Computer Math

Topics to be covered

- Multiplication
- Division
- Floating Point numbers

Multiplication

- Figures 3.5, 3.6, and 3.7 in your text
- On final, you are responsible for figure 3.7

Multiplication

Multiplication

9	7	0000 0111
x 24	x 5	0000 0101
36	35	0000 0111
180		0000 0000
216		0000 0111
		0001 1100
		0010 0011 = 35

Figure 3.5

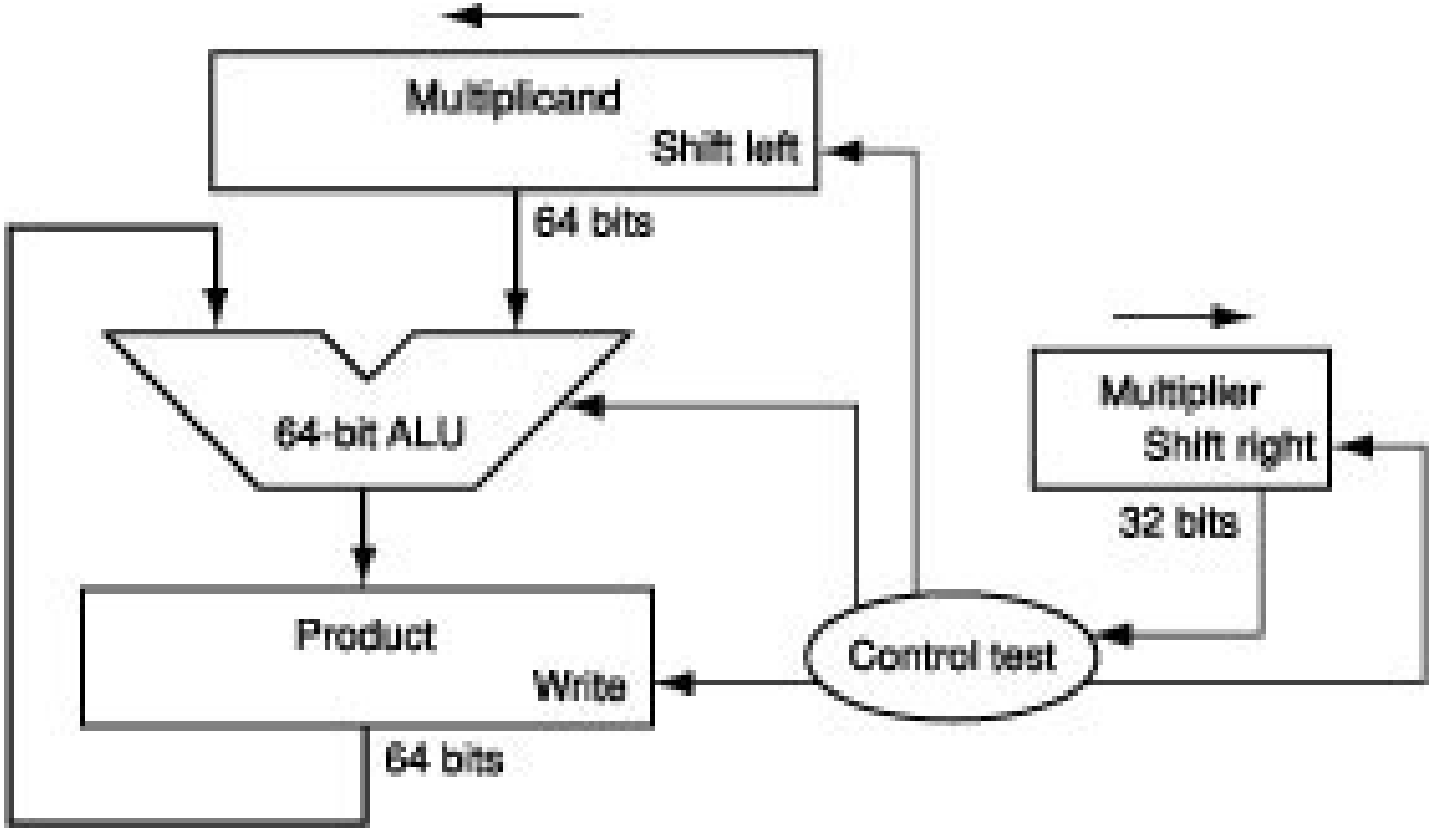
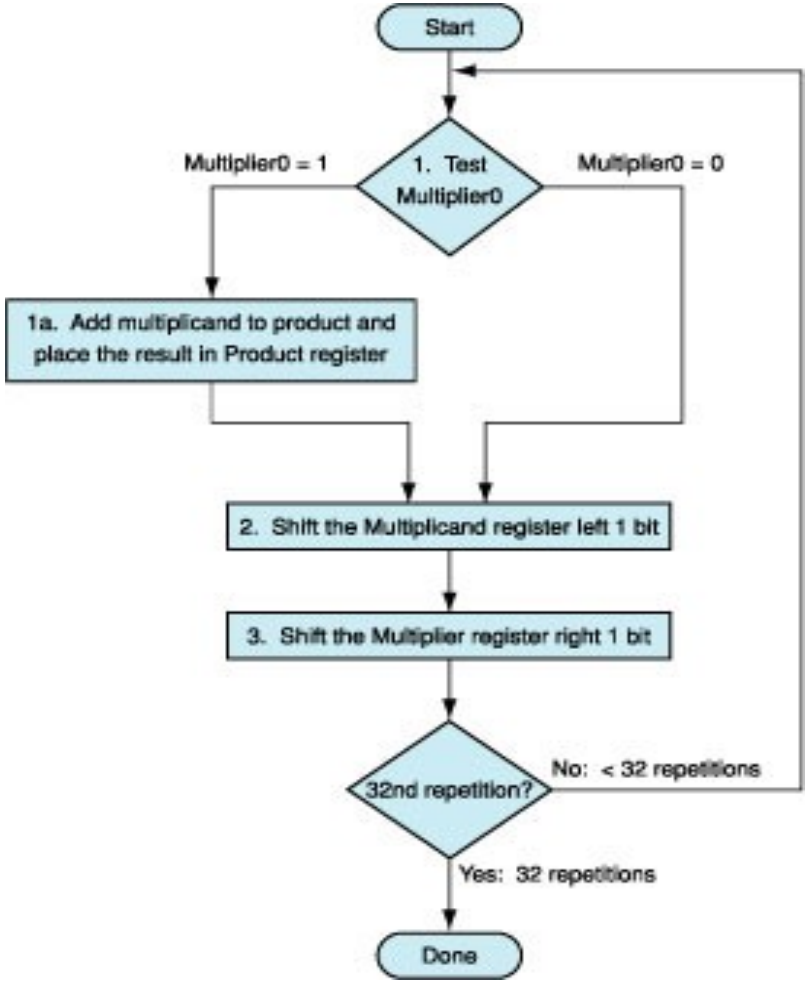

