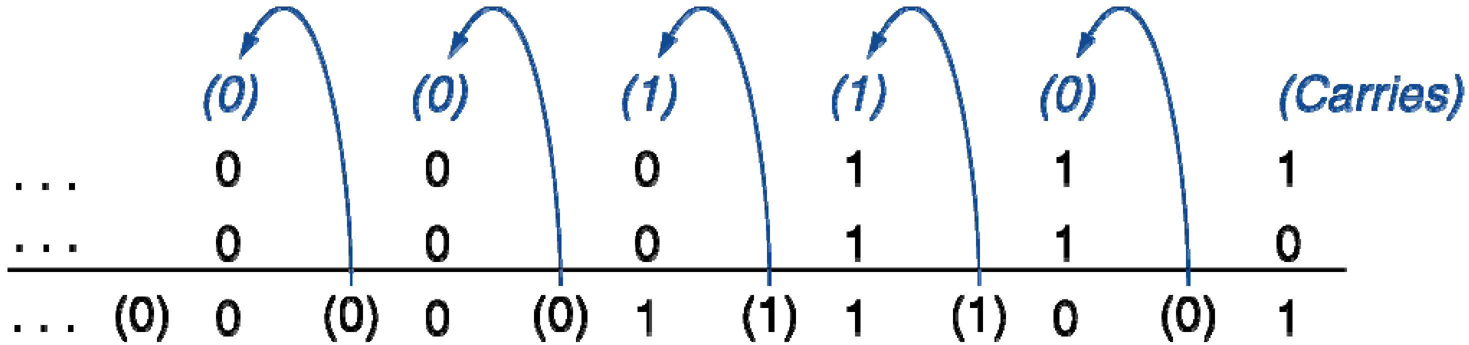
Arithmetic

Integer Addition

- Example: 7 + 6

Integer Addition

- Example: 7 + 6



Integer Subtraction

- How to represent negative numbers?

Sign and Magnitude Representation

- Use Sign Bits
  - Add a sign bit, 0 means positive, 1 means negative

  - $0000\ 0000\ 0000\ 0000\ \ldots\ 0011_2 = +3_{10}$
  - $1000\ 0000\ 0000\ 0000\ \ldots\ 0011_2 = -3_{10}$
  - $0000\ 0000\ 0000\ 0000\ \ldots\ 1010_2 = +10_{10}$
  - $1000\ 0000\ 0000\ 0000\ \ldots\ 1010_2 = -10_{10}$

Disadvantages of Sign and Magnitude

- Ambiguity:
  - Put the sign to left or right?, 0 or 1 ?
  - Two representations for zero!
- Design issues:
  - More complex hardware for adders if they don't know in advance what is the proper sign
  - Software programmers

