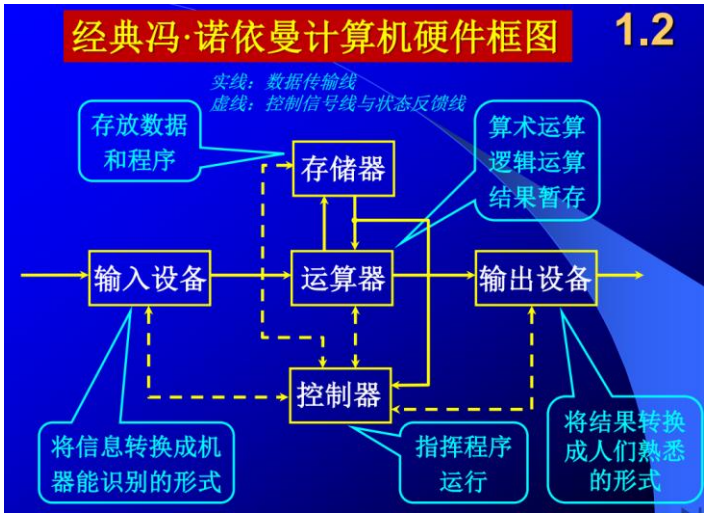
第一章 计算机系统概论

冯诺依曼体系结构



五个基本部件：**运算器、控制器、存储器、输入设备和输出设备**

存储器存放数和指令，形式上两者没有区别，但计算机应能区分数据还是指令；

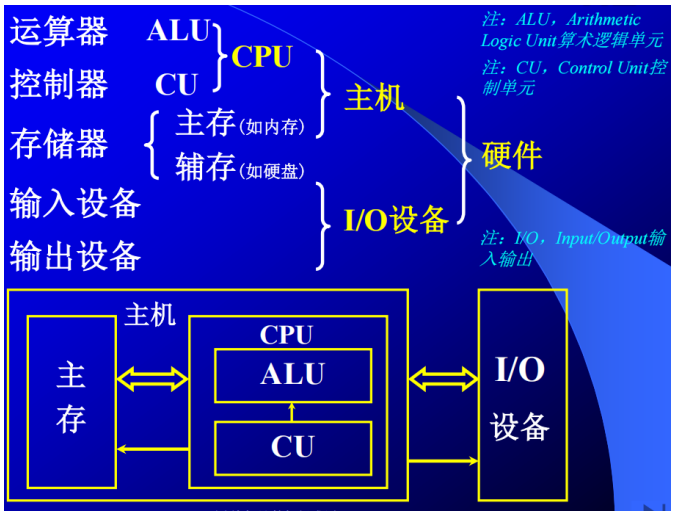
控制器应能自动执行指令；

采用“存储程序”工作方式，存储与计算分离

哈佛结构

程序指令存储与数据存储分开的结构

现代计算机硬件框图



存储器的基本组成

MAR: Memory Address Register

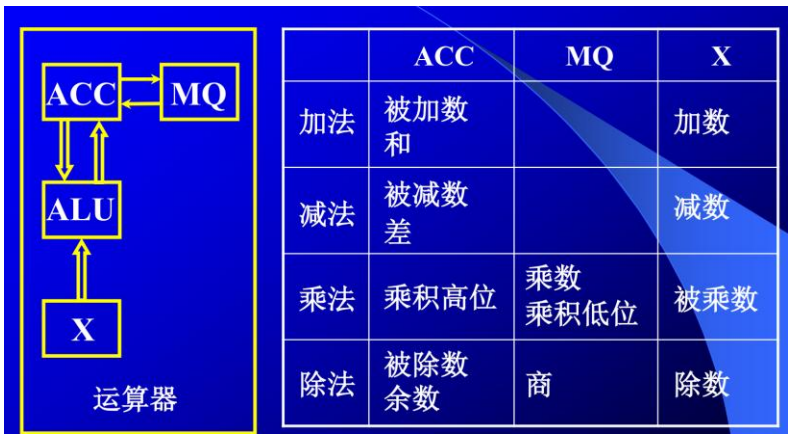
反映存储单元的个数，设地址位数为 m，则存储单元个数为 2^m 个

MDR: Memory Data Register

反映存储字长，设存储字长为 w

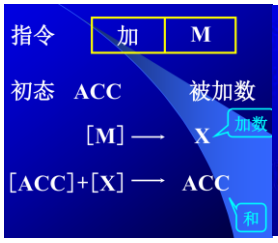
总存储容量=$2^m \times w$ bit

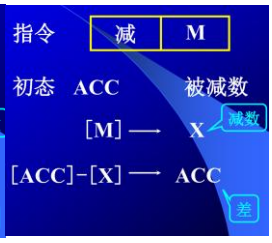
运算器的基本组成

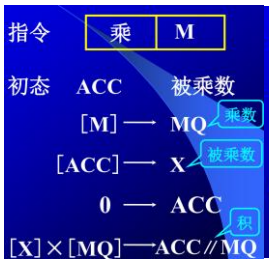


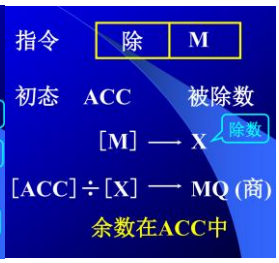
	ACC	MQ	X
加法	被加数 和		加数
减法	被减数 差		减数
乘法	乘积高位	乘数 乘积低位	被乘数
除法	被除数 余数	商	除数