Multiplicand
Shift left
64 bits
64-bit ALU
Multiplier
Shift right
32 bits
Product
Write
Control test
64 bits

Figure 3.6


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

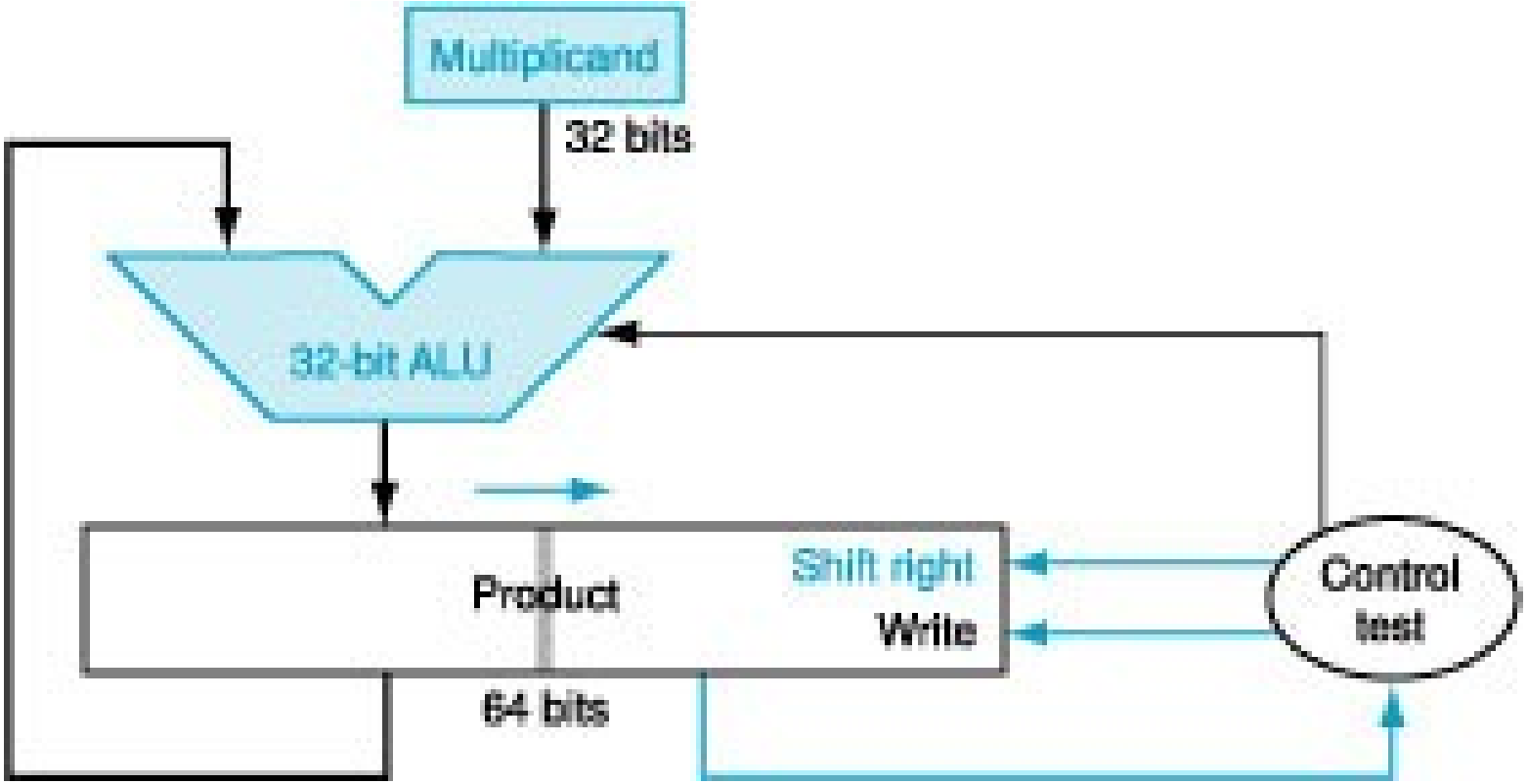
Example (9x11)

Iteration	Multiplier	Multiplicand	Product
0 (Start)	1011	0000 1001	0000 0000
(End of) 1	0101	0001 0010	0000 1001
2	0010	0010 0100	0001 1011
3	0001	0100 1000	0001 1011
4	0000	1001 0000	0110 0011

Problems with this diagram

- Need two 64 bit registers, but one is temporary.
- The Multiplicand must be moved to this temporary register
- The Multiplier is changed, so another register is needed.
- 64 bit adder is used, though only 32 bits should be needed.

