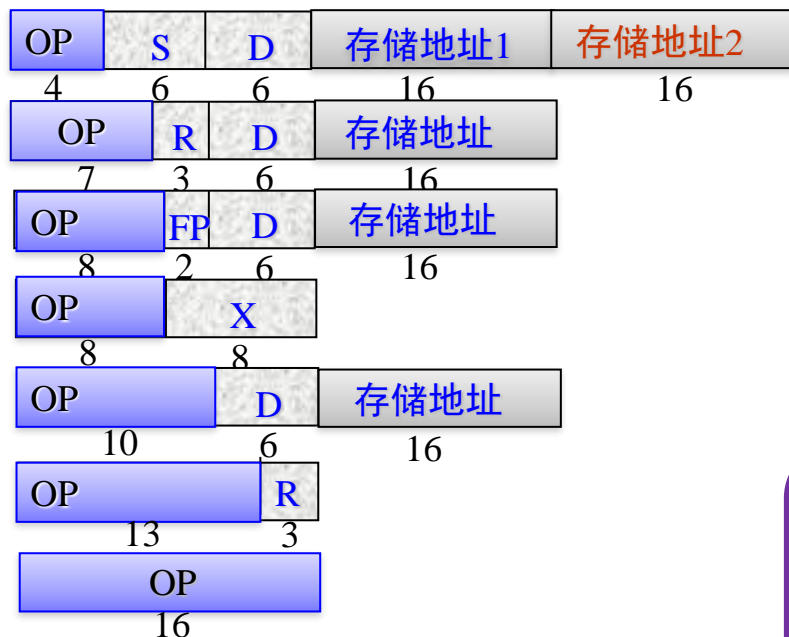


操作码设计

扩展操作码

□ **基本思想**：指令的操作码部分采用**可变长度**的编码

操作码的编码长度分成**几种固定长**的格式，操作码的位数**随地址数的减少而增加**，被大多数指令集采用



PDP-11中典型指令格式

优点

- 缩短指令长度
- 减少程序总位数
- 增加指令字所能表示的操作信息

一个重要原则：

使用频度高的指令：短的操作码

使用频度低的指令：较长的操作码

操作码设计

回顾一下指令格式设计的基本原则

指令尽量短

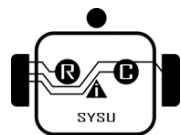
要有足够的操作码位数

指令编码必须具有唯一的解释

指令字长应是字节的整数倍

均衡设计、指令尽量规整

合理选择地址字段的个数



地址码结构

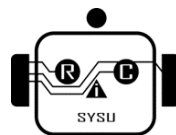
□ 地址个数

地址个数越少 → 指令长度越短 → 指令功能越简单

地址个数越少 → 所需指令条数越多 → 增加了程序复杂度和执行时间

- 堆栈结构：零地址指令
- 累加器结构：一地址指令
- 通用寄存器结构：二、三地址指令

指令中
地址个数一般
不超过3个



地址码结构

□ 一条指令包含1个**操作码**和多个**地址码**

■ **零地址指令**

OP

(1) 无需操作数。如：空操作 / 停机等

(2) 所需操作数为默认的。如：堆栈等

■ **一地址指令**

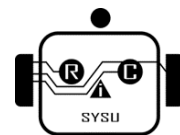
OP

A1

其地址既是源操作数地址，也是存放结果地址

(1) 单目运算：如：取反 / 取负等

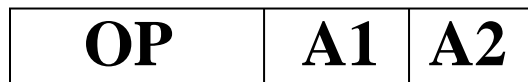
(2) 双目运算：另一操作数为默认的 如：累加器等



地址码结构

□ 一条指令包含1个操作码和多个地址码

■ 二地址指令(最常用)



分别存放双目运算中两个源操作数地址，并将其中一个地址作为存放结果地址

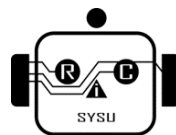
■ 三地址指令(RISC风格)



分别为双目运算中两个源操作数地址和一个结果地址

■ 多地址指令

用于成批数据处理的指令，如：向量指令等



地址码结构

例： $Y = (A - B) \div (C + D \times E)$

二地址指令

指令		操 作
MOVE	Y,A	$Y \leftarrow (A)$
SUB	Y,B	$Y \leftarrow (Y) - (B)$
MOVE	T,D	$T \leftarrow (D)$
MUL	T,E	$T \leftarrow (T) \times (E)$
ADD	T,C	$T \leftarrow (T) + (C)$
DIV	Y,T	$Y \leftarrow (Y) \div (T)$



一地址指令

指令		操 作
LOAD	D	$AC \leftarrow (D)$
MUL	E	$AC \leftarrow (AC) \times (E)$
ADD	C	$AC \leftarrow (AC) + (C)$
STOR	Y	$Y \leftarrow (AC)$
LOAD	A	$AC \leftarrow (A)$
SUB	B	$AC \leftarrow (AC) - (B)$
DIV	Y	$AC \leftarrow (AC) \div (Y)$
STOR	Y	$Y \leftarrow (AC)$

指令地址的结构，需要在硬件实现的简单和程序的复杂性之间做权衡

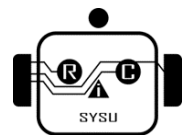
本讲内容

- 什么是计算机语言？

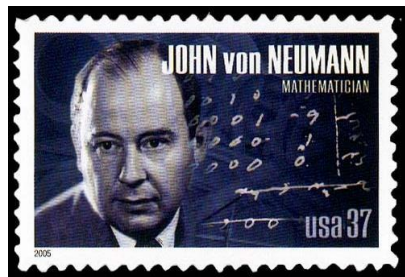
 - 指令集概述 Introduction to Instruction Set

