Chapter 3. Sequential Multiplication/Division

§1. Shift Operations

In hardware, shifting is a very convenient and low-cost operation and it is very useful in many algorithms for multiplication and division. However, to perform shifting operations, one need to view a representation as an infinite sequence.

Given a representation $X = (x_{n-1}, \ldots, x_0)$, we have its infinite extension can be obtained as follows.

- The S-m method:
$$\cdots, 0, 0, \{x_{n-2}, \cdots, x_0\}.0, 0, \cdots$$
- The radix complement method:
$$\cdots, x_{n-1}, x_{n-1}, \{x_{n-1}, \cdots, x_0\}.0, 0, \cdots$$
- The diminished-radix complement method:
$$\cdots, x_{n-1}, x_{n-1}, \{x_{n-1}, \cdots, x_0\}.x_{n-1}, x_{n-1}, \cdots$$

Example 1

- *Let $X = 1011_2$ be represented with s-m method. Then a representation of X with $k = 6$ and $m = 2$ is $X = 001011.00_2$.*
- *Let $X = 1011_2$ be represented with 2's complement method. Then a representation of X with $k = 6$ and $m = 2$ is $X = 111011.00_2$.*
- *Let $X = 1011_2$ be represented with 1's complement method. Then a representation of X with $k = 6$ and $m = 2$ is $X = 111011.11_2$.*

§2. Multiplication

Let the multiplier and the multiplicand be denoted by X and A, respectively, with the following sequence of digits
$$X = x_{n-1} \ldots, x_0, \quad A = a_{n-1} \ldots, a_0,$$

where x_{n-1} and a_{n-1} are the sign digits. Let P be the product of X and A,

$$P = X \cdot A = (p_{2n-1}p_{2n-2}\cdots p_0), \text{ where } p_{2n-1} \text{ is the sign bit.}$$

§2.1. Sequential Multiplication algorithm

In the following we will introduce sequential multiplication algorithm in 2's complement representation. Sequential multiplication using other signed representation methods, such like sign-magnitude and 1's complement method, can be derived in a similar manner. For a two's complement number X, we have

$$X = (x_{n-1}\ldots, x_0) = -x_{n-1}2^{n-1} + \tilde{X},$$

where

$$\tilde{X} = (x_{n-2}\ldots, x_0) = \sum_{i=0}^{n-2} x_i 2^i.$$

Let $\tilde{X} = (x_{n-2}\ldots, x_0)$, then we have the following multiplication algorithm for $U = \tilde{X} \cdot A$.

Algorithm 1 Multiplication of $\tilde{X}$ and A

Input: $\tilde{X}$ and A
Output: $2^{-(n-1)}\tilde{X} \cdot A$
1: $P^{(0)} = 0$;
2: **for** $j = 0$ To $n-2$ **do**
3: $\quad P^{(j+1)} = (P^{(j)} + x_j \cdot A)2^{-1}$
4: **end for**
5: $P^{(n-1)}$ is the output.

A proof of the correctness of the algorithm:

$$\begin{aligned}
P^{(n-1)} &= (P^{(n-2)} + x_{n-2}\cdot A)2^{-1} \\
&= ((P^{(n-3)} + x_{n-3}\cdot A)2^{-1} + x_{n-2}\cdot A)2^{-1} \\
&\vdots \\
&= ((x_0 2^{-(n-1)} + x_1 2^{-(n-2)} + \cdots + x_{n-3}2^{-2} + x_{n-2}2^{-1})A \\
&= 2^{-(n-1)}\sum_{i=0}^{n-2} x_i 2^i A \\
&= 2^{-(n-1)}\tilde{X}A.
\end{aligned}$$

$\square$

Case 1. If $X \geqslant 0$, then $x_{n-1} = 0$. The product of $X \cdot A$ can be obtain as

$$2^{n-1} P^{(n-1)} = \tilde{X} \cdot A = X \cdot A.$$

Case 2. If $X < 0$, then $x_{n-1} = 1$. The it follows

$$X \cdot A = (-2^{n-1} + \tilde{X}) \cdot A = -2^{n-1} A + \tilde{X} \cdot A = 2^{n-1}(-A + P^{(n-1)}).$$

So a final correction step is needed:

$$P^{(n-1)} + (-A).$$

Algorithm 2 shows Algorithm 1 after incorporating the final correction step.

