Advanced Application Programming

(084490, Fall 2020)

Prof. Dong-Chan Kim

Department of Information Security, Cryptology, and Mathematics

Kookmin University

dckim@kookmin.ac.kr

Updated: November 2, 2020

Contents

Chapter 1

Big Integer Representation

1.1 Symbols

`0b`	prefix for binary representation, e.g., `0b10001101`
`0x`	prefix for hexadecimal representation, e.g., `0xaabbccdd`
$\mathbb{Z}$	set of integers
w	bit length of a word ($w \in \{8, 32, 64\}$)
W	integer 2^w
$\lfloor x \rfloor$	floor function ($= \max\{a \in \mathbb{Z} : a \le x\}$)
$\lceil x \rceil$	ceil function ($= \min\{a \in \mathbb{Z} : a \ge x\}$), ($\because \lfloor -x \rfloor = -\lceil x \rceil$)
$[a, b)$	$\{x \in \mathbb{Z} : a \le x < b\}$
$\mathsf{BitLen}(x)$	minimum bit length to represent a non-negative integer x ($= \lfloor \log_2 x \rfloor + 1$), $\mathsf{BitLen}(0) = 1$
$\mathsf{WordLen}_w(x)$	minimum w-bit word length to represent a non-negative integer x ($= \lceil \mathsf{BitLen}(x)/w \rceil$)
$\mathsf{Sign}(x)$	sign of an integer x; $\mathsf{Sign}(x) := \begin{cases} 0, & x \ge 0, \\ 1, & x < 0. \end{cases}$

Every integer $A \in \mathbb{Z}$ can be represented by $A = (-1)^b \sum_{j=0}^{n-1} A_j W^j$ ($A_j \in [0, W)$) for some $n \in \mathbb{Z}_{>0}$ and $b \in \{0, 1\}$.

1.2 Bit Arithmetic

$a \gg r$ r-bit right shift of a bit string a

$a \ll r$ r-bit left shift of a bit string a

$\neg a$ 비트열 a의 부정(NOT)

$a \wedge b$ 길이가 같은 두 비트열 a와 b의 논리곱(AND)

$a \vee b$ 길이가 같은 두 비트열 a와 b의 논리합(OR)

$a \oplus b$ 길이가 같은 두 비트열 a와 b의 배타적논리합(eXclusive-OR)

a	b	$\neg a$ (NOT)	$a \wedge b$ (AND)	$a \vee b$ (OR)	$a \oplus b$ (XOR)
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	0	1	1
1	1	0	1	1	0

1.3 Data Representation

1.3.1 Bit Index

n개의 비트로 구성된 n-비트열 $a = (a_j)_{j=0}^{n-1}$ $(a_j \in \{0, 1\})$은 다음과 같이 빅엔디안(big endian) 방식으로 표기한다.

$$a = (a_{n-1}, a_{n-2}, \ldots, a_0)$$

1.3.2 Binary and Hexadecimal Representation

4비트열은 다음과 같이 2진수 또는 16진수로 표기한다.

bit	bin.	hexa.	bit	bin.	hexa.	bit	bin.	hexa.	bit	bin.	hexa.
$(0,0,0,0)$	`0b0000`	`0x0`	$(0,1,0,0)$	`0b0100`	`0x4`	$(1,0,0,0)$	`0b1000`	`0x8`	$(1,1,0,0)$	`0b1100`	`0xc`
$(0,0,0,1)$	`0b0001`	`0x1`	$(0,1,0,1)$	`0b0101`	`0x5`	$(1,0,0,1)$	`0b1001`	`0x9`	$(1,1,0,1)$	`0b1101`	`0xd`
$(0,0,1,0)$	`0b0010`	`0x2`	$(0,1,1,0)$	`0b0110`	`0x6`	$(1,0,1,0)$	`0b1010`	`0xa`	$(1,1,1,0)$	`0b1110`	`0xe`
$(0,0,1,1)$	`0b0011`	`0x3`	$(0,1,1,1)$	`0b0111`	`0x7`	$(1,0,1,1)$	`0b1011`	`0xb`	$(1,1,1,1)$	`0b1111`	`0xf`

예) $(0,0,0,1,1,0,1,1) =$ `0b00011011` $=$ `0x1b`

1.3.3 Multi-Precision Integer

w비트 워드 A를 다음의 비트열 (a_j)로 정의한다. (내림차순 색인)

$$A = (a_j) = (a_{w-1}, a_{w-2}, \ldots, a_1, a_0), \quad a_j \in \{0, 1\}.$$

워드 A는 음이 아닌 정수 $\sum_{j=0}^{w-1} a_j 2^j \in [0, W)$로 간주한다.

$$A = (a_j) = (a_{w-1}, a_{w-2}, \ldots, a_1, a_0),\ a_j \in \{0, 1\} \quad \Leftrightarrow \quad A = \sum_{j=0}^{w-1} a_j 2^j,\ a_j \in \{0, 1\}.$$

Example 1.1. 8비트 워드 $A = (0, 0, 0, 1, 1, 0, 1, 1) = \texttt{0b00011011} \quad \Leftrightarrow \quad 2^4 + 2^3 + 2^1 + 2^0 = 27$.

워드열 $A = (A_j)_{j=0}^{t-1}$는 음이 아닌 정수 $\sum_{j=0}^{t-1} A_j W^j \in [0, W^t)$로 간주한다.

$$A = (A_j)_{j=0}^{t-1} = (A_{t-1}, A_{t-2}, \ldots, A_1, A_0),\ \mathsf{BitLen}(A_j) = w$$
$$\Leftrightarrow \quad A = \sum_{j=0}^{t-1} A_j W^j,\ A_j \in [0, W).$$

Example 1.2. $W = 2^8$인 경우,

$$\begin{aligned}
A = 4660 \quad &= \texttt{0x1234} \quad \Leftrightarrow \quad A = (\texttt{0x12}, \texttt{0x34}),\\
A = 2^{17} - 1 \quad &= \texttt{0x1ffff} \quad \Leftrightarrow \quad A = (\texttt{0x01}, \texttt{0xff}, \texttt{0xff}),\\
A = 2^{18} - 2^7 \quad &= \texttt{0x3ff80} \quad \Leftrightarrow \quad A = (\texttt{0x03}, \texttt{0xff}, \texttt{0x80}).
\end{aligned}$$

1.4 Software-Friendly Arithmetic

$$A \times 123 \quad \text{vs.} \quad A \times 128\ ? \quad \rightarrow \quad A \times 128 \quad = \quad A \ll 7$$

$$\lfloor A/123 \rfloor \quad \text{vs.} \quad \lfloor A/128 \rfloor\ ? \quad \rightarrow \quad \lfloor A/128 \rfloor \quad = \quad A \gg 7$$

$$A \bmod 123 \quad \text{vs.} \quad A \bmod 128\ ? \rightarrow A \bmod 128 = A \texttt{ \& 0x7f}$$

Let $A = (A_j)_{j=0}^{n-1} = (A_{n-1}, A_{n-2}, \ldots, A_1, A_0)$.

$$\left\lfloor \frac{A}{W^i} \right\rfloor \quad \rightarrow \quad A \gg wi = (\underbrace{0, \ldots, 0}_{i}, A_{n-1}, \ldots, A_{i+1}, A_i)$$

$$A \times W^i \quad \rightarrow \quad A \ll wi = (A_{n-1}, \ldots, A_1, A_0, \underbrace{0, \ldots, 0}_{i})$$

$$A \bmod W^i \quad \rightarrow \quad A\&(2^{wi} - 1) = (A_j)_{j=0}^{i-1} = (A_{i-1}, \ldots, A_0)$$

1.5 Big Integer Structure

큰 정수 구조체에는 다음 정보가 포함되어야 한다.

(*i*) 부호 정보: Negative, Non-negative

(*ii*) 워드열의 워드 개수

(*iii*) 워드열

큰 정수 구조체를 정의할 때, 워드열을 동적으로 메모리를 할당할지, 정적으로 할당할지를 결정해야 한다. 처리 대상 정수의 크기 제한이 없는 경우 워드열을 동적으로 메모리 할당해야 하지만, 암호 연산과 같이 고정 길이의 정수가 처리대상인 경우 속도 효율성을 위해 정적 메모리 할당을 사용하여 구조체를 정의한다.

C code (Dynamic Memory Allocation)

```
typedef struct{
    int sign;        // NEGATIVE or NON-NEGATIVE
    int wordlen;    // wordlen >= 0
    word* a;         // address for big integer
} bigint;
```

C code (Static Memory Allocation)

```
typedef struct{
    int sign;        // NEGATIVE or NON-NEGATIVE
    int wordlen;    // wordlen >= 0
    word a[MAX_WORD_LEN];         // address for big integer
} bigint;
```

주요 큰 정수 오픈소스 라이브러리가 사용하는 구조체는 다음과 같다.

C code (OpenSSL: bn_local.h)

```
#define BN_ULONG        unsigned long long
struct bignum_st {
    BN_ULONG *d;                /* Pointer to an array of 'BN_BITS2' bit chunks. */
    int top;                    /* Index of last used d +1. */
    /* The next are internal book keeping for bn_expand. */
    int dmax;                   /* Size of the d array. */
    int neg;                    /* one if the number is negative */
    int flags;
};
typedef struct bignum_st BIGNUM;
```

C code (mbed TLS: bignum.h)

```
typedef uint64_t mbedtls_mpi_uint;
typedef struct mbedtls_mpi
{
    int s;              /*!<  Sign: -1 if the mpi is negative, 1 otherwise */
    size_t n;           /*!<  total # of limbs  */
    mbedtls_mpi_uint *p;          /*!<  pointer to limbs  */
}
mbedtls_mpi;
```

본 고에서 사용하는 Sage 코드 용 정수 구조는 다음과 같이 정의한다.

```
'''
    Big Integer Operation.
    Integer Structure
    <-> [sign, [A(0), A(1), ..., A(n-1)]]
            sign = 1 <-> negative
            sign = 0 <-> non-negative
'''
def get_sign(a):
    return ZZ(a.sign() < 0)

def get_structure(a, w):
    return [get_sign(a), a.digits(base = 2^w)]

def show_bigint(A, w):
    print ("sign=%d::"%(A[0]), end = '')
    for j in range(len(A[1])-1,-1,-1):
        if w == 8:
            print ("%02x:"%(A[1][j]), end = '')
        elif w == 32:
            print ("%08x:"%(A[1][j]), end = '')
        elif w == 64:
            print ("%016x:"%(A[1][j]), end = '')
    print("\n", end = '')

a = 0x11223344556677889900ff
print ("a = 0x%x"%(a))

w = 8
A = get_structure(a, w); show_bigint (A, w)

w = 32
A = get_structure(a, w); show_bigint (A, w)

w = 64
A = get_structure(a, w); show_bigint (A, w)
```

