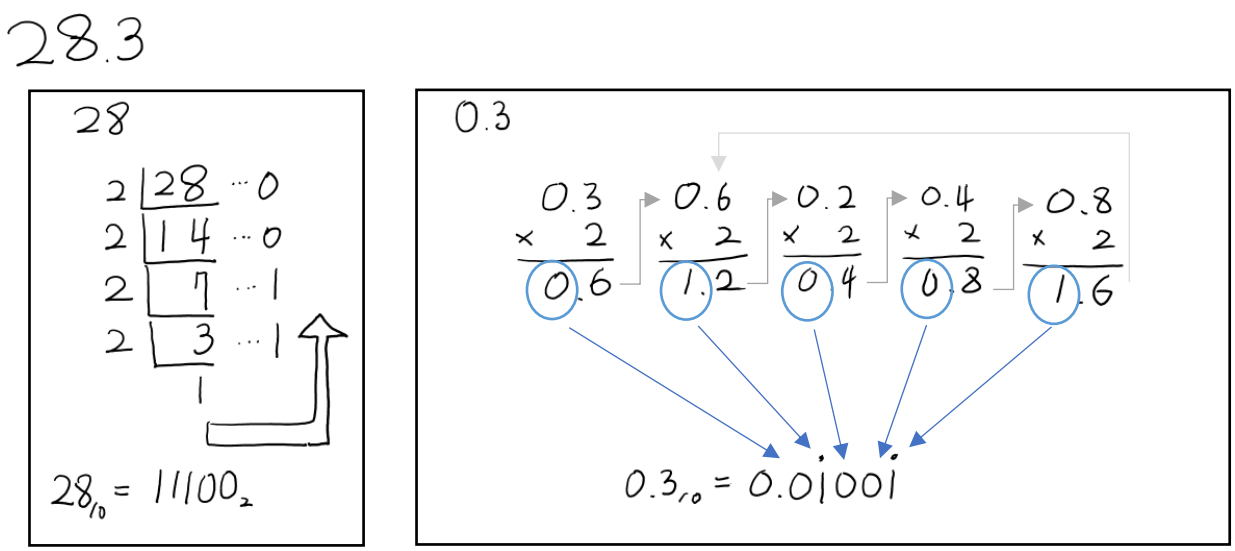
1. 배경지식

(1) 10진수 실수를 2진수 고정소수점으로 바꾸기

고정소수점은 10진수 실수를 28.3으로 나타내는 것처럼, 소수점의 위치를 고정하여 정수 부분과 실수부분을 표현하는 방법이다. 10진수 실수를 고정소수점을 이용한 2진수로 표현하는 것은 실수를 정수부분과 실수부분으로 나누어 진행한다. 예를 들어, 28.3을 2진수 고정소수점으로 표현하려면, 정수부분인 28을 2진수로 바꾸어 11100로 표현한다. 남은 실수부분인 0.3을 $0.0\dot{1}00\dot{1}$ 로 표현할 수 있다. 따라서 28.3은 $11100.0\dot{1}00\dot{1}$로 나타낼 수 있다.



(2) 고정소수점을 유동소수점으로 바꾸기

유동소수점은 고정소수점과 달리 소수점이 고정되어있지 않고, 수를 유효숫자와 지수로 나타낸다. IEEE 754 표준을 이용한 방법은 32개의 비트를 하나의 sign bit, 8개의 비트를 exponent bit, 나머지 23개의 비트를 mantissa bit으로 이용한다.

2진수 고정소수점으로 표현된 실수를 2진수 유동소수점으로 표현하는 것은 소수점 왼쪽의 정수를 1만 남기고 모두 오른쪽으로 이동시키고, 옮긴 자릿수만큼을 지수로 표현하는 것이다. 위에서 예를 든 28.3의 경우 11100.01001에서 정수를 1만 남기고 소수점을 이동시키면 1.110001001이 되고, 소수점은 4자리를 이동하였다.

$$11100.01001 = 1.110001001 \times 2^4$$

이것을 IEEE 754표준에 맞게 표현하려면 sing, exponent, mantissa 정보를 생성하여 조합하면 된다. 가장 먼저 sign비트는 표현하려는 실수의 부호를 의미하고, exponent는 지수에 127을 더한 값과 같다. 마지막으로 mantissa는 소수점 아래의 23자리 수

이다. 따라서 28.3을 IEEE 754표준에 맞게 표현하면 sign bit: 0, exponent bit: 10000011, mantissa: 110001001… 이므로, 0 10000011 11000100110011001100110으로 나타난다.

