Principle of Computer Composition

第1章 计算机系统概论

1.1 计算机系统简介

一、计算机的软硬件概念

1. 计算机系统

1. 计算机系统

- 硬件
  - 计算机的实体，如主机、外设
- 软件
  - 由具有各类特殊功能的信息（程序）组成

2. 软件

- 系统软件： 管理整个计算机系统
  - 语言处理程序
  - 操作系统
  - 服务型程序
  - 数据库管理系统
  - 网络软件
- 应用软件
  - 按任务需要编制成的各种程序

二、计算机系统的层次结构

- 高级语言 （提供编译/解释程序，再在机器上运行）
  - 虚拟机器
  - 用编译程序翻译成汇编语言程序
- 汇编语言 （符号语言，与机器语言一一对应）
  - 虚拟机器
  - 用汇编语言翻译成机器语言程序
- 操作系统 （管理软硬件程序）
  - 虚拟机器
  - 用机器语言解释操作系统
- 机器语言 （由01构成）
  - 实际机器
  - 用微指令解释机器指令
- 微指令系统
  - 微程序机器
  - 由硬件直接执行微指令

1.2 计算机的基本组成

冯·诺依曼计算机的特点

设计思想：存储程序，并按地址顺序访问

1. 计算机由五大部件组成：输入、输出、运算器、控制器、存储器
2. 指令和数据以同等地位存于存储器，可按地址寻访
3. 指令和数据用二进制表示
4. 指令由操作码和地址码组成
5. **存储程序**
6. 以运算器为中心

哈佛结构

设计思想：数据和指令分别放在两个存储器，能提高效率。

指令格式举例

"[]"表示存储在内存中。

指令都以二进制表示。

指令 = 操作码（指明进行说明操作的编号） + 地址码（储存数据/取出指令的地址）

取数 $\alpha \qquad \qquad \qquad [\alpha] \rightarrow ACC$

存数 $\beta \qquad \qquad \qquad \qquad ACC\rightarrow [\beta]$

存储器的基本组成

核心： 存储体 + MAR + MDR

- 存储体 - 存储单元 - 存储元件（0 / 1）
    - 存储单元： 存放一串二进制代码 （每个存储单元赋予一个地址，存储单元按地址寻址）
    - 存储字：存储单元中二进制代码的组合
    - 存储字长：存储单元中二进制代码的位数
- MAR（存储器地址寄存器）：反应存储单元的个数（存放存储单元地址编码）
- MDR（存储器数据寄存器）：反应存储字长
    - /要存入CPU的数据
    - /从存储体取出来的数据
    - /要存入存储体的数据

运算器的基本组成及操作过程

核心： ALU + （3个寄存器）： ACC + X + MQ

累加器型运算器的操作过程

	ACC（累加器）	**MQ**	**X**
加法	被加数$\$ 和		加数

	ACC（累加器）	MQ	X
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

1. 加法操作过程

指令： 加 + M

“加”为操作方法是加法，M为加数在内存单元的地址。

状态	操作	说明
初态	ACC 被加数	在执行加法前需要先用一条取数指令，把被加数存入ACC
	$[M] \rightarrow X$	取M中的加数存放到X寄存器中
	$[ACC]+[M] \rightarrow ACC$	用ALU算逻运算单元完成加法，存入累加器ACC

2. 减法操作过程

指令： 减 + M

“减”为操作方法是减法，M为减数在内存单元的地址。

状态	操作	说明
初态	ACC 被减数	在执行加法前需要先用一条取数指令，把被减数存入ACC
	$[M] \rightarrow X$	取M中的减数存放到X寄存器中
	$[ACC]-[M] \rightarrow ACC$	用ALU算逻运算单元完成减法，存入累加器ACC

控制器

核心: 控制单元 CU + (两个寄存器）： PC + IR

功能

- 解释指令
- 保证指令的按序执行

基本组成

完成一条指令:

- 取指令（要知道指令的地址，指令保存在内存单元中。**在PC中存放指令的地址，用PC取指令存入IR。PC自动指向下一条要执行的指令**）
- 分析指令（取指令的操作码部分进行分析，**控制单元可以从IR中将指令的操作码取出来进行分析**）
- **执行指令 （CU 控制单元：控制器的核心）**

PC（程序计数器）：

- 存放当前欲执行指令的地址
- 具有计数功能（PC） + 1 $\rightarrow$ PC （**使指令可以连续的执行**）

IR（指令寄存器）：

- 存放当前欲执行的指令

CU（控制单元）：控制器的核心

- 发送各种控制信号

$ax^2 + bx + c$ 程序的运行过程

- 将程序通过输入设备送至计算机
- 程序首地址 $\rightarrow$ PC (设置从哪条开始)
- 启动程序运行（运行程序，即连续运行多条指令）
- 取指令 PC $\rightarrow$ MAR $\rightarrow$ M $\rightarrow$ MDR $\rightarrow$ IR，(PC) + 1 $\rightarrow$ PC
- 分析指令 OP(IR)[表示指令的操作码] $\rightarrow$ CU： 取数操作 把x取到ACC中
- 执行指令 AD(IR)[表示指令的地址码] $\rightarrow$ MAR $\rightarrow$ M $\rightarrow$ MDR $\rightarrow$ ACC
- 乘法指令...
- 打印结果
- 停机

