CS/ECE 3810: Computer Organization

Lecture 5: Arithmetic for Computers

Anton Burtsev
September, 2022

Arithmetic for Computers

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

Integer addition

- Example 7 + 6

$$
\begin{array}{rr}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\
+ & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\
\hline
\end{array}
$$

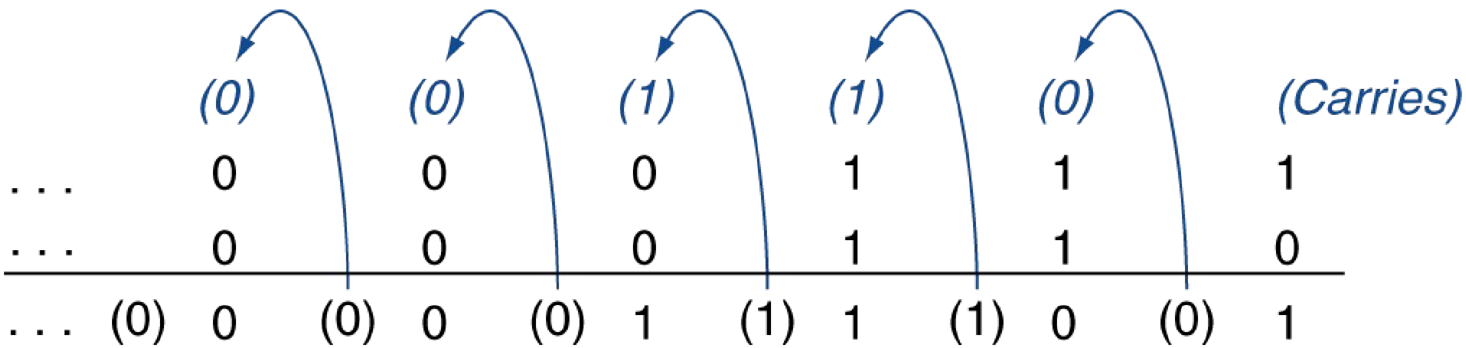
Integer addition

- Example 7 + 6

$$
\begin{array}{ll}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\
+ & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101_{two} = 13_{ten}
\end{array}
$$

Integer Addition

- Example: 7 + 6



Integer Addition

- Example: 7 + 6

	(0)	*(0)*	*(1)*	*(1)*	*(0)*	*(Carries)*
. . .	0	0	0	1	1	1
. . .	0	0	0	1	1	0
. . .	(0) 0	(0) 0	(0) 1	(1) 1	(1) 0	(0) 1

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

Integer subtraction

- Example 7 – 6
  - Directly:

$$
\begin{array}{lr}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\
- & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}
\end{array}
$$

  - Or as adding (-6)

$$
\begin{array}{lr}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\
+ & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010_{two} = -6_{ten} \\
\hline
= & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}
\end{array}
$$

Integer Subtraction

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

+7:	0000 0000 … 0000 0111
<u>–6:</u>	<u>1111 1111 … 1111 1010</u>
+1:	0000 0000 … 0000 0001

- Overflow if result out of range
  - Subtracting two +(positive) or two –negative operands, no overflow
  - Subtracting +(positive) from –(negative) operand
    - Overflow if result sign is 0
  - Subtracting –(negative) from +(positive) operand
    - 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
  - 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