- 指令格式

- 寻址方式

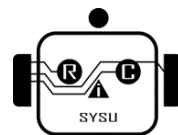
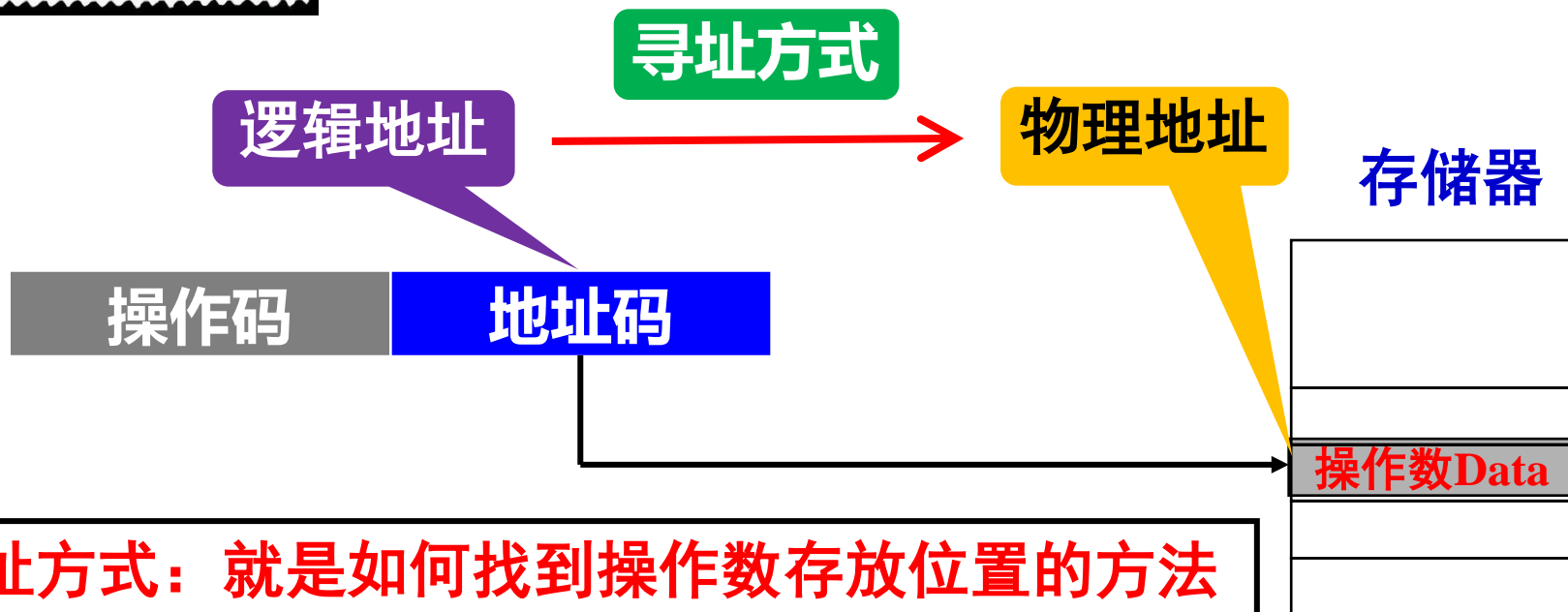


寻址方式



学习了操作码的编码之后，另一个问题就是地址码如何编码。

地址码编码由操作数的寻址方式决定。



寻址方式的概念



为什么不能**直接**把**操作数**放在**地址码**里面，这样就**不用**寻址方式了？？？？？

你说的有道理,非常合理!!!

但是**地址码**的**位数有限**，**更大的操作数**如何处理？操作数写到地址码也影响通用性

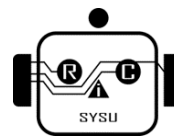


操作数Data


操作码

地址码

操作数只能这么多位吗？
更大的数怎么办？



寻址方式的概念



好吧，为什么不能**直接**把**操作数地址**放在**地址码**里面，这样也就不用**寻址方式**了？？？

地址码的位数有限，能放下多少个操作数地址？



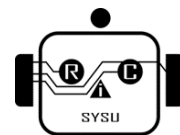
操作数物理地址

操作码


地址码

m

地址码太短，只能访问 2^m 个内存单元或寄存器中的操作数？



寻址方式的概念



好吧，为什么不能**直接把操作数地址**放在**地址码**里面，这样也就**不用寻址方式**了？？？？？

地址码的位数有限，能放下多少个操作数地址？



操作数物理地址

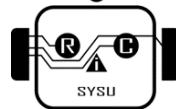
操作码

地址码

m

寻址方式出现的目的

- **扩大访存范围**
- **提高访问数据的灵活性和有效性**
- **支持软件技术的发展：多道程序设计**



寻址方式的概念

指令系统中的寻址

通常寻址方式特指“**操作数寻址**”

寻找：操作数
操作的对象放在哪了？

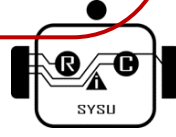
寻找：指令
下一条指令在哪啊？

指令寻址

- 指令寻址——简单
- ❖ 正常：PC增值
- ❖ 跳转 (jump / branch / call / return)：操作数就是第一条指令的地址

操作数寻址

- 操作数寻址——复杂
- ❖ 操作数的来源：寄存器 / 外设端口 / 主(虚)存 / 堆栈
- ❖ 操作数的数据结构：位 / 字节 / 半字 / 字 / 双字 / 一维表 / ...



基本寻址方式

基本寻址方式 (1) 立即数寻址



↑
源操作数

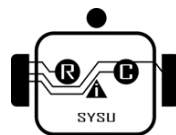
➤ 没错，就是之前讨论的——
不要寻址方式的方法
源操作数直接在指令中

- 指令地址字段直接给出操作数本身
- 立即数寻址只能作为双操作数指令的源操作数

例：MOV AX, 1000H

特点

1. 指令执行时间很短，无需访存
2. 操作数的大小受地址字段长度的限制
3. 广泛使用



基本寻址方式

基本寻址方式 (2)

存储器直接寻址



存储器



➤ $EA = A$, $Operand = (A)$

➤ 例: `MOV AX, [1000H]`

1. 处理简单、直接
2. 寻址空间受到指令的地址字段长度限制
3. 较少使用, 8位计算机和一些16位计算机

操作数在**存储器**中, 指令地址字段
直接给出操作数在存储器中的**地址**

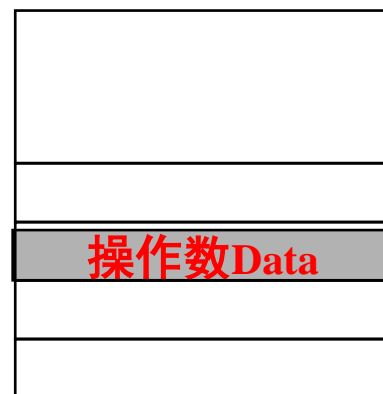
基本寻址方式

基本寻址方式 (3)

