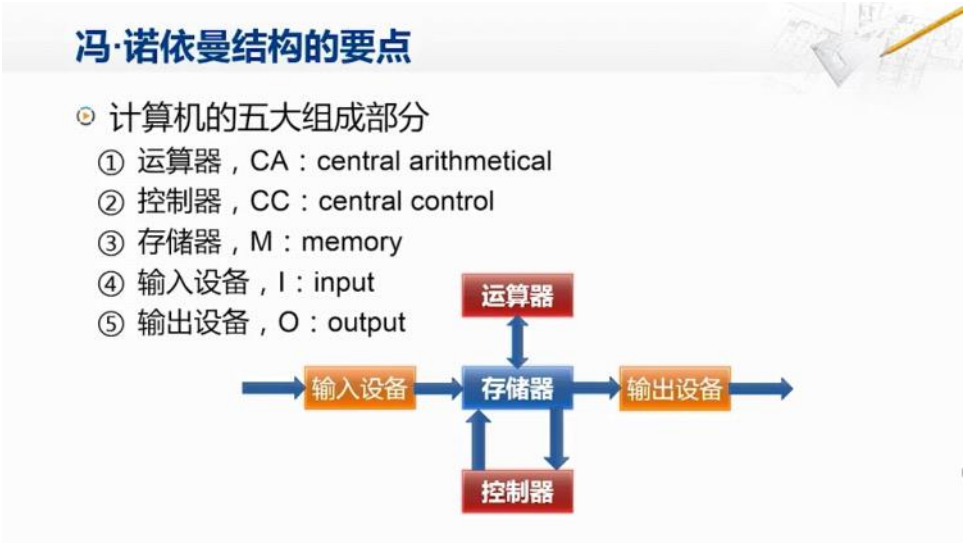
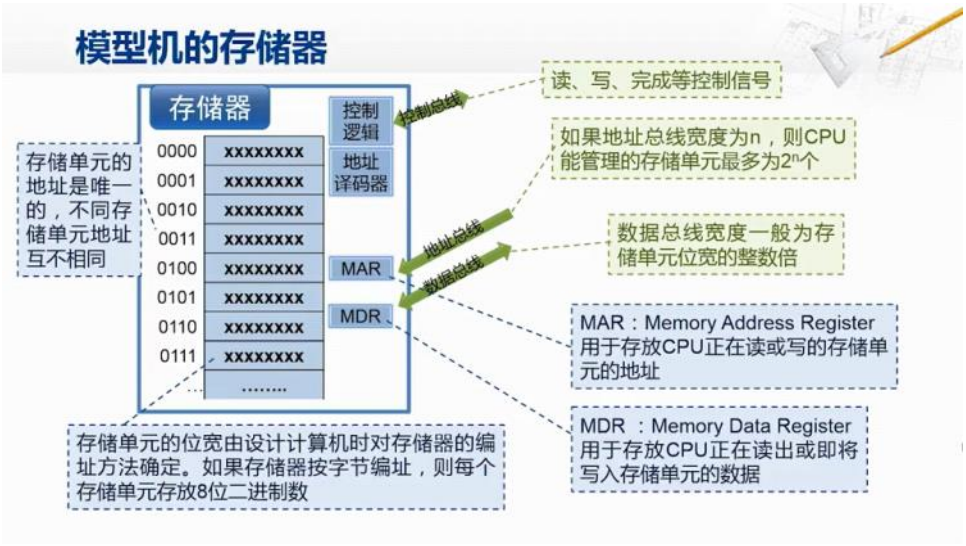
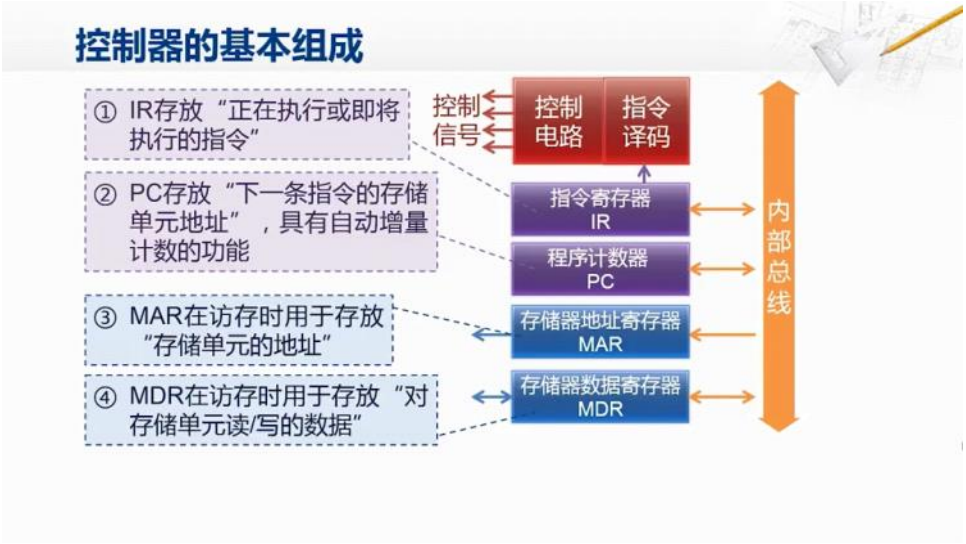
计算机组成chapter1&2

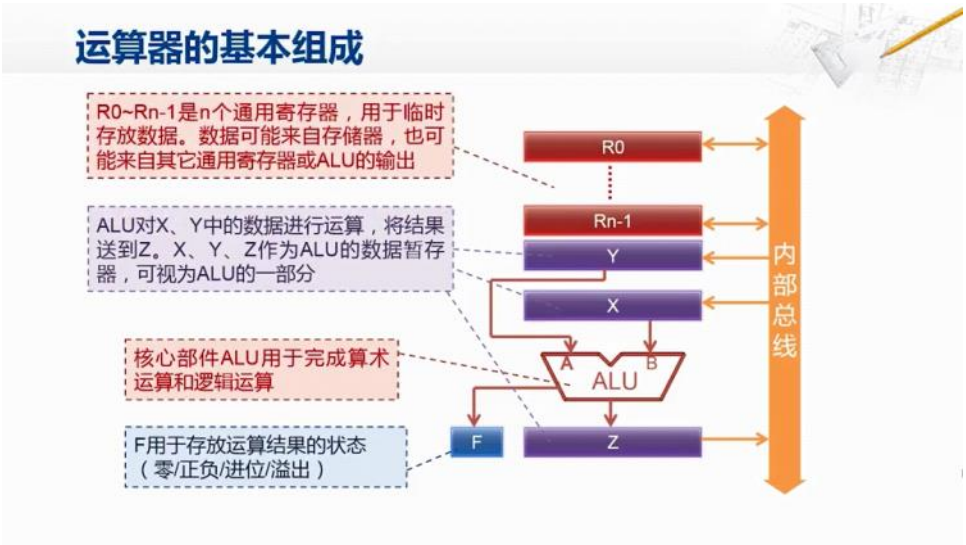
Tuesday, February 23, 2016 5:28 PM



数据和程序均以二进制的形式不加区别地存放在存储器中





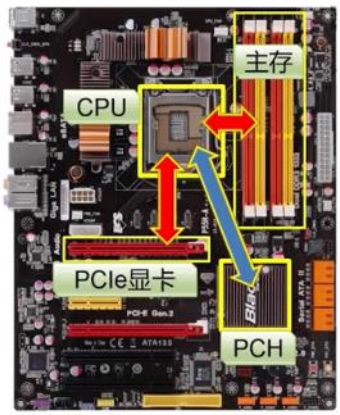


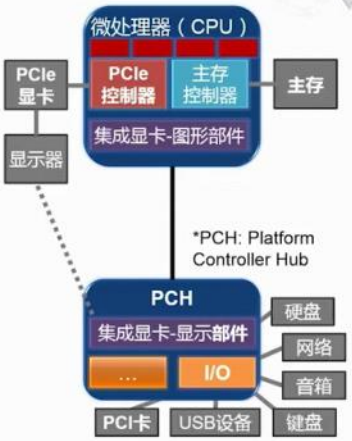
南北桥架构的演变：

北桥逐渐消失，CPU直连主存和显卡。

南桥改名PCH

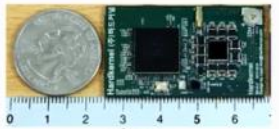
南北桥架构的演变（3）





系统芯片（Sytem-on-a-Chip, SoC）

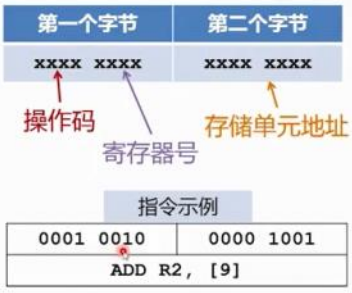
- 将计算机或其他电子系统集成为单一芯片的集成电路
- 在智能手机、平板电脑等移动计算设备上得到广泛应用



