Jak liczy komputer

Agnieszka MIĘDLAR

Każdy początkujący programista – więcej, każdy początkujący użytkownik tego znakomitego liczydła jakim jest komputer – zastanawia się, jak to sprytne urządzenie rozróżnia liczby i w jaki sposób przeprowadza na nich obliczenia. Odpowiedź na to pytanie jest prosta, komputer przechowuje wszystkie liczby w postaci binarnej. Rolą kompilatora jest odpowiednie przekonwertowanie naszych zapisków na jedyny zrozumiały dla komputerów język zer i jedynek. Jak w praktyce realizowana jest ta konwersja? To pytanie wymaga już dłuższych wyjaśnień. Nasze rozważania rozpocznijmy od abstrakcyjnego pojęcia liczby. Każdemu znane są liczby naturalne, rzeczywiste itp. Nas interesować będą systemy liczbowe, czyli zbiory reguł określone w celu jednolitego zapisywania i nazywania liczb.

System pozycyjny jest złożonym systemem liczbowym, w którym liczby zapisywane są w postaci ciągów cyfr umieszczonych na pozycjach ponumerowanych kolejno od 0 (rozpoczynając od prawej strony). Każdej pozycji w ciągu przyporządkowana jest pewna wartość liczbowa, tzw. *waga pozycji* (wyjątek stanowi system znak-moduł, o czym później). Cyfra znajdująca się na danej pozycji określa wielokrotność wagi tej pozycji. Do zapisu liczby w systemie pozycyjnym o podstawie p, używamy dokładnie p cyfr $0, \ldots, p-1$. Jak doskonale wiemy, w przypadku systemu podstawie $p = 10$ (system dziesiętny), ciągi liczbowe składają się z cyfr należących do zbioru $\{0, 1, \ldots, 9\}$.

Przypuśćmy, że dany jest n-cyfrowy ciąg $c_{n-1}c_{n-2}\ldots c_1c_0$ będący zapisem liczby w systemie pozycyjnym o podstawie p. W celu lepszej ilustracji spójrzmy na poniższą tabelę:

pozycja	$n-1$	$n-2$	...	1	0
cyfra	c_{n-1}	c_{n-2}	...	c_1	c_0
waga	w_{n-1}	w_{n-2}	..	w_1	w_0

w której $c_i \in \{0, \ldots, p-1\}$. Pozycji i przyporządkowana jest waga w_i. Wartość tak zapisanej liczby wyznaczamy jako sumę wielokrotności wag pozycji, co możemy zapisać w postaci:

$$\sum_{i=0}^{n-1} c_i \cdot w_i$$

Np. w systemie dziesiętnym ciąg cyfr 321 oznacza $3 \cdot 10^2 + 2 \cdot 10^1 + 1 \cdot 10^0$

Spróbujmy teraz zapoznać się z kilkoma powszechnie stosowanymi systemami pozycyjnymi: *naturalnym kodem binarnym*, *zapisem znak-moduł*, *kodem uzupełnień do 1* oraz nieco bardziej szczegółowo z *kodem uzupełnień do 2*. Zasadnicza różnica omawianych systemów związana jest z określeniem wag poszczególnych pozycji, co wpływa na zakres reprezentowanych w systemie liczb, ale o tym za chwilę.

Naturalny kod binarny (Natural Binary Code) to nic innego, jak odkryty przez Chińczyków ponad 4 tys. lat temu, a rozpowszechniony w wieku XVIII przez Leibniza system binarny. Jest to pozycyjny system liczbowy o podstawie $p = 2$, w którym do zapisu liczb, zgodnie z własnościami systemu pozycyjnego, używa się jedynie cyfr 0 i 1. Wagą i-tej pozycji w ciągu cyfr jest 2^i. Za sprawą Claude'a Shannona oraz Konrada Zuse system binarny stał się podstawą działania współczesnych maszyn cyfrowych, w których do reprezentacji dwóch cyfr 0 i 1 wykorzystano różnice napięć elektrycznych. Shannonowi, ojcu teorii informacji, zawdzięczamy również powstanie pojęcia *bit* (skrót od ang. *binary digit*), określającego najmniejszą jednostkę informacji, czyli w praktyce po prostu cyfrę 0 lub 1. n-bitowy ciąg w naturalnym kodzie binarnym pozwala na reprezentację liczb z zakresu $[0, 2^n - 1]$.

pozycja	$n-1$	$n-2$	...	1	0
bit	b_{n-1}	b_{n-2}	...	b_1	b_0
waga	2^{n-1}	2^{n-2}	..	2^1	2^0

Jeśli ciąg $b_{n-1}b_{n-2}\ldots b_1b_0$ jest liczbą zapisaną w naturalnym kodzie binarnym, to jej wartość jest równa:

$$\sum_{i=0}^{n-1} b_i \cdot 2^i$$

Na przykład liczba 57 w systemie dwójkowym to 111001, więc jej ośmiobitowa reprezentacja w naturalnym kodzie binarnym to 00111001.

