The Arithmetic-Logic Unit (ALU)

Combinational circuit that performs operations in CPU

Given inputs *a* and *b*, and an operation code, produce an output

Operation codes:

000: AND
001: OR
010: NOR
100: ADD
101: SUB
• • •

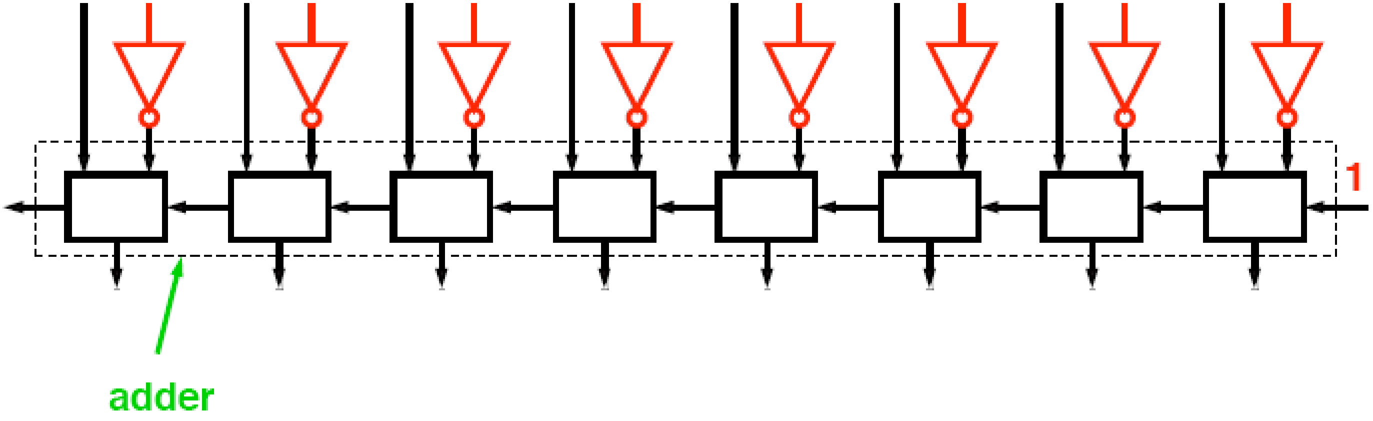
Unlike the adder, this is a general-purpose unit

Subtraction

To calculate $a - b$, use $a + (-b)$.

To calculate $-b$, flip all the bits and add 1.

$\Rightarrow$ build it using an adder



Of course, *any* adder will do ...

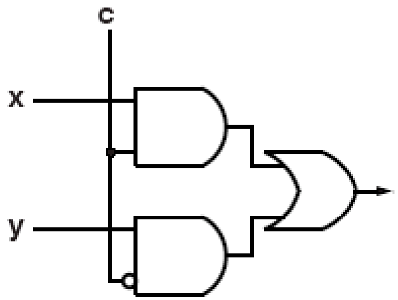
– use block carry-lookahead adder from last time!

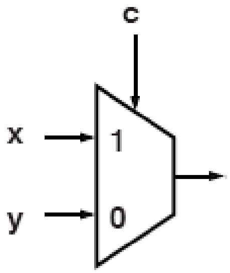
Combined Add/Subtract Unit

Given: one bit of control c, two N bit inputs a and b. compute $a + b$ if $c = 0$, $a - b$ if $c = 1$.

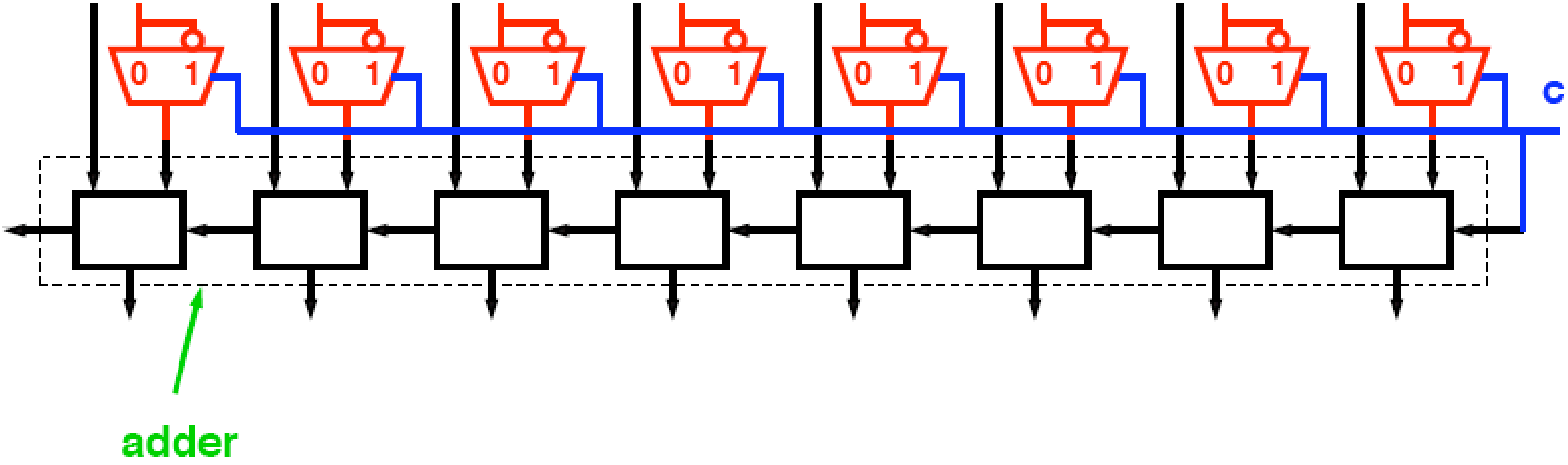
- Carry-in to the adder is c
- one input: a
- other input: b if $c = 0$, complement of b if $c = 1$.