计算机的操作过程

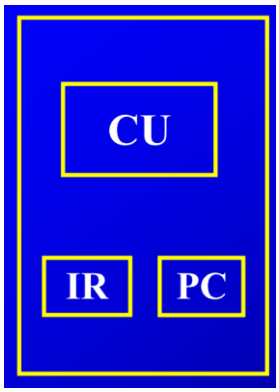




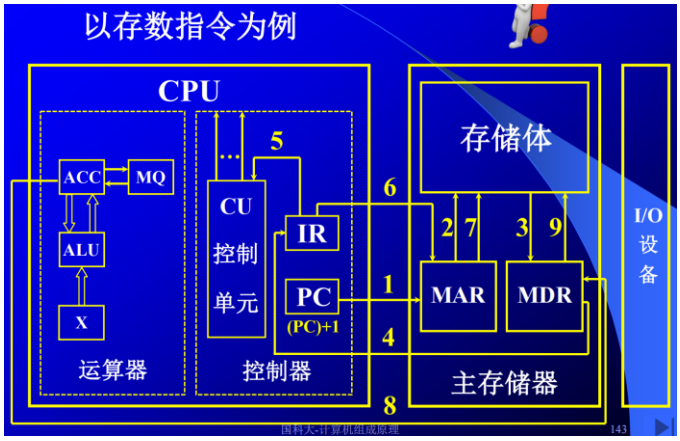




控制器的基本组成



*主机完成一条指令的过程



PC 此刻存储的地址,不一定是 0

存数

① PC → MAR
② 读
③ [o] → MDR
④ MDR → IR
⑤ IR → CU
⑥ Addr(IR) → MAR
⑦ 读mem (找到 Addr(mem))
⑧ ACC → MDR
⑨ MDR → Addr(mem)
⑩ PC+1

e.g 取数指令 即 Addr

取址
① PC → MAR　当前要执行的指令地址
② 读存储体 (根据MAR)
③ [o] → MDR
④ MDR → IR

分析 ⑤ IR → CU

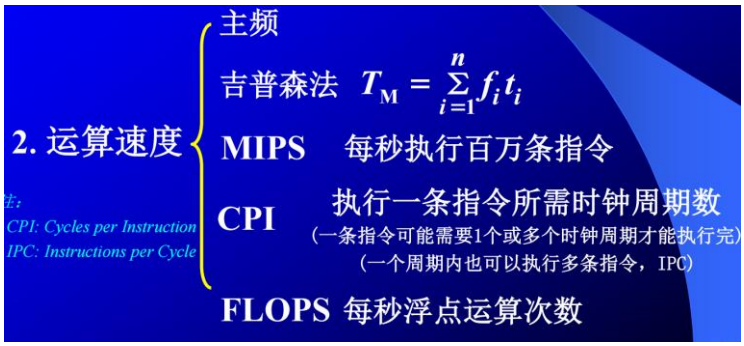
执行
⑥ Addr(IR) → MAR
⑦ 读 mem
⑧ [Addr] → MDR
⑨ MDR → ACC
⑩ PC+1　按字节编址下一条指令的长度
如: 32bit　PC <= PC +4

计算机硬件的主要技术指标

PPA 原则：Performance、Power、Area

（1）机器字长

（2）运算速度



（3）存储容量

主存容量+辅存容量

单位转换：1B = 2^3b　1KB = 2^{10} B　1MB = 2^{20} B　1GB = 2^{30} B

*性能评估指标

缩写/全称	解释	计算方式	影响因素
IC **Instruction Count**	一个程序的指令条数		程序、ISA 和编译器
CPI **Cycles Per Instruction**	**一个程序中：每个周期执行多少条指令** **其倒数为：每条指令要执行多少个周期**	CPI= $\frac{\text{Clock Cycles}}{\text{Instruction Count}}$	与 cpu 的硬件设计有关 与 ISA 无关
假定CPI_i、F_i是各指令CPI和在程序中的出现频率，则程序综合/平均CPI为: $\text{CPI}_{\text{avg}} = \sum_{i=1}^{n} CPI_i \times F_i$　where $F_i = \frac{C_i}{Instruction\_Count}$			
CPU Time	**一个程序的总运行时长**	CPU Time=IC×CPI×Clock Cycle Time(即 1/F)	
MIPS **Million Instructions Per Second**	**每秒钟运行几百万条指令**	MIPS=$\frac{IC}{CPU\ Time}\div 10^6$ 指令条数/总时长 =$\frac{F}{CPI}\div 10^6$ 每条指令所需周期×每秒有多少周期	

