5118007-02 Computer Architecture

Ch. 3 Arithmetic for Computers

9 Apr 2024

Shin Hong

Integer Arithmetic

- addition and subtraction
  - dealing with overflow
- multiplication
- division

Addition

- Digits are added bit-by-bit from right to left, considering carries passed from the right

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

Subtraction

- A subtraction is basically the same as a combination of negation and addition

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

Overflow (1/2)

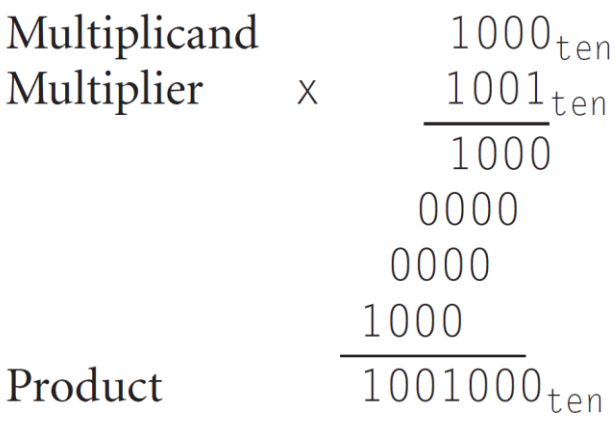
- An overflow occurs when the resulting value cannot be represented within given bits
  - only when adding two numbers of the same sign, and not possible when two numbers of different signs are added

- Overflow for signed numbers
  - adding two positive numbers yields a negative number
  - subtracting two negative numbers yields a positive number
  - the occurrence of overflow with `add`, `addi`, `sub` instructions causes an exception

Overflow (2/2)

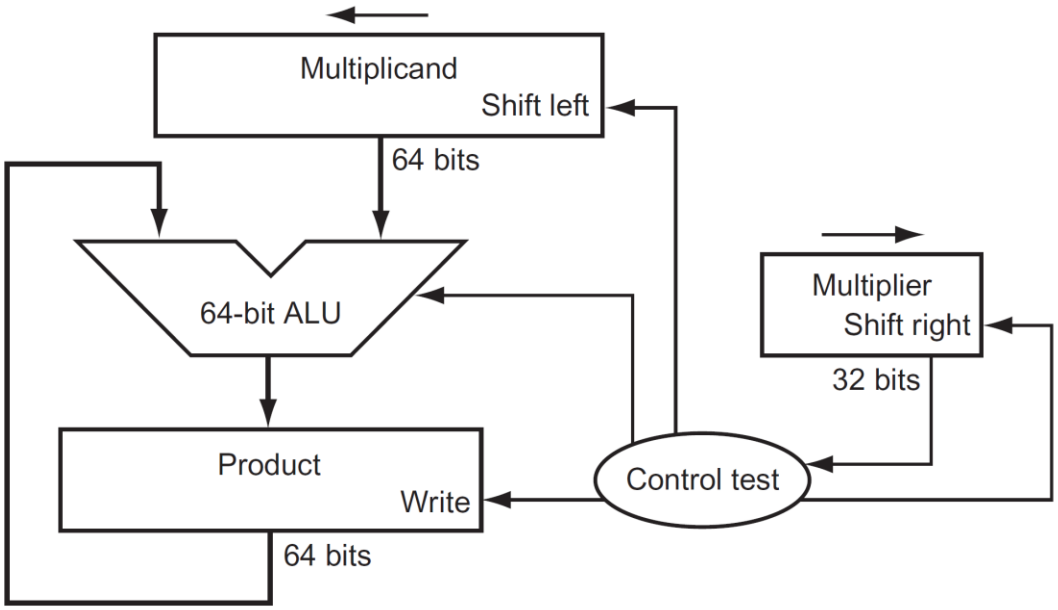
- Overflow for signed numbers
  - addition of two numbers yields a smaller number
  - subtraction of two numbers yields a greater number
  - overflow with `addu`, `addiu`, `subu` do not cause exception, thus C compiler uses these instructions for arithmetics.

