

程序的机器级表示（1）

柴云鹏

2022.10

Outline

- CPU发展
- 自下而上：构建计算机体系
- CPU Architecture
- Memory and Registers

ENIAC

- 第一台电子计算机，宾夕法尼亚大学，1946年
 - 建造的目的是取代人工，计算导弹弹道
 - ENIAC用了18000电子管、1500继电器、重30吨、占地170m²、耗电140kw、每秒计算5000次加法
 - 相当于手工计算的20万倍、机电式计算机的1000倍
 - 纸带输入程序和数据，人工时间占比很高

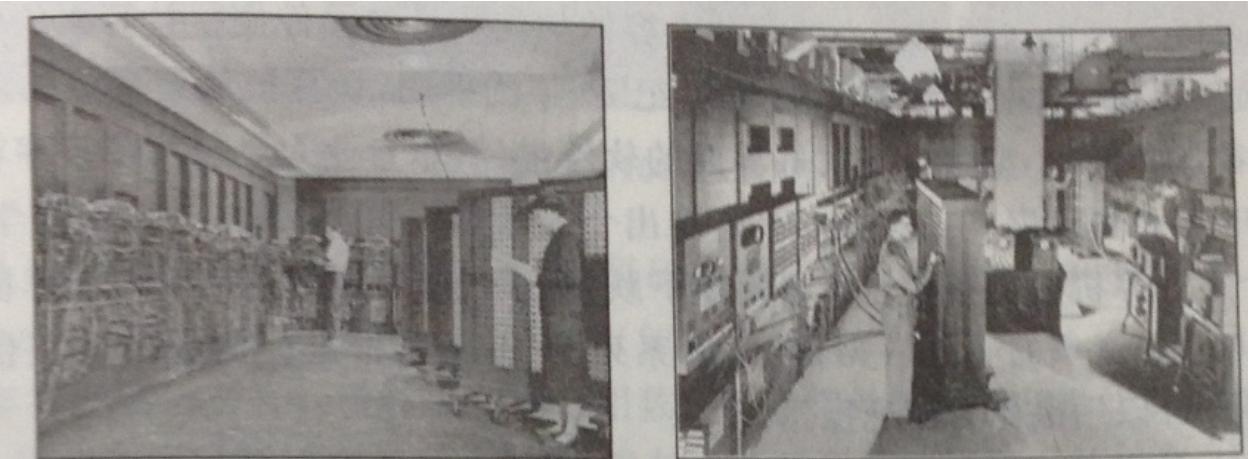


图 2.8 世界上第一台数字式电子计算机 ENIAC

企业级计算机：IBM和DEC

IBM S/360, 1960s

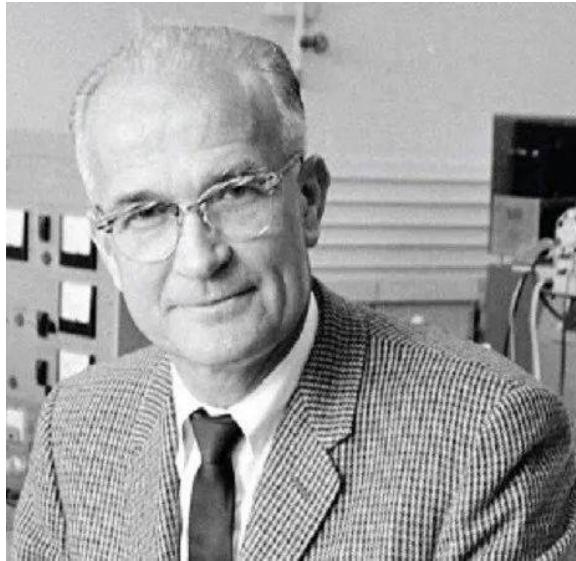


DEC PDP-11



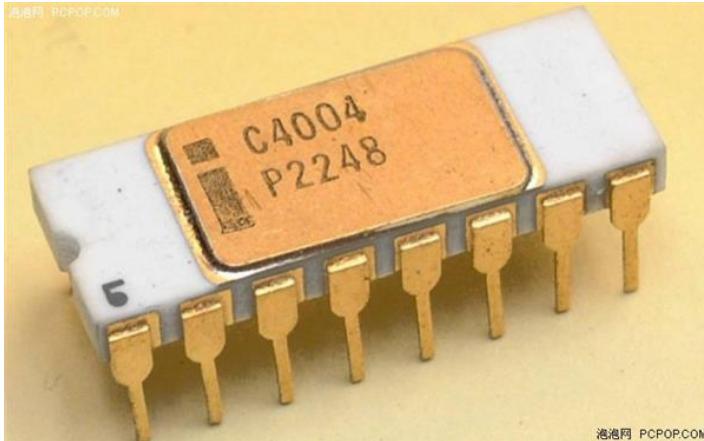
仙童

- 1957年，“晶体管之父”肖克利建立公司开始晶体管研发
- 旗下“八叛徒”建立仙童公司，后建立Intel, AMD等公司



Intel x86 CPU

- Intel 成立于 1968 年，创始人包括来自仙童的葛罗夫 (A. Grove) 和戈登·摩尔 (Gordon Moore)；AMD 成立于 1969 年，创始人是仙童的销售杰里·桑德斯 (Jerry Sanders)
- 1971 年第一款 4 位微处理器 Intel 4004 问世，1974 年第一款 8 位微处理器 Intel 8080 问世



Intel 4004 是第一个商用微处理器

采用 4004 的 **Busicom Unicom⁶** 计算器

X86 family

8088CPU

- 8086(1978, 29K)
 - The heart of the **IBM PC & DOS (8088)**
 - 16-bit, 1M bytes addressable, 640K for users
 - x87 for floating pointing
- 80286(1982, 134K)
 - 80186 编号给了不太重要的一个芯片
 - 为了与摩托罗拉68000系列CPU竞争
 - Basis of the IBM PC-AT & Windows
- i386(1985, 275K)
 - 32 bits architecture, flat addressing model
 - Support a Unix operating system

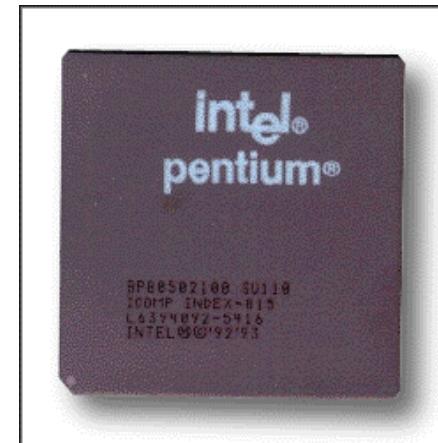


IBM PC机



X86 family

- I486(1989, 1.9M)
 - Integrated the **floating-point** unit onto the processor chip
- Pentium(1993, 3.1M)
 - Improved performance, added minor extensions
- PentiumPro(1995, 5.5M)
 - P6 microarchitecture
 - Conditional mov
- Pentium II(1997, 7M)
 - Continuation of the P6



X86 family



- Pentium III(1999, 8.2M)
 - New class of instructions for **manipulating vectors of floating-point numbers**(SSE, Stream SIMD Extension)
 - Later to 24M due to the incorporation of the **level-2 cache**
- Pentium 4(2001, 42M)
 - Netburst microarchitecture with high clock rate but high power consumption
 - SSE2 instructions, new data types (eq. Double precision)

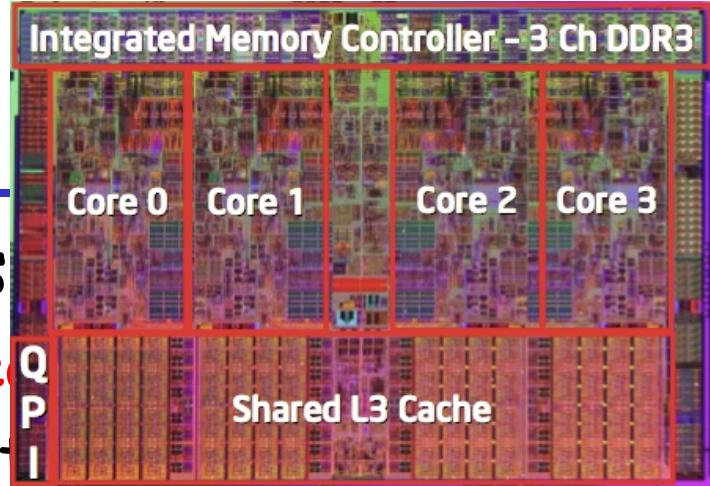


X86 family

- Pentium 4E: (2004, 125M transistors).
 - Added *hyperthreading*
 - run two programs simultaneously on a single processor
 - EM64T, 64-bit extension to IA32
 - First developed by Advanced Micro Devices (AMD)
 - x86-64
- Core 2: (2006, 291M transistors)
 - back to a microarchitecture similar to P6
 - multi-core (multiple processors on a single chip)
 - Did not support hyperthreading

X86 family

- Core i7: (2008, 781 M trans)
 - Incorporated **both** hyperthreading
 - the initial version supporting multiple threads of execution on each core
- Core i7: (2012, 2.27B transistors, Sandy Bridge-EP)
 - 8 cores on each chip
 - 2.7G (3.3G)
- Core i7: (2015, 5.5B transistors, Haswell-EP)
 - 18-core Xeon



