

Single Instruction Multiple Data (SIMD)
przetwarzanych jest wiele zestawów danych
przez jedną instrukcję



- MMX (MultiMedia eXtensions lub Matrix Math eXtensions)
- liczby całkowite
- SSE (Streaming SIMD Extenstions)
- liczby całkowite i liczby zmiennoprzecinkowe
- AVX (Advanced Vector Extensions)
- zastępuje i rozszerza SSE



Instrukcje AVX
są sposobem zrównoleglenia programów
na poziomie danych



Rejestry AVX ymm oraz zmm

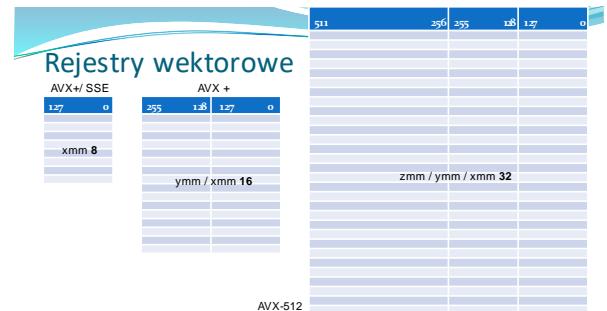
Dla rozkazów AVX dedykowano rejestryst:

- 16 rejestrów 256 bitowych dla AVX oraz AVX2
ymm/15 - ymm/o
- 32 rejestryst 512 bitowe dla AVX-512
zmm/31 - zmm/o
- Cześć instrukcji (większość) operuje na młodszej części rejestryst
xmm/[15/31] - xmm/o

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

7



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

Rejestry XMM

Przykład: VADDPD $\text{xmm1} = \text{xmm2} + \text{xmm3}/\text{m128}$

zmm/7	zmm/6	zmm/5	zmm/4	zmm/3	zmm/2	zmm/1	zmm/0
				ymm/3	ymm/2	ymm/1	ymm/0
				ymm/3	ymm/2	ymm/1	ymm/0
				xmm/1	xmm/0		
511		256	255	128	127	64	0
				xmm2/4	xmm2/0		
				+	+		
				xmm3/1 lub m128/1	xmm3/0 lub m128/0		
				=	=		
				xmm1/1	xmm1/0		

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

9

Rejestry YMM

Przykład: VADDPD $\text{ymm1} = \text{ymm2} + \text{ymm3}/\text{m256}$

zmm/7	zmm/6	zmm/5	zmm/4	zmm/3	zmm/2	zmm/1	zmm/0
				ymm/3	ymm/2	ymm/1	ymm/0
				ymm/3	ymm/2	ymm/1	ymm/0
				xmm/1	xmm/0		
511		256	255	128	127	64	0
				xmm2/4	xmm2/0		
				+	+	+	+
				ymm3/1 lub m256/3	ymm3/0 lub m256/2	xmm3/1 lub m128/1	xmm3/0 lub m128/0
				=	=	=	=
				ymmu/3	ymmu/2	xmm1/1	xmm1/0

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

10

Rejestry XMM

Przykład: VADDPD $\text{zmm1} = \text{zmm2} + \text{zmm3}/\text{m512}$

zmm/7	zmm/6	zmm/5	zmm/4	zmm/3	zmm/2	zmm/1	zmm/0
				ymm/3	ymm/2	ymm/1	ymm/0
				ymm/3	ymm/2	ymm/1	ymm/0
				xmm/1	xmm/0		
511		256	255	128	127	64	0
				xmm2/4	xmm2/0		
				+	+	+	+
				zmm3/1 lub m512/6	zmm3/0 lub m512/5	xmm3/1 lub m128/6	xmm3/0 lub m128/5
				=	=	=	=
				zmmu/7	zmmu/6	zmmu/5	zmmu/4
				ymmu/3	ymmu/2	xmm1/1	xmm1/0

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

11

Typy danych AVX

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

12

Typy danych dla AVX

- Liczby całkowite (packed)

Nazwa	zakres	Oznaczenie w AVX
Bajt	8 bitów	B
Słowo	16 bitów	W
Podwójne słowo	32 bity	D
Poczwórne słowo	64 bity	Q

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

13

Typy danych dla AVX

- Liczby zmienno-przecinkowe (packed i scalar)

nazwa	zakres	Oznaczenie w AVX
Pojedyncza precyza	32 bity	SS, PS
Podwójna precyza	64 bity	SD, PD

- Specyfikatory zmiany typu:

xmmword ptr [adres]
ymmword ptr [adres]

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

14

Typy danych AVX liczby całkowite

Przykład na rejestrach 128 bitowych xmm



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

15

Typy danych AVX

Liczby zmienno-przecinkowe

PS – vector liczb pojedynczej precyzji
PD – vector liczb podwójnej precyzji
SS – scalar pojedynczej precyzji
SD – scalar podwójnej precyzji

Powyższe oznaczenia mają odzwierciedlenie w nazwach instrukcji, „mnemonikach” dla liczb zmienno-przecinkowych.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

16

Instrukcje typu AVX

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

17

Instrukcje typu AVX

Pytanie: po co stosować instrukcje typu *scalar* skoro wykorzystują jedną najmłodszą część rejestru.

- nie zawsze działały na wektorach.
- pominiecie koprocesora.
- gdy liczba danych jest niepodzielna przez liczbę elementów wektora.

Jesli AVX operuje na „skalarach”, starsze części rejestru są prawie zawsze zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

18

Instrukcje typu AVX

Instrukcje AVX podobnie jak instrukcje SSE są instrukcjami wektorowymi, jednak **nie poleca się łączenia w jednym programie/podprogramie instrukcji AVX z SSE**, ponieważ powoduje to **znaczone spowolnienie działania programu/podprogramu.**

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

19

Instrukcje typu AVX

Intel® Integrated Performance Primitives

- https://software.intel.com/sites/landingpage/IntrinsicsGuide/#techs=AVX_AVX2_FMA
- <https://software.intel.com/content/www/us/en/develop/tools/oneapi/optiments/ipp.html>

Ekwivalent np. dla **VADDPD**

```
_mm128d _mm128_add_pd (_mm128d a, _mm128d b);
_mm256d _mm256_add_pd (_mm256d a, _mm256d b);
_mm512d _mm512_add_pd (_mm512d a, _mm512d b);
```

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

20

Systematyka instrukcji AVX

- Dokumentacja firmy Intel wyróżnia 12 kategorii instrukcji AVX.
- Na potrzeby niniejszego wykładu zastosowano podział na instrukcje AVX dotyczące liczb całkowitych, liczb zmienno-przecinkowych oraz oddzielną grupę FMA, która w AVX operuje wyłącznie na liczbach zmienno-przecinkowych. W AVX dostępna jest również grupa instrukcji szyfrujących wykorzystująca algorytm AES (ang. *Advanced Encryption Standard*)

