Chapter 3

Overflow and Multiplication

Supplementary Slides

Prepared by Fairoz Nower Khan

Lecture 14

- Overflow
- Long Multiplication
- Optimized Multiplication
- Faster Multiplier (basics)
- MIPS Multiplication

Slides prepared by Fairoz Nower Khan Miss

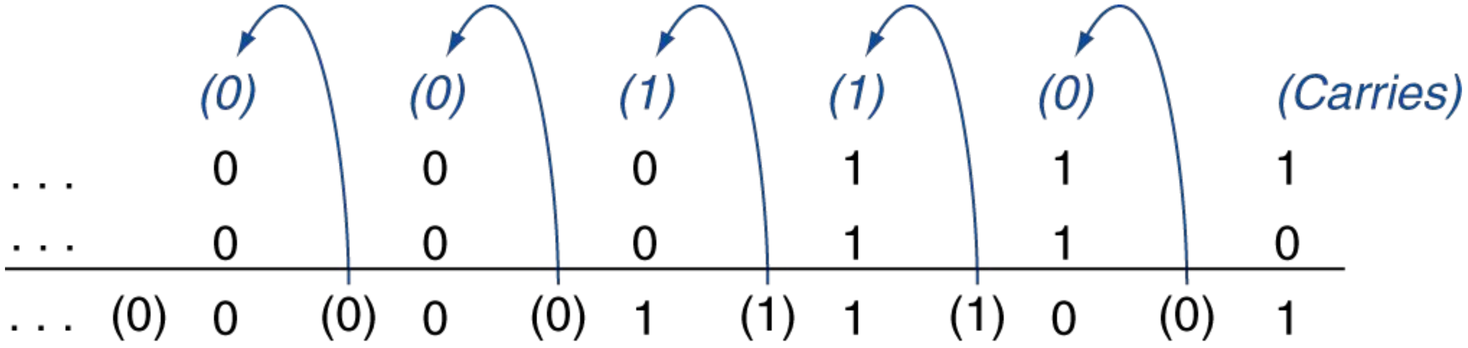
Arithmetic for Computers

§3.1 Introduction

- Operations on integers
  - Addition and subtraction
  - Multiplication and division
  - Dealing with overflow
- Floating-point real numbers
  - Representation and operations

Integer Addition

- Example: 7 + 6



- Overflow if result out of range
  - Adding +ve and –ve operands, no overflow
  - Adding two +ve operands
    - Overflow if result sign is 1
  - Adding two –ve operands
    - Overflow if result sign is 0

§3.2 Addition and Subtraction

Overflow when Addition

Range for n bit Signed Number = -2^(n-1) to + 2^(n-1)-1

Range for 5 bit signed number = -2^(4) to +2^(4)-1
= -16 to +15

No overflow: when adding one positive and one negative binary number

0 1 0 1 1	(11)
1 1 1 1 1	(-1)
0 1 0 1 0	(10)

0 0 0 0 1	(1)
1 1 1 1 0	(1's complement)
1 1 1 1 1	(2's complement)

Overflow: when adding two positive binary numbers *Overflow if sign bit of result is 1*

0 1 0 1 1	(11)
0 1 1 1 1	(15)
1 1 0 1 0	(-6)

2's Complement to Decimal Conversion

11010 = -2^4 + 2^3 + 2^1
= -16 + 8 + 2 = -6

0 1 0 1 0	(10)
0 1 1 0 0	(12)
1 0 1 1 0	(-10)

Overflow: when adding two negative binary numbers *Overflow if sign bit of result is 0*

1 0 1 0 1	(-11)
1 0 0 0 1	(-15)
0 0 1 1 0	(6)

1 0 0 1 1 0 (-26)

2's Complement to Decimal Conversion

100110 = -2^5 + 2^2 + 2^1
= -26

Integer Subtraction

- Add negation of second operand
- Example: 7 – 6 = 7 + (–6)

+7:	0000 0000 … 0000 0111
–6:	1111 1111 … 1111 1010
+1:	0000 0000 … 0000 0001

- 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

3 - 15 = -12

-11 - (-14) = -3

Overflow when Subtraction

Range for 5 bit signed number =
-2^(4) to +2^(4)-1
= -16 to +15

Overflow: when subtracting a negative number from a positive number *Overflow if sign bit of result is 1*

3 - (-15) = 3 + 15

0 0 0 1 1	(3)		1 0 0 0 1	(-15 in 2’s complement)
1 0 0 0 1	(-15)		1 0 0 0 0	(1’s complement)
0 1 1 1 1	(15)		0 1 1 1 1	(15)
1 0 0 1 0	(-14)			

