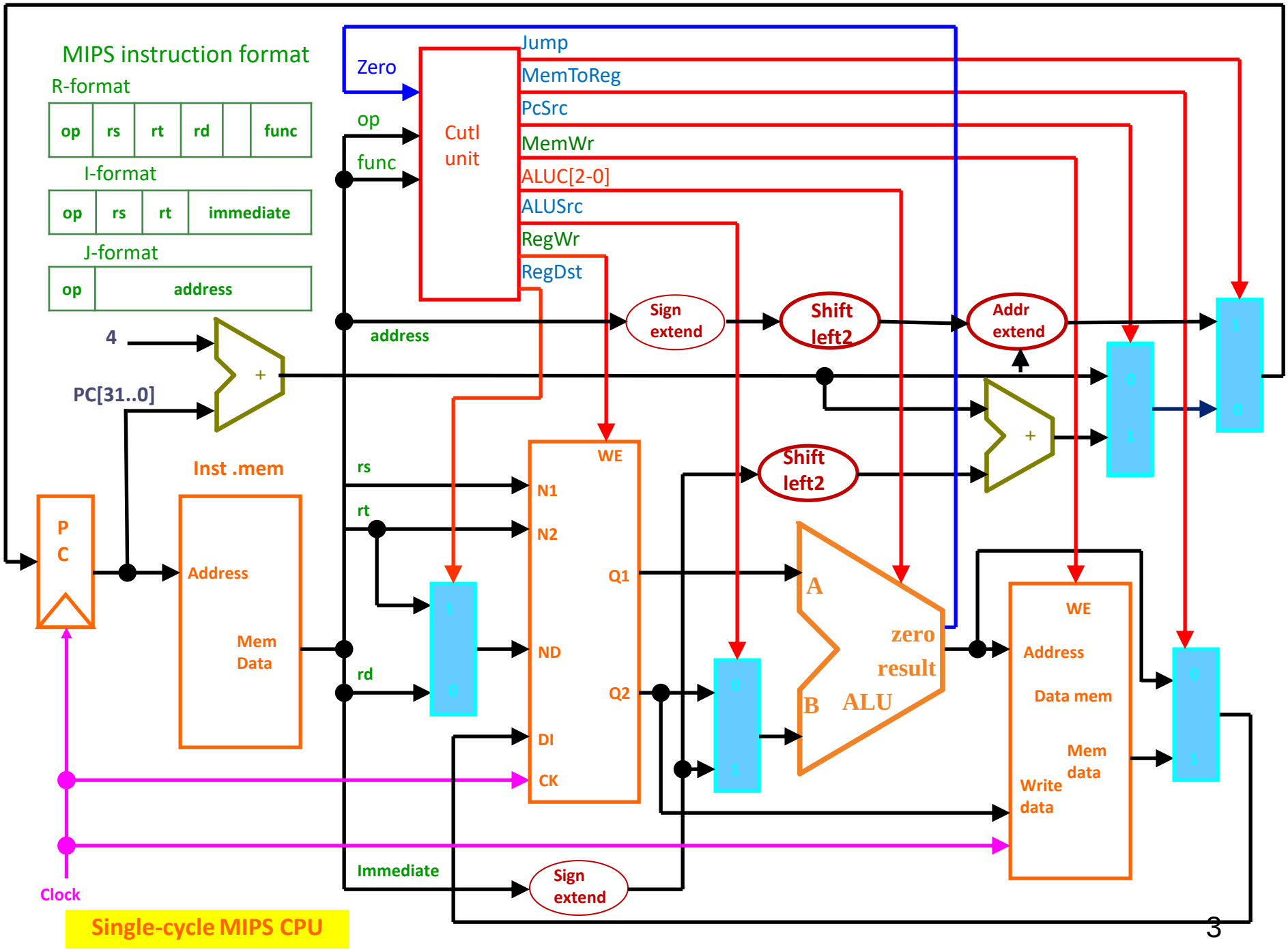
计算机与网络技术

第5讲　简易CPU设计（续）、CPU性能

课程回顾

- CPU设计基本步骤
- 系统指令数据通路设计
- 系统指令控制逻辑设计
- 系统指令数据通路与控制逻辑集成


MIPS instruction format
R-format
op
rs
rt
rd
func
I-format
op
rs
rt
immediate
J-format
op
address
Zero
op
func
Cutl unit
Jump
MemToReg
PcSrc
MemWr
ALUC[2-0]
ALUSrc
RegWr
RegDst
Sign extend
Shift left2
Addr extend
address
4
PC[31..0]
+
Inst .mem
P C
Address
Mem Data
rs
rt
rd
WE
N1
N2
ND
DI
CK
Q1
Q2
Shift left2
A
B
ALU
zero
result
WE
Address
Data mem
Mem data
Write data
Immediate
Sign extend
Clock
Single-cycle MIPS CPU

系统指令控制逻辑设计

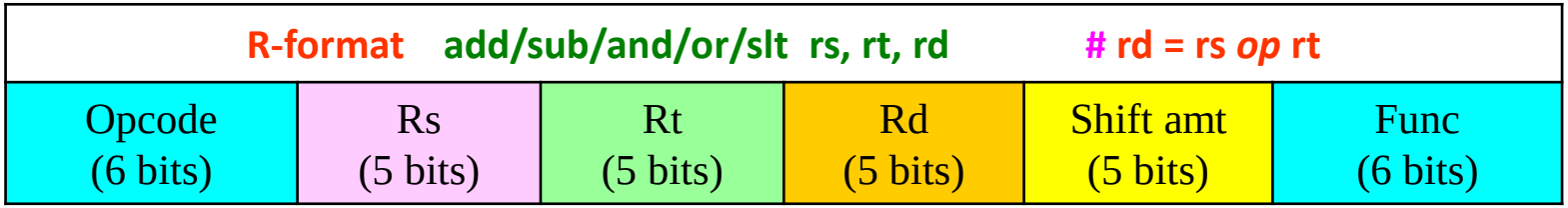
系统指令的控制逻辑集合

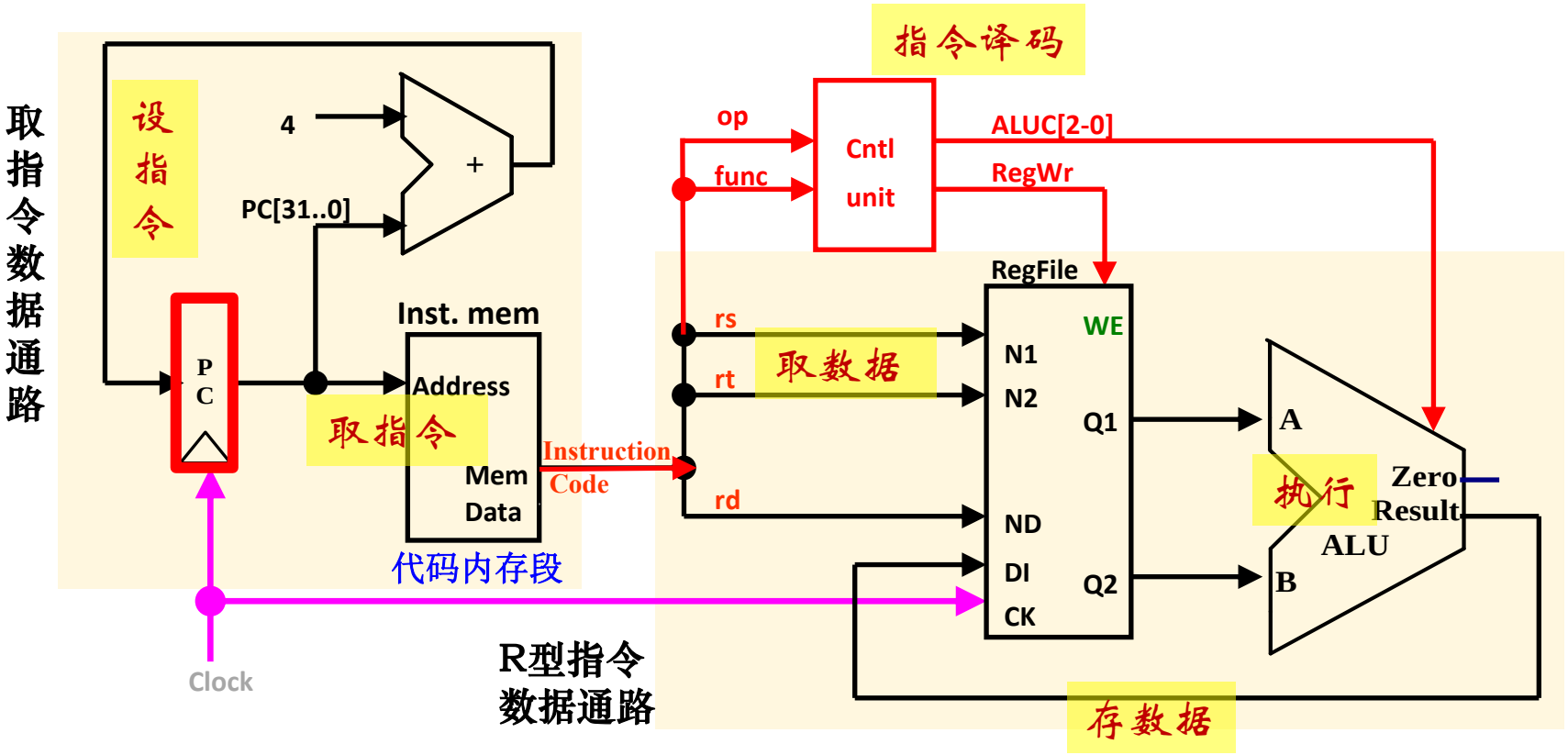
Control Signals	Instructions								
	and	or	add	sub	ori	lw	sw	beq	j
Jump	0	0	0	0	0	0	0	0	**1**
PcSrc	0	0	0	0	0	0	0	**Zero**	X
ALUSrc	0	0	0	0	**1**	**1**	**1**	0	X
ALUC[2]	0	0	0	1	0	0	0	1	X
ALUC[1]	0	0	1	1	0	1	1	1	X
ALUC[0]	0	1	0	0	1	0	0	0	X
MemToReg	0	0	0	0	0	**1**	X	X	X
RegDst	0	0	0	0	**1**	**1**	X	X	X
RegWr	**1**	**1**	**1**	**1**	**1**	**1**	0	0	0
MemWr	0	0	0	0	0	0	**1**	0	0

提纲

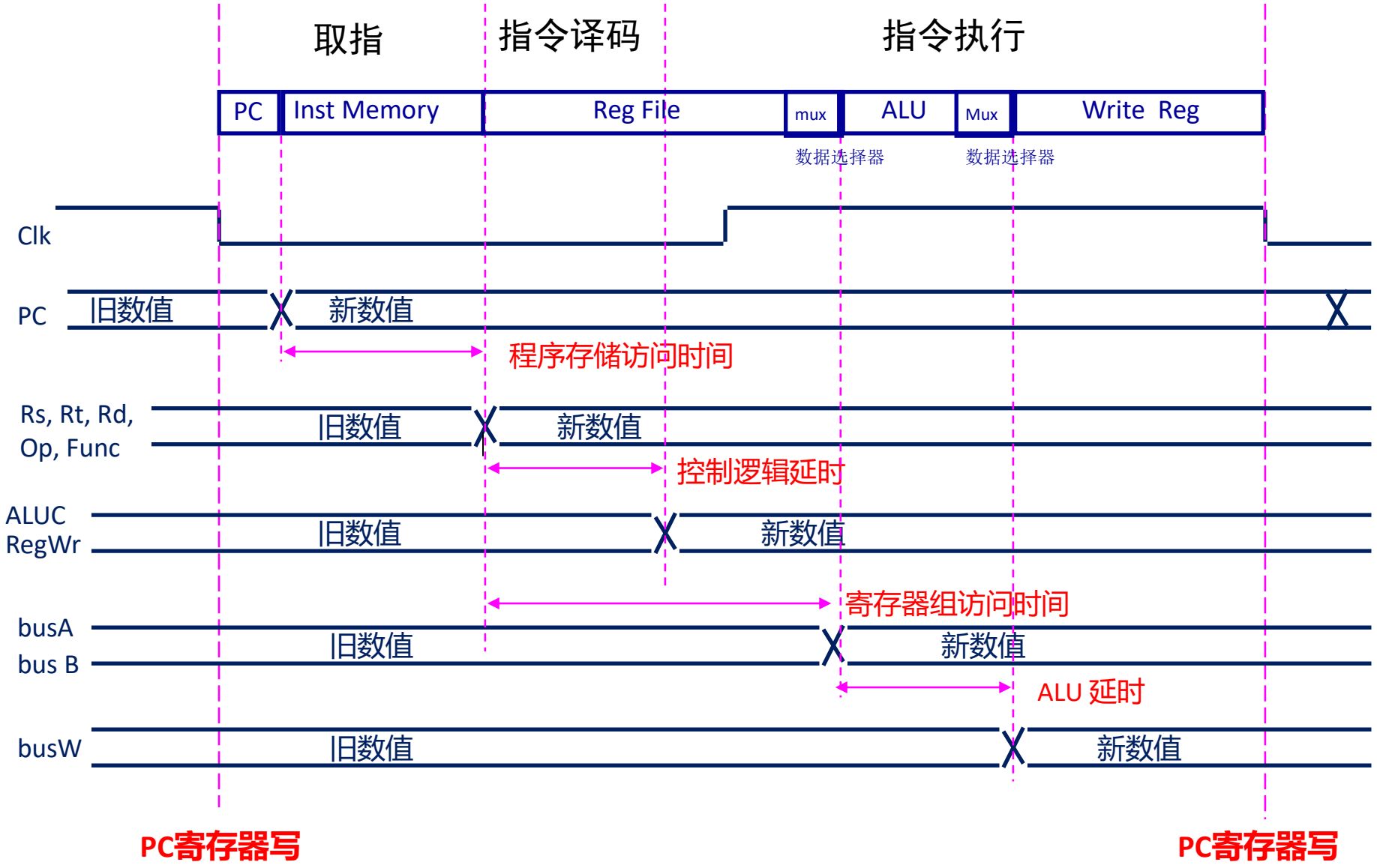
- CPU时序分析
- CPU指令周期
- 计算机性能评价与提升
- 运算方法与运算器

32位MIPS系统R指令





R格式指令时序

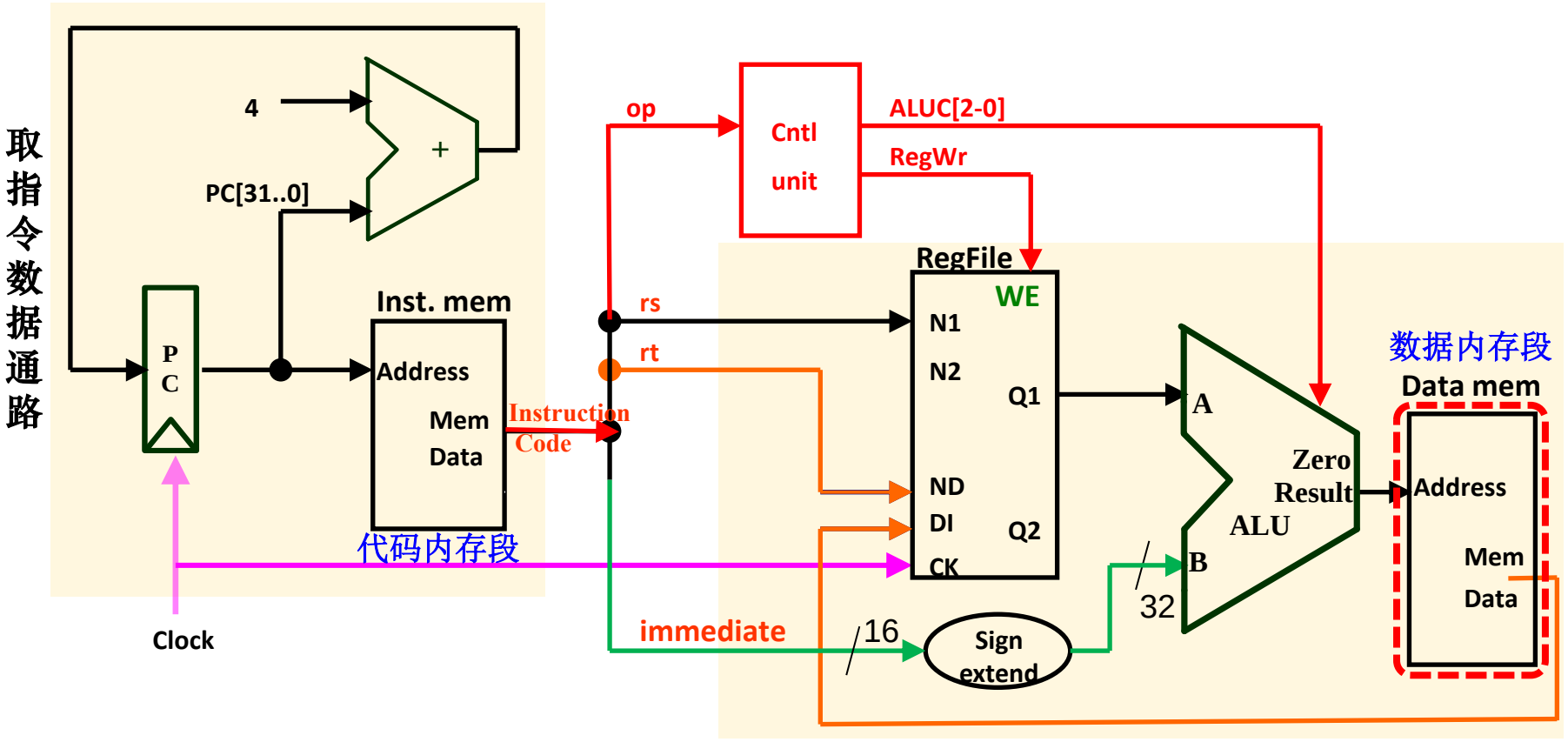

取指
指令译码
指令执行
PC
Inst Memory
Reg File
mux
ALU
Mux
Write Reg
数据选择器
数据选择器
Clk
PC
旧数值
新数值
程序存储访问时间
Rs, Rt, Rd,
Op, Func
旧数值
新数值
控制逻辑延时
ALUC
RegWr
旧数值
新数值
寄存器组访问时间
busA
bus B
旧数值
新数值
ALU 延时
busW
旧数值
新数值
PC寄存器写
PC寄存器写

