매우 큰 수치 데이터 처리를 위한 정수형 겁수의 사칙연산과 논리연산

김귀연*, 박승범*, 김병기*, 최준용**
*전남대학교 전자컴퓨터공학부
**경북대학교 전자전기컴퓨터공학부
*e-mail:gykim@chonnam.ac.kr

Four Arithmetical and Logical Operations of the Integer Type Kalpa Number for Operating Extremely Large Number

Gwi-Yeon Kim*, Seung-Beom Park*, Byung-Ki Kim*, Jun-Yong Choi**
*ECE , Chonnam National University
**EECS, Kyungpook National University

요 약

수학, 물리학, 전자전기공학에서 취급하는 수치 자료는 매우 크거나 정밀도가 높을 수 있고, 이 경우 계산과정에서 흔히 오버플로우나 언더플로우가 발생한다. 본 연구는 이런 현상을 최소화하기 위하여 겁수(Kalpa Number) 표현 방법을 제안하고, 정수형 겁수의 사칙연산과 논리연산을 구현하였다. 겁수를 이용할 경우 실행속도는 느리지만 매우 큰 수의 연산이 가능하다.

1. 서론

디지털 컴퓨터는 현대 사회에서 매우 중요한 위치를 차지하고 있다. 컴퓨터는 의학치료, 기상예측, 우주과학, 전기전자공학, 수학, 물리학 등의 여러 분야에서 다양한 목적으로 사용되고 있다[1].

이들 각 분야의 프로그램에 사용되는 자료는 컴퓨터 내부에서 한정된 크기만 표현되므로 수치가 이보다 크거나 정밀도가 높을 경우 원래 수치의 근사치가 저장되는 오버플로우나 언더플로우로 나타난다.

수학 분야에서 베르누이(Bernoulli) 수나 오일러(Euler) 수를 구하는 문제[2]는 매우 큰 수를 취급하므로 컴퓨터를 이용하여도 정확한 값을 구하기 어렵다. 또한, 전자의 움직임을 설명하는 프로그램을 실행하다 보면 계산 과정에서 $10^{\pm 308}$ 범위를 초과하는 값들을 쉽게 발견할 수 있으며, 정밀한 값을 구하기 어려우므로 근사값을 구하여 계산한다[3].

수학분야에서 π의 정밀한 값을 찾기 위한 노력으로 HFLOAT 또는 apfloat 과 같은 프로그램이 개발되어 사용되고 있다. 이 프로그램들을 이용하면 π의 수백만 ~ 수십억 자리까지 계산이 가능하다[4][5].

CLN(Class Library for Numbers)은 매우 크거나 정밀한 수를 취급할 수 있도록 개발된 프로그램이다. cl_LF 형은 부호 비트 1, 지수 비트 32, 64 이상의 가수(mantissa) 비트를 사용한다, 따라서 지수의 크기가 2^{31} 인 매우 큰 수를 표현할 수 있고, 가수부분은 크기에 제한을 두지 않는다[6].

겁수(Kalpa Number)는 매우 크거나 정밀한 수를 소프트웨어적인 방법으로 처리하기 위하여 제안된 방법이다[7]. 본 논문은 겁수에 대하여 소개하고 정수형 겁수(Kint)의 사칙연산과 논리 연산을 구현한다.

2. 겁수(Kalpa Number)

겁수(Kalpa Number)는 겁수 표현법으로 저장된 수를 의미하며, 정수형 겁수와 실수형 겁수 두 가지 형(type)을 제안하고 있다[7].

2.1 겁수 표현법

겁수는 정수형인 경우 부호를 위한 문자형 변수 하나와 크기를 표현하는 문자형 배열을 사용한다. 부호는 양수일 경우 1, 음수일 경우 -1을 저장한다. 수의 크기는 문자형의 배열에 10진법으로 0~9 범위의 수를 저장한다.

