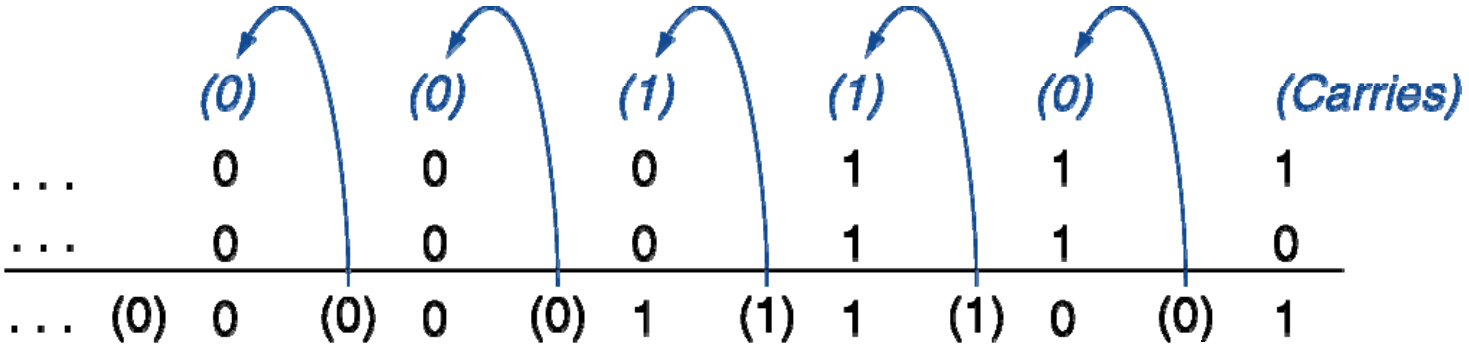
Arithmetic

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

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

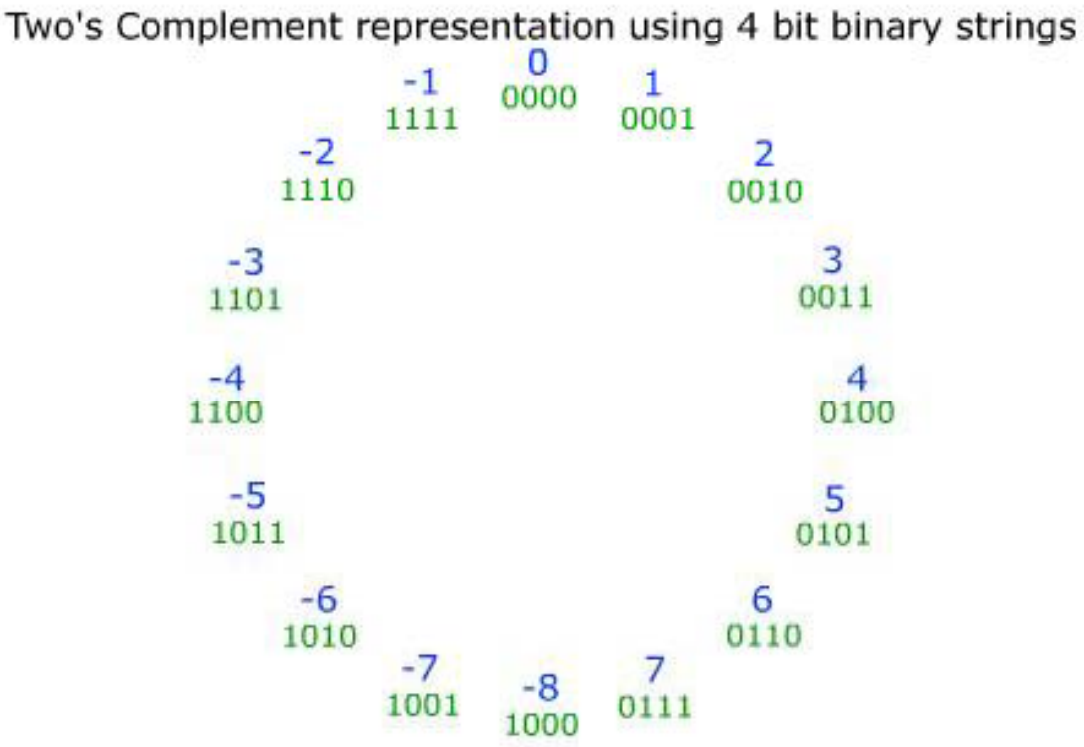
Unsigned Numbers

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

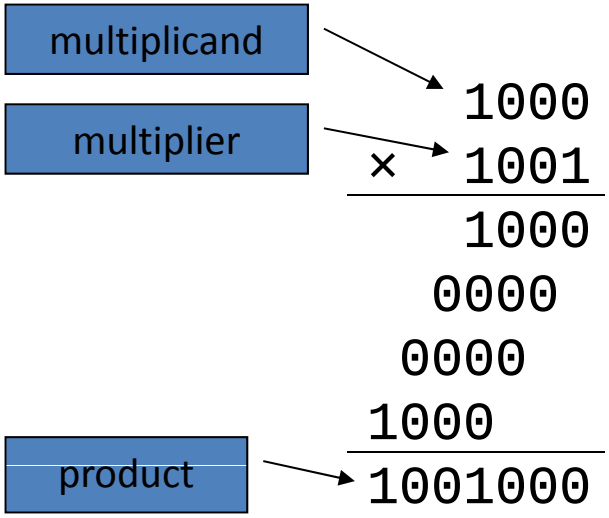
- If overflow occurs,
  - Address of instruction causing overflow is saved in a register
  - Computer jumps to a pre-defined address to invoke a special procedure

Arithmetic for Multimedia

