

CMPE 411

Computer Architecture

Lecture 8

Performing Division

September 26, 2017

[www.csee.umbc.edu/~younis/CMPE411/
CMPE411.htm](http://www.csee.umbc.edu/~younis/CMPE411/CMPE411.htm)



Lecture's Overview

Previous Lecture:

- Algorithms for multiplying unsigned numbers
(Evolution of optimization, complexity)
- Booth's algorithm for signed number multiplication
(Different approach to multiplying, 2-bit based operation selection)
- Multiple hardware design for integer multiplier
(Hardware cost-driven optimization , fast multiplication)

This Lecture:

- Algorithms for dividing unsigned numbers
- Handling of sign while performing a division
- Hardware design for integer division

Dividing Unsigned Numbers

- ❑ Paper and pencil example (unsigned):

$$\begin{array}{r} \text{Quotient } 1001 \\ \text{Divisor } 1000 \overline{) 1001010} \\ \underline{-1000} \\ 10 \\ 101 \\ 1010 \\ \underline{-1000} \\ 10 \end{array}$$

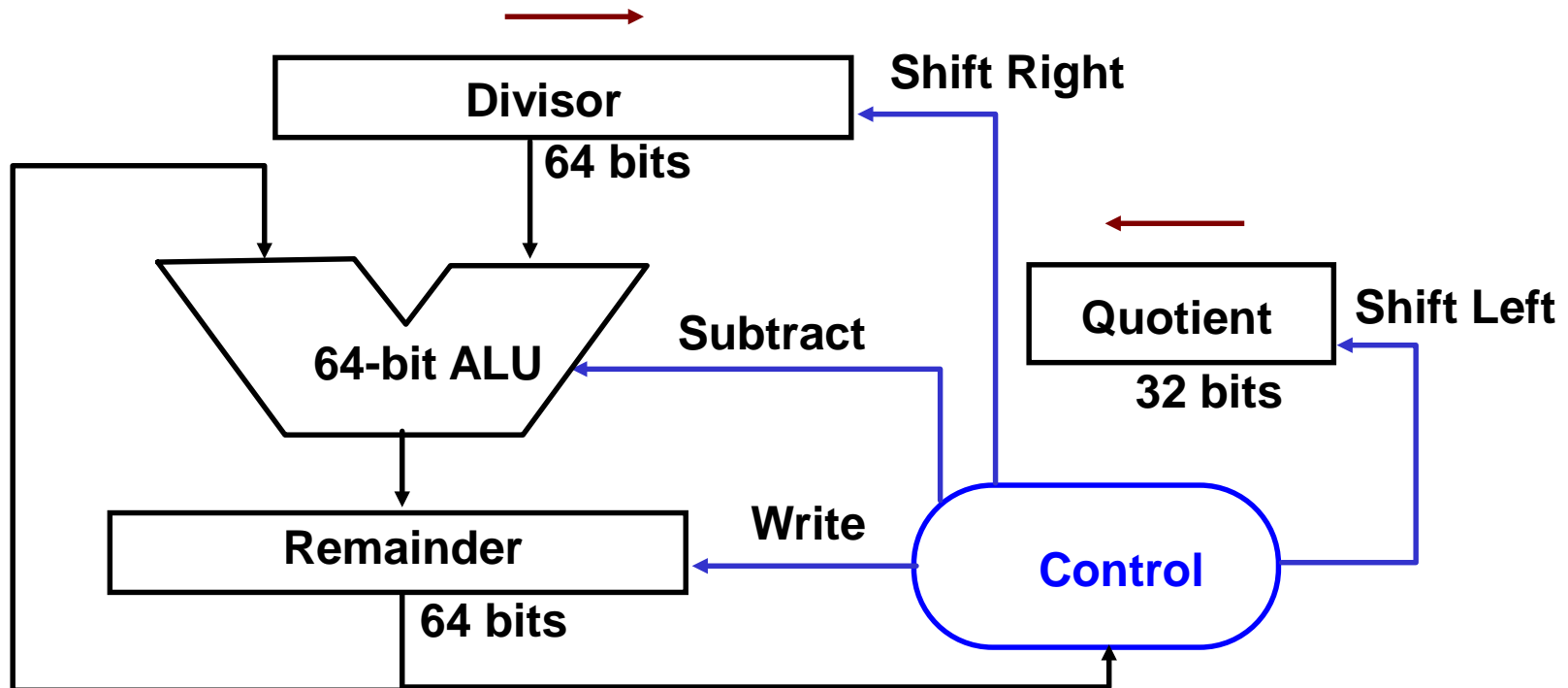
Dividend

Remainder (or Modulo result)

