3170105932 杨钊

1.5、(a). P1的每秒指令数：$3\times10^9 \div 1.5 = 2\times10^9 = 2000$ MIPS

P2：$2.5\times10^9 \div 1 = 2.5\times10^9 = 2500$ MIPS

P3：$4\times10^9 \div 2.2 \approx 1818$ MIPS

∴ 处理器P2有最快的指令执行速率

(b) 10秒钟，P1时钟周期数：$N_{T_1} = 3\times10^9\times10 = 3\times10^{10}$

P1执行指令数：$N_{I_1} = 2\times10^9\times10 = 2\times10^{10}$

10秒钟，P2时钟周期数：$N_{T_2} = 2.5\times10^9\times10 = 2.5\times10^{10}$

P2执行指令数：$N_{I_2} = 2.5\times10^9\times10 = 2.5\times10^{10}$

10秒钟，P3时钟周期数：$N_{T_3} = 4\times10^9\times10 = 4\times10^{10}$

P3执行指令数：$N_{I_3} = 1.818\times10^9\times10 = 1.818\times10^{10}$

(c) $CPI_{(P1)} = 1.5\times1.2 = 1.8$，$CPI_{(P2)} = 1.2$，$CPI_{(P3)} = 2.2\times1.2 = 2.64$

$Time_{(new)} = Time_{(old)}\times0.7 = 7s$

则 $Time(P_1) = 2\times10^{10}\times1.8\div7 = 5.14$ GHz

$Time(P_2) = 2.5\times10^{10}\times1.2\div7 = 4.29$ GHz

$Time(P_3) = 1.818\times10^{10}\times1.2\div7 = 6.86$ GHz

1.6、对于P1而言，执行该程序耗时：

$T_1 = t_{A1} + t + t_{C1} + t_{D1}$

$= (10^5 + 2\times10^{\ }2 + 5\times10^5\times3 + 2\times10^5\times3)\div(2.5\times10^9) = 1.04$ ms

对于P2而言，执行该程序耗时：

$T_2 = t_{A2} + t_{B2}\ \ t_{C2} + t_{D2}$

$= (10^5\times2 + 2;\ ^5\times2 + 5\times10^5\times2 + 2\times10^5\times2)\div(2.5\times10^9) = 0.8$ ms

所以P2架构实[illegible]程序更快

(a). $CPI_{(P1)} = 2.6$ $^6/10^6 = 2.6$

$CPI_{(P2)}$ 2×1 $/10^6 = 2$

(b) P1所需时钟[illegible]期：$N1 = 2.6\times10^6$

P2为N2 2;^{6}

1.14、运行该程序总时 $_{total} = (50\times10^6 + 110\times10^6 + 80\times10^6\times4 + 16\times10^6\times2)\div(2\times10^9)$

$= 256$ ms

(1) FP指令部分 T_{FP} $50\times10^6\div(2\times10^9) = 25$ ms $< \frac{256}{2}$ ms

∴ 仅通过优化FP指令无法使程序运行速度达到原来2倍

(2) 原本 $T_{L/S} = 80\times10^6\div(2\times10^9) = 160$ ms

要使 $T'_{total} = \frac{1}{2}T_{total}$，则 $T'_{L/S} = 160\text{ms} - 128\text{ms} = 32$ms

则 $CPI_{(L/S)} = 2\times10^9\times32\times10^{-3}\div(80\times10^6) = 0.8$，即L/S指令的CPI缩减为原来的20%

(3) $CPI_{(FP)} = CPI_{(INT)} = 0.6$，$CPI_{(L/S)} = 2.8$，$CPI_{(Branch)} = 1.4$

$T_{total} = (50\times10^6\times0.6 + 110\times10^6\times0.6 + 80\times10^6\times2.8 + 16\times10^6\times1.4)\div(2\times10^9) = 171.2$ ms

运行时间改善了 $\frac{256-171.2}{256}\times100\% = 33.13\%$

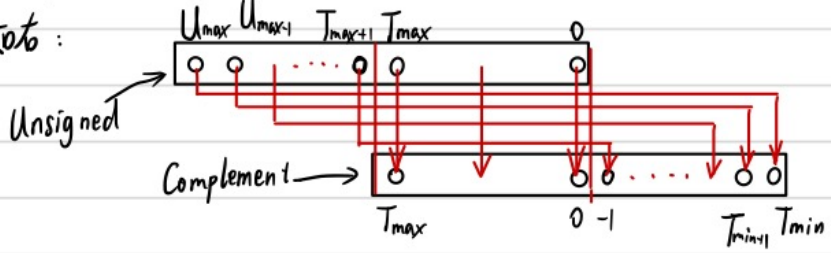
1、首先有 $B2U(x)=\sum_{i=0}^{w-1}x_i\cdot 2^i$，$B2T(x)=-x_{w-1}\cdot 2^{w-1}+\sum_{i=0}^{w-2}x_i\cdot 2^i$