1.3 计算机硬件的主要技术指标

1. 机器字长：

- CPU一次能处理数据的位数。（通常与CPU中的寄存器位数有关）

2.运算速度

- **主频/时钟周期（f）**: $\quad$ 主时钟的频率（非直接）
  - CPU的时钟周期（T）： T = 1 / f $\quad$ 单位： us, ns
- 核数，每个核支持的线程数（非直接）
  - **吉普森法（静/动态使用频率）*
- CPU 执行时间： $\quad$ CPU执行一般程序所占用的CPU时间
  - CPU 执行时间 = CPU的时钟周期数 $\times$ CPU的时钟周期
- CPI： $\quad$ $\quad$ $\quad$ $\quad$ 执行一条指令所需的**平均**时钟周期数
  - CPI = 执行某段程序所需的CPU时钟周期数 $\div$ 程序包含的指令条数
- **MIPS**： $\quad$ $\quad$ $\quad$ $;$ 平均每秒执行多少百万条定点指令数
  - MIPS = 指令数 $\div$ （程序执行时间 $\times 10^6$）
- **FLOPS**： $\quad$ $\quad$ $\quad ; 每秒浮点运算次数 （更科学）
  - FLOPS = 程序中浮点操作次数 $\div$ 程序执行时间（s）

3. 存储容量

存放二进制信息的总位数

- 主存容量
  - 存储单元个数 $\times$ 存储字长
    - 如 $\quad$MAR MDR 容量
    - $\qquad$ 10 $\quad$ 8 $\quad$ 1K * 8位
  - 字节数
    - 如$\qquad 2^{13}b = 1 KB$
- 辅存容量
  - 字节数 80GB

6.1 无符号数和有符合数

有符号数:

真值： 日常中的数据

机器数： 需要存储3部分： 符号 + 小数点位置 + 数据信息

其中小数点的位置：为约定的方式给出。没有任何硬件标志

三种机器数的小结（原码，反码，补码）

- 最高位为符号位，书写上用“，”（整数）或“ . ”（小数）将数值部分和符号位隔开
- 对于正数，原码 = 补码=反码
- 对于负数，符号位为1，
  - 补码：其数值部分原码初符号位外每位取反，末位加1
  - 反码：其数值部分原码初符号位外每位取反
- 补码取原码：其数值部分原码初符号位外每位取反，末位加1

例1： 设机器数字长为8位（其中1位为符号位）。对于整数，当其分别代表无符号数、原码、补码和反码时，对应的真值范围各为多少?

解:

- 无符号: 0-255
- 原码 : -127-127
- 反码 : -127-127
- 补码 : -128-127

例2： 已知[y]补， 求[-y]补

解： 包括符号位在内，每位求反，末位加1.

移码表示法

1. 定义

☑ 移 = $2^n + x \quad (2^n > x >= -2^n)$

x 为真值； n 为真值的位数

注意：

- 不论正负，都把真值加上2^n
- 求法类似补码，但是符号位相反
- 只有整数，无小数（表示浮点小数中的阶码）

2. 移码 vs 补码

设 x = 1100100

- 补码：0,1100100
- 移码：1,1100100

设 x = -1100100

- 补码：1,0011100
- 移码：0,0011100

移码和补码只相差一个符号位。

定点表示和浮点表示

定点机及小数和整数的表示范围

1. 定点小数
    - 小数点位置：最高位（符号位）右侧
    - 表示范围： $0 < |x| < 1 - 2^{-n}$
2. 定点整数表示范围
    - 小数点位置：最低位右侧
    - 表示范围： $0 < |x| < 2^n -1$

浮点表示

问题

- 为什么引入浮点数表示
- 浮点数表示格式是什么
    - 尾数的符号，长度，格式
    - 阶码的符号，长度，格式
- 用那种机器数格式表示尾数和阶码
- 尾数和阶码基值必须是2么?
- 基值大小对浮点数的影响
    - 大小
    - 表示范围
    - 最大值，最小值
- 为什么引入规格化？怎么规格化?
- 浮点数表示格式标准?

为什么引入浮点数表示?

- 简化程序员编程难度
- 扩展数的表示范围

一、浮点表示

N = S * r^j
S 尾数； j 阶码； r 尾数的基值 (r = 2、4、8、16....)

- S 小数、可正可负
- j 整数、可正可负

例： r = 2 时， N = 11.0101 = $0.110101 * 2^{10}$ 规格化数

1. 机器中浮点数的存储

j: 阶符（1位）+ 阶码数值部分（整数m位） + S: 数符（1-2位） + 尾数的数值部分（小数n位）

2. 表示范围

- 最小负数
- 最大负数
- 最小正数
- 最大正数

上溢 阶码 > 最大阶码 计算出错

下溢 阶码 < 最小阶码 按机器零处理

** 阶码： 控制数的范围

尾数： 控制数的精度

3. 对浮点数进行规格化

- 作用：尽可能保证数据的精度
- 做法：小数点后面真值最高位为1.
  - 例: 0.1011 yes
  - 0.001001 no
- 基值不一样，规格化不一样。
  - 例： r = 4, 尾数最高2位不全为0.

