Integer Multiplication

Multiplying two numbers:

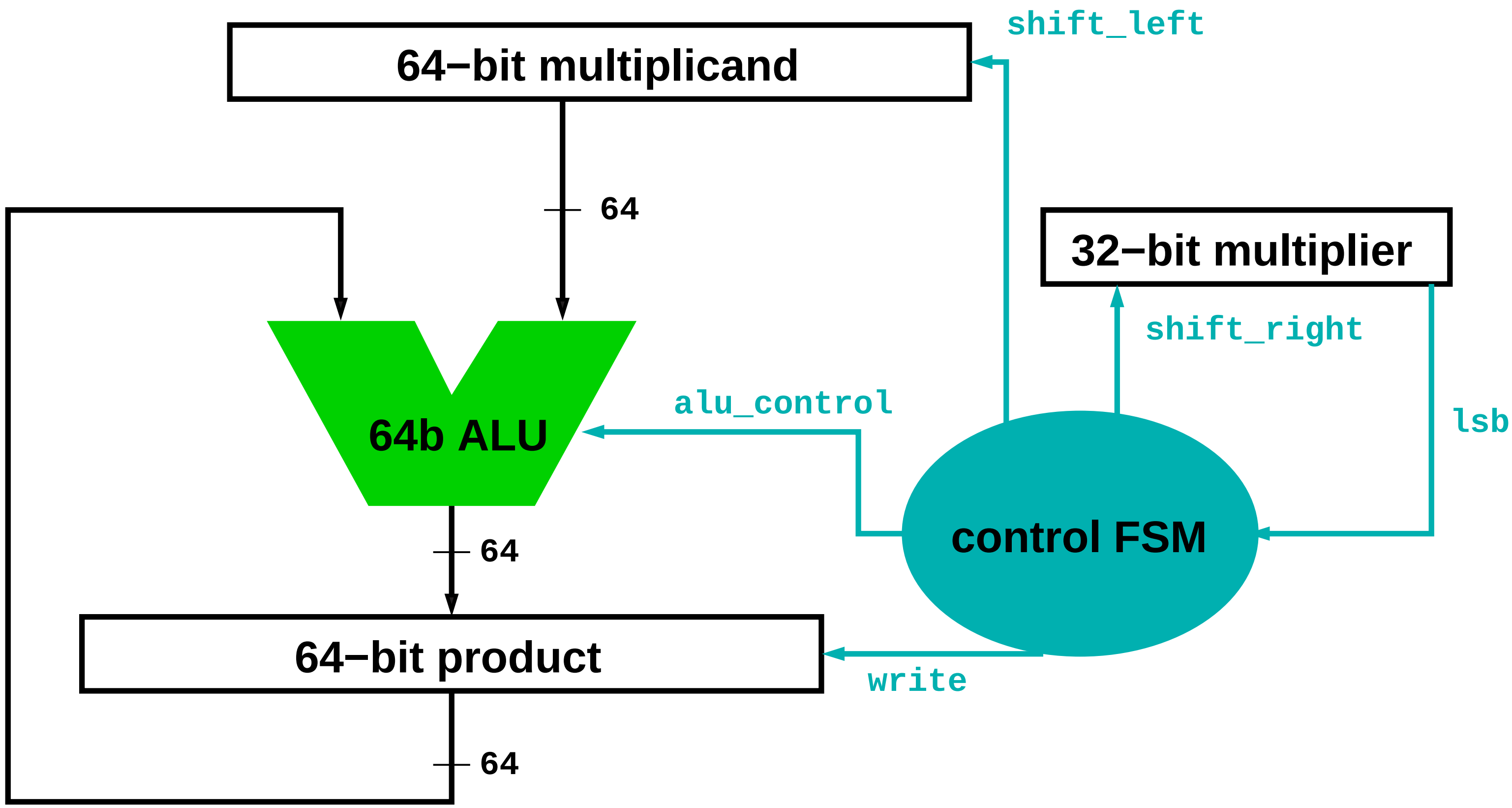
$$
\begin{array}{lcccccccc}
\text{multiplicand} & & & & & 1 & 0 & 1 & 0 \\
\text{multiplier} & & & & \times & 1 & 0 & 0 & 1 \\
\cline{5-9}
 & & & & & 1 & 0 & 1 & 0 \\
 & & & & 0 & 0 & 0 & 0 & \\
 & & & 0 & 0 & 0 & 0 & & \\
 & + & 1 & 0 & 1 & 0 & & & \\
\cline{2-9}
 & & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
\cline{2-9}
\end{array}
$$

m-bits $\times$ n-bits = $(m + n)$-bit result

m-bits: $2^m - 1$ is the largest number

$$\Rightarrow (2^m - 1)(2^n - 1) = 2^{m+n} - 2^m - 2^n + 1$$

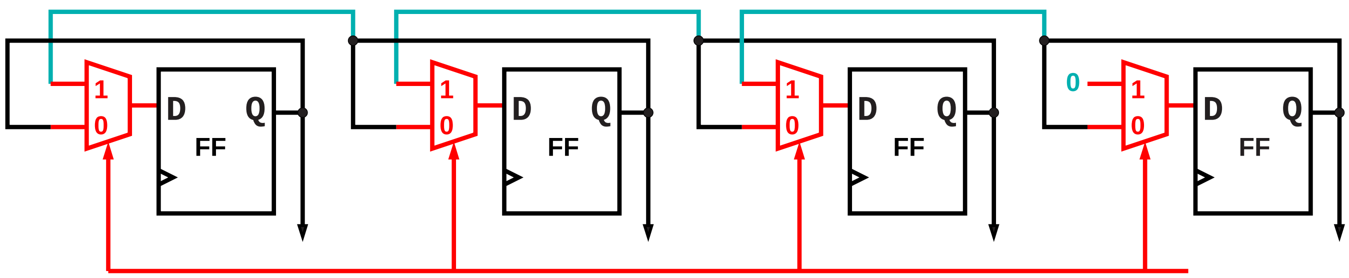
Integer Multiplication: First Try



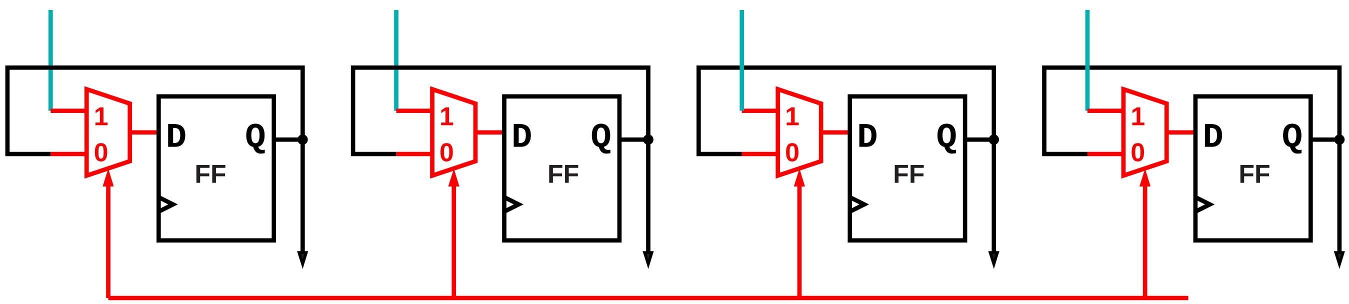
How do we build this?