Sage Output

```
a = 0x11223344556677889900ff
sign=0::11:22:33:44:55:66:77:88:99:00:ff:
sign=0::00112233:44556677:889900ff:
sign=0::0000000000112233:44556677889900ff:
```

Chapter 2

Basic Operations

2.1 Create BigInt, Delete BigInt, Zeroize BigInt

```
void bi_delete(bigint** x)
{
    if(*x == NULL)
        return;

#ifdef ZERORIZE
    array_init((*x)->a, (*x)->wordlen);
#endif

    free((*x)->a);
    free(*x);
    *x = NULL;
}
```

```
void bi_new(bigint** x, int wordlen)
{
    if(*x != NULL)
        bi_delete(x);

    *x = (bigint*)malloc(sizeof(bigint));
    (*x)->sign = NON_NEGATIVE;
    (*x)->wordlen = wordlen;
    (*x)->a = (word*)calloc(wordlen, sizeof(word));
}
```

2.2 Set BigInt, Show BigInt

$$\text{Sign, Word-String} \longrightarrow \text{Big Integer}$$

or

$$\text{Sign, String} \longrightarrow \text{Big Integer}$$

```
void bi_set_by_array(bigint** x, int sign, word* a, int wordlen); // bigint x <- sign and array a
void bi_set_by_string(bigint** x, int sign, char* str, int base); // bigint x <- base string

void bi_show_hex(bigint* x);  // show x in hexa. representation.
void bi_show_dec(bigint* x);  // show x in deci. representation.
void bi_show_bin(bigint* x);  // show x in binary representation.
```

2.3 Refine BigInt (Remove Last Zero Words)

```
void bi_refine(bigint* x)
{
    if(x == NULL)
        return;

    int new_wordlen = x->wordlen;
    while(new_wordlen > 1) // at least one word needed
    {
        if(x->a[new_wordlen - 1] != 0)
            break;
        new_wordlen--;
    }
    if (x->wordlen != new_wordlen)
    {
        x->wordlen = new_wordlen;
        x->a = (word*)realloc(x->a, sizeof(word)*new_wordlen);
    }

    if((x->wordlen == 1) && (x->a[0] == 0x0))
        x->sign = NON_NEGATIVE;
}
```

2.4 Assign BigInt ($Y \leftarrow X$)

```
void bi_assign(bigint** y, bigint* x)
{
    if(*y != NULL)
        bi_delete(y);

    bi_new(y, x->wordlen);
    (*y)->sign = x->sign;
    array_copy((*y)->a, x->a, x->wordlen);
}
```

2.5 Generate Random BigInt

```
void bi_gen_rand(bigint**x, int sign, int wordlen)
{
    bi_new(x, wordlen);
    (*x)->sign = sign;
    array_rand((*x)->a, wordlen);

    bi_refine(*x);
}
```

```
void array_rand(word* dst, int wordlen)
{
    byte* p = (byte*)dst;
    int cnt = wordlen * sizeof(word);
    while(cnt > 0)
    {
        *p = rand() & 0xff;
        p++;
        cnt--;
    }
}
```

2.6 Get Word Length/Bit Length/j-th Bit of BigInt

2.7 Get Sign and Flip Sign of BitInt

2.8 Set One, Set Zero, Is Zero?, Is One?

```
void bi_set_one(bigint** x)
{
    bi_new(x, 1);
    (*x)->sign = NON_NEGATIVE;
    (*x)->a[0] = 0x1;
}
```

```
void bi_set_zero(bigint** x)
{
    bi_new(x, 1);
    (*x)->sign = NON_NEGATIVE;
    (*x)->a[0] = 0x0;
}
```

IsZero$(A) : A = 0$?

Input: $A = (-1)^b \sum_{j=0}^{n-1} A_j W^j$ for some n and b
Output: `True` or `False`

```
1: if Sign(A) = 1 or A[0] ≠ 0 then
2:     return False
3: end if
4: for j = n − 1 downto 1 do
5:     if A_j ≠ 0 then
6:         return False
7:     end if
8: end for
9: return True
```

IsOne$(A) : A = 1$?

Input: $A = (-1)^b \sum_{j=0}^{n-1} A_j W^j$ for some n and b
Output: `True` or `False`

```
1: if Sign(A) = 1 or A[0] ≠ 1 then
2:     return False
3: end if
4: for j = n − 1 downto 1 do
5:     if A_j ≠ 0 then
6:         return False
7:     end if
8: end for
9: return True
```

2.9 Compare

n개의 워드열로 표현되는 음이 아닌 정수 $A \in [W^{n-1}, W^n)$와 m개의 워드열로 표현되는 음이 아닌 정수 $B \in [W^{m-1}, W^m)$의 대소 비교 함수 CompareABS를 다음과 같이 정의한다.

$$\mathsf{CompareABS}(A, B) := \begin{cases} 1, & A > B, \\ 0, & A = B, \\ -1, & A < B. \end{cases}$$

CompareABS 함수를 이용하여 임의의 두 정수의 대소 비교 함수 Compare를 정의한다.

CompareABS(A, B) **where** $A, B \in \mathbb{Z}_{\geq 0}$

Input: $A \in [W^{n-1}, W^n)$, $B \in [W^{m-1}, W^m)$
Output: $1(A > B)$ or $-1(A < B)$ or $0(A = B)$

```
1: n, m ← WordLen_w(A), WordLen_w(B)
2: if n > m then
3:     return 1                                  ▷ A > B
4: else if n < m then
5:     return −1                                 ▷ A < B
6: end if
7: for j = n − 1 down to 0 do                    ▷ Case: n = m
8:     if A_j > B_j then
9:         return 1                              ▷ A > B
10:    else if A_j < B_j then
11:        return −1                             ▷ A < B
12:    end if
13: end for
14: return 0                                     ▷ A = B
```

Compare(A, B) **where** $A, B \in \mathbb{Z}$

Input: A, B
Output: $1(A > B)$ or $-1(A < B)$ or $0(A = B)$

```
1: if Sign(A) = 0 and Sign(B) = 1 then
2:     return 1                                  ▷ A > B
3: end if
4: if Sign(A) = 1 and Sign(B) = 0 then
5:     return −1                                 ▷ A < B
6: end if
7: ret ← CompareABS(A, B)                        ▷ Case: AB ≥ 0
8: if Sign(A) = 0 then                           ▷ A, B ≥ 0
9:     return ret
10: else
11:    return ret × (−1)
12: end if
```

2.10 Left/Right Shift: $A \ll r,\ A \gg r$

Let $A = (A_j)_{j=0}^{n-1} = (A_{n-1}, A_{n-2}, \ldots, A_1, A_0) \in [0, W^n)$.

$$A \ll r \ \Leftrightarrow\ A \times 2^l, \quad A \gg r \ \Leftrightarrow\ \left\lfloor \frac{A}{2^r} \right\rfloor.$$

Case: $r \geq wn$. $\quad A \gg r = \left\lfloor \frac{A}{2^r} \right\rfloor = 0.$

Case: $r = wk$

$$A \ll wk = A \times W^k = (A_{n-1}, \ldots, A_1, A_0, \underbrace{0, \ldots, 0}_{k}),$$

$$A \gg wk = \left\lfloor \frac{A}{W^k} \right\rfloor = (\underbrace{0, \ldots, 0}_{k}, A_{n-1}, \ldots, A_{k+1}, A_k) = (A_{n-1}, \ldots, A_{k+1}, A_k). \quad (k < n)$$

Case: $r = wk + r'$ where $0 < r' < w$

$$A \ll r = A \times 2^{wk+r'} = (A \times 2^{r'}) \times W^k = (B_n, B_{n-1}, \ldots, B_1, B_0, \underbrace{0, \ldots, 0}_{k}),$$

where

$$\begin{aligned} B_j &= (A_j \ll r') \| (A_{j-1} \gg (w - r')), \quad (j = 0, 1, \ldots, n,\ A_{-1} = A_n = 0), \\ &= \begin{cases} A_0 \ll r', & j = 0, \\ (A_j \ll r') \| (A_{j-1} \gg (w - r')), & j = 1, \ldots, n-1, \\ A_{n-1} \gg (w - r'), & j = n. \end{cases} \end{aligned}$$

$$A \gg r = \left\lfloor \frac{A}{2^{wk+r'}} \right\rfloor = \left\lfloor \frac{A}{W^k} \times \frac{1}{2^{r'}} \right\rfloor = (\underbrace{0, \ldots, 0}_{k}, B_{n-1}, \ldots, B_{k+1}, B_k) = (B_{n-1}, \ldots, B_{k+1}, B_k). \quad (k < n)$$

where

$$\begin{aligned} B_j &= (A_{j+1} \ll (w - r')) \| (A_j \gg r'), \quad (j = k, k+1, \ldots, n-1,\ A_n = 0), \\ &= \begin{cases} (A_{j+1} \ll (w - r')) \| (A_j \gg r'), & j = k, k+1, \ldots, n-2, \\ A_{n-1} \gg r', & j = n-1. \end{cases} \end{aligned}$$

2.11 Reduction: $A \bmod 2^r$

Let $A = (A_j)_{j=0}^{n-1} = (A_{n-1}, A_{n-2}, \ldots, A_1, A_0) \in [0, W^n)$.

Case: $r \geq wn$. $A \bmod 2^r = A$.

Case: $r = wk$ **where** $k < n$

$$A \bmod W^k = A \wedge (2^{wk} - 1) = (A_j)_{j=0}^{k-1} = (A_{k-1}, \ldots, A_0).$$

Case: $r = wk + r'$ **where** $k < n$ **and** $0 < r' < w$

$$A \bmod 2^r = A \bmod 2^{wk+r'} = A \bmod W^k 2^{r'} = (B_k, B_{k-1}, \ldots, B_0),$$

where

$$B_j = \begin{cases} A_j, & j = 0, \ldots, k-1, \\ A_j \wedge (2^{r'} - 1), & j = k. \end{cases}$$

Chapter 3

Addition

두 양의 정수 $A \in [W^{n-1}, W^n)$와 $B \in [W^{m-1}, W^m)$의 합 $A + B$는 다음의 하한과 상한을 가진다.

$$W^{l-1} \leq A + B < W^{l+1}, \text{ where } l := \max(n, m).$$

Proof. A와 B를 다음과 같이 표현할 수 있다.

$$\begin{aligned} A &= aW^{n-1} + A', \\ B &= bW^{m-1} + B', \end{aligned} \quad \text{where } a, b \in (0, W), \quad \begin{aligned} 0 &\leq A' < W^{n-1}, \\ 0 &\leq B' < W^{m-1}. \end{aligned}$$

$n \geq m$이라고 가정할 때, 다음이 성립한다.

$$\begin{aligned} W^{n-1} \leq \max(A, B) \leq A + B &= aW^{n-1} + A' + bW^{m-1} + B' \\ &< (a + b)W^{n-1} + (W^{n-1} + W^{n-1}) = (a + b + 2)W^{n-1} \\ &\leq (W - 1 + W - 1 + 2)W^{n-1} = 2W^n \leq W^{n+1}. \end{aligned}$$

□

상기 정리에 의해 n개의 워드열로 표현하는 양의 정수 $A \in [W^{n-1}, W^n)$와 m개의 워드열로 표현하는 양의 정수 $B \in [W^{m-1}, W^m)$의 합 $A + B$는 최대 $\max(n, m) + 1$개의 워드열로 표현된다.

