컴퓨터 산술과 논리 연산 (2)
(제 6주 차)

서울사이버대학교

오 창 환

학습 목표

- 정수의 산술 연산을 설명할 수 있다.
- 부동소수점 수의 표현을 설명할 수 있다.
- 부동소수점 산술 연산을 설명할 수 있다.

학습 내용

- 정수의 산술 연산
- 부동소수점 수의 표현
- 부동소수점 산술 연산

정수의 산술 연산 (1)

(1) 덧셈

- 2의 보수로 표현된 수들의 덧셈 방법은 먼저 두 수를 더하고, 만약 올림수가 발생하면 버리면 됨.
- 만약 덧셈 결과값의 부호가 1이라면, 그 값은 2의 보수로 표현된 음수임.
- 아래와 같은 예가 있음.

(+3) + (+4) = +7

```
  0011
+ 0100
------
  0111  = +7
```

(−3) + (+3) = 0

```
  1101
+ 0011
------
버림←①  0000  = 0
```

(−6) + (+2) = −4

```
  1010
+ 0010
------
  1100  = −4
```

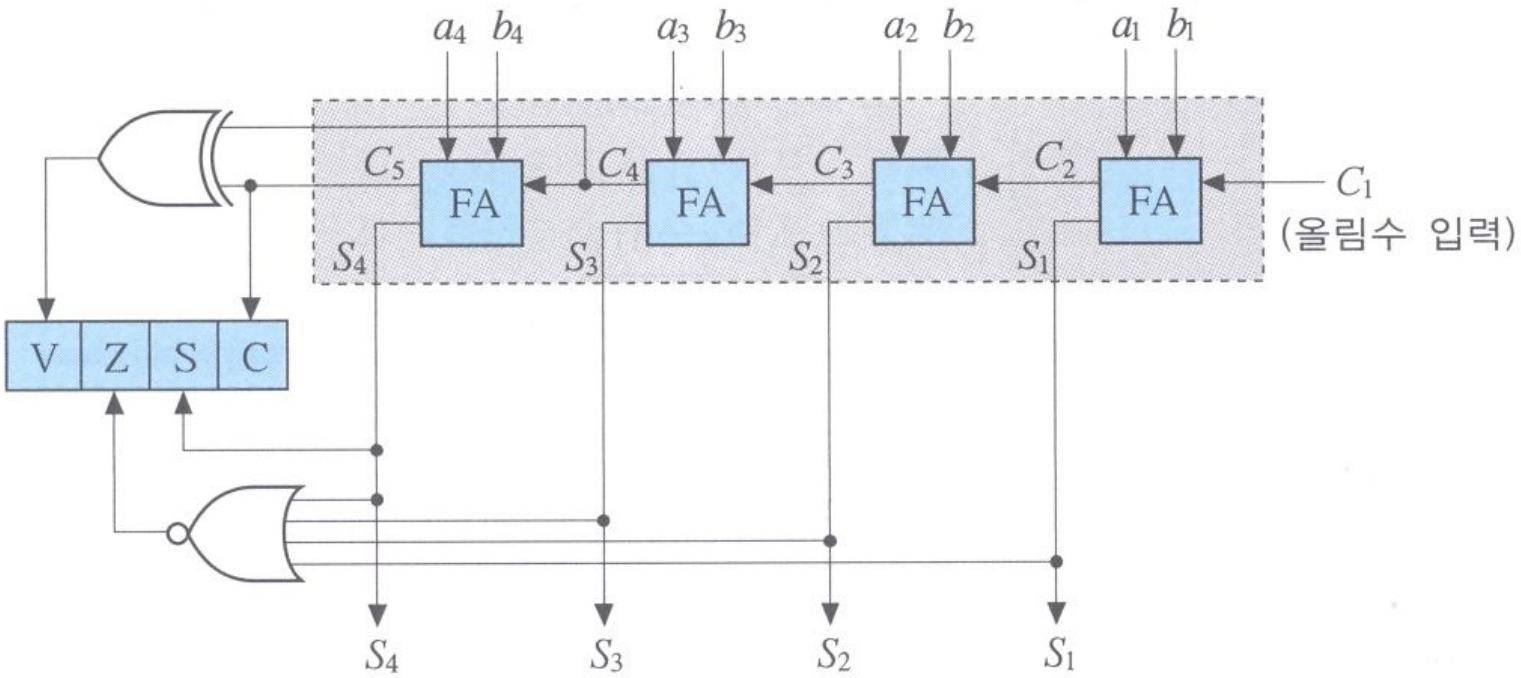
(−4) + (−1) = −5

```
  1100
+ 1111
------
버림←①  1011  = −5
```

정수의 산술 연산 (2)

- 덧셈을 수행하는 하드웨어를 병렬 가산기(parallel adder)라고 부르는데, 이것은 데이터의 비트 수만큼의 전가산기(full-adder)들로 구성됨.
- 예를 들어서 4 비트 병렬 가산기는 아래 그림과 같이 네 개의 전가산기들로 이루어짐.
- 전가산기들은 서로간에 올림수 비트(carry bit)를 전송하는 선으로 연결되는데, 하위 단계의 전가산기에서 발생한 올림수가 상위 단계 전가산기의 올림수 입력이 되도록 연결됨.
- 합의 최상위 비트는 부호 비트이므로 부호(S) 플래그로 직접 연결되고, 양수이면 S 플래그가 0으로, 음수이면 1로 세트 됨.
- 덧셈 결과에 대한 올림수 (C) 플래그는 최상위 단계의 전가산기로부터 발생하는 올림수(C_5)에 의해 세트됨.
- 모든 비트들을 NOR 게이트를 통과시켜서, 영(zero)을 나타내는 Z 플래그를 세트 함.