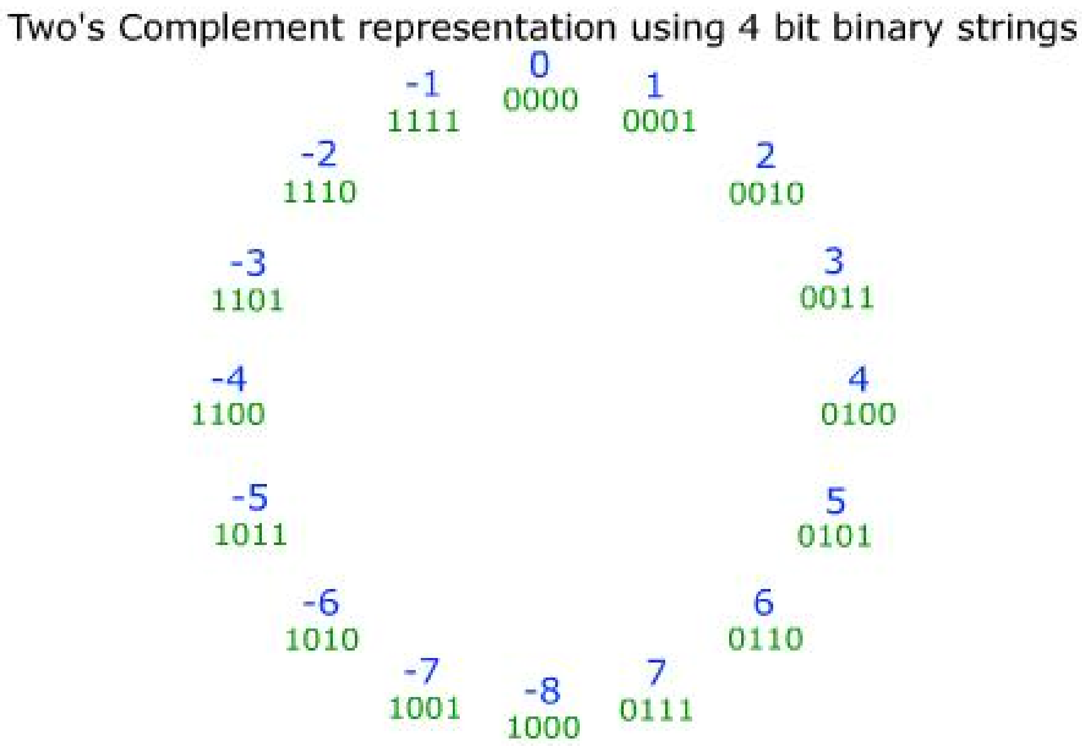
Unsigned Numbers

Unsigned Numbers

- Allows only positive numbers

0000 (0)
0001 (1)
0010 (2)
0011 (3)
0100 (4)
0101 (5)
0110 (6)
0111 (7)
1000 (8)
1001 (9)
1010 (10)
1011 (11)
1100 (12)
1101 (13)
1110 (14)
1111 (15)

Two's complement



- 2-1=1
- 1-1=0
- 0-1=-1
- …

Two's complement

- $x + x_c = -1$
  - $x + x_c + 1 = 0$
    - $x_c + 1 = -x$

- Short cut technique
  - Complement all bits (change all 1's to 0's and all 0's to 1's)
  - Then add 1
  - Ignore carries for the last (left most) column

Two's complement

- Short cut technique
  - Complement all bits (change all 1's to 0's and all 0's to 1's) then add 1
- Assume word length of 8 bits. Express -4 in two's complement form.

Two's complement

- Short cut technique
  - Complement all bits (change all 1's to 0's and all 0's to 1's) then add 1
- Assume word length of 8 bits. Express -4 in two's complement form.

+4 =	0000 0100
Complement :	1111 1011
-4 =	1111 1100

Integer Subtraction

- Add 5 to -5 using two's complement

Overflows

- Subtraction: Overflow if result out of range
  - Subtracting two +ve or two –ve operands, no overflow
  - Subtracting +ve from –ve operand
    - Overflow if result sign is 0
  - Subtracting –ve from +ve operand
    - Overflow if result sign is 1

- Addition: Overflow if result out of range
  - Adding two +ve operands
    - Overflow if result sign is 1

Dealing with Overflow

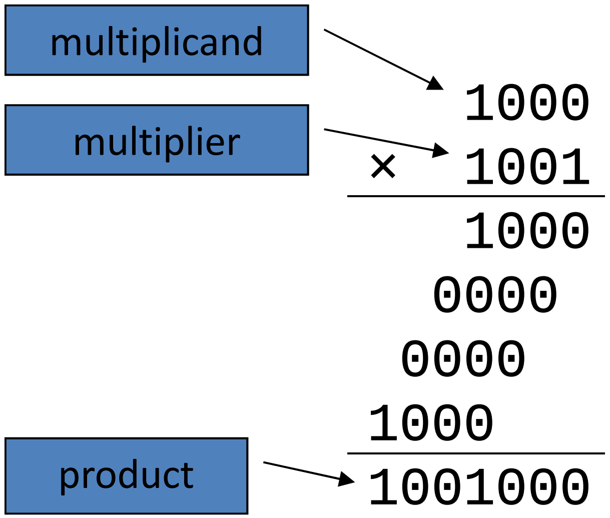
- Some languages (e.g., C) ignore overflow
  - Use MIPS `addu`, `addui`, `subu` instructions
- Other languages (e.g., Ada, Fortran) require raising an exception
  - Use MIPS `add`, `addi`, `sub` instructions

Dealing with Overflow

- If overflow occurs,
  - Address of instruction causing overflow is saved in a register
  - Computer jumps to a pre-defined address to invoke a special procedure