- ❑ See how big a number can be subtracted, creating quotient bit on each step
Binary $\Rightarrow 1 * \text{divisor}$ or $0 * \text{divisor}$
- ❑ Dividend = Quotient x Divisor + Remainder
- ❑ 3 versions of divide, successive refinement

Divide Hardware (version 1)

- ❑ 64-bit Divisor register, 64-bit ALU, 64-bit Remainder register, and 32-bit Quotient register
- ❑ The 32-bit value of the Divisor starts in the left half of the 64-bit register
- ❑ The Divisor is shifted to the right every step to align with the Dividend
- ❑ The Remainder register is initialized with the value of the Dividend
- ❑ Control decides when to shift the Divisor and the Quotient registers and when to write new value into the Remainder register

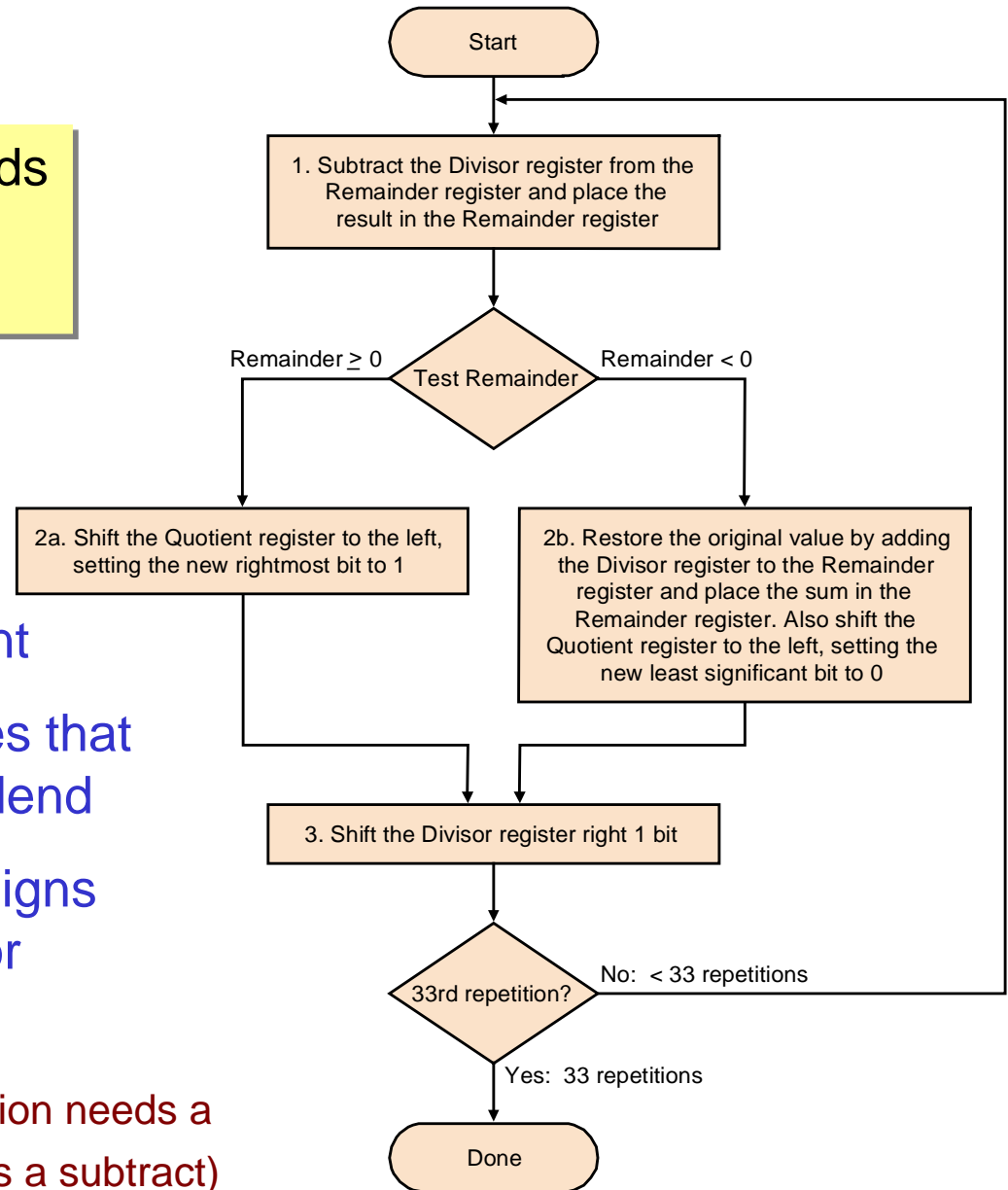


* Figure is courtesy of Dave Patterson

Divide Algorithm Version 1

Dividing two n-bit numbers needs n+1 steps to generate n-bit Quotient and Remainder

- ➔ If the Remainder is positive, a 1 is generated in the Quotient
- ➔ A negative Remainder indicates that Divisor did not go into the Dividend
- ➔ Shifting the Divisor in step 3 aligns the Divisor with the Dividend for next iteration
- ➔ Repeat for 33 times? (First iteration needs a shift for divisor and last iteration needs a subtract)



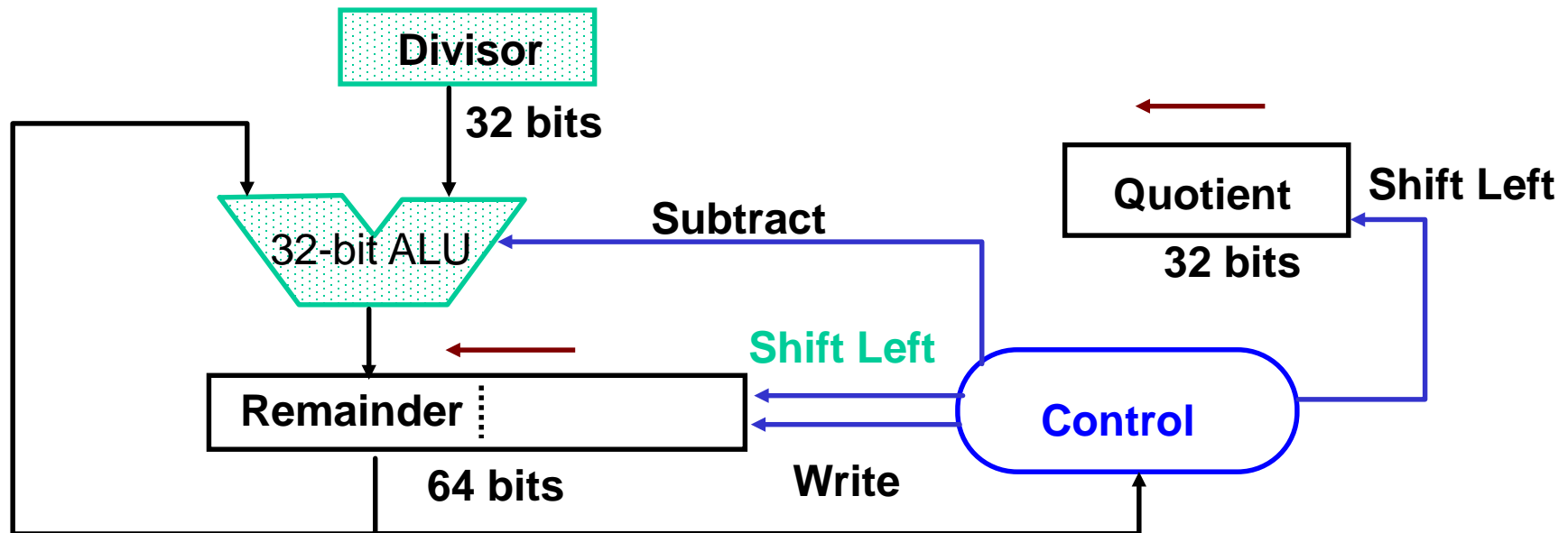
An Example

Follow the division algorithm (version 1) to divide 7 by 2 using only 4-bit binary representation