Example 3.1. $W = 2^8$인 경우,

$$\begin{array}{r} 2^8-1 \\ +\quad 2^8-1 \\ \hline 2^8+(2^8-2) \end{array} \quad \Rightarrow \quad \begin{array}{r} \texttt{0xff} \\ +\quad \texttt{0xff} \\ \hline \texttt{0x01 0xfe} \end{array} \qquad \begin{array}{r} 2^{16}-1 \\ +\quad 2^{16}-1 \\ \hline 2^{16}+(2^{16}-2) \end{array} \quad \Rightarrow \quad \begin{array}{r} \texttt{0xff 0xff} \\ +\quad \texttt{0xff 0xff} \\ \hline \texttt{0x01 0xff 0xfe} \end{array}$$

$$\begin{array}{r} 2^8-1 \\ +\quad 1 \\ \hline 2^8+0 \end{array} \quad \Rightarrow \quad \begin{array}{r} \texttt{0xff} \\ +\quad \texttt{0x01} \\ \hline \texttt{0x01 0x00} \end{array} \qquad \begin{array}{r} 2^{16}-1 \\ +\quad 1 \\ \hline 2^{16}+0 \end{array} \quad \Rightarrow \quad \begin{array}{r} \texttt{0xff 0xff} \\ +\quad \texttt{0x01} \\ \hline \texttt{0x01 0x00 0x00} \end{array}$$

3.1 Single-Precision Addition

3.1.1 $A + B$

C언어에서 `unsigned int`로 정의한 두 32비트 음이 아닌 정수 $A, B \in [0, 2^{32})$의 덧셈은 다음과 같이 계산한다.

$$A + B := A + B \bmod 2^{32}.$$

앞으로 $+$와 $+ \bmod 2^w$를 구분하기 위해 $+ \bmod 2^w$는 ⊞로 표기한다.

C code

```
unsigned int x = 0x12345678;
unsigned int y = 0xffffffff;
unsigned int z = x + y; // z <- x + y mod 2^(32)
printf("%08x + %08x = %08x\n", x, y, z);
```

```
bash-3.2$ ./a.out
12345678 + ffffffff = 12345677
```

$$\begin{aligned}
\texttt{0x12345678} \boxplus \texttt{0xffffffff} &= \texttt{0x12345678} + \underbrace{\texttt{0xffffffff}}_{=2^{32}-1} \bmod 2^{32} \\
&= \texttt{0x12345678} - \texttt{0x1} \bmod 2^{32} \\
&= \texttt{0x12345677}.
\end{aligned}$$

따라서 두 w비트 워드를 실제 정수로 간주하고 덧셈하기 위해서는 carry 처리가 필요하다. $A + B \geq W$인 경우에 carry가 발생하는데 이는 $A \boxplus B < A$로 판별할 수 있다. Proposition 3.1은 이를 보여준다.

Proposition 3.1. *두 정수 $A, B \in [0, W)$에 대해 다음이 성립한다.*

(*i*) *(나눗셈 알고리듬) $A+B$를 W로 나눈 몫은 $\left\lfloor \frac{A+B}{W} \right\rfloor$이며, 나머지는 $A \boxplus B$이다. 즉,*

$$A + B = \left\lfloor \frac{A+B}{W} \right\rfloor W + (A \boxplus B).$$

(*ii*) $\left\lfloor \frac{A+B}{W} \right\rfloor \in \{0, 1\}$.

(*iii*) $0 \leq A + B < W^{2a}$

(*iv*) $A + B \geq W$ *if and only if* $A \boxplus B < A$.

[a] $A + B$를 저장하기 위해 최대 2개의 워드가 필요함을 의미

Proof. (*i*)은 자명함.

(*ii*), (*iii*) $0 \leq A + B < W + W = 2W < W^2$.

(*iv*) ($\Rightarrow$) Since $W \leq A + B < 2W$, $A \boxplus B = (A + B) - W = A - (W - B) < A$.

($\Leftarrow$) $A + B < W \Rightarrow A \boxplus B = A + B \geq A$. □

Proposition 3.1에 의해 단일 워드 덧셈 유사부호는 다음과 같다.

$A + B$ where $A, B \in [0, W)$

Input: $A, B \in [0, W)$

Output: $c \in \{0, 1\}$ and $C \in [0, W)$ such that $A + B = cW + C$

1: $c \leftarrow 0$

2: $C \leftarrow A \boxplus B$ ▷ $A \boxplus B = (A + B) \wedge (2^w - 1)$

3: **if** $C < A$ **then**

4: $\quad c \leftarrow 1$ ▷ $A + B \geq W \Leftrightarrow A \boxplus B < A$

5: **end if**

6: **return** c, C

3.1.2 $A + B + c$

단일 워드 덧셈을 다중 워드 덧셈으로 확장하기 위해서는 carry가 추가적으로 더해지는 경우를 고려해야 한다. 이를 위해 다음의 명제가 필요하다.

Proposition 3.2. *두 정수 $A, B \in [0, W)$와 비트 $c \in \{0, 1\}$에 대해 다음이 성립한다.*

(i) $\left\lfloor \frac{A+B+c}{W} \right\rfloor \in \{0, 1\}$.

(ii) $0 \leq A + B + c < W^{2a}$.

(iii) *If* $W \leq A + B$, *then* $0 \leq (A \boxplus B) + c < W$.

[a] $A + B + c$는 최대 두 워드로 표현할 수 있음

Proof. $(i), (ii)$ $0 \leq A + B + c \leq (W - 1) + (W - 1) + 1 = 2W - 1 < W^2$.

(iii)는 Proposition 3.1의 (iii)에 의해 다음이 성립한다.

$$0 \leq (A \boxplus B) + c < A + c \leq W - 1 + 1 = W.$$

□

Proposition 3.2에 의해 $A + B + c$는 다음과 같이 계산한다.

$\left\lfloor \frac{A+B+c}{W} \right\rfloor, A \boxplus B \boxplus c \leftarrow \textbf{ADD}^{\textbf{ABc}}(A, B, c)$ **where** $A, B \in [0, W)$ **and** $c \in \{0, 1\}$

Input: $A, B \in [0, W)$ and $c \in \{0, 1\}$

Output: $c' \in \{0, 1\}$ and $C \in [0, W)$ such that $A + B + c = c'W + C$

1: $c' \leftarrow 0$
2: $C \leftarrow A \boxplus B$
3: **if** $C < A$ **then**
4: $\quad c' \leftarrow 1$
5: **end if**
6: $C \leftarrow C \boxplus c$
7: **if** $C < c$ **then** ▷ 라인 3이 `True`인 경우, Proposition 3.2 (iii)에 의해 반드시 `False`
8: $\quad c' \leftarrow c' + 1$
9: **end if**
10: **return** c', C

3.2 Multi-Precision Addition

3.2.1 ADDC: Addition of Same Signed Two Integers

0이 아닌 동일한 부호의 두 정수 A, B의 덧셈 $\mathsf{ADDC}(A, B)$에 대한 유사부호는 다음과 같다.

$A + B \leftarrow$ ADDC(A, B) where WordLen$(A) \geq$ WordLen(B) and Sign$(A) =$ Sign(B)

Input: $A = (-1)^b \sum_{j=0}^{n-1} A_j W^j, B = (-1)^b \sum_{j=0}^{m-1} B_j W^j$, where $A_j, B_j \in [0, W)$ and $b \in \{0, 1\}$

Output: $A + B = (-1)^b \sum_{j=0}^{l} C_j W^j$, where $C_j \in [0, W)$ and $l \in \{n-1, n\}$.

1: $B_j \leftarrow 0$ for $j = m, m+1, \ldots, n-1$.
2: $c \leftarrow 0$
3: **for** $j = 0$ to $n-1$ **do**
4: $\quad c, C_j \leftarrow \mathsf{ADD}^{\mathsf{ABc}}(A_j, B_j, c)$
5: **end for**
6: $C_n \leftarrow c$
7: **if** $C_n = 1$ **then**
8: $\quad$ **return** $(-1)^b \sum_{j=0}^{n} C_j W^j$
9: **else**
10: $\quad$ **return** $(-1)^b \sum_{j=0}^{n-1} C_j W^j$
11: **end if**

Remark. 두 양의 정수 $A = \sum_{j=0}^{n-1} A_j W^j \in [W^{n-1}, W^n)$, $B = \sum_{j=0}^{m-1} B_j W^j \in [W^{m-1}, W^m)$ $(A_j, B_j \in [0, W))$의 덧셈은 두 $\max(n, m)$워드 정수로 간주하여 연산한다. 예를 들어 $A = (\texttt{0x12}, \texttt{0x34})$, $B = (\texttt{0x88})$인 경우 B를 $B = (\texttt{0x00}, \texttt{0x88})$으로 재정의한다. 이 방식은 $n \geq m$라고 가정할 때, A를 $A = A^{(H)} W^m + A^{(L)}$ $(A^{(L)} < W^m)$로 표현하여 다음과 같이 계산한다.

$$
\begin{aligned}
A + B = \left(\sum_{j=0}^{n-1} A_j W^j \right) + \left(\sum_{j=0}^{m-1} B_j W^j \right) &= \left(\sum_{j=m}^{n-1} A_j W^j \right) + \left(\sum_{j=0}^{m-1} (A_j + B_j) W^j \right) \\
&= A^{(H)} W^m + \mathsf{ADDC}(A^{(L)}, B) \\
&= \left(C_m W^m + \sum_{j=m}^{n-1} A_j W^j \right) + \left(\sum_{j=0}^{m-1} C_j W^j \right) \\
&= \left(C_m + \sum_{j=0}^{n-1-m} A_{j+m} W^j \right) W^m + \left(\sum_{j=0}^{m-1} C_j W^j \right) \\
&= \left(\left(C_m + \sum_{j=0}^{n-1-m} A_{j+m} W^j \right) \ll wm \right) \oplus \left(\sum_{j=0}^{m-1} C_j W^j \right).
\end{aligned}
$$

이때 $C_m \in \{0, 1\}$임을 이용하여 $\left(C_m + \sum_{j=0}^{n-1-m} A_{j+m} W^j \right)$를 구현한다.