单周期

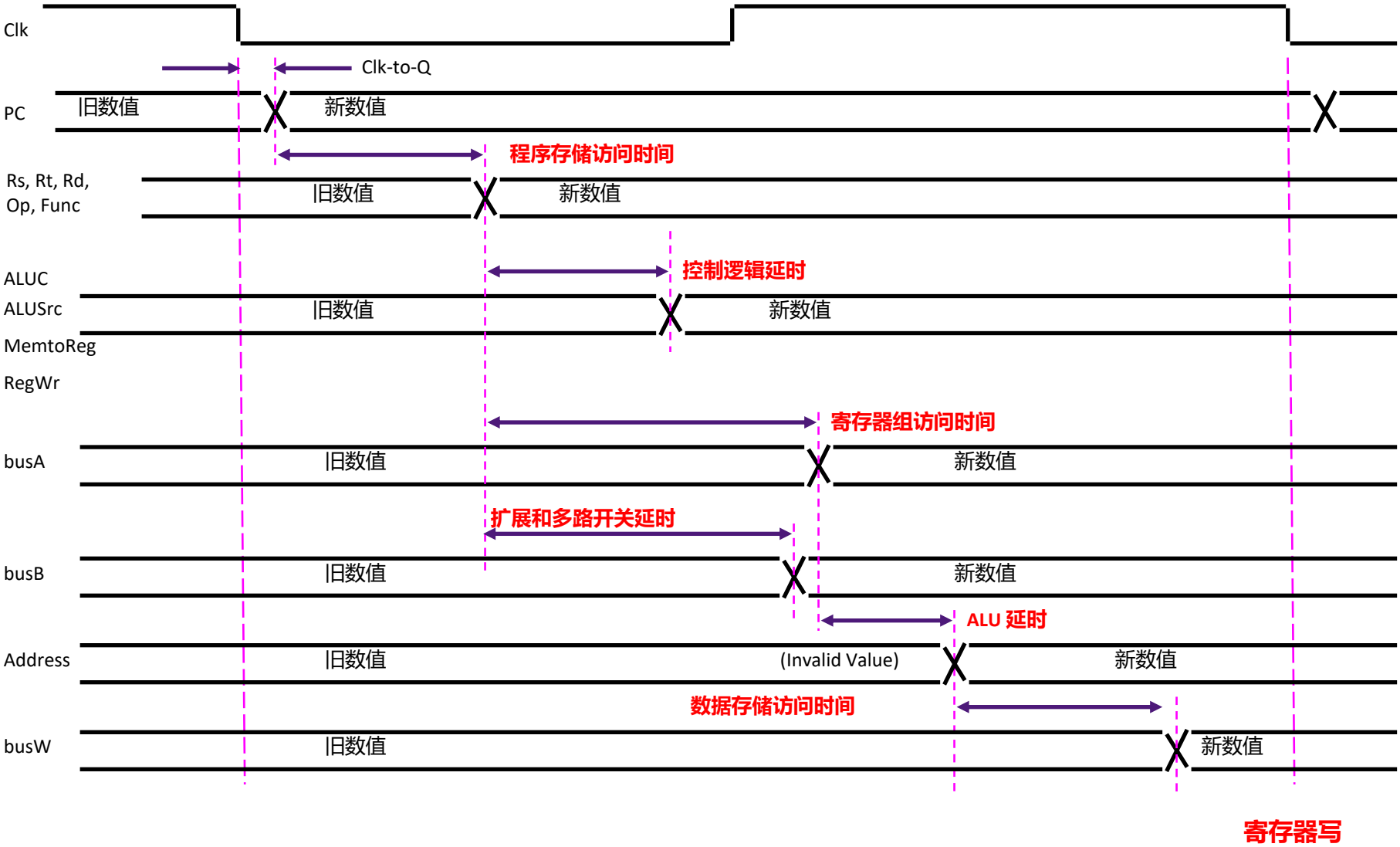
32位MIPS系统I指令

面向访存的I指令（lw）

I-format lw rt, #imm16(rs)			# rt = Memory[rs + imm16]
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Immediate (16 bits)



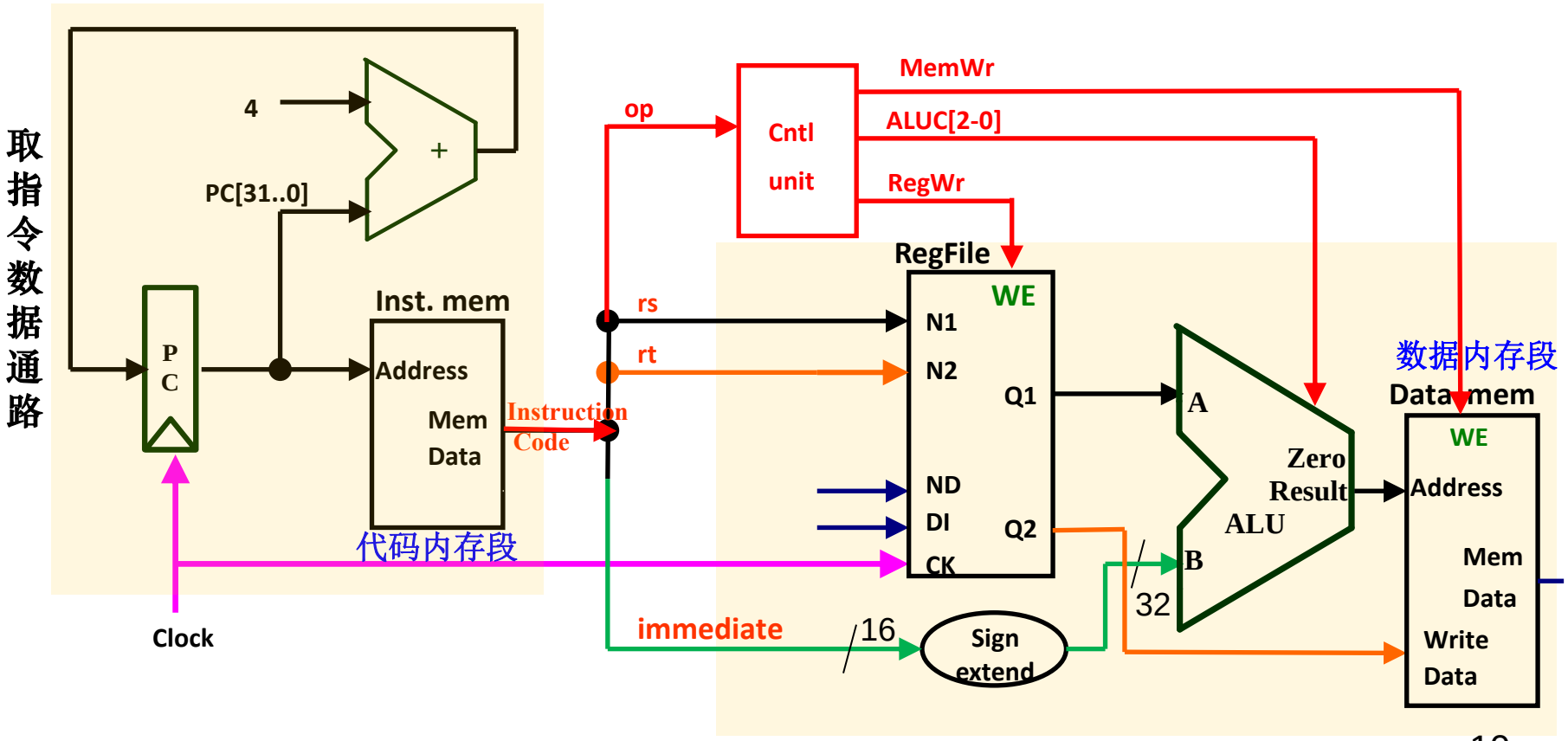
lw 指令时序


Clk
Clk-to-Q
PC
旧数值
新数值
程序存储访问时间
Rs, Rt, Rd,
Op, Func
旧数值
新数值
ALUC
控制逻辑延时
ALUSrc
旧数值
新数值
MemtoReg
RegWr
寄存器组访问时间
busA
旧数值
新数值
扩展和多路开关延时
busB
旧数值
新数值
ALU 延时
Address
旧数值
(Invalid Value)
新数值
数据存储访问时间
busW
旧数值
新数值
寄存器写

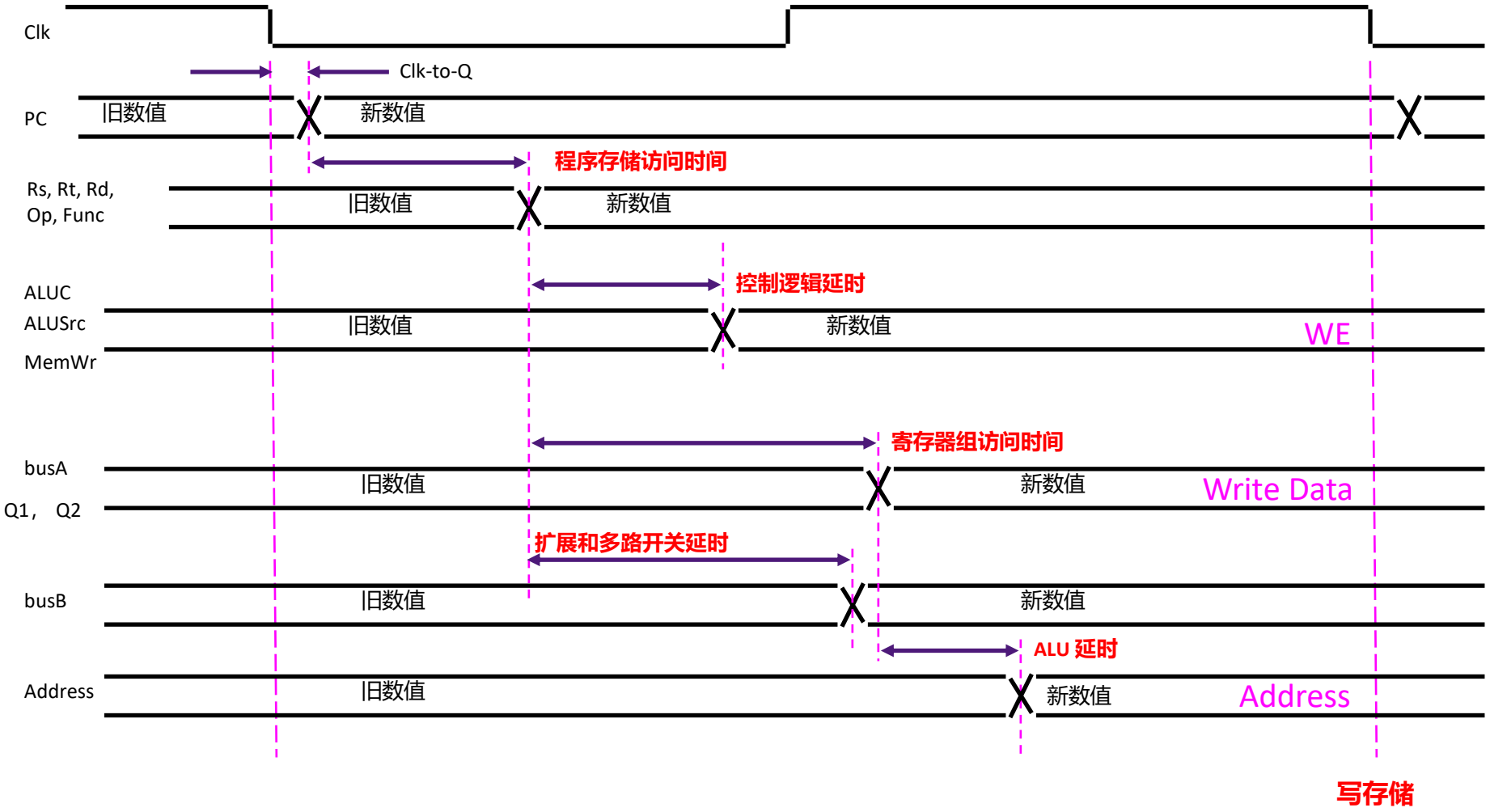
32位MIPS系统I指令

面向访存的I指令（sw）

I-format sw rt, #imm16(rs)			# Memory[rs + imm16] = rt
Opcode (6 bits)	Rs (5 bits)	Rt (5 bits)	Immediate (16 bits)



sw 指令时序

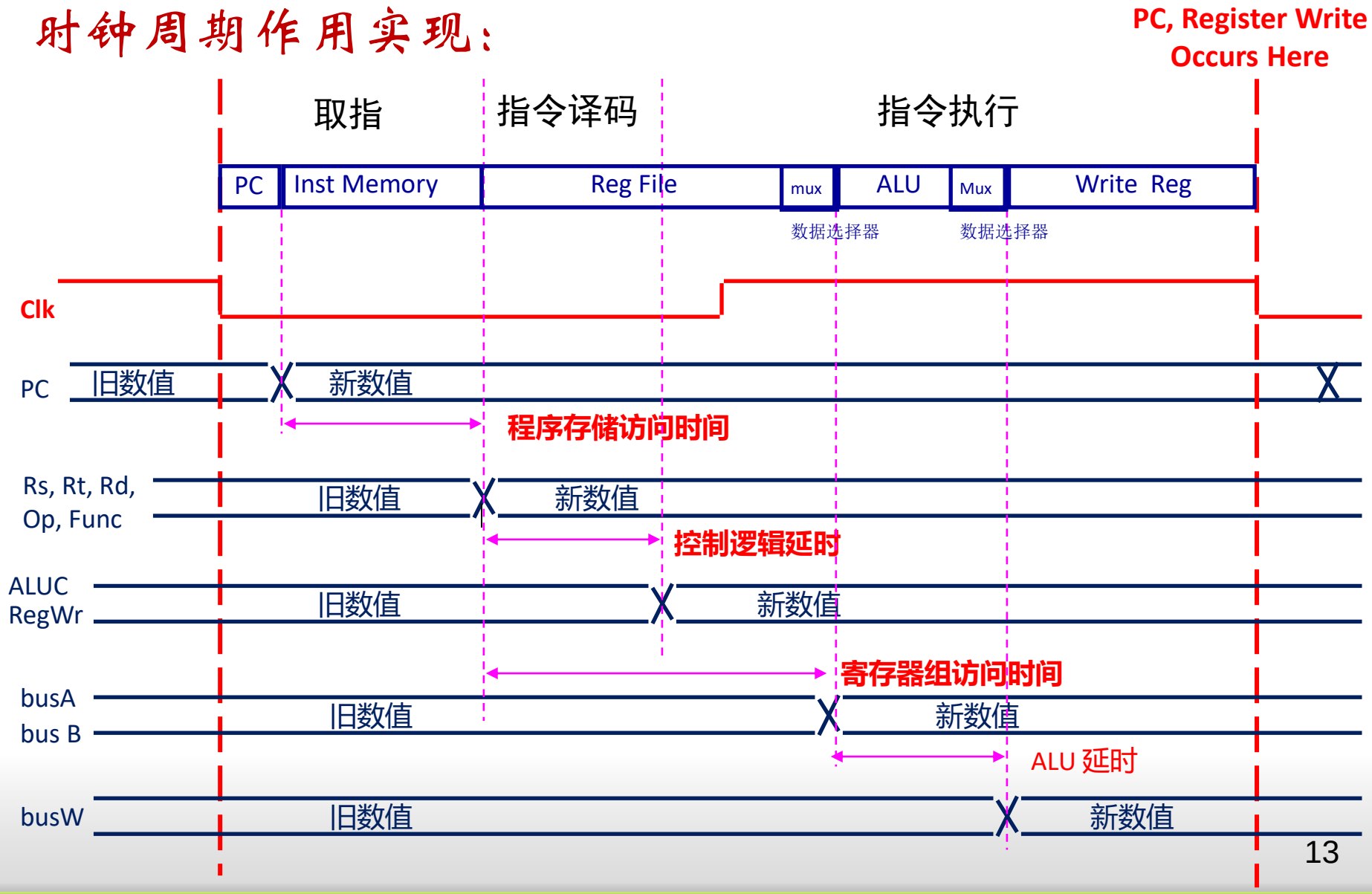

Clk
Clk-to-Q
PC
旧数值
新数值
程序存储访问时间
Rs, Rt, Rd,
Op, Func
旧数值
新数值
控制逻辑延时
ALUC
ALUSrc
MemWr
旧数值
新数值
WE
寄存器组访问时间
busA
Q1， Q2
旧数值
新数值
Write Data
扩展和多路开关延时
busB
旧数值
新数值
ALU 延时
Address
旧数值
新数值
Address
写存储

