2주차 2차시 2진수 연산

[학습목표]

1. 데이터에 따른 2진수의 표현을 예를 들어 설명할 수 있다.
2. 2진수의 기본적인 논리 연산을 설명할 수 있다.

학습내용1 : 데이터의 2진수 표현

<일반적인 디지털 장치에서는 2진수로 양의 정수, 음의 정수, 소수를 표현>

예시 : 부호가 있고 소수점을 포함하는 동일 값의 10진수와 2진수를 나타낸 예

- $(-13.625)_{10}$ = $(-1101.101)_2$

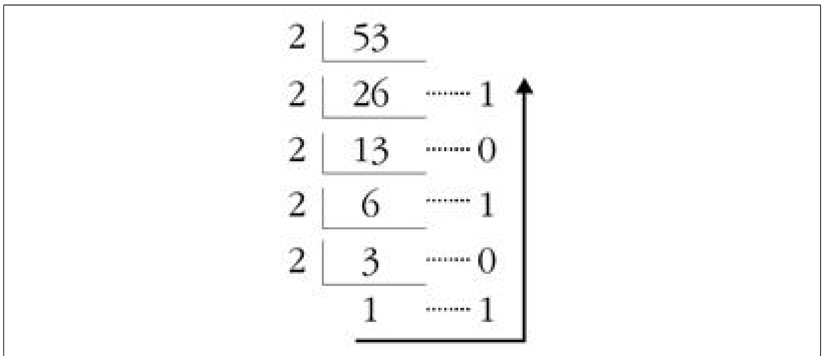
- 2진법으로 부호를 갖는 정수와 소수를 표현하려면 추가적으로 부호와 소수점의 기호를 사용하여야 하므로 단순한 진법 변환으로 해결되지 않음

1. 정수의 표현

<자연수 또는 양수의 10진수를 2진법으로 변환하면 부호가 없는 2진법으로 표현함>

1) 10진수를 2진수의 기수 2로 연속해서 나누고 이때 얻어지는 나머지가 2진수의 가수가 되어 2진수로 표현함

- 10진수 $(53)_{10}$을 부호 없는 2진수로 표현하는 과정 : $(53)_{10}$ = $(110101)_2$



- 예시 : 자연수의 10진수들을 부호가 없는 2진수로 표현한 예

$(57)_{10}$ = $(00111001)_2$, $(0)_{10}$ = $(00000000)_2$
$(1)_{10}$ = $(00000001)_2$, $(128)_{10}$ = $(10000000)_2$
$(255)_{10}$ = $(11111111)_2$

2. 부호가 존재하는 2진 정수의 표현

<디지털 장치에서는 부호를 구분할 수 있는 (+)와 (-)같은 별도의 기호는 존재하지 않고 최상위 비트 자리를 부호 비트로 할당하고 0이면 양수, 1이면 음수로 정의>

1) 나머지 비트들은 적절한 형태로 크기 값을 표현
- 부호화-크기 표현(signed-magnitude representation)
- 1의 보수 표현(1's complement representation)
- 2의 보수 표현(2's complement representation)

3. 부호화-크기 표현

<n비트로 구성된 2진수에서, 최상위 비트는 부호비트(signed bit)이고 나머지 n-1개의 비트들은 수의 절대 크기(magnitude)를 나타냄>

- $(+9)_{10}$ = $(0\ 0001001)_2$
- $(-9)_{10}$ = $(1\ 0001001)_2$
- $(+35)_{10}$ = $(0\ 0100011)_2$
- $(-35)_{10}$ = $(1\ 0100011)_2$

<부호화-크기 방법으로 표현된 2진수(a^{n-1} a^{n-2} ... a^1 a^0)를 10진수로 변환하는 방법임>
- 부호 비트를 통해서 부호를 결정, 크기 비트는 일반적인 10진수 변환방법과 동일함

$$A = (-1)^{a_{n-1}} (a_{n-2}\times2^{n-2} + a_{n-3}\times2^{n-3} + \dots + a_1\times2^1 + a_0\times2^0)$$

$$(0\ 0100011)_2 = (-1)^0(0\times2^6 + 1\times2^5 + 0\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0)$$

$$= (32 + 2 + 1) = (35)_{10}$$

<가장 간단한 개념으로 부호를 표현하지만, 덧셈과 뺄셈 연산을 수행하기 위해서는 부호비트와 크기 부분을 별도로 처리하여야 함>
- 0(zero)의 표현이 두 개 존재하므로 표현할 수 있는 수의 범위가 줄어듦

$$(0\ 0000000)_2 = (+0)_{10} \quad (1\ 0000000)_2 = (-0)_{10}$$

4. 보수를 이용한 부호를 갖는 2진수의 표현

* 1의 보수(1's complement) 표현 : 모든 비트들을 반전 (0 → 1, 1 → 0)
* 2의 보수(2's complement) 표현 : 모든 비트들을 반전하고, 결과값에 1을 더함

1) 보수를 이용한 2진수의 부호변경
- $(+9)_{10}$ = $(0\ 0001001)_2$
- $(-9)_{10}$ = $(1\ 1110110)_2$ (1의 보수)

- $(-9)_{10}$ = 1 1110111 (2의 보수)
- $(+35)_{10}$ = $(0\ 0100011)_2$
- $(-35)_{10}$ = $(1\ 1011100)_2$ (1의 보수)
- $(-35)_{10}$ = $(1\ 1011101)_2$ (2의 보수)

- 보수를 이용하면 부호비트가 자연스럽게 변경되고 그 크기도 적절한 형태로 변경됨
- 2의 보수는 0에 대한 표현이 하나만 있으며, 산술 연산이 용이함
- 2의 보수는 가장 효율적이기 때문에 컴퓨터를 비롯한 디지털 장치에 부호를 갖는 2진수를 표현하는데 사용함

