05장 타겟 머신의 구조

5.2.1 컴퓨터 산술

예 5.15

16진수

n자리의 부호가 없는 이진수 $d_{n-1}d_{n-2}...d_2d_1d_0$의 값은 $\sum_{0 \le i < n} d_i 2^i$다. 주어진 십진 값과 일치하는 비트 패턴이 명백하지 않으며 0과 1의 나열로 나타낸 비트 패턴은 번거롭기 때문에 컴퓨터 과학자들은 일반적으로 정수 값을 16진수(hexadecimal)로 나타낸다. 16진수에서 글자 a~f는 6개의 추가적인 자릿수로 이용되며 각기 십진 수 10~15의 값을 나타낸다(◎심화학습에 있는 그림 5.7을 보자). 2^4=16이므로 16진수의 각 자릿수는 이진수의 4비트와 정확히 대응되며, 이 덕분에 16진수와 이진수 간의 변환은 매우 쉽다. 아스키로 더 쉽게 표현하기 위해 16진수는 뒷부분에 16이라는 첨자를 다는 대신 가끔 앞에 `0x`를 붙인다. ◎(심화학습에 있는) 그림 5.7을 참조하면 16진수 값 `0xabcd`가 이진 값 `1010 1011 1100 1101`과 일치함을 알 수 있다. 이와 마찬가지로 `0x400` = 2^{10} = `1024`로 보통 `1K`로 나타내며, `0x100000` = 2^{20} = `1048576`으로 보통 `1M`로 나타낸다.

설계와 구현

메가바이트는 어느 정도인가?

$2^{10} \sim 10^3$이라는 사실은 “간단한” 어림 계산은 쉽게 해주지만 정확도가 요구될 때는 때때로 혼란을 야기할 수 있다. 1K나 1M은 어떤 의미를 의도한 것일까? 안타깝게도 답은 문맥에 따라 다르다. 주 메모리 크기와 주소는 전형적으로 2의 제곱으로 측정하는 반면 다른 양은 10의 제곱으로 측정한다. 그러므로 1GHz, 1GB 개인용 컴퓨터는 1초에 1,000,000,000번 새로운 명령어를 시작할 수 있지만 1,073,741,824바이트의 메모리를 가진다. 100GB 하드디스크는 10^{11}바이트를 가진다.

0 0 0 0	0	1 0 0 0	8
0 0 0 1	1	1 0 0 1	9
0 0 1 0	2	1 0 1 0	a
0 0 1 1	3	1 0 1 1	b
0 1 0 0	4	1 1 0 0	c
0 1 0 1	5	1 1 0 1	d
0 1 1 0	6	1 1 1 0	e
0 1 1 1	7	1 1 1 1	f

그림 5.7 | 16진수

2의 보수

예 5.16

2의 보수

부호가 있는 정수에 대한 가장 명백한 표현은 한 비트를 부호(+ 또는 −)로 사용하고 부호가 없는 숫자로 크기를 나타내기 위해 나머지 `n-1` 비트를 사용하는 것이다. 불행히도 이 접근 방식은 덧셈과 뺄셈에 서로 다른 알고리즘(그리고 그러므로 별도의 회로)을 필요로 한다. 거의 보편적으로 쓰이는 방법은 2의 보수 산술이다. 이 방법은 오버플로우를 무시했을 때 부호가 없는 n자리 수에 대한 산술이 실제로 수학자들이 정수 모듈로 2^n 고리라고 부르는 것이 대한 산술이라는 점을 이용한다. 예를 들어 `A+B`는 실제로 `(A+B) mod 2`^{n}이다. 그러나 산술 중인 비트 패턴을 숫자 $0...2^n-1$로 해석해야 할 특별한 이유는 없다. 실제로 숫자의 나열에서 2^n개 정수로 된 연속적인 범위를 아무거나 선택한 후 이에 대해 나머지 연산을 수행 중이라고 할 수도 있다. 특히 범위 $-2^{n-1}...2^{n-1}-1$을 선택할 수 있다.