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

21

Systematyka instrukcji AVX

- Pośród rozkazów typu AVX wyróżniamy grupy: AVX, AVX2, FMA
- AVX operuje na rejestrach ymm oraz xmm
- AVX2 wyłącznie na rejestrach ymm
- FMA na rejestrach ymm
- Instrukcje szyfrujące algorytmem AES na rejestrach xmm
- Najogólniejszą systematyką instrukcji AVX jest podział na instrukcje dla liczb całkowitych i dla liczb zmienno-przecinkowych, w tych ostatnich sytuują się instrukcje FMA.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

22

Budowa rozkazu AVX

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

23

Budowa rozkazu AVX liczby całkowite

- Mnemoniki prawie wszystkich rozkazów AVX rozpoczynają się od liter v (od słowa Vector);
- 1 literowy skrót p od „packed”, jednak umieszczony na początku rozkazu określa operację na liczbach całkowitych;
- 3-4 literowy skrót wykonywanego działania (add,sub,mul...);
- niektóre instrukcje są z nasyceniem „saturation” skróty s;
- jednoliterowy skrót określa zakres operacji, może być (B) bytes, (W) word, (D) double word, (Q) quad word.

Rozkaz **vpaddb** wykonuje dodawanie (add) wektorowo/równolegle (p) liczb całkowitych w zakresie 8 bitów (b) i ewentualnie z nasyceniem

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

24



Budowa rozkazu AVX liczby zmiennoprzecinkowe

- Mnemoniki prawie wszystkich rozkazów AVX rozpoczynają się od litery v (od słowa Vector);
- 3-4 literowy skrót wykonywanego działania (np. add, mul, itp.);
- litera p lub s określa, eng packed lub eng.scalar;
- litera d lub s oznacza stopień precyzyji: double lub single.

vaddpd

Rozkaz **vaddpd** wykonuje dodawanie (add) wektorowo/równolegle (p) liczb zmiennoprzecinkowych podwójnej precyzyji (d).

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

25



Budowa rozkazu AVX operacje FMA

- Mnemoniki prawie wszystkich rozkazów AVX rozpoczynają się od litery v (od słowa Vector);
- 2 literowy skrót dla FMA (skumulowane wyniki dodawania);
- 3-4 literowy skrót wykonywanego działania (add, sub, mul, itp.);
- 3 cyfry określające kolejność mnożenia i dodawania, może być 132, 231 (np. 132 mnoży 1 x 3, i dodaje 2.);
- litera p lub s określa sposób wykorzystania rejestru eng. „packed”, lub „scalar”;
- litera d lub s oznacza stopień precyzyji „double” lub „single”;

vfmaadd132pd

Rozkaz **vfmaadd132pd** mnoży (m) i dodaje (add) równolegle/packed (p) liczby zmiennoprzecinkowe podwójnej precyzyji (d), kolejność operacji arytmetycznych jest oznaczona przez trzy cyfry 132, czyli wg przekładu mnoży i. i 3. argument, do iloczynu dodaje 2. argument.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

26



Bajt sterujący imm8

Część instrukcji AVX wykorzystuje, jako argument, **bajt sterujący imm8**

imm8[7]	imm8[6]	imm8[5]	imm8[4]	imm8[3]	imm8[2]	imm8[1]	imm8[0]
1	0	1	0	1	1	1	0

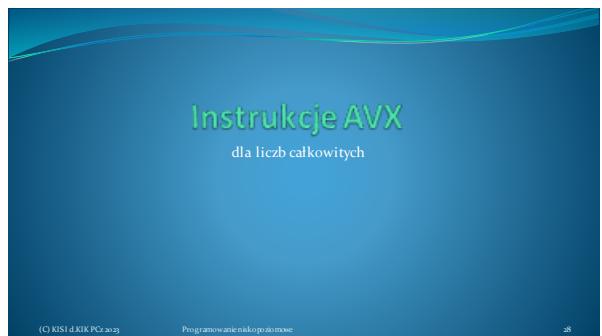
W instrukcjach bajt sterujący najczęściej jest zapisywany w postaci liczby szesnastkowej.

np. 0ech

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

27



Instrukcje AVX

dla liczb całkowitych

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

28



Instrukcje AVX dla liczb całkowitych

- Instrukcje przesłania
- Operacje matematyczne
- Operacje porównania
- Operacje przesunięcia (bitowe, arytmetyczne, logiczne)
- Instrukcje logiczne
- Instrukcje zerowania
- Instrukcje wyrównania
- Instrukcje dodatkowe (ładowanie ustawień)

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

29



Operacje przesłania AVX

Instrukcje przesłania:

- VMOVD, VMOVQ
 VMOVDQA, VMOVDQU, VMOVNTDQA
 VMOVNTDQ, VLDDQU
 VPMOV[S/Z]XBW, VPMOV[S/Z]XBD
 VPMOV[S/Z]XBQ, VPMOV[S/Z]XWD
 VPMOV[S/Z]XWQ, VPMOV[S/Z]XDQ
 VPMOVMSKB, VMASKMOVDQU, VPMASKMOV[D/Q]

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

30

Operacje przesłania AVX

- Instrukcje kompresji/rozpakowania:**

VPACK[S/U]SWB, VPACK[S/U]SDW,
VPUNPCKHBW, VPUNPCKHW, VPUNPCKHDQ,
VPUNPCKHQDQ, VPUNPCKLBW, VPUNPCKLWD,
VPUNPCKLDQ, VPUNPCKLQDQ

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

9

Operacje przesłania AVX

- Instrukcje przetasowania:** VPSHUFB, VPSHUFD
VPSHUFHW, VPSHUFLW

- Instrukcje permutacji:** VPERMD, VPERMQ

- Instrukcje mieszające:** VPBLENDB, VPBLENDW, VPBLEND

- Instrukcje rozgłaszenia:** VPBROADCASTB, VPBROADCASTW
VPBROADCASTD, VPBROADCASTQ

- Instrukcje zbierania:** VPGATHERDD, VPGATHERQD
VPGATHERDQ, VPGATHERQQ

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

31

Instrukcja przesłania

VMOV[D/Q]

vmovd xmm1, r/m32	xmm1 ← r/m32
vmovq xmm1, r/m64	xmm1 ← r/m64
vmovq xmm1, xmm2/m64	xmm1 ← xmm2/m64
vmovd r/m32, xmm1	r/m32 ← xmm1
vmovq r/m64, xmm1	r/m64 ← xmm1
vmovq xmm1/m64, xmm2	xmm1/m64 ← xmm2

VMOV / VMOVQ przesyła podwójne lub poczwórnego słowo z rejestru ogólnego przeznaczenia/pamięci do rejestru xmm lub odwrotnie. Używany jest jeden najmłodszy element rejestru xmm, starsze elementy są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