单靠 CPI 不能反映 CPU 性能！

假设有一种 CPU，执行每种指令都需要 1 个时钟周期，即单周期 CPU

那么单周期 CPU 的 CPI = 1，但单周期 CPU 的性能不够好！

第六章 计算机的运算方法

6.1 计算机中数的编码与表示

数值数据表示的三要素：

1.进位计数制　2.定、浮点表示　3.编码方式

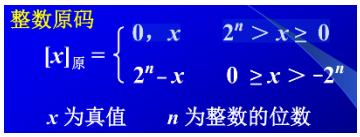
无符号（整）数

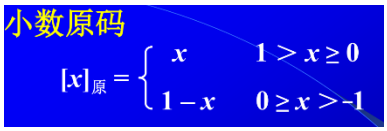
表示范围：0~2^n-1　8 位：0~255　16 位：0~65535

有符号数

真值 x（带有正负号）　机器数（符号数字化表示）

原码：实际上就是 0/1, |x|　符号，绝对值

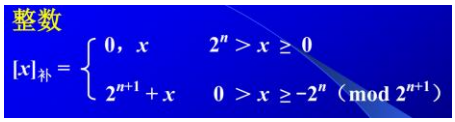


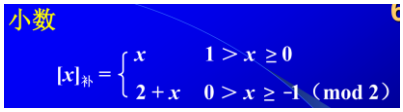


范围：[-2^n+1, 2^n-1]　[$-1+2^{-n}$, $1-2^{-n}$]

[+0.0000]原= 0.0000　[- 0.0000]原= 1.0000

补码：





快捷方式：当真值为负时，补码可用原码除符号位外每位取反，末位加 1 求得

设数值位有 n 位

范围：$[-2^n, 2^n-1]$　$[-1.0, 1-2^{-n}]$

[+0]补= [-0]补= 0, 00…0（n 个 0）

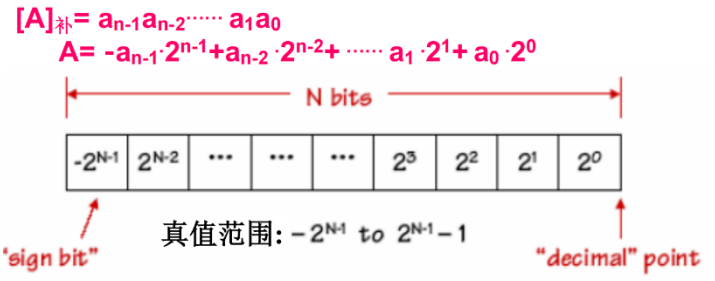
$[-2^n]$补=1, 00…0（n 个 0）

[-1.0]补=1.00…0（n 个 0）

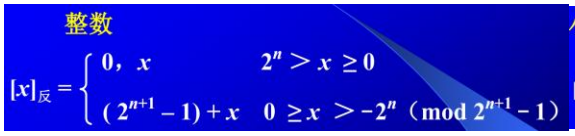
还原为原码：

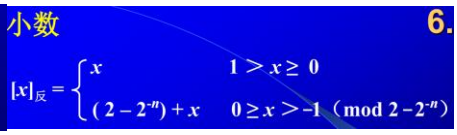
符号为 0，则为正数，数值部分同

符号为 1，则为负数，数值各位取反，末位加 1



反码：

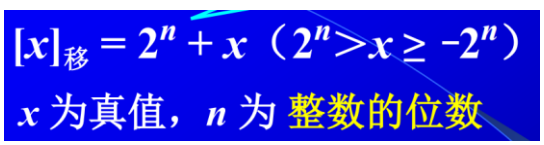


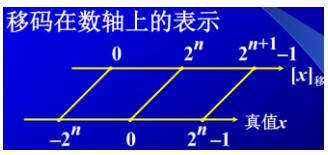


范围：$[-2^n+1, 2^n-1]$　$[-1+2^{-n}, 1-2^{-n}]$

[+0]反=0,0000　[-0]反= 1,1111

移码：





注意 x 是真值，而不是补码

解决了补码不便于比大小的不足（消除了符号位的干扰）

与补码仅差一个符号位，且符号位相反：说明**补码比大小时，符号位取反，直接按无符号数比较数值即可**

也可以说[X]移最高位用 1 表示正号，用 0 表示负号

[+0]移 = [-0]移 = 1,0000

[-10000]移 = 2^4-10000=0,0000 最小真值的移码为全 0

移码主要用来表示浮点数的阶码(指数)