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 \Rightarrow +Div, shift left Q, Q0=0	0000	0010 0000	0000 0111
	3: Shift right Divisor	0000	0001 0000	0000 0111
2	1: Rem = Rem - Div	0000	0001 0000	1111 0111
	2b: Rem < 0 \Rightarrow +Div, shift left Q, Q0=0	0000	0001 0000	0000 0111
	3: Shift right Divisor	0000	0000 1000	0000 0111
3	1: Rem = Rem - Div	0000	0000 1000	1111 1111
	2b: Rem < 0 \Rightarrow +Div, shift left Q, Q0=0	0000	0000 1000	0000 0111
	3: Shift right Divisor	0000	0000 0100	0000 0111
4	1: Rem = Rem - Div	0000	0000 0100	0000 0011
	2a: Rem \geq 0 \Rightarrow shift left Q, Q0=1	0001	0000 0100	0000 0011
	3: Shift right Divisor	0001	0000 0010	0000 0011
5	1: Rem = Rem - Div	0001	0000 0010	0000 0001
	2a: Rem \geq 0 \Rightarrow shift left Q, Q0=1	0011	0000 0010	0000 0001
	3: Shift right Divisor	0011	0000 0001	0000 0001

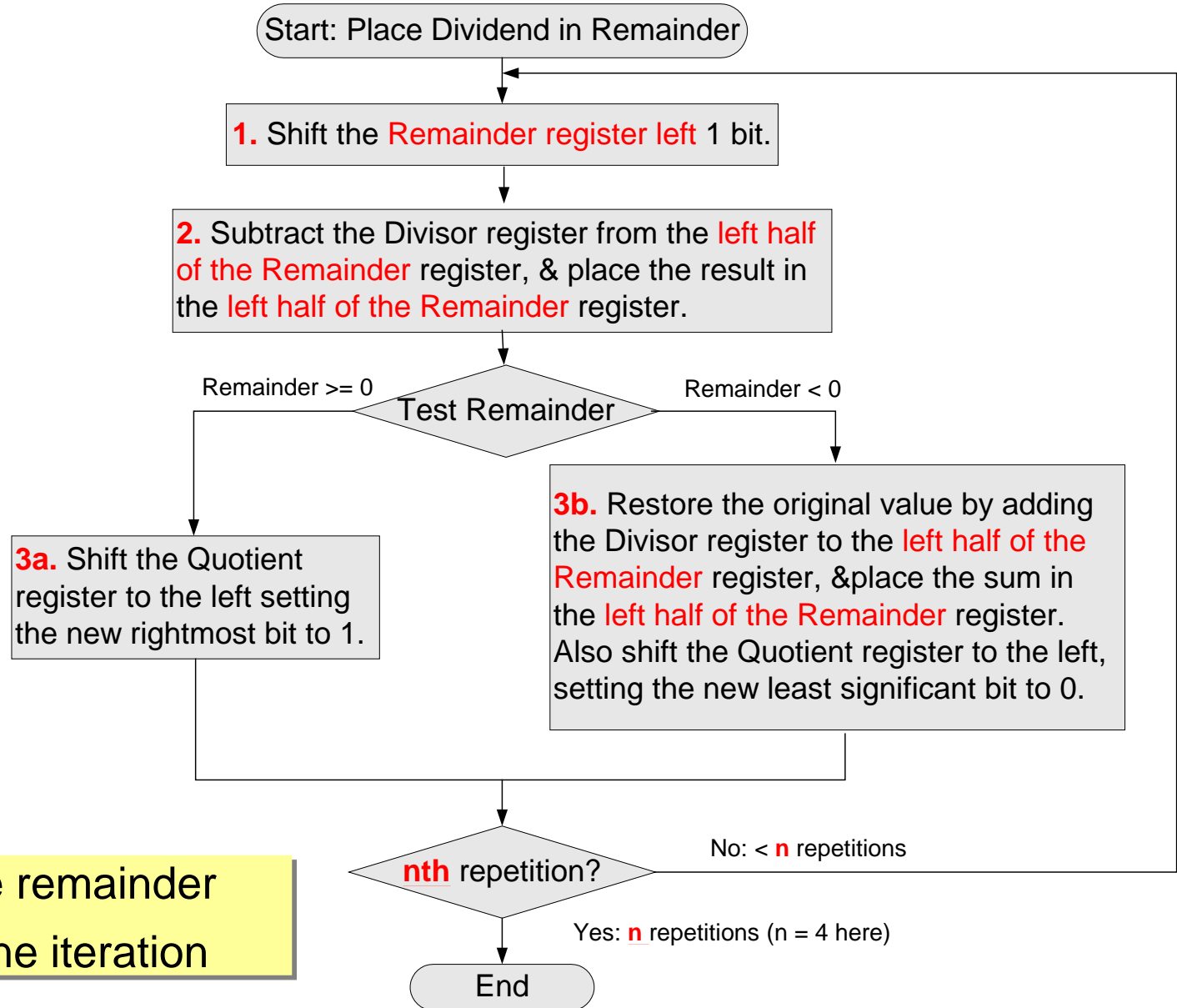
Divide Hardware (version 2)