寄存器直接寻址



通用寄存器组



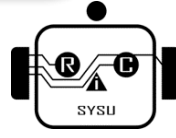
➤ $EA = R$, $Operand = (R)$

➤ 例: `MOV BX, AX`

1. 只需要很短的地址字段
2. 无需访存, 指令执行速度快
3. 地址范围有限, 可以编程使用的通用寄存器不多
4. 使用最多, 是提高性能的常用手段

操作数在寄存器中, 指令地址字段

直接给出存放操作数的寄存器编号



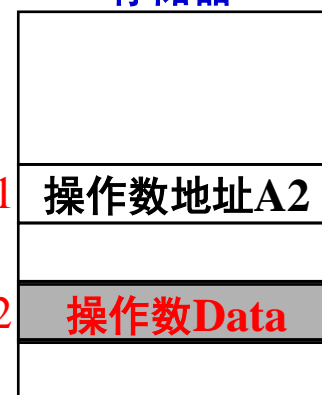
基本寻址方式

基本寻址方式 (4)

存储器间接寻址



存储器

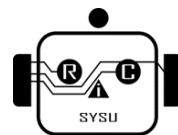


$EA = (A), \text{Operand} = ((A))$

例: MOV R1, @(1000H)

- 寻址空间大, 灵活, 便于编程
- 至少需要两次访存才能取到操作数
- 执行速度慢

- 操作数和操作数地址都在存储器中
- 指令地址字段直接给出操作数地址在存储器中的地址



基本寻址方式

基本寻址方式 (5)

寄存器间接寻址

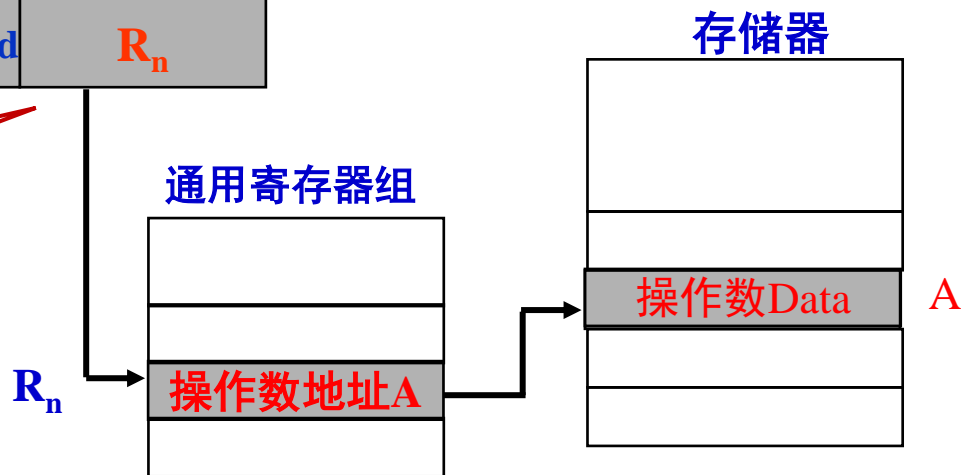


$EA = (R)$, $Operand = ((R))$

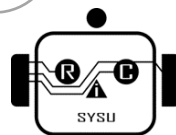
例: `MOV AX, [BX]`

➤ 比存储器间接寻址少访问存储器一次

➤ 寻址空间大, 使用比较普遍



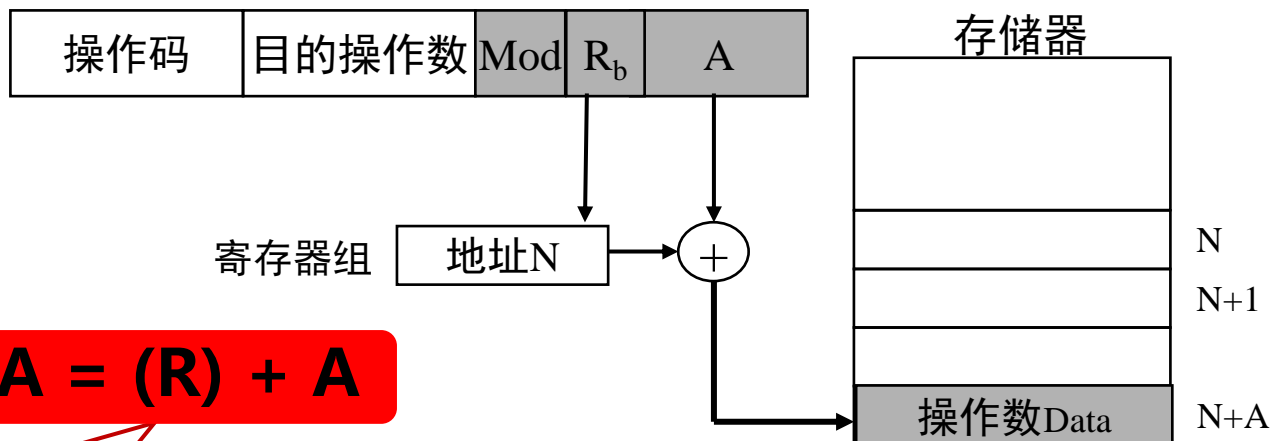
- 操作数在存储器中
- 操作数地址在寄存器中
- 指令地址字段给出的寄存器的内容是操作数在存储器中的地址



基本寻址方式

基本寻址方式 (6)

