論 文
56-2-31

MATLAB을 이용한 부동소수점 연산의 특이사항 분석

Analysis of Some Strange Behaviors of Floating Point Arithmetic using MATLAB Programs

鄭 台 相†
(Tae-Sang Chung)

Abstract - A floating-point number system is used to represent a wide range of real numbers using finite number of bits. The standard the IEEE adopted in 1987 divides the range of real numbers into intervals of $[2^E, 2^{E+1})$, where E is an integer represented with finite bits, and defines equally spaced equal counts of discrete numbers in each interval. Since the numbers are defined discretely, not only the number representation itself includes errors, but in floating-point arithmetic some strange behaviors are observed which cannot be agreed with the real world arithmetic. In this paper, errors with floating-point number representation, those with arithmetic operations, and those due to order of arithmetic operations are analyzed theoretically with help of and verification with the results of some MATLAB program executions.

Key Words : Floating Point Number System, Error Analysis, Rounding, MATLAB

1. 서 론

Computer의 유한한 binary 자릿수에 넓은 범위의 실수를 표현하기 위하여 부동소수점 수 표현법, 즉 FPNS(Floating Point Number System)을 사용한다. 무한의 실수를 유한 자리수의 binary로 표현하여야 함으로, 수 표현상에 내재된 오차가 있으며, 이러한 수의 연산에도 오차가 따른다. 부동소수점 연산의 오차/오류에 의한 연구 분야와 산업체에 심대한 영향을 미친 사건들이 참고 문헌 [1,2]에 기술되어 있다. 부동소수점에 관련하여서는 여러 컴퓨터회사가 독자적인 표현법으로 사용하여 왔는데 표현법의 차이로 같은 프로그램도 이를 기계언어로 컴파일하는 컴파일러의 차이, 그리고 실행하는 컴퓨터 기종과 word의 bit 수가 다름에 따라 약간씩 다른 결과를 주는 등의 불편함이 있었으나, 1985년 미국 표준국과 IEEE가 ANSI-IEEE Std 754-1985라는 32 bit와 64 bit의 FPNS 표준을 제시하였다[3]. ANSI/IEEE가 부동소수점의 연산에 관한 표준을 제시하여 범용성과 오차에 대한 처리 방법들이 표준화 되면서 새로운 하드웨어와 소프트웨어의 제작이 가능하게 되어 Intel CPU 등 현대의 범용 CPU들과 C, C++, MATLAB[4] 등의 프로그램 언어들이 이 표준을 따른다.

MATLAB 프로그램 언어는 32 bit의 single precision과 64 bit의 double precision의 FPN를 지원하지만, 명시적으로 규약하지 않는 경우 default로 64 bit의 double precision의 수로 변수를 표현한다. 본 논문에서는, 64-bit double-precision FPN의 표현법과 이산화에 따른 오차를 분석하고, 또 FPN의 연산에 따른 오차가 수학적 예측과 반대의 결과로 증폭 혹은 축소되는 예들을 컴퓨터 연산이론에 의하여 분석하며, 수학적으로 동일한 결과가 부동소수점의 연산 순서에 따라 달라지는 특이사항을 분석하고, MATLAB의 statement와 집행 결과를 10진수와 hexadecimal로 함께 출력하여 이론적 분석을 검증한다.

† 교신저자, 正會員 : 중앙대학교 전자전기공학부 교수
E-mail : tschung@digital.cau.ac.kr
接受日字 : 2006年 11月 20日
最終完了 : 2006年 11月 21日

2. 본 론

2.1 부동소수점 수 표현법

1985년 제정된 ANSI/IEEE Std 754-1985는 single precision의 32 bit 혹은 double precision의 64 bit를 사용하는 두 종류의 FPN를 규정하였다. 수 표현은 아래와 같이 세 부분으로 나누어지는데, 부호 (sign), 지수(exponent), 그리고 가수(mantissa) 혹은 유효숫자(significand)로 구분되어 있다.