Standard element: MUX (multiplexor)





Combined Add/Subtract Unit

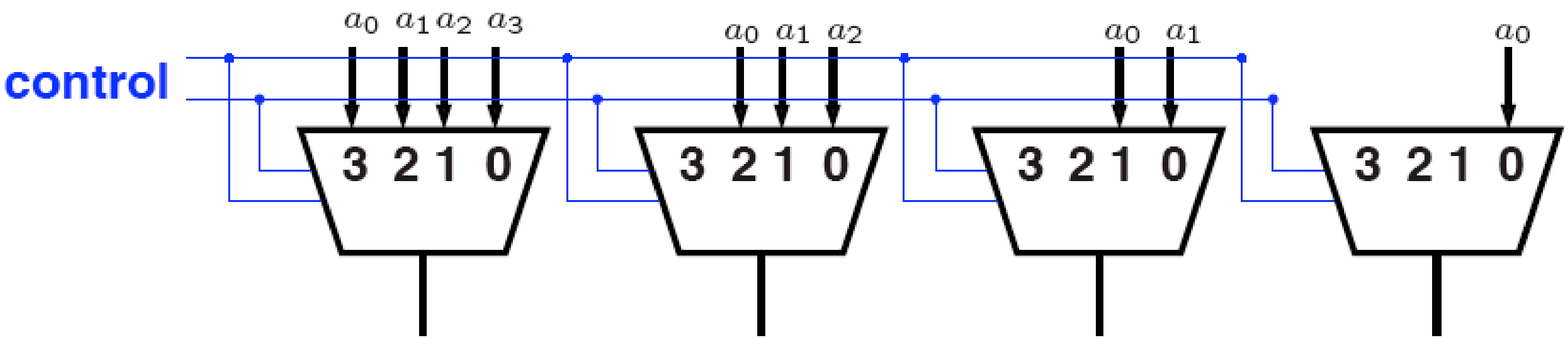


- Hierarchical design
- Reuse components
- Replication

Shifter

4-input MUX?

Simple shifter:



Arithmetic Logic Unit (ALU)

Example ALU: given inputs a and b, and an operation code, produce output.

Operation code:

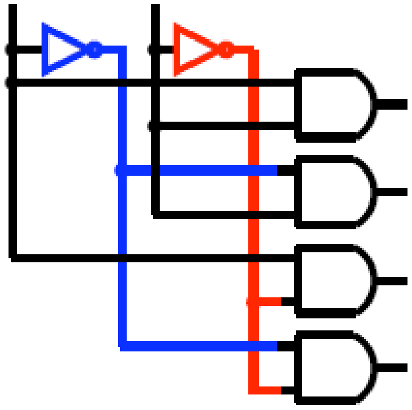
- 000: AND
- 001: OR
- 010: NOR
- 011: ADD
- 111: SUB

How do we implement this ALU?

Selecting An Operation

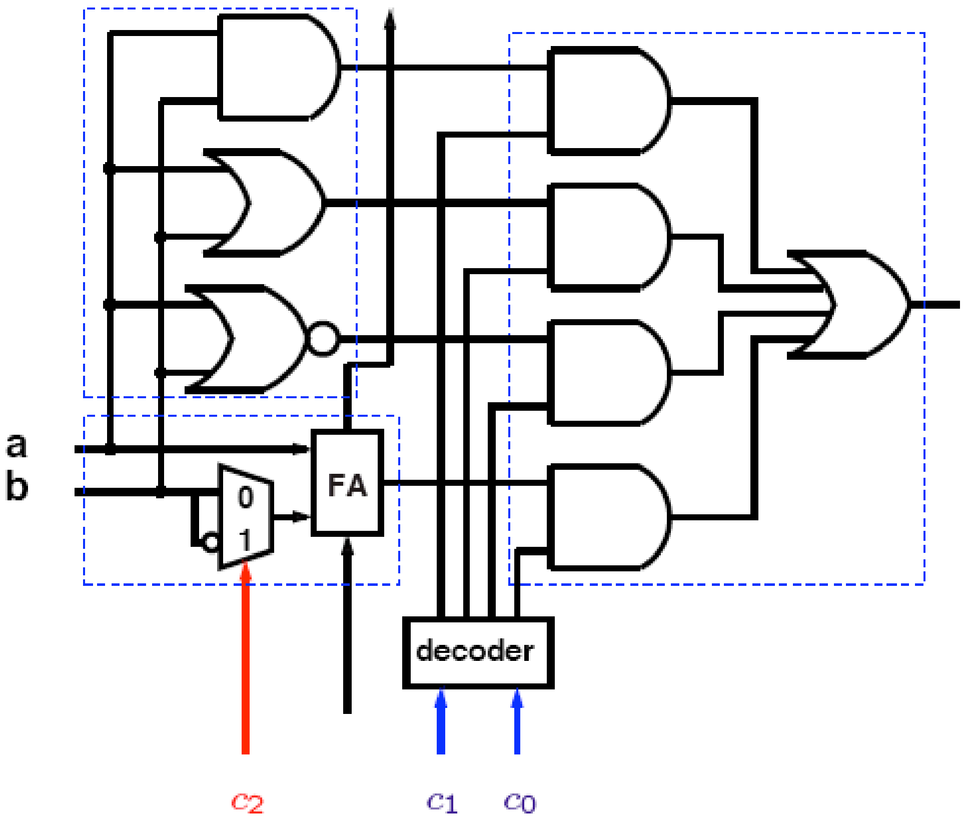
2-bit decoder: 2 bit input, 4 bit output

- **input:** 00, **output:** 0001
- **input:** 01, **output:** 0010
- **input:** 10, **output:** 0100
- **input:** 11, **output:** 1000