Multiplication

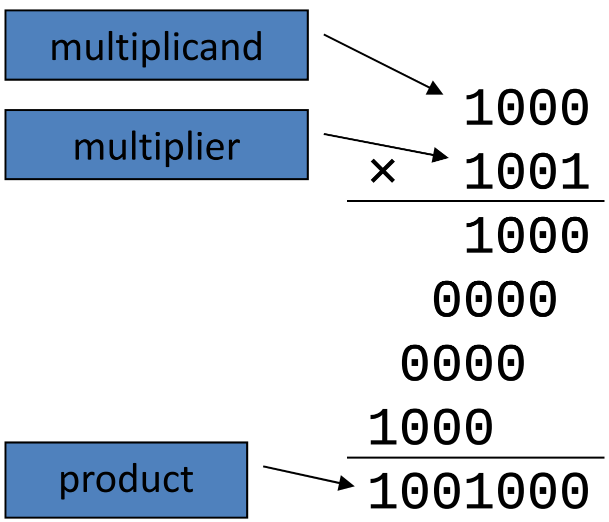
- Start with long-multiplication approach



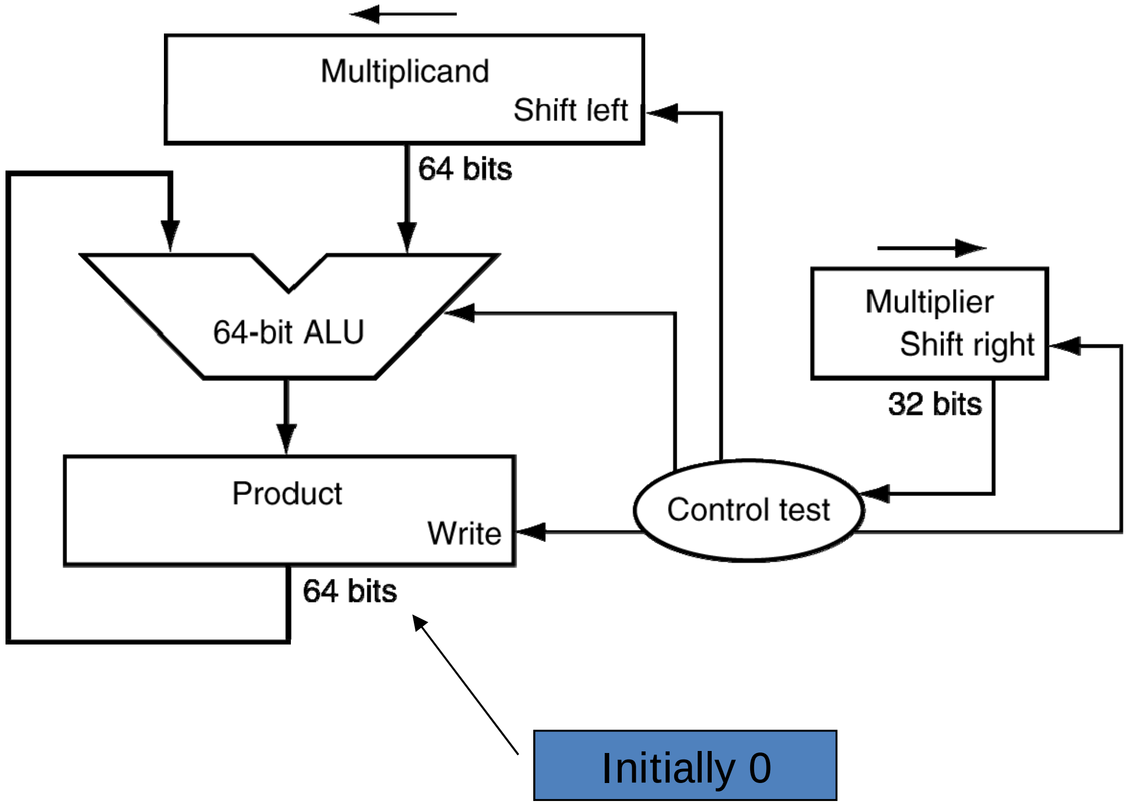
Length of product is the sum of operand lengths

