# 计算思维论题1：
## -为什么计算机看上去那么强大？

2013年9月27

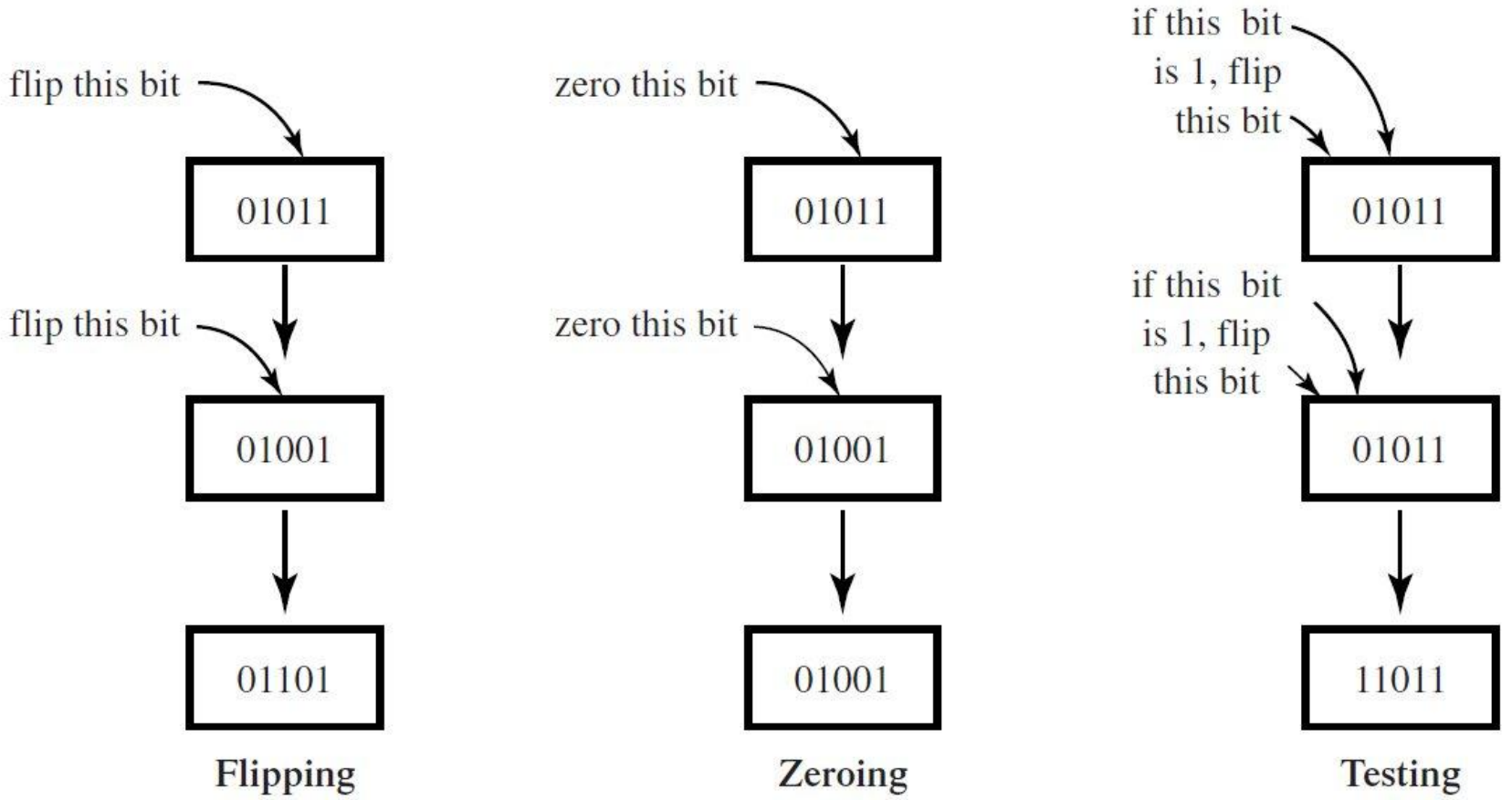
# Computers are amazing machines.

