计算机组成原理复习指南

针对于ppt所划知识点的逐一解答。 *powered by BoffinZhang*

目录

数据的编码表示

1. 原码、反码、补码

对于一个二进制数，通常我们会单独考虑符号位，如$[-8]_{10}=[-1000]_2$ 但是在机器码之中，我们以额外的一位二进制数代替符号位，0表示正，1表示负，这样就有$[-8]_{10}=[11000]_{原}$ 这种表示方法就叫做**原码**。 我们将原码**除了符号位以外的其它各位***按位取反*，就得到了反码，$[11000]_{原}=[10111]_{反}$ 我们再将反码**加1**，就得到补码$[11000]_{原}=[10111]_{反}=[11000]_{补}$ 注：补码也可以通过**取反加一**的方式得到

原码。

- n位二进制数的表示范围 - 原码：$-(2^{n-1}-1) \sim 2^{n-1}-1$ - 反码：$-(2^{n-1}-1) \sim 2^{n-1}-1$ - 补码：$-2^{n-1} \sim 2^{n-1}-1$

2. 移码

- 移码通常用于表示浮点数的阶码
- 以IEEE754浮点数中的8位移码为例，设移码为E，真值为e，有$E=e+[011111111]_2$即$E=e+127$。
- n位移码通式：$E=e+2^{n-1}$
- 表示范围：$e \in [-2^{n-1}, 2^{n-1}-1]$（和补码范围相同）

3. 浮点数

- 一般表示法：N位**阶码e**，K位尾数M,
  真值为$M*2^e$
- IEEE754（32位浮点）：1位符号位S，8位移码E=e+127，23位尾数M，真值为：
  $(-1)^s*(1.M)*2^{E-127}$
- *IEEE754浮点数的特殊情形
  - E=0,M=0：表示机器0
  - E=0,M!=0：表示非规格化浮点数$(-1)^S*(0.M)*2^{E-126}$
  - 1<=E<=254：表示正常的规格化浮点数$(-1)^s*(1.M)*2^{E-127}$
  - E=255,M=0：无穷大inf
  - E=255,M!=0：非数值NaN，对应$0/0$

4. 规格化浮点数的表示范围与精度取决什么?

- 表示范围取决于**阶码**的位数
- 精度取决于**尾数**的位数

5. 浮点运算中，判断尾数是否规格化的方法?

- 依据PPT讲解，尾数规格化就是通过**移位**的方式将尾数化为$0.1101...*2^e$的形式

指令系统

1. 掌握一些常见的寻址方式

数据寻址方式是根据**指令中给出的地址码字段**寻找**真实操作数地址**的方式，**形式地址**经过某种运算而得到的能够直接访问主存的地址称为**有效地址**（Effective Address，简记作EA）。 **指令中的形式地址->(寻址方式)->有效地址** 下面是几种基本的寻址方式

1. 立即寻址 指令中直接给出的立即数就是操作数。

OP	立即数

2. 寄存器寻址 指令中的地址码部分给出某一个通用寄存器的编号R_i，这个指定的寄存器存放着操作数。此时，操作数S与寄存器R_i的关系为：

$$S = (R_i)$$

3.直接寻址 指令中的地址码字段给出的地址A就是操作数的有效地址，即形式地址等于有效地址：

$$EA = A$$

4.间接寻址 指令中给出的地址A是**存放着*操作数地址*的主存单元的地址**，通常会在指令格式中划出一位作为直接或间接寻址的标志位，间址寻址时标志位@=1。 一级间接寻址时，指令中存储的A是操作数地址的地址，(A)是操作数的地址，((A))就是操作数。

$$S = ((A))$$

如此可以进行多次间接寻址。

5. 寄存器间接寻址 指令中的地址码给出的是某一个通用寄存器的编号，被指定的寄存器存放的是**操作数的有效地址**。操作数S与寄存器号R_i的关系为：

$$S = ((R_i))$$

6. 变址寻址 变址寻址把**变址寄存器R_x的内容**与指令中给出的**形式地址A**相加，**形成操作数的有效地址**，即$EA = (R_x) + A$，而实际的操作数地址为：

$$S = ((R_x) + A)$$

变址寻址最典型的用法是将**指令中的形式地址作为基准地址**，而变址寄存器中的内容作为修改量，在需要频繁修改地址时，只要修改变址值就可以了（用于汇编语句），对于**数组运算、字符串操作**等成批的数据处理是很有用的。

