Computer Organization

605.204

Module Two

Part Two

Integer Arithmetic

Summer 2014

Module Two

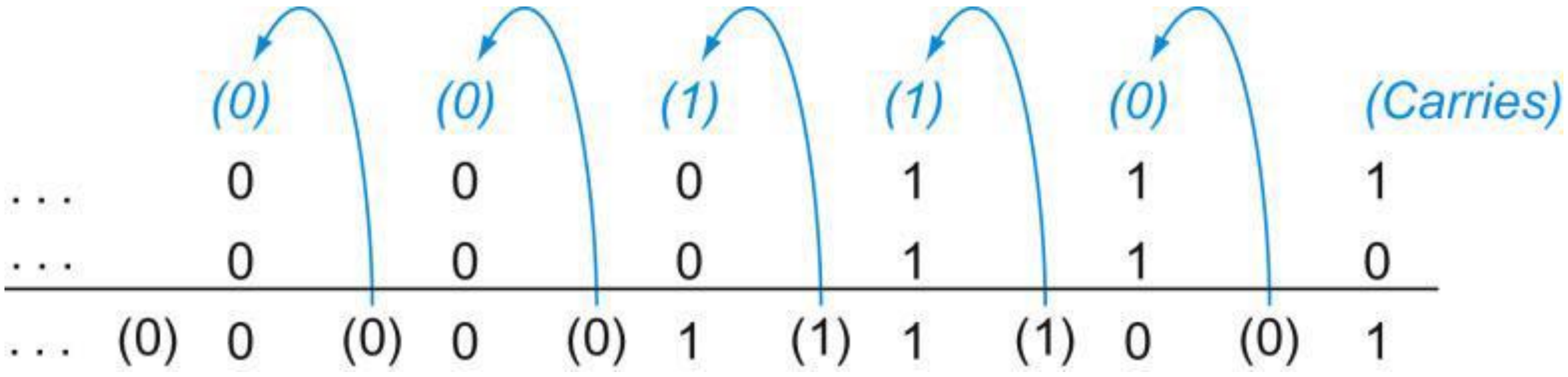
- Part Two
- In this presentation, we are going to talk about :
- Integer Addition and Subtraction
- Overflow
- Integer Multiplication and Division

Previously

- Previously we talked about:
- Integer Computer Numbers
- Now: Integer Arithmetic

Integer Addition

- Bitwise right to left



```
      (0)      (0)      (1)      (1)      (0)      (Carries)
...    0        0        0        1        1        1
...    0        0        0        1        1        0
... (0) 0   (0) 0   (0) 1   (1) 1   (1) 0   (0) 1
```

- Always a SUM and a CARRY

Integer Addition & Subtraction

- Just like in grade school (carry/borrow 1s)

```
   110
   0111            0111            0110
+  0110         -  0110         -  0101
   1101            0001            0001
```

- Two's complement operations
  - subtraction using addition of negative numbers

```
   0111     7
+  1010    -6
1  0001
```

Change the Sign and Add

Overflow !

- **Overflow** is simply a result too large for the finite computer word:

$$
\begin{array}{rrr}
 & 0111 & 7 \\
+ & 0001 & 1 \\
\hline
 & 1000 & -8?
\end{array}
$$

BTW: *The **overflow** term is somewhat misleading,*

$$
\begin{array}{rrr}
 & 0111 & 7 \\
+ & 1110 & -2 \\
\hline
\boxed{1} & 0101 & 5
\end{array}
$$

it does not mean a Carry “flowed over”

Detecting Overflow

- No overflow when adding a positive and a negative number
- No overflow when signs are the same for subtraction
- **Overflow occurs when the value affects the sign:**
  - when adding two positives yields a negative
  - or, adding two negatives gives a positive
  - or, subtract a negative from a positive and get a negative
  - or, subtract a positive from a negative and get a positive

When Overflow happens

- An exception (interrupt) occurs
  - Control jumps to predefined address for exception processing
  - Interrupted address is saved for possible resumption
- Details based on software system / language
  - example: flight control vs. homework assignment
- Don't always want to detect overflow
  - some MIPS instructions: `addu, addiu, subu`

Overflow Discussion

- Consider the operations A + B, and A – B
  - Can overflow occur if B is 0 ?
  - Can overflow occur if A is 0 ?