5. 10진수의 2의 보수로 표현된 2진수 변환 과정

1) 10진수 $(-25)_{10}$를 2의 보수로 표현된 2진수를 변환하는 과정

① 1단계 : 10진수를 부호가 없는 2진수로 변환함
- $(25)_{10}= (11001)_2$

② 2단계 : 부호 비트를 삽입함
- $(25)_{10}= (011001)_2$

③ 3단계 : 1의 보수를 구함
- $(011001)_2 \Rightarrow (100110)_2$

④ 4단계 : 2의 보수를 구함
- $(100110)_2 \Rightarrow (100111)_2$

⑤ 다음 결과를 얻을 수 있음
- $(-25)_{10} \Rightarrow (100111)_2$

6. 2의 보수로 표현된 2진수를 10진수로 변환

- 2의 보수로 표현된 양의 정수(최상위 비트: a_{n-1} = 0)는 부호 비트를 제외한 크기의 비트들은 실제의 크기를 나타냄
- 부호 없는 2진수를 10진수로 변환하는 방법과 동일함

$$A = a_{n-2}\times2^{n-2} + a_{n-3}\times2^{n-3} + \ldots\ a_1\times2^1 + a_0\times2^0$$

1) 2의 보수로 표현된 음의 정수(최상위 비트: a_{n-1} = 1)
- 부호 비트에 해당하는 최상위 비트의 자릿수를 2의 승수로 표현하고 (-)를 붙여서 음수가 되도록 함
- 나머지 비트는 양의 정수와 동일함

$$A = -\ 2^{n-1} + (a_{n-2}\times2^{n-2} + a_{n-3}\times2^{n-3} +\cdots\ a_1\times2^1 + a_0\times2^0)$$

＊ 예시

$$(10101110)_2 = -128 + (1 \times 2^5 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1)$$
$$= (-82)_{10}$$

- 2진수 음의 정수를 보수를 이용하여 양의 정수로 만들고 이것을 10진수로 변환
- 그리고 최종 단계에서 (-) 부호를 붙이는 방식

＊ 예시

1단계: 2의 보수를 이용하여 음수를 양수로 변환
(10101110→01010010)
2단계: $(01010010)_2 = -(1 \times 2^6 + 1 \times 2^4 + 1 \times 2^1)$
$= -(64 + 16 + 2) = (-82)_{10}$

7. 2진수의 표현 범위

1) 2의 보수를 사용한 3비트 이진수 표현의 예

＊ 예시

$+3 = (011)_2$	$-1 = (111)_2$
$+2 = (010)_2$	$-2 = (110)_2$
$+1 = (001)_2$	$-3 = (101)_2$
$+0 = (000)_2$	$-4 = (100)_2$

- 표현할 수 있는 수의 범위는 -4 ~ 3이 됨
- 이것은 -2^{3-1} ~ $2^{3-1}-1$로 표현됨

<n비트 데이터의 경우로 일반화 해서 수의 범위를 나타내면 다음과 같음>
- $-2^{n-1} \leq N \leq 2^{n-1}-1$

8. 부호가 있는 8비트 이진수의 표현

1) 부호화-크기 표현 : $-(2^7 - 1)$ ~ $+(2^7 - 1)$

2) 1의 보수 : $-(2^7 - 1)$ ~ $+(2^7 - 1)$

3) 2의 보수 : -2^7 ~ $+(2^7 - 1)$

10진수	부호화-크기표현	1의보수	2의보수
127	01111111	01111111	01111111
126	01111110	01111110	01111110
…	…	…	…
1	00000001	00000001	00000001
+0	00000000	00000000	00000000
-0	10000000	11111111	X
-1	10000001	11111110	11111111

9. 비트 확장(Bit Extension)

<부호가 존재하는 데이터의 비트 수를 늘리는 연산을 비트확장이라고 함>

1) 부호화-크기 표현의 비트확장
- 부호 비트를 확장되는 최상위 자리로 이동시키고, 나머지 새로 확장되는 크기 비트들은 0으로 채움

- $(+21)_{10}=(00010101)_2$ (8비트)
  $\rightarrow (+21)_{10}=(0000000000010101)_2$ (16비트)
- $(-21)_{10}=(10010101)_2$ (8비트)
  $\rightarrow (-21)_{10}=(1000000000010101)_2$ (16비트)

- 16비트로 부호 확장된 양의 정수에 대한 2의 보수를 구하여 부호를 변경함

$$\begin{aligned}&0000000000010101\\ \Rightarrow\ &1111111111101010\\ \Rightarrow\ &1111111111101011\end{aligned}$$

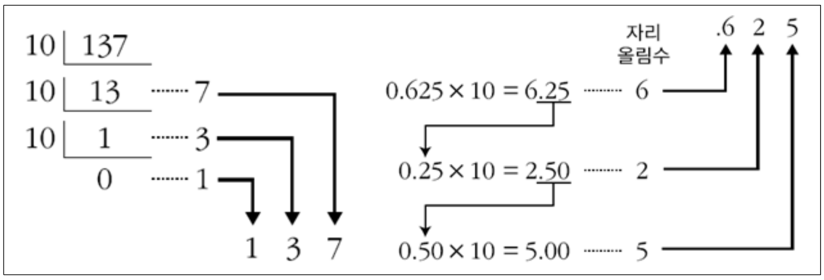
- 구해진 2의 보수는 음의 정수와 동일하므로 부호 비트 확장의 방법이 정당함을 보여줌

10. 소수(Decimal Fraction)의 표현

1) 정수와 소수를 포함한 10진수
- $(137.625)_{10}$= 137 + 0.625

2) 10의 지수 승 형태로 표시
- $137 + 0.625 = 1\times10^2 + 3\times10^1 + 7\times10^0 + 6\times10^{-1} + 2\times10^{-2} + 5\times10^{-3}$
- 정수부분의 가수는 기수 10으로 연속으로 나눗셈을 수행해 얻은 나머지로 구할 수 있음
- 소수부분은 지수가 음의 정수이므로 가수는 나눗셈의 반대인 곱셈을 연속적으로 수행하는 것임

