系统思维概述

抽象化

数据抽象

徐志伟 zxu@ict.ac.cn

提纲

- 系统思维概述
  - **以一耦万**：一套抽象栈支持万千应用
    - 通过一套抽象，将模块组合成为系统，无缝执行众多计算过程
  - 三个目标：周到、整体、应对复杂性
  - 三个利器：**抽象化**、**模块化**、**无缝衔接**， Acu-Ex**ams**-CP
- 什么是抽象
  - 抽象三性质（COG，齿轮性质）
  - 数据抽象
  - 控制抽象（含模块抽象）
- 文件操作

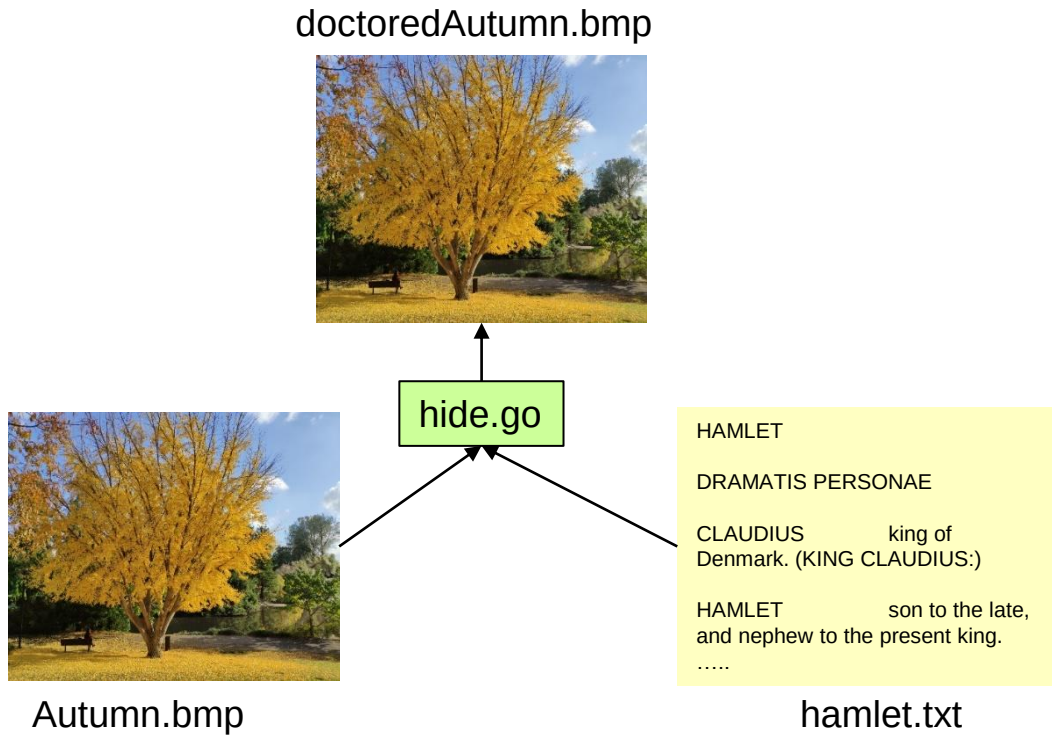
课件中可能包含素材引用，特此致谢！

果壳场景

1. 设计**Unicode**表示全球字符

我们已知如何用ASCII编码表示英文字符。设计新编码，表示全球字符，如A、Θ、😊、中、¥

2. **信息隐藏**



章题引用：两千多年前

民可使由之，不可使知之。

Enable people to follow the way, without them having to understand [the internals of] it.

— 孔子（Confucius），551–479 BCE

- 愚民政策？
  - 肯定不是。孔子是教育家，实践有教无类。
- 英文翻译可能更准确反映原意
  - 此句在讲“抽象”，类似于信息隐藏原理
    - 黑箱模型即可，不要强迫人掌握白箱模型

章题引用：2022年

计算思维的核心是抽象

At the heart of computational thinking is abstraction.

—Alfred Aho and Jeffrey Ullman, 2022

Alfred Aho

Jeffrey Ullman

Turing Lecture 图灵奖演讲论文

Aho, A. and Ullman, J. Abstractions, their algorithms and their compilers. CACM 65, 2 (Feb, 2022), 76-91.

计算抽象，是能够应对复杂性、比特精准、自动执行、以一耦万 的 信息变换抽象

抽象栈思路在科学技术领域广泛使用

考维尔太极图

引自美国国家科学基金会主任瑞塔•考维尔（Rita Colwell）于2002年11月在国际超级计算会议（SC'02）上所作的主旨报告。沿着集成法主轴（Integration）看，标注分别为原子级、分子级、细胞级、组织、器官、生物体、栖息地、种群、群落社区、生态系统、行星级、宇宙级。

考维尔于21世纪初展示的生物复杂性图，体现了自然科学在21世纪的两种基本发展趋势：

1. 跨学科趋势，一门学问（如生物学）将在多个层次/级别/尺度、从多个学科角度研究；
2. 集成法（Integration）将补充盛行几百年的规约法（Reductionism）。

https://nsf.gov/news/speeches/colwell/rc02_sc2002/sld020.htm

1. 什么是系统思维?

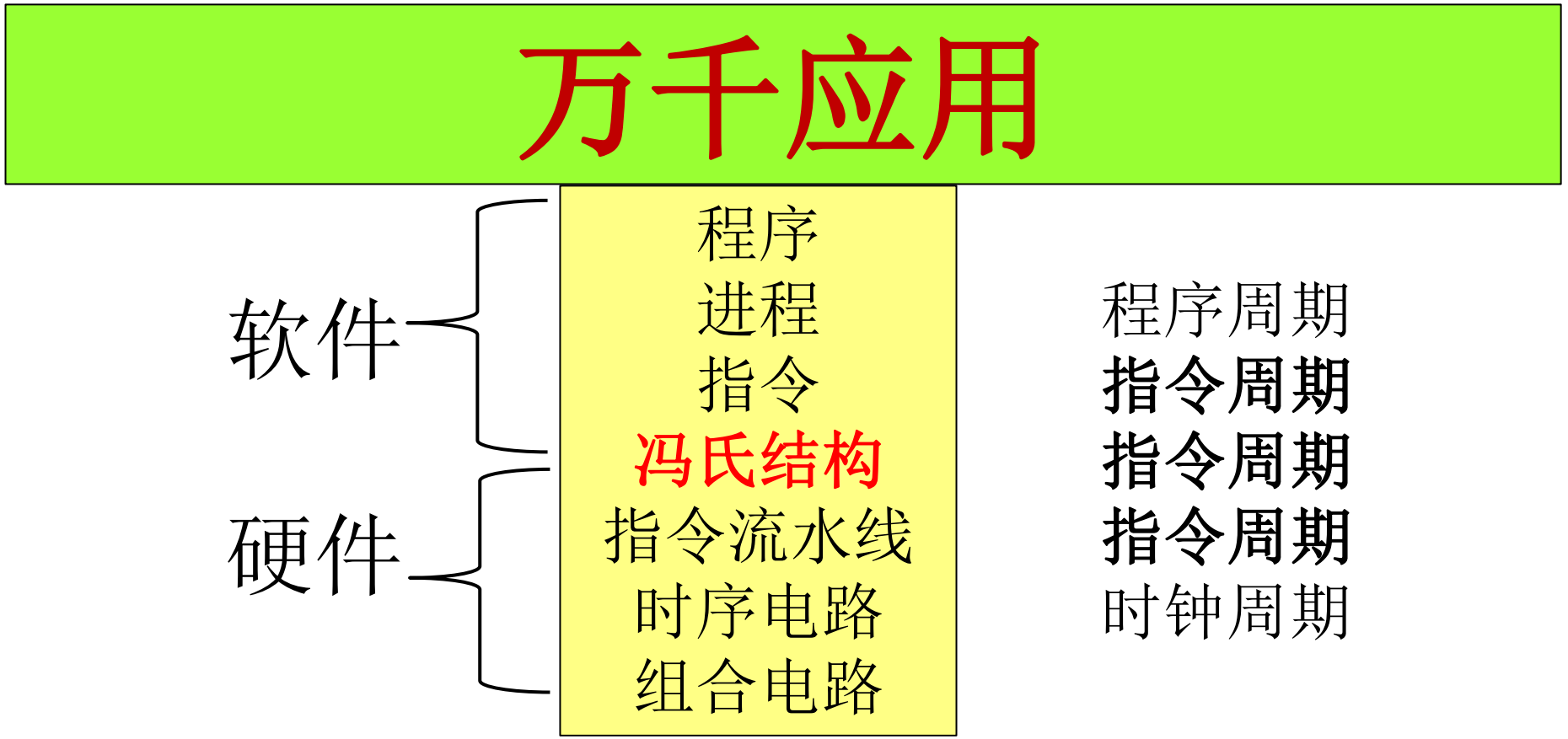
- 系统思维的作用
  - 系统思维使得计算过程实用，从而构造出实用的系统

- 基本思维方法
- 以一耦万：使用一套**抽象**，组合**模块**成为系统，**无缝执行**众多计算过程
  - 向编程者提供抽象，比特精准地将抽象映射到最底层硬件
    - 霍尔悖论展示的递归抽象

- 系统思维的内涵
  - 三个追求：周到性、整体性、应对复杂性
  - 两个基本思路：抽象栈、周期
  - 三个利器：抽象化、模块化、无缝衔接

通过抽象栈与周期实现以一耦万

- 一套抽象栈支持万千应用



通过抽象栈与周期实现以一耦万

知行合一要求：掌握一个程序的实现即可

万千应用

```
fib[0] = 0
fib[1] = 1
for i := 2; i < 51; i++ {
   fib[i] = fib[i-1] + fib[i-2]
}
```

程序
进程
指令
冯氏结构
指令流水线
时序电路
组合电路

抽象栈的作用

比特精准地将高级抽象映射到最底层硬件

```
fib[0] = 0
fib[1] = 1
for i := 2; i < 51; i++ {
   fib[i] = fib[i-1] + fib[i-2]
}
```

万千应用

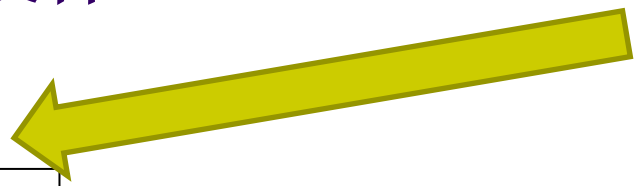
程序
进程
指令
冯氏结构
指令流水线
时序电路
组合电路

比特精准地将抽象映射到最底层硬件

万千应用

程序
进程
指令
冯氏结构
指令流水线
时序电路
组合电路

```
MOV 0, R1
MOV R1, M[R0]
MOV 1, R1
MOV R1, M[R0+8]
MOV 2, R2    // i:=2
MOV 0, R1    // label Loop
ADD M[R0+R2*8-16], R1
ADD M[R0+R2*8-8], R1
MOV R1, M[R0+R2*8-0]
INC R2          // i++
CMP 51, R2  // i < 51?
JL Loop   // if '<' goto Loop
```



```
fib[0] = 0
fib[1] = 1
for i := 2; i < 51; i++ {
   fib[i] = fib[i-1] + fib[i-2]
}
```

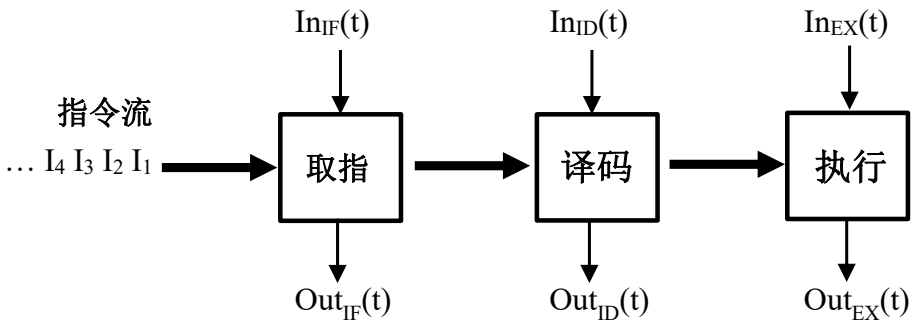
比特精准地将抽象映射到最底层硬件

万千应用

程序
进程
指令
冯氏结构
指令流水线
时序电路
组合电路

```
MOV 0, R1
MOV R1, M[R0]
MOV 1, R1
MOV R1, M[R0+8]
MOV 2, R2    // i:=2
MOV 0, R1    // label Loop
ADD M[R0+R2*8-16], R1
ADD M[R0+R2*8-8], R1
MOV R1, M[R0+R2*8-0]
INC R2          // i++
CMP 51, R2  // i < 51?
JL Loop   // if '<' goto Loop
```

```
fib[0] = 0
fib[1] = 1
for i := 2; i < 51; i++ {
   fib[i] = fib[i-1] + fib[i-2]
}
```



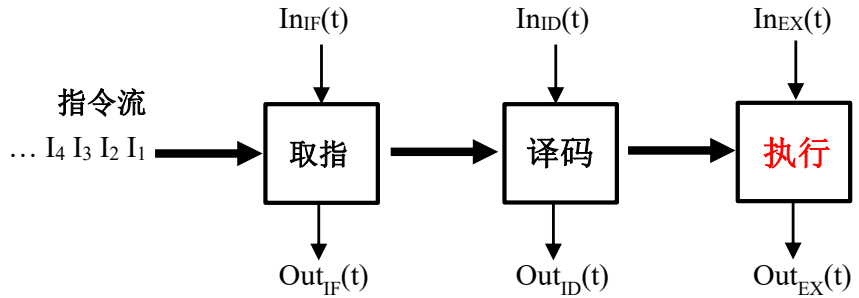
比特精准地将抽象映射到最底层硬件

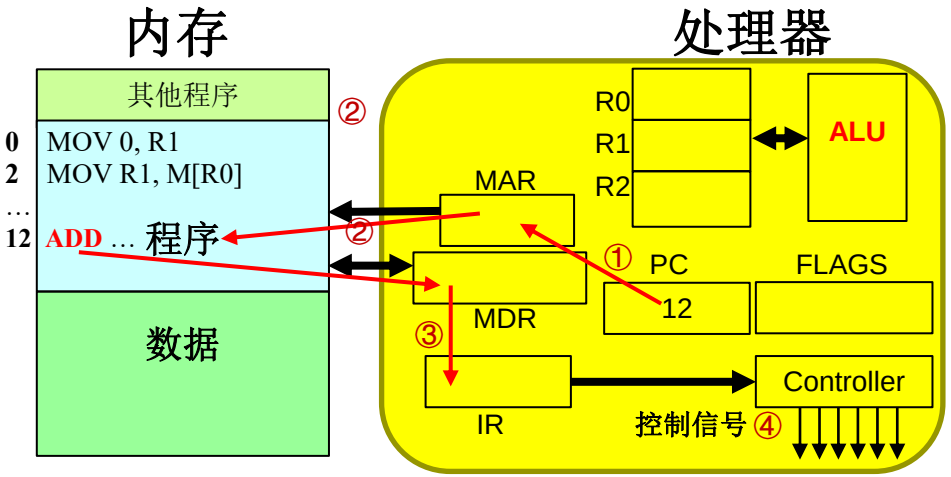
万千应用

程序
进程
指令
冯氏结构
指令流水线
时序电路
组合电路

```
MOV 0, R1
MOV R1, M[R0]
MOV 1, R1
MOV R1, M[R0+8]
MOV 2, R2    // i:=2
MOV 0, R1    // label Loop
ADD M[R0+R2*8-16], R1
ADD M[R0+R2*8-8], R1
MOV R1, M[R0+R2*8-0]
INC R2       // i++
CMP 51, R2   // i < 51?
JL Loop   // if '<' goto Loop
```

```
fib[0] = 0
fib[1] = 1
for i := 2; i < 51; i++ {
   fib[i] = fib[i-1] + fib[i-2]
}
```