실수형인 경우 수와 지수의 부호를 위한 문자형 변수 두 개와 지수 및 가수를 표현하는 두 개의 문자형 배열을 사용한다.

예를 들면 (그림 1)은 정수형일 경우 $(+452896\cdots5)_{10}$, 지수일 경우 $10^{+452896\cdots5}$, 가수일 경우 $(0.452896\cdots5)_{10}$를 의미한다. 가장 왼쪽의 1은 부호 '+'를 의미한다.

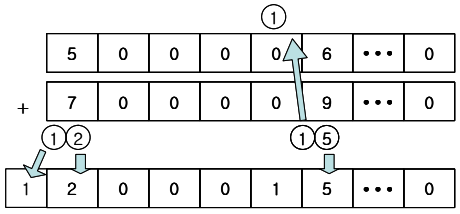
(그림 1) 겁수 표현법

2.2 겁수의 사칙연산 전략

겁수와 겁수간의 연산을 위하여 다양한 연산자를 정의하여야 한다. 본 절은 겁수의 사칙 연산만을 논의한다.

(1) 겁수의 덧셈과 뺄셈

정수형 겁수의 덧셈과 뺄셈은 종이에 연필로 덧셈, 뺄셈하는 것과 동일한 방법을 사용한다. 예를 들면 양수인 두 정수형 겁수 덧셈의 경우 (그림 2)와 같은 방법을 이용한다. (그림 2)에서 가장 왼쪽에 발생한 올림수는 배열의 왼쪽 공간에 추가하여 저장한다.



(그림 2) 정수형 겁수의 덧셈

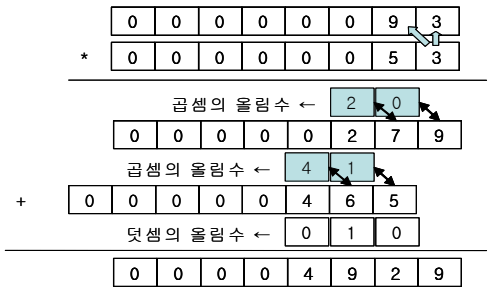
뺄셈의 경우 절대값이 큰 수에서 작은 수를 빼고, 계산 결과의 수에 절대값이 큰 쪽의 부호를 저장한다.

실수형 겁수의 덧셈과 뺄셈은 지수가 작은 수를 지수가 큰 수와 같도록 소수점을 이동한다. 예를 들면, $0.12*10^5$와 $0.34*10^2$을 덧셈할 경우 $0.34*10^2$을 $0.00034*10^5$ 으로 조정한다. 지수의 값이 동일하면, 가수부분은 정수형 겁수의 덧셈 또는 뺄셈과 동일한 방법으로 계산한다.

즉, 가수의 덧셈은 (그림 2)와 유사하게 처리된다. 이때 가장 왼쪽에 발생한 올림수는 배열의 왼쪽 공간에 추가하여 저장하고, 지수를 1 증가한다. 뺄셈도 유사한 방법으로 처리한다.

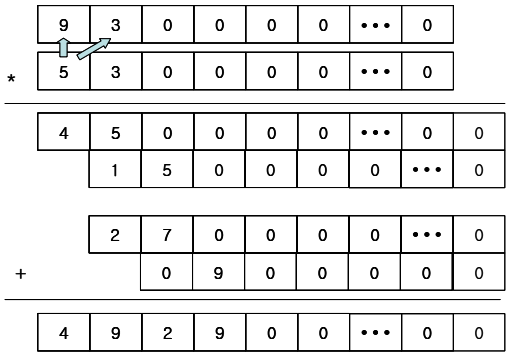
(2) 겁수의 곱셈

정수형 겁수의 곱셈은 종이에 연필을 이용하여 계산하는 방법과 동일하게 처리한다. 예를 들면 93*53은 (그림 3)과 같이 처리한다.