Algorithm 2 Multiplication of X and A

Input: X and A
Output: $2^{-(n-1)} X \cdot A$

Call Algorithm 1;
if $x_{n-1} = 1$ **then**
    $P^{(n-1)} = P^{(n-1)} + (-A)$;
end if
$P^{(n-1)}$ is the output.

Example 2 *The multiplier is positive and the multiplicand A is negative, both in the two's complement representation. Let $X = 3$ and $A = -5$, and then the product can be obtained by using Algorithm 1 as shown in follows.*

A		*1*	*0*	*1*	*1*				*−5*
X	$\times$	*0*	*0*	*1*	*1*				*3*
$P^{(0)} = 0$		*0*	*0*	*0*	*0*				
$x_0 = 1 \Rightarrow$ *Add* A	$+$	*1*	*0*	*1*	*1*				
		1	*0*	*1*	*1*				
Shift to get $P^{(1)}$		*1*	*1*	*0*	*1*	*1*			
$x_1 = 1 \Rightarrow$ *Add* A	$+$	*1*	*0*	*1*	*1*				
		1	*0*	*0*	*0*	*1*			
Shift to get $P^{(2)}$		*1*	*1*	*0*	*0*	*0*	*1*		
$x_2 = 0 \Rightarrow$ *Shift to get* $P^{(3)}$		*1*	*1*	*1*	*0*	*0*	*0*	*1*	
$x_3 = 0 \Rightarrow$ *output* $P^{(3)}$		*1*	*1*	*1*	*0*	*0*	*0*	*1*	$= 2^{-3} \times (-15)$

Example 3 *The multiplier is negative and the operands are in the two's complement form. Let $X = -3$ and $A = -5$, in n-bit 2's complement form, where $n = 4$.*

A		*1*	*0*	*1*	*1*				-5
X	$\times$	*1*	*1*	*0*	*1*				-3
$P^{(0)} = 0$		*0*	*0*	*0*	*0*				
$x_0 = 1 \Rightarrow$ *Add A*		*1*	*0*	*1*	*1*				
		1	*0*	*1*	*1*				
Shift to get $P^{(1)}$		*1*	*1*	*0*	*1*	*1*			
$x_1 = 0 \Rightarrow$ *Shift to get* $P^{(2)}$		*1*	*1*	*1*	*0*	*1*	*1*		
$x_2 = 1 \Rightarrow$ *Add A*	$+$	*1*	*0*	*1*	*1*				
		1	*0*	*0*	*1*	*1*	*1*		
Shift to get $P^{(3)}$		*1*	*1*	*0*	*0*	*1*	*1*	*1*	
$x_3 = 1 \Rightarrow$ *Correction* $($*Add* $-A)$	$+$	*0*	*1*	*0*	*1*				
Get $P^{(3)}$		*0*	*0*	*0*	*1*	*1*	*1*	*1*	$= 2^{-3} \times 15$

§2.2. Sequential Multiplier (optional)

Can you design a circuit architecture that performs Algorithm 1?

§3. Division

Given a dividend X and divisor D, a quotient Q and a remainder R have to be calculated so as to satisfy

$$X = Q \cdot D + R \text{ with } R < D \text{ and } D \neq 0. \tag{1}$$

In most fixed-point arithmetic structures, we assume the size of the operands as follows. For multiplication operation,

(multiplier: single-length) $\times$ (multiplicand: single-length) $=$ (product: double-length).

While for division operation we have

(dividend: double-length) $=$ (quotient: single-length)$\times$(divisor: single-length)$+$(remainder: single-length).

There are two main types of division algorithms, restoring and non-restoring. In the following we introduce only the restoring division algorithm.

§3.1. Sequential Restoring Division

For simplicity we assume that

(i). X, D, Q and R are non-negative numbers.

(ii). Let $X < D$ so that Q is always a fraction and has a form of $Q = 0.q_1q_2 \cdots q_m$, where $m = n - 1$.

Algorithm 3 Sequential Division 1 (Restoring)

Input: X and D, $D > X \geqslant 0$.
Output: $Q = 0.q_1 \cdots q_m$ and $r_m = 2^m \cdot R$

$r_0 = X$;
for $i = 1$ To m **do**
    IF $2r_{i-1} > D$ THEN $q_i = 1$ ELSE $q_i = 0$
    $r_i = 2r_{i-1} - q_i \cdot D$
end for
The output are $Q = 0.q_1q_2 \cdots q_m$ and $r_m = 2^m \cdot R$.