가장 작은 n 자리 2의 보수 값인 -2^{n-1}은 1과 그 뒤로 이어지는 n−1개의 0으로 나타낸다. 그 다음 값들은 반복적으로 일반적인 자리−값 덧셈을 사용해서 1씩 더해가면서 얻을 수 있다. 이러한 표현 방법은 몇 가지의 바람직한 특징을 가진다.

1. 0과 양수는 부호가 없는 형식과 동일한 비트 패턴을 가진다.
2. 음수의 최상위 비트는 항상 1이며 0과 양수의 최상위 비트는 항상 0이다.
3. 하나의 덧셈 알고리즘을 이용해서 음수와 음수가 아닌 수의 모든 조합을 더할 수 있다.

4비트 2의 보수의 목록은 ◎(심화학습에 있는) 그림 5.8과 같다.

0 1 1 1	7	1 1 1 1	−1
0 1 1 0	6	1 1 1 0	−2
0 1 0 1	5	1 1 0 1	−3
0 1 0 0	4	1 1 0 0	−4
0 0 1 1	3	1 0 1 1	−5
0 0 1 0	2	1 0 1 0	−6
0 0 0 1	1	1 0 0 1	−7
0 0 0 0	0	1 0 0 0	−8

그림 5.8 | 4비트 2의 보수. 대응되는 양수가 없는 음수 (−8)이 존재한다는 사실에 주의하자. 물론 0은 하나뿐이다.

예 5.17

2의 보수 덧셈에서의 오버플로우

부호가 없는 이진수와 2의 보수 이진수 모두에 대한 덧셈 알고리즘은 이미 익숙한 10진수의 왼쪽 방향 덧셈의 2진수 버전이다. 유일한 차이점은 오버플로우 발생 여부를 탐지하는 데 사용하는 기법이다. 정의상 두 정수(비트 패턴이 아니라 비트 패턴이 나타내는 실제 정수)의 합이 2^n비트로 나타낼 수 있는 범위를 벗어날 때마다 오버플로우가 발생해야 한다. 부호가 없는 정수의 경우 오버플로우는 최상위 자릿수에 자리 올림이 있을 때 발생한다. 2의 보수의 경우에는 최상위 자릿수로의 올림 값이 최상위 자릿수의 올림 값과 다를 때 오버플로우가 발생한다. 예를 들어 4비트의 2의 보수의 경우 `1100 + 0110(−4 + 6)`은 가장 왼쪽 자릿수로의 올림 값과 그 자릿수의 올림 값이 모두 1이므로 오버플로우가 아니다. 그러나 `0101 + 0100(5 + 4)`와 `1011 + 1100(−5 + −4)`는 모두 오버플로우를 야기한다. 첫 번째 경우 최상위 자릿수로의 올림 값과 이 자리의 올림 값이 1과 0이며, 두 번째 경우에는 0과 1이다.

서로 다른 기계는 다른 방법으로 오버플로우를 처리한다. 일부 기계는 오버플로우가 발생하면 트랩(인터럽트)을 생성한다. 어떤 기계는 소프트웨어에서 검사할 수 있는 비트를 설정한다. 또 어떤 기계는 앞선 두 선택 사항을 각기 지원하는 두 개의 `add` 명령어를 제공한다. 일부 기계는 특수 레지스터의 비트 값에 따라 두 선택 사항 중 하나를 수행할 수 있는 하나의 `add`를 제공한다.

2의 보수의 덧셈에 대한 역원은 모든 비트를 뒤집은 후(역자주: 0은 1로, 1은 0으로) 1을 더하고 가장 왼쪽 자리의 올림을 모두 버림으로써 얻을 수 있다. 뺄셈은 덧셈기를 사용해서 아주 쉽게하게 구현할 수 있는데, 우선 빼는 수의 비트를 뒤집고 최상위 자릿수로의 "올림"으로서 1을 더한 후 보통의 "덧셈"을 수행하면 된다. 부호가 있는 수의 곱셈과 나눗셈은 덧셈과 뺄셈보다는 다소 더 까다롭지만 여전히 이해하기 쉽다.

