Zastosowania procesorów sygnałowych

SYSTEMY LICZBOWE

Opracowanie: Grzegorz Szwoch

Politechnika Gdańska, Katedra Systemów Multimedialnych

1 + 1 = 10

- Procesor wykonuje jedynie operacje na ciągach bitów:
  1010011001100110 + 0010011001100110 1 = 1111001100110011
- Procesor nie interpretuje co to znaczy „1010011001100110”.
  To może być 42598, -22938, -0,7 lub coś jeszcze innego.
- System liczbowy – metoda zapisu liczb dziesiętnych za pomocą sekwencji 0 i 1 w taki sposób, aby wynik operacji bitowych był poprawny.
- Na tym wykładzie omówimy najważniejsze systemy liczbowe stosowane w PS:
  - całkowitoliczbowe,
  - zmiennoprzecinkowe,
  - stałoprzecinkowe.

Liczby całkowite bez znaku

Zapis liczb całkowitych bez znaku (*unsigned integer*) na *N* bitach:

- najstarszy bit (MSB): 2^{N-1}
- *i*-ty bit od prawej: 2^{i-1}
- najmłodszy bit (LSB): 1

Liczba dziesiętna = suma wag, dla których bit = 1.

2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
1	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0
32768		8192			1024	512			64	32			4	2	

= 42598

Liczby całkowite ze znakiem

Zapis liczb całkowitych ze znakiem (*signed integer*) musi zapewniać poprawny wynik operacji. Np.: (-1 + 2 = 1) → (x + 0000 0000 0000 0010 = 0000 0000 0000 0001).
Stąd: x = 1111 1111 1111 1111.

Zamiana liczby dodatniej na ujemną (zapis w kodzie uzupełnień do 2, U2):

- negujemy wszystkie bity (0 → 1, 1 → 0),
- dodajemy 1.

Przykłady dla liczb 8 bitowych:

(-1) 0000 0001 → 1111 1110 → 1111 1111

(-123) 0111 1011 → 1000 0100 → 1000 0101

Najstarszy bit jest równy 0 dla liczb dodatnich i 1 dla ujemnych, z tego powodu jest nazywany bitem znaku.

Rozdzielczość i zakres

Rozdzielczość:

- najmniejsza możliwa różnica między dwoma liczbami,
- wynosi $2^0 = 1$, a więc nie można zapisywać liczb niecałkowitych, np. 0,5.

Zakres:

- najmniejsza i największa liczba możliwa do zapisania,
- zakres zależy od liczby bitów *N* przeznaczonych na zapis liczby,
- liczby bez znaku: od 0 do (2^N-1), np. 16 bitów: od 0 do 65535,
- liczby ze znakiem: od -2^{N-1} do $(2^{N-1} – 1)$, np. 16 bitów: od -32768 do 32767,
- jeżeli nie potrzebujemy liczb ujemnych, użycie typu bez znaku zwiększa nam górny zakres dwukrotnie.

Typy stałoprzecinkowe w języku C

- *int* – co najmniej 16 bitów (Intel, ARM: 32 bity; DSP C5535: 16 bitów).
- *char* – 8 bitów, bajt (*byte*).
- *short* – 16 bitów, 2 bajty, słowo (*word*).
- *long* – 32 bity, 4 bajty, podwójne słowo (*double word*).
- *long long* – dłuższy niż *long* (Intel, ARM: 64 bity, DSP C5535: 40 bitów).

Każdy typ występuje w dwóch wersjach:

- *unsigned* – bez znaku, np. *unsigned int*,
- *signed* – ze znakiem (typ domyślny), np. *signed long* (to samo co *long*).

W języku C każdy typ może mieć zdefiniowane aliasy (alternatywne nazwy),
np. 16-bitowy *short* może też mieć nazwy: *DATA* (na DSP C5535), *int16_t*, *WORD*, itp.

Ułożenie bajtów w pamięci

Jeżeli liczba zajmuje np. 4 bajty, w jakiej kolejności są one zapisywane w pamięci?

- Od najmłodszego do najstarszego bajtu (*little endian*): B4, B3, B2, B1
  – wszystkie procesory Intela i (domyślnie) ARM.
- Od najstarszego do najmłodszego bajtu (*big endian*): B1, B2, B3, B4
  – niektóre procesory, transmisja sieciowa.
- Na procesorze sygnałowym zazwyczaj można wybrać sposób zapisu (przy kompilacji kodu). Najczęściej wybiera się zapis *little endian*.