The most complicated step in the above division algorithm is the comparison between $2r_{i-1}$ and D. The output of r_m can be verified as follows.

$$\begin{aligned} r_m &= 2r_{m-1} - q_m \cdot D \\ &= 2(2r_{m-2} - q_{m-1} \cdot D) - q_m \cdot D \\ &\vdots \\ &= 2^m r_0 - (q_m + 2q_{m-1} + \cdots + 2^{m-1}q_1) \cdot D \end{aligned}$$

Since $r_0 = X$ it follows

$$\begin{aligned} r_m \cdot 2^{-m} &= X - (q_1 2^{-1} + q_2 2^{-2} + \cdots + q_m \cdot 2^{-m}) \cdot D \\ &= X - Q \cdot D \\ &= R \end{aligned}$$

Example 4 *(restoring) Let $X = (0.100000)_2 = 1/2$ and $D = (0.110)_2 = 3/4$. Since $X < D$, we have*

$r_0 = X$			0	.1	0	0	0	0	0	
$2r_0$		0	1	.0	0	0	0	0		
Add $-D$	+	1	1	.0	1	0				
$r_1 = 2r_0 - D \geq 0$		0	0	.0	1	0	0	0		*set* $q_1 = 1$
$2r_1$		0	0	.1	0	0	0			
Add $-D$	+	1	1	.0	1	0				
$2r_1 - D < 0$		1	1	.1	1	0	0			*set* $q_2 = 0$
$r_2 = 2r_1$		0	0	.1	0	0	0			
$2r_2$		0	1	.0	0	0				
Add $-D$	+	1	1	.0	1	0				
$r_3 = 2r_2 - D \geq 0$		0	0	.0	1	0				*set* $q_3 = 1$

The final results are $Q = (0.101)_2 = 5/8$ and $r_3 = 1/4 \Rightarrow R = r_m 2^{-m} = r_3 2^{-3} = 1/32$.

When the operands are integers, we can verify that the condition for performing division using Algorithm 3 is that $X < 2^{n-1}D$ where n denotes single-length. In this case, $Q = q_1 \cdots q_m$ is an $m = n-1$ bits integer and the remainder $R = r_{n-1}$.

Algorithm 4 Sequential Division 2(Restoring)

Input: X and D, $2^{n-1}D > X \geqslant 0$.
Output: $Q = q_1 \cdots q_m$ and $r_m = R$

1: $r_0 = X$;
2: **for** $i = 1$ To m **do**
3: IF $2r_{i-1} > D$ THEN $q_i = 1$ ELSE $q_i = 0$
4: $r_i = 2r_{i-1} - q_i \cdot D$
5: **end for**
6: The output are $Q = q_1 q_2 \cdots q_m$ and $r_m = R$.

Example 5 *(restoring) Let $X = (0100000)_2 = 32$ and $D = (0110)_2 = 6$. Since $X < 2^{n-1}D$, we have*

$$
\begin{array}{llccccc|cccl}
r_0 = X & & & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \\
2r_0 & & 0 & 1 & 0 & 0 & 0 & 0 & 0 & & \\
\mathit{Add}\ -D & + & 1 & 1 & 0 & 1 & 0 & & & & \\
\hline
r_1 = 2r_0 - D > 0 & & 0 & 0 & 0 & 1 & 0 & 0 & 0 & & \textit{set } q_1 = 1 \\
2r_1 & & 0 & 0 & 1 & 0 & 0 & 0 & & & \\
\mathit{Add}\ -D & + & 1 & 1 & 0 & 1 & 0 & & & & \\
\hline
2r_1 - D < 0 & & 1 & 1 & 1 & 1 & 0 & 0 & & & \textit{set } q_2 = 0 \\
r_2 = 2r_1 & & 0 & 0 & 1 & 0 & 0 & 0 & & & \\
2r_2 & & 0 & 1 & 0 & 0 & 0 & & & & \\
\mathit{Add}\ -D & + & 1 & 1 & 0 & 1 & 0 & & & & \\
\hline
r_3 = 2r_2 - D > 0 & & 0 & 0 & 0 & 1 & 0 & & & & \textit{set } q_3 = 1
\end{array}
$$

The final results are $Q = (101)_2 = 5$ and $R = r_3 = 2$.

§3.2. Sequential Restoring Divider (optional)

Can you design a circuit architecture that performs restoring division algorithm (Algorithms 3 or 4)?