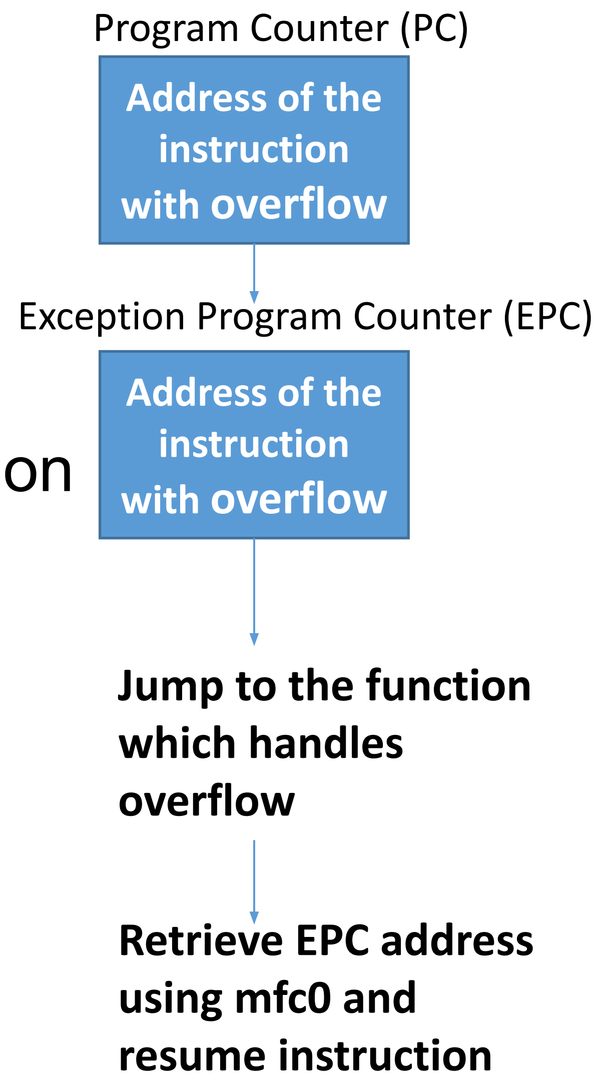
Overflow: when subtracting a positive number from a negative number *Overflow if sign bit of result is 0*

-8 - 10 = -18

1 1 0 0 0	(-8)		0 1 0 1 0	(10)
0 1 0 1 0	(10)		1 0 1 0 1	(1’s complement)
1 0 1 1 0	(-10)		1 0 1 1 0	(2’s complement)
0 1 1 1 0	(14)			

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
  - On overflow, invoke exception handler
    - Save PC in exception program counter (EPC) register
    - Jump to predefined handler address
    - mfc0 (move from coprocessor reg) instruction can retrieve EPC value, to return after corrective action