3.2.2 ADD: Addition of Two Integers

두 정수 A, B의 덧셈 $\mathsf{ADD}(A, B)$에 대한 유사부호는 다음과 같다.

$A + B \leftarrow \textbf{ADD}(A, B)$ **where** $A, B \in \mathbb{Z}$

Input: $A, B \in \mathbb{Z}$

Output: $A + B$

```
1: if A = 0 then
2:     return B
3: end if
4: if B = 0 then
5:     return A
6: end if
7: if A > 0 and B < 0 then
8:     return SUB(A, |B|)
9: end if
10: if A < 0 and B > 0 then
11:     return SUB(B, |A|)
12: end if
13: if WordLen(A) ≥ WordLen(B) then
14:     return ADDC(A, B)
15: else
16:     return ADDC(B, A)
17: end if
```

Chapter 4

Subtraction

두 양의 정수 $A \in [W^{n-1}, W^n)$와 $B \in [W^{m-1}, W^m)$에 대해 $A \geq B$일 때, 차 $A - B$는 다음의 하한과 상한을 가진다.

$$0 \leq A - B < A < W^n.$$

상기 정리에 의해 n개의 워드열로 표현하는 양의 정수 $A \in [W^{n-1}, W^n)$와 m개의 워드열로 표현하는 양의 정수 $B \in [W^{m-1}, W^m)$에 대해 $A \geq B$인 경우 차 $A - B$는 최대 $\max(n, m)$개의 워드열로 표현된다.

Example 4.1. $W = 2^8$인 경우,

$$\begin{array}{r} 2^8 - 1 \\ - \quad 1 \\ \hline 2^8 - 2 \end{array} \quad \Rightarrow \quad \begin{array}{r} \texttt{0xff} \\ - \ \texttt{0x01} \\ \hline \texttt{0xfe} \end{array} \qquad \begin{array}{r} 2^{16} - 1 \\ - \quad 1 \\ \hline 2^{16} - 2 \end{array} \quad \Rightarrow \quad \begin{array}{r} \texttt{0xff 0xff} \\ - \quad \texttt{0x01} \\ \hline \texttt{0xff 0xfe} \end{array}$$

$$\begin{array}{r} 1 \\ - \quad 2^8 - 1 \\ \hline -(2^8 - 2) \end{array} \quad \Rightarrow \quad \begin{array}{r} \texttt{0x01} \\ - \ \texttt{0xff} \\ \hline - \ \texttt{0xfe} \end{array} \qquad \begin{array}{r} 1 \\ - \quad 2^{16} - 1 \\ \hline -(2^{16} - 2) \end{array} \quad \Rightarrow \quad \begin{array}{r} \texttt{0x01} \\ - \ \texttt{0xff 0xff} \\ \hline - \ \texttt{0xff 0xfe} \end{array}$$

4.1 Single-Precision Subtraction

4.1.1 $A - B$

C언어에서 `unsigned int`로 정의한 두 32비트 음이 아닌 정수 A와 B의 뺄셈은 다음과 같이 계산한다.

$$A - B := A - B \bmod 2^{32}.$$

앞으로 $-$와 $- \bmod 2^w$를 구분하기 위해 $- \bmod 2^w$는 ⊟로 표기한다.

다음의 예제를 살펴보자.

C code
```
unsigned int x = 0xffffffff;
unsigned int y = 0x12345678;
unsigned int z1 = x - y; // z1 <- x - y mod 2^(32)
unsigned int z2 = y - x; // z2 <- y - x mod 2^(32)
printf("0x%08x - 0x%08x = 0x%08x\n", x, y, z1);
printf("0x%08x - 0x%08x = 0x%08x\n", y, x, z2);
```

Output
```
0xffffffff - 0x12345678 = 0xedcba987
0x12345678 - 0xffffffff = 0x12345679
```

상기 예제의 결과에서 0xffffffff - 0x12345678 = 0xedcba987는 자명하다. 그런데 0x12345678와 0xffffffff의 차는 $-$0xedcba987인데, 0x12345678 - 0xffffffff = 0x12345679이 나온 이유는 "$-$"를 "$-\bmod 2^{32}$"로 계산했기 때문이다.

$$\texttt{0x12345678} \boxminus \texttt{0xffffffff} = \texttt{0x12345678} - \underbrace{\texttt{0xffffffff}}_{2^{32}-1} \bmod 2^{32}$$
$$= \texttt{0x12345678} + \texttt{0x1} \bmod 2^{32} = \texttt{0x12345679}.$$

따라서 결과값에 -2^{32}를 더해주면 정상적인 결과값이 나온다.

$$-2^{32} + \texttt{0x12345679} = -\texttt{0xedcba987}.$$

두 정수 $A, B \in [0, W)$에 대해 $A - B \in [-(W-1), W-1]$이며, 다음과 같이 표현할 수 있다.

$$A - B = \begin{cases} A - B, & A \geq B, \\ -W + \underbrace{(W + A - B)}_{=A \boxminus B \in [0, W)}, & A < B. \end{cases}$$

따라서 $A - B$는 $-bW + C$ 형태로 표현된다. $(b \in \{0, 1\},\ C \in [0, W))$

$A - B$

Input: $A, B \in [0, W)$

Output: $b \in \{0, 1\},\ C \in [0, W)$ such that $A - B = -bW + C$

```
1: C ← A ⊟ B
2: if A ≥ B then
3:     b ← 0
4: else                                   ▷ Case A < B
5:     b ← 1
6: end if
7: return b, C
```

4.1.2 $A - b - B$

Proposition 4.1. *두 정수 $A, B \in [0, W)$와 비트 $b \in \{0, 1\}$에 대해 다음이 성립한다.*

(i) $-W \leq A - b - B < W$.

(ii) $\left\lfloor \frac{A-b-B}{W} \right\rfloor \in \{-1, 0\}$.

(iii) $-W \leq A - b - B < 0$*이면,* $A - b - B = -W + (A \boxminus b \boxminus B)$*이다.*

(iv) $A - b < 0$*인 경우는* $(A, b) = (0, 1)$*인 경우뿐이다. 따라서* $A - b = -1 = -W + (W - 1)$*로 표현할 수 있고, 이 경우* $(A \boxminus b) - B \in [0, W)$*이다.*

Proof. $(i), (ii), (iii)$ 자명함. (iv) $(A \boxminus b) - B = (W - 1) - B \in [0, W)$ □

$\left\lfloor \frac{A-b-B}{W} \right\rfloor, A \boxminus b \boxminus B \leftarrow \textbf{SUB}^{\textbf{AbB}}(A, b, B)$ **where** $A, B \in [0, W)$ **and** $b \in \{0, 1\}$

Input: $A, B \in [0, W), b \in \{0, 1\}$

Output: $b \in \{0, 1\},\ C \in [0, W)$ such that $A - B = -bW + C$

1: $C \leftarrow A \boxminus b$
2: **if** $A < b$ **then** ▷ $A = 0$ and $b = 1$
3: $\quad b \leftarrow 1$
4: **else**
5: $\quad b \leftarrow 0$
6: **end if**
7: **if** $C < B$ **then**
8: $\quad b \leftarrow b + 1$
9: **end if**
10: $C \leftarrow C \boxminus B$
11: **return** b, C

4.2 Multi-Precision Subtraction

4.2.1 SUBC: $A \geq B > 0$

양의 두 정수 $A, B (A \geq B > 0)$의 뺄셈 $\mathsf{SUBC}(A, B)$에 대한 유사부호는 다음과 같다.

$A - B \leftarrow$ SUBC(A, B) where $A \geq B > 0$

Input: $A = \sum_{j=0}^{n-1} A_j W^j \in [W^{n-1}, W^n), B = \sum_{j=0}^{m-1} B_j W^j \in [W^{m-1}, W^m), (A \geq B > 0)$, where $A_j, B_j \in [0, W)$

Output: $A - B = \sum_{j=0}^{l-1} C_j W^j \in [0, W^n)$

1: $B_j \leftarrow 0$ for $j = m, m+1, \ldots, n-1$
2: $b \leftarrow 0$
3: **for** $j = 0$ to $n-1$ **do**
4: $\quad b, C_j \leftarrow \mathsf{SUB}^{\mathsf{AbB}}(A_j, b, B_j)$
5: **end for**
6: $l \leftarrow \min\{j : C_{n-1} = C_{n-2} = \cdots = C_j = 0\}$
7: **return** $\sum_{j=0}^{l-1} C_j W^j$

4.2.2 SUB: Subtraction of Two Integers

두 정수 $A, B \in \mathbb{Z}$의 뺄셈 $\mathsf{SUB}(A, B)$의 유사부호는 다음과 같다.

