Computer Culture Advanced

信息表示与处理

数字化：人类感知世界到人机协同计算世界的桥梁

从硬件实验到上层编码

数字化：感知型 与 逻辑型

所有信息进入计算机前，都必须被**数字化**，不同的信息，有不同的**转换**策略。

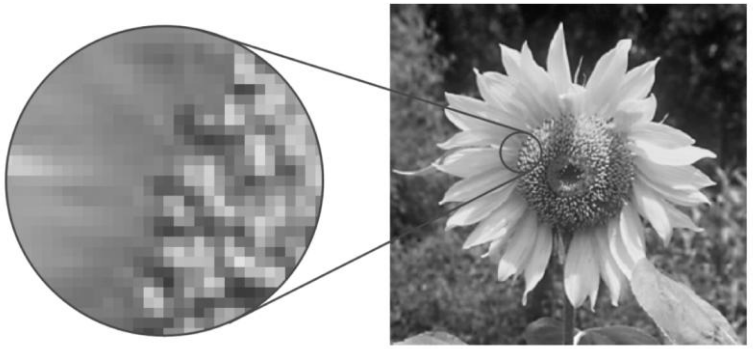
逻辑型数字化是基于人预先定义的**规则**完成**编码**表示。计算机硬件仅处理 0/1 符号，**含义、语义、结构是人通过编码规则赋予的**。

典型应用：数值编码、文本编码、矢量图形。

感知型数字化是对连续变化的物理世界，通过**采样**、**量化**，得到有限离散数值（对真实世界的**近似记录**），再对数值**编码**，变成计算机可存储、处理的 0/1 比特序列。典型应用：位图图像、音频和视频。

信息加工处理机器

（基于规则绘制的**图形**）可无限缩放



（基于有限记录还原的**图像**）缩放失真

本次课的主题：逻辑型编码

逻辑型 编码

人为定义编码规则 →将符号映射成 0/1比特数串 → 硬件 f 完成符号变换 → 解码 还原信息语义

$$y = f(x)$$

主要介绍

- **数值编码**（补码 / 浮点数）
- **文本编码**（ASCII / GB2312 / UTF8）
- **矢量图形编码**（SVG / AI / CDR）

带着这个问题听课：**为什么不同类型的信息要采用不同的编码方案?**

根本原因：**信息本身的结构特征不同，同时需要匹配不同的工程目标（硬件运算、存储开销、传输效率、解码复杂度等）**

数值数据编码：基于进位规则的唯一映射

二进制进位计数制

- 基数固定为 **2**（二进制），每位只有 0 或 1 两个符号，符合**逢二进一**原则
- 位置有权值，从右向左，第 i 位权重 = 2^i，最低位 i =0

$$(1101.11)_2 = 1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}$$

$$(13.75)_{10}$$

10^1 10^0 10^{-1} 10^{-2}

$$(13.75)_{10} = (1101.11)_2$$

2^{-1} 2^{-2}

$$(1101.11)_2$$

2^3 2^2 2^1 2^0

减法运算8-4 = ?

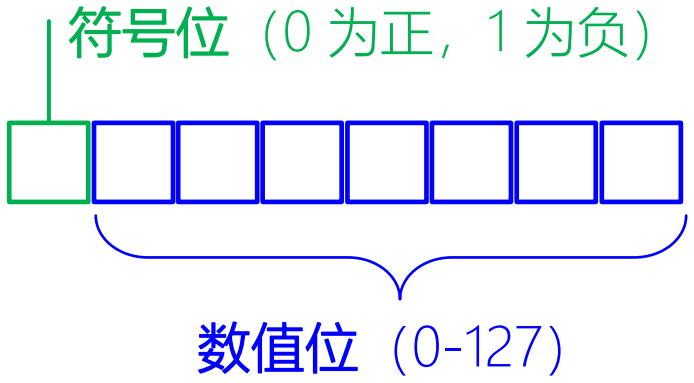
复用加法电路?

加法电路运算实现8+（-4）= ?

符号位怎么编码进二进制数中?

