2.2 ADDITION

Digits are added bit by bit from right to left, with carries passed to the next digit to the left.

Eg) Add 6_{ten} to 7_{ten}

0000 0000 0000 0000 0000 0000 0000 $0110_2 \rightarrow 6_{10}$

0000 0000 0000 0000 0000 0000 0000 $0111_2 \rightarrow 7_{10}$

+

0000 0000 0000 0000 0000 0000 0000 $1101_2 \rightarrow 13_{10}$

Overflow occure when the result form an operation cannot be represented with the available hardware, in this case a 32-bit word.

When adding operands with different signs, overflow cannot occur. The reason is the sum must be no larger than one of the operands.

Eg) -10 +4 -6. Since the sum is no larger than an oerand, the sum must fit in 32 bits. No overflow.

2.2.1 SUBTRACTION

Subtraction uses addition : the appropriate operand is simply negated.

ie) In subtraction, take the 2's complement of negative number and then add.

Eg) Subtract 6_{10} from 7_{10}

0000 0000 0000 0000 0000 0000 0000 $0111_2 \rightarrow 7_{10}$

0000 0000 0000 0000 0000 0000 0000 $0110_2 \rightarrow 6_{10}$

-

0000 0000 0000 0000 0000 0000 0000 $0001_2 \rightarrow 1_{10}$

This is done directly. It can also be done via addition using 2's complement representation of -6

$6_{10} \rightarrow$ 0000 0000 0000 0000 0000 0000 0000 0110_2

Negation of $6_{10} \rightarrow$ 0000 0000 0000 0000 0000 0000 0000

1111 1111 1111 1111 1111 1111 1111 1001

1

$(-6_{10}) \rightarrow$ 1111 1111 1111 1111 1111 1111 1111 1010

$7_{10} \rightarrow$ 0000 0000 0000 0000 0000 0000 0000 0111_2

$-6_{10} \rightarrow$ 1111 1111 1111 1111 1111 1111 1111 1010_2

$=1_{10} \rightarrow$ 0000 0000 0000 0000 0000 0000 0000 0001_2

2.2.2 MULTIPLICATION

In multiplication the first operand is called the multiplicand. Second operand is called the multiplier. The final result is called the product. Fig 1 represents the flow chart of multiplication.

Eg) Multiplying 1000_{ten} & 1001_{10}

```
          1 0 0 0      → Multiplicand
          1 0 0 1      → Multilplier
          ------------
          1 0 0 0
        0 0 0 0
      0 0 0 0
    1 0 0 0
  ---------------------
    1 0 0 1 0 0 0      → product
```

The length of the multiplication of an n-bit multiplicand and an m-bit multiplier is a product. ie) n + m bits long.

To simply the multiplication we have two steps,

1) If multiplier digit is 1, just place a copy of the multiplicand. (1 x multiplicand)
2) If the multiplier digit is 0, place 0 (0 x multiplicand)

Steps

Step 1: check whether the LSB of multiplier is 0 or 1

1) If it is '1'
   - → add " Product + Multiplicand = product"
   - → Shift the multiplicand register left 1 bit
   - → shift the multiplier register right 1 bit
2) If it is '0'
   - →shift the multiplicand register left 1 bit
   - → shift the multiplier register right 1 bit

Eg) Multiply 2_{10} X 3_{10} (or) 0010_2 X 0011_2

Iteration	**step**	**multiplier**	**multiplicand**	**product**
0	Initial step values	0011	0000 0010	0000 0000
1	1: 1-> product =product + Multiplicand	0011	0000 0010	0000 0010
	2: shift left Multiplicand	0011	0000 0100	0000 0010
	3: shift right multiplier	0001	0000 0100	0000 0010
2	1: 1-> product =product + Multiplicand	0001	0000 0100	0000 0110
	2: shift left Multiplicand	0001	0000 1000	0000 0110
	3: shift right multiplier	0001	0000 1000	0000 0110
3	1: 0-> 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-> 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