We will discuss this question, this week during the Office Hours

Integer Multiplication

- More complicated than addition
  - accomplished via shifting and addition

- More time and more area

- Let's look at 3 versions based on grade school algorithm

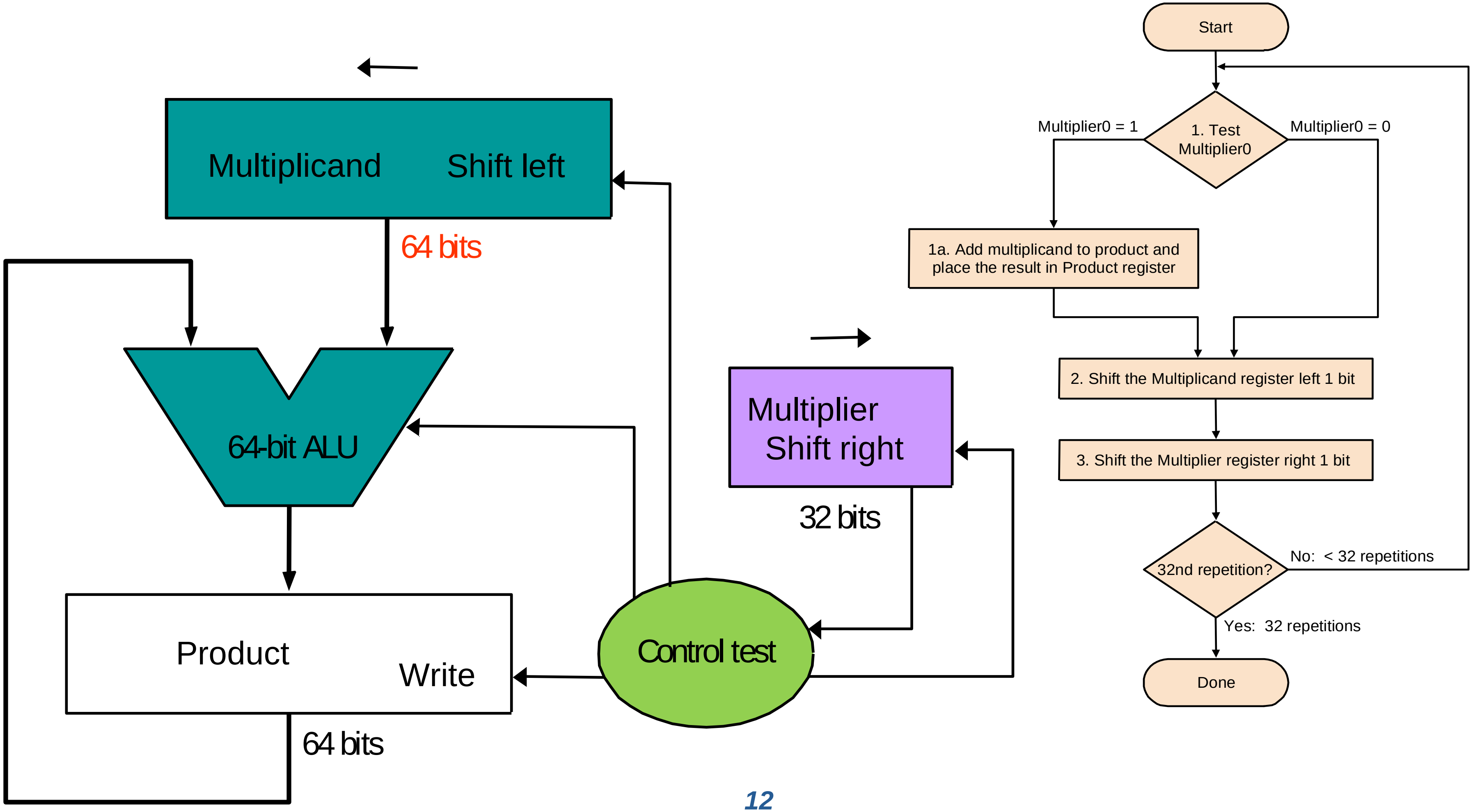
```
   0010    (multiplicand)
 x 1011    (multiplier)
```