- They seem to be able to do anything. They fly aircraft and spaceships, and control power stations and hazardous chemical plants. Companies can no longer be run without them, and a growing number of sophisticated medical procedures cannot be performed in their absence. They serve lawyers and judges who seek judicial precedents in scores of documented trials, and help scientists in performing immensely complicated and involved mathematical computations.
- They route and control millions of telephone calls in networks that span continents. They execute tasks with enormous precision—from map reading and typesetting to graphical picture processing and integrated circuit design. They can relieve us of many boring chores, such as keeping a meticulous track of home expenses, and at the same time provide us with diverse entertainment such as computer games or computerized music. Moreover, the computers of today are hard at work helping design the even more powerful computers of tomorrow.

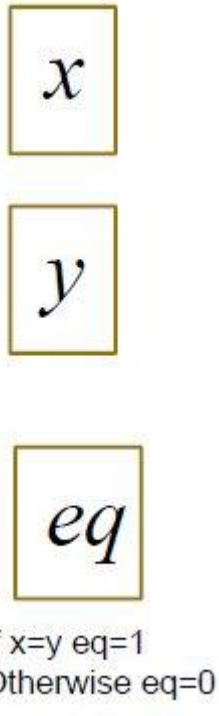
# 实际上，计算机是什么？

- even the most sophisticated computer is really only a large, well-organized volume of bits, and moreover it can normally carry out only a small number of extremely simple operations on them, such as zeroing, flipping, and testing

# 计算机其实很笨！


flip this bit
01011
flip this bit
01001
01101
Flipping
zero this bit
01011
zero this bit
01001
01001
Zeroing
if this bit
is 1, flip
this bit
01011
if this bit
is 1, flip
this bit
01011
11011
Testing

# 但是，计算机确实很神奇！




```
Equality test (x,y)
    zero eq;
    flip eq; /* equality on
    test x flip eq;
    test y flip eq; /* equality on only turn twice
```

你能将这个操作扩展到，
比如，32位内的整数吗？

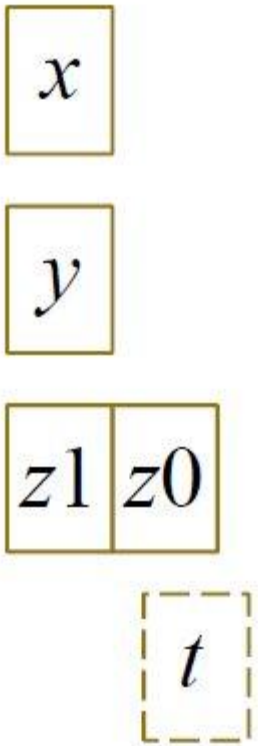
我们用x0,x1,…,x31表示x的32个bits，用y0,y1,…,y31表示y的32个bits

# 一个稍微复杂的例子

- 计算两个1bit的二进制数的和(增加exit和goto操作)

add(x,y)

1. zero z0
2. zero z1
3. equality test(x,y)
4. test eq goto 7
5. flip z0
6. exit
7. zero t
8. flip t
9. equality test(x,t)
10. test eq flip z1




问题：

如果只用zero、flip、test exit、goto操作，能完成这个任务吗？

# 封装+抽象

- 用最原子的操作封装成更“高级”的操作
  - 用三个基本操作可以实现“相等”操作
  - 用“相等”操作可以实现一位的加法操作
  - 用一位的加法“间接操作”可以实现普通加法
  - a:= x + y

实际上，现代计算机内部电路能提供的“原子”操作远不止这5个基本操作
不同的原子操作 == 不同的计算机

实际上，我们在编写程序时使用“封装”的高级操作可以“随心所欲”
不同级别的语言、不同级别的软件库

# 一段等价的汇编和高级语言程序

```
//--计算|x-y|的C代码-
int absdiff(int x, int y)
{
  if (x < y)
    return y - x;
  else
    return x - y;
}
```

```
//----------汇编代码----------
 movl 8(%ebp),%edx    ; 取x的值
 movl 12(%ebp),%eax   ; 取y的值
 cmpl %eax,%edx       ;比较x和y的值
 jl .L3                ;如果y<x 转到.L3
 subl %eax,%edx       ; 计算y-x
 movl %edx,%eax       ; 返回值
 jmp .L5               ; 跳转到.L5
.L3:                   ;y<x
   subl %edx,%eax     ;计算x-y
.L5:                   ; 完成
```

