운영체제- 201921725 안성현

패트리어트 미사일 오차

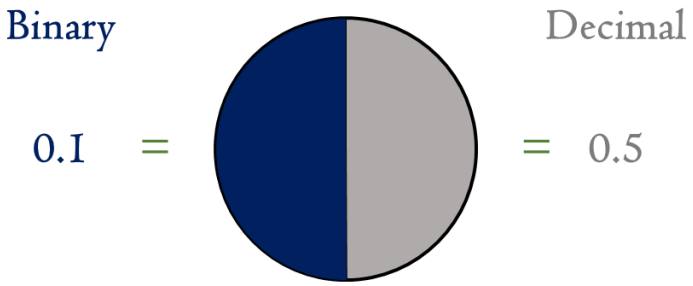
<2021/03/30>

가톨릭대학교 1

안녕하세요. 컴퓨터정보공학부 19학번 안성현입니다.
저는 '미국 패트리어트 미사일 오차'에 대한 이야기를 하려고 합니다. 대표적인 소수점 오차로 인한 실제 사고인데요. 컴퓨터공학도라면 한 번씩은 들어본, 그리고 한 번씩은 상기해봐야 될 이야기라고 생각해서 이렇게 발표를 준비해봤습니다.

부동소수점

- 이진법
  - 숫자 0과 1을 이용해서 수를 표현하는 방법
  - 이진법 0.1 = 십진법 0.5

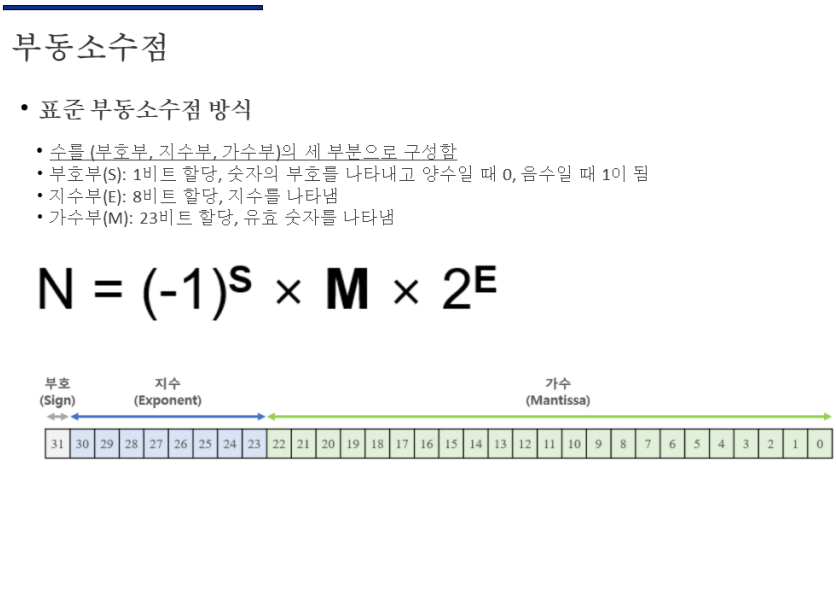


가톨릭대학교 2

미사일 오차에 대한 이야기를 하기 전에 컴퓨터가 어떻게 숫자를 저장하는지부터 살펴보겠습니다. 컴퓨터에서 숫자는 특별한 소프트웨어적 처리가 없으면 이진법에 기반한 방법으로 저장됩니다. 일반적으로 floating point 부동소수점 방법으로 숫자를 저장하게 됩니다.

이진법과 십진법, 부동소수점 방법은 다들 아실거라고 생각합니다. 이진법은 숫자 0과 1을 이용해서 수를 표현하는 방법이고 십진법은 숫자 0~9 10개를 이용해서 수를 표현하는 방법입니다. 그래서 이진수 0.1을 2번 더하면 1이 되고 십진수는 0.1을 10번 더해야 1이 됩니다. 여기서 1이라는 숫자는 완전한 수, 전체를 의미하잖아요. 그래서 이진수 0.1은 십진수 0.5와 같습니다.