정수의 산술 연산 (3)



• 4 비트 병렬 가산기와 상태 비트 제어회로

정수의 산술 연산 (4)

- 덧셈 과정에서 수의 표현 범위가 초과된 경우에는 전혀 틀린 결과를 산출하게 되는데 예를 들어서, 2의 보수로 표현된 4 비트 데이터의 표현 범위는 -8부터 +7까지 이지만 아래 예에서와 같이 덧셈 결과가 그 범위를 초과하게 되면 결과값이 틀리게 되는데, 이것을 오버플로우(overflow)라고 함.
- 덧셈 오버플로우 예

(+6) + (+3) = +9

```
  0110
+ 0011
------
  1001  = -7 (오버플로우)
```

(-7) + (-6) = -13

```
  1001
+ 1010
------
버림←①  0011  = +3  (오버플로우)
```

- 오버플로우는 덧셈 과정에서 최상위 비트들 및 그 다음 비트들의 덧셈 과정에서 발생하는 올림수들 (그림의 C_5와 C_4)이 서로 다른 경우에 발생함.
  따라서 덧셈 과정에서의 오버플로우(V)는 다음과 같이 두 올림수들 간의 XOR를 수행하여 검출할 수 있음.

$$V = C_4 \oplus C_5$$

정수의 산술 연산 (5)

(2) 뺄셈

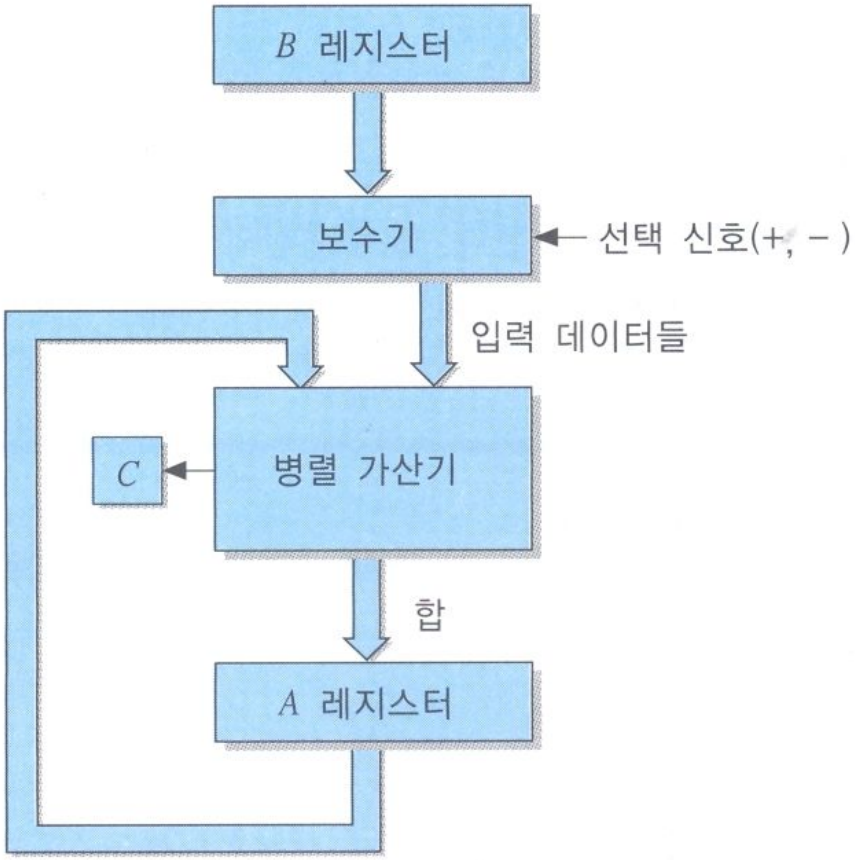
- 정수들의 뺄셈은 다음과 같이 덧셈을 이용하여 수행할 수 있음.

  A – (+B) = A + (–B)

  A – (–B) = A + (+B)

- 뺄셈은 덧셈을 이용하여 수행되므로 ALU에 뺄셈을 위한 하드웨어를 별도로 두지 않고 가산기를 이용하여 뺄셈을 함.
- 아래 그림에서 A 레지스터로부터 들어오는 수(A)는 병렬 가산기로 직접 입력되고, B 레지스터로부터 들어오는 수(B)는 보수기를 통하여 병렬 가산기로 들어오게 되는데 보수기는 입력 데이터에 대한 2의 보수를 출력하는 회로임.
- 보수기로는 제어 신호가 가해지는데, 덧셈인 경우에는 그 신호가 0이 되어 입력 데이터 A를 그대로 통과시키고, 뺄셈인 경우에는 제어 신호로 1이 가해져서 입력 데이터 B가 보수기에 의해 2의 보수로 변환된 다음에 병렬 가산기로 입력되며, 덧셈의 결과는 다른 레지스터가 아닌 A 레지스터에 저장됨.
- 뺄셈에서도 그 결과가 표현 가능한 범위를 초과한 경우에는 오버플로우가 발생함.

정수의 산술 연산 (6)



• 덧셈과 뺄셈 겸용 하드웨어의 블록 구성도

2 교시

정수의 산술 연산 (7)