임의의 2의 보수와 이 수의 덧셈에 대한 역원을 부호가 없는 값인 것처럼 최종 올림 비트를 유지하면서 더하면 그 결과는 2^n이 된다는 점에 주목하자. 이 사실은 "2의 보수"라는 이름의 근원이다. 물론 올림 비트를 무시하면 `k + (−k)`의 값으로 예견할 수 있는 0을 얻는다.

부동소수점 실수

예 5.18

편향 지수

IEEE 754 표준은 두 가지 크기의 부동소수점 실수를 정의한다. 단정도 실수는 부호 비트, 8비트의 지수, 23비트의 유효수를 가진다. 단정도 실수는 대략 10^{-38}에서 10^{38}까지의 수를 나타낼 수 있다. 배정도 실수는 11비트의 지수와 52비트의 유효수를 가지며 대략 10^{-308}에서 $10^{30}8$까지의 수를 나타낼 수 있다. 지수는 가장 작은 음수를 빼는 방법으로 편향(biased)시키는데, 이를 통해 지수는 부호가 없는 수로 나타낼 수 있게 된다. 예를 들어 단정도에서 지수 12는 12 − (−127) = 139 = 0x8b로 나타내며 지수 −12는 −12 − (−127) = 115 = 0x73으로 나타낸다.

IEEE 표준에서 대부분의 값은 유효수가 1에서 2 사이의 값이 될 때까지 유효수를 이동시킴으로써 정규화한다(나타내는 값이 변하지 않게 지수도 조정한다). 정규화의 결과로 유효수에 있는 비트의 개수는 실제로 명시적으로 나타낸 것보다 하나가 더 많아진다. 값 1.어떤 수 × 2^{exp}에서 1은 여분으로 표현에서는 생략한다. 이 규칙에 대한 예외는 0 근처에서 일어난다. 매우 작은 수는 (감소된 정밀도를 사용해서) 0.어떤 수 × 2^{min+1}로 나타낼 수 있다. 여기서 min은 해당 형식에서 쓸 수 있는 가장 작은(음수) 지수다. 좀 더 오래된 다수의 부동소수점 실수 표준에서는 이러한 비정규 수를 허용하지 않으며, 이는 표현 가능한 가장 작은 양수와 0 사이에 표현 가능한 가장 작은 두 양수 간의 간격보다 더 큰 간격을 야기한다. IEEE 표준은 비정규 수를 포함하기 때문에 점진적 언더플로우를 제공한다. 비정규 수는 지수부에 0(가장 작은 음수 지수를 나타냄)을 가지며 0이 아닌 분수로 나타낸다.

예 5.19

IEEE 부동소수점 실수

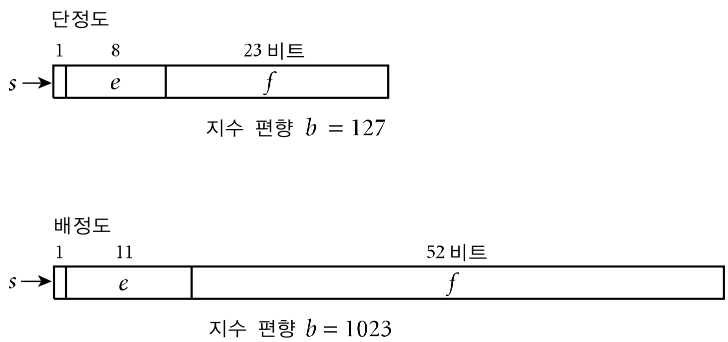
IEEE 754 표준의 규약을 그림 5.9에 나타냈다. 단정도와 배정도 형식 외에도 이 표준은 판매자–정의 "확장" 단정도와 배정도 숫자를 제공한다(여기서는 보이지 않았다). 확장 형식은 각기 최소한 32와 64개의 유효수 비트(명시적으로는 31과 63개)를 가져야 한다.