提纲

- CPU时序分析
- **CPU指令周期**
- 计算机性能评价与提升
- 运算方法与运算器

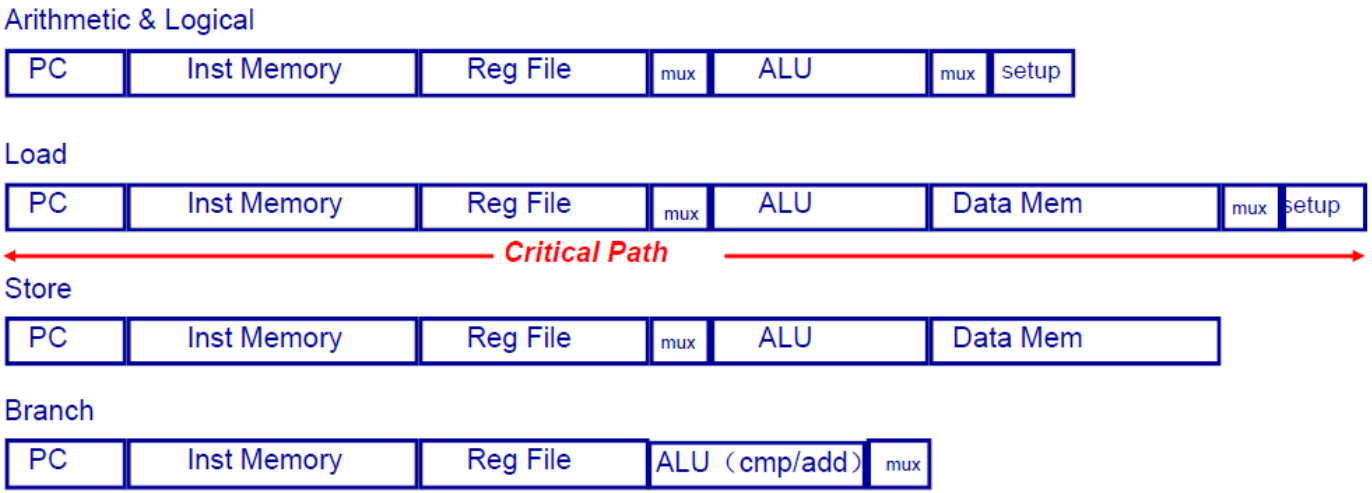
CPU指令周期

时钟周期作用实现：



CPU指令周期

CPU指令周期选择

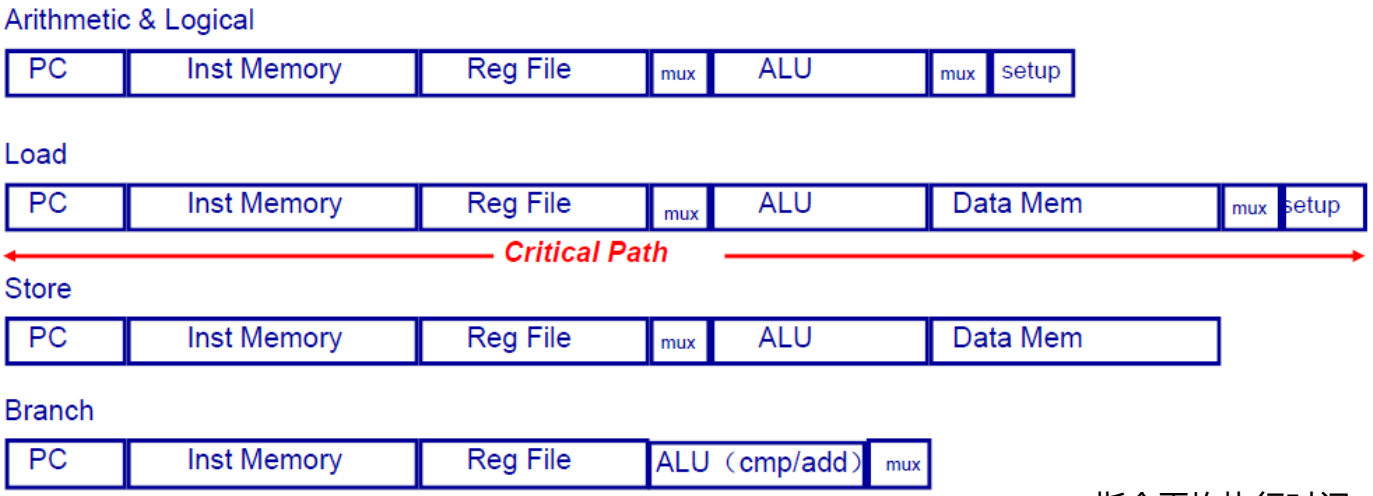


指令周期：从取指令、分析指令到执行完该指令所需的全部时间

时钟周期：Clock脉冲信号的周期

CPU指令周期

单时钟周期指令



?

指令平均执行时间：8.78
最长指令执行时间：13

- Single Cirlce Processor：所有指令在时钟的一个周期内完成
- 一个时钟周期必须满足最长指令执行时间要求
- 真实的存储器与理想存储器性能有很大差别
  - cannot always get the job done in one (short) cycle

Inst. Type	Inst. Mem	Read Reg	ALU	Data Mem	Write Reg	Total /ns	Percet ent
R Type	3	1	2		1	7	44%
Lw	3	1	2	6	1	13	24%
Sw	3	1	2	6		12	12%
Branch	3	1	2			6	18%
Jump	3					3	2%

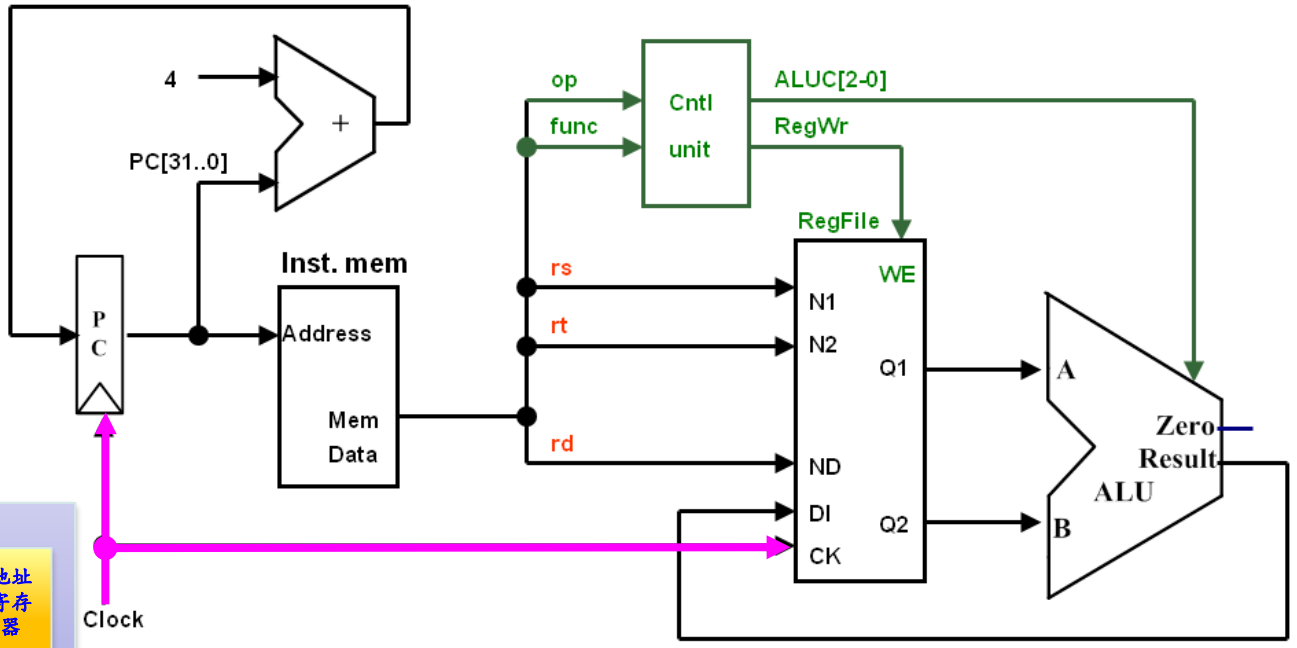
CPU指令周期

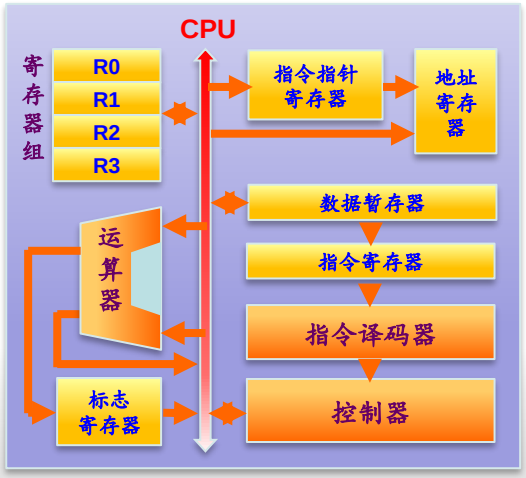
单时钟周期指令

- 效率低下
  - 不同类型的指令所完成的工作量有很大的差别，所要用到的部件和所通过的数据通路不同，所用的时间的长短也有很大的差别
- 增加实现成本
  - 每个时钟周期中功能部件最多被使用一次，如果要在执行一条指令的过程中，多次使用某一部件，那么就需要重复设置该部件

CPU指令周期

CPU指令周期选择

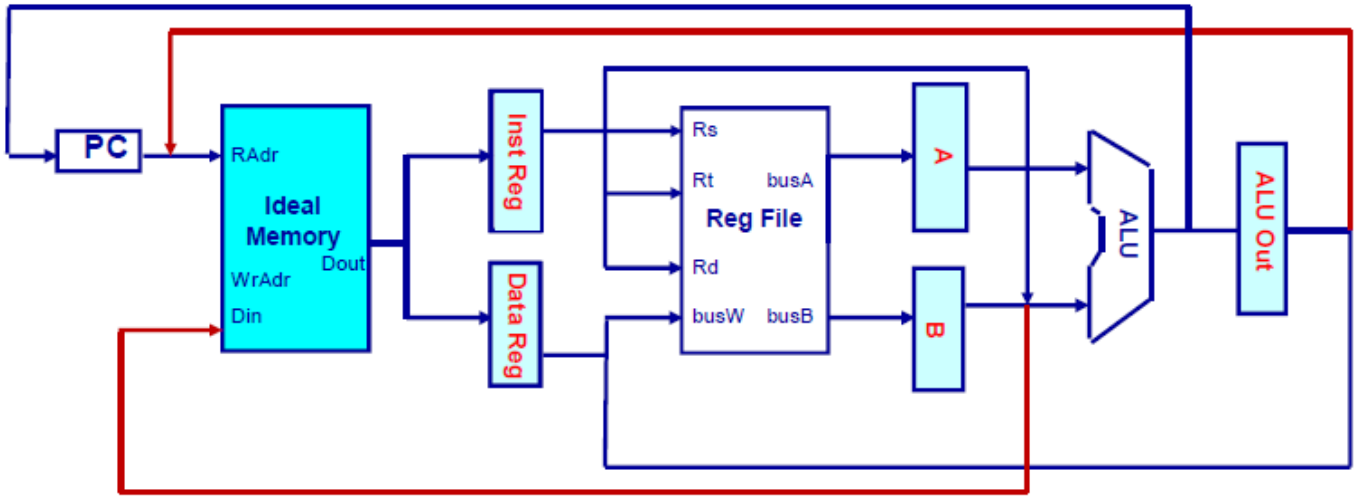


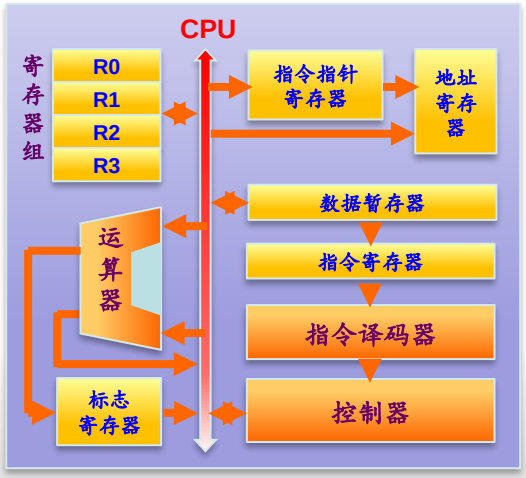


?

CPU指令周期

多时钟周期指令

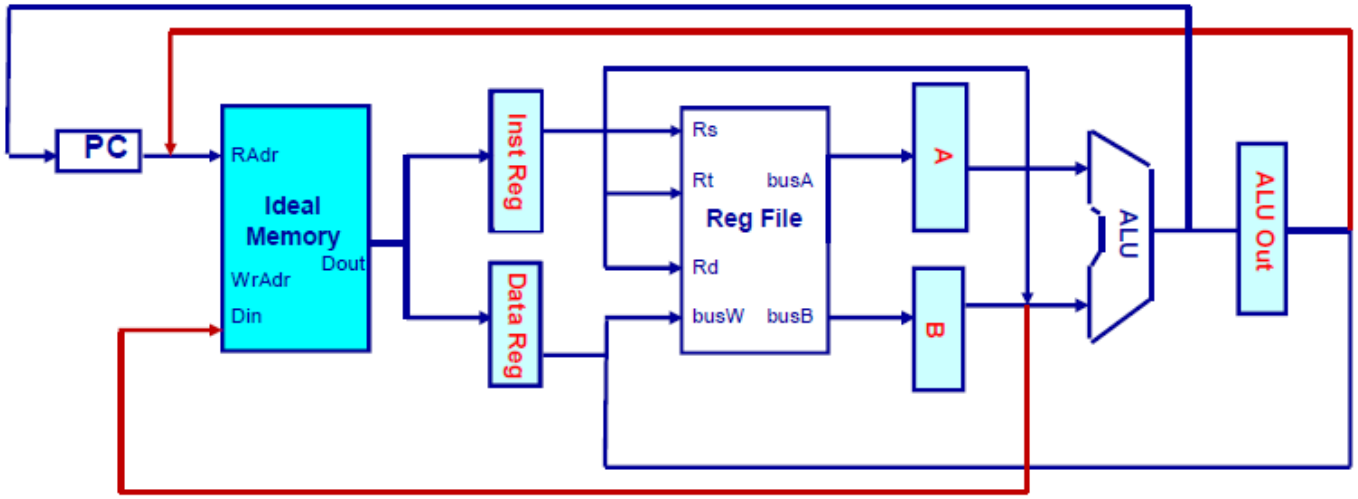




- 将指令分解为多个步骤，每个步骤一个时钟周期，执行时间长度基本相当
- 增加中间寄存器，每个时钟周期暂存一次中间结果

CPU指令周期

多时钟周期指令

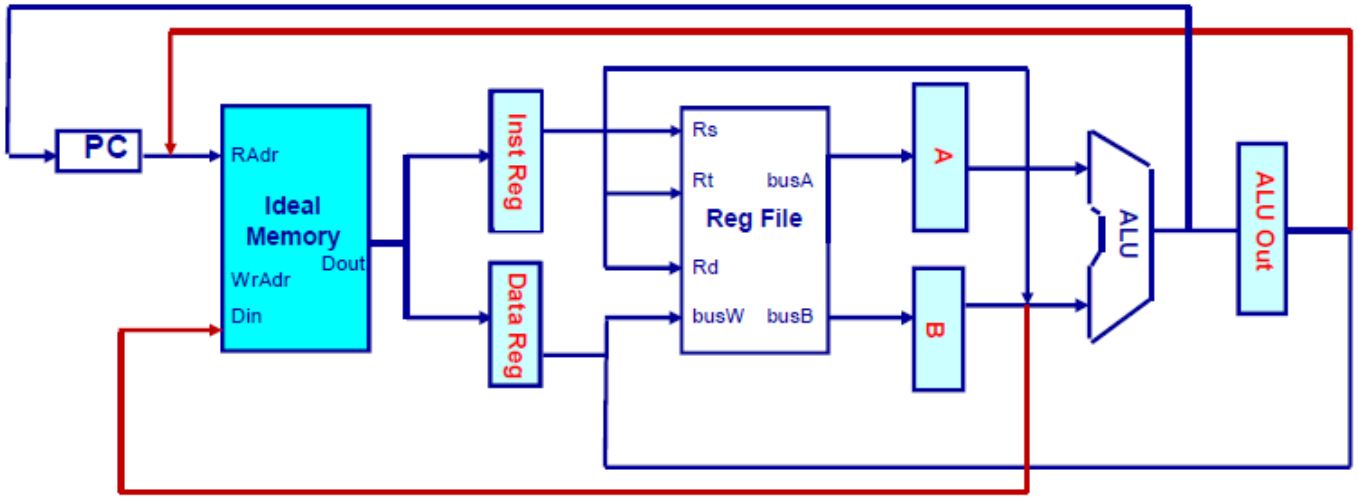


需要设置哪些临时寄存器取决于两个因素：

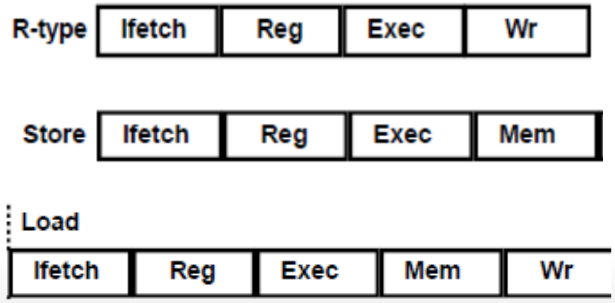
- 什么样的组合逻辑电路正好适合作为一个周期；
- 哪些数据在该指令后面的执行周期中需要用到。