Registers And Shift Registers

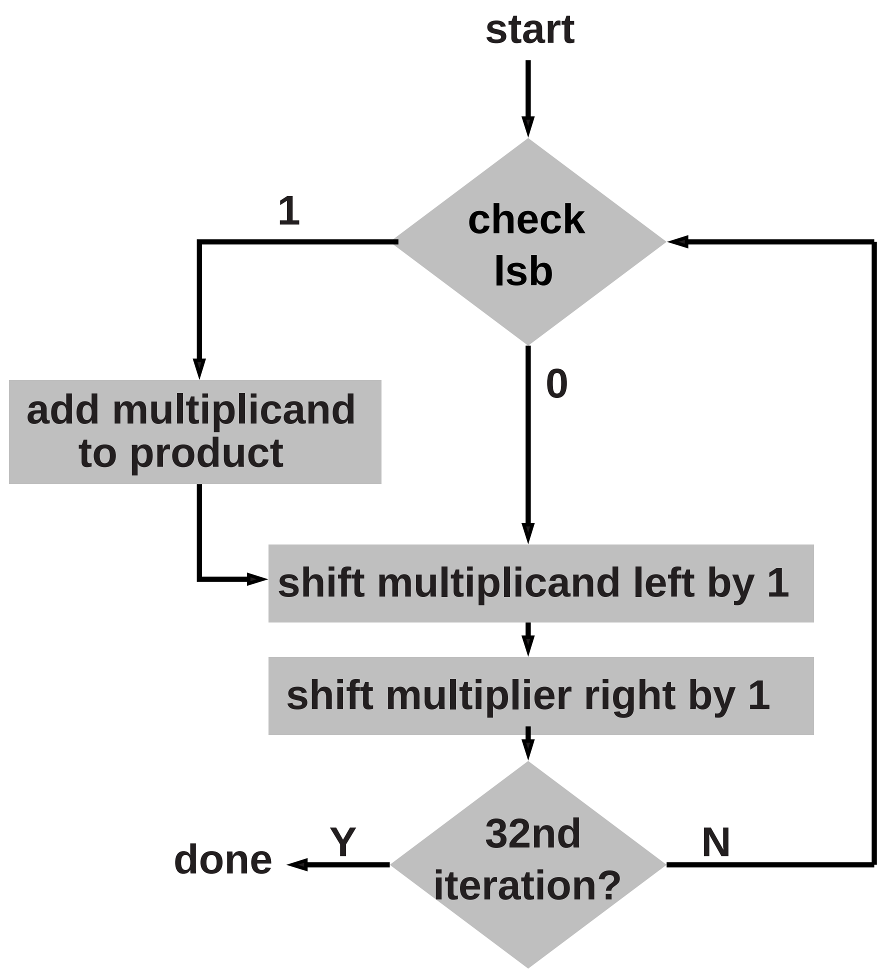
Register with shift left:



Register with write:



Control



start
check lsb
1
0
add multiplicand to product
shift multiplicand left by 1
shift multiplier right by 1
32nd iteration?
Y
N
done

Integer Multiplication

Observations:

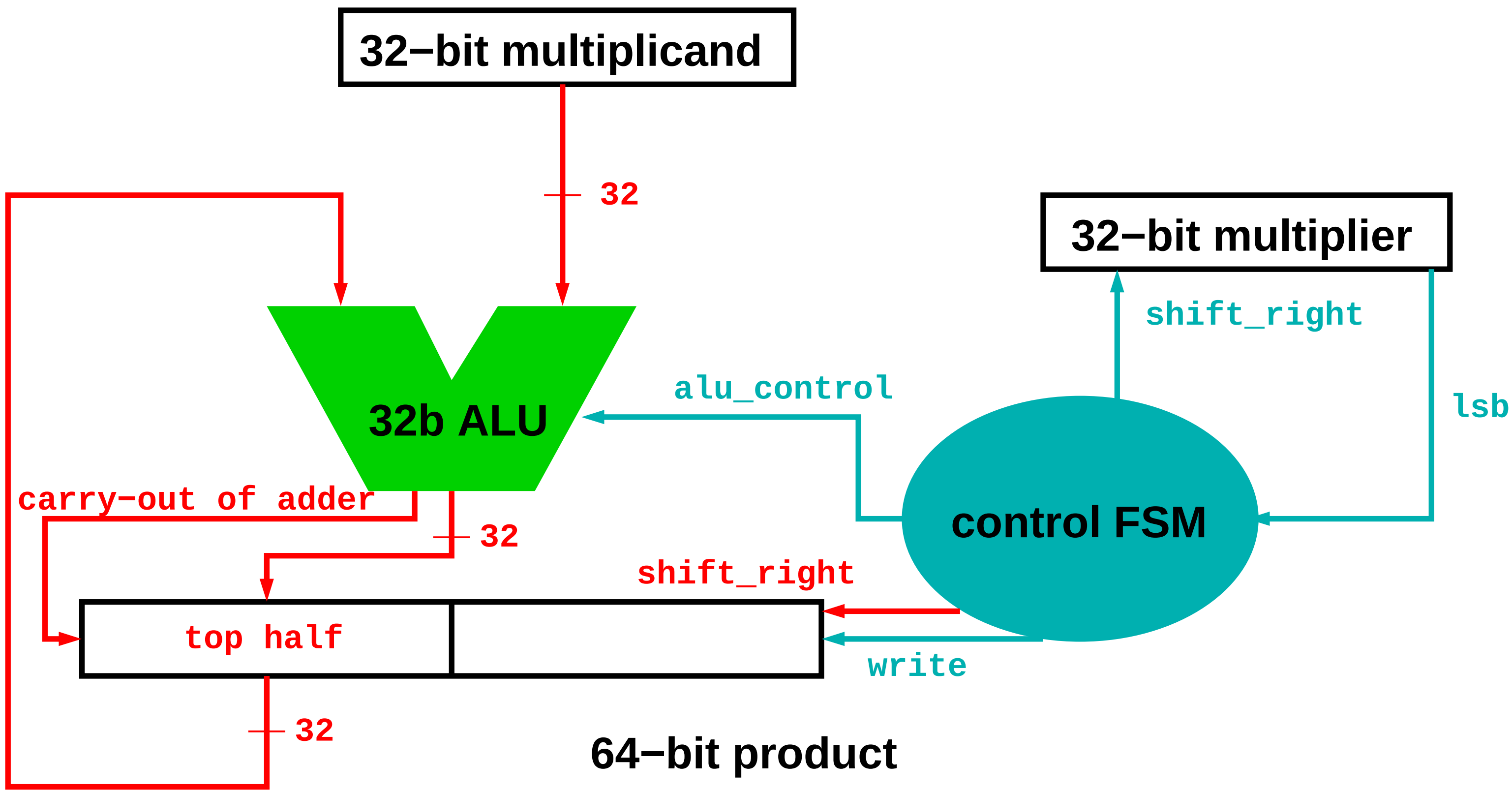
- 32 iterations for multiplication $\Rightarrow$ 32 cycles
- How long does 1 iteration take?
- Suppose 5% of ALU operations are multiply ops, and other ALU operations take 1 cycle.

  $$\Rightarrow CPI_{alu} = 0.05 \times 32 + 0.95 \times 1 = 2.55!$$
- Half of the bits of the multiplicand are zero

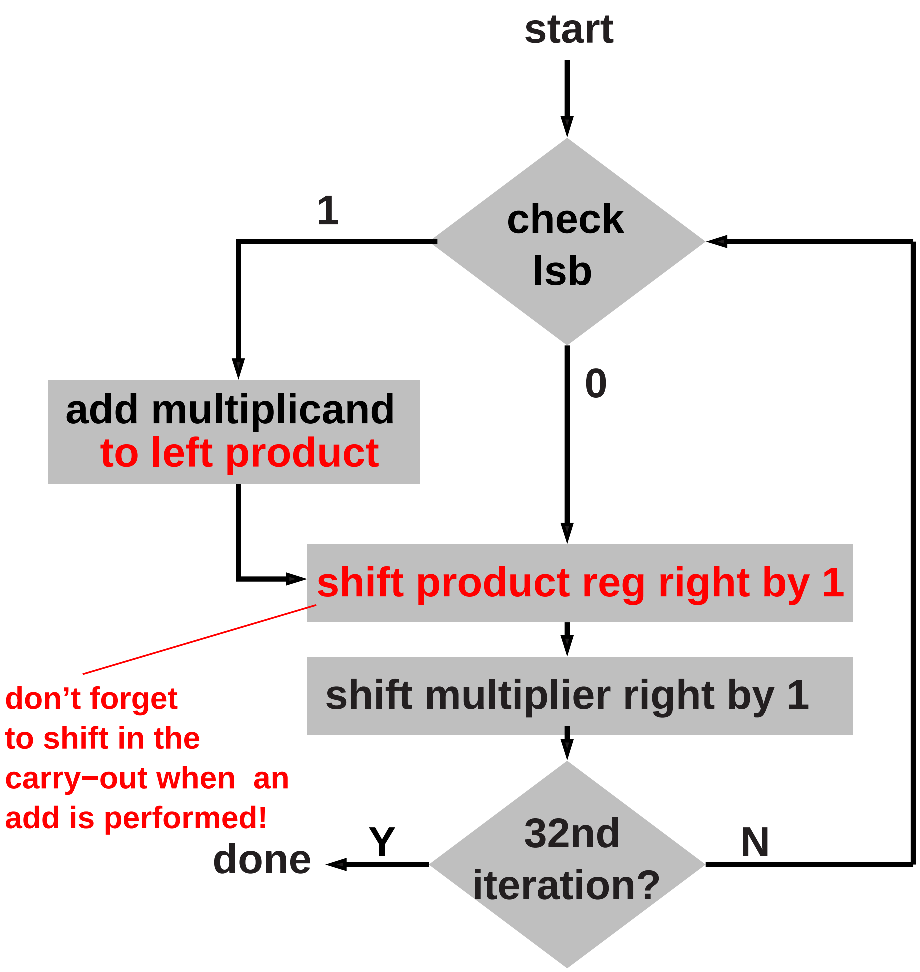
  $\Rightarrow$ 64-bit adder is wasted
- 0's inserted when multiplicand shifted left