∴ $B2U(x)-B2T(x)=x_{w-1}\cdot 2^w$

∴ $U2T=B2T(U2B)=x_{w-1}\cdot 2^w-B2U(U2B)=x_{w-1}\cdot 2^w-x$

即 $U2T(x)=x_{w-1}\cdot 2^w-x$

U→T对照图如右：



2、(a).① 0b10010011 = 147 = 0x93

② 63 = 0b00111111 = 0x3F

③ 0b00100100 = 36 = 0x24

④ 0 = 0b00000000 = 0x00

⑤ 39 = 0b00100111 = 0x27

⑥ 437 = 0b0001101101 = 0x1B5

⑦ 0x0123 = 0b0000 0001 0010 0011 = 291

(b).① 0xD3AD = 0b1101 0011 1010 1101

② 0xB33F = 0b1011 0011 0011 1111

③ 0x7EC4 = 0b0111 1110 1100 0100

(c).① $2^{16}=2^6$ Ki = 64Ki ② $2^{34}=2^4$ Gi = 16Gi

③ $2^{27}=2^7$ Mi = 128Mi ④ $2^{61}=2$ Ei

⑤ $2^{43}=2^3$ Ti = 8Ti ⑥ $2^{47}=2^7$ Ti = 128Ti

⑦ $2^{36}=2^6$ Gi = 64Gi ⑧ $2^{58}=2^8$ Pi = 256 Pi

(d).① 2Ki = 2^{11} ② 256Pi = 2^{58} ③ 512Ki = 2^{19}

④ 64Gi = 2^{36} ⑤ 16Mi = 2^{24} ⑥ 128Ei = 2^{67}

3、(a). 对于数x，实现3x可以通过：x<<1 +x

(b). 对于数x，实现7x可通过：x<<3 - x

-x可以通过补码实现，即 x<<3 + (-x)

(c). 对于数x，实现x/3可以通过：$\sum_{i=1}^{15}(x<<(2i))$

对于等比数列：$\frac{1}{4}$，$\frac{1}{16}$，$\frac{1}{64}$，……

$S_n=\lim_{n\to\infty}\sum_{i=1}^{\infty}\frac{1}{4^i}=\frac{1}{3}$

∴ $\frac{1}{3}x=\frac{1}{4}x+\frac{1}{16}x+\frac{1}{64}x+\cdots=x<<2+x<<4+x<<6\cdots\cdots$

由于int型为32位

∴ $\frac{1}{3}x\approx x<<2+x<<4+x<<6+\cdots\cdots+x<<30$

4、(a). $x = 0b\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1010$

$ux = 0b\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$

$y = 0b\ 0|000\ 00|\ 0010|\ 001|\ |001|\ |001|\ |001|001|$

$z = 0b\ |100\ 0000\ 0011|\ 1111|\ 1011|\ 0011|001|\ 0011|001|\ 0011|001|\ 0011|\ 0011|\ 0011|001|$

(b) $x \cdot y$ 时，float型的 x 会自动转换成 double 以便与double 的 y 运算

则 $(double)x = 0b0100\_0000\_0010\_0101\_0011\_0011\_0011\_0011\_0010\_0000\_0000\_0000\_0000\_0000\_0000\_0000$

$x \cdot y = 0b\ 1100\_0000\_0110\_1111\_1111\_1111\_0111\_1000\_0101\_0001\_1110\_1100\_0000\_0000\_0000\_0000$

