Assignment3

3.9

Assume 151 and 214 are signed 8-bit decimal integers stored in two’s complement format. Calculate 151 + 214 using saturating arithmetic. The result should be written in decimal. Show your work.

$$151 = 1001\ 0011$$
$$214 = 1101\ 0110$$

Rewrite using 2's complement representation

$$151 = 0110\ 1100 = -105$$
$$214 = 0010\ 1001 = -42$$

So, in calculation

$$-105 + (-42) = -147$$

$$\begin{array}{r} 0110\ 1100 \\ +\ \ 0010\ 1001 \\ - - - - - - - - - \\ 1001\ 0011 \end{array}$$

This result is overflow, so by using saturating arithmetic, **the result will be save as -128**

3.10

Assume 151 and 214 are signed 8-bit decimal integers stored in two’s complement format. Calculate 151 - 214 using saturating arithmetic. The result should be written in decimal. Show your work.

$$151 = 1001\ 0011$$
$$214 = 1101\ 0110$$

Rewrite using 2's complement representation

$$
\begin{gathered}
151 = 0110\ 1100 = -105 \\
214 = 0010\ 1001 = -42
\end{gathered}
$$

So, in calculation

$$-105 - (-42) = -63$$

$$
\begin{array}{r}
0110\ 1100 \\
-\ \ 0010\ 1001 \\
-\ -\ -\ -\ -\ -\ -\ - \\
0100\ 0011
\end{array}
$$

This result is not overflow, so by using saturating arithmetic, **the result will be save as -63**

3.11

Assume 151 and 214 are unsigned 8-bit integers. Calculate 151 + 214 using saturating arithmetic. The result should be written in decimal. Show your work.

$$
\begin{gathered}
151 = 1001\ 0011 \\
214 = 1101\ 0110
\end{gathered}
$$

$$
\begin{array}{r}
1001\ 0011 \\
+\ \ 1101\ 0110 \\
-\ -\ -\ -\ -\ -\ -\ - \\
1\ 0110\ 1101
\end{array}
$$

This result is overflow, so by using saturating arithmetic, **the result will be save as 255**

3.13

Using a table similar to that shown in Figure 3.6, calculate the product of the hexadecimal unsigned 8-bit integers 62 and 12 using the hardware described in Figure 3.5. You should show the contents of each register on each step.

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 $\Longrightarrow$ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 $\Longrightarrow$ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 $\Longrightarrow$ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 $\Longrightarrow$ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

FIGURE 3.6 Multiply example using algorithm in Figure 3.4. The bit examined to determine the next step is circled in color.

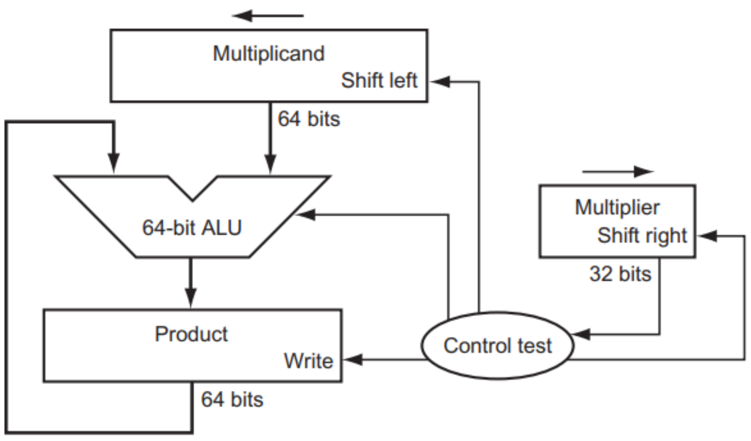


FIGURE 3.3 First version of the multiplication hardware. The Multiplicand register, ALU, and Product register are all 64 bits wide, with only the Multiplier register containing 32 bits. (Appendix B describes ALUs.) The 32-bit multiplicand starts in the right half of the Multiplicand register and is shifted left 1 bit on each step. The multiplier is shifted in the opposite direction at each step. The algorithm starts with the product initialized to 0. Control decides when to shift the Multiplicand and Multiplier registers and when to write new values into the Product register.

$$62 \times 12$$

$$\begin{aligned} 62_{16} &= 0110\ 0010_2 \\ 12_{16} &= 0001\ 0010_2 \end{aligned}$$

