



中国科学院大学
University of Chinese Academy of Sciences



计算机科学导论 计算系统思维

徐志伟
中科院计算所
zxu@ict.ac.cn

编程练习

- 练习：请安装Go环境，并运行Hello程序
- 练习：请用Go编程语言实现自己姓名的字符相加编码符号变换。提示：采用ASCII字符集
- 每个同学将自己姓名的汉语拼音的字符相加，并打印出结果数值的字符。该练习涉及三类符号
 - 即汉语拼音字符（可简略为ASCII字符）Xu Zhi Wei
 - 整数数值符号（861），即“861”是一个符号，一个整数值
 - 三个十进制数值的字符（8，6，1）
- “徐志伟”同学的姓名编码最后屏幕显示是“861”
 - “徐志伟”同学的汉语拼音共包含10个字符，即Xu Zhi Wei
 - ASCII字符对应的数值相加之和是 $88 + 117 + 32 + 90 + 104 + 105 + 32 + 87 + 101 + 105 = 861$
 - 最后应打印出'8'，'6'，'1'三个ASCII字符

程序：计算过程的体现、载体

- 计算过程：通过操作数字符号变换信息的过程

- 数字符号

- 有名字
- 有类型
- 有值

- 表示数据（名词）

- 表示操作（动词）

- 函数、方法

- 可以组合

- 可以分解

```
1 package main
2 import "fmt"
3 func main() {
4     var name string = "Xu Zhi Wei"
5     var sum int = 0
6     var i int
7     for i = 0; i < len(name); i++ {
8         sum = sum + int(name[i])
9     }
10    var sum_bytes [5]byte
11    var j int
12    for j = len(sum_bytes) - 1; sum != 0; j-- {
13        sum_bytes[j] = byte(sum % 10) + '0'
14        sum = sum / 10
15    }
16    var k int
17    for k = j + 1; k < len(sum_bytes); k++ {
18        fmt.Printf("%c", sum_bytes[k])
19    }
20    fmt.Println()
21 }
```

Go语言大量使用函数、库、库函数

- 是特殊符号

- 函数func
- 库library
- 包package
- Dot notation

- 直接用

- 不要纠缠细节
- 编程课会讲细节
- 教科书有必要解释
- 容易习惯
- 无意识、有意识、下意识

```
1 package main
2 import (
3     "fmt"
4     "math/rand"
5     "runtime"
6     "time"
7 )
8 func main() {
9     rand.Seed(time.Now().UnixNano())
10    runtime.GC()
11    var m0 runtime.MemStats
12    runtime.ReadMemStats(&m0)
13    n := 128*1024*1024
14    var large_array = make([]int, n)
15    for i := 0; i < len(large_array); i++ {
16        large_array[i] = rand.Int()
17    }
18    start := time.Now()
19    quicksort(large_array[0:n])
20    elapsed := time.Since(start)
21    runtime.GC()
22    var m1 runtime.MemStats
23    runtime.ReadMemStats(&m1)
24    memUsage := m1.Alloc - m0.Alloc
25    fmt.Printf("Sort took %.3f seconds\n", float64(elapsed)/1000.0/1000.0/1000.0)
26    fmt.Printf("Cost memory %.3f MB\n", float64(memUsage)/1024.0/1024.0)
27    // TODO Your result checking code here
28 }
29 func quicksort(array []int) {
30     // TODO Your quicksort code here
31 }
```

必须考虑异常处理

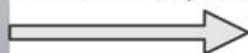
- 真正干的活
(payload)
- 实际代码大部分用于处理异常
 - 互联网访问异常
 - !(http.Get正常返回)
 - !(http得到正常值)
 - 本机I/O操作异常
 - ioutil.ReadAll出错
- 比特精准
 - 不要过分纠缠细节

```
httpresp := http.Get("http://csintro.ucas.ac.cn/static/code_project/Richard_Karp.txt")  
fmt.Println(string(httpresp.Body))
```

```
1 package main  
2 import (  
3     "fmt"  
4     "io/ioutil"  
5     "net/http"  
6     "os"  
7 )  
8 func main() {  
9     httpresp, err := http.Get("http://csintro.ucas.ac.cn/static/code_project/  
    /Richard_Karp.txt"); err != nil || httpresp.StatusCode != http.StatusOK {  
10         if err != nil {  
11             fmt.Fprintln(os.Stderr, err.Error())  
12         } else {  
13             fmt.Fprintln(os.Stderr, httpresp.Status)  
14         }  
15         return  
16     } else {  
17         if data, err := ioutil.ReadAll(httpresp.Body); err != nil {  
18             fmt.Fprintln(os.Stderr, err.Error())  
19         } else {  
20             fmt.Println(string(data))  
21         }  
22     }  
23 }
```



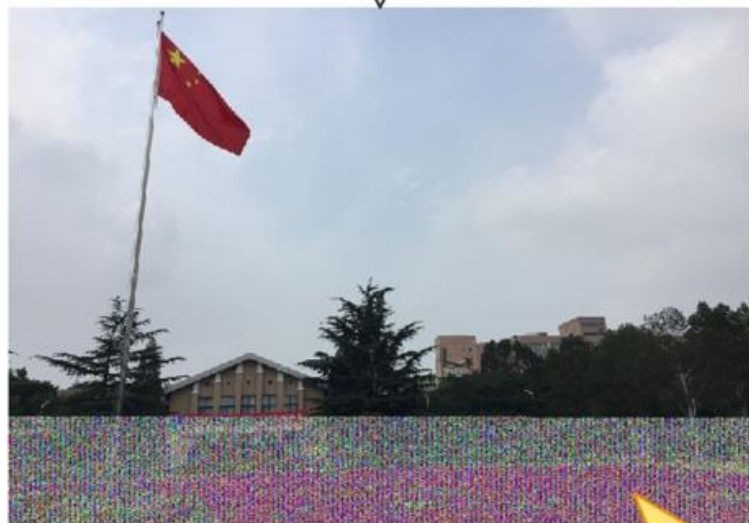
Richard_Karp.txt



B. 无明显视觉差异

Richard_Karp.txt

A. 原图

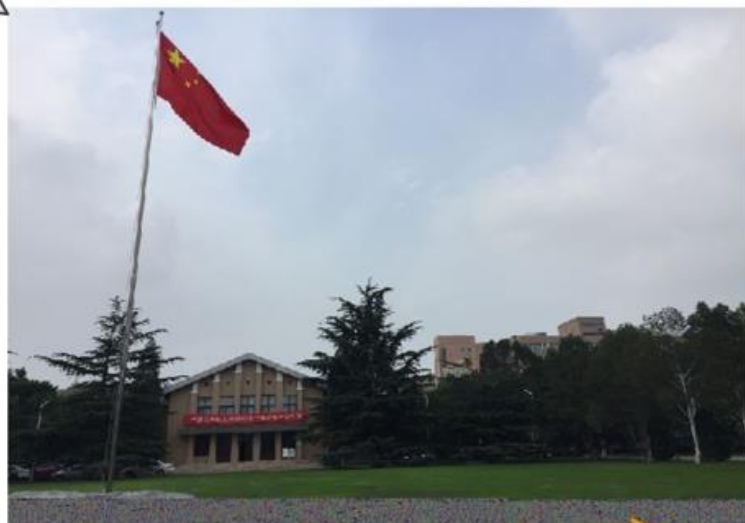


C. 视觉差异明显

文本的2 bits覆盖像素阵列1 byte的最高两位

Richard_Karp.txt

文本的2 bits覆盖像素阵列1 byte的最低两位



D. 视觉差异明显

文本的1 byte覆盖像素阵列的一个byte

本课程的要点

对计算思维的十种理解

理解1：自动执行。计算机能够自动执行由离散步骤组成的计算过程。

自动

理解2：正确性。计算机求解问题的正确性往往可以精确地定义并分析。

理解3：通用性。计算机能够求解任意可计算问题。

通用

理解4：构造性。人们能够构造出聪明的方法让计算机有效地解决问题。

理解5：复杂度。这些聪明的方法（算法）具备时间/空间复杂度。

算法

理解6：连接性。很多问题涉及用户/数据/算法的连接体，而非单体。

联网

理解7：协议栈。连接体的节点之间通过协议栈通信交互。

理解8：抽象化。少数精心构造的计算抽象可产生万千应用系统。

理解9：模块化。多个模块有规律地组合成为计算系统。

抽象

理解10：无缝衔接。计算过程在计算系统中流畅地执行。

系统思维将模块整合成为系统

解决“中间有一段很大的空白”难题

- 计算思维是交响乐
- 同学们如何操作
 - 初步理解《太玄经·差首》：“帝由群雍，物差其容”
 - 一定要看一遍教科书
 - 一定要自己做一遍编程练习

组合性

有效性

正确性

网络	系统
算法	
逻辑与计算模型	

计算系统方法

- 构造计算系统有条理、有门道，不是随意
 - 系统地（**systematic**）、而不是随意地（**ad hoc**）
 - 计算系统往往没有物理世界强加的限制，系统性更加重要
 - 产出一个系统整体，而不是一堆罗列
 - 系统性不是教条，不是用于阻碍创新
 - 既有系统性，又鼓励创新和多样性
 - 为什么计算系统设计者叫**architect**?
 - 为什么要有“计算机体系结构”（**computer architecture**）
- 近70年计算系统的发展
 - 做得好：支持数十亿用户，近百亿设备
 - 做的不好：多样性欠缺，一个病毒可毁坏百万设备

什么是计算系统思维？

- 计算过程必须在计算系统中运行
 - 计算系统包括抽象计算系统或真实计算系统
- 设计与理解计算系统的最大挑战：复杂性
- 系统思维方法
 - 通过抽象，将模块组合成系统，无缝执行计算过程
同义表达：将系统分解成模块的组合（**decomposition**）
 - 巧妙地定义抽象，将部件组合成系统，流畅地执行计算过程，提供应用价值
 - 模块就是精心设计的部件
 - 三个利器，即抽象化、模块化、无缝衔接
 - 抽象是最本质的考虑，模块化与无缝衔接是对抽象的补充
 - 系统思维方法本质上是抽象化方法（**abstraction**）

提纲

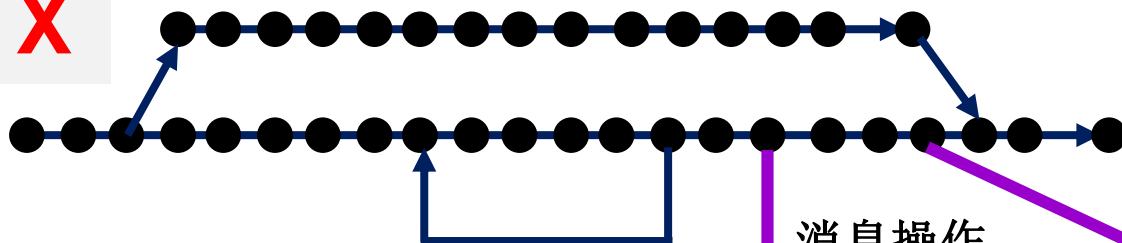
1. 计算过程与系统思维
 1. 应对复杂性挑战
 2. 抽象、模块、无缝衔接
2. 系统思维例子
3. 系统思维要点
 1. 抽象化
 2. 模块化
 3. 无缝衔接
4. 系统思维的创新实例

1. 计算过程与系统思维

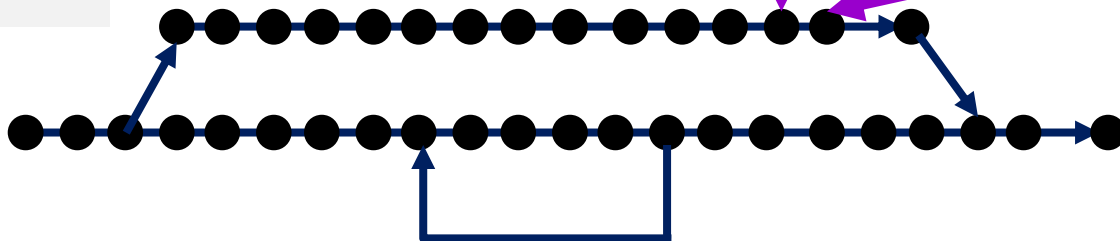
- 计算过程在计算系统中运行
 - 计算系统执行计算过程
 - 通用系统：一个计算系统可执行万千计算过程
 - 专用系统：一个计算系统可执行一个或几个计算过程



X



Y



Z



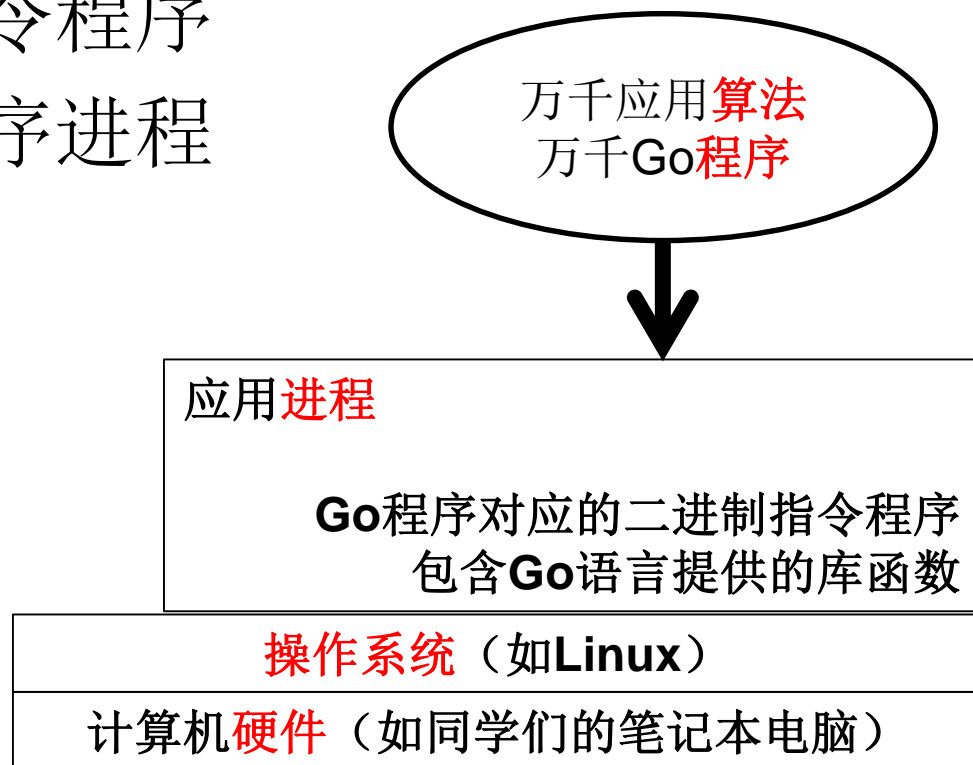
从计算系统思维角度看到的一些 计算机科学基本抽象

- 抽象使得计算机科学成为一门优美的领域
 - 本质：采用一种方法解决该层次的所有问题，应对系统的复杂性，以不变的抽象应系统的万变

数字符号	比特、字节、四进制数、十六进制数、整数、数组、 BMP 图像。	
软件	算法	巧妙的信息变换方法。例如信息隐藏算法。
	程序	算法的代码实现。例如实现信息隐藏算法的 <code>hide.go</code> 程序。
	进程	运行时的程序。例如在 Linux 环境中的 <code>hide</code> 进程。
	指令	程序的最小单位，计算机能够直接执行。
硬件	指令流水线	每条指令都通过“取指-译码-执行-写回”四个操作组成的指令流水线得以自动执行。所有指令都由这一种机制执行，指令流水线由若干时钟周期组成。
	时序电路	等同于 自动机 ，说明每一个时钟周期的操作。时序电路由组合电路与存储单元组成，理论上等同于 图灵机 。
	组合电路	实现二值逻辑表达式（ 布尔逻辑表达式 ）。