X86 family

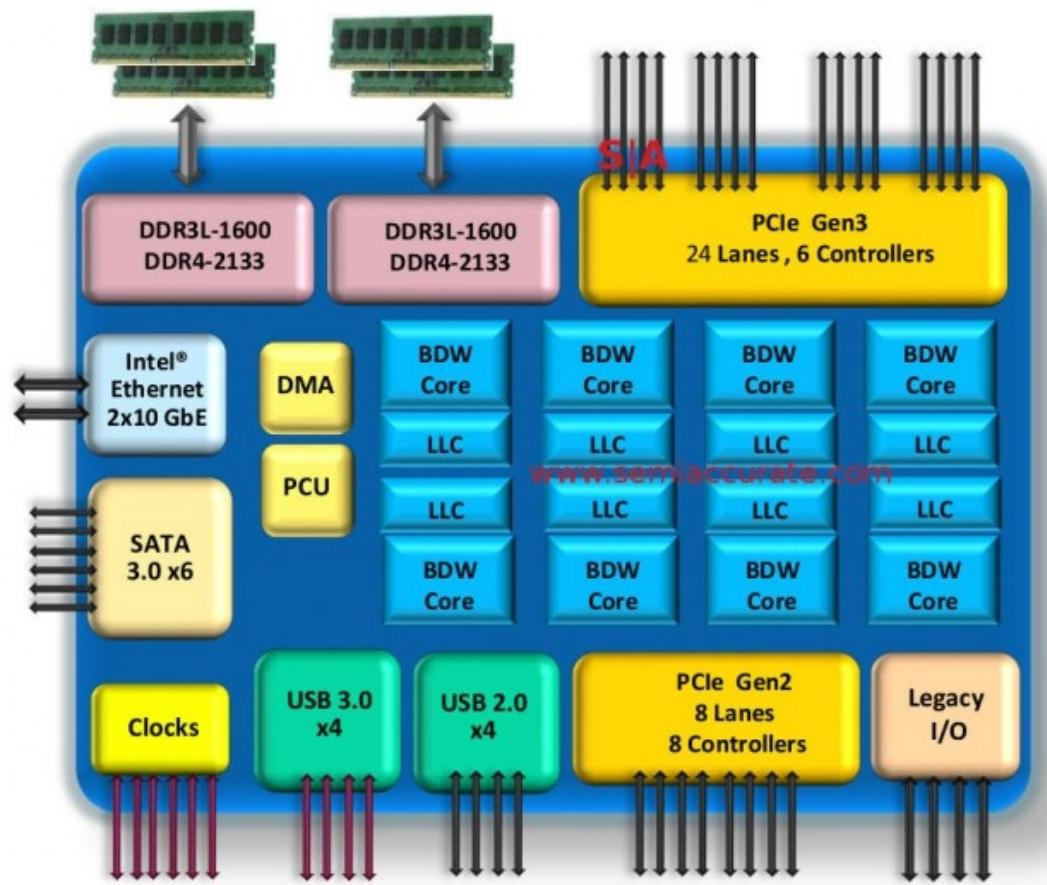
- Core i7 Broadwell 2015

- Desktop Model

- 4 cores
- Integrated graphics
- 3.3-3.8 GHz
- 65W

- Server Model

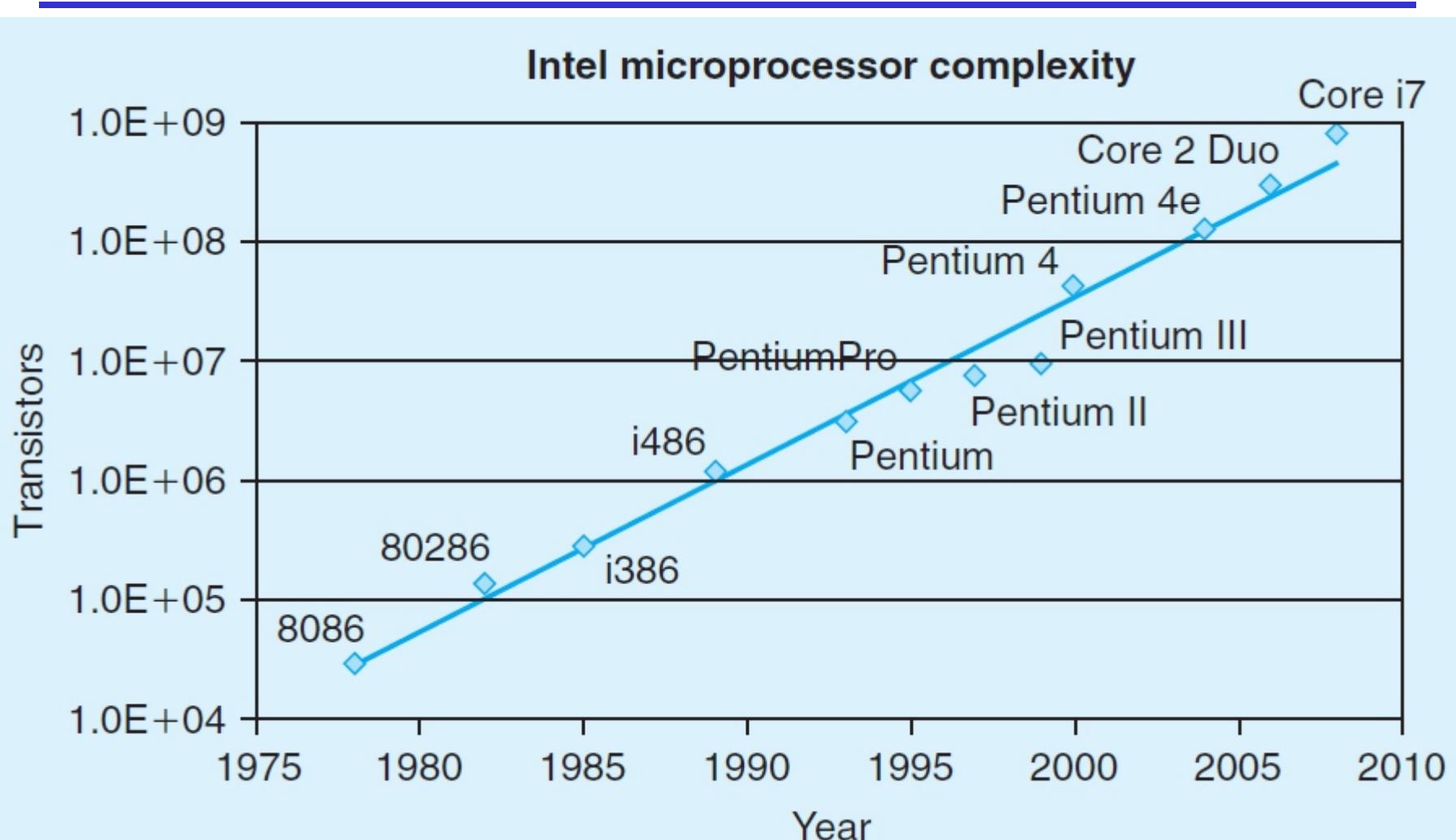
- 8 cores
- Integrated I/O
- 2-2.6 GHz
- 45W



X86 family

- Advanced Micro Devices (AMD)
 - At beginning,
 - lagged just behind Intel in technology,
 - produced less expensive and lower performance processors
- In 1999
 - First broke the 1-gigahertz clock-speed barrier
- In 2002
 - Introduced x86-64
 - The widely adopted 64-bit extension to IA32

Moor's Law



苹果M1 Pro和M1 Max

- M1 Pro
 - 内存带宽200GB/s
 - 337 亿晶体管
- M1 Max
 - 内存带宽400GB/s
 - 570 亿晶体管



Our Coverage

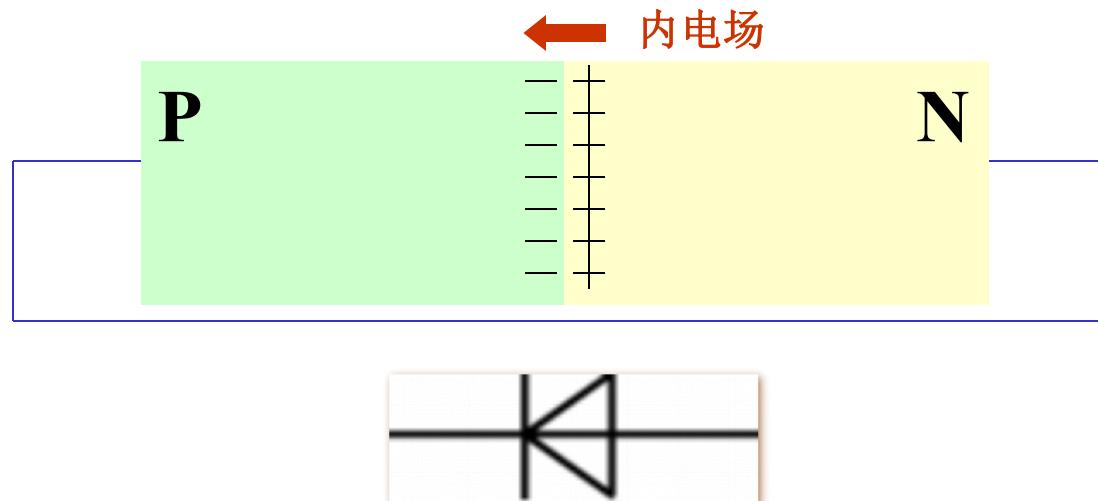
- IA32
 - The traditional x86
 - shark> gcc -m32 hello.c
- x86-64
 - The emerging standard
 - shark> gcc hello.c
 - shark> gcc -m64 hello.c

自下而上：构建计算机体系

- 1) 物理: PN结->门电路
- 2) 数字电路: 门电路->组合逻辑电路/时序逻辑电路 (CPU、寄存器、控制器...)
- 3) 抽象: 冯诺伊曼体系结构
- 4) 硬件接口: 机器语言/指令集
- 5) 软硬件分界面: 汇编语言 (汇编器)
- 6) 编译器: 高级语言->汇编语言
- 7) 链接器: 目标文件 (机器代码序列) ->可执行文件
- 8) 代码执行: 程序->进程 (加载器)
- 9) 进程空间: 虚拟内存管理