  $\Rightarrow$ product LSBs don't change

Using A 32-Bit ALU



New Control

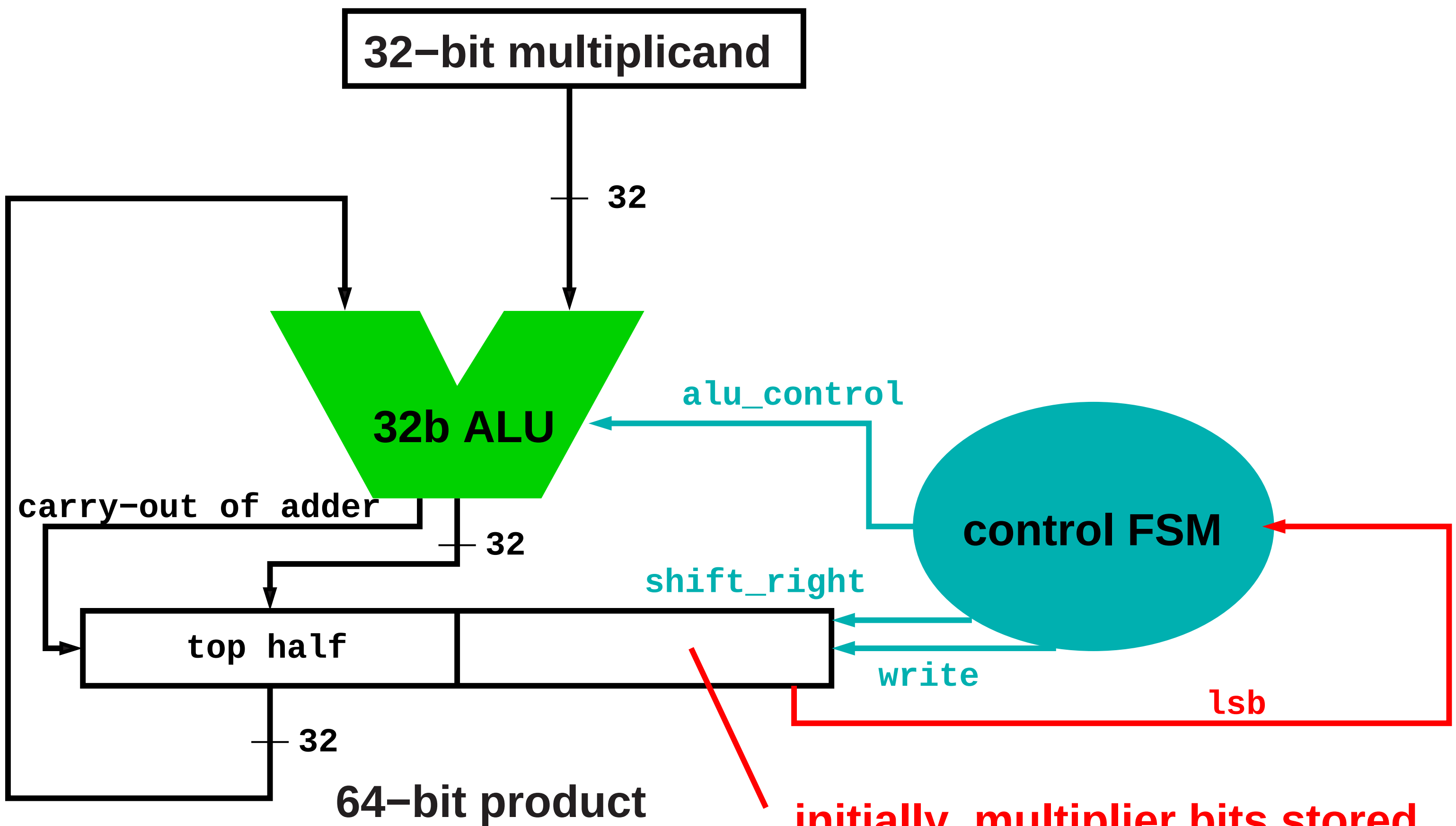


Bottom half of product register is zero initially.

Each iteration:
- adds 1 product bit
- loses one multiplier bit

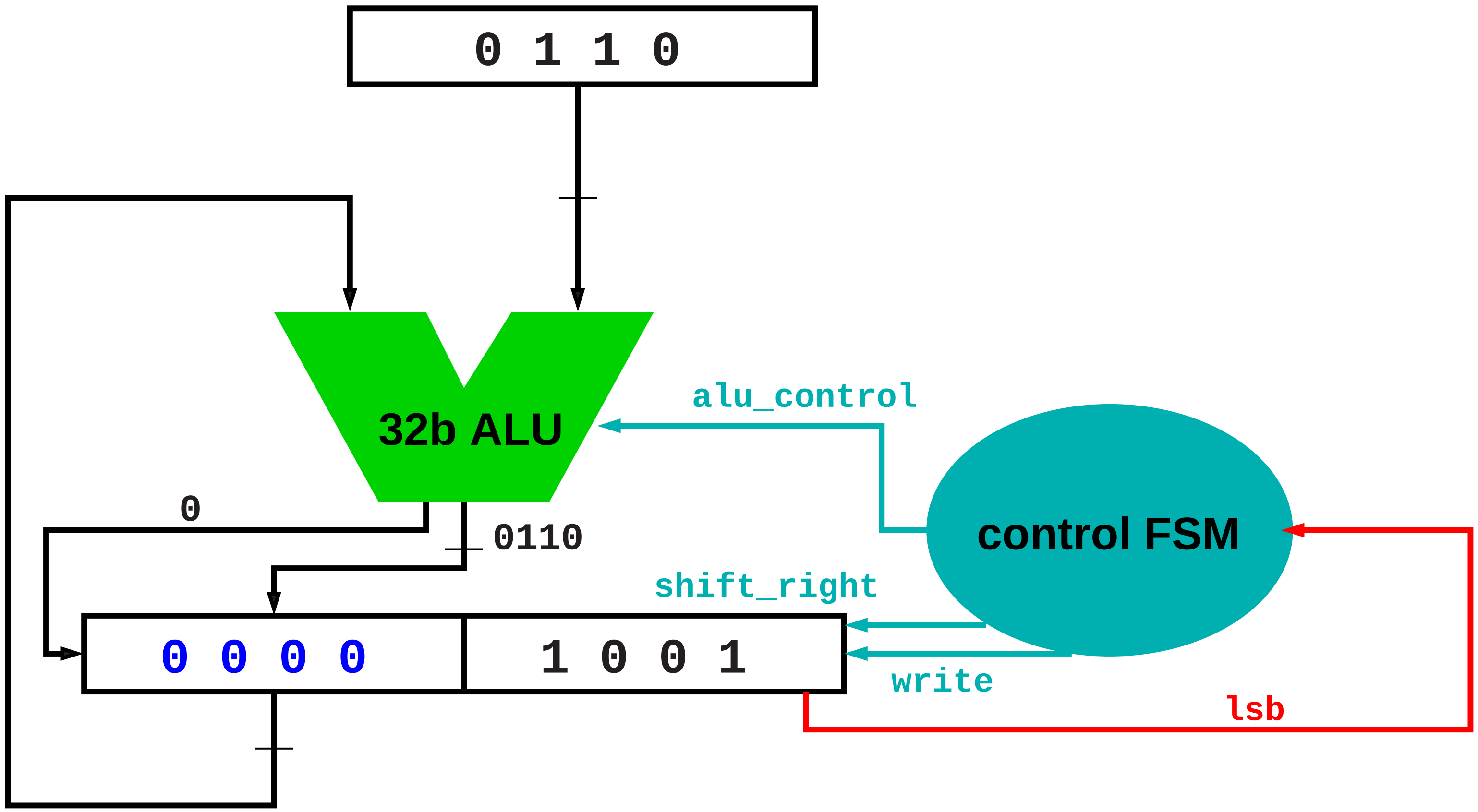
Share storage for product register and multiplier!