一套方法支持万千应用

- 程序员：应用算法→Go语言程序
- 编译器：Go语言程序→ 机器语言程序
- 计算机：执行二进制指令程序
- 操作系统：控制应用程序进程
- 万千应用，一套方法



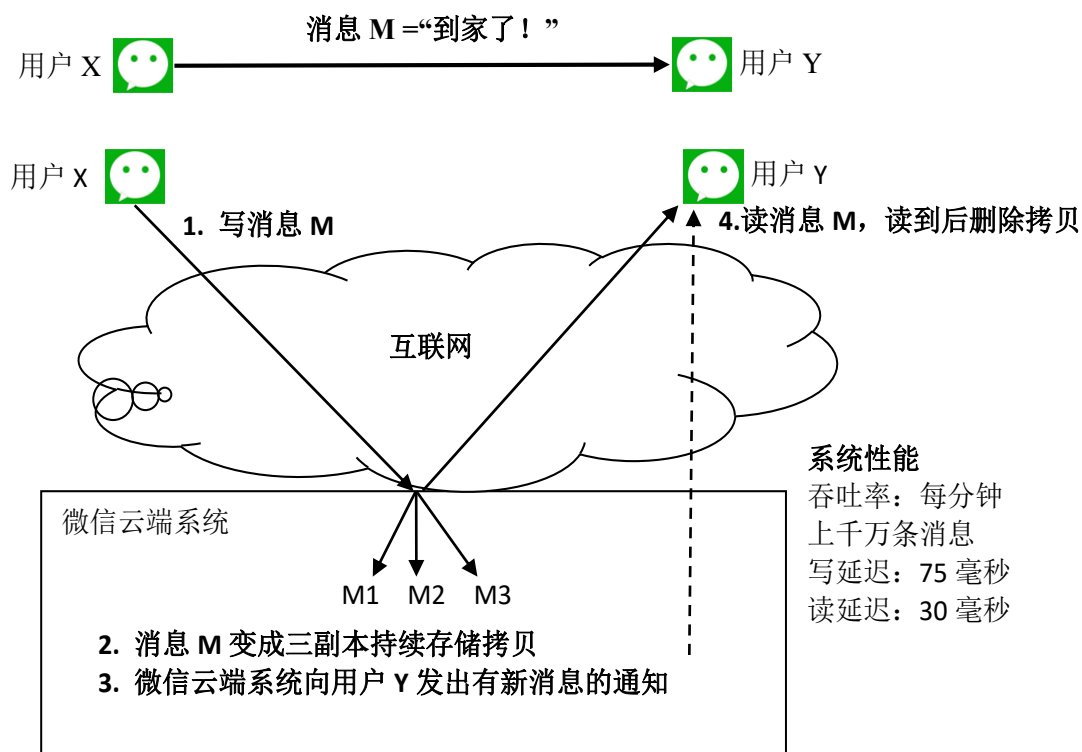
应对复杂性挑战

- 系统思维的主要目的是应对复杂性挑战
 - 微信系统比全世界的米级地图还要复杂的多
 - 多：微信系统涉及20亿亿个晶体管
 - 杂：多种应用需求、多种计算过程
 - 乱：不能靠随意堆砌20亿亿个晶体管解决问题
- 主要手段：抽象、模块、无缝衔接
 - 系统思维的主要手段是抽象化（abstraction）
 - 强调针对信息的抽象：数据抽象与控制抽象
 - 强调“比特精准、可自动执行的抽象”
 - 任何计算系统包含：处理、存储、互连三大子系统

2. 一个系统思维实例

- 不能靠简单地堆砌20亿亿个晶体管
 - 从多个层次（角度）理解一个系统
 - 每个层次仅考虑该层次特有问題，忽略其他问题
 - 用一套抽象概念和方法统一地处理该层次所有的计算过程，解决这些特有问題

用户层抽象

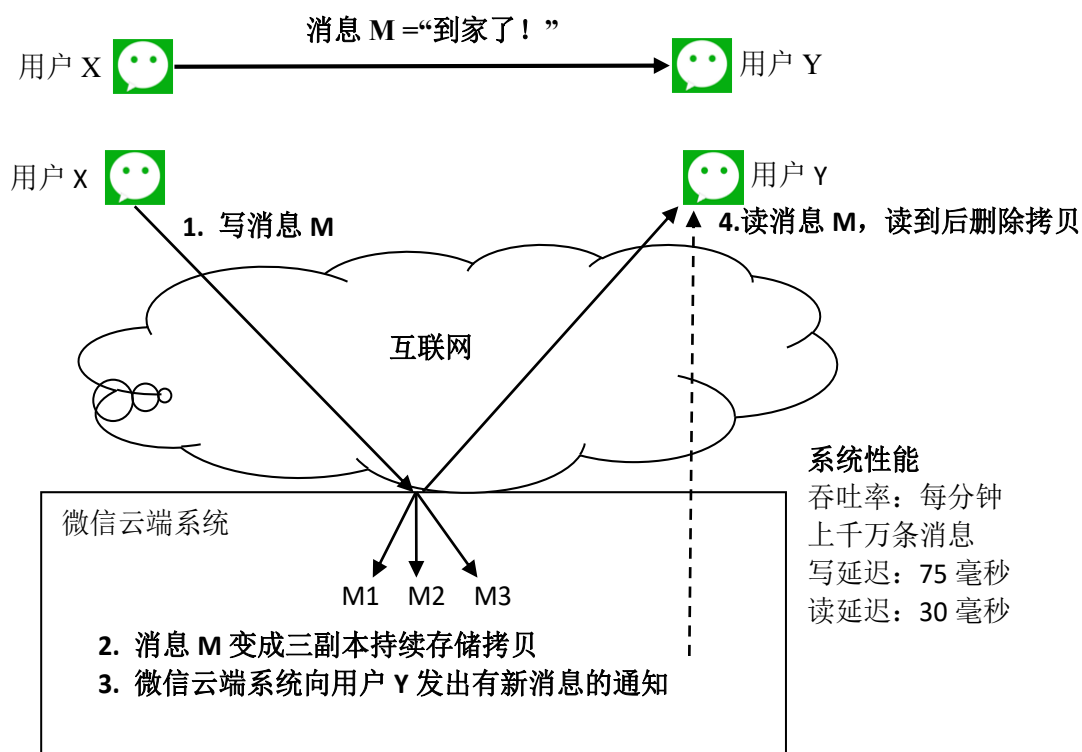


抽象概念	用户 客户端设备 消息
计算过程	浏览消息 收消息 发消息
关心问题	如何浏览消息 如何收消息 如何发消息 消息可以含自拍视频吗 如何加好友 多久好友才能收到 什么样的终端设备 什么样的网络
忽略问题	消息不丢失、不发错

一个系统思维实例

- 不能靠简单地堆砌20亿亿个晶体管
 - 从多个层次（角度）理解一个系统
 - 每个层次仅考虑该层次特有问題，忽略其他问题
 - 用一套抽象概念和方法统一地处理该层次所有的计算过程，解决这些特有问題

应用层抽象



抽象概念

用户、客户端设备、客户端应用软件、消息、微信云端系统

计算过程

编写并发送消息
向微信云端系统写消息
复制三份写入持续存储
发送“有新消息”通知
读取消息

关心问题

如何实现消息的处理和传递、如何保证消息不丢失、不发错，微信系统能够支持多少用户同时在线

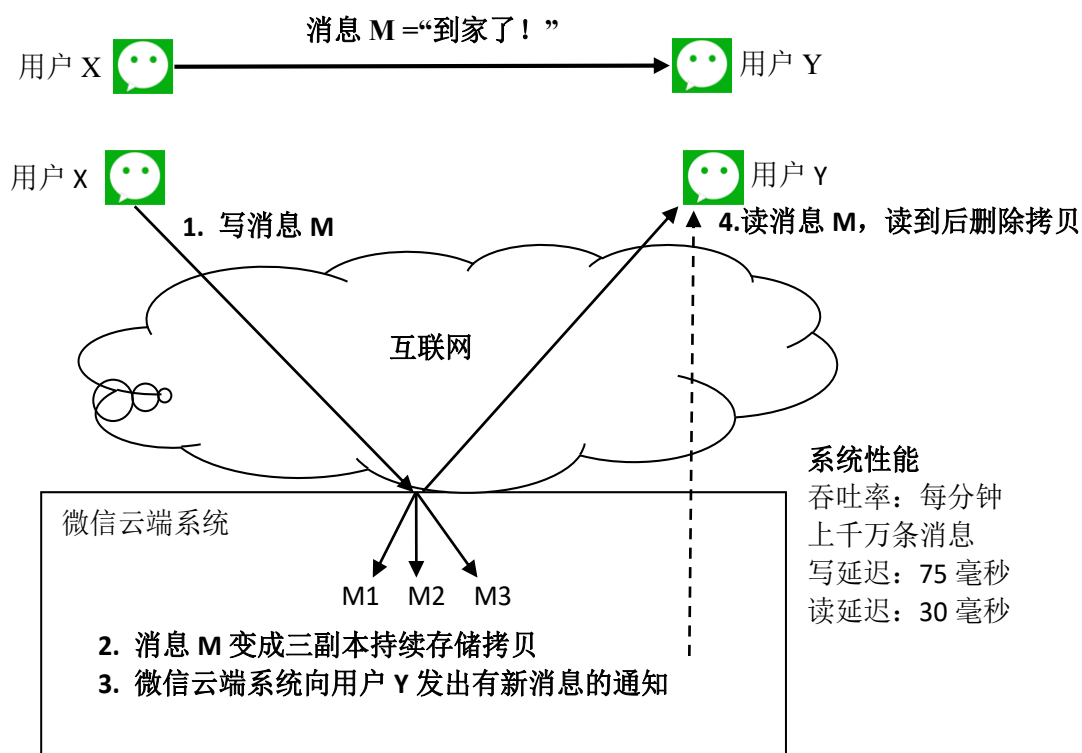
忽略问题

计算过程如何由系统软件 and 系统硬件实现

一个系统思维实例

- 不能靠简单地堆砌20亿亿个晶体管
 - 从多个层次（角度）理解一个系统
 - 每个层次仅考虑该层次特有问題，忽略其他问题
 - 用一套抽象概念和方法统一地处理该层次所有的计算过程，解决这些特有问題

系统层抽象



抽象概念

客户端和云端系统所涉及的硬件、操作系统软件、数据库软件、应用框架软件

计算过程

具体的计算过程

关心问题

让微信系统有效地及时地服务上亿用户，每分钟传递上千万条消息、同时能够保证读/写一条消息的延迟分别控制在**30毫秒**和**75毫秒**，如何控制系统成本

忽略问题

更底层的系统软件与硬件的组成

3. 计算系统思维的要点

- 通过抽象，将模块组合成为系统，无缝执行计算过程
 - 抽象化：一个通用抽象代表多个具体需求
 - 模块化：系统由多个模块组合而成
 - 计算机 = 硬件 + 系统软件 + 应用软件
 - 全系统一致性；信息隐藏原理，接口概念
 - 无缝衔接
 - 避免缝隙：
 - 扬雄周期原理、波斯特尔鲁棒性原理、冯诺依曼穷举原理
 - 重视瓶颈：阿姆达尔定律
 - 系统性能改进受限于系统瓶颈（针对某任务）
 - $\text{加速比} = 1 / ((1-f)/p + f) \rightarrow 1/f$ 当 $p \rightarrow \infty$

3.1 计算系统抽象

- 抽象化：一个通用抽象代表多个具体需求
 - 计算抽象（**abstraction**）：既是计算机科学最重要的方法，也是最重要的产物
 - 一个计算抽象是一个语义精确、格式规范的计算概念
- 硬件实例：一个处理器中的运算器应该如何设计，才能有一个通用处理器？
 - 科学计算、事务处理、文字处理 → 定点部件、浮点部件
 - 新的应用出现怎么办？
 - 图形图像与音视频等多媒体 → 新指令、GPU加速器
 - 人工智能应用兴起 → 寒武纪神经网络处理器、张量处理器（TPU）
- 软件实例：万千应用，万千算法，如何用统一方式执行？

抽象化的要点

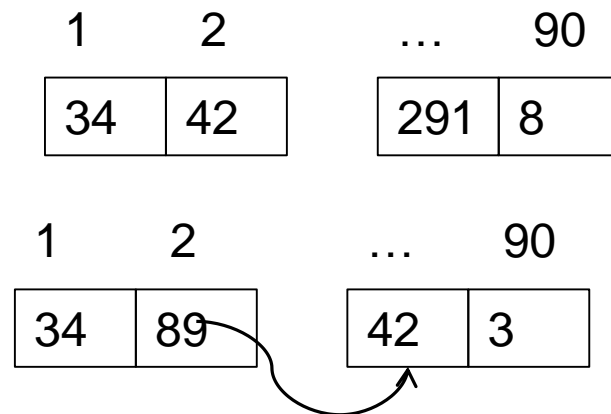
- 一个系统可从多个层次（或多个视野）理解
- 每个层次仅仅考虑有限的、该层次特有的问题
 - 并用一套精确规定的抽象概念和方法，统一处理该层次所有的计算过程，解决这些特有问题
- 抽象三性质：
 - 有限性：每个抽象仅仅考虑一个层次的有限的特有问题，忽略其他层次，忽略同一层次的其他问题
 - 精确性：抽象化的产物是一个计算抽象，它是一个语义精确、格式规范的计算概念
 - 泛化性：一个通用抽象代表多个具体需求
 - 可以触类旁通、用于其他实例（generalization）
 - 只对某个实例有效的抽象，不是好的抽象