朴素符号位编码方案

采用符号-真值法的**编码**
比如，用 8 位二进制表示一个整数（-127 ~ +127）



计算一下：+5 ➕ -5

+5 表示为 00000101
（最高位 0 为正，后 7 位是 5 的二进制真值）

-5 表示为 10000101
（最高位 1 为负，后 7 位是 5 的二进制真值）

```
  00000101
➕ 10000101
----------
  10001010
```

- 加法逻辑混乱
- 存在两个 0
- 减法需要单独设计电路

朴素符号位编码符合人类直觉，但加法器直接计算会错

加法硬件复用引出补码规则

补码规则的底层逻辑（模运算）

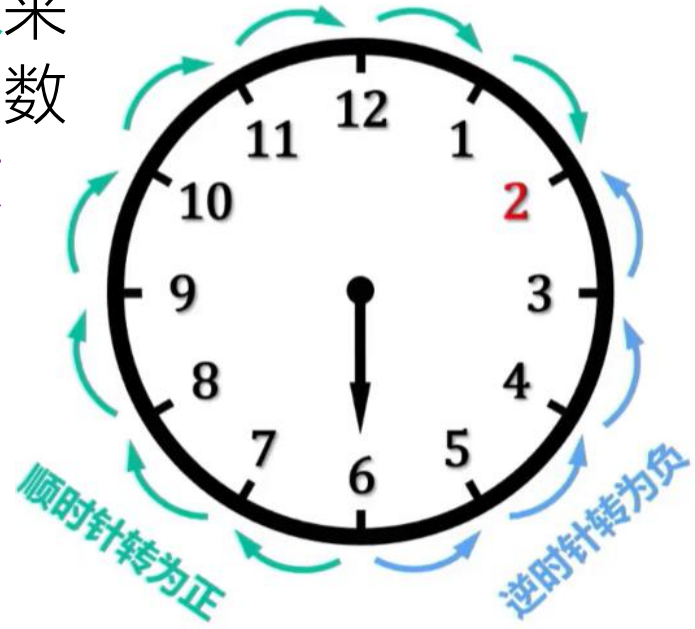
模是一个数值计量系统的计量范围（mod）。只要确定了**模**，就可找到一个**与负数等价的正数**来代替此负数，该正数就是负数的**补数**，超出计量范围的数都应该自动舍去模数。

指针从**6**拨到**2**的两种方式：

- 逆时针拨动4下（减法）
- 顺时针拨动8下（加法）

-4和8是一对**补数**

6-4 = 6+(12-4) = 6+8



钟表**模**运算

举例

以 **8** 位为例，**模** = **256**

5-5=5+(256-5)=256

自动丢掉最高位（溢出），超出范围舍去，剩下 **0**

```
  00000101   (+5的补码)
+ 11111011   (-5的补码)
-----------
 100000000
```

8位模空间运算时，超出范围的1被丢弃（溢出），结果为 00000000

已知： 清华1911年建校，今年2026年

要求： 不能直接做减法 2026-1911

约定： 年份有效表示0~9999，用补码规则，
减法转换为加法

如何用补码规则，只用加法运算，计算出今年清华建校多少周年？

$$N = m \times 10^{e}$$

类比十进制的科学计数法

二进制的**浮点数**编码方案

$$N = (-1)^{s} \times m \times 2^{e}$$

以最常用的 32 位 **单精度浮点数**（IEEE754）为例，这套人为定义的编码，把 32 位划分成三段含义：

符号位(s)	指数位(e)	尾数位(m)
1 位	8 位	23 位

如果让你设计一个能表示**宇宙年龄（138亿年）**和**普朗克时间10^{-44}**的**数**，你会**怎么做**？

从补码（定点整数）方案理解编码

补码是固定位宽模空间下的工程编码方案：在固定位宽的模运算体系中，将负数映射成一个等效正数。使计算机只用加法电路，就能完成带符号整数的加减运算。

- **硬件视角**：加法器只认高低电平（0/1），只会做加法
- **编码的本质**：人为定义的映射规则
- **编码的价值**：把符号信息嵌入比特序列，改造数字的语义
- **编码的选择**：编码方案不是唯一的，要适配硬件工程目标