Multiplication



- Basically, multiplication can be implemented as a sequential operations of right-shifts, left-shifts and additions
- Multiplication of a *n*-bits multiplicand and a *m*-bits multiplier produces a *n*+*m* product
  - overflow may occur

Sequential Multiplication Hardware



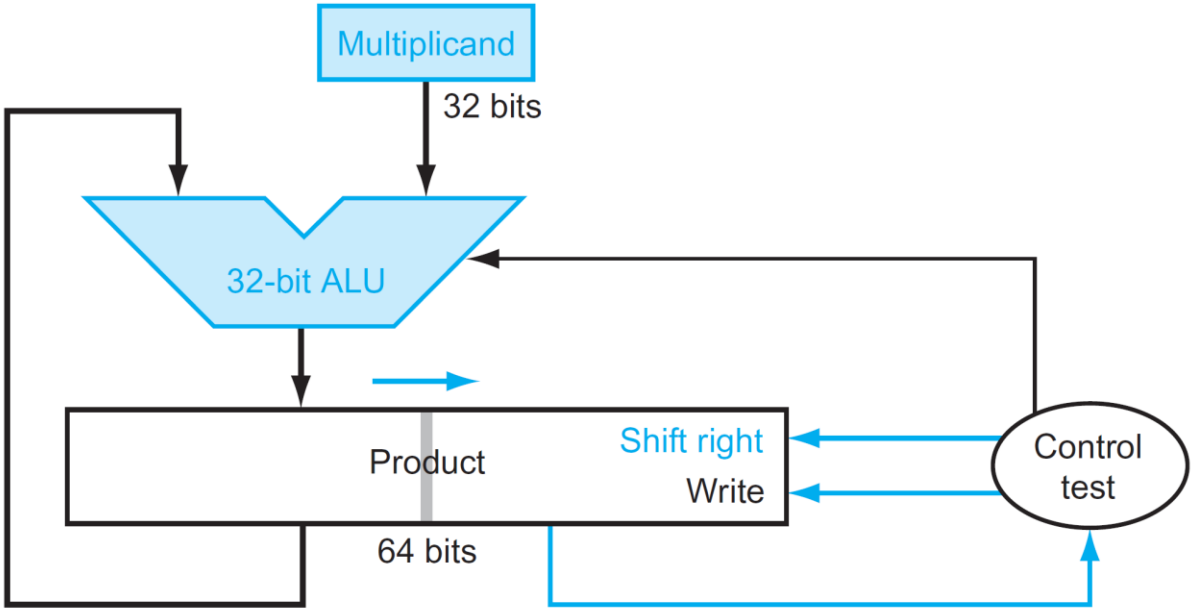

```
foreach bit of multiplier begin
   if the LSB of multiplier is 1 then
      add multiplicand and product;
   end-if
   shift-left the multiplicand ;
   shift-right a multiplier ;
end-for
```

Example

- Multiply $0010_{(2)}$ and $0011_{(2)}$

Iteration	Step	Multiplier	Multiplicand	Product
0	Initial values	0011	0000 0010	0000 0000
1	1a: 1 ⟹ Prod = Prod + Mcand	0011	0000 0010	0000 0010
	2: Shift left Multiplicand	0011	0000 0100	0000 0010
	3: Shift right Multiplier	0001	0000 0100	0000 0010
2	1a: 1 ⟹ Prod = Prod + Mcand	0001	0000 0100	0000 0110
	2: Shift left Multiplicand	0001	0000 1000	0000 0110
	3: Shift right Multiplier	0000	0000 1000	0000 0110
3	1: 0 ⟹ No operation	0000	0000 1000	0000 0110
	2: Shift left Multiplicand	0000	0001 0000	0000 0110
	3: Shift right Multiplier	0000	0001 0000	0000 0110
4	1: 0 ⟹ No operation	0000	0001 0000	0000 0110
	2: Shift left Multiplicand	0000	0010 0000	0000 0110
	3: Shift right Multiplier	0000	0010 0000	0000 0110