33

Instrukcja przesłania z wyrównaniem

VMOVDQ[A/U]

vmovdq[a/u] xmm1, xmm2/m128	vmovdq[a/u] ymm1, ymm2/m256
vmovdq[a/u] xmm2/m128, xmm1	vmovdq[a/u] ymm2/m256, ymm1

Przesyła całą zawartość (128 lub 256 bitów) źródła do celu, jeśli cellem lub źródłem jest pamięć, wówczas w wersji (A) dane w pamięci muszą być wyrównane (ang. aligne) do granicy 16 (m128) lub 32 (m256) bajtów, w przeciwnym wypadku zgłaszany jest wyjątek ochrony pamięci. Aby przesłać dane do/z niewyrównanych lokalizacji pamięci należy w instrukcji zamiast (A) użyć (U) (ang. unaligne).

cel (r) ← źródło (r/m) cel (r/m) ← źródło

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

34

Instrukcja przesłania z wyrównaniem

VMOVNTDQ[A]

vmovntdq xmm1, m128	cel (r) ← źródło (m)
vmovntdq ymm1, m256	
vmovntdq m128, xmm1	cel (m) ← źródło (r)
vmovntdq m256, ymm1	

Przesyła całą zawartość z/do pamięci m128/m256 do/z rejestru xmm/yymm.

Dla instrukcji w wersji (A) pamięć musi być wyrównana (ang. aligne) do granicy 16 (m128) lub 32 (m256) bajtów, w przeciwnym wypadku zgłaszany jest wyjątek ochrony pamięci, alternatywnie można zastosować instrukcję bez litery A. NT oznacza (non-temporal hint) przesłanie z pominięciem pamięci podręcznej (cache).

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

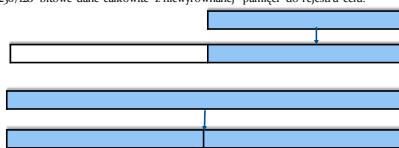
35

Instrukcja ładowanie danych z pamięci

VLDQU

vlddq u xmmy, m128
vlddq u ymm1, m256

Ładuje 256/128 bitowe dane całkowite z niewyrównanej pamięci do rejestru celu.



Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

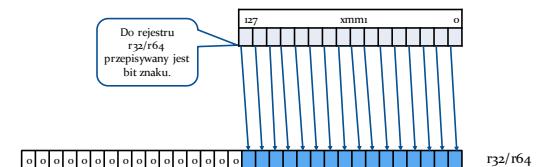
36

Instrukcja przesłania VPMOVMSKB

vpmovmskb reg, xmmi rejestr \leftarrow xmmi
 vpmovmskb reg, ymmi rejestr \leftarrow ymmi (AVX2)

Przesyła najstarsze bity (bit znaku) **każdego bajtu** rejestru xmmi/ymmi po kolejno do rejestru r32/r64, dla xmmi przenosi 16 bitów do r32, dla ymmi przenosi 32 bity do r64, pozostałe starsze bity są zerowane.

Instrukcja przesłania VPMOVMSKB



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

37

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

38

Instrukcje przesłania z konwersją

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

39

Instrukcje przesłania z konwersją

VPMOV[S/Z]XB[W/D/Q], VPMOV[S/Z]XW[D/Q]
 VPMOV[S/Z]XDQ

vpmov[s/z]xbw xmmi, xmm2/m64	vpmov[s/z]xwd xmmi, xmm2/m64
vpmov[s/z]xbd xmmi, xmm2/m32	vpmov[s/z]xwq xmmi, xmm2/m32
vpmov[s/z]xbq xmmi, xmm2/m16	vpmov[s/z]xdq xmmi, xmm2/m64

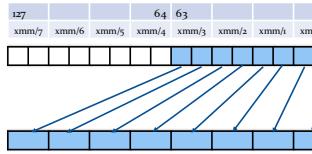
Instrukcja zamienia (konwertuje) ze znakiem / bez znaku:

bajty na słowa/podwójne słowa/poczwórne słowa, **slowa** na podwójne słowa/poczwórne słowa, **podwójne słowa** na poczwórne słowa przepisują odpowiednio wartości z młodszą częścią rejestru xmm2/(m64||m32||m16) **do rejestru celu xmmi**.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

40



xmm2/m128
xmmi

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

41

Instrukcje przesłania z konwersją

VPMOV [S/Z] XB[W/D/Q], VPMOV [S/Z] XW[D/Q],
 VPMOV [S/Z] XDQ (AVX2)

vpmov[s/z]xbw ymmi, xmm2/m128	vpmov[s/z]xwd ymmi, xmm2/m128
vpmov[s/z]xbd ymmi, xmm2/m64	vpmov[s/z]xwq ymmi, xmm2/m64
vpmov[s/z]xbq ymmi, xmm2/m32	vpmov[s/z]xdq ymmi, xmm2/m128

Instrukcja zamienia (konwertuje) ze znakiem / bez znaku:

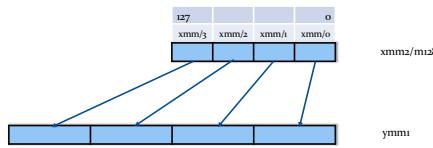
bajty na słowa/podwójne słowa/poczwórne słowa, **slowa** na podwójne słowa/poczwórne słowa, **podwójne słowa** na poczwórne słowa przepisują odpowiednio wartości z młodszą częścią z rejestru ymm2/(m128||m64||m32) **do rejestru celu ymmi**, na młodszą część rejestru.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

42

Instrukcja przesłania z konwersją VPMOVVSXQ



(C) KISI d.KIK PCz 2023

Programowanie nikopozomowe

43

Instrukcje przesłania - przykład:

```
vmovdqa ymm1, ymmword ptr[rdi] ; rdi = int * tab[1]
vmovdqa ymm1, ymmword ptr[rdi+4*rax] ; rdi = int * tab[1]; rax = n
```

Dla typu tablicowego ładowanie całego rejestru ymm1/ymm2 z adresu pierwszego (o) elementu tablicy oraz od kolejnych elementów (widokromieś a) o wielkości dwójnego słowa.

```
vmovdqa ymm5, ymmword ptr [ebx] ; ebx = unsigned short * a
vmovvpxwd ymm5, ymm5 ; konwersja z 16 do 32 bitów
```

Ponieważ obliczenia na 16 bitach mogły by doprowadzić do przepięcenia (overflow), typ unsigned short konwertujemy na unsigned int 32 bitów zachowując przy tym ilość elementów wektora równą 8.