计算机指令结构

指令的格式

- 每条指令等长，均为2个字节
- 第一个字节的高4位是操作码
  - LOAD：0000；ADD：0001
  - STORE：0010；JMP：0011
  - 目前只提供4条指令，最多可扩展到16条
- 第一个字节的低4位是寄存器号
  - R0~R3：0000~0011
  - 目前只提供4个寄存器，最多可扩展到16个
- 第二个字节是存储单元地址
  - 最大可以使用256个字节的存储器



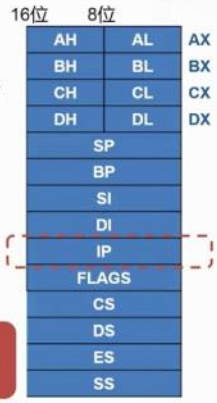

x86-64的寄存器模型
说明
代表x86-64寄存器模型的新增部分
代表IA-32寄存器模型的新增部分
代表8086的寄存器模型
64位
32位
16位
8位
RAX
RBX
RCX
RDX
RSP
RBP
RSI
RDI
RIP
RFLAGS
AH
AL
AX
BH
BL
BX
CH
CL
CX
DH
DL
DX
SP
BP
SI
DI
IP
FLAGS
CS
DS
ES
SS
FS
GS
R8
R9
...
R15

8086的指令指针寄存器

- 指令指针寄存器 IP（Instruction Pointer）
  - 保存一个内存地址，指向当前需要取出的指令
  - 当CPU从内存中取出一个指令后，IP会自动增加，指向下一指令的地址（注：实际情况会复杂的多）
  - 程序员不能直接对IP进行存取操作
  - 转移指令、过程调用/返回指令等会改变IP的内容

IP寄存器的寻址能力：
2^{16}=65536(64K)字节单元

8086对外有20位地址线
寻址范围：2^{20}=1M字节单元

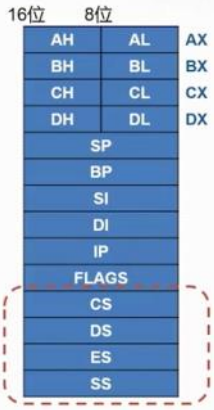


解决矛盾方法：段加偏移的物理地址生成方式

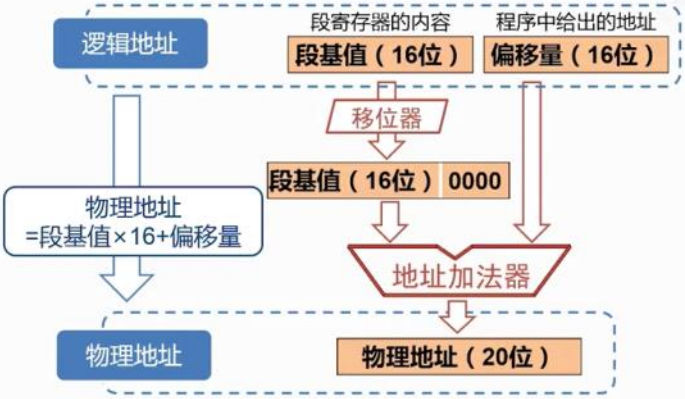
8086的段寄存器

- 段寄存器（Segment Register）
  - 与其它寄存器联合生成存储器地址

CS	代码段寄存器（Code Segment）
DS	数据段寄存器（Data Segment）
ES	附加段寄存器（Extra Segment）
SS	堆栈段寄存器（Stack Segment）



8086的物理地址生成



MOV指令和寻址方式的示例

```
MOV EBX, 40
MOV AL, BL
MOV ECX, [1000H]

MOV [DI], AX

MOV WORD PTR[BX+SI*2+200H], 01H
```

直接给出操作数

给出存放操作数的寄存器名称

给出存放操作数的存储器地址

给出存放“存放操作数的存储器地址”的寄存器名称

给出“存放操作数的存储器地址”的计算方法

注：BYTE PTR ：字节长度标记
WORD PTR ：字长度标记
DWORD PTR ：双字长度标记

不同类型的MOV指令编码

76543210	76543210	76543210	76543210	76543210	76543210
1100011w	mod 000 r/m	DISP-LO	DISP_HI	data	data if w=1
100010dw	mod reg r/m	DISP-LO	DISP_HI		
10001110	mod 0 SR r/m	DISP-LO	DISP_HI		
10001100	mod 0 SR r/m	DISP-LO	DISP_HI		
1010000w	addr-lo	addr-hi			
1010001w	addr-lo	addr-hi			
1011wreg	data	data if w=1			

（1）加法类指令

ADD指令（加）

- 格式：ADD DST, SRC
- 操作：DST←DST+SRC

ADC指令（带进位的加）

- 格式：ADC DST, SRC
- 操作：DST←DST+SRC+CF

INC指令（加1）

- 格式：INC OPR
- 操作：OPR←OPR+1

```
ADD BL, 8
ADD WORD PTR[BX], DX

ADD EAX, ECX
ADC EBX, EDX

INC CL
```

示例

INC Reg
指令长度为1字节

76543210
01000reg

（2）减法类指令

CMP指令（比较）

- 格式：CMP　DST, SRC
- 操作：DST - SRC
- 说明：减法操作，但不回写结果，仅影响标志位

还有逻辑指令NOT/AND，移位指令SHL DST,CNT（左移）

（2）移位指令

SHR指令（逻辑右移）

- 格式：SHR　DST, CNT
- 说明：相当于无符号数除以2^n的运算

SAR指令（算术右移）

- 格式：SAR　DST, CNT
- 说明：相当于带符号数除以2^n的运算

（1）无条件转移指令 - 直接转移

- 短转移：JMP　SHORT　LABEL
  - 操作：IP←IP+8位的位移量（-128~127Byte）
- 近转移：JMP　NEAR PTR　LABEL
  - 操作：IP←IP+16位的位移量（±32KByte）
- 远转移：JMP　FAR PTR　LABEL
  - 操作：IP←LABEL的偏移地址；CS ←LABEL的段基值

1. 位移量是一个带符号数，为LABEL的偏移地址与当前EIP/IP值之差
2. 从80386开始，近转移可以使用32位的位移量

说明

	操作码				
短转移	EB	8-bit位移量			
近转移	E9	16-bit位移量			
远转移	EA	IP	IP	CS	CS

处理器控制指令

- 作用
  - 控制CPU的功能
  - 对标志位进行操作

分组	格式	功能
标志操作指令	STC	把进位标志CF置1
	CLC	把进位标志CF清0
	CMC	把进位标志CF取反
	STD	把方向标志DF置1
	CLD	把方向标志DF清0
	STI	把中断标志IF置1
	CLI	把中断标志IF清0
外同步指令	HLT	暂停
	WAIT	等待
	ESC	交权
	LOCK	封锁总线（指令前缀）
空操作	NOP	空操作

串传送指令说明

MOVSB指令（字节串传送）

- 格式：MOVSB
- 操作：在存储器中将指定位置的一个字节单元传送到另一个指定的位置

REP前缀（无条件重复）

- 格式：REP 串操作指令
- 操作：当CX≠0时，重复执行串操作指令

串操作指令的特性

- 隐含操作数
  - 源串地址为DS:SI，目的串地址为ES:DI
  - 串的长度在CX寄存器中

- 处理完一个串元素后的操作（硬件自动完成）
  ① 修改SI和DI，指向下一个串元素
  ② 若使用重复前缀，则CX ←CX-1

串传送方向（标志寄存器中的DF标志位）

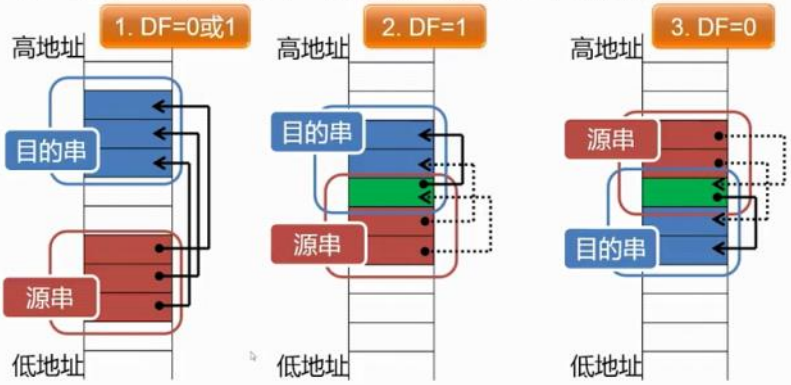
- 设置DF=0
  - 从“源串”的低地址开始传送
  - 传送过程中，SI和DI自动增量修改
- 设置DF=1
  - 从“源串”的高地址开始传送
  - 传送过程中，SI和DI自动减量修改