Sign	지수 Biased Exponent	가수 Significand s = 1.F (1 is hidden)
±	E+bias	F

최상위의 1 bit는 수 전체의 sign으로 1이면 음수 0이면 양수를 뜻한다. 지수는 single precision의 경우는 8 bit의 127 초과 코드를, 그리고 double precision의 경우는 11 bit의 1023 초과 코드를 사용한다. 그리고 가수부분은 각각 23 bit와 52 bit를 사용한다. 이 구분에 의하여 부동소수점 수는 부호, 지수, 그리고 가수부분에 의하여 $\pm 1.F \times 2^E$로 해석되며, 같은 가수의 표현에 지수가 달라지면, 2진수의 bit 배열은 같고, 소수점의 위치만 옮겨진다. 예를 들어 지수 $E=-1, 0, 1, 2$에 대하여 FPN은 2진법으로 각각 $0.1\,f_{-1}f_{-2}\cdots f_{-52}$, $1.f_{-1}f_{-2}\cdots f_{-52}$, $1\,f_{-1}.f_{-2}\cdots f_{-52}$, 그리고 $1\,f_{-1}f_{-2}.f_{-3}\cdots f_{-52}$가 된다. 이렇게 지수에 따라 소수점이 옮겨 다니는 이유로 이러한 수 표현법을 부동소수점(浮動小

數點) 수 표현법(Floating Point Number System)이라고 한다. 지수부분과 가수부분의 중복해석을 피하는 편의상 본 논문에서는 64 bit double precision의 경우에 대하여만 설명한다.

Double precision의 11 bit의 지수부분은 2진수로 0부터 2047까지의 수이나, 이를 1023 초과 코드로 해석하여 지수는 -1023에서 1024까지의 범위가 된다. 이 중 양쪽 끝의 두 지수 -1023과 1024를 특수 목적에 사용하기 위하여 제외하고, -1022부터 1023까지를 2의 승수를 표현하는 지수로 사용한다. 지수 E에 따라 양의 실수 범위를 $[2^E, 2^{E+1})$의 연속적 구간으로 나눌 수 있는데, 따라서 전 표현 구간은 $[2^{-1022}, 2^{1024})$가 된다. 예를 들어 $E=-1, 0, 1, 2$인 경우는 각각 구간 $[0.5, 1)$, $[1, 2)$, $[2, 4)$, $[4, 8)$을 표현 한다.

지수에 의하여 구분되는 실수의 전구간은 $[2^E, 2^{E+1}) = [1, 2)\times 2^E$의 형태로 변형할 수 있는데, E는 지수부분이 표시하므로, $[1, 2)$ 범위의 수를 추가로 규정하여야 하는데, 가수부분이 이 역할을 담당한다. 가수가 1과 2 사이의 값(1은 포함하고 2는 포함하지 않음)이므로, 1을 제외한 소수점 이하의 수가 52 bit로 표현되며, 정수부 1은, 항상 있는 것이므로, 52 bit 표현 중에 포함하지 않고 숨겨진 것으로 간주한다. 가수는 통상 1.F 형태로 해석하는데, 2진법으로 $1.f_{-1}f_{-2}\cdots f_{-52}$의 형태가 되며 정밀도는 2^{-52}이 된다. 이 가수는 52 bit가 전부 0인 경우는 1.0이 되며, 52 bit 전부가 1이 되는 경우의 $2-2^{-52}$가 되므로, $1 \le 1.F \le 2-2^{-52}$의 범위에서 2^{-52}의 등 간격으로 2^{52}개, 즉 약 4,504조의 숫자를 표현하게 된다. FPN의 유효 숫자는 2진법으로는 숨겨진 1을 포함하여 53자리로 고정되지만, 수 표현의 간격이 $2^{-52} \approx 2.22\times 10^{-16}$이 되므로 1과 2사이의 FPN 수는 10진법으로 소수점 이하 17번째 이하의 정밀도는 FPN 표현에 아무런 영향을 미칠 수 없으므로, FPN은 10진법의 자연과학적 표현법(scientific notation)상 정수부를 포함하여 17 자리의 유효숫자를 가지게 된다.