偏移寻址



$$EA = (R) + A$$

➤ 相对寻址

$$EA = (PC) + A$$

相对于当前指令处 位移量为A的单元

➤ 基址寻址：

$$EA = (B) + A$$

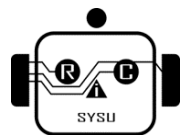
相对于基址(B)处 位移量为A的单元

➤ 变址寻址：

$$EA = (I) + A$$

相对于形式地址A处位移量为(I)的单元

直接寻址 + 寄存器间接寻址



相对寻址实现子程序的浮动和相对转移

***表示相对寻址方式**

公共子程序

JMP * +10	50
ADD AX,BX	72

存储器

.....	
JMP * +10	100
ADD AX,BX	122
⋮	
JMP * +10	250
ADD AX,BX	272
⋮	

子程序内地址关系**相对独立**，与用户程序的地址无关，不管浮动到哪里，总能实现程序中AX与BX的内容相加。

问题：采用相对寻址的转移指令中第一个字节是操作码OP，第二个字节是位移量，用**补码**表示，则转移目标地址的范围为多少？

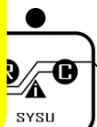
例：若转移指令地址为2000H，转移目标地址为1FF0H，总是在取指令的同时对PC增量，则转移指令第二个字节位移量是多少？

$$1FF0H - 2002H = EEH (-18)$$

主存按字/字节编址？目的地址

$$=(PC+\Delta)+2\times Disp?$$

位移量是8位
-128 ~ +127

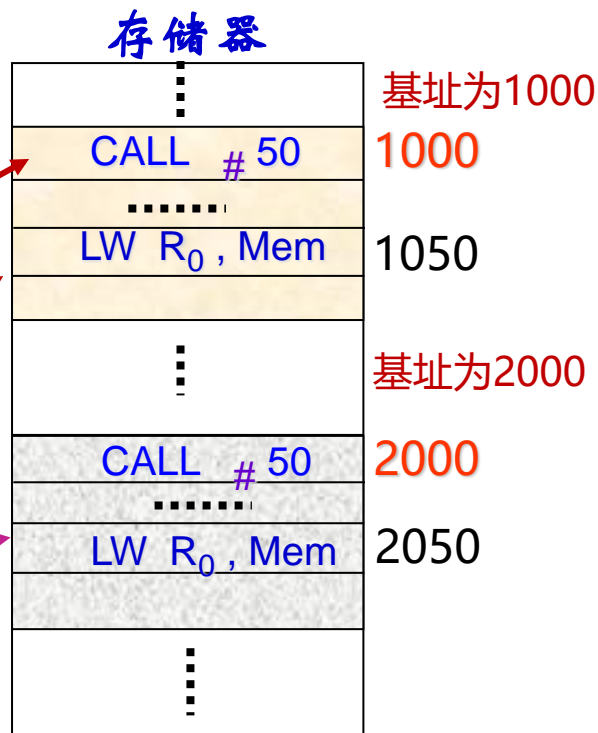
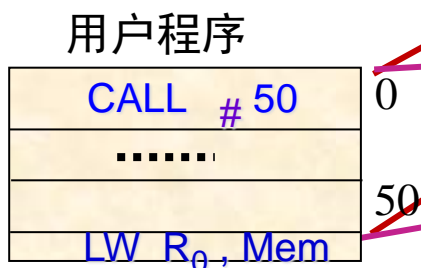


基址寻址实现程序的重定位

基址寻址实现程序重定位

例：假定程序的某一条指令是要调用绝对地址为50的一个过程。若这个程序被调入分区1(地址1000)，则指令跳转的目标地址50，真正需要调用的地址是1050。若这个程序被调入分区2(地址2000)，那么该程序就会去调用2050。这就是“重定位”。

#表示基址寻址方式



用户程序由OS装入系统后会分配一个基地址，存放在基址寄存器B中

$$EA = (B) + A$$

基址寄存器的内存由操作系统确定，在程序执行过程中不能由用户随意改变！

基本寻址方式

变址寻址实现线性表元素的存取

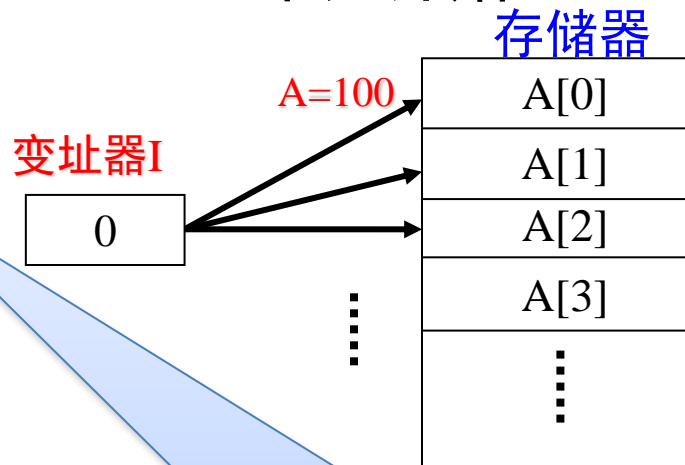
自动变址

指令地址字段A给出数组基址，变址寄存器I每次自动加/减数组元素的长度X。

$$EA = (I) + A, \quad I = (I) \pm X$$

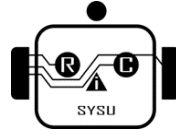
- 在元素地址从低→高地址增长时，“+”
- 在元素地址从高→低地址增长时，“-”
- 在没有硬堆栈的情况下，用来建立软堆栈提供对线性表的访问。