Iteration	Step	Multiplicand	Product
0	Initialize	01100010	00000000|00010010
1	Rightmost bit of the product = 0	01100010	00000000|00010010
1	Shift product right	01100010	00000000|00001001
2	Rightmost bit of the product = 1	01100010	01100010|00001001
2	Shift product right	01100010	00110001|00000100
3	Rightmost bit of the product = 0	01100010	00110001|00000100
3	Shift product right	01100010	00011000|10000010
4	Rightmost bit of the product = 0	01100010	00011000|10000010
4	Shift product right	01100010	00001100|01000001
5	Rightmost bit of the product = 1	01100010	01101110|01000001
5	Shift product right	01100010	00110111|00100000
6	Rightmost bit of the product = 0	01100010	00110111|00100000
6	Shift product right	01100010	00011011|10010000
7	Rightmost bit of the product = 0	01100010	00011011|10010000
7	Shift product right	01100010	00001101|11001000
8	Rightmost bit of the product = 0	01100010	00001101|11001000
8	Shift product right	01100010	00000110|11100100

Thus, the product is

$$0000\ 0110\ 1110\ 0100 = 06E4_{16}$$

3.16

Calculate the time necessary to perform a multiply using the approach given in Figure 3.7 if an integer is 8 bits wide and an adder takes 4 time units.

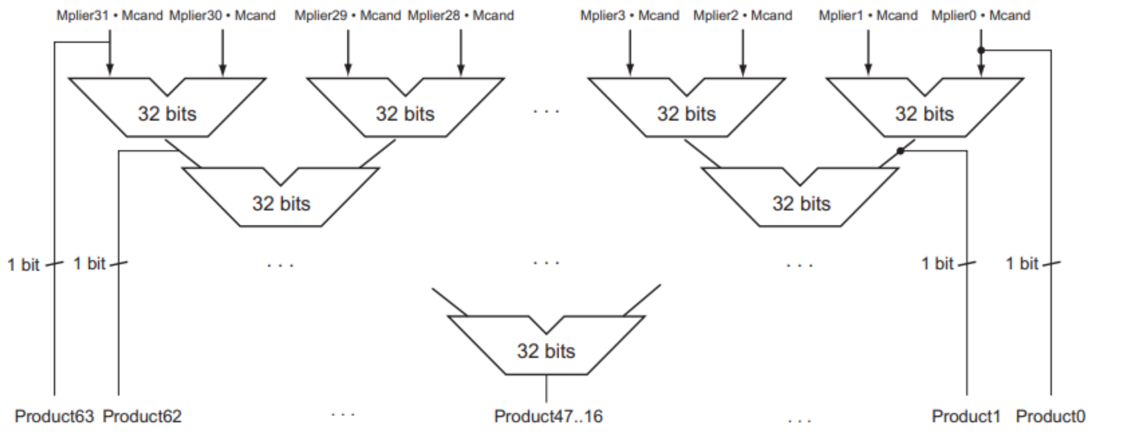


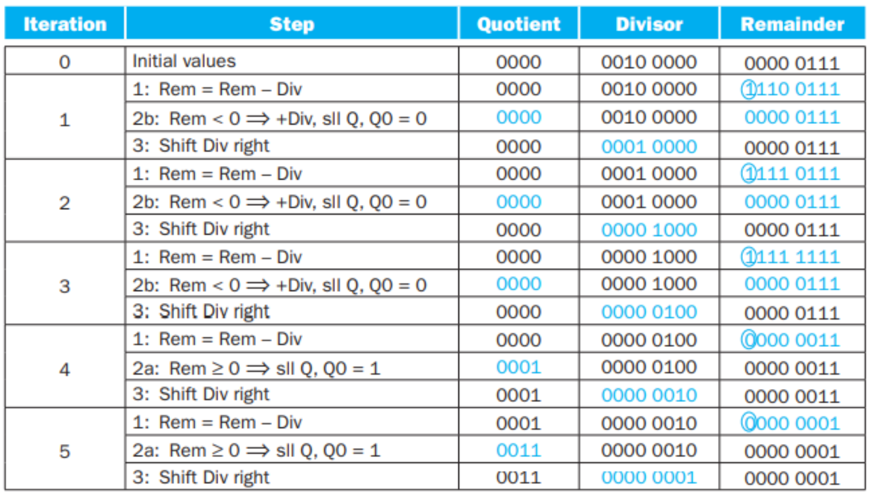
FIGURE 3.7 Fast multiplication hardware. Rather than use a single 32-bit adder 31 times, this hardware "unrolls the loop" to use 31 adders and then organizes them to minimize delay.