计算机科学的抽象化特色

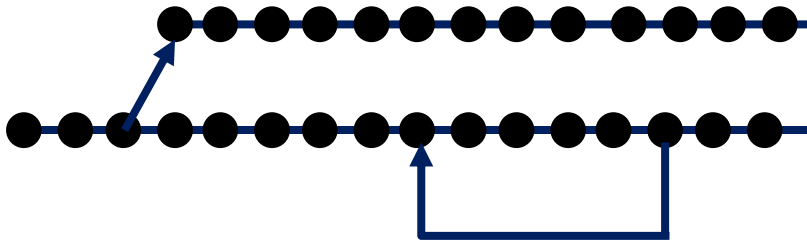
- 强调信息，即数据以及对数据的操作
 - 包括存储操作、运算操作、通信操作
 - 存储操作：将数据存放在某个地方（比如内存），或者从某个地方取出来放在寄存器中。该“地方”的名字称为存放该数据的地址。
 - 运算操作：加、减、乘、除、与、或、非、移位等
 - 通信操作：将数据从一个地方（比如硬盘）传递到另一个地方（比如内存）。数据传递到显示器时，相应的信息就被显示出来了。
 - 数据抽象：一类数据及其操作
 - 控制抽象：控制多个步骤组合实现计算过程
- 强调“比特精准、可自动执行的抽象”

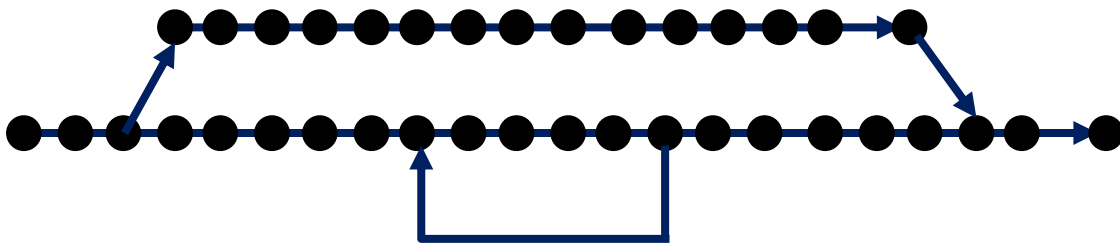
数据抽象（数据类型）

- 比特（bit）：基本二进制数位
- 4比特（hex）：16进制数位
 - 0:0000, 1:0001, 2:0010, ..., 9:1001, A:1010,..., F:1111
- 字节（byte）：8位字节
 - 应用实例：ASCII码、Unicode码
- 字（word）：32/64位的整数、浮点数、字符串
- 指针（pointer）、数组（array）
-
- 文档、文件
- App



符号、操作、步骤、过程

- 一个过程是一个有限的步骤序列
 - 下图包含几个计算过程？
 - 每个步骤包含“1个操作+1个下一步骤”
 - 下一步骤可是顺序或跳转
 - 操作可是：
 - 算术逻辑运算
 - 访存操作
 - 跳转操作
 - 输入输出操作
 - 它们都是数字符号变换操作
- 

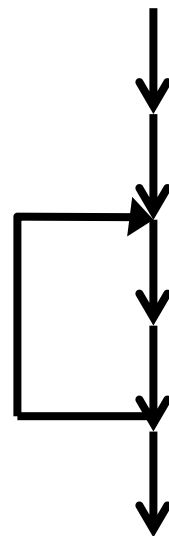


计算系统的环节

- 任何计算系统都有处理、存储、互连通信模块
- 任何模块抽象都要考虑数据、操作、控制三方面
 - 例子：如何应对输入输出（I/O）设备的多样性？
 - 鼠标键盘显示器,音箱麦克风,硬盘,打印机,网络WiFi
 - 回答：发明系统抽象
 - 操作系统
 - 驱动程序：字符设备、块设备
 - 文件概念：open, close, read, write; ioctl
 - 互连通信硬件接口
 - 串口、并口
 - PCI总线
 - USB总线

操作 与控制

- 算术逻辑运算： $+$, $-$, \times , \div ; \wedge , \vee , \neg ; 移位
- 控制操作：跳转、条件跳转、同步、异常处理
- 存储操作：读、写、打开、关闭
- 元数据操作：创建、删除、访问控制
- 连接通信操作：网络部分
- 高级操作抽象（往往是）低级原语的组合（算法）
 - 高级操作有自身的接口
 - 例子：发一条微信、拷一个图片 **vs.** 加密一个文件



ASCII字符编码

语义精确、格式规范

$D_6D_5D_4 \backslash D_3D_2D_1D_0$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	,	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

数据的地址、内容、格式、语义

假设一段内存内容如右所示

- 常见地址: byte address, word address
- 内容: 黄框里的值 (十进制)
- 格式与语义

- ASCII字符串

- 122 120 117 64 105 99 116 46 97 99 46 99 110 13 13 13
z x u @ i c t . a c . c n CR CR CR
- zxu@ict.ac.cn (后面填充了三个回车符)

- 64位整数

- 8028048
6908526
6383171
7212509

8028048	0
6908526	1
6383171	2
7212509	3



字节地址		64位地址
0	122	0
1	120	
2	117	
3	64	
4	105	1
5	99	
6	116	
7	46	
8	97	2
9	99	
10	46	
11	99	
12	110	3
13	13	
14	13	
15	13	

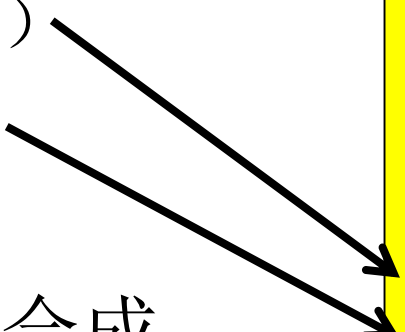
ASCII和Unicode都是数据抽象

它们如何体现抽象三性质

- 有限性
 - 解决“用数字符号表示世界上各种语言字符”的问题
 - 忽略了字体（是宋体、隶书还是黑体）、大小（是小五还是四号字体）、如何对齐、如何具体显示打印等等问题
- 精确性
 - 是语义精确、格式规范的计算概念，没有歧义
 - “中” “国” 对应于0x4E2D和0x56FD
 - 格式规范，每个字符存放在相邻两个字节中
- 通用性（泛化性）；发明时微信尚未出现
 - 不针对某个计算机、某个软件、或某个应用场景
 - 不论何时何地，不管是什么电脑、使用什么操作系统和应用软件、处于何种应用场景，用统一的方法解决问题

控制抽象

- 三种常见的控制抽象（顺序、条件跳转、调用）
 - **顺序**（sequential order, 或sequence）是最基本、最常用的、缺省的控制抽象
 - **跳转**（条件跳转）
 - **调用**一个子程序
- 这三种抽象可组合成更高级的控制抽象
 - 步骤2、3、4、5组合成新的步骤2（**循环**抽象）



步骤0: 开始计算过程
步骤1: Sum=0
步骤2: i=0
步骤3: Sum = A[i] + B[i] + Sum
步骤4: i = i + 1
步骤5: 如果i<30亿, 跳转到步骤3
步骤6: 打印出结果Sum
步骤7: 终止计算过程

步骤0: 开始计算过程
步骤1: Sum=0
步骤2: for (i=0; i<30亿; i++) Sum = A[i] + B[i] + Sum;
步骤3: 打印出结果Sum
步骤4: 终止计算过程

3.2 模块化：计算系统与计算模块

- 模块化：系统由多个模块组合而成
 - 计算机 = 硬件 + 系统软件 + 应用软件
 - 计算机硬件 = 处理器 + 存储器 + 输入输出（I/O）设备
 - 处理器 = 运算器 + 控制器 + 寄存器 + 数据通路 + ...
 -
 - 最终到达计算机的基本操作
- 全系统一致性，不能相互矛盾
 - 不同的表示需要自动的翻译、解析
- 信息隐藏原理
 - 规范（specification）与实现（implementation）分离
 - 接口概念
 - 只能看到和使用模块接口，看不到模块内部

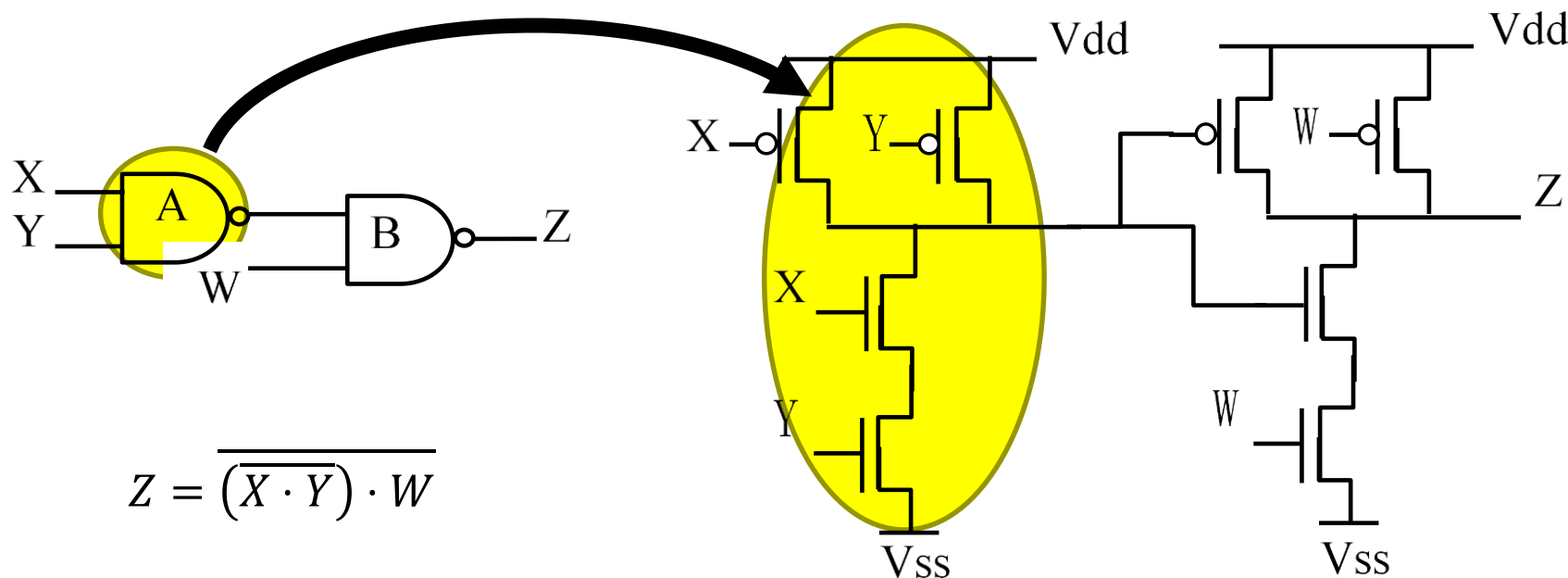
信息隐藏原理

- 精心定义模块的接口，将外界调用模块所需要的信息放在模块接口处，将外界调用模块不需要的信息放在模块内部隐藏起来
 - 隐藏内部信息 (**information hiding**)
 - 每个模块仅暴露其接口 (**interface**) 以及通过接口可见的外部行为，隐藏所有内部细节行为和内部信息
 - 区分规范与实现 (**separation of specification and implementation**)
 - 精确给出每个模块的规范 (**specification**)，即其接口和外部行为规定，独立于内部实现 (**implementation**)
 - 抽象并重用模块
 - 给出每种模块的抽象化描述，给予每种模块抽象特定的命名指称，并尽量重用模块的抽象

信息隐藏原理 实例

- 图中三者表达同样的逻辑电路
- 但抽象不同，暴露的信息不同

W	X	Y	Z
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



接口：逻辑值+逻辑操作


电压值+晶体管操作

用真值表定义基本逻辑运算门

- 基本逻辑运算

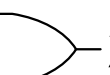
- 与:

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

 X, Y inputs, Z output


- 或:

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

 X, Y inputs, Z output


- 非:

X	Z
0	1
1	0

 X input, Z output

- 异或:

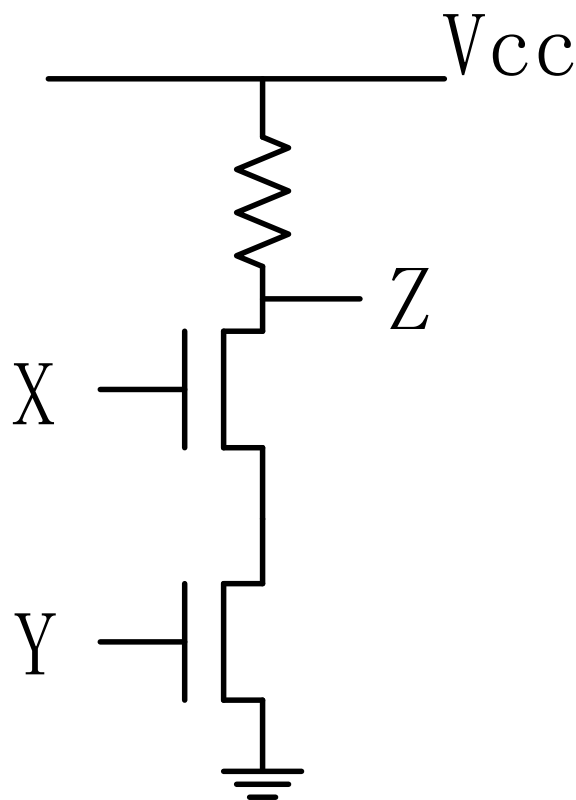
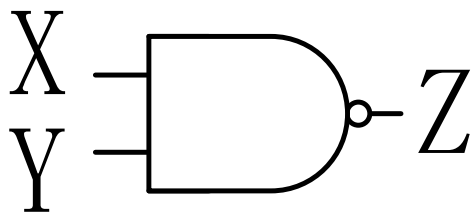
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