# 但，本质上都是：

However, the outward appearance is of peripheral importance when compared to the bits and their internal arrangement. It is the bits that "sense" the external stimuli arriving from the outside world via buttons, levers, keys on a keyboard, electronic communication lines, and even microphones and cameras. It is the bits that "decide" how to react to these stimuli and respond accordingly by directing other stimuli to the outside via displays, screens, printers, loudspeakers, beepers, levers, and cranks.

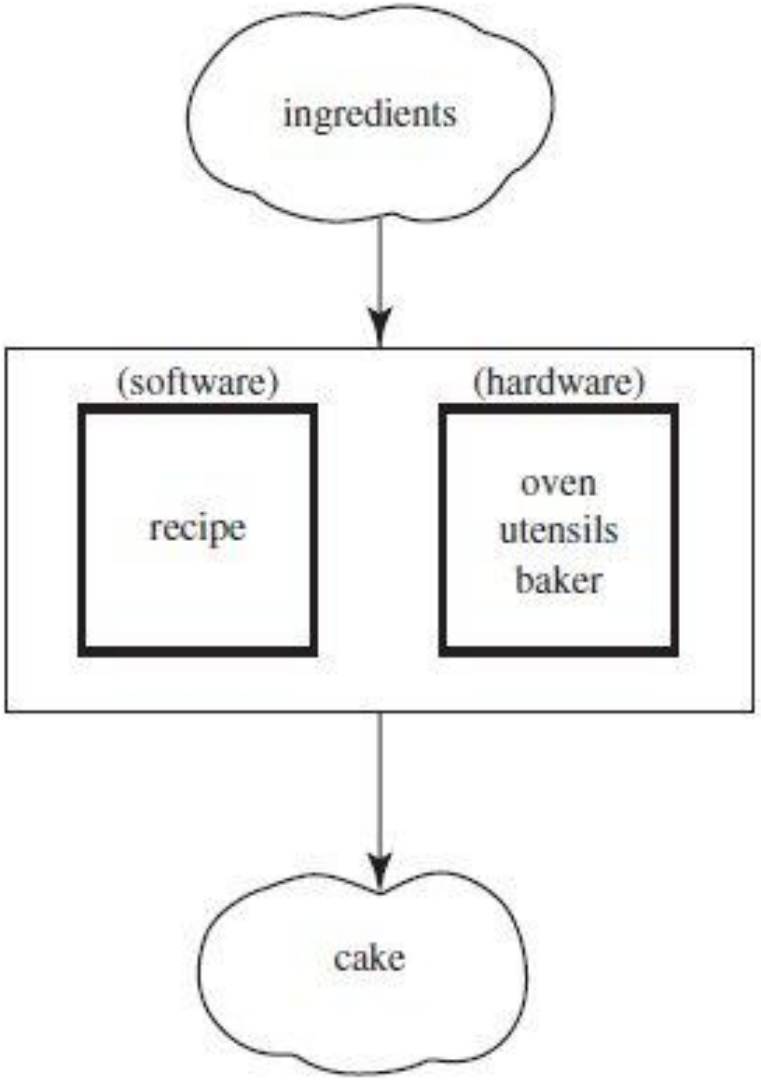
# 关于问题的广义理解

- 求解3x=6和求解ax=b的区别
  - “哪个x能够满足3x=6”:问题的实例
  - 在给定任意的a,b时，哪个x能够满足ax=b：问题！
- 计算机解题：
  - 解一般的题

# 人类如何解题？

- George Polya: “How to Solve It?”
  - Understanding the problem: “What you are given and what you are supposed to figure out”
  - Devising a plan: “How will you attack the problem?”
  - Carrying out the plan: Solve the problem.
  - Looking back: check the result, and…

# 人类如何解题的？烘蛋糕


ingredients
(software)
recipe
(hardware)
oven
utensils
baker
cake

# 计算机也会解题？

- 计算机如何理解问题?
  - 输入是什么? 输出是什么?
- 如何针对计算机制定计划?
  - 什么样的”计划”可能在计算机上实现?
  - 什么样的形式才能让计算机知道该怎么做?
- 执行计划–“计算机解题”
  - 只有这个才真正是计算机做的!
- 回头看
  - 为什么结果是正确的?
  - 效率能提高吗?