Multiplication

Start with long-multiplication approach

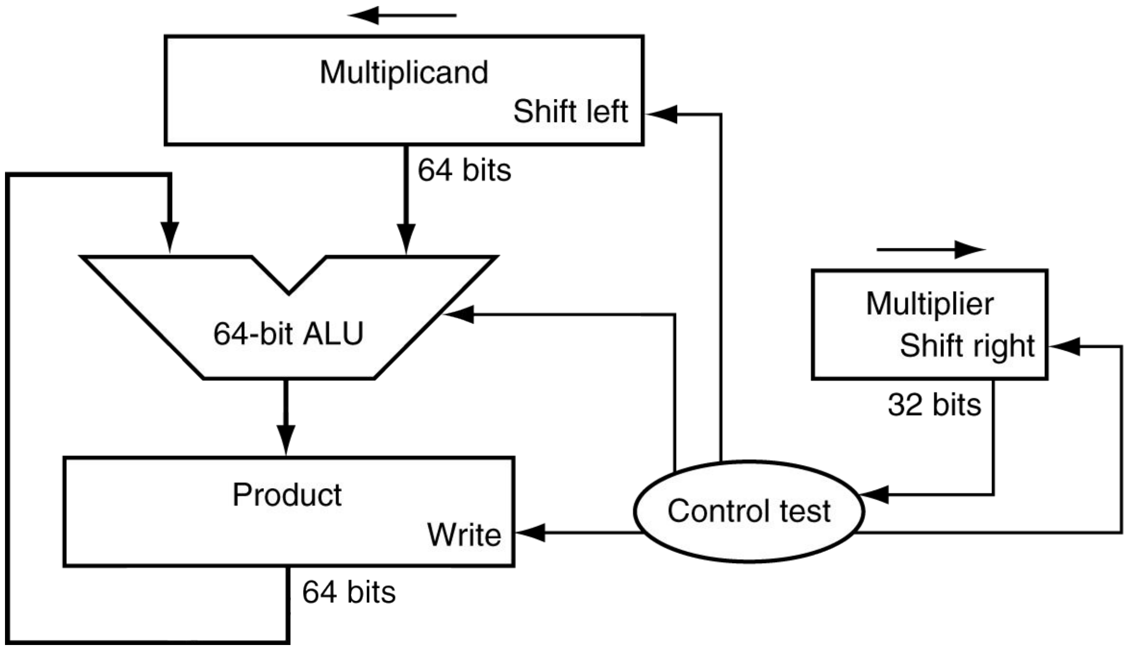
multiplicand

multiplier

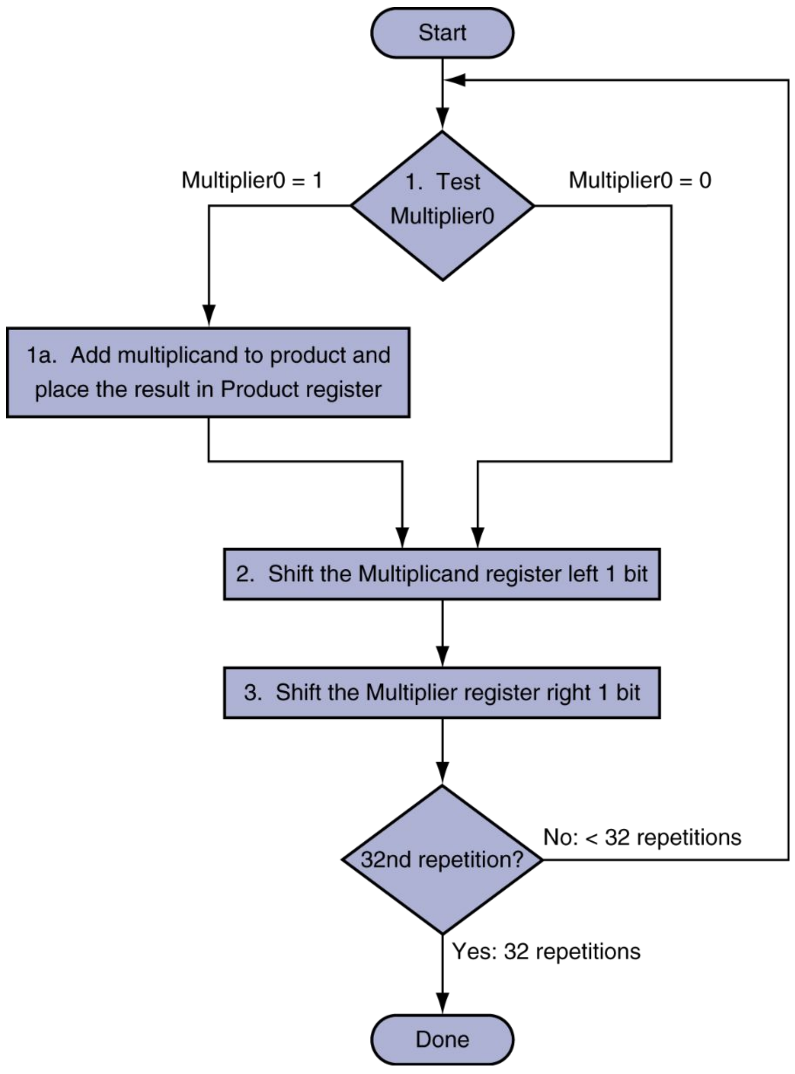
product

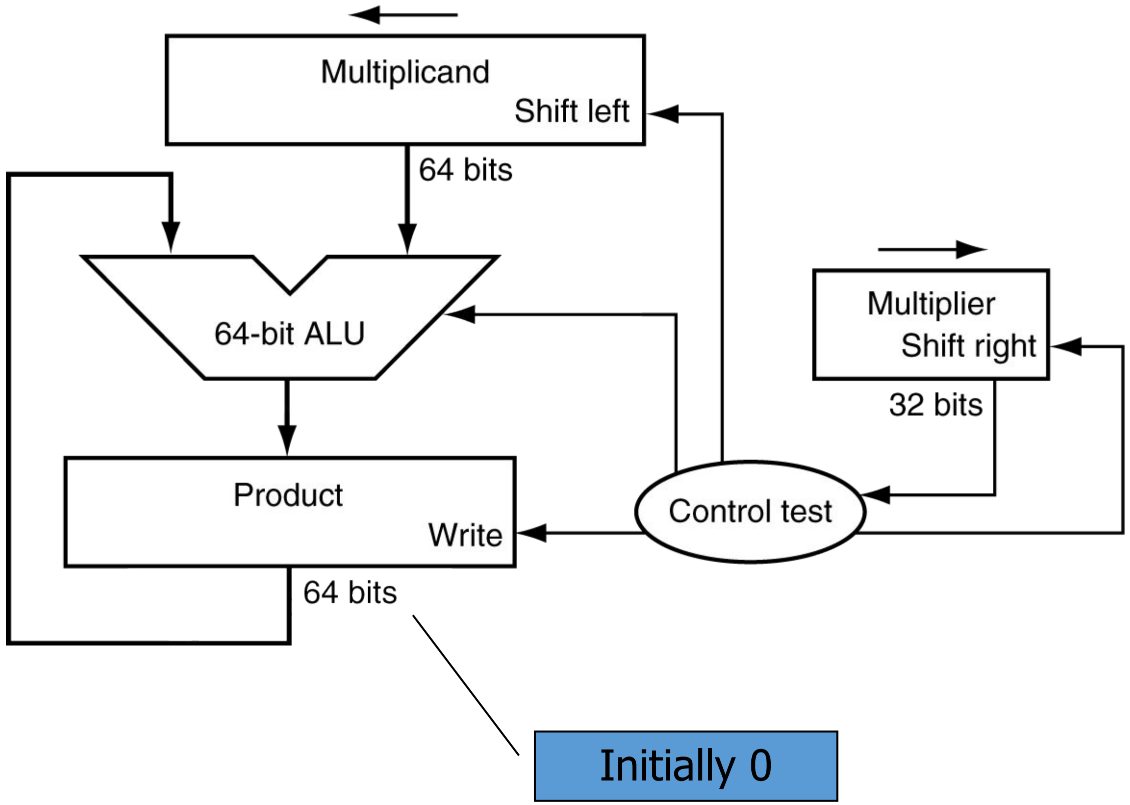
```
      1000
  ×   1001
  --------
      1000
     0000
    0000
   1000
  --------
   1001000
```

