Designing a Divider

With contributions from
J. Kubiatowicz (CS152)

Divide: Paper & Pencil

```
                  1001        Quotient
               ┌──────
Divisor 1000 │ 1001010        Dividend
              -1000
                 10
                 101
                 1010
                -1000
                   10        Remainder (or Modulo result)
```

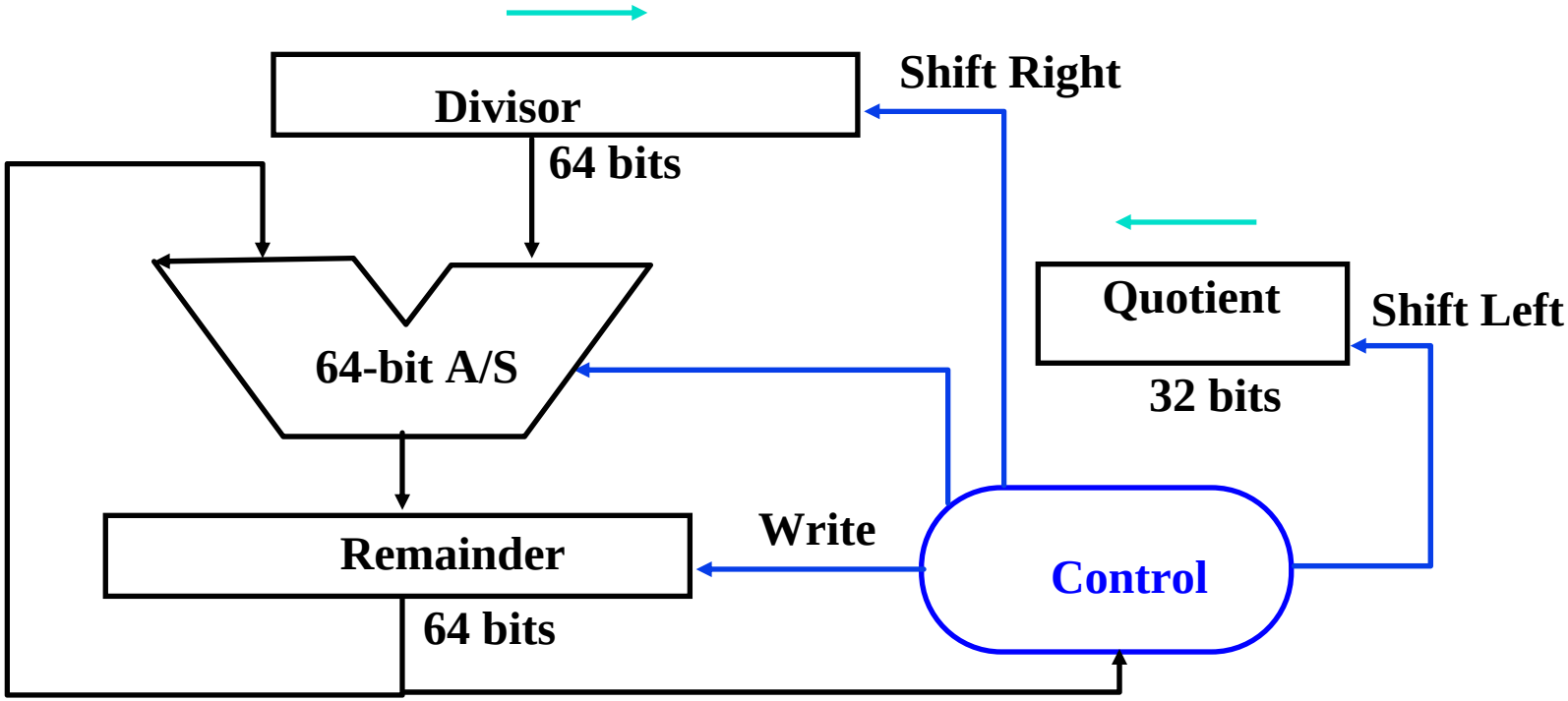
See how big a number can be subtracted, creating quotient bit on each step

Binary => 1 * divisor or 0 * divisor

Dividend = Quotient x Divisor + Remainder
=> | Dividend | = | Quotient | + | Divisor |