便于浮点数加减运算时的对阶操作(判断阶码大小)

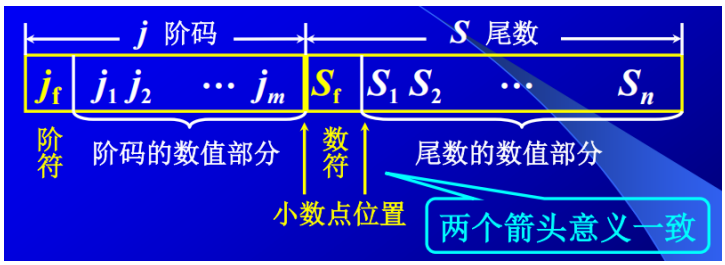
- **为什么用补码表示有符号整数？**
  - **补码运算系统是模运算系统，加、减运算统一**
  - **数0的表示唯一，方便使用**
  - **比原码和反码多表示一个最小负数**
  - **与移码相比，其符号位和真值的符号对应关系清楚**

6.2 数的定点表示和浮点表示

定点表示

定点机表示范围	小数定点机	整数定点机
原码	$-(1-2^{-n})\sim+(1-2^{-n})$	$-(2^n-1)\sim+(2^n-1)$
补码	$-1.0\sim+(1-2^{-n})$	$-2^n\sim+(2^n-1)$
反码	$-(1-2^{-n})\sim+(1-2^{-n})$	$-(2^n-1)\sim+(2^n-1)$