Length of product is the sum of operand lengths



Multiplication Hardware (Long Multiplication)





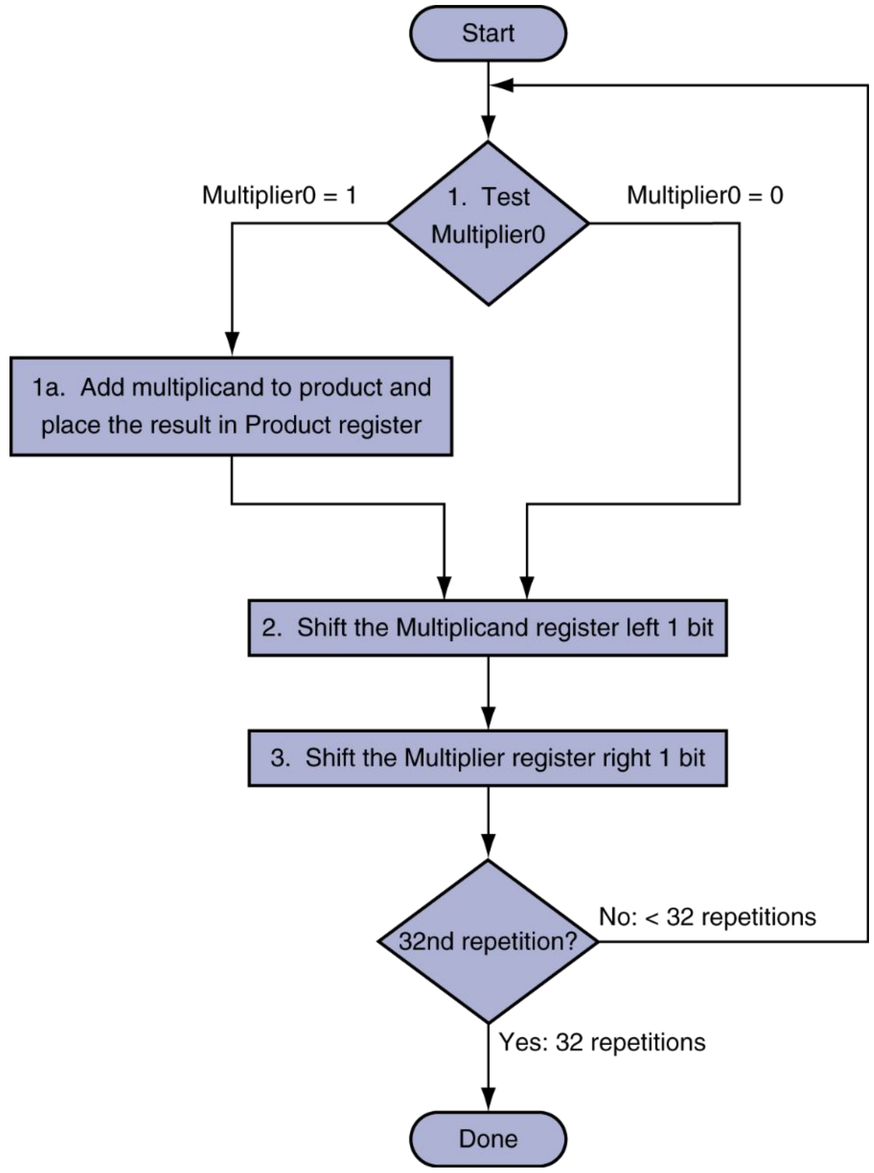
Multiplicand
1000 (8)

Multiplier
1001 (9)

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

Number of iterations = Number of bits in multiplier

Long Multiplication Approach



Multiplication

1000	Multiplicand
1001	Multiplier

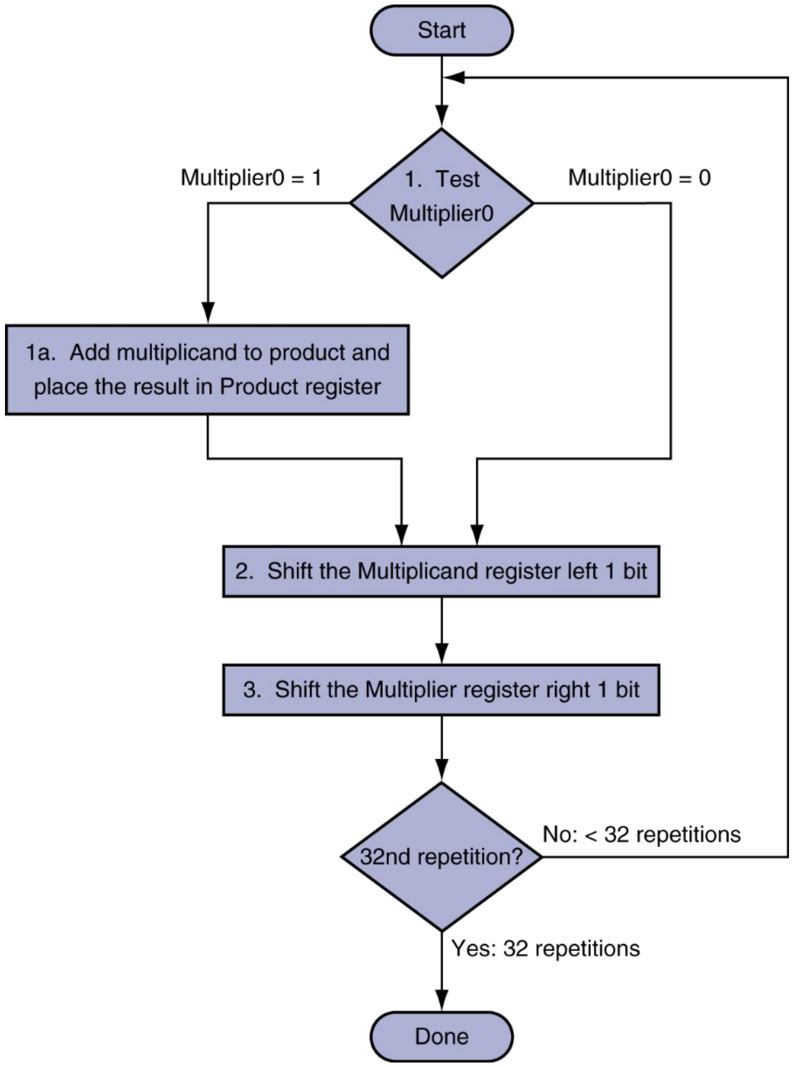
```
   1000
  0000x
 0000xx
1000xxx
-------
1001000
```