2.2 가감 연산에 있어서의 특이 사항 분석

부동소수점 수의 가감산의 특이사항의 예제로 사용하기 위하여 10진수 0.2와 8의 FPN 표현법을 구하여 보자. 0.2를 2진법으로 표현하기 위하여 0.2에 연속적으로 2를 곱하는 과정에 나오는 정수부를 제거하면서 계속 곱하기를 하면 나오는 정수부는 0011을 반복한다. 따라서 0.2는 2진수에서 소수점 이하에서 0011을 반복하는 순환소수가 되며, 이를 1보타 크거나 같은 가수로 표현하기 위하여 2^{-3}을 곱하고 소수점을 3자리 왼쪽으로 shift하여 표현하면, 소수점 이하에서 1001을 반복하는 순환소수가 된다. 이를 소수점 이하 52자리까지로 반올림에 의한 이산화 하는 과정은 다음과 같다:

$$
\begin{aligned}
0.2_{10} &= 0.\underbrace{0011}_{}\,\underbrace{0011}_{}\cdots \\
&= 1.\underbrace{1001}_{1}\,\underbrace{1001}_{2}\cdots\underbrace{1001}_{13}\,\underbrace{1001}_{14}\cdots\times 2^{-3} \\
&\Rightarrow 1.\underbrace{1001}_{1}\,\underbrace{1001}_{2}\cdots\underbrace{1010}_{13}\times 2^{-3}
\end{aligned}
$$

0.2의 FPN 표현은 가수 부분이 소수점 이하 53번째가 1일 뿐만 아니라 그 이하에도 0이 아닌 수가 반복적으로 나타나는데, 소수점 53번째 이하 무한대까지의 합은 2^{-53}보다 크므로 2^{-52}로 반올림이 되어, 전체적으로 0.2보다 크게 표현됨을 알 수 있다. FPN(x)를 실수 x에 대한 부동소수점 표현법이라고 하고, Err(x)를 FPN(x)의 실수 x에 대한 이산화 오차로 정의하여 보자. 위의 0.2에 대한 이산화 결과에 의하여 $FPN(0.2) > 0.2$이 성립하고 이산화 오차의 범위 $Err(0.2) = FPN(0.2) - 0.2 < 2^{-53}$이 됨을 알 수 있다. 지수 -3의 1023 초과 코드는 $1020_{10} = 01111111100_2$이 되고, 숨겨지는 1을 제외하고, 양수부호(0), 지수, 가수를 조합하면 FPN(0.2) 표현은 아래와 같으며, 이의 16진법 표현법은 3F,C9,99,99,99,99,99,99,9A가 된다.

$$
0.2_{10} \Rightarrow \underbrace{0}_{s}\,|\,\underbrace{01111111100}_{exp+1023}\,|\,\underbrace{1001}_{1}\cdots\underbrace{1001}_{12}\,\underbrace{1010}_{13}
$$

정수 8은 1.0×2^3이 되므로 오차 없이 정확히 표현되는 수로 $FPN(8) = 8$과 $Err(8) = 0$이 된다. 지수 3에 대한 초과 1023 코드는 $1026_{10} = 10000000010_2$ 이며 숨겨진 1을 제외한 52 bit의 가수부는 전부 0이 된다. 따라서 FPN 표현법은 16진법으로 40,20,00,00,00,00,00,00이 될 것이다. 다음의 MATLAB 문장이 위의 conversion을 확인하여 준다. MATLAB에서 0.2를 소수점이하 18자리까지 10진법으로 프리트하면 내부적으로 0.2보다 크게 표현됨을 알 수 있고, hexadecimal로 프린트 하면 위에서 수작업으로 conversion한 값과 같음을 알 수 있다.

```
fprintf('  %20.18f   %bx\n', 0.2, 0.2)
fprintf('  %20.18f   %bx\n', 8, 8)
```

```
  0.200000000000000010   9a9999999999c93f
  8.000000000000000000   0000000000002040
```