自下而上：构建计算机体系

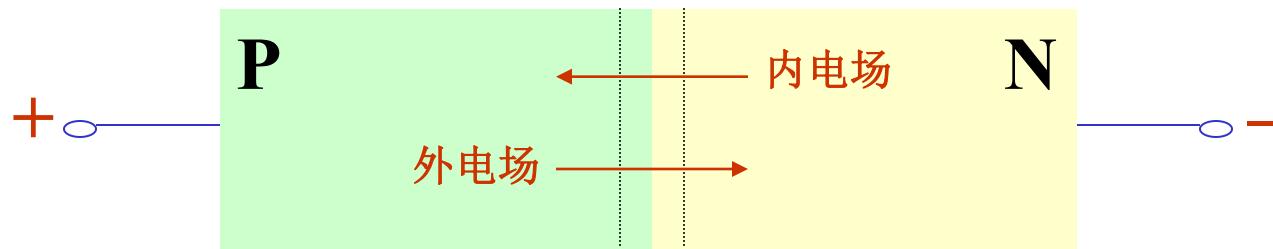
- 1) 物理: PN结->二极管->门电路
 - PN结: P型半导体和N型半导体构成
 - 单向导电性
 - 加正向电压时PN结导通
 - 加反向电压时PN结截止



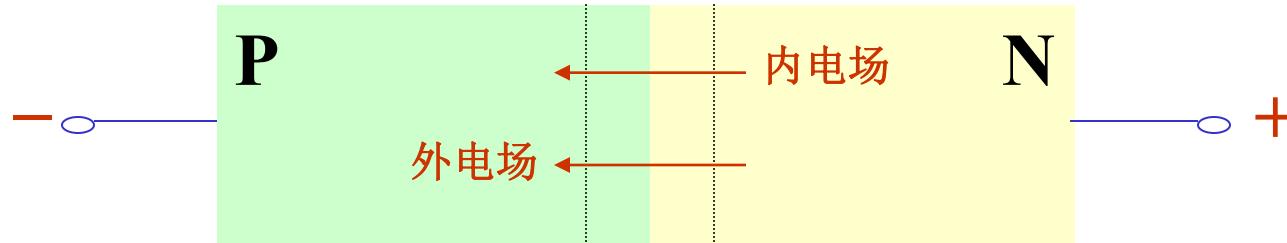
自下而上：构建计算机体系

在PN结两端适当加以外部电压，就可以打破上述平衡，实现单向导电的目的。

加正向电压时PN结导通：加正向电压时，内外电场方向相反，阻挡层变窄，相当于PN结的电阻很小。因此加正向电压时，PN结导通。



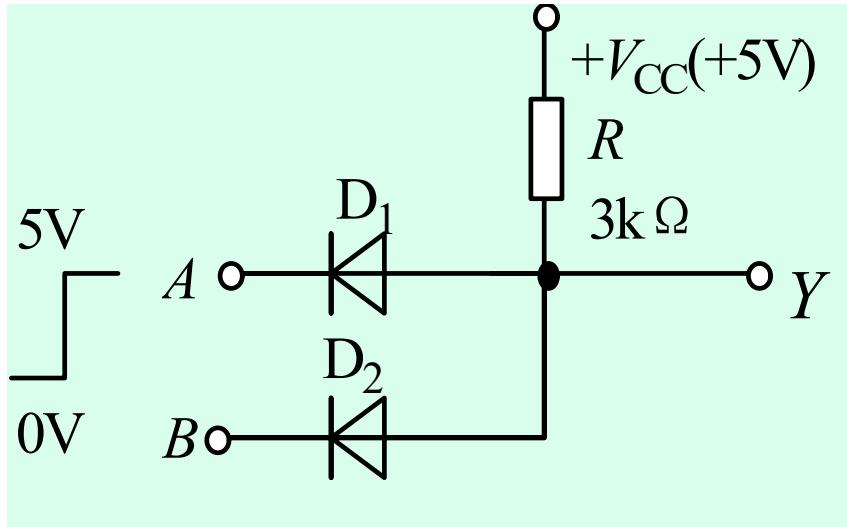
加反向电压时PN结截止：加反向电压时，内外电场方向相同，阻挡层变宽，相当于PN结的电阻很大。因此加反向电压时，PN结截止。



自下而上：构建计算机体系

- 1) 物理: PN结->二极管 (一种晶体管) ->门电路
 - 与门 (逻辑运算)

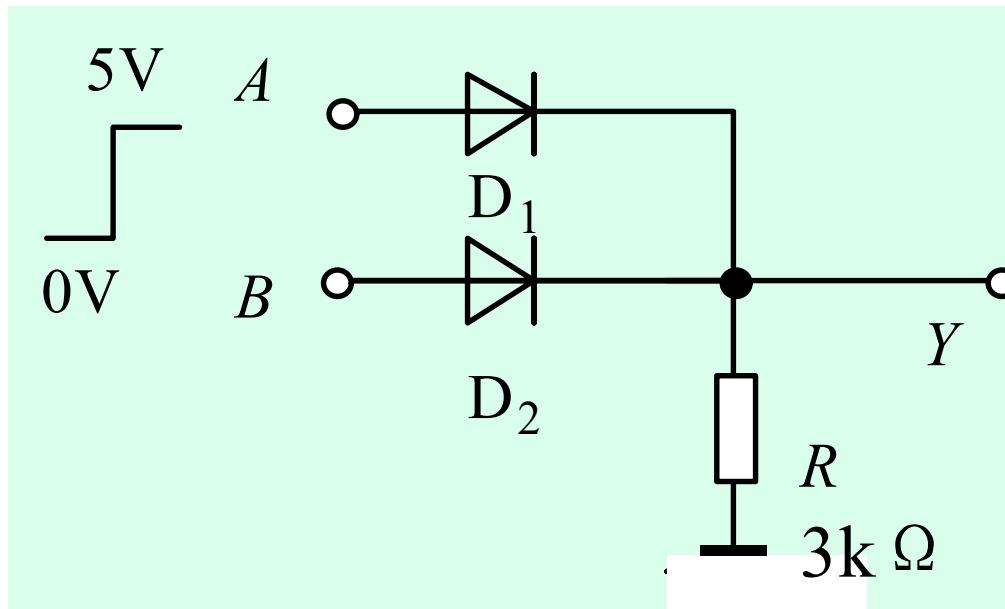
$$Y=AB$$



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

自下而上：构建计算机体系

- 1) 物理: PN结->二极管->门电路
 - 与门 (逻辑运算)



$$Y = A + B$$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

自下而上：构建计算机体系

- 2) 数字电路：门电路->组合逻辑电路/时序逻辑电路（CPU、寄存器、控制器...）
 - 组合逻辑电路：
 - 加法器、ALU、译码器
 - 时序逻辑电路
 - 寄存器、计数器

Xn	Yn	Hn	Cn
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

半加器：

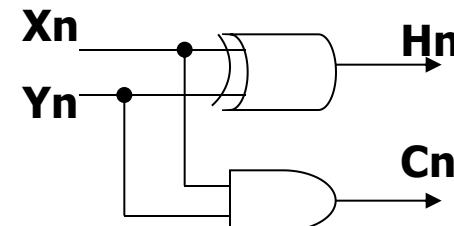
不考虑进位输入时，两数码 X_n , Y_n 相加称为半加

H_n : 本位二进制值

C_n : 想更高位的进位

$$H_n = X_n \cdot Y_n + X_n \cdot Y_n = X_n \oplus Y_n$$

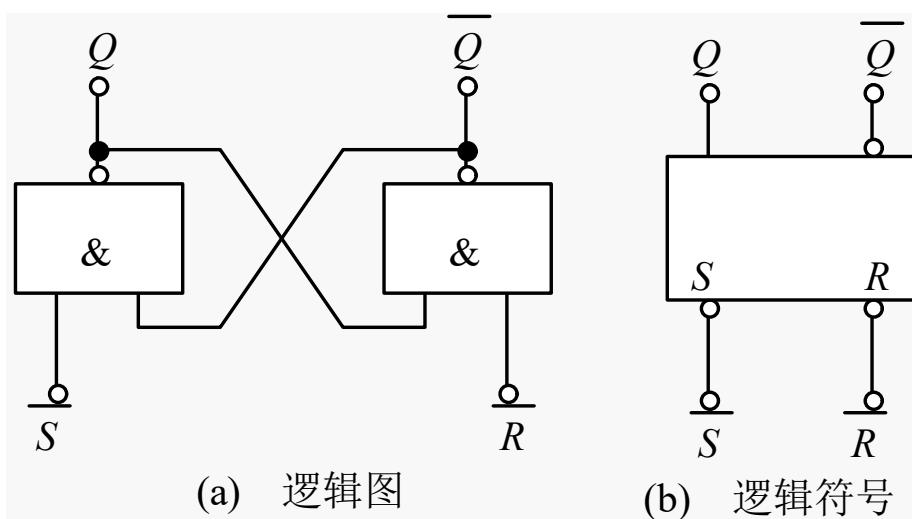
$$C_n = X_n \cdot Y_n$$



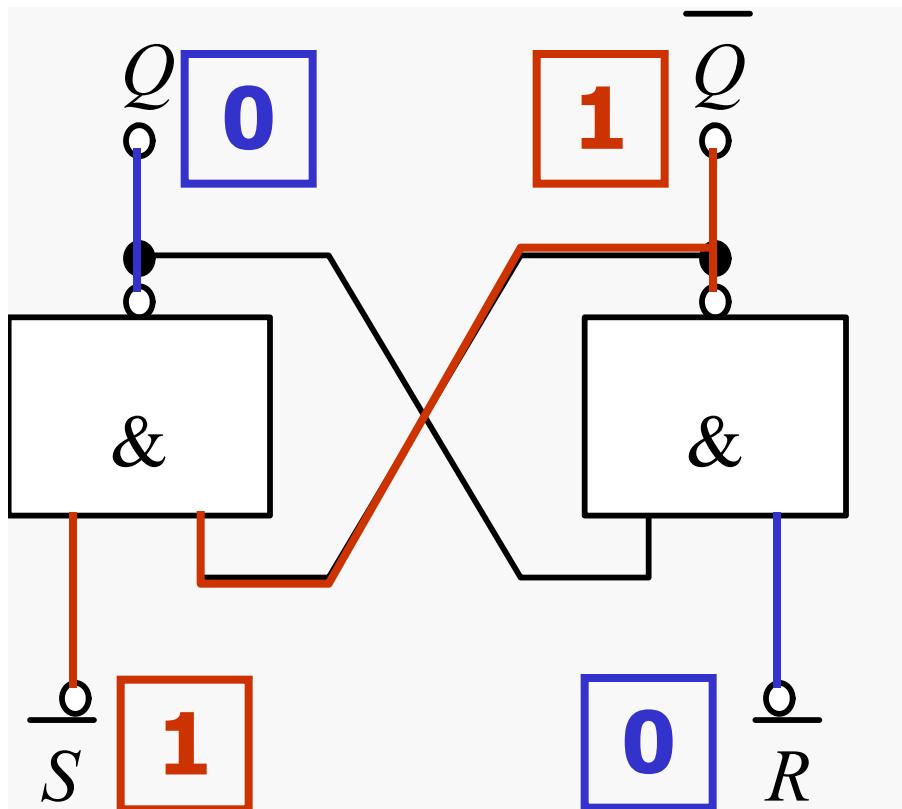
自下而上：构建计算机体系

- 2) 数字电路：门电路->组合逻辑电路/时序逻辑电路（CPU、寄存器、控制器...）
 - 时序逻辑电路举例：用与非门构造RS触发器
 - 具有记忆功能
 - 不断点的前提下