# 一个例子程序

- 求某个机构的工资总额

- 用这个程序解读“计算机解题”的要素
  - 数据+算法
  - 语言+程序

# 人解题

- 理解问题：这是一个求和的问题。
- 解题计划：使用加法就够了！
- 实施计划：找纸和笔，逐个加上工资；或者使用一个计算器，逐个加上工资。
- 检查结果：工资总和是否和上月差不多？

# 人解题

| 外名 | 工种 | 考勤 | 日工资 | 工资总额 |
|---|---|---|---|---|
| | 队长 | 30 | 350 | 10500 |
| | 班长 | 30 | 300 | 9000 |
| | 班长 | 30 | 300 | 9000 |
| | 库管 | 30 | 140 | 4200 |
| | 壮工 | 30 | 140 | 4200 |
| | 壮工 | 30 | 100 | 3000 |
| | 壮工 | 5 | 100 | 500 |
| 杰 | 瓦工 | 30 | 220 | 6600 |
| 民 | 瓦工 | 30 | 220 | 6600 |
| 彬 | 抹灰 | 30 | 220 | 6600 |
| 欣 | 抹灰 | 30 | 220 | 6600 |
| 轻 | 抹灰 | 30 | 220 | 6600 |
| 利 | 抹灰 | 30 | 220 | |


结果

# 计算机解题

- 针对计算机理解问题：
  - 对所有的工资求和问题
  - 输入n个工资信息；输出n个工资的和
- 针对计算机制定计划：
  - 工资数据放在哪里？怎么放？
  - 工资数据如何读入计算机？
  - 如何求和？
  - 求和后的数据如何输出？如何显示？

# 计算机解题

- 执行计划
  - 数据放在数组里（当然也可以放在文件里，这里只是作为一个例子）
  - 逐个扫描数组
  - 使用循环语句将数组里的数据逐个相加
  - 输出最终结果