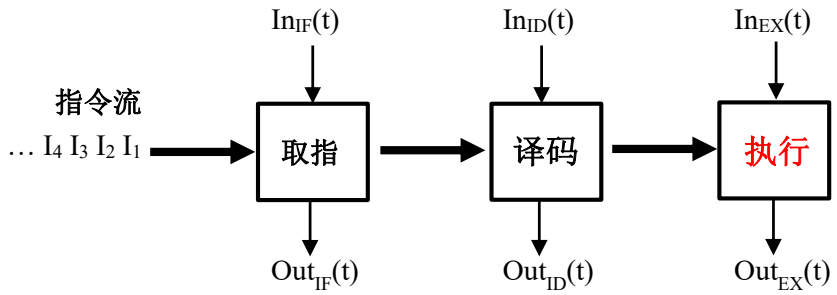
比特精准地将抽象映射到最底层硬件

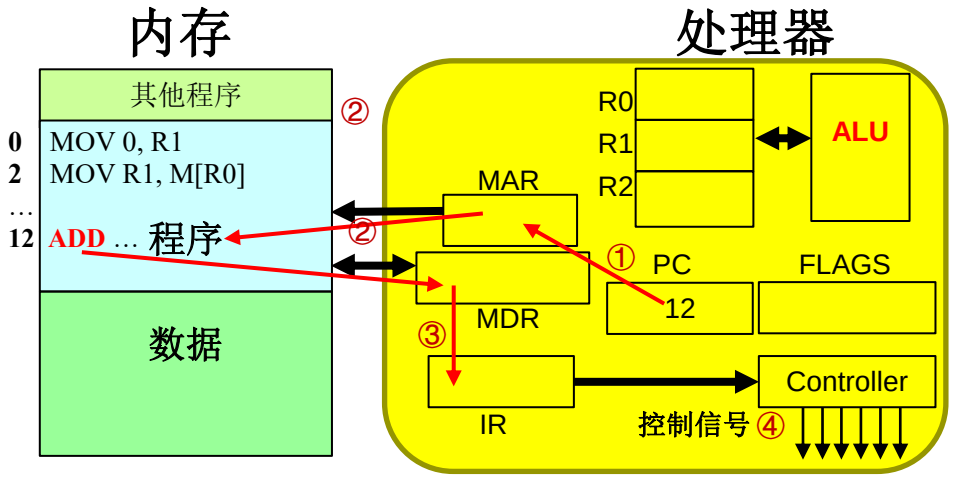
万千应用

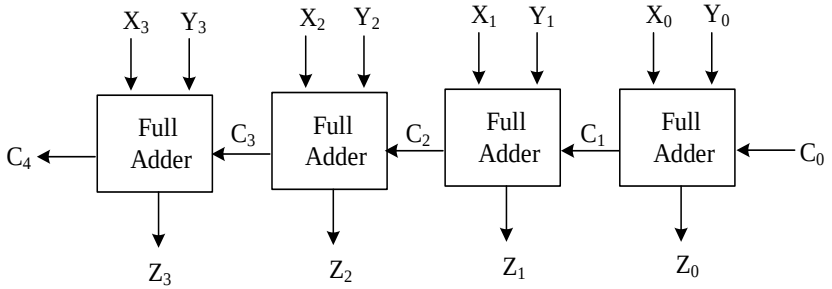
程序
进程
指令
冯氏结构
指令流水线
时序电路
组合电路

```
MOV 0, R1
MOV R1, M[R0]
MOV 1, R1
MOV R1, M[R0+8]
MOV 2, R2    // i:=2
MOV 0, R1    // label Loop
ADD M[R0+R2*8-16], R1
ADD M[R0+R2*8-8], R1
MOV R1, M[R0+R2*8-0]
INC R2       // i++
CMP 51, R2   // i < 51?
JL Loop   // if '<' goto Loop
```

```
fib[0] = 0
fib[1] = 1
for i := 2; i < 51; i++ {
   fib[i] = fib[i-1] + fib[i-2]
}
```







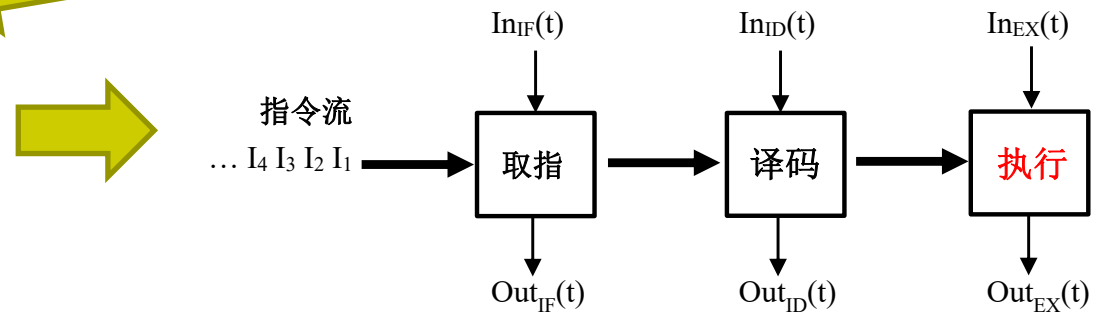
比特精准地将抽象映射到最底层硬件

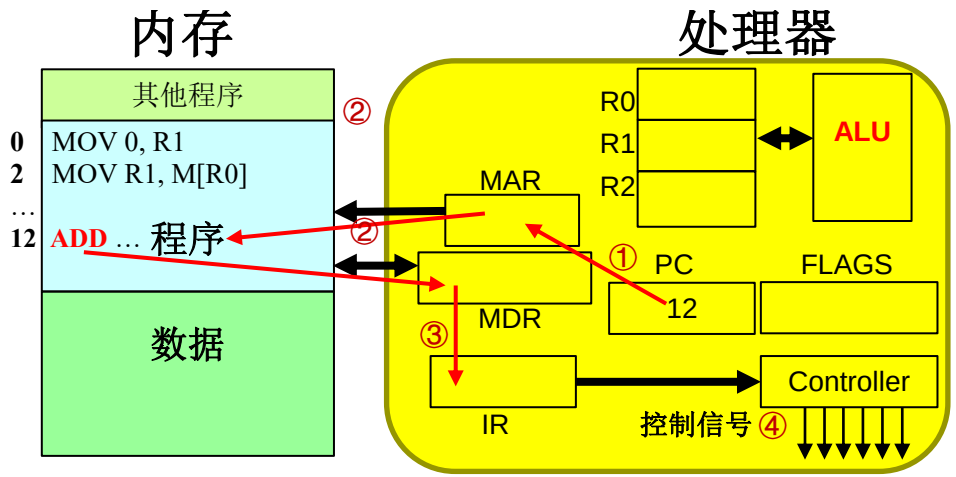
万千应用

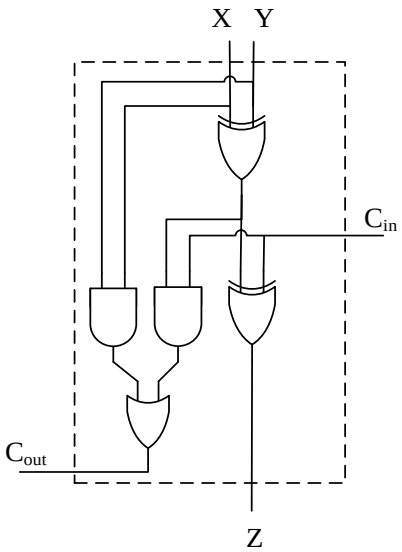
程序
进程
指令
冯氏结构
指令流水线
时序电路
组合电路

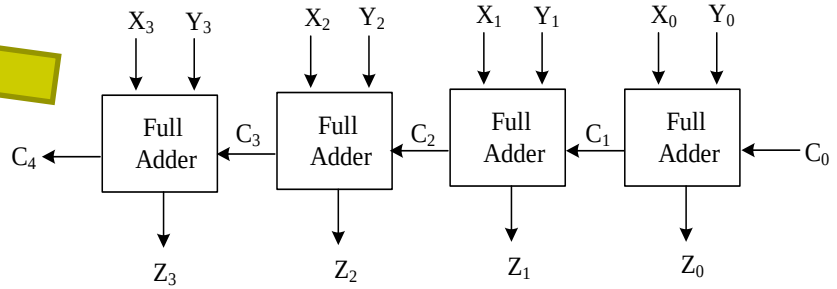
```
fib[0] = 0
fib[1] = 1
for i := 2; i < 51; i++ {
   fib[i] = fib[i-1] + fib[i-2]
}
```

```
MOV 0, R1
MOV R1, M[R0]
MOV 1, R1
MOV R1, M[R0+8]
MOV 2, R2     // i:=2
MOV 0, R1     // label Loop
ADD M[R0+R2*8-16], R1
ADD M[R0+R2*8-8], R1
MOV R1, M[R0+R2*8-0]
INC R2        // i++
CMP 51, R2    // i < 51?
JL Loop   // if '<' goto Loop
```





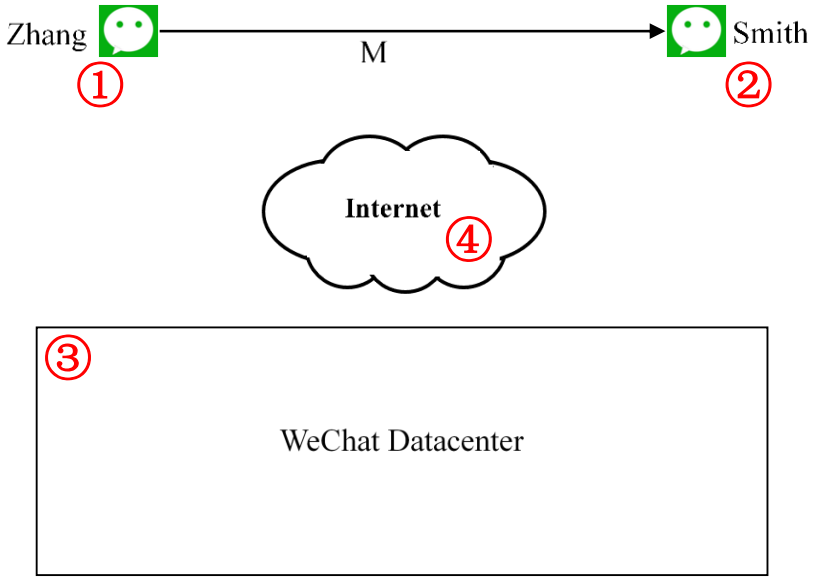


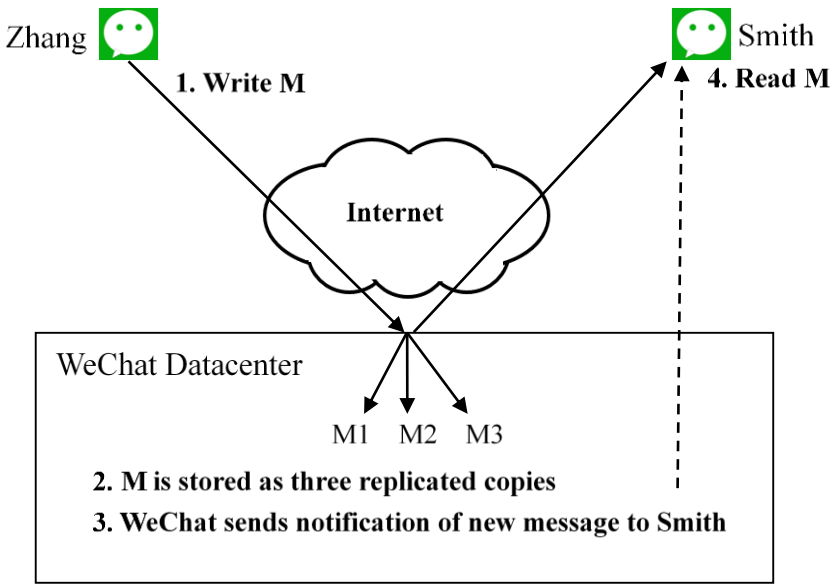


1.1 Being thorough 周到性

- 何处存储张蕾发给Smith的微信消息M
- 考虑端到端全系统，涵盖所有应用场景、所有必要细节
  - 那些细节？功能、易用性、性能、容错、隐私、可扩展性，等
  - 什么是端到端？从哪端到哪端？
    - 横向：从发送端到接收端
    - 纵向：考虑整个技术栈

① 张蕾的手机
② Smith的笔记本电脑
③ 微信数据中心
④ 互联网中的其他地方

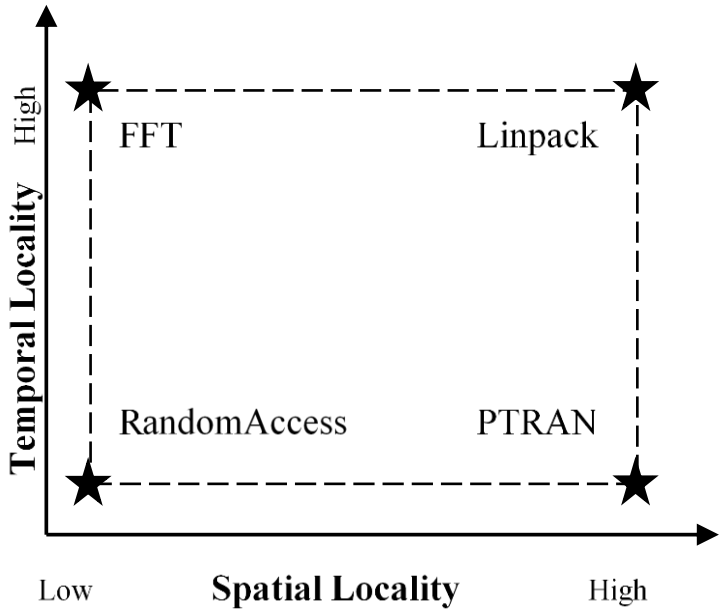




考虑所有应用场景：基准程序

- 如何知道超级计算机A比超级计算机B快1000倍?
  - 有成千上万种应用程序
- 如何考虑所有情况？
- 方法：覆盖局部性四角的基准程序集（benchmarks）
  - Low-low, low-high, high-low, high-high
  - Other applications are in the area enclosed by the extreme points

Computer performance is critically influenced by locality.
Temporal locality 时间局部性: data and instructions currently used tend to be used again in near future.
Spatial locality 空间局部性: locations nearby a reference item will also likely be referenced.