(3) 곱셈

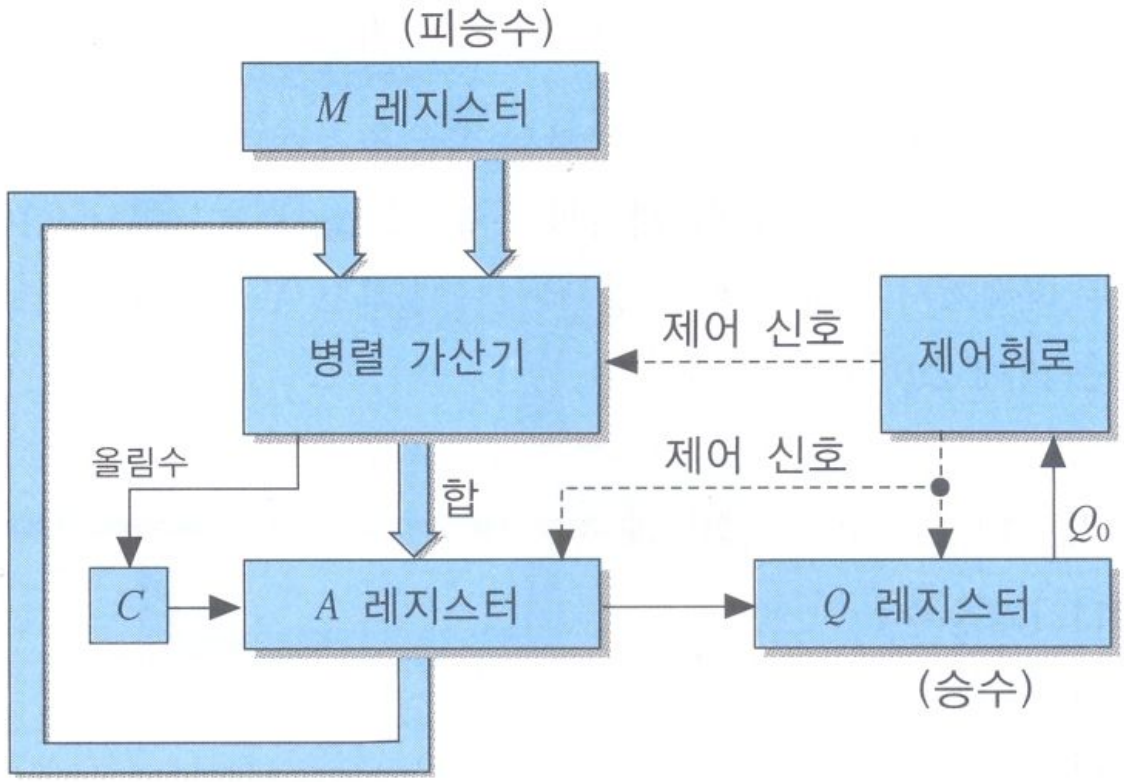
- 덧셈이나 뺄셈에 비하여, 곱셈은 하드웨어로 처리되든 소프트웨어로 처리되든 복잡한 연산임.
- 아래 예에서는 부호 없는 4 비트 정수들을 곱하는 과정을 보여주고 있는데 이 과정에서 승수의 각 비트에 대해 한 개씩의 부분 적이 발생되며, 부분 적들이 모두 더해져서 최종 결과가 발생됨. 그리고 두 개의 n비트 2진 정수들을 곱하면 결과값은 최대 2n비트가 됨.

```
    1011 (피승수)
  x 1101 (승수)
  ------
    1011
   0000   (부분 적들)
  1011
 1011
--------
10001111 (최종 결과)
```

정수의 산술 연산 (8)

- 컴퓨터에서 곱셈을 수행할 때는 부분 적이 발생할 때마다 그들의 합을 구해나감으로써 부분 적들을 저장해두기 위한 레지스터의 수를 최소화함. 또한 부분 적을 구하는 시간을 절약하기 위해 승수의 비트가 1일 경우에는 덧셈과 쉬프트를 하고, 0인 경우에는 쉬프트만 함.
- 아래 승산기 그림에서 그 중심에는 병렬 가산기가 위치하고, 그 입력들은 피승수가 저장되어 있는 M 레지스터와 결과값이 저장되는 A 레지스터로부터 들어옴. 그리고 가산기의 출력은 다시 A 레지스터에 저장되며, 이때 덧셈 과정에서 발생한 올림수는 1 비트 길이의 C 레지스터에 저장됨.
- C 및 A 레지스터는 승수가 저장되어 있는 Q 레지스터와 직렬로 연결되어 있어서 전체적으로 쉬프트 연산을 수행할 수 있도록 구성됨.
- Q 레지스터의 최하위 비트인 Q_0 비트는 제어회로로 입력되는데 제어회로는 Q_0 비트를 검사하여, 그 결과에 따라 적절한 제어 신호들을 발생함으로써 곱셈 연산이 순차적으로 수행되도록 해 줌.

정수의 산술 연산 (9)



• 부호 없는 정수 승산기의 하드웨어 구성도

정수의 산술 연산 (10)

• 상기 예의 곱셈이 수행되는 동안 각 사이클에서는 다음과 같이 연산이 진행됨.

[초기 상태] M 레지스터에 피승수인 1011, Q레지스터에 승수인 1101을 각각 저장하며, 결과값이 저장될 A 레지스터의 모든 비트들과 올림수(C) 비트를 0으로 초기화 함.

[사이클 1] Q 레지스터의 최하위 비트(Q_0)를 제어 회로로 보내어 검사하는데, 이 예에서는 그 비트가 1이므로 M 레지스터의 피승수 1011을 A레지스터의 내용과 더하고, 결과는 다시 A 레지스터에 저장함. 그런 다음에 직렬 연결된 상태인 A 및 Q 레지스터를 하나의 8비트 레지스터로 간주하여 오른쪽으로 한 비트씩 쉬프트 함. 이때 올림수 C 비트는 A 레지스터의 최상위 비트로 들어오며, Q 레지스터의 최하위 비트는 버림.

[사이클 2] Q_0가 0이므로, 덧셈은 하지 않고 A 및 Q 레지스터를 오른쪽으로 한 비트씩 쉬프트 함.

정수의 산술 연산 (11)

[사이클 3] Q_0가 1이므로, [사이클 1]에서와 같이 피승수와 A 레지스터 내용을 더하고, A 및 Q 레지스터를 오른쪽으로 한 비트씩 쉬프트 함.

[사이클 4] Q_0가 1이므로, [사이클1]에서와 같이 피승수와 A 레지스터 내용을 더하고 A 및 Q 레지스터를 오른쪽으로 한 비트씩 쉬프트 함.

최종 결과값의 상위 4 비트인 1000은 A 레지스터에,

그리고 하위 4 비트인 1111은 Q 레지스터에 각각 저장되어 있음.