编码（真值→补码）：

- 正数：补码与原码（真值）相同
- 负数：符号位固定为 1，数值位按位取反后加 1

解码（补码→真值）：

- 正数：直接读取数值位，即为真值
- 负数：对负数的补码整体按位取反后加 1，得到对应的原码，再根据符号位判断正负，读出真值

编码↔解码：信息处理的一对配套动作

编码 是把信息翻译成机器能处理的符号（信息在机器内部的表示）

解码 是把机器内部表示的符号还原成人能理解的信息

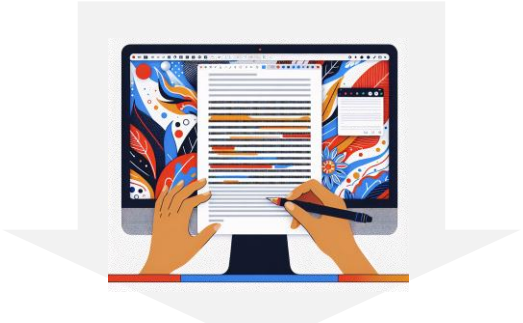
示例

- **加法器硬件实验**：
  十进制数字（人看懂）→编码成二进制 0/1 输入加法器
  加法器输出 0/1 →数码管解码，变回我们可读的十进制数字
- **补码运算**：
  带正负的整数真值 →编码转为补码送入加法器
  运算得到补码结果 →解码还原成带正负的真值
- **浮点数编码**：
  极大 / 极小实数 →编码为符号位 + 指数位 + 尾数位的比特序列
  读取比特序列 →解码还原为实数
- **英文字符编码**（ASCII）：
  英文字符 →编码成 0/1 表示
  读取 0/1 →解码，配合字体文件，渲染出屏幕上的文字
- **SVG 矢量图形**
  绘图指令（文字描述画圆）→编码为文本文件
  应用程序读取 0/1 →解码，配合渲染器绘出图形

编码规则由人定义。**编码和解码必须成对配套**，使用同一套规则，否则出现乱码、解析错误

$y = f(x)$ + 编码解码

x：输入信号（开关产生的高低电平，一串 0/1）
y：输出信号（电路输出引脚的高低电平，一串 0/1）
f：由逻辑门的导线连接**拓扑结构**决定的变换规则
只要电路接线不变，f 的转换规则就固定不变。



信息加工处理机器

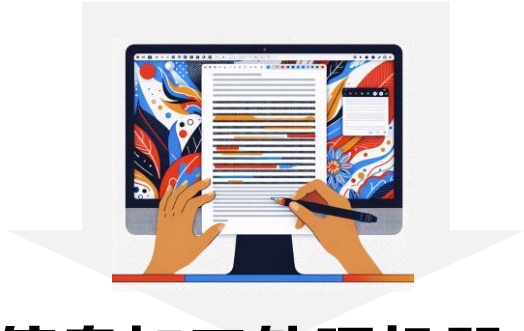
如果我们想处理不同类型的信息，有**两种路径**：

1. **改 f**（电路拓扑）：重新接线，改变硬件的符号变换规则
2. **改 编码规则**：硬件 f 不变，只是改变输入输出 0/1 的含义

总之，**硬件 f** 只负责完成 0/1 符号之间的变换，硬件本身不理解 0/1 的含义。**编码**把现实世界各类信息翻译为硬件 f 能够接收的 0/1 符号 x，运算得到输出 y 后，再通过**解码**将 0/1 还原成人能够理解的信息。

计算模型　　信息翻译层

$y = f(x)$ + 编码解码



信息加工处理机器

文字、图形这类非数值信息，同样可以使用这套模型吗

非数值数据编码：从数字到文本

编码的本质：建立唯一无歧义的映射关系

字符集：ASCII 一共收录 128 个符号

编码规则：人为约定，把表格排列成 16 行 ×8 列，用 7 比特来记录每个字符在表格里的行列位置。这个位置编号，直接作为字符对应的唯一的二进制数串

存储约定：7 比特的编码，在计算机里占用 1 字节存放，最高位补 0

编码（查表映射）

	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

ASCII编码1963年发布

编码的本质：建立唯一无歧义的映射关系

逻辑型数字化的本质是**符号映射**

映射步骤：字符集 → 编码空间 → 编码

❶确定**字符集**

GB2312国标码（1980年）

汉字：6763个（一级汉字3755个按拼音排序，二级汉字3008个按部首排序）。

非汉字：682个（拉丁字母、日文假名、俄文字母、标点符号等）。

❷在**编码空间**里给每个一个**唯一编号**

区位码（94区× 94位）

	帧	症	郑	证	芝	枝	支	吱	蜘	知	肢	脂	汁	之	织
职	直	植	殖	执	值	侄	址	指	止	趾	只	旨	纸	志	挚
掷	至	致	置	帜	峙	制	智	秩	稚	质	炙	痔	滞	治	窒
中	盅	忠	钟	衷	终	种	肿	重	仲	众	舟	周	州	洲	诌
粥	轴	肘	帚	咒	皱	宙	昼	骤	珠	株	蛛	朱	猪	诸	诛
逐	竹	烛	煮	拄	瞩	嘱	主	著	柱	助	蛀	贮	铸	筑	

中 —区位码→ **5448** (3630_H)