- 그리고 정수부분으로 발생하는 자리올림수가 가수가 됨



11. 소수를 포함하는 10진수의 2진수 표현

1) 2진수로 변환하기 위해서는 2의 지수 승으로 표현해야 함

$$
\begin{aligned}
&(137.625)_{10} \\
&= 1\times10^{2} + 3\times10^{1} + 7\times10^{0} + 6\times10^{-1} + 2\times10^{-2} + 5\times10^{-3} \\
&= A_m\times2^{m} + \cdots + A_1\times2^{1} + A_0\times2^{0} + A_1\times2^{-1} + A_2\times2^{-2} + \cdots \\
&\quad + A_m\times2^{-m}
\end{aligned}
$$

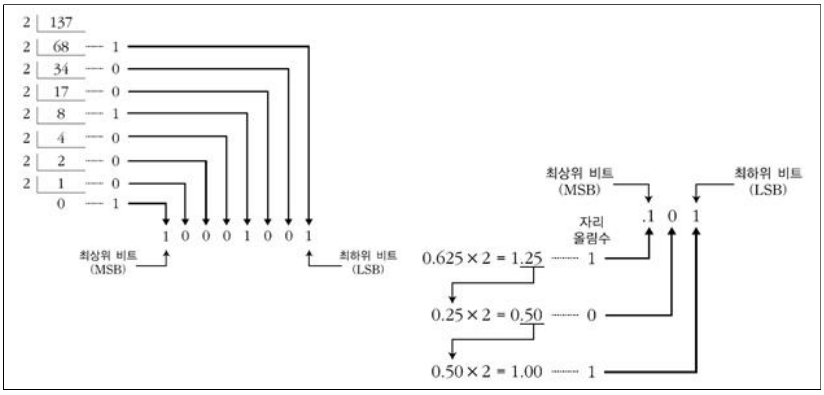
2) 정수부분은 2로 연속적인 나눗셈을 소수부분은 2로 연속적인 곱셈을 수행함

① 1단계 : 정수부분과 소수부분을 분리함

② 2단계 : 정수부분의 10진수를 2진수로 변환함
- $(137)_{10}$ = $(10001001)_{10}$

③ 3단계 : 소수부분의 10진수를 2진수로 변환함
- $(0.625)_{10}$ = $(0.101)_{2}$

④ 4단계 : 얻어진 정수와 소수의 2진수를 합함
- $(137.625)_{10}$ = $(10001001)_{2}$ + $(0.101)_{2}$ = $(10001001.101)_{2}$



12. 소수점을 포함하고 있는 이진수의 십진수로 변환

1) 정수부분은 기존의 방법과 같이 2의 지수 승을 이용하여 분해함

- 소수점 이하는 2의 (-)지수 승을 사용함

$$A = a_{n-1} \times 2^{-1} + a_{n-2} \times 2^{-2} + \dots + a_1 \times 2^{-(n-1)} + a_0 \times 2^{-n}$$

2) 소수를 포함하는 이진수 $(1101.101)_2$를 십진수로 변환함

2^3	2^2	2^1	2^0	.	2^{-1}	2^{-2}	2^{-3}
1	1	0	1	.	1	0	1

$$\begin{aligned}&(1101.101)_2\\&= 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}\\&= 8+4+1+0.5+0.125 = (13.625)_{10}\end{aligned}$$

13. 부동소수점(Floating-point)의 표현

1) 고정소수점(fixed-point)표현
- 소수가 고정된 소수점을 통해서 구분하여 표현된 방식 : $(17.60)_{10}$
- 표현 범위의 한계가 있어 아주 큰 값과 매우 작은 값을 표현하는 것이 불가능함

2) 부동소수점 표현
- 지수를 사용 소수점의 위치를 이동하여 수의 표현 범위를 확대함

  $(176,000)_{10} = 1.76 \times 10^5$,

  $(0.000176)_{10} = 1.76 \times 10^{-4}$
- 부동소수점 수(Floating-point Number)을 표현하는 일반적인 형식임 → $\pm M \times B^{\pm E}$
- 여기서, ±는 부호로서 + 혹은 -을 나타내며 M은 가수(significand), B는 기수(base), E는 지수(exponent)를 나타냄

3) 2진 부동소수점 수(binary floating-point number)
- 가수 M은 0과 1로 구성이 되는 2진수 이며, 기수 B는 2가 됨

  $+1.100101 \times 2^3$

14. 2진 부동소수점 수는 표현

1) 단일-정밀도(single-precision) 부동소수점 수 : 32비트로 표현

2) 복수-정밀도(double-precision) 부동소수점 수 : 64비트로 표현

3) 단일-정밀도 부동소수점 수 형식
- 32비트가 세 개의 필드로 구성됨

31	30 ~ 23	22 ~ 0
부호(S) 필드	지수(E) 필드	가수(M) 필드

- 부호 필드(S 필드)는 1비트로 0이면 양수이고 1이면 음수
- 지수 필드는 지수 값을 저장하는 곳임
- 8비트이므로 256(2^8)개를 표현할 수 있음
- 가수 필드는 23비트이므로 8388608(2^{23})개를 표현할 수 있음
- 따라서 고정 소수점 수와 비교해서 표현할 수 있는 수의 범위가 훨씬 넓음

4) 각 필드의 비트 할당 문제는 표현하는 수의 범위와 정밀도를 결정
- 지수(E) 필드의 비트 수가 늘어나면, 표현 가능한 수의 범위가 확장됨
- 가수(M) 필드의 비트 수가 늘어나면, 이진수로 표현할 수 있는 수가 많아져서 정밀도가 증가
- 따라서 적절한 비트할당이 필요함