 X, Y inputs, Z output

- 基本逻辑运算的组合可产生任意组合逻辑表达式
- 真值表可定义任意组合逻辑表达式

用晶体管电路实现基本逻辑运算 与非门例子

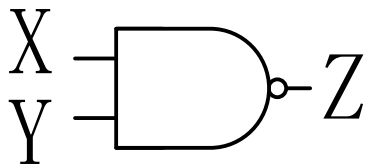
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0



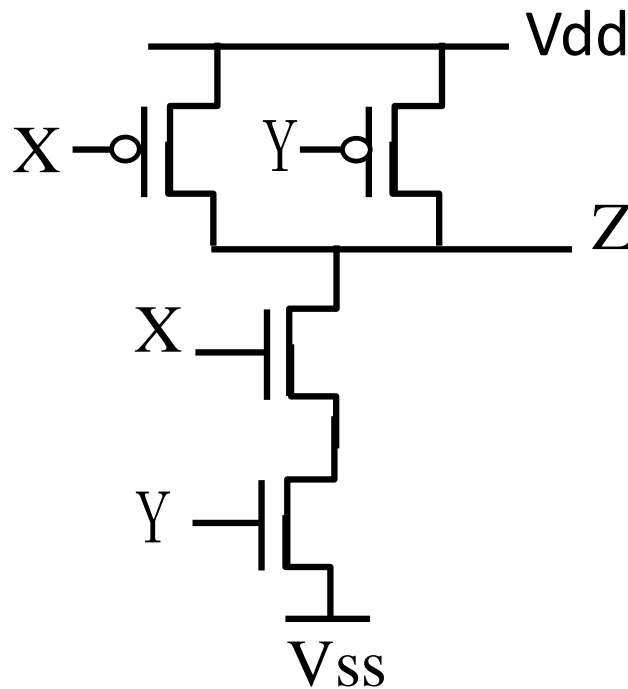
用晶体管电路实现基本逻辑运算 与非门例子

与非门

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0



CMOS 电路



任意组合电路 都可以通过组合基本门电路实现

- 理论依据
 - 第二章的析取范式、合取范式定理

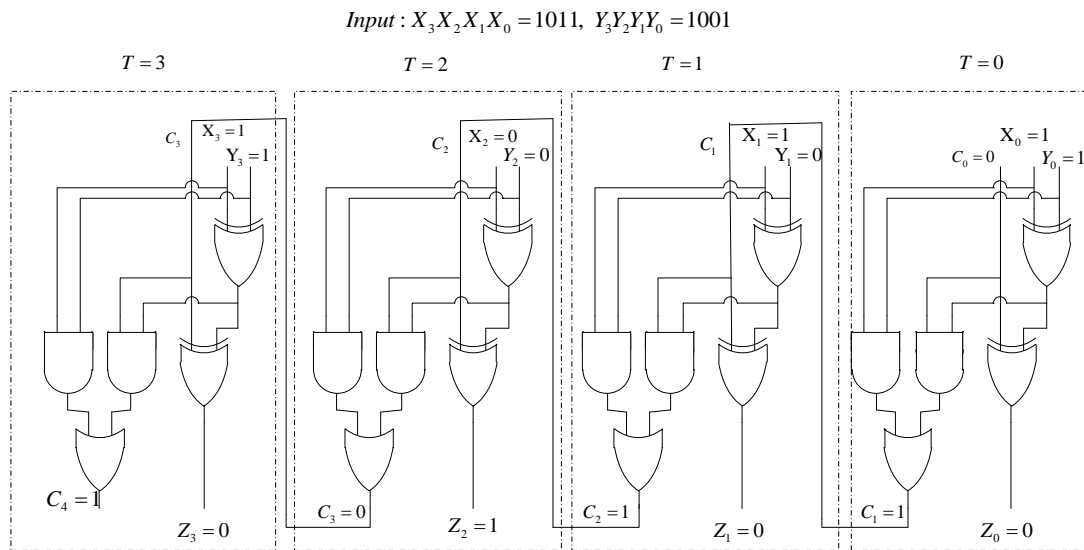
逻辑门名称	两个输入变量表达式	n 个输入变量表达式
与门	$Z = X \cdot Y$	$y = x_1 \cdot x_2 \cdots x_n$
或门	$Z = X + Y$	$y = x_1 + x_2 + \cdots + x_n$
异或门	$Z = X \oplus Y$	$y = x_1 \oplus x_2 \oplus \cdots \oplus x_n$
与非门	$Z = \overline{X \cdot Y}$	$y = \overline{x_1 \cdot x_2 \cdots x_n}$

二进制加法例子: $11+9=20 \rightarrow 1011+1001=10100$

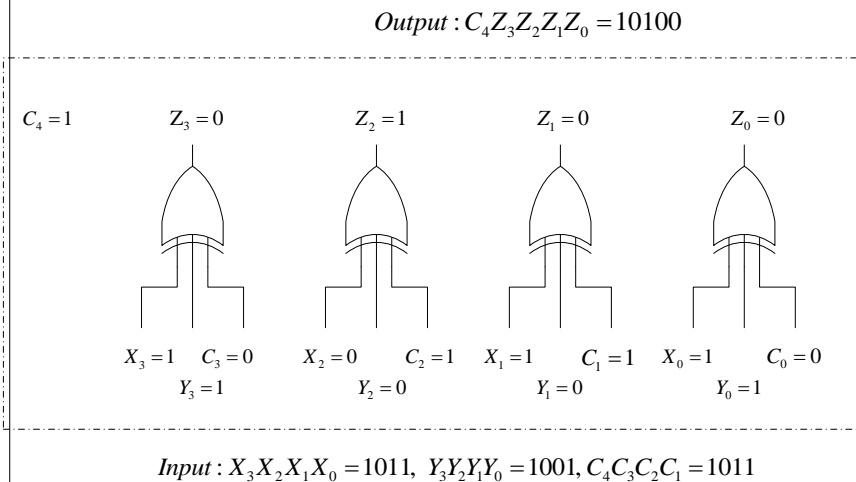
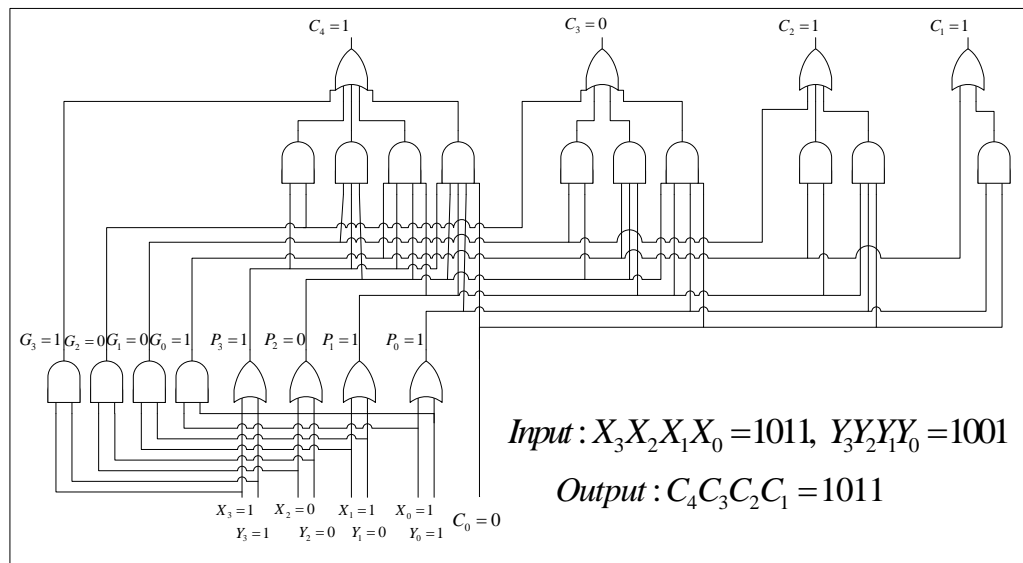
$X_3X_2X_1X_0$		1	0	1	1
$Y_3Y_2Y_1Y_0$	+	1	0	0	1
		<hr/>			
C_1				1	<u>0</u>
		<hr/>			
C_2			1	<u>0</u>	
		<hr/>			
C_3		0	<u>1</u>		
		<hr/>			
C_4		<u>1</u>	<u>0</u>		
		<hr/>			
$Z_3Z_2Z_1Z_0$		1	0	1	0
				0	

二进制加法实现: $11+9=20 \rightarrow 1011+1001=10100$

- 全加器+波纹进位
- $3*4 = 12$ 门延迟
- 优化加法器
- $3+1=4$ 门延迟

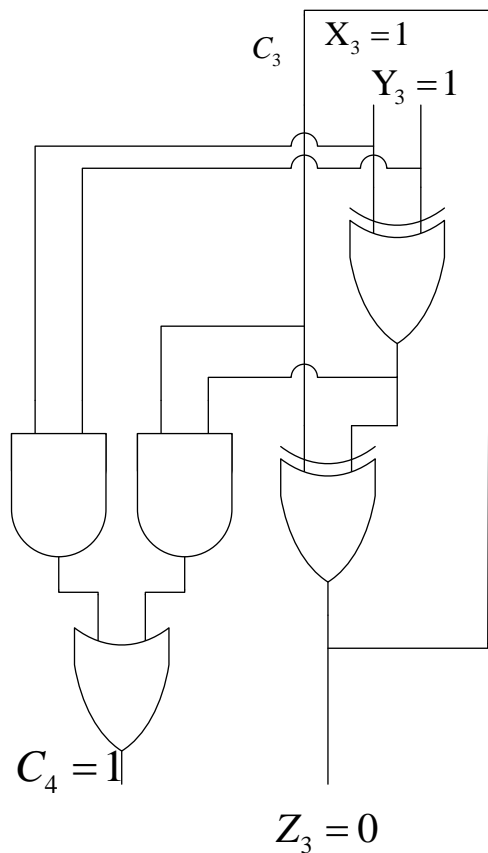


Output: $C_4Z_3Z_2Z_1Z_0 = 10100$

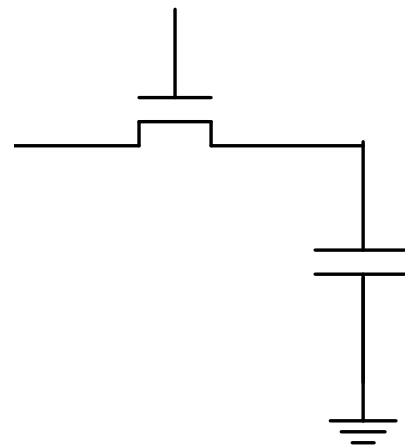


包含状态的电路称为时序逻辑电路 (如何表示状态?)

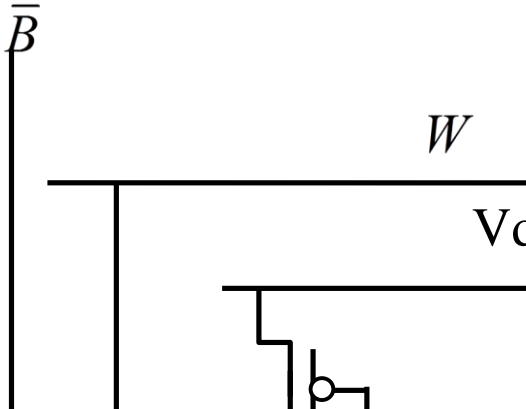
- 有回路的电路

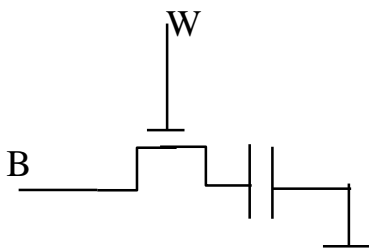
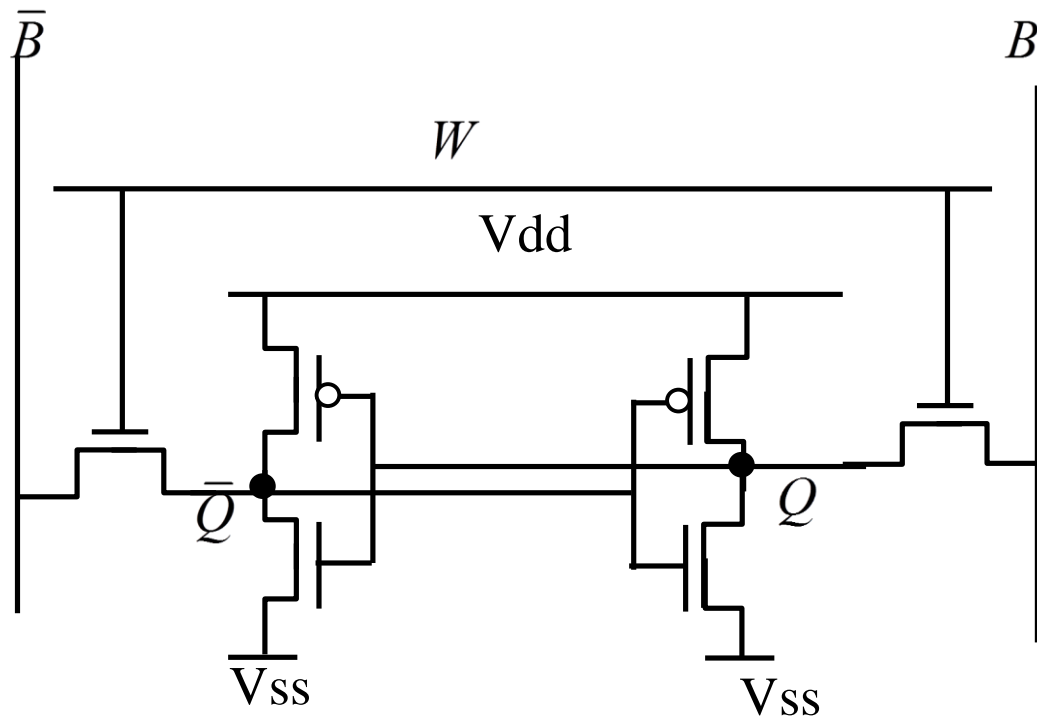