若一维数组A从内存100号单元开始



- 若每个元素为1个字节,则
 $I = (I) \pm 1$
- 若每个元素为4个字节,则
 $I = (I) \pm 4$

变址寄存器的内存由用户设定，在程序执行过程中其值可变，而指令字中的形式地址A是不可变的！



基本寻址方式

基址寻址

◆ 对于一道程序，基址是不变的，程序中的所有地址都是相对于基址变化的

◆ 在基址寻址中，偏移量位数较短

◆ 基址寻址立足于面向系统，主要是解决程序逻辑空间与存储器物理空间的无关性

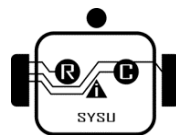
变址寻址

◆ 对于变址寻址，形式地址给出的是一个存储器地址基准，变址寄存器存放的是相对于该基准地址的偏移量

◆ 而在变址寻址中，偏移量位数足以表示整个存储空间

◆ 而变址寻址立足于用户，主要是为编写高效访问一片存储空间的程序

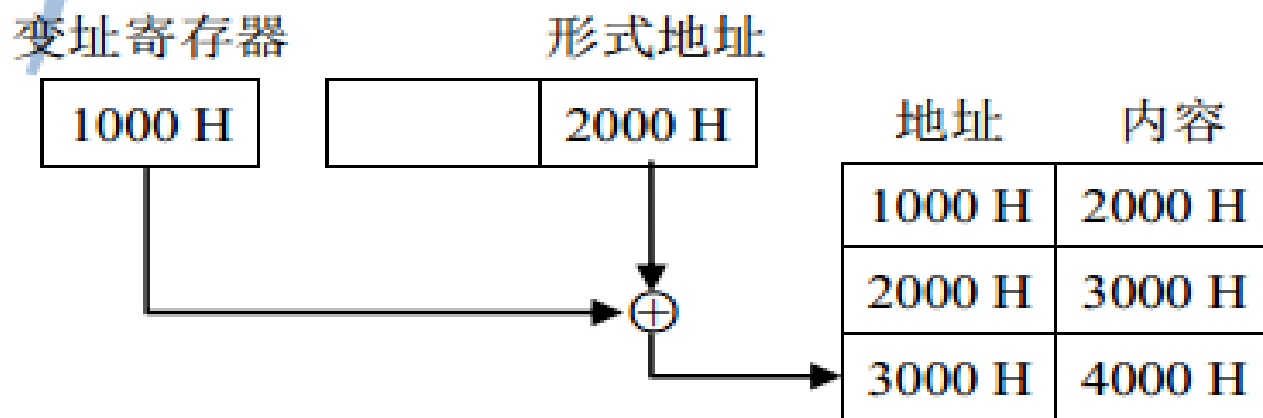
软件设计的重要支持基础



寻址方式举例

2013年全国统考：假设变址寄存器R的内容为1000H，指令中的形式地址为2000H；地址1000H中的内容为2000H，地址2000H中的内容为3000H，地址3000H中的内容为4000H，则变址寻址方式下访问到的操作数是()。

A. 1000H B. 2000H C. 3000H D. 4000 H



寻址方式举例

2009年全国统考：某机器字长16位，主存按字节编址，转移指令采用相对寻址，由两个字节组成，第一字节为操作码字段，第二字节为相对位移量字段。假定取指令时，**每取一个字节PC自动加1**。若某转移指令所在主存地址为2000H，相对位移量字段的内容为06H，则该转移指令成功转移[💡]的目标地址是()。

A.2006H

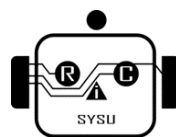
B.2007H

C.2008H

D.2009H

考查知识点

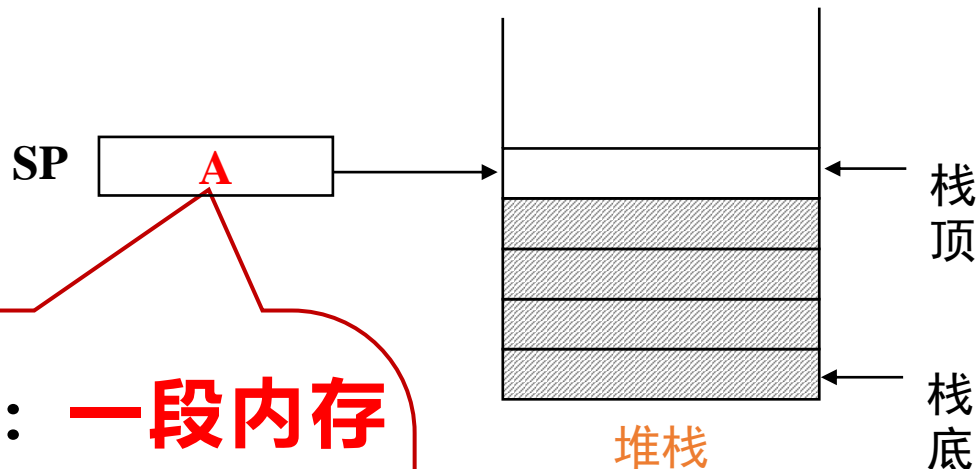
- 偏移寻址的三种方式
- 形式地址到物理地址的变换



基本寻址方式

基本寻址方式 (7)

堆栈寻址



➤ 堆栈的结构：**一段内存区域**

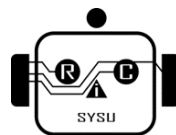
➤ 栈底、栈顶

➤ 堆栈指针(SP)：一个**特殊寄存器**，指向栈顶

PUSH (从寄存器到堆栈)

POP (从堆栈到寄存器)

零地址数



基本寻址方式

基本寻址方式 (7)

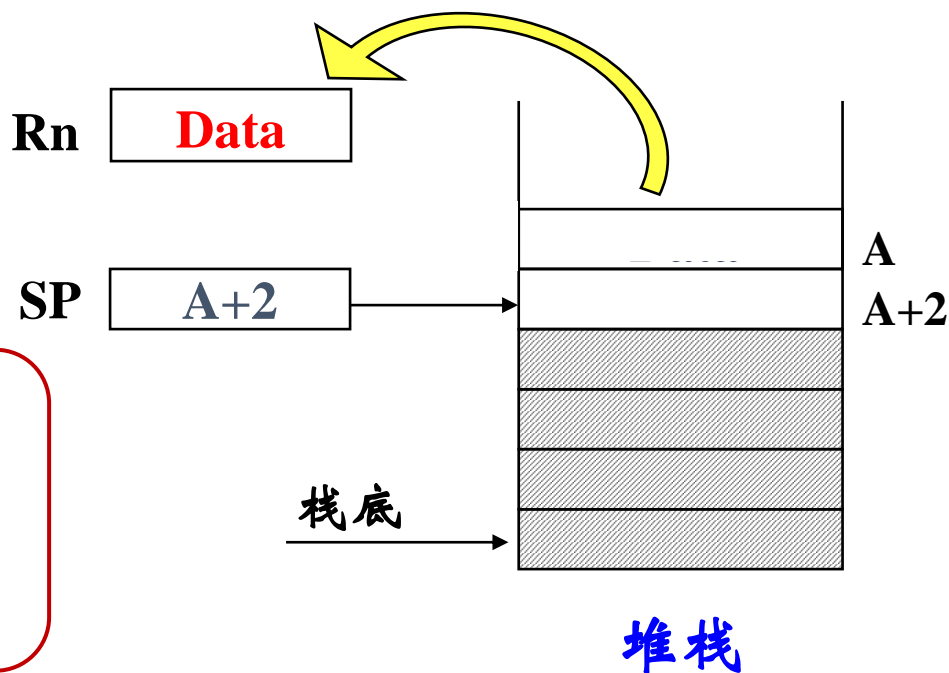
堆栈寻址

堆栈操作示例

□ 堆栈操作

出栈操作: **POP Rn**

$Rn \leftarrow ((SP)), SP \leftarrow (SP) + 2$



基本寻址方式

例题1：基本寻址方式

例：累加器型指令

- ①立即数寻址 MOV AX, 500H
- ②直接寻址 MOV AX, [500H]
- ③寄存器寻址 MOV AX, BX
- ④寄存器间接寻址 MOV AX, [BX]
- ⑤基址寻址 MOV AX, 500H[BX]

PC=250

BX=400

AX=?