(0.1을 1번 더하면 0.1, 1번 더 더하면 1이 됨. 왜냐하면 2라는 개념이 없으므로<숫자가 0과 1뿐이므로> 다음 1로 넘어가는 구조)



부동소수점 방식도 간단하게 리뷰 해보겠습니다. 부동소수점 표현에서 수는 (부호부,지수부,가수부)의 세 부분으로 구성됩니다. 부호부(S)는 1비트가 할당되며, 숫자의 부호를 나타내고 양수일 때 0, 음수일 때 1이 됩니다. 지수부(E)는 8비트가 할당되며, 지수를 나타냅니다. 가수부(M)는 23비트가 할당되며, 유효숫자를 나타냅니다. 그래서 총 32비트로 수를 표현하게 됩니다.

부동소수점

- 부동소수점 방식 표현(-314.625)
  - 부호부(S): 부호가 음수이므로 32비트의 가장 앞자리는 1

  - 가수부(M): 소수점을 이동시켜서 소수점 왼쪽에 1만 남도록 함 (정규화) -> 소수점의 오른쪽 부분을 23비트의 앞에서부터 채움

$100111010.101_{(2)}$

$100111010.101_{(2)} = 1.00111010101 \times 2^8$

$1.00111010101_{(2)}$

예를 들어 -314.625를 부동소수점 방식으로 표현해 보겠습니다.

첫 번째로 부호부 처리 방법입니다. 부호가 음수이므로 32비트의 가장 앞자리는 1이 됩니다.

두 번째로 가수부 처리 방법입니다. 314.625를 2진수로 나타내면 100111010.101(2)이 됩니다. 여기서 2진수의 소수점을 아래와 같이 왼쪽으로 이동시켜서 소수점 왼쪽에 1만 남도록 합니다. 이렇게 소수점 왼쪽에 1만 남도록 한 표현을 정규화된 표현이라고 합니다. 그리고 소수점의 오른쪽 부분을 가수부 23비트의 앞에서부터 채워줍니다. 남은 자리는 0으로 채워지게 됩니다.

부동소수점

- 부동소수점 방식 표현(-314.625)
  - 지수부(E): 정규화 식의 지수에 bias인 127을 더함 -> 2진수 변환 -> 8비트의 지수부에 채움