- Graphics and media processing operates on vectors of 8-bit and 16-bit data
  - Use 64-bit adder, with partitioned carry chain
    - Operate on 8×8-bit, 4×16-bit, or 2×32-bit vectors
  - SIMD (single-instruction, multiple-data)
- Saturating operations
  - On overflow, result is largest representable value
    - c.f. 2s-complement modulo arithmetic
  - E.g., clipping in audio, saturation in video

Multiplication

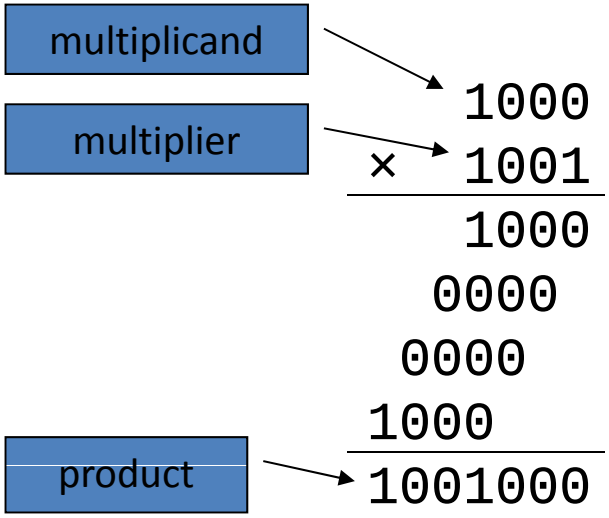
- Start with long-multiplication approach