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

- Start with long-multiplication approach

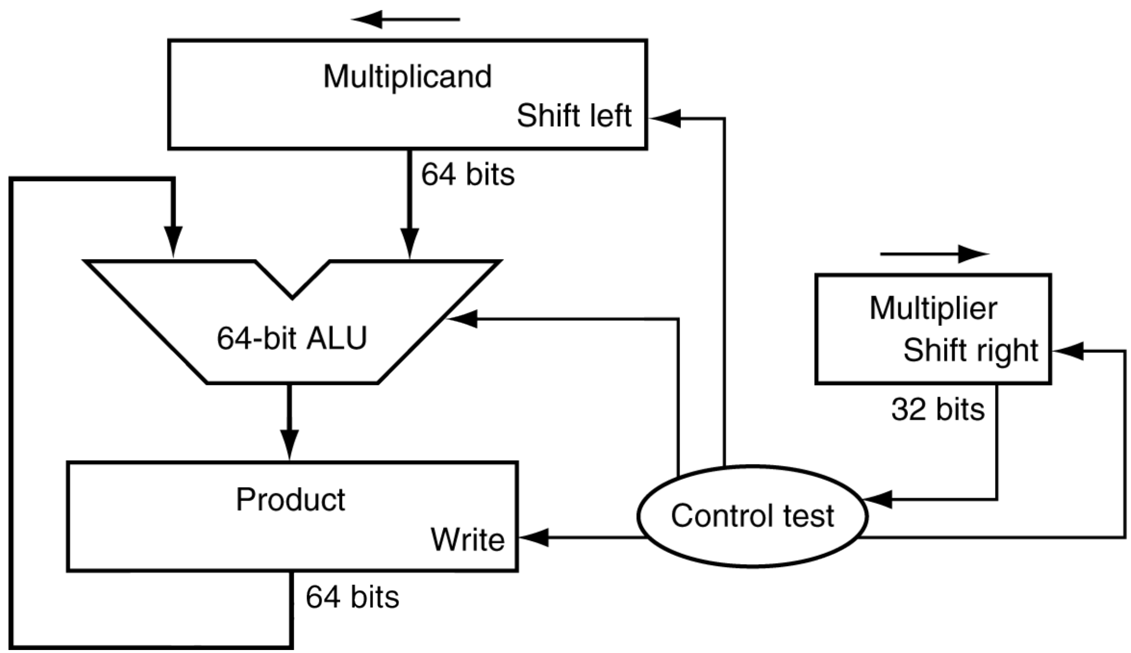
multiplicand

multiplier

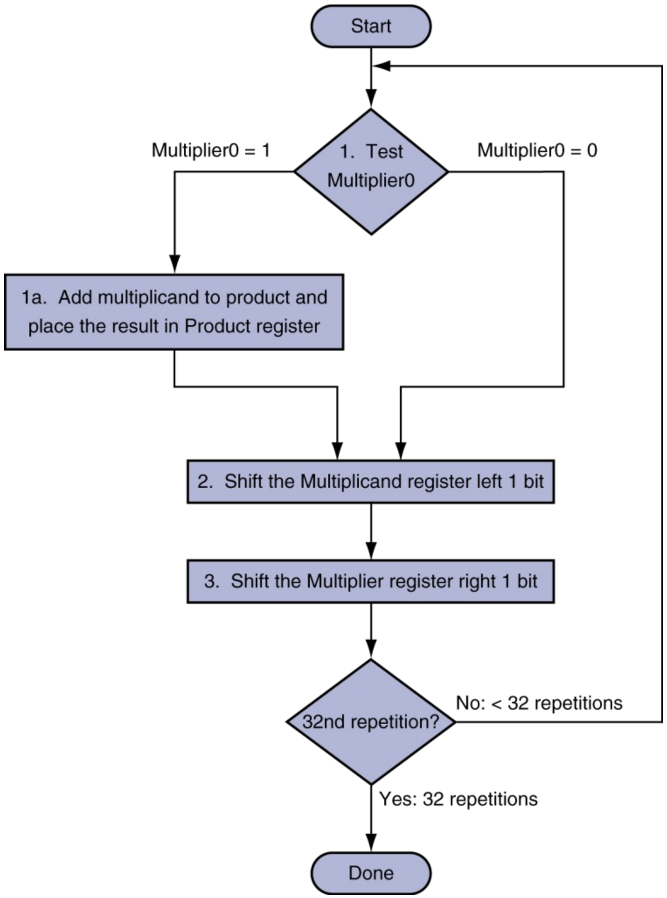
product