4. 浮点数规格化

尾数左右移动，并配合阶码增减。

r	类型	操作
r=2	左规	尾数左移1位，阶码减1
r=2	右规	尾数右移1位，阶码加1
r=4	左规	尾数左移2位，阶码减1

r	类型	操作
r=4	右规	尾数右移2位，阶码加1

基值r越大，表示的浮点数的范围越大，个数越多 基值r越大，表示的浮点数的精度降低

5. 机器零

- `x,xxxx;0.00···0`: 浮点数的尾数为0，不管阶码的值，按机器零处理。
- `1,00···0;x.xx···x`: 阶码等于或者小于所能表示的最小值。
- `0,0000;0.00···0`: 阶码用移码，尾数用补码表示，机器零为：

6.IEEE754

形式： 数符 + 阶码（含阶符） + [.] + 尾数

规定：

- 尾数必须用规格化表示，即非“0”的有效位最高位为“1”（隐含）。增大了尾数的尾数。尾数域表示的真值：1.M
- 基数固定为2.
- 阶码用移码表示,偏移值为 127 = $2^8 - 1$

	符号位S	阶码	尾数	总位数
短实数	1	8	23	32
长实数	1	11	52	64
临时实数	1	15	64	80

6.3 定点运算 p74

一、移位运算（拓展）

意义

小数点并不动，数据相对于小数点左右移动。

左移 绝对值扩大 右移 绝对值缩小

####算术移位规则

- 符号位不变

	码制	添补代码
正数	原码、补码、反码	0
负数	原码	0
	补码	左移 添0

码制	添补代码
	右移 添1
反码	1

算数移位和逻辑移位

算数移位 有符号位的移位 （最高位不参加）

逻辑移位 无符号位的移位（所有数据域都参与移位）

二、加减法运算

补码加减法运算的公式

1. 加法

   整数 $[x]_补 + [y]_补 = [x + y]_补$ (mod 2^{n+1})

   小数 $[x]_补 + [y]_补 = [x + y]_补$ (mod 2)

2. 减法

   整数 $[x - y]_补 = [x]_补 + [-y]_补$ (mod 2^{n+1})

   小数 $[x - y]_补 = [x]_补 + [-y]_补$ (mod 2)

补码的加法运算，连同符号位一起相加，符号位产生进位自动丢掉。

溢出判断

1. 用一位符号位判断溢出

   - 分析： 异号运算 不会发生溢出；同号运算 可能发生溢出
   - 硬件实现： 最高有效位的进位 $\oplus$ 符号位的进位 = 1

2. 两位符号位判断溢出 $$ [x]_{补'} = \begin{cases} x, & \text{1 > x $\ge$ 0} \[3ex] x + 4, & \text{-1 $\ge$ x $\gt$ 0 （mod 4）} \end{cases} $$

- 判读标准
  - 结果的双符号位相同 未溢出
  - 结果的双符号位不同 溢出 10， xxxx; 01,xxxx
- 最高符号位代表： 真正的符号； 后者代表：溢出部分

补码加减法的硬件配置

三、乘法运算

计算机中二进制乘法的原理

- 乘法运算可用加和移位实现： n = 4, 加4次，右移4次

- 乘数末尾决定是否与原部分积相加/加0. 右移之后，高位用乘法结果右移丢失的最低位添补。
- 被乘数只与部分积的高位相加
- 3个寄存器，2个具有移位功能

原码的乘法

1） 原码一位乘运算规则：符号位进行 $\oplus$ ，小数部分取绝对值进行计算

2）原码一位乘递推公式：部分积进行逻辑右移，其余类似乘法运算公式的操作。

特点：

1. 绝对值运算
2. 移位过程中，采取逻辑移位，左侧补零。
3. 用移位次数判断乘法是否结束，n次加法,n次移位。

计算机中的除法运算

- 符号单独处理，剩余部分做除法；
- 被除数不为0， 为0时，直接判断= 0；
- 小数除法的结果仍为小数；整数除法结果 > 1

除法运算方法

1） 恢复余数法

- 余数为正 上商 1
- 余数为负 上商 0，恢复余数

2） 加减交替法（不恢复余数法）

- 余数为正 上商 1 余数左移 $+ [-|y|]_补$
- 余数为负 上商 0 余数左移 $+ [|y|]_补$

特点：

- 上商 n + 1 次
- 第一次上商判溢出
- 移n次，加 n + 1 次

7.1 机器指令

一、指令的一般格式

操作码字段（地址可以分开存放） + 地址码字段

1. 操作码：

反映机器对什么类型的数据，做什么操作。指明操作数的寻址方式。

- 长度固定：用于指令字长较长的情况 （译码操作方便）

- 长度可变：操作码分散在指令字的不同字段中
- 扩展操作码技术：操作码位数随地址数的减少而增加
    - 保留扩展标志的扩展方式。
    - 例：指令格式：$OP + A_1 + A_2 + A_3$ ，其中OP 和 A_i都为4位。OP可以由（0000~1110）表示，此时三地址指令操作码共15种。）OP = 1111时，A_1 = （0000~1110），可表示15种二地址码指令。以此类推...
- 短操作码一般表示常用指令，长操作码一般表示非常用指令。