먼저 0.2와 8의 FPN 표현법을 이용하여 FPN 연산의 특이 사항을 아래 프로그램을 이용하여 관찰하여 보자. 오차 없이 정확히 표현되는 8에 0.2보다 크게 표현되는 FPN(0.2)을 더하면 8.2보다 큰 수가 되는 것이 수학적 결론이지만, FPN의 연산에서는 이것이 부주의한 결론이 됨을 아래 프로그램의 실행 결과로 알 수 있다. 이 프로그램의 첫 번째 문장의 해석상 주의할 점, 혹은 컴파일 순서는, 변수 a에 8+0.2 즉 십진으로 계산된 8.4의 FPN이 배정되는 것이 아니고, 프로그램 언어의 컴파일 기법상, 8의 FPN을 먼저 구하고, 다음에 0.2의 FPN을 구한 후, FPN(8)+FPN(0.2)를 하고, 합산한 결과의 FPN을 구하여 변수 a에 배정하는 것이며, 두 번째 문장은, FPN 표현법인 변수 a의 값에 0.2의 FPN을 구하여 합산한 결과의 FPN 값을 변수 b에 배정하는 것이다.

```
a = 8 + 0.2;
b = a + 0.2;
fprintf('  %20.18f   %bx\n', a, a)
fprintf('  %20.18f   %bx\n', b, b)
```

```
  8.199999999999999300   6666666666662040
  8.399999999999998600   cccccccccccc2040
```

0.2의 FPN이 0.2보다 큰 수로 표현됨을 알았지만, 이를 오차 없이 표현되는 8의 FPN에 연속적으로 더하는 경우, 그 결과가 각각 8.2과 8.4와의 대소 관계가 수학적 결론과 달라지는 이유를 다음과 같이 분석할 수 있다. 수 8은 $[8, 16) = [1, 2)\times 2^3$의 구간에 속하는 수로서 지수가 3인 반면에, 0.2는 구간 $[0.125, 0.25) = [1, 2)\times 2^{-3}$에 속하여 지수가 -3이 되며, 따라서 두 수를 합산하기 위하여 지수를 3 (3과 -3 중의 큰 수)으로 통일하고, 0.2의 가수를 (숨겨진 1을 포함하여) 두 지수의 차이 6=3-(-3) 자리만큼 오른쪽으로 shift한 후에 합산을 하여야 한다. 이 과정이 다음의 식에 나타나 있다:

$$
\begin{aligned}
8_{10} &= 1.\underline{0000}\,\underline{0000}\cdots\underline{0000}\,\underline{0000} && \times 2^3\\
0.2_{10} &\Rightarrow 1.\underline{1001}\,\underline{1001}\cdots\underline{1001}\,\underline{1010} && \times 2^{-3}\\
&= 0.\underline{0000}\,\underline{0110}\cdots\underline{0110}\,\underline{0110}\,\underline{011010}\times 2^3 && > 0.2\\
8_{10}+0.2_{10} &\Rightarrow 1.\underline{0000}\,\underline{0110}\cdots\underline{0110}\,\underline{0110}\,\underline{011010}\times 2^3 && > 8.2\\
&\Rightarrow 1.\underline{0000}\,\underline{0110}\cdots\underline{0110}\,\underline{0110} && \times 2^3 < 8.2
\end{aligned}
$$

위 식에서 011010은 0.2의 가수 부분이 소수점 이하 52번째에서 오른쪽으로 shift되어 나오는 bit들이며, 이 bit들을 포함한 0.2의 가수와 8의 가수를 합산한 결과는 당연히 8.2보다 큰 수가 될 것이다. 그러나 가수의 합산 결과에서 53번째 bit이 0이므로 $1.f_{-1}f_{-2}\cdots f_{-52}$ 형태로 소수점이하 52자리까지 이산화 하는 과정에 53번째 이하의 6 bit는 절삭되며, 이 때문에 합산의 결과가 오히려 8.2보다 작아진다. 연산 FPN(FPN(8)+FPN(0.2)) 결과를 부호, 지수, 가수 부분을 조합하여 16진법으로 표시하면 40,20,66,66,66,66,66,66이 됨을 쉽게 확인할 수 있으며, 이는 위 프로그램의 프린트 결과와 일치한다. 위 프로그램의 실행 결과에서 보듯이 8의 FPN에 0.2보다 큰 FPN(0.2)을 2번 연속으로 순차적으로 더하여도 8.4보다 작은 결과가 되는 사실도 같은 방법으로 분석/확인할 수 있다.