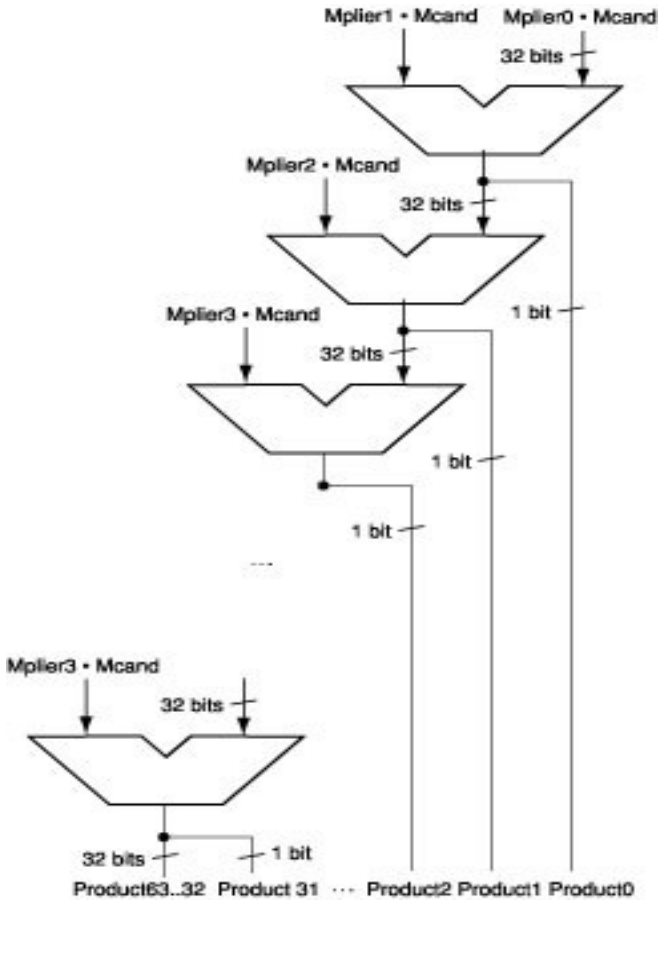
Figure 3.7


Multiplicand
32 bits
32-bit ALU
Product
Shift right
Write
Control
test
64 bits

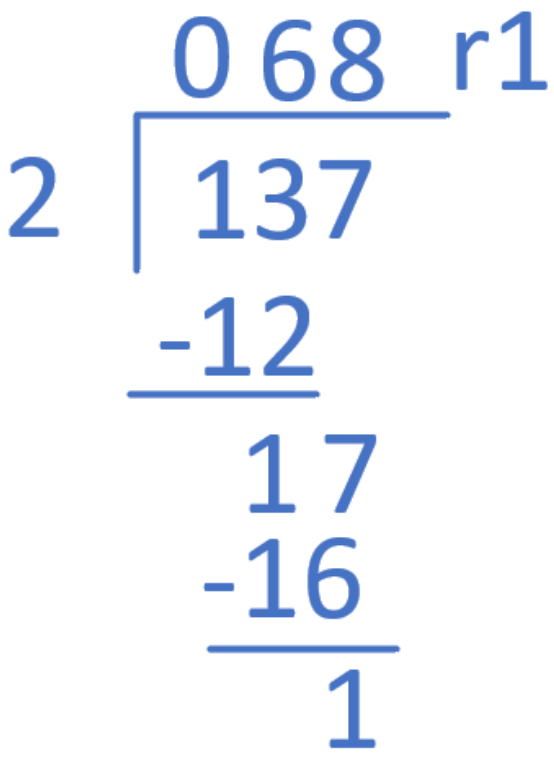
Example (9x11)

Iteration	Multiplicand	Product
0 (Start)	1001	0000 1011
1 (Note shift is after add)	1001	0100 1101
2	1001	0110 1110
3	1001	0011 0111
4	1001	0110 0011

Faster Multiplicer



Division



Division

7/2

Step 1: Check to see if 0010 > 0000 by subtraction

0010 | 0000 0111

+111 0

+111 0

Negative, so did not work

Step 2: Right shift and check to see if greater than by subtraction

```
          00
0010 | 0000 0111
     +11 10
     ---------
     +11 11
```

Negative, so did not work

Step 3: Right shift and check to see if greater than by subtraction

```
          001
0010 | 0000 0111
     +1 110
     ---------
       0 001
```

Positive, so worked. Leave value, put a 1 in the quotient and cont.