Multiplicand
01001 (9)

Multiplier
01010 (10)

Long Multiplication Approach

	Multiplicand 00000 01001	Multiplier 01010	Product 00000 00000
1	00000 10010	00101	00000 00000
2	00001 00100	00010	00000 10010
3	00010 01000	00001	00000 10010
4	00100 10000	00000	00010 11010
5	01001 00000	00000	00010 11010



Number of iterations = Number of bits in multiplier

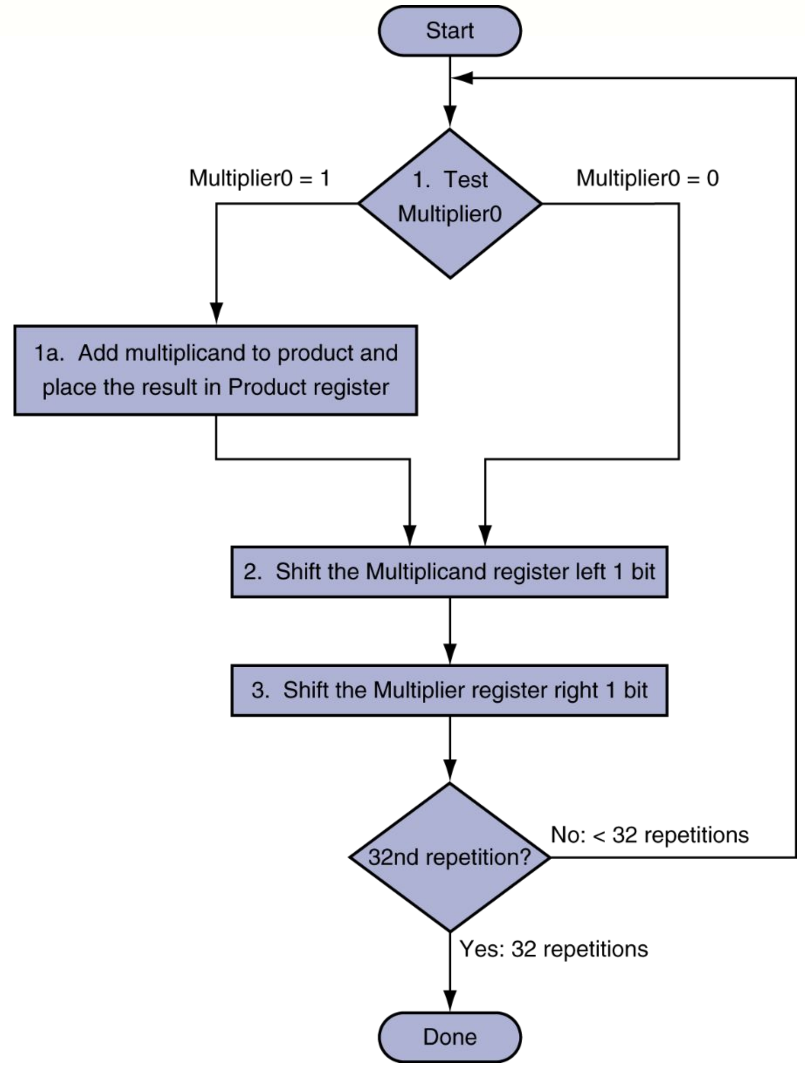
Perform multiplication between 101100 (Multiplicand) and 10110 (Multiplier) using the Long multiplication approach for 6 bit Architecture.