부동소수점 실수 연산은 책 한 권을 통째로 할애할 수 있을 만큼 복잡하다. 컴파일러 작성자가 특히 관심을 가질 수 있는 IEEE 표준의 특징은 다음과 같다.

- 음수가 아닌 부동소수점 실수를 나타내는 데 사용하는 비트 패턴은 정수와 같은 방식으로 정렬된다. 결과적으로 부동소수점 실수의 크기를 결정할 때 일반적인 정수 비교 연산을 사용할 수 있다.
- 0은 전부 0으로 구성된 비트 패턴으로 나타낸다. (혼란스럽게도) "음수 0"도 있는데 부호 비트만 1이고 나머지는 전부 0으로 구성된 패턴이다.
- 두 비트 패턴은 양의 무한대와 음의 무한대를 나타내기 위해 사용한다. 이 값들은 예측 가능한 방식으로 동작한다. 예를 들어 임의의 양수가 0으로 나누어지면 양의 무한대가 된다. 마찬가지로 양의 무한대의 아크탄젠트 값은 $\pi/2$다.
- 몇 가지 비트 패턴은 특수한 "숫자가 아님(not-a-number, NaN)" 값을 위해 사용한다. 이 값들은 음수의 이중 근호 값, 양의 무한대와 음의 무한대의 합, 0 나누기

0 등과 같이 터무니없는 연산에 의해 생성된다. NaN에 대한 거의 모든 연산의 결과는 다른 NaN이다. 결과적으로 많은 알고리즘에서 내부 오류 검사를 완화할 수 있다. 오류가 없는 경우에 이치에 맞는 여러 단계를 따른 후 최종 결과가 NaN이 아님을 보장하기 위해 결과만 확인하면 된다. 보통 산술 연산에 의해 생성되지 않는 일부 NaN은 컴파일러가 초기화되지 않은 변수나 기타 특수 상황을 명시적으로 나타내기 위해 설정할 수 있다. 이러한 알림 NaN은 사용되는 경우 인터럽트를 발생할 수 있다.

정수와 부동소수점 실수 연산 모두에 대한 소개와 참고할 수 있는 문헌 정보는 헤네시와 패턴슨의 컴퓨터 구조 교과서에서 데이비드 골드버그가 쓴 부록[HP03, App. H]에서 찾아볼 수 있다. 이 부록은 books.elsevier.com/companions/1558605967/appendices/1558605967−appendix-h.pdf에서 볼 수 있다.



	e	f	값
0	0	0	± 0
무한	$2b+1$	0	$\pm \infty$
정규	$1 \le e \le 2b$	<any>	$\pm 1.f \times 2^{e-b}$
비정규	0	$\neq 0$	$\pm 0.f \times 2^{1-b}$
NaN(숫자 아님)	$2b+1$	$\neq 0$	NaN

그림 5.9 | IEEE 754 부동소수점 실수 표준. 정규화된 수의 경우 지수는 정밀도에 따라 e−127 또는 e−1023이다. 유효수는 역시 정밀도에 따라 1 + f X 2−23 또는 1 + f X 2−52이다. 항목 f는 분수부 또는 분수로 불린다. e가 모두 1인 비트 패턴(단정도의 경우 255, 배정도의 경우 2047)은 무한과 NaN을 나타낸다. e는 0이지만 f는 0이 아닌 비트 패턴은 비정규(점진적 언더플로우) 수를 나타내는 데 사용한다.

✔ 확인문제

30. 2의 보수의 덧셈에 대한 역원(반대 부호 값)을 계산하는 방법을 설명하라.

31. 2의 보수 덧셈에서 오버플로우를 탐지하는 방법을 설명하라.

32. 2의 보수는 부호를 나타내기 위해 하나의 비트를 사용하는가? 설명하라.

33. IEEE 754 부동소수점 실수 산술의 주요 특징을 요약하라.

34. 단정도와 배정도 부동소수점 실수 값의 대략적인 범위는 무엇인가? 배정도 값의 정밀도는 무엇인가(비트로 답하라)?

35. 부동소수점 실수 NaN은 무엇인가?