2.地址码

1. 四地址： $OP + A_1 + A_2 + A_3 + A_4$

    - A_1 第一操作数地址
    - A_2 第二操作数地址
    - A_3 结果的地址
    - A_4 下一条指令地址

    $(A_1)OP(A_2) \rightarrow A_3$， 共4次访存（取指令+取两个操作数+存放结果）

    设指令字长为32 位，操作码固定8位，则寻址范围$2^6 = 64$。范围过小。

2. 三地址： $OP + A_1 + A_2 + A_3$

    - A_1 第一操作数地址
    - A_2 第二操作数地址
    - A_3 结果的地址
    - *A_4 用PC代替

    $(A_1)OP(A_2) \rightarrow A_3$， 共4次访存

    设指令字长为32 位，操作码固定8位，则寻址范围$2^8 = 256$。

3. 二地址： $OP + A_1 + A_2$

    - A_1 第一操作数地址
    - A_2 第二操作数地址
    - *用A_1 / A_2 代替 A_3 结果的地址
    - *A_4 用PC代替

      $(A_1)OP(A_2) \rightarrow A_1$

    或 $\quad (A_1)OP(A_2) \rightarrow A_2$， 共4次访存

    设指令字长为32 位，操作码固定8位，则寻址范围$2^{12} = 4K$。

4. 一地址： $OP + A_1$

    - A_1 第一操作数地址
    - *A_2 第二操作数地址默认保存在内存中的某个位置（例如 ACC）

- *用A_1 / A_2 代替 A_3 结果的地址
- *A_4 用PC代替

  $(ACC)OP(A_1) \rightarrow ACC$ ， 共2次访存（对于ACC的操作，不需要访存，省去两次）

设指令字长为32 位，操作码固定8位，则寻址范围$2^{24} = 16M$。

5. 零地址： OP

例如：

- 给ACC清零（ACC可以隐含取值）
- 给ACC中数据取反
- 停机指令

二、指令字长

决定因素：

- 操作码的字长
- 操作码地址的长度
- 操作码地址的个数

1. 指令字长固定

指令字长 = 储存字长

2. 指令字长可变

按字节的倍数变化

7.2操作数类型和操作种类

一、操作数类型

地址 $\qquad$ 无符号整数

数字 $\qquad$ 定点数、浮点数、十进制数

字符 $\qquad$ ASCII

逻辑数 $\quad$ 逻辑运算

二、数据在存储器中的存放方式

字节编址，数据在存储器中的存放方式

- 从任意位置开始存储
  - 优点：不浪费存储空间
  - 缺点：所有类型的数据都要浪费两个存储周期的时间。读写控制复杂
- 从一个存储字的起点位置开始访问

    - 优点：无论访问何种类型数据，均可在一个周期内完成，读写控制简单。
    - 缺点：浪费存储资源。
- 边界对准方式：从地址的整数倍位置开始访问

三、操作类型

1. 数据传送

类型：

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
类型	RR型	RS型	SR型	SS型
例如	MOVE	STORE $\$ MOVE$\$PUSH	LOAD$\$ MOVE$\$POP	MOVE

基础操作：

传送 $\qquad$ MOV $\quad; ;$ 清零 $\qquad$ CLA

存数 $\qquad$ STO $\qquad$ 置1 $\qquad$ SET

取数 $\qquad$ LAD $\qquad$ 进栈 $\qquad$ PUS

交换 $\qquad$ EXC $\qquad$ 退栈 $\qquad$ POP

2.算术逻辑操作

算术运算：

加法 $\qquad$ ADD $\qquad$ 绝对值 $\quad$ ABS

减法 $\qquad$ SUB $\qquad$ 变负 $\qquad$ NEG

乘法 $\qquad$ MUL $\qquad$ 增量 $\qquad$ INC

除法 $\qquad$ DIV $\qquad$ 减量 $\qquad$ DEC

逻辑运算：

与 $\qquad$ AND $\quad$ 测试 $\qquad$ TES

或 $\qquad$ OR $\qquad$ 比较 $\qquad$ COM

非 $\qquad$ NOT $\qquad$

异或 $\quad$ XOR $\qquad$

3.移位操作

算术移位 $\qquad$ 逻辑移位 循环移位（带进位和不带进位）ROT

4.转移

- 无条件转移 JMP
- 有条件转移 JZ JO JC
- 调用和返回
- 陷阱与陷阱指令（意外事故的终端，由硬件自动完成）
- 输入输出
    - 入 $\qquad$ 端口中的内容$\rightarrow$ CPU的寄存器
    - 出 $\qquad$ CPU的寄存器$\rightarrow$ 端口中的内容