信号输出端， $Q=0$ 、 $\bar{Q}=1$ 的状态称**0状态**， $Q=1$ 、 $\bar{Q}=0$ 的状态称**1状态**

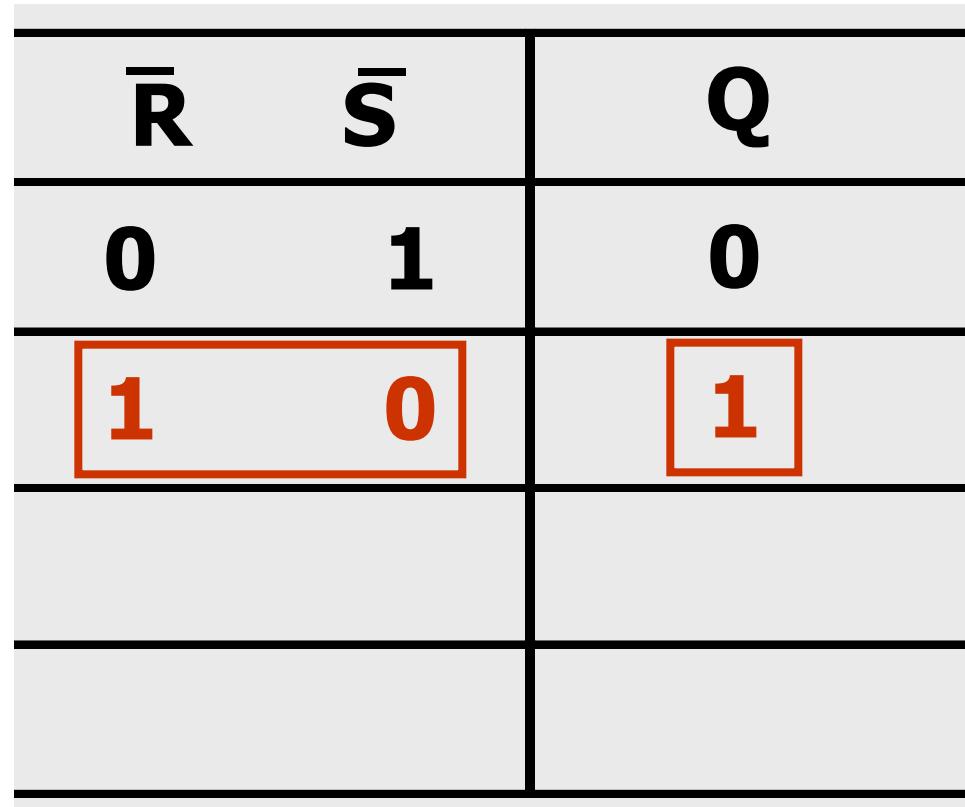
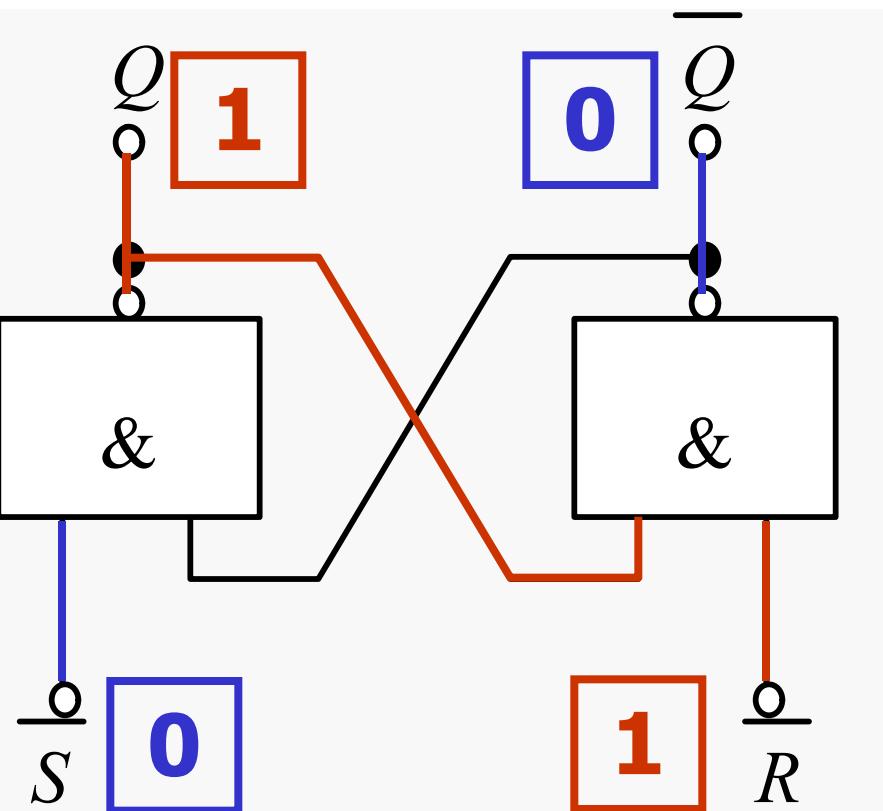