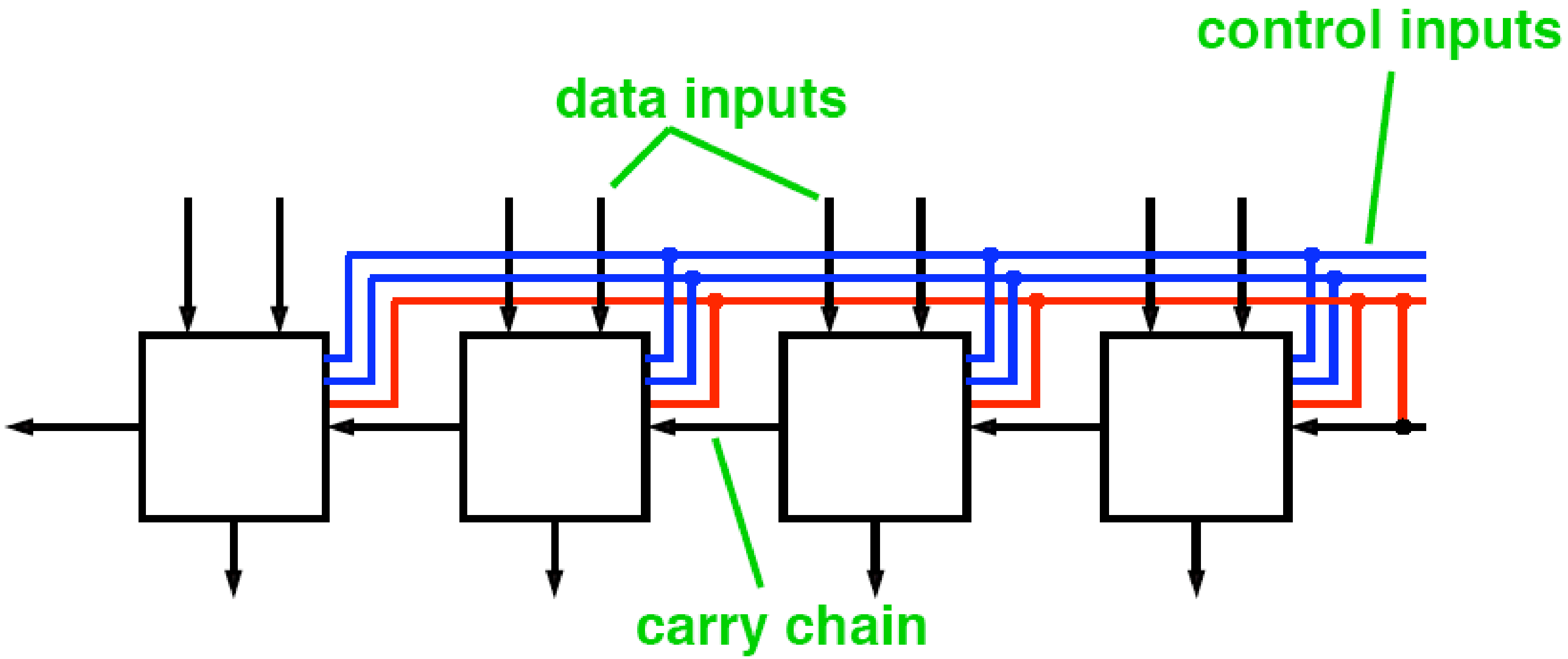
ALU: One Bit

Use decoder to select operation, and use combined add/subtract unit.



ALU: Multiple Bits

Chain ALU *bit slices* to get an N bit ALU:



How can we use a better adder in the ALU?

Overflow Detection

Overflow = result of operation cannot be represented

Unsigned N-bit addition:

- Overflow = result requires more than N bits
  $\Rightarrow$ carry-out of MSB is 1

Signed addition:

- Adding two positive numbers
- Adding two negative numbers

Overflow $\equiv$ carry-in to MSB $\neq$ carry-out of MSB

Comparison

When is $a < b$?

- $a < b \equiv a - b < 0$
- Subtract b from a, check sign of result
- Sign bit is MSB

When is $a = b$?

- $a = b \equiv a - b = 0$
- Subtract b from a, check if all bits are zero
- Use NOR gate

Integer Multiplication

Multiplying two numbers:

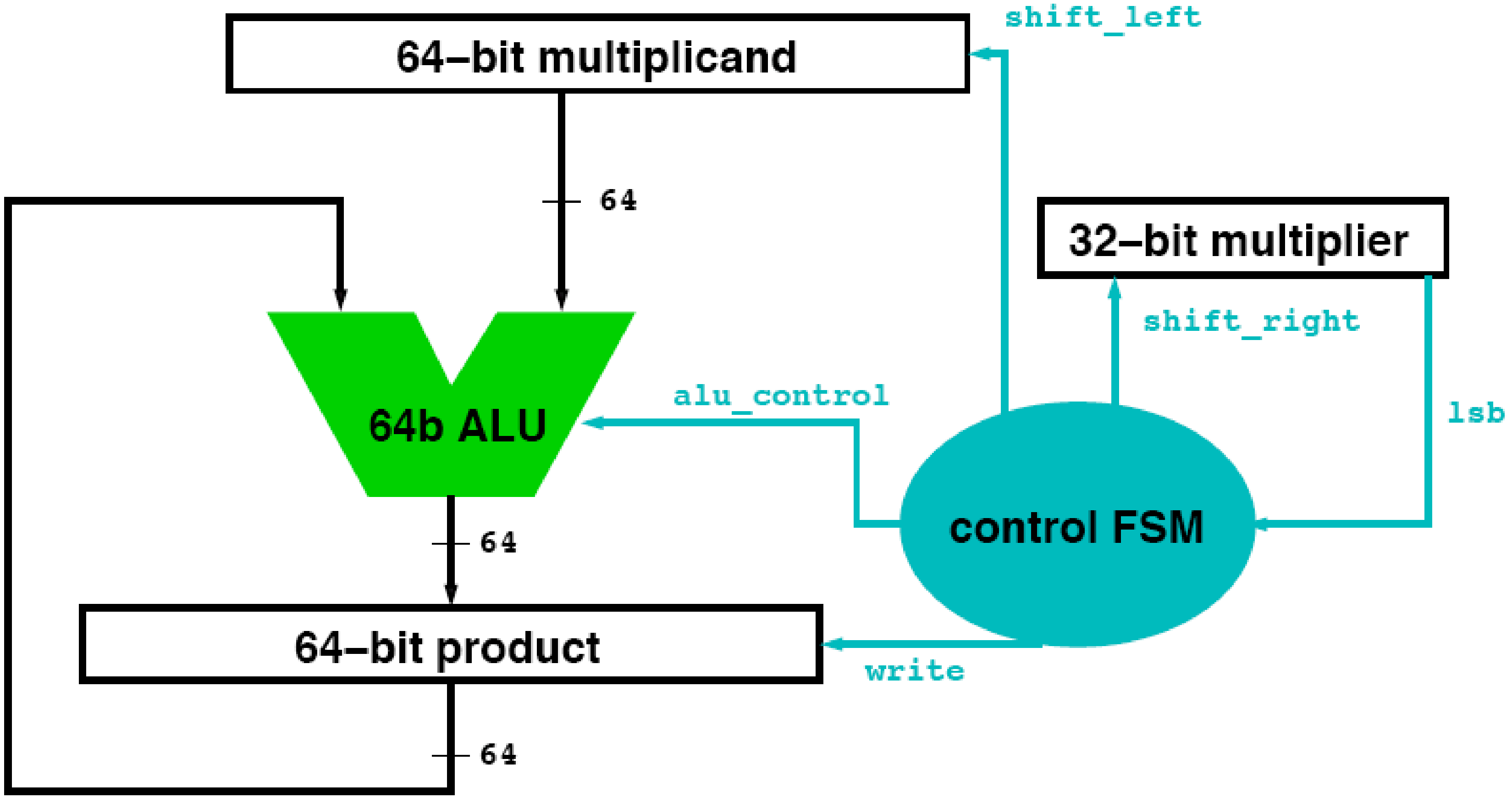
$$\begin{array}{lccccccccc}
\text{multiplicand} & & & & & 1 & 0 & 1 & 0 \\
\text{multiplier} & & & & \times & 1 & 0 & 0 & 1 \\ \cline{5-9}
 & & & & & 1 & 0 & 1 & 0 \\
 & & & & 0 & 0 & 0 & 0 & \\
 & & & 0 & 0 & 0 & 0 & & \\
 & + & 1 & 0 & 1 & 0 & & & \\ \cline{2-9}
 & & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ \cline{2-9}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