7.3 寻址方式

寻址方式：（找操作数/指令的地址）

- 确定 本条指令的操作数地址
- 确定 下一条要执行指令的指令地址

一、指令寻址

顺序寻址 $\quad$ (PC) + 1 $\rightarrow$ PC

跳跃寻址 $\quad$ 由转移指令指出

二、数据寻址

形式地址$\quad$ 指令字中的地址

有效地址$\quad$ 操作数的真实地址

约定$\quad$ 指令字长=存储字长=机器字长

格式：

操作码	寻址特征	形式地址A

1.立即寻址

形式地址A： 操作数

格式： `OP + # + A`

- #：立即数寻址的特征
- A：立即数：可正可负 补码
- 取操作数：取A
- 指令执行阶段访存：0次

2.直接寻址

EA = A $\quad$ 有效地址由形式地址直接给出

格式： `LDA + 寻址特征 + A`

- A：有效地址
- 取操作数：地址A的内存$\rightarrow$ 操作数
- 缺点
  - 操作数的地址不易修改（必须修改A，不利于指令循环）
- 指令执行阶段访存：1次

3.隐含寻址

一个操作数的地址隐含在OP（操作数）中

格式： `ADD + 寻址特征 + A`

- A：直接寻址
- 取操作数：
  - 一个操作数：A的直接寻址
  - 另一个操作数隐含在 ACC 中
- 指令字减少了一个地址字段，缩短指令字长
- 指令执行阶段访存：1次

4.间接寻址

EA = (A) $\quad$ 有效地址由形式地址间接提供

格式： `OP + @ + A`

a. 一次间址

- @：间接寻址的特征
- 取操作数：A地址所在内存 $\rightarrow$ 有效地址EA $\rightarrow$ EA地址所在内存 $\rightarrow$ 操作数
- 优点
  - 扩大寻址范围：[A 可以很短，EA可以较长。]
  - 便于编程，可以只修改EA。
- 指令执行阶段访存：2次

a. 多次间址

- A：包含可以寻找到操作数的中间地址。
- 多次进行一次间址过程。
- 指令执行阶段访存：多次

5.寄存器寻址

EA = R$_i \quad$ 有效地址：寄存器编号

格式： OP + 寻址特征 + R$_i$

- R$_i$：操作数
- 取操作数：寄存器 $\rightarrow$ 操作数
- 优点：

    - 只访问寄存器，执行速度快。
    - 寄存器个数有限，可缩短指令字长。
- 指令执行阶段访存：0次

6.寄存器间接寻址

EA = (R$_i) \quad$ 有效地址：寄存器编号。

格式： OP + 寻址特征 + R$_i$

- R$_i$：操作数的地址。操作数存放在内存单元。
- 取操作数：寄存器 $\rightarrow$ 内存地址 $\rightarrow$ 内存单元 $\rightarrow$ 操作数
- 优点
    - 便于指令循环操作
- 指令执行阶段访存：1次

7.基址寻址

a. 采用专用寄存器作基址寄存器

EA = (BR) + A $\quad$ BR为基址寄存器。

格式： OP + 寻址特征 + A$

- A: 偏移量
- 取操作数：(BR) + A $\rightarrow$ 地址值 $\rightarrow$ （地址值）
- 优点
    - 可扩大寻址范围
    - 有利于多道程序
    - BR内容由操作系统或管理程序确定
    - 程序执行过程中BR内容不变，形式地址A可变

b. 采用通用寄存器作基址寄存器

EA = (R$_0$) + A $\quad$ BR为基址寄存器。

格式： OP + 寻址特征 + R$_0$ + A

- A: 偏移量
- 取操作数：(R$_0$) + A $\rightarrow$ 地址值 $\rightarrow$ （地址值）

8.基址寻址

EA = (IX) + A $\quad$ IX 为专用变址寄存器

$\qquad \qquad \qquad$ 通用寄存器也可以作为变址寄存器

格式： OP + 寻址特征 + A$

- A: 偏移量
- 取操作数：(IX) + A $\rightarrow$ 地址值 $\rightarrow$ （地址值）
- 优点

- 可扩大寻址范围
- 有利于多道程序
- A固定，IX可以被修改（和基址相反）。[便于数组操作：A作为数组起点，IX为数组下标]

9.相对寻址

EA = (PC) + A $\quad$ A 是相对于当前指令的位移量（可正可负，补码）

格式： OP + 寻址特征 + A$

- A: 要执行指令和当前指令的 相对距离
- 取操作数：(PC) + A $\rightarrow$ 地址值 $\rightarrow$ （地址值）
- 优点
  - 程序浮动

10.堆栈寻址

1. 堆栈的特点

- 硬堆栈 $\quad$ 多个寄存器
- 软堆栈 $\quad$ 指定的存储空间

先进后出（一个入出口） **栈顶地址**由SP指出(一般栈顶是低地址。底是高地址，越往下，地址越大)

进栈 (SP) - 1 $\rightarrow$ SP

出栈 (SP) + 1 $\rightarrow$ SP

2. SP的修改与主存编址方法有关

- 按字编址 $\quad$(SP) + / -1
- 按字节编址
  - 储存字长16位 $\quad$(SP) + / -2
  - 储存字长32位 $\quad$(SP) + / -4