정수의 산술 연산 (12)

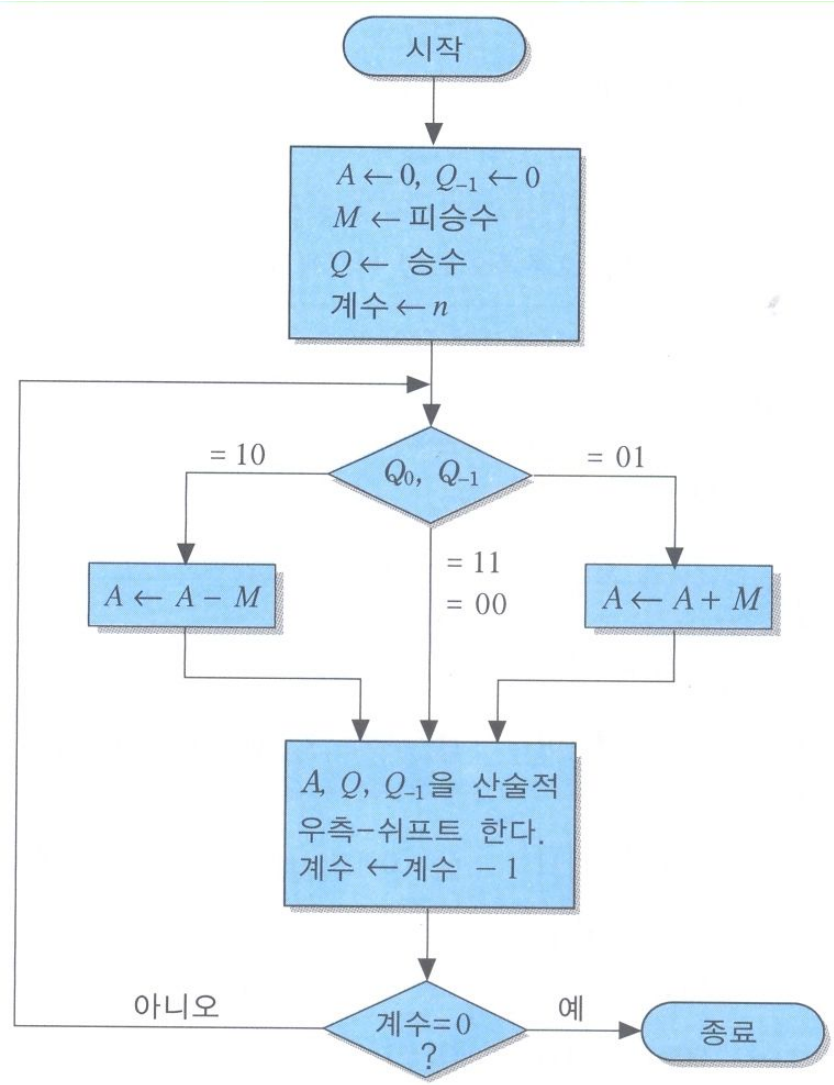
	C	A	Q	
[초기 상태]	0	0000	1101	
[사이클 1]	0	1011	1101	; $Q_0 = 1$이므로, $A \leftarrow A + M$.
	0	0101	1110	; 우측 쉬프트(C–A–Q)
[사이클 2]	0	0010	1111	; $Q_0 = 0$이므로, 쉬프트(C–A–Q)만 한다.
[사이클 3]	0	1101	1111	; $Q_0 = 1$이므로, $A \leftarrow A + M$.
	0	0110	1111	; 우측 쉬프트(C–A–Q)
[사이클 4]	1	0001	1111	; $Q_0 = 1$이므로, $A \leftarrow A + M$.
	0	1000	1111	; 우측 쉬프트(C–A–Q)

- 곱셈이 수행되는 동안의 레지스터 내용들

정수의 산술 연산 (13)

- 2의 보수들 간의 곱셈 방법으로 가장 널리 사용되고 있는 Booth 알고리즘은 기존의 곱셈 회로를 다음과 같이 수정하면 됨.
  ① M 레지스터와 병렬 가산기 사이에 보수기를 추가함.
  ② Q 레지스터의 우측에 Q_{-1} 이라고 부르는 1 비트 레지스터를 추가하고, 그 출력이 Q_0와 함께 제어 회로로 입력되도록 함.
- 앞에서 설명한 부호 없는 곱셈의 경우와 마찬가지로, 피승수와 승수는 M 레지스터와 Q 레지스터에 각각 저장되고, Q 레지스터의 최하위 비트인 Q_0의 오른쪽에는 Q_{-1} 이라는 한 비트 길이의 레지스터가 추가됨.
  A 레지스터와 Q_{-1} 비트는 0으로 초기화되며, 곱셈의 결과는 A 및 Q 레지스터에 저장됨.
- 제어 회로는 승수의 비트들을 검사할 때, Q_0뿐만 아니라 그 우측 비트(Q_{-1})도 함께 검사함.
- 이러한 과정은 승수의 비트 수 만큼인 n번 반복됨.

정수의 산술 연산 (14)



• Booth 알고리즘의 흐름도

정수의 산술 연산 (15)

• Booth 알고리즘을 이용한 곱셈 (−7 x 3)의 예

(M) 1001 초기값 : A = 0000, Q_{-1}=0, 계수 =4

(Q) x 0011

	A	Q	Q_{-1}	계수	
(AQ)	0111	0011	0		; (Q_0 Q_{-1}) = (10)이므로, A로부터 피승수 1001을 빼는데, 실제로는 그 보수인 0111을 더함.
	0011	1001	1	3	; AQQ_{-1}을 산술적 우측−쉬프트 하고, 계수에서 1을 뺌.
	0001	1100	1	2	; (Q_0 Q_{-1}) = (11)이므로, AQQ_{-1}에 대한 산술적 우측 쉬프트만 하고, 계수에서 1을 뺌.
	1010	1100	1		; (Q_0 Q_{-1}) = (01)이므로, A에 피승수 1001을 더함.
	1101	0110	0	1	; AQQ_{-1}을 산술적 우측−쉬프트 하고, 계수에서 1을 뺌.
	1110	1011	0	0	; (Q_0 Q_{-1}) = (00)이므로, AQQ_{-1}을 산술적 우측−쉬프트 함. 계수에서 1을 빼면 0이므로 계산이 종료되었음.