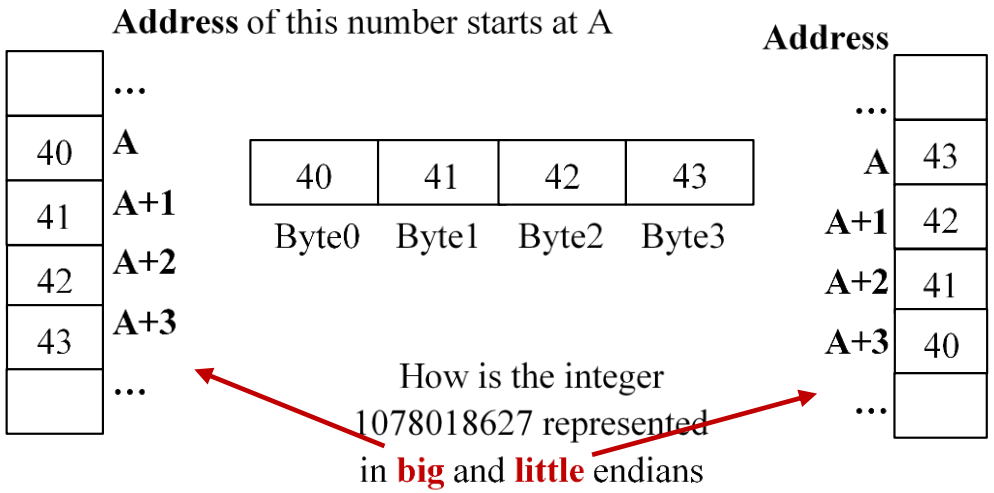


不能忽略必要细节
必须考虑到必要但无聊细节

- 数据的大端（big endian）与小端（little endian）表示
  - 大端：将最高位字节0x40放在起始地址A；0x41在A+1，0x42在A+2， 0x43在A+3
  - 小端：将最低位字节0x43放在起始地址A；0x42在A+1，0x41在A+2， 0x40在A+3
- 尽快定下一个即可
  - Danny Cohen: “Agreement upon an order is more important than the order agreed upon.”
- 现状：两种都在使用，必要时转换
  - 大端派：TCP/IP 网络，MIPS处理器
  - 小端派： x86，ARM，RISC-V处理器

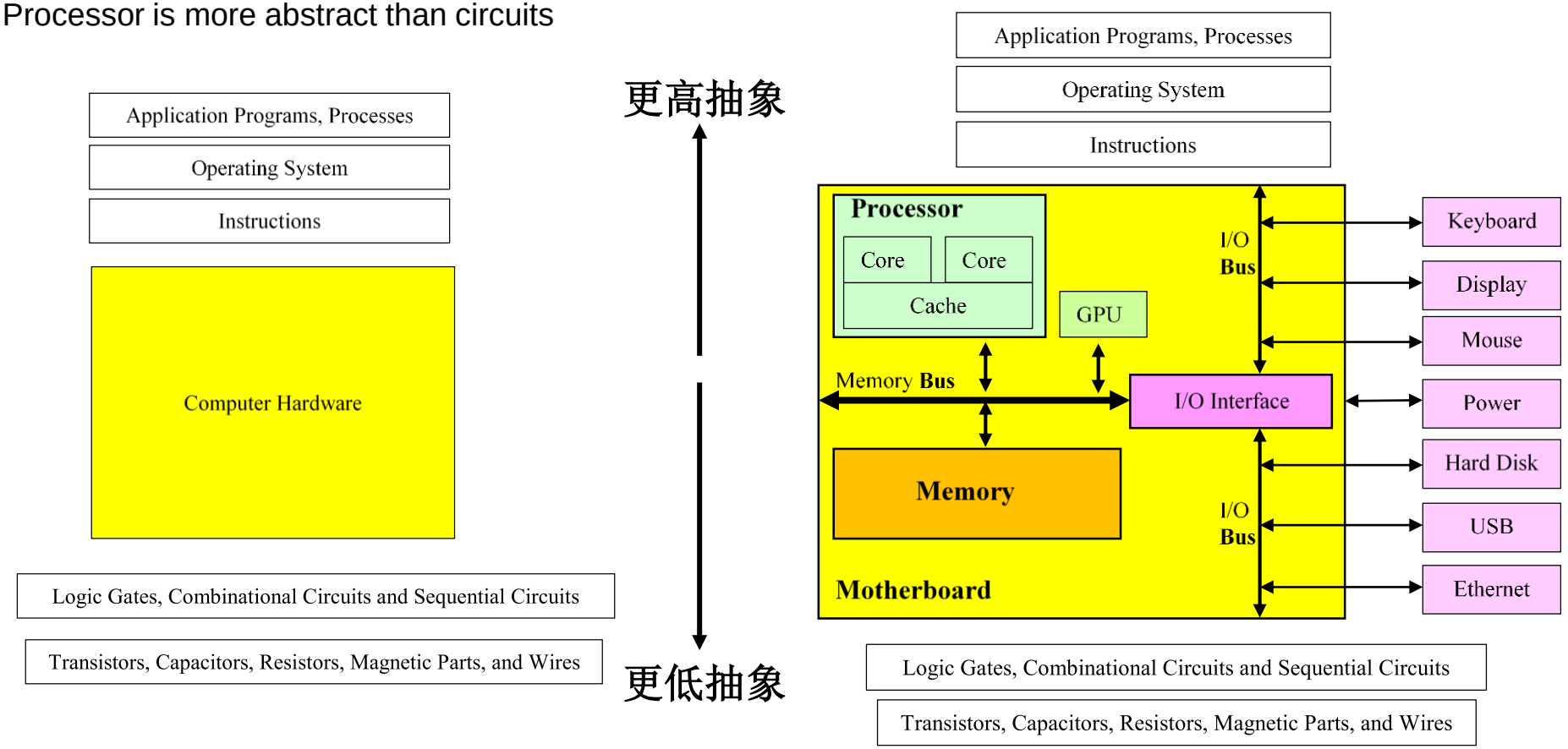
32位整数 0x40414243 = 1078018627_{10}
包含4个字节
最高位 Byte0: 01000000=0x40,
Byte1: 01000001=0x41,
Byte2: 01000010=0x42,
最低位 Byte3: 01000011=0x43.

0x40, 0x41, 0x42, 0x43 四个字节如何摆放在内存地址 A, A+1, A+2, A+3?



1.2 Being systematic 整体性
整体地、系统地，而不是随意地、无章法地

- 技术栈方法，即抽象栈方法（以一耦万）
  - Use one stack of layers of abstractions to support many applications
    - Instead of one stack for an application, in an *ad hoc* way
  - Upper layer provides higher abstraction than lower layers
    - Processor is more abstract than circuits



1.3 Coping with complexity
应对复杂性

- 系统复杂性不同于算法复杂度
- 系统越复杂，越难设计、实现、使用
  - 越难被人驾驭

- 复杂性客观存在，有四个表象和因素
  - **多**：系统规模大，部件多
  - **杂**：系统的部件有多种多类，异构性大
  - **乱**：系统的组织杂乱无章
  - **变**：系统的组织或部件随意变化

- 应对思路：惠勒间接原理

微信系统
- **多**：用户数超过十亿
- **杂**：设备种类高达数千
- **乱**：灵活，并非杂乱无章
- **变**：每天都在升级变化

惠勒间接原理

- 任何计算机科学问题都可以通过又一个间接层解决
  Any problem in computer science can be solved with another level of indirection

- 大卫•惠勒提出了这个看起来很夸张的原理

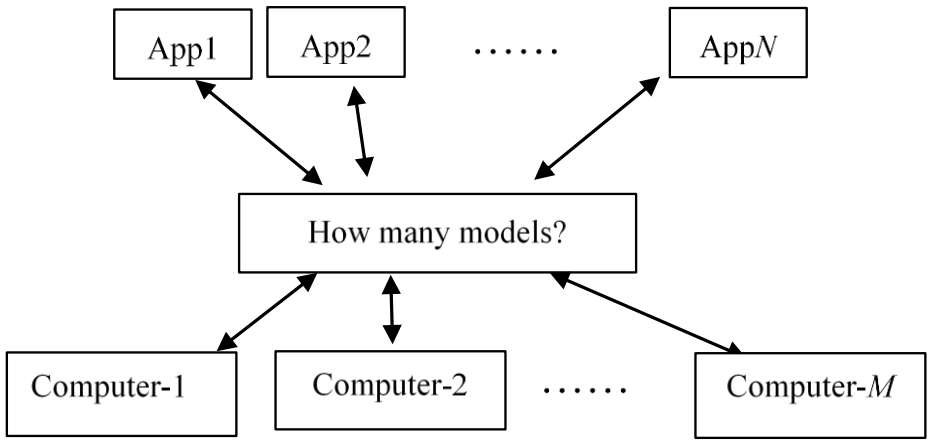
- 巴特勒·兰普森（Butler Lampson）在1993年图灵奖演说中引用了这个原理，使它成为一句智慧类名言

大卫•惠勒
David Wheeler
1927-2004

汇编语言发明者
剑桥大学教授

桥接模型：应对“变”带来的复杂性

- 本课程的300名同学们在做Go编程时，看见多少种计算机型号？



Millions of applications

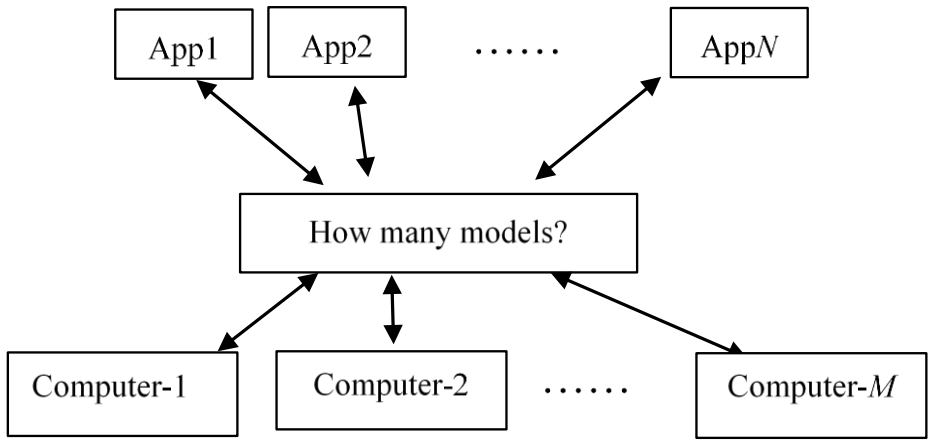
Thousand types of computers

桥接模型：应对“变”带来的复杂性

- 本课程的300名同学们在做Go编程时，看见多少种计算机型号？

A. 1种

- 冯诺依曼模型是桥接模型（bridging model）
  - 桥接应用与真实计算机
  - 针对冯诺依曼模型写的程序，可变换成真实计算机上运行的程序
  - 性能差别只有 O(1)
  - 针对图灵机写的程序，在真实计算机上运行，性能差别可能高达 $O(N^4)$



Millions of applications

1种计算机：**von Neumann Model**

Thousand types of computers

2. What is abstraction? 什么是抽象？

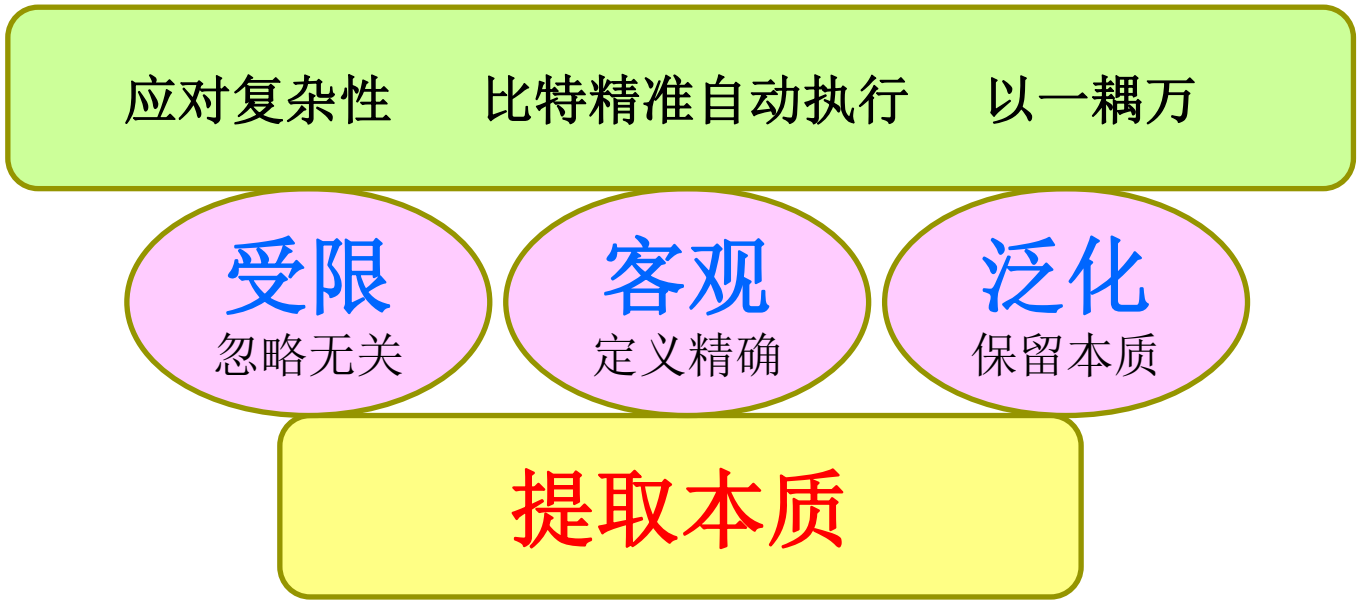
- 抽象 = 抽象过程
  - The creative process of abstracting a high-level entity from low-level instances by focusing on the essential
- 抽象 = 抽象过程的结果
  - Also the outcome of the creative process of abstracting

本课程涉及的
四种抽象

- 数据抽象
- 控制抽象
- **体系结构**
- 硬件电路

The COG properties 抽象的齿轮三性质

- 受限（Constrained），客观（Objective），泛化（Generalizable）
  - 受限，才能忽略/隐藏无关细节，从而应对复杂性
  - 客观，才能将本质定义成为精确概念，从而支持比特精准与自动执行
  - 泛化，才能提取本质、保留本质，从而实现以一耦万



抽象例子：万维网（World Wide Web）

- 抽象过程：采用超链接获取全球信息资源
- 抽象产物：Tim Berners-Lee发明的万维网抽象
- WWW的齿轮三性质
  - 受限（Constrained）
    - 聚焦“采用超链接获取全球万维网资源”，忽略了如何存储处理等细节
  - 客观（Objective）
    - 精确地定义了URL、HTML、HTTP等万维网技术标准
  - 泛化（Generalizable）
    - 最初（1989年），资源 = 文档
    - 30年后，万维网资源抽象从文档泛化到

Multimedia	多媒体
Programs	程序
Services	服务
Data	数据
Things	智能物体

https://www.w3.org/comm/assets/logos/Web@30_logo/Logo_web/PNG/Logo_Logo_horizontal.png

惠勒间接原理应用实例：Unicode + UTF-8

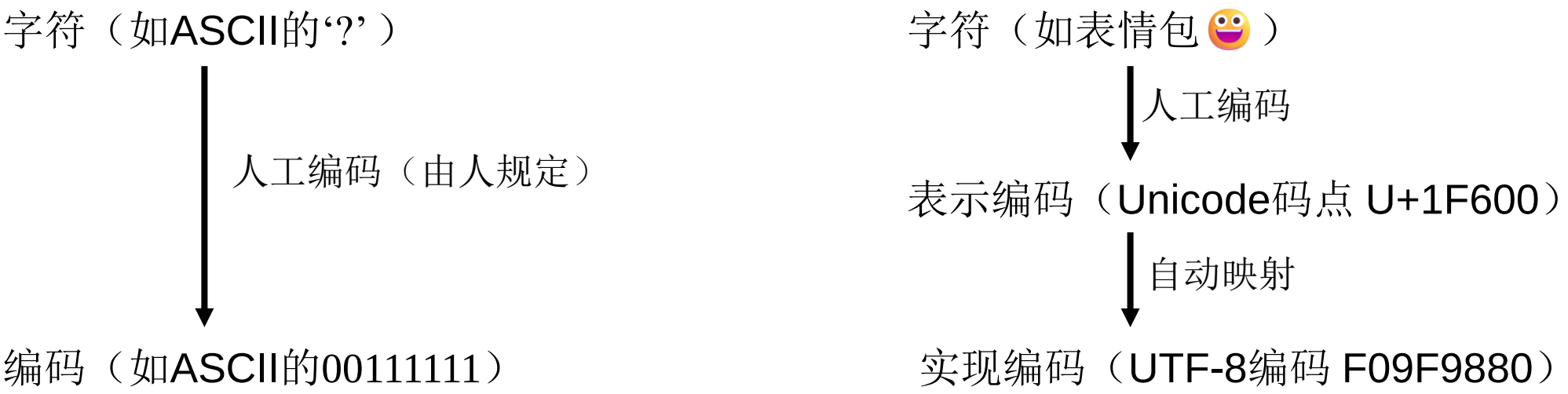
- 问题：Encoding the world‘s writing systems，即推广ASCII思路“编码世界上所有字符”
  - “世界上所有字符”的集合远大于ASCII字符集，且在不断增长；当前预留了百万字符的空间
- 简单粗暴方法：英文字符需要1字节编码 → 世界上所有百万字符需要4字节编码
- 1M英文字符构成的程序文件需要多少存储空间？
- 1M英文字符构成的程序文件需要4MB，而不是ASCII码的1MB

- 惠勒间接原理应用，添加一个间接层（表示编码）

惠勒间接原理应用实例：Unicode + UTF-8

- 问题：Encoding the world‘s writing systems，即推广ASCII思路“编码世界上所有字符”
  - “世界上所有字符”的集合远大于ASCII字符集，且在不断增长；当前预留了百万字符的空间
- 简单粗暴方法：英文字符需要1字节编码 → 世界上所有百万字符需要4字节编码
- 1M英文字符构成的程序文件需要多少存储空间？
- 1M英文字符构成的程序文件需要4MB，而不是ASCII码的1MB

