Wstęp do programowania

Reprezentacje liczb

Liczby naturalne, całkowite i rzeczywiste w układzie binarnym

System dwójkowy

- W komputerach stosuje się dwójkowy system pozycyjny do reprezentowania zarówno liczb naturalnych, całkowitych, jak i rzeczywistych
- Kodowanie to umożliwia reprezentację jedynie skończonego podzbioru wszystkich tych nieskończonych zbiorów.
- Nie każda liczba naturalna, całkowita, rzeczywista ma swój dokładny odpowiednik komputerowy
- W szczególności żadna z liczb niewymiernych ($\sqrt{2}$,π,...) nie ma swojego dokładnego odpowiednika.
- Co gorsza, nie ma go nawet tak normalna liczba, jak 1/10.

Liczby naturalne

- Przypisujemy kolejnym pozycjom od prawej do lewej kolejne potęgi dwójki: 1,2,4,8,16,32,64,128,256,...
- Zatem np liczba 99 ma przedstawienie w systemie dwójkowym następujące:
- $99 = 1*64+1*32+0*16+0*8+0*4+1*2+1*1=(01100011)_2$

0	1	1	0	0	0	1	1
128	64	32	16	8	4	2	1

Jak zamienić liczbę na postać binarną?

- Można dodawać możliwie duże potęgi dwójki tak aby nie przekroczyć zadanej.
- Można tez dzielić liczbę przez 2 zapisując reszty (zera lub jedynki) i odczytując je wspak.

0	1	3	6	12	24	49	98
0	1	1	0	0	0	1	0

System szesnastkowy

- Często, aby skrócić zapis dwójkowy, grupuje się bity po 4 i interpretuje za pomocą cyfr szesnastkowych.
- Bajt = 8 bitów = 2 cyfry szesnastkowe
- Przykłady:
  - $10010001 = (91)_{16}$ (=9*16+1)
  - $11111111 = (FF)_{16}$ (=15*16+15)
  - $00000000 = (00)_{16}$ (=0*16+0)
  - $00001000 = (08)_{16}$ (=0*16+8)

System szesnastkowy

Ciąg bitów	Reprezentacja szesnastkowa	Reprezentacja dziesiętna
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Liczby ujemne

- Są 3 systemy kodowania liczb ujemnych:
  - Kod znak-moduł prosty
  - Kod znak-moduł odwrotny
  - Kod uzupełnieniowy (do dwóch)
- Dodatnie liczby całkowite – identycznie we wszystkich kodach
- Ujemne zupełnie inaczej. Rezerwujemy pierwszy bit jako bit znaku. Jedynka oznacza, że mamy do czynienia z liczbą ujemną

Kod znak-moduł (prosty)

- Musimy ustalić wpierw na ilu bitach będziemy reprezentować liczby.
- Ujemne liczby po pierwszej obowiązkowej jedynce mają moduł. Na przykład w ośmiobitowej reprezentacji liczba -5 ma reprezentację 10000101, podobnie jak 5 ma reprezentację 00000101.

Kod znak-moduł odwrotny

- Po bicie znaku liczby ujemne mają negatyw modułu: jedynki zastępujemy zerami, a zera jedynkami
- Na przykład w ośmiobitowym kodzie liczba -5 ma reprezentację 11111010.
- Liczba 5 ma ten sam kod 00000101, co w kodzie znak-moduł.

Kod uzupełnieniowy

- Pierwszy bit reprezentuje wartość -2^n dla (n+1)-bitowej reprezentacji
- Np d;a 8 bitów bit pierwszy reprezentuje wartość $-128=-\mathbf{2}^7$
- Zatem -5 ma reprezentację 11111011, bo -128+64+32+16+8+2+1=-5
- Zauważmy, że od poprzedniej reprezentacji kod ten różni się tylko jednym – ostatnim bitem. Tak jest zawsze!

Arytmetyka binarna (dodawanie)

- W każdym z kodów dodawanie się realizuje nieco inaczej. Najprościej jest – o dziwo! – w kodzie uzupełnieniowym, gdzie po prostu wystarczy wykonać dodawanie nie zważając na znak i wynik wychodzi taki, jak powinien.

Przykłady dodawania w U2

+	0	0	0	0	0	1	0	1	5
	0	0	0	0	0	1	1	1	7
	0	0	0	0	1	1	0	0	12

Dodajemy bit po bicie od prawej do lewej, uwzględniając bity przeniesienia.

Dodawanie w U2

	1	1	1	1	1	0	1	1	-5
✚	0	0	0	0	0	1	1	1	7
	0	0	0	0	0	0	1	0	2

- Jedynkę, która powstaje jako przeniesienie ostatniego bitu po prostu ignorujemy!

Dodawanie w U2

-

	0	0	0	0	0	1	0	1	5
✚	1	1	1	1	1	0	0	1	-7
	1	1	1	1	1	1	1	0	-2

- Tym razem przeniesienia nie trzeba było ignorować

Dodawanie w U2

-

	0	1	1	1	1	1	1	1	127
✚	0	0	0	0	0	0	0	1	1
	1	0	0	0	0	0	0	0	-128

- 127 jest największą dodatnią wartością, którą można uzyskać na 8 bitach. Gdy dodamy do niej jedynkę, dostaniemy najmniejszą możliwą wartość, i to ujemną, czyli -128.