위에서 0.2보다 큰 수 FPN(0.2)를 8에 더하면 오히려 0.2보다 적은 값이 더하여 지는 예를 분석하였지만, 이것이 일관된 패턴은 아니다. FPN(0.2)을 0.2의 지수와 4의 차이가 나는 구간 [2, 4)에 속하는 수, 예를 들어 3에 더하면 FPN(0.2)보다 더욱 큰 수를 더하는 효과가 됨을 다음의 프로그램에서 확인할 수 있다. 이에 대한 분석이 뒤 따르는데, 위에서와 같은 방법으로 해석할 수 있다.

```
fprintf(' %20.18f   %bx\n', 3+0.2, 3+0.2)
   3.20000000000000200   9a99999999990940
```

$$
\begin{aligned}
3_{10} &= 1.\underline{1000}\,\underline{0000}\cdots\underline{0000}\,\underline{0000} && \times 2^1\\
0.2_{10} &\Rightarrow 1.\underline{1001}\,\underline{1001}\cdots\underline{1001}\,\underline{1010} && \times 2^{-3}\\
&= 0.\underline{0001}\,\underline{1001}\cdots\underline{1001}\,\underline{1001}\,\underline{1010}\times 2^1 && > 0.2\\
3_{10}+0.2_{10} &\Rightarrow 1.\underline{1001}\,\underline{1001}\cdots\underline{1001}\,\underline{1001}\,\underline{1010}\times 2^1 && > 3.2\\
&\Rightarrow 1.\underline{1001}\,\underline{1001}\cdots\underline{1001}\,\underline{1010} && \times 2^1 \gg 3.2
\end{aligned}
$$

2.3 부동 소수점 연산 순서에 따른 특이 사항 분석

앞의 프로그램 예에서 오차 없이 표현되는FPN(8)=8에 0.2보다 크게 표현되는 FPN(0.2)을 두 번 연속으로 합산하면 수학적 결과에 배치되게 각각 8.2와 8.4보다 작은 수로 표현되는 특이 사항을 분석하였다. 그런데 계산 순서를 바꾸어 먼저 FPN(0.2)+FPN(0.2)을 집행하고 이를 8.0에 합산하면 그 결과는 8.4보다 큰 수가 되어 8에 FPN(0.2)를 계속 두 번 합산하는 결과와 다르게 되는 특이사항이 발생한다. 다음의 MATLAB 프로그램에서 이런 현상을 확인하고, 이를 연산 이론적으로 분석하여 보자. 이 프로그램에서 "fprintf" function의 상수 parameter인 (8+0.2)+0.2와 8+(0.2+0.2)에 대한 주의가 필요하다. 이 상수들은 10진수 연산의 결과인 8.4의 FPN을 parameter로 사용하는 것이 아니고 8과 0.2의 FPN을 기술된 순서와 괄호에 의한 순서에 따라 FPN 연산을 한 결과를 function call에 parameter로 보내고 있으며, 따라서 두 결과는 같지 않다.

```
fprintf('0.2+0.2          %20.18f  %bx\n', ...
                                  0.2+0.2, 0.2+0.2)
fprintf('(8+0.2)+0.2      %20.18f  %bx\n', ...
                                  0.2)+0.2, (8+0.2)+0.2)
fprintf('8+(0.2+0.2)      %20.18f  %bx\n', ...
                                  8+(0.2+0.2), 8+(0.2+0.2))
```

0.2+0.2	0.400000000000000020	9a9999999999d93f
(8+0.2)+0.2	8.399999999999998600	cccccccccccc2040
8+(0.2+0.2)	8.400000000000000400	cdcccccccccc2040

$$
\begin{aligned}
8_{10} &= 1.\underline{0000}\,\underline{0000}\cdots\underline{0000}\,\underline{0000} && \times 2^3\\
0.2_{10}+0.2_{10} &\Rightarrow 1.\underline{1001}\,\underline{1001}\cdots\underline{1001}\,\underline{1010} && \times 2^{-2}\\
&= 0.\underline{0000}\,\underline{1100}\cdots\underline{1100}\,\underline{1100}\,\underline{11010}\times 2^3 && > 0.4
\end{aligned}
$$

$$
\begin{aligned}
8_{10}+(0.2_{10}+0.2_{10}) &\Rightarrow 1.\underline{0000}\,\underline{1100}\cdots\underline{1100}\,\underline{1100}\,\underline{11010}\times 2^3 && > 8.4\\
&\Rightarrow 1.\underline{0000}\,\underline{1100}\cdots\underline{1100}\,\underline{1101} && \times 2^3 \gg 8.4
\end{aligned}
$$

