Chapter-3

"Arithmetic for Computers"

Addition: $(7)_{10} + (6)_{10}$

bits with this color represent carry forward.

```
   1    1   0
   0    1   1    1
+  0    1   1    0
-------------------
1 (1) 1 (1) 0 (0) 1
```

adding bits right to left.

Subtraction: $(7)_{10} - (6)_{10} = (7)_{10} + (-6)_{10}$

$6 = 110$

$+6 = 0110$

$= 1001$

$+1$

$-6 = (1010)_{2s}$

```
          1   1   1   0
+7 =        0   1   1   1
-6 =        1   0   1   0
--------------------------
1 (1) 0 (1) 0 (1) 0 (0) 1
```

Overflow (signed)

Addition:

=> Add two same signed number

if (answer also has same sign):

No overflow

else

Overflow

=> Sub two same signed number

Never Over Flow.

=> Add two different signed number

Never Over Flow.

=> Sub two different number

$+A - (-B) = +A + B$ => rule 1

$-A - (+B) = -A + (-B)$ => rule 1

In case of unsigned integers, overflows are ignored.

are used to indicate memory addresses

Compilers detect unsigned overflows using simple branch instruction.

$6 = 1\ 1\ 0$

$7 = 1\ 1\ 1$

$1\ 1\ 0\ 1$

logic: If the sum is less than either of the addends.

Overflow

if the difference is greater than the minuend.

13 ← Subtrahend

-3

10

ALU = Arithmetic Logic Unit

Long Multiplication

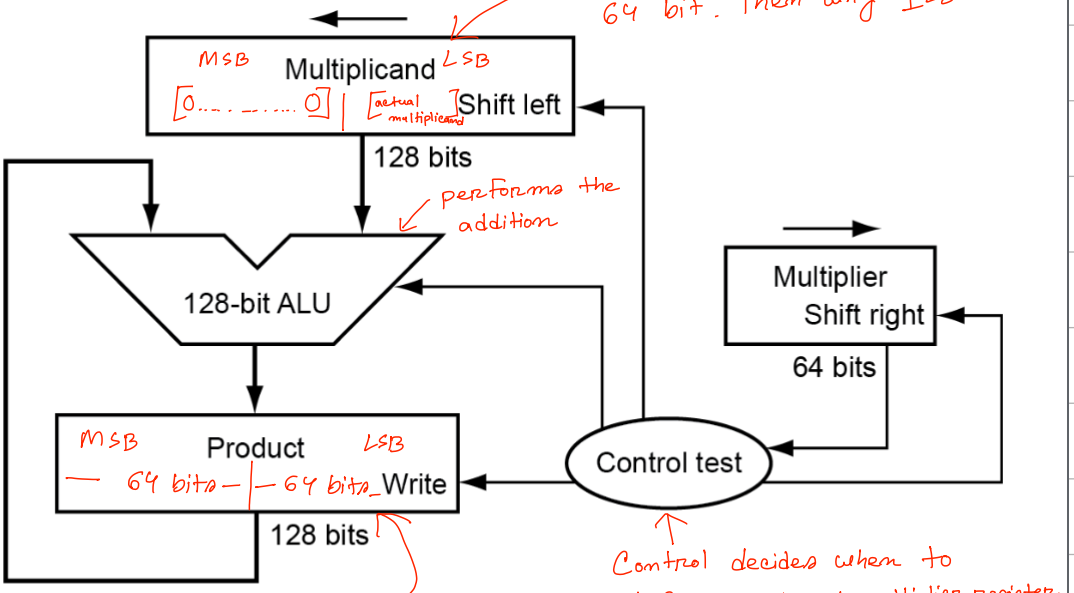
```
  1000   → Multiplicand
x 1001   → Multiplier
------
  1000
 0000
00 00
1000
------
1001000  → Product
```

Maximum,

length of the product

= (length of multiplicand + length of multiplier)