15. 정규화된 표현(Normalized Representation)

1) 부동소수점의 수는 지수의 값에 따라 표현이 여러 가지 존재함

0.1001×2^5, 100.1×2^2, 0.01001×2^6

2) 정규화된 표현은 부동소수점 수에 대한 표현을 통일하기 위한 방법

$\pm\ 0.1bbb...b \times 2^E$

- 위의 예에서 정규화된 표현은 0.1001×2^5이 됨

3) 단일 정밀도 부동소수점의 수에서 정규화된 표현
- 예시 : 0.1001×2^5을 필드 별로 비트로 표현하면
- 부호(S) 비트 = 0 / 지수(E) = 00000101
- 가수(M) = 1001 0000 0000 0000 0000 000

부호(S)필드	지수(E)필드	가수(M)필드
0	00000101	1001 0000 0000 0000 0000 000

16. 바이어스된 지수값

1) 정규화된 표현에서 소수점 우측의 첫 번째 비트는 항상 1로 생략 가능
- 가수 필드 23비트를 이용하여 생략된 소수점 아래 첫 번째 1을 포함하여 24자리의 수까지 표현 가능하게 되어 1비트를 더 표현할 수 있음

2) 지수의 바이어스된 수(biased number)로 표현
- 지수 필드의 지수는 양의 값뿐만 아니라 음의 값을 가지므로 부호에 대한 표현을 가능하게 함
- 음수의 표현뿐만 아니라 0에 대한 표현에서 모든 비트가 0이 됨

3) 바이어스된 값은 원래의 지수 비트값에서 바이어스 값을 더해서 얻음
- 지수값이 4이면 이를 8비트의 이진수로 표현하면 00000100이 되며, 이것을 128로 바이어스된 값을 구하기 위해서는 128의 이진수값 10000000을 더해줌
- 지수값 : $(4)_{10}$ = $(00000100)_2$
- 128로 바이어스 된 지수값 : 00000100 + 10000000 = 10000100

17. 지수 비트 패턴과 128로 바이어스된 지수 값

지수 비트 패턴	절대값	128로 바이어스된 지수값
11111111	255	+127
11111110	254	+126
…	…	…
10000100	132	+4
10000011	131	+3
10000010	130	+2
10000001	129	+1
10000000	128	0
01111111	127	-1
01111110	126	-2
…	…	…
00000001	1	-127
00000000	0	-128

18. 바이어스 값을 이용한 부동소수점 수

1) 바이어스 값이 128일 때, - $(13.625)_{10}$에 대한 부동소수점 수의 표현
- $(13.625)_{10}$ = $(1101.101)_2$ = 0.1101101×2^4

2) 이것을 필드 별로 비트열로 나타내면

부호(S) 비트 = 1 (-)
지수(E) = 00000100 + 10000000 = 10000100 (바이어스 128을 더한다)
가수(M) = 10110100000000000000000
(소수점 우측의 첫 번째 1은 제외)

부호(S)필드	지수(E)필드	가수(M)필드
1	10000100	1011 0100 0000 0000 0000 0000

19. 부동소수점 수의 표현 범위

1) 정규화된 표현과 바이어스된 지수를 이용한 부동소수점 수의 표현
- 32비트 부동소수점 수의 표현 범위

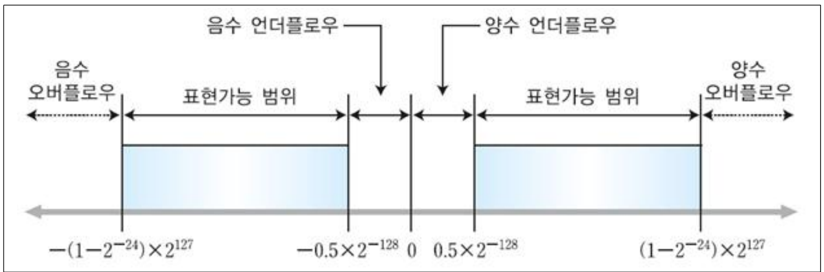
0.5×2^{-128} ~ $(1-2^{-24})\times2^{127}$ 사이의 양수들

$-(1-2^{-24})\times2^{127}$ ~ -0.5×2^{-128} 사이의 음수들

2) 아주 작은 수와 그리고 아주 큰 수는 표현이 불가능함
- 제외되는 범위는 다음과 같음

- $(1-2_{-24})\times2^{127}$보다 작은 음수
  → 음수 오버플로우(negative overflow)
- 0.5×2^{-128} 보다 큰 음수
  → 음수 언더플로우(negative underflow)
- 0
- 0.5×2^{-128} 보다 작은 양수
  → 양수 언더플로우(positive underflow)
- $(1-2^{-24})\times2^{127}$ 보다 큰 양수
  → 양수 오버플로우(positive overflow)

3) 32비트 부동소수점 수의 표현 범위



20. IEEE 754 표준

1) 국제 표준 : $\pm$ 1.bbbb⋯bbb × $2^{\pm E}$

단일-정밀도 형식	복수-정밀도 형식
지수 : 8비트	지수 : 11비트
바이어스 = 127	바이어스 : 1023
가수 : 23비트	가수 : 52비트
표현 영역 : 10^{-38} ~ 10^{38}	표현영역 : 10^{-308} ~ 10^{308}

2) 10진수 $-(13.625)_{10}$ 의 IEEE 754 표현
- 13.62510 = 1101.101 = 1.101101 × 2^3

3) 표준형을 단일-정밀도 형식으로 표시
- 바이어스 값이 127이고 정수 값인 소수점 좌측의 1은 제외됨

- **부호(S) = 1(-)**
- **지수(E)**
  = 00000011 + 01111111 = 10000010
  (바이어스 127을 더함)
- **가수(M) = 10110100000000000000000**
  (소수점 좌측의 1은 제외함)