→ −21 (곱셈 결과)

정수의 산술 연산 (16)

(4) 나눗셈

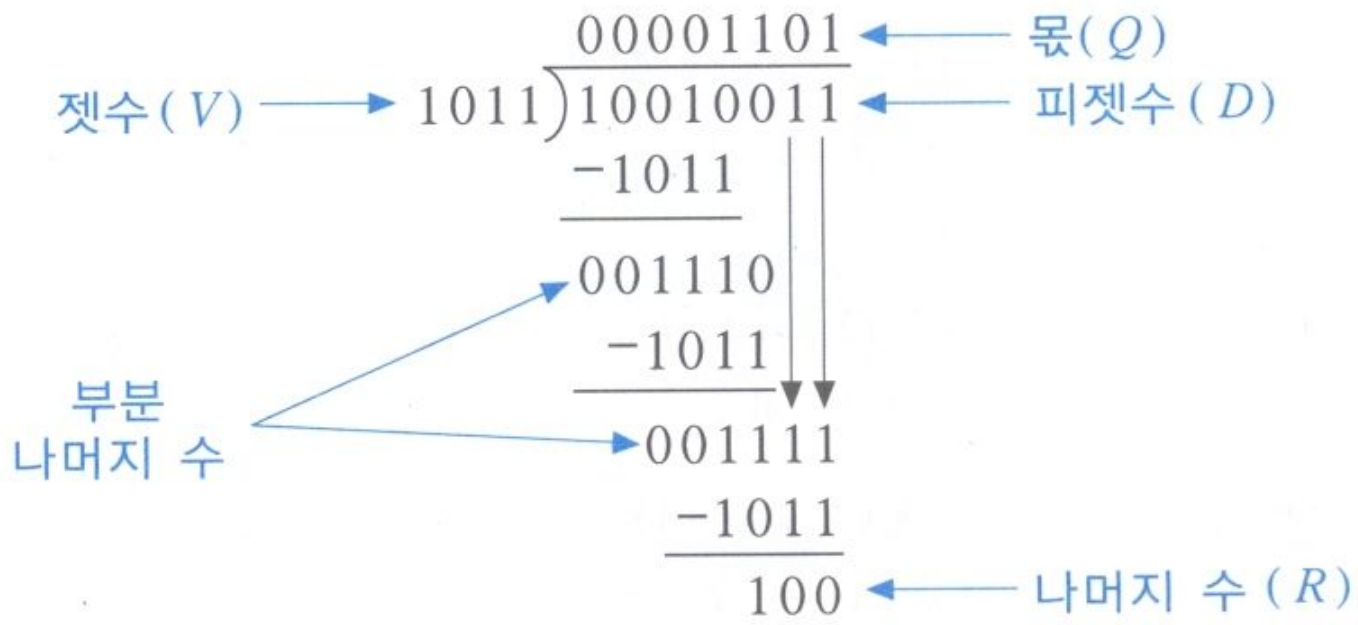
- 나눗셈은 곱셈보다 다소 더 복잡하지만 동일한 원리에 근거를 두고 있음. 즉, 곱셈에서와 마찬가지로 나눗셈의 알고리즘도 반복적인 쉬프트와 덧셈 또는 뺄셈으로 이루어짐.
- 아래 그림은 부호 없는 2진 정수에 대한 나눗셈의 예를 보여주고 있는데, 먼저 피젯수의 비트들을 좌측에서부터 우측으로 차례대로 검사하여서, 그 비트들이 나타내는 값이 젯수보다 같거나 큰 경우가 될 때(젯수가 피젯수를 나눌 수 있게 될 때)까지 한 비트씩 이동하면서 검사를 반복함.

  그렇게 될 때까지는 몫의 좌에서 우로 가면서 0이 채워지며, 나눌 수 있게 되면 몫에 1이 주어지고 부분 피젯수의 값에서 젯수를 빼는데 그 결과를 부분 나머지 수라 함.

  여기서부터 나눗셈은 사이클 형태를 이루는데 각 사이클에서 피젯수로부터의 한 비트씩이 부분 나머지에 추가되며, 그 결과값이 젯수와 같거나 커질 때까지 그 과정이 반복됨.

  그리고 새로운 부분 나머지 수를 얻기 위해 이 수로부터 젯수를 빼는데 이 과정은 피젯수의 모든 비트들에 대해 적용될 때까지 계속됨.

정수의 산술 연산 (17)