Division

- Figures 3.10, 3.11, and 3.12 in your book
- You are responsible for figure 3.12
- Math for division
  - Example 74/8 from text, page183

Figure 3.10

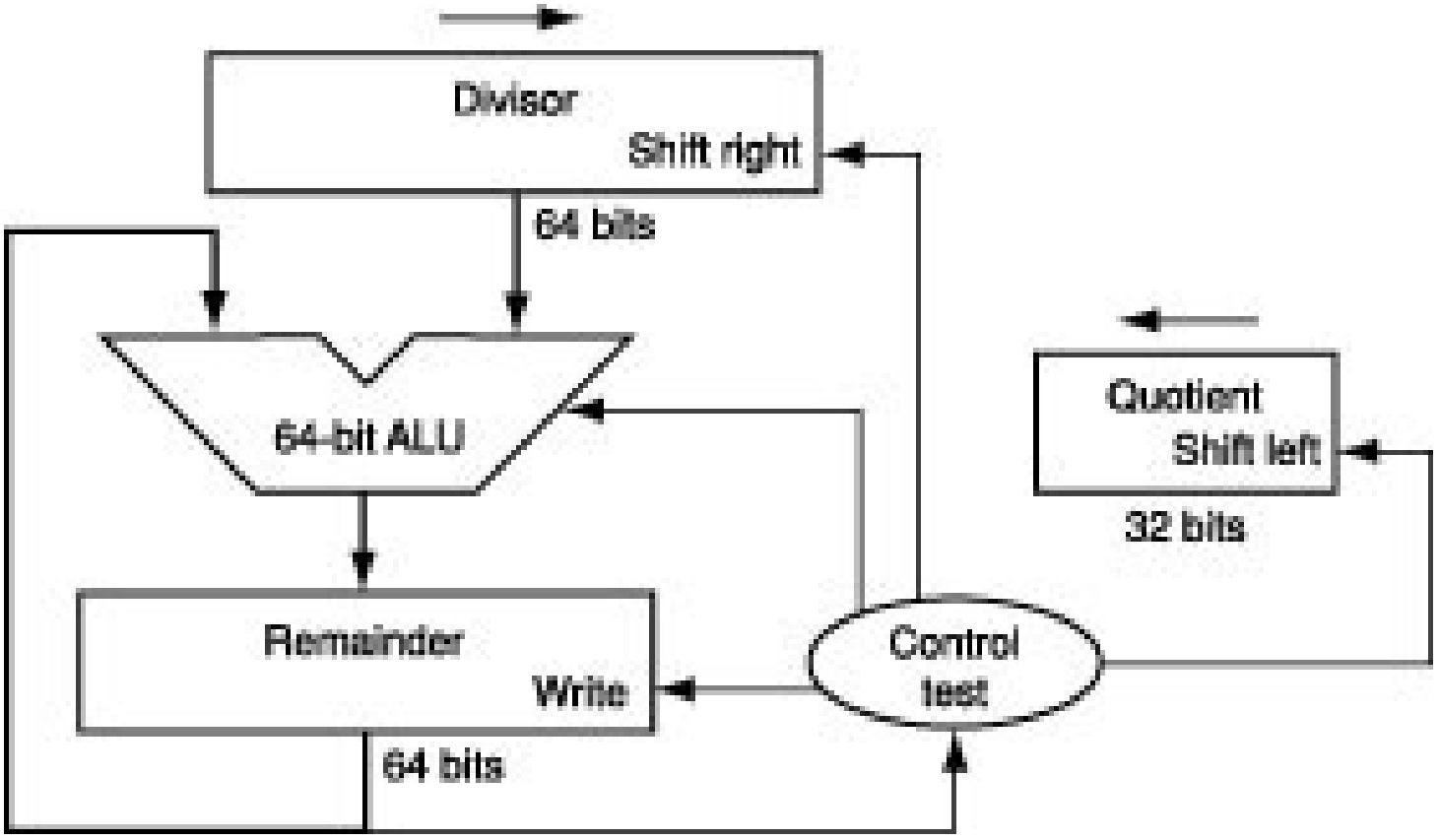
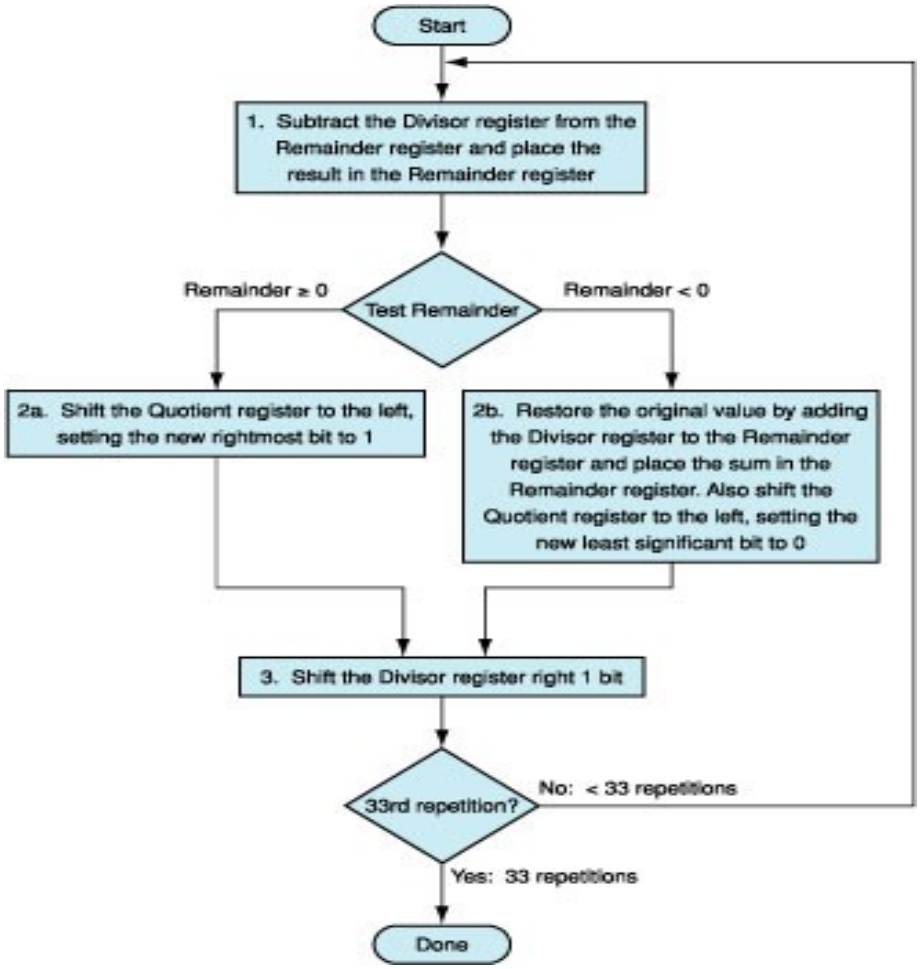

Divisor
Shift right
64 bits
64-bit ALU
Quotient
Shift left
32 bits
Remainder
Write
Control
test
64 bits