(C) KISI d.KIK PCz 2023

Programowanie nikopozomowe

44

Instrukcje przesłania

warunkowego

(C) KISI d.KIK PCz 2023

Programowanie nikopozomowe

45

Instrukcja przesłania warunkowego VMASKMOVDQU

vmaskmovdqu xmm1, xmm2

Celem jest obszar 16 bajtów pamięci adresowany przez DS: DI / EDI / RDI.
Bajty ze źródła xmm1 są przesyłane do celu pod warunkiem, że siódme bity (bity znaku) odpowiadających im bajtów z xmm2 są jedynkami.

xmm2 = maska

if xmm2[i][7] = 1 then m128[i] = xmm1[i]

i – jest numerem bajtu

(C) KISI d.KIK PCz 2023

Programowanie nikopozomowe

46

Instrukcja przesłania warunkowego VPMASKMOV[D/Q]

```
vpmaskmov[d/q] xmm1, xmm2, m128
vpmaskmov[d/q] ymm1, ymm2, m256
vpmaskmov[d/q] m128, xmm1, xmm2
vpmaskmov[d/q] m256, ymm1, ymm2
```

Przesyła podwójne/poczwórne słowa z pamięci m128/m256 lub xmm2/ymm2 do rejestru celu xmm2/ymm2 lub pamięci m128/m256, pod warunkiem, że bit znaku odpowiadający wartości z rejestru maski (drugi operand) xmm2/xmm2 lub xmm1/ymm1 jest ustawiony na jeden, w przeciwnym wypadku zapisuje zero.

if źródło1[i][31/63] == 1 then cel[i] ← źródło2[i] else cel[i] ← 0

Bit od 128/256 do MSB są ignorowane.

(C) KISI d.KIK PCz 2023

Programowanie nikopozomowe

47

Instrukcje kompresji

(C) KISI d.KIK PCz 2023

Programowanie nikopozomowe

48

Instrukcja kompresji

VPACK[S/U]SWB

vpack[s/u]swb xmmi, xmm2, xmm3/m256 (AVX2)

Konwertuje słowa ze znakiem na bajty ze znakiem (S) / bez znaku(U) z rejestru xmm2 / ymm2 wpisując do młodszych części rejestru xmmi, a z xmm3/m256 / ymm3/m256 wpisując do starszych części rejestru ymmi / ymm2. Starsza część rejestru ymmi jest wypełniana danymi ze starszych części rejestrów hi ymm2 i hi ymm3/m256. Wynik jest zapisywany ze znakiem(S) / bez znaku(U) oraz z nasycieniem.

Bity od 128 i zgł do MSB są zerowane.

(C) KISI d.KIK PCz 2023

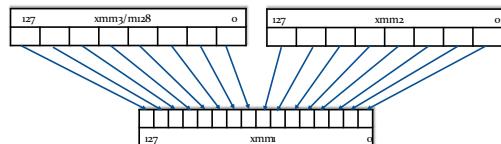
Programowanie niskopoziomowe

49

Instrukcja kompresji

VPACK[S/U]SWB

vpack[s/u]swb xmmi, xmm2, xmm3/m256 (AVX2)



(C) KISI d.KIK PCz 2023

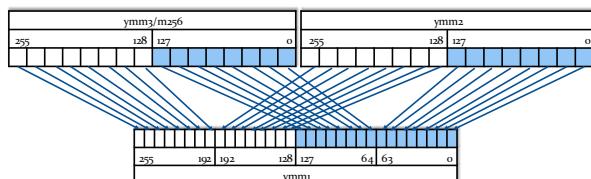
Programowanie niskopoziomowe

50

Instrukcja kompresji

VPACK[S/U]SWB

vpack[s/u]swb ymmi, ymm2, ymm3/m256 (AVX2)



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

51

Instrukcja kompresji

VPACK[S/U]SDW

vpack[s/u]sdw xmmi, xmm2, xmm3/m256 (AVX2)
vpack[s/u]sdw ymmi, ymm2, ymm3/m256 (AVX2)

Konwertuje podwójne słowa ze znakiem na słowa ze znakiem(S)/bez znaku(U) z rejestru xmm2/ lo ymm2 wpisując do młodszych części rejestru xmmi/ lo ymmi, z xmm3/m256 / lo ymm3/ lo m256 wpisując do starszych części rejestru celu xmmi/lo ymmi. Starsza część rejestru ymmi, jest wypełniana danymi ze starszych części rejestrów hi ymm2 oraz hi ymm3/m256. Wynik jest zapisywany ze znakiem(S) / bez znaku(U) oraz z nasycieniem.

Bity od 128 i zgł do MSB są zerowane.

(C) KISI d.KIK PCz 2023

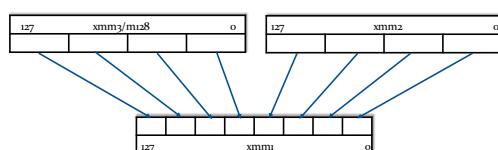
Programowanie niskopoziomowe

52

Instrukcja kompresji

VPACK[S/U]SDW

vpack[s/u]sdw xmmi, xmm2, xmm3/m256



(C) KISI d.KIK PCz 2023

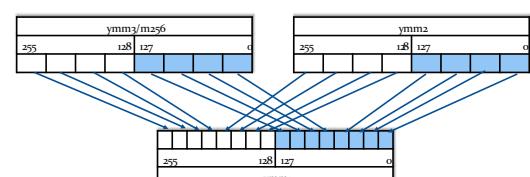
Programowanie niskopoziomowe

53

Instrukcja kompresji

VPACK[S/U]SDW

vpack[s/u]sdw ymmi, ymm2, ymm3/m256 (AVX2)



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

54

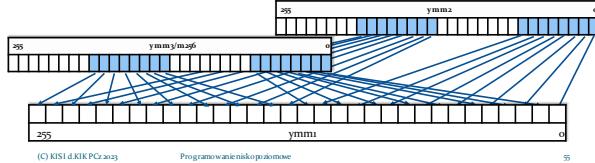
Instrukcja kompresji

VPUNPCKLBW / VPUNPCKLWD / VPUNPCKLDQ / VPUNPCKLQDQ

vpunpcklbw xmm1, xmm2, xmm3/m128

vpunpcklbw ymm1, ymm2, ymm3/m256 (AVX2)

Młodsze bajty ze 128 bitowych części rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepłotem do rejestru celu ymm/xmm.



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

55

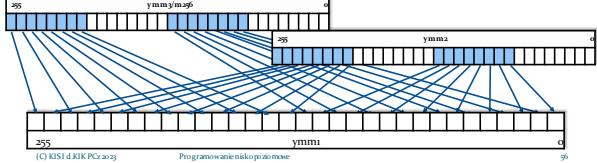
Instrukcja kompresji

VPUNPCKHBW / VPUNPCKHWD / VPUNPCKHDQ / VPUNPCKHQDQ

vpunpckhbw xmm1, xmm2, xmm3/m128

vpunpckhbw ymm1, ymm2, ymm3/m256 (AVX2)

Starsze bajty ze 128 bitowych części rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepłotem do rejestru celu ymm/xmm.



