计算机组成原理
作业 1

李祥泽
2018011331

1

False.

处理器的频率不仅和指令系统有关, 更和具体的硬件实现和工艺有关. 举例来说, 使用 RISC 指令系统的 *ARM Cortex-A72* 主频是 2.5 GHz, 使用 CISC 指令系统的 *Intel Core i7-880* 主频是 3.06 GHz.

2

True.

由补码乘法的布斯算法可知, 进行补码的乘法, 需要的操作只有: 1. 加法 (包括减法), 2. 移位, 3. 比较两个 bit 并按其结果控制加法器.

3

True.

因为加减交替法不用保存原来的余数, 而保存余数需要使用寄存器.

4

B.

5

D.

6

D.

7

B.

8

A.

9

推导布斯算法

根据补码的定义, 当 $x < 0$ 时, $[x]_{补} = 2^{n+1} + x$. 若将其补码记为 $[x]_{补} = x_{n-1}x_{n-2}\cdots x_1x_0$, 则有:

$$x = -2^{n-1}x_{n-1} + \sum_{i=0}^{n-2} 2^i x_i$$

如此有:

$$\begin{aligned}
[x*y]_{补} &= [x]_{补} * (-2^{n-1}y_{n-1} + \sum_{i=0}^{n-2} 2^i y_i) \\
&= [x]_{补} * [-2^{n-1}y_{n-1} + \sum_{i=0}^{n-2} (2^{i+1} - 2^i) y_i] \\
&= [x]_{补} * [2^{n-1}(y_{n-2} - y_{n-1}) + 2^{n-2}(y_{n-3} - y_{n-2}) + \cdots + 2^0(y_{-1} - y_0)] \qquad (约定\ y_{-1} = 0) \\
&= [x]_{补} * \sum_{i=0}^{n-1} 2^i (y_{i-1} - y_i)
\end{aligned}$$

因此, 可以直接用补码进行乘法运算. 其运算过程是: (假定是 m 位补码 x 与 n 位补码 y 相乘得到 z)

1. 准备一个长度为两个运算数长度之和的部分积寄存器, 要求支持算术右移, 高位 (与被乘数等长) 初始化为 0, 低位 (与乘数等长) 初始化为乘数; 另需一个长度与被乘数相等的加法器, 左操作数为部分积的高 m 位, 结果也更新到结果寄存器的高 m 位; 在部分积最低位之外还需有一位附加位 , 初始化为 0.

   说明: 这里, 部分积寄存器的低位 (暂时) 表示 $y_{n-1}\ldots y_0$, 而附加位 (暂时) 表示 y_{-1}, 高位 (暂时) 表示 $z_{m-1}\ldots z_0$. 经过下面所述的 k 步计算之后 (每次会导致结果寄存器右移 1 位), 其高 $m+k$ 位表示 $z_{m+k-1}\ldots z_0$, 低 $n-k$ 位表示 $y_{n-1}\ldots y_k$, 附加位表示 y_{k-1}.

2. 做 `附加位 - 结果最低位`, 根据其结果做:

   - `+1: 部分积的高 m 位 += 被乘数`
   - `0 : pass`
   - `-1: 部分积的高 m 位 -= 被乘数`

   说明: 这里实现的是 $[x]_{补} * 2^i(y_{i-1} - y_i)$.

3. 将部分积寄存器算术右移 1 位, 最右侧溢出的位移入附加位. 如果已经移了 n 次, 乘法结束, 转 **4**; 否则, 转 **2**.

   说明: 这相当于 `i++`.

4. 此时部分积寄存器的值就是结果.

计算 $3 \times (-7)$

在 5 位补码下, 3 为 `00011`, -7 为 `11001`. 结果为 10 位.

```
0:(ori) 00000 11001 0
1: (-x) 11101 11001 0
  (sra) 11110 11100 1
2: (+x) 00001 11100 1
  (sra) 00000 11110 0
3:  (0) 00000 11110 0
  (sra) 00000 01111 0
4: (-x) 11101 01111 0
  (sra) 11110 10111 1
5:  (0) 11110 10111 1
  (sra) 11111 01011 1
```

其结果为 `1111101011`, 即 `-21`; 如果结果截断为相同位数, 为 `01011 = 11`.

10

根据 $2^{r-1} \geq k + r$, 4 位数据需要 4 位校验位. 排列为

$$p_1 p_2 d_1 p_3 d_2 d_3 d_4 p_4$$

生成方法是

$$\begin{aligned}
p_1 &= d_1 \oplus d_2 \oplus d_4 &&= 0 \\
p_2 &= d_1 \oplus d_3 \oplus d_4 &&= 1 \\
p_3 &= d_2 \oplus d_3 \oplus d_4 &&= 0 \\
p_4 &= d_1 \oplus d_2 \oplus d_3 \oplus d_4 \oplus p_1 \oplus p_2 \oplus p_3 &&= 0
\end{aligned}$$

即, 实际传输的数据是 `01100110`. 按题设, 误传为 `01100010`, 记为 $p'_1 p'_2 d'_1 p'_3 d'_2 d'_3 d'_4 p'_4$. 纠错过程为:

$$\begin{aligned}
s_1 &= p'_1 \oplus d'_1 \oplus d'_2 \oplus d'_4 &&= 0 \\
s_2 &= p'_2 \oplus d'_1 \oplus d'_3 \oplus d'_4 &&= 1 \\
s_3 &= p'_3 \oplus d'_2 \oplus d'_3 \oplus d'_4 &&= 1 \\
s_4 &= d'_1 \oplus d'_2 \oplus d'_3 \oplus d'_4 \oplus p'_1 \oplus p'_2 \oplus p'_3 \oplus p'_4 &&= 1
\end{aligned}$$

由 $s_4 = 1$ 知是 1 位出错; 由 $s_1 s_2 s_3 = 011$ 知出错的位不参与生成 s_1, 但参与生成 s_2 和 s_3. 所以出错的是 d'_3.