标志操作指令	
STD	把方向标志DF置1
CLD	把方向标志DF清0

标志位 \ 串元素位宽	字节	字
方向标志DF=0	SI←SI+1; DI←DI+1	SI←SI+2; DI←DI+2
方向标志DF=1	SI←SI-1; DI←DI-1	SI←SI-2; DI←DI-2

方向标志的作用

- 应对“源串”和“目的串”的存储区域部分重叠的问题



LOOPNE/LOOPNZ指令说明

LOOPNE/LOOPNZ指令（不为零/不相等时循环）

- 格式：LOOPNE LABEL
  或 LOOPNZ LABEL

- 操作
  ① CX←CX-1
  ② 若CX≠0且ZF=0，转移到LABEL处继续执行
  否则，结束循环，顺序执行下一条指令

循环控制指令示例

- 在100个字符的字符串中寻找第一个$字符

```
        MOV CX, 100
        MOV SI, 0FFFH
NEXT:   INC  SI
        CMP  BYTE PTR [SI], '$'
        LOOPNZ  NEXT
```

在循环出口分析查找情况

ZF=0 CX=0	查找完毕，在串中没有$字符
ZF=1 CX≠0	已找到$字符，通过CX的内容可确定位置
ZF=1 CX=0	已找到$字符，在串的最后一个字符处

XLAT指令示例

```
TAB   DB   3FH, 06H, 5BH, 4FH, 66H
      DB   6DH, 7DH, 07H, 7FH, 6FH
...
MOV   BX, OFFSET  TAB
...
MOV   AL, 4
XLAT
...
MOV   AL, 6
XLAT
...
```

