
Designing a Divider

With contributions from
J. Kubiawicz (CS152)



Divide: Paper & Pencil

		1001	Quotient
Divisor	1000	1001010	Dividend
		<u>-1000</u>	
		10	
		101	
		1010	
		<u>-1000</u>	
		10	Remainder (or Modulo result)

See how big a number can be subtracted, creating quotient bit on each step

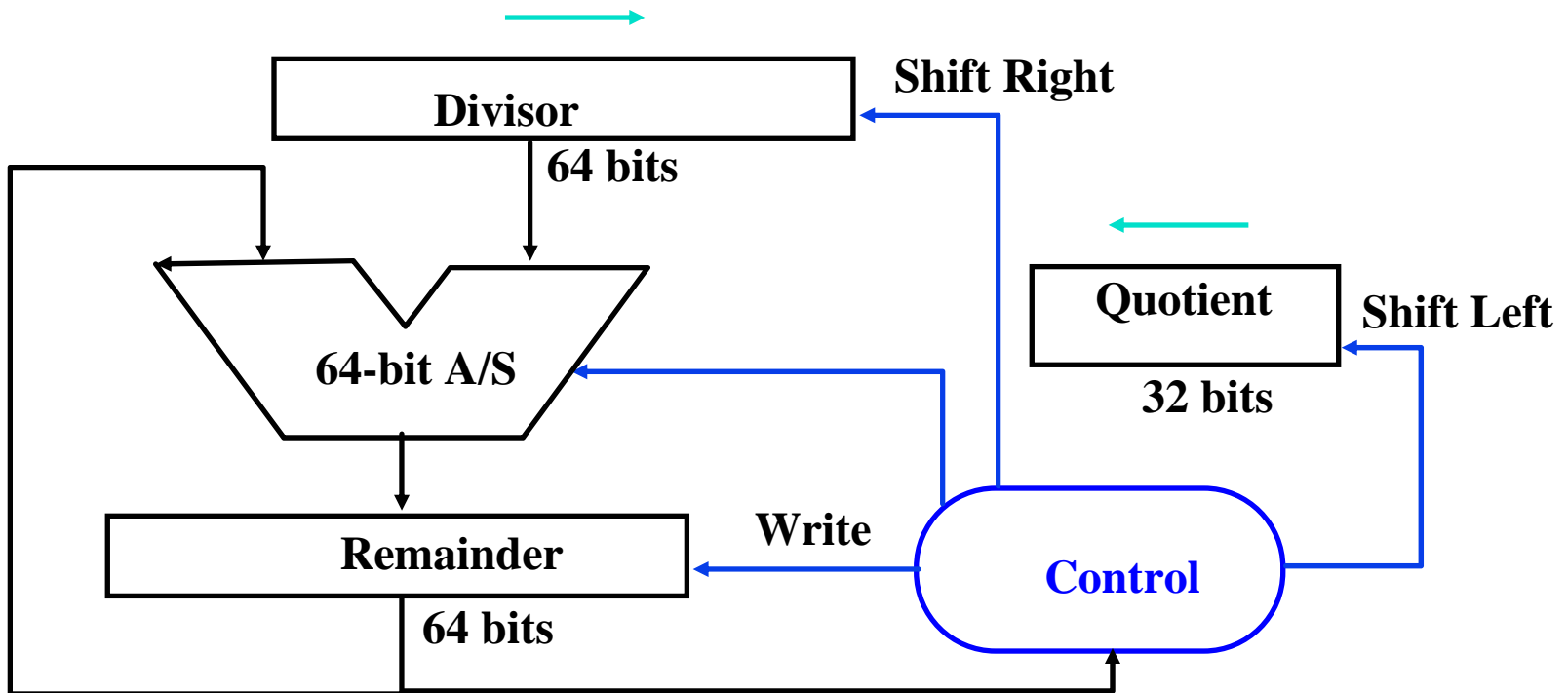
Binary => 1 * divisor or 0 * divisor

Dividend = Quotient x Divisor + Remainder

=> | Dividend | = | Quotient | + | Divisor |

DIVIDE HARDWARE Version 1

- 64-bit Divisor reg, 64-bit ALU, 64-bit Remainder reg, 32-bit Quotient reg



Divide Algorithm Version 1

Start: Place Dividend in Remainder

° Takes $n+1$ steps for n -bit Quotient & Rem.