(그림 3) 정수형 겁수의 곱셈 방법

실수형 겁수 곱셈은 지수와 가수를 분리하여 연산한다. 지수는 정수형 겁수의 덧셈으로 처리하고, 가수는 정수형 겁수의 곱셈과 비슷한 방법으로 계산한다. 예를 들면 0.93*0.53의 계산방법은 (그림4)와 같다.

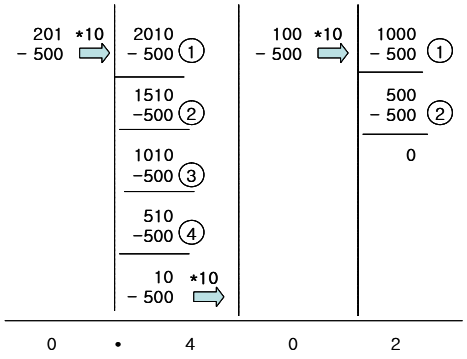


(그림 4) 가수의 곱셈 방법

가수의 곱셈은 가장 왼쪽부터 실행한다. (그림 4)의 가장 오른쪽 바이트는 가장 왼쪽 바이트의 값이 0일 경우 소수점을 오른쪽으로 한번 이동하기 위한 임시 저장 공간이다. 이 경우 지수는 1 감소한다.

(3) 겁수의 나눗셈

정수형 겁수의 나눗셈은 분자에서 분모를 계속적으로 뺄셈한다. 또한 실수형 겁수의 곱셈은 지수와 가수를 분리하여 처리한다. 지수 부분은 분모의 지수에 -1을 곱하여 분자의 지수에 더한 후, 가수의 나눗셈을 수행한다. 정수형 겁수의 나눗셈과 실수형 겁수의 가수 나눗셈은 동일하므로 여기서는 가수의 나눗셈을 예로 든다. 0.201/0.500의 경우 (그림 5)와 같이 수행된다. (그림 5)에서 소수점 왼쪽의 수가 0이 아니면 소수점을 왼쪽으로 한번 이동하고, 지수를 1 증가한다.



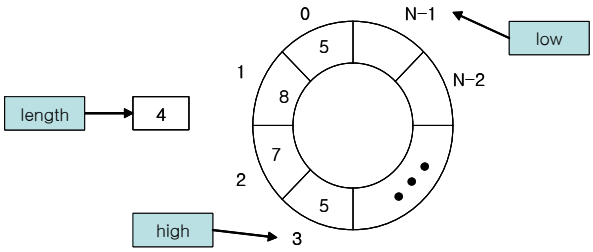
(그림 5) 가수의 나눗셈

3. 정수형 겁수의 사칙연산과 논리연산 구현

본 절은 정수형 겁수를 구현하는 방법과 정수형 겁수(Kint)의 사칙연산 및 논리연산에 대한 설계와 구현을 논의한다. 원시코드는 C++ 언어로 작성하였다.

3.1 정수형 겁수의 구현

정수형 겁수 저장을 위한 문자형 배열은 원형 큐(Circular Queue) 형태를 이용한다.



(그림 6) 숫자 저장을 위한 원형 큐

예를 들어 $(5785)_{10}$는 (그림 6)과 같이 저장한다. high는 최고차 수의 인덱스, low는 최저차 수의 인덱스보다 1적은 수, 숫자의 길이는 4가 된다. 이와 같은 속성을 갖는 정수형 겁수는 Kint 클래스로 정의한다. Kint 클래스 속성은 <표 1>과 같다.

<표 1> Kint 클래스의 속성

속 성	설 명
char sign;	부호
char digit[N];	숫자 저장 배열
long high;	최고차수의 지수(index)
long low;	최저차수의 지수(index)
long length;	숫자의 길이

Kint 클래스의 생성자는 <표 2>와 같다. 정수형, 숫자 배열 형태의 수치를 정수형 겁수로 변환한다.