- 具有存储信息功能
物理器件 (如电容)
- DRAM 单元



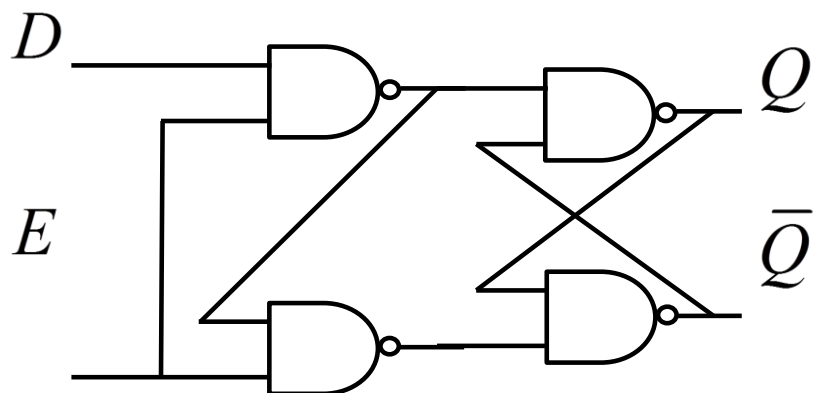
静态存储器单元 SRAM单元

- 用六个晶体管组成1比特存储单元
 - 为什么叫静态
 - 三种存储器
 - DRAM
 - SRAM
 - NVM
- 



D-触发器

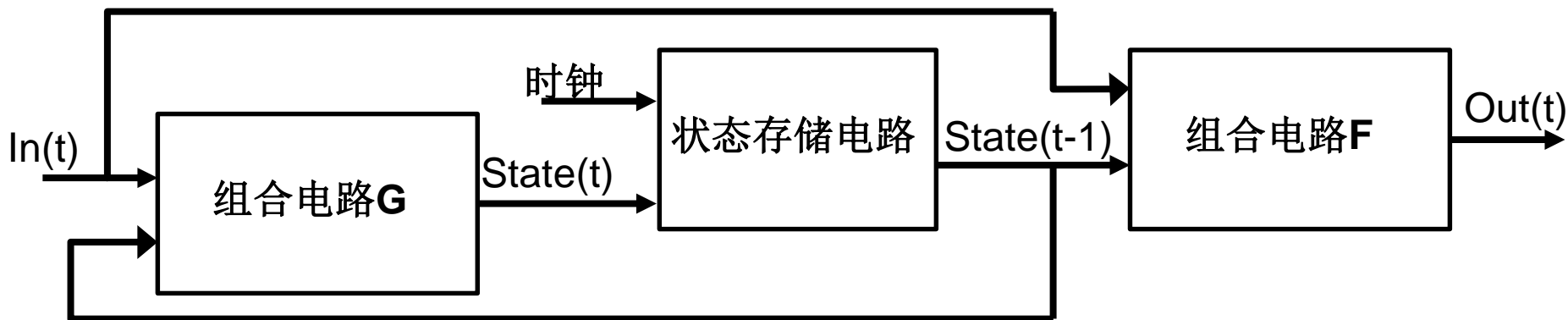
- Delay Flip-Flop



E	D	Q	Q_{next}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

组合电路与状态电路产生自动机 即时钟同步的时序电路

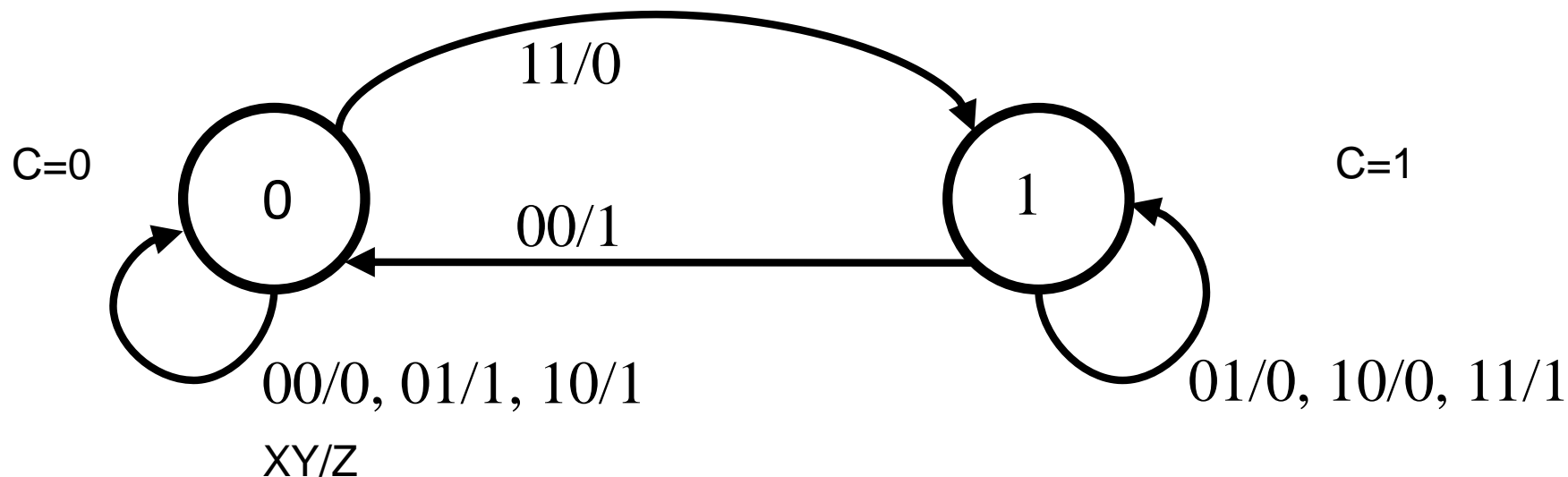
- 第 t 时刻的输出是 t 时刻输入与 $t-1$ 时刻状态的函数
- 第 t 时刻的状态是 t 时刻输入与 $t-1$ 时刻状态的函数
 - $\text{Out}(t) = F(\text{In}(t), \text{State}(t-1))$
 - $\text{State}(t) = G(\text{In}(t), \text{State}(t-1))$



自动机实现4位串行加法过程

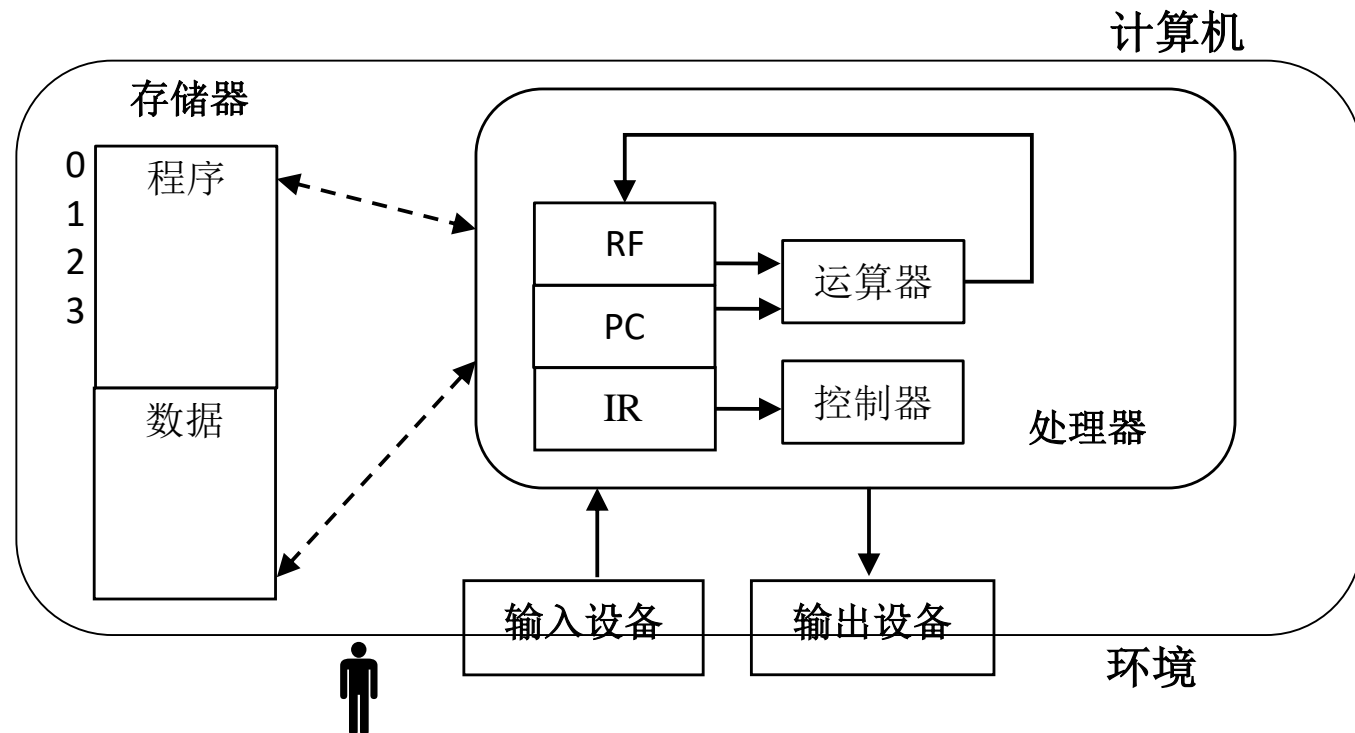
● $1011+1001 = 10100$

- $t=0$ 的初始状态: $C=0$ 。自动机处于左边状态。
- $t=1$: $X=1, Y=1$; 有向边11/0适用, 自动机转移到右边状态, 输出 $Z=0$ 。
- $t=2$: $X=1, Y=0$; 有向边10/0适用, 自动机保持在右边状态, 输出 $Z=0$ 。
- $t=3$: $X=0, Y=0$; 有向边00/1适用, 自动机转移到左边状态, 输出 $Z=1$ 。
- $t=4$: $X=1, Y=1$; 有向边11/0适用, 自动机转移到右边状态, 输出 $Z=0$ 。



存储程序计算机（冯诺依曼模型）

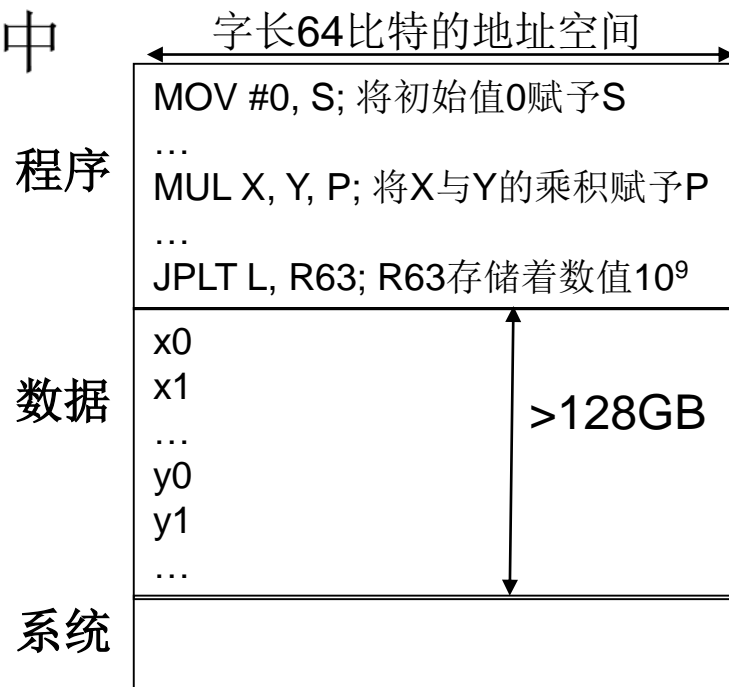
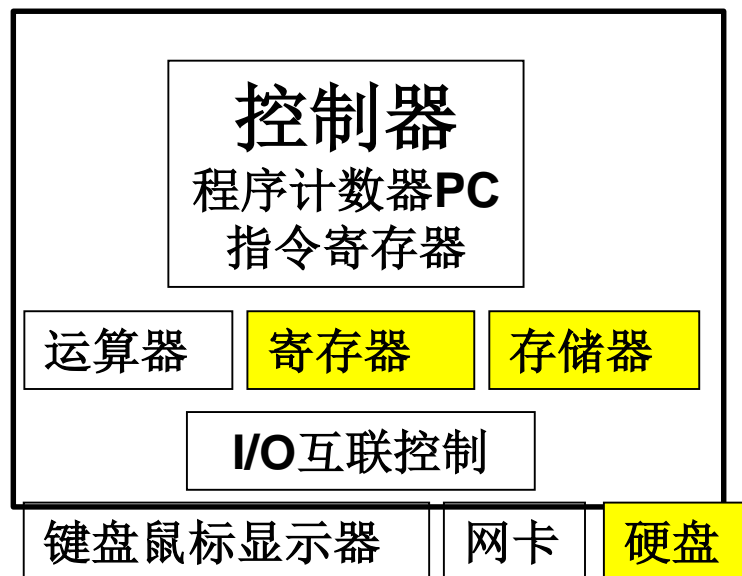
- 二进制数据及其算术逻辑操作
- 计算机 = 处理器 + 存储器 + 输入输出设备
 - 处理器 = 运算器 + 控制器 + 寄存器
 - 冯诺依曼瓶颈（von Neumann bottleneck）
- 存储程序计算机（stored program computer）
- 串行执行



软件与硬件的配合

- 开机序列干什么？
- 如何求10亿对实数的内积？
 - $S = \sum_{k=0}^{n-1} (x_k \times y_k), n = 10^9$
- 算法与程序
 - 假设程序与输入数据已在存储器中

```
      S = 0; k = 0
L:    P = x[k] × y[k]
      S = S + P
      k = k + 1
      Jump L if k < 109
      Print S
```