- 惠勒间接原理应用，添加一个间接层（表示编码）



Unicode实例：字符→码点

- 抽象过程：Encoding the world's writing systems
- 抽象结果： Unicode， (with COG properties)
  - 受限 Constrained: focus on the essential; ignore irrelevant details
    - Unicode码点 U+5174 表示字符‘兴’，忽略了很多东西，如高兴、兴奋等含义
    - 忽略，才能应对复杂性
  - 客观 Objective: a named, objective entity, no vagueness or ambiguity
    - Syntactically and semantically precisely defined by Unicode standards
    - 定义精准概念，使得 比特精准与自动执行 成为可能
  - 泛化 Generalizable: to unseen instances or unexpected scenarios
    - Able to handle unseen instances and unexpected scenarios，如当代才出现的表情包 😀（U+1F600）
    - 泛化性是为什么一套抽象能够“以一耦万”的重要原因

Symbol	Description	Unicode
T	English capital letter T	U+0054
Ω	Greek letter Omega	U+03A9
€	The Euro sign	U+20AC
志	A Chinese character	U+5FD7
𐍈	A Gothic letter	U+10348

Unicode表示 + UTF-8实现

- “全球字符的二进制表示”问题通过两步解决
  - Unicode表示：将每个字符**映射**到其Unicode码点
  - UTF-8实现：将每个Unicode码点**实现**为其UTF-8编码，即在内存中的摆放
- 英文字符
  - ‘?’的Unicode码点是U+003F=00000000 00111111，位于第1行，需要1个字节
  - 其UTF-8编码为 0xxxxxxx = 00111111

Unicode码点范围	对应的UTF-8编码1~6个字节	字节数
0000~007F	**0xxxxxxx**	**1**
0080~07FF	110xxxxx 10xxxxxx	2
0800~FFFF	1110xxxx 10xxxxxx 10xxxxxx	3
10000~1FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	4
200000~3FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx	5
4000000~7FFFFFFF	111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx	6

Unicode表示 + UTF-8实现

- “全球字符的二进制表示”问题通过两步解决
  - Unicode表示：将每个字符**映射**到其Unicode码点
  - UTF-8实现：将每个Unicode码点**实现**为其UTF-8编码，即在内存中的摆放
- 中文字符
  - ‘你’的Unicode码点是U+4F60= 0100 111101 100000，位于第3行，需要3个字节
  - UTF-8编码为 1110xxxx 10xxxxxx 10xxxxxx = 11100100 10111101 10100000 = E4BDA0

起始字节 跟随字节 跟随字节

Unicode码点范围	对应的UTF-8编码1~6个字节	字节数
0000~007F	0xxxxxxx	1
0080~07FF	110xxxxx 10xxxxxx	2
0800~FFFF	**1110xxxx 10xxxxxx 10xxxxxx**	**3**
10000~1FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	4
200000~3FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx	5
4000000~7FFFFFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx	6

Unicode表示 + UTF-8实现

- “全球字符的二进制表示”问题通过两步解决
  - Unicode表示：将每个字符**映射**到其Unicode码点
  - UTF-8实现：将每个Unicode码点**实现**为其UTF-8编码，即在内存中的摆放
- 中文字符
  - ‘你’的Unicode码点是U+4F60= 0100 111101 100000，位于第3行，需要3个字节
  - UTF-8编码为 1110xxxx 10xxxxxx 10xxxxxx = 11100100 10111101 10100000 = E4BDA0

起始字节 跟随字节 跟随字节

- 既能表示全球字符，又能减小所需字节数
  - 1M英文字符构成的程序文件，只需1MB；直接使用4字节Unicode存储需要4MB

Unicode码点范围	对应的UTF-8编码1~6个字节	字节数
0000~007F	0xxxxxxx	1
0080~07FF	110xxxxx 10xxxxxx	2
0800~FFFF	1110xxxx 10xxxxxx 10xxxxxx	3
10000~1FFFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx	4
200000~3FFFFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx	5
4000000~7FFFFFFF	111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx	6

3. 数据抽象

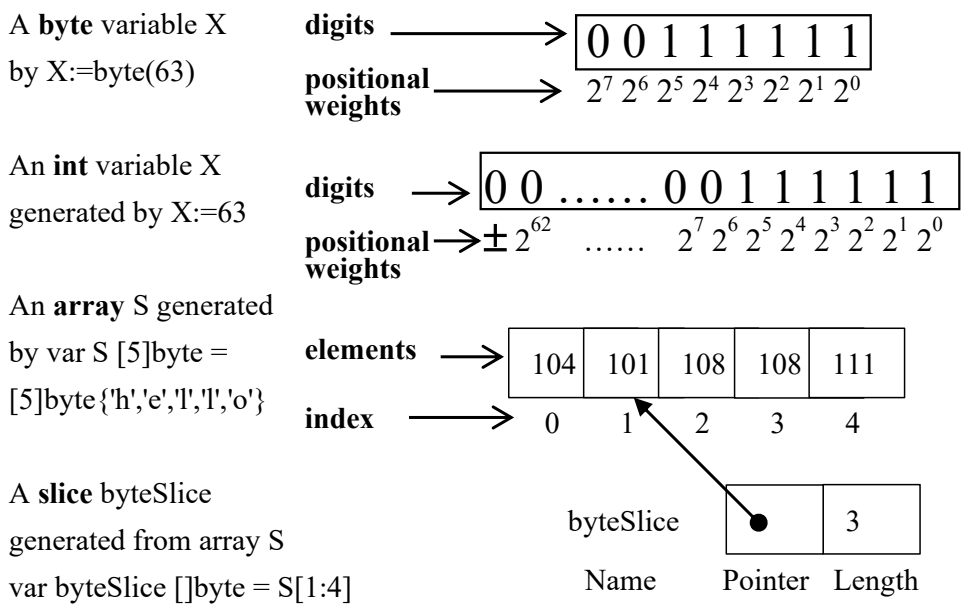
- 数据抽象 = 数据类型 = 数据结构
- 编程基础中已经涉及下列数据类型的入门介绍
  - 整数、数组、切片
  - ASCII字符、符号串
- 如何表示并操作各种数据类型？
  - 任意长度的整数：斐波那契数F(100)=
  - 全球字符（the world’s text）
  - 实数：圆周率π
  - 比特：信息隐藏实验中，隐藏字节到像素阵列
  - 文件：信息隐藏实验中，创建、读、写文件
- 系统如何支持数据类型？
  - 寻址模式、指针（pointer）、结构（struct）

3.1 集中回顾比特、字节、整数、数组、切片

- 请写一个Go程序，打印出各种数据类型的值

```
X := byte(63)                    // X is a byte variable. What if changed to X:=63?
fmt.Printf("Decimal: %d\n", X)   // Decimal: 63
fmt.Printf("Hex: %X\n", X)       // Hex: 3F
fmt.Printf("Character: %c\n", X) // Character: ?
fmt.Printf(“Binary: %b\n”, X)    // Binary: 111111；X是8比特字节，事实上是00111111，忽略了leading 0’s

var S [5]byte = [5]byte{'h','e','l','l','o'}
            // S=[104, 101, 108, 108, 111]
var byteSlice []byte = S[1:4]
            // byteSlice=[101, 108, 108]
            // slice is built from underlying array
            切片是动态数组
fmt.Println("array S = ", S)
            // Array S = [104 101 108 108 111]
fmt.Println("byteSlice = ", byteSlice)
            // byteSlice = [101 108 108]
```



当代计算机与人类使用位值计数法
positional number systems

- 为什么？远比非位值计数法方便
  - 假设使用罗马计数法，一种非位值计数法，值与位置无关
  - Q: MMXXI – MCMLIV = ?
    - Value of MMXXI = M + M + X + X + I = 1000+1000+10+10+1=2021
    - Value of MCMLIV = M + CM + L + IV = 1000+900+50+4=1954
      - Note that CM and IV are short-hand symbols, e.g., IV = IIII = I+I+I+I
  - A: 2021-1954 = 67 = LXVII； MMXXI – MCMLIV = LXVII

  - 注意： MMXXI的两个M都表示1000（一千），与位置无关

罗马计数法与十进制位值计数法的数位对应

Roman	M	D	C	L	X	V	I	IV	IX	XL	XC	CD	CM
Decimal	1000	500	100	50	10	5	1	4	9	40	90	400	900

示例：十进制数$a = 2021.1954$

- 十进制数a有8个数位：2、0、2、1、1、9、5、4
  - 其索引指定位置，分别为3、2、1、0、-1、-2、-3、-4
  - 数位（digit）也翻译为数字；第i位的位置权重是10^i
- 它的值是 $a = \sum_{i=-4}^{3}(a_i \times 10^i) = 2021.1954$
  - 10 是**底数（base）**，i是**索引（index）**，{0,1,2,3,4,5,6,7,8,9}是**数位集（digit set）**
- 位值计数法的要点：数的值由数字和数字所在位置决定
  - 最左边的2在千位（索引为3）；第二个2在十位（索引为1）
  - 最左边的2 表示　$a_3 \times 10^3 = 2 \times 10^3 = 2000$
  - 第二个2 表示　$a_1 \times 10^1 = 2 \times 10^1 = 20$

位值记数法的其他例子

- 自然数的一般定义
  - 任何n-digit number $a = \sum_{i=0}^{n-1}(a_i \times b^i)$，如含小数，可有**i < 0**
    - *b* is the base, *i* is the index, {0,…,b-1} is the digit set
    - *i* is also called position
- 二级制Binary　　如，$01000001 = 1 \times 2^6 + 1 \times 2^0 = 64 + 1 = 65$
  - $a = \sum_{i=0}^{n-1}(a_i \times 2^i)$, $b = 2$, digit set = {0, 1}
  - Used in computers. This is what computers can understand

位值记数法的其他例子

- 一般定义
  - 任何n-digit number $a = \sum_{i=0}^{n-1}(a_i \times b^i)$，如含小数，可有**i < 0**
    - *b* is the base, *i* is the index, {0,…,b-1} is the digit set
    - i is also called position
- 二级制Binary　　如，$01000001 = 1 \times 2^6 + 1 \times 2^0 = 64 + 1 = 65$
  - $a = \sum_{i=0}^{n-1}(a_i \times 2^i)$, $b = 2$, digit set = {0, 1}
  - Used in computers. This is what computers can understand
- 十进制Decimal　　如，$65 = 6 \times 10^1 + 5 \times 10^0 = 60 + 5 = 65$
  - $a = \sum_{i=0}^{n-1}(a_i \times 10^i)$, $b = 10$, digit set = {0,1,2,3,4,5,6,7,8,9}
  - Used by humans and high-level language programs

位值记数法的其他例子

- 一般定义
  - 任何n-digit number $a = \sum_{i=0}^{n-1}(a_i \times b^i)$，如含小数，可有i < 0
    - *b* is the base, *i* is the index, {0,…,b-1} is the digit set
    - i is also called position
- 二级制Binary　　如，$01000001 = 1 \times 2^6 + 1 \times 2^0 = 64 + 1 = 65$
  - $a = \sum_{i=0}^{n-1}(a_i \times 2^i)$, $b = 2$, digit set = {0, 1}
  - Used in computers. This is what computers can understand
- 十进制Decimal　　如，$65 = 6 \times 10^1 + 5 \times 10^0 = 60 + 5 = 65$
  - $a = \sum_{i=0}^{n-1}(a_i \times 10^i)$, $b = 10$, digit set = {0,1,2,3,4,5,6,7,8,9}
  - Used by humans and high-level language programs
- 十六进制Hexadecimal　如，$41 = 4 \times 16^1 + 1 \times 16^0 = 64 + 1 = 65$
  - $a = \sum_{i=0}^{n-1}(a_i \times 16^i)$, $b = 16$, digit set = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}
  - Used by programmers and high-level language programs

还有其他位值记数法吗？

- 差异极大的
  - 能用无理数做底数吗？例如，用π做底数
- 可以
- 例子
  - Bergman's number system (the τ number system)
    - Can represent some irrational numbers exactly in finite digits
    - Created by George Bergman, a 12-year junior high school student
    - Base $b = \tau = (1 + \sqrt{5})/2 \approx 1.6180339$, digit set = {0, 1}
    - $14 = \mathbf{100100.110110} = \tau^5 + \tau^2 + \tau^{-1} + \tau^{-2} + \tau^{-4} + \tau^{-5}$
  - Fibonacci number system (FNS)
    - Fibonacci numbers 1, 2, 3, 5, 8, … as positional weights
    - digit set = {0, 1}
    - $14 = \mathbf{11001} = 1 \times 8 + 1 \times 5 + 0 \times 3 + 0 \times 2 + 1 \times 1$

位值记数法的五种例子

Decimal	Hexadecimal	Binary	The τ Number System	FNS
$10^1 10^0$	16^0	$2^3 2^2 2^1 2^0$	$\tau^5\tau^4\tau^3\tau^2\tau^1\tau^0\tau^{-1}\tau^{-2}\tau^{-3}\tau^{-4}\tau^{-5}\tau^{-6}$	8 5 3 2 1
0	0	0000	0	00000
1	1	0001	1	00001
2	2	0010	10.01	00010
3	3	0011	100.01	00100
4	4	0100	101.01	00101
5	5	0101	1000.1001	01000
6	6	0110	1010.0001	01001
7	7	0111	10000.0001	01010
8	8	1000	10001.0001	10000
9	9	1001	10010.0101	10001
10	A	1010	10100.0101	10010
11	B	1011	10101.0101	10100
12	C	1100	100000. 101001	10101
13	D	1101	100010.001001	11000
14	**E**	**1110**	**100100.110110**	**11001**
15	F	1111	100101.001001	11010

3.2 如何操作比特?

- Answer: Operate on the byte or int variable containing the bit
  - Through some mask mechanism
- Example: Inverting the rightmost bit of a byte 最右比特取非
  - Input: 0011111**1**; Output: 0011111**0**
- Code explained with the corresponding operations

```
x := byte(63)       // assign 63=00111111 to variable x
v := ^x             // bitwise NOT of x, i.e., v=11000000
v = v & 0x1         // bitwise AND to retain the right-most bit of v
x = x & 0xFE        // bitwise AND to clear the right-most bit of x
x = x | v           // bitwise OR to get the final result
```

Mask mechanism
掩码机制

x = 0011111**1** Given input

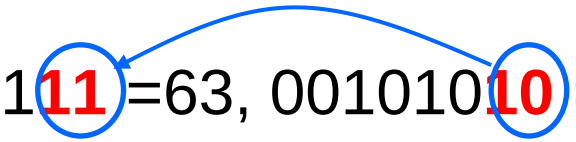
$v = \overline{0}\ \overline{0}\ \overline{1}\ \overline{1}\ \overline{1}\ \overline{1}\ \overline{1}\ \overline{1} = 1100000$**0** Bitwise NOT

v = 11000000 & 00000001 = 0000000**0** Bitwise AND

x = 00111111 & 11111110 = 0011111**0** Bitwise AND

x = 00111110 | 00000000 = 0011111**0** Bitwise O

置换字节的最低有效位两位
(used in Project Text Hider)



- Input: 00111111 =63, 00101010 =42; Output: 00111110
- Code explained with the corresponding operations

```
x := byte(63)       // assign 63=00111111 to variable x
v := byte(42)       // assign 42=00101010 to variable v
v = v & 0x3         // bitwise AND to retain the right-most 2 bits of v
x = x & 0xFC        // bitwise AND to clear the right-most 2 bits of x
                            // retaining the leftmost 6 bits
x = x | v           // bitwise OR to get the final result
```

Mask mechanism

x = 00111111		Given input
v = 00101010		Given input
v = 00101010 & 00000011	= 00000010	Bitwise AND
x = 00111111 & 11111100	= 00111100	Bitwise AND
x = 00111100 | 00000010	= 00111110	Bitwise OR

Note: 0x3 = 00000011; 0xFC = 00111100

***3.3 IEEE 754 浮点数

Use floating-point numbers to represent reals

- 有同学在个人作品实验使用
- 如何表示π ≈ 3.1415927 ?
  - A simple way: converting whole and fraction into binary
    - 3.1415927 = 11.00100100001111111011011
  - Another way: 3.1415927 = 3.1415927×10^0
  - May not be unique
    - $3.1415927 \times 10^0 = 31415927 \times 10^{-7} = 0.31415927 \times 10^1 = \ldots$
    - 科学计数法要求小数点在尾数的第一数位之后