Remainder	Quotient	Divisor
0000	0111	0000
0010	0000	0000

1. Subtract the Divisor register from the Remainder register, and place the result in the Remainder register.

Remainder ≥ 0 Test Remainder Remainder < 0

2a. Shift the Quotient register to the left setting the new rightmost bit to 1.

2b. Restore the original value by adding the Divisor register to the Remainder register, & place the sum in the Remainder register. Also shift the Quotient register to the left, setting the new least significant bit to 0.

3. Shift the Divisor register right 1 bit.

n+1 repetition? No: $< n+1$ repetitions

Yes: $n+1$ repetitions ($n = 4$ here)

Done

Divide Algorithm I example (7 / 2)

	Remainder		Quotient		Divisor
	0000	0111	00000		0010 0000
1:	1110	0111	00000		0010 0000
2:	0000	0111	00000		0010 0000
3:	0000	0111	00000		0001 0000
1:	1111	0111	00000		0001 0000
2:	0000	0111	00000		0001 0000
3:	0000	0111	00000		0000 1000
1:	1111	1111	00000		0000 1000
2:	0000	0111	00000		0000 1000
3:	0000	0111	00000		0000 0100
1:	0000	0011	00000		0000 0100
2:	0000	0011	00001		0000 0100
3:	0000	0011	00001		0000 0010
1:	0000	0001	00001		0000 0010
2:	0000	0001	00011		0000 0010
3:	0000	0001	00011		0000 0001

Answer:

Quotient = 3

Remainder = 1

Observations on Divide Version 1

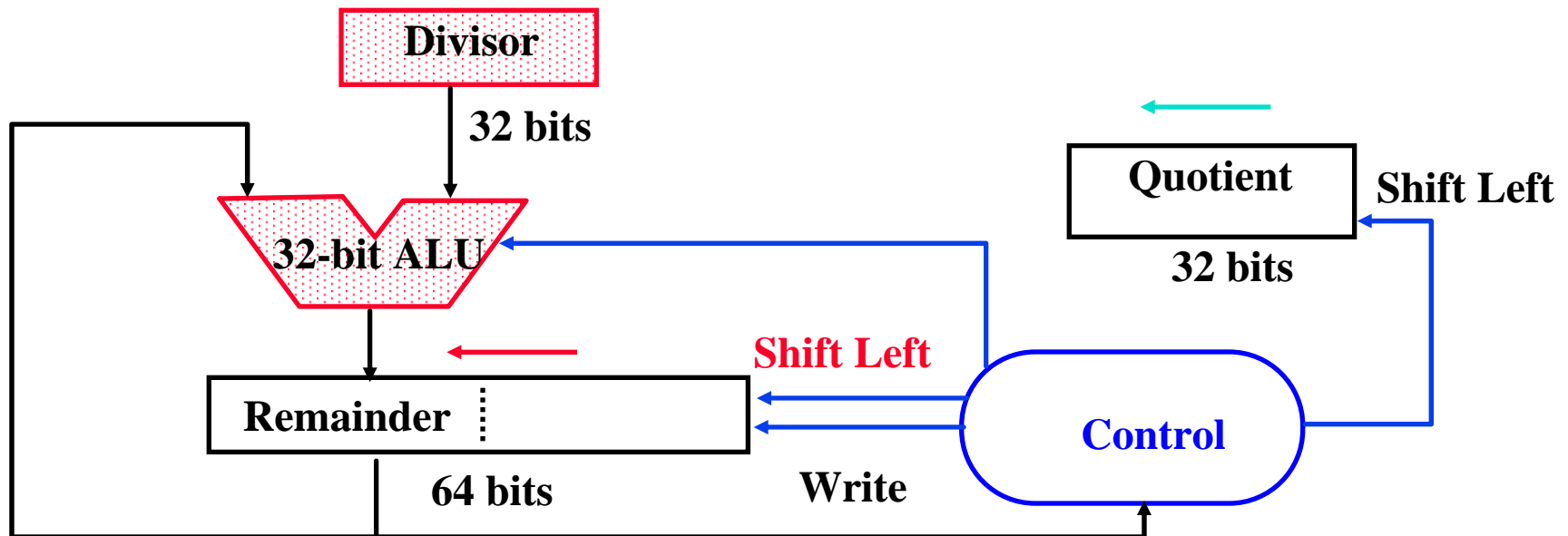
- **1/2 bits in divisor always 0**
=> 1/2 of 64-bit adder is wasted
=> 1/2 of divisor is wasted
- **Instead of shifting divisor to right, shift remainder to left?**