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

59

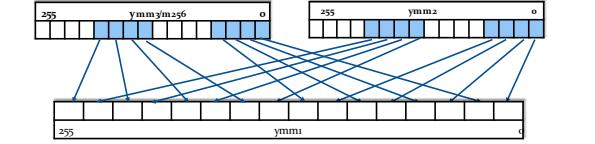
Instrukcja kompresji

VPUNPCKLBW / **VPUNPCKLWD** / VPUNPCKLDQ / VPUNPCKLQDQ

vpunpcklwd xmm1, xmm2, xmm3/m128

vpunpcklwd ymm1, ymm2, ymm3/m256 (AVX2)

Młodsze słowa ze 128 bitowych części rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepłotem do rejestru celu ymm/xmm.



Bity od 128/256 do MSB są zerowane.

Programowanie niskopoziomowe

57

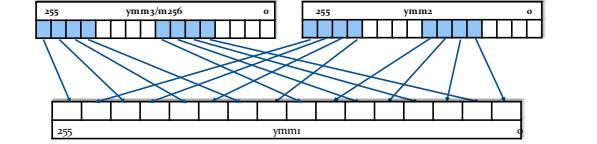
Instrukcja kompresji

VPUNPCKHWB / **VPUNPCKHWD** / VPUNPCKHDQ / VPUNPCKHQDQ

vpunpckhwb xmm1, xmm2, xmm3/m128

vpunpckhwd ymm1, ymm2, ymm3/m256 (AVX2)

Starsze słowa ze 128 bitowych części rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepłotem do rejestru celu ymm/xmm.



Bity od 128/256 do MSB są zerowane.

Programowanie niskopoziomowe

58

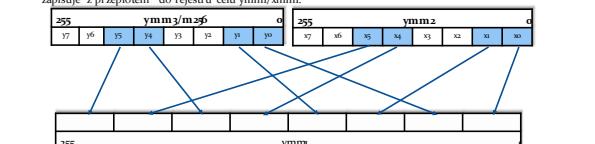
Instrukcja kompresji

VPUNPCKLBW / VPUNPCKLWD / **VPUNPCKLDQ** / VPUNPCKLQDQ

vpunpckldq xmm1, xmm2, xmm3/m128

vpunpcklddq ymm1, ymm2, ymm3/m256 (AVX2)

Młodsze podwójne słowa ze 128 bitowych części rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepłotem do rejestru celu ymm/xmm.



Bity od 128/256 do MSB są zerowane.

Programowanie niskopoziomowe

59

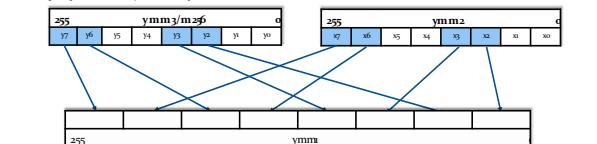
Instrukcja kompresji

VPUNPCKHWB / VPUNPCKHWD / **VPUNPCKHDQ** / VPUNPCKHQDQ

vpunpckhbw xmm1, xmm2, xmm3/m128

vpunpckhqdq ymm1, ymm2, ymm3/m256 (AVX2)

Starsze podwójne słowa ze 128 bitowych części rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepłotem do rejestru celu ymm/xmm.



Bity od 128/256 do MSB są zerowane.

Programowanie niskopoziomowe

60

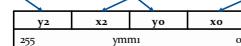
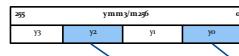
Instrukcja kompresji

VPUNPCKLBW / VPUNPCKLWD / VPUNPCKLDQ / **VPUNPCKLQDQ**

vpunpcklqdq xmm1, xmm2, xmm3/m128

vpunpcklqdq ymm1, ymm2, ymm3/m256 (AVX2)

Młodsze poczwórne słowa ze 128 bitowych części rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepłotem do rejestru celu ymm1/xmm1.



Bity od 128 do 0 są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

61

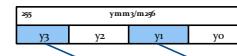
Instrukcja kompresji

VPUNPCKHBW / VPUNPCKHWD / VPUNPCKHDQ / **VPUNPCKHQDQ**

vpunpckhqdq xmm1, xmm2, xmm3/m128

vpunpckhqdq ymm1, ymm2, ymm3/m256 (AVX2)

Starze poczwórne słowa ze 128 bitowych części rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepłotem do rejestru celu ymm1/xmm1.



Bity od 128 do 0 są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

62

Instrukcje wyluskiwania / wstawiania

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

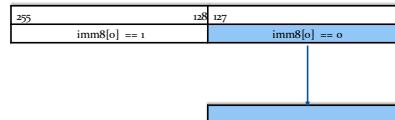
63

Instrukcja wyluskiwania

VEXTRACTI₁₂₈ / VINSERTI₁₂₈

vextracti₁₂₈ xmm1/m128, ymm2, imm8 (AVX2)

Przepisuje 128 bitów z rejestru ymm2 do rejestru xmm1 lub m128. Jeśli zerowy bitu bajtu sterującego imm8[o] = 0 przepisuje młodszą część, a jeśli imm8[o] = 1 przepisuje starszą część rejestru ymm2.



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

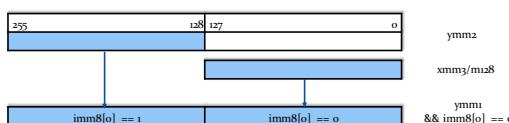
64

Instrukcja wstawiania

VEXTRACTI₁₂₈ / VINSERTI₁₂₈

vinserti₁₂₈ ymm1, ymm2, xmm3/m128, imm8 (AVX)

Przepisuje cały rejestr ymm2 do ymm1 następnie przepisuje rejestr xmm3 lub m128 również do rejestru celu ymm1 zależnie od ustawienia bitu bajtu sterującego: imm8[o] = 0 przepisuje xmm3/m128 na młodszą część, gdy imm8[o] = 1 na starszą część celu ymm1.



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

65

Instrukcje tasowania

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

66

Instrukcja tasowania VPSHUFB

vpshufb xmmi1, xmmz, xmm3/m128

vpshufb ymmi, ymmz, ymm3/m256 (AVX2)

Tasuje bajty z xmmi1/ymmi, w zależności od bitu znaku kolejnych bajtów rejestru xmm3/m128 / ymm3/m256. Jeśli bit znaku jest ustawiony odpowiedni bajt rejestru celu xmmi1/ymmi jest zerowany, jeśli bit znaku xmmi1/ymmi / m128/m256 nie jest ustawiony wówczas z takiego bajtu jest tworzony 4 bitowy indeks wskazujący numer bajtu ze 128-bitowej części, który ma być przepisany z xmmi1/ymmi do właściwego xmmi1/ymmi.

```
i = numer bajtu
if xmm3/m128[i][7] == 1 then xmmi1[i] = 0
else { index[3..0] = xmm3/m128[i][3..0]
        xmmi1[i] = xmm2[index]
    }
```

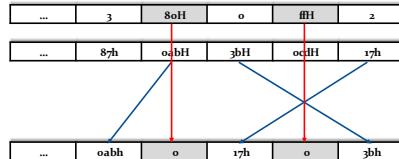
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

67

Instrukcja tasowania VPSHUFB

VPSHUFB



xmm3/m128
xmm2
xmmi1

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

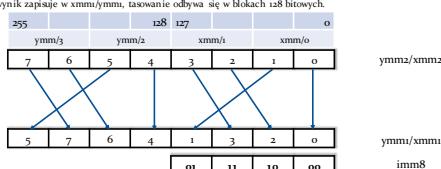
68

Instrukcja tasowania VPSHUFD

vpshufd xmmi1, xmmz/m128, imm8

vpshufd ymmi, ymmz/m256, imm8 (AVX2)

Tasuje po dwie słowa z rejestru xmmi1/ymmi/ymmz/m256 według dwubitowych wartości bajtu sterującego imm8 (argument kolejności), wynik zapisuje w xmmi1/ymmi, tasowanie odbywa się w blokach 128 bitowych.



