TUTORIAL 9 COMPUTER ARITHMETIC

Overview

- We will review/learn the following concepts in this tutorial:

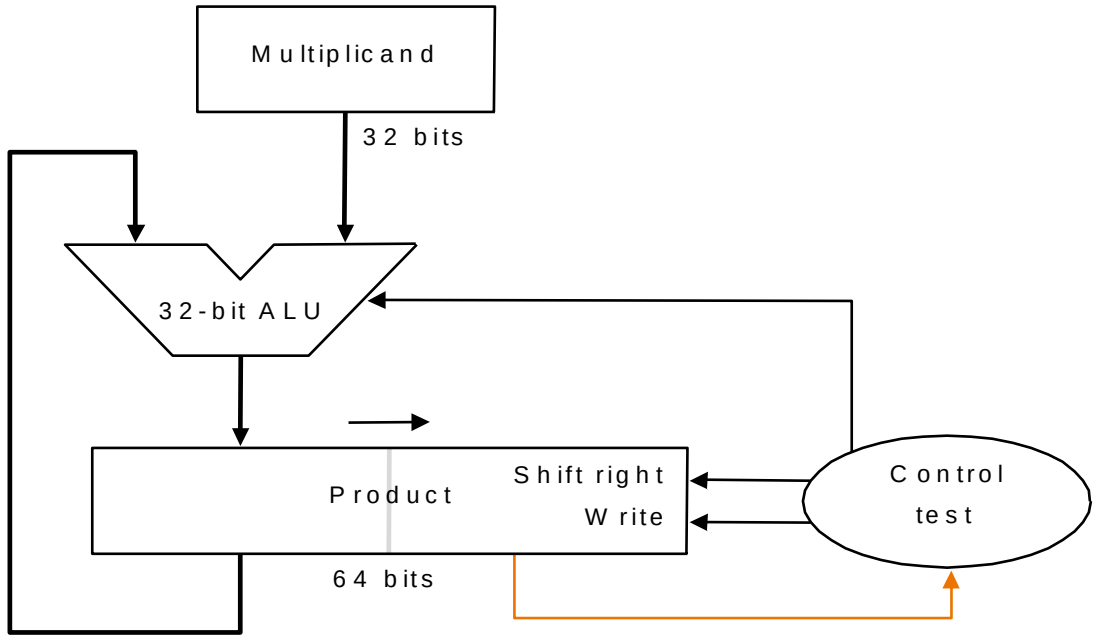
- Unsigned multiplication (optimized version)
- Unsigned division (version 1 -> optimized version)
- Booth's algorithm for signed multiplication (optional)

Multiplication Hardware: Optimized Version

- **32-bit ALU**

- **Two registers**
  - Multiplicand register: 32 bits
  - Product register: 64 bits (right half also used for storing multiplier)

- **Operations:**
  - The right half of the product register is initialized to the multiplier, and its left half is initialized to 0
  - The two right-shifts at each step for version 2 are combined into only a single right-shift because the product and multiplier registers have been combined



Example

- Multiplication of two 4-bit **unsigned** numbers 0101 and 0110

Multiplier

Iteration	Multiplicand (M)	Product (P)	Remark
0	0110	0000 0101	Initial state
1		0110 0101	Left(P) = Left(P) + M
		0011 0010	P = P >> 1
2		0011 0010	No operation
		0001 1001	P = P >> 1
3		0111 1001	Left(P) = Left(P) + M
		0011 1100	P = P >> 1
4		0011 1100	No operation
		0001 1110	P = P >> 1

Exercise 1

- Do unsigned multiplication 5 x 7 (0101 and 0111) with optimized hardware, and fill in the table below.