指数
Exponent

Significant 尾数

IEEE 754 浮点数的特色

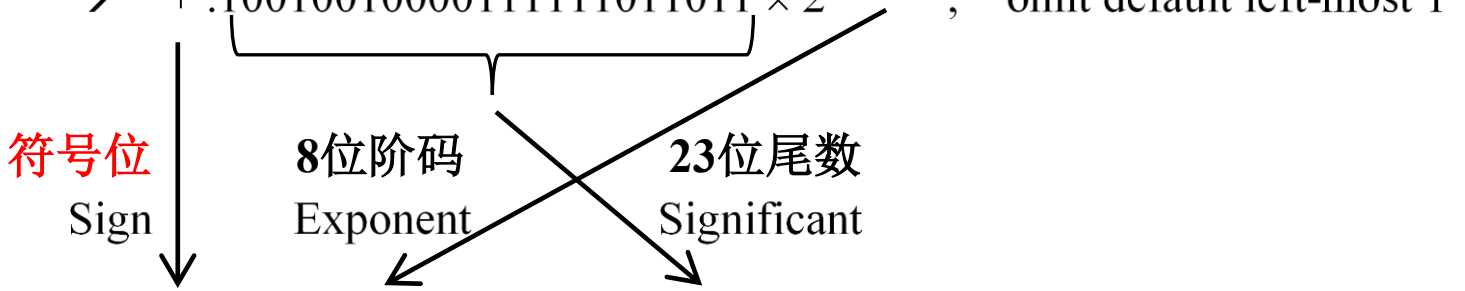
Use floating-point numbers to represent reals

- 规格化数的巧妙尾数表示
  - Default left-most 1
- 用移码表示阶码（Biased exponent）to speed up comparison
  - 00000001 + 127 = 128 = 10000000

$\pi \approx 3.1415927 \times 10^0 \approx 1.5707964 \times 2^1$

$\approx \quad +1.10010010000111111011011 \times 2^{00000001}$;

$\rightarrow \quad +.10010010000111111011011 \times 2^{00000001}$; omit default left-most 1



$= \quad \mathbf{0}\mathbf{10000000}10010010000111111011011$; the IEEE 754 representation

IEEE 754 浮点数的特色　为什么获图灵奖？

- 定义了五类浮点数，支持代数完备性
  - 规格化数、非规格化数、零、无穷大、非实数（NaNs，如$\sqrt{-5}$）
  - W := 7.0/(Y/0.0)/(Z/0.0)产生正确结果0.0，而不是由于除零异常而中止

比特	31	30 29……23	22 21 …… …… 1 0	值
规格化数	S	非全0非全1；$0 < E < 255$	规格化尾数M	$(-1)^S \times 1.M \times 2^{(E-127)}$
最大	0	11111110	11111111111111111111111	$1.M \times 2^{127}$
π	0	10000000	10010010000111111011011	$1.570796 \times 2^1 \approx 3.141592$
最小	0	00000001	00000000000000000000000	$1.0 \times 2^{(1-127)} = 2^{-126}$
非规格化数	S	00000000	非全零尾数M	$(-1)^S \times 0.M \times 2^{-126}$
最大	0	00000000	11111111111111111111111	$(1 - 2^{-23}) \times 2^{-126}$
最小	0	00000000	00000000000000000000001	$2^{-23} \times 2^{-126} = 2^{-149}$
零	0	00000000	00000000000000000000000	零 0
	1	00000000	00000000000000000000000	负零 −0
无穷大	0	11111111	00000000000000000000000	+∞
	1	11111111	00000000000000000000000	−∞
非实数	S	11111111	非全零尾数	NaN

浮点数往往是近似值，不应该使用==做比较操作

- How to test the equality of two floating-point numbers?

```
> go run ./testPoint123.go
0.1+0.2 == 0.3
0.1+0.2 != 0.3
0.1+0.2 == 0.3
>
```

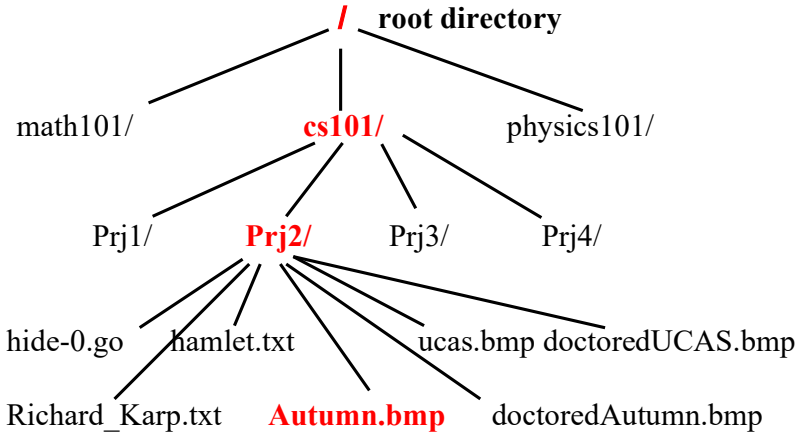
There may be a small difference between 0.1+0.2 and 0.3

```
package main   // testPoint123.go
import "fmt"
import "math"
func main() {
        if 0.1 + 0.2 == 0.3 {
                fmt.Println("0.1+0.2 == 0.3")
        } else {
                fmt.Println("0.1+0.2 != 0.3")
        }
        X := 0.1          // var X float64 = 0.1
        Y := 0.2
        Z := 0.3
        if X + Y == Z {
                fmt.Println("0.1+0.2 == 0.3")
        } else {
                fmt.Println("0.1+0.2 != 0.3")
        }
        if math.Abs(X+Y - Z) < math.Pow(10, -12) {
                fmt.Println("0.1+0.2 == 0.3")
        } else {
                fmt.Println("0.1+0.2 != 0.3")
        }
}
```

- To test if (X+Y) is equal to Z
  - Don't use (X+Y)==Z
  - Test if the absolute value of the difference is less than epsilon, i.e., use Abs(X+Y-Z) < 10^{-12}

4. The file abstraction　文件与文件系统

- **持久存储**数据和程序文件，下电后还存在
- 文件系统是一颗树
  - 叶子节点（文件），内部节点（目录）
  - 文件名（路径）；绝对路径，相对路径
  - 目录；缺省目录，当前目录，父目录，根目录
- 数据与元数据
  - 文件格式，文件大小，访问权限
- 如何读入文件
  - 有利于定位及操作
- 如何支持循环
  - 迭代操作文件元素



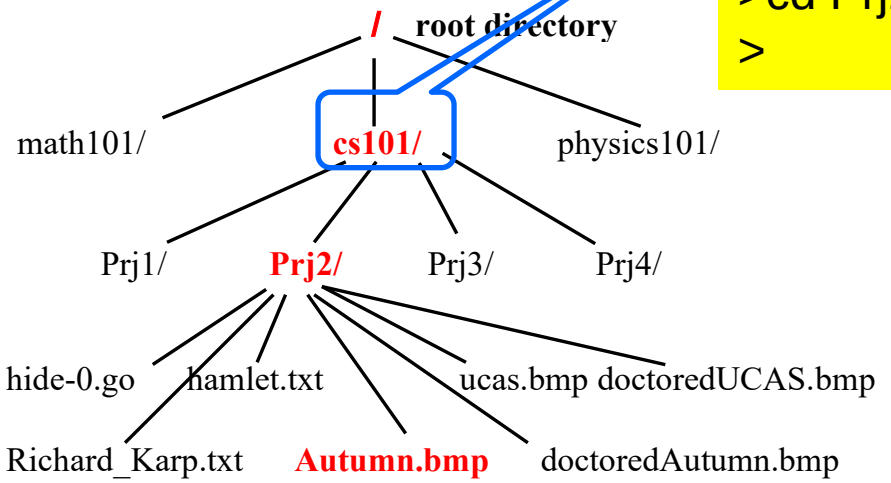
4.1 文件与文件系统

- 文件系统是一颗树
  - Leaf nodes are files 文件; internal nodes are directories 目录 (special files)

- 用文件名（又称为文件路径，简称路径）指明某一文件
  - Absolute path 绝对路径: all the way from the root (/)
    - Absolute path of **Autumn.bmp**: /cs101/Prj2/Autumn.bmp
  - Relative path 相对路径 of **Autumn.bmp**
    - Related to the *current directory* 当前目录，
      i.e., *working directory* 工作目录
    - ./Autumn.bmp if the working directory is /cs101/Prj2/
    - ../Prj2/Autumn.bmp if the working directory is /cs101/Prj2/
  - Home directory 缺省目录
    - The default directory when log in. Assume /cs101 is the home directory

主目录是/cs101/

```
>pwd
/cs101/
>cd Prj2
>
```



Hide text in a picture

- Write a program hide-0.go
  - to hide the text of Shakespeare's *Hamlet*
    - in a picture file Autumn.bmp
    - such that the doctored file shows no visible difference from the original picture
- and another program show-0.go to recover the text

HAMLET

DRAMATIS PERSONAE

CLAUDIUS king of Denmark. (KING CLAUDIUS:)

HAMLET son to the late, and nephew to the present king.

…..

ACT ISCENE I Elsinore. A platform before the castle.
…..
PRINCE FORTINBRAS Let four captains
Bear Hamlet, like a soldier, to the stage;
For he was likely, had he been put on,
To have proved most royally: and, for his passage,
The soldiers' music and the rites of war
Speak loudly for him.
Take up the bodies: such a sight as this
Becomes the field, but here shows much amiss.
Go, bid the soldiers shoot.
[A dead march. Exeunt, bearing off the dead
bodies; after which a peal of ordnance is shot off]
…..
] 换行键0x0A

Original picture

After careless hiding

After careful hiding

一个动态网页思路

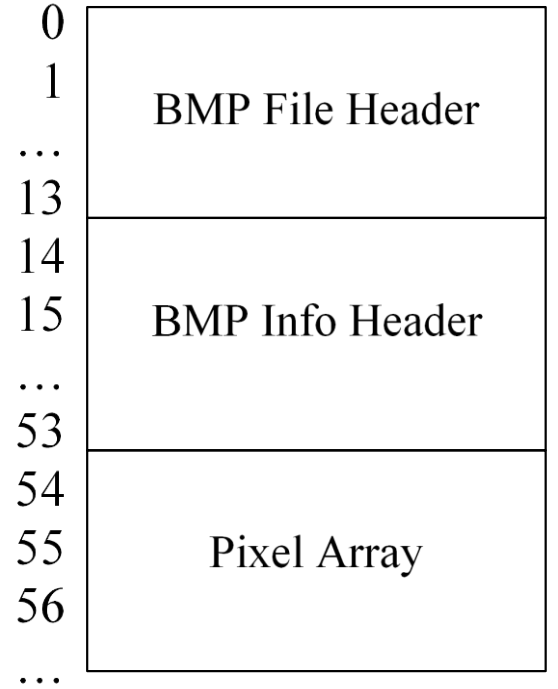
4.2 数据与元数据

- A file is a group of bits ***and*** may be structured
  - 文件是一组比特，可能是有结构的
    - 无结构例子：F(500) = **1394 2322 4561 6978 8013 9724 3828 7040 7283 9500 7025 6587 6973 0726 4108 9629 4832 5571 6228 6329 0691 5576 5887 6222 5212 94125**
- 有结构例子：Data and metadata of file Autumn.bmp
  - Data: bits of the actual picture (Pixel Array)
  - Metadata 元数据 : data about the picture data 文件格式
    - BMP format in the file: File Header, Info Header
    - Other data associated with the file

Addresses 0~53 hold metadata

The pixel array for actual image data starts at address 54

Autumn.bmp



The extension (扩展名) bmp says it's a bit map image file

Other types of metadata

- Can be seen by running “ls -l Autumn.bmp”
  - ls -l Autumn.bmp中，l是小写的L，不是大写的I

  > -rw-rw-rw- 1 zxu zxu 9144630 Jul 22 2020 Autumn.bmp
  - The file name, the file size, the time of creation (last modification)
  - Access permissions 三类三种权限
    - Rights to read, write, and execute a file by the owner of the file, by the group the owner belonging to, and by other users
- Example of access permissions
  - ioutil.WriteFile("./doctoredAutumn.bmp", p, 0666)
  - Every user can read and write, but cannot execute
    - -rw-rw-rw-
    - 0666 = 0110110110

提醒：多数Linux系统会设置一个umask值，来避免用户创建出危险的高权限文件。默认umask值为0022，因此以0666创建文件时，实际创建出的文件权限为0666 & (~0022) = 0666 & 0755 = 0644

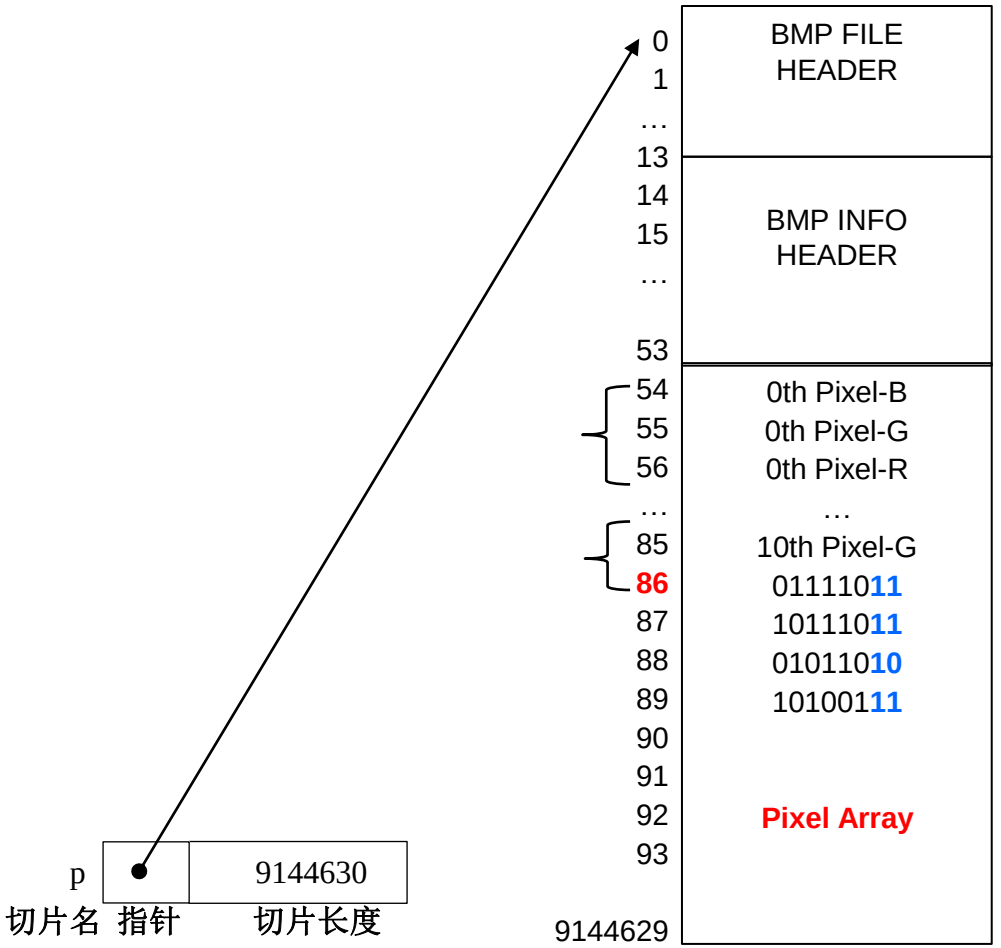
Owner			Group			Others		
r	w	e	r	w	e	r	w	e
1	1	-	1	1	-	1	1	-

0666：拥有者（用户本人）、组、他人可读、可写，不可执行
这是一个数据文件，不是程序文件

0700表示什么权限？

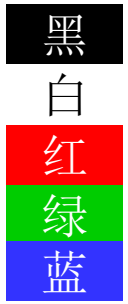
4.3 如何读文件

- 将文件以合适的格式读进内存，便于**定位**
  - 后续操作方便地定位到文件所需区域并处理该区域的数据
- 什么是合适的格式呢？
  - 是**字节切片**（byte slice）
- 使用Go语言提供的库函数
  - p, _ := ioutil.ReadFile("./Autumn.bmp")
  - 将当前目录中的Autumn.bmp图像文件读进内存的字节切片变量p
  - 文件内容被读进内存，形成一个包含9144630个元素的字节数组，成为切片p的底层数组

