Chapter 3

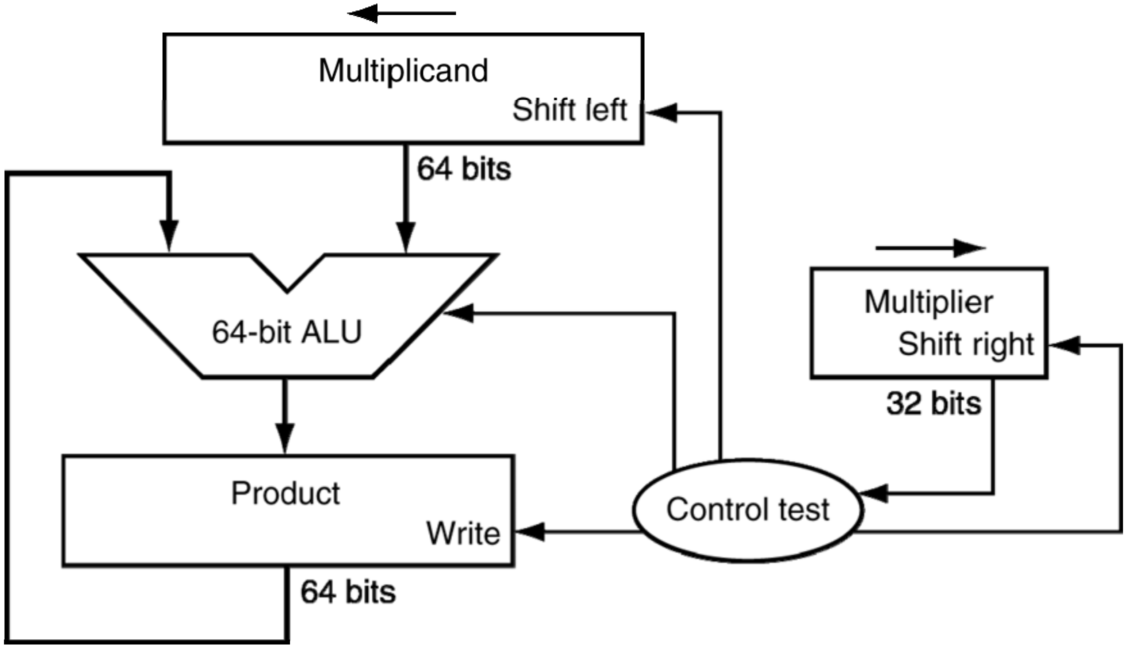
Arithmetic for Computers

Multiplication

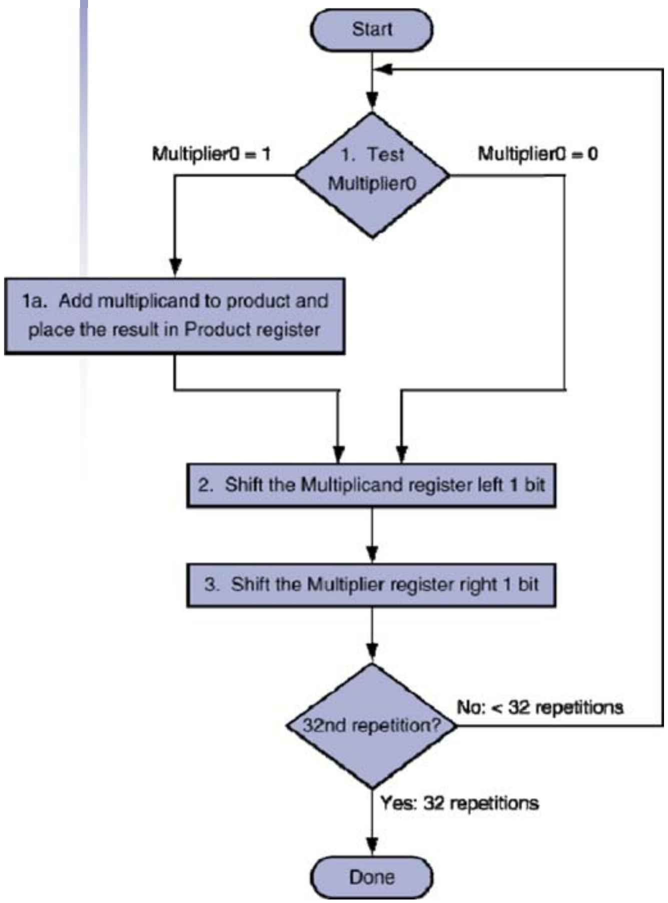
- Start with long-multiplication approach