- Negative numbers: convert to positive, multiply, adjust the sign

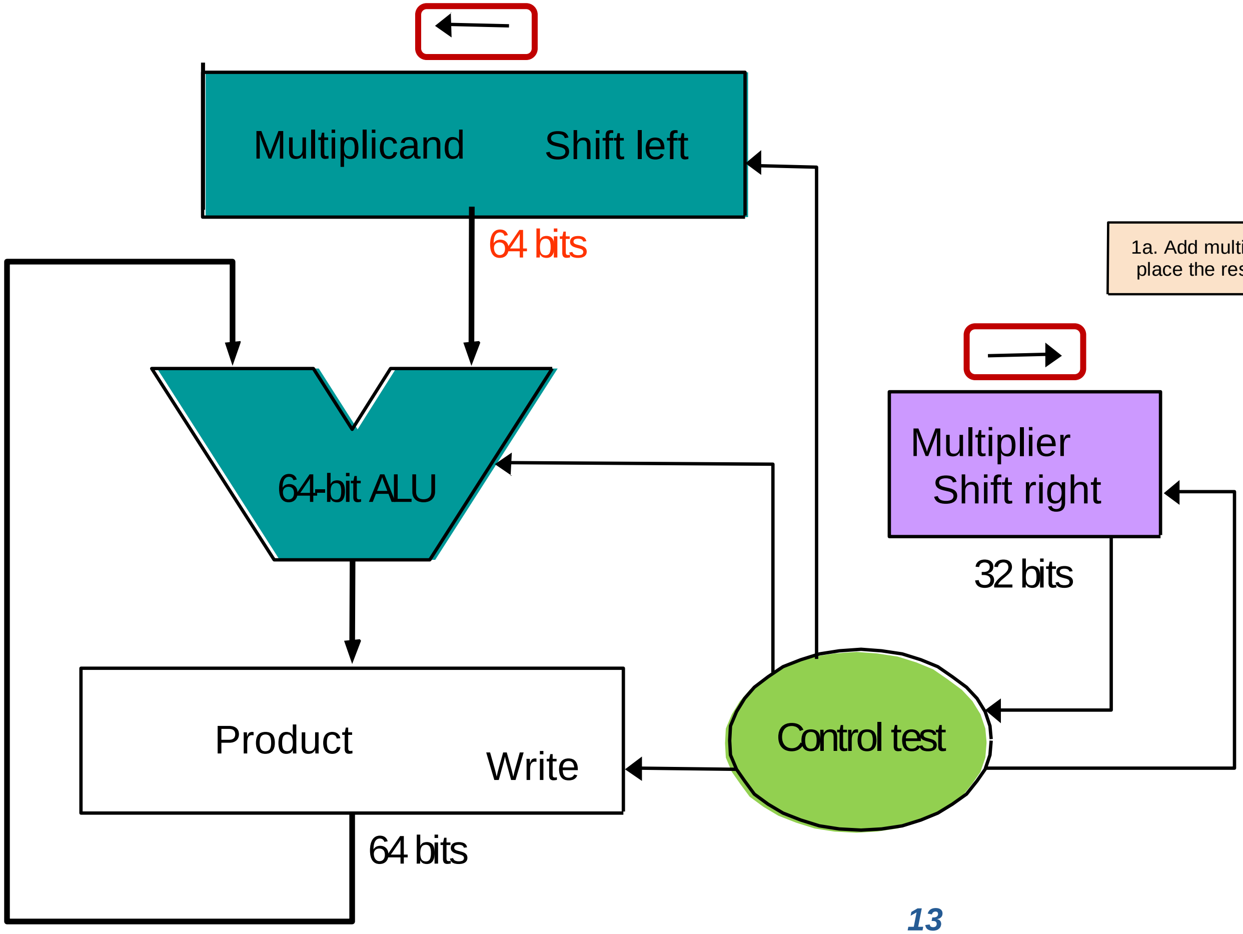
Grade school multiply

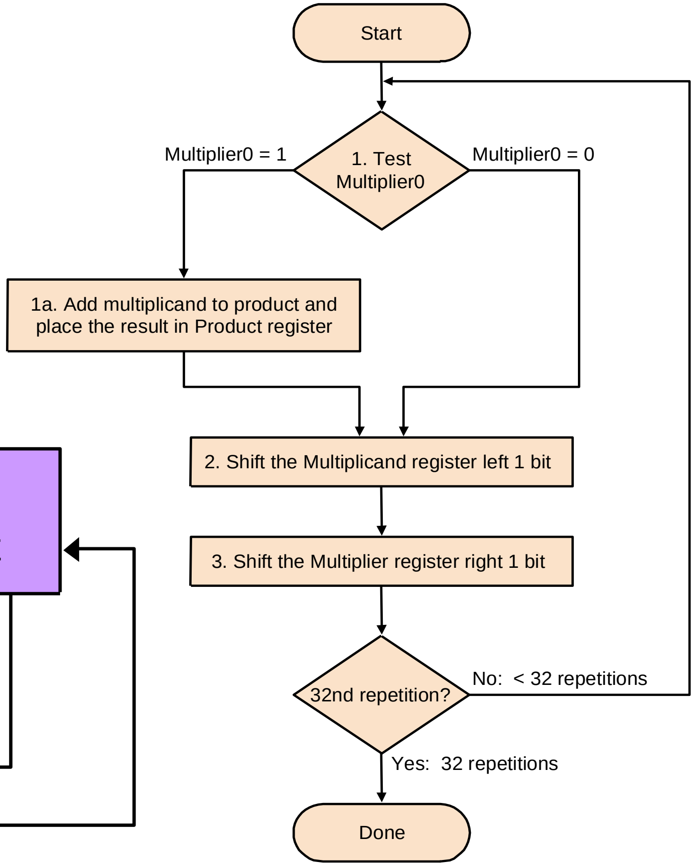
$$\begin{array}{r l}
0010 & \text{(multiplicand)}\ \ 2_{10} \\
\times\ \ \underline{1011} & \text{(multiplier)}\ \ \ \ 11_{10} \\
0010 & \\
0010 & \\
0000 & \\
\underline{0010} & \\
0010110 & 22_{10}
\end{array}$$