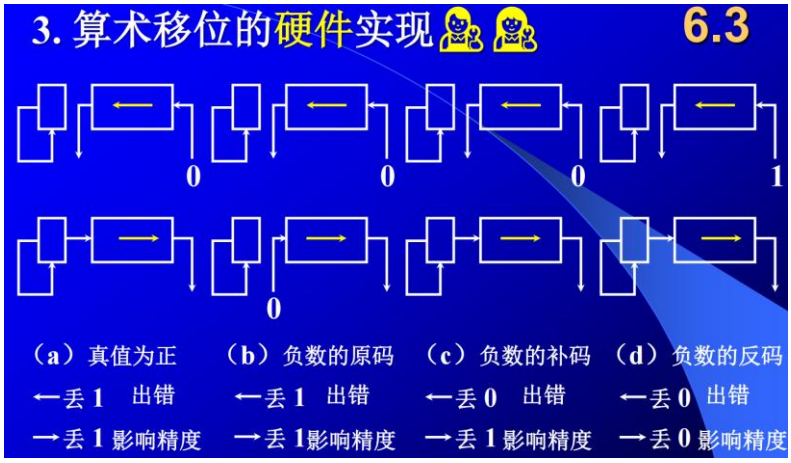
浮点表示

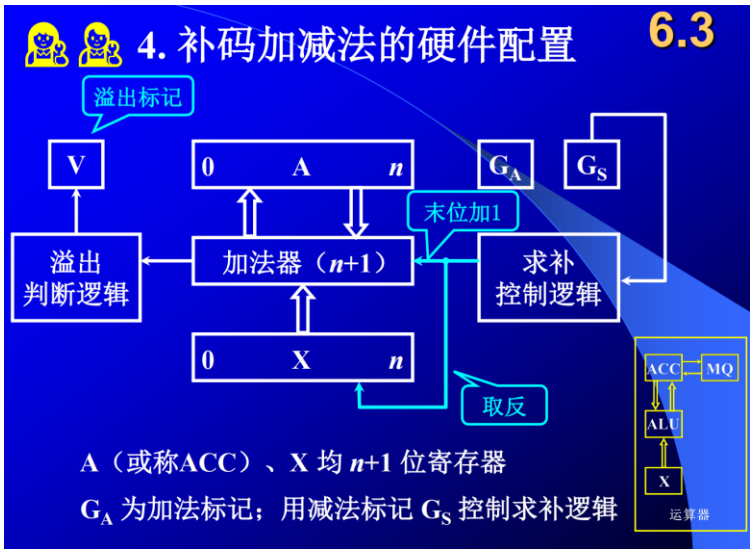


j_f 和 m 共同表示小数点的实际位置

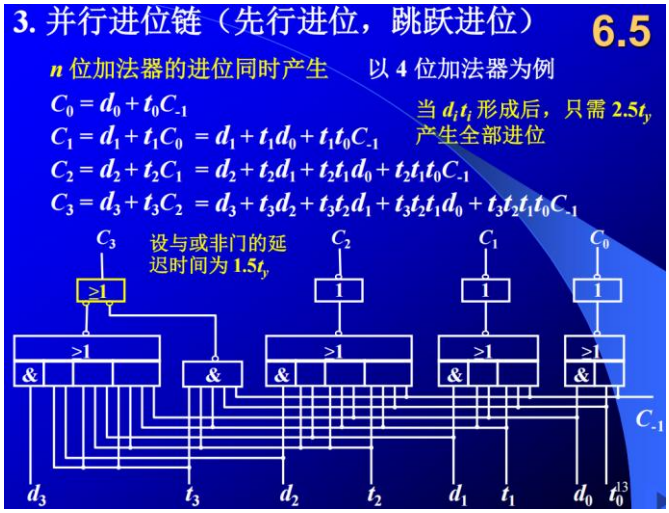
6.2 定点运算

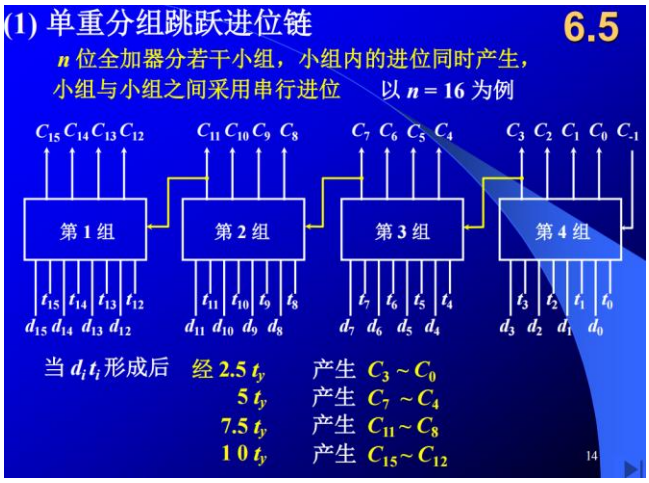
各种运算流程见笔记

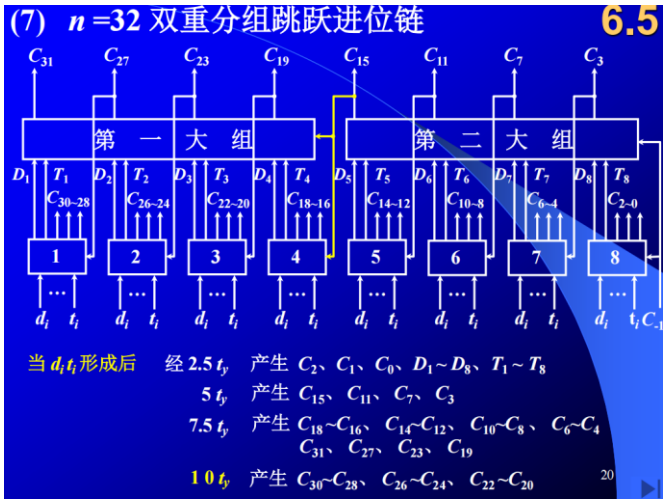




快速进位链







并行进位的延时为 2.5ty

分组后的延时取决于没组输出的位数 m，t=m×2.5ty

若四位数字分成一组，则 t=10ty

单周期处理器

Processor (CPU):

• Datapath:　指令执行过程中，数据所经过的路径，包括路径中的部件。

是指令执行的部件

组合元件和存储元件通过总线或分散方式连接而成的进行数据存储、处理和传送的路径。

• Control: 对指令进行译码，生成指令对应的控制信号，控制数据通路的动作。是指令的控制部件，对执行部件发出控制信号

CPUTime(ET) = IC× CPI × Cycle Time

MIPS

•无内部互锁流水级的微处理器

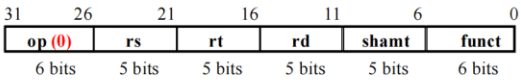
（Microprocessor without Interlocked Piped Stages）

• All instructions 32-bits long

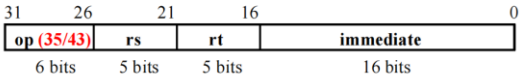
• 3 Formats:

The MIPS Subset

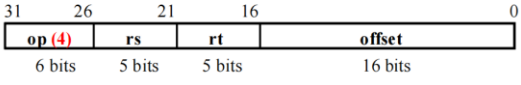
R-Type
- add rd, rs, rt
- sub, and, or, slt



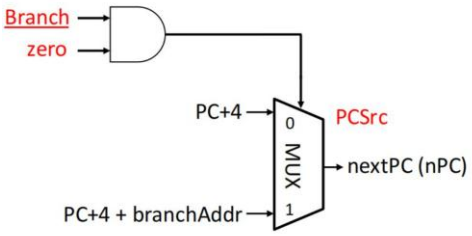
LOAD and STORE
- lw rt, rs, imm16
- sw rt, rs, imm16



BRANCH:
- beq rs, rt, imm16



- Revisit “next address logic”:
  - PCSrc should be 1 if branch, 0 otherwise



Branch	zero	PCSrc
0	0	0
0	1	0
1	0	0
1	1	1

How does this change if we add `bne`?