<표 2> Kint 클래스의 생성자

분 류	원 형
생성자	Kint(); Kint(Kint *ki); Kint(long n); Kint(int n); Kint(char * str);

Kint 클래스의 메소드는 <표 3>과 같다. 사칙연산, 논리연산, 속성 값 반환함수, 절대값 반환, 대소비교, 출력함수가 제공된다.

<표 3> Kint 클래스의 메소드

분 류	원 형
사칙연산	Kint operator+(Kint ki); Kint operator-(Kint ki); Kint operator*(Kint ki); Kint operator/(Kint ki);

논리연산	int operator==(Kint ki);
	int operator!=(Kint ki);
	int operator>(Kint ki);
	int operator>=(Kint ki);
	int operator<(Kint ki);
	int operator<=(Kint ki);

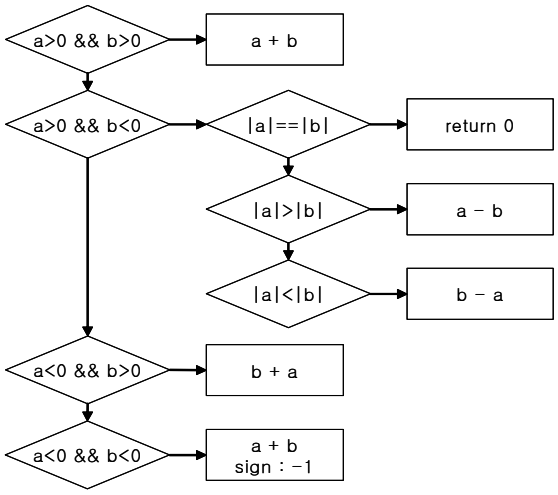
<표 3 계속>

분 류	원 형

기타	long getLength(void);
	long getIndexHigh(void);
	long getIndexLow(void);
	void prt(void);
	Kint absKalpaInt(Kint ki);
	int absKalpaMax(Kint &ki);
	int KiMax(Kint &ki);

3.2 덧셈과 뺄셈의 구현

두 겁수 a와 b의 덧셈 연산은 (그림 7)과 같이 수행된다. 뺄셈은 b의 부호에 -1을 곱한 후 덧셈으로 처리한다. 또한 덧셈과 뺄셈에서 a==0 이면 b를, b==0 이면 a를 반환한다.



(그림 7) 덧셈 흐름도

3.3 곱셈과 나눗셈의 구현

곱셈은 (그림 3)과 동일하게 처리한다. 소스코드는 (그림 8)과 같다.

```
mult.length =  this->length + ki.length - 1;
k = mult.low;
for(i=1; i<=ki.length; i++)
{           b = ki.digit[(ki.low+i)%N];
            carry=0;
            for(j=1; j<=this->length; j++)
            {   a = this->digit[(this->low+j)%N];
                total = a*b + carry + mult.digit[(k+j)%N];
                carry = total/10;       sum = total%10;
                mult.digit[(k+j)%N] = sum;
                mult.high = (k+j)%N;
            }
            if(carry>0)
            {   mult.digit[(k+j)%N] = carry;
                mult.high = (k+j)%N;
            }
            inc(k);
}
if(carry>0) mult.length++;
if(this->sign * ki.sign == 1) mult.sign = 1;
else mult.sign = -1;
return mult;
```