250H	MOV
251H	500H
252H	Next Instruction
→ 400H	700H
→ 500H	800H
700H	600H
800H	300H
→ 900H	200H



基本寻址方式

那么多寻址方式，我怎么知道**具体**的寻址方式是怎么确定的啊？

嗯嗯！！问得非常好，让我来告诉你



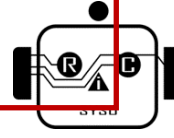
寻址方式的确定

(1) 在操作码中给定寻址方式

如：MIPS指令中仅有一个主(虚)存地址，且指令中仅有一、二种寻址方式

(2) 专门的寻址方式位：如：X86指令

0/1字节	0/1字节	1/2字节	0/1字节	0/1/2字节	0/1/2字节
指令前缀	段超越	操作码	寻址方式	位移量	立即数



复合寻址方式和寻址方式实例

复合寻址

间接寻址



相对寻址

间接寻址



变址寻址

在寻址的灵活性和硬件的复杂性之间权衡

先间接

➤ 间接相对式

$$EA = (PC) + (A)$$

➤ 间接变址式

$$EA = (X) + (A)$$

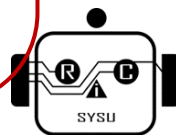
后间接

➤ 变址间接式

$$EA = ((X) + A)$$

➤ 相对间接式

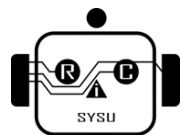
$$EA = ((PC) + A)$$



例题2：指令系统设计举例

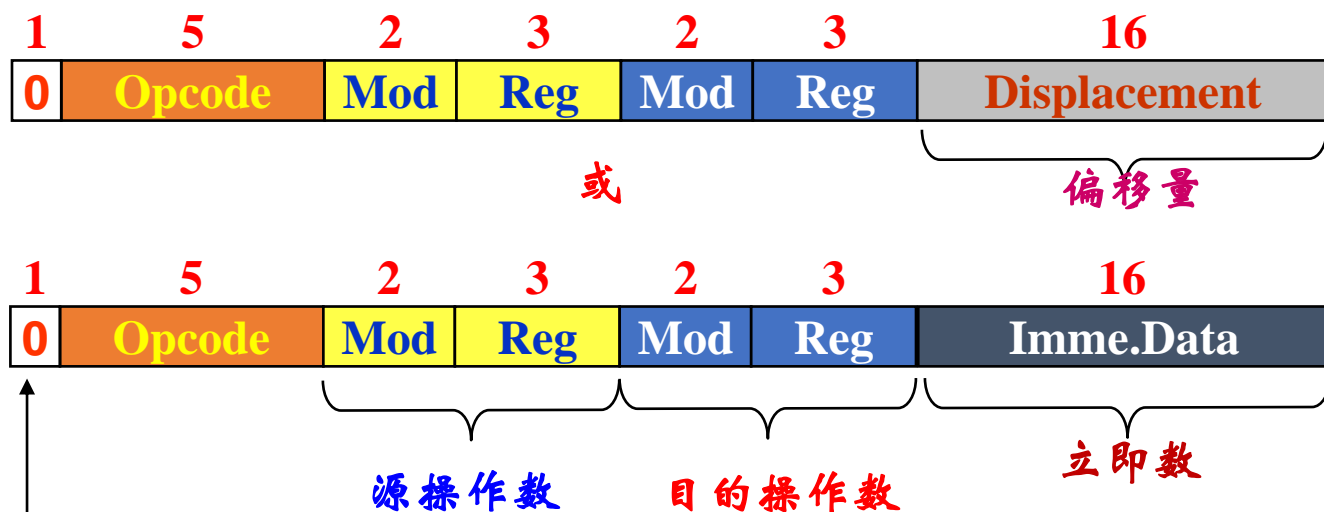
例：某机器字长为16位，数据总线16位，内存容量64KB，8个16位通用寄存器R0~R7。该指令系统的基本要求：

- 四种寻址方式：立即寻址，寄存器直接寻址，寄存器间接寻址，变址寻址；立即数和变址寻址时位移量均可达16位
- 32条双操作数指令（其中必有一操作数是寄存器直接寻址）
- 128条单操作数指令

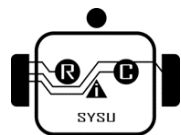


例题2：指令系统设计举例

双操作数指令格式



双操作数指令标志



例题2：指令系统设计举例

单操作数指令格式

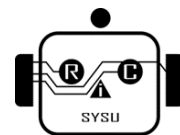


或



源/目的
操作数

单操作数指令标志



例题3：指令设计系统举例

2010年全国统考题：某计算机字长为16位，主存地址空间大小为128KB，按字编址。采用单字长指令格式，指令各字段定义如下：

15	12	11	6	5	0
OP	Ms	Rs	Md	Rd	

源操作数

目的操作数

转移指令采用相对寻址，相对偏移量用补码表示，寻址方式定义如下

Ms/Md	寻址方式	助记符	含义
000B	寄存器直接	Rn	操作数=(Rn)
001B	寄存器间接	(Rn)	操作数= ((Rn))
010B	寄存器间接、自增	(Rn)+	操作数=((Rn)), (Rn)+1->Rn
011B	相对	D(Rn)	转移目标地址=(PC)+(Rn)