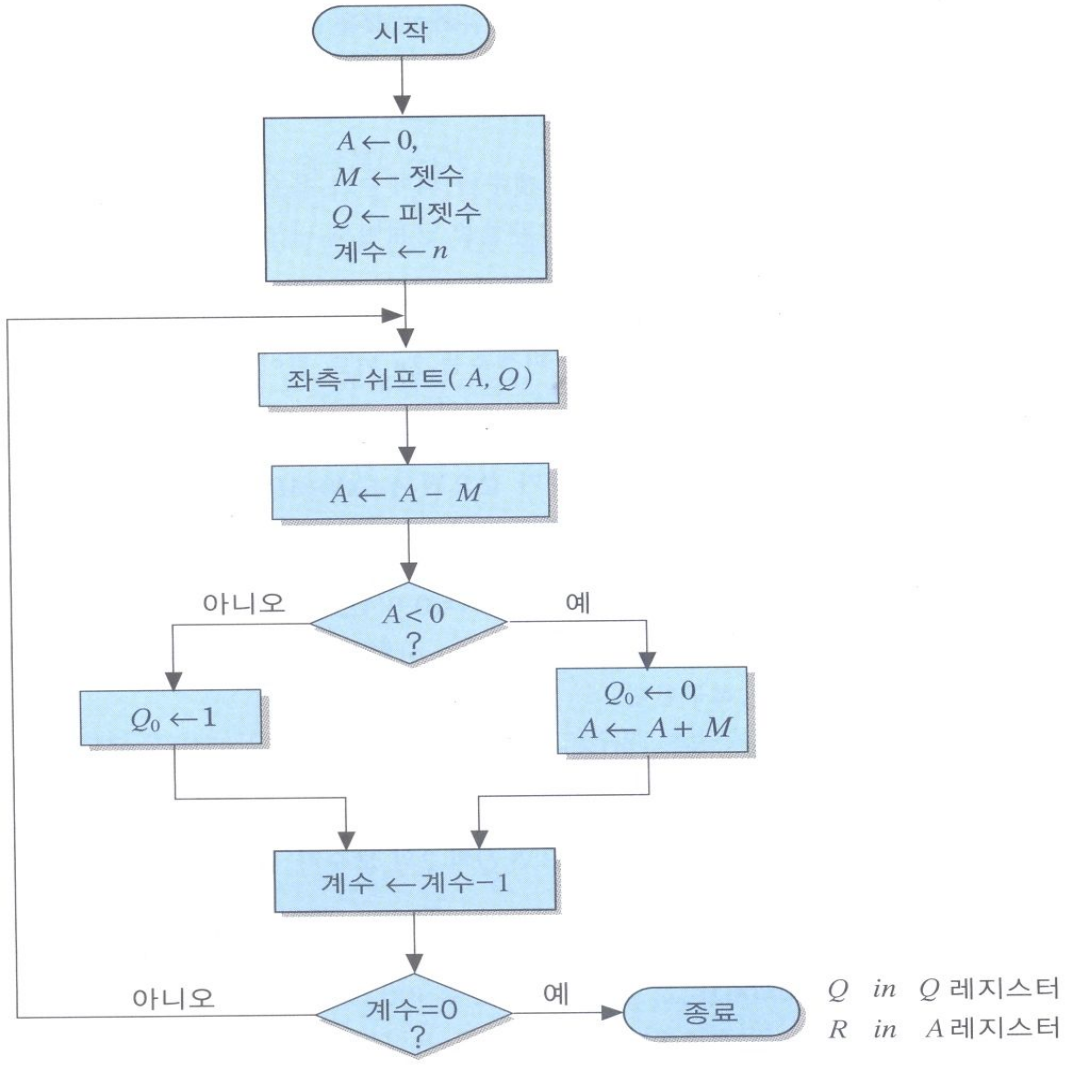
• 부호 없는 2진 정수의 나눗셈

정수의 산술 연산 (18)

- 아래 그림은 나눗셈에 대한 컴퓨터 알고리즘을 보여주고 있음.
- 젯수는 M 레지스터에, 피젯수는 Q 레지스터에 각각 저장함.
- 각 단계에서 A와 Q 레지스터 모두 왼쪽으로 한 비트씩 쉬프트 함.
- A가 부분 나머지 수를 나눌 수 있는지 알기 위하여 A로부터 M을 뺌.
- 만약 나눌 수 있는 경우, Q_0는 1의 값을 가지고,그렇지 않은 경우에는 Q_0는 0이 되고 M의 이전 값을 복구시키기 위하여 M을 A에 다시 더함. 이 과정이 끝나면 계수를 1 감소하고, 이 과정을 n번 반복함.

  결국 몫은 Q 레지스터에, 나머지는 A 레지스터에 남게 됨.

정수의 산술 연산 (19)



• 부호 없는 2진 나눗셈 알고리즘의 흐름도

정수의 산술 연산 (20)

• 상기의 알고리즘을 약간 수정하면 젯수나 피젯수가 음수인 경우에도 적용할 수 있는데, 2의 보수로 표현된 수들간의 나눗셈 과정을 단계별로 나누어 설명하면 아래와 같음.

[초기 상태] 젯수는 M 레지스터에, 피젯수는 A와 Q 레지스터에 저장함. 각 레지스터가 n비트일 때, 피젯수는 2n 비트 길이의 2의 보수로 나타냄.

[사이클 1] A와 Q 레지스터를 좌측으로 한 비트씩 쉬프트 함.

[사이클 2] 만약 M과 A의 부호가 같으면 A ← A−M, 다르면 A ← A+M을 수행함.

[사이클 3] 연산 전과 후의 A의 부호가 같으면 위의 연산은 성공임.

(a) 연산이 성공이거나 A=0이면, Q_0 ←1로 세트 함.

(b) 연산이 실패이고 A ≠ 0이면, Q_0 ← 0으로 하고, A를 이전의 값으로 복구함.

[사이클 4] Q에 비트 자리 수가 남아 있다면, 단계 2에서 4까지를 반복함.

[사이클 5] 나머지 수는 A에 남음. 만약 젯수와 피젯수의 부호가 같으면 몫은 Q의 값이고, 그렇지 않으면 Q의 2의 보수가 몫이 됨.

정수의 산술 연산 (21)

A	Q	$M = 1101$ (-3)
0000	0111	; 초기 상태
0000	1110	; 좌측-쉬프트($A-Q$).
1101		; A와 M의 부호가 서로 다르므로, $A \leftarrow A + M$.
0000	1110	; A의 부호가 바뀌었으므로, A의 원래값을 복구.
0001	1100	; 좌측-쉬프트($A-Q$).
1110		; A와 M의 부호가 서로 다르므로, $A \leftarrow A + M$.
0001	1100	; A의 부호가 바뀌었으므로, A의 원래값을 복구.
0011	1000	; 좌측-쉬프트($A-Q$).
0000		; A와 M의 부호가 서로 다르므로, $A \leftarrow A + M$.
0000	1001	; $A = 0$이므로, $Q_0 \leftarrow 1$로 세트.
0001	0010	; 좌측-쉬프트($A-Q$).
1110		; A와 M의 부호가 서로 다르므로, $A \leftarrow A + M$.
0001	0010	; A의 부호가 바뀌었으므로, A의 원래값을 복구.