Iteration	Multiplicand (M)	Product (P)	Remark
0	0101		Initial state
1			
2			
3			
4			

Exercise 2

- **Write down the sequence of MIPS instructions for the following C++ code, assuming variable a, b are stored in $s0 and $s1 respectively (you may assume there is no overflow condition)**
  - b = a * 5;

Division

- Division is the reciprocal operation of multiplication

- **Paper-and-pencil example** (1001010_{ten} / 1000_{ten}):

```
                        1001      Quotient
Divisor  1000    / 1001010        Dividend
                 -1000000
                   0001010
                   0001010
                   0001010
                    -1000
                        10       Remainder
```

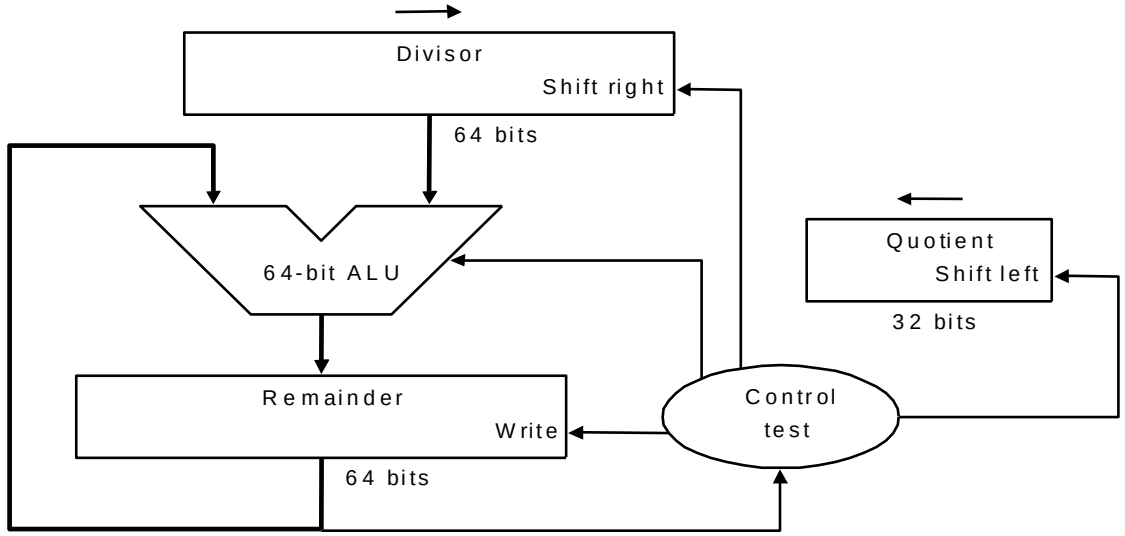
Dividend = Quotient x Divisor + Remainder