注：(X)表示有存储地址 X 或寄存器 X 的内容



考查知识点

- 对指令格式的理解
- 对寻址方式的理解

例题3：指令设计系统举例

2010年全国统考题：某计算机字长为16位，主存地址空间大小为128KB，按字编址。采用单字长指令格式，指令各字段定义如下：

15	12	11	6	5	0
OP	Ms	Rs	Md	Rd	

源操作数

目的操作数

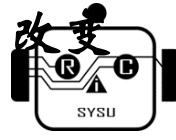
转移指令采用相对寻址，相对偏移量用补码表示，寻址方式定义如下

Ms/Md	寻址方式	助记符	含义
000B	寄存器直接	Rn	操作数=(Rn)
001B	寄存器间接	(Rn)	操作数= ((Rn))
010B	寄存器间接、自增	(Rn)+	操作数=((Rn)), (Rn)+1->Rn
011B	相对	D(Rn)	转移目标地址=(PC)+(Rn)

注：(X)表示有存储地址 X 或寄存器 X 的内容

请回答下列问题：

- (1)该指令系统最多可有多少条指令？该计算机最多有多少个通用寄存器？存储器地址寄存器(MAR)和存储器数据寄存器(MDR)至少各需要多少位？
- (2)转移指令的目标地址范围是多少？
- (3)若操作码0010B表示加法操作(助记符为add)，寄存器R4和R5的编号分别为100B和101B，R4的内容为1234H，R5的内容为5678H，地址1234H中的内容为5678H，地址5678H中的内容为1234H，则汇编语句“add (R4), (R5)+” (逗号前为源操作数，逗号后为目的操作数)对应的机器码是什么(用十六进制表示)？该指令执行后，哪些寄存器和存储单元的内容会改变？改变后的内容是什么？



例题3: 指令设计系统举例

请回答下列问题:

(1) 该指令系统最多可有多少条指令? 该计算机最多有多少个通用寄存器? 存储器地址寄存器(MAR)和存储器数据寄存器(MDR)至少各需要多少位?

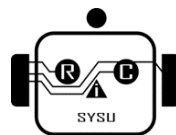
解:

最多可有 $2^4 = 16$ 条指令; $2^3 = 8$ 个通用寄存器;
MAR、MDR 至少需 16 位



考查知识点

- 指令指令系统的指令总数取决于操作码的位数
- 寄存器数决定了它的编码位数
- 存储地址寄存器(MAR)的位数取决于主存地址空间大小
- 存储数据寄存器(MDR)取决于机器字长



例题3: 指令设计系统举例

(2) 转移指令的目标地址范围是多少?

解: 转移指令的目标地址范围: $-2^{15} \sim +(2^{15} - 1)$



考查知识点

- 对寻址方式的理解和应用
- 对补码表示范围的掌握

15	12	11	6	5	0
OP	Ms	Rs	Md	Rd	

Ms/Md	寻址方式	助记符	含义
000B	寄存器直接	Rn	操作数=(Rn)
001B	寄存器间接	(Rn)	操作数= ((Rn))
010B	寄存器间接、自增	(Rn)+	操作数=((Rn)), (Rn)+1->Rn
011B	相对	D(Rn)	转移目标地址=(PC)+(Rn)

注: (X)表示有存储地址 X 或寄存器 X 的内容

例题3：指令设计系统举例

(3)若操作码0010B表示加法操作(助记符为add), 寄存器R4和R5的编号分别为100B和101B, R4的内容为1234H, R5的内容为5678H, 存储地址1234H中的内容为5678H, 5678H中的内容为1234H,

则汇编语言为add (R4), (R5)⁺ (逗号前为源操作符, 逗号后为目的操作数) 对应的机器码是什么(用十六进制表示)?

该指令执行后, 哪些寄存器和存储单元的内容会改变? 改变后的内容是什么?

15	12	11	6	5	0
OP	Ms	Rs	Md	Rd	

Ms/Md	寻址方式	助记符	含义
000B	寄存器直接	Rn	操作数=(Rn)
001B	寄存器间接	(Rn)	操作数= ((Rn))
010B	寄存器间接、自增	(Rn)+	操作数=((Rn)), (Rn)+1->Rn
011B	相对	D(Rn)	转移目标地址=(PC)+(Rn)

注: (X)表示有存储地址 X 或寄存器 X 的内容



例题3：指令设计系统举例

(3)若操作码0010B表示加法操作(助记符为add), 寄存器R4和R5的编号分别为100B和101B, R4的内容为1234H, R5的内容为5678H, 存储地址1234H中的内容为5678H, 5678H中的内容为1234H, 则汇编语言为add (R4), (R5)+ (逗号前为源操作符, 逗号后为目的操作数) 对应的机器码是什么(用十六进制表示)?该指令执行后, 哪些寄存器和存储单元的内容会改变? 改变后的内容是什么?

解:

汇编语言: add (R4), (R5)+

所对应的机器码是

该指令执行后, R5的内容改变, 其值变为5679H;

存储地址为5678H的存储单元内容亦改变, 其值变为68ACH

15	12	11	6	5	0
OP	Ms	Rs	Md	Rd	

Ms/Md	寻址方式	助记符	含义
000B	寄存器直接	Rn	操作数=(Rn)
001B	寄存器间接	(Rn)	操作数= ((Rn))
010B	寄存器间接、自增	(Rn)+	操作数=((Rn)), (Rn)+1->Rn
011B	相对	D(Rn)	转移目标地址=(PC)+(Rn)

注: (X)表示有存储地址 X 或寄存器 X 的内容

例题3：指令设计系统举例

(3)若操作码0010B表示加法操作(助记符为add)，寄存器R4和R5的编号分别为100B和101B，R4的内容为1234H，R5的内容为5678H，存储地址1234H中的内容为5678H，5678H中的内容为1234H，则汇编语言为add (R4), (R5)+ (逗号前为源操作符，逗号后为目的操作数)对应的机器码是什么(用十六进制表示)?该指令执行后，哪些寄存器和存储单元的内容会改变?改变后的内容是什么?