软件

- 基础软件

- 固件（**firmware**）：实现最基本的功能，通常“固化”在只读存储器芯片中
- 系统软件：操作系统、编译器
- 中间件：数据库软件、万维网服务器

- 应用软件

- 办公软件、通信软件、行业软件、游戏软件、电子商务软件、上网软件等

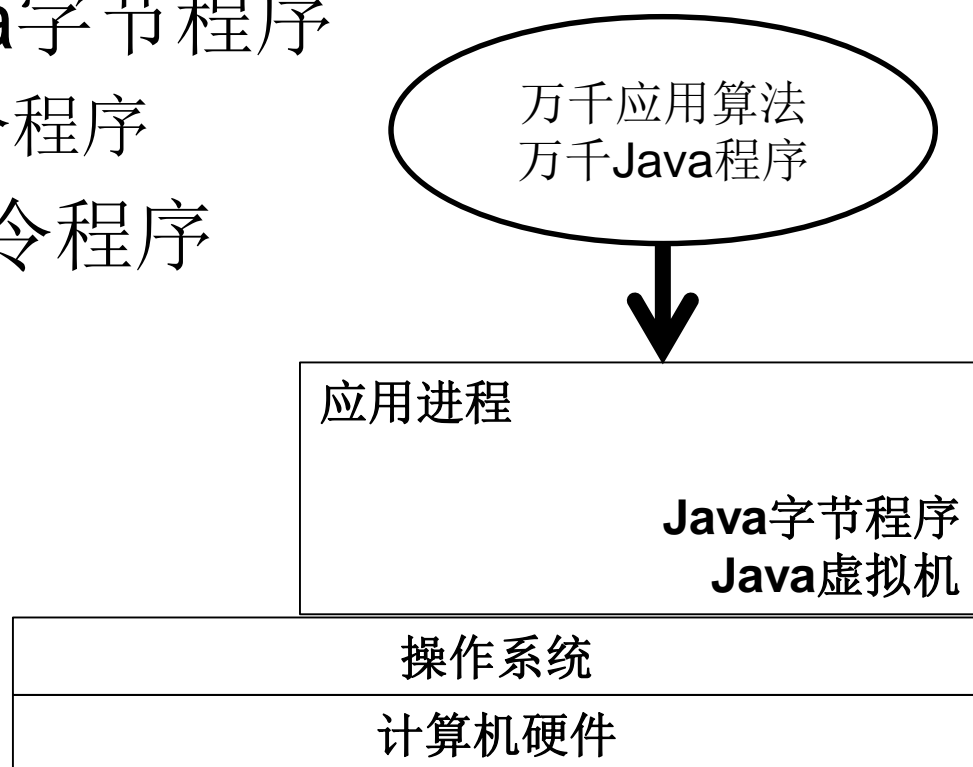
- 二进制码：机器语言程序

- 源码：汇编语言或高级语言程序

应用软件
中间件
系统软件
固件
电脑硬件

一套方法支持万千应用： 高级语言应用程序的执行

- 程序员：应用算法→Java应用程序
- 编译器：Java应用程序→Java字节程序（bytecode）
- Java虚拟机：解析 Java字节程序
 - 解析的结果是二进制指令程序
- 计算机：执行二进制指令程序
- 操作系统控制应用程序
- 万千应用，一套方法



3.3 自动执行与无缝衔接（无缝级联）

- 自动执行高阶：无缝衔接
 - 扬雄周期原理
 - 波斯特尔鲁棒性原理
 - 冯诺依曼穷举原理
 - 阿姆达尔定律
- 自动执行难题尚未完全解决
 - 基本解决了单机自动执行难题
 - 在多台计算机上执行的计算过程如何自动执行？
 - 在人机物三元计算的万物互联网时代如何自动执行？
 - 网络思维（名字空间、拓扑、协议栈）
 - IEEE CS 2022报告：无缝智能

无缝智能尚未实现

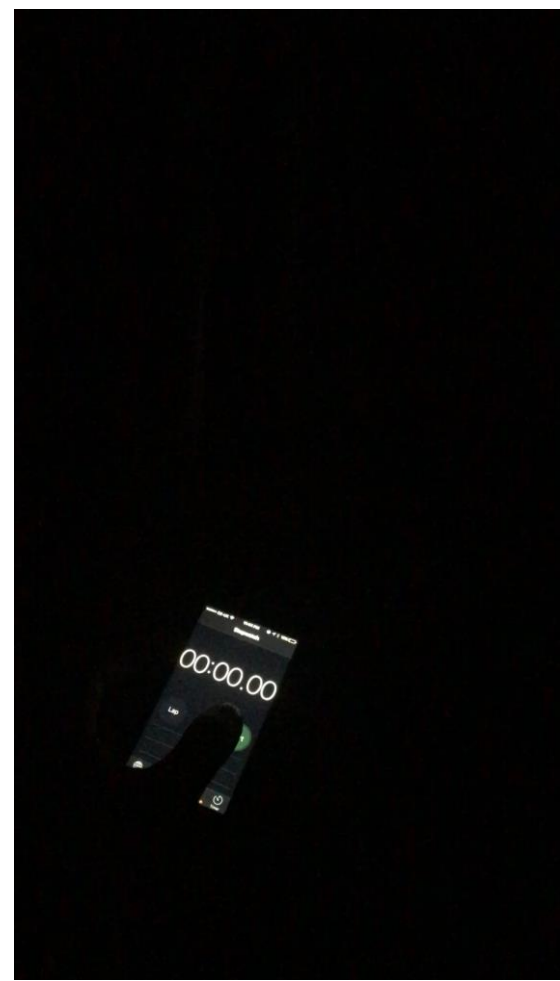
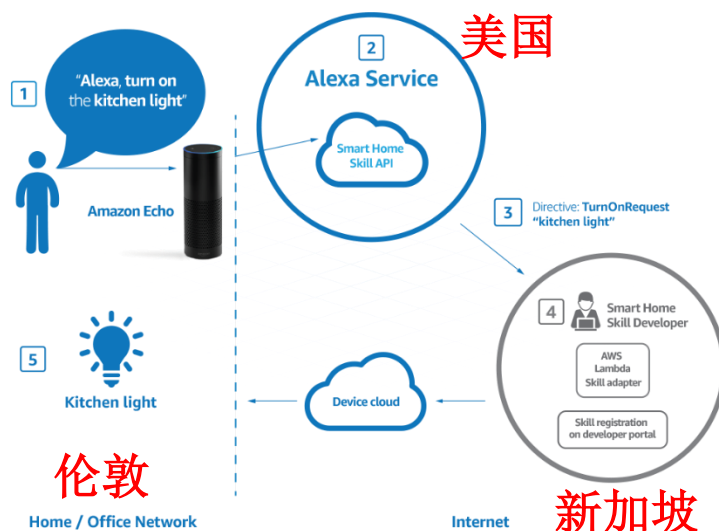
- 不只是简单的数值计算，而是智能计算
 - 计算机能够：理解、感知、认知，甚至呈现情感、美感、意识
- 无缝：用户体验流畅

案例：执行“请开灯”命令（2017.4.13实验）

世界领先的Amazon Echo 物端系统
响应延迟：3-5秒

心理学准则：
用户体验到无缝智能
→ 响应延迟 < 0.12秒

中科院计算所
物端系统研究目标
响应延迟 < 0.1秒



扬雄周期原理

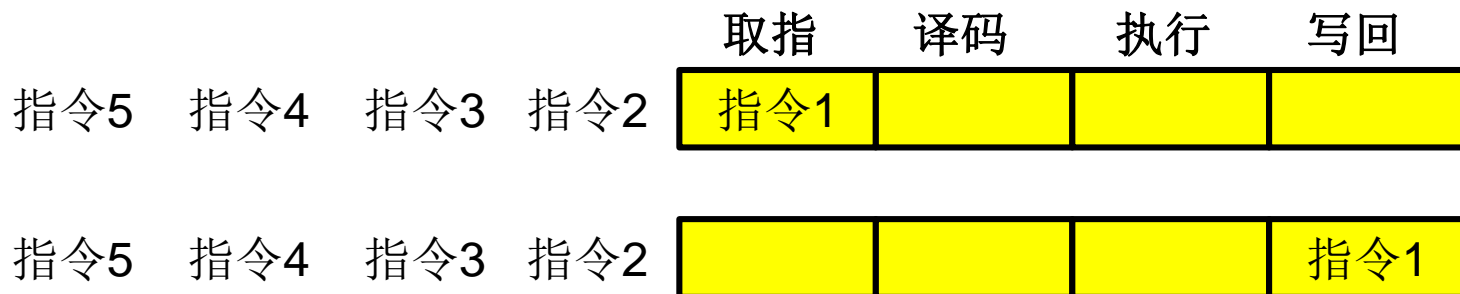
优于尼采的永恒轮回：die ewige Wiederkunft

- 汉代扬雄所著的《太玄经》
 - 《太玄经•周首》：“阳气周神而反乎始，物继其汇”
 - 宋代司马光诠释道：“岁功既毕，神化既周”
- 扬雄周期原理体现组合性（composability）方法
 - 执行完一个XX周期，周而复始执行下一个XX周期
 - 计算过程由程序周期组合而成
 - 程序周期由指令周期组合而成
 - 指令周期由时钟周期组合而成
 - 一个时钟周期的操作对应于一个自动机的变换
 - XX周期具备可数个类别
 - 例如，指令周期有内部周期、总线周期

扬雄周期原理实例：指令流水线

- 每条指令的执行都包含四个操作阶段（stage）
 - 取指操作： $IR \leftarrow M(PC)$
 - 译码操作： $Signals = Decode(IR)$
 - 执行操作： $Result \leftarrow Op$, 或 $Address \leftarrow Op$
 - 写回操作：
 - $RF(i) \leftarrow Result$,
 - $RF(i) \leftarrow M(Address)$ 或 $M(Address) \leftarrow RF(i)$
- 执行的同时， $PC \leftarrow PC + 1$ ；周而复始

流水线延时
= 1纳秒
处理器主频
= 1 GHz



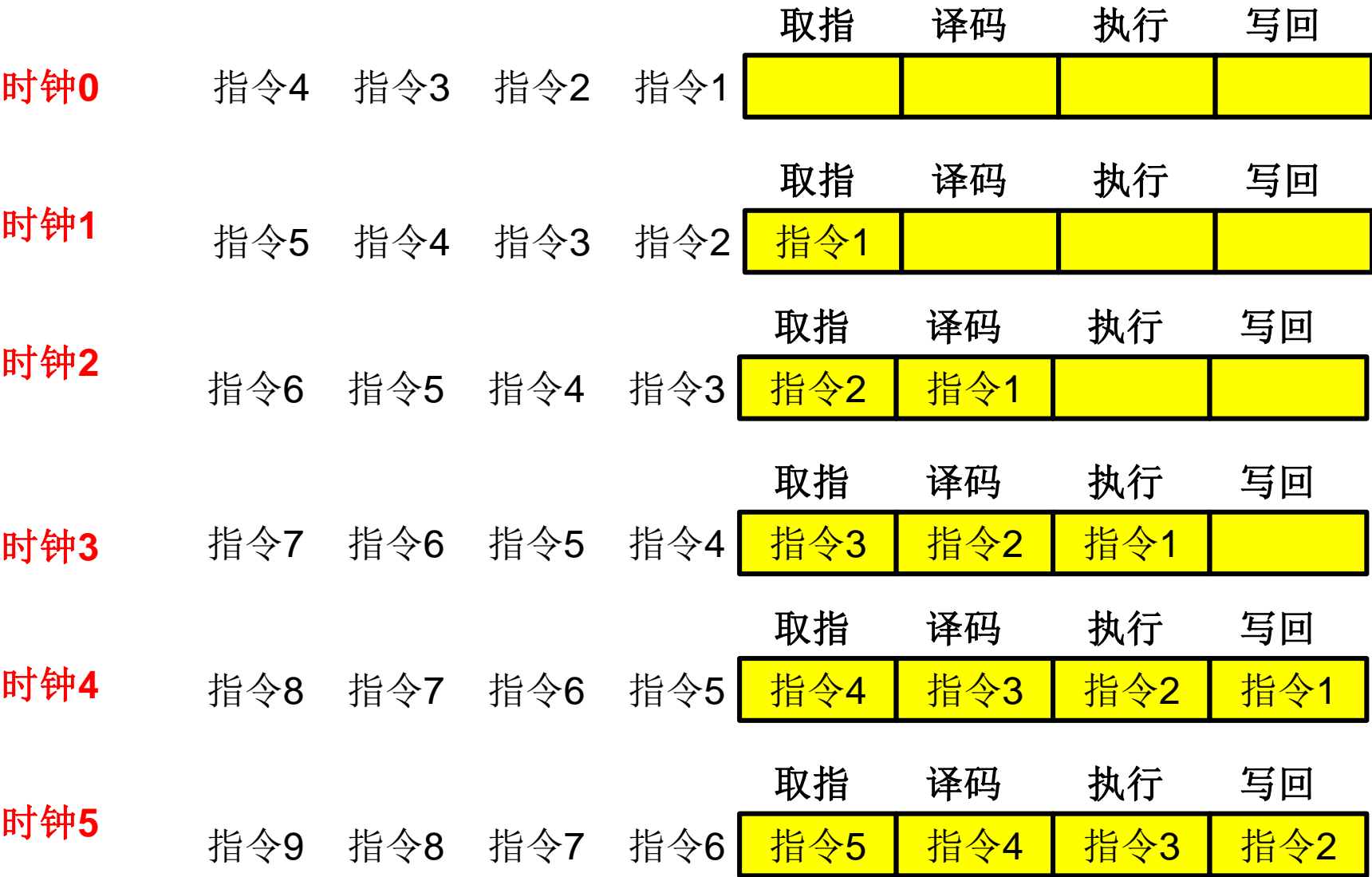
时钟1

时钟4

扬雄周期原理实例：指令流水线

流水线延时= 1纳秒，处理器主频= 4 GHz

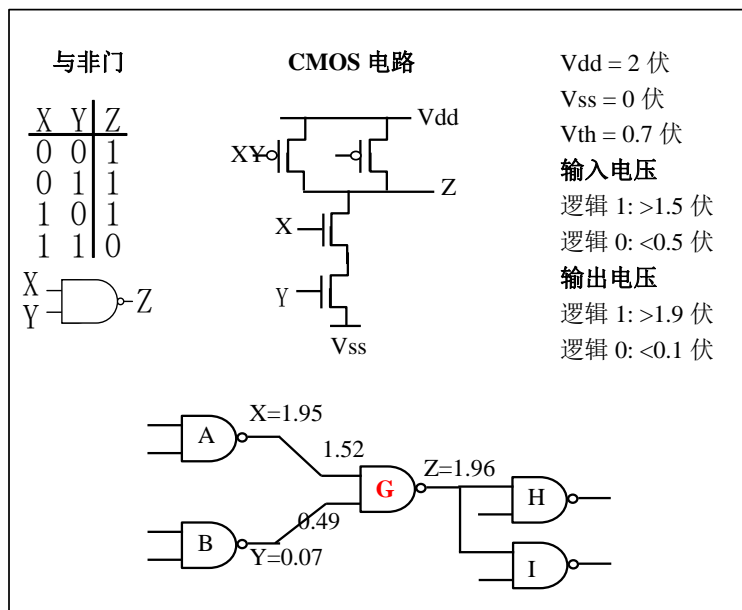
假设数据和指令都在缓存中，速度是4 GOPS（或4 GFLOPS）



宽进严出原理

波斯特尔鲁棒性原理 (Postel's Robustness Principle)