Division in Binary

- **Paper-and-pencil 4-bit example (0111_2 / 0010_2):**

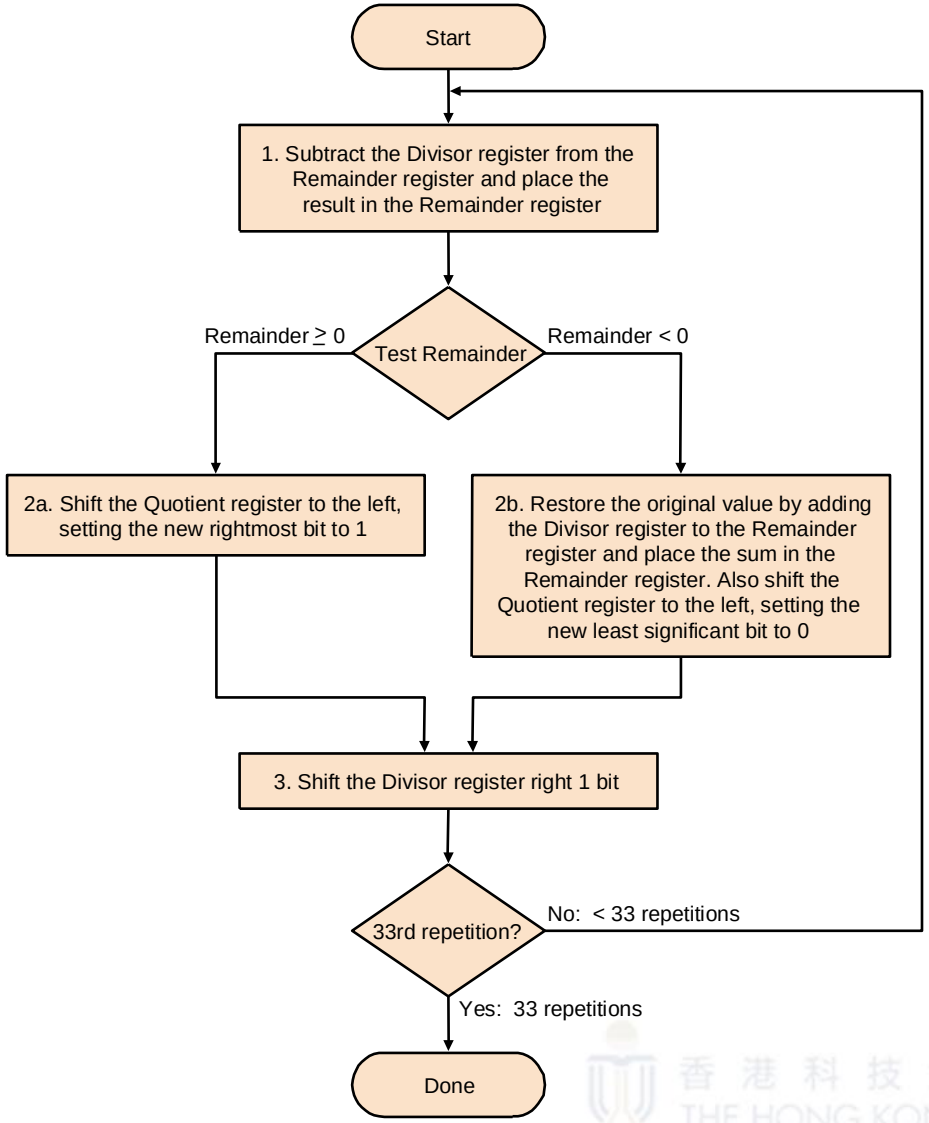
```
                      00011      Quotient
Divisor  0010    /00000111      Dividend
                -00100000
                 -0010000
                  -001000
                   -00100
                    00011
                   -00010
                    00001      Remainder
```

Division Sequential Hardware - Version 1



- **64-bit ALU**
- **Divisor register: 64 bits, Quotient register: 32 bits, Remainder register: 64 bits**
- **Operations:**
  - 32-bit divisor starts in the left half of divisor register; is shifted right 1 bit at each step
  - Quotient register is initialized to 0; shifted left 1 bit at each step
  - Remainder register is initialized with the dividend
  - Control decides
    - when to shift the divisor and quotient registers
    - when to write new values into the remainder register