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

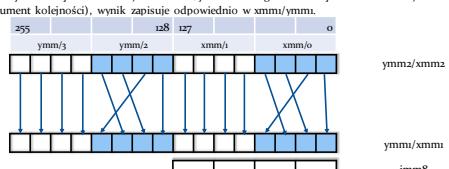
69

Instrukcja tasowania VPSHUFLW

vpshuflw xmmi1, xmmz/m128, imm8 (AVX)

vpshuflw ymmi, ymmz/m256, imm8 (AVX2)

Tasuje wektory młodszych słów z rejestru xmmi1/ymmi2 według dwubitowych wartości bajtu sterującego imm8 (argument kolejności), wynik zapisuje odpowiednio w xmmi1/ymmi.



ymmz/xmm2
ymmi/xmmi
imm8

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

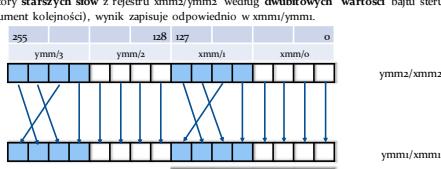
70

Instrukcja tasowania VPSHUFHW

vpshuflw xmmi1, xmmz/m128, imm8 (AVX)

vpshuflw ymmi, ymmz/m256, imm8 (AVX2)

Tasuje wektory starszych słów z rejestru xmmi1/ymmi2 według dwubitowych wartości bajtu sterującego imm8 (argument kolejności), wynik zapisuje odpowiednio w xmmi1/ymmi.



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

71

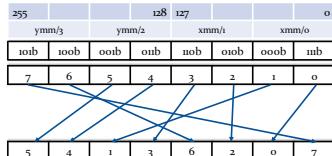
Instrukcje permutacji

Instrukcja ta sowania

VPERMD

vpermD ymm1, ymm2, ymm3/m256 (AVX2)

Wykonuje permutację wektorów podwójnych słów z rejestru **ymm3/m256** według porządku podanego w **ymm2**, najmłodsze **3 bity** odpowiedniego podwójnego słowa rejestru ymm3 wyznaczają **z którego miejsca** w ymm3/m256 zostanie skopiowane podwójne słowo do miejsca położenia „**adresu**” (ymm2). Wynik jest zapisywany w **ymm1**.



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

73

ymm3/m256
ymm2/xmm2

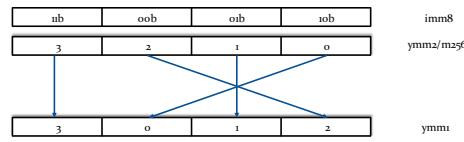
ymm1/xmm1

Instrukcja permutacji

VPERMQ

vpermQ ymm1, ymm2/m256, imm8 (AVX2)

Wykonuje permutację wektorów poczwórnego słowa z rejestru **ymm2/m256** według porządku określonego w **imm8**, kolejne dwubitowe pola **imm8[0..3]** określają, spod którego indeksu „**adresu**” zostaną skopiowane poczwórne słowa z **ymm2/m256**. Wynik jest zapisywany do **ymm1**.



Bioryd zbiory do MSB są zeroowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

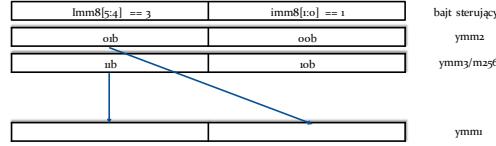
74

Instrukcja permutacji

VPERM2I128

vperm2i128 ymm1, ymm2, ymm3/m256, imm8 (AVX2)

Wykonuje **permutację** dwóch wektorów 128 bitowych z rejestrów **ymm2** oraz **ymm3/m256**, bajt sterujący **imm8** odpowiada za sposób przepisania, pola **imm8[5..4]** i **imm8[0..1]** są indeksami wskazującymi skąd należy pobrać starszą i młodszą część rejestru celu, bity **imm8[7] = 1** i **imm8[3] = 1** powodują wyzerowanie starszej i młodziej części.



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

75

bajt sterujący
ymm2
ymm3/m256

ymm1

Instrukcja permutacji przykład: transponowanie macierzy dynamicznej

void Transponuj4x4(double **tab) { __asm {

```
push esi;
mov esi, tab
mov eax, [esi]
mov ecx, [esi + 8]
mov edx, [esi + 12]
mov esi, [esi + 4]
vmovdqu ymm0, ymmword ptr [eax]
vmovdqu ymm1, ymmword ptr [eax]
vmovdqu ymm0, ymmword ptr [ecx]
vmovdqu ymm2, ymmword ptr [ecx]
vmovdqu ymm3, ymmword ptr [edx]
pop esi;
}
```

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

76

```
vmovdqu ymm1, ymmword ptr [edx]
vperm2i128 ymm3, ymm0, ymm1, 20h
vperm2i128 ymm5, ymm0, ymm1, 31h
vpunpcklqdq ymm0, ymm2, ymm3
vpunpckhqdq ymm1, ymm2, ymm3
vpunpcklqdq ymm2, ymm4, ymm5
vpunpckhqdq ymm3, ymm4, ymm5
```

```
vmovdqu ymmword ptr [eax], ymm0
vmovdqu ymmword ptr [esi], ymm1
vmovdqu ymmword ptr [ecx], ymm2
vmovdqu ymmword ptr [edx], ymm3
```

```
pop esi;
}
```

Instrukcje mieszające

VPBLENDVB

vpblendvb xmm1, xmm2, xmm3/m128, xmm4 (AVX)

vpblendvb ymm1, ymm2, ymm3/m256, ymm4 (AVX2)