Integer Multiplication Hardware

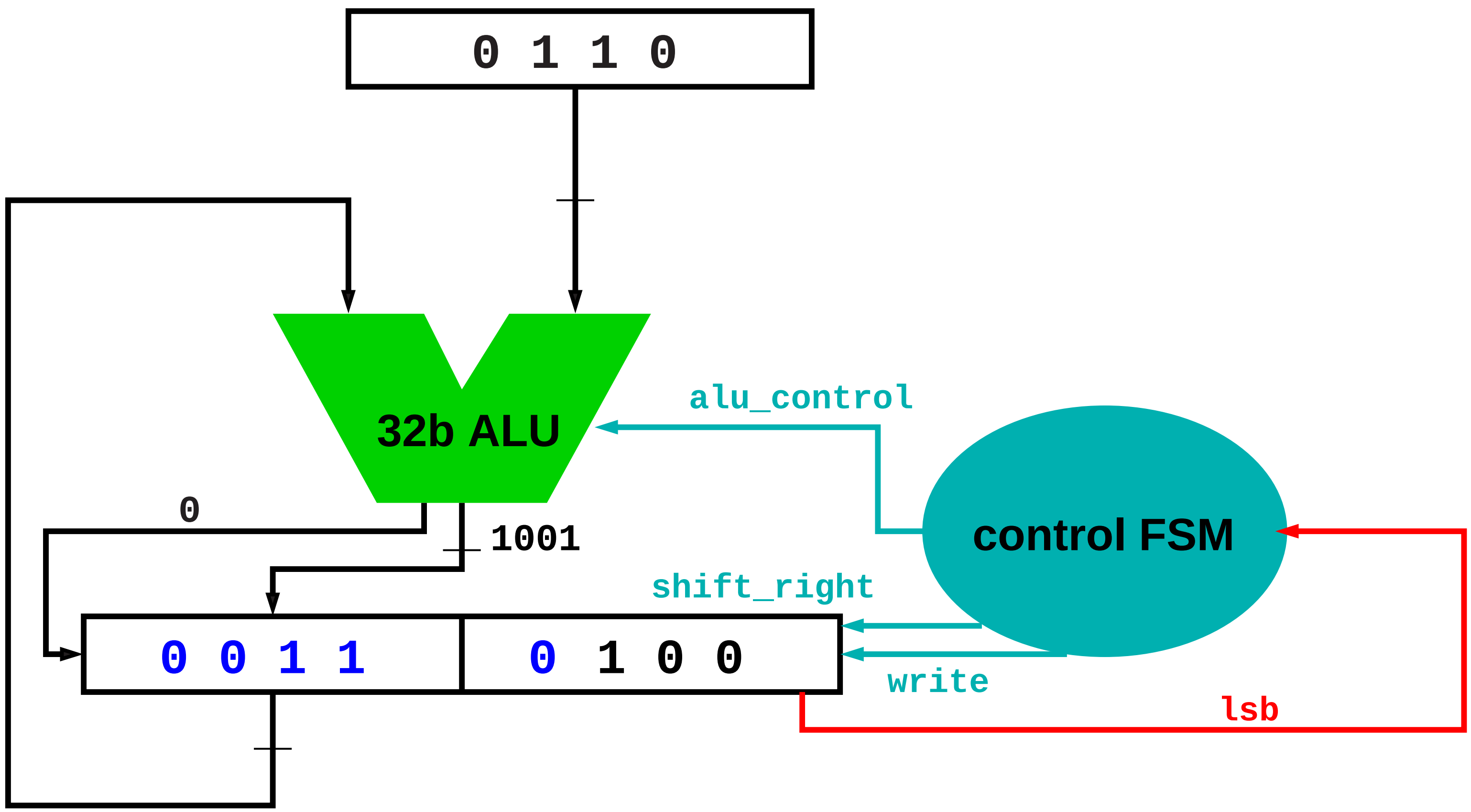


initially, multiplier bits stored here.