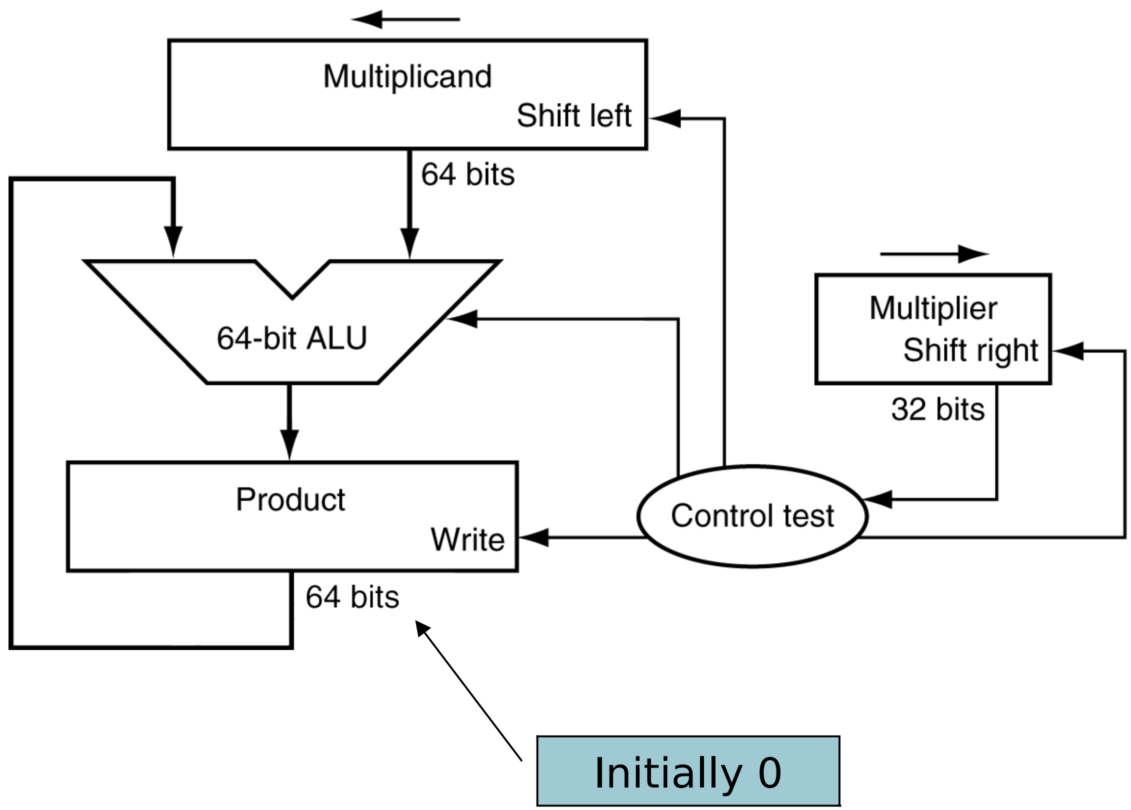
```
       1000
×      1001
-----------
       1000
      0000
     0000
    1000
-----------
    1001000
```

Length of product is the sum of operand lengths



Multiplication Hardware





Can we do it faster?