$100111010.101_{(2)} = 1.00111010101 \times 2^8$

$8 + 127 = 135$

$10000111_{(2)}$

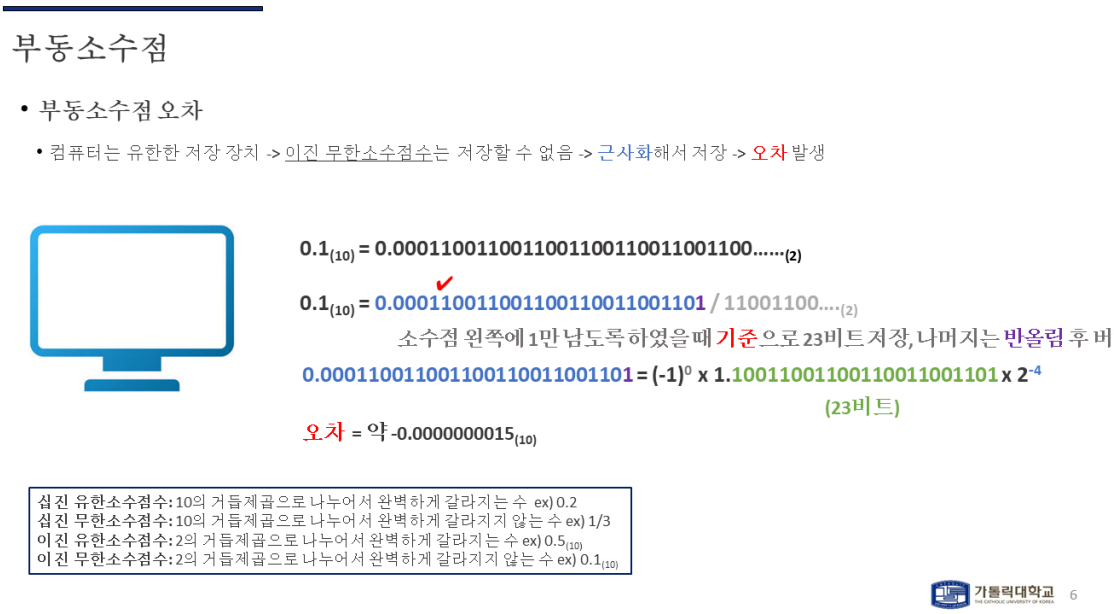
$N = (-1)^S \times M \times 2^E$

$-314.625_{(10)} = 100111010.101_{(2)}$

$100111010.101 = (-1)^1 \times 1.00111010101 \times 2^8$

마지막으로 지수부 처리 방법입니다. 가수부 처리할 때 구한 정규화 식에서 지수는 8이 됩니다.

이 지수에 bias인 127을 더해줍니다. 이렇게 더해진 수를 2진수로 변환하고 8비트에 채워주면 됩니다.



이 부동소수점 방식에서 우리가 주의 깊게 바라봐야 될 부분이 있습니다. 바로 컴퓨터가 유한한 저장 장치를 가지고 있기 때문에 수를 32비트로 저장한다는 것이고 이 말은 이진 무한소수점수를 저장할 수 없다는 의미가 됩니다. 결국 이진 무한소수점수를 저장할 때는 정규화된 표현 기준으로 23비트를 저장하고 남은 뒷자리는 반올림 후 버리게 됩니다.

이진 무한소수점수인 0.1 =0.0001100[1100](2) 을 예시로 하면 오차는 십진수로 약 -0.0000000015가 된다고 합니다. 이렇게만 보면 아주 작은 수에 불과한데 이런 작은 수가 어떤 참극을 불러 일으켰는지.. 지금부터 패트리어트 미사일 오차에 대한 이야기를 해보겠습니다.

패트리어트 미사일 오차

- MIM-104 패트리어트 미사일
  - 1990년대 적의 스커드 미사일과 근거리 탄도 미사일을 요격하는데 사용된 미사일
  - 아래의 단계를 거쳐서 적의 미사일을 추적하고 파괴한다

1) 레이더가 속도, 위도, 경도, 방위각 등의 정보를 기반으로 미사일의 특징을 가진 공중 물체를 찾아낸다.
2) 레이터 시스템 내의 '레인지 게이트'가 미사일이 다음에 나타나게 될 영공의 구역을 계산한다.
3) 적의 미사일이 계산된 구역에 들어오면 패트리어트가 자신의 미사일을 발사한다.

미국에 MIM-104라고 하는 패트리어트 미사일이 있습니다. 이 미사일은 1990년대 적의 스커드 미사일과 근거리 탄도 미사일을 요격하는데 사용이 되었다고 합니다. 이 패트리어트 시스템은 아래의 단계를 거쳐서 적의 미사일을 추적하고 파괴합니다.

첫 번째, 레이더가 속도, 위도, 경도, 방위각 등의 정보를 기반으로 미사일의 특징을 가진 공중 물체를 찾아냅니다.

두 번째, 레이더 시스템 내의 '레인지 게이트'가 미사일이 다음에 나타나게 될 영공의 구역을 계산합니다.

세 번째, 적의 미사일이 계산된 구역에 들어오면 패트리어트가 자신의 미사일을 발사합니다.

