CSCI 341 Computer Organization

– *Chapter 3: Arithmetic for Computers*

Numerical precision is the very soul of science.

Sir D'Arcy Wentworth Thompson
On Growth and Form, 1917

Topics

- Integer Addition and Subtraction (3.1, 3.2)
- Integer Multiplication (3.3)
- Integer Division (3.4)
- Floating point arithmetic (3.5)

Addition & Subtraction

- **Just like in grade school (carry/borrow 1s)**

```
  0111          0111          0110
+ 0110        - 0110        - 0101
```

- **Two's complement makes operations easy**
  - subtraction using addition of negative numbers

```
  0111
+ 1010
```

- **Overflow occurs when**
  - The number that is the proper result of some operations cannot be represented by the hardware bits

Why does Overflow happen?

- **Limited precision in computers**
  - -8 ≤ 4-bit signed number ≤ 7
  - 0 ≤ 4-bit unsigned number ≤ 15
- **Result too large or too small**
  - Example (4-bit unsigned numbers):

+15	1111
+3	0011
+18	10010

When does Overflow happen?

- **No overflow**
  - when adding a positive and a negative number
  - when signs are the same for subtraction
- **Overflow occurs when the value affects the sign:**
  - Carry out occurs into the sign bit
    - when adding two positives yields a negative or vice versa
  - Borrow occurs from the sign bit
    - subtract a negative from a positive and get a negative
    - or, subtract a positive from a negative and get a positive

Overflow vs. Carryout

- **Carry out:**
  - a possible carry value of 1 that results from the addition of the leftmost bits of the operands.
- **Overflow**
  - Is not about the carry-out bit
  - There can be a possible carry value of 1 that results from the addition of the leftmost bits of the operands
  - Is about the sign of the MSB of the result vs. the operands

- **With respect to the addition of two k-bit 2's Complement (2C) integers, occurrence of a carry out is neither necessary nor sufficient for overflow.**

Not Necessary

- **Consider the 2C addition of 10 and 7:**

```
  111
  01010
  00111
 -------
  10001
```

- **Here there is no carry out and yet we have overflow.**
  - 17 lies outside the 5-bit 2C representation range.

Not Sufficient

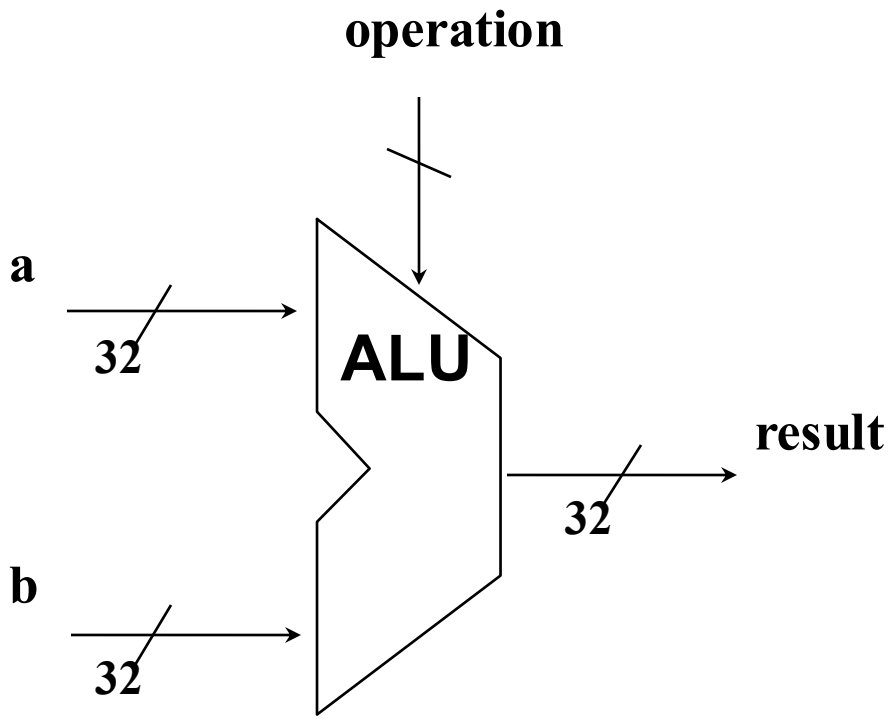
- **Consider the 2's Complement addition of 14 and -5**

Carry out → 1 111
01110
11011
01001

- **Here we have a carry out and yet there is no overflow.**
  - this sum is correct.
  - Overflow never occurs if one operand is nonnegative and the other is negative.