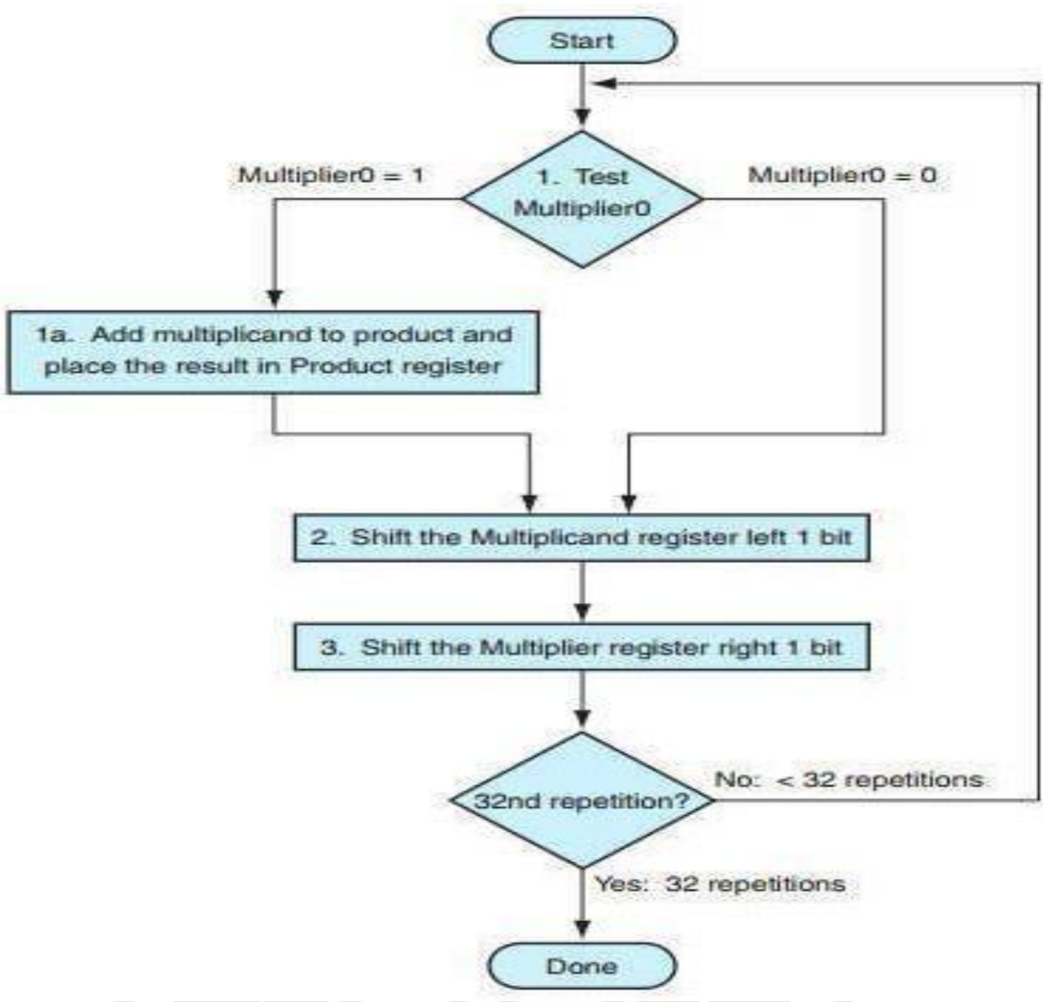
Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done

Fig 1: Steps for Multiplication

Source: David A. Patterson and John L. Hennessey, —Computer Organization and Design‖, Fifth edition, Morgan Kauffman / Elsevier

The multiplicand register, ALU and product register are all 64 bits wide, with only the multiplier register containing 32 bits. Fig 2 and 3 represent the first version of the multiplication hardware and refined version of the multiplication hardware

Control decides when to shift the multiplicand and multiplier registers and when to right new values into the product register.