패트리어트 미사일 오차

- 1991년 2월 25일 사고
  - 이라크의 미사일을 격추하는데 실패를 해서 스커드 미사일이 육군 막사에 떨어짐
  - 28명의 대원이 사망하고 98명이 부상을 당함
  - 레이더가 날아오는 스커드를 일차적으로 탐지 -> 레인지 게이트가 다음 위치를 예측하였지만 발견되지 않음 -> 가짜 경보로 판단
- 미국 일반 조사위: 수치 계산의 부정확성이 주요 원인

가톨릭대학교 8

그런데 1991년 2월 25일 사고가 일어나게 됩니다. 이라크의 스커드 미사일을 격추하는데 실패를 해서 스커드 미사일이 육군 막사에 떨어졌고 28명의 대원이 사망하고 98명이 부상을 당했다고 합니다. 레이더가 날아오는 스커드를 일차적으로 탐지하였지만 이 미사일의 다음 위치를 예측한 레인지 게이트에서 스커드가 발견되지 않아 가짜 경보로 판단한 것이 참사로 이어진 것입니다.

미국 일반 조사위는 수치 계산의 부정확성이 주요 원인이라고 결론을 내렸습니다.

패트리어트 미사일 오차

- 패트리어트 시스템의 수치 저장
  - 24비트 고정 소수점 레지스터(fixed-point-registers)만을 사용함
  - 고정 소수점 방식은 (정수부, 소수부)로 저장하고 비트 수를 초과하면 버리는 구조
  - 24비트 이후 소수점 뒷부분이 잘리는 라운드오프 에러가 발생

$0.1_{(10)} = 0.0001100110011001100110011001100\ldots\ldots_{(2)}$

$0.1_{(10)} = 0.00011001100110011001100 / 11001100\ldots_{(2)}$

(24비트) 25비트부터 버림

오차 $= 0.00011001100110011001100110011001100\ldots\ldots_{(2)} - 0.00011001100110011001100_{(2)}$

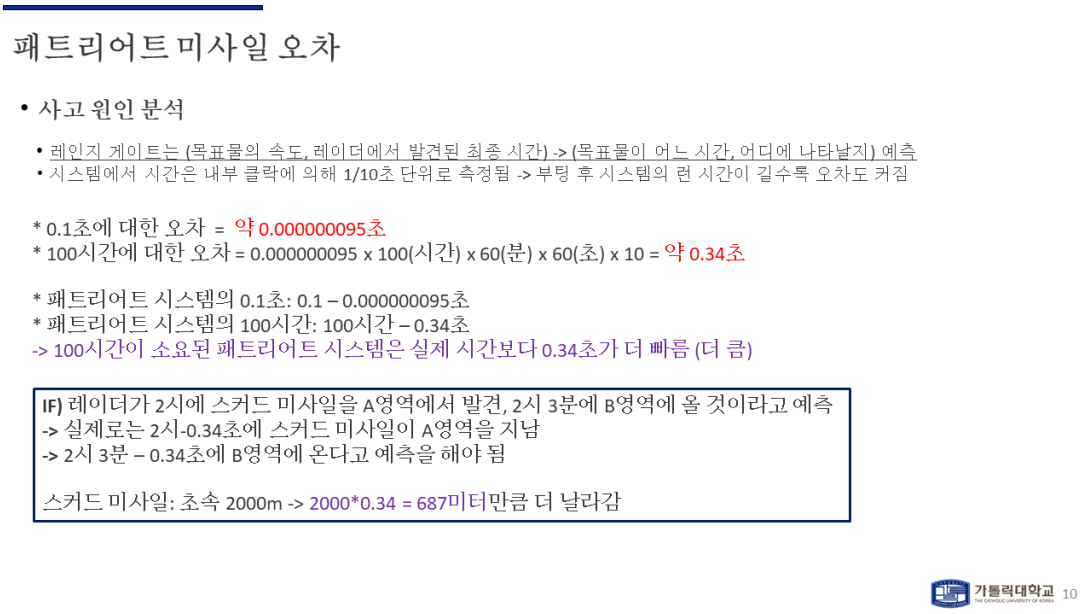
$= 0.00000000000000000000000011001100\ldots_{(2)}$

$=$ 약 $0.000000095_{(10)}$

가톨릭대학교 9

패트리어트 미사일 시스템은 고정 소수점 방식으로 수치를 저장합니다. 앞에서 설명한 부동 소수점 방식은 좀 일반적인 이야기였고요. 고정 소수점 방식은 부동 소수점 방식 이전에 사용된 좀

더 간단한 방법입니다. 단순하게 숫자를 정수부, 소수부로 나눈 다음에 비트 수를 초과하면 버리는 구조입니다. 그래서 24비트 레지스터를 사용했다고 하였으니 25비트부터는 버리는 구조인 것입니다. 따라서 0.1에 대한 오차는 10진수로 약 0.000000095가 됐다고 합니다. 이렇게 소수점이 잘려서 원본과 차이가 나는 오류를 **라운드 오프 에러**라고 합니다.