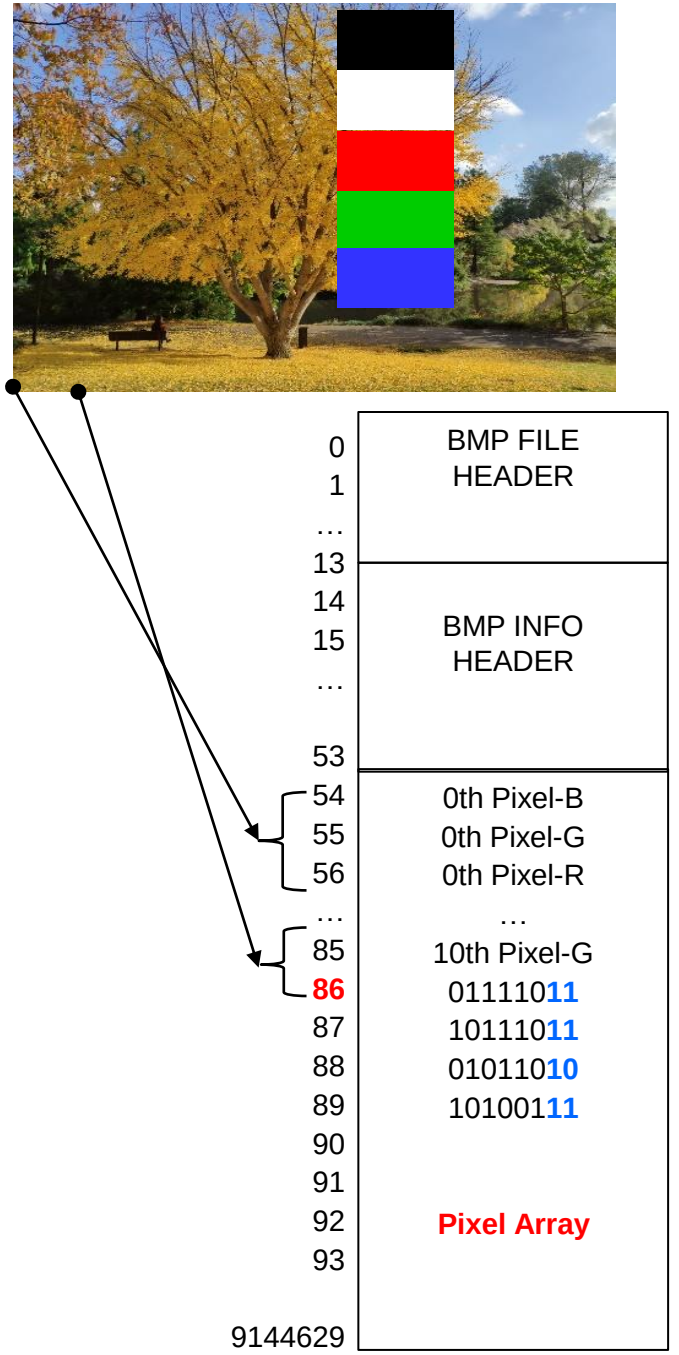


How is the image of Autumn.bmp stored in Pixel Array?

- Pixel = Picture Element
- Pixel Array holds the pixels of the image, starting from the low-left corner going right, and then up, row by row
- Each pixel has three bytes for
  - Color depth values of RGB, i.e., the primary colors of red, green, blue
  - BGR values = (0, 0, 0) → Black
  - BGR values = (255, 255, 255) → White
  - BGR values = (0, 0, 255) → Red
  - BGR values = (0, 255, 0) → Green
  - BGR values = (255, 0, 0) → Blue

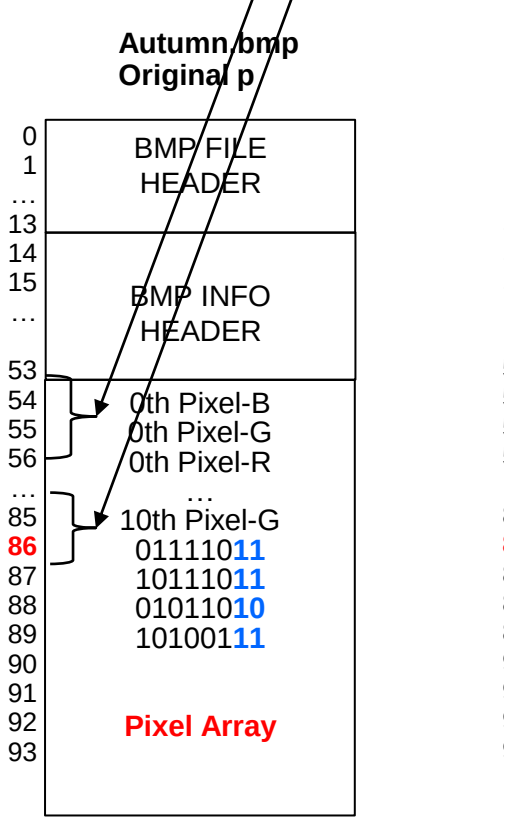


- The first pixel is the 0th element of Pixel Array
  - Uses addresses 54, 55, and 56 to store its three BGR color depth values



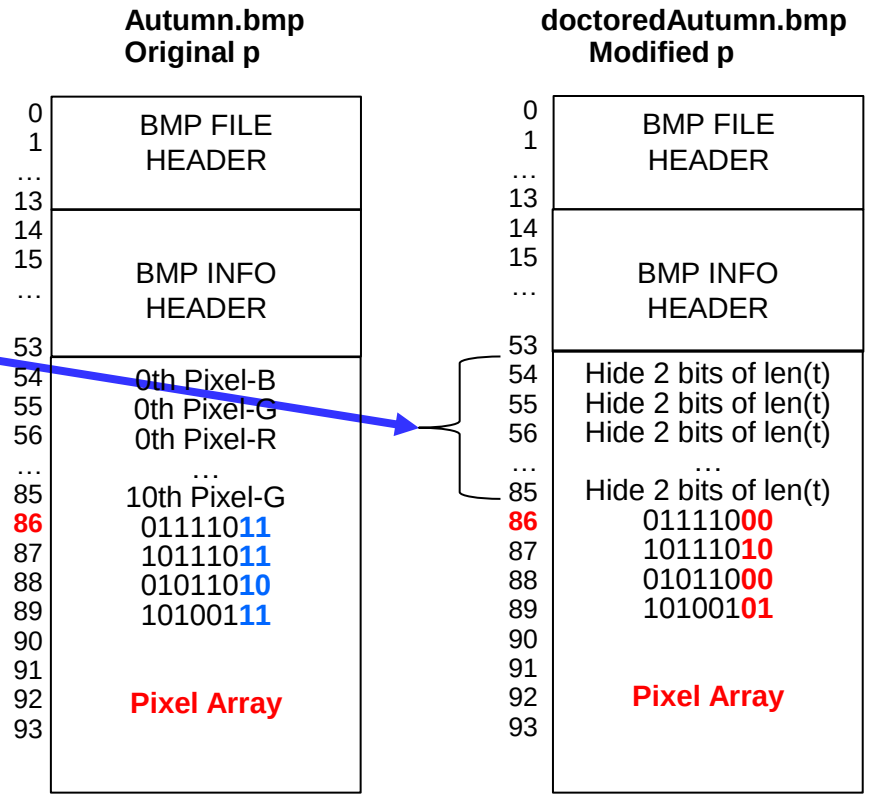
4.4 如何在图片文件中隐藏信息

- 如何在图片文件Autumn.bmp中隐藏文本文件hamlet.txt的长度
- p, _ := ioutil.ReadFile("./Autumn.bmp") to read the image file into byte slice p
- Recall that a function can return multiple values
  - This function returns two values, the second of which is not needed by this code
  - Use a placeholder symbol '_'
  - Also called the blank identifier



How to hide the length of a text file in a picture?

- p, _ := ioutil.ReadFile("./Autumn.bmp")
  to read the image file into byte slice p
- t, _ := ioutil.ReadFile("./hamlet.txt")
  to read the text file into byte slice t
- Length len(t) is a 64-bit integer
  - Hide every 2 bits in a byte of p
  - Need 32 bytes
  - S = 54, T = 32
- modify(len(t), p[S:S+T], T)
  to hide len(t) in p[54:86]



How to hide the length of a text file in a picture?

- p, _ := ioutil.ReadFile("./Autumn.bmp")
  to read the image file into byte slice p
- t, _ := ioutil.ReadFile("./hamlet.txt")
  to read the text file into byte slice t
- Length len(t) is a 64-bit integer
  - Hide every 2 bits in a byte of p
  - Need 32 bytes
  - S = 54, T = 32
- modify(len(t), p[S:S+T], T)
  to hide len(t) in p[54:86]

```
func modify(txt int, pix []byte, size int) {
   for i := 0; i < size; i++ {
      replace last 2 bits of pix[i]
           with the last 2 bits of txt
      repeat with the next 2 bits of txt
   }
}
```

Autumn.bmp
Original p

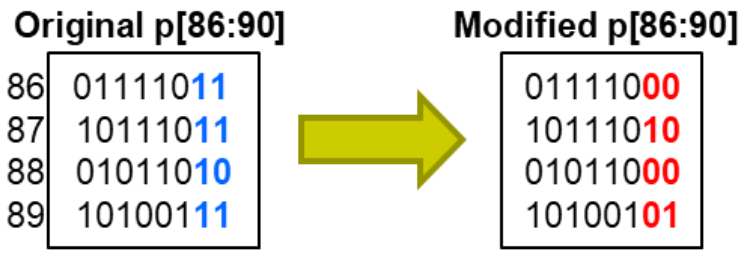
0 1 … 13	BMP FILE HEADER
14 15 … 53	BMP INFO HEADER
54 55 56 … 85 86 87 88 89 90 91 92 93	0th Pixel-B 0th Pixel-G 0th Pixel-R … 10th Pixel-G 01111011 10111011 01011010 10100111 **Pixel Array**

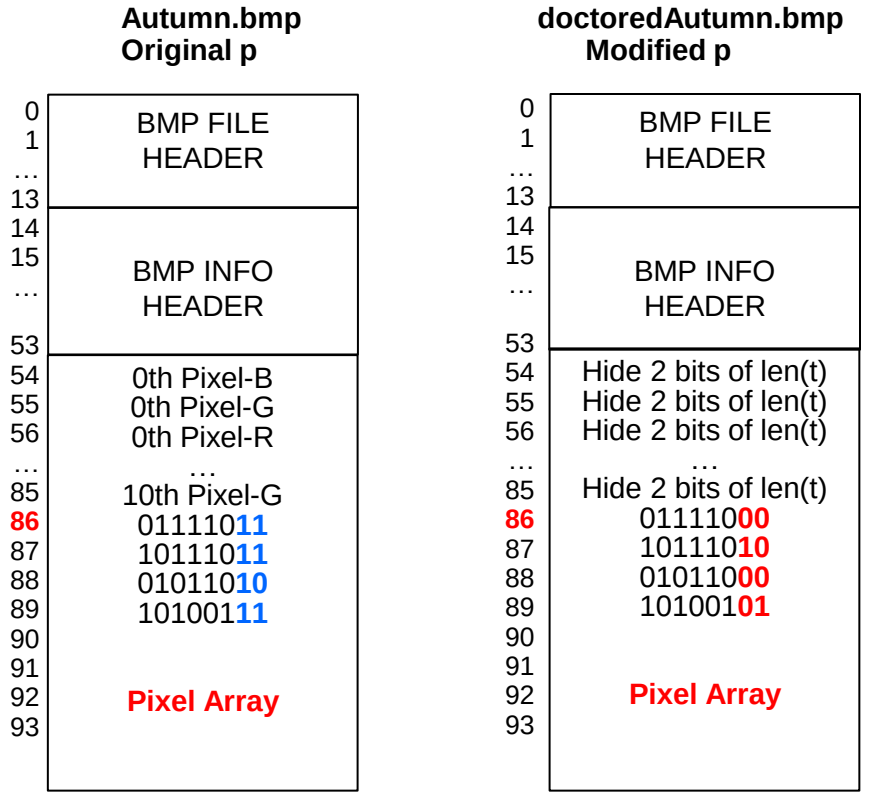
doctoredAutumn.bmp
Modified p

0 1 … 13	BMP FILE HEADER
14 15 … 53	BMP INFO HEADER
54 55 56 … 85 86 87 88 89 90 91 92 93	Hide 2 bits of len(t) Hide 2 bits of len(t) Hide 2 bits of len(t) … Hide 2 bits of len(t) 01111000 10111010 01011000 10100101 **Pixel Array**

How to hide the contents of a text file in a picture?

- p, _ := ioutil.ReadFile("./Autumn.bmp")
  to read the image file into byte slice p
- t, _ := ioutil.ReadFile("./hamlet.txt")
  to read the text file into byte slice t
- t[0] holds the 1st character ‘H’ = 72
- modify(int(t[0]), p[S+T:S+T+C], C)
  where
  - t[0] is ‘H’ = 72 = **01001000**
  - S = 54, T = 32, C is 4
  - p[S+T:S+T+C] is p[86:90]





How to hide the contents of a text file in a picture?

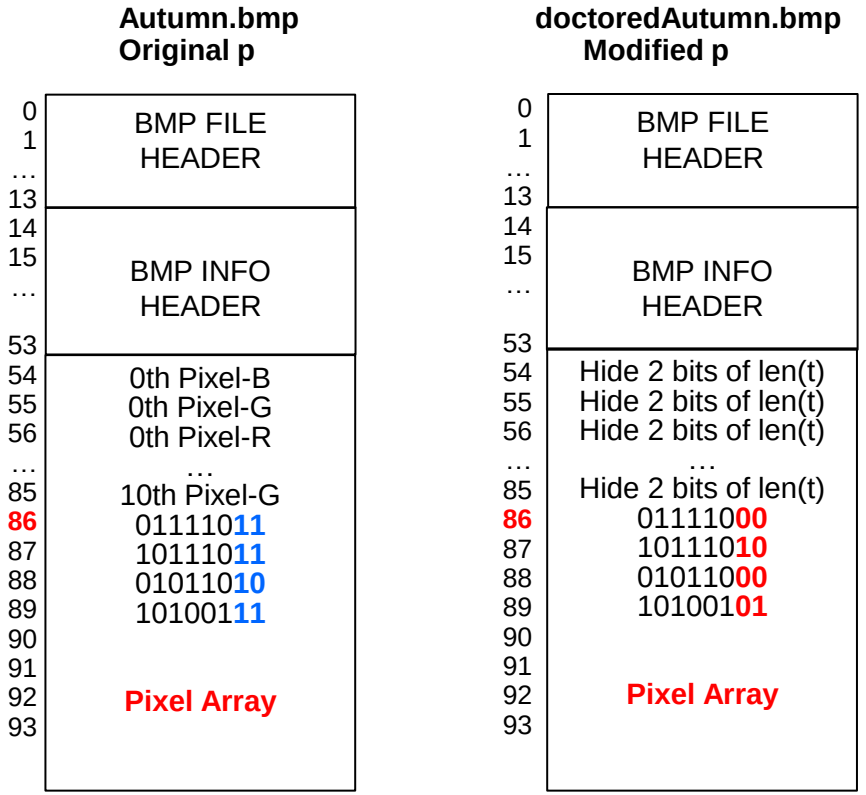
- p, _ := ioutil.ReadFile("./Autumn.bmp")
  to read the image file into byte slice p
- t, _ := ioutil.ReadFile("./hamlet.txt")
  to read the text file into byte slice t
- 使用循环语句隐藏t[i], i = 0~len(t)

```
for i:=0; i<len(t); i++{
   offset := S+T+(i*4)
   modify(int(t[i]), p[offset:offset+C], C)
}
```

比例因子是4，因为每个字符有8位，需要p的4个字节。这4个字节的偏移量分别是0、1、2、3。第0次迭代执行modify(int(t[0]), p[86:90], 4)，修改P[86]， P[87]， P[88]， P[89]

基址+索引*4+偏移量

- 回忆支持循环的基址+索引+偏移量寻址模式

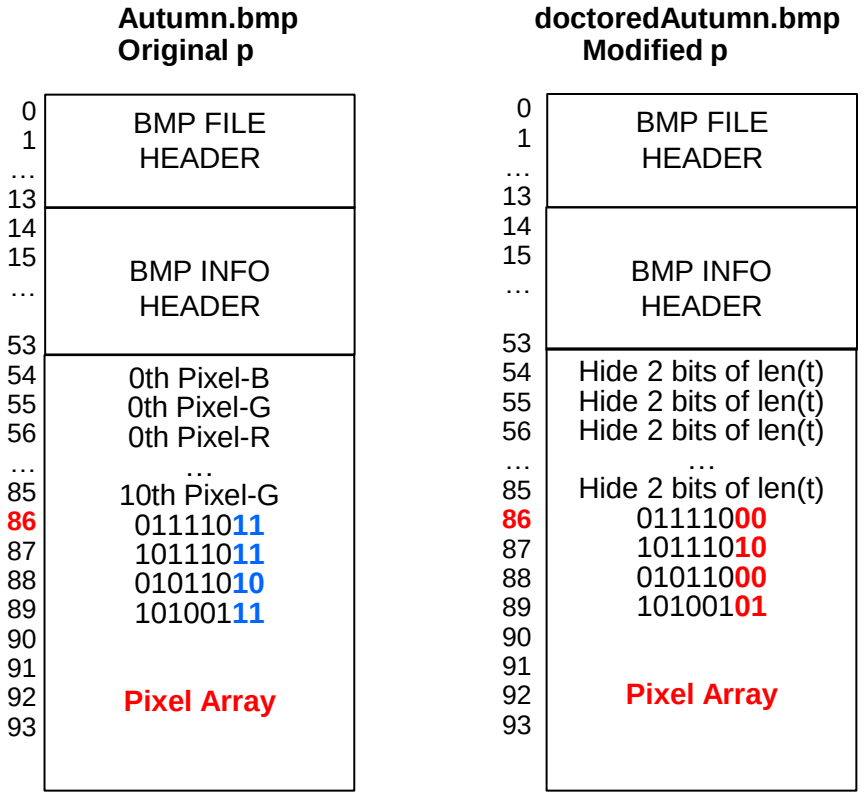


How to hide the contents of a text file in a picture?