Przekroczenie zakresu

Co się stanie jeżeli wynik operacji nie zmieści się w zadanej liczbie bitów?

Następuje przekroczenie zakresu (*range overflow*) – nadmiarowe bity są tracone.

Przykład dla dodawania liczb 16-bitowych.

- Liczby bez znaku (maksymalna wartość: 65535)
  (65530 + 10) → [1] 0000 0000 0000 0100 → 4
  - nadmiarowa jedynka zostaje utracona (flaga *overflow* procesora ustawiona na 1),
  - dostajemy wynik *modulo* 2^{16}.
- Liczby ze znakiem (maksymalna wartość: 32767)
  (32760 + 10) → 1000 0000 0000 0010 → -32766 (!!!)
  - jedynka „przeskakuje” na bit znaku,
  - następuje „zawinięcie” zakresu, dostajemy ujemny wynik.

Jak się chronić przed przepełnieniem zakresu?

- Akumulatory procesorów sygnałowych mają nadmiarowe bity (np. 32+8), zapobiega to przepełnieniu przy zapisywaniu pośrednich wyników obliczeń.
- Zapisując wynik do pamięci musimy wybrać „wystarczająco pojemny” typ liczbowy, np. *long* zamiast *short*.
- Jeżeli nie możemy wybrać dłuższego typu, możemy przeskalować liczby, np.:
  - zamiast $y = a + b$ zapisujemy: $y = 2 * (a/2 + b/2)$,
  - używając przesunięcia bitowego: $y = (a>>1 + b>>1)<<1$
  - zapobiegamy przepełnieniu, ale tracimy najmłodszy bit liczb a i b.

Przy okazji – operatory języka C:

- $x>>n$: przesunięcie bitowe o n miejsc w prawo, odpowiada dzieleniu przez 2^n,
- $x<<n$: przesunięcie bitowe o n miejsc w lewo, odpowiada mnożeniu przez 2^n.

Zapis zmiennoprzecinkowy

- Zapis zmiennoprzecinkowy (*floating point*) pozwala zapisywać dowolne liczby, także niecałkowite.
- Procesor musi posiadać jednostkę do wykonywania obliczeń na takich liczbach (FPU, *floating point unit*). Taki procesor nazywamy procesorem zmiennoprzecinkowym.
- Procesor C5535 z projektu ZPS, tak jak wiele innych PS, nie posiada takiej jednostki – jest procesorem stałoprzecinkowym. Nie można więc używać typów zmiennoprzecinkowych.
- Operacje wykonywane na liczbach zmiennoprzecinkowych są dłuższe i wymagają więcej cykli procesora niż operacje na liczbach stałoprzecinkowych. Zatem wymagane są szybsze procesory sygnałowe (większa częstotliwość taktowania), co przekłada się na większe zużycie energii.

Zapis zmiennoprzecinkowy

Każda liczba jest reprezentowana przez:

- S – znak (0: dodatnia lub 1: ujemna),
- M – mantysę *(mantissa*),
- E – wykładnik (*exponent*),
- b – bazę (*base*, zwykle b = 2).

$$(-1)^s \cdot 1,M \cdot b^{(E-127)}$$

Przykład:

$2\pi = 6.283185307179586 = 1,570796326795 \cdot 2^2$

S = 0, M = 570796326795, E = 129, b = 2

Typy zmiennoprzecinkowe

Typy zmiennoprzecinkowe zdefiniowano w standardzie IEEE 754.

- *float* – typ pojedynczej precyzji
  - 32 bity: 1 bit znaku, 23 bity mantysy, 8 bitów wykładnika
  - ok. 7 znaczących miejsc po przecinku
  - zakres: od $\pm 3{,}4 \cdot 10^{-38}$ do $\pm 3{,}4 \cdot 10^{38}$
- *double* – typ podwójnej precyzji
  - 64 bity: 1 bit znaku, 52 bitów mantysy, 11 bitów wykładnika
  - ok. 15 znaczących miejsc po przecinku
  - zakres: od $\pm 1{,}7 \cdot 10^{-308}$ do $\pm 1{,}7 \cdot 10^{308}$

Rozdzielczość jest zmienna, zależy od wartości liczby.

Typ *double* zapewnia większą precyzję zapisu liczb, kosztem dłuższych obliczeń.