字节型数据表

数据表起始地址的偏移量

04H AL

66H AL

06H AL

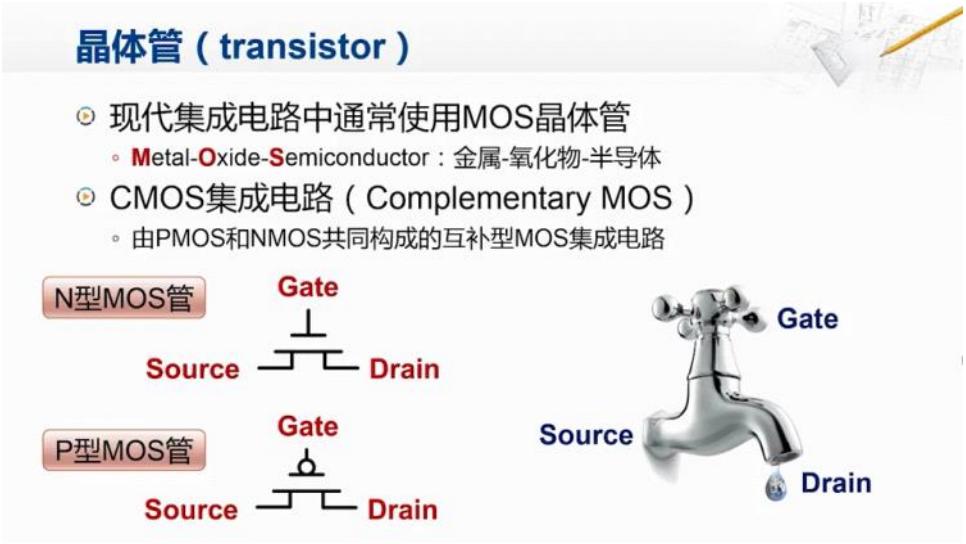
7DH AL

MIPS指令：有三个操作数，将三和二操作结果存入一中

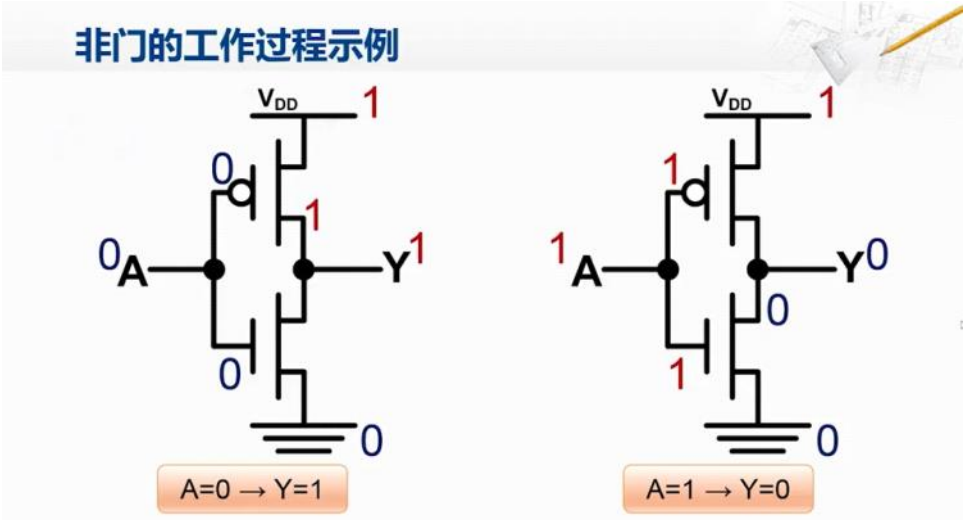
Chapter3

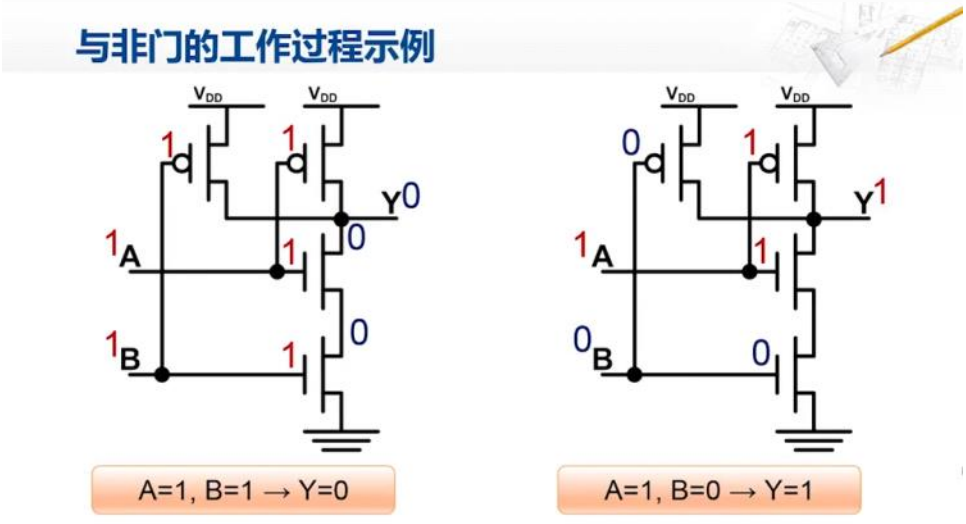
Sunday, February 28, 2016 2:38 PM

门电路的基本原理：

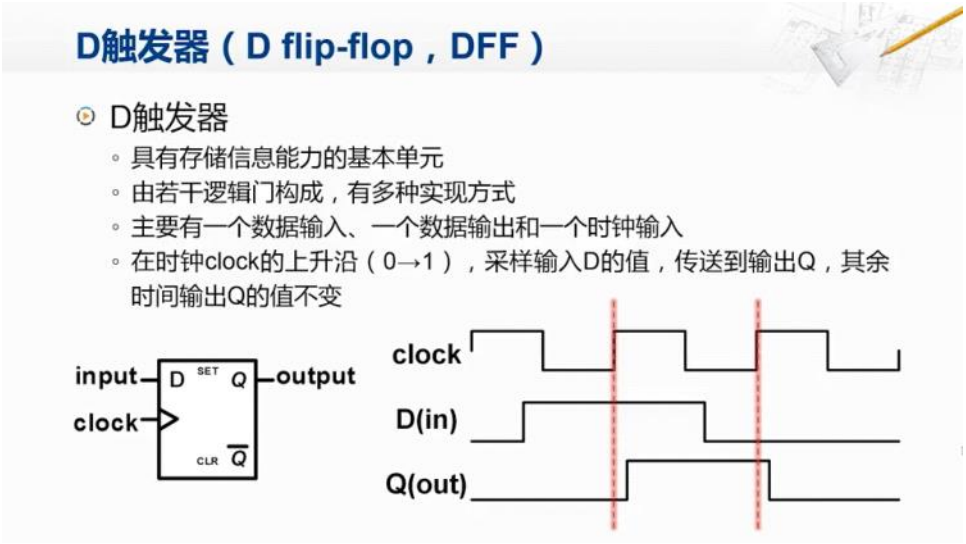


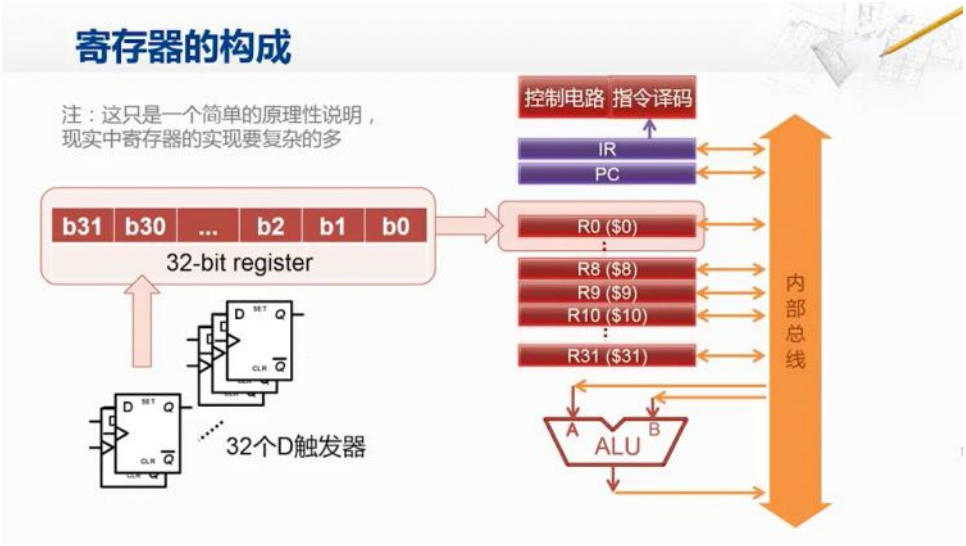
N型gate接高电平导通，P型接低电平。



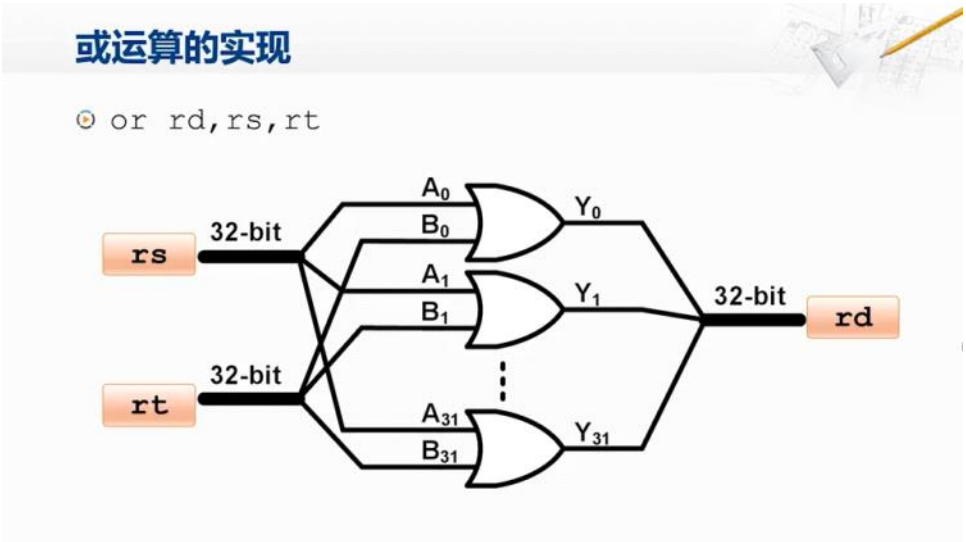


寄存器的基本原理：

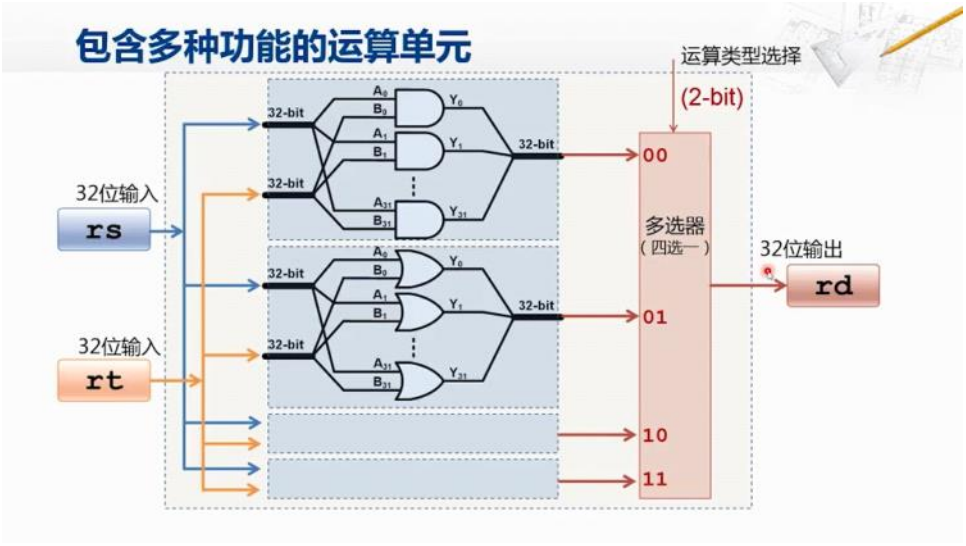




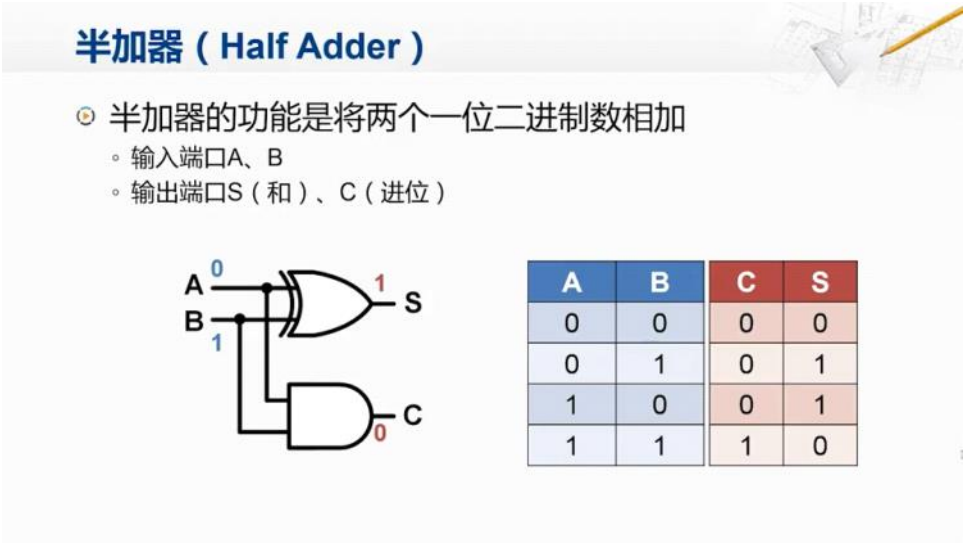
逻辑运算的实现：



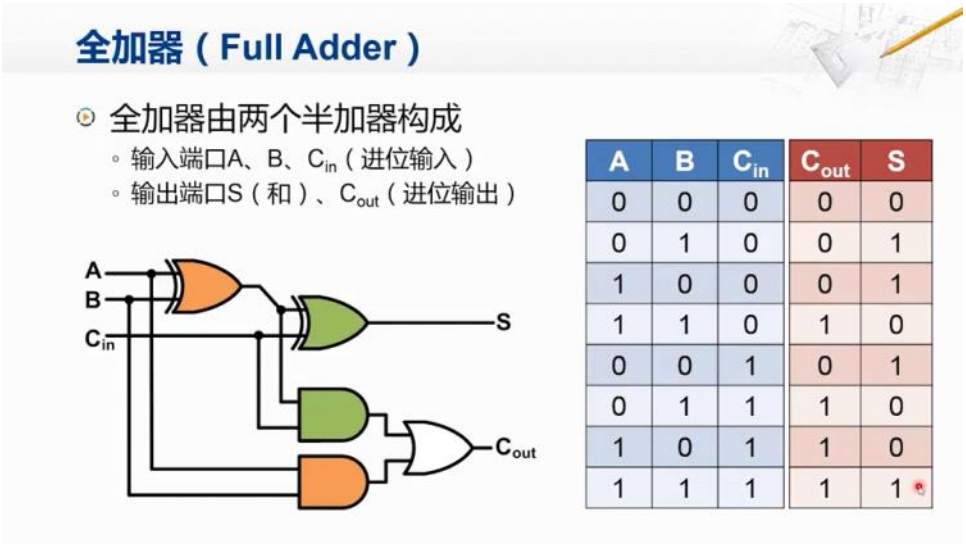
接32个或门



半加器由异或门和与门构成



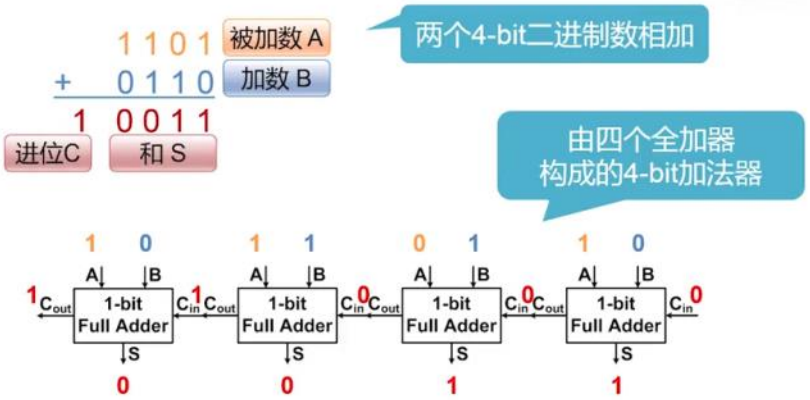
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



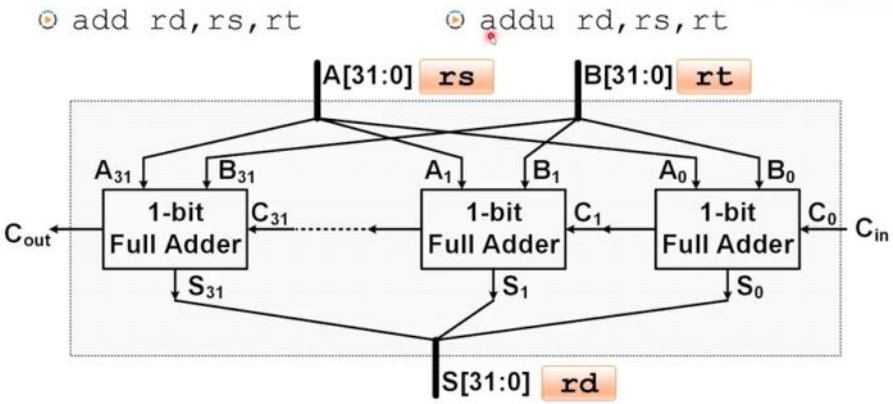
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