- p, _ := ioutil.ReadFile("./Autumn.bmp")
  to read the image file into byte slice p
- t, _ := ioutil.ReadFile("./hamlet.txt")
  to read the text file into byte slice t
- To hide all t[i], i = 0 to len(t)

```
for i:=0; i<len(t); i++{
   offset := S+T+(i*4)
   modify(int(t[i]), p[offset:offset+C], C)
}
```

- Each iteration hides t[i] in p[S+T+(i*4):S+T+(i*4)+C]
  - Where S = 54, T = 32, C = 4
- That is, t[i] is hidden in p[86+(i*4)], p[86+(i*4)+1], p[86+(i*4)+2], p[86+(i*4)+3]
- E.g., t[1]='A', is hidden in p[90:94]



Check results

- Write the complete hide-0.go
- Execute and display result

> go run hide-0.go

> display doctoredAutumn.bmp

- The Text Hider project
  - Produce hide.go with good coding practices
  - Also need to write show.go

- Change hide-0.go to hide-1.go
  - by modifying the most significant 2 bits (the rightmost 2 bits) of each byte of Pixel Array

> go run hide-1.go

> display doctoredAutumn.bmp

Original Autumn.bmp

doctoredAutumn.bmp
Modifying rightmost 2 bits

doctoredAutumn.bmp
Modifying leftmost 2 bits

谢谢 Thank You

Q&A

zxu@ict.ac.cn

中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY

进阶知识（***）结构体与指针

- Array is simple
  - Linear, consecutive arrangement of N elements of the same type
  - Example: var A [5]byte defines a byte array A of 5 elements
    - A[0], A[1], A[2], A[3], A[4] are all of type byte　　数组：表示一组连续存储的同构（同类）元素
- What if the elements are of different types?　不同类怎么办？

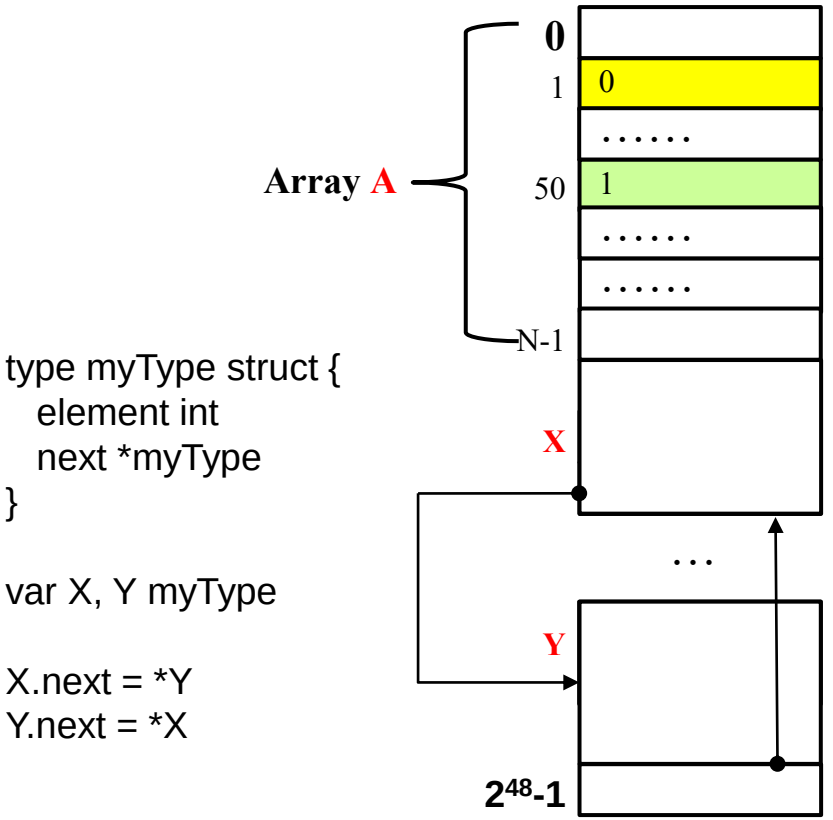
- Use **struct**　用结构体表示一组异构元素
  - A data structure with different types of elements
  - Elements are called fields
  - Use the dot notation to access fields, e.g.,
    - JoanSmith.ID accesses the student ID of Joan Smith，不是JoanSmith[1]
    - FanWang.active = false assigns the false value to the active field of student Fan Wang, indicating that he is not actively enrolled
  - 对比：数组使用索引（index）指向特定元素，如A[3]指定3号元素

```
type Student struct {
   name          string
   ID            int
   majorCode     byte
   active        bool
   contact       string
}
var JoanSmith, FanWang Student
```

Pointers and addressing modes
指针与寻址模式

- Array is simple
  - Linear, consecutive arrangement of N elements of the same type
    数组：一组顺序连续的同类数据
  - Example: a byte array A
    - Next element of A[i] is A[i+1]
    - If A[i] is at address 50, A[i+1] is at 51
  - What if not consecutive?
    假如不连续怎么办？

- Pointer brings flexibility
  - Nonlinear arrangements, where the elements can jump around
  - Example: two variables X and Y connected by pointers
    - Indicated by the two arrows

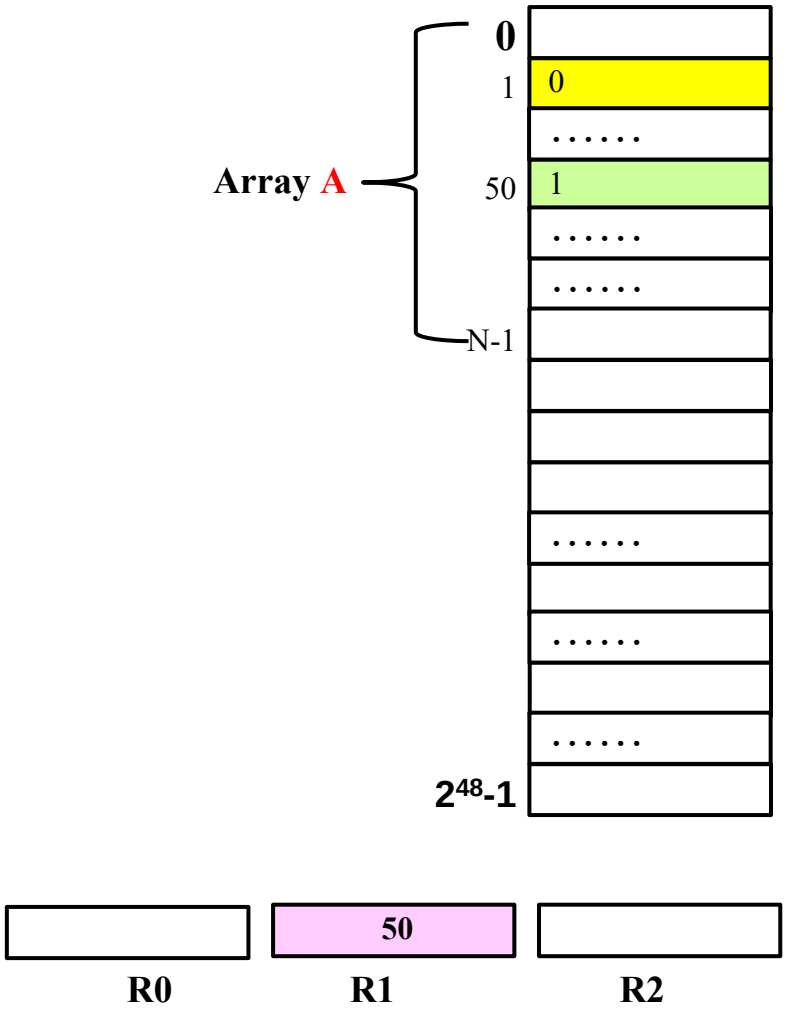


The number 48 is due to the fact that Golang has a 48-bit virtual memory space

三类寻址模式
Contrasting three addressing modes

- Array is simple
  - Linear, consecutive arrangement
- Pointer brings flexibility
  - Nonlinear arrangements

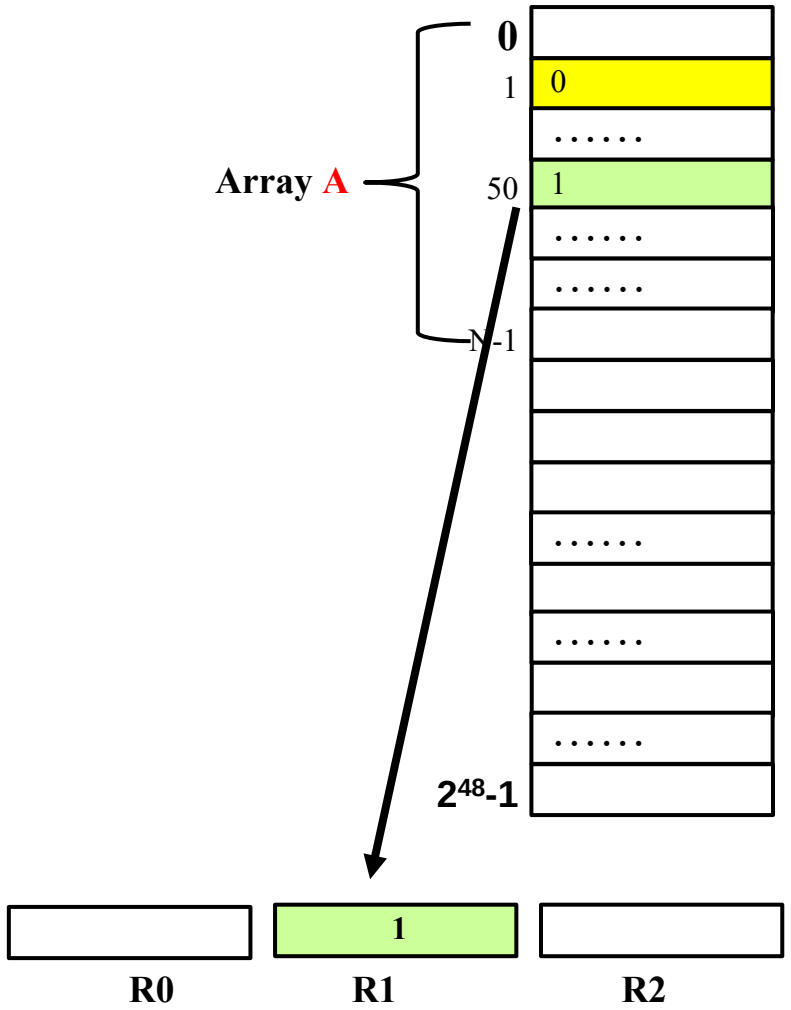
- Pointers are supported by the *indirect addressing* mode
  - Contrasting three addressing modes
    - **Immediate addressing** 立即数寻址: MOV 50, R1;
      - 50 → R1, i.e., R1=50
    - No memory access



Contrasting three addressing modes

- Array is simple
  - Linear, consecutive arrangement
- Pointer brings flexibility
  - Nonlinear arrangements

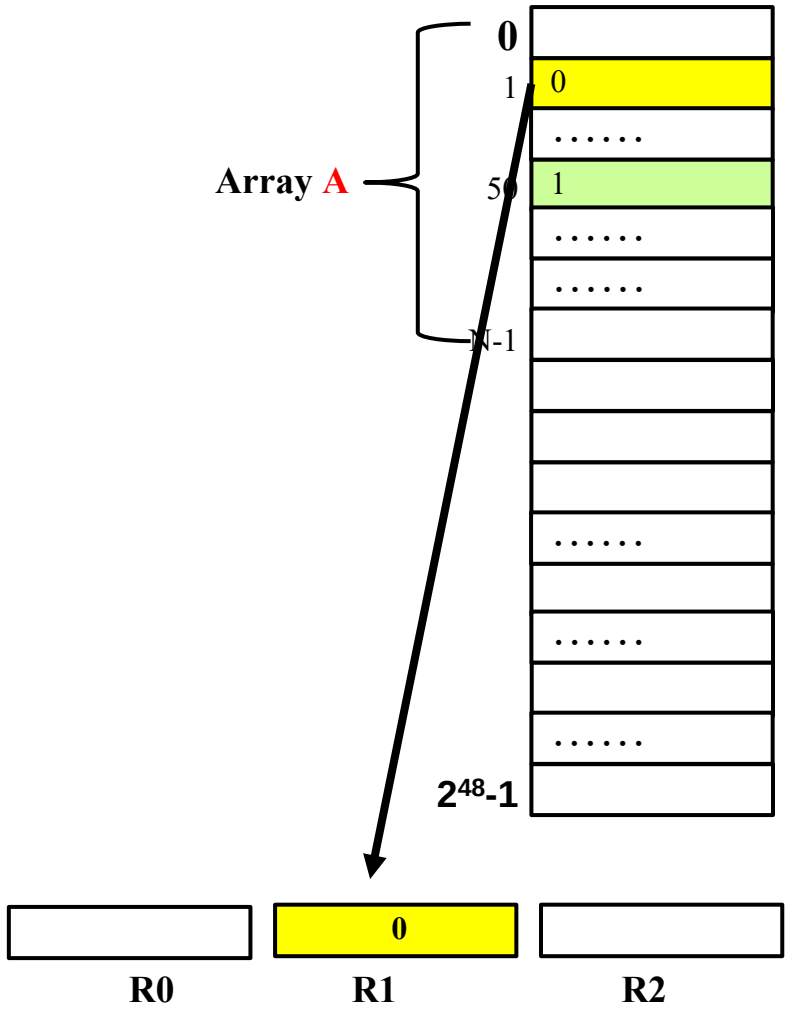
- Pointers are supported by the *indirect addressing* mode
  - Contrasting three addressing modes
    - Immediate addressing 立即数寻址: MOV 50, R1;
      - 50 → R1, i.e., R1=50
    - **Direct addressing** 直接寻址: MOV M[50], R1;
      - M[50] → R1, i.e., R1=1
      - Often the common case