- Start with long-multiplication approach

multiplicand

multiplier

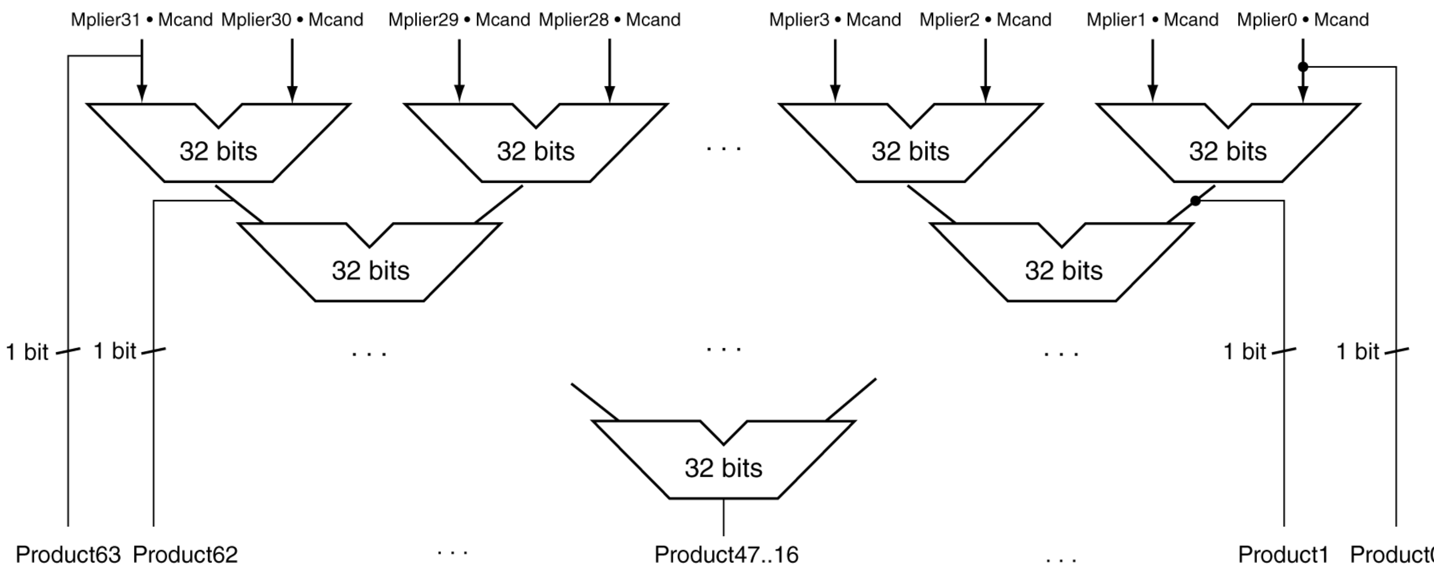
product

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

Length of product is the sum of operand lengths

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

Division

Division

			$\underline{1001_{ten}}$	Quotient
Divisor	1000_{ten}	|	1001010_{ten}	Dividend
			$\underline{-1000}$	
			10	
			101	
			1010	
			$\underline{-1000}$	
			10_{ten}	Remainder

At every step,

- shift divisor right and compare it with current dividend
- if divisor is larger, shift 0 as the next bit of the quotient
- if divisor is smaller, subtract to get new dividend and shift 1 as the next bit of the quotient

Division

1001$_{ten}$ Quotient

Divisor 1000$_{ten}$ | 1001010$_{ten}$ Dividend

0001001010	0001001010	0000001010	0000001010
100000000000	0001000000	0000100000	0000001000
Quo: 0	000001	0000010	000001001

At every step,

- shift divisor right and compare it with current dividend
- if divisor is larger, shift 0 as the next bit of the quotient
- if divisor is smaller, subtract to get new dividend and shift 1 as the next bit of the quotient

Divide Example

- Divide 7_{ten} ($0000\ 0111_{two}$) by 2_{ten} (0010_{two})

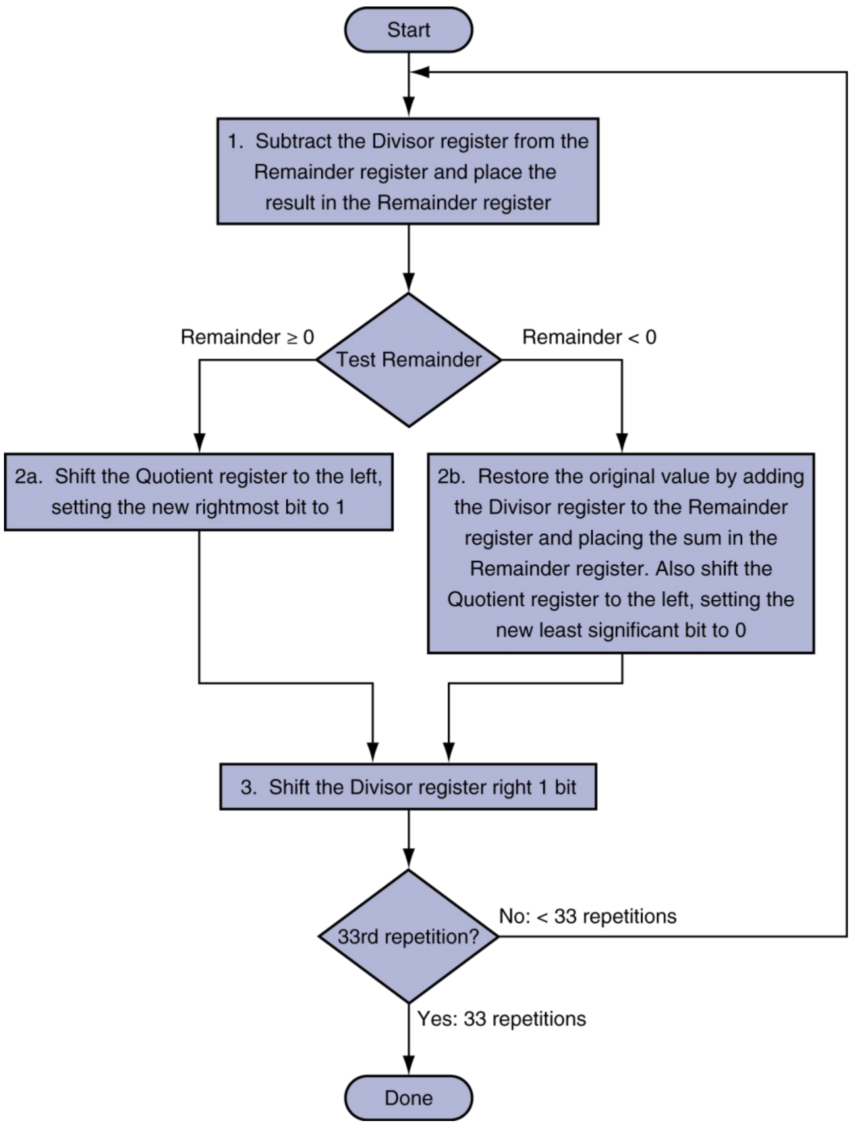
Iter	Step	Quot	Divisor	Remainder
0	Initial values			
1				
2				
3				
4				
5				