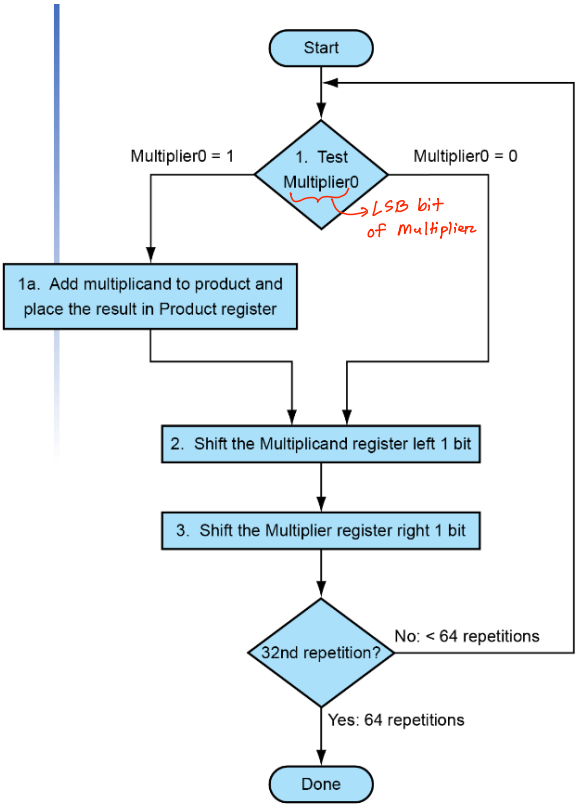
this was also supposed to be 64 bit. Then why 128 bits?

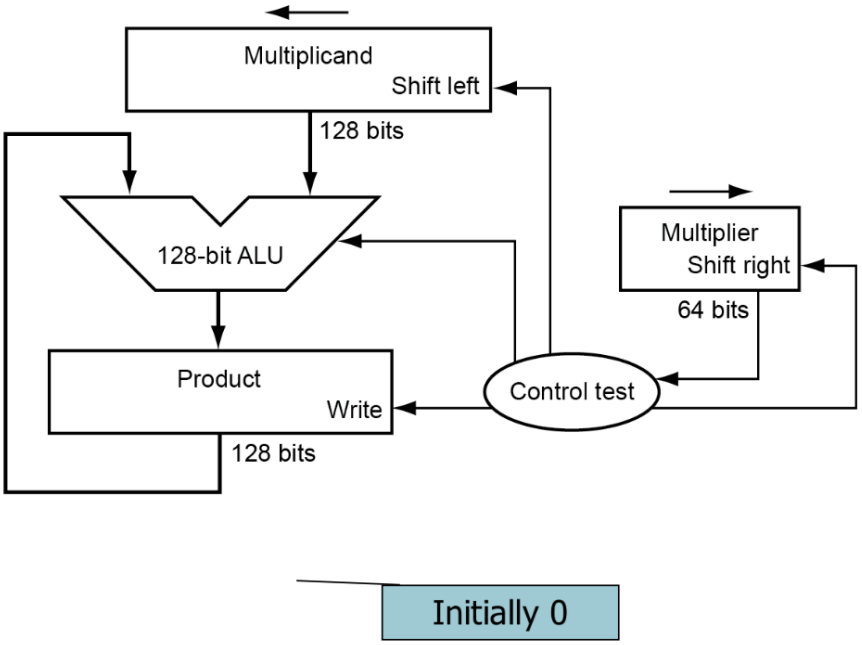


Multiply two 64 bit values, product length can be (64+64)=128 bit.