First Version

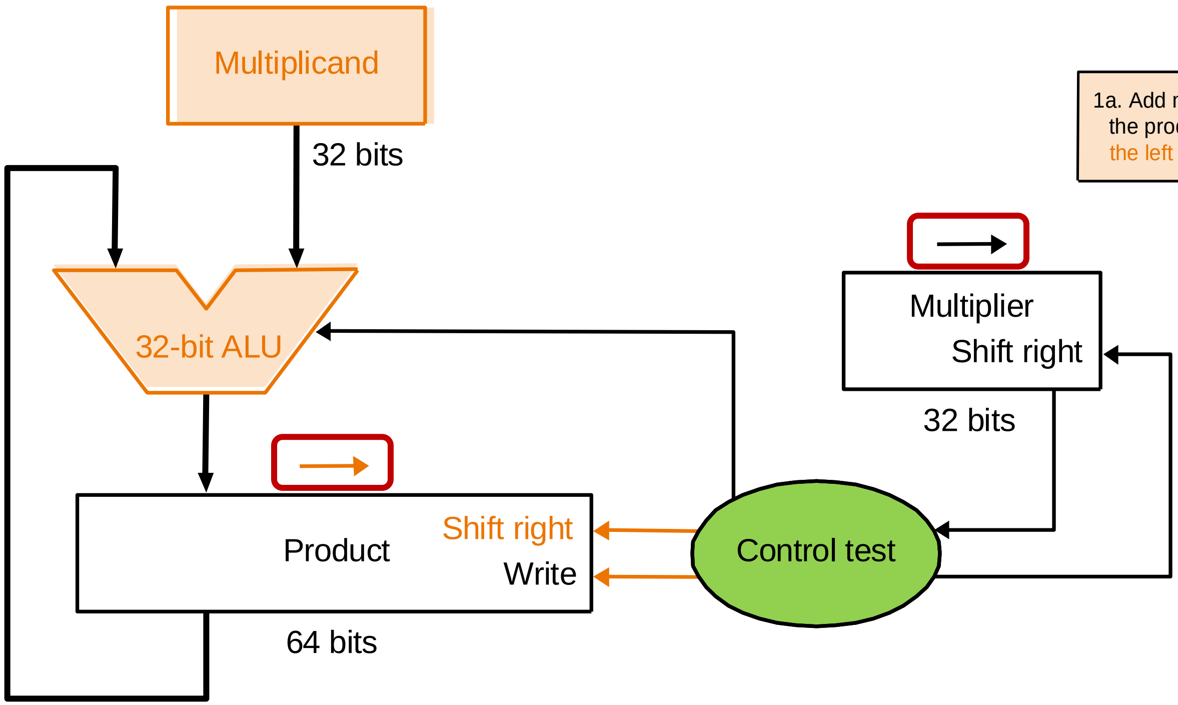


First Version




Start
Multiplier0 = 1
1. Test Multiplier0
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

Second Version



Multiplicand

32 bits

32-bit ALU

Multiplier

Shift right

32 bits

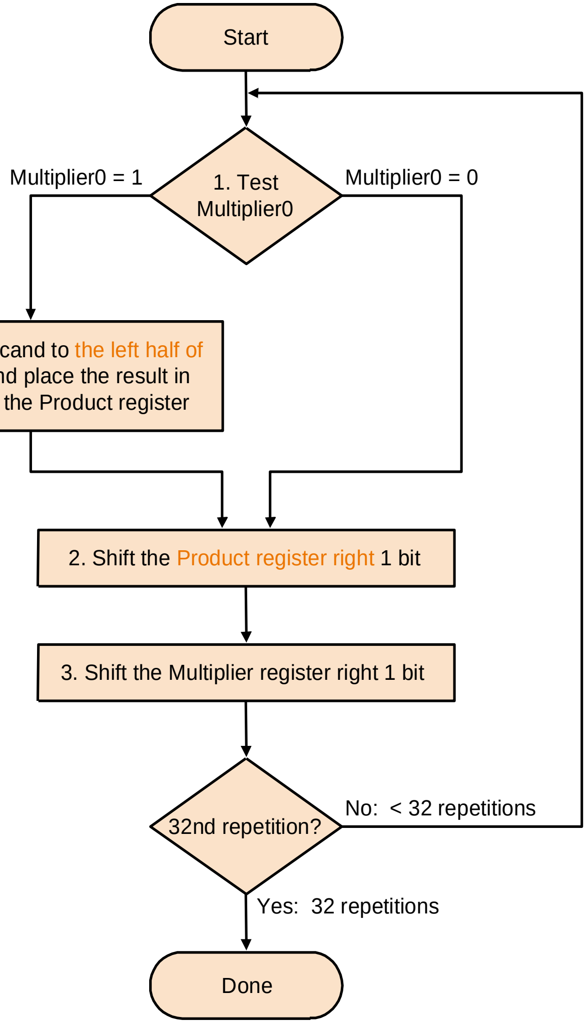
Product

Shift right

Write

64 bits

Control test



Start

1. Test Multiplier0

Multiplier0 = 1

Multiplier0 = 0

1a. Add multiplicand to the left half of the product and place the result in the left half of the Product register