7. 基址寻址 基址寻址将基址寄存器R_b的内容与给出的位移量D相加，形成操作数的有效地址，即$EA = (R_b) + D$，而实际的操作数为：

$$S = ((R_b) + D)$$

基址寻址原来是大型计算机采用的一种技术，用来将用户的**逻辑地址**转化成主存的**物理地址**。

这一部分大致了解即可，不用死记硬背，实际工程中会依照指令集设计不同而采用不同的寻址方式，考试时也会给出具体的规则。

2. 相对寻址

相对寻址的有效地址为：$EA = (PC) + D$由程序计数器（PC）提供基准地址，指令中给出的地址码字段作为位移量D。 这种寻址方式有两个特点：

- 操作数的地址不是固定的，随之PC值的变化而变化，并且与指令地址之间总是相差一个固定值。当指令地址变换是，由于其位移量不变，使得操作数与指令在可用的存储区内一起**浮动**。

- 指令给出的位移量D可正可负，采用补码表示，若位移量为n位二进制数，则相对寻址的寻址范围为：$(PC)-2^{(n-1)} \sim (PC)+2^{(n-1)}-1$

3. 堆栈

1. 寄存器堆栈 在计算机中用一组专门的寄存器构成寄存器堆栈，称作**硬堆栈**。（可以想想用寄存器组成堆栈会有多贵） 这种堆栈的**栈顶**是固定的，他们之间具有对应位自动推移的功能，可以将一个寄存器的内容推移到相邻的另一个寄存器中。 **2. 存储器堆栈** 从主存中划出一部分作为**软堆栈**，**堆栈的大小可变，栈底固定，栈顶浮动**，需要用一个专门的硬件计算器作为**栈顶指针**，即**专用寄存器SP**。 将存储器想像成一摞书，地址小的在上面，地址大的在下面。

低地址
——
——
…
高地址

构造堆栈可以有两种方式：

- 自底向上生成（向低地址方向） 进栈：

```
(SP) - 1 -> SP;    //修改栈指针
(A) -> (SP);       //将A中的内容压入栈顶单元
```

出栈：

```
((SP)) -> A;       //将栈顶单元的内容弹出
(SP) + 1 -> SP;    //修改栈指针
```

- 自顶向下生成（向高地址方向）略

4. 十六进制编码格式与通用约定

考试时，机器指令都要以**十六进制**形式进行编码，自行翻译。 PPT中给出的一个指令格式：

操作码	寄存器号	寻址方式	寄存器号	寻址方式
15~12	11~9	8~6	5~3	2~0

地址、取值的助记符，以及寻址方式的约定：

字段代码	寻址方式	地址助记符	含义
000	寄存器直接	R_i	操作数=(R_i)
001	寄存器间接	(R_i)	操作数=((R_i))

运算器

1. 四位先行(快速)进位链的逻辑表达式

对于并行加法器中的第i个全加器： **输入信号**：

- 从低位（i-1）来的进位输入C_{i-1}
- 两个加数A_i和B_i

输出信号：

- 本位和S_i与进位输出C_i

运算的逻辑表达式为：

$S_i = A_i \oplus B_i \oplus C_{i-1}$

$C_i = A_iB_i + (A_i \oplus B_i)C_{i-1}$

如果记$G_i = A_iB_i$（进位产生函数）,$P_i = A_i \oplus B_i$（进位传递函数），就可以很简单地表示每一位上的进位：$C_i = G_i + P_iC_{i-1}$我们用上面的式子多次展开，就可以实现**快速进位加法器**，其四阶的表达式如下：(**十分重要**)

$C_1 = G_1 + P_1C_0$ $C_2 = G_2 + P_2C_1 = G_2 + P_2G_1 + P_2P_1C_0$
$C_3 = G_3 + P_3C_2 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1C_0$
$C_4 = G_4 + P_4C_3 = G_4 + P_4G_3 + P_4P_3G_2 + P_4P_3P_2G_1 + P_4P_3P_2P_1C_0$ …

2. 计算机中的补码运算

补码加减运算非常简单，只要加就行了，具体规则如下：

1. 参与运算的两个操作数均用补码表示
2. 符号位作为数的一部分参加运算
3. 如若加法，直接相加；如若减法，则将被减数与减数的机器负数相加
4. 运算结果仍然是补码

补码运算的直接好处在于可以**简化运算器的设计**。

*EXTRA 符号扩展 比如你想把一个8位数和另外一个32位数相加，就必须将8位数转换成32位数的形式，这就被称为**符号扩展**。

- 原码的符号扩展：符号位移到最高位，多出来的其它位用0填充。
- **补码的符号扩展**：所有多出来的数位用补码的**符号位**填充。

*EXTRA 溢出检测 待续

！**双符号位补码**

3. 浮点数的运算

我们这里只考虑**规格化**浮点数的四则运算问题，其中尾数的基数r=2（2进制数）。 设参与运算的两个非0规格化浮点数分别为：$A = M_A \times 2^{E_A}$ $B = M_B \times 2^{E_B}$