GB2312编码

00110110 00110000

❸按照规则把这个**位置编号**转换成二进制

编码的本质：建立唯一无歧义的映射关系

映射步骤：选定**字符集** →分配字符编号（**编码空间**） → 二进制**编码**

❶确定**字符集**

小字符集（ASCII）
大字符集扩展（GB18030）
跨语言统一集合（Unicode）

❷根据应用场景
配套**字符编码规则**

同一套 Unicode 字符集，可以选用 UTF-8、UTF-16 等不同编码。

Unicode 国际字符集把世界上所有语言的文字、符号全部收录进同一本总表，给每个字符分配一个全球唯一编号，彻底解决各国各搞一套编码表的问题

UTF(**U**nicode **T**ransformation **F**ormat)是 Unicode 的存储实现规则，后缀数字代表编码的基础单元大小。

例如：**UTF-8** 是变长编码，根据字符频率用1~4个字节表示一个Unicode字符（英文字符兼容 ASCII 占1字节，常用中文字符占 3 字节），兼顾兼容性与存储效率

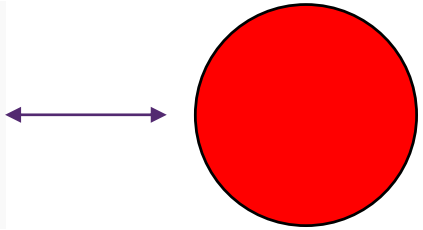
非数值数据编码：从文本到图形

编码对象：**图形**的基础形状种类是有限集合（圆、矩形…），但形状摆放位置、大小、颜色，可以有无穷多种组合

编码方案：**图元 + 参数**（绘图说明的结构化描述）
图元指定是什么基础形状，参数指定这个形状的位置、大小、颜色等属性
矢量图形拥有一套几何生成规则，但图元本身不能单独确定画面，必须配套具体参数，才能唯一确定图形长什么样

图形由坐标、几何方程描述，不是预先固定像素点阵，因此称为矢量图形

```
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">
  <!-- 画一个红色填充、黑色轮廓的圆 -->
  <circle cx="100" cy="100" r="80" fill="red" stroke="black" stroke-width="2"/>
</svg>
```



SVG矢量图形编码示例

矢量逻辑编码（业务语义）+ UTF8编码（文件存储的字符编码）

图形如何编码

SVG文件：绘图描述文本（图元 + 参数）→UTF-8文本编码→二进制0/1

```
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">
  <!-- 画一个红色填充、黑色轮廓的圆 -->
  <circle cx="100" cy="100" r="80" fill="red" stroke="black" stroke-width="2"/>
</svg>
```

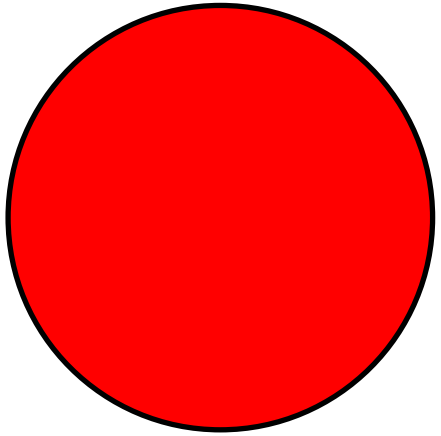
记事本打开的SVG文件

- **指令**: `circle` (画圆)　　生成画面（圆）的结构化描述
- **参数**:
  - `cx="100"`: 圆心 X 坐标在 100
  - `cy="100"`: 圆心 Y 坐标在 100
  - `r="80"`: 半径为 80

  图元基本参数

  - `fill="red"`: 内部填充红色
  - `stroke="black"`: 轮廓为黑色
  - `stroke-width="2"`: 轮廓宽度为 2