Algorithm - Version 1



Observations Version 1

Similar to the first version of the multiplication hardware

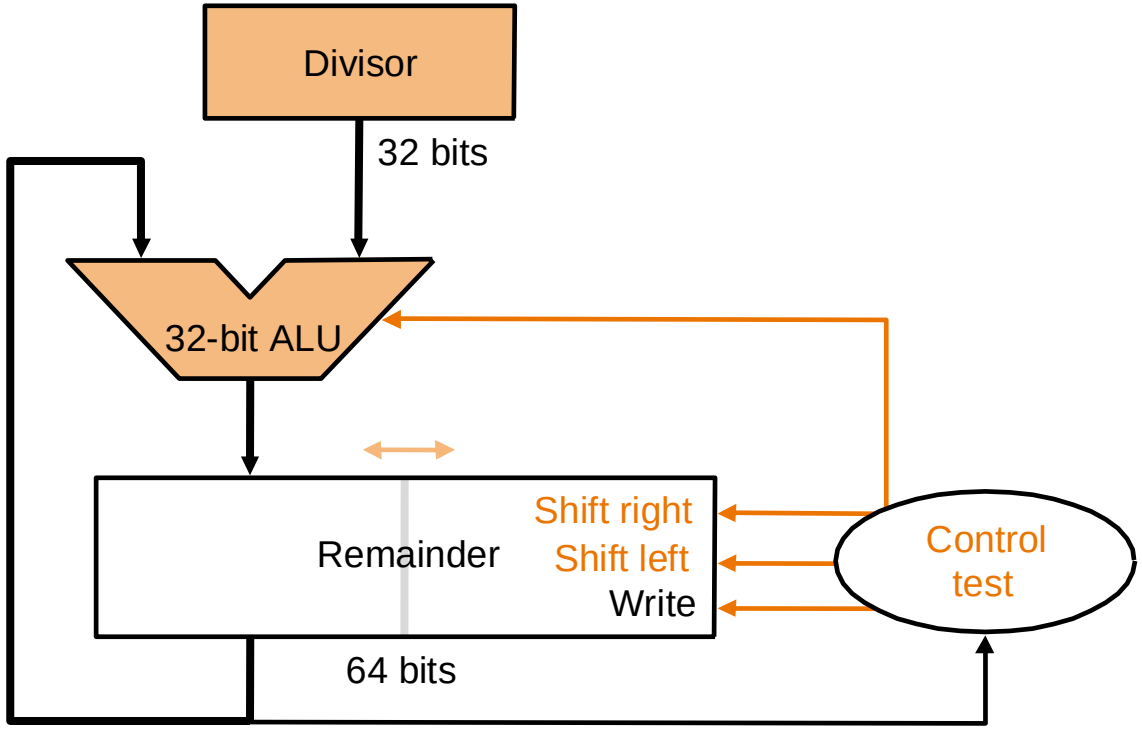
- **At most half of the divisor register has useful information**
  - Both the divisor register and ALU could potentially be cut in half
- **Shift divisor register to right =>Shift remainder register to left**
  - Produce the same alignment
  - But, simplify hardware necessary for the ALU and divisor register
- **Combine the remainder and quotient registers**

Example Optimized Division

Paper-and-pencil example (0111_2 / 0010_2):

```
                    0 0011      Quotient
Divisor  0010    | 00000111     Dividend
                -0010
                  00001110
                -0010
                  00011100
                -0010
                  00111000
                -0010
                  00011000
                  00110000
                -0010
                  0001          Remainder
```

Division Hardware Optimized Version



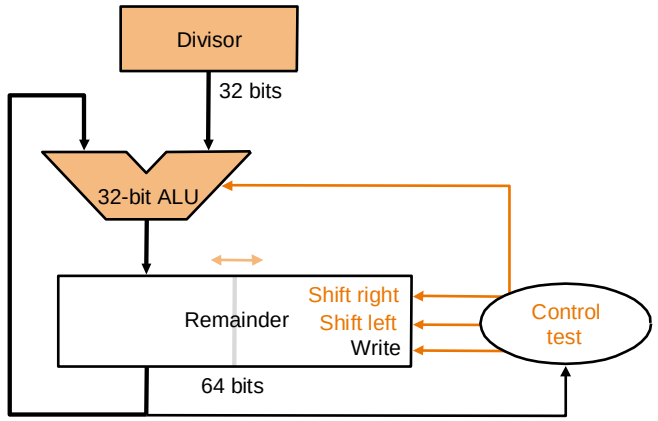
(changes made to previous version are highlighted in orange color)

Division Hardware Optimized Version (cont.)