$A - B \leftarrow$ SUB(A, B) where $A, B \in \mathbb{Z}$

Input: $A, B \in \mathbb{Z}$
Output: $A - B \in \mathbb{Z}$

```
 1: if A = 0 then
 2:     return −B
 3: end if
 4: if B = 0 then
 5:     return A
 6: end if
 7: if A = B then
 8:     return 0
 9: end if
10: if 0 < B ≤ A then
11:     return SUBC(A, B)
12: else if 0 < A < B then
13:     return −SUBC(B, A)
14: end if
15: if 0 > A ≥ B then
16:     return SUBC(|B|, |A|)
17: else if 0 > B > A then
18:     return −SUBC(|A|, |B|)
19: end if
20: if A > 0 and B < 0 then
21:     return ADD(A, |B|)
22: else                                  ▷ Case: A < 0 and B > 0
23:     return −ADD(|A|, B)
24: end if
```

Chapter 5

Multiplication

두 양의 정수 $A \in [W^{n-1}, W^n)$와 $B \in [W^{m-1}, W^m)$의 곱 AB는 다음의 하한과 상한을 가진다.

$$W^{n+m-2} \leq AB < W^{n+m}.$$

상기 정리에 의해 n개의 워드열로 표현하는 양의 정수 $A \in [W^{n-1}, W^n)$와 m개의 워드열로 표현하는 양의 정수 $B \in [W^{m-1}, W^m)$의 곱 AB는 최대 $n+m$개의 워드열로 표현된다.

$$n + m - 1 \leq \mathsf{WordLen}(AB) \leq n + m.$$

Example 5.1. $W = 2^8$인 경우,

$$\begin{array}{r} 2^8-1 \\ \times\quad 2^8-1 \\ \hline (2^8-2)2^8+1 \end{array} \Rightarrow \begin{array}{r} \texttt{0xff} \\ \times\quad \texttt{0xff} \\ \hline \texttt{0xfe 0x01} \end{array} \qquad \begin{array}{r} 2^8-1 \\ \times\quad 1 \\ \hline 2^8-1 \end{array} \Rightarrow \begin{array}{r} \texttt{0xff} \\ \times\ \texttt{0x01} \\ \hline \texttt{0xff} \end{array}$$

$$\begin{array}{r} 2^{16}-1 \\ \times\quad 2^{16}-1 \\ \hline (2^{16}-2)2^{16}+1 \end{array} \Rightarrow \begin{array}{r} \texttt{0xff 0xff} \\ \times\quad \texttt{0xff 0xff} \\ \hline \texttt{0xff 0xfe 0x00 0x01} \end{array}$$

$$\begin{array}{r} 2^{16}-1 \\ \times\quad 1 \\ \hline 2^{16}-1 \end{array} \Rightarrow \begin{array}{r} \texttt{0xff 0xff} \\ \times\quad \texttt{0x01} \\ \hline \texttt{0xff 0xff} \end{array}$$

5.1 Single-Precision Multiplication

두 정수 $A, B \in [0, W)$에 대해 곱 $AB \in [0, W^2)$는 다음과 같이 $\frac{w}{2}$비트 정수 곱셈 4회로 계산한다.

$$\begin{aligned} W^2 > AB &= (A_1 W^{\frac{1}{2}} + A_0)(B_1 W^{\frac{1}{2}} + B_0) \\ &= ((\underbrace{A_1 B_1}_{w/2\text{-bit mul.}})W + \underbrace{A_0 B_0}_{w/2\text{-bit mul.}}) + (\underbrace{A_1 B_0}_{w/2\text{-bit mul.}} + \underbrace{A_0 B_1}_{w/2\text{-bit mul.}})W^{\frac{1}{2}} \\ &= ((A_1 B_1 \ll w) + A_0 B_0) + ((\underbrace{A_1 B_0 + A_0 B_1}_{<2W}) \ll \frac{w}{2}). \end{aligned}$$

where

$$A = A_1 W^{\frac{1}{2}} + A_0, \quad B = B_1 W^{\frac{1}{2}} + B_0, \quad A_0, A_1, B_0, B_1 \in [0, W^{\frac{1}{2}}).$$

AB

Input: $A, B \in [0, W)$

Output: $C \in [0, W^2)$ such that $AB = C$

1: $A_1, A_0 \leftarrow A \gg \frac{w}{2}, A \bmod 2^{\frac{w}{2}}$

2: $B_1, B_0 \leftarrow B \gg \frac{w}{2}, B \bmod 2^{\frac{w}{2}}$

3: $T_1, T_0 \leftarrow A_1B_0, A_0B_1$ ▷ $T_1, T_0 \in [0, W)$

4: $T_0 \leftarrow T_1 \boxplus T_0$

5: $T_1 \leftarrow T_0 < T_1$ ▷ $T_1W + T_0 = A_1B_0 + A_0B_1$, $T_1 \in \{0, 1\}$

6: $C_1, C_0 \leftarrow A_1B_1, A_0B_0$ ▷ $C_1, C_0 \in [0, W)$

7: $T \leftarrow C_0$

8: $C_0 \leftarrow C_0 \boxplus (T_0 \ll \frac{w}{2})$

9: $C_1 \leftarrow C_1 + (T_1 \ll \frac{w}{2}) + (T_0 \gg \frac{w}{2}) + (C_0 < T)$ ▷ $C_1 \in [0, W)$

10: $C \leftarrow (C_1 \ll w) + C_0$

11: **return** C

```
w = 32

cnt = 0
while(cnt < 1000):
    A = ZZ.random_element(2^w)
    B = ZZ.random_element(2^w)
    #A = B = 2^w - 1

    A1, A0 = A >> (w/2), A%2^(w/2)
    B1, B0 = B >> (w/2), B%2^(w/2)
    T1, T0 = A1*B0, A0*B1
    T0 = (T1 + T0)%2^(w)
    T1 = T0 < T1
    C1, C0 = A1*B1, A0*B0
    T = C0
    C0 = (C0 + (T0 << (w/2)))%2^(w)
    C1 = C1 + (T1 << (w/2)) + (T0 >> (w/2)) + (C0 < T)
    C = (C1 << w) + C0

    print (A*B == C, A, B, C)
    cnt = cnt + 1
```

Example 5.2. Let $W = 2^8$. Compute `0xab` $\times$ `0xfe`.

5.2 Multi-Precision Multiplication

다음은 양의 두 정수 $A, B \in [0, W^n)$의 곱 $\mathsf{MULC}(A, B)$을 계산하는 주요 알고리듬이다. 이때, 계산 복잡도는 단일워드 곱셈 횟수를 의미한다.

MULC Algorithm	Computational Complexity	Year	Ref.
Schoolbook	$O(n^2)$		
Karatsuba	$O(n^{\log_2 3}) = O(n^{1.585})$	1960	[3]
Toom-Cook	$O(n^{\log_3 5}) = O(n^{1.465})$	1963	[1]
Schönhage-Strassen	$O(n \log_2 n \log_2 \log_2 n)$	1971	[4]
Fürer	$O(n \log_2 n)$	2007	[2]

임의의 두 정수 A, B의 곱 $\mathsf{MUL}(A, B)$를 다음과 같이 계산한다.

$AB \leftarrow \textbf{MUL}(A, B)$ **where** $A, B \in \mathbb{Z}$

Input: $A, B \in \mathbb{Z}$
Output: $C = AB \in \mathbb{Z}$

if $A = 0$ or $B = 0$ **then**
    return 0
end if
if $A = 1$ **then**
    return B
end if
if $A = -1$ **then**
    return $-B$
end if
if $B = 1$ **then**
    return A
end if
if $B = -1$ **then**
    return $-A$
end if
$C \leftarrow \mathsf{MULC}(|A|, |B|)$
return $(-1)^{\mathsf{Sign}(A)+\mathsf{Sign}(B)}C$

5.2.1 Schoolbook Multiplication

기본적 알고리듬(Schoolbook)의 계산복잡도는 $O(nm)$이다.

$$C = AB = \left(\sum_{j=0}^{n-1} A_j W^j\right)\left(\sum_{i=0}^{m-1} B_i W^i\right) = \sum_{j=0}^{n-1}\left(\sum_{i=0}^{m-1}(A_j B_i)W^{i+j}\right) \in [0, W^{n+m}).$$

Schoolbook : $AB \leftarrow$ **MULC**(A, B) **where** $A \in [W^{n-1}, W^n)$ **and** $B \in [W^{m-1}, W^m)$

Input: $A = \sum_{j=0}^{n-1} A_j W^j, B = \sum_{j=0}^{m-1} B_j W^j$, where $A_j, B_j \in [0, W)$
Output: $C = AB \in [0, W^{n+m})$

```
1: C ← 0
2: for j = 0 to n − 1 do
3:     for i = 0 to m − 1 do
4:         T ← A_j B_i                                   ▷ 0 ≤ T < W^2
5:         T ← T ≪ w(i + j)
6:         C ← ADDC(C, T)
7:     end for
8: end for
9: return C
```

5.3 Fast Multiplication

5.3.1 Karatsuba

양의 두 정수 $A \in [W^{n-1}, W^n), B \in [W^{m-1}, W^m)$는 다음과 같이 표현될 수 있다.

$$A = A_1 W^l + A_0, \quad B = B_1 W^l + B_0, \quad A_j, B_j \in [0, W^l),\ l = (\max(n, m) + 1) \gg 1.$$

상기식을 이용해서 다음과 같이 A와 B의 곱을 계산한다.

$$\begin{aligned} AB &= (A_1 W^l + A_0)(B_1 W^l + B_0) \\ &= ((A_1 B_1) W^{2l} + (A_0 B_0)) + (A_0 B_1 + A_1 B_0) W^l \\ &= ((A_1 B_1) W^{2l} + (A_0 B_0)) + ((A_0 - A_1)(B_1 - B_0) + A_0 B_0 + A_1 B_1) W^l. \end{aligned}$$

상기 식은 $2l$워드 정수 A와 B의 곱을 l워드 정수 곱셈 3회로 처리한다[1]. 이 분할 정복 방식(divide and conquer)을 Karatsuba 곱셈이라 한다.

[1]만일 $A_0 B_1 + A_1 B_0$을 $(A_1 + A_0)(B_1 + B_0) - A_0 B_0 - A_1 B_1$로 계산할 경우, $(A_1 + A_0)(B_1 + B_0)$는 $(l+1)$워드 정수 곱셈이기 때문에 구현에 적합하지 않다.

n워드 정수의 Karatsuba 곱셈은 1워드 곱셈 연산 기준으로 $O(n^{\log_2 3})$의 계산 복잡도를 가진다.

Proof. n워드 정수의 일반적 곱셈량을 $T(n)$이라 할 때, Karatsuba 방식의 계산복잡도는 다음과 같이 계산된다.

$$\begin{aligned} T(n) &= 3T(n/2) + a_0 n + b_0 \\ &= 3^2 T(n/2^2) + a_1 n + b_1 \\ &= 3^3 T(n/2^3) + a_2 n + b_2 \\ &= \cdots \\ &= 3^{\log_2 n} T(1) + an + b. \end{aligned}$$

□

Remark. Karatsuba 알고리듬은 2^k워드로 표현하는 정수들의 곱일 때, 성능이 극대화된다.