  样式属性参数



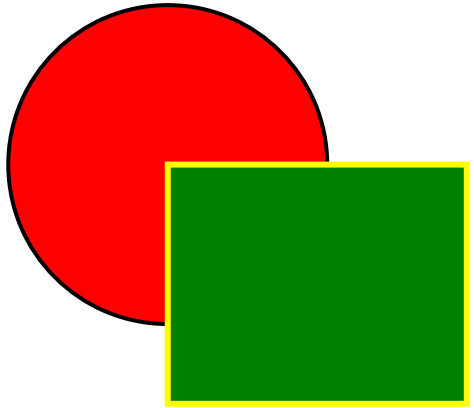
PowerPoint幻灯片中插入该SVG文件
内置的渲染器根据绘制说明画出图形

思考：SVG 的 circle 图元描述，是编码还是编程

矢量图形编辑练习

矢量图形：编码包含几何生成规则和参数，渲染时实时计算坐标

矢量图形编辑：不修改像素，只增加几何运算规则、变换参数，修改绘图描述文本



任务：修改SVG绘图描述，对圆和矩形执行布尔差集运算（圆减去矩形），再将运算得到的新图形整体逆时针旋转 45°，得到右侧缺口扇形图形

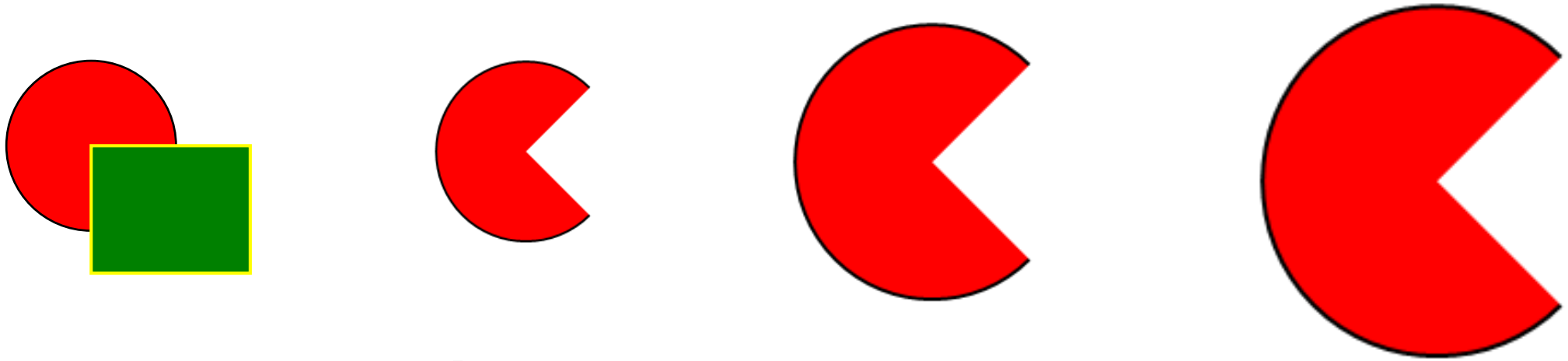
```
<svg width="300" height="300" xmlns="http://www.w3.org/2000/svg">
  <!-- 画一个红色填充、黑色轮廓的圆 -->
  <circle cx="100" cy="100" r="80" fill="red" stroke="black" stroke-width="2"/>
  <!-- 再画一个绿色填充、黄色轮廓的矩形 -->
  <rect x="100" y="100" width="150" height="120" fill="green" stroke="#FFFF00" stroke-width="3"/>
</svg>
```

矢量图形编辑练习

```
<svg width="300" height="300" xmlns="http://www.w3.org/2000/svg">
  <!-- defs：新增 <defs> 定义遮罩规则，实现圆形减去矩形 -->
  <defs>
    <!-- 定义遮罩mask -->
    <mask id="cutMask">
      <!-- 白色矩形：全部保留区域 -->
      <rect x="0" y="0" width="300" height="300" fill="white"/>
      <!-- 黑色矩形：这个区域透明，用来挖掉圆形的一部分 -->
      <rect x="100" y="100" width="150" height="120" fill="black"/>
    </mask>
  </defs>
  <!-- 红色圆形，应用遮罩：黑色矩形区域被挖去，实现圆减去矩形 -->
  <circle cx="100" cy="100" r="80" fill="red" stroke="black" stroke-width="2"  mask="url(#cutMask)"/>
</svg>
```

矢量图形编辑练习

```
<svg width="300" height="300" xmlns="http://www.w3.org/2000/svg">
  <!-- defs：新增 <defs> 定义遮罩规则，实现圆形减去矩形 -->
  <defs>
    <!-- 定义遮罩mask -->
    <mask id="cutMask">
      <!-- 白色矩形：全部保留区域 -->
      <rect x="0" y="0" width="300" height="300" fill="white"/>
      <!-- 黑色矩形：这个区域透明，用来挖掉圆形的一部分 -->
      <rect x="100" y="100" width="150" height="120" fill="black"/>
    </mask>
  </defs>
  <!-- g分组：把图形打包成一个整体，添加变换规则：整体逆时针旋转45°，旋转中心(100,100) -->
  <g transform="rotate(-45,100,100)">
    <circle cx="100" cy="100" r="80" fill="red" stroke="black" stroke-width="2" mask="url(#cutMask)"/>
  </g>
</svg>
```



矢量图形的编辑：

- 全程修改文本描述（指令 + 参数）

矢量图形的显示：

- 渲染器读取全部几何规则，实时计算图形坐标并重新绘制
- 放大/缩小只是在不同比例上重新绘制，图形本身不会失真