解：

汇编语言： add (R4), (R5)+

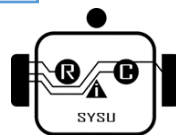
所对应的机器码是： 0010 001100 010101B = 2315H;

该指令执行后，R5的内容改变，其值变为5679H；存储地址为5678H的存储单元内容亦改变，其值变为68ACH



考查知识点

- 指令格式和寻址方式的理解
- 如何将汇编语句翻译为二进制机器指令



例题4: 指令系统设计举例

2014年考研题:

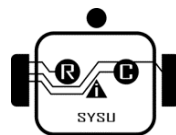
某程序中有如下循环代码段

P: “for($i=0; i < N; i++$) $sum+=A[i];$ ”, 假设编译时变量 sum 和 i 分别分配在寄存器 $R1$ 和 $R2$ 中, 常量 N 在寄存器 $R6$ 中, 数组 A 的首地址在寄存器 $R3$ 中。程序段P起始地址为0804 8100H, 对应的汇编代码和机器代码如表所示:



考查知识点

- 高级语言和机器级代码表示之间的关系



例题4: 指令系统设计举例

2014年考研题:

某程序中有如下循环代码段

P: “for($i=0; i < N; i++$) $sum+=A[i];$ ”, 假设编译时变量 sum 和 i 分别分配在寄存器 $R1$ 和 $R2$ 中, 常量 N 在寄存器 $R6$ 中, 数组 A 的首地址在寄存器 $R3$ 中。程序段P起始地址为0804 8100H, 对应的汇编代码和机器代码如表所示:

编号	地址	机器代码	汇编代码	注释
1	08048100H	00022080H	loop:slt R4,R2,2	$(R2) < 2 \rightarrow R4$
2	08048104H	00832020H	add R4,R4,R3	$(R4) + (R3) \rightarrow R4$
3	08048108H	8C850000H	load R5,0(R4)	$((R4) + 0) \rightarrow R5$
4	0804810CH	00250820H	add R1,R1,R5	$(R1) + (R5) \rightarrow R1$
5	08048110H	20420001H	addi R2,R2,1	$(R2) + 1 \rightarrow R2$
6	08048114H	1446FFFAH	bne R2,R6,loop	if $(R2) \neq (R6)$ goto loop

例题4: 指令系统设计举例

执行上述代码的计算机M采用32位定长指令字，其中分支指令bne采用如下格式：

31	26	25	21	20	16	15	0
OP		Rs		Rd		OFFSET	

其中OP为操作码；Rs和Rd为寄存器编号；OFFSET为偏移量，用补码表示。

请回答下列问题，并说明理由。

①M的存储器编址单位是什么？

编号	地址	机器代码	汇编代码	注释
1	08048 00H	00022080H	loop:sll R4,R2,2	(R2)<<2→R4
2	08048 04H	00832020H	add R4,R4,R3	(R4)+(R3)→R4
3	08048 08H	8C850000H	load R5,0(R4)	((R4)+0)→R5
4	08048 0CH	00250820H	add R1,R1,R5	(R1)+(R5)→R1
5	08048 10H	20420001H	addi R2,R2,1	(R2)+1→R2
6	08048 14H	1446FFFAH	bne R2,R6,loop	if(R2)≠(R6) goto loop

例题4: 指令系统设计举例

执行上述代码的计算机M采用32位定长指令字，其中分支指令bne采用如下格式：

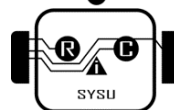
31	26	25	21	20	16	15	0
OP		Rs		Rd		OFFSET	

其中OP为操作码；Rs和Rd为寄存器编号；OFFSET为偏移量，用补码表示。

请回答下列问题，并说明理由。

② 已知sll指令实现左移功能，数组A中每个元素占多少位？

解：32位。 因为R2里面存放的是数组元素的下标i，将R2中的内容左移两位，相当于乘以4，然后加上R3中存放的数组的首地址，得到元素所在内存的地址，然后每循环一次，R2中内容自增1。因为存储器按字节编址，每计算一次移动4个位置，故每个数组元素占4个字节，即32位。



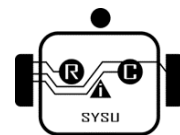
例题4: 指令系统设计举例

执行上述代码的计算机M采用32位定长指令字，其中分支指令bne采用如下格式：

编号	地址	机器代码	汇编代码	注释
1	08048100H	00022080H	loop:sll R4,R2,2	$(R2) \ll 2 \rightarrow R4$
2	08048104H	00832020H	add R4,R4,R3	$(R4) + (R3) \rightarrow R4$
3	08048108H	8C850000H	load R5,0(R4)	$((R4) + 0) \rightarrow R5$
4	0804810CH	00250820H	add R1,R1,R5	$(R1) + (R5) \rightarrow R1$
5	08048110H	20420001H	addi R2,R2,1	$(R2) + 1 \rightarrow R2$
6	08048114H	1446FFFAH	bne R2,R6,loop	if(R2) \neq (R6) goto loop

③ 表中bne指令的OFFSET字段的值是多少？已知bne指令采用相对寻址方式，当前PC内容为bne指令地址，通过分析表中指令地址和bne指令内容，推断出bne指令的转移目标地址计算公式。

解：OFFSET=FFFAH(-6)



例题4: 指令系统设计举例

执行上述代码的计算机M采用32位定长指令字，其中分支指令bne采用如下格

编号	地址	机器代码	汇编代码	注释
	00H	00022080H	loop:sll R4,R2,2	(R2)<<2→R4

考查知识点

- 通过指令地址分析存储器编址最小单位
- 通过指令分析出数组元素占多少位
- 推断跳转指令的转移目标地址计算公式

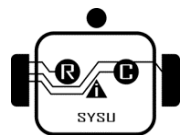
③ 表中bne指令的OFFSET字段的值是多少？已知bne指令采用相对寻址方式，当前PC内容为bne指令地址，通过分析表中指令地址和bne指令内容，推断出bne指令的转移目标地址计算公式。

解：OFFSET=FFFAH(-6)

指令bne所在地址为 0804 8114H，转移目标地址为 0804 8100H，因为 $08048100H = 08048114H + 4 + (-6) \times 4$ ，所以指令bne的转移目标地址计算公式为： $(PC) + 4 + OFFSET \times 4$

补充题1

- 在一个36位长的指令系统中，设计一个扩展操作码，使之能表示下列指令：
 - 7条具有两个14位地址和一个5位地址的指令；
 - 600条具有一个14位地址和一个5位地址的指令；
 - 100条无地址指令。



联系方式

□ Acknowledgements:

□ This slides contains materials from following lectures:

- Computer Architecture (ETH, NUDT, USTC)

□ Research Area:

- 计算机视觉与机器人应用计算加速,
- 人工智能和深度学习芯片及智能计算机

□ Contact:

- 中山大学计算机学院
- 管理学院D101 (图书馆右侧)
- 机器人与智能计算实验室
- cheng83@mail.sysu.edu.cn