自下而上：构建计算机体系

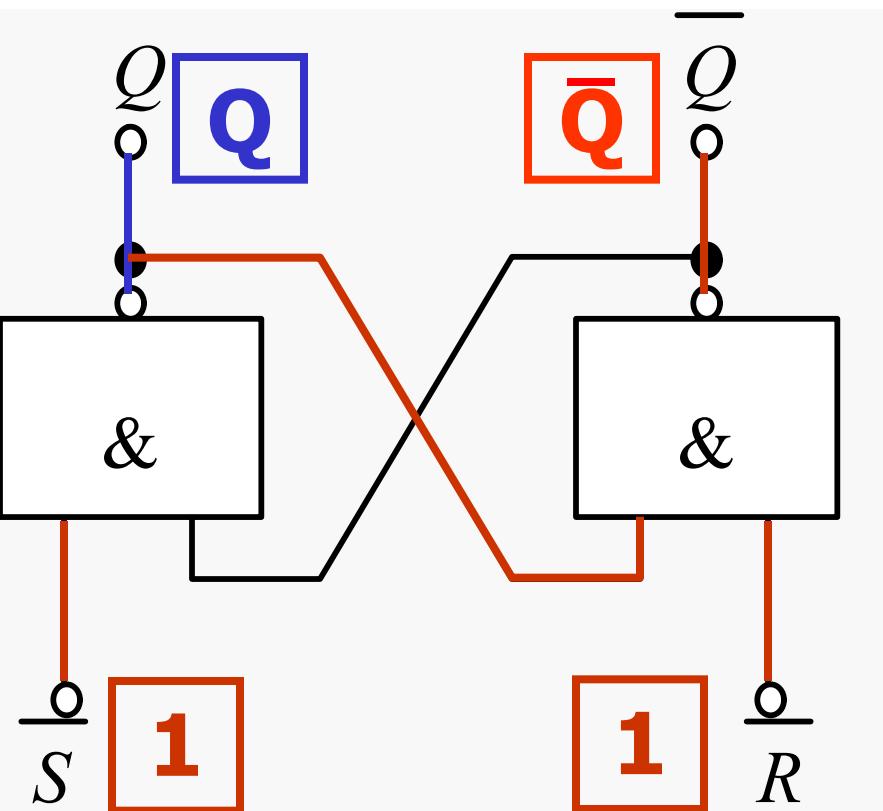


\bar{R}	\bar{S}	Q
0	1	0

自下而上：构建计算机体系

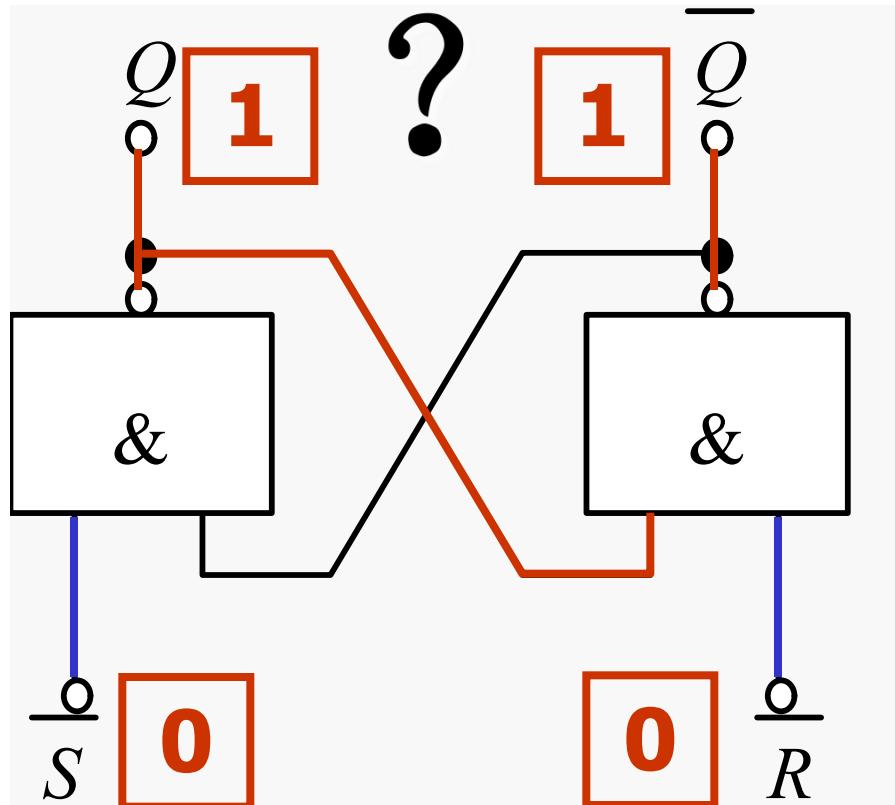


自下而上：构建计算机体系



\bar{R}	\bar{S}	Q
0	1	0
1	0	1
1	1	不变

自下而上：构建计算机体系



\bar{R}	\bar{S}	Q
0	1	0
1	0	1
1	1	不变
0	0	不定

自下而上：构建计算机体系

现态：触发器接收输入信号之前的状态，也就是触发器原来的稳定状态。

\bar{R}	\bar{S}	Q^n	Q^{n+1}	功能
0	0	0	不用	不允许
0	0	1	不用	
0	1	0	0	$Q^{n+1} = 0$
0	1	1	0	置 0
1	0	0	1	$Q^{n+1} = 1$
1	0	1	1	置 1
1	1	0	0	$Q^{n+1} = Q^n$
1	1	1	1	保持

次态：触发器接收输入信号之后所处的新的稳定状态。

自下而上：构建计算机体系

- 1949年在英国剑桥大学，计算机先驱 Maurice Wilkes与EDSAC计算机内存的合影
- 图中内存叫做汞延迟线内存
 - 需要依赖比光速慢很多的音速实现存储功能
 - 在玻璃管的一端是扬声器，另一端则是麦克风。输入的数据脉冲会被转换成声音，然后借助汞传递到玻璃管的另一头，并被重新转换为电信号脉冲，进行整理，然后重新转换并通过环路发送出去。



自下而上：构建计算机体系

• 外存的存储原理

- 磁盘

- 磁化方向->感应电流

- 闪存 (Flash)

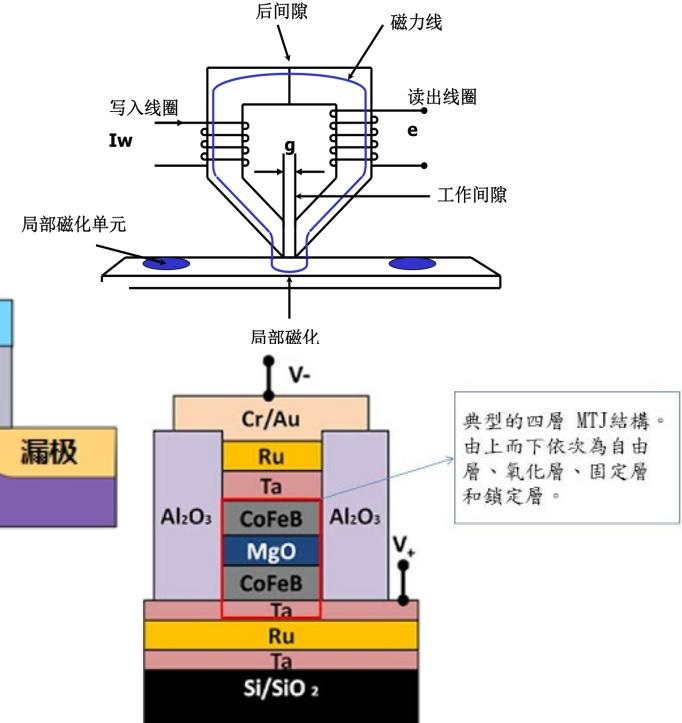
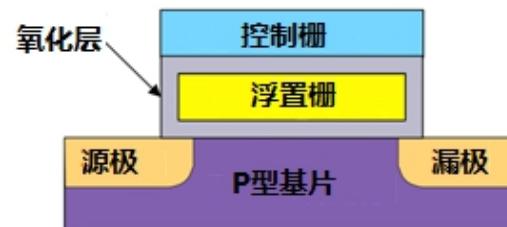
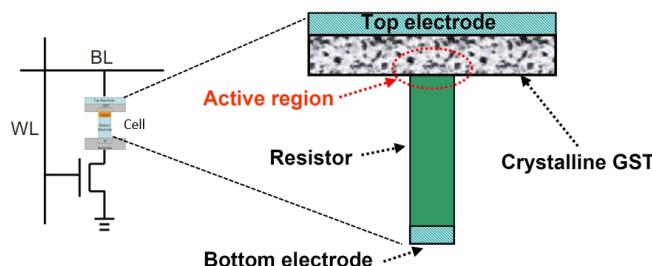
- 是否捕获/捕获多少电子->电阻/电流

- 相变存储器 (PCM)