- **32-bit ALU**

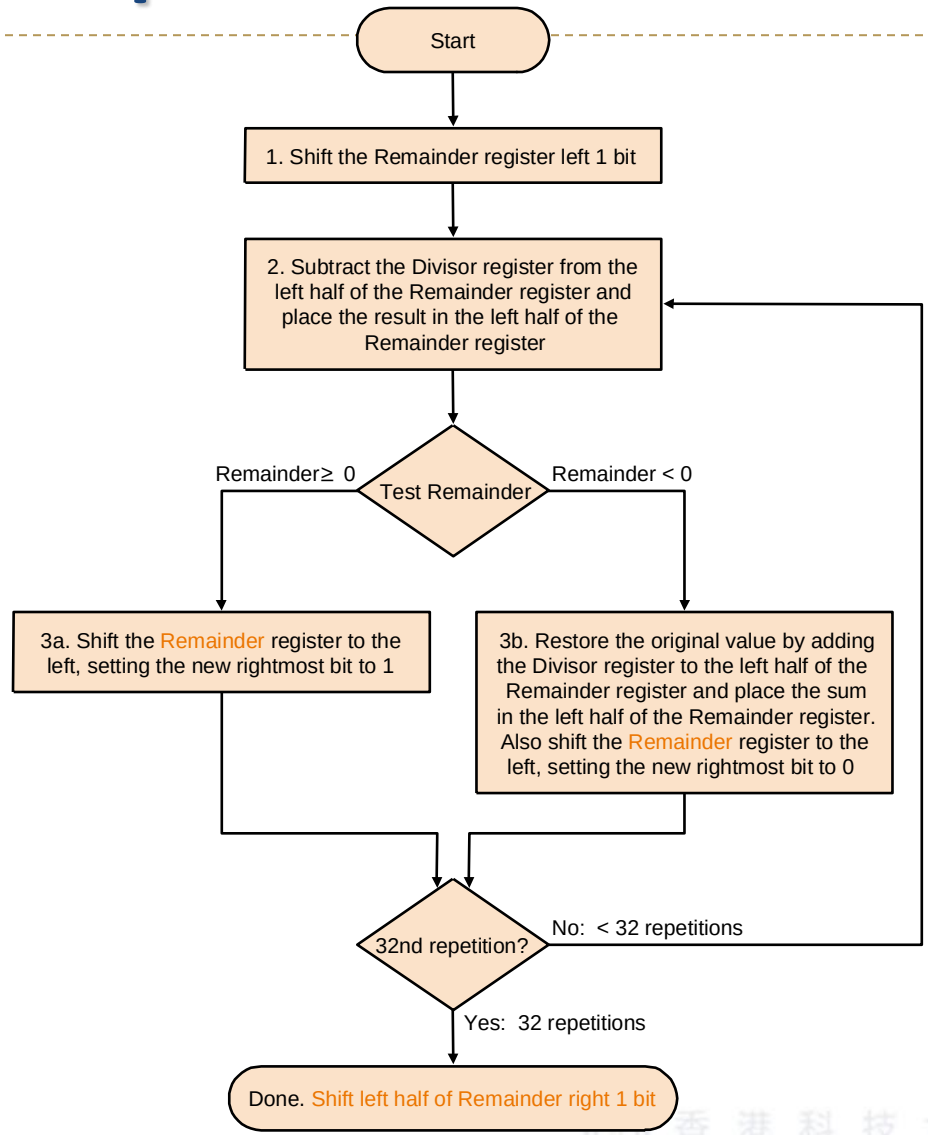
- **Two registers:**
  - **Divisor register: 32 bits**
  - **Remainder register: 64 bits**

  (right half also used for storing quotient)



- **Operations:**
  - 32-bit divisor is always subtracted from the left half of remainder register
    - The result is written back to the left half of the remainder register
  - The right half of the remainder register is initialized with the dividend
    - Left shift remainder register by one before starting
  - The new order of the operations in the loop is that the remainder register will be **shifted left one time too many**
    - Thus, **final correction step:** must **right shift back only the remainder** in the left half of the remainder register

Division Algorithm Optimized Version



Example 1

- Division of a 4-bit unsigned number (0111) by another one (0011)