Number of Iteration = bit length of register

Multiplicand	Multiplier
101100 (44)	010110 (22)

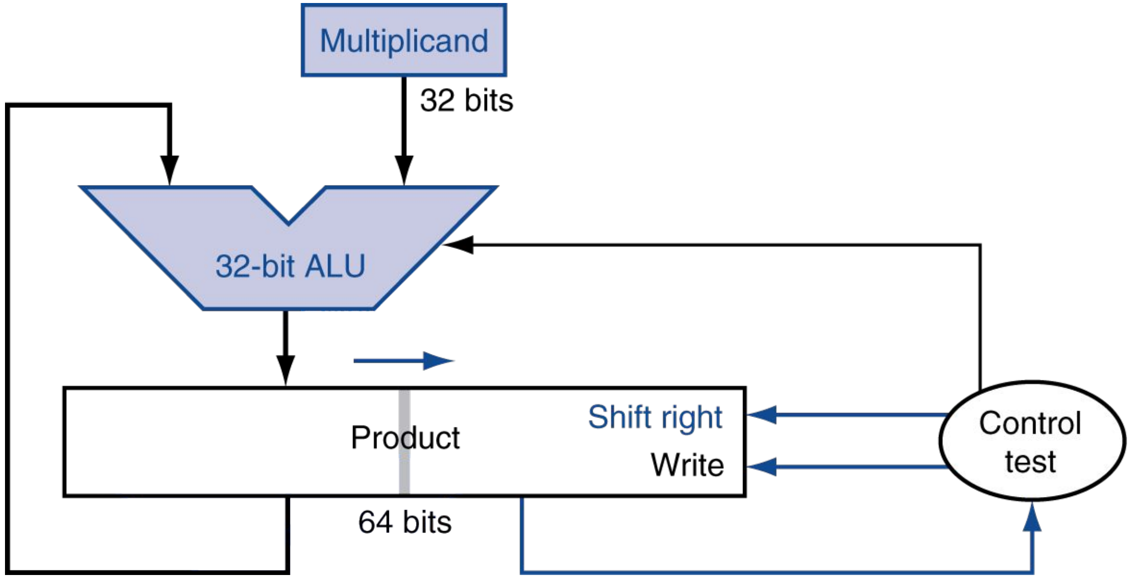
	Multiplicand 000000 101100	Multiplier 010110	Product 000000 000000
1	000001 011000	001011	000000 000000
2	000010 110000	000101	0000001 011000
3	000101 100000	000010	000100 001000
4	001011 000000	000001	000100 001000
5	010110 000000	000000	001111 001000
6	101100 000000	000000	001111 001000

Long Multiplication Approach



Optimized Multiplier

- Perform steps in parallel: add/shift



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

No Changes here

Multiplicand
1000 (8)

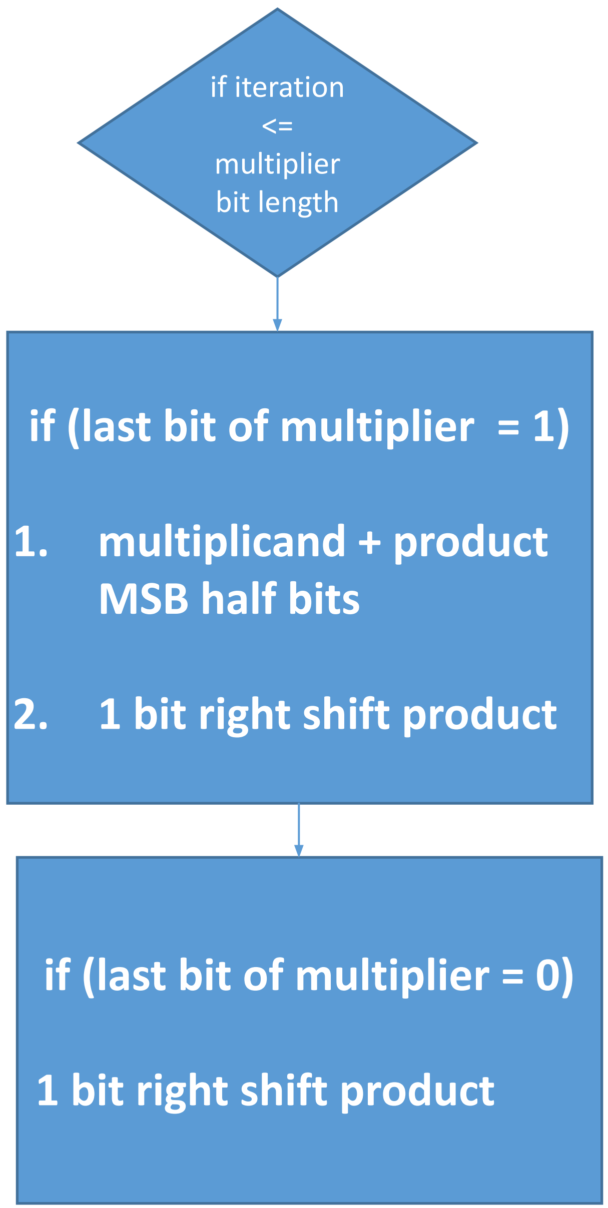
Multiplier
1001 (9)

Product LSB half bits = Multiplier bits

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

Number of iterations = Number of bits in multiplier

Optimized Multiplication Approach



No Changes here

Multiplicand 1000 (8)