We organize 8 adders in a binary tree, the height of this tree require $log_2(8) = 3$ levels.

The operations on each level of the tree are parallelized, so we need $4 \times 3 = 12$ time units.

3.18

Using a table similar to that shown in Figure 3.10, calculate **74 divided by 21** using the hardware described in Figure 3.8. You should show the contents of each register on each step. Assume both inputs are unsigned 6-bit integers.



Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	1110 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	1111 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	1111 1111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	0000 0011
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	0000 0001
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

FIGURE 3.10 Division example using the algorithm in Figure 3.9. The bit examined to determine the next step is circled in color.

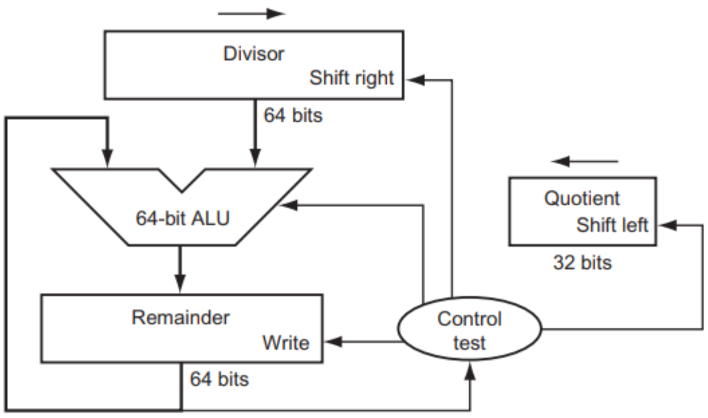


FIGURE 3.8 First version of the division hardware. The Divisor register, ALU, and Remainder register are all 64 bits wide, with only the Quotient register being 32 bits. The 32-bit divisor starts in the left half of the Divisor register and is shifted right 1 bit each iteration. The remainder is initialized with the dividend. Control decides when to shift the Divisor and Quotient registers and when to write the new value into the Remainder register.

$$74 \div 21$$

$$74_8 = 111\ 100_2$$
$$21_8 = 010\ 001_2$$

STEP	ACTION	QUOTIENT	DIVISOR	REMAINDER
0	Initial Vals	000 000	010 001 000 000	000 000 111 100
1	Rem=Rem−Div	000 000	010 001 000 000	101 111 111 100
1	Rem<0,R+D,Q<<	000 000	010 001 000 000	000 000 111 100
1	Rshift Div	000 000	001 000 100 000	000 000 111 100
2	Rem=Rem−Div	000 000	001 000 100 000	111 000 011 100
2	Rem<0,R+D,Q<<	000 000	001 000 100 000	000 000 111 100
2	Rshift Div	000 000	000 100 010 000	000 000 111 100
3	Rem=Rem−Div	000 000	000 100 010 000	111 100 101 100
3	Rem<0,R+D,Q<<	000 000	000 100 010 000	000 000 111 100
3	Rshift Div	000 000	000 010 001 000	000 000 111 100
4	Rem=Rem−Div	000 000	000 010 001 000	111 110 110 100
4	Rem<0,R+D,Q<<	000 000	000 010 001 000	000 000 111 100
4	Rshift Div	000 000	000 001 000 100	000 000 111 100
5	Rem=Rem−Div	000 000	000 001 000 100	111 111 111 000
5	Rem<0,R+D,Q<<	000 000	000 001 000 100	000 000 111 100
5	Rshift Div	000 000	000 000 100 010	000 000 111 100
6	Rem=Rem−Div	000 000	000 000 100 010	000 000 011 010
6	Rem>0,Q<<1	000 001	000 000 100 010	000 000 011 010
6	Rshift Div	000 001	000 000 010 001	000 000 011 010
7	Rem=Rem−Div	000 001	000 000 010 001	000 000 001 001
7	Rem>0,Q<<1	000 011	000 000 010 001	000 000 001 001
7	Rshift Div	000 011	000 000 001 000	000 000 001 001

So, the resulting quotient and remainder are

$$quotient = 000\ 011 = 3_8$$
$$remainder = 001\ 001 = 11_8$$