CPU指令周期

多时钟周期指令

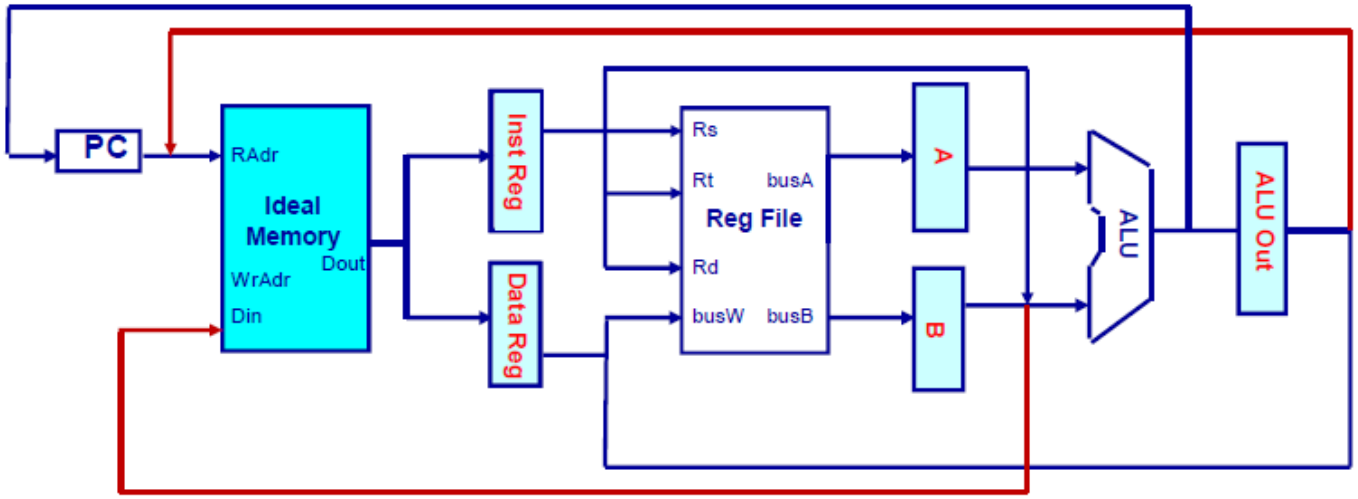


- Ifetch: Instruction Fetch，从指令存储器取指
- Reg/Dec: 寄存器读取、指令解码
- Exec: ALU执行（计算存储器地址）
- Mem: 从数据存储器读取/准备写入数据
- Wr: 将数据写回到通用寄存器



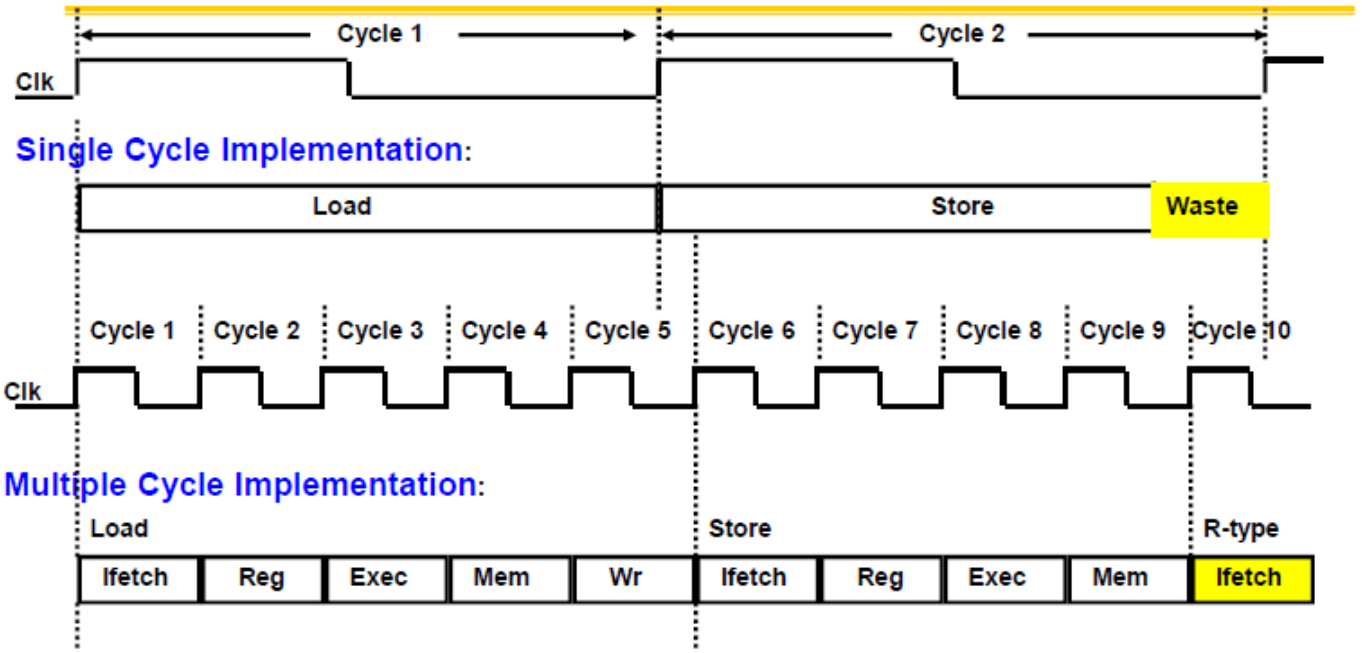
CPU指令周期

多时钟周期指令



- 可以共享同一个功能部件（如果是在不同的时钟周期使用该部件的话）：
  一个ALU，一个存储器，节省硬件
  - 指令存储器和数据存储器可以合并为一个存储器
  - 做PC+4的加法器可以和ALU合并

CPU指令周期

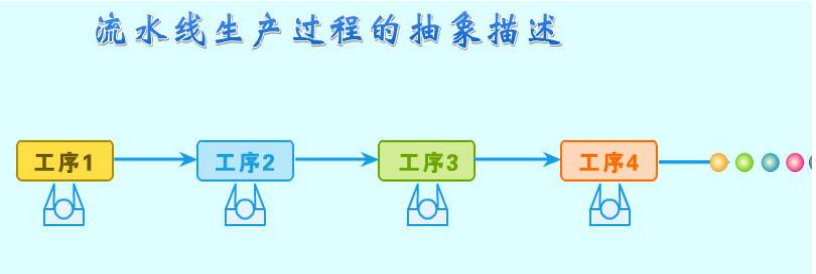


Inst. Type	Inst. Mem	Read Reg	ALU	Data Mem	Write Reg	Total /ns	Percet ent
R Type	3	1	2		1	7	44%
Lw	3	1	2	6	1	13	24%
Sw	3	1	2	6		12	12%
Branch	3	1	2			6	18%
Jump	3					3	2%

指令平均执行时间：8.78
最长指令执行时间：13

CPU指令周期

工业生产流水线



CPU指令周期

流水线技术

- 把一个重复的过程分解为若干个子过程，每个子过程由专门的功能部件来实现。
- 把多个处理过程在时间上错开，依次通过各功能段，这样，每个子过程就可以与其它的子过程并行进行。
- 流水线中的每个子过程及其功能部件称为流水线的级或段，段与段相互连接形成流水线。
- 流水线的段数称为流水线的深度。

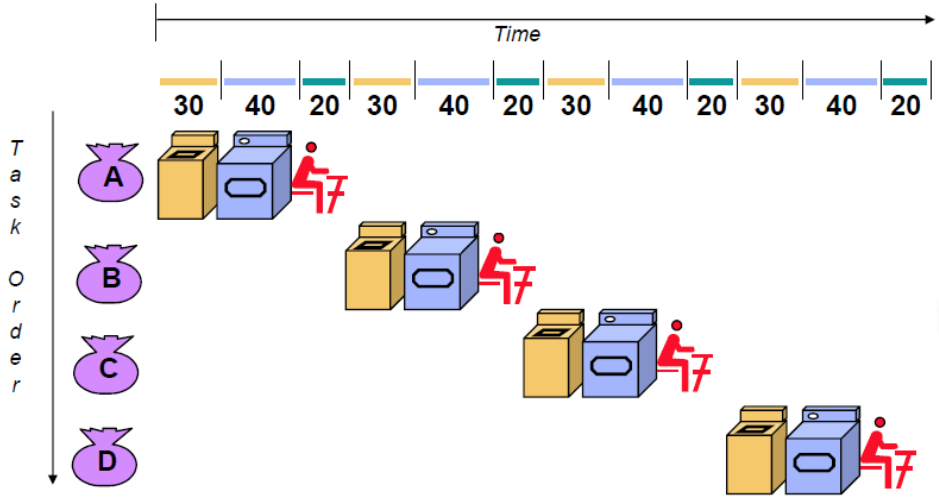
CPU指令周期

流水线技术

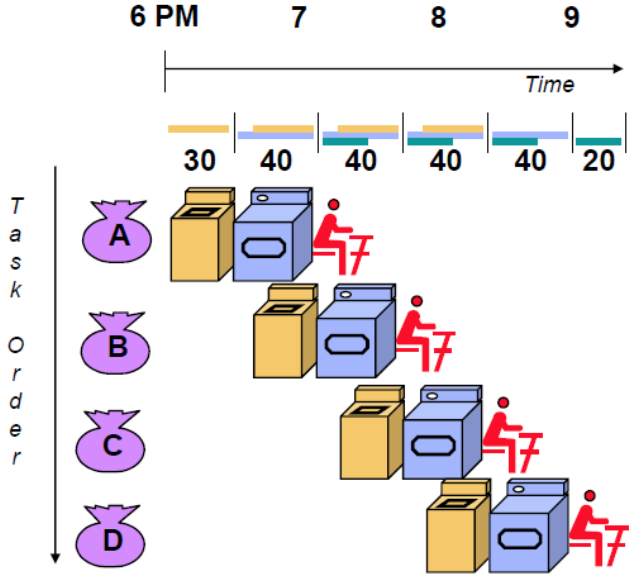
以一个洗衣房为例：

- 处理器：洗衣房
- 4条指令：A, B, C, D
  - 指令：洗衣服
  - 指令执行过程：洗衣—烘干—熨烫
  - 指令各阶段延时：30分钟—40分钟—20分钟