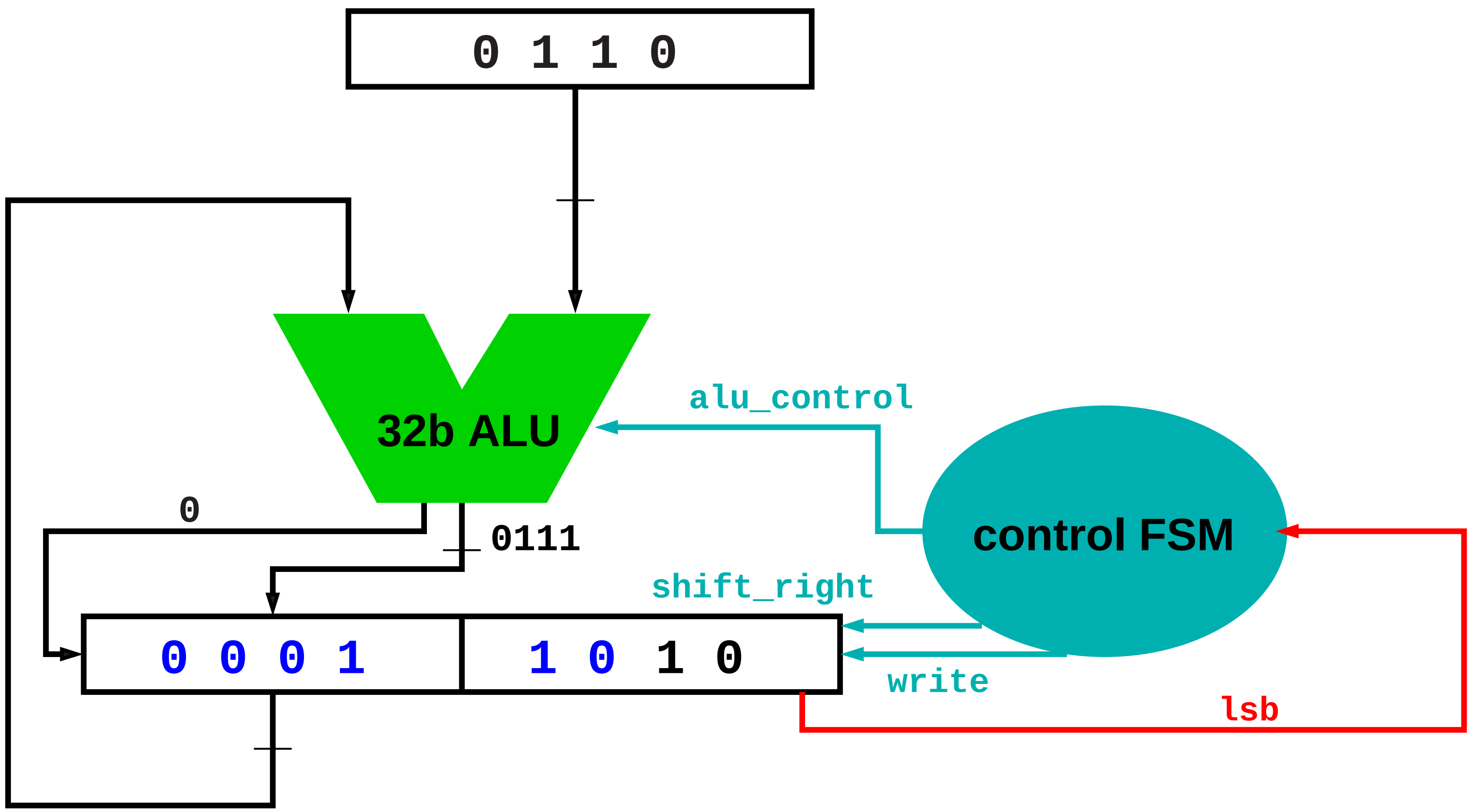
Integer Multiplication Hardware



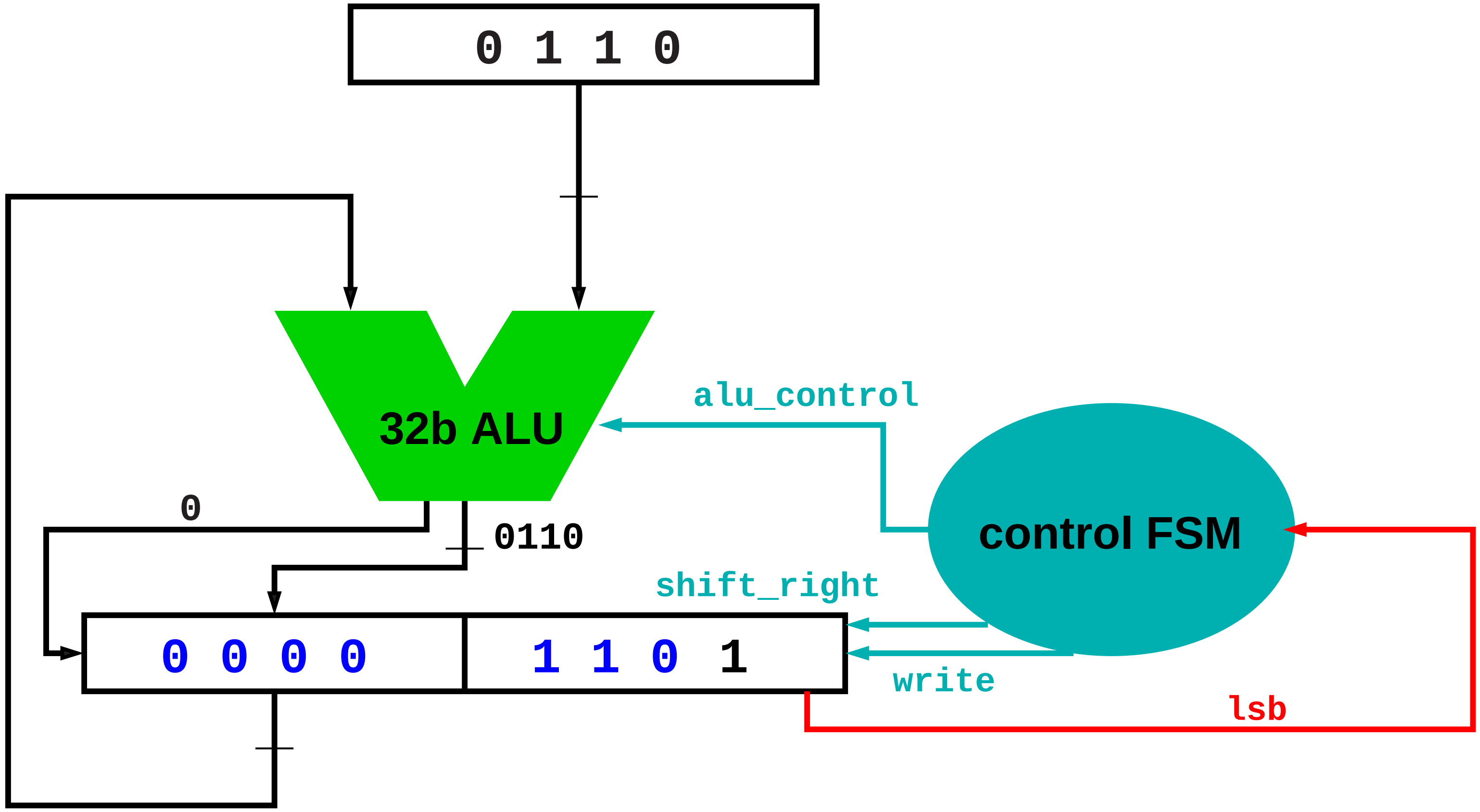
Integer Multiplication Hardware



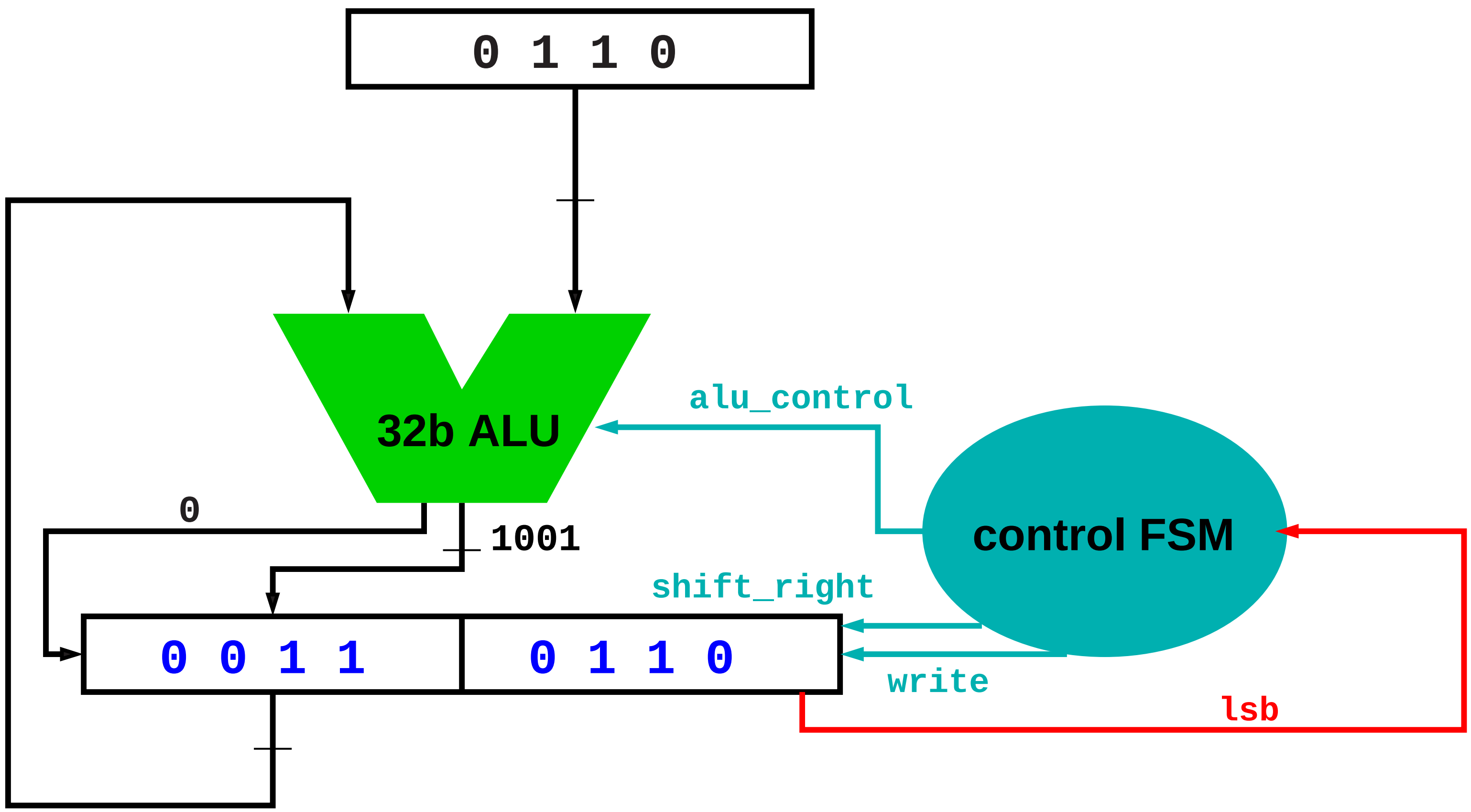
Integer Multiplication Hardware



Integer Multiplication Hardware



Integer Multiplication Hardware



Integer Multiplication

- Each step requires an add and shift
- MIPS: `hi` and `lo` registers correspond to the two parts of the product register
- Hardware implements `multu`
- Signed multiplication:
  - Determine sign of the inputs, make inputs positive
  - Use `multu` hardware, fix up sign
  - Better: Booth's algorithm

Booth Multiplication

Example:

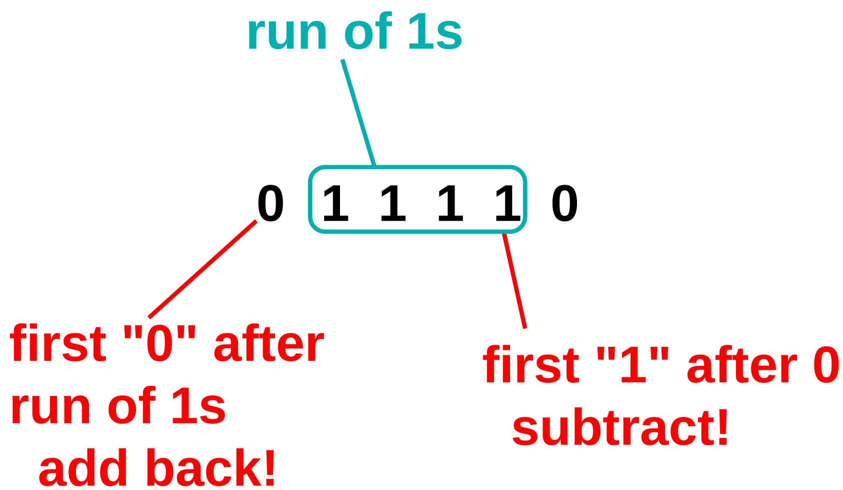
multiplicand					1	0	1	0
multiplier				×	0	1	1	0
					0	0	0	0
				1	0	1	0	
			1	0	1	0		
	+	0	0	0	0			
		0	1	1	1	1	0	0

Instead we could subtract early and add later...

$$6x = 2x + 4x = -2x + 8x$$

$$11110000 = 10000XXXX - 0001XXXX$$

Booth Multiplication



Current	Right	Explanation
1	0	beginning of run of 1s
0	1	end of run of 1s
1	1	middle of run of 1s
0	0	middle of run of 0s

Originally for speed: shifts faster than adds

Booth Multiplication

Depending on current and previous bits, do one of the following:

- 00: middle of a run of 0s $\Rightarrow$ no operation
- 01: end of a run of 1s $\Rightarrow$ add multiplicand to left half of product
- 10: start of a run of 1s $\Rightarrow$ subtract multiplicand from left half of product
- 11: middle of a run of 1s $\Rightarrow$ no operation

As before, shift product register right by 1 bit per step.