Iteration	Divisor (D)	Remainder (R)	Remark
0	0011	0000 0111 0000 1110	Initial state R = R << 1
1		1101 1110 0000 1110 0001 1100	Left(R) = Left(R) – D Undo R = R << 1, R_0 = 0
2		1110 1100 0001 1100 0011 1000	Left(R) = Left(R) – D Undo R = R << 1, R_0 = 0
3		0000 1000 0001 0001	Left(R) = Left(R) – D R = R << 1, R_0 = 1
4		1110 0001 0001 0001 0010 0010	Left(R) = Left(R) – D Undo R = R << 1, R_0 = 0
extra		0001 0010	Left(R) = Left(R) >> 1

Remainder **Quotient**

correction

Example 2

- Division of a 4-bit unsigned number (1011) by another one (0011)

Iteration	Divisor (D)	Remainder (R)	Remark
0	0011	0000 1011 0001 0110	Initial state R = R << 1
1		1110 0110 0001 0110 0010 1100	Left(R) = Left(R) – D Undo (L(R) = L (R) + D) R = R << 1, R_0 = 0
2		1111 1100 0010 1100 0101 1000	Left(R) = Left(R) – D Undo R = R << 1, R_0 = 0
3		0010 1000 0101 0001	Left(R) = Left(R) – D R = R << 1, R_0 = 1
4		0010 0001 0100 0011	Left(R) = Left(R) – D R = R << 1, R_0 = 1
extra		0010 0011	Left(R) = Left(R) >> 1

Remainder correction Quotient

Exercise 3

- Divide unsigned division 8 (1000) by 3 (0011) with optimized hardware, fill in the table below.

Iteration	Divisor (D)	Remainder (R)	Remark
0	0011		Initial state
1			
2			
3			
4			
extra			

Exercise 4

- **Write down the sequence of MIPS instructions for the following C++ code, assuming variable a, b are stored in $s0 and $s1 respectively**
  - b = a / 3;

Signed Multiplication: Booth's Algorithm

- Let's consider multiplying 0010_2 and 0110_2

	Convention		Booth	
Multiplicand	0010		0010	
Multiplier	x 0110		0110	
	+ 0000	+	0000	shift
	+ 0010	-	0010	subtract
	+ 0010	+	0000	shift
	+ 0000	+	0010	add
Product	= 0001100	=	0001100	

Idea of Booth's Algorithm

- Looks at two bits of multiplier at a time from right to left
  - 00: No-op; 01: Addition; 10: Subtraction; 11: No-op.
- Assume that shifts are much faster than adds
- Basic idea to speed up the calculation: avoid unnecessary additions

Booth's Algorithm Example

- **Multiply 14 times -5 using 5-bit numbers (10-bit result).**
  - 14 in binary: 01110; -14 in binary: 10010 (so we can add when we need to subtract the multiplicand); -5 in binary: 11011
  - Expected result: -70 in binary: 11101 11010

Step	Multiplicand	Action	Multiplier upper 5-bits 0, lower 5-bits multiplier, 1 "Booth bit" initially 0
0	01110	Initialization	00000 11011 0
1	01110	10: Subtract Multiplicand	00000+10010=10010 10010 11011 0
		Shift Right Arithmetic	11001 01101 1
2	01110	11: No-op	11001 01101 1
		Shift Right Arithmetic	11100 10110 1

Booth's Algorithm Example (con't)

3	01110	01: Add Multiplicand	11100+01110=01010 (Carry ignored because adding a positive and negative number cannot overflow.) 01010 10110 1
		Shift Right Arithmetic	00101 01011 0
4	01110	10: Subtract Multiplicand	00101+10010=10111 10111 01011 0
		Shift Right Arithmetic	11011 10101 1
5	01110	11: No-op	11011 10101 1
		Shift Right Arithmetic	11101 11010 1

Exercise 5

- Do signed multiplication of two signed number +2 and -3 (0010 and 1101) with Booth's algorithm, fill in the table below.

Iteration	Multiplicand (M)	Product (P)	Remark
0	0010		Initial state
1			
2			
3			
4			