左边和等于二进制的右边

4-bit加法器



加法运算的实现示例



检查加法是否溢出：

溢出仅限于有符号数，正正得负或者负负得正时出现溢出

“进位”和“溢出”示例

- 注意区分“进位”和“溢出”
  - 有“溢出”时，不一定有“进位”
  - 有“进位”时，不一定有“溢出”

无符号数：3 + 5 = 8
有符号数：3 + 5 = (-8)

```
  0 0 1 1
+ 0 1 0 1
0 1 0 0 0
```

有“溢出”，无“进位”

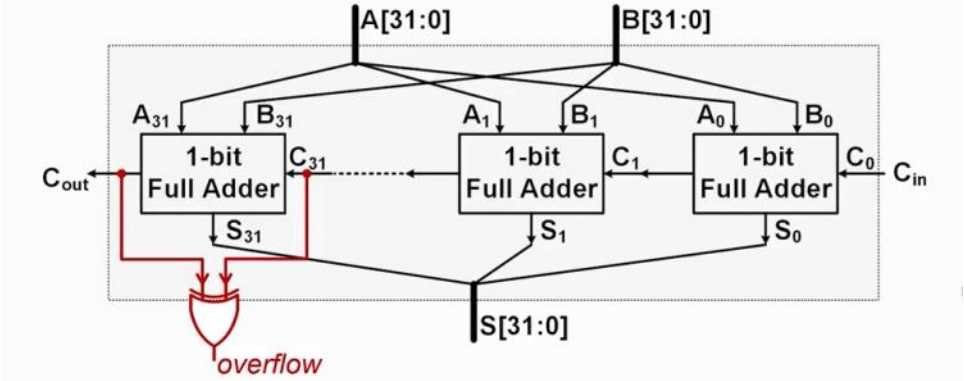
无符号数：14 + 12 = 10
有符号数：(-2)+(-4)=(-6)

```
  1 1 1 0
+ 1 1 0 0
1 1 0 1 0
```

有“进位”，无“溢出”

“溢出”的检查方法

- “最高位的进位输入”不等于“最高位的进位输出”



异或门连接最高位，结果为1则溢出

对“溢出”的处理方式：MIPS

- 提供两类不同的指令分别处理

（1）将操作数看做有符号数，发生“溢出”时产生异常

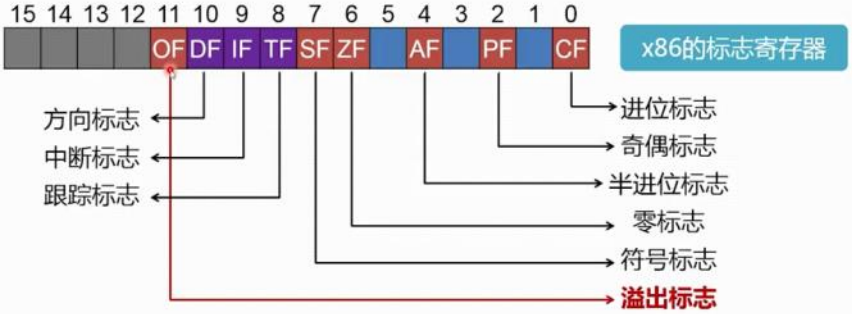
```
◦ add rd,rs,rt          # R[rd]=R[rs]+R[rt]
◦ addi rt,rs,imm        # R[rt]=R[rs]+SignExtImm
```

（2）将操作数看做无符号数，不处理“溢出”

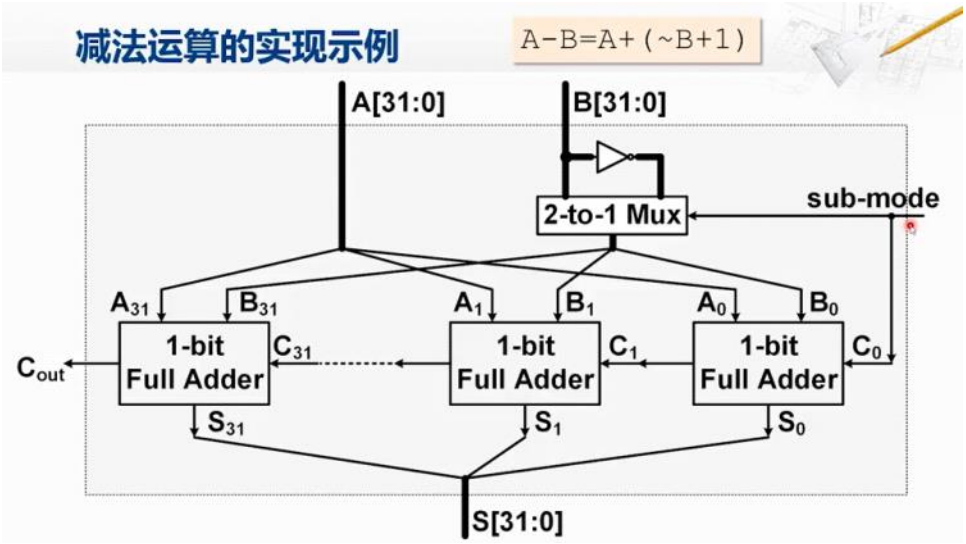
```
◦ addu rd,rs,rt         # R[rd]=R[rs]+R[rt]
◦ addiu rt,rs,imm       # R[rt]=R[rs]+SignExtImm
```

