Zad 4.

środa, 8 marca 2023 16:44

Zadanie 4. Zmienne «x» i «y» o typie «`uint32_t`» przechowują czteroelementowe wektory typu «`uint8_t`». Tj. wektor $\{x_3, x_2, x_1, x_0\}$ reprezentujemy w zmiennej «x» przypisując jej wartość $\sum_{i=0}^{3} x_i \cdot 2^{8i}$. Jak szybko obliczyć zmienną «z» przechowującą wektor $\{z_3, z_2, z_1, z_0\}$, gdzie $z_i = x_i \oplus y_i$, gdy:

- $\oplus$ jest operacją dodawania,
- $\oplus$ jest operacją odejmowania.

Obliczając wynik należy zapobiec wystąpieniu **przeniesienia** (ang. *carry*) lub **pożyczki** (ang. *borrow*) propagujących się do bardziej znaczącego bajtu.

Wskazówka: Spróbuj rozwiązać zadanie samodzielnie, a następnie przeczytaj §2.17 książki „*Uczta programistów*".

$X = \underbrace{X_3}_{8}\ \underbrace{X_2}_{8}\ \underbrace{X_1}_{8}\ \underbrace{X_0}_{8} \qquad y = y_3\ y_2\ y_1\ y_0$

a) $z = \underbrace{X_3 + Y_3}_{8}\ \underbrace{X_2 + y_2}_{8}\ \underbrace{X_1 + y_1}_{8}\ \underbrace{X_0 + y_0}_{8}$ ← uint32_t

maskujemy najstarszy bit w każdym bajcie i wykonujemy dodawanie z boku bez nich

```
  1010 1111
+ 0101 1000
-----------
1 0000 0111
```

przeniesienie do starszego bajtu

(1) $0x7F7F\_ = \underbrace{0111\ 1111}_{8}\ 011\ldots_2$

(2) $0x8080\ldots = \overbrace{1000\ 0000}^{}\ 100\ldots_2$

wynik dodawania zmaskowanych bitów otrzymujemy za pomocą $(x \wedge y) \& (2)$

dokładniej najstarsze bity $x_i + y_i$

ostateczny wynik dostajemy xorem ← w zasadzie xor to dodawanie ignorujące przeniesienie

podsumowując: na najstarszych bitach chcemy ignorować przeniesienie więc jak używamy ich w sumie to używamy xora, natomiast pozostałe możemy sumować normalnie

```
14 uint32_t add_vectors(uint32_t x, uint32_t y) {
15     return (x ^ y) & 0x80808080 ^ (x & 0x7F7F7F7F) + (y & 0x7F7F7F7F);
16 }
```

+ > & > ^

b)

```
12 uint32_t sub_vect(uint32_t x, uint32_t y) {
13     uint32_t d = (x | 0x80808080) - (y & 0x7F7F7F7F);
14     uint32_t d = ~((x ^ y) | 0x7F7F7F7F ^ d);
15     return d;
16 }
```

(1) (2) (3) (4)

idea jest bardzo podobna. Najpierw wykonujemy odejmowanie na młodszych (2) bitach elementów wektora, dopisując przed tym sztucznie 1 w miejsca najstarszych bitów (1). następnie wykonujemy odejmowanie bez pożyczki na najstarszych bitach (4) i wypełniamy resztę jedynkami. Negacja z XORA zamienia różne bity na 0, a te same na 1 więc (3) nie zmieni z obliczonych młodszych bitów, natomiast poprawi te starsze, które brały udział w „sztucznej pożyczce", równości