串行洗衣房（6小时完成4个任务）



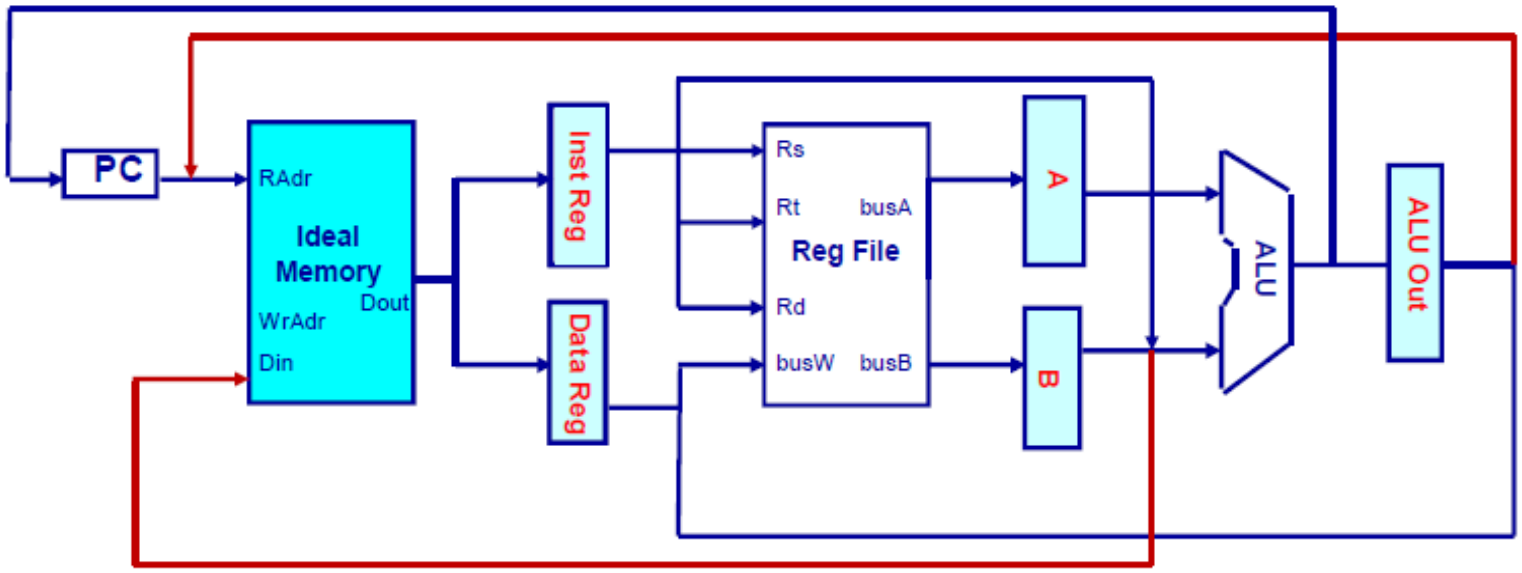
流水化洗衣房（3.5小时完成4个任务）



- 顺序处理需要6 小时完成 4 项任务
- *如果她学会了流水线操作，需要多长时间？*

CPU指令周期

CPU流水线

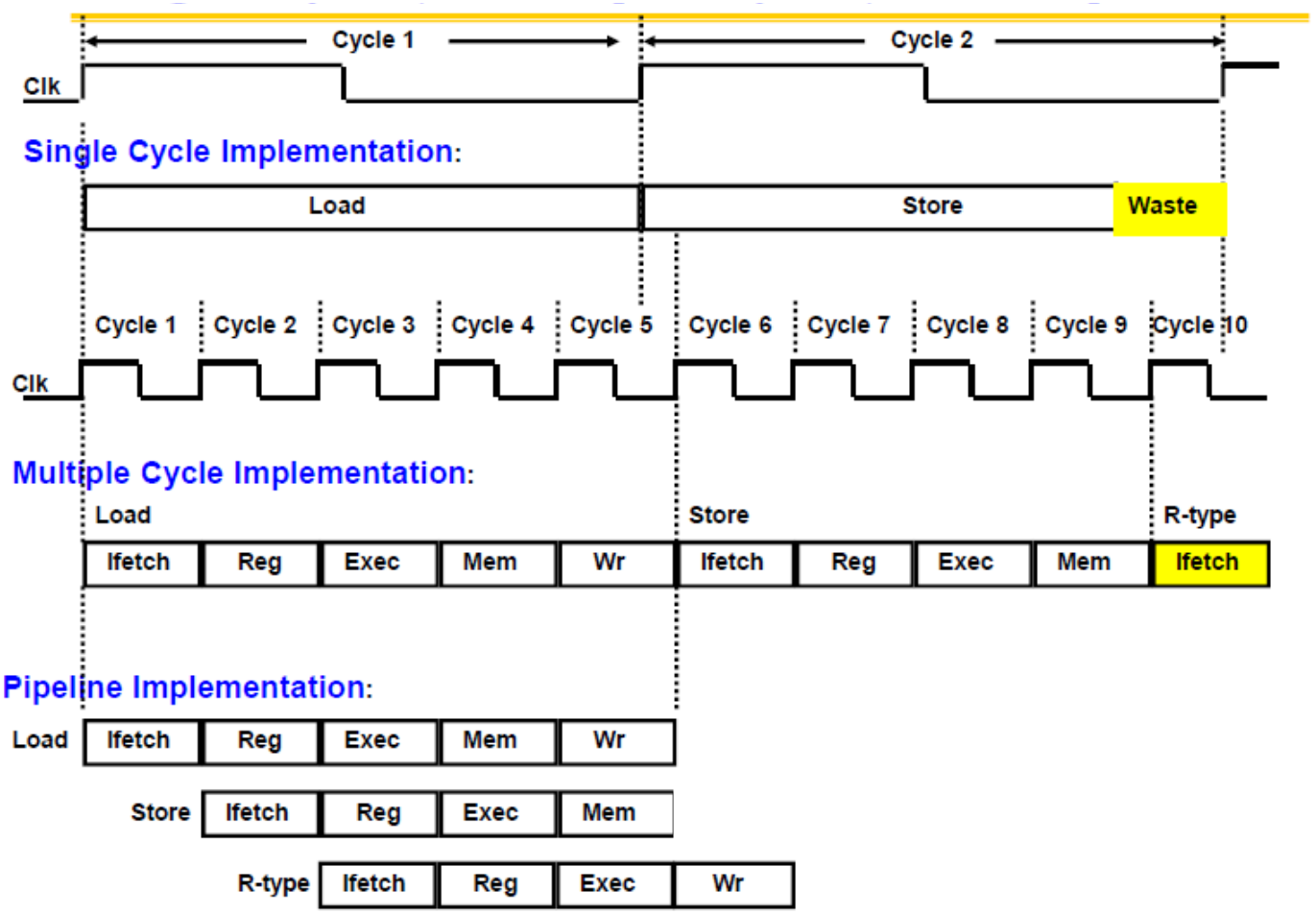


Pipeline Implementation:

Load	Ifetch	Reg	Exec	Mem	Wr
Store	Ifetch	Reg	Exec	Mem	
R-type	Ifetch	Reg	Exec	Wr	

CPU指令周期

CPU流水线



CPU指令周期

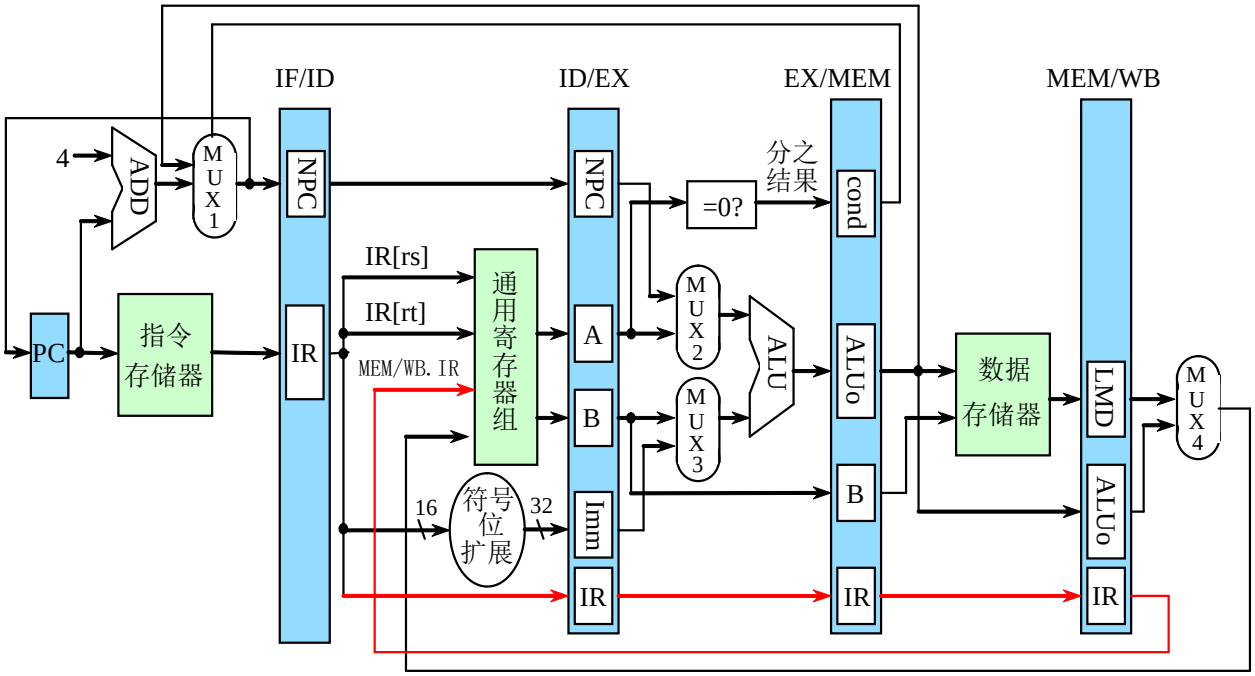
从不同的角度和观点，可以把流水线分成多种不同种类：

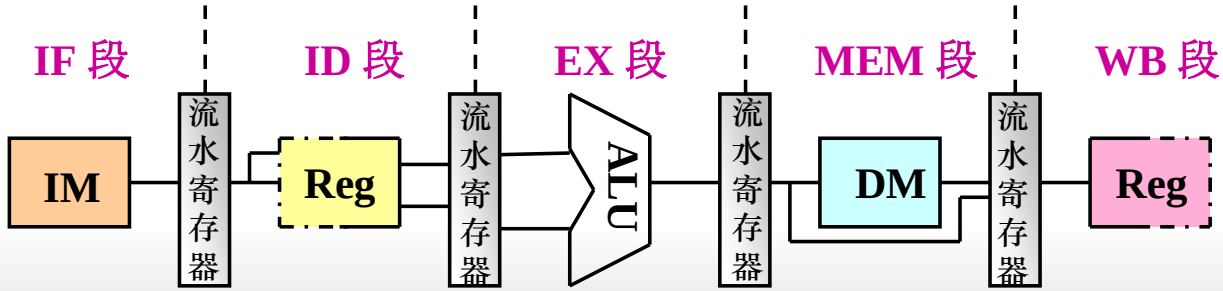
- 部件级流水线（运算操作流水线）：把处理机中的部件分段，再把这些分段相互连接起来，使得各种类型的运算操作能够按流水方式进行。
- 处理机级流水线（指令流水线）：把指令的执行过程按照流水方式处理。把一条指令的执行过程分解为若干个子过程，每个子过程在独立的功能 部件中执行。
- 系统级流水线（宏流水线）：把多台处理机串行连接起来，对同一数据流进行处理，每个处理机完成整个任务中的一部分。

CPU指令周期

MIPS（Microprocessor without interlocked piped stages）

无内部互锁流水级的微处理器





1.Instruction Fetch

2.Deconde and Register Read

3.Execute

4.Memory

5. Write Back

提纲

- CPU时序分析
- CPU指令周期
- **计算机性能评价与提升**
- 运算方法与运算器

计算机性能评价与提升

计算机技术不断发展的驱动力及目标：

用最小化的资源/成本（时间&空间）

获取

最大化的性能/运算速度

计算机运算速度：

- 以每秒执行多少条指令或完成多少次浮点运算来表示
- 单位：MIPS（百万条指令/秒，Million Instructions Per Second）
  MFLOPS（百万次浮点运算/秒）
- 分别反映了计算机的定点运算能力和浮点运算能力

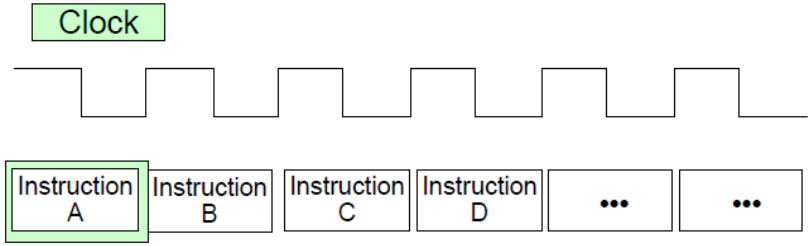
$$\text{MIPS} = \frac{\text{指令条数}}{\text{执行时间}} \times 10^{-6}$$

$$\text{MFLOPS} = \frac{\text{浮点运算次数}}{\text{执行时间}} \times 10^{-6}$$

计算机性能评价与提升

CPU性能公式

运行特定功能的程序，所用时间越短越好！



$$总CPU时间 = CPI \times IC/F$$

F：时钟频率（CPU主频），反映计算机实现技术、生产工艺和组织结构

CPI：执行每条指令平均时钟周期数，反映计算机体系结构（指令集等）

IC：执行程序所用的指令数，反映计算机体系结构（指令集）和编译技术

$$P = F/(CPI \times 10^6)$$

P：指令执行速度用于评测处理器性能，常表示为百万指令每秒，缩写为MIPS (Million Instructions Per Second)

计算机性能评价与提升

Intel IA-32 (x86)：家族系列

8008/8080

8086/8088

2/3/486

芯　片	推出日期	主频（MHz）	晶体管数量	可寻址容量	说　明
4004	4/1971	0.108	2300	640	第一片单片微处理器
8008	4/1972	0.108	3500	16KB	第一片 8 位微处理器
8080	4/1974	2	6000	64KB	第一片通用单片 CPU
8086	6/1978	5 ~ 10	29 000	1MB	第一片 16 位单片 CPU
8088	6/1979	5 ~ 8	29 000	1MB	用于 IBM PC
80286	2/1982	8 ~ 12	134 000	16MB	出现了内存保护
80386	10/1985	16 ~ 33	275 000	4GB	第一片 32 位 CPU
80486	4/1989	25 ~ 100	1.2M	4GB	嵌入 8KB 高速缓存
Pentium	3/1993	60 ~ 233	3.1M	4GB	双流水线；新型号有 MMX
Pentium Pro	3/1995	150 ~ 200	5.5M	4GB	嵌入两级高速缓存
Pentium II	5/1997	233 ~ 450	7.5M	4GB	Pentium Pro 加 MMX 指令
Pentium III	2/1999	650 ~ 1400	9.5M	4GB	增加了用于 3D 图形处理的 SSE 指令
Pentium 4	11/2000	1300 ~ 3800	42M	4GB	超线程；更多的 SSE 指令
Core Duo	1/2006	1600 ~ 3200	152M	2GB	单芯片上的双核
Core	7/2006	1200 ~ 3200	410M	64GB	64 位 4 核体系结构
Core i7	1/2011	1100 ~ 3300	1160M	24GB	集成的图形处理器

总CPU时间 = CPI×IC/F

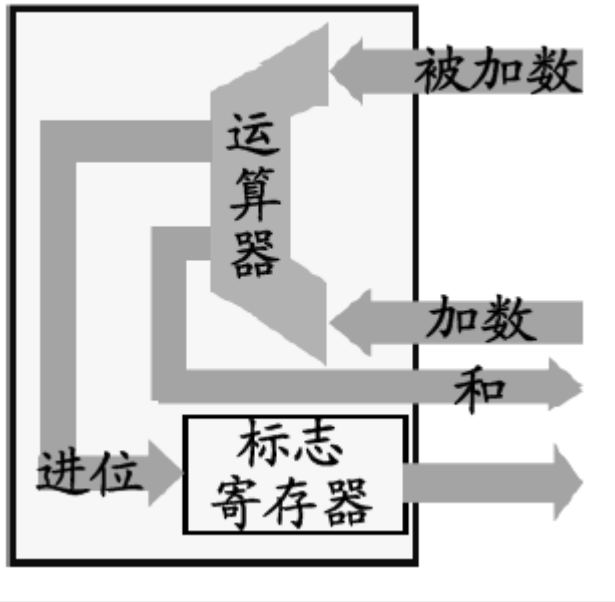
计算机性能评价与提升

CPU基本字长（8、16、32、64位）

总CPU时间 = CPI×IC/F

—— 直接参与运算的数据字的二进制位数

决定了CPU寄存器、ALU、数据总线等的位数，直接影响着硬件性能和造价



181 + 143 = 324（256 + 68）

```
    10110101    被加数8位
+   10001111    加数8位
进位 1 111111
-------------
    01000100    和8位
```

数值≤ 2^N-1

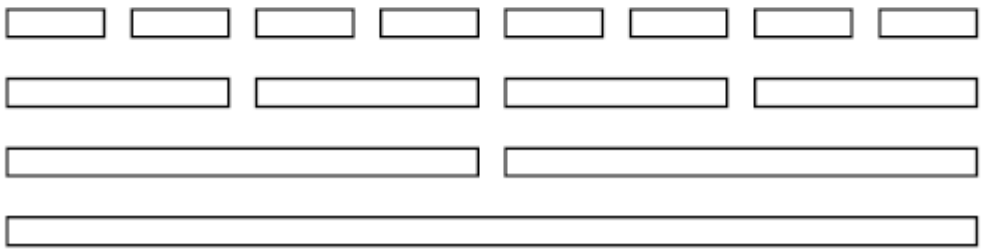
计算机性能评价与提升

CPU基本字长（8、16、32、64位）

总CPU时间 = CPI×IC/F

2,892,339,267 + 3,284,342,104 = 6,176,681,371

```
          1010 1100 0110 0101 1001 1000 0100 0011   被加数
+         1100 0011 1100 0011 0001 0101 0101 1000   加数
进位   1     1 1111 1     111     1        1
-------------------------------------------------
          0111 0000 0010 1000 1010 1101 1001 1011   和
```

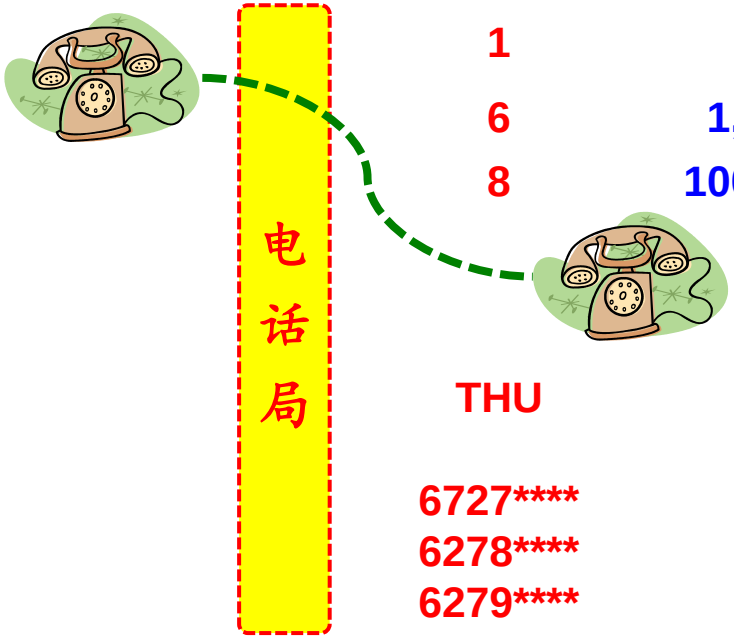


4位 8次
8位 4次
16位 2次
32位 1次

计算机性能评价与提升

CPU地址总线位数

地址总线什么用？

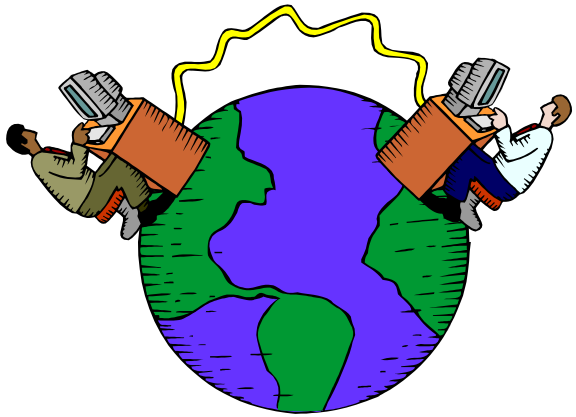


电话号码位数	电话用户数量
1	10
6	1,000,000
8	100,000,000

THU

6727****
6278****
6279****

30,000



IPv4地址　0.0.0.0 ～ 255.255.255.255

32位，地址容量2^{32} (约43亿)

