Computer Organization(H)
Southern University of Science and Technology
Mengxuan Wu
12212006

Theory Assignment 2

Mengxuan Wu

Problem 1

a)

The weighted average CPI for two implementations are as follows:

$$
\begin{aligned}
CPI_1 =& 1 \times 10\% + 2 \times 20\% + 3 \times 50\% + 3 \times 20\% = 2.6 \\
CPI_2 =& 2 \times 10\% + 2 \times 20\% + 2 \times 50\% + 2 \times 20\% = 2
\end{aligned}
$$

b)

The clock cycles for the two implementations are as follows:

$$
\begin{aligned}
Cycles_1 =& 1.0 \times 10^6 \times 2.6 = 2.6 \times 10^6 \\
Cycles_2 =& 1.0 \times 10^6 \times 2 \quad = 2 \times 10^6
\end{aligned}
$$

c)

The CPU time for the two implementations are as follows:

$$
\begin{aligned}
Time_1 =& \frac{2.6 \times 10^6}{2.5 \times 10^9} \approx 1.04 \times 10^{-3} s \\
Time_2 =& \frac{2.0 \times 10^6}{3.0 \times 10^9} \approx 0.67 \times 10^{-3} s
\end{aligned}
$$

Hence, the second implementation is faster.

Problem 2

a)

The value of `x30` after the addition is `0x5000000`. An overflow does occur.

Since the values stored in registers considered as signed integers, `0x8000000` and `0xD000000` are both negative numbers. The sum of two negative numbers should be negative, but the result is positive. Therefore, an overflow occurs.

b)

The value of `x30` after the subtraction is `0xB0000000`. It is the direct result.

As we subtract a smaller negative number from a larger negative number, the result should be negative. And the result is negative, so no overflow occurs, and the result is correct.

Problem 3

a)

For an 8-bit signed integer, the range is -2^7 to $2^7 - 1$, or -128 to 127.

The result of $23 + 112 = 135$, which is out of the range, an overflow occurs. Since we are using saturating arithmetic, the result should be the maximum value of the range, which is 127.

b)

The result of $23 - 112 = -89$, which is not out of the range. Then no further action is needed.

Problem 4

$$62_{16} = 01100010_2 \qquad 14_{16} = 00010100_2$$

Iteration	**Multiplicand**	**Product**	**Operation**
0	01100010	00000000_00010100	Initialization
1	01100010	00000000_00001010	Shift right
2	01100010	00000000_00000101	Shift right
3	01100010	01100010_00000101	Add multiplicand
	01100010	00110001_00000010	Shift right
4	01100010	00011000_10000001	Shift right
5	01100010	01111010_10000001	Add multiplicand
	01100010	00111101_01000000	Shift right
6	01100010	00011110_10100000	Shift right
7	01100010	00001111_01010000	Shift right
8	01100010	00000111_10101000	Shift right

The result is $62_{16} \times 14_{16} = 7A8_{16} = 0111\_1010\_1000_2$.

Problem 5

$$62_{10} = 111110_2 \qquad 21_{10} = 010101_2$$

Iteration	Divisor	Remainder	Quotient	Operation
0	010101_000000	000000_111110	000000	Initialization
1	010101_000000	101011_011110	000000	Subtract divisor
	010101_000000	000000_111110	000000	Restore, shift 0 to quotient
	001010_100000	000000_111110	000000	Divisor shift right
2	001010_100000	110110_011110	000000	Subtract divisor
	001010_100000	000000_111110	000000	Restore, shift 0 to quotient
	000101_010000	000000_111110	000000	Divisor shift right
3	000101_010000	111011_101110	000000	Subtract divisor
	000101_010000	000000_111110	000000	Restore, shift 0 to quotient
	000010_101000	000000_111110	000000	Divisor shift right
4	000010_101000	111110_010110	000000	Subtract divisor
	000010_101000	000000_111110	000000	Restore, shift 0 to quotient
	000001_010100	000000_111110	000000	Divisor shift right
5	000001_010100	111111_101010	000000	Subtract divisor
	000001_010100	000000_111110	000000	Restore, shift 0 to quotient
	000000_101010	000000_111110	000000	Divisor shift right
6	000000_101010	000000_010100	000000	Subtract divisor
	000000_101010	000000_010100	000001	Shift 1 to quotient
	000000_010101	000000_010100	000000	Divisor shift right
7	000000_010101	111111_111111	000001	Subtract divisor
	000000_010101	000000_010100	000010	Restore, shift 0 to quotient
	000000_001010	000000_010100	000010	Divisor shift right

The result is $62 \div 21 = 2_{10} = 00000010_2$ with a remainder of $20_{10} = 00010100_2$.

Problem 6

a)

The number can be decomposed as follows:

Number	Sign	Exponent	Fraction
`0x0C000000`	0	$11000_2 = 24_{10}$	0

The number is $(-1)^0 \times (1+0) \times 2^{24-127} = 2^{-103}$.

b)

$$63.25_{10} = 111111.01_2 = 1.1111101_2 \times 2^5 = 1.1111101_2 \times 2^{132-127}$$

Hence, the single precision floating point representation is `0_10000100_11111010000000000000000` or `0x427D0000`.