부호(S)필드	지수(E)필드	가수(M)필드
1	10000010	1011 0100 0000 0000 0000 000

학습내용2 : 2진수 산술 연산

〈덧셈, 뺄셈, 곱셈, 나눗셈 등의 산술연산과 참, 거짓을 판별하는 논리연산이 있음〉

1. 진수의 산술 연산

〈부호를 갖는 2진 정수의 산술 연산은 2의 보수를 활용하여 수행됨〉
- 부동소수점의 수에 대한 산술 연산은 지수부분과 가수부분을 분리해서 독립적으로 수행됨

1) 정수의 산술 연산
- 부호변경(2의 보수) : A = A'+ 1 (A': 1의 보수)
- 덧 셈 : C = A + B
- 뺄 셈 : C = A - B
- 곱 셈 : C = A × B
- 나눗셈 : C = A / B

+19 :	00010011
1의 보수 :	11101100
	+ 1
−19 :	11101101

2) 부호변경
- 2의 보수를 사용함
- 음의 정수를 2진수로 표현할 때 사용함

2. 2진 정수의 덧셈 연산

1) 오버플로우가 발생하지 않는 덧셈 연산

* 양수와 양수의 덧셈
- 최상위 비트에서 자리 올림이 발생하지 않아 계산의 결과에서 오류가 발생하지 않고 정확한 답을 출력함
- $(+2)_{10}$ + $(+3)_{10}$ = $(+5)_{10}$

$$\begin{array}{rr} & 0010 \\ + & 0011 \\ \hline & 0101 \end{array}$$

* 최상위 비트에서 자리 올림이 발생하지 않은 음수와 양수의 덧셈
- 최상위 비트에서 자리 올림수가 발생하지 않음
- 따라서 부호비트가 변경되는 등의 잘못된 계산이 발생하지 않아서 정확한 값을 얻을 수 있음
- $(-6)_{10}$ + $(+3)_{10}$ = $(-3)_{10}$

$$\begin{array}{rr} & 1010 \\ + & 0011 \\ \hline & 1101 \end{array}$$

- 덧셈 결과로 발생하는 최상위 비트에서 자리 올림 수는 버림 하지만 연산에는 오류가 없이 올바른 답을 얻을 수 있음
- $(-3)_{10}$ + $(+5)_{10}$ = $(+2)_{10}$

$$\begin{array}{rr} & 1101 \\ + & 0101 \\ \hline 1 & 0010 \end{array}$$

버림

* 음수와 음수의 덧셈
- 음수와 음수의 덧셈은 필연적으로 최상위 비트에서 자리 올림이 발생하고 자리 올림 수는 버림을 통해서 버려짐
- 그러나 그 결과는 정확한 값임
- $(-2)_{10} + (-4)_{10} = (-6)_{10}$

$$\begin{array}{rr} & 1110 \\ + & 1100 \\ \hline 1 & 1010 \\ \text{버림} & \end{array}$$

2) 오버플로우가 발생하는 덧셈 연산
- 덧셈 결과가 표현할 수 있는 범위를 초과하여 결과값이 틀리게 되는 상태를 덧셈의 오버플로우 상태라고 함

* 양수와 양수의 덧셈
- 덧셈의 결과는 자리 올림으로 인해서 부호비트가 변경 잘못된 결과 발생함
- $(+4)_{10} + (+5)_{10} = (+9)_{10}$

$$\begin{array}{rr} & 0100 \\ + & 0101 \\ \hline & 1001 \end{array} = (-7)_{10}$$

* 음수와 음수의 덧셈
- 자리 올림이 발생하고 부호 비트는 변경됨
- $(-7)_{10} + (-6)_{10} = (-13)_{10}$

$$\begin{array}{rr} & 1001 \\ + & 1010 \\ \hline 1 & 0011 \end{array} = (+3)_{10}$$

3. 2진수 정수의 뺄셈 연산

1) 2의 보수를 사용 결과적으로 덧셈을 수행함
- A - (+B) = A + (-B), A - (-B) = A + (+B)
- 빼지는 수 A를 피감수(minuend)라 하며, 빼는 수 B를 감수(subtrahend)라 함

* 오버플로우가 발생하지 않는 뺄셈 연산 : 연산 결과가 디지털 장치에서 표현할 수 있는 범위 안에 존재 연산의 결과는

정확함

2) 최상위 비트에서 자리 올림수가 발생하지 않는 뺄셈

- 10진수에서는 감수를 음수화하고 그 다음 뺄셈을 덧셈으로 고치고 계산을 $(+2)_{10} - (+5)_{10} = (+2)_{10} + (-5)_{10} = (-3)_{10}$ → 0010+1011=1101
- 최상위 비트에서 자리 올림 발생하는 뺄셈 $(+5)_{10} - (+2)_{10} = (+5)_{10} + (-2)_{10} = (+3)_{10}$ → 0101+1110= 1 0011

$$
\begin{array}{rr}
 & 0101 \\
+ & 1110 \\
\hline
1 & 0011 \quad = (+3)_{10}
\end{array}
$$

버림

- 부호비트의 변경은 없으나 자림 올림이 발생하였음
- 자리 올림 수는 버려지게 되고 나머지를 취하면 올바른 값을 얻을 수 있음

3) 뺄셈 결과가 그 범위를 초과하여 틀리게 되는 상태를 뺄셈 오버플로우라고 함

4) 최상위 비트에서 자리올림이 발생하지 않는 경우

- 부호비트가 변경이 발생하지만 자림 올림이 발생하지 않은 경우
- $(+7)_{10} - (-5)_{10} = (+7)_{10} + (+5)_{10} = (+12)_{10}$ →

$$
\begin{array}{rr}
 & 0111 \\
+ & 0101 \\
\hline
 & 1100 \quad = (-4)_{10}
\end{array}
$$