- 加热温度->结晶状态->电阻

- MTT-MRAM

- 电子自旋

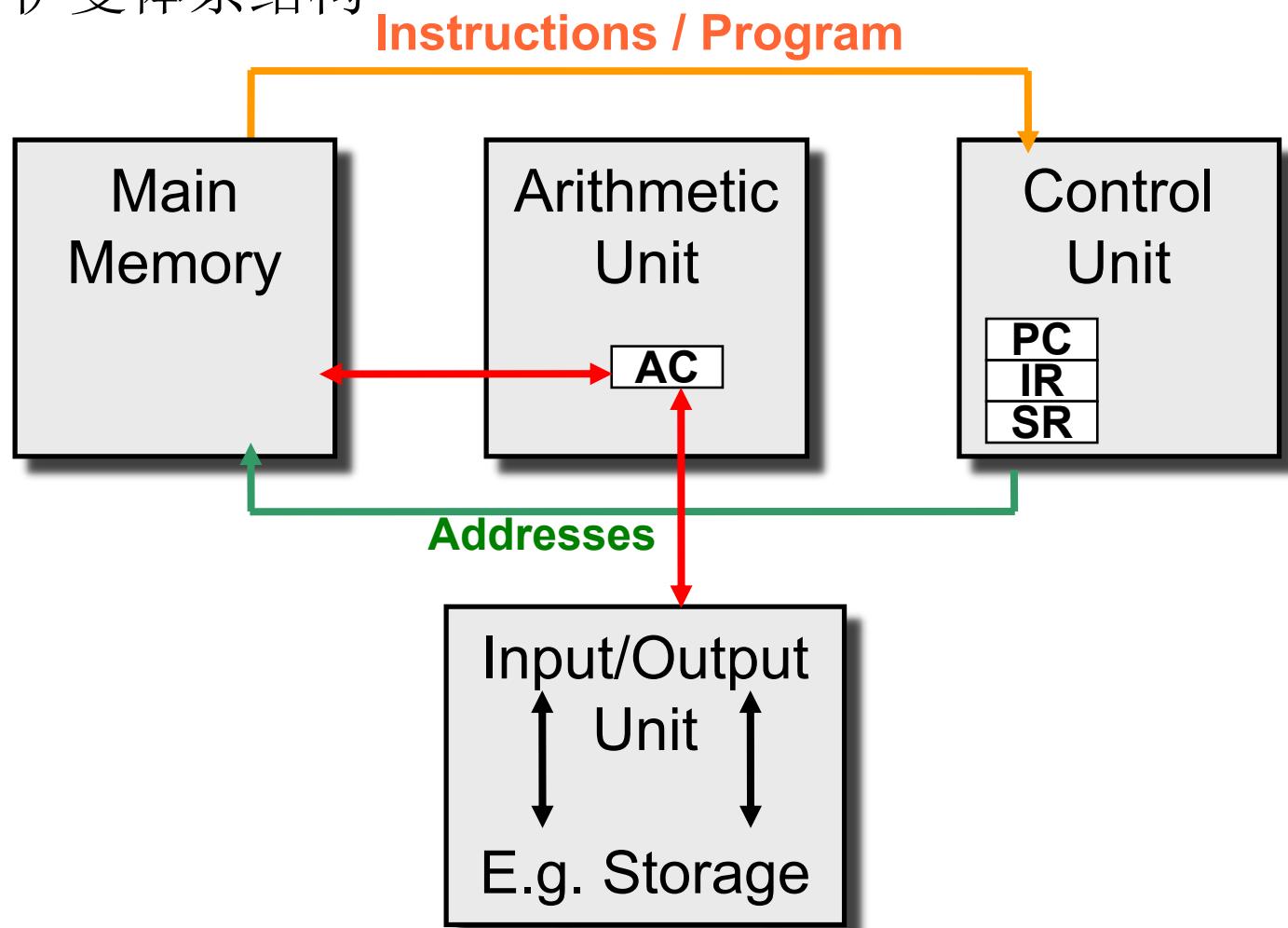


世界上第1块硬盘诞生于1956年：IBM公司的305 RAMAC



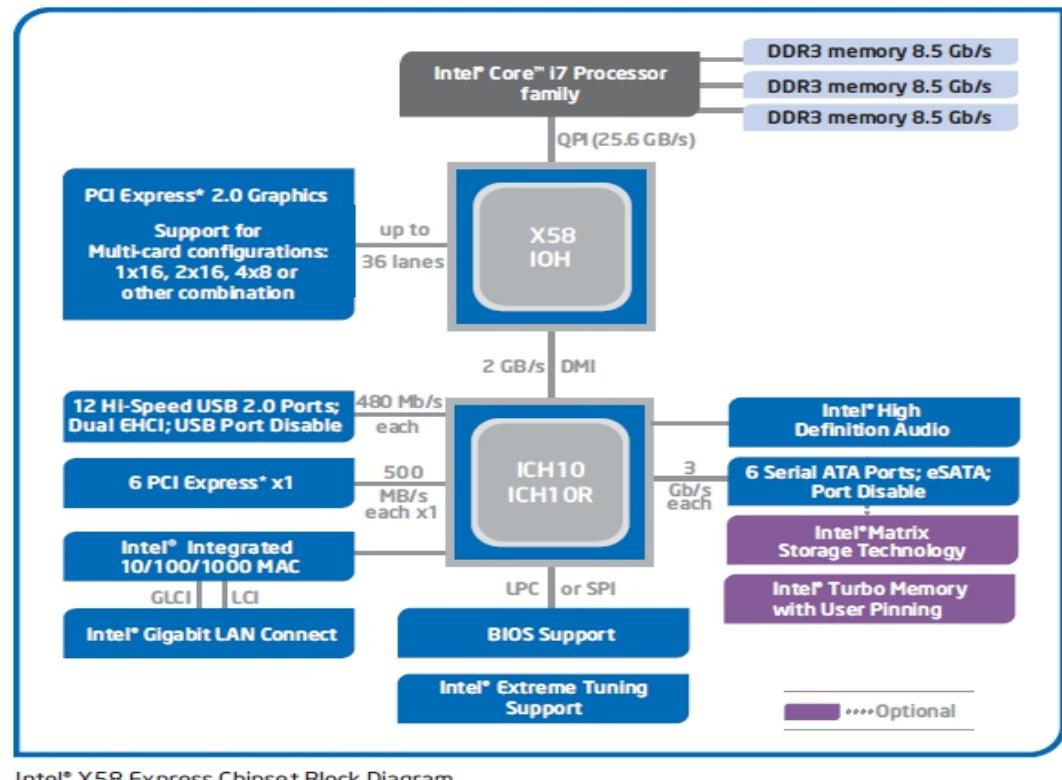
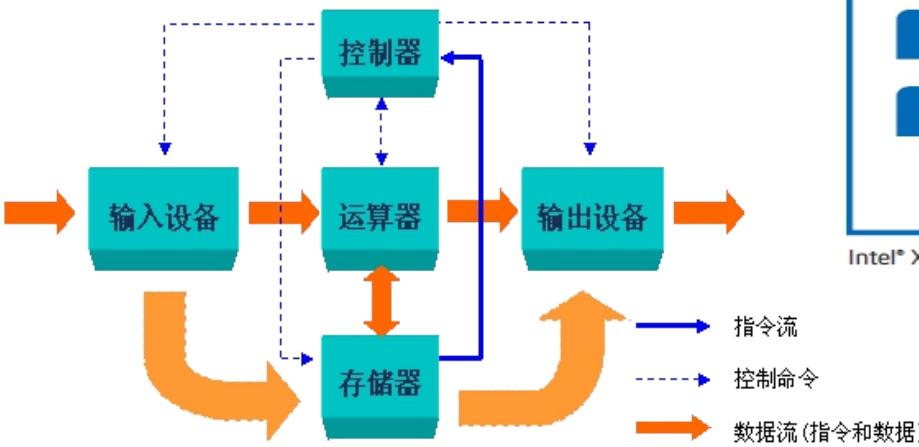
自下而上：构建计算机体系

- 3) 抽象：冯诺伊曼体系结构



自下而上：构建计算机体系

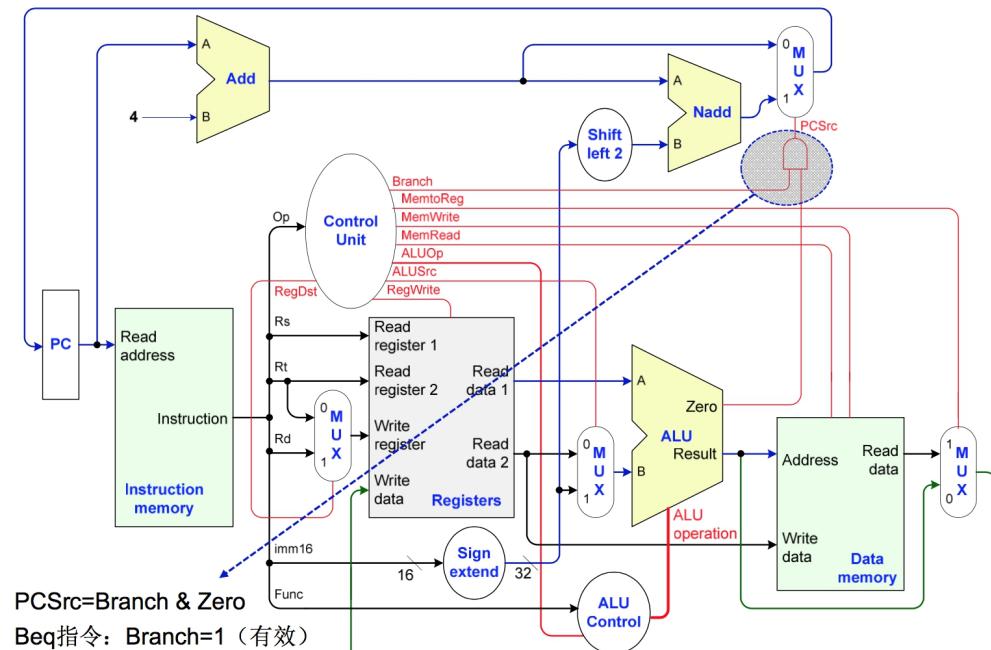
- 运算器、控制器（寄存器、Cache）->CPU
- 存储器->内存
- 输入输出设备->磁盘、显示器、打印机、键盘鼠标、...



自下而上：构建计算机体系

• CPU设计

- 出口：指令集（机器语言），如加法、乘法、跳转、过程调用等
- 运算器（ALU）
 - 组合逻辑电路，进行各种计算
- 控制器
 - 输入：当前指令、指令阶段
 - 输出：所有器件的控制信号
- 寄存器
 - 少量临时存储器件，寄存器快于 Cache 快于内存
 - 时序逻辑电路，具有记忆功能



自下而上：构建计算机体系

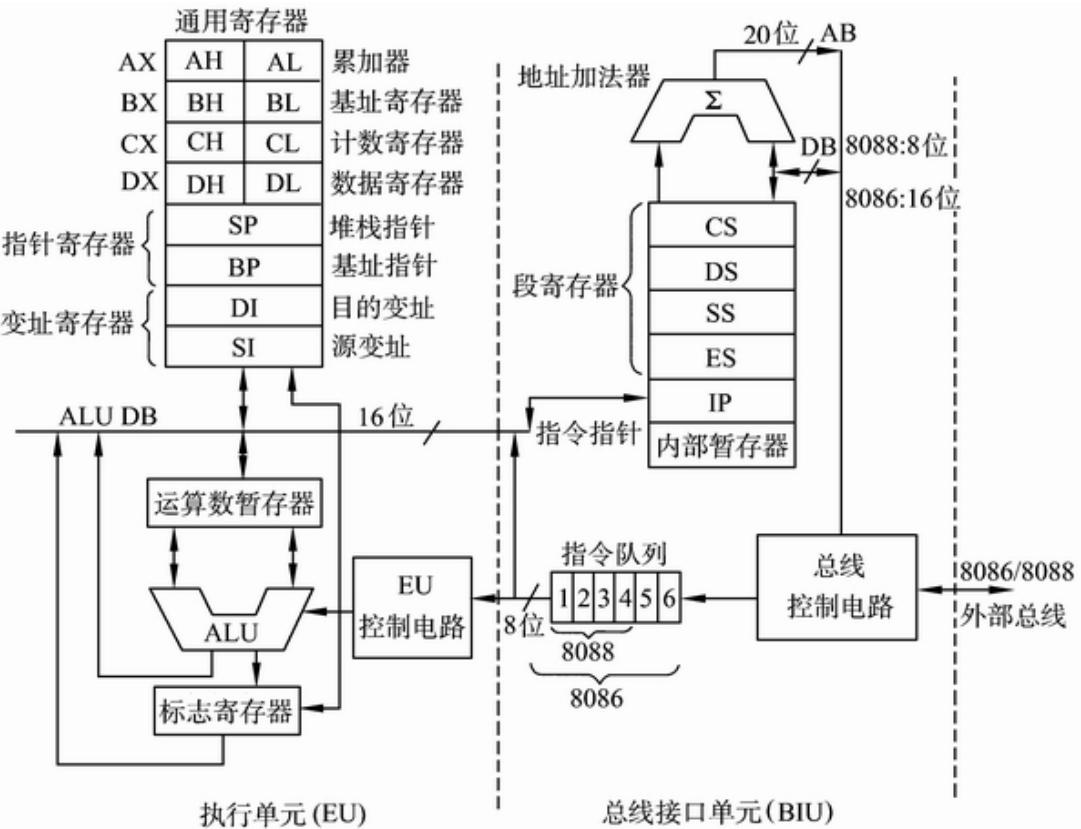
- CPU工作流程

```
While (1) {  
    Fetch_an_instruction();  
    Understand_the_instruction()  
    Execute_the_instruction();  
}
```

- Instructions:

- 移动数据(among register, memory, and I/O)
 - 算数、逻辑、位运算
 - 跳转类 (指令跳转, 函数调用等)

Intel 8086处理器的结构



自下而上：构建计算机体系

- 4) 硬件接口：机器语言/指令集
- 5) 软硬件分界面：汇编语言（汇编器）

```
long plus(long x, long y);

void sumstore(long x, long y,
              long *dest)
{
    long t = plus(x, y);
    *dest = t;
}
```

objdump -d sum

```
0000000000400595 <sumstore>:
400595: 53                      push   %rbx
400596: 48 89 d3                mov    %rdx,%rbx
400599: e8 f2 ff ff ff    callq  400590
<plus>
40059e: 48 89 03                mov    %rax,(%rbx)
4005a1: 5b                      pop    %rbx
4005a2: c3                      retq
```