Branch	BranchType	zero	PCSrc
0	X	X	0
1	0 (beq)	0	0
1	0 (beq)	1	1
1	1 (bne)	0	1
1	1 (bne)	1	0

(3) 硬件修改

- 增加逻辑门：将 `BranchType` 信号与 `zero` 的反相信号（NOT）结合，生成新的条件判断。
  - 例如：用 XOR 门 实现 `BranchType` 和 `zero` 的互斥判断（见下图示意）。

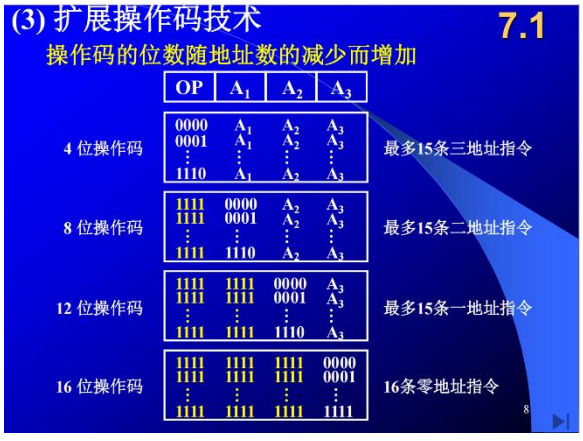
(BranchType ⊕ zero) ∧ Branch 即可

各类指令要求的时间长度为：

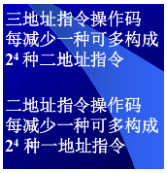
Instruction class	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-type	200	50	100	0	50	400 ps
Load word	200	50	100	200	50	600 ps
Store word	200	50	100	200		550 ps
Branch	200	50	100	0		350 ps
Jump	200					200 ps

第七章 指令系统

7.1.1 指令格式



变长的代价是前面每组会损失一个操作种类（整体来看损失更大），因为需要标识操作码的位数



例 7.1 假设指令字长为 16 位，操作数的地址码为 6 位，指令有零地址、一地址、二地址三种格式。

（1）设操作码固定，若零地址指令有 P 种，一地址指令有 Q 种，则二地址指令最多有几种？

（2）采用扩展操作码技术，若二地址指令有 X 种，零地址指令有 Y 种，则一地址指令最多有几种？

解：（1）根据操作数地址码为 6 位，则二地址指令中操作码的位数为 $16-6-6=4$。这 4 位操作码可有 $2^4=16$ 种操作。由于操作码固定，则除去了零地址指令 P 种，一地址指令 Q 种，剩下二地址指令最多有 $16-P-Q$ 种。

（2）采用扩展操作码技术，操作码位数可变，则二地址、一地址和零地址的操作码长度分别为 4 位、10 位和 16 位。可见二地址指令操作码每减少一种，就可多构成 2^6 种一地址指令操作码；一地址指令操作码每减少一种，就可多构成 2^6 种零地址指令操作码。

因二地址指令有 X 种，则一地址指令最多有 $(2^4-X)\times 2^6$ 种。设一地址指令有 M 种，则零地址指令最多有 $[(2^4-X)\times 2^6-M]\times 2^6$ 种。

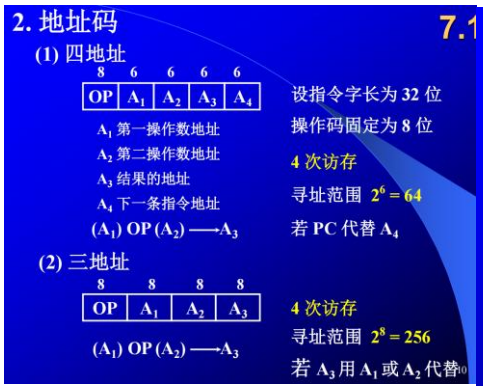
根据题中给出零地址指令有 Y 种，即

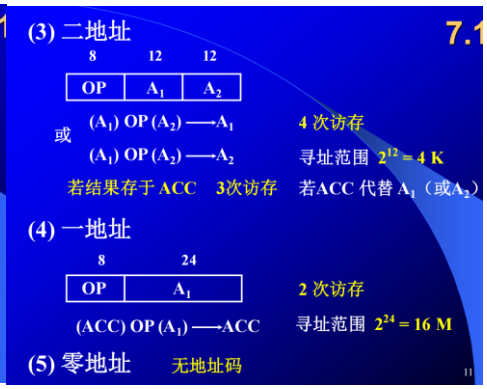
$$Y=[(2^4-X)\times 2^6-M]\times 2^6$$

则一地址指令

$$M=(2^4-X)\times 2^6-Y\times 2^{-6}$$

在设计操作码不固定的指令系统时，应尽量考虑安排指令使用频度（即指令在程序中出现的概率）高的指令占用短的操作码，对使用频度低的指令可占用较长的操作码，这样可以缩短经常使用的指令的译码时间。当然，考虑操作码长度时也应考虑地址码的要求。