Refined Version



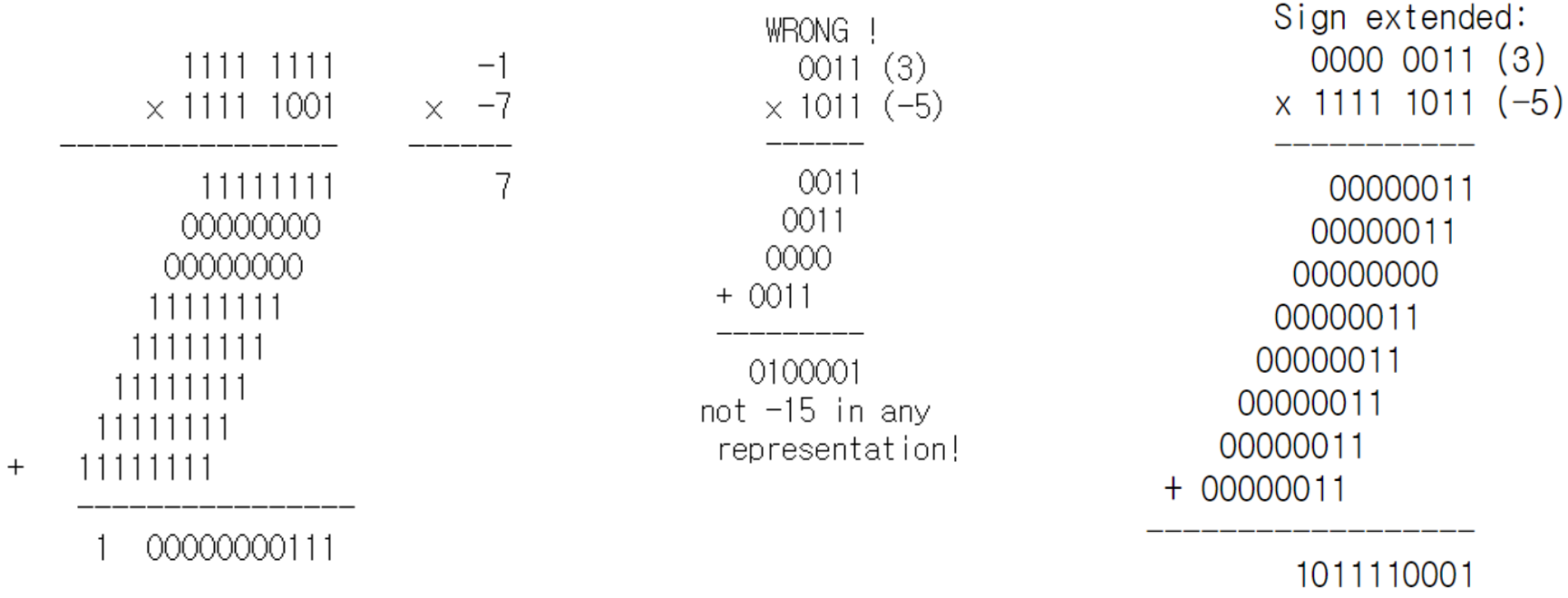
Signed Multiplication

- Naive approach
  - store the signs of multiplicand and multiplier respectively
  - convert them to positive numbers
  - perform multiplication
  - apply the proper sign to the product

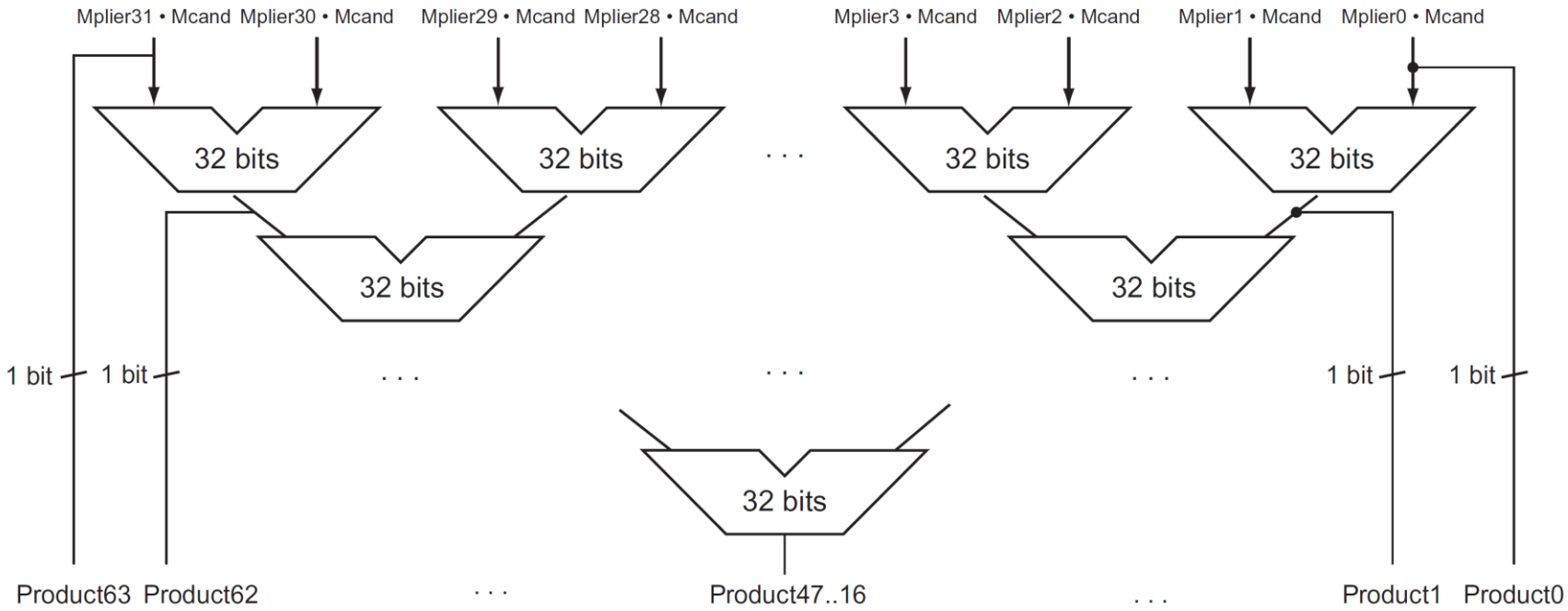
- Interestingly, the aforementioned multiplication algorithm works sound for two's compelemtns
  - https://pages.cs.wisc.edu/~markhill/cs354/Fall2008/beyond354/int.mult.html