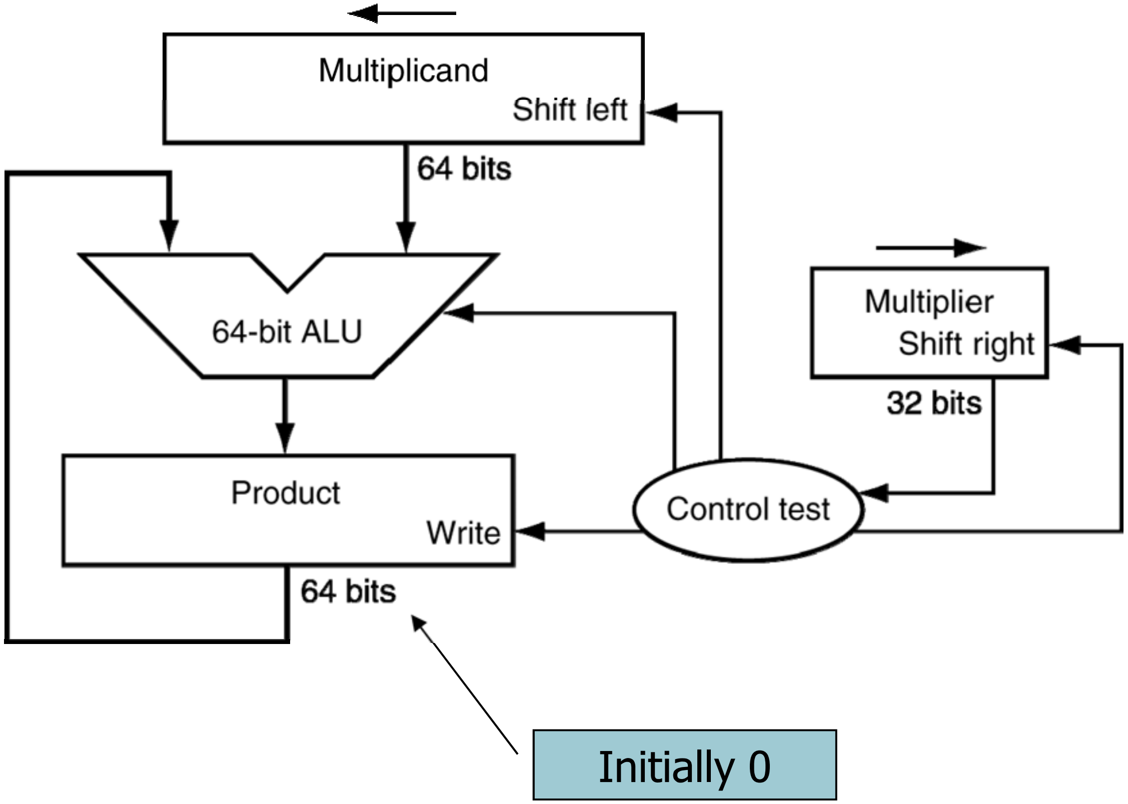
```
multiplicand →     1000
multiplier   →   × 1001
                 -------
                   1000
                  0000
                 0000
                1000
                -------
product      →  1001000
```