DIVIDE HARDWARE Version 1

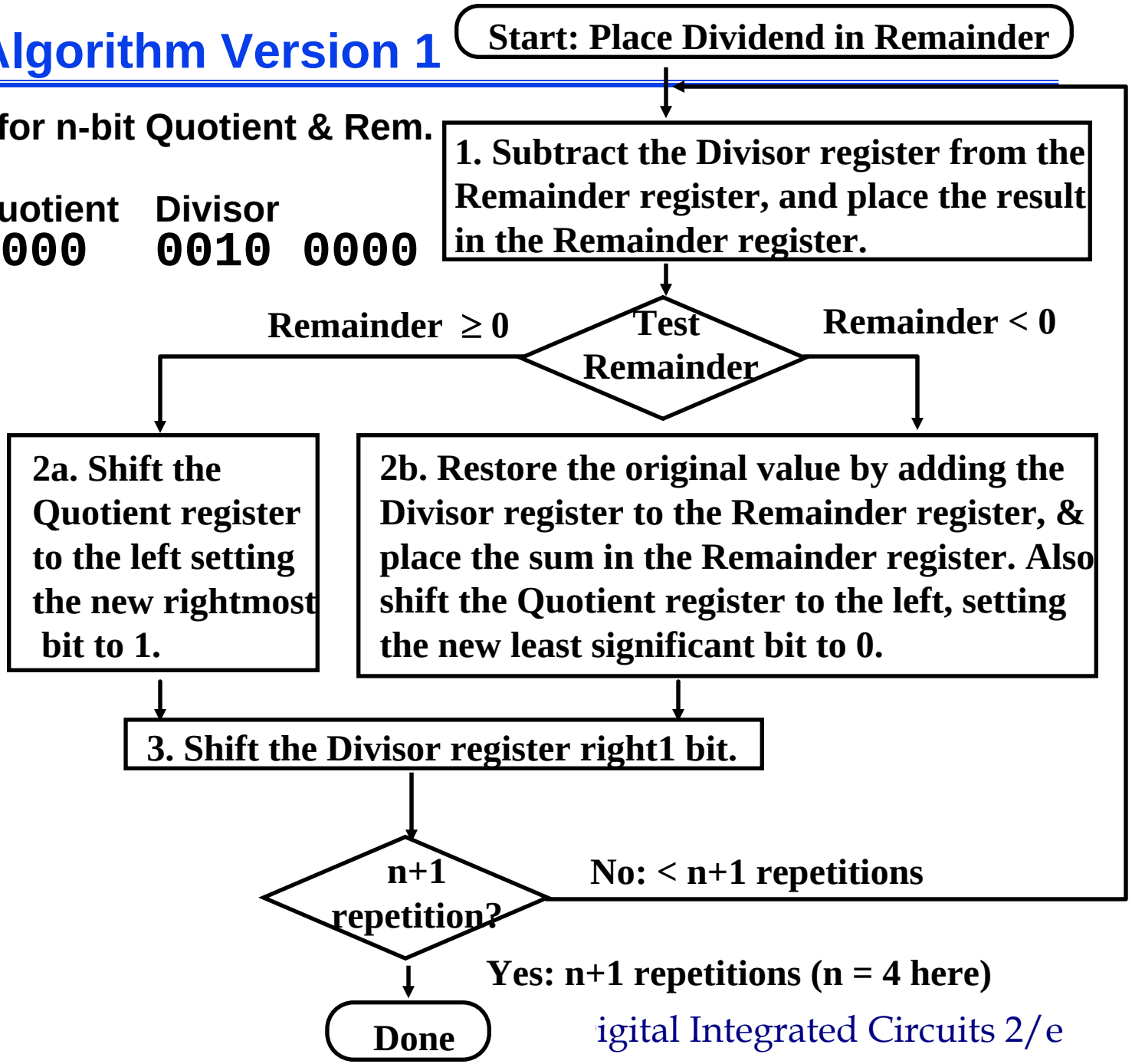
° 64-bit Divisor reg, 64-bit ALU, 64-bit Remainder reg, 32-bit Quotient reg