- Jon Postel, RFC761, 1980:
 - be conservative in what you send, be liberal in what you accept
 - be tolerant of input, be strict on output
- 避免误差、漂移、错误的积累；请求不会被“弹回去”



宽进：晶体管的输入电压>1.5伏（而不是>0.7伏）时，对应逻辑1；输入电压<0.5伏（而不是<0.7伏）时，对应逻辑0。高电平和低电平之间至少有1伏的间隔。允许高电平在1.5伏和2伏之间漂移，允许低电平在0.5伏和0伏之间漂移。也就是说，输入端允许大约0.5伏的漂移。

严出：对应逻辑1，晶体管的输出电压>1.9伏（而不是>0.7伏）；对应逻辑0，输出电压<0.1伏（而不是<0.7伏）。高电平和低电平之间至少有1.8伏的间隔。仅允许高电平在1.9伏和2伏之间漂移，仅允许低电平在0.1伏和0伏之间漂移。也就是说，输出端仅允许小于0.1伏的漂移。

冯诺依曼穷举原理

来源： First Draft of a Report on the EDVAC

- 人们必须事先给计算机全面的指示，**绝对穷举所有细节**（“**in absolutely exhaustive detail**”），使得计算机能够自动处理所有情况
 - 这些细节包括：
 - 计算机需要执行的程序的指令、程序的输入数据、程序需要的函数等
 - 其他实现细节包括：
 - 计算机开机后执行的第一条指令
 - 计算机正常执行程序时，下一条指令是什么
 - 执行程序出现异常时，有哪些异常，每种异常如何处理

开机后执行的第一条指令

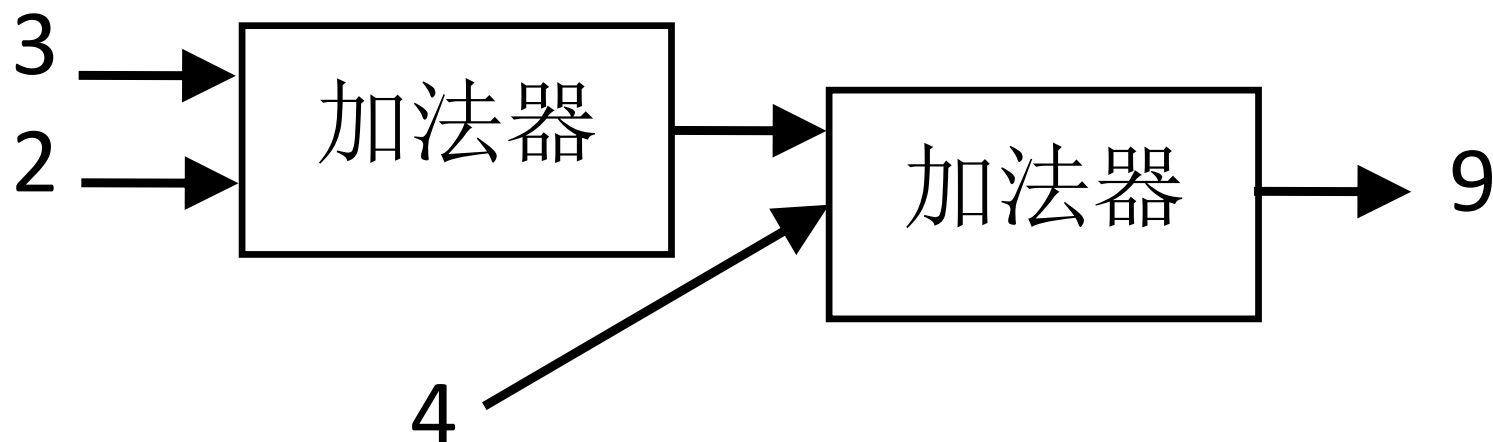
- x86计算机开机后执行的第一条指令
 - 位于地址FFFFFFFF0 (0xFFFFFFFF0)
 - 其内容是一条跳转指令JUMP 000F0000
 - 最底层的一个系统软件（称为BIOS，即Basic Input-Output System）的第一条指令，称为BIOS入口地址
- 龙芯计算机开机后执行的第一条指令
 - 位于地址FFFFFFFFBFC00000
 - 其内容是一条特殊的赋值指令，将处理器的状态寄存器复位（置为零，清零），这也称为初始化处理器的状态寄存器

下一条指令是什么 下一条指令在哪里

- 三种方式
 - 哈佛马克一号（Harvard Mark I）
 - 学名是“Automatic Sequence Controlled Calculator”（自动顺序控制计算机）
 - “直线程序”，没有跳转指令，不支持循环控制
 - ENIAC改造后
 - 在每一条指令的内容中，明确地列出下一条指令的地址
 - 今天的计算机系统
 - 采用“程序计数器”（program counter，即PC）方式

如何求3个数之和？

- 直接用硬件逻辑电路实现
- 步骤可以是硬件



如何求30亿个数之和？

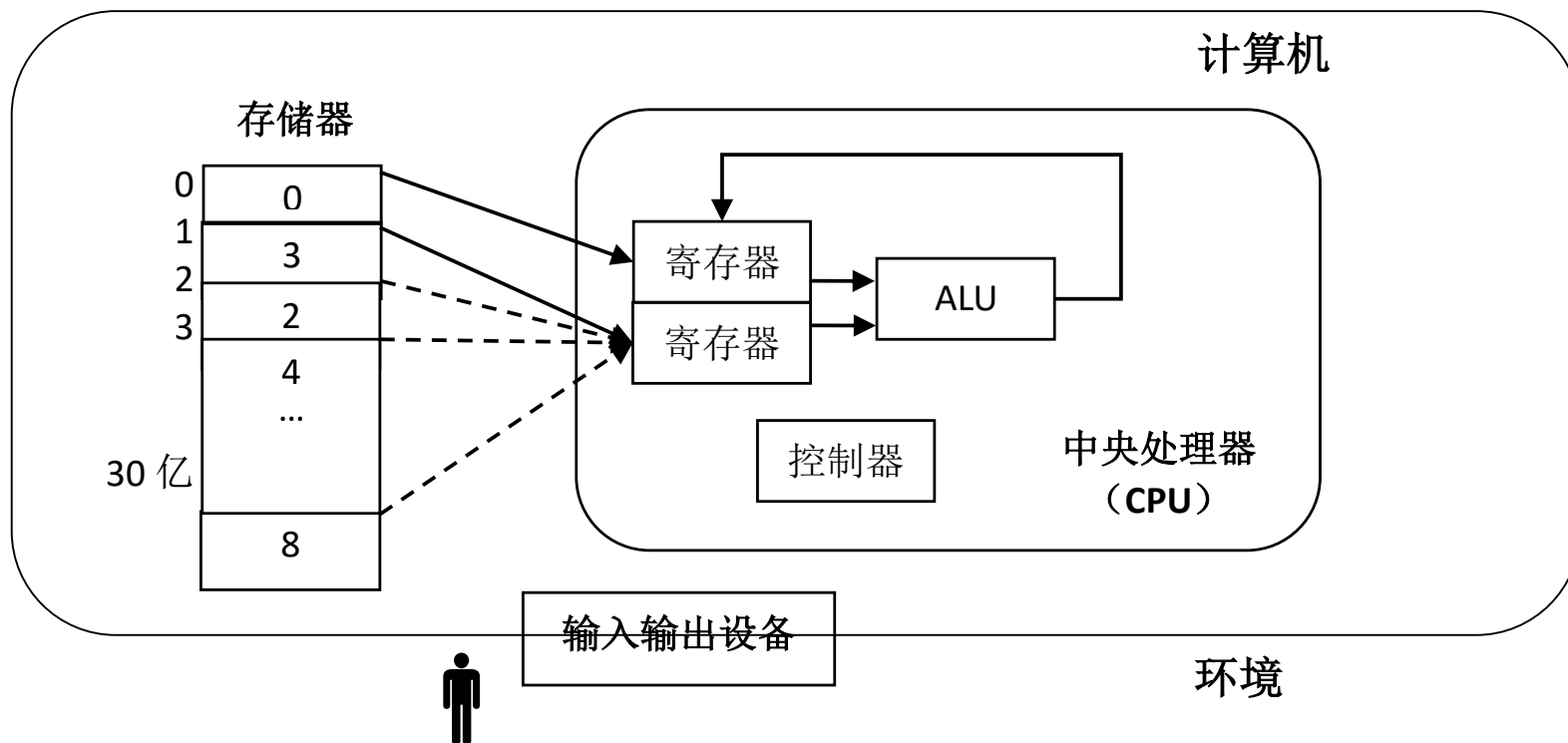
Step 1（第一步骤）：将存储器地址0里的数取到第一个寄存器

Step 2（第二步骤）：将存储器地址1里的数取到第二个寄存器

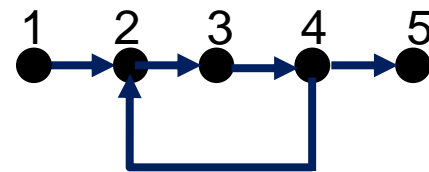
Step 3（第三步骤）：将两个寄存器的数在ALU相加，结果送到第一个寄存器

Step 4（第四步骤）：如果第二步的地址<30亿，将其地址增1，回到第二步骤

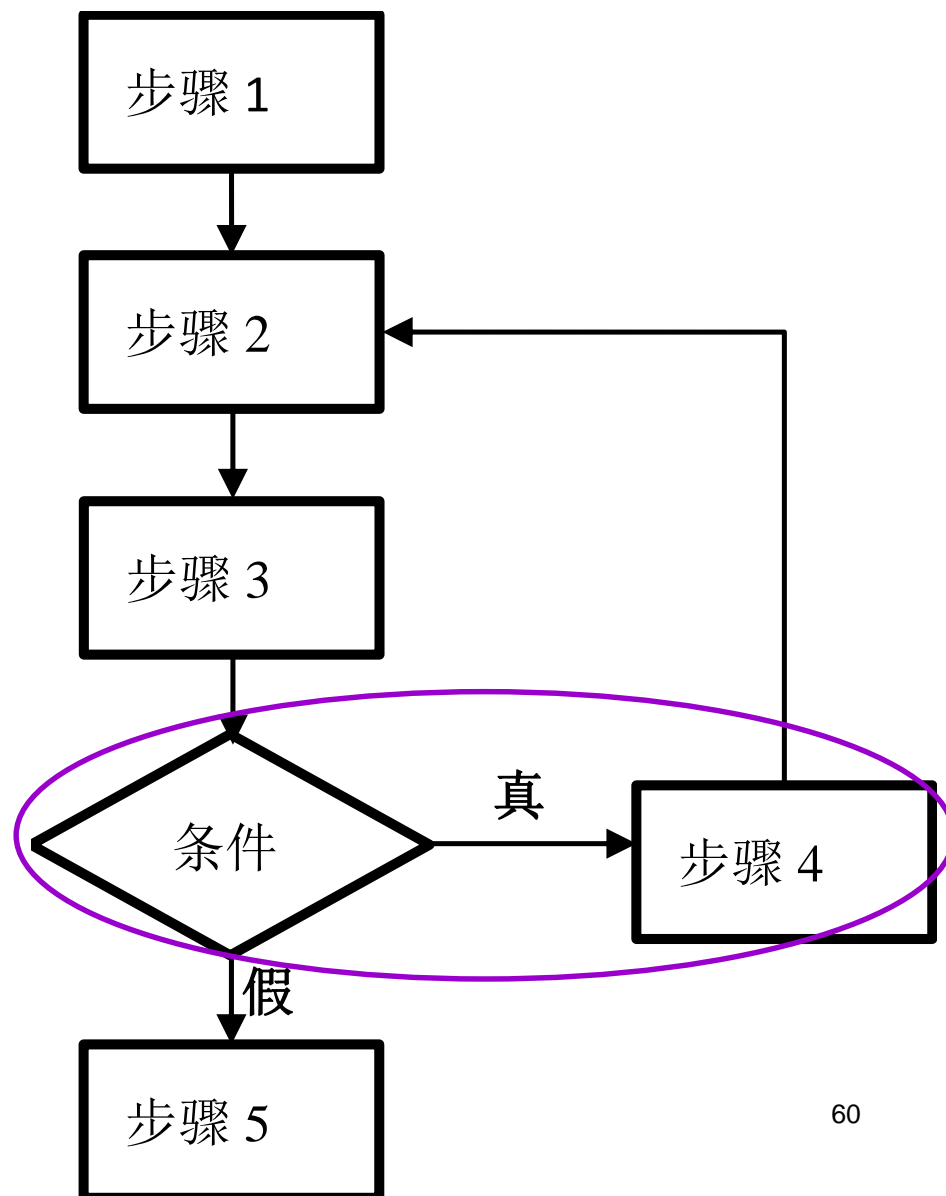
Step 5（第五步骤）：将第一个寄存器的数存入存储器地址0



用流程图描述计算过程



- Step 1（第一步骤）
 - 将存储器地址0里的数取到第一个寄存器
- Step 2（第二步骤）
 - 将存储器地址1里的数取到第二个寄存器
- Step 3（第三步骤）
 - 将两个寄存器的数送到ALU相加，结果送到第一个寄存器
- Step 4（第四步骤）
 - 如果第二步的地址 < 30 亿，将其地址增1，回到第二步骤
- Step 5（第五步骤）
 - 将第一个寄存器的数存入存储器地址0