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

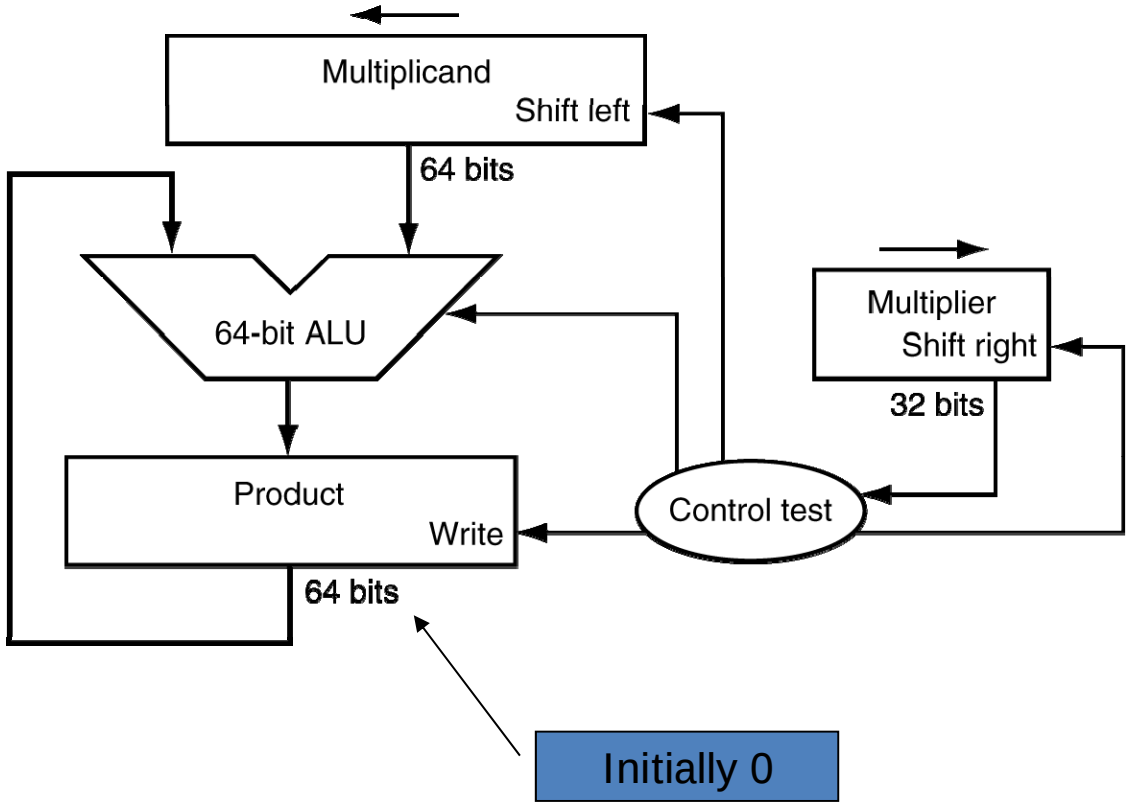
multiplicand

multiplier

product

```
  1000
× 1001
------
  1000
 0000
0000
1000
------
1001000
```

Length of product is the sum of operand lengths



Multiplication

- Start with long-multiplication approach

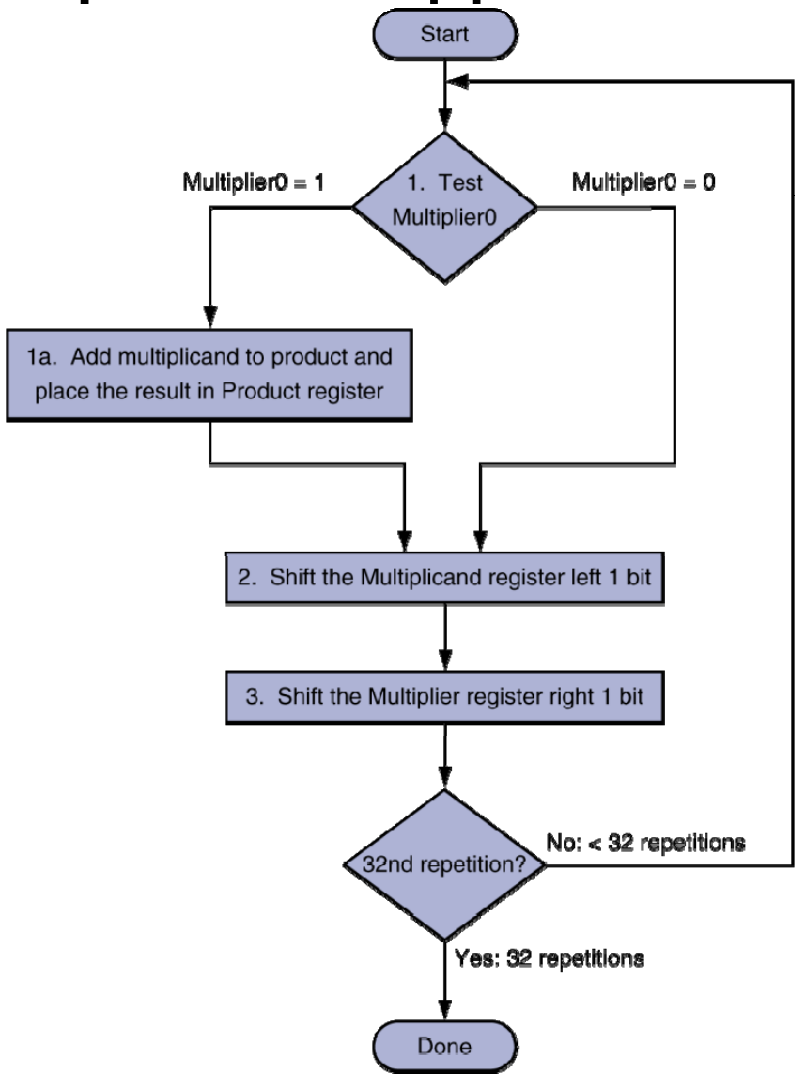
multiplicand → 1000

multiplier → × 1001

```
  1000
 0000
0000
1000
1001000
```

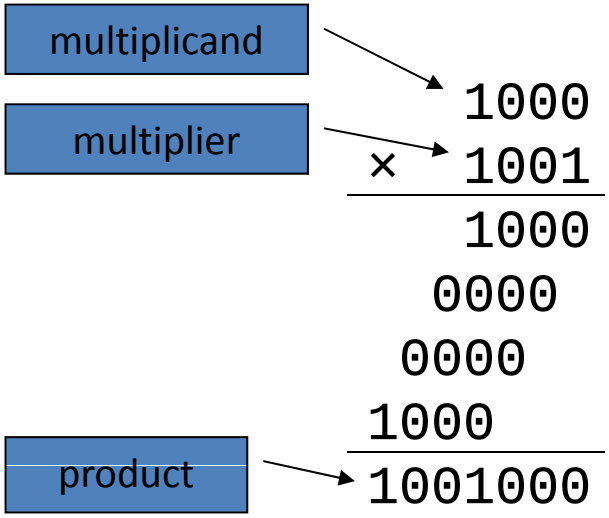
product → 1001000

Length of product is the sum of operand lengths