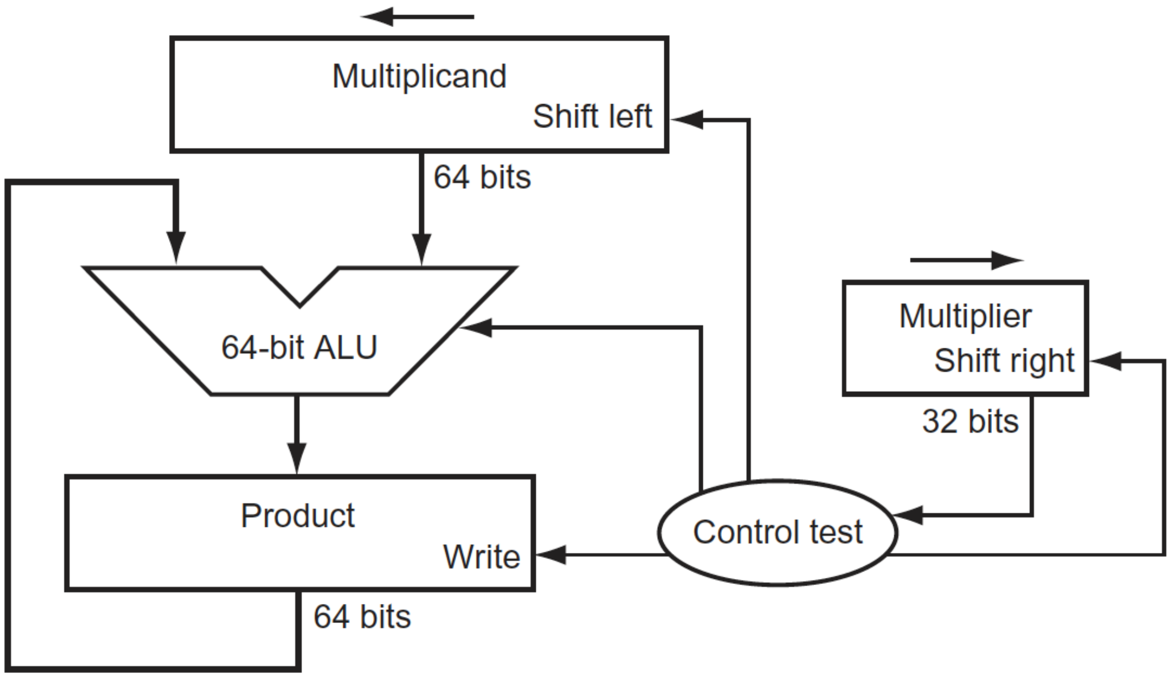

Fig 2: first version of the multiplication hardware

Source: David A. Patterson and John L. Hennessey, —Computer Organization and Design‖, Fifth edition, Morgan Kauffman / Elsevier

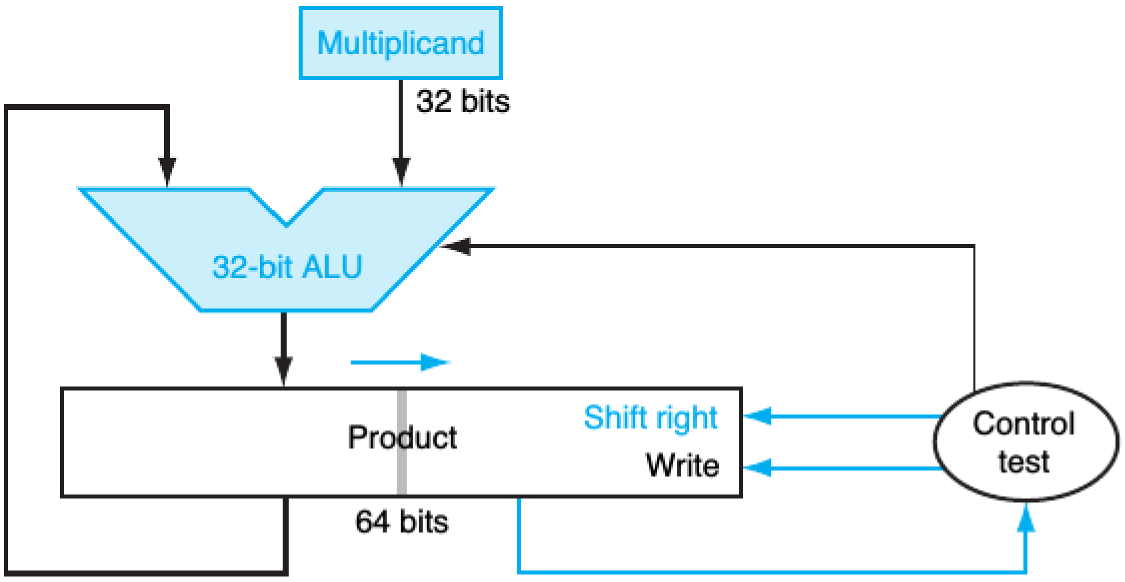

Fig 3: Refined version of the multiplication hardware

Source: David A. Patterson and John L. Hennessey, —Computer Organization and Design‖, Fifth edition, Morgan Kauffman / Elsevier

The speed-up comes from performing the operations in parallel. The hardware is further optimized to halve the width of adder and register by noticing the unused portions.