Integer Division

		0101	*quotient*
divisor	0010	1011	*dividend*
		0010	
		− 0010	
		0011	
		0010	
		− 0010	
		0001	*remainder*

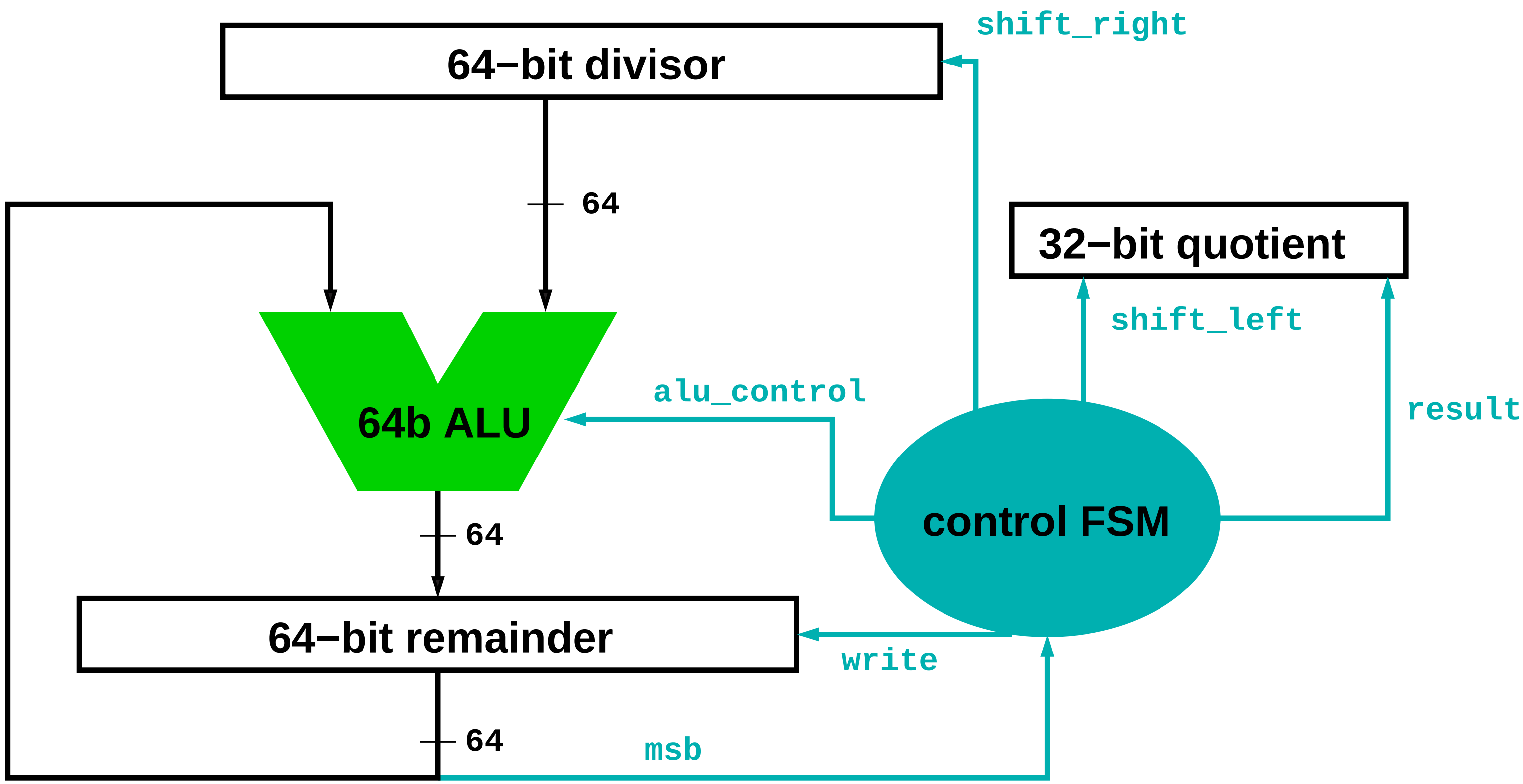
Red: steps where subtracting would result in a negative number, i.e. quotient bit is zero.

Integer Division

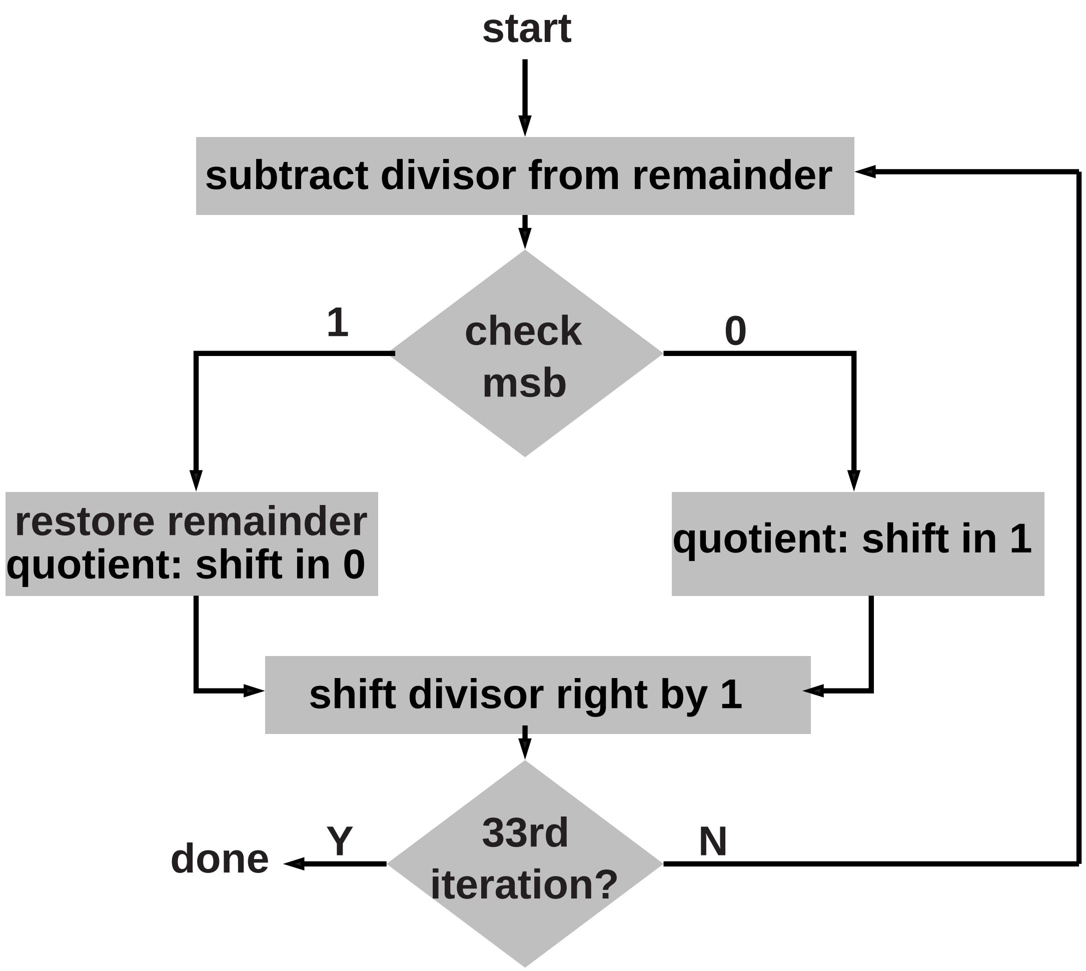
```
                  0101      quotient
divisor    0010 | 1011      dividend

              00010000
            - 00001000
            ----------
              00000011
              00000100
            - 00000010
            ----------
              00000001      remainder
```

Pad out the dividend and divisor to 8 bits.