앞의 2.2항에서의 분석에 의하면 FPN(8)과 FPN(0.2)의 지수가 각각 3과 -3이 되어 차이가 6이므로, 두 수를 더할 때 지수를 통일하기 위하여 FPN(0.2)의 가수부분의 하단 6 bit, 즉 011010이 shift right 되어 나오는데 합산 후 소수점 이하 53번째 자리의 값이 0이므로 52 bit만을 취할 때 6 bit 전체가 절삭되어, 결과적으로 8.2보다 작아짐을 확인하였다. FPN(0.2)를 다시 합산하여도 똑같은 과정을 거쳐 truncation이 일어나서 8.4보다 작은 결과가 됨을 알 수 있다. 그러나 먼저 FPN(0.2)를 두 번 합산한 결과를 8에 더하는 경우에는 상황이 달라진다. FPN(0.2)+FPN(0.2)을 실행하면 가수부분은 FPN(0.2)와 같고, 지수만 FPN(0.2)의 지수보다 1이 증가하여 -2가 되므로, 결과는 2*FPN(0.2)가 되어 0.4보다 큰 수가 된다. FPN(8)의 지수와 FPN(0.2)+FPN(0.2)의 지수의 차이는 5가 되므로, 합산 시 지수를 통일하기 위하여 FPN(0.2)+FPN(0.2)의 가수부분을 5 bit shift right 하여야 하는데, 11010이 shift out 된다. 합산 후 소수점 이하 52 bit만을 취함에 있어서, 53번째 자리의 값이 1이고 그 이하의 값이 0이 아니므로 절삭하지 않고 반올림이 일어나며, 결과적으로 8.4보다 더욱 커짐을 알 수 있다.

2.4 MATLAB 프로그램 언어에 있어서 FPN에 관련한 명제의 Compile 방법에 따른 영향

위 2.2항과 2.3항에서 부동소수점 연산의 결과가 수학적 결론과 배치되기도 하고, 또한 수학적으로 동일한 사항이

연산의 집행 순서에 따라 결과에 차이가 남을 이론적으로 분석하였다. 예제의 결과를 종합하면 다음과 같다.

$$\begin{cases} FPN(8) & =8 \\ FPN(0.2) & >0.2 \\ FPN(FPN(8)+FPN(0.2)) & <8.2 \\ FPN([FPN(8)+FPN(0.2)]+FPN(0.2)) & <8.4 \\ FPN(FPN(8)+[FPN(0.2))+FPN(0.2)]) & >8.4 \end{cases}$$

다음의 세 개의 간단한 MATLAB 프로그램들을 비교/분석하여 보자. 세 개의 프로그램은 형태는 다르지만 8부터 9까지의 사이에서 0.2의 등간격으로 데이터를 정의하는 프로그램으로 쉽게 간주할 수 있다. 첫 번째 것은 일반적으로 C 언어에서 등간격의 데이터를 작성하는 예에서 흔히 보는 프로그램으로, 8에서부터 계속하여 0.2씩 더하여 데이터가 정의될 것이다. 두 번째는 MATLAB 언어의 matrix 혹은 array 데이터를 정의하는 프로그램으로, 첫 번째와 동일한 데이터가 정의될 것으로 추측할 수 있다. 그런데 두 프로그램에서의 정의된 데이터를 10진수와 hexadecimal로 프린트한 결과를 비교하면 이 추측이 맞지 않는다는 것을 알 수 있다.