可以用一些硬件资源（如 PC、ACC）代替指令字中的地址码字段：

- 可扩大指令操作数的寻址范围
- 可缩短指令字长
- 可减少访存次数

7.1.2 指令字长

取决于 **操作码的长度、操作数地址的长度、操作数地址的个数**

若指令字长固定，则指令字长=存储字长

若指令字长不固定，则按字节倍数变化

数 01FB H 按字节写作:

以高字节地址为字地址的存储方式：大端序：高位数字在低地址

第一个字节:01H 第二个字节:FBH

以低字节地址为字地址的存储方式：小端序：低位数字在低地址

第一个字节:FBH 第二个字节:01H

按边界对齐：要求数据的地址为相应的存储字边界地址

不按边界对齐：节省空间，增加访存次数

7.2 操作类型

1.数据传送

源	寄存器	寄存器	存储器	存储器
目的	寄存器	存储器	寄存器	存储器
例如	MOVE	STORE MOVE PUSH	LOAD MOVE POP	MOVE
置“1”，清“0”				

2. 算术逻辑操作

3. 移位操作：算术移位、逻辑移位、循环移位（带进位和不带进位）

4. 转移：(1) 无条件转移 (2) 条件转移（Branch）(3) 调用和返回 (4) 陷阱（Trap）与陷阱指令

5. 输入输出

输入：端口->寄存器　　输出：寄存器->端口

7.3 寻址方式

指令寻址

1. 顺序寻址：PC+1

2. 跳跃寻址：通过转移类指令实现。转移地址的生成方式有直接寻址和相对寻址。

数据寻址

形式地址：一般地址码字段都是形式地址，记作 A

真实地址：有寻址方式和形式地址共同确定，记作 EA

1. 立即寻址（立即数寻址）：不用寻址，操作数本身就在指令中。寻址特征：#

2. 直接寻址：EA=A，指令中的形式地址即为操作数的真实地址

    缺点：限制范围；必须修改 A 的值才能修改操作数的地址

3. 隐含寻址：（全部或部分）操作数地址隐含在操作码或某个寄存器中

    减少了一个地址字段，有利于缩短指令字长

4. 间接寻址：有效地址由形式地址间接提供 EA=(A)。

有一次间接寻址和多次间接寻址。**多次间接寻址时用存储字的首位来标识间接寻址是否结束。所以间接寻址的寻址范围为 $2^{存储字长-1}$**

执行阶段 2(或 n+1 次)访存。

5. 寄存器寻址：（操作数存在寄存器中）有效地址即为寄存器编号，EA=Ri

执行阶段不访存

6. 寄存器间接寻址：有效地址在寄存器中 EA =（Ri）

便于编制循环程序（方便回到跳转点）

7. 基址寻址：在程序的执行过程中 基址寄存器内容不变，形式地址 A 可变

（1）采用专用寄存器作基址寄存器：EA =（BR）+ A　BR 为基址寄存器

在程序的执行过程中 BR 内容不变，形式地址 A 可变

（2）采用通用寄存器作基址寄存器：由用户指定哪个通用寄存器作为基址寄存器，但基址寄存器的内容由操作系统确定

基址寻址有利于多道程序

8. 变址寻址：EA =（IX）+A，**IX 的内容由用户给定**

在程序的执行过程中 IX 内容可变，形式地址 A 不变

便于处理数组问题：A 设定为数组首地址，通过改变 IX 来访问任一元素

适合编制循环程序

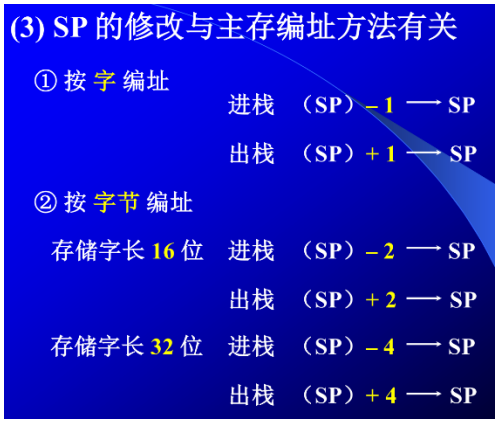
9. 相对寻址：EA =（PC）+ A ，A 是相对于当前指令的位移量（可正可负，补码）