- ❑ In the first version of divide hardware, half the bits in Divisor always 0
=> 1/2 of 64-bit adder is wasted & 1/2 of divisor is wasted
- ❑ Uses only 32-bit Divisor register, 32-bit ALU, 64-bit Remainder register, and 32-bit Quotient register
- ❑ Since the least significant bits of the Divisor would not change, the Remainder could be shifted to the left instead of shifting the Divisor to the right
- ❑ 1st step cannot produce a 1 in quotient bit (divide by zero)
=> switch order to shift first and then subtract, can save 1 iteration
- ❑ The most significant 32-bits would be used by the ALU as a result register



* Figure is courtesy of Dave Patterson

Divide Algorithm Version 2



Early shifting the remainder register saves one iteration

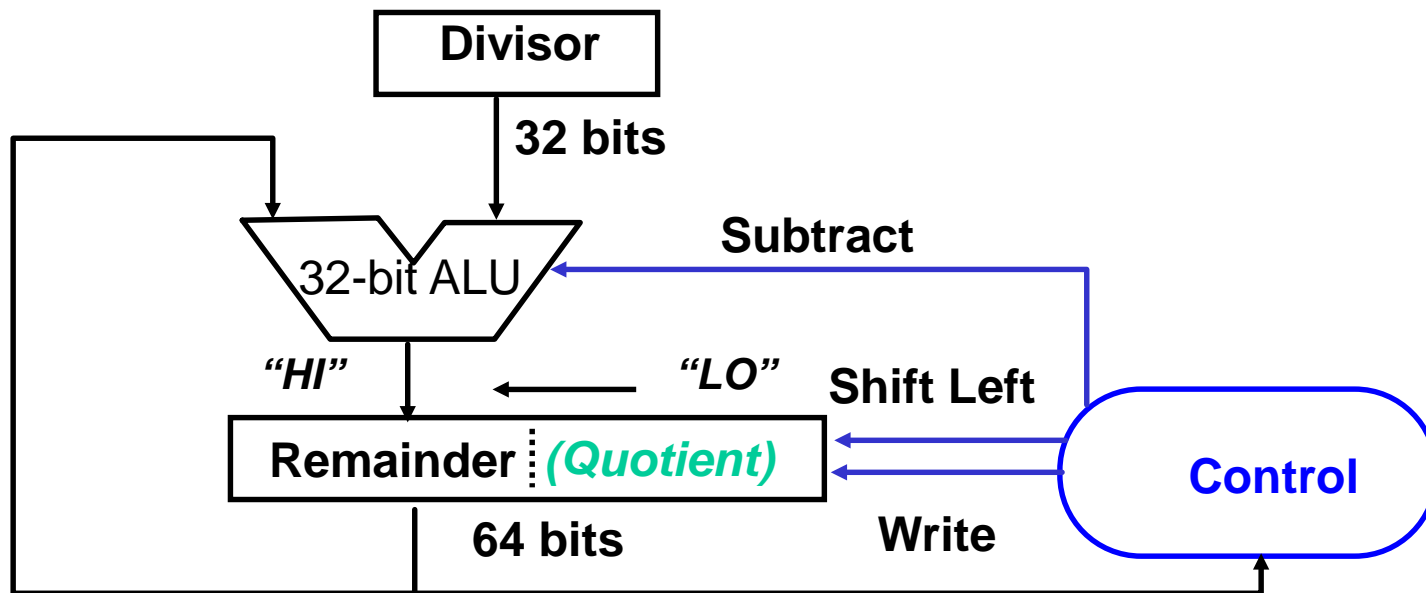
An Example

Follow the division algorithm (version 2) to divide 7 by 2 using only 4-bit binary representation