而 z 为 float 型

$\therefore z = x \cdot y = 0b1100\ 0111\_1011\_1111\_1111\_1011\_1100\_0010$

(c) $x = (-1)^{S_1} M_1 \cdot 2^{E_1}$，$y = (-1)^{S_2} M_2 \cdot 2^{E_2}$

x，y 二进制见 (b)中(double)x 与 (a)中z

$(-1)^{S_2} M_2$ ← $E_2 - E_1$ →

\+ $(-1)^{S_1} M_1$

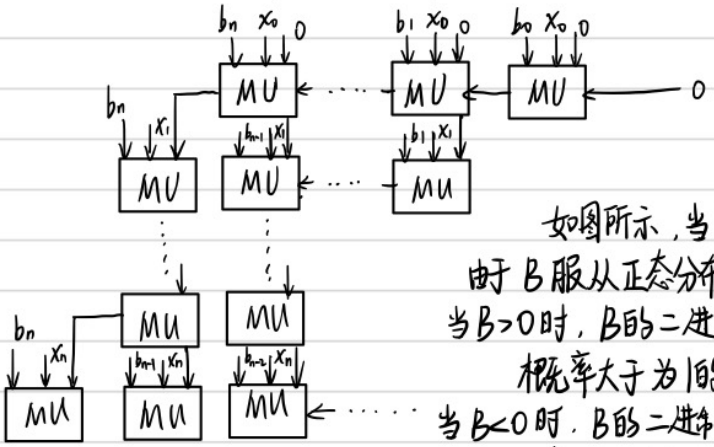
$x + y = (double)x + y = 2^3 \times (M_1 - 2M_2)$

其中 $M_1 = (1.0101\_0011\_0011\_0011\_0010)_2$

$M_2 = (1.1111\_1011\_0011\_0011\_0011\_0011\_0011\_0011\_0011\_0011\_0011\_0011)_2$

$\therefore z = (float)(x+y) = -21.1$

5、首先应该使用二进制数的补码形式进行运算



如图所示，当 $b_i = 0$ 时，$x_j \cdot b_i$ 无功耗

由于 B 服从正态分布，所以 B 大概率在 0 附近

当 $B > 0$ 时，B 的二进制原码与补码为 0 的位数概率大于为 1 的位数

当 $B < 0$ 时，B 的二进制原码为 1 的位数概率大于为 0 位数，但补码中位数为 0 概率大于位数为 1.

如 $-1 = 0b\ 1111\_1111\_1111\_1111\_1111\_1111\_1111\_1111$ 原码

$-1 = 0b1000\_0000\_0000\_0000\_0000\_0000\_0000\_0001$ 补码

在补码基础上，可以通过 Booth 编码的方法减少部分积数量从而优化 MAC

将乘数按三位一组进行划分，相互重叠一位，每一组按下表编码，并形成一个部分积

Y_n	Y_{n+1}	操作
0	0	+0，右移一位
0	1	+[X]补，右移一位
1	0	+[-X]补，右移一位
1	1	+0，右移一位

二进制补码 $y=-2^{n-1}y_{n-1}+2^{n-2}y_{n-2}+\cdots+y_0$ ……①

①式可等效于：$y=2^{n-1}(-y_{n-1}+y_{n-2})+2^{n-2}(-y_{n-2}+y_{n-3})+\cdots+(-y_0+y_{-1})$ ……②

Booth编码采用②式的形式，将多个部分积之和转化成只需计算首尾两个部分积，大大减小部分积数目，从而实现对乘加器的优化

具体步骤如下：1、被乘数X与乘数B均以补码形式参加乘法运算，结果是积的补码

2、部分积和被乘数X采用双符号位，乘数B采用单符号位

3、初始部分积为0，运算前，在乘数B的补码末位添加一位B_{n+1}，初值为0

4、根据Y_n，Y_{n+1}的值，按照上表进行累加右移操作，右移时遵循补码移位规则

5、累加n+1次，右移n次，最后一次不右移

具体硬件框架如下所示：

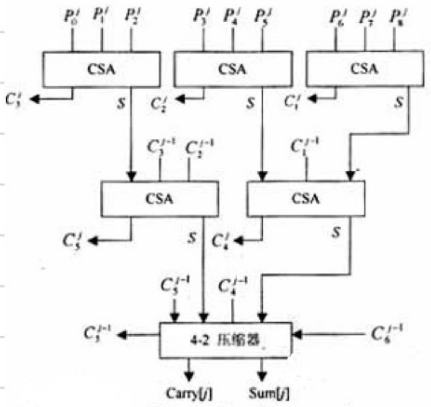


图3 Wallace 树

例如：

操作	乘数	操作	部分积
0	1001 1111		0000 0000 1000 1110
1	1001 1111	00	0000 0000 0100 0111
2		10	- 1001 1111
			0110 0001 0100 0111
			0011 0000 1010 0011
3		11	0001 1000 0101 0001
4		11	0000 1100 0010 1000
5		01	+1001 1111
			1010 1011 0010 1000
			1101 0101 1001 0100
6		00	1110 1010 1100 1010
7		00	1111 0101 0110 0101
8		10	- 1001 1111
			0101 0110 0110 0101