2. Shift the Product register right 1 bit

3. Shift the Multiplier register right 1 bit

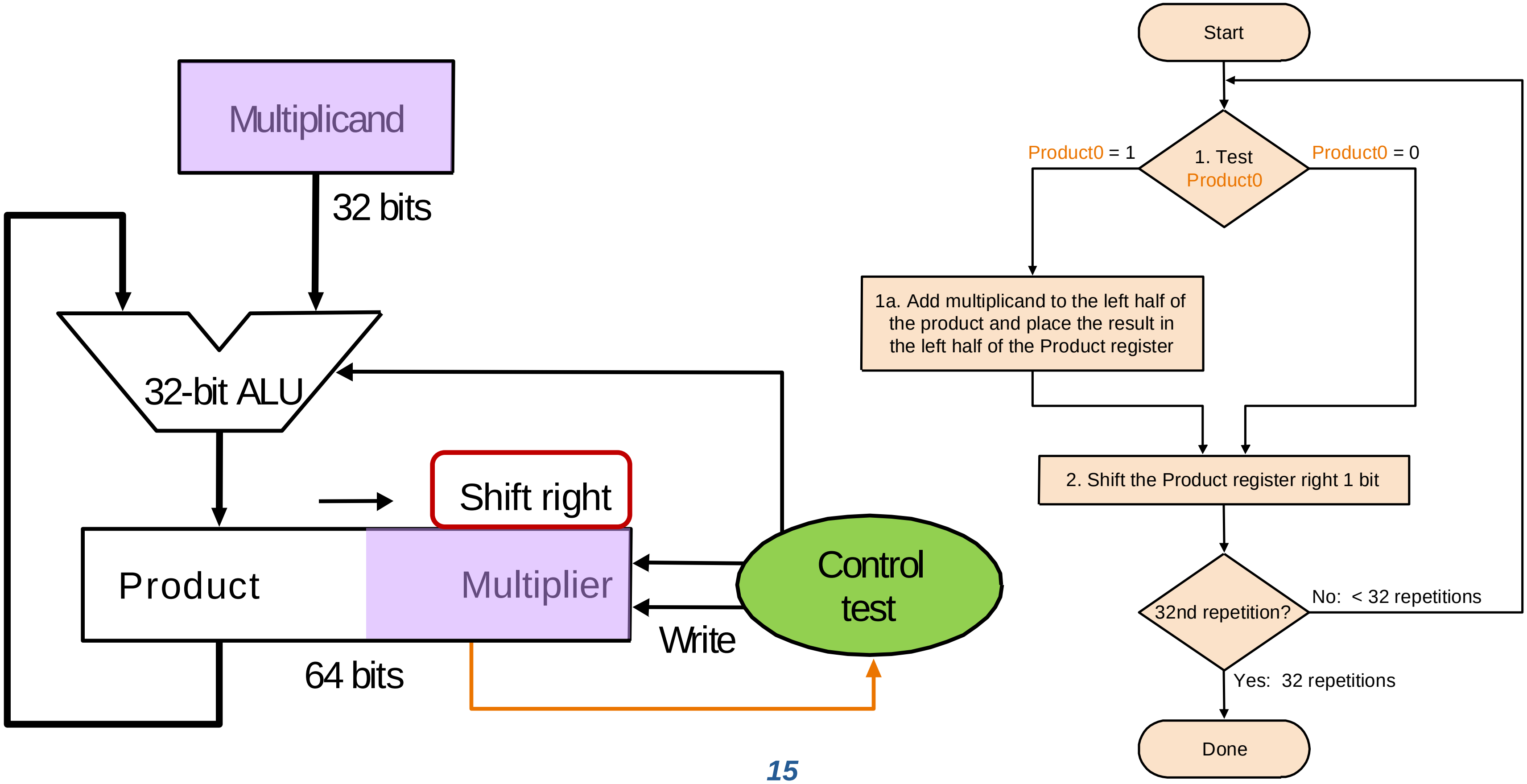
32nd repetition?