$$AB = ((A_1B_1)W^{2l} + (A_0B_0)) + ((A_0 - A_1)(B_1 - B_0) + A_0B_0 + A_1B_1)W^l.$$

Karatsuba Multiplication: $AB \leftarrow \textbf{MULC}^{Karatsuba}(A, B)$

Input: $A = \sum_{j=0}^{n-1} A_j W^j \in [0, W^n), B = \sum_{j=0}^{m-1} B_j W^j \in [0, W^m)$, where $A_j, B_j \in [0, W)$
Output: $C = AB \in [0, W^{n+m})$

procedure ($\mathsf{MULC}^{Karatsuba}(A, B)$)
  if $flag \geq \min(\mathsf{WordLen}(A), \mathsf{WordLen}(B))$ **then**
    return $\mathsf{MUL}(A, B)$
  end if
  $l \leftarrow (\max(\mathsf{WordLen}(A), \mathsf{WordLen}(B)) + 1) \gg 1$
  $A_1, A_0 \leftarrow A \gg lw, A \bmod 2^{lw}$ ▷ $A_j \in [0, W^l)$
  $B_1, B_0 \leftarrow B \gg lw, B \bmod 2^{lw}$ ▷ $B_j \in [0, W^l)$
  $T_1, T_0 \leftarrow \mathsf{MULC}^{Karatsuba}(A_1, B_1), \mathsf{MULC}^{Karatsuba}(A_0, B_0)$ ▷ $T_j \in [0, W^{2l})$
  $R \leftarrow (T_1 \ll 2lw) + T_0$ ▷ $R = T_1 \| T_0 \in [0, W^{4l})$
  $S_1, S_0 \leftarrow \mathsf{SUB}(A_0, A_1), \mathsf{SUB}(B_1, B_0)$
  $S \leftarrow (-1)^{\mathsf{Sign}(S_1) \oplus \mathsf{Sign}(S_2)} \mathsf{MULC}^{Karatsuba}(|S_1|, |S_0|)$ ▷ $|S| \in [0, W^{2l})$
  $S \leftarrow \mathsf{ADD}(S, T_1)$
  $S \leftarrow \mathsf{ADD}(S, T_0)$
  $S \leftarrow S \ll lw$
  $R \leftarrow \mathsf{ADD}(R, S)$
  return R
end procedure

```
reset()

w = 8

def get_sign(a):
    if a < 0:
        return 1
    return 0

def Mul_K(a, b):
    n = floor(len(a.bits())/w) + 1
    m = floor(len(b.bits())/w) + 1

    if 10 >= min(n, m):
        return a*b

    l = (max(n, m) + 1) >> 1
    lw = l*w
    a1, a0 = a >> lw, a%(2^lw)
    b1, b0 = b >> lw, b%(2^lw)
    t1, t0 = Mul_K(a1, b1), Mul_K(a0, b0)
    r = (t1 << 2*lw) + t0
    s1, s0 = a0 - a1, b1 - b0
    s = (-1)^(get_sign(s1) + get_sign(s0))\
                            * Mul_K(abs(s1), abs(s0))
    s = s + t1
    s = s + t0
    s = s << lw
    r = r + s
    return r

cnt = 0
while cnt < 100:
    print (cnt)
    k = 1024
    a = ZZ.random_element(2^k)
    b = ZZ.random_element(2^k)
    print (hex(a))
    print (hex(b))

    c1 = Mul_K(a, b)
    c2 = a * b
    print (c1 == c2)
    if c1 != c2:
        print (hex(c1))
        print (hex(c2))
        print (hex(c1 ^^ c2))
        break
    cnt = cnt + 1
```

5.3.2 Toom-Cook

5.3.3 Schönhage-Strassen

5.3.4 Fürer

Chapter 6

Squaring

6.1 Single-Precision Squaring

1워드 정수 $A \in (-W, W)$는 다음과 같이 표현할 수 있다.

$$A = (-1)^b(A_1 W^{\frac{1}{2}} + A_0), \quad b \in \{0, 1\}, A_j \in [0, W^{\frac{1}{2}}).$$

상기 표현식을 이용하여 제곱 $A^2 \in [0, W^2)$를 다음과 같이 계산한다.

$$\begin{aligned} W^2 > A^2 &= (A_1 W^{\frac{1}{2}} + A_0)^2 \\ &= (A_1^2 W + A_0^2) + (2A_0 A_1) W^{\frac{1}{2}} \\ &= ((A_1^2 \ll w) + A_0^2) + (A_0 A_1 \ll (\frac{w}{2} + 1)). \end{aligned}$$

상기 식에서 $A_1^2, A_0^2, A_0 A_1 \in [0, W)$이기 때문에 $(i, j = 0, 1)$ 제곱 A^2는 $\frac{w}{2}$비트 정수 곱셈 3회로 계산할 수 있다.

A^2

Input: $A \in (-W, W)$

Output: $C \in [0, W^2)$ such that $A^2 = C$

1: $A_1, A_0 \leftarrow |A| \gg \frac{w}{2}, |A| \bmod 2^{\frac{w}{2}}$

2: $C_1, C_0 \leftarrow A_1^2, A_0^2$ ▷ $C_1, C_0 \in [0, W)$

3: $C \leftarrow (C_1 \ll w) + C_0$ ▷ $C \in [0, W^2)$

4: $T \leftarrow A_0 A_1$ ▷ $T \in [0, W)$

5: $T \leftarrow T \ll (\frac{w}{2} + 1)$ ▷ $T \in [0, W^2)$

6: $C \leftarrow C + T$ ▷ Two words addition

7: **return** C

6.2 Multi-Precision Squaring

6.2.1 Schoolbook Squaring

t워드 두 양의 정수 $A = \sum_{j=0}^{t-1} A_j W^j, B = \sum_{j=0}^{t-1} B_j W^j \in [W^{t-1}, W^t)$ $(A_j, B_j \in [0, W))$의 곱은 다음과 같이 계산할 수 있다.

$$\begin{aligned} AB &= \left(\sum_{j=0}^{t-1} A_j W^j \right) \left(\sum_{i=0}^{t-1} B_i W^i \right) \\ &= \sum_{j=0}^{t-1} \sum_{i=0}^{t-1} (A_j B_i) W^{i+j} \\ &= \sum_{k=0}^{2t-2} \left(\sum_{\substack{0 \le i,j, \le k \\ i+j=k}} (A_j B_i) \right) W^k. \end{aligned}$$

이때 만일 $A = B$이면, 모든 $i \neq j$에 대해 $A_j B_i = A_i B_j$이 동일한 값이기 때문에, 곱셈 결과를 다음과 같이 재사용할 수 있다.

$$\begin{array}{ccccc} A_0A_0 & A_0A_1 & A_0A_2 & A_0A_3 & \cdots \\ A_1A_0 & A_1A_1 & A_1A_2 & A_1A_3 & \cdots \\ A_2A_0 & A_2A_1 & A_2A_2 & A_2A_3 & \cdots \\ A_3A_0 & A_3A_1 & A_3A_2 & A_3A_3 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

따라서 A^2을 다음과 같이 중복 연산을 제거하며 계산한다.

$$
\begin{aligned}
A^2 = \left(\sum_{j=0}^{t-1} A_j W^j\right)\left(\sum_{i=0}^{t-1} A_i W^i\right) &= \sum_{j=0}^{t-1}\left(\sum_{i=0}^{t-1}(A_j A_i)W^{i+j}\right) \\
&= \left(\sum_{j=0}^{t-1}\sum_{i=j}(A_i A_i)W^{2i}\right) + \left(\sum_{j=0}^{t-1}\sum_{i\neq j}(A_i A_j)W^{i+j}\right) \\
&= \sum_{j=0}^{t-1}\left(A_j^2 W^{2j} + \sum_{i=j+1}^{t-1} 2(A_j A_i)W^{i+j}\right) \\
&= \underbrace{\sum_{j=0}^{t-1} A_j^2 W^{2j}}_{=A_{t-1}^2 \| A_{t-2}^2 \| \cdots \| A_0^2} + 2\left(\sum_{j=0}^{t-1}\sum_{i=j+1}^{t-1}(A_j A_i)W^{i+j}\right).
\end{aligned}
$$

$A^2 \leftarrow$ **SQUC**(A)

Input: $A = \sum_{j=0}^{t-1} A_j W^j \in [W^{t-1}, W^t)$, $A_j \in [0, W)$

Output: $C = A^2 \in [0, W^{2t})$

1: $C_1, C_2 \leftarrow 0$
2: **for** $j = 0$ to $t - 1$ **do**
3: $\quad T_1 \leftarrow A_j^2$ ▷ $T_1 \in [0, W^2)$
4: $\quad T_1 \leftarrow T_1 \ll 2jw$
5: $\quad C_1 \leftarrow T_1 + C_1$ ▷ $C_1 \leftarrow T_1 \| C_1$
6: $\quad$ **for** $i = j + 1$ to $t - 1$ **do**
7: $\quad\quad T_2 \leftarrow A_j A_i$ ▷ $T \in [0, W^2)$
8: $\quad\quad T_2 \leftarrow T_2 \ll (i + j)w$
9: $\quad\quad C_2 \leftarrow$ ADD(C_2, T_2)
10: $\quad$ **end for**
11: **end for** ▷ $C_1 = A_{t-1}^2 \| A_{t-2}^2 \| \cdots \| A_0^2$
12: $C_2 \leftarrow C_2 \ll 1$
13: **return** ADD(C_1, C_2)

6.2.2 Karatsuba Squaring

$2n$워드로 표현되는 정수 $A \in (-W^{2n}, W^{2n})$는 다음과 같이 표현할 수 있다.

$$A = (-1)^b(A_1W^n + A_0), \quad b \in \{0, 1\}, A_j \in [0, W^n),$$

상기식을 이용해서 A^2을 다음과 같이 n워드 정수 제곱 2회와 곱셈 1회로 계산한다.

$$\begin{aligned} A^2 &= (A_1W^n + A_0)(A_1W^n + A_0) \\ &= (A_1^2W^{2n} + A_0^2) + 2A_0A_1W^n \\ &= ((A_1^2 \ll 2nw) + A_0^2) + (A_0A_1 \ll (nw + 1)). \end{aligned}$$

$\mathsf{SQUC}^{Karatsuba}(A)$

Input: $A = (-1)^b \sum_{j=0}^{n-1} A_j W^j$, where $b \in \{0, 1\}$, $A_j \in [0, W)$

Output: $C \in [0, W^{2n})$ such that $A^2 = C$

1: **procedure** $(\mathsf{SQUC}^{Karatsuba}(A))$
2: **if** $flag \geq \mathsf{WordLen}(A)$ **then**
3: **return** $\mathsf{SQUC}(A)$
4: **end if**
5: $l \leftarrow (\mathsf{WordLen}(A) + 1) \gg 1$
6: $A_1, A_0 \leftarrow |A| \gg lw, |A| \bmod 2^{lw}$ ▷ $A_j \in [0, W^l)$
7: $T_1, T_0 \leftarrow \mathsf{SQUC}^{Karatsuba}(A_1), \mathsf{SQUC}^{Karatsuba}(A_0)$ ▷ $T_j \in [0, W^{2l})$
8: $R \leftarrow (T_1 \ll 2lw) + T_0$ ▷ $R \leftarrow T_1 \| T_0$
9: $S \leftarrow \mathsf{MULC}^{Karatsuba}(A_1, A_0)$ ▷ $S \in [0, W^{2l})$
10: $S \leftarrow S \ll (lw + 1)$
11: $R \leftarrow \mathsf{ADDC}(R, S)$
12: **return** R
13: **end procedure**

Multi-precision 곱셈을 이용해서 임의의 정수 A의 제곱 $\mathsf{SQU}(A)$를 다음과 같이 계산한다.