오버플로우

- 부호변경은 오버플로우가 발생한 것으로 답은 정확하지 않음

5) 최상위 비트에서 자리 올림이 발생하는 경우

- 부호비트가 변경되고 또한 최상위 비트에서 자리 올림이 발생한 경우
- $(-6)_{10} - (+4)_{10} = (-6)_{10} + (-4)_{10} = (-10)_{10}$ →

$$
\begin{array}{rr}
 & 1010 \\
+ & 1100 \\
\hline
1 & 0110 \quad = (+6)_{10}
\end{array}
$$

오버플로우

버림

- 얻어진 결과에서 버림의 과정을 거쳤지만 그래도 뺄셈의 오버플로우가 발생한 경우로 계산결과는 잘못된 것임

4. 2진 정수의 곱셈 연산

1) A × B = C

2) 곱하는 수(B)를 승수라고 하며, 곱하여 지는 수(A)를 피승수라고 함

3) 부호가 없는 2진수의 곱셈
- 승수의 각 숫자에 대하여 부분 합을 계산, 승수의 한 비트가 0이면 부분 합도 0이 됨
- 그러나 1이면 부분 합은 피승수와 동일하게 됨
- 최종 결과값은 부분 합을 한 자릿수씩 왼쪽으로 이동하고, 더하여 구함
- 피승수 1101과 승수 1011과의 곱셈 과정

				1	1	0	1	피승수(13)
			×	1	0	1	1	승수(11)
				1	1	0	1	
			1	1	0	1		부분합
		0	0	0	0			
	1	1	0	1				
1	0	0	0	1	1	1	1	곱의 결과(143)

- 4비트의 두 수가 서로 곱셈을 수행하면, 2배인 8비트의 길이의 결과를 출력
- 일반화하면, 두 N비트 2진 정수를 곱한 결과값의 길이는 2N 비트가 됨

4) 2의 보수에 의해서 부호를 갖는 2진수의 곱셈
- 음수를 양수로 변환하고 부호가 없는 곱셈을 수행하고, 만약 승수와 피승수의 부호가 서로 다르다면 결과 값에 2의 보수를 취하여 부호를 변경함
= 0010 × 1001

- 부호가 있는 2진수의 곱셈을 수행하기 위해서 음수 1001의 2의 보수를 구함
= 1001 → 0111

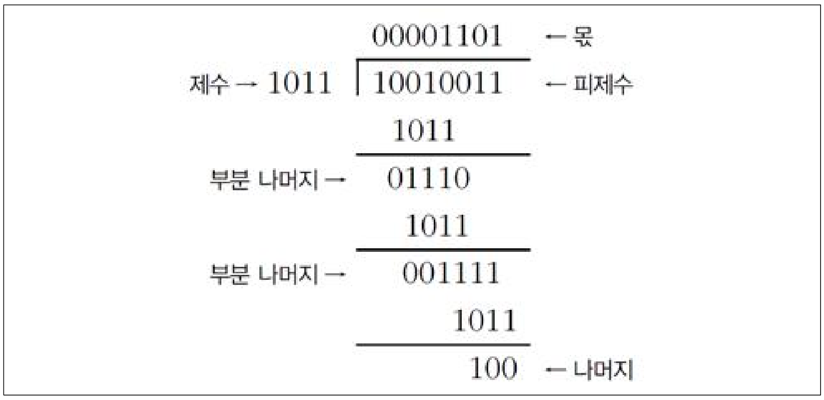
- 부호 없는 2진수의 곱셈을 수행함

				0	0	1	0	피승수(2)
			×	0	1	1	1	승수(7)
				0	0	1	0	
			0	0	1	0		
		0	0	1	0			
	0	0	0	0				
0	0	0	0	1	1	1	0	곱의 결과(14)

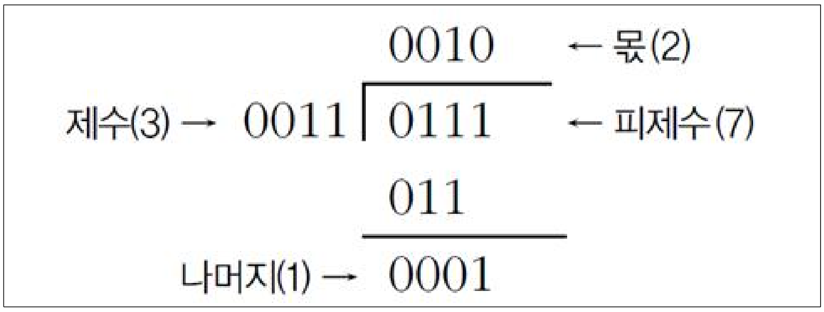
- 피승수와 승수의 부호가 서로 다르므로 얻어진 결과를 다시 2의 보수화를 통해서 부호를 변경함
- 결과적으로 $(-14)_{10}$를 얻게 됨 : 00001110 → 11110010

5) D ÷ V = Q ⋯ R
- 나누어지는 수 D를 피제수(dividend)라고 하며, 나누는 수 V를 제수(divisor)라고 함
- 나눗셈의 결과로 몫(quotient) Q와, 나머지 수(remainder) R을 얻음

6) 부호 없는 2진 정수의 나눗셈
- 10진수의 나눗셈과 동일함
- 10010011 ÷ 1011를 계산하는 과정 :



7) 2의 보수를 사용하여 부호가 표현된 2진 정수의 나눗셈 연산
- 2의 보수를 통해서 모두 양수로 고친 다음 부호 없는 2진수의 나눗셈 수행
- 제수와 피제수의 부호가 다른 경우에는 몫을 부호를 변경함
- 예) 0111 ÷ 1101
- 1101은 음수이므로 2의 보수화를 통해서 양수를 구함
- 부호가 없는 2진수의 나눗셈을 수행함
- 제수와 피제수가 동일하지 않으므로 얻어진 몫을 2의 보수로 표현함
- 0010 → 1110
- 따라서 몫은 $(-2)_{10}$가 얻어짐