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010	0000 0111
1	1: Shift Rem to left 1	0000	0010	0000 1110
	2: Rem = Rem - Div	0000	0010	1110 1110
	3b: Rem < 0 \Rightarrow +Div, shift left Q, Q0=0	0000	0010	0000 1110
2	1: Shift Rem to left 1	0000	0010	0001 1100
	2: Rem = Rem - Div	0000	0010	1111 1100
	3b: Rem < 0 \Rightarrow +Div, shift left Q, Q0=0	0000	0010	0001 1100
3	1: Shift Rem to left 1	0000	0010	0011 1000
	2: Rem = Rem - Div	0000	0010	0001 1000
	3: Shift left Quotient, Q0=1	0001	0010	0001 1000
4	1: Shift Rem to left 1	0001	0010	0011 0000
	2: Rem = Rem - Div	0001	0010	0001 0000
	3: Shift left Quotient, Q0=1	0011	0010	0001 0000

Divide Hardware Version 3

- ❑ Remainder register wastes space that exactly matches size of Quotient
⇒ combine Quotient register and Remainder register
- ❑ Uses only 32-bit Divisor register, 32-bit ALU, 64-bit Remainder register, and 0-bit Quotient register
- ❑ The same number of shift operations would apply to both the Remainder and the Quotient ⇒ the Remainder needs to be corrected at the end
- ❑ The most significant 32-bits are still being used by ALU as a result register