IPv6地址　128位，地址容量2^{128}

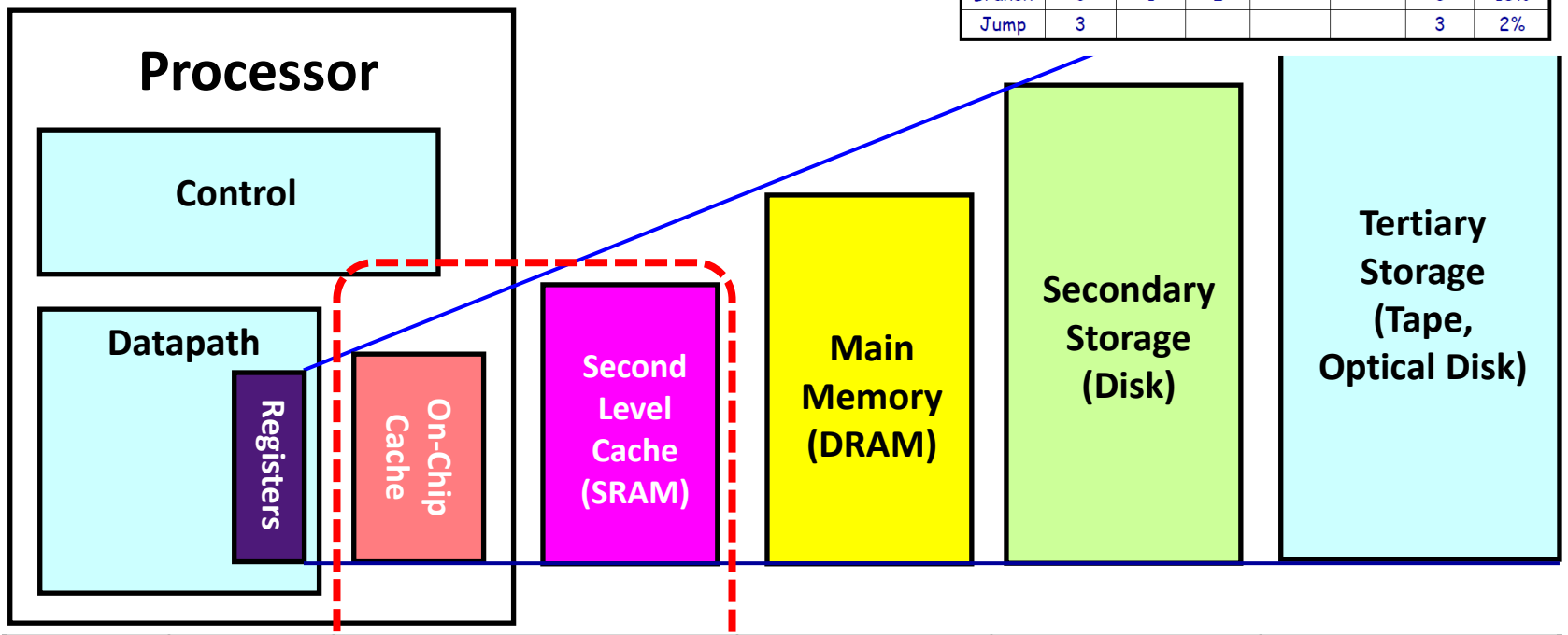
地址和数据总线位数需要一样吗？8G内存计算机，用32位CPU装32位Windows会有什么问题？

计算机性能评价与提升

总CPU时间 = CPI×IC/F

CPU缓存（Cache）

Inst. Type	Inst. Mem	Read Reg	ALU	Data Mem	Write Reg	Total /ns	Percent
R Type	3	1	2		1	7	44%
Lw	3	1	2	6	1	13	24%
Sw	3	1	2	6		12	12%
Branch	3	1	2			6	18%
Jump	3					3	2%



Speed	1ns	10ns	50-100ns	ms	s
Size	<100	KB-MB	MB-GB	GB-TB	TB

提纲

- CPU时序分析
- CPU指令周期
- 计算机性能评价与提升
- **运算方法与运算器**

CPU乘法运算器

总CPU时间 = CPI×IC/F

计算机中一般是用加法运算与移位运算来实现乘除法运算，可以采用以下3种方式：

- 采用计算机中的加减运算指令、移位指令以及控制指令组成循环程序，通过反复的加减操作，得到运算结果（速度太慢）
- 采用加法器、移位寄存器、计数器等逻辑器件组成乘除运算部件
- 采用专用的运算部件，提高乘除法运算的速度

CPU乘法运算器

人工乘法计算过程

两个正数X＝1011　Y＝1101

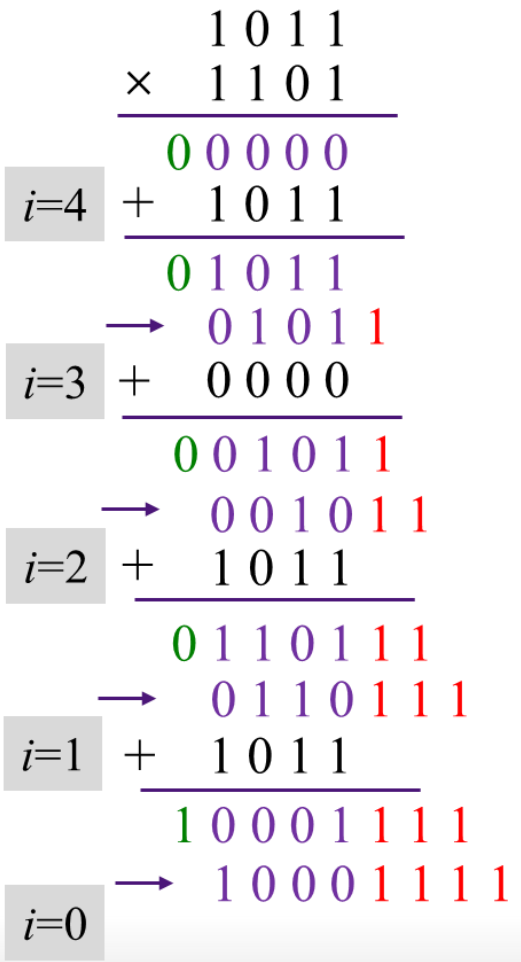
```
      1 0 1 1
×     1 1 0 1
-------------
      1 0 1 1      ←---- 位积P0
    0 0 0 0        ←---- 位积P1
  1 0 1 1          ←---- 位积P2
1 0 1 1            ←---- 位积P3
-------------
1 0 0 0 1 1 1 1
```

- **乘法器可以由加法实现**
- **存在的问题**
  - 部分积左移次数不同
  - 需要多输入全加器
  - 需要长度为2n的积寄存器

CPU乘法运算器

原码一位乘法

- 针对人工乘法计算存在问题的改进方法
  - 部分积左移次数不同
    - 右移部分积
  - 需要多输入全加器
    - 循环累加
  - 需要长度为2n的积寄存器
    - 从部分积和乘数寄存器取结果



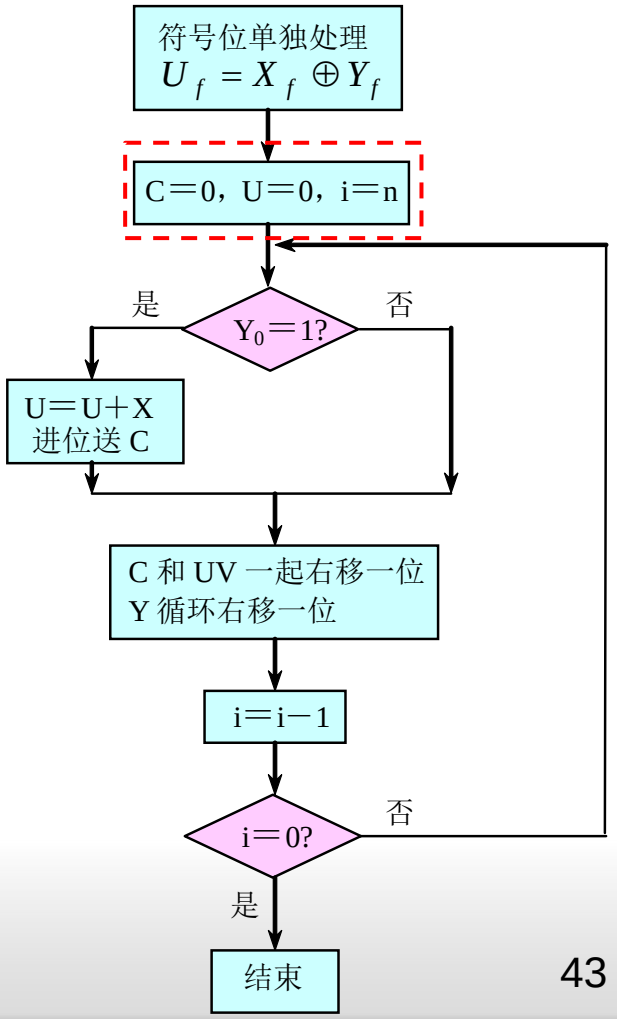
运算方法与运算器

CPU乘法运算器　　原码一位乘法

X：被乘数
Y：乘数

```
        1 0 1 1
      × 1 1 0 1
    ---------------
      0 0 0 0 0          CU
i=4 + 1 0 1 1            +X (Y0=1)
    ---------------
      0 1 0 1 1          CU
    → 0 1 0 1 1          UV
i=3 + 0 0 0 0            +0 (Y0=0)
    ---------------
      0 0 1 0 1 1        CUV
    → 0 0 1 0 1 1        UV
i=2 + 1 0 1 1            +X (Y0=1)
    ---------------
      0 1 1 0 1 1 1      CUV
    → 0 1 1 0 1 1 1      UV
i=1 + 1 0 1 1            +X (Y0=1)
    ---------------
      1 0 0 0 1 1 1 1    CUV
    → 1 0 0 0 1 1 1 1    UV
i=0
```

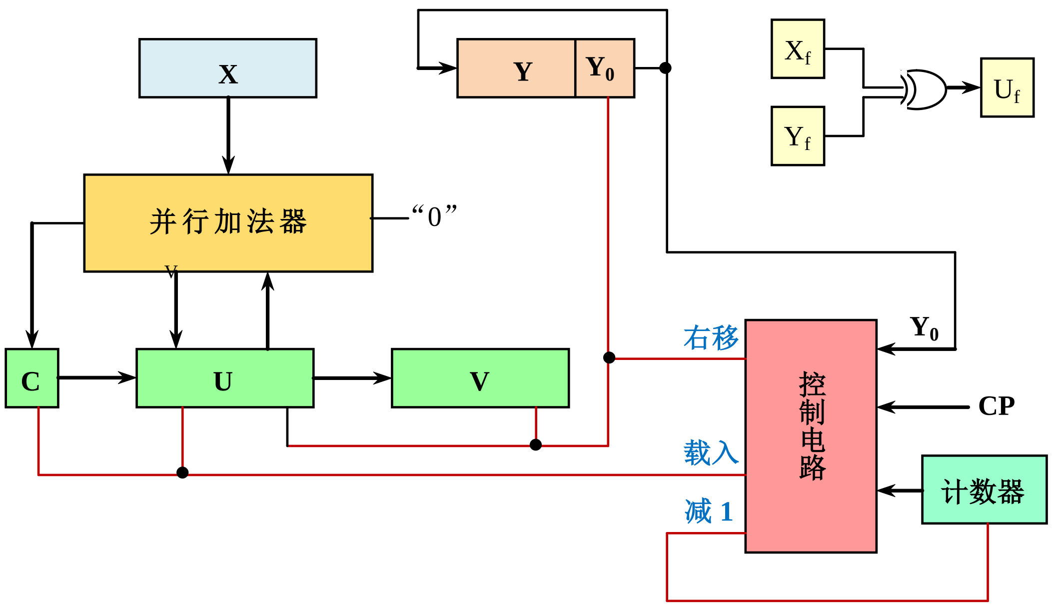
X：被乘数　Y: 乘数
Y0: 乘数右移后最低位
C：加法器进位输出　U V: 部分积



CPU乘法运算器

实现原码一位乘法的逻辑电路

	1 0 1 1	X：被乘数
	× 1 1 0 1	Y：乘数
	0 0 0 0 0	CU
i=4	+ 1 0 1 1	+X (Y_0=1)
	0 1 0 1 1	CU
	→ 0 1 0 1 1	UV
i=3	+ 0 0 0 0	+0 (Y_0=0)
	0 0 1 0 1 1	CUV
	→ 0 0 1 0 1 1	UV
i=2	+ 1 0 1 1	+X (Y_0=1)
	0 1 1 0 1 1 1	CUV
	→ 0 1 1 0 1 1 1	UV
i=1	+ 1 0 1 1	+X (Y_0=1)
	1 0 0 0 1 1 1 1	CUV
i=0	→ 1 0 0 0 1 1 1 1	UV