Integer Division



Integer Division



Integer Division

Observations:

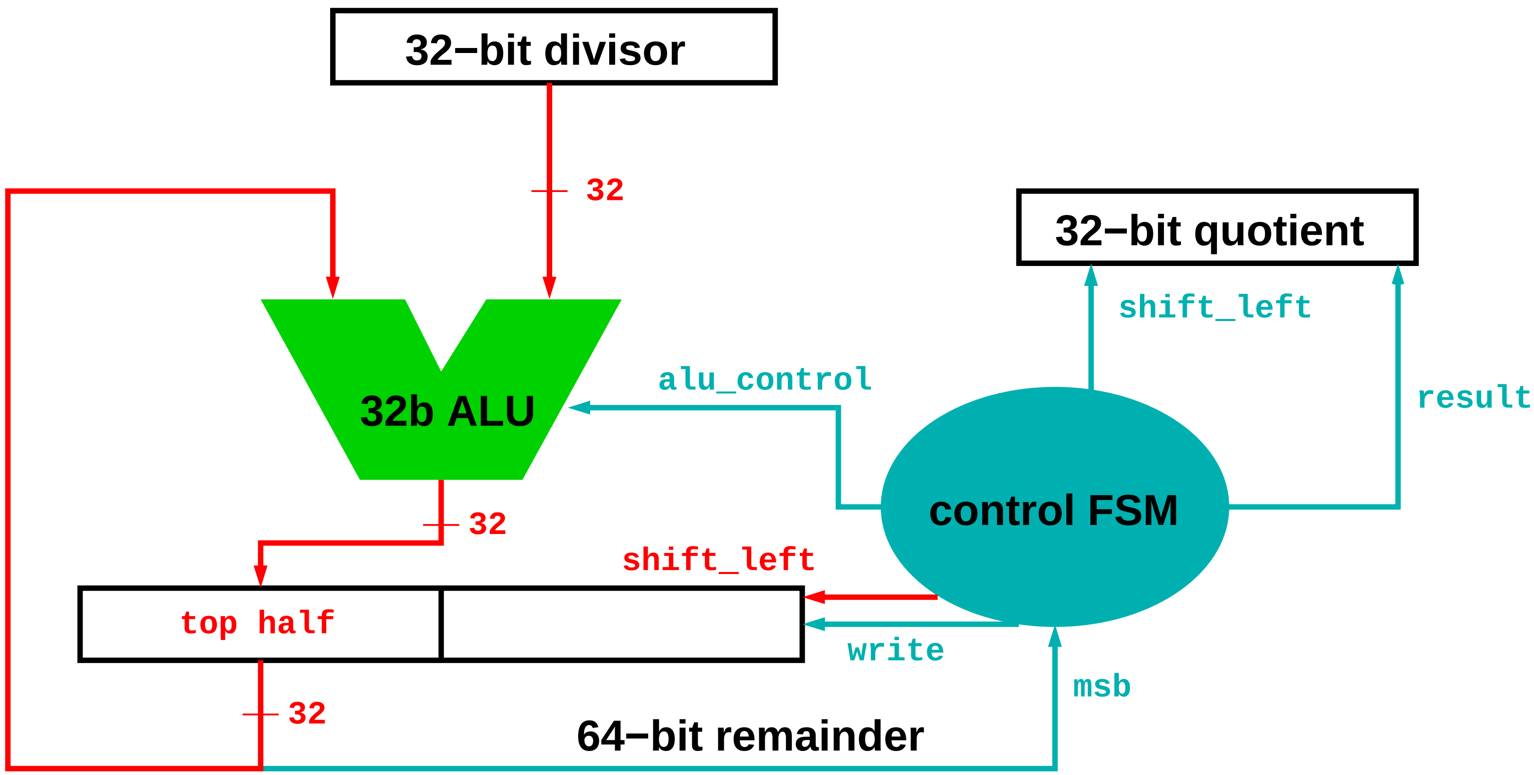
- Half the bits in the divisor are zero

  $\Rightarrow$ 64-bit ALU wasted
- Instead of shifting divisor right, we can shift remainder left
- When does the first iteration shift in a 1 into the quotient?

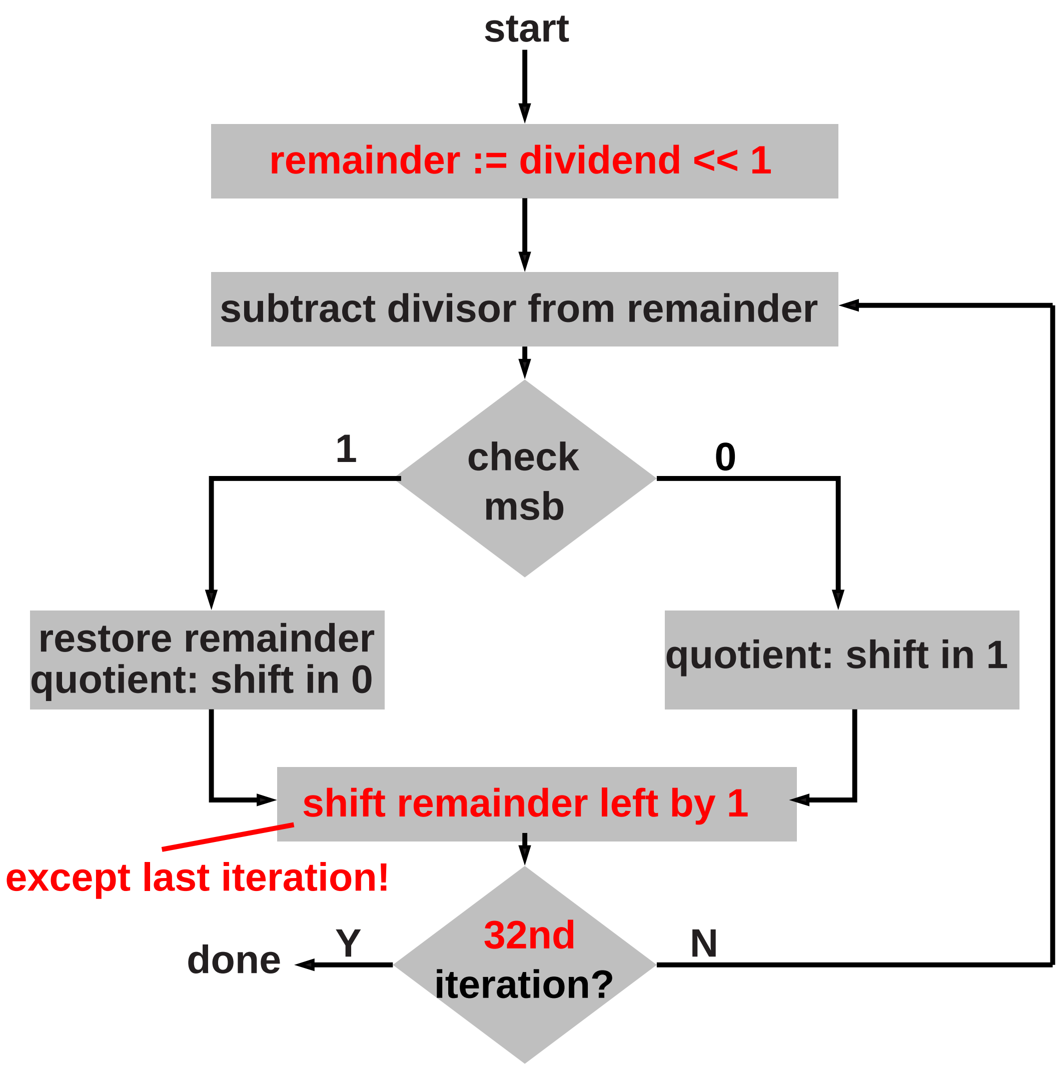
  $\Rightarrow$ save 1 iteration

What is the initial value of the divisor?