Arithmetic Logical Unit (ALU)

- **Operations**
  - add
  - sub
  - and
  - or
  - xor
  - beq
  - bne
  - blt
  - bge
  - Used in lw, sw



Topics

- Integer Addition and Subtraction (3.1, 3.2)
- ➔**Integer Multiplication (3.3)**
- **Integer Division (3.4)**
- **Floating point arithmetic (3.5)**

Multiplication is vexation, division is as bad; The rule of three doth puzzle me, and practice drives me mad.

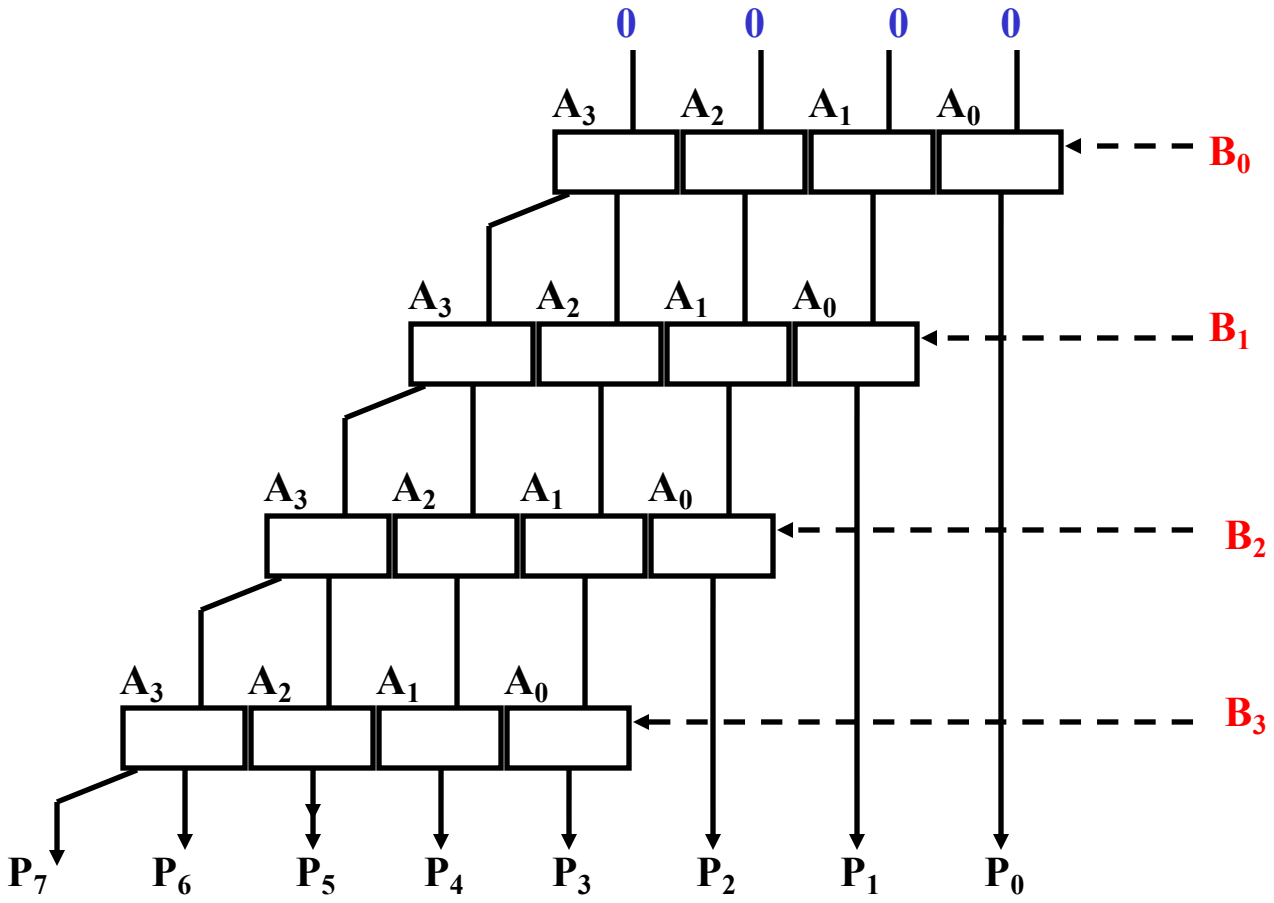
Anonymous, Elizabethan manuscript, 1570

Unsigned Multiplication

					0	1	0	1	1	0
					1	1	0	0	1	1
+					0	1	0	1	1	0
+				0	1	0	1	1	0	
+			0	0	0	0	0	0		
+		0	0	0	0	0	0			
+	0	1	0	1	1	0				
0	1	0	1	1	0					
1	0	0	0	1	1	0	0	0	1	0