Zapis stałoprzecinkowy liczb niecałkowitych

- System liczbowy to tylko interpretacja znaczenia bitów.
- Można więc przyjąć taką interpretację (zapis na *N* bitach):
  - najmłodszy bit ma wartość $2^{-(N-1)}$,
  - bit (N-1) ma wartość 2^{-1},
  - najstarszy bit jest bitem znaku.
- Zyskujemy możliwość zapisywania liczb niecałkowitych.

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
znak	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}
0	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0
+		2^{-2}			2^{-5}	2^{-6}			2^{-9}	2^{-10}			2^{-13}	2^{-14}	

$= 0{,}29998 \approx 0{,}3$

Reprezentacja Q15

- Reprezentacja liczb ułamkowych jako liczby całkowitej, 16-bitowej, ze znakiem, nazywa się zapisem Q15:
  1 bit znaku, 0 bitów części całkowitej, 15 bitów części ułamkowej.
- Rozdzielczość (minimalna różnica między liczbami) jest równa $2^{-15} = 0{,}000030517578125$.
- Zakres: od -1 do $(1-2^{-15})$, czyli do 0,99996948242187 5 (+1 jest już poza zakresem).
- Wszelkie uwagi dotyczące przepełnienia zakresu są nadal aktualne.
- Taki system liczbowy nazywamy stałoprzecinkowym (*fixed point*); można zapisywać liczby niecałkowite, ale pozycja bitowa przecinka dziesiętnego jest stała.
- Zapis Q umożliwia wykonywanie operacji na liczbach niecałkowitych za pomocą stałoprzecinkowego procesora sygnałowego (nie posiadającego FPU).

Reprezentacja QM.N

- Możemy uogólnić ten zapis do formy QM.N: 1 bit znaku,
  M bitów części całkowitej, *N* bitów części ułamkowej.
- Rozdzielczość: 2^{-N}.
- Zakres: od -2^{M} do $(2^{M} – 2^{-N})$.
- Przykład: 1 bit części całkowitej i 14 bitów części ułamkowej
  daje nam zapis Q1.14. Zakres: od -2 do 1,9999389648437 5.
- Przy ustalonym sposobie zapisu QM.N, przecinek dziesiętny pozostaje zawsze na tej samej pozycji (przed *N*-tym bitem od prawej). Stąd termin „zapis stałoprzecinkowy”.
- Podobnie można zapisywać liczby bez znaku. Mamy wtedy zapis UQM.N (np. UQ16).

Zapis Q na typach całkowitych

- Nie ma w języku C specjalnego typu dla liczb Q. Stosujemy zwykłe typy całkowite, np. *short*.
- Aby zapisać liczbę Q, musimy interpretować bity tak, jakby były zwykłymi liczbami całkowitymi, np.:
  0,3 → 10011001100110 → 9830
- Co sprowadza się do zależności (dla QM.N):
  - konwersja z liczby ułamkowej *x* na liczbę całkowitą *q*:
    $q = x \cdot 2^N$ (z zaokrągleniem)
  - z liczby całkowitej *q* na liczbę ułamkową *x*:
    $x = q / 2^N$
- Np. dla liczb Q15: $q = 32768 \cdot x$; $x = q / 32768$.

Zapis Q na typach stałoprzecinkowych

Przykład dla zapisu Q15 ($2^N = 2^{15} = 32768$):

0,3 → 0,3 * 32768 → 9830,4 → 9830

0,6 → 0,6 * 32768 → 19660,8 → 19661

Obliczenie wyrażenia (0,3 + 0,6):

9830 + 19661 = 29491

Konwersja na liczbę ułamkową:

29491 → 29491 / 32768 → 0,89999

(przypominamy: rozdzielczość jest równa 0,00003)

Kwantyzacja

- Przy konwersji musimy zaokrąglać wynik do najbliższej dostępnej liczby:
  9830,4 → 9830 19660,8 → 19661
- Jest to kwantyzacja liczb.
- Błąd (szum) kwantyzacji: różnica między wartościami po kwantyzacji a oryginalnymi.
- Przy zapisie liczb stałoprzecinkowych na procesorach, kwantyzacja ma bardzo duży wpływ na dokładność obliczeń.
- Np. dla filtrów cyfrowych typu IIR może spowodować niestabilność prawidłowo zaprojektowanego filtru.
- Błąd kwantyzacji maleje ze wzrostem liczby bitów: jest mniejszy w Q31 niż w Q15.
- W zapisie zmiennoprzecinkowym błąd kwantyzacji jest znacznie mniejszy.