SQU(A)

Input: $A = (-1)^a \sum_{j=0}^{n-1} A_j W^j$, where $b \in \{0, 1\}$, $A_j \in [0, W)$

Output: $C \in [0, W^{2n})$ such that $A^2 = C$

1: **if** $A = 0$ or $A = \pm 1$ **then**
2: **return** $|A|$
3: **end if**
4: **return** $\mathsf{SQUC}(A)$

Chapter 7

Division

음이 아닌 두 정수 $A = \sum_{j=0}^{n-1} A_j W^j \in [W^{n-1}, W^n)$과 $B = \sum_{j=0}^{m-1} B_j W^j \in [W^{m-1}, W^m)$에 대해 나눗셈 알고리듬(division algorithm)은 다음을 만족하는 몫(quotient) Q와 나머지(remainder) R을 찾는다.

$$A = BQ + R, \quad (0 \le R < B).$$

$B = 0$인 경우는 연산이 정의되지 않으며[1], $A < B$인 경우에는 $Q = 0$, $R = A$가 된다. 따라서 $A \ge B > 0$인 경우에 대한 나눗셈 연산이 필요하다.

Memo 7.1. $A < 0$인 경우, $|A|$에 대해서 몫 Q'과 나머지 R'를 구하면 $-Q' - 1$이 A를 B로 나누었을 때 몫, $B - R$이 나머지가 된다.

$$\begin{aligned} |A| = BQ' + R' \quad &\Rightarrow \quad -|A| = -BQ' - R' \\ &\Rightarrow \quad -|A| = -BQ' - B + B - R' \\ &\Rightarrow \quad A = -|A| = \underbrace{(-Q' - 1)}_{=Q} B + \underbrace{(B - R')}_{=R}. \end{aligned}$$

[1] $B \le 0$인 경우 R의 범위를 $0 \le R < |B|$로 정의한다.

Proposition 7.1. *몫 Q는 최대 $n-m+1$ 워드 정수이며, 나머지 R은 최대 m워드 정수이다.*

Proof.

$$Q = \frac{A-R}{B} \leq \frac{A}{B} < \frac{W^n}{W^{m-1}} = W^{n-m+1}.$$

□

다음은 가장 기본적인 나눗셈 알고리듬의 유사부호이다.

Division Algorithm

Input: $A, B \in \mathbb{Z}_{\geq 0}$
Output: `INVALID` or (Q, R) such that $A = BQ + R$, $(0 \leq R < B)$.

1: **if** $B \leq 0$ **then**
2: **return** `INVALID`
3: **end if**
4: **if** $A < B$ **then**
5: **return** $(0, A)$
6: **end if**
7: **if** $B = 1$ **then**
8: **return** $(A, 0)$
9: **end if**
10: $(Q, R) \leftarrow (0, A)$
11: **while** $R \geq B$ **do**
12: $(Q, R) \leftarrow (Q + 1, R - B)$
13: **end while**
14: **return** (Q, R)

7.1 Binary Long Division

$W = 2$인 경우 A와 B의 나눗셈은 다음과 같이 계산한다. 이를 이진 긴 나눗셈 알고리듬(binary long division)이라고 한다.

Long Division Algorithm (binary version)

Input: $A = \sum_{j=0}^{n-1} a_j 2^j, B \in \mathbb{Z}$ $(a_j \in \{0,1\},\ A \geq B > 0)$
Output: (Q, R) such that $A = BQ + R$ $(0 \leq R < B)$.

1: $(Q, R) \leftarrow (0, 0)$
2: **for** $j = n - 1$ downto 0 **do**
3: $\quad R \leftarrow 2R + a_j$ ▷ $R \leftarrow (R \ll 1) \oplus a_j$
4: $\quad$ **if** $R \geq B$ **then**
5: $\quad\quad (Q, R) \leftarrow (Q + 2^j, R - B)$ ▷ $Q \leftarrow Q \oplus (1 \ll j)$
6: $\quad$ **end if**
7: **end for**
8: **return** (Q, R)

상기 알고리듬은 뺄셈 연산$(R - B)$ 기준으로 계산 복잡도가 $O(\log_2 A)$이다.

Sage code

```
def div_long(a,b):
    q, r = 0, 0
    n = floor(log(a,2) + 1) - 1
    while n >= 0:
        r = 2*r + ((a >> n)&1)
        if r >= b:
            q = q + (1 << n)
            r = r - b
        n = n - 1
    return q, r
```

7.2 Multi-Precision Long Division

이진 긴 나눗셈 알고리듬은 다중 워드 단위 긴 나눗셈 DIV로 확장할 수 있다.

$\mathsf{DIVCC}(A,B)$	$\Rightarrow$	$\mathsf{DIVC}(A,B)$	$\Rightarrow$	$\mathsf{DIV}(A,B)$
				$A = \sum_{j=0}^{n-1} A_j W^j$ $B = \sum_{j=0}^{m-1} B_j W^j$ $(A_j, B_j \in [0, W))$

7.2.1 DIV(A, B)

Long Division Algorithm (Multi-precision version)

Input: $A = \sum_{j=0}^{n-1} A_j W^j, B$ $(A_j \in [0, W))$
Output: $(Q = \sum_{j=0}^{n-1} Q_j W^j, R)$ such that $A = BQ + R$ $(0 \le R < B,\ Q_j \in [0, W))$.

1: **if** $B \le 0$ **then**
2: **return** `INVALID`
3: **end if**
4: **if** $A < B$ **then**
5: **return** $(0, A)$
6: **end if**
7: $(Q, R) \leftarrow (0, 0)$ ▷ $Q_j = 0$ for all j
8: **for** $i = n - 1$ downto 0 **do**
9: $R \leftarrow RW + A_i$
10: $(Q_i, R) \leftarrow \mathsf{DIVC}(R, B)$ ▷ $R < BW$, $0 \le R' < B$
11: **end for**
12: **return** (Q, R)

```
1: for i = n − 1 downto 0 do
2:     R ← RW + A_i
3:     (Q_i, R) ← DIVC(R, B)                    ▷ R < BW, 0 ≤ R′ < B
4: end for
```

상기 알고리듬에서 함수 $\mathsf{DIVC}(A, B)$는 $0 \le A < BW$를 만족하는 A와 B에 대한 나눗셈 연산으로, 반환값인 몫 Q_i는 $[0, W)$ 집합의 원소이다.

Proof. $0 \le R < B$인 경우, $0 \le RW + A_i \le (B-1)W + (W-1) = BW - 1 < BW$를 만족한다. 또한 $BW > A = BQ + R$이면, $B(W - Q) > R \ge 0$이 되기 때문에 $W > Q$이어야 한다. □

7.2.2 DIVC(A, B)

DIVC(A, B)는 $0 \le A < BW$를 만족하는 두 정수 $A = \sum_{j=0}^{n} A_j W^j$와 $B = \sum_{j=0}^{m-1} B_j W^j$의 나눗셈 연산이다. 만일 $0 \le A < B$인 경우는 몫이 0, 나머지가 A임을 바로 알 수 있기 때문에, $B \le A < BW$인 경우만 생각하면 된다.

A와 B를 다음과 같이 표현한다.

$$A = \sum_{j=0}^{m} A_j W^j, \quad B = \sum_{j=0}^{m-1} B_j W^j, \quad (A_j, B_j \in [0, W]).$$

DIVC(A, B)는 다음의 근사 몫 $\widehat{Q}$으로부터 실제 Q를 얻는 방식으로 구현한다.

$$\begin{aligned} \widehat{Q} &:= \min\left(\left\lfloor \frac{A_m W(n-m) + A_{m-1}}{B_{m-1}} \right\rfloor, W-1\right) \\ &= \begin{cases} \min\left(\left\lfloor \frac{A_m W + A_{m-1}}{B_{m-1}} \right\rfloor, W-1\right), & n = m+1, \\ \left\lfloor \frac{A_{m-1}}{B_{m-1}} \right\rfloor, & n = m. \end{cases} \end{aligned}$$

Proposition 7.2. $n = m + 1$인 경우 다음이 성립한다.

(i) $A_m = B_{m-1}$이면, $\widehat{Q} = W - 1$이다.

(ii) $A_m < B_{m-1}$이면, $\widehat{Q} = \left\lfloor \frac{A_m W + A_{m-1}}{B_{m-1}} \right\rfloor$이다.

Proof. (i)

$$\widehat{Q} = \min\left(\left\lfloor \frac{A_m W + A_{m-1}}{A_m} \right\rfloor, W - 1\right) = \min\left(W + \left\lfloor \frac{A_{m-1}}{A_m} \right\rfloor, W - 1\right) = W - 1.$$

(ii)

$$A_m W + A_{m-1} \leq A_m W + (W - 1) = (A_m + 1)W - 1 < (A_m + 1)W \leq B_{m-1} W$$

$$\Rightarrow \quad \left\lfloor \frac{A_m W + A_{m-1}}{B_{m-1}} \right\rfloor \leq \frac{A_m W + A_{m-1}}{B_{m-1}} < W.$$

□

Theorem 7.1. $0 < B \leq A < BW$를 만족하는 두 정수 $A = \sum_{j=0}^{n-1} A_j W^j$, $B = \sum_{j=0}^{m-1} B_j W^j \in [W^{m-1}, W^m)$ $(A_j, B_j \in [0, W))$는 다음을 만족한다.

$$A - \widehat{Q}B < B.$$

Proof. $\widehat{Q} = W - 1$인 경우, $A < BW$이기 때문에 자명하게 성립한다.

$$A - \widehat{Q}B = A - (W-1)B = A - BW + B < B.$$

$\widehat{Q} = \left\lfloor \frac{A_m W(n-m) + A_{m-1}}{B_{m-1}} \right\rfloor$인 경우, $X - \lfloor X \rfloor < 1$이므로 $\frac{A_m W(n-m) + A_{m-1}}{B_{m-1}} - \widehat{Q} < 1$이 되어 다음의 부등식을 얻는다.

$$\widehat{Q}B_{m-1} \geq A_m W(n-m) + A_{m-1} - B_{m-1} + 1.$$

상기 부등식에 의해 다음이 유도된다.

$$\begin{aligned} A - \widehat{Q}B = A - \widehat{Q}(B_{m-1}W^{m-1} + \cdots + B_0) &\leq A - \widehat{Q}B_{m-1}W^{m-1} \\ &\leq A - (A_m W(n-m) + A_{m-1} - B_{m-1} + 1)W^{m-1} \\ &\leq (A_{m-2}W^{m-2} + A_{m-3}W^{m-3} + \cdots + A_0) + (B_{m-1} - 1)W^{m-1} \\ &< W^{m-1} + (B_{m-1} - 1)W^{m-1} = B_{m-1}W^{m-1} \leq B. \end{aligned}$$

□

Corollary 7.1. *Theorem 7.1에 의해 $Q \leq \widehat{Q}$임을 알 수 있다.*

Proof.

$$A - \widehat{Q}B < B \quad \Rightarrow \quad \frac{A}{B} - \widehat{Q} < 1 \quad \Rightarrow \quad \frac{A}{B} < 1 + \widehat{Q}$$

$$\Rightarrow \quad Q - 1 = \left\lfloor \frac{A}{B} \right\rfloor - 1 \leq \frac{A}{B} - 1 < \widehat{Q}.$$

□

Theorem 7.2. $0 < B \le A < BW$를 만족하는 두 정수 $A = \sum_{j=0}^{n-1} A_j W^j$, $B = \sum_{j=0}^{m-1} B_j W^j \in [W^{m-1}, W^m)$ $(A_j, B_j \in [0, W))$에 대해, $W \le 2B_{m-1}$이면 (즉, $B_{m-1} \ge 2^{w-1}$), A를 B로 나누었을 때 몫 Q는 다음을 만족한다.

$$W \le 2B_{m-1} \quad \Rightarrow \quad \widehat{Q} \le Q + 2.$$

Memo 7.2. B_{m-1}의 MSB값이 1이면 $B_{m-1} \ge 2^{w-1}$이다.