7.4 指令格式举例

*一、设计时需要考虑的因素

1. 指令系统的兼容性

2. 其他因素

- 操作类型： 指令个数及操作难易度
- 数据类型： 哪些类型
- 指令格式：
  - 指令字长是否固定
  - 操作码位数、是否采用扩展操作码技术
  - 地址码位数、地址个数、寻址方式类型
- 寻址方式：指令寻址、操作数寻址
- 寄存器个数：直接影响指令的执行时间

7.5 RISC技术

一、RISC的产生和发展

RISC (Reduced Instruction Set Computer) CISC (Complex Instruction Set Computer)

2-8规律： 典型程序中80%的语句仅仅使用处理机中20%的指令

二、RISC的主要特征

- 选用使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
- 指令 长度固定、指令格式种类少、寻址方式少
- 只有 LOAD/STORE 指令访存
- CPU中有多个通用寄存器
- 采用流水技术 一个时钟周期内完成一条指令
- 采用组合逻辑实现控制器

三、CISC的主要特征

四、RISC和CISC的比较

8.2 指令周期

一、基本概念

1. 指令周期：

定义： 取出 + 执行指令 的全部时间

指令周期 = 取指周期 + 执行周期

完成一条指令：

- 取指（存入IR）、分析（操作码 + 寻址方式）
- 执行（完成全部运算）

*2. 每条指令的指令周期不同

*3. 具有间接寻址的指令周期

*4. 具有中断周期的指令周期

5. 指令周期流程

取指周期 $\rightarrow$ 执行周期 $\rightarrow$ ...

6.CPU 工作周期的标志

二、指令周期的数据流

8.3 指令流水

系统的并行性

1.并行的概念：

- 并发：多个事件在同一时间段发生
- 同时：多个事件在同一时刻发生

2. 并行性的等级

- 过程级 （程序、进程）$\qquad$ 软件实现
- 指令级（指令之间/内部）$\quad$ 硬件实现

三、指令流水原理

1. 指令的串行执行

取指令1 $\rightarrow$ 执行指令1 $\rightarrow$ 取指令2 $\rightarrow$ 执行指令2 ...

取指令 和 执行指令 分布完成，总有一个部件空闲

2. 指令的二级流水

取指令1 $\rightarrow$ 执行指令1

$\qquad \qquad$ 取指令2 $\rightarrow$ 执行指令2 ...

取指和执行阶段时间上完全重叠，指令周期减半。

3. 影响指令流水效率加倍的因素

1） 执行时间 $\gt$ 指令时间

取指令部件 $\rightarrow$ **指令部件缓存区** $\rightarrow$ 执行指令部件

2） 条件转移指令对指令流水的影响

四、流水线性能

2. 运算流水线

以浮点加减运算为例：

- 对阶
- 尾数求和
- 规格化

对阶功能部件 $\rightarrow$ 锁存器 $\rightarrow$ 尾数加部件 $\rightarrow$ 锁存器 $\rightarrow$ 规格化部件 $\rightarrow$ 锁存器。

分段原则： 每段操作时间尽量一致

9.1 微操作命令分析

微操作命令： 指令解释过程中，由控制单元发出的指令。

一、取指周期

指令	操作说明
PC $\rightarrow$ MAR $\rightarrow$ 地址线	
1 $\rightarrow$ R	发出读命令
M(MAR) $\rightarrow$ MDR	
MDR $\rightarrow$ IR	
OP(IR) $\rightarrow$ CU	译码部分：把指令操作码给CU
(PC) + 1 $\rightarrow$ PC	

二、间址周期

指令	说明
指令形式地址 $\rightarrow$ MAR	
Ad(IR) $\rightarrow$ MAR	
1 $\rightarrow$ R	
M(MAR) $\rightarrow$ MDR	
MDR $\rightarrow$ Ad(IR)	取得操作数

三、执行周期

1. 非访存指令

(1) CLA 清A

(2) COM 取反

(3) SHR 算数右移

(4) CSL 循环左移

(5) STP 停机指令

2. 访存指令

(1) 加法指令 ADD X

> ACC中的一个加数和给出内存地址码X的另一个加数相加，结果保存在ACC中。此时，ACC中已经保存了一个被加数，即被加数已经在CPU中。

指令	说明

指令	说明
Ad(IR) $\rightarrow$ MAR	取存在内存中的加数

1 $\rightarrow$ R M(MAR) $\rightarrow$ MDR (ACC) + (MDR) $\rightarrow$ ACC

(2) 存数指令 STA X

> 把ACC中保存的数据，存到内存中的给定地址X。

指令	说明
Ad(IR) $\rightarrow$ MAR	把内存的地址给MAR
1 $\rightarrow$ W	CU给存储器发写命令

ACC $\rightarrow$ MDR MDR $\rightarrow$ M(MAR)

(3) 取数指令 LDA X

> 把内存中指定的地址单元的数据保存到ACC中。

指令	说明
Ad(IR) $\rightarrow$ MAR	把内存的地址给MAR
1 $\rightarrow$ R	读命令
M(MAR) $\rightarrow$ MDR	取出数据放入MDR