Length of product is the sum of operand lengths



Multiplication Hardware

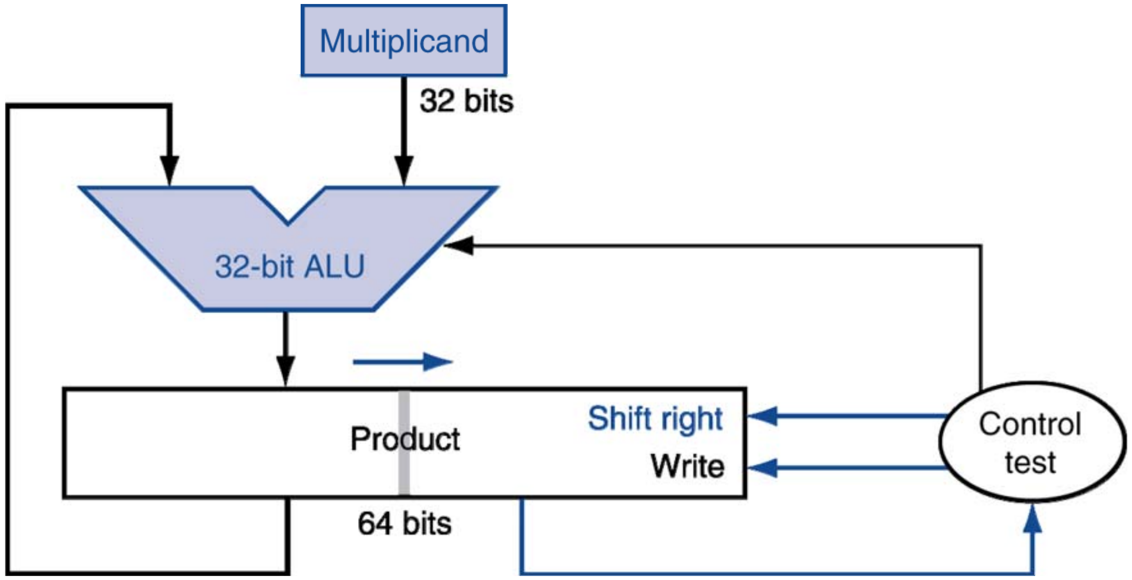

Start
1. Test Multiplier0
Multiplier0 = 1
Multiplier0 = 0
1a. Add multiplicand to product and place the result in Product register
2. Shift the Multiplicand register left 1 bit
3. Shift the Multiplier register right 1 bit
32nd repetition?
No: < 32 repetitions
Yes: 32 repetitions
Done