Divide Algorithm Version 1

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

Remainder	Quotient	Divisor
0000 0111	0000	0010 0000



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

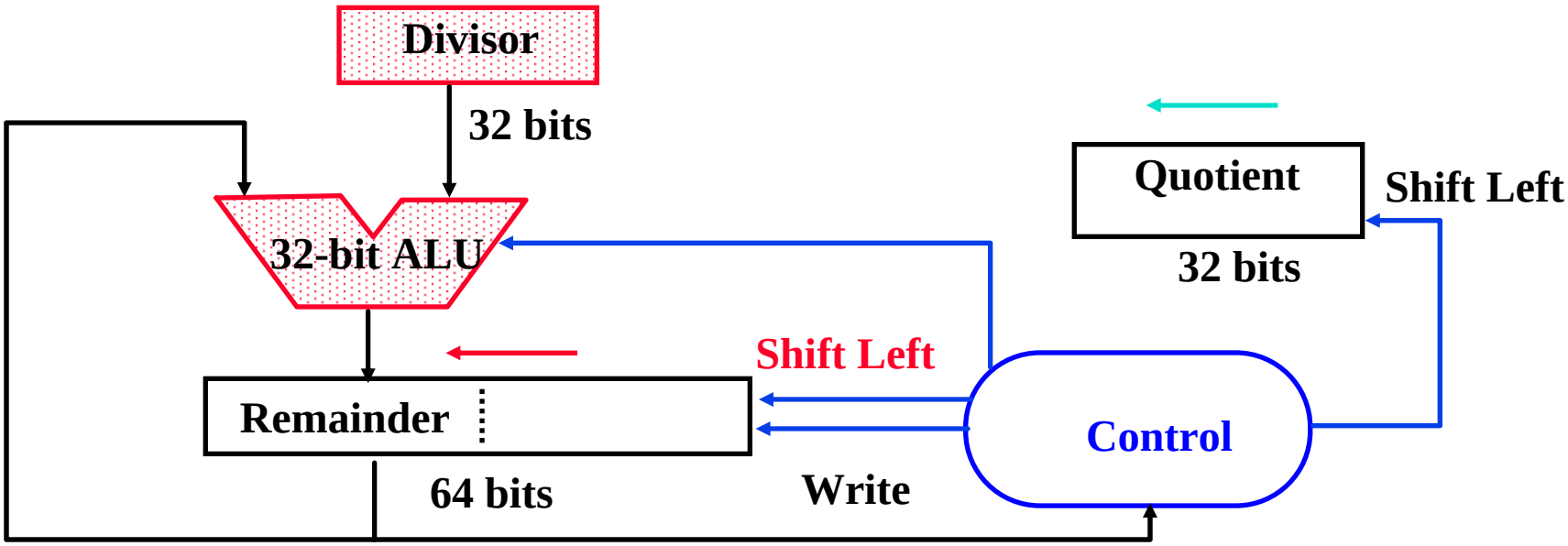
- Instead of shifting divisor to right, shift remainder to left?

Divide Algorithm I example: wasted space

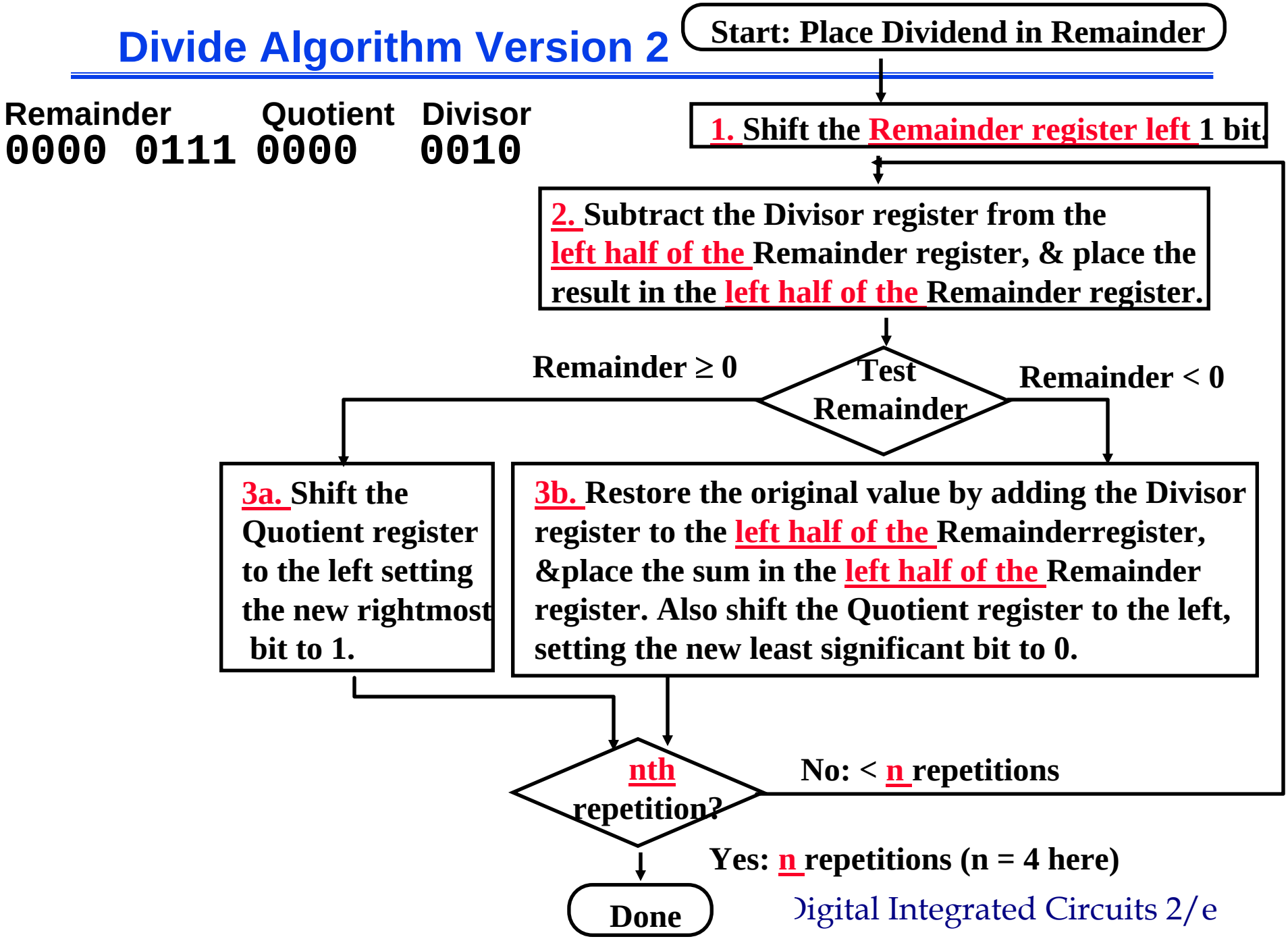
	Remainder	Quotient	Divisor
	0000 0111	00000	0010 0000
1:	1110 0111	00000	0010 0000
2:	0000 0111	00000	0010 0000
3:	0000 0111	00000	0001 0000
1:	1111 0111	00000	0001 0000
2:	0000 0111	00000	0001 0000
3:	0000 0111	00000	0000 1000
1:	1111 1111	00000	0000 1000
2:	0000 0111	00000	0000 1000
3:	0000 0111	00000	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