X：被乘数　Y：乘数　U、V：部分积　计数器：减 1 计数器

C：加法器的进位输出　X_f、Y_f、U_f：被乘数、乘数、乘积的符号位

右移：寄存器的右移控制信号　载入：寄存器的数据载入（写入）控制信号

减 1：计数器的减 1 控制信号

CPU乘法运算器

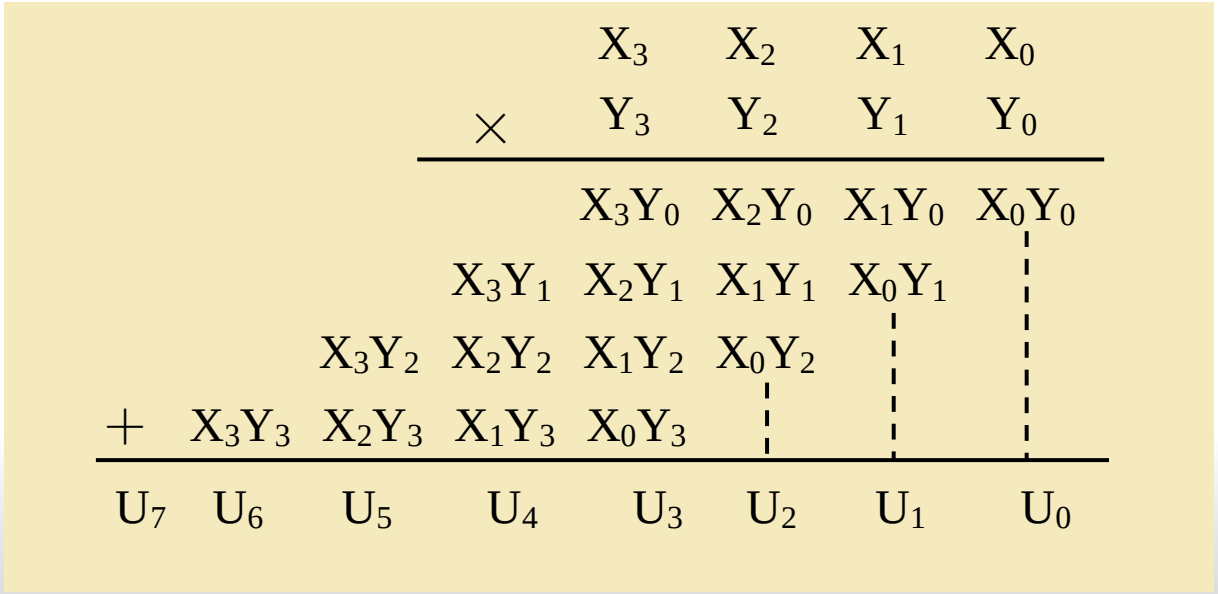
阵列乘法器

设有两个无符号的4位二进制定点数：

$$X=X_3X_2X_1X_0$$

$$Y=Y_3Y_2Y_1Y_0$$

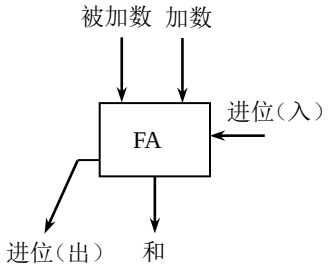
- 人工计算U＝X×Y的过程



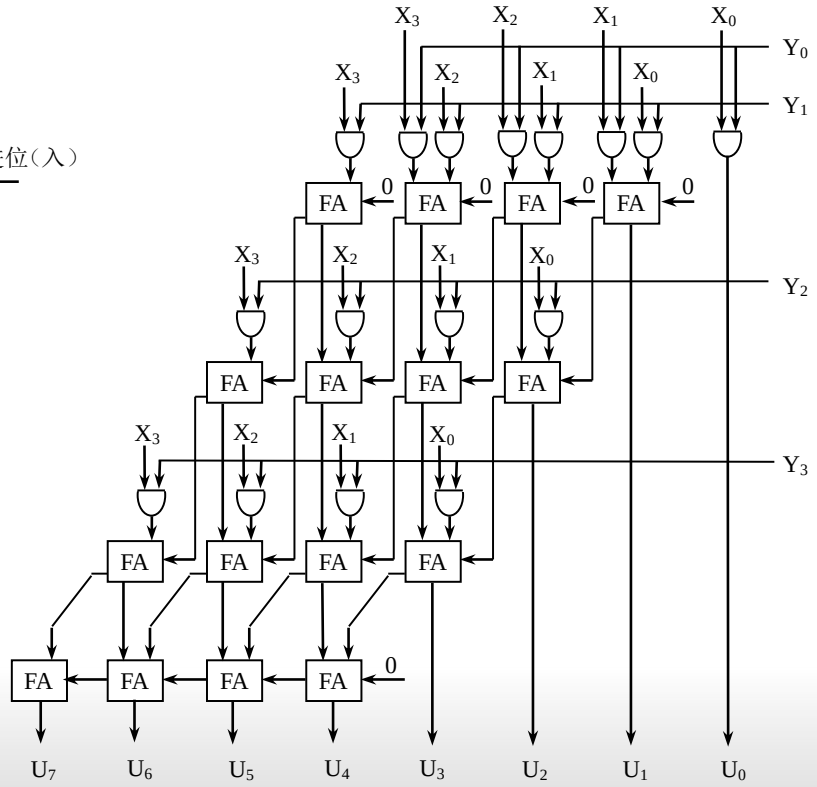
CPU乘法运算器

一个4×4位的阵列乘法器

阵列乘法器所用的加法器数量很多，但内部规则性强，运算速度快。



（a）全加器

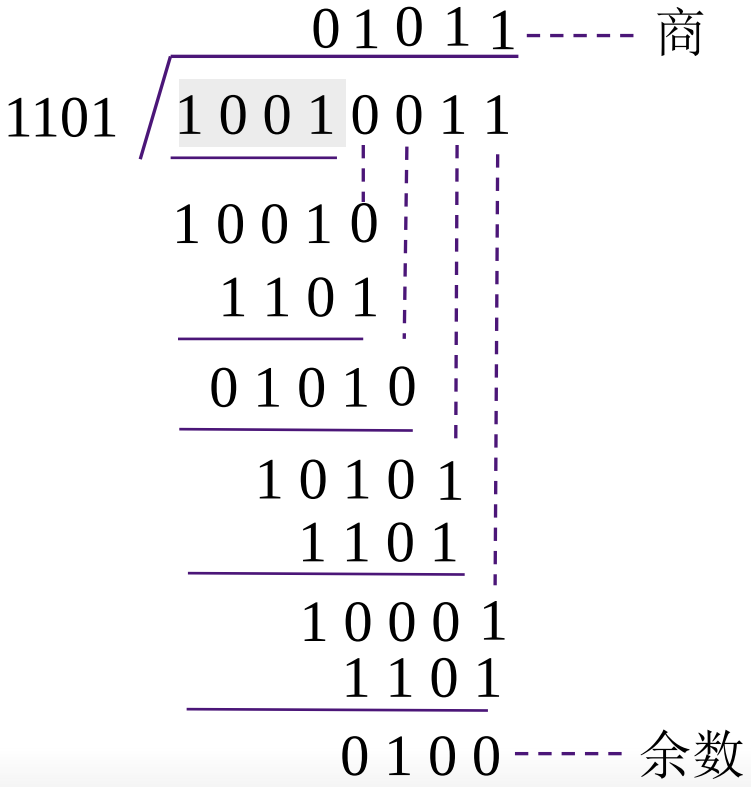


（b）4×4 位阵列乘法器

运算方法与运算器

CPU除法运算器

人工除法计算过程



- **除法器可以由加法实现**
- **存在的问题**
  - 计算机中通常是通过减法运算比较部分余数和除数大小；如果部分余数小于除数，需要恢复原值
  - 除数放置位置逐次向右错开一位，加法器规模变大

CPU除法运算器

原码恢复余数法

```
          0 1 0 1 1 -------商
1 1 0 1 ) 1 0 0 1 0 0 1 1
          0 0 0 0      ←---1001＜1101，商 0
            1 0 0 1 0  ←---取被除数的下一位 0
          1 0 0 1 0    ←---部分余数左移一位
            1 1 0 1    ←---10010≥1101，商 1，减除数
            0 1 0 1 0  ←---取被除数的下一位 0
          0 1 0 1 0    ←---部分余数左移一位
            0 0 0 0    ←---01010＜1101，商 0
            1 0 1 0 1  ←---取被除数的下一位 1
          1 0 1 0 1    ←---部分余数左移一位
            1 1 0 1    ←---10101≥1101，商 1，减除数
            1 0 0 0 1  ←---取被除数的下一位 1
          1 0 0 0 1    ←---部分余数左移一位
            1 1 0 1    ←---10001≥1101，商 1，减除数
            0 1 0 0  -------- 余数
```

（b）修改后

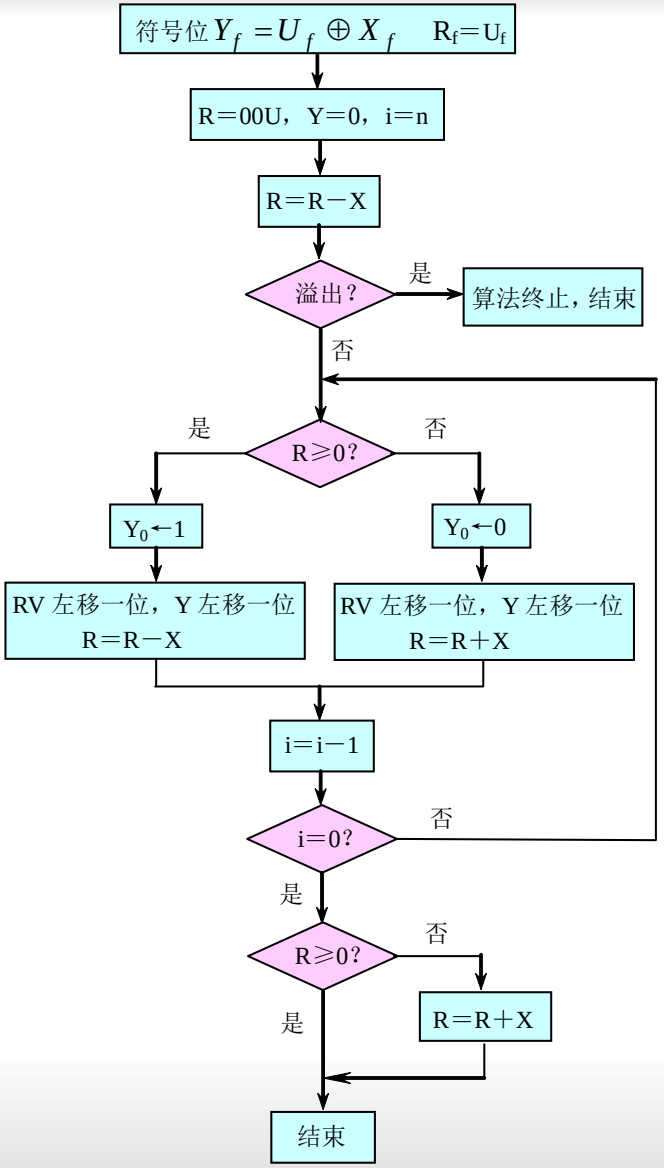
- 部分余数左移一位，R=R-X
- 如果R≥0表示够减，上商为1
- 如果R<0，表示不够减，上商为0，并恢复余数（R=R+X)
- 直至求得商的各位

CPU除法运算器

原码恢复余数法

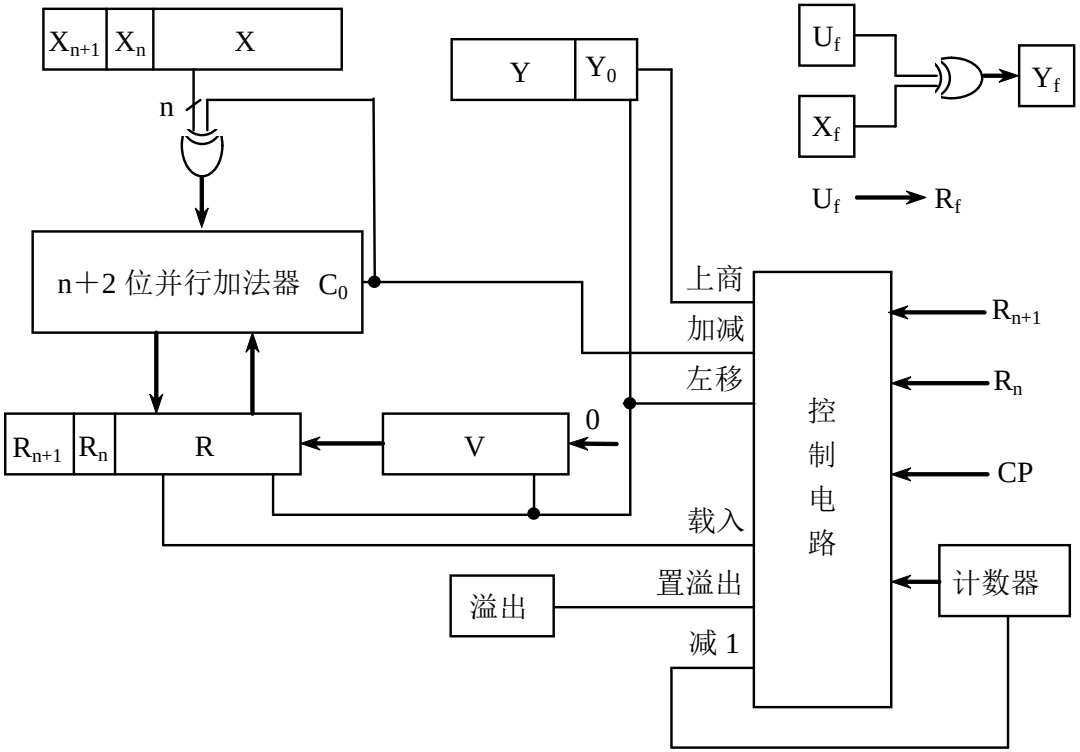
```
                 0 1 0 1 1 -------商
1 1 0 1 ┌ 1 0 0 1 0 0 1 1
        │ 0 0 0 0          ←----1001＜1101，商 0
        │   1 0 0 1 0      ←----取被除数的下一位 0
        │ 1 0 0 1 0        ←----部分余数左移一位
        │   1 1 0 1        ←----10010≥1101，商 1，减除数
        │   0 1 0 1 0      ←----取被除数的下一位 0
        │ 0 1 0 1 0        ←----部分余数左移一位
        │   0 0 0 0        ←----01010＜1101，商 0
        │   1 0 1 0 1      ←----取被除数的下一位 1
        │ 1 0 1 0 1        ←----部分余数左移一位
        │   1 1 0 1        ←----10101≥1101，商 1，减除数
        │   1 0 0 0 1      ←----取被除数的下一位 1
        │ 1 0 0 0 1        ←----部分余数左移一位
        │   1 1 0 1        ←----10001≥1101，商 1，减除数
        └   0 1 0 0  -------- 余数
```

（b）修改后



CPU除法运算器

原码恢复余数法的逻辑电路



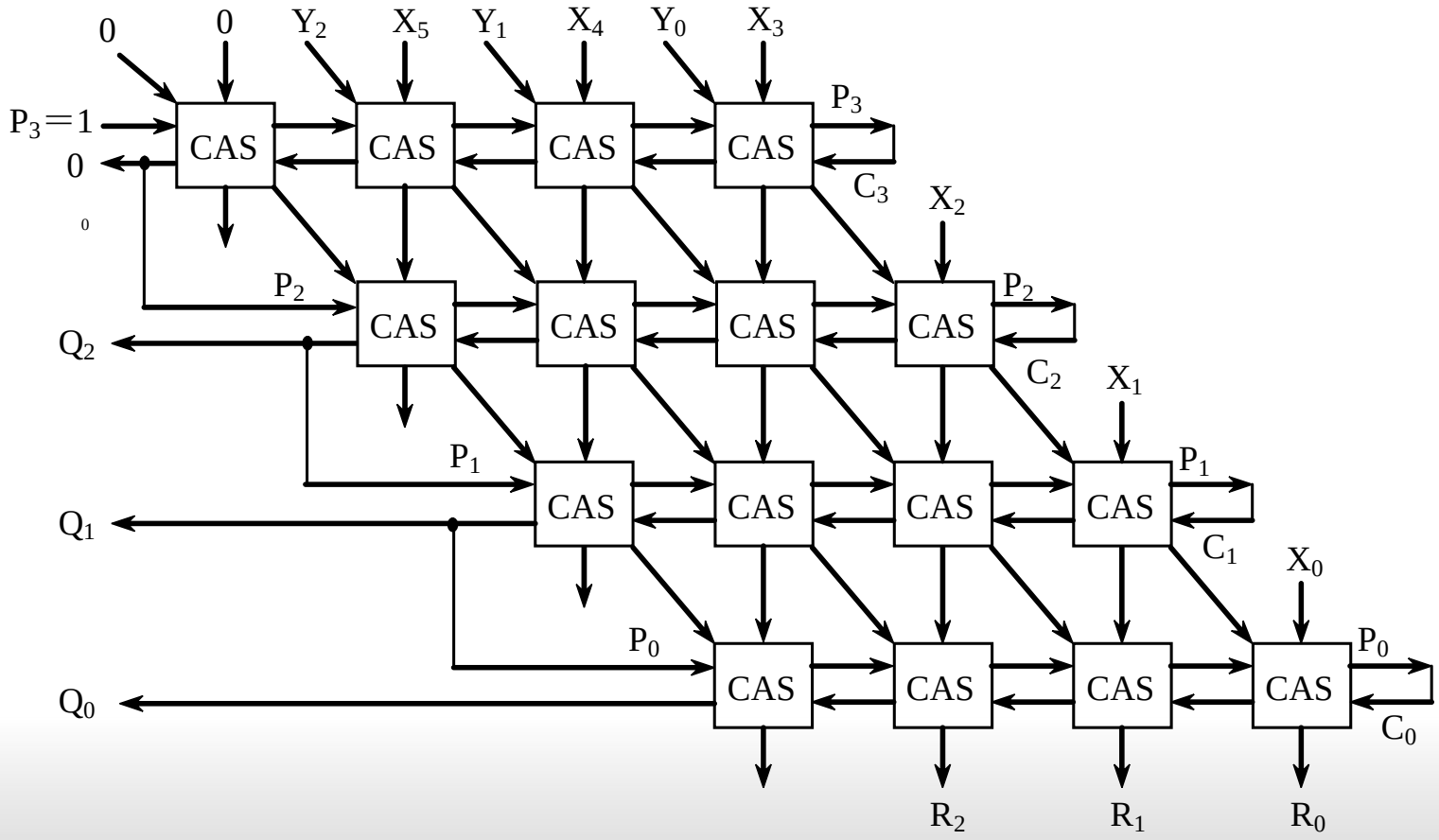
X：除数　　Y：商　　R、V：被除数或余数

U_f、X_f、Y_f、R_f：被除数、除数、商、余数的符号位

上商：控制将 Y_0 置 1 或 0　　加减：控制并行加法器进行加减法运算

CPU除法运算器

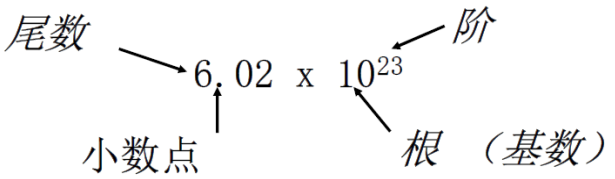
阵列除法器



运算方法与运算器

CPU浮点数运算器

- n 位能表示哪些数据?
  - 无符号整数： 0 to 2^n-1
  - 有符号整数： $-2^{(n-1)}$ to $2^{(n-1)}-1$



科学计数法

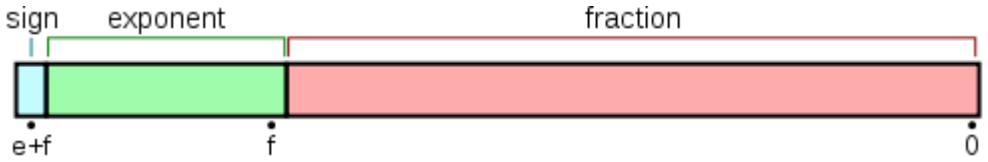
- 规格化：1.0×10^{-9}
- 非规格化：0.1×10^{-8}， 10.0×10^{-10}