对“溢出”的处理方式：x86

- 溢出标志OF（Overflow Flag）
  - 如果把操作数看做有符号数，运算结果是否发生溢出
  - 若发生溢出，则自动设置OF=1；否则，OF=0

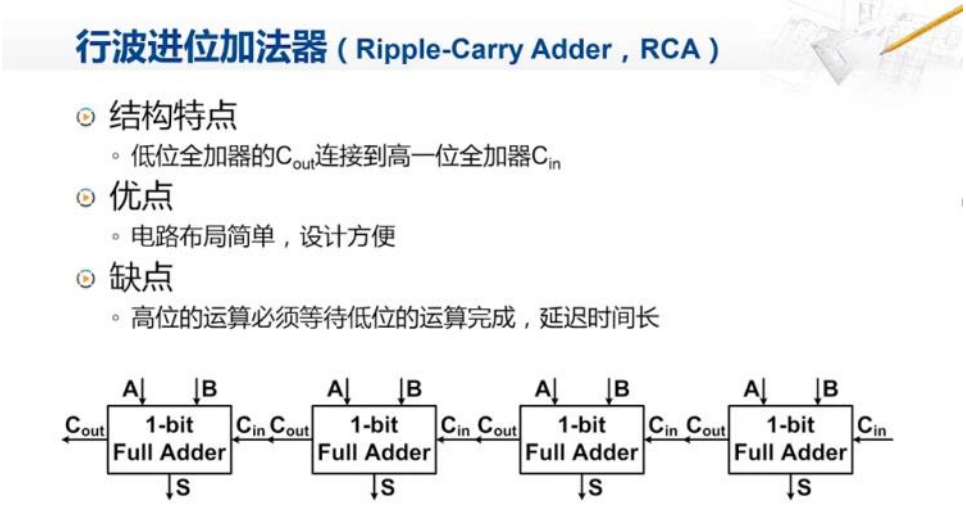


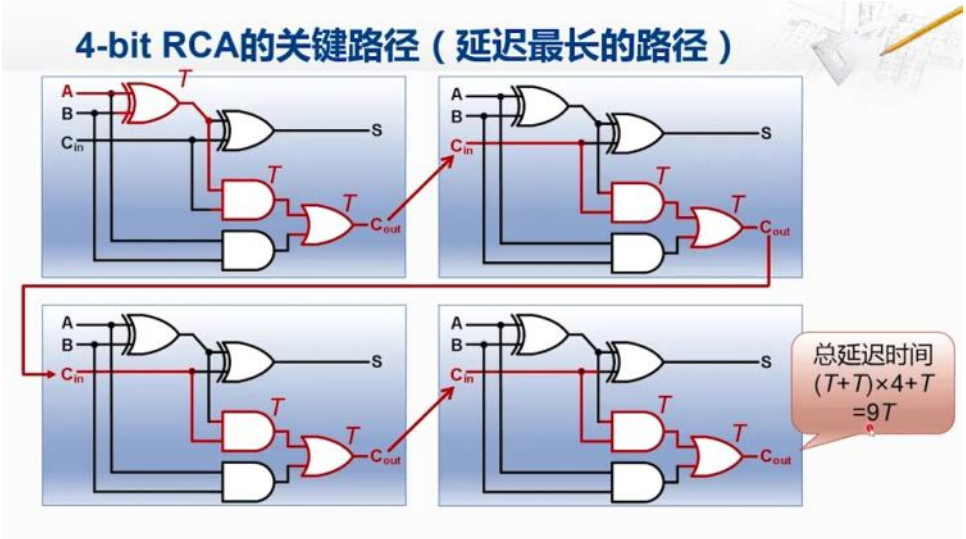
减法运算就是对相反数的加法，相反数等于原数逐位取反后加上一（推理过程：x+(~x)=11111111=-1）



通过选择器，若为加法，C0=0，直接计算；若为减法，C0=1，B逐位取反进行计算。

加法器的优化：





32bit RCA延迟时间65T

优化思路：提前计算出进位信号（输入的三数中只要有两个以上1即可）

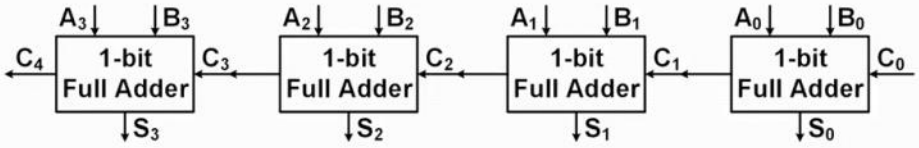
进位输出信号的分析

$$C_{i+1} = (A_i \cdot B_i) + (A_i \cdot C_i) + (B_i \cdot C_i)$$
$$= (A_i \cdot B_i) + (A_i + B_i) \cdot C_i$$

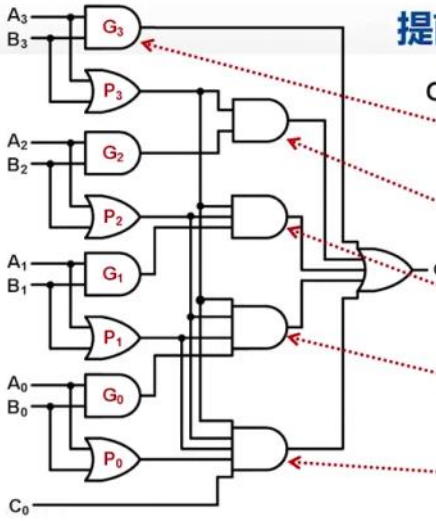
设：

- 生成（Generate）信号：$G_i = A_i \cdot B_i$
- 传播（Propagate）信号：$P_i = A_i + B_i$

则：$C_{i+1} = G_i + P_i \cdot C_i$



提前计算C_4的电路实现



$C_4 =$

G_3

$+$

$P_3 \cdot G_2$

$+$

$P_3 \cdot P_2 \cdot G_1$

$+$

$P_3 \cdot P_2 \cdot P_1 \cdot G_0$

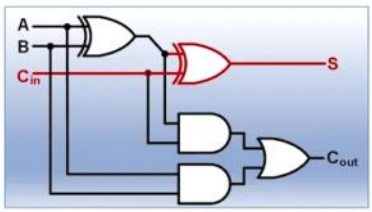
$+$

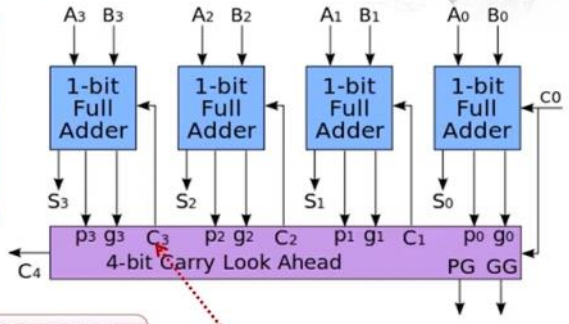
$P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$

优点：计算C_{i+1}的延迟时间固定为三级门延迟，与加法器的位数无关

缺点：如果进一步拓宽加法器的位数，则电路变得非常复杂

超前进位加法器（Carry-Lookahead Adder，CLA）





最后一级全加器还需要1级门延迟

总延迟时间为4级门延迟

计算C_3需要3级门延迟

参考值：4-bit行波进位加法器的总延迟时间为9级门延迟

32-bit加法器的实现

- 如果采用行波进位
  - 总延迟时间为65级门延迟

- 如果采用完全的超前进位
  - 理想的总延迟时间为4级门延迟
  - 实际上电路过于复杂，难以实现

$$C_{31}=G_{30}+P_{30}\cdot G_{29}+P_{30}\cdot P_{29}\cdot G_{28}+\ldots$$
$$+P_{30}\cdot P_{29}\cdot P_{28}\cdot\ldots\cdot P_2\cdot P_1\cdot P_0\cdot C_0$$

需要32输入的与门和或门？！

- 通常的实现方法
  - 采用多个小规模的超前进位加法器拼接而成
  - 例如，用4个8-bit的超前进位加法器连接成32-bit加法器

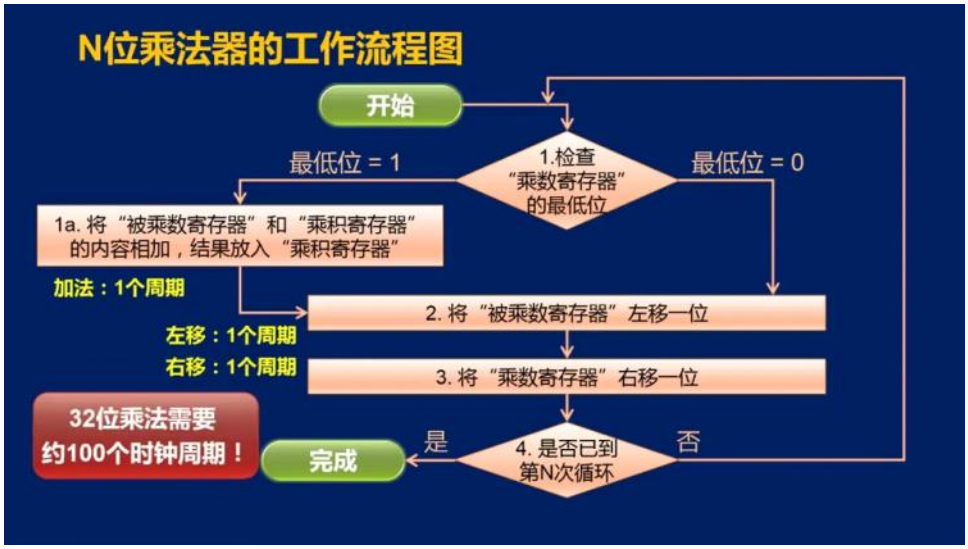
	延迟时间	时钟频率
32-bit RCA	1.3ns	769MHz
单个CLA	0.08ns	/
4级CLA	0.26ns	3.84GHz