- 浮点数加减运算
  1. 对阶——将两个浮点数的小数点对齐 $E_A = E_B$，无需对阶； $E_A > E_B$，则M_B右移。每右移一位，$E_B + 1 -> E_B$，直到$E_A = E_B$； $E_A < E_B$，则M_A右移。每右移一位，$E_A + 1 -> E_A$，直到$E_A = E_B$；
  2. 尾数加减$M_A \pm M_B \rightarrow M_C$
  3. 尾数结果规格化 规格化的尾数M应当满足$\frac{1}{2} \leq |M| < 1$ **左规&右规** 设尾数采用双符号位补码表示，经过加减运算，会出现以下六种情况： a. 00.1$\times\times...\times$ b. 11.0$\times\times...\times$ c. 00.0 $\times\times...\times$ d. 11.1$\times\times...\times$ e. 01.$\times\times\times...\times$ f. 10.$\times\times\times...\times$
     - 其中，a和b已经符合规格化浮点数的定义，已是规格化数。
     - c和d不是规格化数，需要**尾数左移**以实现规格化，每左移一位，阶码减一，直到**两个符号位和小数点后第一位不同**为止，这就实现了**左规**，左规可以有很多次。将左规条件用符号化语言叙述，即：左规 $= \overline{C_{s1} C_{s2}} \overline{C_1} + C_{s1} C_{s2} C_1$其中，$C_{s1}, C_{s2}$代表尾数$M_c$的两个符号位，$C_1$为$M_C$的最高位。
     - e和f两种情况在**定点加减运算中称为溢出**，但是在浮点运算时，说明此时尾数的绝对值大于1，右移一次即可，并且右移最多只会有一次：右规 $= C_{s1} \oplus C_{s2}$(没看懂)
  4. 舍入 正常尾数的最低位之后的全部数位通常会在运算后直接丢失。
  5. 溢出判断 浮点数的溢出需要根据阶码的符号位来判断。 若阶码也用双符号位补码表示 $[E_C]_{补} = 01.\times\times...$表示上溢，浮点数真正溢出，做溢出中断处理。 $[E_C]_{补} = 10.\times\times...$表示下溢，浮点数值趋近于0，按照机器0处理。

浮点数加减法计算样例可参照书107页。

- 浮点数乘除运算——未完待续

4. 左规、右规

左规、右规 这个在上文中已经提到，不再赘述。

5. 浮点运算器的实现

浮点运算器由**两个定点运算部件**构成，分别完成对**阶码**和**尾数**的处理。

- 阶码运算部件：完成阶码**加**、**减**和尾数规格化时对阶码的调整。
- 尾数运算部件：完成尾数的**四则运算**，判断尾数是否规格化，判断溢出。

注：阶码只需要加减，而尾数需要四则运算。

6. PSW寄存器

（参见书168页，状态标志寄存器） **状态标志寄存器**是CPU内的一个**专用寄存器**，用来存放程序状态字（PSW，Program Status Word），反映**处理器的状态**和**ALU运算结果的某些特征**，以及**控制指令的执行状态**。 各类机器中的状态标志寄存器规定不尽相同，这里给出8086微处理器的例子。 8086的状态标志寄存器有6个状态标志和3个控制标志。 6个状态标志：

- 进位标志位（CF）：
- 辅助进位标志位（AF）；
- 溢出标志位（OF）：
- 零标志位（ZF）；
- 符号标志位（SF）；
- 校验标志位（PF）；

3个控制标志：

- 方向标志（DF），表示串操作指令中字符串操作的方向。
- 终端允许标志位（IF），表示CPU是否能够相应外部的可屏蔽中断请求。
- 陷阱标志位（TF），为了方便程序的调试，使处理器的执行进入单步方式而设置的控制标志位。

7. 原码一位乘法

这里的“一位乘法”并不是说只计算1位二进制数，而是说仿照手写竖式的方式，逐位相乘，再将部分和相加， 在计算机中，由于位宽限制，通常不会完全按照手写的方式设计乘法器，一是因为**加法器内很难实现多个数据同时相加**，二是因为**加法器的位数需要设置为寄存器位数的两倍**才能满足计算需要。 因此，计算机中通常把n位乘法转化为n次**累加和移位**。 具体步骤如下：

1. 参加运算的操作数取其绝对值；
2. 令乘数的最低位为判断位，若为1，加被乘数，若为0，加0；
3. 累加后的部分积以及乘数右移一位；
4. 重复n次第2步和第3步；
5. 符号位单独处理；