MDR $\rightarrow$ ACC

3.转移指令

(1) 无条件转移 JMP X

> 要转移的地址，就是该指令的地址码部分。

$Ad(IR) \rightarrow PC$

(2) 条件转移 BAN X（负则转）

$A_0·Ad(IR) + \overline{A_0}(PC) \rightarrow PC$

4.三类指令周期

- 非访存 指令周期
  - 取指周期
  - 执行周期
- 直接访存 指令周期
  - 取指周期
  - 执行周期
- 间接访存 指令周期

    - 取指周期
    - 间址周期
    - 执行周期
- 转移 指令周期
    - 取指周期
    - 执行周期
- 间接转移 指令周期
    - 取指周期
    - 间址周期
    - 执行周期

四、中断周期

9.2 控制单元的功能

一、控制单元的外特性

指令寄存器，控制单元CU，标志，时钟

控制信号：CPU内部的控制信号，来自系统总线的，传到系统总线的

1. 时钟信号

1） CU 受时钟控制

一个时钟脉冲

- 发一个操作命令
- 发一组需要同时执行的操作命令

2）IR 指令寄存器 $\quad$ OP(IR) $\rightarrow$ CU

控制信号与操作码有关（由译码结果对应不同操作发出不同控制信号）

3）标志 CU 受标志控制

4）外来信号

2.输出信号

1）CPU内的各种控制信号

- CPU内寄存器间数据传输：$R_i \rightarrow R_j$
- 取下一条指令：$(PC) + 1\rightarrow PC$
- CPU内部的数逻运算：ALU，+，-，与，或 ……

2）送至控制总线的信号

- 访存控制信号：$\overline{MREQ}$
- 访IO / 存储器的控制信号：$\overline{IO}/M$
- 读命令：$\overline{RD}$

- 写命令：$\overline{WR}$
- 中断响应信号：INTA
- 总线响应信号：HLDA

三、多级时序系统

1.机器周期

1）机器周期的概念

- 所有指令执行过程中的一个基准时间

2）确定机器周期需考虑的因素

- 每条指令的执行步骤
- 每一步骤所需的时间

3）基准时间的确定

- 已完成最复杂指令功能的时间为准
- 通常 以访问一次存储器的时间为基准

若 指令字长 = 存储字长 $\qquad$ 取指周期 = 机器周期

2.时钟周期（节拍、状态）

- 一个机器周期内可完成若干个微操作，每个微操作需一定的时间
- 将一个机器周期分成若刚个时间相等的时间段（节拍、状态、时钟周期）
- 由时钟周期控制，微操作的先后顺序
- 时钟周期是控制计算机操作的最小时间单位
- 用时钟周期控制产生一个或几个微操作命令

3.多级时序系统

指令周期 > 机器周期 > 时钟周期（节拍）

4.机器速度与机器主频的关系

还和包含的机器周期数和一个机器周期包含时钟周期数，以及流水/非流水方式有关。

四、控制方式

1.同步控制方式

任一微操作均由统一基准时标的时序信号控制

- 采用定长的机器周期
- 采用不定长的机器周期
- 采用中央控制和局部控制相结合的方法

2.异步控制方式

- 采用应答方式

3. 联合控制方式

- 同步与异步相结合

4. 人工控制方式

- Reset
- 连续和单条指令转换开关
- 符合停机开关

微指令和机器指令的区别

- 指令：表示不同
- 时间：微指令更快速
- 流程不同：微指令直接带转移字段（控制转移字段）

第10章 控制单元的设计

10.1 组合逻辑设计

一、组合逻辑控制单元框图

- CU
- 节拍信号

二、微操作的节拍安排

案例：

- 采用同步控制方式，有统一的节拍控制信号控制。
- 一个机器周期内有3个节拍（时钟周期）[一个机器周期内有多少个节拍的设计，与控制指令的条数和复杂程度有关]
- CPU内部结构采用非总线方式

CU控制的内容

- 发布所有的时钟信号C_i
- 发布控制ALU进行某一具体操作的信号
- 修改标志

1.安排微操作时序的原则

- 微操作的先后顺序不得随意更改
- 被控制对象不同的微操作，尽量安排在一个节拍内完成（并行执行的微操作，微操作之间没有相互联系，这样的操作尽量安排在一个节拍内完成）
- 占用时间较短的微操作，尽量安排在一个节拍内完成，并允许有先后顺序。[例：一个微操作安排在上升沿完成，另一个微操作安排在下降沿完成。]

2. 取指周期微操作的节拍安排

节拍信号	操作	说明
T_0	$PC \rightarrow MAR\1 \rightarrow R$	原则二；PC-MAR是内存到寄出器，所用的时间比T_3中R-R的传输所用的时间更短
T_1	$M(MAR) \rightarrow MDR\(PC) + 1\rightarrow PC$	原则二；没有先后循序，可以互换
T_2	$MDR \rightarrow IR\OP(IR) \rightarrow ID$	原则三；用时短，故可以安排量一起并行执行；第二行是IR的操作码的译码。

3. 间址周期微操作的节拍安排

节拍信号	操作	说明
T_0	$A_d(IR) \rightarrow MAR\1 \rightarrow R$	原则二
T_1	$M(MAR) \rightarrow MDR$	
T_2	$MDR \rightarrow A_d(IR)$	