첫 번째 프로그램의 실행에서 보듯이 8.4에 대하여 8에 FPN(0.2)>0.2를 계속하여 두 번 합산하면, 그 결과는 8.4보다 작아짐을 2.2항에서 분석하였다. 그런데 두 번째 프로그램에서 보듯이 8.4가 FPN에서 8.4보다 크게 정의되는데 이는 8에서 0.2간격씩 두 번 떨어져 있는 숫자는 0.2를 두 번 먼저 더한 후에 8에 더하였음을 2.3하의 결과로 알 수 있다. 두 번째 프로그램과 세 번째 프로그램은 같은 결과를 주는데, 세 번째 프로그램은 먼저 0.2를 0부터 5가지의 숫자에 각각 곱하여 그 결과를 8에 각각 합산한 것이다.

```
a(1) = 8.0;
for k = 1:5
  fprintf(' %20.18f  %bx\n',a(k),a(k))
  a(k+1) = a(k) + 0.2;
end
```

8.000000000000000000	0000000000002040
8.199999999999999300	6666666666662040
8.399999999999998600	cccccccccccc2040
8.599999999999997900	3233333333332140
8.799999999999997200	9899999999992140

```
b = 8:0.2:9;
for k = 1:5
  fprintf(' %20.18f  %bx\n',b(k),b(k))
end
```

8.000000000000000000	0000000000002040
8.199999999999999300	6666666666662040
8.400000000000000400	cdcccccccccc2040
8.599999999999999600	3333333333332140
8.800000000000000700	9a99999999992140

```
ind = 0:4;
c = 8 + 0.2*ind;
for k = 1:5
  fprintf(' %20.18f  %bx]n',c(k),c(k))
end
```

8.000000000000000000	0000000000002040
8.199999999999999300	6666666666662040
8.400000000000000400	cdcccccccccc2040
8.599999999999999600	3333333333332140
8.800000000000000700	9a99999999992140

위 프로그램의 분석에 의하면 MATLAB에서 a:b:c의 데이터 정의 문장은 a에서 c까지 b의 간격으로 정의하는 데이터는 a에 연속적으로 간격 b를 더하여 정의하는 것이 아니고, b를 1부터 시작하여 필요한 정수까지의 수열에 b를 먼저 곱한 후에 이를 a에 각각 더함으로 이루어짐을 알 수 있다.

3. 결 론

최근 IT의 발달로 인하여 FPN processor의 필요성이 증대되었으며, 따라서 현장에서 특화된 IT 제품개발에 바로 적용하기 위하여, 그리고 32/64 bit의 높은 정밀도의 FPN를 대신에 16 bit의 정밀도를 요하는 축약된 FP-processor를 설계하는 등의 이유로, FPN과 이의 연산, 그리고 연산에 수반되는 특이사항들에 대한 완전한 이해가 필요하게 되었다. 본 논문에서는 실수를 FPN로 표현함에 있어서 이산화 과정에서 발생하는 특이사항과 FPN의 연산에 관련한 특이 사항, 그리고 연산순서에 따른 특이사항, MATLAB 언어의 특수 문장에 있어서의 내부적 계산 순서 등을 MATLAB 프로그램의 간단한 예로써 이론적으로 분석하였으며, 이를 통하여 FPN에 관련한 프로그램의 작성에 주의를 환기시키고 특성화된 FPN processor 설계에 도움이 되고자 하였다. 사용한 예들이 FPN과 관련한 특이사항의 모두를 수집하거나 지적한 것은 아니지만 이론적 분석과 연습의 기초가 되기 위한 것이다.

참 고 문 헌

[1] Moler, C., " A Tale of Two Numbers," SIAM News, Vol. 28, No. 1, pp. 1, 16, 1995.

[2] http://www.math.psu.edu/dna/455.f96/disasters.html

[3] IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985), IEEE Press, 1985

[4] The Student Edition of MATLAB, User's Guide, Prentice Hall, Englewood Cliffs, NJ 1995

[5] Kuck, D. J., D. S. Parker and A. H. Sameh, "Analysis of Rounding Methods in Floating-Point Arithmetic," IEEE Trans. Computers, Vol. 26, No. 7, pp. 643-50, 1977