m-bits × n-bits = $(m+n)$-bit result

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

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

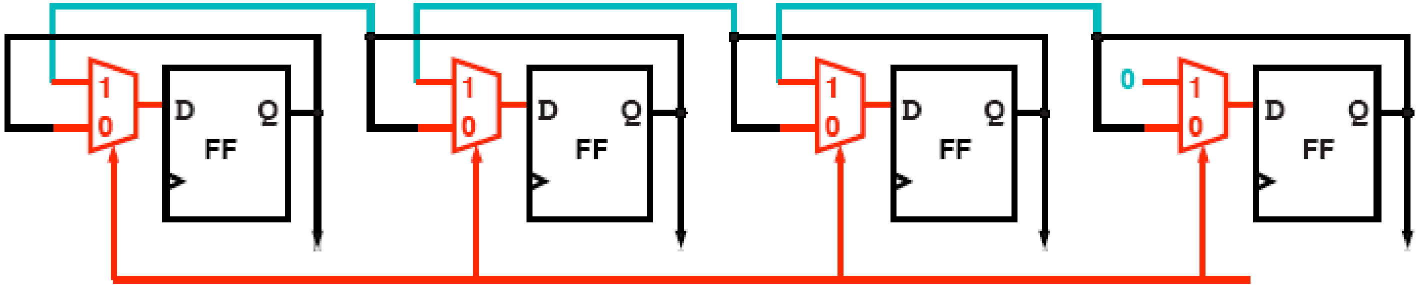
Integer Multiplication: First Try



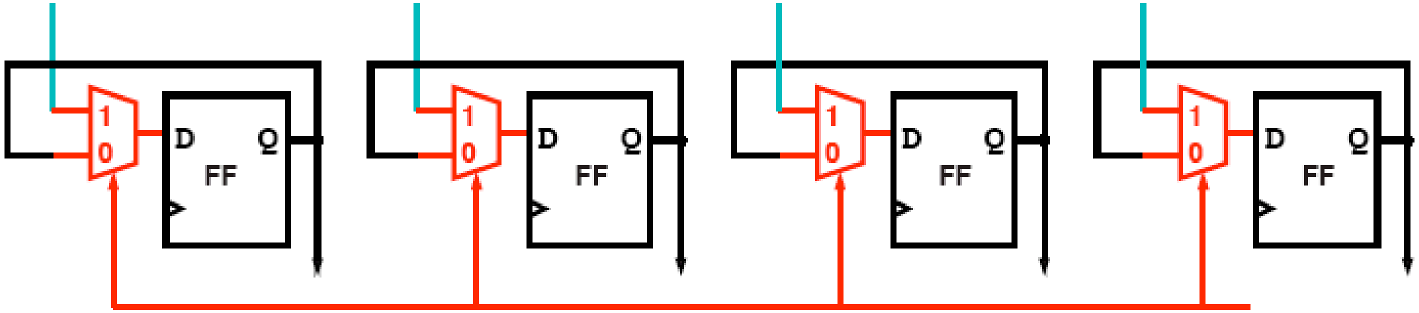
How do we build this?

Registers And Shift Registers

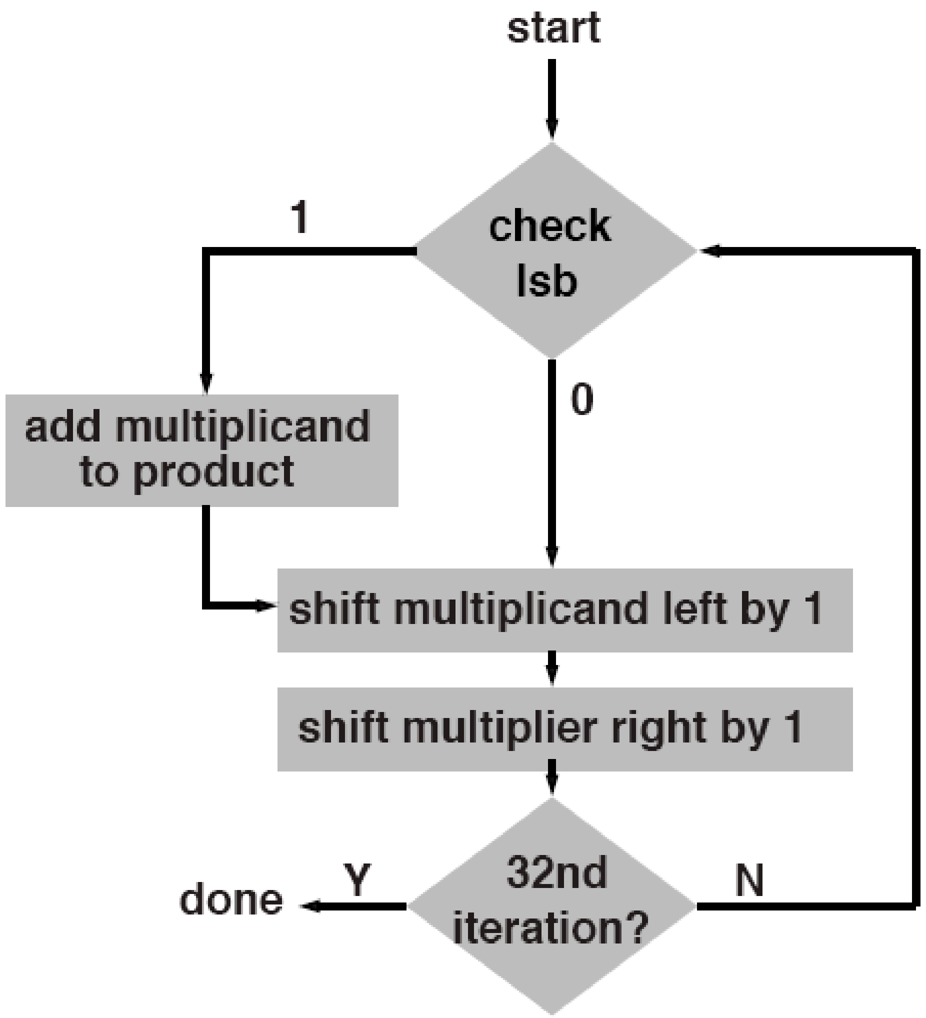
Register with shift left:



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
done
N

Integer Multiplication

Observations:

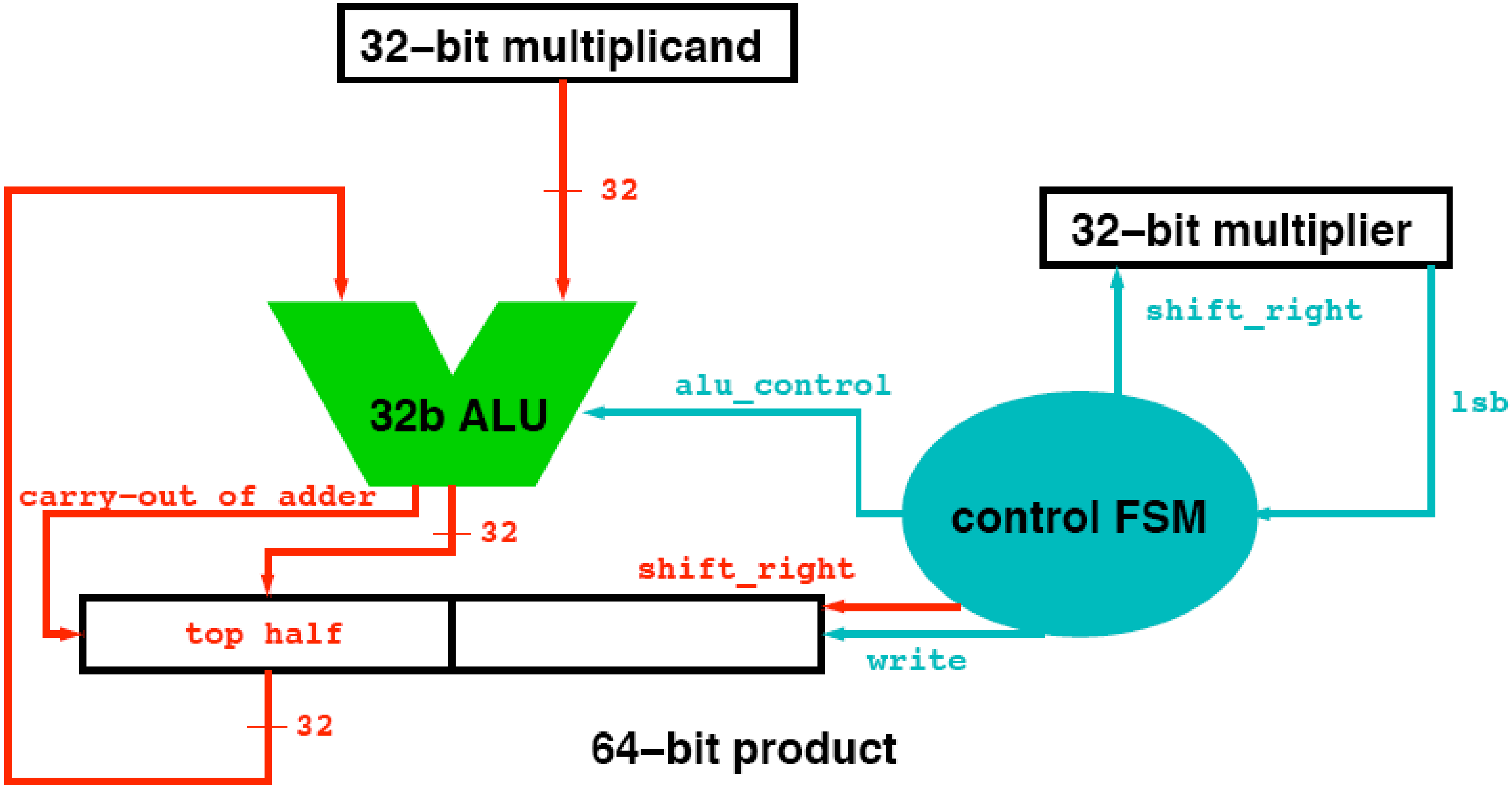
- 32 iterations for multiplication ⇒ 32 cycles
- How long does 1 iteration take?
- Suppose 5% of ALU operations are multiply ops , and other ALU operations take 1 cycle.

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

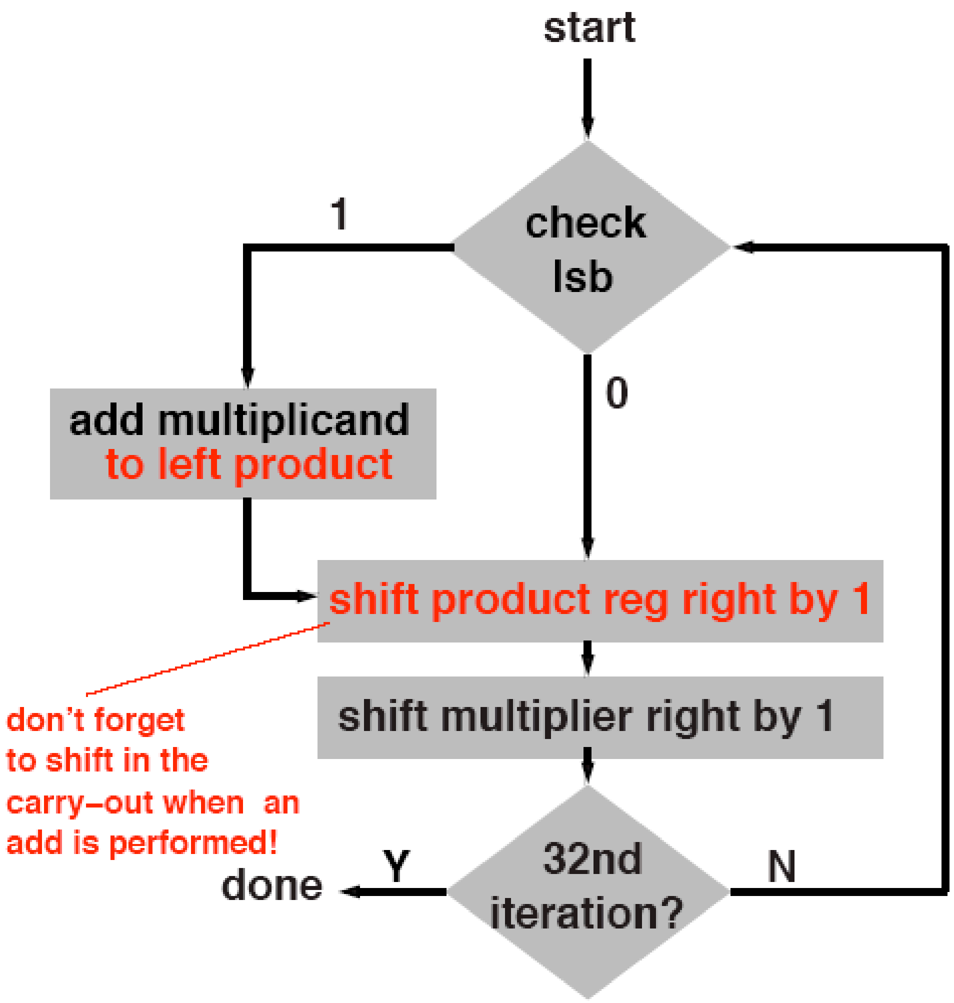
  ⇒ 64-bit adder is wasted
- 0's inserted when multiplicand shifted left