No: < 32 repetitions

Yes: 32 repetitions

Done

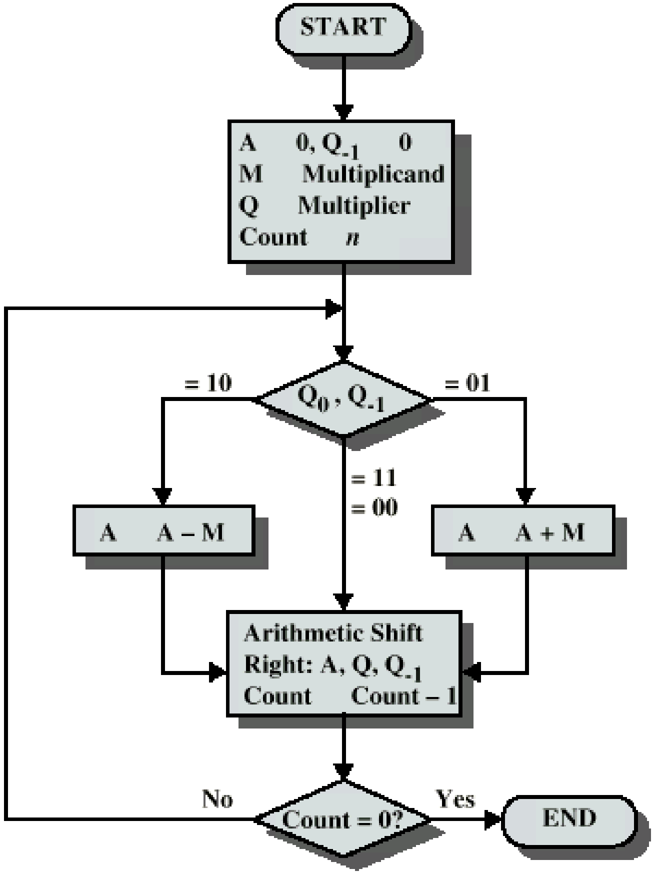
Third Version



Booth’s Algorithm

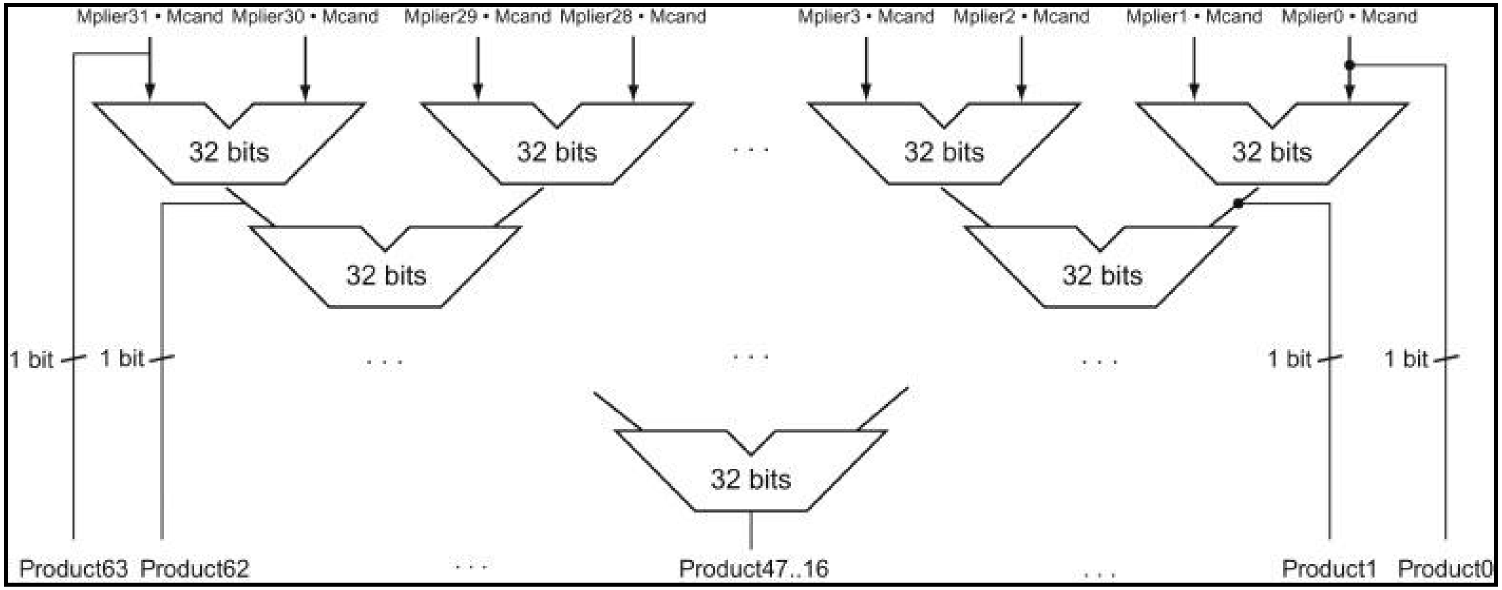
A.D. Booth

Technique for multiplying two n-bit **two’s complement integers** without regard to sign of multiplier or multiplicand.