Miesza wektory bajtów z rejestru **xmm2/ymm2** oraz **xmm3/ymm3** lub **m128/m256** **według** bitu znaku każdego bajtu w **xmm4/ymm4**, wynik zapisuje w **xmm1/ymm1**.

```
i < 0, 1> lub <0, 1>, 31 - indeks bajtu
if źródł0[i][j] >= 1 => cel[i] = źródł0z[i]
else cel[i] = źródł0w[i]

if xmm4[i][j] >= 1 => xmm1[i] = xmm3/m256[i]
else xmm1[i] = xmm2[i]

if ymm4[i][j] >= 1 => ymm1[i] = ymm3/m256[i]
else ymm1[i] = ymm2[i]
```

Bioryd zbiory do MSB są zeroowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

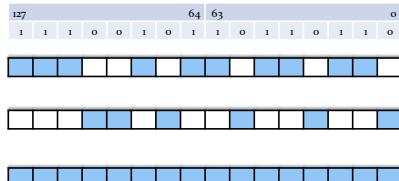
78

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

77

Instrukcja miesząca VPBLENDVB



xmm4
bit znaku
xmm3/m128

xmm2

xmmu

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

79

Instrukcja miesząca VPBLENDW

vblendw ymm1, ymm2, ymm3/m128, imm8 (AVX)
vblendw ymm1, ymm2, ymm3/m256, imm8 (AVX2)

W oparciu o bajt kontrolny miesza wektory slow; wybiera elementy wektora z rejestru xmm3/ymm3 lub m128/m256 dla imm8[i] = i, albo elementy wektora xmm2/ymm2 dla imm8[i] = 0. Dla indeksu 8-15 należy wziąć imm8[i]-8]. Wynik zapisuje w xmmu/ymmu.

```
i < 0, 7>, lub <0, 15> - indeks slowa
if imm8[i] modulo 8 = i then cel[i] = źródło2[i]
else cel[i] = źródło1[i]
if imm8[i] = 0 then xmmu[i] = xmm3/m128[i]
else xmmu[i] = xmm2[i]
if imm8[i] modulo 8 = i then ymmu[i] = ymm3/m256[i]
else ymmu[i] = ymm2[i]
```

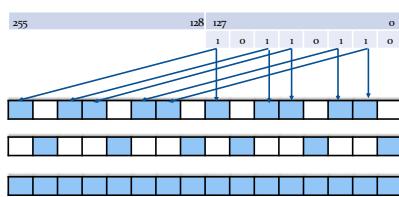
Bajty z 8-15 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

80

Instrukcja miesząca VPBLENDW



imm8

xmm3/m128

xmm2

xmmu

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

81

Instrukcja miesząca VPBLENDDD

vblenddd ymm1, ymm2, ymm3/m256, imm8 (AVX2)

Miesza wektory podwójnych słów z rejestru ymm2 oraz ymm3 lub m256, w oparciu o specyfikację z bajtu kontrolnego imm8, wynik zapisuje w ymmu.

```
i < 0, 7> - indeks podwójnego slowa
if imm8[i] = 1 then cel[i] = źródło2[i]
else cel[i] = źródło1[i]
```

```
if imm8[i] = 1 then ymmu[i] = ymm3/m256[i]
else ymmu[i] = ymm2[i]
```

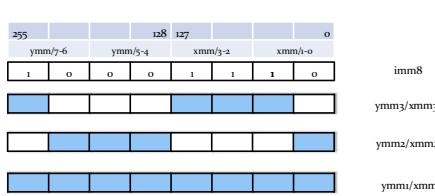
Bajty z 8-15 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

82

Instrukcja miesząca VPBLENDDD



imm8

ymm3/xmm3

ymm2/xmm2

ymmu/xmmu

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

83

Instrukcje rozgłaszające

84

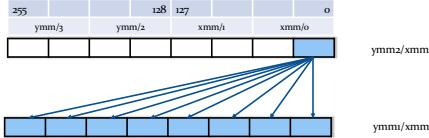
Instrukcja rozgłasza

VPBROADCAST[B/W/D/Q]

`vpbroadcast[b/w/d/q] xmmi, ymm2/m8/m16/m32/m64 (AVX2)`

`vpbroadcast[b/w/d/q] ymmi, ymm2/m8/m16/m32/m64 (AVX2)`

Rozgłasza tę samą wartość 8/16/32/64 bitową ze źródła ymmi/ymm2 lub pamięci m8/m16/m32/m64 do wszystkich elementów wektora rejestru celu xmmi/ymmi. Jeśli operand źródłowy jest rejestrem wartości rozgłoszana w najmłodszym elemencie wektora. Bity od 128/256 do MSB są zerowane.



(C) KISI d.KIK PCz 2023

Programowanie nielokalizowane

85

Instrukcje zbierania

Instrukcja zbierania

VPGATHER[D/Q][D/Q]

`vpgatherdd xmmi, vm32x, xm13`

`vpgatherqd xmmi, vm64y, xmms3`

`vpgatherdd ymmi, vm32y, ymm3`

`vpgatherqd ymmi, vm64y, ymm3`

`vpgatherdq xmmi, vm32x, xm13`

`vpgatherdq ymmi, vm64x, xmms3`

`vpgatherhq ymmi, vm32y, ymm3`

`vpgatherhq ymmi, vm64y, ymm3`

Bit y8 zign. do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie nielokalizowane

87

Drugie [D/Q] dotyczy typu danych wartości pobranych z pamięci.

Pierwsze [D/Q] dotyczy adresu, jednocześnie określa maksymalną ilość pobieranych elementów z pamięci.

Instrukcja kompletuje wektor ymmi/ymm3 używając adresów w postaci podwójnych/poczwrótnych słów zdefiniowanych w `vm32x/y/vm64x/y` używając jako indeksów podwójnych/poczwrótnych słów zapisanych w `xmm3/ymm2` do wskazanej lokalizacji pamięci składowane są wartości podwójnego/poczwrórnego słowa.

Pobierane z pamięci wartości są zapisywane do rejestru celu `xmmi/ymmi` tylko wówczas gdy najstarsze bity odpowiednich elementów wektora maski `xmm3/ymm3` są równe 1.

Instrukcja zbierania (szczegółowo) AVX2

VPGATHER[D/Q][D/Q]

`adres_fizyczny[i] = adres_bazowy + index[i]*skalowanie + przesunięcie`

adres_bazowy – adres danych, określa użyty rejestr GPR

index[i] – i-ty element rejestru `xmm2/ymm2` (`xmm2/ymm2` używane są jedynie indeksy)

skalowanie – określa rozmiar danych (1, 2, 4, 8)

przesunięcie – wartość w bajtach

(C) KISI d.KIK PCz 2023

Programowanie nielokalizowane

88

Instrukcja zbierania (szczegółowo) AVX2

VPGATHER[D/Q][D/Q]

Adresowanie cd.

W opisie instrukcji `vm32x` wskazuje wektor czterech 32-bitowych wartości adresów dla konkretnego `xmm`, `vm32y` wektor ośmiu 32-bitowych wartości indeksów dla konkretnego `ymm`.