计算过程自动执行的要点：PC机制

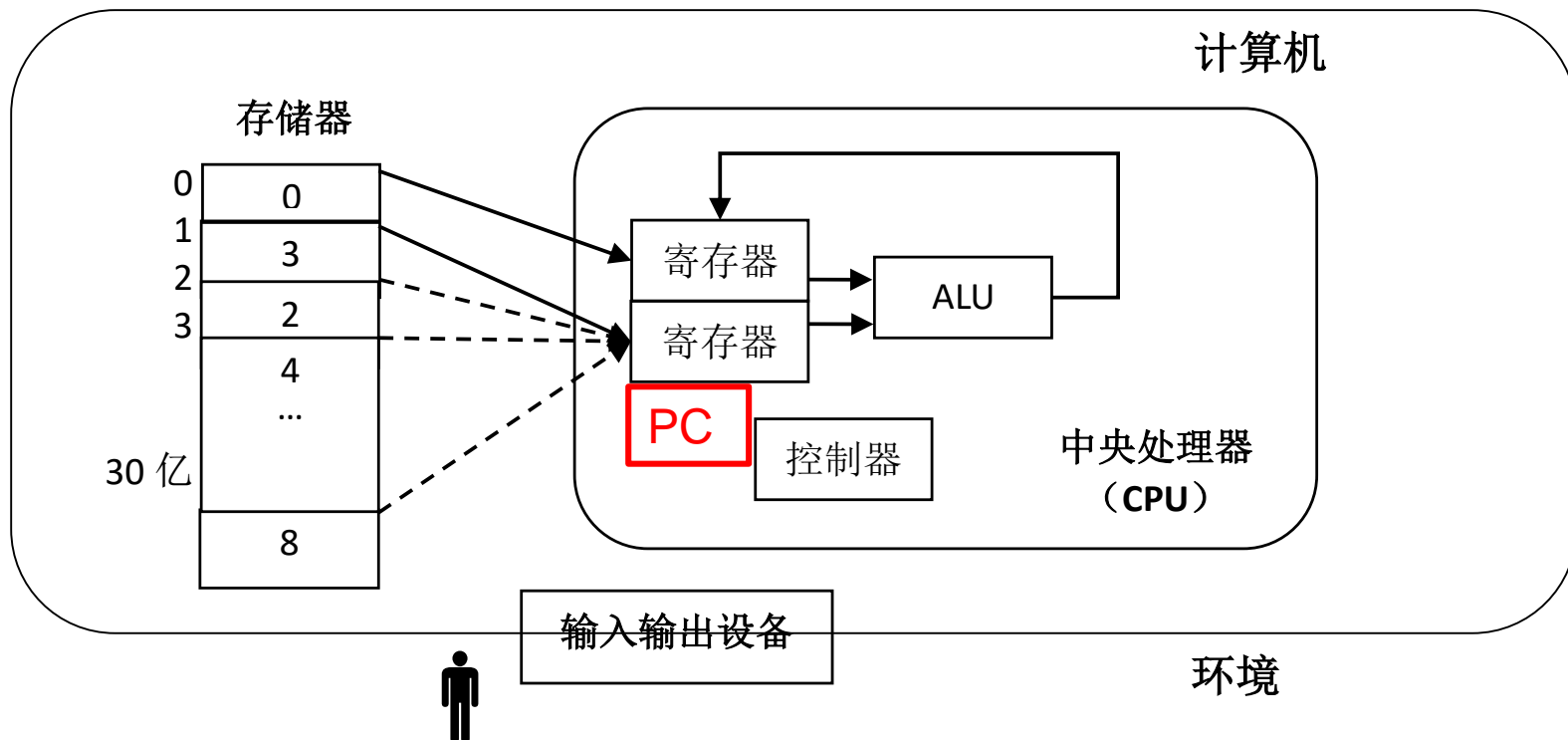
Step 1（第一步骤）：将存储器地址0里的数取到第一个寄存器

Step 2（第二步骤）：将存储器地址1里的数取到第二个寄存器

Step 3（第三步骤）：将两个寄存器的数在ALU相加，结果送到第一个寄存器

Step 4（第四步骤）：如果第二步的地址<30亿，将其地址增1，回到第二步骤

Step 5（第五步骤）：将第一个寄存器的数存入存储器地址0



PC =

Step 2

Step 3

Step 4

Step 2

Step 3

Step 4

...

Step 2

Step 3

Step 4

Step 5

异常处理

- 中断
 - 执行完毕当前指令，然后执行异常处理程序
- 硬件出错
 - 立即执行一个事先设计好的异常处理程序
- 保底异常
 - 为了做到穷举，计算机一般会设计一个保底异常（通常被称为**machine check**），以覆盖其他规定的异常没有覆盖的情况

重视瓶颈：阿姆达尔定律

- 系统性能改进受限于系统瓶颈
 - 加速比 = $1 / ((1-f)/p + f) \rightarrow 1/f$ 当 $p \rightarrow \infty$
 - “假如一个系统可以分成两部分X和Y, $X+Y=1$, $0 \leq X \leq 1$, $0 \leq Y \leq 1$ 。Y能够改善（即缩小它的数值），X不能被改善（即X是瓶颈）。那么，系统最多能被改善到 $1/X$ 。”
 - 一台电脑使用了**500 MHz**主频的处理器芯片，假设 $X=Y=0.5$ ，即程序代码只有一半可以随处理器速度的增加而改善
 - 那么，即使我们将处理器的速度提高**1000倍**（提到**500 GHz**）、**1万倍**、甚至无穷大，整个电脑的速度也最多只能变到 $1/0.5=2$ ，即提高一倍

4.系统思维的创新实例

ENIAC

- 1943年初，美国陆军出现弹道计算迫切需求
- 宾州大学“电子计算机”概念、立项、实施
 - 1941年，毛捷利与艾克特开始合作
 - 1942年，毛捷利备忘录：《高速真空管器件的计算用途》
 - 1943年4月2-9日，正式立项
 - 1943年5月31日，ENIAC项目正式启动
 - 1945年11月，ENIAC研制成功
- ENIAC的主要特征
 - 数字计算机：而不是模拟计算机
 - 电子计算机：所有运算操作用电子速度执行
 - 足够的可靠性（用了10年）：18000个真空管的大系统

IBM 360

- 1960年，IBM总裁华森启动战略性业务调整
 - 全球计算机使用量数千台
 - 8+6款晶体管电脑+X款真空管电脑，各不相干
- 1960-1962.12，IBM 360总体方案出炉
 - 通用计算机 + 计算机家族 + 兼容性
 - 计算机体系结构 + 系统实现，两者的分离
 - 1963.1，IBM总部批准实施
- 应对普遍而剧烈的反对意见
 - 市场竞争力：单一系统太冒险、易被复制
 - 改软件、改外设：成本太大、时间太长
 - 解决问题的技术：自动翻译、软件模拟、硬件仿真
- 1964.4，IBM发布S/360，1965年发货

IBM 360经验（如何鼓励创新）

● 三种创新

● 新问题：兼容性问题

- 设计一代具有竞争力的电脑，改变多个互不兼容的产品线带来的软件移植、市场、研究开发、生产、管理的杂乱和低效率状况。

● 新概念：通用性（360）和电脑家族

● 从根本上解决兼容性问题

- 效果（What）：从汇编语言和外部设备的角度看，家族成员都是一样的（相互兼容）
- 措施（How）：家族成员有同样标准的指令系统、地址格式、数据格式和与外部设备的接口

● 新指标：“大内存”（提高100倍）等，而不是“核能”

● 应对致命弱点（knockoff）：动态寻址

● 今天的意义：昆虫纲悖论

个人电脑

- 1973, Xerox PARC Alto
- 1974, MITS Altair 8800
 - 1975, MS BASIC解析器
- 1976, Apple-I, (1979, VisiCalc)
- 1981, IBM PC
 - Intel CPU, MS DOS, 主板总线, BIOS
- 1996, 联想SpeedEasy无跳线主板



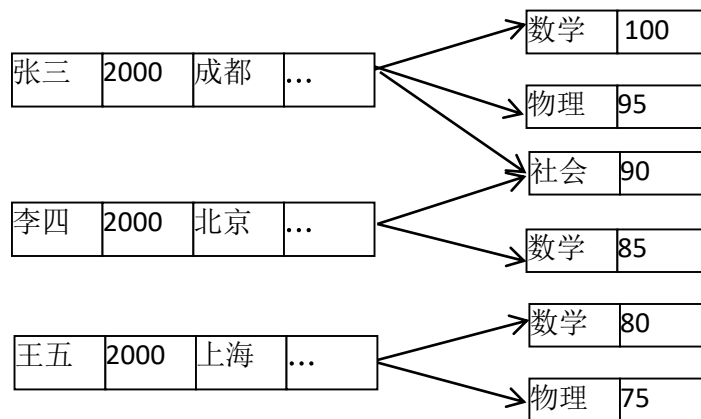
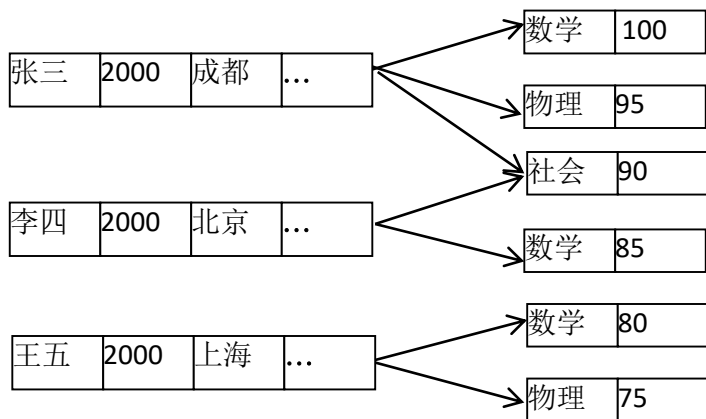
Worldwide PC Vendor Unit Shipment Estimates (Wiki)

Source	Date	Lenovo	HP	Dell	Acer Inc.	Asus	Others
IDC	Q2,2014	19.6%	18.3%	14.0%	8.2%	6.2%	33.6%
Gartner	Q2,2014	19.2%	17.7%	13.3%	7.9%	6.9%	35.0%

学术抽象生命力长久——数据库实例

找出“至少一门功课在90分以上的同学”

● 网络数据库与层次数据库



● 关系数据库（1970年发明）

- 关系表（行、列、键）
- 关系代数、范式

学生姓名	年级	家庭住址
张三	2000	成都
李四	2000	北京
王五	2000	上海

学生姓名	课程	成绩
张三	数学	100
张三	物理	95
张三	社会	90
李四	数学	85
李四	社会	90
王五	数学	80
王五	物理	75

操作系统创新实例

- 1965, MIT Project MAC, Multics
 - Time sharing, file system, process, shell
- 1972, Unix
 - 可移植性: Unix及其软件工具支持很多厂家的计算机
 - 用C语言实现Unix及其软件工具
 - 灵活性: 允许用户和编程人员添加新的软件工具
 - pipe, shell scripts; Keep It Simple, Stupid (KISS)
- 1991, Linux
 - 开放源码的“集中-分散-集中”开发模式
 - 优良的设计原理
 - 实用的原则、有限目标的原则（实用、速度快、可移植）
 - 简洁设计的原则（归纳共性、极小接口、模块）

计算机科学是研究计算过程的科学

十种理解是一个整体：如，抽象是能够构造性自动执行的抽象

理解1：自动执行。计算机能够自动执行由离散步骤组成的计算过程。

自动

理解2：正确性。计算机求解问题的正确性往往可以精确地定义并分析。

理解3：通用性。计算机能够求解任意可计算问题。

通用

理解4：构造性。人们能够构造出聪明的方法让计算机有效地解决问题。

算法

理解5：复杂度。这些聪明的方法（算法）具备时间/空间复杂度。

理解6：连接性。很多问题涉及用户/数据/算法的连接体，而非单体。

联网

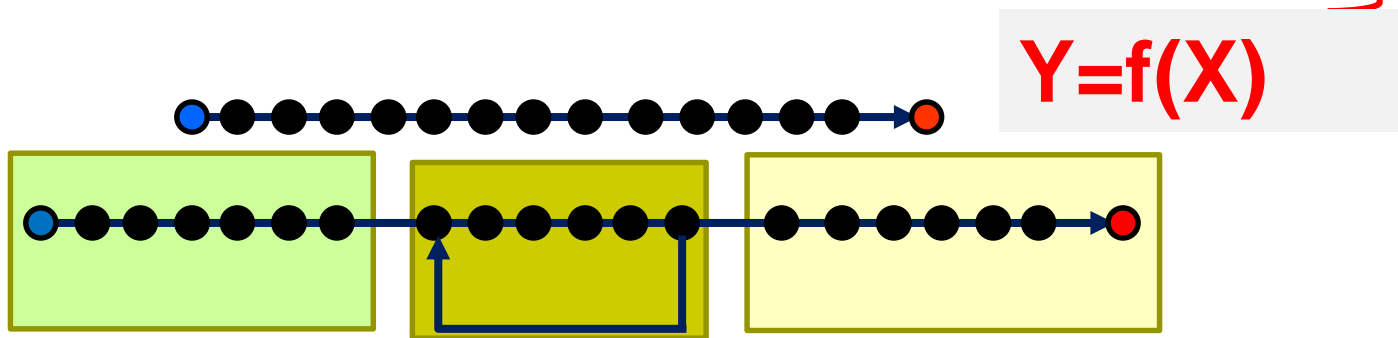
理解7：协议栈。连接体的节点之间通过协议栈通信交互。

理解8：抽象化。少数精心构造的计算抽象可产生万千应用系统。

理解9：模块化。多个模块有规律地组合成为计算系统。

抽象

理解10：无缝衔接。计算过程在计算系统中流畅地执行。



从计算系统思维角度看到的一些 计算机科学基本抽象

- 抽象使得计算机科学成为一门优美的领域
 - 本质：采用一种方法解决该层次的所有问题，应对系统的复杂性，以不变的抽象应系统的万变

数字符号	比特、字节、四进制数、十六进制数、整数、数组、 BMP 图像。	
软件	算法	巧妙的信息变换方法。例如信息隐藏算法。
	程序	算法的代码实现。例如实现信息隐藏算法的 <code>hide.go</code> 程序。
	进程	运行时的程序。例如在 Linux 环境中的 <code>hide</code> 进程。
	指令	程序的最小单位，计算机能够直接执行。
硬件	指令流水线	每条指令都通过“取指-译码-执行-写回”四个操作组成的指令流水线得以自动执行。所有指令都由这一种机制执行，指令流水线由若干时钟周期组成。
	时序电路	等同于 自动机 ，说明每一个时钟周期的操作。时序电路由组合电路与存储单元组成，理论上等同于 图灵机 。
	组合电路	实现二值逻辑表达式（ 布尔逻辑表达式 ）。

谢谢 Thank You

Q&A

zxu@ict.ac.cn



中国科学院
INSTITUTE OF COMPUTING TECHNOLOGY