  ⇒ product LSBs don't change

Using A 32-Bit ALU



New Control


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

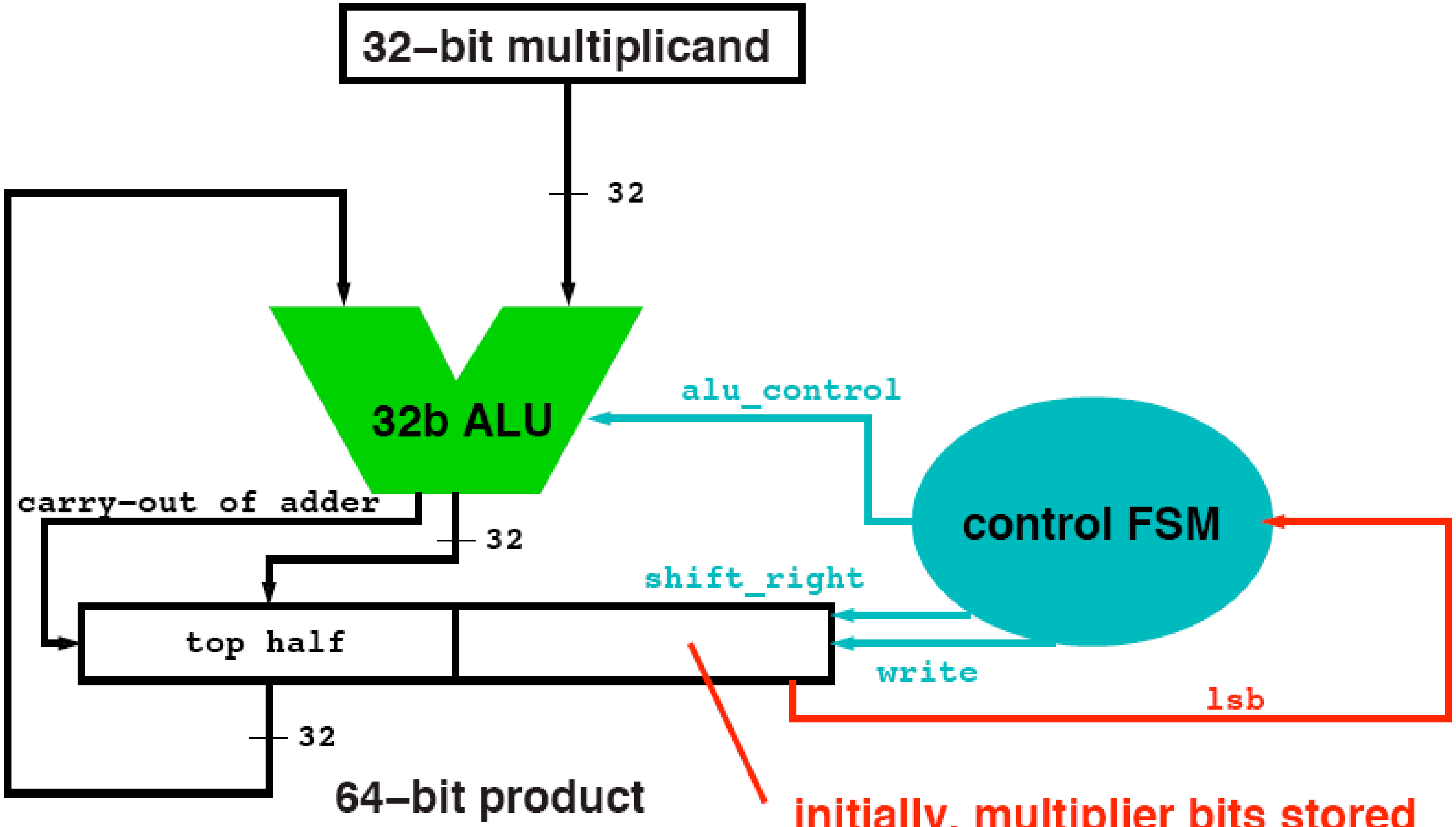
don't forget to shift in the carry–out when an add is performed!

Bottom half of product register is zero initially.

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

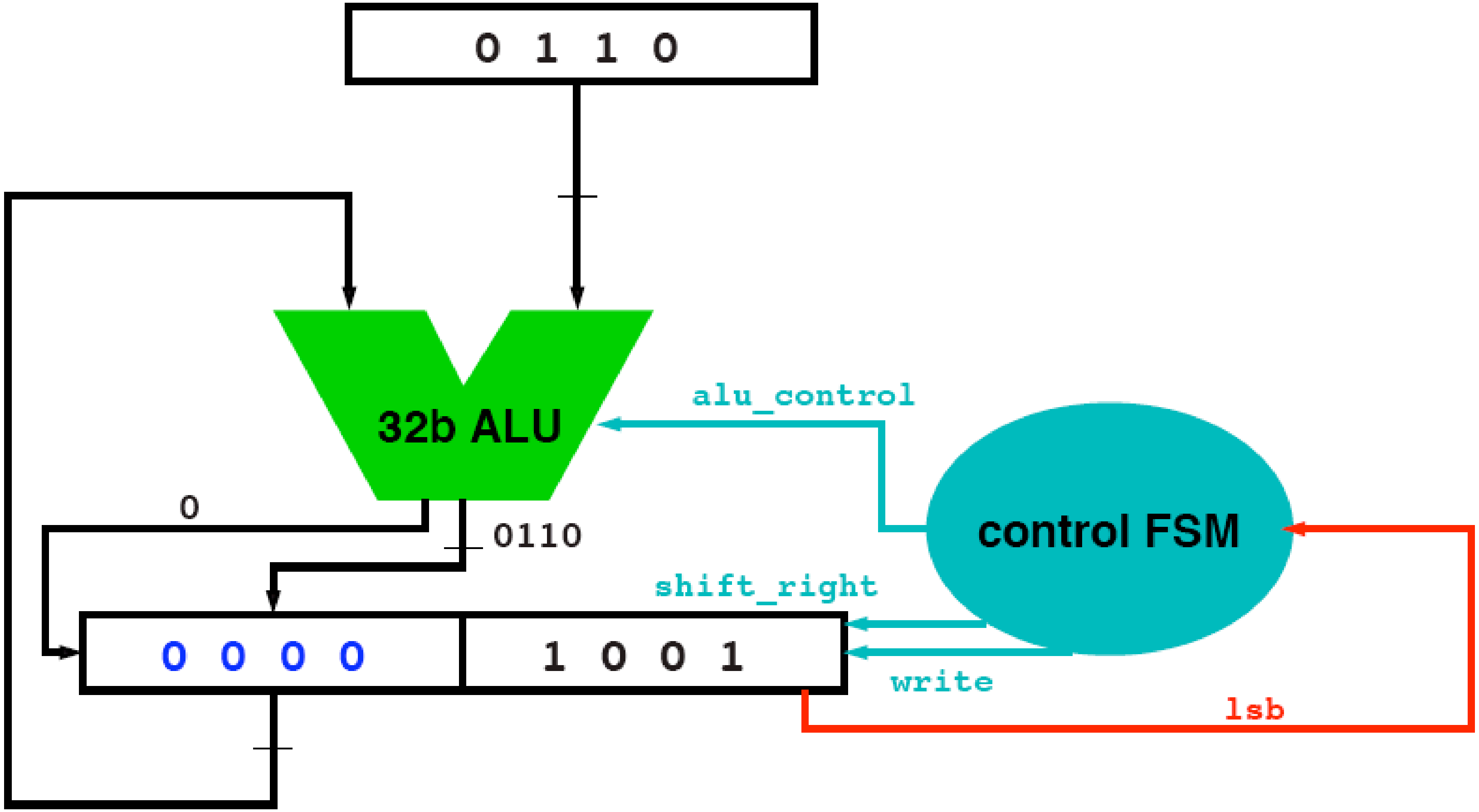
Share storage for product register and multiplier!