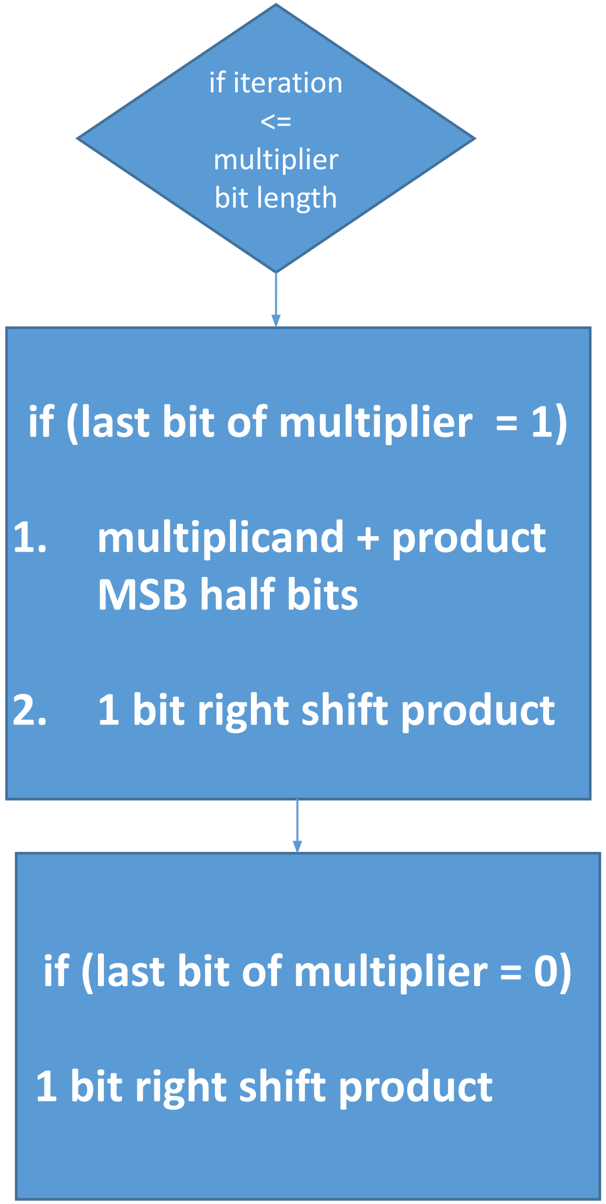
Multiplier 1001 (9)

Product LSB half bits = Multiplier bits

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

Number of iterations = Number of bits in multiplier

Optimized Multiplication Approach



Optimized Multiplication Approach

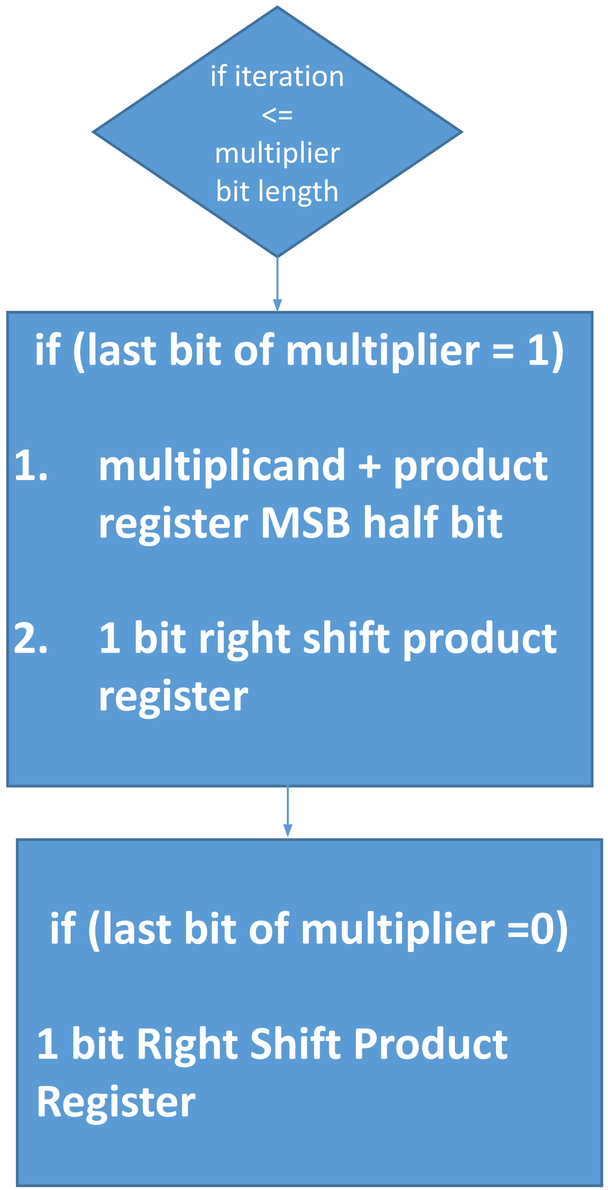
Multiplicand 01001 (9)

Multiplier 01010 (10)

Product LSB half bits = Multiplier bits

	Multiplicand 01001	Product 00000 01010
1	01001	00000 00101
2	01001	01001 00101 00100 10010
3	01001	00010 01001
4	01001	01011 01001 00101 10100
5	01001	00010 11010