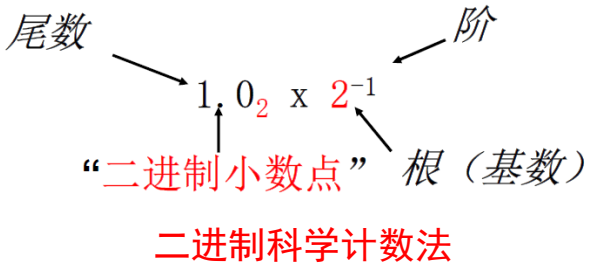
$X=M\times10^{E}$

- 其它数据呢?
  - 非常大整数? 如：一个世纪的秒数=3,155,760,00010 （3.1557610×10^{9}）
  - 非常小的数? 如：原子的直径=0.0000000110 （1.010×10^{-8}）
  - 有理数? 如：循环小数 2/3=0.666666666…
  - 无理数（无限不循环小数）? 如：√2=1.414…、e=3.718…、π=3.141…

运算方法与运算器

CPU浮点数运算器





- 浮点数的计算机内部表示（IEEE 754 浮点数标准）　$X = M \times 2^E$
  - 自1980年起被几乎所有计算机采纳
  - 对于规格化形式：$\pm 1.xxxxxxxxxx_2 * 2^{yyyy_2}$
    - 符号位（Sign）：1表示负数，0表示正数
    - 阶码（exponent）：表示y的偏正值（实际的指数大小+127）
    - 有效位：对于规格化小数，隐含高位为1；用尾数（fraction）表示其余位x
  - 0：不可能出现为1的位，在阶码中保留0给数0
  - 总长度：为字长的整数倍
    - 32位单精度数（C语言float类型），其中阶码8位、尾数23位
    - 64位双精度数（C语言double类型），其中阶码11位、尾数52位

运算方法与运算器

CPU浮点数运算器

单精度浮点数各种极值情况：

类别	正负号	实际指数	有偏移指数	指数域	尾数域	数值
零	0	-127	0	0000 0000	000 0000 0000 0000 0000 0000	0.0
负零	1	-127	0	0000 0000	000 0000 0000 0000 0000 0000	−0.0
1	0	0	127	0111 1111	000 0000 0000 0000 0000 0000	1.0
-1	1	0	127	0111 1111	000 0000 0000 0000 0000 0000	−1.0
最小的规约数	*	-126	1	0000 0001	000 0000 0000 0000 0000 0000	$\pm 2^{-126} \approx \pm 1.18\times 10^{-38}$
最大的规约数	*	127	254	1111 1110	111 1111 1111 1111 1111 1111	$\pm(2-2^{-23}) \times 2^{127} \approx \pm 3.4\times 10^{38}$
正无穷	0	128	255	1111 1111	000 0000 0000 0000 0000 0000	+∞
负无穷	1	128	255	1111 1111	000 0000 0000 0000 0000 0000	−∞
NaN	*	128	255	1111 1111	non zero	NaN

* 符号位可以为0或1 .

运算方法与运算器

CPU浮点数运算器

- **加减算法实现**
  - 对阶，求阶差E_X-E_Y，使阶码小的数的尾数右移阶差位，其阶码取大的阶码值
  - 对尾数进行加、减法，求得结果
  - 保持阶的值
  - 规格化
  - 舍入，可能再次规格化
  - 进行溢出检查（阶码）
- **乘除算法实现**
  - 对尾数进行乘、除法，求得结果
  - 阶码加、减：乘为E_X+E_Y，除为E_X-E_Y
  - 规格化
  - 舍入，可能再次规格化
  - 进行溢出检查（阶码）

$X=M\times 2^E$

例1：

$1.0110\times 2^3+1.1000\times 2^2$

对阶：

$1.0110\times 2^3+0.1100\times 2^3$

加法：

10.0010×2^3

规格化：

1.0001×2^4

例2：

$1.0001\times 2^3-1.1110\times 2^1$

对阶：

$1.0001\times 2^3-0.01111\times 2^3$

减法：

0.10011×2^3

规格化：

1.0011×2^2

CPU浮点数加减法运算

设有两个规格化浮点数X和Y，分别为：

X：$X_E X_M$

Y：$Y_E Y_M$

其中：X_E和Y_E：阶码

X_M和Y_M：尾数

1. 判0操作
   - 两个操作数X和Y中有为“0”的，则不需要进行运算，直接就能设置运算结果（为0），运算结束
   - 否则进入下一步

$X = M \times 2^E$

例1：

$1.0110 \times 2^3 + 1.1000 \times 2^2$

对阶：

$1.0110 \times 2^3 + 0.1100 \times 2^3$

加法：

10.0010×2^3

规格化：

1.0001×2^4

例2：

$1.0001 \times 2^3 - 1.1110 \times 2^1$

对阶：

$1.0001 \times 2^3 - 0.01111 \times 2^3$

减法：

0.10011×2^3

规格化：

1.0011×2^2

CPU浮点数加减法运算

2. 对阶

- 使小数点对齐
- 需要对其中的一个操作数进行变换，使两个操作数的阶码相等

- 求阶差$\triangle E = X_E - Y_E$
- 若$\triangle E > 0$，则表示X的阶码大于Y的阶码，需调整操作数Y。
  - 将Y的尾数Y_M右移；
  - 每右移一位，其阶码Y_E加1，直到两数的阶码相等为止。
- 若$\triangle E < 0$，则表示X的阶码小于Y的阶码，需调整操作数X，调整的方法与上面的一样。

$X = M \times 2^E$

例1：

$1.0110 \times 2^3 + 1.1000 \times 2^2$

对阶：

$1.0110 \times 2^3 + 0.1100 \times 2^3$

加法：

10.0010×2^3

规格化：

1.0001×2^4

例2：

$1.0001 \times 2^3 - 1.1110 \times 2^1$

对阶：

$1.0001 \times 2^3 - 0.01111 \times 2^3$

减法：

0.10011×2^3

规格化：

1.0011×2^2

CPU浮点数加减法运算

3. 尾数加/减

- 将两数的尾数X_M和Y_M，按照相应的定点加减运算规则进行加减法运算，得到运算结果尾数

4. 结果规格化并判溢出

- 若得到的运算结果的绝对值大于1，则需要右规
  - 将该结果右移一位，相应的阶码加1
  - 右规最多只有一位
- 若得到的运算结果的绝对值小于1，则需要左规
  - 将该结果左移，每左移一位，相应阶码减1
  - 直到运算结果的绝对值大于等于1/2为止
- 在规格化时，阶码每次加1或减1以后，都要判断阶码是否超出所能表示的范围。
  - 若阶码上溢，即阶码大于可能表示的最大正数，则置溢出标志，或者将结果当作+∞或-∞处理。
  - 若阶码下溢，即阶码小于可能表示的最小负数，这时，可以置溢出标志，也可以将结果当作0处理。

$X=M\times 2^E$

例1：

$1.0110\times 2^3+1.1000\times 2^2$

对阶：

$1.0110\times 2^3+0.1100\times 2^3$

加法：

10.0010×2^3

规格化：

1.0001×2^4

例2：

$1.0001\times 2^3-1.1110\times 2^1$

对阶：

$1.0001\times 2^3-0.01111\times 2^3$

减法：

0.10011×2^3

规格化：

1.0011×2^2

CPU浮点数加减法运算

5. 舍入处理

- 常用的舍入方法有以下几种：
  - **0舍1入法**
  - **截断法**
  - **朝＋∞舍入法**
  - **朝－∞舍入法**

- 几个数值在不同舍入方法下的舍入结果

数值	0舍1入法	截断法	朝＋∞舍入法	朝－∞舍入法
.0110 1001	.0111	.0110	.0111	.0110
－.0110 1001	－.0111	－.0110	－.0110	－.0111
.1000 0111	.1000	.1000	.1001	.1000
－.1000 0111	－.1000	－.1000	－.1000	－.1001
.1000 0000	.1000	.1000	.1000	.1000

$X = M \times 2^E$

例1：

$1.0110 \times 2^3 + 1.1000 \times 2^2$

对阶：

$1.0110 \times 2^3 + 0.1100 \times 2^3$

加法：

10.0010×2^3

规格化：

1.0001×2^4

例2：

$1.0001 \times 2^3 - 1.1110 \times 2^1$

对阶：

$1.0001 \times 2^3 - 0.01111 \times 2^3$

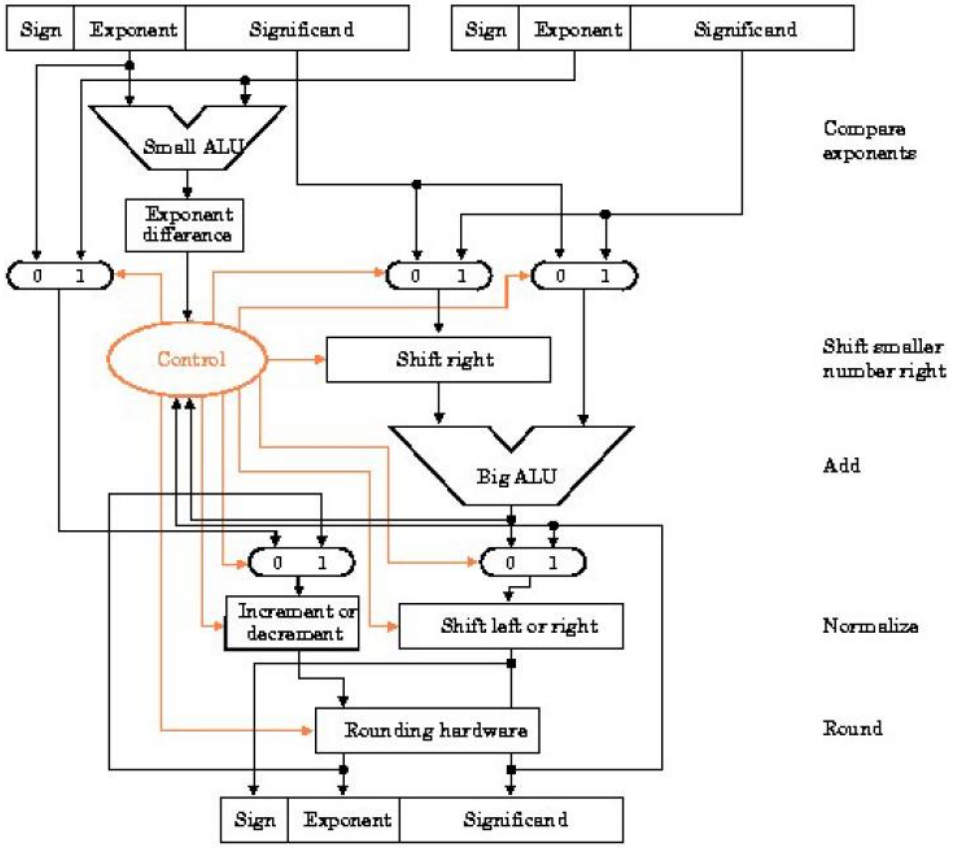
减法：

0.10011×2^3

规格化：

1.0011×2^2

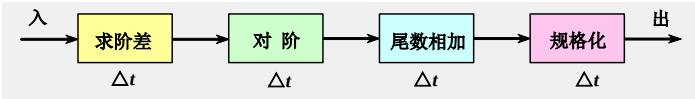
CPU浮点数运算器

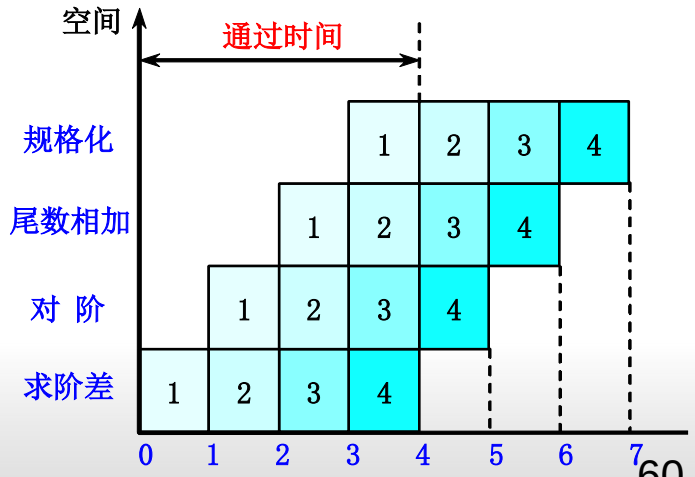


ARM支持的附加功能单元：VFP浮点计算引擎

浮点加法流水线

- 把流水线技术应用于运算的执行过程，就形成了运算操作流水线，也称为部件级流水线。
- 把浮点加法的全过程分解为求阶差、对阶、尾数　相加、规格化四个子过程，理想情况下速度提高3倍

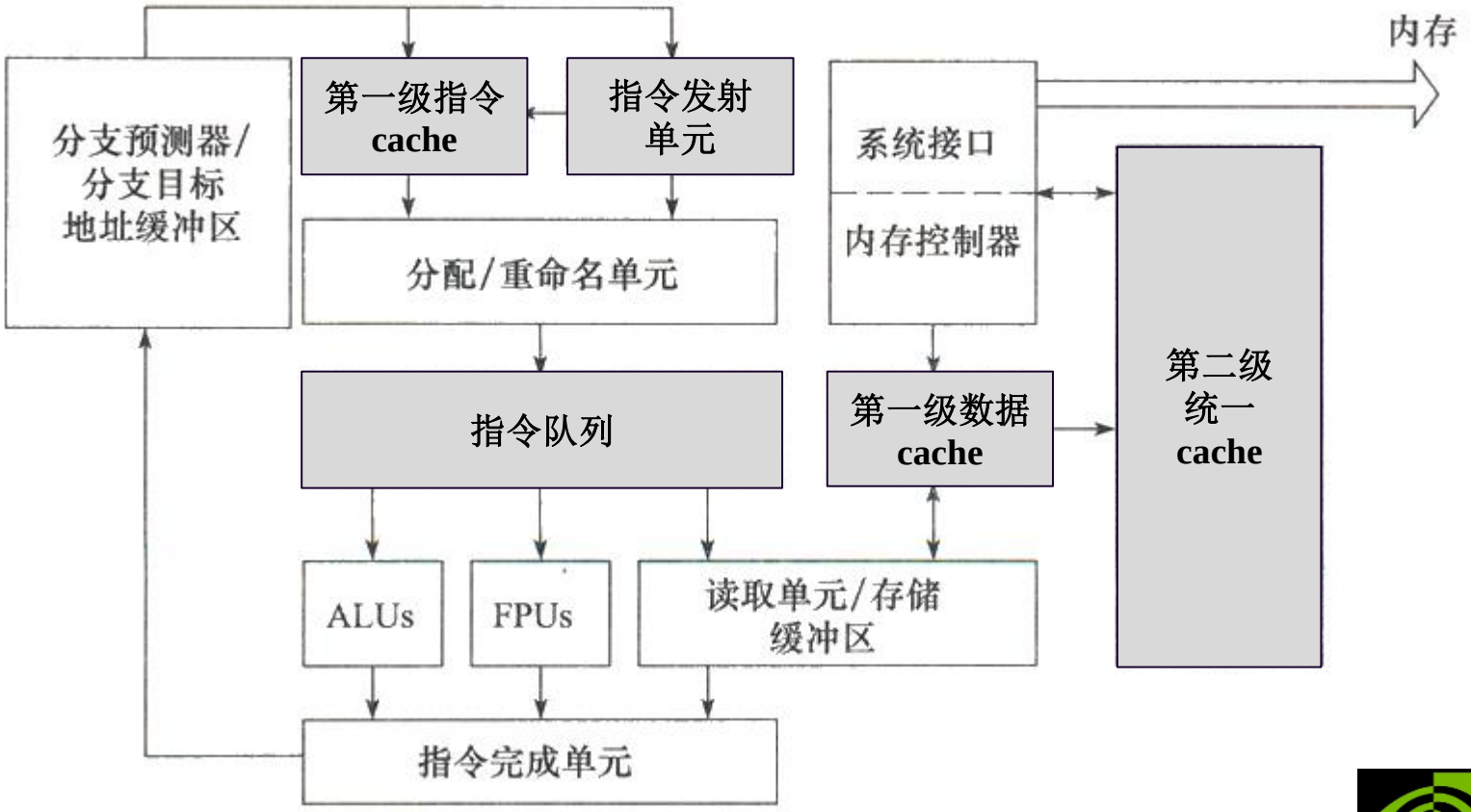




运算方法与运算器

ARM：微体系结构（TI OMAP4430）

总CPU时间 = CPI×IC/F



ARM支持的附加功能单元：VFP浮点计算引擎、NEON整数SIMD向量计算单元

课后作业：

复习：

《计算机组成原理》第1.4章节《计算机的性能指标》、第2.3.2章节《数的浮点表示》、第10章《运算方法与运算器》

预习：

《计算机组成原理》第7.6章节《指令系统的发展和改进》、第8.8章节《经典微处理器》

思考题：

《计算机组成原理》 P225页习题 8.5、8.8、8.10、8.11题，P271页习题10.12题

谢 谢

请不要将课件上传到公共网络平台上~~