2.2.3 DIVISION

- Divides two operands called the dividend and divisor. Fig 6 shows the Division flow chart.
- Result called the quotient, a combined by a second result called remainder,

Dividend: A number being added

Dividend = Quotient X Divisor + Reminder

Divisor: A number that the dividend is divided by

Quotient The primary result of a division

Remainder: The Secondary result of a division.

Eg) Divide 1 0 0 1 0 10 $_{ten}$ by 10 0 0$_{ten}$

```
                1 0 0 0                 → Quotient
          ______________
  1 0 0 0 √1 0 0 1 0 1 0                → Divident
            1 0 0 0
          --------------
                1 0 1 0
                1 0 0 0
                ---------
                     10                 → Remainder
```

Division Algorithm:

Step 1:

Subtract the divisor register from the remainder register and place the result in the remainder register.

ie) Remainder = Remainder - Divisor

Step 2: Check whether the MSB of remainder ie) 0 (or) 1

1) If MSB of remainder is 1, then
   a) Remainder = Remainder + Divisor
   b) Shift quotient register to left
   c) Set empty space of quotient , $q_0 = 0$
2) If MSB of remainder is 0, then
   a) Shift quotient register to left
   b) Set , $q_0 = 1$

Step 3: Shift the divisor register 1 bit right.

Eg) Dividing 7_{10} by 2_{10} ie) 0 0 0 0 0 1 1 1 by 0 0 1 0

Iteration	**steps**	**Quotient**	**Divisor**	**Remainder**
0	Initial step values	0000	0010 0000	0000 0111
1	1: Reminder = Remainder - Div	0000	0010 0000	1110 0111
	2: Rem<0 => +Div, sll Q, Q_0=0	0000	0010 0000	0000 0111
	3: shift Div right	0000	0001 0000	0000 0111
2	1: Reminder = Remainder - Div	0000	0001 0000	1111 0111
	2: Rem<0 => +Div, sll Q, Q_0=0	0000	0001 0000	0000 0111
	3: shift Div right	0000	0000 0000	0000 0111
3	1: Reminder = Remainder - Div	0000	0000 1000	1111 1111
	2: Rem<0 => +Div, sll Q, Q_0=0	0000	0000 1000	0000 0111
	3: shift Div right	0000	0000 0100	0000 0111
4	1: Reminder = Remainder - Div	0000	0000 0100	0000 0011
	2: Rem<0 => sll Q, Q_0=1	0001	0000 0100	0000 0011
	3: shift Div right	0001	0000 0010	0000 0011
5	1: Reminder = Remainder - Div	0001	0000 0010	0000 0001
	2: Rem<0 => sll Q, Q_0=1	0011	0000 0010	0000 0001
	3: shift Div right	0011	0000 0001	0000 0001

Division Hardware:

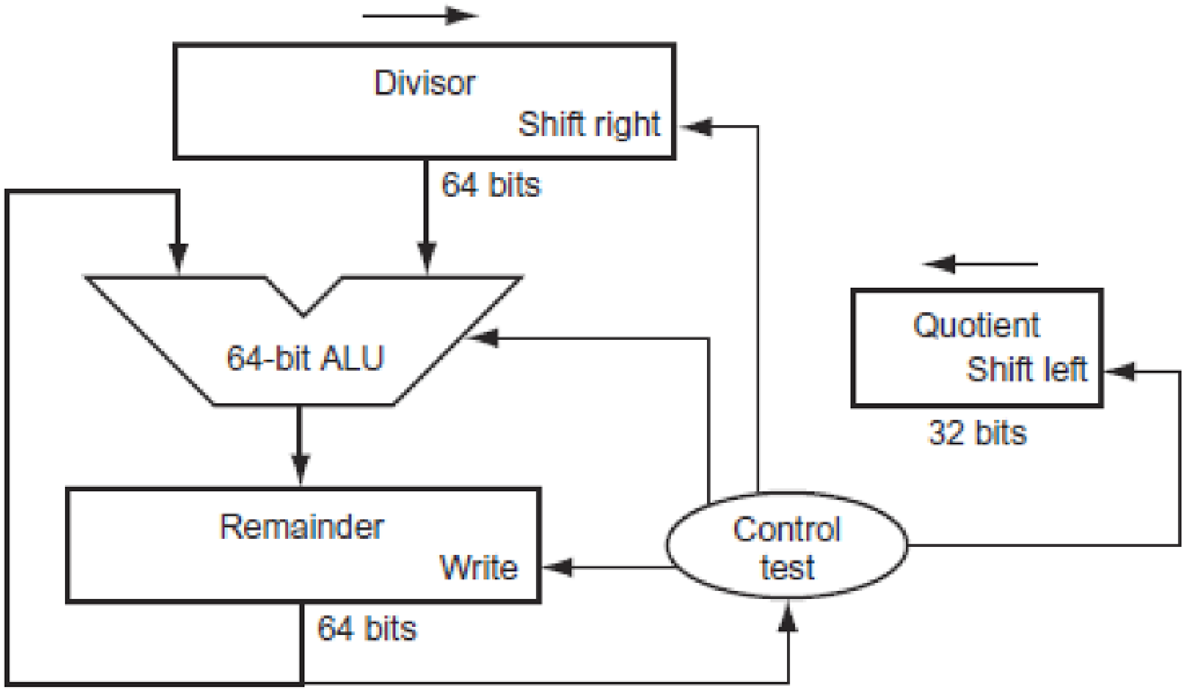

Fig 4 first version of the division hardware

Source: David A. Patterson and John L. Hennessey, ―Computer Organization and Design‖, Fifth edition, Morgan Kauffman / Elsevier

- Divisor reg, ALU and Remainder register are 64 bits wide. The above Fig 4 shows the first version of the division hardware.
- Quotient reg → 32 bits
- Remainder reg is initialized with the dividend.
- 32 bit divisor starts in the left half of the divisor reg and its shifter right, bit each iteration.
- Control decides when to shift the quotient and divisor reg and when to write the new value into remainder register.

The hardware can be refined to be faster and cheaper. The bellow Fig 5 shows an improved version of division hardware. The speed up comes from shifting the operands and quotient simultaneously with the subtraction. This refinement halves the width of the adder and register by noticing where there are unused portions of register and adders.

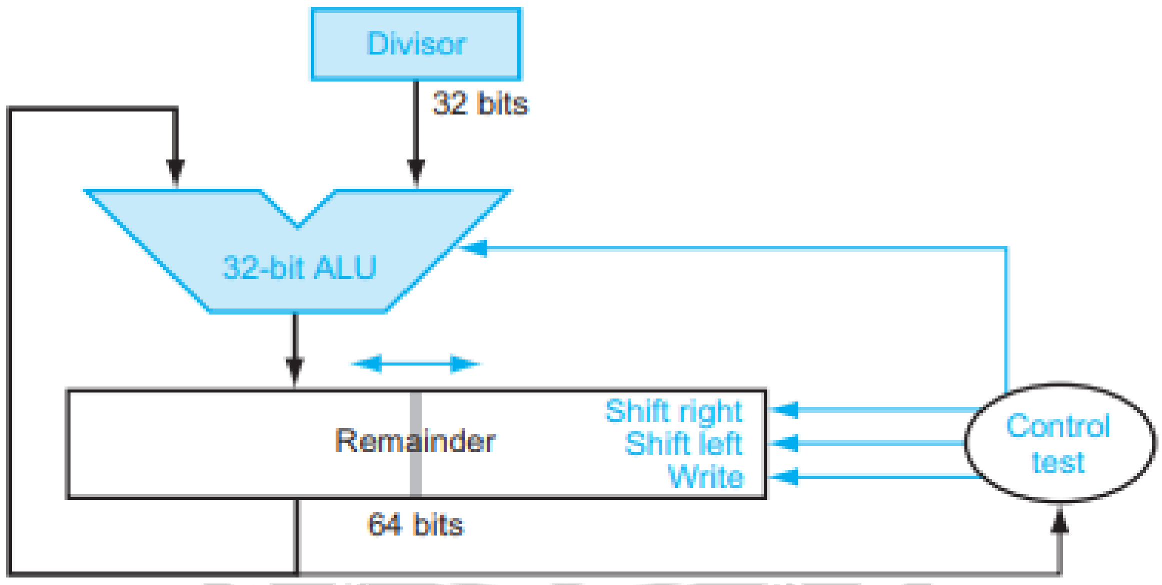

Fig 5: An improved version of division hardware

Source: David A. Patterson and John L. Hennessey, —Computer Organization and Design‖, Fifth edition, Morgan Kauffman / Elsevier