연산 결과 : 몫 = Q의 내용(0010)에 대한 2의 보수 = 1110 (−2)
나머지 = A의 내용 = 0001 (+1)

• 2의 보수 나눗셈의 예 : (7) ÷ (−3)

3 교시

부동소수점 수의 표현 (1)

- 2진수 표현 방법들을 사용하면 양수와 음수를 모두 나타낼 수 있고, 또한 그 표현 방식에서 2진 소수점을 사용하면 소수도 표현할 수 있지만, 이 방법으로는 표현 가능한 수의 범위에 한계가 있어서, 아주 큰 수와 매우 작은 소수는 표현할 수 없음.
- 10진수에서는 과학적 표기를 사용하여 그러한 문제를 극복할 수 있는데, 예를 들어서 274,000,000,000,000은 2.74×10^{14} 으로, 0.00000000000274는 2.74×10^{-12}으로 표기할 수 있음.
- 이와 같이 소수점의 위치가 필요에 따라 이동하는 표현 방법을 부동소수점 표현이라고 하며, 그와 같이 표현된 수를 부동소수점 수라고 부르는데, 이 표현 방법을 사용하면 매우 큰 수와 작은 수도 간결하게 표현할 수 있음.

부동소수점 수의 표현 (2)

- 부동소수점 수의 일반적인 형태는 아래와 같음.

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

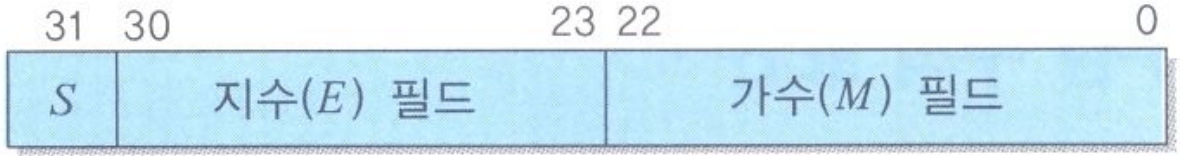
  여기서 S는 수의 부호, M은 가수, B는 기수, E는 지수를 각각 나타냄.

  이들 중에서 B는 10진수에서는 10으로, 2진수에서는 2로 고정되므로

  데이터 표현 형식에는 포함시키지 않아도 됨.

  예를 들어 2.74×10^{14} 에서 가수 M은 2.74이고, 지수 E는 +14, 그리고 기수 B는 10임.

- 디지털 컴퓨터에서는 2진수 체계가 사용되므로 위의 방식으로 표현하는 수는 2진 부동소수점 수가 되는데, 이 경우에도 구성 요소는 10진수의 경우와 동일하며 기수 B =2인 것만 다름.
- 아래 그림은 전형적인 32 비트 부동소수점 수의 한 형식인데, 1비트 부호, 8비트 지수, 23비트 가수로 구성되어 있음.



• 32 비트 부동소수점 형식의 예

부동소수점 수의 표현 (3)

- 이 형식에서는 가수(M)는 정밀도를 결정해주고, 지수(E)는 표현 가능한 수의 범위를 결정함. 즉, 가수 필드의 비트 수가 많아질수록 수를 더 정확하게 표현할 수 있으며, 지수 필드의 비트 수가 많아지면 2^E의 절대값의 최대값이 커지므로 표현할 수 있는 수의 범위가 증가하게 됨. 그러나 데이터 길이가 고정된 상태에서 어느 한 필드의 비트 수를 증가시키면 다른 필드의 비트 수가 감소하므로 적절히 조정하는 것이 필요함.
- 앞의 형식에서 지수가 2의 보수로 표현된다면 지수가 8비트이므로 그 범위는 $-2^7 \sim (2^7-1)$ 사이가 됨.
  가수 필드의 맨 좌측 비트의 왼쪽에 소수점이 있다고 가정한다면 23비트 가수로는 소수점 아래 23번째 자리수 (2^{-23})까지 표현할 수 있음.

부동소수점 수의 표현 (4)

- 부동소수점 수 표현에서는 어느 한 수에 대하여 여러 개의 표현들이 존재할 수 있는데, 예를 들어서, 아래의 표현들은 모두 같은 값을 나타냄.

  0.1101×2^5

  110.1×2^2

  0.01101×2^6

  이로 인한 혼란을 막는 방법으로 아래와 같은 정규화된 표현으로 나타내는 것임.

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

- 정규화된 표현이란 이와 같이 소수점의 바로 오른 편에 있는 비트가 반드시 1이 되도록 위치를 조정하는 것을 말하며, 그에 따라 지수가 결정됨.

  결과적으로 위의 수에 대한 정규화 표현은 0.1101×2^5이 됨.

- 컴퓨터 형식으로 저장하는 경우에는 소수점의 좌측에 있는 0은 포함시킬 필요가 없고, 소수점의 우측에 있는 비트들만 가수(M) 필드에 들어가게 되므로 가수의 크기는 항상 $0.5 \leq M \leq 1.00$이 됨.

부동소수점 수의 표현 (5)

- 결과적으로 0.1101×2^{5}을 컴퓨터 형식으로 표현하면 다음과 같음.

부호(S) 비트 = 0
지수(E) = 00000101
가수(M) = 1101 0000 0000 0000 0000 000

데이터 표현:

S	E	M
0	00000101	11010000000000000000000

- 그런데 이러한 정규화된 표현의 경우에 2진 소수점 아래의 첫 번째 비트는 항상 1이므로, 이 비트를 반드시 저장할 필요가 없고 후에 그 수를 이용한 산술 계산 과정에서 그 비트가 있는 것으로 처리해주면 될 것이므로 23 비트 가수 필드로 소수점 아래 24번째 비트까지 저장할 수 있게 됨.

부동소수점 수의 표현 (6)

- 지수를 바이어스 된 수(biased number)로 표현하면 2^E의 절대값이 거의 0에 가까운 수를 표현할 수 있음.

  바이어스가 128인 경우에 $M \times 2^{+2}$라는 수에 대한 지수 비트들은 +2(00000010)에 128 (10000000)을 더하면 10000100이 됨.

  $M \times 2^{-2}$ 에 대한 지수 비트들은 −2(11111110)에 128(10000000)을 더하면 01111110임.