(그림 8) 곱셈 소스코드 일부

나눗셈은 (그림 5)와 유사하게 이루어진다. 소스코드는 (그림 9)와 같다. 한편, 분자가 0이면 0을 반환하고, 분모가 0이면 오류임을 알리는 문장을 출력한다. 또한 분모가 1이면 분자를 반환한다.

```
absDivisor = absKalpaInt(ki);
absDividend = absKalpaInt(*this);

divLen = ki.getLength();
divPoint = this->getIndexHigh();

for(i=0; i<divLen; i++)
{   temp.digit[temp.low] = this->digit[divPoint];
    dec(divPoint);
    dec(temp.low);
    inc(temp.length);
}
quotient.sign = this->sign * ki.sign;
while(1)
{           count = 0;
            while(absDivisor <= temp)
            {           temp = temp - absDivisor;
                        count ++;
            }
            quotient.digit[quotient.low] = count;
            dec(quotient.low);
            inc(quotient.length);
            if(divPoint == this->low) break;
            temp.digit[temp.low] = this->digit[divPoint];
            dec(divPoint);
            dec(temp.low);
            inc(temp.length);
}
return quotient;
```

(그림 9) 나눗셈 소스코드 일부

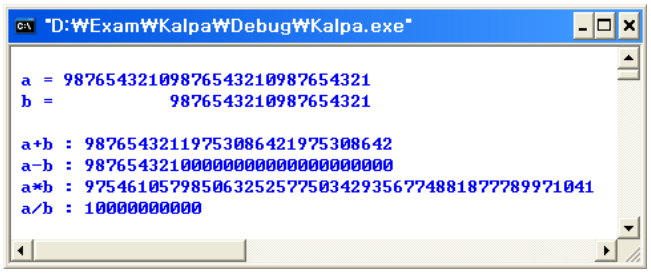
3.4 논리연산 구현

겁수의 사칙연산을 위하여 논리연산이 필요하다. 구현된 논리연산은 <표 3>과 같으며, 두 겁수 a, b의 논리연산은 <표 4>와 같이 처리한다.

<표 4> 논리연산의 처리

논리연산	처리내용
a==b	참이면 1, 거짓이면 0 반환
a!=b	return !(a==b)
a>=b	참이면 1, 거짓이면 0 반환
a>b	참이면 1, 거짓이면 0 반환
a<=b	참이면 1, 거짓이면 0 반환
a<b	참이면 1, 거짓이면 0 반환

두 겁수 a, b의 사칙연산 실행 예는 (그림 10)과 같다.



(그림 10) 두 겁수의 사칙연산

4. 결론

수학, 물리학, 전자전기공학에서 취급하는 수치 자료는 매우 크거나 정밀도가 높을 수 있고, 이 경우 계산과정에서 흔히 오버플로우나 언더플로우가 발생한다. 본 연구는 이런 현상을 최소화하기 위하여 겁수(Kalpa Number) 표현 방법을 제안하고, 정수형 겁수의 사칙연산과 논리연산을 구현하였다. 정수형 겁수는 수의 부호와 크기를 분리하여 부호는 하나의 문자형 변수에 저장하고, 수의 크기는 문자형 배열에 저장한다. 문자형 배열은 원형 큐를 이용하였다.

정수형 겁수의 연산을 위하여 사칙연산, 논리연산, 기타의 연산을 구현하였다. 사칙연산은 종이에 연필을 사용하여 계산하는 것과 유사한 방법으로 처리한다.

실수형 겁수의 연산이나, 삼각함수, 지수함수 등에 필요한 연산 구현은 추후 제시할 것이다. 겁수를 이용할 경우 실행속도는 느리지만 매우 큰 수의 연산이 가능하다.

참고문헌

[1] 권영빈, 김준년, 컴퓨터공학, pp.4~25, 대영사, 1988.

[2] http://www.bernoulli.org/

[3] Joung Young Sug, at al. "Quantum transition processes in deformation potential interacing systems using the equilibrium density projection technique", Physical Review B vol. 64, 2001.

[4] http://www.jjj.de/hfloat/

[5] http://www.apfloat.org/apfloat/

[6] http://www.ginac.de/CLN/

[7] 김귀연, 김병기, 최준용, "매우 큰 수치 데이터 처리를 위한 겁수(Kalpa Number) 표현방법", KFIS 2007년 춘계학술대회, 한국 퍼지 및 지능 시스템학회, 2007. 4.