注意（PC）是该寻址指令的下一条指令所在地址，即先更新 PC，再进行寻址

有利于编写浮动程序（数据与指令的相对位置不变）

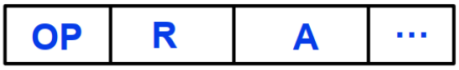
10. 堆栈寻址：操作数存放再堆栈中，由 SP 指向栈顶（存/取地址）

也可以视为一种隐含寻址。写进出栈 SP 的更新时，注意栈底地址是大于还是小于栈顶地址。



注意按字编址和按字节编址的区别！！！

假设：A=地址字段值，R=寄存器编号，
EA=有效地址， (X)=X中的内容



方式	算法	主要优点	主要缺点
立即 #	操作数=A	指令执行速度快	操作数幅值有限
直接	EA=A	有效地址计算简单	地址范围有限
间接 @	EA=(A)	有效地址范围大	多次存储器访问
寄存器	操作数=(R)	指令执行快，指令短	地址范围有限
寄间接	EA=(R)	地址范围大	额外存储器访问
偏移	EA=(R)+A	灵活	复杂
堆栈	EA=栈顶	指令短	应用有限

偏移方式：将直接方式和寄存器间接方式结合起来
有：基址BR / 变址IX / 相对(*)PC 三种
MIPS不区分基址还是变址，统一为偏移寻址方式

7.4 指令格式设计

考虑因素：**指令系统的兼容、操作类型、数据类型、指令格、寻址方式、寄存器个数**

7.5 RISC 技术

执行频度高的简单指令，因复杂指令的存在（如被迫增加指令长度、译码流程等），执行速度无法提高

RISC 的主要特征

1. **选用频度较高的一些简单指令，复杂指令的功能由简单指令来组合**
2. **指令 长度固定、指令格式种类少、寻址方式少**
3. **只有 LOAD / STORE 指令访存**
4. **CPU 中有多个 通用 寄存器**
5. **采用 流水技术 一个时钟周期 内完成一条指令**

6. 采用 组合逻辑 实现控制器

7. 采用 优化 的 编译 程序

注意：上述这些特点是纯 RISC 机的特点，不是所有的 RISC 机都有上述特点

CISC 的主要特征

1. 指令系统 复杂庞大，各种指令使用频度相差大

2. 指令 长度不固定、指令格式种类多、寻址方式多

3. 访存 指令 不受限制

4. 大多数指令需要 多个时钟周期 执行完毕

5. 采用 微程序 控制器

6. CPU 中设有 专用寄存器

7. 难以 用 优化编译 生成高效的目的代码

CISC 的比较

与 CISC 机相比，RISC 机的主要优点可归纳如下：

1. 充分利用 VLSI 芯片的面积

CISC 机的控制器大多采用微程序控制（详见第 10 章），其控制存储器在 CPU 芯片内所占的面积为 50% 以上（如 Motorola 公司的 MC68020 占 68%）。而 RISC 机控制器采用组合逻辑控制（详见第 10 章），其硬布线逻辑只占 CPU 芯片面积的 10% 左右。可见它可将空出的面积供其他功能部件用，例如用于增加大量的通用寄存器（如 Sun 微系统公司的 SPARC 有 100 多个通用寄存器），或将存储管理部件也集成到 CPU 芯片内（如 MIPS 公司的 R2000/R3000）。以上两种芯片的集成度分别小于 10 万个和 20 万个晶体管。

随着半导体工艺技术的提高，集成度可达 100 万至几百万个晶体管，此时无论是 CISC 还是 RISC 都将多个功能部件集成在一个芯片内。但此时 RISC 已占领了市场，尤其是在工作站领域占有明显的优势。

2. 提高计算机运算速度

RISC 机能提高运算速度，主要反映在以下 5 个方面。

① RISC 机的指令数、寻址方式和指令格式种类较少，而且指令的编码很有规律，因此 RISC 的指令译码比 CISC 的指令译码快。

② RISC 机内通用寄存器多，减少了访存次数，可加快运行速度。

③ RISC 机采用寄存器窗口重叠技术，程序嵌套时不必将寄存器内容保存到存储器中，故又提高了执行速度。

④ RISC 机采用组合逻辑控制，比采用微程序控制的 CISC 机的延迟小，缩短了 CPU 的周期。

⑤ RISC 机选用精简指令系统，适合于流水线工作，大多数指令在一个时钟周期内完成。

3. 便于设计，可降低成本，提高可靠性

RISC 机指令系统简单，故机器设计周期短，如美国加州伯克莱大学的 RISC Ⅰ机从设计到芯片试制成功只用了十几个月，而 Intel 80386 处理器（CISC）的开发花了三年半时间。

RISC 机逻辑简单，设计出错可能性小，有错时也容易发现，可靠性高。

4. 有效支持高级语言程序

RISC 机靠优化编译来更有效地支持高级语言程序。由于 RISC 指令少，寻址方式少，使编译程序容易选择更有效的指令和寻址方式，而且由于 RISC 机的通用寄存器多，可尽量安排寄存器