지수 비트 패턴	절대값	실제 지수값	
		바이어스=127	바이어스=128
11111111	255	+ 128	+ 127
11111110	254	+ 127	+ 126
⋮	⋮	⋮	⋮
10000001	129	+ 2	+ 1
10000000	128	+ 1	0
01111111	127	0	− 1
01111110	126	− 1	− 2
⋮	⋮	⋮	⋮
00000001	1	− 126	− 127
00000000	0	− 127	− 128

• 8 비트 바이어스 된 지수값들

부동소수점 수의 표현 (7)

- 반대의 경우로서, 이 형식에 따라 표현된 지수 비트 패턴으로부터 실제 지수값을 찾아 내기 위해서는 예를 들어서 지수 필드가 129(10000001)라면, 실제 지수값은 128(10000000)을 뺀 +1(00000001)이 되어서 실제 수의 계산에서는 $M \times 2^{+1}$로 처리됨.
- 예로서, N = −13.625에 대한 부동소수점 표현은 아래와 같음.

$13.625_{10} = 1101.101_2 = 0.1101101 \times 2^4$

부호(S) 비트 = 1 (−)

지수(E) = 00000100 + 10000000 = 10000100 (바이어스 128을 더한다)

가수(M) = 10110100000000000000000 (소수점 우측의 첫 번째 1은 제외)

〈결과〉

S	E	M
1	10000100	10110100000000000000000

부동소수점 산술 연산 (1)

(1) 덧셈과 뺄셈

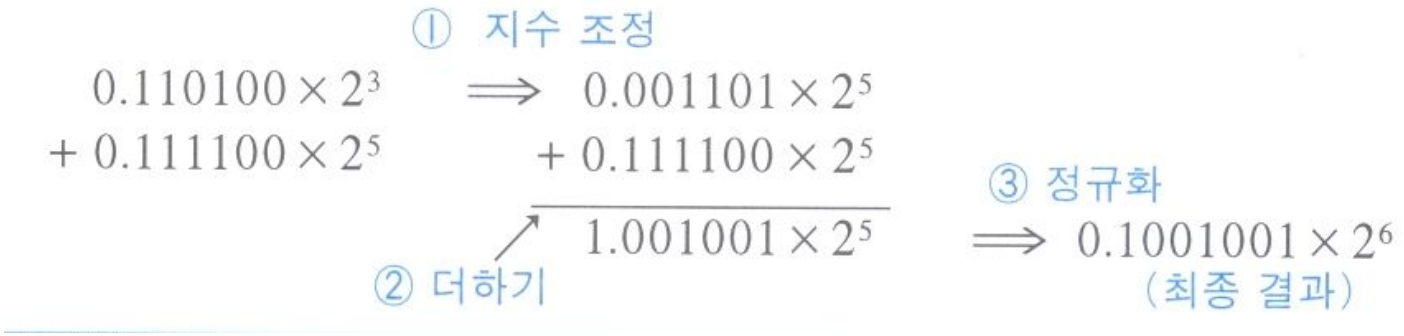
- 덧셈과 뺄셈 알고리즘은 다음과 같은 단계들로 이루어짐.

  ① 지수들이 일치되도록 조정함.

  ② 가수들을 더하기 혹은 빼기 함.

  ③ 결과를 정규화함.

- 소수점 위치 조정은 둘 중에 더 작은 수를 우측으로 (지수의 증가) 또는 더 큰 수를 좌측으로 (지수의 감소) 쉬프트 함으로써 수행됨.



- 부동소수점 덧셈의 예

부동소수점 산술 연산 (2)

(2) 곱셈과 나눗셈

• 2진 부동소수점 곱셈 과정은 다음과 같은 단계들로 이루어짐.

① 가수들을 곱함.

② 지수들을 더함

③ 결과를 정규화함.

$$(0.1011 \times 2^3) \times (0.1001 \times 2^5)$$

① 가수 곱하기 : $1011 \times 1001 = 01100011$

② 지수 더하기 : $3 + 5 = 8$

③ 정규화 : $0.01100011 \times 2^8 = 0.1100011 \times 2^7$ (결과값)

• 부동소수점 곱셈의 예

셀프 테스트

• 2의 보수 형태로 -5 + (+6)을 계산할 때에 버림이 발생하면 1을, 발생하지 않으면 0을 기입하라.

* 1

해설) -5에 대한 2의 보수는 1011이고 +6에 대한 2의 보수는 0110 이므로 이들 두 수, 즉 1011+0110 = 10001 에서 왼쪽 끝의 1은 버려서 계산 결과는 0001이 됨.

• Booth 알고리즘에서 Q레지스터에 저장되는 것은 곱셈의 무엇인가?

* 승수

해설) Booth 알고리즘에서 Q레지스터에는 곱셈의 승수가 입력됨.

• 바이어스가 128로 표현된 부동소수점 수의 지수값이 135인 경우에 실제 지수값은 얼마인가?

* 7

해설) 바이어스가 128로 표현된 부동소수점의 지수값이 135이면 실제 지수값은 135-128 = 7 이 됨.

요점 정리

- 2의 보수로 표현된 수들의 덧셈 방법은 먼저 두 수를 더하고, 만약 올림수가 발생하면 버리면 되고, 덧셈 결과값의 부호가 1이라면, 그 값은 2의 보수로 표현된 음수임.
- 덧셈을 수행하는 하드웨어를 병렬 가산기(parallel adder)라고 부르는데, 이것은 데이터의 비트 수만큼의 전가산기(full-adder)들로 구성됨.
- 컴퓨터에서 곱셈을 수행할 때는 부분 적이 발생할 때마다 그들의 합을 구해나감으로써 부분 적들을 저장해두기 위한 레지스터의 수를 최소화함.
- 나눗셈의 알고리즘도 반복적인 쉬프트와 덧셈 또는 뺄셈으로 이루어짐.
- 소수점의 위치가 필요에 따라 이동하는 표현 방법을 부동소수점 표현이라고 하며, 그와 같이 표현된 수를 부동소수점 수라고 부르는데, 이 표현 방법을 사용하면 매우 큰 수와 작은 수도 간결하게 표현할 수 있음.
- 부동소수점 수의 일반적인 형태는 $N = (-1)^{S} M \times B^{E}$ 임.
- 정규화된 표현이란 소수점의 바로 오른 편에 있는 비트가 반드시 1이 되도록 위치를 조정하는 것을 말하며, 그에 따라 지수가 결정됨.