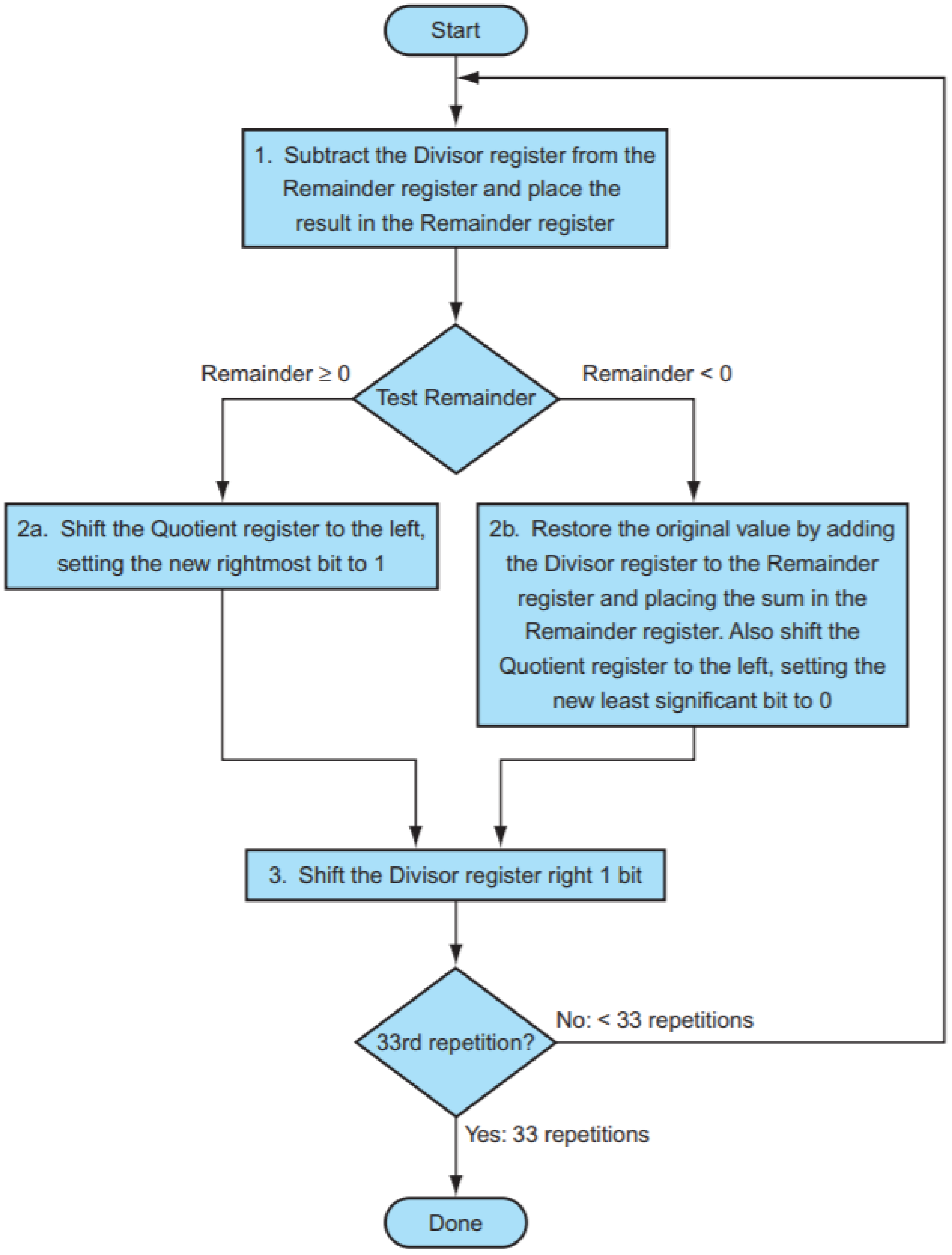
Start
1. Subtract the Divisor register from the Remainder register and place the result in the Remainder register
Test Remainder
Remainder ≥ 0
Remainder < 0
2a. Shift the Quotient register to the left, setting the new rightmost bit to 1
2b. Restore the original value by adding the Divisor register to the Remainder register and placing 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
33rd repetition?
No: < 33 repetitions
Yes: 33 repetitions
Done

Fig 6: Division Flow Chart

Source: David A. Patterson and John L. Hennessey, ―Computer Organization and Design‖, Fifth edition, Morgan Kauffman / Elsevier