# 计算机解题

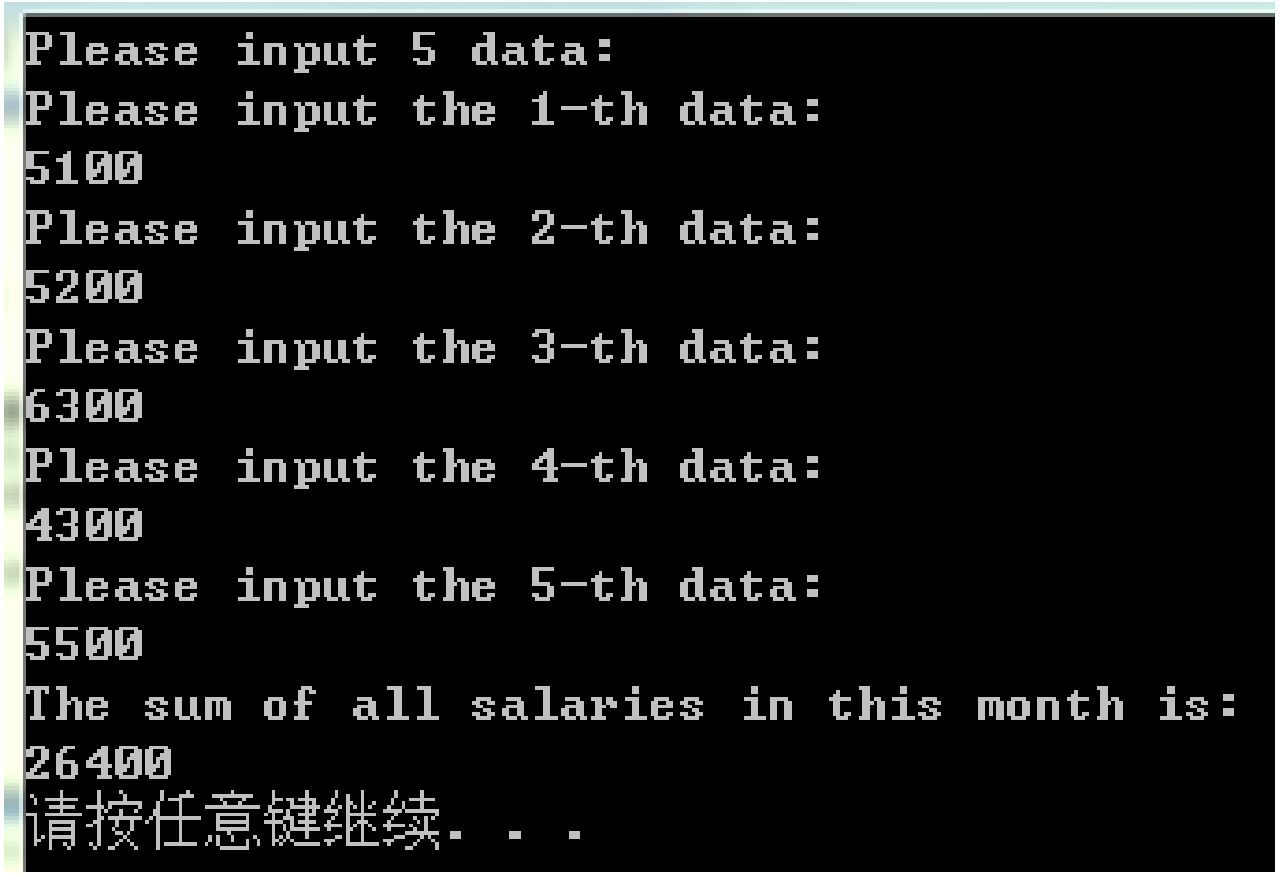
- // SalarySum.cpp: 主项目文件。
- //预处理
- #include "stdafx.h"
- #include <iostream>
- #define N 5
- using namespace std;
- //声明
- float Salary[N];
- void Init();
- double Sum();

```
//主函数
int main()
{
	double AllSalary = 0;
	//数据初始化
	Init();

	//工资求和
	AllSalary = Sum();

	//输出工资总和
	cout<<"The sum of all salaries in this month is: "<<AllSalary<<endl;
	 return 0;
}
```

```
//求工资的总和
double Sum()
{
	//使用一个循环，求和后将结果返回
	double All = 0;
	for (int i = 0;i<N;i++)
	{
		All = All + Salary[i];
	}
	return All;
}
```

```
//数据初始化程序
void Init()
{
	//录入数据,放入数组；也可以通过读文件的方法
	cout<<"Please input "<<N<<" data:"<<endl;
	for (int i = 0;i<N;i++)
	{
		//提示输入
		cout<<"Please input the "<<i+1<<"-th data:"<<endl;
		//放入数组
		cin>>Salary[i];
	}
}
```

# 运行结果


Please input 5 data:
Please input the 1-th data:
5100
Please input the 2-th data:
5200
Please input the 3-th data:
6300
Please input the 4-th data:
4300
Please input the 5-th data:
5500
The sum of all salaries in this month is:
26400
请按任意键继续. . .

# 计算机解题与数学

- 对问题的理解必须用严格的数学语言描述。
  - 其前提是必须建立问题的数学模型。
  - 可用的数学模型必须是计算机能对其进行操作的。
  - 让计算机能理解的解题plan必须建立在严密的数学基础上

- 将plan表示为计算机能执行的“指示”的语言必须建立在严密的数学基础上

- 分析计算机计算的结果必须使用数学方法：
  - 用逻辑证明结果正确；
  - 动用必要的数学手段分析解法的效率。

# 结论

- 从如此简单的基本元件和操作来完成不可思议的复杂行为?
  - 计算机如何做到?
- 核心答案
  - 过程和描述这个过程的算法
  - 数据，输入的数字化描述
- 辅助机制
  - 计算机能理解的高级语言和程序+硬件执行机制