Faster Multiply

- Use 16 + 8 + 4 + 2 + 1 = 31 adders; operate in parallel

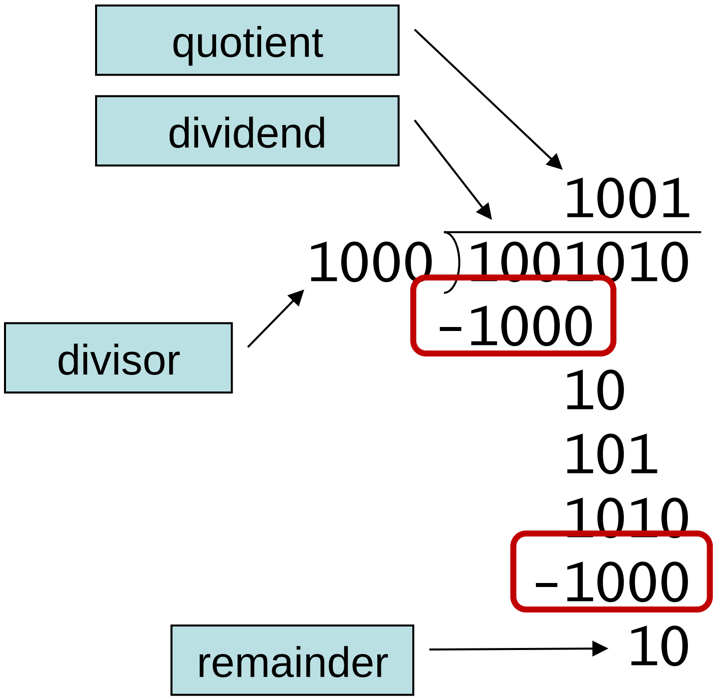


- Combine the results; time = 5

- Use shift left 1 = multiply by 2

Division

- Long division from fourth grade
- Binary digits
- If the divisor <= dividend digits
  - then put 1 in the quotient
  - and subtract
  - otherwise put 0 in the quotient,
  - and bring down the next digit
  - Until remainder less than divisor.