注：参照28nm制造工艺，门延迟设为0.02ns

Chapter4

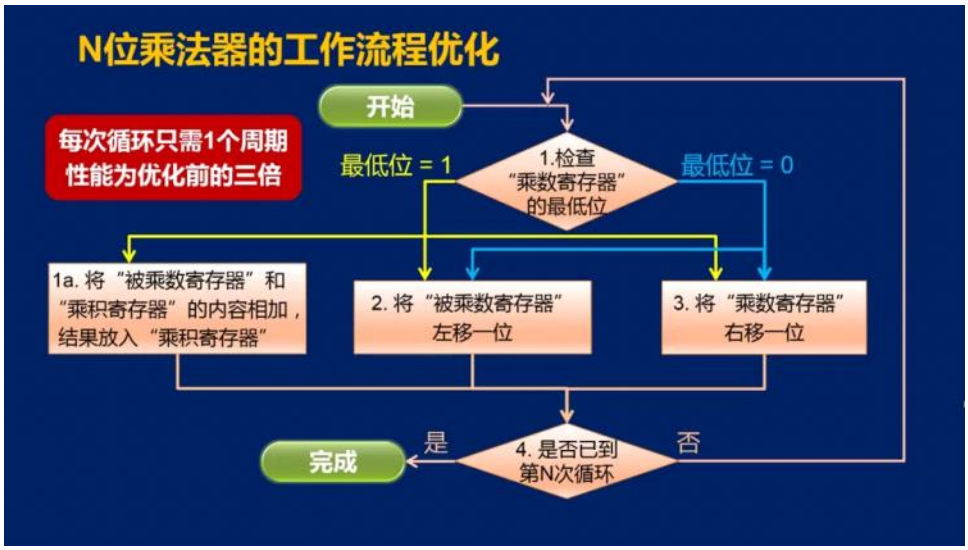
Sunday, February 28, 2016 11:50 PM

乘法器的实现：

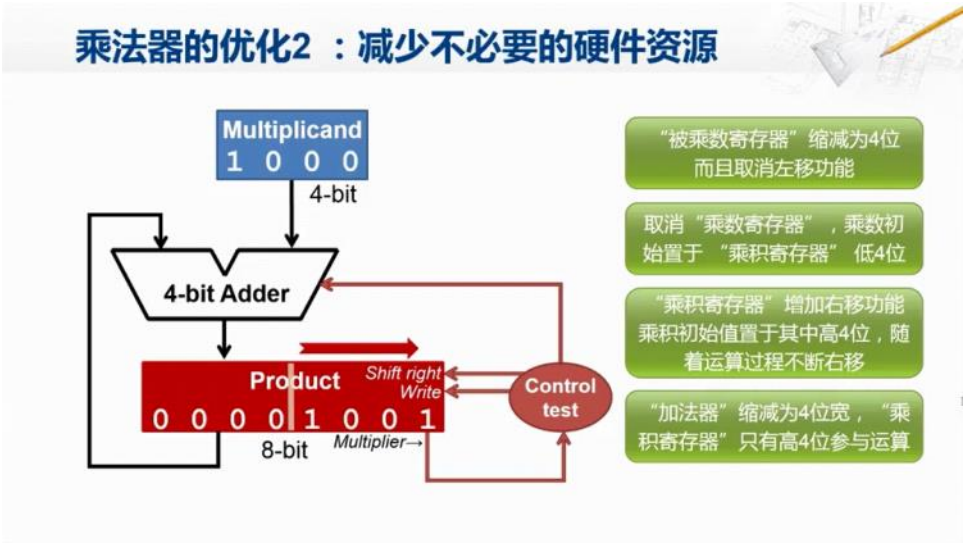


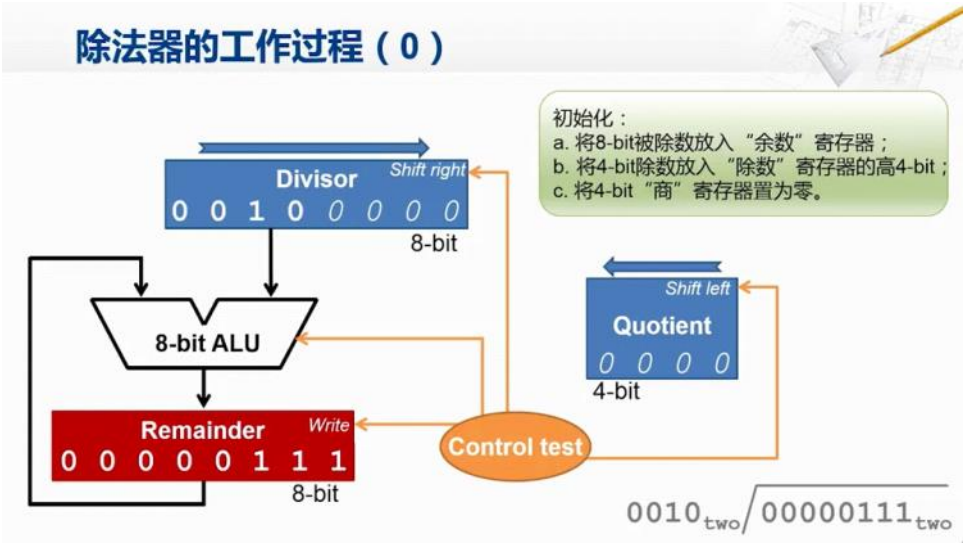
乘法器的优化：

1. 时钟上升沿到来之前乘积寄存器是不会改变的，三个步骤可以同时进行。



2. 各寄存器空间存在浪费

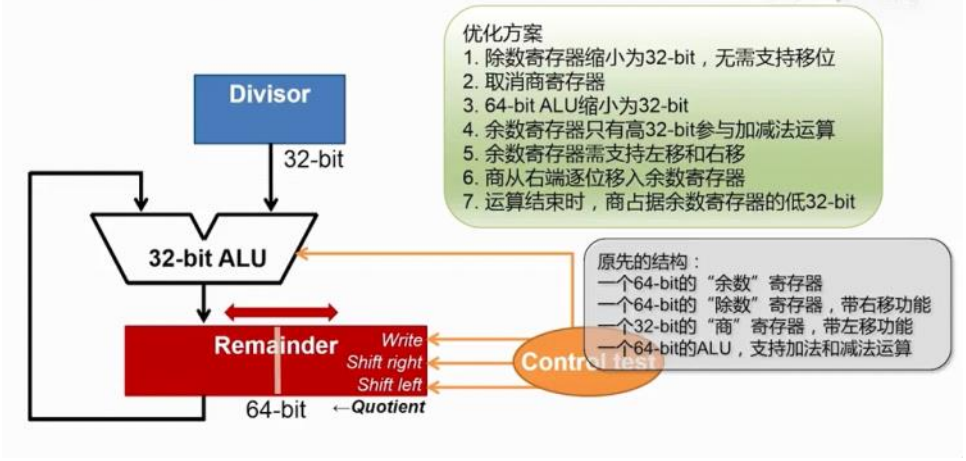




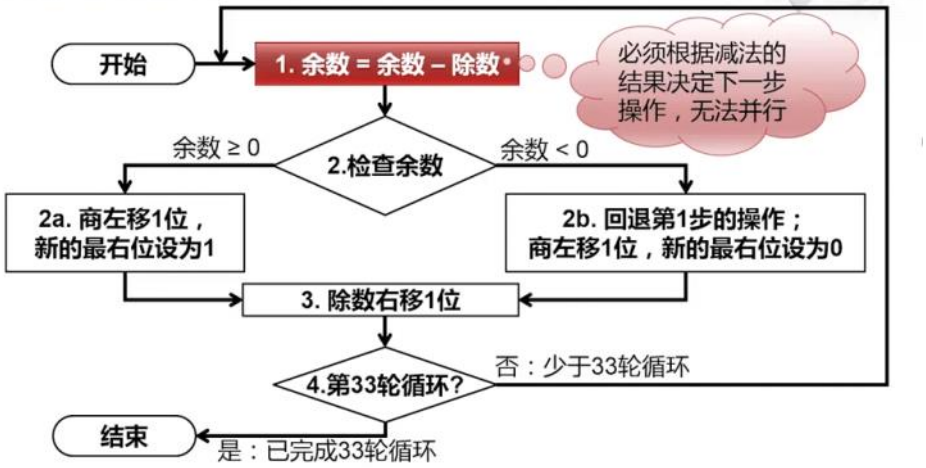
执行五轮减法，并将结果存入余数寄存器，若结果大于0，则商左移一位，右边补一个1，除数右移一位；若结果小于0，则余数寄存器的值回退到上一步，除数右移一位，商左移一位，补一个0.