But we do not have any 128 bit registers in RISC-V

Hence, we use two registers to store the values





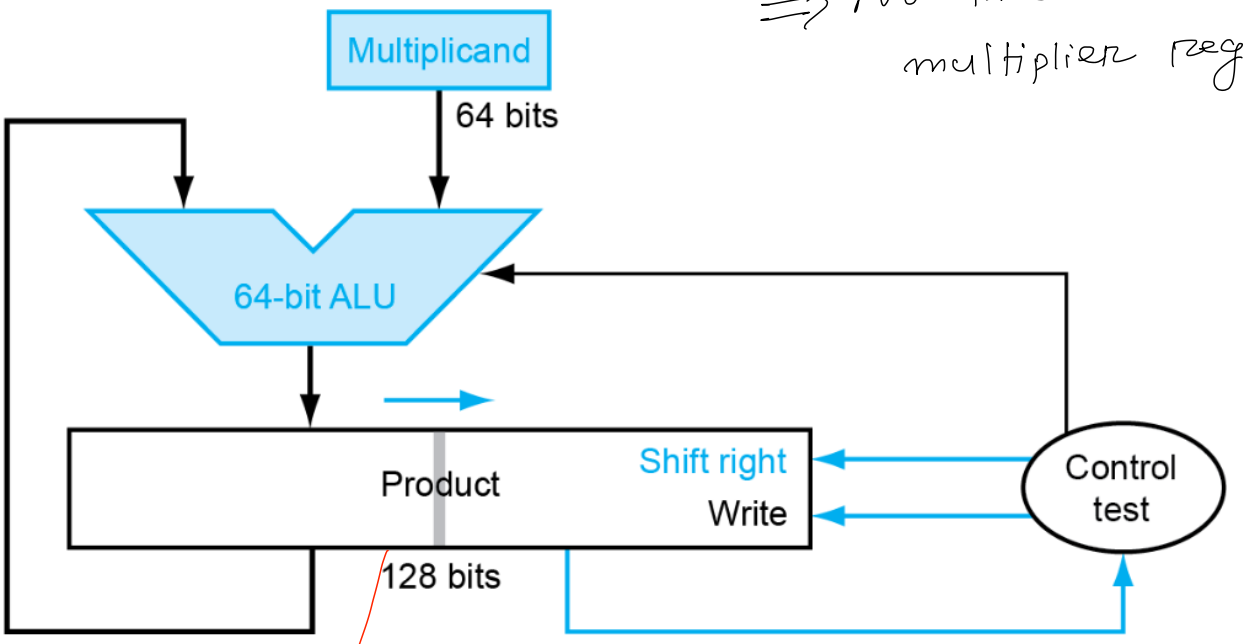
Number of iteration = Number of bits in Multiplier

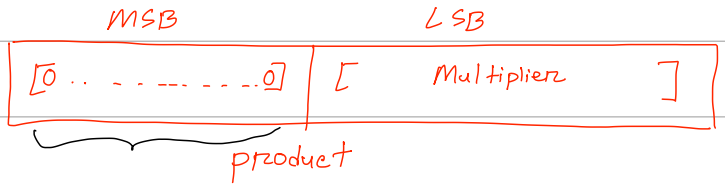
⇒ 1000 × 1001

iteration	Multiplier	Multiplicand	Product
0	1001	0000 1000	0000 0000
	1001	0000 1000	0000 1000
1	1001	0001 0000	0000 1000
	0100	0001 0000	0000 1000
	0100	0001 0000	0000 1000
2	0100	0010 0000	0000 1000
	0010	0010 0000	0000 1000
	0010	0010 0000	0000 1000
3	0010	0100 0000	0000 1000
	0001	0100 0000	0000 1000
	0001	0100 0000	0100 1000
4	0001	1000 0000	0100 1000
	0000	1000 0000	0100 1000

Optimized Multiplication

⇒ No individual multiplier reg.





Number of iteration = Number of bits in Multiplier

logic:

```
if (iteration <= multiplier bit length):
    if (multiplier_0 == 1):
        product_MSB = Multiplicand + product_MSB
        product = right shift product by 1
    elif (multiplier_0 == 0):
        product = right shift product by 1
```

$8 \times 9 = 1000 \times 1001$

iteration	multiplicand	product_MSB	product
0	1000	0000	1001
1	1000	1000	1001
		0100	0100
2	1000	0010	0010
3	1000	0001	0001
4	1000	1001	0001
		0100	1000

Floating Point

How does RISC-V support numbers with ~~fractions~~ ?

Scientific Notation is just a way to represent very large or very small number.

$\Rightarrow 4500000 = 4.5 \times 10^{6}$

Coefficient, Base, Exponent

$\Rightarrow 0.00453 = 4.53 \times 10^{-3}$

Decimal

- $\Rightarrow 5.64 \times 10^{33}$
- $\Rightarrow -2.34 \times 10^{56}$

→ Normalized.

- $\Rightarrow 109.64 \times 10^{33}$
- $\Rightarrow 0.002 \times 10^{-4}$
- $\Rightarrow +987.02 \times 10^{9}$

→ Not Normalized.

In Binary,

$\pm 1.xxxx_2 \times 2^{yyy}$

IEEE-754 → defines floating point standard.

- Single precision. (32-bit)
- Double precision. (64-bit)

In case of double precision, using more bits, you can represent a larger or a smaller number than single precision.