Dodajmy, że 8 bitów to *bajt*, a w komputerach stosowane są zazwyczaj liczby o długościach $n = 8, 16, 32$ i 64 bitów. We wszystkich przykładach będzie się pojawiał najkrótszy format 8-bitowy.

Kolejnym etapem było stworzenie systemu umożliwiającego prostą reprezentację nie tylko liczb naturalnych, ale również liczb całkowitych. Najprostszym systemem wykorzystującym n bitów do zapisu liczb z zakresu $\left[-2^{n-1}+1, 2^{n-1}-1\right]$ jest tzw. **zapis znak-moduł (SM – Signed Magnitude)**. I tak, wszystkie bity w ciągu, poza najstarszym (ostatnim bitem począwszy od prawej strony), interpretować będziemy jako liczbę zapisaną w naturalnym kodzie binarnym i nazywać modułem liczby, a ostatni bit będzie bitem znaku. Jeżeli liczba jest dodatnia, to bit znaku przyjmuje wartość 0, jeżeli ujemna – wartość 1. Dysponując ośmioma bitami ($n = 8$), możemy zapisać na nich wszystkie liczby całkowite z przedziału $[-127, 127]$.

Ciąg $b_{n-1}b_{n-2}\ldots b_1b_0$ w zapisie znak-moduł odpowiada zatem liczbie:

$$(-1)^{b_{n-1}} \cdot \left(\sum_{i=0}^{n-2} b_i \cdot 2^i\right)$$

Na uwagę zasługują dwie szczególne własności systemu znak-moduł. Uważny Czytelnik zwrócił z pewnością uwagę na fakt, iż bit znaku w reprezentacji znak-moduł nie posiada wagi, a jego wartość, jak sama nazwa wskazuje, decyduje jedynie o znaku liczby. Zauważmy również, iż w zapisie znak-moduł mamy dwa sposoby reprezentacji 0: jako 10000000 oraz 00000000.

Dla wprawy możemy na przykład zapisać w 8-bitowym systemie znak-moduł liczbę -13:

bit	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
wartość	1	0	0	0	1	1	0	1
waga	$+/-$	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Kolejnym etapem na drodze do opracowania koncepcyjnie prostego i łatwego z punktu widzenia przeprowadzania operacji arytmetycznych zapisu liczb całkowitych, było opracowanie tzw. **kodu uzupełnień do 1 (U1, 1C – One's complement)**. W tym systemie waga najstarszego bitu jest równa $-2^{n-1}+1$. Wobec tego ciąg bitów $b_{n-1}b_{n-2}\ldots b_1b_0$ w kodzie U1 oznacza liczbę:

$$b_{n-1}\cdot(-2^{n-1}+1)+\sum_{i=0}^{n-2} b_i\cdot 2^i$$

Zakres liczby zapisanej na n bitach w kodzie uzupełnień do 1 jest taki sam jak zakres n bitowej liczby w zapisie znak-moduł i wynosi $\left[-2^{n-1}+1, 2^{n-1}-1\right]$. Ponownie pojawia się jednak problem dwóch reprezentacji zera w postaci ciągów 00000000 oraz 11111111.

Reprezentację liczby dodatniej w kodzie U1 wyznaczamy standardowo. Natomiast w celu zapisania w formacie U1 liczby ujemnej wyznaczamy rozwinięcie dwójkowe jej modułu, w razie potrzeby uzupełniając bitami o wartości 0 do ustalonej długości formatu, po czym zamieniamy wszystkie bity wyznaczonego ciągu na przeciwne. Liczbę -13 zapiszemy więc tak:

bit	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
wartość	1	1	1	1	0	0	1	0
waga	-2^7+1	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Kod uzupełnień do 2 (U2, 2C – Two's Complement) jest w chwili obecnej najpopularniejszym zapisem liczb całkowitych. Ze względu na łatwość wykonywania operacji arytmetycznych jest podstawą funkcjonowania współczesnych komputerów, co skłania nas do poświęcenia mu szczególnej uwagi. Koncepcja kodu uzupełnień do dwóch przypomina naturalny kod binarny i kod U1. Wagą i-tej pozycji jest 2^i, wyjątek stanowi waga najstarszego bitu (bitu znaku), która przyjmuje wartość -2^{n-1}. Wartością ciągu zapisanego w tym kodzie jest więc

$$b_{n-1}\cdot(-2^{n-1})+\sum_{i=0}^{n-2} b_i\cdot 2^i$$

Stosując kod U2, na n bitach możemy reprezentować liczby całkowite z zakresu $\left[-2^{n-1}, 2^{n-1}-1\right]$, a zatem mamy dodatkową liczbę ujemną. Rozwiązany został przy tym problem podwójnej reprezentacji zera.

Nazwa systemu jest związana ze szczególną zależnością między reprezentacją danej liczby x i liczby przeciwnej $-x$. Z wyjątkiem $x=0$ suma tych reprezentacji (jeśli potraktujemy ciągi bitów jako liczby dodatnie) jest zawsze równa 2^n. Wynika stąd praktyczny przepis na wyznaczanie w systemie U2 liczby przeciwnej: najpierw negujemy wszystkie bity danej liczby, a następnie dodajemy do wyniku liczbę 1. Sprawdźmy: negację bitu można zapisać jako przekształcenie $b_i \to 1-b_i$, zatem w wyniku opisanej procedury otrzymujemy liczbę:

$$-(1-b_{n-1})2^{n-1}+\textstyle\sum_{i=0}^{n-2}(1-b_i)2^i+1$$

$$=-(-b_{n-1}2^{n-1}+\textstyle\sum_{i=0}^{n-2} b_i2^i)-2^{n-1}+2^0+\ldots+2^{n-2}+1$$

$$=-(-b_{n-1}2^{n-1}+\textstyle\sum_{i=0}^{n-2} b_i2^i)$$

czyli faktycznie liczbę przeciwną do początkowej.

Możemy wyznaczyć reprezentację naszej ulubionej liczby -13:

bit	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
wartość	1	1	1	1	0	0	1	1
waga	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Kod uzupełnień do 2 zyskał swoją popularność dzięki prostym zasadom przeprowadzania operacji arytmetycznych. Dodawanie i odejmowanie w systemie U2 jest analogiczne do wykonywania działań w zwykłym systemie dwójkowym. Pojawiające się przeniesienia poza bit znaku możemy zwyczajnie zignorować. Oto przykład:

	57	00111001
$+$	(-13)	11110011
$=$	44	00101100

Do tego łatwo zanegować liczbę (o czym była już mowa), zatem za darmo mamy też odejmowanie. Ogromnie upraszcza to budowę układów odpowiedzialnych za operacje arytmetyczne w procesorach. Dla porównania, w systemie znak-moduł musimy prowadzić obliczenia jedynie na modułach liczb, a następnie na podstawie określonych reguł ustalana jest wartość bitu znaku. Także w systemie U1 nie jest aż tak prosto: na przykład nie moglibyśmy dodać w ten sposób do zera (zapisanego jako 11111111) liczby 1 (00000001), bo wynikiem byłoby 00000000, czyli... zero. W systemie U2 mamy za to zjawisko tzw. przewijania licznika, będące zmorą niejednego programisty. Sytuacja taka ma miejsce, gdy do największej liczby dodatniej w naszej reprezentacji (01111111) dodamy liczbę 1 (00000001). W efekcie otrzymamy najmniejszą reprezentowalną liczbę ujemną (10000000). Liczba ujemna może też być wynikiem dodawania dwóch dużych liczb dodatnich.

Podsumowując, przedstawiliśmy pokrótce najpowszechniejsze systemy reprezentacji liczb całkowitych stosowane w maszynach obliczeniowych i komputerach. Niektóre z nich jedynie zaistniały w świecie komputerów (np. system znak-moduł w komputerach IBM 7090) inne, jak kod uzupełnień do 2, ciągle przeżywają okres swojej świetności. W tym momencie powinno się pojawić pytanie „A co z liczbami rzeczywistymi?". Otóż w przypadku liczb rzeczywistych mamy do czynienia z tzw. *systemami zmiennopozycyjnymi*, ale to już zupełnie inna historia...