Mnożenie liczb QM.N

Jak obliczyć (0,3 · 0,6) stosując zapis Q15?

- Konwersja jak poprzednio: 0,3 → 9830, 0,6 → 19661.
- 9830 · 19661 = 193267630
- Mnożenie dwóch liczb Q15 daje wynik w formacie Q30!
- Mnożenie liczb QM.N daje Q(2M).(2N)
- Konwersja wyniku na liczbę dziesiętną:
  $193267630 / 2^{30} = 0{,}1799945$
- Na procesorze bardzo często konwertuje się wynik mnożenia z Q30 na Q31 (mnożąc przez 2, czyli <<1). Ułatwia to późniejszą konwersję na Q15.

Dzielenie liczb na procesorze stałoprzecinkowym jest baaaardzo wolne i należy go unikać. Wyjątek: dzielenie przez potęgi 2, obsługiwane przesunięciem bitowym (>>).

Mnożenie liczb QM.N

Mamy wynik mnożenia liczb Q15 w formacie Q30. Jak z tego odzyskać liczbę Q15?

- Najpierw podzielić przez 2^{15}, przesuwając w prawo o 15 bitów (>>15).
- Następnie odrzucić starsze bity – pozostawić młodszych 16 bitów.
- W ten sposób wykonuje się zaokrąglenie w dół.
- Aby zaokrąglić do najbliższej liczby, trzeba na początku dodać 2^{14} (jedynka na najstarszym z odrzucanych bitów), a potem przesunąć i obciąć wynik.

Mnożenie:	9830 · 19661 = 193267630
Zaokrąglenie:	193267630 + (1<<14) = 193284014
Konwersja na Q15:	193284014 >> 15 = 5898
Dziesiętnie:	5898 /32768 = 0,17999267

Niedopełnienie

- Co się stanie jeżeli po przesunięciu wyniku mnożenia zostaną same zera?
- (0,003 · 0,002) → 98 · 66 = 6468 (Q30)
- 6468 >> 15 = 0 (Q15) !!!
- Wynik (0,003 · 0,002) = 0,000006 jest zbyt małą liczbą do zapisania w Q15 (jest mniejszy niż rozdzielczość Q15 równa 0,00003).
- Powstaje niedopełnienie (*underflow*), powodujące wyzerowanie wyniku.
- Wszystkie kolejne mnożenia (np. w filtrze) też dadzą zero!
- Przed niedopełnieniem próbujemy się chronić stosując dłuższe typy liczbowe (np. Q31) i reorganizując kolejność operacji.

Mnożenie Q15 w języku C

Jak poprawnie zapisać wynik mnożenia liczb Q15 w języku C?

```
short a = 9830;
short b = 19661;
/* short y = ??? */
```

W ten sposób się nie da: wynik nie zmieści się na 16 bitach, starsze bity zostaną obcięte:

```
short y = a * b;  // y = 1966 (1011 1000 0101 0000 0111 1010 1110)
```

Może tak? Typ *long* ma 32 bity, więc wynik powinien się zmieścić.
Niestety, wciąż ten sam błędny wynik. Dlaczego?

```
long y = a * b;  // y = 1966
```

Mnożenie Q15 w języku C

- Kompilator najpierw oblicza wyrażenie po prawej stronie: (*a* * *b*).
- Typ wyniku jest zgodny z „najdłuższym” typem argumentu. Oba argumenty są typu *short*, więc wynik też ma typ *short*. Nadmiarowe bity zostają obcięte.
- Wynik (błędny) jest zapisywany do tego, co jest po lewej stronie (typ *long*).
- Aby uzyskać prawidłowy wynik, musimy „promować” argument do dłuższego typu.
- Rzutowanie (*cast*) w języku C wykonuje się podając nowy typ w nawiasie przed wyrażeniem lub zmienną:

```
long y = (long)a * b;  // y = 193267630
```

- Teraz jeden z argumentów jest typu *long*, więc wynik będzie też zapisany jako *long*.
- Rzutowanie stałej liczbowej (literału) na typ *long* można wykonać tak:

```
long y = a * 19661L;
```

Mnożenie liczb Q15

A zatem, aby wykonać mnożenie (0,3 · 0,6) i zapisać wynik w formacie Q15 (jako *short*), musimy to zrobić tak:

```
short y = (short)(((long)a * b) >> 15);
```

albo z zaokrągleniem:

```
short y = (short)((((long)a * b) + (1<<14)) >> 15);
```