(3) 유동소수점의 덧셈과 뺄셈

유동소수점의 덧셈과 뺄셈은 계산하려는 두 수의 sign, exponent, mantissa data를 추출하여 계산한다. 각각의 mantissa에 정수1을 붙여 1.mantissa형태로 만든 뒤, 두 수의 exponent를 비교하여 서로 지수를 맞춘다. 이후 두 mantissa의 덧셈 혹은 뺄셈을 진행한다. 계산한 결과를 다시 유동소수점 방식으로 표현하면 계산과정이 종료된다.

2. 코드내용

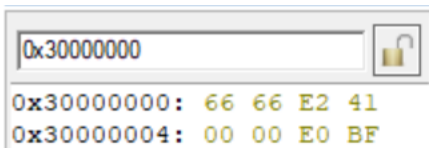
작성한 코드는 28.3 – 1.75를 계산하는 코드이다. 계산에 사용된 실수를 부동소수점 방식으로 나타내면 다음과 같다.

10진수	28.3	-1.75	26.55
Sign	0	1	0
Exponent	1000_0011	0111_1111	1000_0011
Mantissa	110_0010_0110_0110_0110_0110	110_0000_0000_0000_0000_0000	101_0100_0110_0110_0110_0110
부동소수점	01000001111000100110011001100110	10111111111000000000000000000000	01000001110101000110011001100110

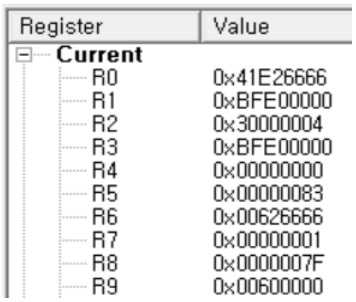
위 bit data에 대해 1.mantissa형태로 만들면 1110_0010_0110,,, 과 1110_0000,, 형태가 되고, 두 Exponent를 비교하면 28.3의 exponent가 4만큼 크다. Exponent값이 작은 -1.75의 mantissa값을 4만큼 오른쪽으로 shift하여 0000_1110_000,, 형태로 만들고 두 mantissa값을 빼면, 26.55의 mantissa값을 얻는다. 이것을 다시 부동소수점으로 나태내는 과정을 거쳐서 최종결과를 얻는다.

프로그램이 실행되면 가장 먼저 r1과 r2에 부동소수점으로 나타낸 두 수를 메모리에 저장하는 것부터 실행한다. 이후 문제풀이의 시작 부분에서 다시 계산할 두 수를 읽어온 뒤, sign, exponent, mantissa 부분을 각각 추출하여 r4,r5,r6과 r7,r8,r9에 저장한다. 다음으로 1.mantissa 형태를 만들기 위해 r10의 23번째 bit에만 1을 채운 뒤, mantissa가 저장된 r5, r8에 각각 더한다. Exponent를 비교하여 지수가 더 작은 수를 LSR연산하여 지수를 맞춘다. 두 exponent가 모두 양수이기 때문에 SUB명령어를 이용하여 계산하였다. 지수까지 모두 맞춘 mantissa를 두 수의 부호에 따라 더하거나 빼는 연산을 실행하고, mantissa를 normalize한다. 이 부분에서는 B명령어를 이용한 서브루틴을 사용하였다. 앞에서 mantissa에 더해준 r10을 다시 빼고, sign, exponent, mantissa를 조합하여 결과를 얻는다.

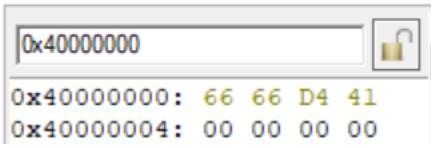
3. 결과



문제풀이 이전 메모리에 저장한 내용



sign, exponent, mantissa 추출 결과



프로그램 실행 결과

4. 고찰

처음 문제풀이를 시작할 때 레지스터에 32bit값을 저장하지 못하여 문제를 겪었다. 이 문제에 대해 인터넷 검색을 통해 이유를 찾을 수 있었다.[i] immediate value는 8bit를 이용하여 저장하기 때문에 나타난 이 문제를 해결하기 위하여 레지스터에 8비트의 값을 저장한 뒤에 LSL 명령어를 이용하여 8비트를 이동시키고, 다시 8비트를 더하는 과정을 반복하여 총 4번에 걸쳐 32bit를 저장하는 코드를 작성하였다. 다른 단순한 방법을 찾지 못한 점이 아쉽다.

레지스터에 저장된 부동소수점 정보에서 sign, exponent, mantissa bit 정보를 추출하는 과정이 인상깊었다. 필요한 비트 수만큼 레지스터에 1을 채우고, LSR을 이용하여 추출할 정보를 적절히 이동시킨 뒤에 AND명령어를 이용하였다.

---

[i] davespace, Immediate Values,

http://www.davespace.co.uk/arm/introduction-to-arm/immediates.html