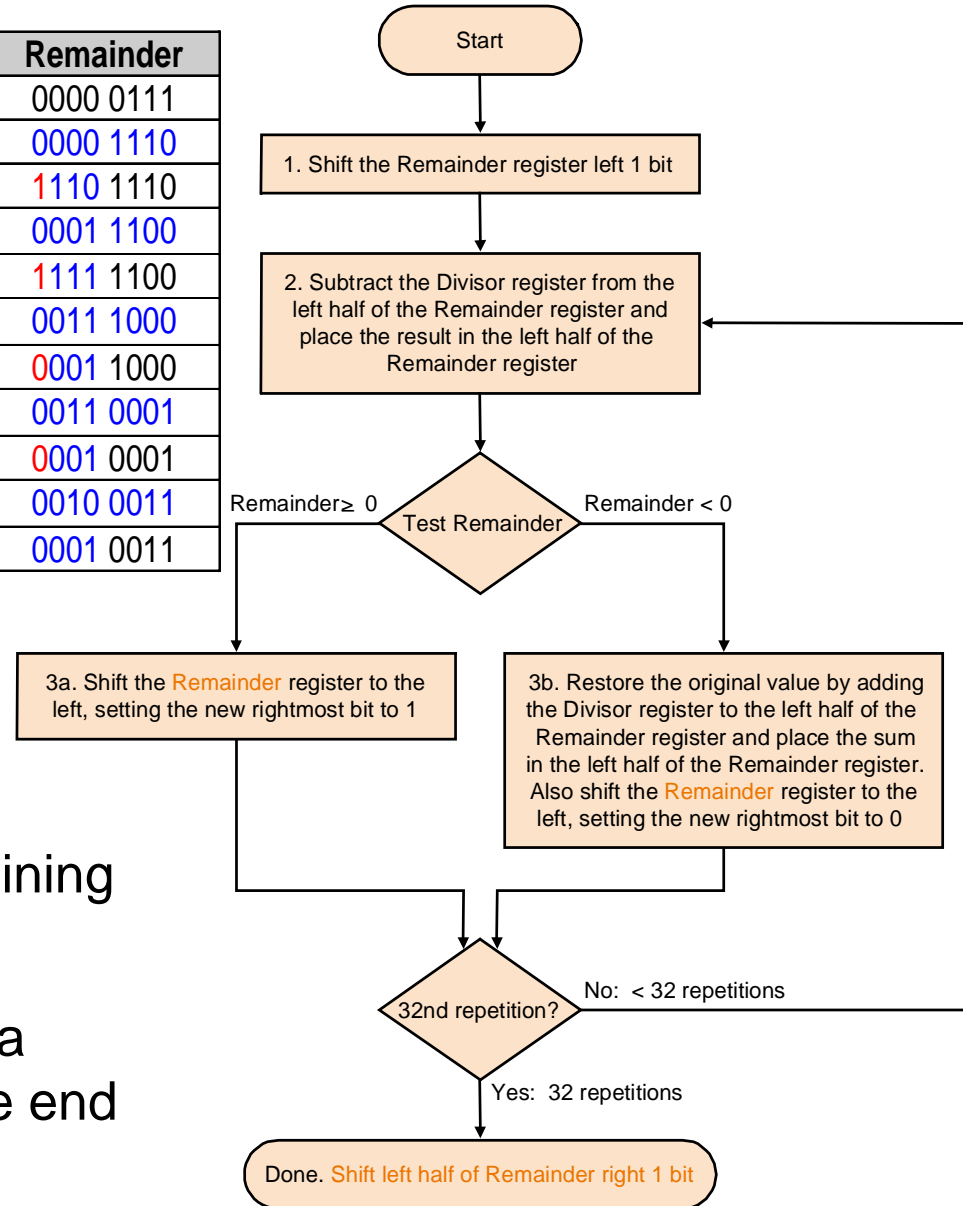


* Figure is courtesy of Dave Patterson

Divide Algorithm Version 3

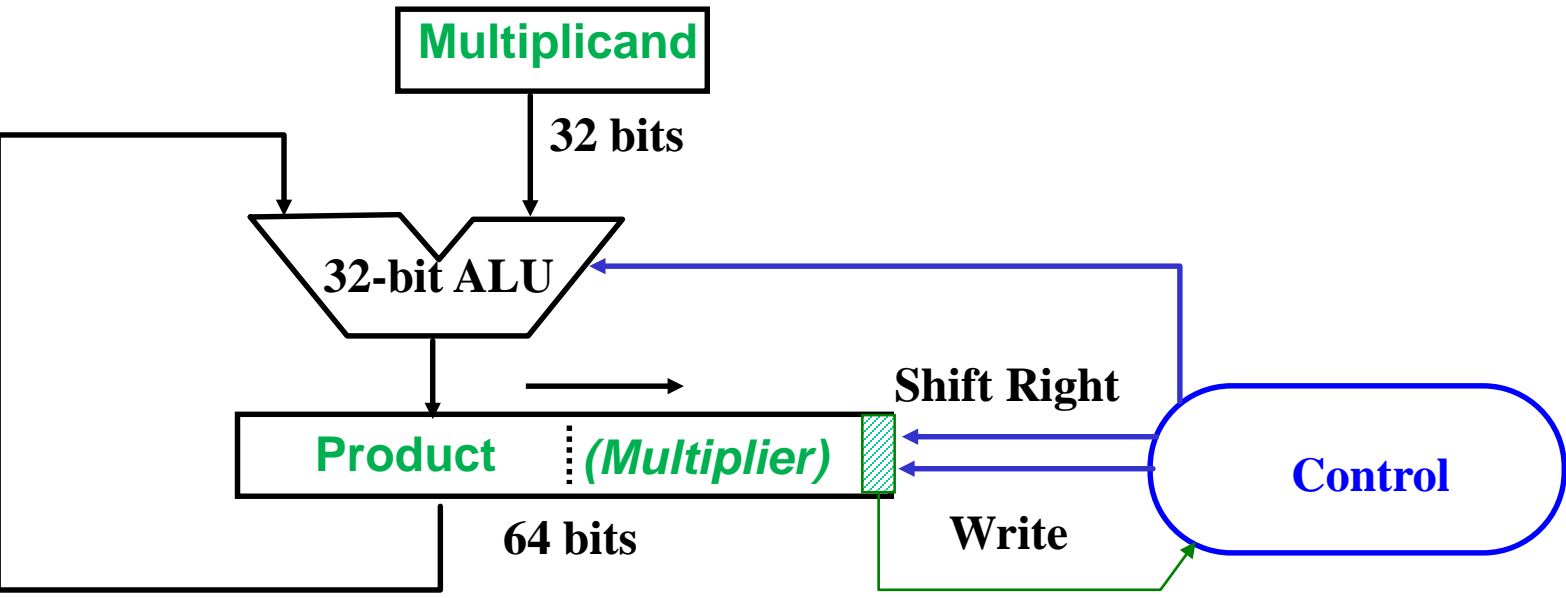
Iteration	Step	Divisor	Remainder
0	Initial values	0010	0000 0111
	Shift Rem left 1	0010	0000 1110
1	2: Rem = Rem - Div	0010	1110 1110
	3b: Rem < 0 \Rightarrow +Div, shift left R, R0=0	0010	0001 1100
2	2: Rem = Rem - Div	0010	1111 1100
	3b: Rem < 0 \Rightarrow +Div, shift left R, R0=0	0010	0011 1000
3	2: Rem = Rem - Div	0010	0001 1000
	3a: Rem \geq 0 \Rightarrow shift left R, R0=1	0010	0011 0001
4	2: Rem = Rem - Div	0010	0001 0001
	3a: Rem \geq 0 \Rightarrow shift left R, R0=1	0010	0010 0011
	Shift left half of Rem right 1	0010	0001 0011