- **`Multiplicand` is shifted left**
- **Check each bit of the `multiplier`**
  - shift right and then check LSB
- **Three registers:**
  - Multiplicand: 64 bits
  - Multiplier: 32 bits
  - Product: 64 bits

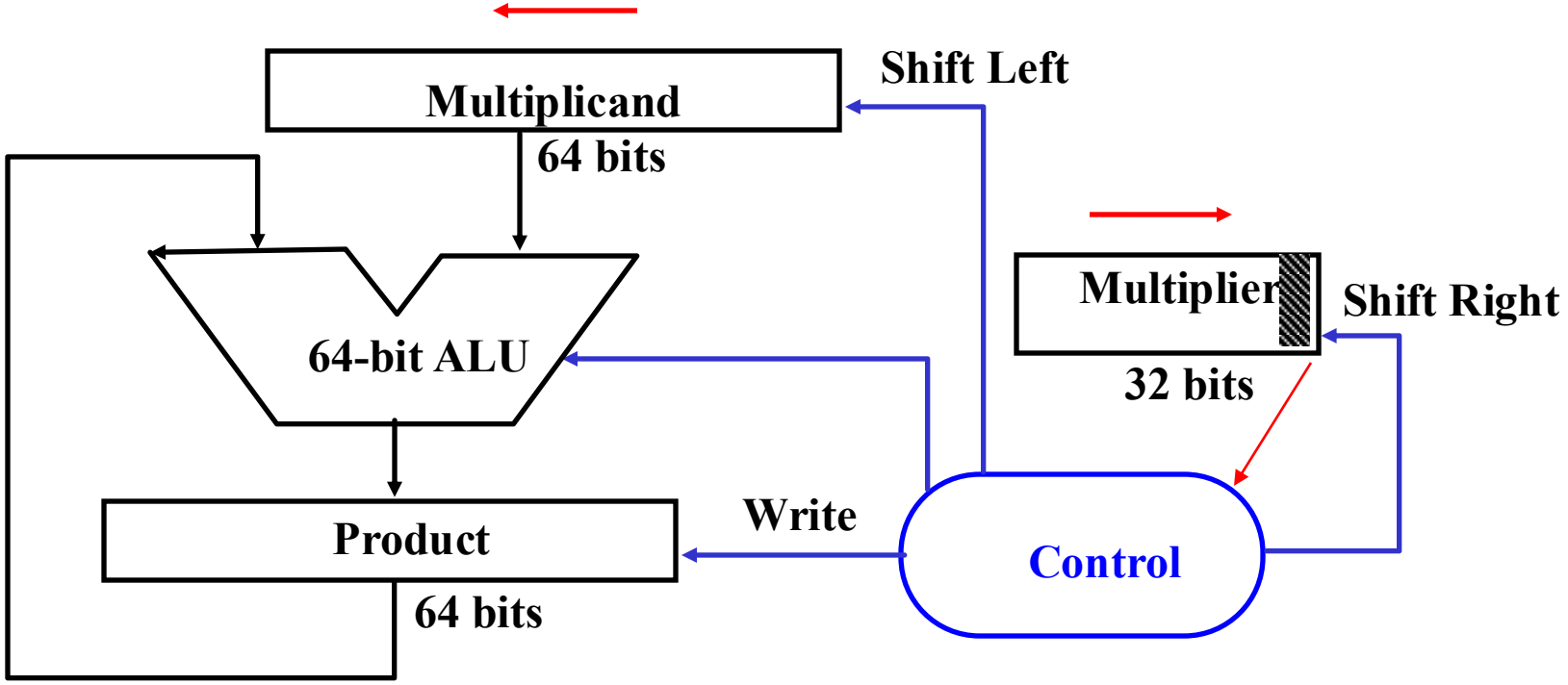
Unsigned Combinational Multiplier



- Stage i accumulates $A * 2^i$ if $B_i == 1$

Multiply Hardware (version 1)