Problem przepełnienia

Większość procesorów nie sprawdza przy dodawaniu, czy wynik nie przekracza przypadkiem zakresu reprezentowalności. Po prostu wykonuje bitowe działania, a wynik interpretuje wbrew zdrowemu rozcisądkowi (suma dwóch liczb dodatnich wychodzi ujemna).

Wniosek: przy dużych liczbach należy uważać, bo to są przyczyny niektórych błędów programistycznych tym złośliwszych, że mogą przejść przez wiele testów.

Ułamki binarne

- Podobnie, jak w układzie dziesiętnym, przedłużamy reprezentację za kropkę binarną (odpowiednik dziesiętnego przecinka) dodając pozycje odpowiadające kolejnym ujemnym potęgom dwójki: ½,¼,...

Ułamki binarne – przykłady

- ½=0.1
- ¼=0.01
- ¾= ½+¼, więc ¾=0.11
- 2/3 = 0.1010101(01)
- 1/10 = 0.000110011100(1100)

Reguły zaokrąglania

- $0.0001100|1100 \approx 0.0001101$
- $0.000110|01100 \approx 0.000110$
- $0.00011|001100 \approx 0.00011$
- $0.0001|1001100 \approx 0.0010$
- $0.000|11001100 \approx 0.001$
- $0.00|011001100 \approx 0.00$
- $0.0|0011001100 \approx 0.0$
- $0.|00011001100 \approx 0$

Notacja stałopozycyjna

- Ustalamy pewną stałą liczbę bitów na część całkowitą i pewną liczbę bitów na część ułamkową. Kropki wtedy nie reprezentujemy, bo i tak wiadomo, między którym a którym bitem się znajduje.
- Wadą notacji stałopozycyjnej jest jej mały zakres i mała precyzja. Reprezentowane wartości są rozmieszczone równomiernie.

Jak sobie radzą z dużymi zakresami fizycy?

- $h = 6.62 * 10^{-34}$ Js
- $R = 10^{25}$m
- ... co jest znacznie czytelniejszym zapisem, niż odpowiednio
  0.00000000000000000000000000000602 Js
  lub 100000000000000000000000000m

Czyli podaje się parę cyfr dokładnych i czynnik skalujący

Reprezentacja zmiennopozycyjna

- Przedstawia się w postaci pary
- (mantysa, cecha)
- mantysa – dokładne cyfry, a cecha – to czynnik skalujący mówiący o tym, o ile miejsc przesunąć kropkę binarną (przecinek dziesiętny)
- $x=m*2^a$ dla mantysy m i cechy a, przy czym zakładamy, że $\frac{1}{2} \leq m < 1$ dla dodatnich mantys oraz $-1 \leq m < -\frac{1}{2}$ dla mantys ujemnych
- Przykład: Liczba 4 ma przedstawienie (½,3), bo $4= \frac{1}{2}*2^3$. ($=1*2^2 = 2*2^o = ...$, ale tylko mantysa ½ spełnia zadane nierówności).

Przykład reprezentacji zmiennopozycyjnej

- Załóżmy, że mamy 3 bity cechy i 4 mantysy. Jak wygląda wtedy reprezentacja liczby ⅓?
- Zamieńmy na ułamek okresowy mnożąc przez 2 części ułamkowe i zapisując części całkowite:
- 0⅓ 0⅔ 1⅓ 0⅔ 1⅓ 0⅔
- 0. 0 1 0 1 0..., czyli 0.(01)
- Ponieważ 0.01... jest mniejsze od ½, więc musimy przemnożyć mantysę przez 2, zatem cecha musi być równa -1, co w zapisie uzupełnieniowym na 3 bitach ma postać 111. (-4+2+1). Mantysa musi być zaokrąglona po 3-ciej cyfrze po kropce binarnej (mamy tylko 4 bity na mantysęm z czego 1 odchodzi na znak odpowiadający wartości -1), więc ostatecznie postać w naszym układzie możliwie jak najlepiej przybliżająca ⅓, to 111 0101 (cecha mantysa).

Błędy zaokrągleń

- Zaokrąglając godzimy się na niedokładne obliczenia: nie działamy na prawdziwych liczbach, tylko na ich przybliżeniach
- Wyniki działań mogą zależeć od kolejności ich wykonania. Na przykład (a+b)+c wcale nie musi dać tego samego wyniku co a+(b+c).
- Może się zdarzyć, że a+b=a, mimo że b≠0
- Błędy mogą się kumulować w miarę postępu obliczeń!

Analiza numeryczna

- To dział informatyki zajmujący się opracowywaniem i analizą algorytmów pod kątem ich dokładności.
- Niektóre metody dają dokładniejsze wyniki niż inne, nawet przy tej samej reprezentacji.
- Oczywiście są też znaczne różnice w czasie działania różnych algorytmów.
- Zazwyczaj im szybszy, tym dokładniejszy!