除法器的优化：

除法器的实现（第二版）



除法的性能优化分析



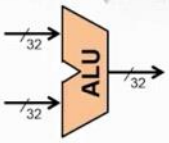
Chapter5

Monday, February 29, 2016 3:40 PM

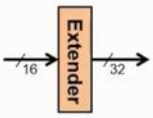
处理器的设计就是如何改写PC。

指令系统的需求

- 算术逻辑单元（ALU）
  - 运算类型：加、减、或、比较相等
  - 操作数：2个32位的数，来自寄存器 或 扩展后的立即数



- 立即数扩展部件
  - 将一个16立即数扩展为32位数
  - 扩展方式：零扩展、符号扩展



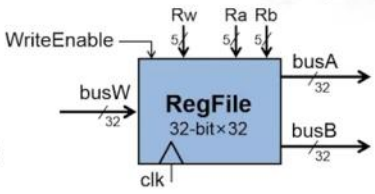
- 程序计数器（PC）
  - 一个32位的寄存器
  - 支持两种加法：加4 或 加一个立即数

指令系统的需求

- 寄存器堆
  - 每个寄存器为32位宽，共32个
  - 支持读操作：rs 和 rt
  - 支持写操作：rt 或 rd
  - 注：这称为“两读一写”的寄存器堆
- 存储器
  - 一个只读的指令存储器，地址和数据均为32位
  - 一个可读写的数据存储器，地址和数据均为32位
  - 注：这两个存储器实际对应了CPU中的指令和数据高速缓存（Cache）

存储组件：寄存器堆

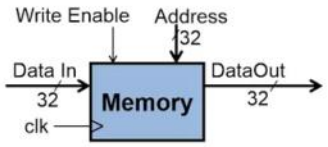
- 内部构成
  - 32个32位的寄存器
- 数据接口信号
  - busA，busB：两组32位的数据输出
  - busW：一组32位的数据输入



- 读写控制
  - Ra(5位)：选中对应编号的寄存器，将其内容放到busA
  - Rb(5位)：选中对应编号的寄存器，将其内容放到busB
  - Rw(5位)：选中对应编号的寄存器，在时钟信号（clk）的上升沿，如果写使能信号有效（WriteEnable==1），将busW的内容存入该寄存器
  - 注：寄存器堆的读操作不受时钟控制

存储组件：存储器

- 数据接口信号
  - Data In：32位的数据输入信号
  - Data Out：32位的数据输出信号



- 读写控制
  - Address：32位的地址信号。该信号指定一个存储单元，将其内容送到数据输出信号
  - Write Enable：写使能信号。在时钟信号（clk）的上升沿，如果写使能信号有效（为1），将数据输入信号的内容存入地址信号指定存储单元
  - 注：存储器的读操作不受时钟控制

处理器的设计步骤

① 分析指令系统，得出对数据通路的需求 ✓

② 为数据通路选择合适的组件 ✓

③ 连接组件建立数据通路

④ 分析每条指令的实现，以确定控制信号

⑤ 集成控制信号，形成完整的控制逻辑

Chapter6

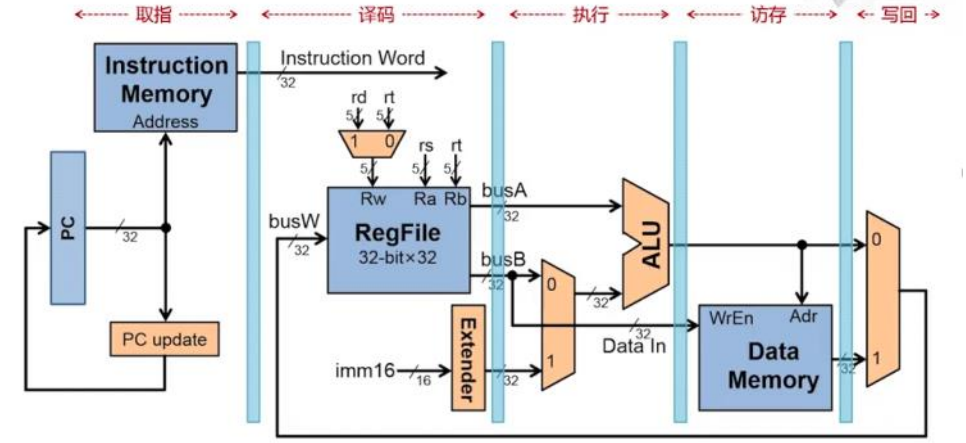
Thursday, March 3, 2016 1:10 PM

流水线：

执行指令的主要步骤（MIPS）

1、取指（Fetch）	从存储器取指令，更新PC
2、译码（Decode）	指令译码，从寄存器堆读出寄存器的值
3、执行（Execute）	运算指令：进行算术逻辑运算 访存指令：计算存储器的地址
4、访存（Memory）	Load指令：从存储器读数据 Store指令：将数据写入存储器
5、回写（Write-back）	将数据写入寄存器堆

添加流水线寄存器

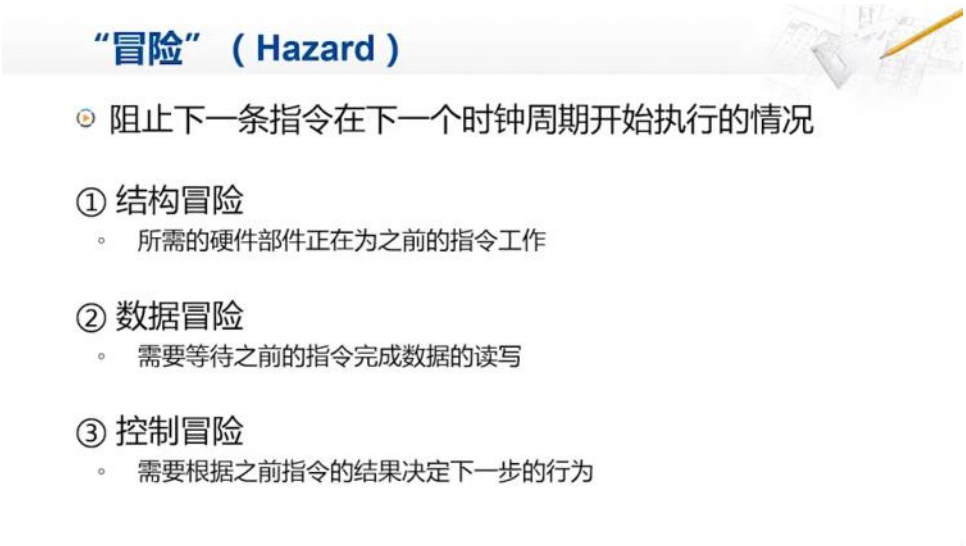


标量流水线：将周期缩短为原来执行一条指令所需的时间

超标量流水线

- 超标量（Superscalar）
  - 通常，具有两条或两条以上并行工作的流水线结构称为**超标量结构**
    - 亦称为“超标量流水线”或直接称为“超标量”
    - 与之相对，之前的流水线则称为“标量流水线”
  - 使用超标量结构的处理器称为**超标量处理器**
- 标量流水线和超标量流水线
  - 单周期→标量流水线：**时间并行性**的优化，主要是对现有硬件的切分
  - 标量流水线→超标量流水线：**空间并行性**的优化，需成倍增加硬件资源

现代的多核cpu通常是在一个cpu芯片中集成了多个超标量处理器核。



结构冒险解决方案：

（指令和数据放在同一个寄存器中）

1.流水线停顿，产生空泡（bubble）

2.指令和数据放在不同寄存器中

（读寄存器和写寄存器同时发生）

前半个周期写寄存器，后半个周期读寄存器，而且设置独立的读写口

数据冒险解决方案：

（一条指令需要前一条指令的运算结果，但是该结果还没写回）

1.流水线停顿，产生空泡。（硬件）

2.插入nop指令（软件）

3.数据前递

（一条指令需要之前指令的访存结果）：

流水线停顿加数据前递。

控制冒险解决方案：

（尚未确定是否发生分支，如何进行下一次取指）

1.流水线停顿，产生空泡。

Chapter7

2016年3月4日 16:52

存储器（运算器，主存，BIOS芯片，硬盘）的特性：
非易失性
可读可写（bios只读）
随机访问（主存和bios）
访问时间
---
cache：SRAM
Main memory：DRAM