6. 부동소수점 수의 산술연산

<부동소수점 수의 산술은 가수와 지수의 연산을 분리해서 수행함>

1) 부동소수점 수의 덧셈과 뺄셈
- 지수들이 동일한 값을 갖도록 일치, 가수의 소수점이 좌우로 이동함
- 다음으로 가수들 간의 덧셈과 혹은 뺄셈을 수행함
- 2진수의 경우, 마지막으로 결과를 정규화함

2) 2의 보수를 사용하여 부호가 표현된 2진 정수의 나눗셈 연산
- 10진수의 부동소수점의 수의 덧셈과 뺄셈
- A = 0.3×10^2, B = 0.2×10^3

덧셈 연산

A + B
$= 0.3\times10^2 + 0.2\times10^3$
$= 0.3\times10^2 + 2\times10^2$
$= 2.3\times10^2$

뺄셈 연산

A - B
$= 0.3\times10^2 - 0.2\times10^3$
$= 0.3\times10^2 - 2\times10^2$
$= -1.7\times10^2$

- 2진수의 부동소수점 수의 덧셈과 뺄셈

$$\begin{array}{rcrcl} 0.110010 \times 2^2 & \xrightarrow{\text{지수 조정}} & 0.011001 \times 2^3 & & \\ +0.111011 \times 2^3 & & \underline{+0.111011 \times 2^3} & \text{정규화} & \\ & & 1.010100 \times 2^3 & \rightarrow & 0.101010 \times 2^4 \end{array}$$

7. 부동소수점 수의 곱셈

<가수끼리는 곱셈 연산을 수행하고 지수끼리는 덧셈을 수행함>

1) 10진수의 부동소수점 수의 곱셈

✓ A = 0.3×10^2, B = 0.2×10^3

A × B
$= (0.3\times10^2)\times(0.2\times10^3)$
$= (0.3\times0.2)\times10^{2+3}$
$= 0.06\times10^5$

2) 2진수 부동소수점 수의 곱셈 과정
- 가수들을 곱하는 것과 지수들을 더하는 과정은 동일하며, 결과값을 정규화하는 것이 추가됨
- $(0.1011 \times 2^3) \times (0.1001 \times 2^5)$
- 가수 곱하기 : 1011 × 1001 = 01100011
- 지수 더하기 : 3 + 5 = 8
- 정규화 : $0.01100011 \times 2^8 = 0.1100011 \times 2^7$

8. 부동소수점 수의 나눗셈

<가수부분은 나눗셈 연산을 수행하고, 지수부분은 뺄셈 연산의 수행>

1) 10진수의 부동소수점 수의 나눗셈

✓ $A \div B$
$= 0.3 \times 10^2 \div 0.2 \times 10^3$
$= (0.3 \div 0.2) \times 10^{2-3}$
$= 1.5 \times 10^{-1}$

2) 2진수 부동소수점 수의 나눗셈
- 첫 번째로 가수들을 나누고 피제수의 지수에서 제수의 지수를 빼고 마지막으로 결과값을 정규화하는 과정을 추가 함
- $(0.1100 \times 2^5) \div (0.1100 \times 2^3)$
- 가수 나누기 : 1100 ÷ 1100 = 1
- 지수 뺄셈 : 5 - 3 = 2
- 정규화 : 0.1×2^3

9. 부동소수점 수의 연산 과정에서 발생 가능한 문제

<부동소수점의 수의 표현 범위에서 오버플로우와 언더플로우가 발생하는 영역이 존재, 연산의 결과가 이런 영역에 포함되면 오류가 발생함>

1) 지수 오버플로우(exponent overflow)
- 양의 지수값이 최대 지수값을 초과하여 발생하는 오류
- 수가 너무 커서 표현될 수 없음을 의미하는 것으로 +∞ 또는 -∞로 나타냄

2) 지수 언더플로우(exponent underflow)
- 음의 지수값이 최대 지수값을 초과하는 경우
- 수가 너무 작아서 표현될 수 없음을 의미하므로 0으로 표시됨

3) 가수 언더플로우(mantissa underflow)
- 가수의 소수점 위치 조정 과정에서 비트들이 가수의 우측 편으로 넘치는 경우로서 반올림(rounding)을 사용해서 문제를 해결함

4) 가수 오버플로우(mantissa overflow)
- 같은 부호를 가진 두 가수들을 덧셈하였을 때 올림수가 발생하는 경우로 재조정(realignment) 과정을 통하여 정규화하여서 해결함

학습내용3 : 2진수 논리 연산

1. 2진수의 논리 연산

<논리 연산은 주어진 명제에 대하여 참(true)과 거짓(false)를 결정하는 연산임>
- 컴퓨터와 같은 디지털 장치에서는 많은 산술 연산뿐만 아니라 다양한 논리 연산을 지원함

1) 기본적인 논리 연산

① AND 연산
- 2진수의 모든 입력이 모두 1일 때, 1을 출력하고 나머지의 경우에는 0을 출력 함

② OR 연산
- 2진수의 입력 중 하나만 1이면, 1을 출력하고, 모든 입력이 0일 때는 0을 출력함

③ Exclusive-OR(XOR) 연산
- 2진수의 입력중 1의 개수가 홀수개 일 경우에 출력이 1이 됨

④ NOT 연산
- 입력을 부정(not)하는 출력

2. 기본적인 논리연산의 진리표

입력X	입력Y	X AND Y	X OR Y	X XOR Y	NOT X	NOT Y
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	1
1	1	1	0	0	0	0

3. 컴퓨터 응용 논리 연산(1)

1) 선택적-세트(Selective-set) 연산
- 2진수의 특정 비트를 선택하여서 1로 세트시키는 연산
- 데이터 A가 1001 0010일 때, 하위 4비트 모두를 1로 세트하려고 함
- 데이터 B를 0000 1111로 하고 A와 OR 연산을 수행함