Multiplication

- Start with long-multiplication approach



Length of product is the sum of operand lengths

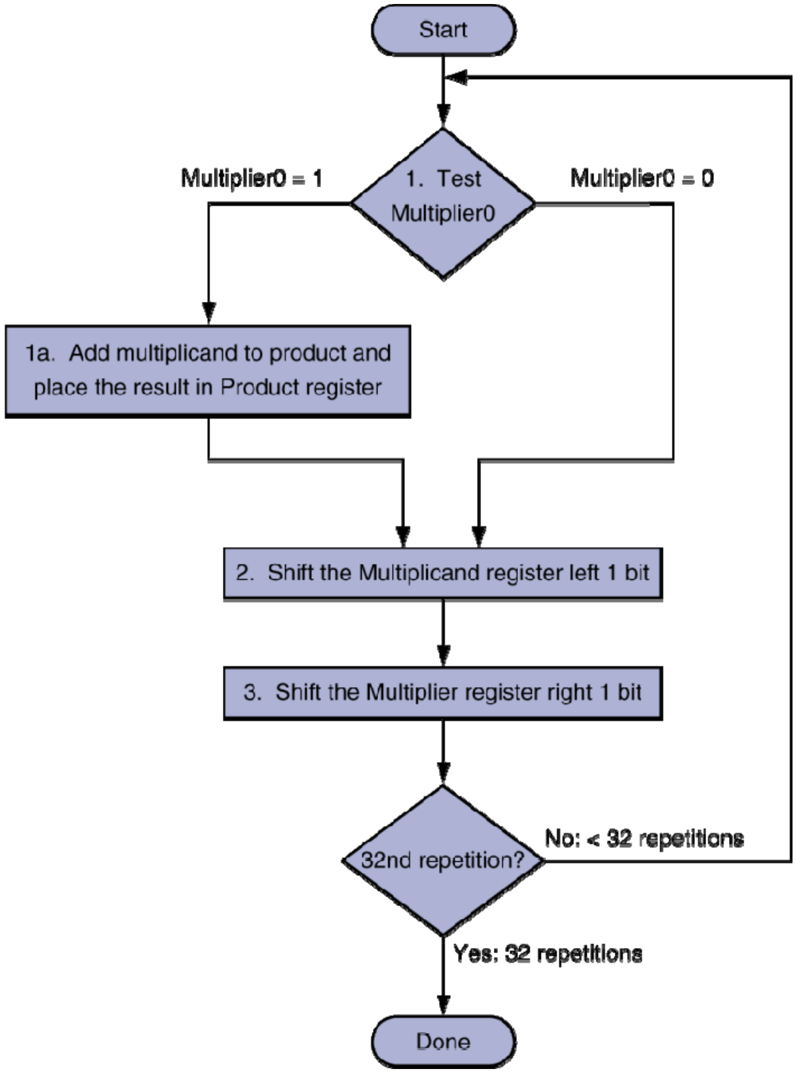


Multiplication

- Start with long-multiplication approach

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

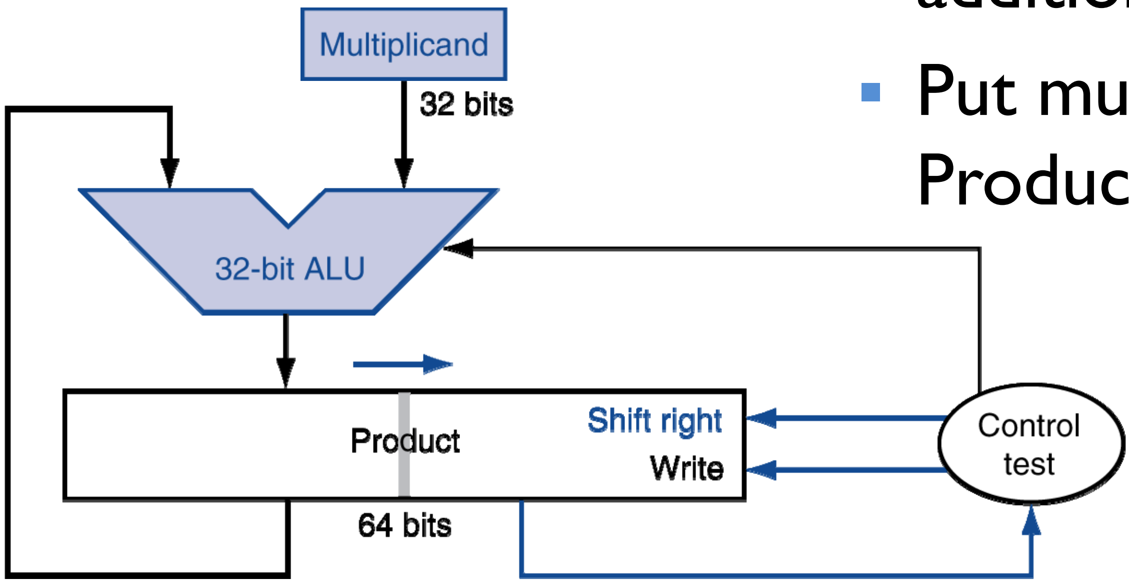
Length of product is the sum of operand lengths