3. 执行周期微操作的节拍安排

1）CLA 清零

节拍信号	操作
T_0	
T_1	
T_2	$0 \rightarrow ACC$

2）COM 取反

节拍信号	操作
T_0	
T_1	
T_2	$\overline{AC} \rightarrow AC$

3）SHR 右移

节拍信号	操作
T_0	
T_1	

节拍信号	操作
T_2	$L(AC) \rightarrow R(AC) \ AC_0 \rightarrow AC_0$

4）CSL 循环左移

节拍信号	操作
T_0	
T_1	
T_2	$R(AC) \rightarrow L(AC) \ AC_0 \rightarrow AC_n$

5）STP

节拍信号	操作
T_0	
T_1	
T_2	$0 \rightarrow G$

6）ADD X

7）STA X

8）LDA X

9）JMP X

10）BAN X

***4. 中断周期微操作的节拍安排**

三、组合逻辑设计步骤

1. 列出操作时间表

2. 写出微操作命令的最简表达式

3. 画出逻辑图

10.2 微程序设计

一、微程序设计思想的产生

完成一条机器指令

- 微指令 1（10100000)
    - 微操作命令1 + 微操作命令2 + ...
- ...

- 微指令 m (00010010) 1->控制对应的两个微操作命令的执行
    - ... + 微操作命令n

总结:

- 层次线
    - 完成一条机器指令，对应一个微程序。
    - 一个微程序包含了若干条微指令。
    - 一条微指令包含一个或多个微操作的控制信号。
- 存储逻辑
    - 由微程序控制实行一条指令的执行。
    - 微指令的操作顺序，就是微操作执行过程中的先后顺序。
    - 微指令/微指令构成的微程序保证存在只读存储器ROM中。执行过程中，把ROM中存储的指令一条一条读出，根据微指令中有效控制信号的位置，发送对应的信号。

二、微程序控制单元框图及工作原理

1. 机器指令对应的微程序

- 取指周期微程序
- 间址周期微程序
- 中断周期微程序
- 对应LDA操作的微程序
- 对应STA操作的微程序

2. 微程序控制单元的基本框图

- 控制存储器（ROM 只读）：保存微程序/微指令
- CMAR（控制存储器地址寄存器）：保存微指令的地址
- 地址译码
- CMDR（控制存储器数据寄存器）：
- 微地址形成部件
    - （因为执行周期中每个指令需要执行的指令是不同的，微指令的首地址不同。因此，需要一个部件生成要找到的微指令的地址。）
    - 接受IR中送来的操作码部分。
- 顺序逻辑（CMAR传来的地址有多种来源）
    - 标志
    - CLK

数据传输逻辑

$OP(IR) \rightarrow 微地址形成部件（形成微程序指令地址）\rightarrow 顺序逻辑（选择正确的地址）\rightarrow CMAR \rightarrow 地址译码 \rightarrow 控制存储器（取指令） \rightarrow CMDR \rightarrow (CMDR)下地址 \rightarrow 顺序逻辑$

CMDR的操作码部分可以直接发送信号至CPU内部和系统总线的控制信号。

微指令基本格式

操作控制 + 顺序控制

操作控制：

- 由一系列01构成，每一位代表一个微操作控制信号。1表示有效。

顺序控制

- 下一条微指令的地址

小结

- 一条微指令完成多个操作。
- 完成一个微程序中的所有微指令，则完成一个指令。如：取指操作。

工作原理：见笔记

三、微指令的编码方式（控制方式）

1. 直接编码（直接控制）方式

在微指令的操作控制字段中，每一位代表一个微操作命令。

某位为“1”，则表示该控制信号有效。

无需译码，速度最快。

2. 字段直接编码方式

将微指令的控制字段分成若干“段”，每段经译码后发出控制信号。

显式编码：每个字段中的命令是互斥的。

缩短了微指令字长，增加了译码时间。

3. 字段间接编码方式

隐式编码：每个字段结果与自己和其他的译码结果都相关。

4. 混合编码

直接编码和字段编码（直接和间接）混合使用。

5. 其他

四、微指令序列地址的形成

1. 微指令的下地址段指出

2. 根据机器指令的操作码形成

3. 增量计数器（如：执行阶段很多微地址是顺序 + 1）

4. 分支转移：

格式：控制造作字段 + 转移方式 + 转移地址

- 转移方式: 指明判别条件
- 转移地址：指明转移成功后的去向

5. 通过测试网络

6. 由硬件产生微程序入口地址

五、微指令格式

1. 水平型微指令

一次能定义并执行多个并行操作

> 直接编码、字段直接编码、字段间接编码、直接和字段混合编码

2. 垂直型微指令

类似机器指令操作码的方式

由微操作码字段规定微指令的功能

3. 两种微指令格式的比较

- 水平型微指令比垂直型微指令 并行操作能力强，灵活性强。
- 水平型微指令执行一条机器指令所要的微指令数目少，速度快。
    - 一条微指令的控制信号多
    - 微程序比较短，微指令数比较短，不需要译码
- 垂直型微指令用较短的微程序结构换取较长的微指令结构
- 水平型微指令与机器指令差别较大