Contrasting three addressing modes

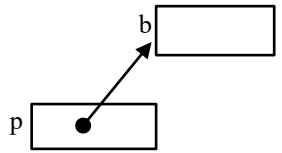
- Array is simple
  - Linear, consecutive arrangement
- Pointer brings flexibility
  - Nonlinear arrangements

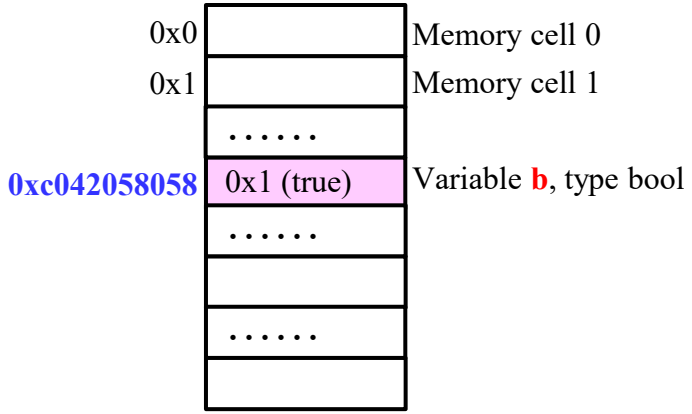
- Pointers are supported by the *indirect addressing* mode
  - Contrasting three addressing modes
    - Immediate addressing 立即数寻址: MOV 50, R1;
      - 50 → R1, i.e., R1=50
    - Direct addressing 直接寻址: MOV M[50], R1;
      - M[50] → R1, i.e., R1=1
    - **Indirect addressing** 间接寻址: MOV M[M[50]], R1;
      - M[M[50]] → R1, i.e., M[M[50]] is M[1], R1=M[1]=0
      - First get the address
      - Then access for the value



逐步展示指针行为
Step-by-step Illustration of pointers

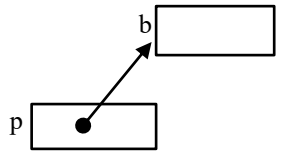
```
func main() {
   b := true            // Boolean variable b
   p := & b             // p holds b's address
   fmt.Println(p)       // Print b's address
   fmt.Println(*p)      // Print b's value; dereference p
   *p =  false          // Modify b's value
   fmt.Println(b)       // Print b's value
   *p = !(*p)           // Use and modify b's value by negation
   fmt.Println(b)
}
```





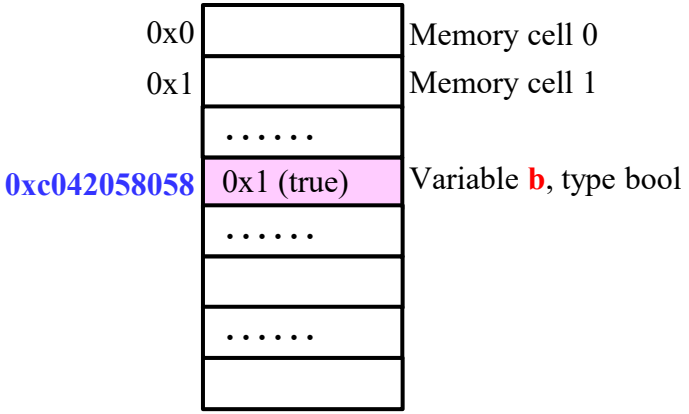
Step-by-step Illustration of pointers

```
func main() {
   b := true          // Boolean variable b
   p := & b           // p holds b's address
   fmt.Println(p)     // Print b's address
   fmt.Println(*p)    // Print b's value; dereference p
   *p =  false        // Modify b's value
   fmt.Println(b)     // Print b's value
   *p = !(*p)         // Use and modify b's value by negation
   fmt.Println(b)
}
```



1-minute Quiz:

What is the output of
the final statement fmt.Println(b)?

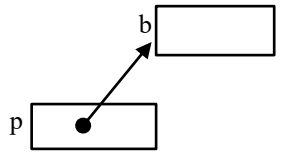


Step-by-step Illustration of pointers

- 理解指针的一个好办法是运行实例程序

```
func main() {
   b := true            // Boolean variable b
   p := & b             // p holds b's address
   fmt.Println(p)       // Print b's address
   fmt.Println(*p)      // Print b's value
   *p =  false          // Modify b's value
   fmt.Println(b)       // Print b's value
   *p = !(*p)           // Use and modify b's value by negation
   fmt.Println(b)
}

> go run ./pointer.go
0xc042058058
true
false
true
>
```



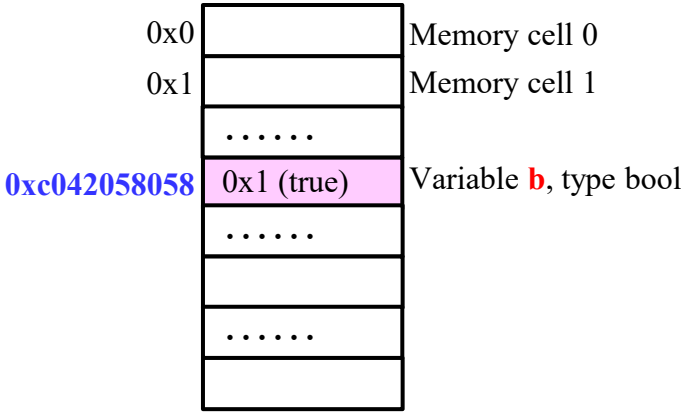
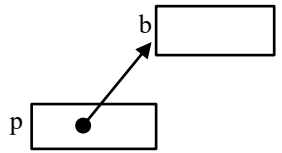


Illustration of pointers

```
func main() {
   b := true            // Boolean variable b
   p := & b             // p holds b's address   此处&是取址操作符
   fmt.Println(p)       // Print b's address
   fmt.Println(*p)      // Print b's value
   *p =  false          // Modify b's value
   fmt.Println(b)       // Print b's value
   *p = !(*p)           // Use and modify b's value by negation
   fmt.Println(b)
}

> go run ./pointer.go
0xc042058058
true
false
true
>
```



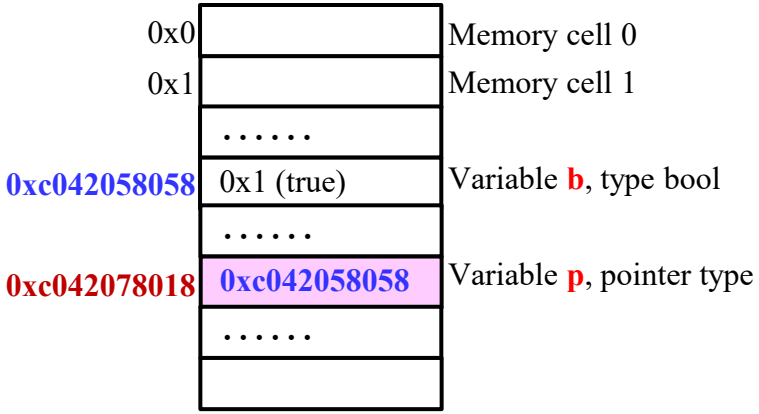


Illustration of pointers

```
func main() {
   b := true          // Boolean variable b
   p := & b           // p holds b's address
   fmt.Println(p)     // Print b's address
   fmt.Println(*p)    // Print b's value
   *p =  false        // Modify b's value
   fmt.Println(b)     // Print b's value
   *p = !(*p)         // Use and modify b's value by negation
   fmt.Println(b)
}

> go run ./pointer.go
0xc042058058
true
false
true
>
```

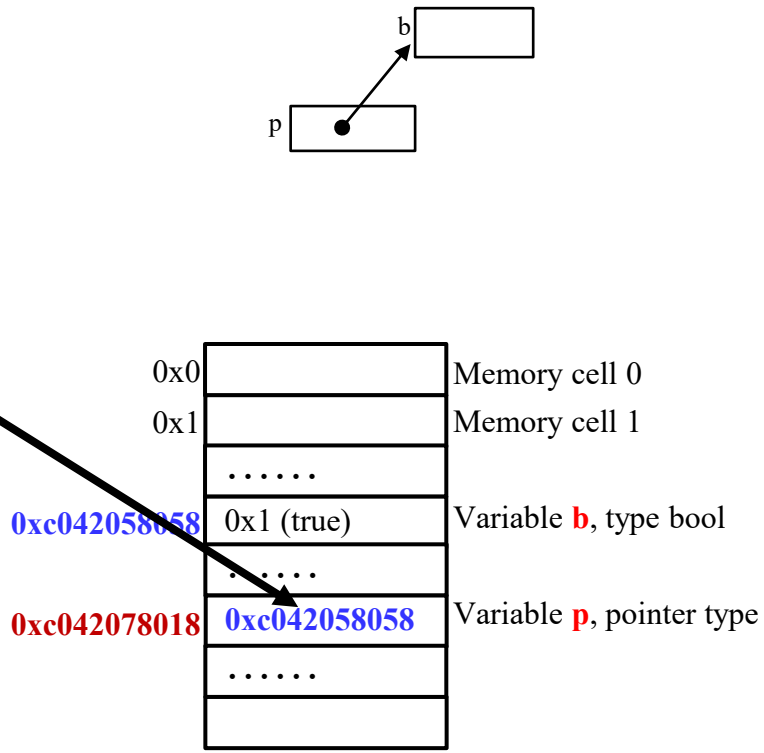


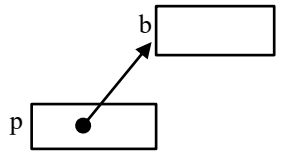
Illustration of pointers

```
func main() {
   b := true          // Boolean variable b
   p := & b           // p holds b's address
   fmt.Println(p)     // Print b's address
   fmt.Println(*p)    // Print what *p holds, i.e., b's value
   *p =  false        // Modify b's value
   fmt.Println(b)     // Print b's value
   *p = !(*p)         // Use and modify b's value by negation
   fmt.Println(b)
}

> go run ./pointer.go
0xc042058058
true
false
true
>
```

此处*是解引用操作符

间接寻址



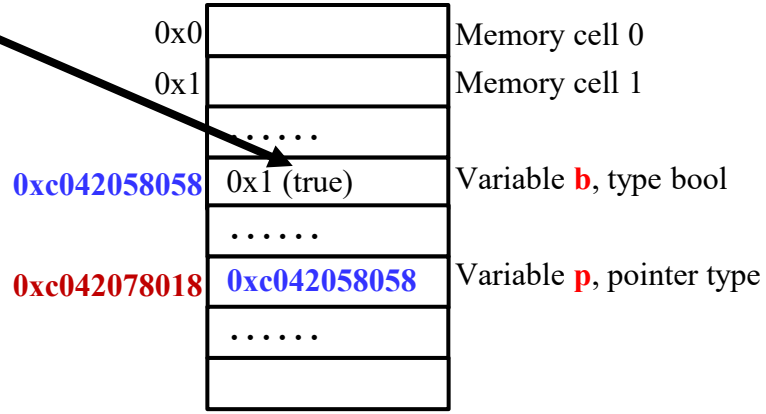
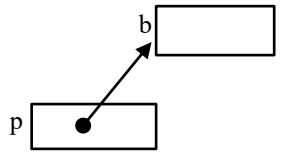


Illustration of pointers

```
func main() {
   b := true            // Boolean variable b
   p := & b             // p holds b's address
   fmt.Println(p)       // Print b's address
   fmt.Println(*p)      // Print b's value
   *p =  false          // Modify b's value  间接寻址
   fmt.Println(b)       // Print b's value
   *p = !(*p)           // Use and modify b's value by negation
   fmt.Println(b)
}

> go run ./pointer.go
0xc042058058
true
false
true
>
```



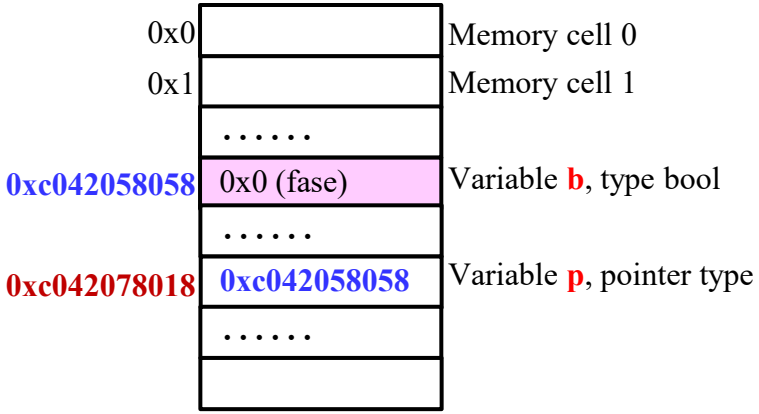
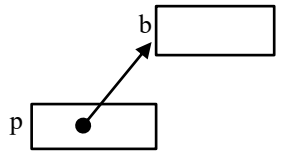


Illustration of pointers

```
func main() {
   b := true            // Boolean variable b
   p := & b             // p holds b's address
   fmt.Println(p)       // Print b's address
   fmt.Println(*p)      // Print b's value
   *p =  false          // Modify b's value
   fmt.Println(b)       // Print b's value
   *p = !(*p)           // Use and modify b's value by negation
   fmt.Println(b)
}

> go run ./pointer.go
0xc042058058
true
false
true
>
```



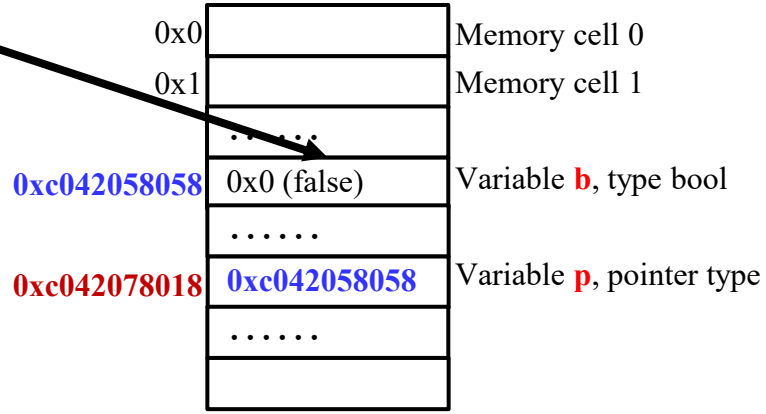
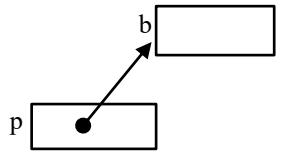


Illustration of pointers

```
func main() {
   b := true            // Boolean variable b
   p := & b             // p holds b's address
   fmt.Println(p)       // Print b's address
   fmt.Println(*p)      // Print b's value
   *p =  false          // Modify b's value
   fmt.Println(b)       // Print b's value
   *p = !(*p)           // Use and modify b's value by negation
   fmt.Println(b)
}
```

```
> go run ./pointer.go
0xc042058058
true
false
true
>
```



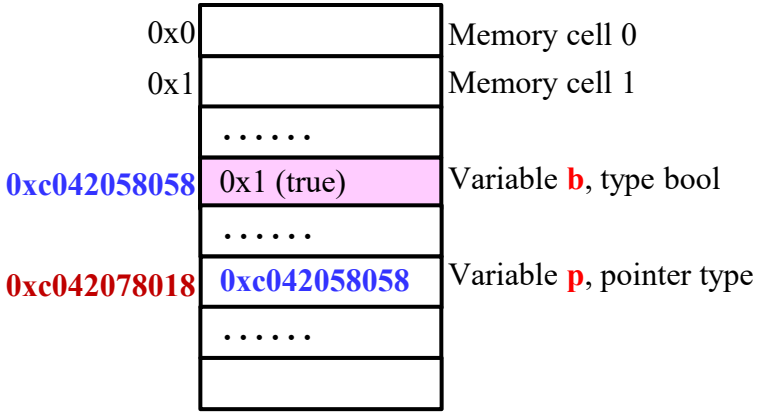


Illustration of pointers

```
func main() {
   b := true            // Boolean variable b
   p := & b             // p holds b's address
   fmt.Println(p)       // Print b's address
   fmt.Println(*p)      // Print b's value
   *p =  false          // Modify b's value
   fmt.Println(b)       // Print b's value
   *p = !(*p)           // Use and modify b's value by negation
   fmt.Println(b)
}

> go run ./pointer.go
0xc042058058
true
false
true
>
```