Dividing 7 by 2

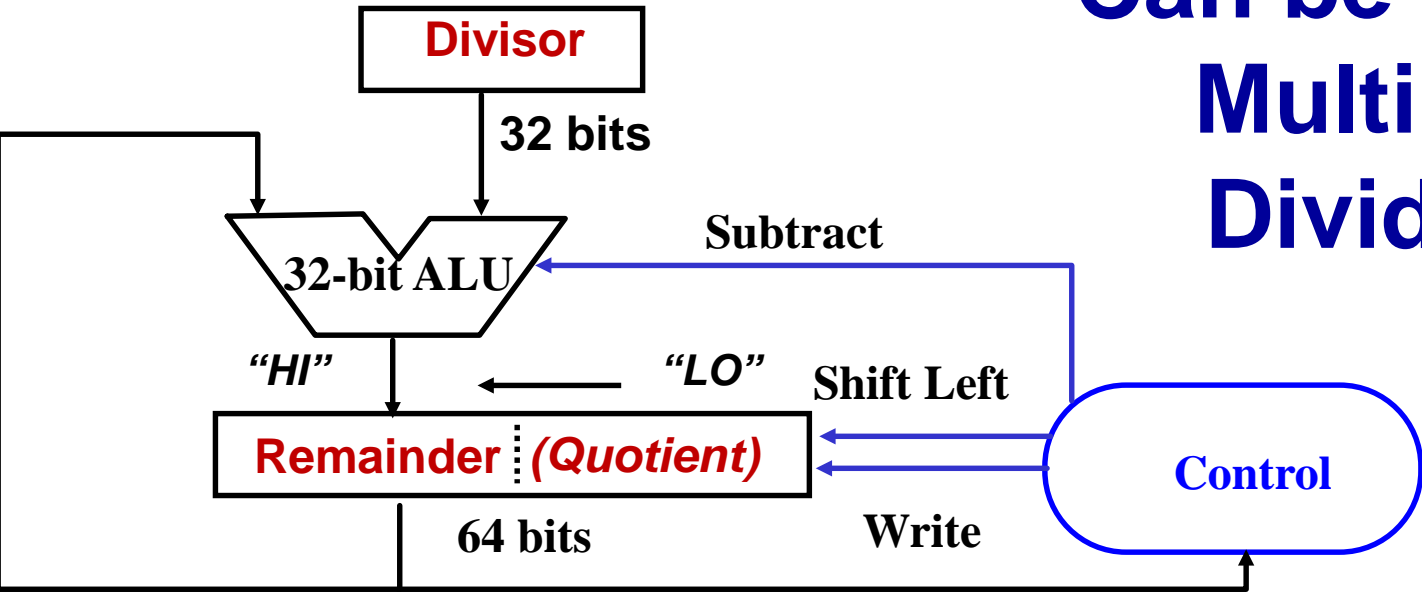


➔ Eliminate Quotient register by combining with Remainder and shifted left

➔ Remainder would be shifted an extra time and need to be corrected at the end



**Can be combined
Multiply and
Divide logic**



Dividing Signed Numbers

Simplest approach is to remember signs, make positive, and complement quotient and remainder if necessary

- **Rule 1:** Dividend and Remainder must have same sign
- **Rule 2:** Quotient negated if Divisor sign & Dividend sign are different

Examples:

$$\textit{Dividend} = \textit{Quotient} \times \textit{Divisor} + \textit{Remainder}$$

$$7 \div 2 = 3, \text{ remainder} = 1$$

$$-7 \div 2 = -3, \text{ remainder} = -1$$

$$7 \div -2 = -3, \text{ remainder} = 1$$

$$-7 \div -2 = 3, \text{ remainder} = -1$$

Conclusion

□ Summary

- ➔ Algorithms for dividing unsigned numbers
(Evolution of optimization, complexity)
- ➔ Handling of sign while performing a division
(Remainder sign matches the dividend's)
- ➔ Hardware design for integer division
(Same hardware as Multiply)

□ Next Lecture

- ➔ Representation of floating point numbers
- ➔ Floating point arithmetic
- ➔ Floating point hardware

Read section 3.4 in 5th Ed., or section 3.4 in 4th Ed. of the textbook