Initially 0

Optimized Multiplier

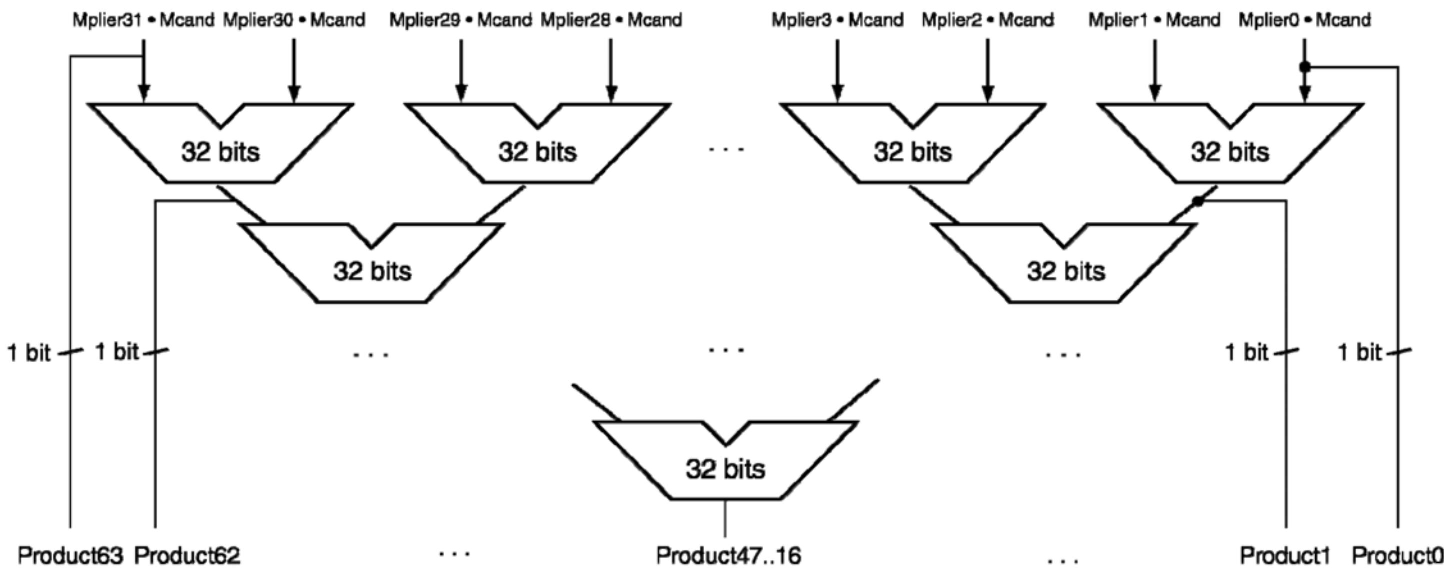
- Perform steps in parallel: add/shift



- One cycle per partial-product addition
  - That’s ok, if frequency of multiplications is low

Faster Multiplier

- Uses multiple adders
  - Cost/performance tradeoff



- Can be pipelined
  - Several multiplication performed in parallel

MIPS Multiplication

- Two 32-bit registers for product
  - HI: most-significant 32 bits
  - LO: least-significant 32-bits
- Instructions
  - `mult rs, rt  /  multu rs, rt`
    - 64-bit product in HI/LO
  - `mfhi rd  /  mflo rd`
    - Move from HI/LO to rd
    - Can test HI value to see if product overflows 32 bits
  - `mul rd, rs, rt`
    - Least-significant 32 bits of product –> rd

Chapter 3

Arithmetic for Computers

Signed Multiplication?

- Make both positive
  - remember whether to complement product when done
- Apply definition of 2’s complement
  - need to sign-extend partial products and subtract at the end
- Booth’s Algorithm
  - elegant way to multiply signed numbers
    - using same hardware as before and save cycles

Motivation for Booth’s Algorithm

- Example 2 x 6 = 0010 x 0110:

```
             0010
x            0110
+            0000      shift (0 in multiplier)
+           0010       add (1 in multiplier)
+          0010        add (1 in multiplier)
+         0000         shift (0 in multiplier)
          00001100
```

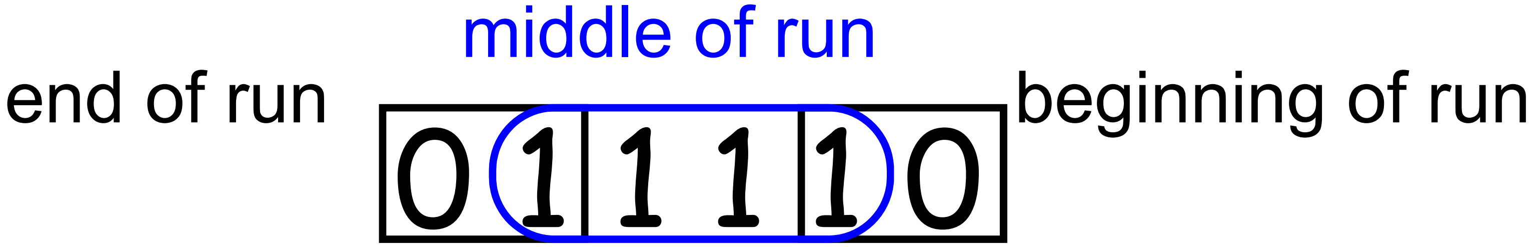
- ALU with add or subtract gets same result in more than one way:

```
14     = 2 + 4 + 8
14     = – 2 + 16
001110 = – 000010 + 010000 = 111110 + 010000
```

- For example

```
             0010
x            0110
             0000 shift (0 in multiplier)
–           0010  sub (first 1 in multpl.)  .
           0000   shift (mid string of 1s)  .
+         0010    add (prior step had last 1)
          00001100
```

Booth’s Algorithm



Current Bit	Bit to the Right	Explanation	Example	Op
1	0	Begins run of 1s	0001111000	sub
1	1	Middle of run of 1s	0001111000	none
0	1	End of run of 1s	0001111000	add
0	0	Middle of run of 0s	0001111000	none

Originally for Speed (when shift was faster than add)

- Replace a string of 1s in multiplier with an initial subtract when we first see a one and then later add for the bit after the last one

$$\begin{array}{r} -1 \\ +\ 10000 \\ \hline 01111 \end{array}$$