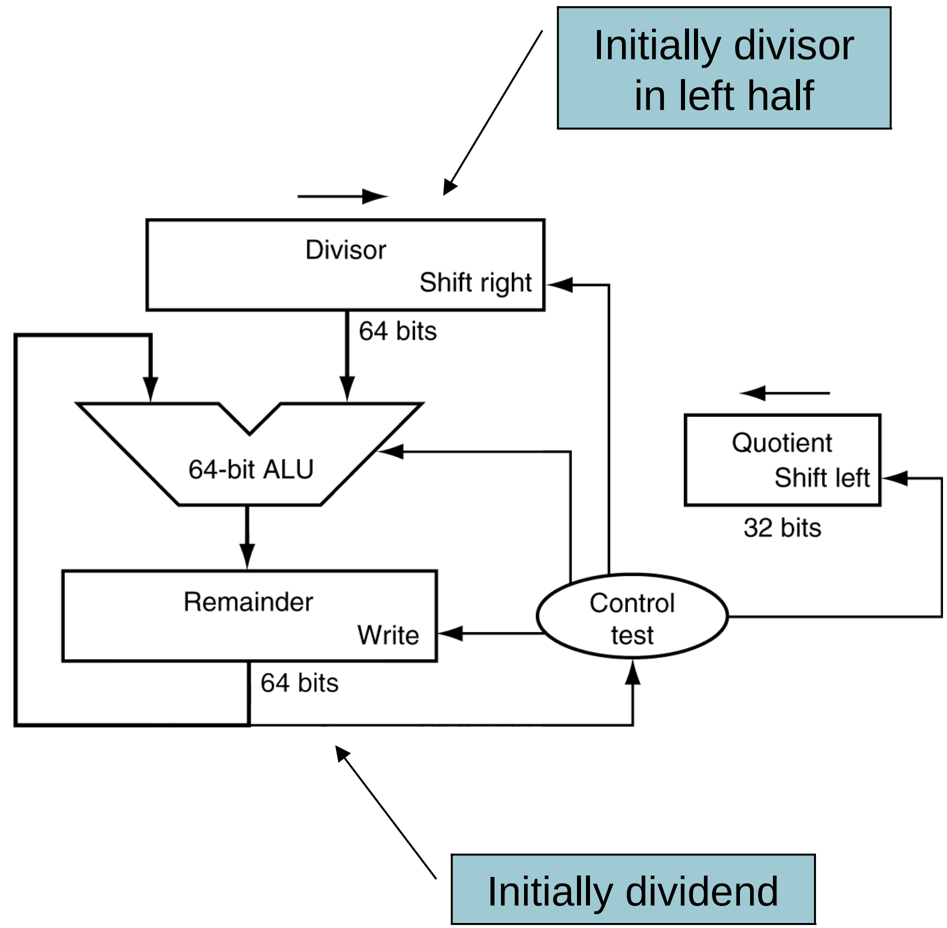
Divide Example

- Divide 7_{ten} ($0000\ 0111_{two}$) by 2_{ten} (0010_{two})

Iter	Step	Quot	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	Rem = Rem – Div Rem < 0 ⇒ +Div, shift 0 into Q Shift Div right	0000 0000 0000	0010 0000 0010 0000 0001 0000	1110 0111 0000 0111 0000 0111
2	Same steps as 1	0000 0000 0000	0001 0000 0001 0000 0000 1000	1111 0111 0000 0111 0000 0111
3	Same steps as 1	0000	0000 0100	0000 0111
4	Rem = Rem – Div Rem >= 0 ⇒ shift 1 into Q Shift Div right	0000 0001 0001	0000 0100 0000 0100 0000 0010	0000 0011 0000 0011 0000 0011
5	Same steps as 4	0011	0000 0001	0000 0001

Division Hardware





MIPS Division

- Use HI/LO registers for result
  - HI: 32-bit remainder
  - LO: 32-bit quotient
- Instructions
  - `div rs, rt  /  divu rs, rt`
  - No overflow or divide-by-0 checking
    - Software must perform checks if required
  - Use `mfhi`, `mflo` to access result

Thank you!