Register

- 80386的寄存器

通用寄存器		
	16	15 0
EAX	AH	AL
EBX	BH	BL
ECX	CH	CL
EDX	DH	DL
ESP	SP	
EBP	BP	
ESI	SI	
EDI	DI	

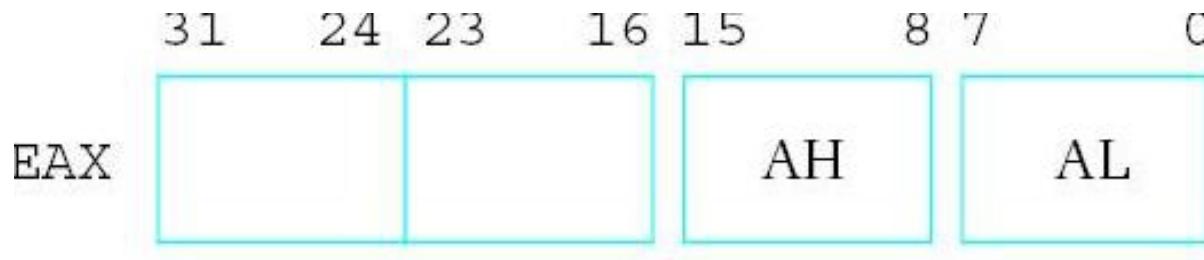
指令指针和标志寄存器		
	16	15 0
EIP		IP
EFLAGS		FLAGS
控制寄存器		
CR0	AH	AL
CR1	BH	BL
CR2	CH	CL
CR3	DH	DL

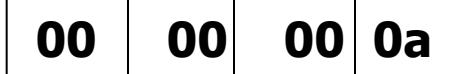
段寄存器	
	0 15
CS	16位选择符
SS	16位选择符
DS	16位选择符
ES	16位选择符
FS	16位选择符
GS	16位选择符

段描述符寄存器	
	0 31
	32位段基址、20位段限、11位其它属性

Register

- EAX, EBX, ECX, EDX



`mov eax, 10` 

00	00	00	0a
----	----	----	----

`mov ax, 10` 

xx	xx	00	0a
----	----	----	----

`mov ah, 10` 

xx	xx	0a	xx
----	----	----	----

`mov al, 10` 

xx	xx	xx	0a
----	----	----	----

Register

- **EAX, EBX, ECX, EDX**
 - 通用寄存器
- **ESP, EBP, ESI, EDI**
 - 32位通用寄存器
 - 各自的低**16**位可以独立使用: **SP, BP, SI, DI**
 - **ESP**: 系统栈的栈指针寄存器
 - **EBP**: 基址指针寄存器
 - **ESI, EDI**: 索引寄存器, 从**ESI**指定位置拷贝到**EDI**指定位置

Register

- 标志寄存器EFLAGS

31...18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留	VM	RF		NT	IOPL	OF	DF	IF	TF	SF	ZF		AF		PF		CF	

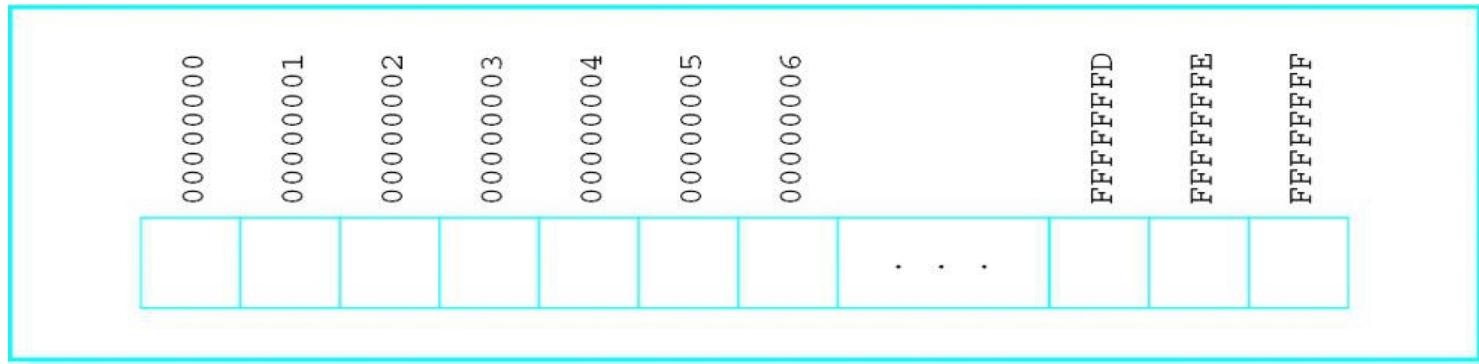
Bit	Mnemonic	Usage
0	CF	carry flag
2	PF	parity flag
6	ZF	zero flag
7	SF	sign flag
10	DF	direction flag
11	OF	overflow flag

Register

- 64位计算机
 - X86-64指令集扩展了IA32的32位寄存器
 - 用'r'代替'e', %eax->%rax, %esp->%rsp
 - 增加了8个寄存器
 - %r8~%r15
 - 浮点数XMM寄存器
 - %xmm0~%xmm15, 128位, 能放下4个float, 或2个double

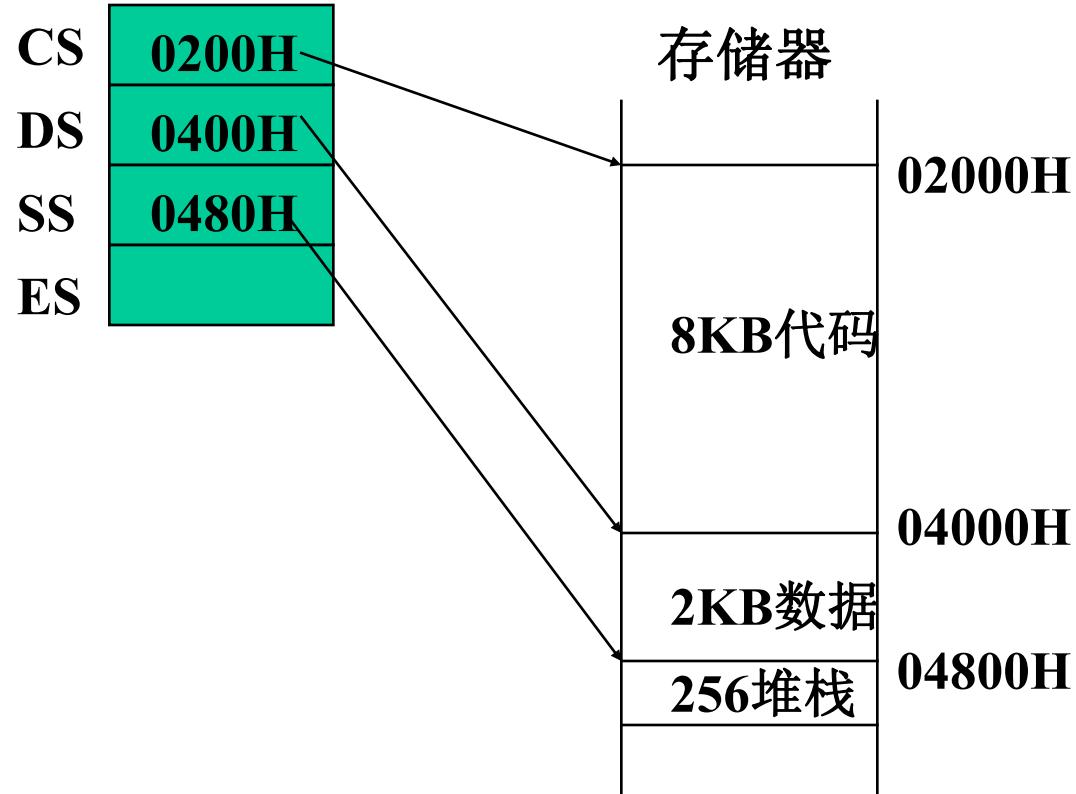
Memory

- 8086~80286
 - 实模式
 - 直接访问物理地址，处理器只能寻址 1 MB 的内存空间
 - 处理器一次只能运行一个程序，随时可被中断
- 80386
 - 实模式： 80386 提供实模式来兼容先前的 8086
 - 保护模式： 程序访问虚拟地址， OS 内核解析为物理地址



Memory

- 段寄存器（为什么内存要分段？）
- 代码段**CS**
 - 对应执行的指令，起始地址对应第一条可执行的指令
- 数据段**DS**
 - 对应程序的数据、常量等
- 堆栈段**SS**
 - 对应暂存任何数据，包括子程序所用的数据
- 扩展段**ES**
 - 对应于特殊的数据和操作使用



Memory

- 内存分段管理
 - 由于**8086**中的地址寄存器都是**16**位的，用户不能直接使用**20**位的物理地址，编程时需要使用**逻辑地址**来寻址存储单元。
 - 逻辑地址由两个**16**位数构成，其形式为：
 - 段的起始地址 : 段内的偏移地址
 - (**16**位段地址) : (**16**位偏移量)

Memory

- 存储空间分为多个逻辑段（段——Segment）：
 - 段的20位的起始地址（xxxxxH）其低4位必须为0（xxxx0H），所以可以将它们省略，然后用1个16位数来表示段的首地址。
 - 每段长度限 $2^{16}=64KB$ ，所以段内偏移地址可以用1个16位数来表示（xxxxH）；
 - 所以有：
 - 段的起始地址 : 段内的偏移地址
 - (16位段地址) : (16位偏移量)

Memory

- 物理地址和逻辑地址
 - 每个存储单元都有一个唯一物理地址 (00000H~FFFFFH)，20位，该地址在指令执行时由地址加法器形成，并进行硬件寻址。
 - 地址加法器的具体做法：段地址左移4位，然后加上偏移地址就得到20位物理地址。
 - 用户编程时采用逻辑地址，其形式为：
段的首地址 : 段内偏移地址
它们由两个16位的无符号数构成。
 - 逻辑地址 “1460H:100H” = 物理地址14700H
 $1460H + 100H = 14700H$