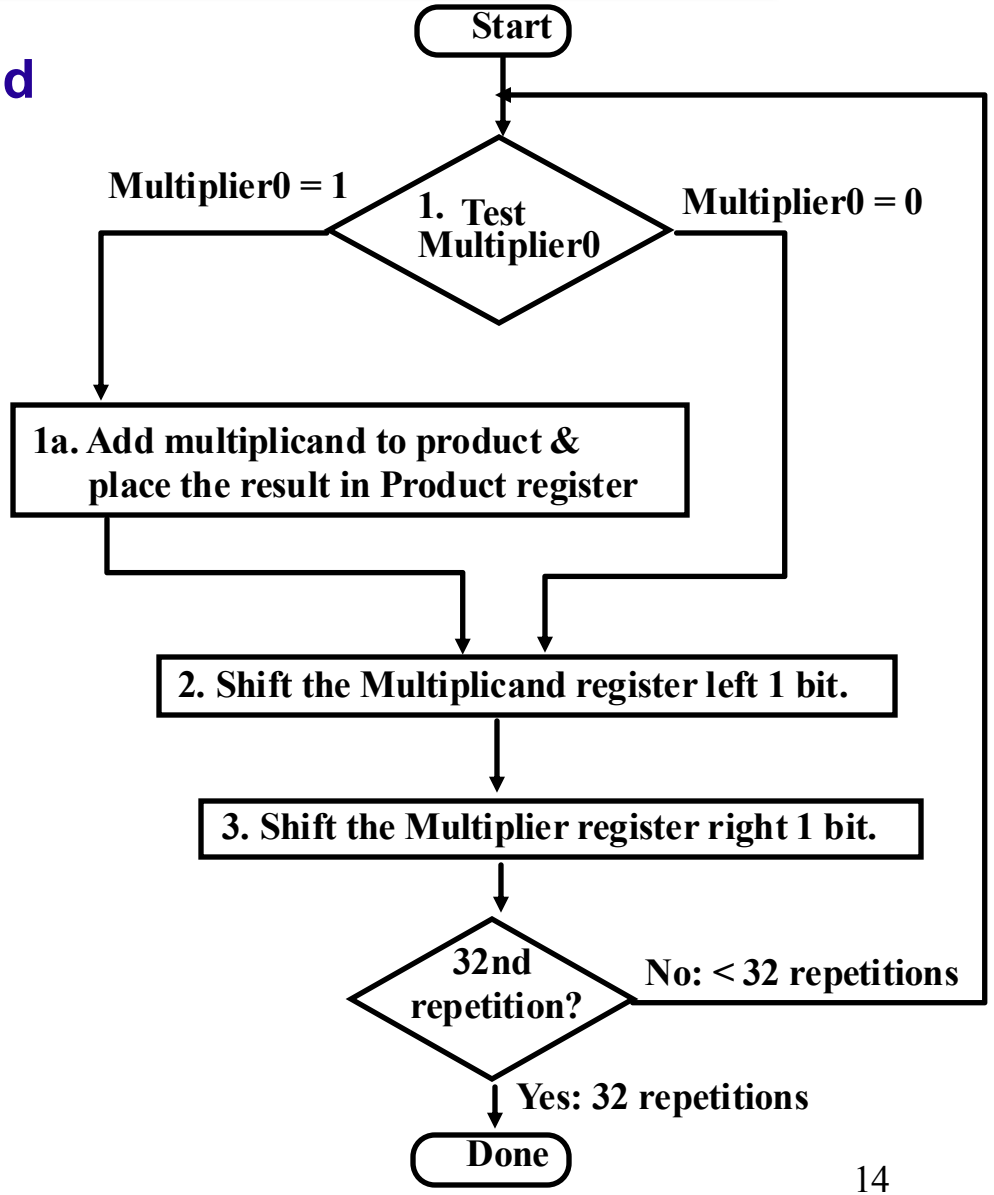
- Shift-add multiply
- 64-bit Multiplicand reg, 64-bit ALU, 64-bit Product reg, 32-bit multiplier reg



Multiply Algorithm Version 1

	Product	Multiplier	Multiplicand
0.	0000 0000	0011	0000 0010
1.	0000 0010	0001	0000 0100
2.	0000 0110	0000	0000 1000
3.	0000 0110	0000	0001 0000
4.	0000 0110	0000	0010 0000

0000 0110



Observations on Multiply Version 1

- 1 clock cycle per step => 96 clocks per multiply
- 1/2 bits in multiplicand always 0
  => 64-bit adder is wasted
- 0 is inserted when shift the multiplicand left
  => least significant bits of product never changed once formed