Integer Division



New Control

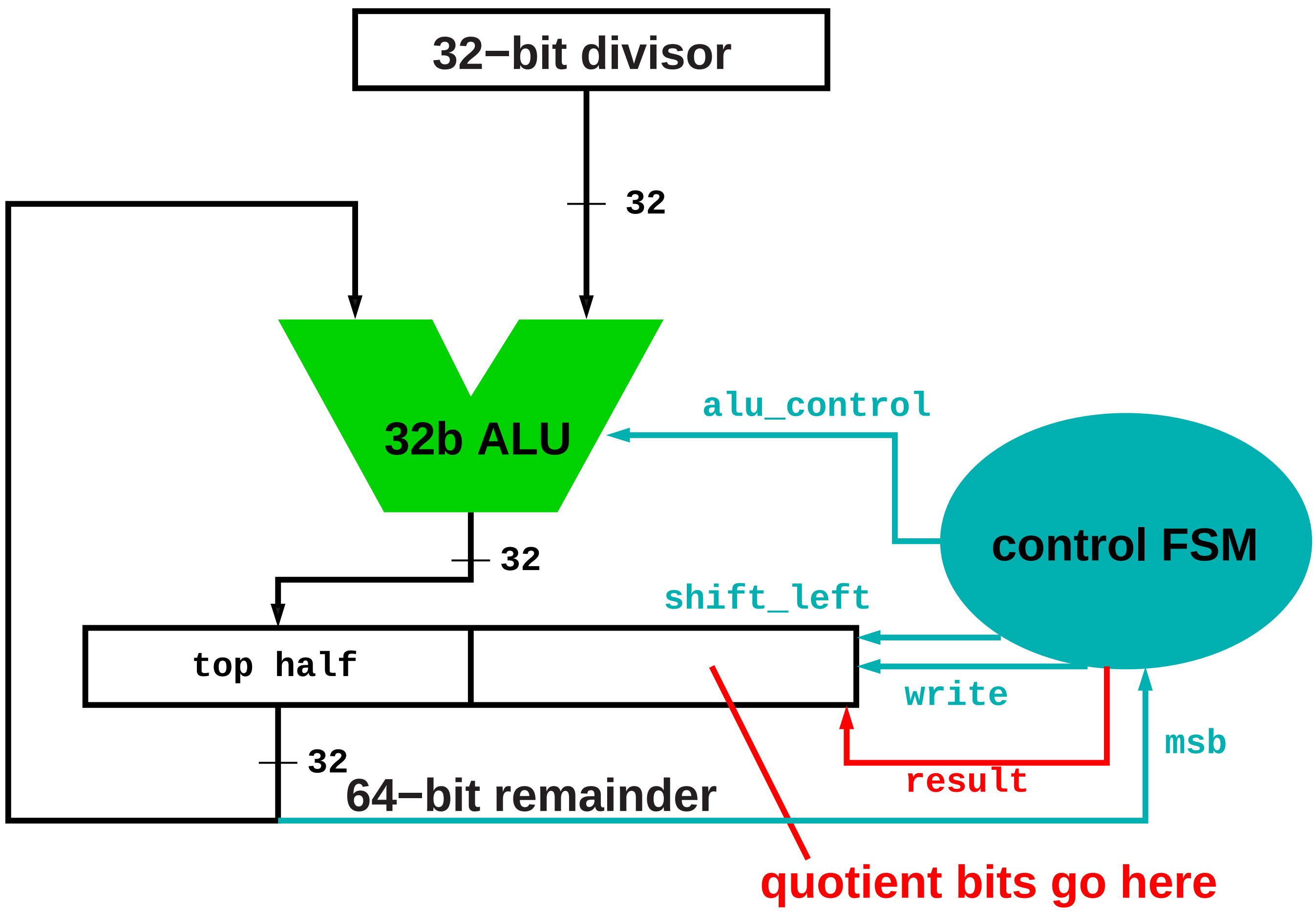


Remainder loses one bit per iteration;

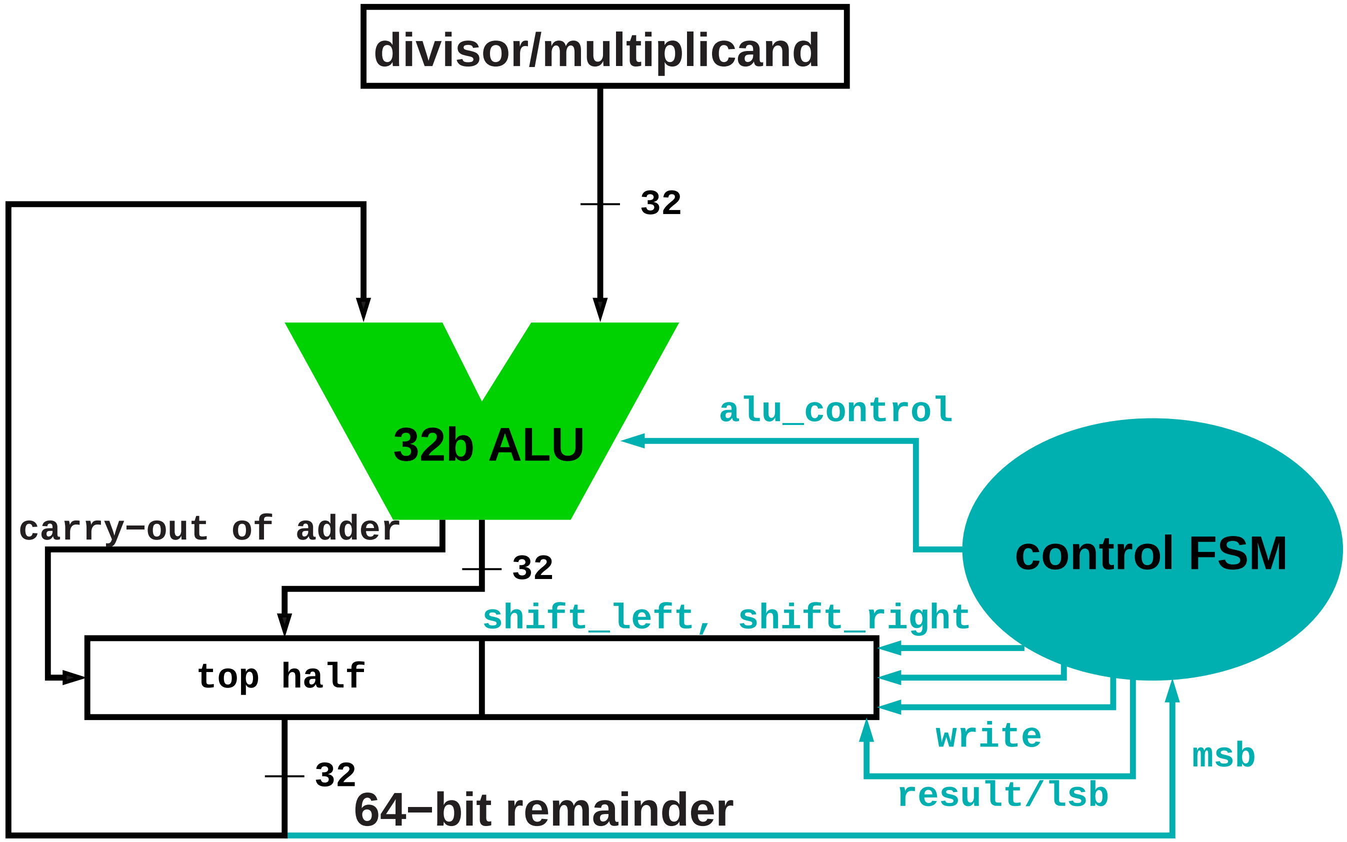
Quotient gains one bit per iteration.

=> share registers!

Final Divider Hardware



Mult/Div



It's the same hardware...

Real Numbers

How do we represent real numbers?

Several issues:

- How many digits can we represent?
- What is the range?
- How accurate are mathematical operations?
- Consistency...

Is $a + b = b + a$?

Is $(a + b) + c = a + (b + c)$?

Is $(a + b) - b = a$?

Fixed Point

Basic idea:

0 1 0 0 1 0 1 0

radix point is here

Choose a *fixed* place in the binary number where the radix point is located.

For the example above, the number is

$$(010.01010)_2 = 2 + 2^{-2} + 2^{-4} = (2.3125)_{10}$$

How would you do mathematical operations?

Floating-Point

Some problematic numbers....

$$6.023 \times 10^{23}$$
$$6.673 \times 10^{-11}$$
$$6.62607 \times 10^{-34}$$

Scientific computations require a number of digits of precision...

But they also need *range*

$\Rightarrow$ permit the radix point to move

$\Rightarrow$ *floating-point numbers*

Floating-Point: Scientific Notation

$$\overset{\text{sign}}{\pm}\ \underbrace{6.023}_{\text{mantissa}} \times \underbrace{10}_{\text{radix (base)}}{}^{\overset{\text{exponent}}{23}}$$

- Number represented as:
  - mantissa, exponent
- Arithmetic
  - multiplication, division: perform operation on mantissa, add/subtract exponent
  - addition, subtraction: convert operands to have the same exponent value, add/subtract mantissas