Na szczęście, procesory sygnałowe dają nam zwykle instrukcje-skróty. Np. na procesorze C5535 to samo wykonamy instrukcją *_smpy*:

```
short y = _smpy(a, b);
```

Wersja z zaokrągleniem – instrukcja *_lsmpyr*:

```
short y = (short)(_lsmpyr(a, b) >> 16);
```

Tryb nasycenia

- Przepełnienie zakresu skutkuje znacznym przekłamaniem wyniku (np. 32760 + 10 = -32766).
- Procesory sygnałowe mają arytmetykę nasycenia (*saturation arithmetic*). Nasycenie polega na obcięciu liczb przekraczających zakres do wartości granicznej. 32760 + 10 = 32767
- Wynik wciąż jest błędny, ale błąd jest ograniczany.
- Na procesorze C5535 mamy instrukcje wykonujące operacje z obsługą trybu nasycenia, o nazwach zaczynających się od _s: _*sadd* (+), _*ssub* (–), _*smpy* (×), _*sround* (zaokrąglenie), itp.

Cechy zapisu zmiennoprzecinkowego

- Ze względu na bardzo szeroki zakres, praktycznie nie ma ryzyka wystąpienia przepełnienia.
- Ryzyko niedopełnienia jest bardzo małe (mniejsze dla typu *double*).
- Nie trzeba stosować żadnych specjalnych zapisów w C:

```
double y = 0.3 * 0.6;
```

- Precyzja obliczeń jest znacznie większa niż dla zapisu stałoprzecinkowego. 0,3 * 0,6 = 0,18, a nie 0,1799945.
- Działania na liczbach zmiennoprzecinkowych wymagają znacznie więcej cykli procesora i często więcej pamięci.

Działania na liczbach zmiennoprzecinkowych w C

Ta sama zasada co dla liczb stałoprzecinkowych obowiązuje również dla zmiennoprzecinkowych. Typ wyniku jest równy „największemu” typowi argumentów.

```
short a = 5;
double wrong = a / 2;          // Nieprawidłowo: wrong=2 (!)
double b = (double)a / 2;      // Prawidłowo, z rzutowaniem.
double c = a / 2.0;
float d = (float)a / 2;
float e = a / 2.0f;
```

Ze względu na ograniczoną precyzję, nie należy bezpośrednio przyrównywać liczb zmiennoprzecinkowych do siebie. Np. (5.0 / 2.5) może być równe 2,5000000000001.

```
if (x / 2.0 == 2.5) {}                 // To się może nie udać.
if (abs(x / 2.0 - 2.5) < 1e-8) {}      // Prawidłowo.
```

Procesory zmiennoprzecinkowe a stałoprzecinkowe

Zalety procesorów zmiennoprzecinkowych:

- duża dokładność obliczeń,
- duża dynamika – bardzo mały szum kwantyzacji,
- wygodne programowanie – nie trzeba konwertować liczb na zapis Q,
- szybsze i dokładniejsze wykonywanie wielu operacji matematycznych (dzielenie, pierwiastek, logarytm, funkcje trygonometryczne, itp.).

Wady procesorów stałoprzecinkowych:

- wolniejsze obliczenia, więcej cykli zużytych na wykonanie operacji (nawet prostego mnożenia),
- większe zużycie pamięci, zwłaszcza gdy stosujemy liczby typu *double,*
- większe zużycie energii,
- wyższy (znacznie) koszt procesora.

Procesory zmiennoprzecinkowe a stałoprzecinkowe

Kiedy użyjemy zmiennoprzecinkowego procesora sygnałowego?

- gdy potrzebujemy wysokiej precyzji obliczeń,
- gdy liczy się duża dynamika, np. przetwarzanie dźwięku z rozdzielczością 32 bity – mniejsze szumy,
- gdy nas na to stać (koszty produkcji urządzeń i zużycia energii),
- gdy istotna jest wygoda programowania operacji zmiennoprzecinkowych.

Kiedy użyjemy stałoprzecinkowego procesora sygnałowego?

- gdy wykonujemy tylko proste operacje PS (FFT, filtry),
- gdy istotna jest cena procesora i niskie zużycie energii,
- gdy sygnał wejściowy nie ma dużej dynamiki (np. sygnał z przetwornika A/C o rozdzielczości 12 bitów).