A = 1 0 0 1 0 0 1 0	연산 전
B = 0 0 0 0 1 1 1 1	선택적-세트(OR) 연산
A = 1 0 0 1 1 1 1 1	연산 후

2) 선택적-보수(Selective-complement) 연산
- 2진수의 특정 비트를 1의 보수로 변경 시키는 연산
- 즉, 지정된 비트가 반전됨
- 데이터 A의 특정 비트들을 1의 보수로 나타내기 위해서, 원하는 특정 비트위치가 1로 세트 된 데이터 B와 XOR 연산을 수행함
- 데이터 A의 하위 4비트를 비트반전 시키기 위해서, 데이터 B를 0000 1111로 하고 데이터 A와 XOR 연산을 수행함

A = 1 0 0 1 0 0 1 0	연산 전
B = 0 0 0 0 1 1 1 1	선택적-보수(XOR) 연산
A = 1 0 0 1 1 1 0 1	연산 후

4. 컴퓨터 응용 논리 연산(2)

1) 마스크(Mask) 연산
- 원하는 비트들을 선택적으로 clear(0)하는데 사용하는 연산
- 데이터 A의 특정 비트들을 0으로 바꾸기 위해서, 원하는 특정 비트위치가 0로 세트 된 데이터 B와 AND 연산을 수행함
- 데이터 A의 상위 4비트를 0으로 clear하기 위해서, 데이터 B를 0000 1111로 하고 A 레지스터와 AND 연산을 수행함

A = 1 0 1 1 0 1 0 1	연산 전
B = 0 0 0 0 1 1 1 1	마스크(AND) 연산
A = 0 0 0 0 0 1 0 1	연산 후

2) 삽입(Insert) 연산
- 2진 데이터내의 특정 위치에 새로운 비트 값들을 삽입하는 연산
- 마스크 연산과 선택적 세트연산을 순차적으로 수행함으로써 완성됨
- 삽입할 비트 위치들에 마스크 연산, 새로이 삽입할 비트들과 OR 연산을 수행함
- 데이터 A = 1011 1010의 하위 4비트에 1100을 삽입하는 경우

A = 1 0 1 1 1 0 1 0	
B = 1 1 1 1 0 0 0 0	마스크(AND) 연산
A = 1 0 1 1 0 0 0 0	마스크 결과
B = 0 0 0 0 1 1 0 0	삽입(OR) 연산
A = 1 0 1 1 1 1 0 0	최종 삽입 결과

5. 컴퓨터 응용 논리 연산(3)

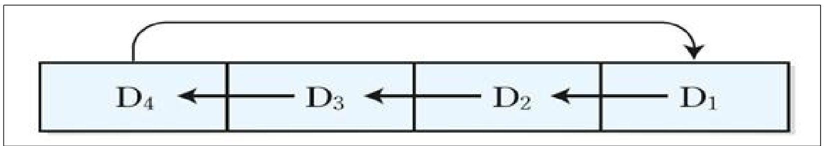
1) 비교(compare) 연산
- 두 데이터를 비교하는 연산으로 exclusive-OR 연산에 의해서 구현됨
- 데이터 A와 B의 내용을 비교 만약 대응되는 비트들의 값이 같으면, 데이터 A의 해당 비트를 0으로 세트
- 서로 다르면, 데이터 A의 해당 비트를 1로 세트
- A =1110 0001과 B=0101 1001과를 비교

A = 1 1 1 0 0 0 0 1	연산 전
B = 0 1 0 1 1 0 0 1	비교(XOR) 연산
A = 1 0 1 1 1 0 0 0	연산 후

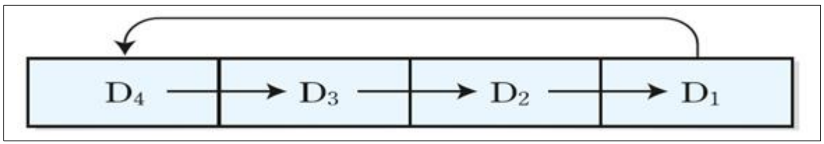
- 연산의 결과는 두 데이터가 같으면 0을 출력하고 다르면 1을 출력함

6. 컴퓨터 응용 논리 연산(4)

1) 순환 이동(Circular Shift)
- 최상위 혹은 최하위에 있는 비트가 반대편 끝에 있는 비트 위치로 이동해서 비트가 회전함
- 순환 좌측-이동(circular shift-left) : 최상위 비트인 D_4가 최하위 비트 위치인 D_1으로 이동



- 순환 우측-이동(circular shift-right) : 최하위 비트인 D_1 이 최상위 비트 위치인 D_4로 이동



7. 컴퓨터 응용 논리 연산(5)

1) 산술적 이동(arithmetic shift)
- 이동 과정에서 부호 비트는 유지하고, 수의 크기를 나타내는 비트들만 이동함

① 산술적 좌측-이동
- D_4 (불변), $D_3 \leftarrow D_2$, $D_2 \leftarrow D_1$, $D_1 \leftarrow 0$

② 산술적 우측-이동
- D_4 (불변), $D_4 \rightarrow D_3$, $D_3 \rightarrow D_2$, $D_2 \rightarrow D_1$

* 예시 : 산술적 이동 예

A = 1 0 1 0 1 1 1 0 : 초기 상태
1 1 0 1 1 1 0 0 : A의 산술적 좌측-시프트 결과
1 1 0 1 0 1 1 1 : A의 산술적 우측-시프트 결과

【학습정리】

1. 2진수 산술 연산에서 중요한 것은 오버플로우 발생시 처리 방법이다.

2. 2진수 논리연산의 기본 개념은 논리게이트의 연산과도 연결되므로 개념 정리가 확실히 되어야 한다.