No Thinking Method

- sign-extend to both multiplicand and multiplier before multiplication



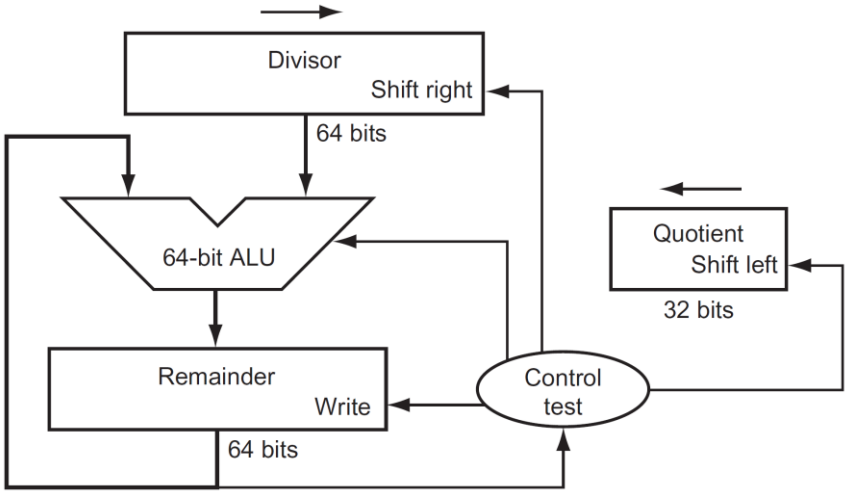
Fast Multiplication with Large Circuits



Division

- Division can be implemented as a series of shift-right, shift-left and subtraction

$$
\begin{array}{rl}
 & 1001_{ten} \quad \text{Quotient} \\
\text{Divisor } 1000_{ten} & \overline{)\,1001010_{ten}} \quad \text{Dividend} \\
 & \underline{-1000} \\
 & \quad 10 \\
 & \quad 101 \\
 & \quad 1010 \\
 & \underline{-1000} \\
 & \quad 10_{ten} \quad \text{Remainder}
\end{array}
$$



Example: $0111_{(2)} / 0010_{(2)}$

Iteration	Step	Quotient	Divisor	Remainder
0	Initial values	0000	0010 0000	0000 0111
1	1: Rem = Rem – Div	0000	0010 0000	①110 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0010 0000	0000 0111
	3: Shift Div right	0000	0001 0000	0000 0111
2	1: Rem = Rem – Div	0000	0001 0000	①111 0111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0001 0000	0000 0111
	3: Shift Div right	0000	0000 1000	0000 0111
3	1: Rem = Rem – Div	0000	0000 1000	①111 1111
	2b: Rem < 0 ⟹ +Div, sll Q, Q0 = 0	0000	0000 1000	0000 0111
	3: Shift Div right	0000	0000 0100	0000 0111
4	1: Rem = Rem – Div	0000	0000 0100	⓪000 0011
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0001	0000 0100	0000 0011
	3: Shift Div right	0001	0000 0010	0000 0011
5	1: Rem = Rem – Div	0001	0000 0010	⓪000 0001
	2a: Rem ≥ 0 ⟹ sll Q, Q0 = 1	0011	0000 0010	0000 0001
	3: Shift Div right	0011	0000 0001	0000 0001

Faster Implementation