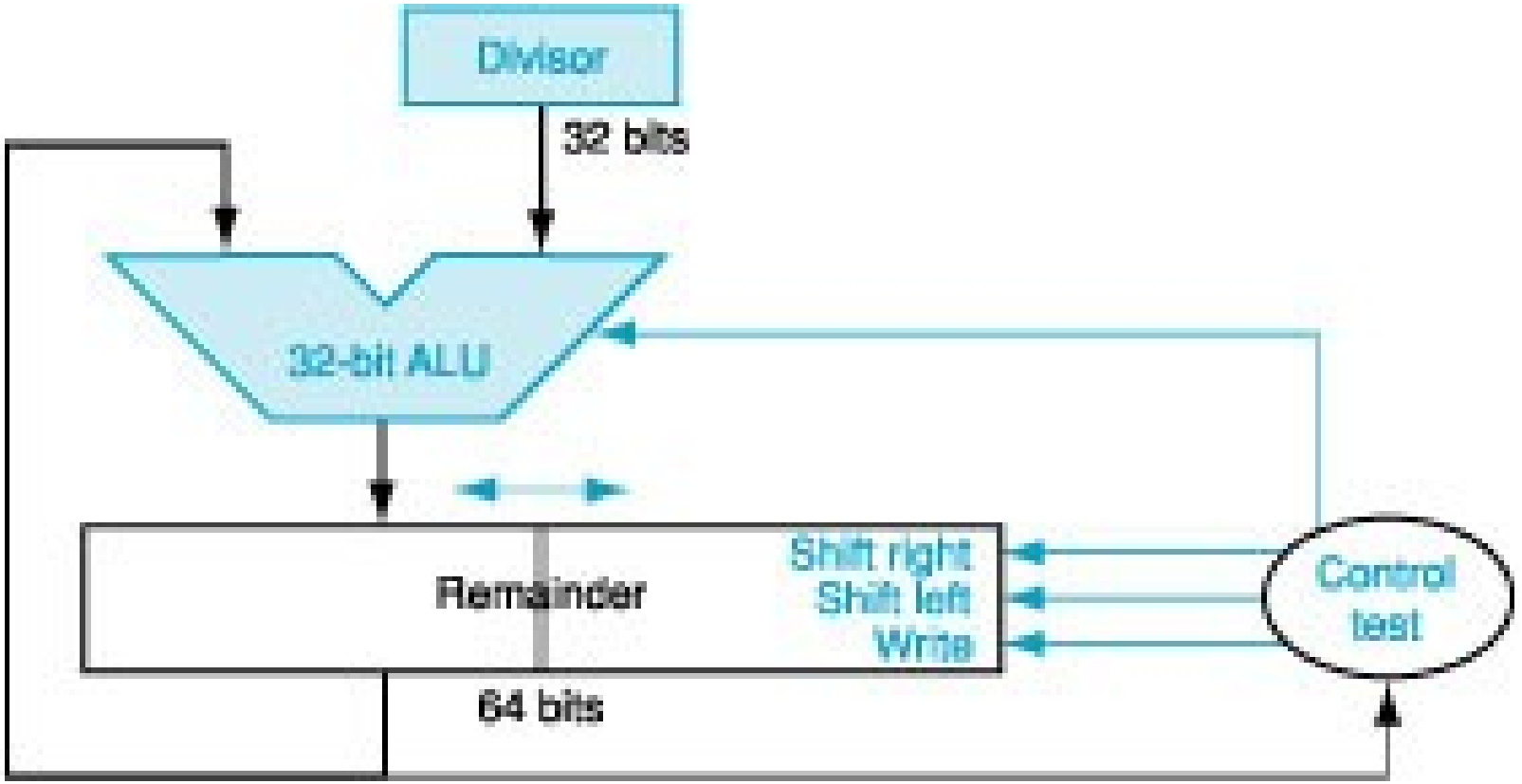
Figure 3.11


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 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
33rd repetition?
No: < 33 repetitions
Yes: 33 repetitions
Done