Proof. Claim 1. $\widehat{Q} \le \left\lfloor \frac{A}{B_{m-1}W^{m-1}} \right\rfloor$.

다음 부등식이 성립한다.

$$B = B_{m-1}W^{m-1} + \underbrace{B_{m-2}W^{m-2} + \cdots + B_0}_{<W^{m-1}} < B_{m-1}W^{m-1} + W^{m-1}.$$

상기 부등식으로부터 $0 \le B - W^{m-1} < B_{m-1}W^{m-1}$이 유도된다.

$$\widehat{Q} = \left\lfloor \frac{A_m W(n-m) + A_{m-1}}{B_{m-1}} \right\rfloor = \left\lfloor \frac{A_m W^m (n-m) + A_{m-1}W^{m-1}}{B_{m-1}W^{m-1}} \right\rfloor \le \left\lfloor \frac{A}{B_{m-1}W^{m-1}} \right\rfloor.$$

Claim 2. $B = W^{m-1}$이면, $\widehat{Q} = Q$.

만일 $B = W^{m-1}$인 경우 상기 부등식은 $\widehat{Q} \leq \left\lfloor \frac{A}{B_{m-1}B} \right\rfloor \leq \lfloor \frac{A}{B} \rfloor = Q$가 된다.

Claim 3. $B > W^{m-1}$이면, $\widehat{Q} - Q < \frac{A}{B}\left(\frac{W^{m-1}}{B-W^{m-1}}\right) + 1$.

$B > W^{m-1}$인 경우, 다음의 부등식이 유도된다.

$$\begin{aligned}
\widehat{Q} - Q &= \left\lfloor \frac{A_m W(n-m) + A_{m-1}}{B_{m-1}} \right\rfloor - \left\lfloor \frac{A}{B} \right\rfloor \\
&< \frac{A}{B_{m-1}W^{m-1}} - \frac{A}{B} + 1 \\
&= \frac{A}{B}\left(\frac{B}{B_{m-1}W^{m-1}} - 1\right) + 1 \\
&< \frac{A}{B}\left(\frac{B}{B - W^{m-1}} - 1\right) + 1 \\
&= \frac{A}{B}\left(\frac{W^{m-1}}{B - W^{m-1}}\right) + 1.
\end{aligned}$$

(To show) If $2 < \widehat{Q} - Q$, then $B_{m-1} < 2^{w-1}$.
$2 < \widehat{Q} - Q$라고 가정하자. $\widehat{Q} - Q < \frac{A}{B}\left(\frac{W^{m-1}}{B - W^{m-1}}\right) + 1$이므로 다음을 얻는다.

$$2 < \widehat{Q} - Q < \frac{A}{B}\left(\frac{W^{m-1}}{B - W^{m-1}}\right) \quad \Rightarrow \quad \frac{A}{B} > 2\left(\frac{B - W^{m-1}}{W^{m-1}}\right)$$
$$\geq 2\left\lfloor\frac{B - W^{m-1}}{W^{m-1}}\right\rfloor = 2(B_{m-1} - 1).$$

$\widehat{Q} \leq W - 1$이기 때문에 다음의 결과가 유도된다.

$$W - 4 \geq \widehat{Q} - 3 \geq Q = \left\lfloor\frac{A}{B}\right\rfloor \geq 2(B_{m-1} - 1) \quad \Rightarrow \quad W - 2 \geq 2B_{m-1}$$
$$\Rightarrow \quad 2^{w-1} = \frac{W}{2} > \frac{W}{2} - 1 \geq B_{m-1}.$$

따라서 $B_{m-1} \geq 2^{w-1}$이면, $\widehat{Q} \leq Q + 2$이 된다. □

$B \le A < BW$인 경우 DIVC를 계산할 때, $B_{m-1} \in [2^{w-1}, 2^w)$ 조건이 있을 때, 상기 정리에 의해 몫 Q는 집합 $\{\widehat{Q}, \widehat{Q}-1, \widehat{Q}-2\}$의 원소이다. 일반적인 DIVC$(A, B)$를 계산하기 위해서 조건 $B_{m-1} \in [2^{w-1}, 2^w)$를 완화해야 하는데, 이는 다음의 정리를 이용해서 해결할 수 있다.

Proposition 7.3. *두 정수 A, B에 대해 A를 B로 나누었을 때 몫을 Q, 나머지 R라고 할 때, 임의의 음이 아닌 정수 k에 대해 $A' = 2^k A$를 $B' = 2^k B$로 나누었을 때 몫 Q', 나머지 R'은 다음을 만족한다.*

$$Q = Q', \quad R = 2^{-k} R'.$$

Proof.

$$A = BQ + R \quad \Rightarrow \quad \underbrace{2^k A}_{=A'} = \underbrace{2^k B}_{=:B'} \underbrace{Q}_{=:Q'} + \underbrace{2^k R}_{=:R'(\in[0,B'))} .$$

□

다음은 DIVC(A, B)의 유사부호이다.

DIVC(A,B)

Input: $A = \sum_{j=0}^{n-1} A_j W^j, B = \sum_{j=0}^{m-1} B_j W^j \in \mathbb{Z}$ $(A_j, B_j \in [0, W)$, $0 \le A < BW)$
Output: $(Q = \sum_{j=0}^{n-1} Q_j W^j, R)$ such that $A = BQ + R$ $(0 \le R < B,\ Q_j \in [0, W))$.

1: **if** $A < B$ **then**
2: **return** $(0, A)$
3: **end if**
4: Compute $k \in \mathbb{Z}_{\ge 0}$ such that $2^k B_{m-1} \in [2^{w-1}, 2^w)$
5: $A', B' \leftarrow 2^k A, 2^k B$
6: $Q', R' \leftarrow$ DIVCC(A', B')
7: $Q, R \leftarrow Q', 2^{-k} R'$
8: **return** (Q, R)

7.2.3 DIVCC(A, B)

DIVCC(A, B)

Input: $A = \sum_{j=0}^{n-1} A_j W^j, B = \sum_{j=0}^{m-1} B_j W^j \in \mathbb{Z}_{>0}$ $(A_j, B_j \in [0, W),\ 0 < B \le A < BW,\ B_{m-1} \ge 2^{w-1})$
Output: $(Q = \sum_{j=0}^{n-1} Q_j W^j, R)$ such that $A = BQ + R$ $(0 \le R < B,\ Q_j \in [0, W))$.

if $n = m$ **then**
    $Q \leftarrow \left\lfloor \frac{A_{m-1}}{B_{m-1}} \right\rfloor$ ▷ $Q \in [0, W)$
end if
if $n = m + 1$ **then**
    if $A_m = B_{m-1}$ **then**
        $Q \leftarrow W - 1$
    else ▷ $A_m < B_{m-1}$
        $Q \leftarrow \left\lfloor \frac{A_m W + A_{m-1}}{B_{m-1}} \right\rfloor$ ▷ $Q \in [0, W)$
    end if
end if
$R \leftarrow A - BQ$
while $R < 0$ **do** ▷ At most 3 loops
    $(Q, R) \leftarrow (Q - 1, R + B)$
end while
return (Q, R)

Question 7.1. 상기 유사부호에서 곱셈 BQ는 어떤 알고리듬으로 계산할 것인가?

Question 7.2. 상기 유사부호에서 $\left\lfloor \frac{A_m W + A_{m-1}}{B_{m-1}} \right\rfloor$를 어떻게 계산할 것인가?

Answer. 다음과 같이 이진 긴 나눗셈 알고리듬을 조건에 맞게 수정하여 계산한다.

Long Division Algorithm (2-word version)

Input: $A = A_1W + A_0, B$ ($A_1, A_0 \in [0, W)$, $B \in [2^{w-1}, 2^w)$, $A_1 < B$, $A_0 = \sum_{j=0}^{w-1} a_j 2^j$)
Output: Q such that $A = BQ + R$ ($0 \le R < B,\ Q_j \in [0, W)$).

1: $(Q, R) \leftarrow (0, A_1)$
2: **for** $j = w - 1$ downto 0 **do**
3: **if** $R \ge 2^{w-1}$ **then**
4: $(Q, R) \leftarrow (Q + 2^j, 2R + a_j - B)$ ▷ $2R + a_j - B = R + a_j - (B - R) < W$
5: **else**
6: $R \leftarrow 2R + a_j$ ▷ $R \in [0, W)$
7: **if** $R \ge B$ **then**
8: $(Q, R) \leftarrow (Q + 2^j, R - B)$
9: **end if**
10: **end if**
11: **end for**
12: **return** Q

Bibliography

[1] Stephen A. Cook and Stål O. Aanderaa. On the minimum computation time of functions. *Transactions of the American Mathematical Society*, 142:291–314, 1969.

[2] Martin Fürer. Faster integer multiplication. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, page 57–66, New York, NY, USA, 2007. Association for Computing Machinery.

[3] A. Karatsuba and Yu. Ofman. Multiplication of many-digital numbers by automatic computers. *Proceedings of USSR Academy of Sciences*, 145(7):293–294, 1962.

[4] Arnold Schönhage and Volker Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3-4):281–292, 1971.