Divide Algorithm I example: wasted space

	Remainder	Quotient	Divisor		
	0000	0111	00000	0010	0000
1:	1110	0111	00000	0010	0000
2:	0000	0111	00000	0010	0000
3:	0000	0111	00000	0001	0000
1:	1111	0111	00000	0001	0000
2:	0000	0111	00000	0001	0000
3:	0000	0111	00000	0000	1000
1:	1111	1111	00000	0000	1000
2:	0000	0111	00000	0000	1000
3:	0000	0111	00000	0000	0100
1:	0000	0011	00000	0000	0100
2:	0000	0011	00001	0000	0100
3:	0000	0011	00001	0000	0010
1:	0000	0001	00001	0000	0010
2:	0000	0001	00011	0000	0010
3:	0000	0001	00011	0000	0010

DIVIDE HARDWARE Version 2

- **32-bit** Divisor reg, **32-bit** ALU, 64-bit Remainder reg, 32-bit Quotient reg



Divide Algorithm Version 2

Remainder	Quotient	Divisor
0000	0111	0010

Start: Place Dividend in Remainder

1. Shift the **Remainder register left** 1 bit

2. Subtract the Divisor register from the **left half of the** Remainder register, & place the result in the **left half of the** Remainder register.

Test Remainder

Remainder ≥ 0 Remainder < 0

3a. Shift the Quotient register to the left setting the new rightmost bit to 1.

3b. Restore the original value by adding the Divisor register to the **left half of the** Remainder register, & place the sum in the **left half of the** Remainder register. Also shift the Quotient register to the left, setting the new least significant bit to 0.

nth repetition?

No: $< n$ repetitions

Yes: n repetitions (n = 4 here)

Done

Divide Algorithm I version 2 (shift remainder)

	Remainder	Quotient	Divisor
	0000 0111	00000	0010
1:	1110 0111	00000	0010
2:	0000 0111	00000	0010
3:	0000 1110	00000	0010
1:	1110 1110	00000	0010
2:	0000 1110	00000	0010
3:	0001 1100	00000	0010
1:	1111 1100	00000	0010
2:	0001 1100	00000	0010
3:	0011 1000	00000	0010
1:	0001 1000	00001	0010
2:	0001 1000	00001	0010
3:	0011 0000	00001	0010
1:	0001 0000	00011	0010
2:	0001 0000	00011	0010

Divide: Revisited

Non-restoring divider

		1001	Quotient
Divisor	1000	$\overline{) 1001010}$	Dividend
		$\underline{-1000}$	
		00010	
		$\underline{-1000}$	
		110101	
		$\underline{+1000}$	
		111010	
		$\underline{+1000}$	
		00010	Remainder (or Modulo result)

Avoids extra step of “restoration” when partial result is negative.
Instead of subtract, adds divisor on next iteration

Divide Algorithm I example: non-restoring

	Remainder	Quotient	Divisor
	0000	0111	00000
	0010	00000	0010
1:	1110	0111	00000
	0010	00000	0010
2:	1100	1110	00000
	0010	00000	0010
1:	1110	1110	00000
	0010	00000	0010
2:	1101	1100	00000
	0010	00000	0010
1:	1111	1100	00000
	0010	00000	0010
2:	1111	1000	00000
	0010	00000	0010
1:	0001	1000	00001
	0010	00001	0010
2:	0011	0000	00001
	0010	00001	0010
1:	0001	0000	00011
	0010	00011	0010