Stałopozycyjna reprezentacja liczb

Do reprezentacji liczb całkowitych stosowane są następujące kody binarne:
- zapis znak-moduł
- zapis U1
- zapis U2

1. Zapis **znak-moduł** ZM (ang. sign-magnitude) utworzono przez dodanie jednego bitu do zapisu NKB. Bit ten, dodawany przed najbardziej znaczącą pozycją słowa, jest bitem znaku. Wartość tego bitu równa 0 oznacza liczbę dodatnią, a 1 liczbę ujemną. Dalsze bity w zapisie NKB oznaczają moduł tej liczby. Dla słów 16-bitowych można przedstawić liczby z zakresu $-2^{15} + 1 <= A <= +2^{15} - 1$, ponieważ największą liczbą zakodowaną na 15 bitach jest $2^{15} - 1$.

2. **U1** to tzw. zapis uzupełnień do 1 (ang. 1s complement). W zapisie tym najbardziej znaczący bit jest również bitem znaku (0 – liczba dodatnia, 1 – ujemna), ale w zależności od jego wartości dalsze bity zapisu mają różne znaczenie. Jeśli bit znaku jest 0 (liczba dodatnia) to dalsze bity są reprezentacją liczby dodatniej w kodzie NKB. Natomiast gdy byt znaku jest 1 (liczba ujemna), to dalsze bity reprezentują moduł liczby ujemnej w taki sposób, że zanegowane ich wartości odpowiadają modułowi tej liczby w kodzie NKB. Zakres liczb tego zapisu jest taki sam jak dla kodu ZM.

3. **U2** to tzw. zapis uzupełnień do 2 (ang. 2s complement). Zapis ten różni się od U1 tylko dla liczb ujemnych. Moduł liczby ujemnej w kodzie U2 jest obliczany w taki sposób, że do zanegowanych pozycji słowa jest arytmetycznie dodawana jedynka i dopiero tak utworzone słowo odpowiada w kodzie NKB modułowi tej liczby.

Arytmetyka stałopozycyjna

Dodawanie liczb dodatnich

Dla wszystkich w/w reprezentacji binarnych, liczby dodatnie przedstawiają się tak samo, a ich dodawanie przebiega w ten sam sposób – jest analogiczne do pisemnego dodawania liczb dziesiętnych, z przeniesieniem jedynki w przypadku dodawania dwóch jedynek, np.

```
 0110
+0011
=1001
```

Dodawanie liczb o przeciwnych znakach (odejmowanie)

1. Dla zapisu ZM algorytm jest następujący:
   - porównaj moduły obu argumentów i przyjmij jako znak wyniku znak liczby o większym module,
   - od większego modułu odejmij moduł mniejszy.

Samo odejmowanie przebiega wg następującego algorytmu:
   - jeżeli na danej pozycji odjemna jest równa 1, to odejmujemy od niej odjemnik o wartości 0 lub 1 i na tym kończymy działanie,

- jeżeli natomiast na danej pozycji odjemna oraz odjemnik wynoszą 0, to rezultatem działania jest 0,
- jeśli na danej pozycji odjemna wynosi 0, a odjemnik 1, to wynik jest równy 1, ale musimy wykorzystać tzw. „pożyczkę" z bardziej znaczącej pozycji. Jeżeli na bardziej znaczącej pozycji (tzn. jeden bit w lewo) jest jedynka, to zapożyczenie polega na jej zamianie na zero. Jeżeli nie ma tam jedynki, to należy dokonać pożyczki z jeszcze bardziej znaczącej pozycji. W rezultacie kolejne zera będą zmieniały się na jedynki, aż do napotkania pierwszej jedynki, np. 9 – 7 (przedstawione są kolejne kroki działania, pożyczka w pierwszym rzędzie, w nawiasie):

```
(0000)      (0110)      (0110)      (0110)
+1001       +100x       +10xx       +1xxx
−0111       −011x       −00xx       −0xxx
 xxxx        xx10        x010        0010
```

2. Dla zapisu U1 algorytm odejmowania polega na dodawaniu wszystkich pozycji liczby wraz z bitem znaku oraz uwzględnienia powstałego przeniesienia. Oznacza to konieczność korekcji w przypadku powstania przeniesienia. Korekcja ta polega na dodaniu przeniesienia (jedynki) do najmniej znaczącej pozycji wyniku. Jest to wada tego zapisu, powodująca wydłużanie czasu trwania operacji arytmetycznych.

3. Wadę tę wyeliminowano przy operacjach w zapisie U2. Algorytm odejmowania jest po prostu dodawaniem wszystkich pozycji wraz z bitem znaku. Jeżeli powstanie przeniesienie na bicie znaku, jest ono ignorowane, a zatem algorytm w tym zapisie daje zawsze poprawny wynik w mniejszej liczbie kroków niż w zapisie U1.

Dzielenie poprzez przesunięcie bitowe

Za pomocą prostego przesunięcia o odpowiednią liczbę bitów w prawo (w najpopularniejszych językach programowania operator '>>') możemy dokonać dzielenia przez odpowiednią potęgę liczby dwa. Z punktu widzenia jednostki obliczeniowej jest to działanie najszybsze (w porównaniu do skomplikowanych algorytmów dzielenia). Na najstarszą pozycję dopisywany jest bit o wartości zero, natomiast najmłodszy bit jest tracony, np.:

dziesiętnie: 19 / 2 = 9
binarnie: 00010011 >> 1 = 00001001

Mnożenie poprzez przesunięcie bitowe

Analogicznie, za pomocą przesunięcia o odpowiednią liczbę bitów w lewo (w najpopularniejszych językach programowania operator '<<') możemy dokonać mnożenia przez odpowiednią potęgę liczby dwa. Na najmłodszą pozycję dopisywany jest bit o wartości zero, natomiast najstarszy bit jest tracony, np.:

dziesiętnie: 19 * 2 = 38
binarnie: 00010011 << 1 = 00100110

Powyższe przykłady dotyczą Naturalnego Kodu Binarnego (NKB). W przypadku kodów ZM, U1 i U2, bit odpowiedzialny za znak zostaje zachowany.

Mnożenie dowolnych liczb

Mnożenie dla kodu Znak-Moduł przebiega następująco:

- porównaj bity znaku i przyjmij znak wyniku (zgodnie z przyjętymi zasadami mnożenia),
- wykonaj mnożenie modułów na zasadzie tradycyjnego „mnożenia słupkowego” znanego z działań na liczbach dziesiętnych.