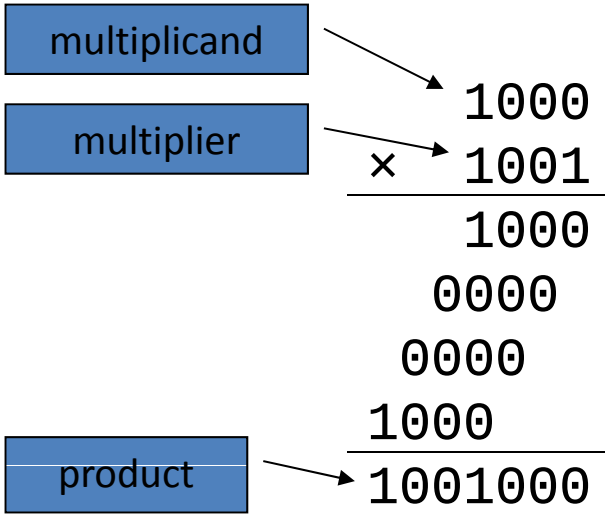


Multiplication

- Shift product to right



Length of product is the sum of operand lengths

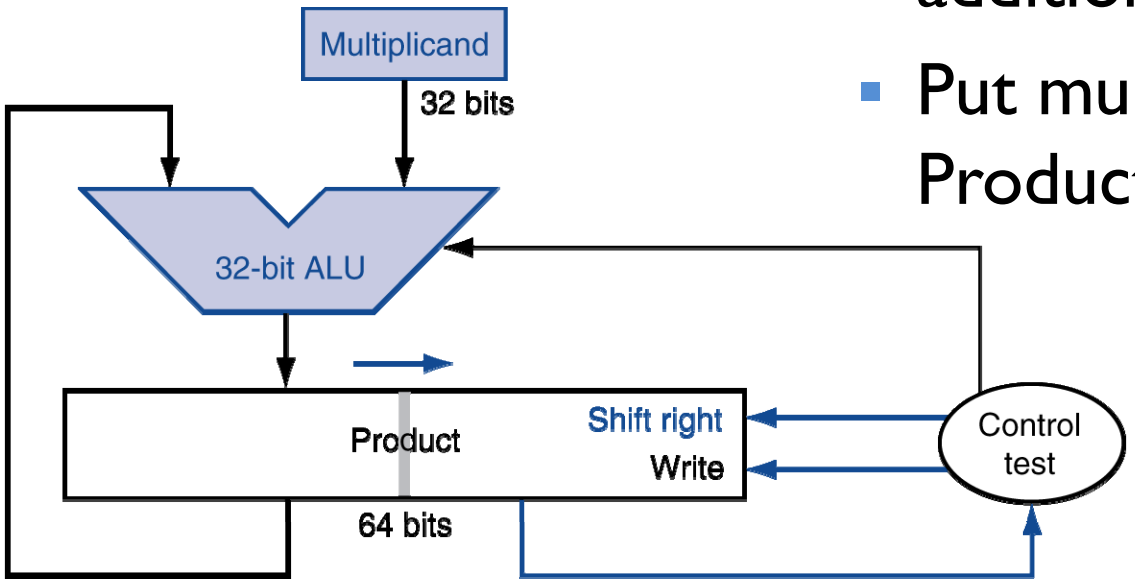


Length of product is the sum of operand lengths

- 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

Optimized Multiplier

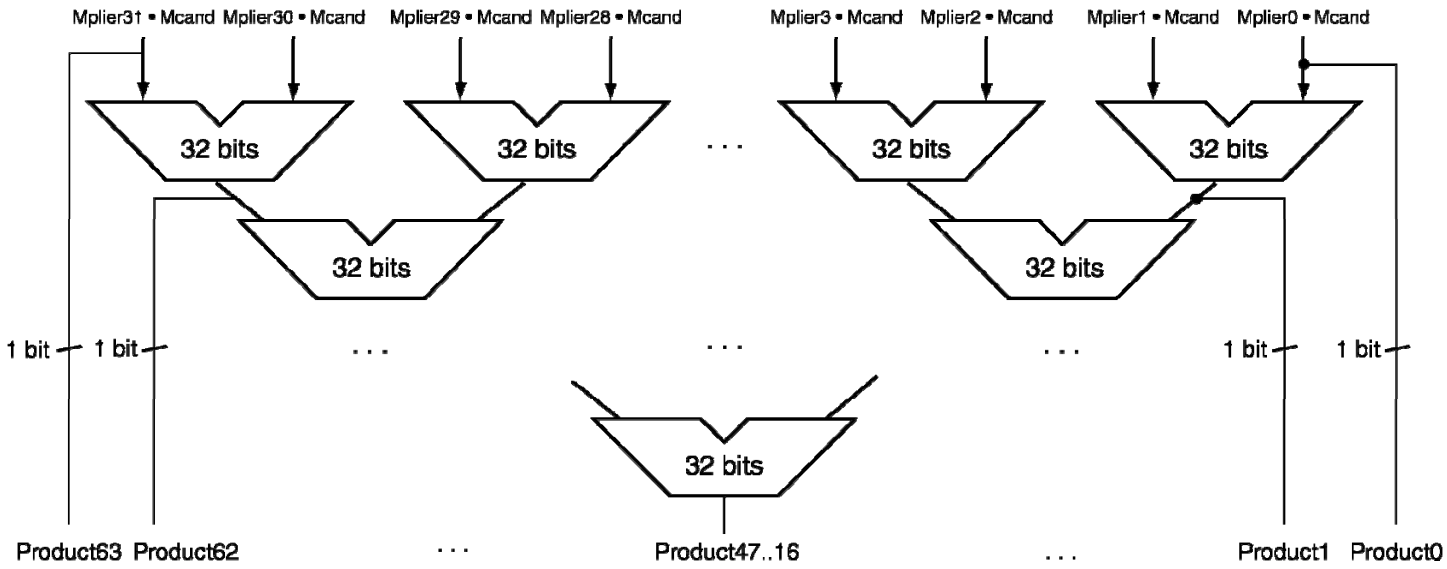
- Save hardware (32 bit adders and multiplicand registers)
- Faster, because shifting and addition can be in parallel
- Put multiplier in 32 bits of the Product register



- 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

Hexadecimal

- Reading binary numbers are tedious
- So, hexadecimal representation is popular
- Base 16 is power of two and hence it is easy to replace each group of 4 binary digits

Hexadecimal

- Base 16
  - Compact representation of bit strings
  - 4 bits per hex digit

0	0000	4	0100	8	1000	c	1100
1	0001	5	0101	9	1001	d	1101
2	0010	6	0110	a	1010	e	1110
3	0011	7	0111	b	1011	f	1111

- Example: eca8 6420 ?
- Example:

0001 0011 0101 0111 1001 1011 1101 1111