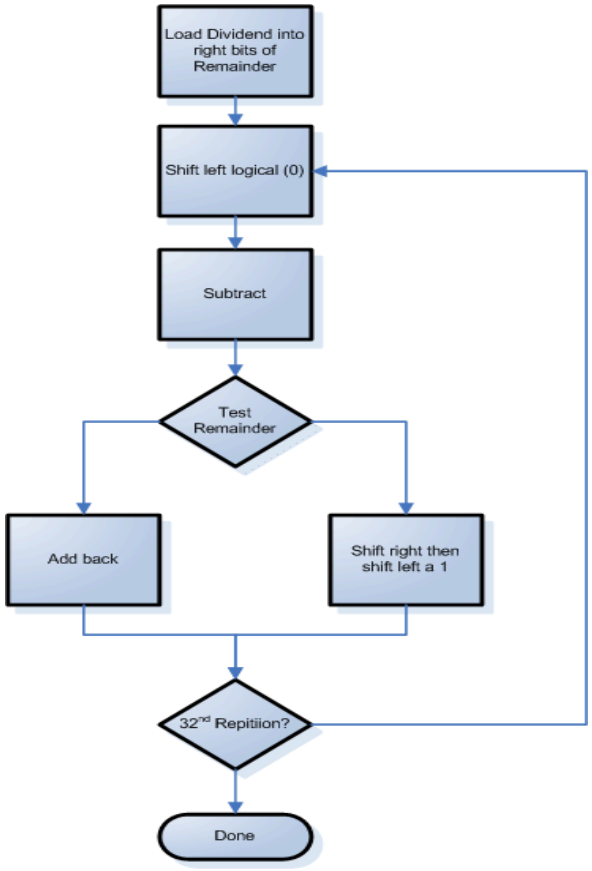
Example Division (7/2) (note: obviously must only effect last 2 bits)

Iteration (end of step)	Quotient	Divisor	Remainder
0	0000	0010 0000	0000 0111
1	0000	0001 0000	0000 0111
2	0000	0000 1000	0000 0111
3	0000	0000 0100	0000 0111
4	0001	0000 0010	0000 0011
5	0011	0000 0001	0000 0001

Figure 13.3 (note: Same hardware as Figure 3.7



Flow Chart


Load Dividend into right bits of Remainder
Shift left logical (0)
Subtract
Test Remainder
Add back
Shift right then shift left a 1
32nd Repitiion?
Done

Example 1 (7/2)

Iteration	Divisor	Complement	Remainder
0	0010	1110	0000 0111
1a	0010	1110	0000 1110
1b	0010	1110	1110 1110
1c	0010	1110	0000 1110 (note minus removed)
2a	0010	1110	0001 1100
2b	0010	1110	1111 1100
2c	0010	1110	0001 1100 (note minus removed)
3a	0010	1110	0011 1000
3b	0010	1110	0001 1000
3c	0010	1110	0001 1001 (note 2 shifts)
4a	0010	1110	0011 0010
4b	0010	1110	0001 0010
4c	0010	1110	0001 0011 (note 2 shifts)

Example 2 (9/2)

Iteration	Divisor	Complement	Remainder
0	0010	1110	0000 1001
1a	0010	1110	0001 0010
1b	0010	1110	1111 0010
1c	0010	1110	0001 0010
2a	0010	1110	0010 0100
2b	0010	1110	0000 0100
2c	0010	1110	0000 0101
3a	0010	1110	0000 1010
3b	0010	1110	1110 1010
3c	0010	1110	0000 1010
4a	0010	1110	0001 0100
4b	0010	1110	1111 0100
4c	0010	1110	0001 0100

Floating Point Numbers

- Floating point numbers are numbers that have a single digit to the left of the decimal point.
- Normalized number has no leading 0s
- Floating point numbers have a base 2 representation

Terms

- Fraction (or mantissa in normalized floating point numbers, since as we will see only the part to the left of the decimal is kept).
- Exponent
- Overflow and underflow
- Single precision and Double precision

Constants

- MAX_VALUE
- MIN_VALUE
- NEGATIVE_INFINITY
- POSITIVE_INFINITY
- NAN

Floating Point format and Examples