Integer Multiplication Hardware

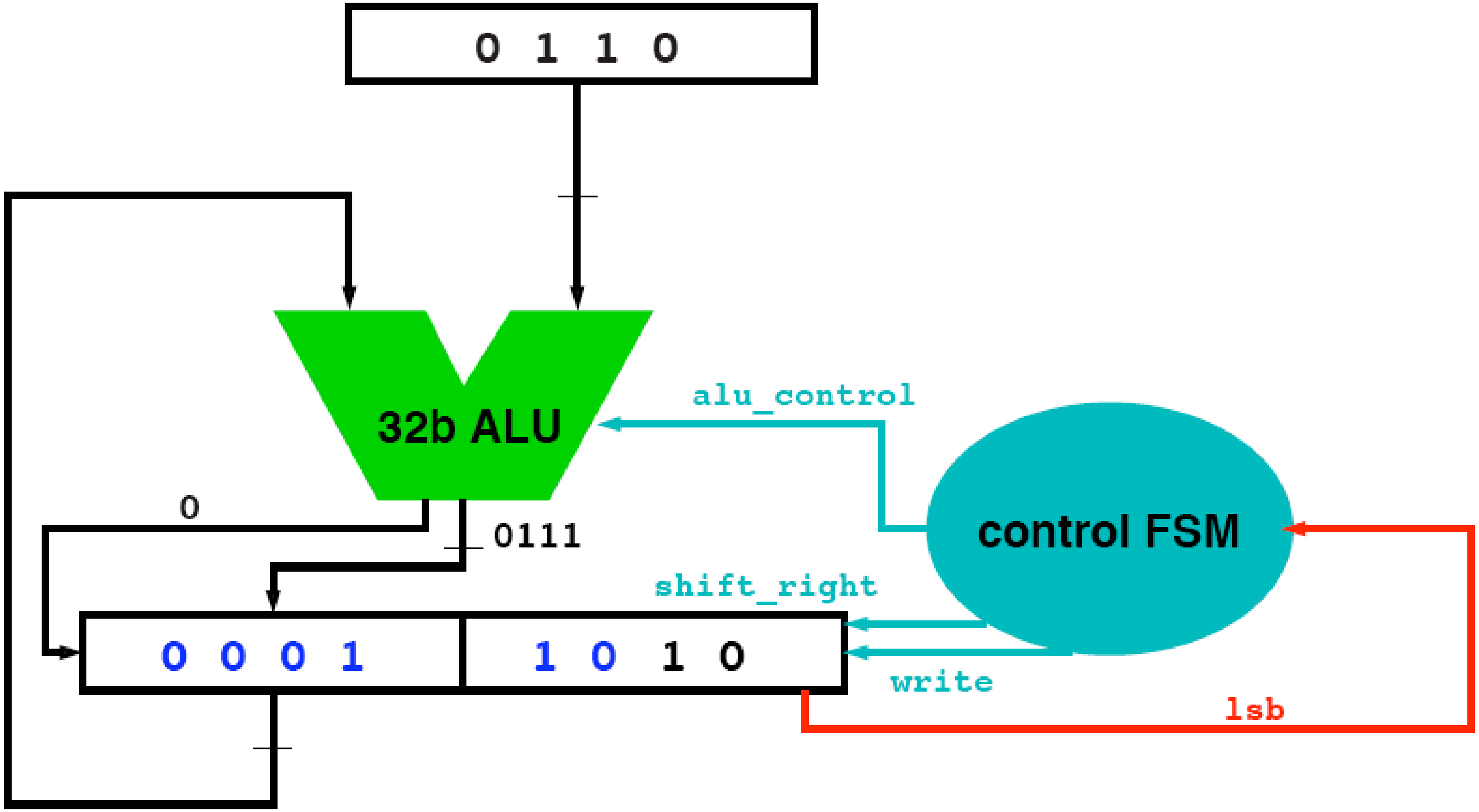


initially, multiplier bits stored here.