quotient

dividend

divisor

remainder

1001

1000) 1001010

-1000

10

101

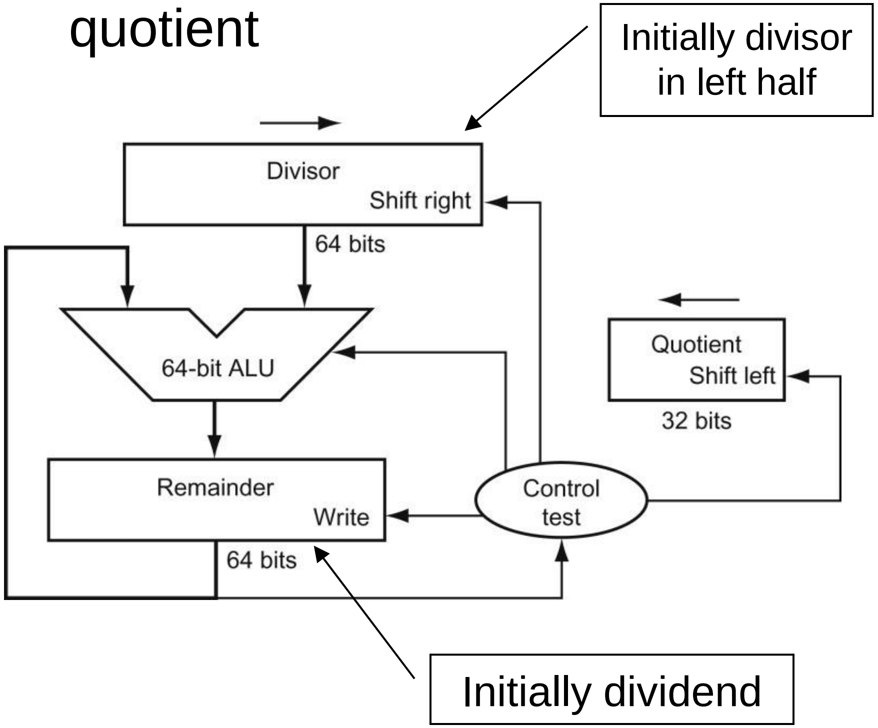
1010

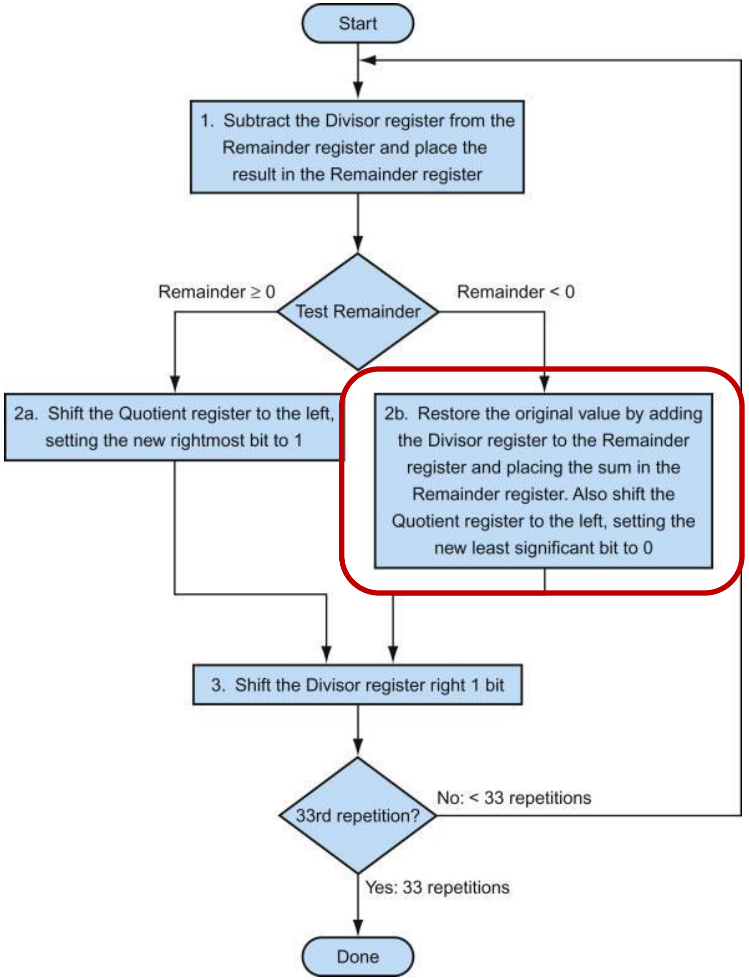
-1000

10

Restoring Division

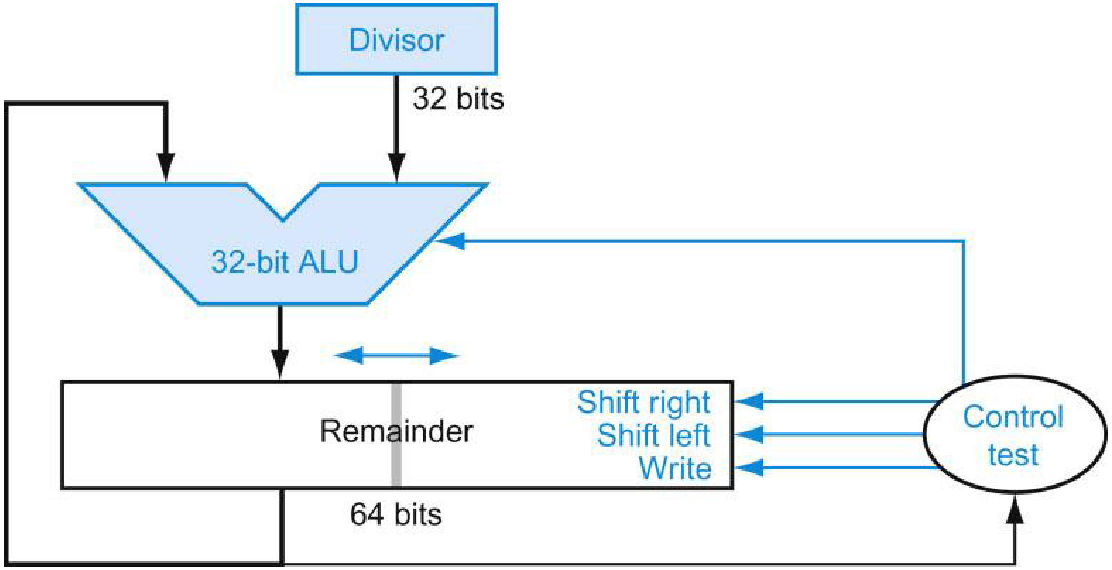
- Subtract
- Step two sets the bits in the quotient




Start
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 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

Improved Division Hardware

- As previously, the dividend is placed in the Remainder register to begin.
- At the end, the remainder register contains both remainder and quotient
- Same as Multiply
- Hi and Lo registers



Signed numbers Division

- Dividend = Quotient x Divisor + Remainder
- The sign of the Quotient is negative if the signs of the Divisor and Dividend are different; otherwise, positive.
- Rule: Dividend and Remainder always have the same sign.
- Discussion:
  - Shift right for divide; complement of shift left for multiply ?

Integer Summary

- Two’s complement representation for integers.
- Subtract - Change the Sign and Add
- Overflow - Value too large for the number of digits
- Multiply - Booth’s Algorithm
- Divide - Signed numbers
- Next: Floating Point Numbers