原码一位乘法运算器电路需要的部件：

- 3个寄存器
- 1个n+2位加法器
- 1个计数器
- n+2个与门
- 1个异或门

8. 浮点数的溢出判断&处理方法

浮点数运算

CPU

- CPU(Central Processing Unit，中央处理单元)，计算机的核心组成部分，包括运算器和控制器。

1. CPU中的常用寄存器，名称和作用

- CPU中的寄存器主要用来暂存运算和控制过程中的中间结果、最终结果，以及控制、状态的信息。
- 可以分为通用寄存器和专用寄存器两大类。
- 在32位MIPS中，32个寄存器都是通用寄存器

1. 通用寄存器 通用寄存器可以用来存放原始数据和计算结果，常见的有如下几个用途：
    - 变址寄存器
    - 计数器
    - 地址指针
    - 累加寄存器Acc
2. 专用寄存器 专用寄存器专门用来完成某一种特殊的功能，CPU中至少需要5个专用寄存器：
    - 程序计数器（PC）
    - 指令寄存器：用来存放从存储器中取出的指令
    - 存储器数据寄存器：在读写主存时，暂存需要读取/写入的指令或数据
    - 存储器地址寄存器：保存当前CPU所访问的主存单元的地址。由于主存频率慢于CPU，所以需要地址寄存器保存地址信息，直到主存读写操作完成。
    - 状态标志寄存器：存放程序状态字

2. 指令周期

> – 指令周期是指从取指令、分析取数到执行完该指令所需的全部时间。

机器周期又称CPU周期，通常一个指令周期会被划分为若干个机器周期，在每个机器周期，CPU会完成一个基本操作。一般的CPU周期分为下面几个部分：

- 取指周期（短）
- 取数周期（长）
- 执行周期（短）
- 中断周期（长）$\text{指令周期} = i \times \text{机器周期}$

3. 单总线数据通路中，指令运行的基本过程

一条指令的运行过程可以分为3个阶段：**取指令阶段、分析取数阶段、执行阶段**。

1. 取指令阶段 将现行指令从主存中取出来并送到指令寄存器中。 在整个取指令过程中，MAR和MDR充当主存的接口寄存器，最终将指令送到IR中。
    - 将**程序计数器（PC）**中的内容送到**存储器地址寄存器（MAR）**，同时送到**地址总线（DB）**
    - 由控制单元（CU）经过控制总线（CB）向存储器发读命令。
    - 从主存中取出的指令通过**数据总线（DB）**送到存储器数据寄存器（MDR）。
    - 将**MDR**的内容送至**指令寄存器（IR）**中。
    - 将PC的内容递增（PC <- PC + 4）。
2. 分析取数阶段 取出指令后，指令译码器(ID)可以识别和区分出不同的指令类型，并依据不同指令的功能完成对主存信息的读取。
3. 执行阶段 执行阶段完成指令规定的各种操作，形成稳定运算结果，并将其存储起来。 这个过程依据指令的不同而不同。 完成执行阶段任务的时间成为**执行周期**。

更加详细的**指令微操作序列**参见书179页。

4. 转移指令的读取与执行

转移指令：**JC A** 这是一个**条件转移指令**

- 若上次运算结果有进位(C=1)，就转移（PC=PC+A)；

- 若上次运算结果无进位(C=0)，就顺序执行下一条指令。 同样需要三个周期来完成读取和执行。

1. **取指周期**
    - PC_{out}和MAR_{in}有效，$(PC) \rightarrow MAR$
    - 控制总线向主存发出读命令，$Read$
    - 存储器将MAR中的地址所指向单元的内容通过数据总线送至MDR，$M(MAR) \rightarrow MDR$
    - MDR_{out}和IR_{in}有效，$(MDR) \rightarrow IR$，取地址完毕，IR中的指令开始控制CU
    - PC内容“加一（机器字）”，$(PC) + 1 \rightarrow PC$
2. **取数周期**
    - JC指令不需要取数，直接进入执行周期。
3. **执行周期**
    - 如果有进位(C=1)，则完成$(PC) + A \rightarrow PC$的操作
        - PC_{out}和Y_{in}有效（Y是存储中间结果的寄存器），记作：$(PC) \rightarrow Y(C = 1)$
        - $AdIR_{out}$（指令中的地址码字段）和Y_{in}有效，同时CU向ALU发送"ADD"控制信号，结果存入寄存器Z，记作：$Ad(IR) + (Y) \rightarrow Z(C = 1)$
        - Z_{out}和PC_{int}有效，$(Z) \rightarrow PC(C = 1)$
    - 如果没有进位(C=0)，则直接执行下一条指令