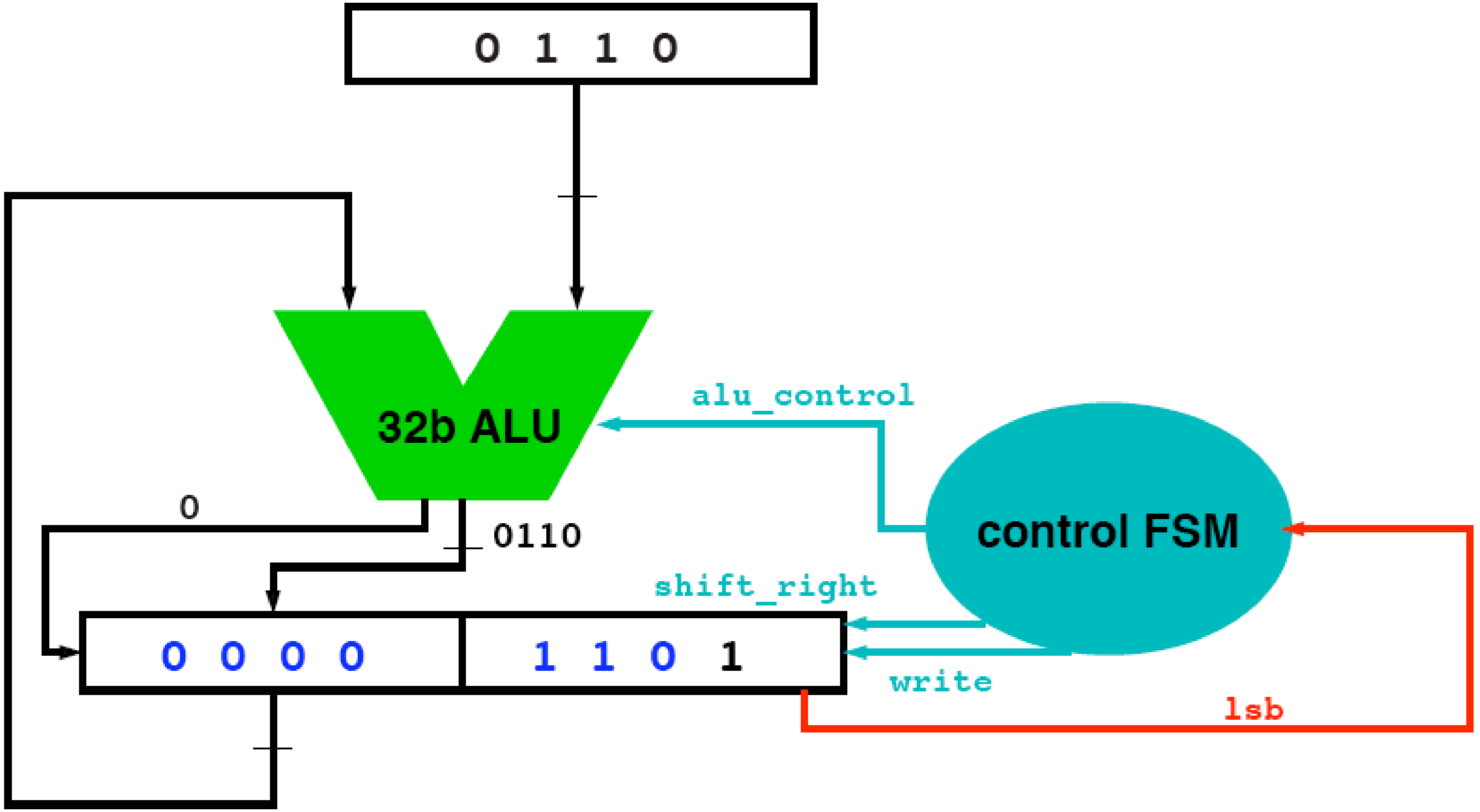
Integer Multiplication Hardware



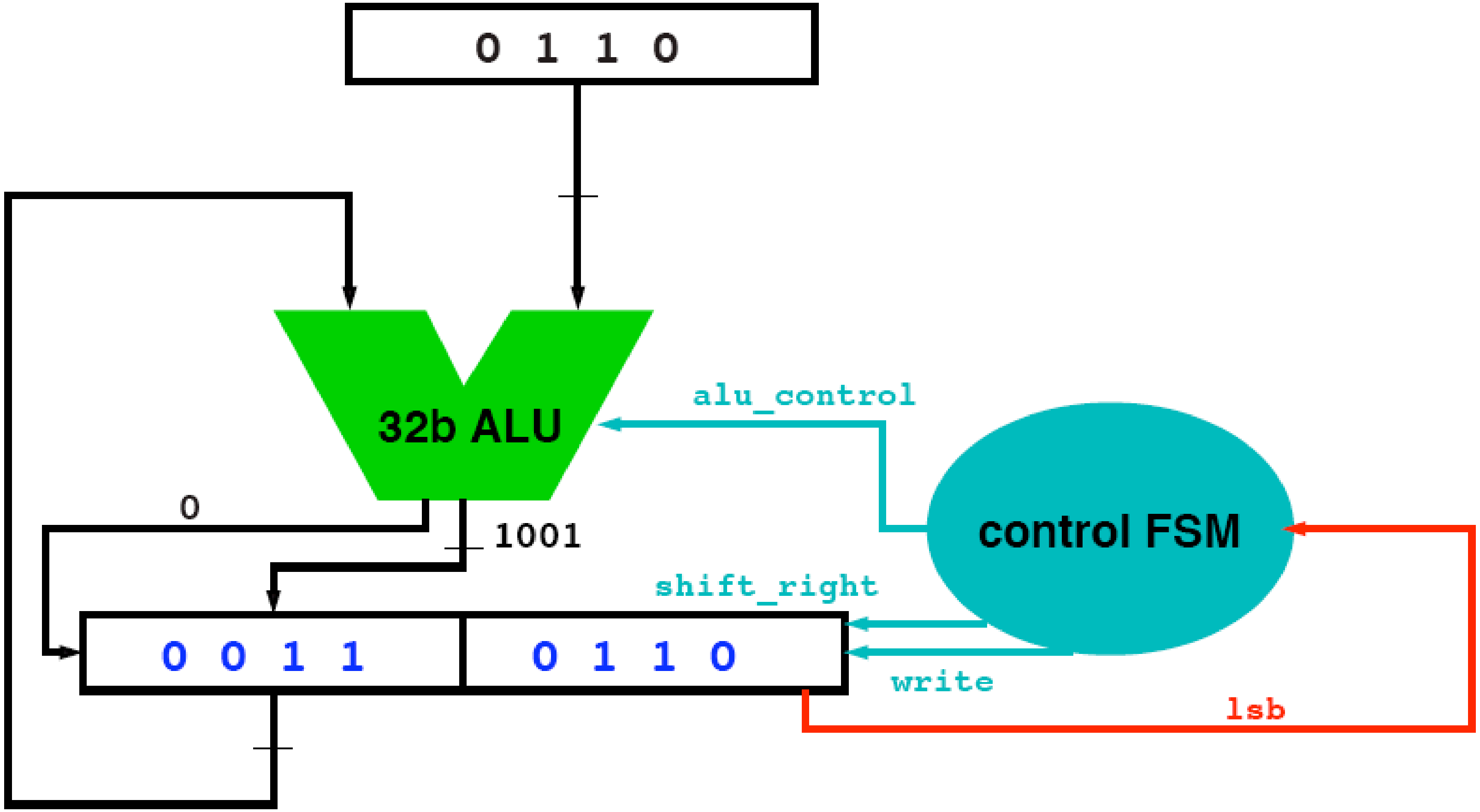
Integer Multiplication Hardware



Integer Multiplication Hardware



Integer Multiplication Hardware



Integer Multiplication

- 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