Memory

逻辑地址 **vs.** 物理地址

16 位 段 地 址	0000
------------	------

+

16 位 偏 移 地 址

20 位 物 理 地 址

Memory

- 段寄存器和逻辑段
 - 典型的程序有三个段：代码段、数据段和堆栈段
 - 8086有4个16位的段寄存器：
 - CS** (代码段寄存器) 用来指明代码段的首地址
 - SS** (堆栈段寄存器) 指明堆栈段的首地址
 - DS** (数据段寄存器) 指明数据段的首地址
 - ES, FS, GS** (附加段寄存器) 指明附加段的首地址

Memory

- $CS: IP = CS * 16 + IP$, 当前指令地址
- $DS: [x] = DS * 16 + x$, 当前数据地址
- $SS: SP = SS * 16 + SP$, 当前栈顶地址
 - 需自己保证push和pop不越界
- 80x86不允许直接将立即数送入段寄存器
 - `Mov ds, 1000H`非法
 - `Mov ax, 1000H` `mov ds, ax`合法

程序员可见状态

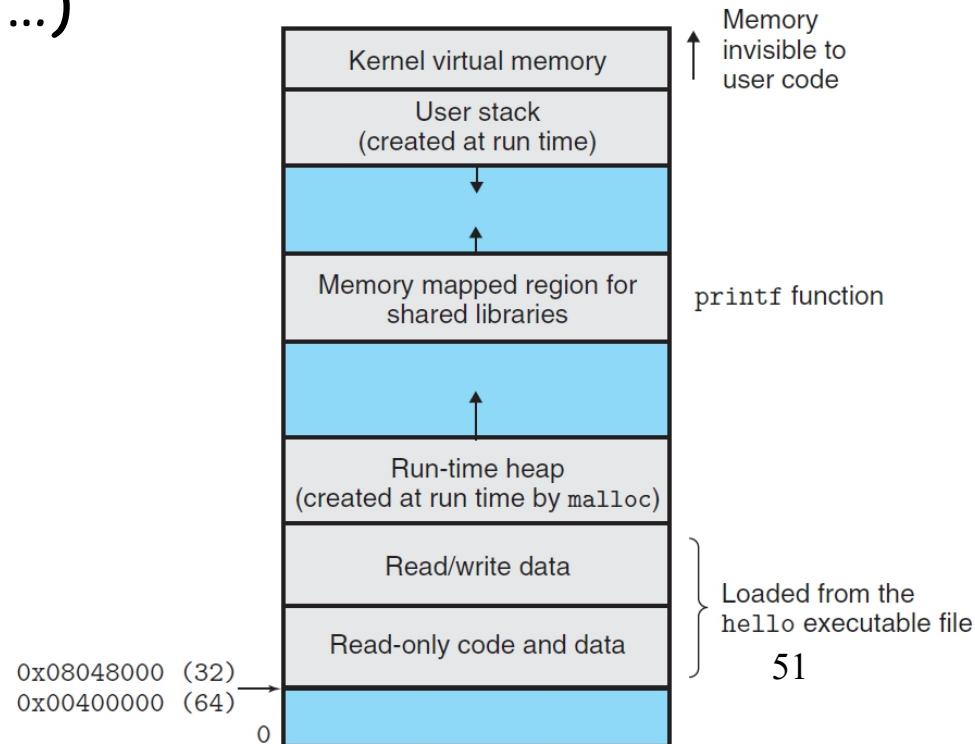
- C语言程序员
 - C语句
 - 各种计算
 - 程序流程（循环、分支、函数调用等）
 - 常量、变量（内存） //寄存器不可见
 - 全局变量/静态变量：数据段
 - 局部变量：堆栈段（注意作用域）
 - 操作系统提供的系统调用
 - `malloc`（堆内存）， `free`
 - `Printf`等， 控制外设（输入输出设备）
 - `Read/write`， 访问文件系统（I/O设备）
 - `Send/receive`， 访问网卡

程序员可见状态

- 汇编程序员
 - 汇编指令
 - 各类运算
 - 跳转指令 (`jmp`, `jcc`->分支、循环；子过程调用`call`)
 - 中断指令 (`int`, 与外设打交道/进行系统调用)
 - 寄存器
 - `Eax`, `ebx`, ... 等通用和专用寄存器
 - 内存
 - 全局变量：放在数据段
 - 堆栈段：可以直接操作 (`push`, `pop`, `move`, ...)
 - 函数调用时自动操作堆栈（保存返回地址等）

程序员可见状态

- 寄存器
 - `%rax, %rbx, %rcx, %rdx, %rbp, %rsp, %rsi, %rdi`
 - `%r8, %r9, %r10, %r11, %r12, %r13, %r14, %r15`
 - `EFlags (SF, ZF, OF, CF, ...)`
 - `CS, DS, SS, ES, FS, GS`
 - `PC (IP)`
 - `Memory`
 - 8086: 物理内存
 - 80386+: 虚拟内存地址



程序员可见状态

x86-64 Integer Registers

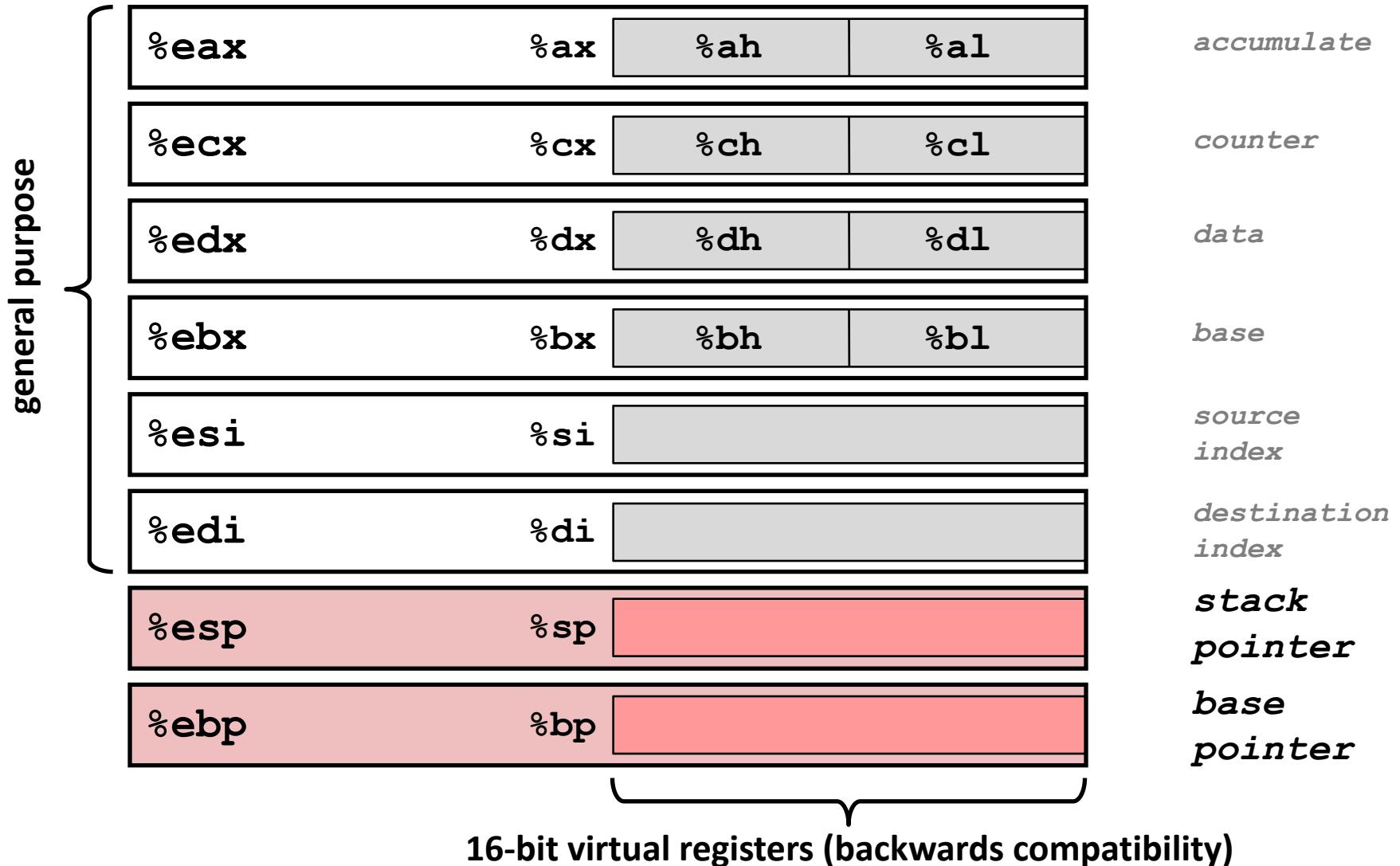
%rax	%eax
%rbx	%ebx
%rcx	%ecx
%rdx	%edx
%rsi	%esi
%rdi	%edi
%rsp	%esp
%rbp	%ebp

%r8	%r8d
%r9	%r9d
%r10	%r10d
%r11	%r11d
%r12	%r12d
%r13	%r13d
%r14	%r14d
%r15	%r15d

- Can reference low-order 4 bytes (also low-order 1 & 2 bytes)

Some History: IA32 Registers

Origin
(mostly obsolete)



课堂练习

- 8086计算机上，如果希望能够寻址到物理地址 $0x54000$ ，请问段地址的最大、最小分别是多少？

课堂练习

- 内存中存放的机器码和对应的汇编指令如下图所示，设CPU初始状态：CS=2000H，IP=0000H，请写出指令执行序列。

地址	内存	对应汇编指令	地址	内存	对应汇编指令
10000H	B8 23 01	mov ax, 0123H	20000H	B8 22 66	mov ax, 6622H
10003H	B8 00 00	mov ax, 0000	20003H	EA 03 00	jmp 1000:3
10006H	8B D8	mov bx, ax		00 10	
10008H	FF	jmp bx	20008H	89 C1	mov cx, ax
10009H	E3				

练习答案

- Mov ax, 6622H
- Jmp 1000:3
- Mov ax, 0000
- Mov bx, ax
- Jmp bx
- Mov ax, 0123H
- →第3步