이 고정 소수점 방식을 기반으로 사고 원인을 분석해보겠습니다.

일단 레인지 게이트는 (목표물의 속도, 레이더에서 발견된 최종 시간)을 기반으로 (목표물이 어느 시간, 어디에 나타날지)를 예측합니다. 즉 이 예측 알고리즘은 속도와 시간 모두가 실시간으로 표현될 것을 요구합니다.

시스템에서 시간은 내부 클락에 의해 1/10초 단위로 측정되고 부팅 후 시스템의 런 시간이 길어질 수록 시간을 나타낸 숫자도 점점 커지는 구조입니다. 그런데 1/10은 이진수 입장에서 무한 소수이니 오차가 발생합니다. 따라서 부팅 후 시스템의 런 시간이 길수록 오차도 커진다고 생각할 수 있습니다.

이 때 패트리어트 미사일 시스템은 약 100시간 연속해서 가동되었다고 합니다.

따라서 라운드 오프 에러로 인한 시간 오차를 계산하면
0.000000095×100(시간)×60(분)×60(초)×10을 해서 0.34초가 나오는 것을 확인할 수 있습니다.

쉽게 말해서 패트리어트 시스템의 0.1초는 실제로 0.1-0.000000095초이고요. 패트리어트 시스템의 100시간은 실제로 100시간-0.34초를 의미합니다. 즉 100시간이 소요된 패트리어트 시스템은 실제 시간보다 0.34초가 더 빠릅니다. (0.34가 더 큽니다)

그럼 예를 들어, 레이더가 2시에 스커드 미사일을 A영역에서 발견하고, 2시 3분에 B영역에 올 것이라 예측을 했다고 가정합시다.

그럼 실제로 스커드 미사일은 2시 - 0.34초 에 A영역을 지났으니 2시 3분 - 0.34초에 B영역에 온다고 예측을 해야 됩니다. 레이더 시간만 믿고 2시 3분에 확인해 봤자 이미 스커드 미사일은 0.34초만큼 더 날라간 셈입니다.

스커드 미사일이 초속 2000미터라고 하니 2000*0.34=687미터가 더 날라갔다는 의미입니다.
안타깝게도 이 버그가 수정된 소프트웨어는 사고가 발생한 다음날이 되어서야 육군 기지에 도착을 했다고 합니다.

참고 자료

- 컴퓨터에서는 0.1 × 0.1 이 0.01 이 아닙니다
  - https://www.youtube.com/watch?v=vOO-oLS0H68&t=316s
- 1991년 미육군 MIM-104 패트리어트 미사일 소프트웨어의 숫자 전환 라운드오프 에러
  - https://grapevine9700.tistory.com/173
- 숫자를 부동소수점 방식으로 표현하기
  - https://codetorial.net/articles/floating_point.html
- 컴퓨터시스템 제3판
  - Computer System A programmer's Perspective (Randal E Bryant, David R O'Hallaron, 김형신 옮김)

저는 이 사례를 듣고 두 가지 정도를 느낀 것 같습니다.

첫 번째로, 우리가 왜 컴퓨터 사이언스를 배우고 소프트웨어를 개발해야 되는가를 느끼게 되었습니다.

두 번째로, 교수님들께서 항상 말씀하시잖아요. '실수가 허용되는 건 학생일 때까지이다'라고 하시는데 정말로 그런 것 같다고 생각이 들었습니다.

감사합니다

<2021/03/30>

지금까지 '패트리어트 미사일 오차'를 발표한 안성현이었습니다.

감사합니다.