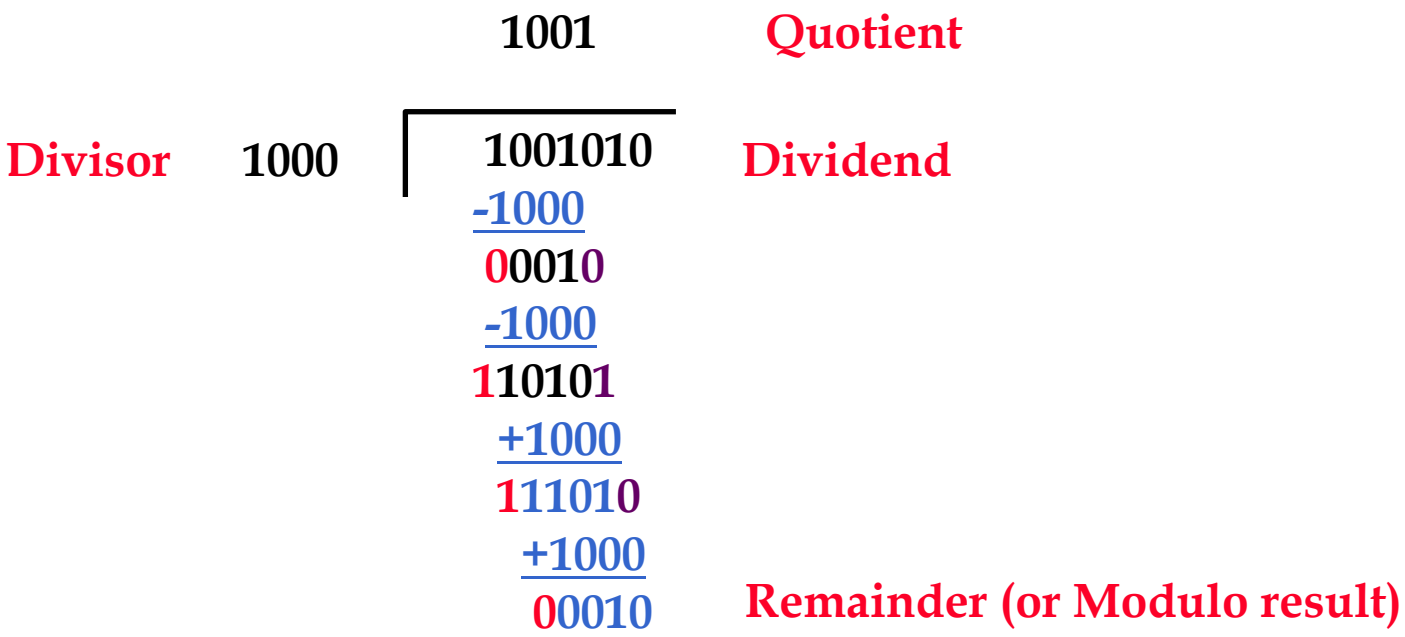


Divide Algorithm I version 2 (shift remainder)

	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



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