Multiplication

- Long multiplication is slow. 3 steps, add, shift, & shift are repeated 32 times – almost 100 cycles for one multiplication instruction!!

Optimized Multiplier

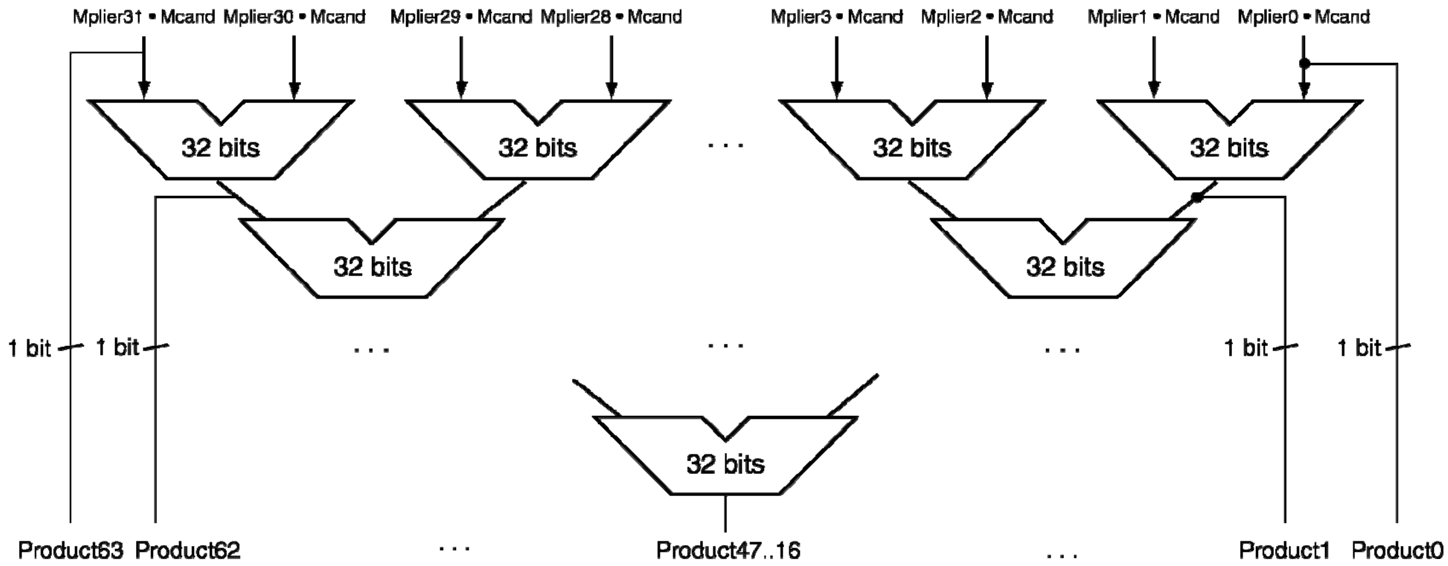


- Save hardware (32 bit adders and multiplicand registers)
- Faster, because shifting and addition can be in parallel
- Put multiplier in 32 bits of the Product register

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

Faster Multiplier

- Uses multiple adders
  - Cost/performance tradeoff



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
  - `mflo and mfhi to move result from HI/LO to a general purpose register`

History

- Registers were precious
  - Single register called Accumulator
  - Add 200

- Separate accumulators for special operations like multiply, divide ..

History

- General purpose registers – any register may be used for any purpose. E.g, MIPS

- Register-memory architecture: allow memory operands others demand operands must always be in registers - MIPS