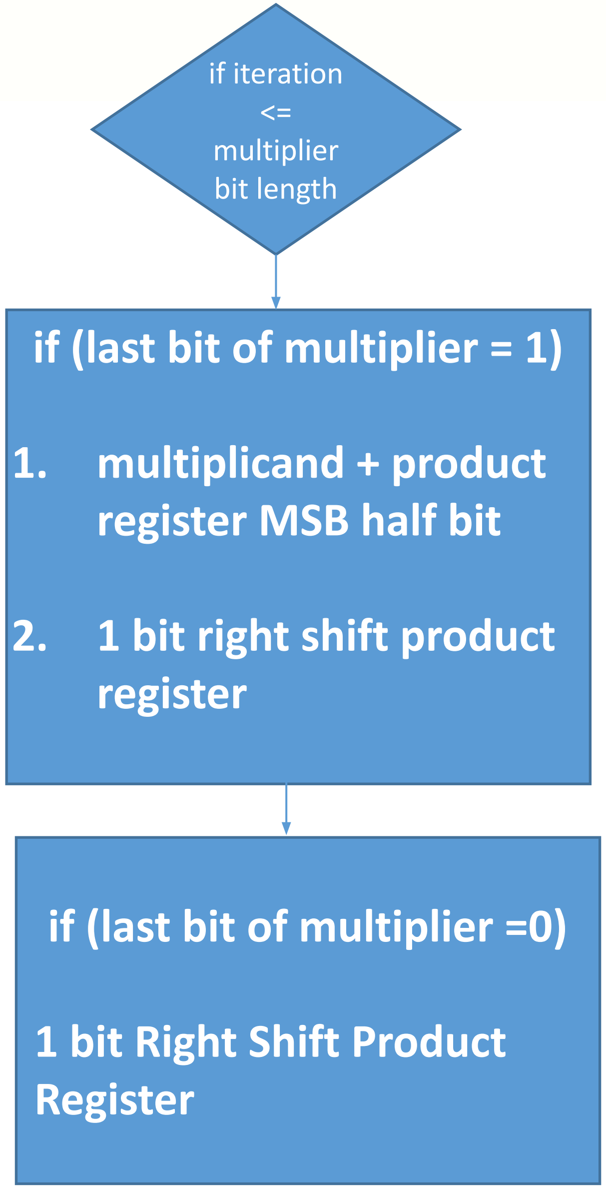
Number of iterations = Number of bits in multiplier



Perform multiplication between 0110 (Multiplicand) and 110 (Multiplier) using the optimized multiplication approach.

Product LSB half bits = Multiplier bits

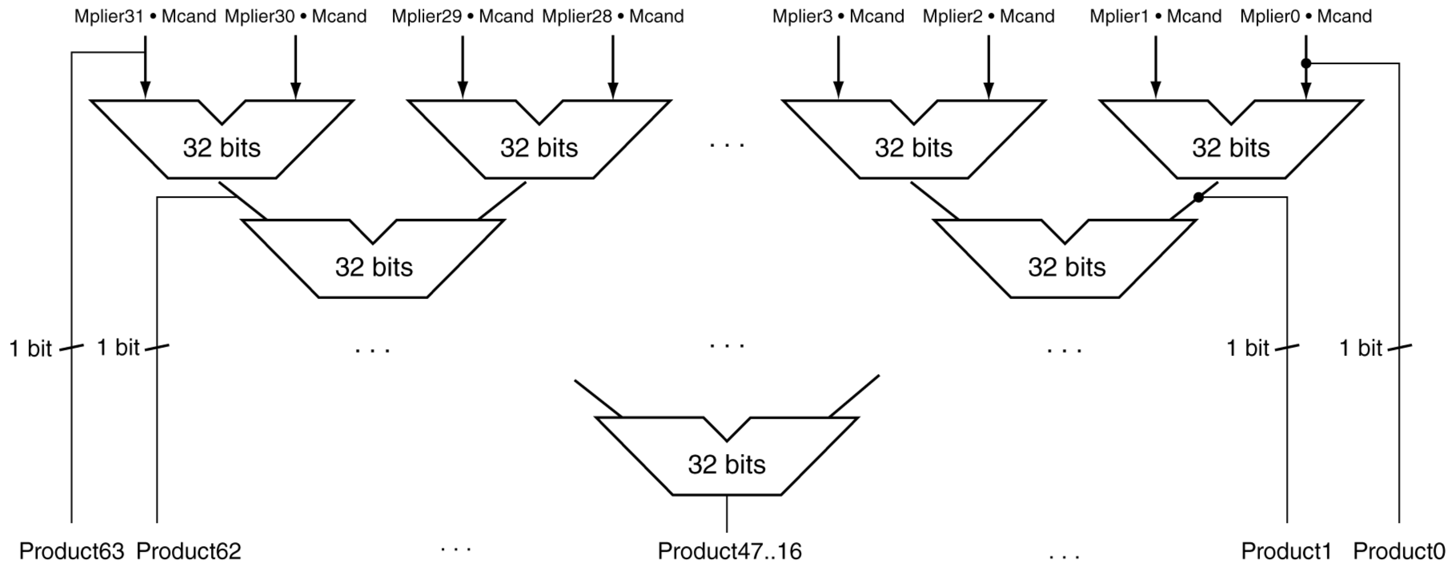
	Multiplicand 0110	Product 0000 0110
1	0110	0000 0011
2	0110	0110 0011 0011 0001
3	0110	1001 0001 0100 1000
4	0110	0010 0100



Number of iterations = Number of bits in multiplier

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