Notacja `vm64x` i `vm64y` wskazuje analogicznie na maksymalnie dwa lub cztery adresy.

(C) KISI d.KIK PCz 2023

Programowanie nielokalizowane

89

Instrukcja zbierania (szczegółowo) AVX2

VPGATHER[D/Q][D/Q]

Działanie instrukcji gather

Pobiera z pamięci o wskazanej lokalizacji określonej tu jako **adres_fizyczny** wartości podwójnych/poczwrótnych słów i zapisuje je do rejestru celu `ymmi/ymm3` tylko wówczas gdy bit znaku odpowiadającego elementu maski `ymm3/xmm3` jest równy jeden, jeśli bit znaku jest równy zero w rejestrze celu zostaje wartość poprzednia. Po wykonaniu operacji pobierania z pamięci elementy maski są zerowane.

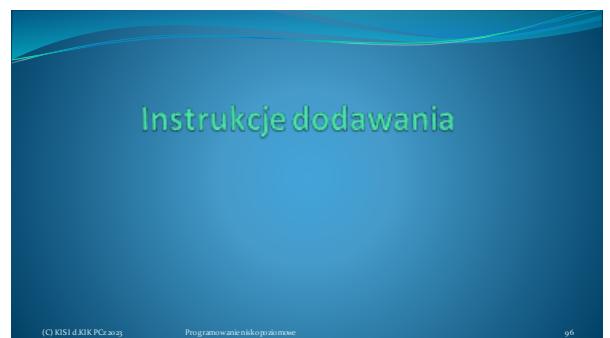
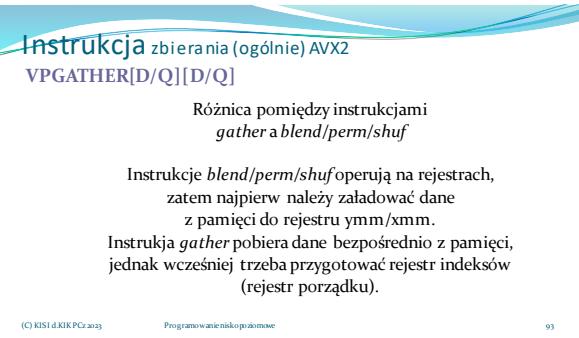
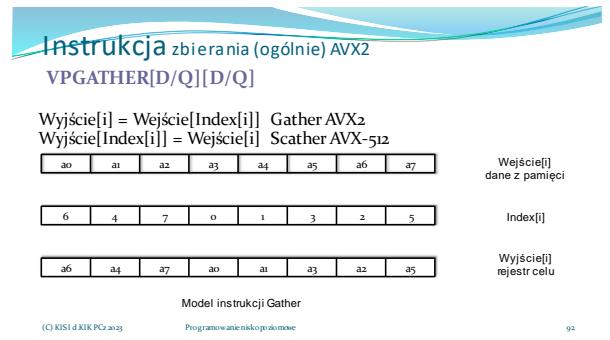
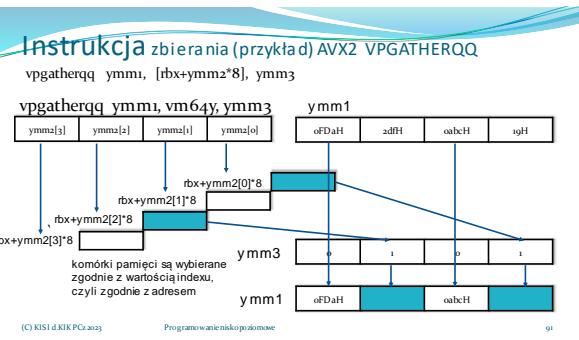
`if xmm3[i][63/31] then xmmi[i] ← [adres_fizyczny(xmm2[i])]`

`if ymm3[i][63/31] then ymmi[i] ← [adres_fizyczny(ymm2[i])]`

(C) KISI d.KIK PCz 2023

Programowanie nielokalizowane

90



Instrukcja dodawania

VPADD[B/W/D/Q]

vpadd[b/w/d/q] xmm1, xmm2, xmm3/m128
vpadd[b/w/d/q] ymm1, ymm2, ymm3/m256 (AVX2)

Do wartości bajtów/słów/podwójnych słów/poczwórnego słów z rejestru **xmm2/ymm2** są dodawane równolegle odpowiednie wartości z rejestru **xmm3/ymm3** lub z pamięci **m128/m256**, wynik jest zapisywany w rejestrze **xmm1/ymm1**.

cel[i] = źródło1[i] + źródło2[i]
xmm1[i] = xmm2[i] + xmm3/m128[i]
ymm1[i] = ymm2[i] + ymm3/m256[i]

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

97

Instrukcja dodawania

VPADDQ

255	192	191	128	127	64	63	o
yymm/3	yymm/2	yymm/1	yymm/o				
+	+	+	+	+	+	+	
=	=	=	=	=	=	=	

ymm2/xmm2

ymm3/xmm3
m256/m128

ymmu/xmmu

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

98

Instrukcja dodawania

VPADDS[B/W]

vpaddsb xmm1, xmm2, xmm3/m128
vpaddsb ymm1, ymm2, ymm3/m256 (AVX2)

Dodawanie ze znakami wektorów bajtów/słów z rejestru **xmm2/ymm2** oraz **xmm3/ymm3** lub pamięci **m128/m256**, wynik jest zapisywany z **nasytemiem** w rejestrze **xmm1/ymm1**.

cel[i] = źródło1[i] + źródło2[i]
xmm1[i] = xmm2[i] + xmm3/m128[i]
ymm1[i] = ymm2[i] + ymm3/m256[i]

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

99

Instrukcja dodawania

VPADDSW

127			64	63	xmm/7	xmm/6	xmm/5	xmm/4	xmm/3	xmm/2	xmm/1	xmm/o
+	+	+	+	+	+	+	+	+	+	+	+	+
=	=	=	=	=	=	=	=	=	=	=	=	=

xmm2

xmm3/m128

xmmu

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

100

Instrukcja dodawania

VPADDUS[B/W]

vpaddusb xmm1, xmm2, xmm3/m128
vpaddusb ymm1, ymm2, ymm3/m256 (AVX2)

Dodawanie bez znaku wektorów bajtów/słów rejestru **xmm2/ymm2** i **xmm3/ymm3** lub pamięci **m128/m256**, wynik jest zapisywany z **nasytemiem** w rejestrze **xmm1/ymm1**.

cel[i] = źródło1[i] + źródło2[i]
xmm1[i] = xmm2[i] + xmm3/m128[i]
ymm1[i] = ymm2[i] + ymm3/m256[i]

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

101

Instrukcja dodawania

VPADDUSW

255	192	191	128	127	64	63	o
yymm/5	yymm/8	xmmn/7					
+	+	+	+	+	+	+	+
=	=	=	=	=	=	=	=

xmm2

xmm3/m128

xmmu

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

102

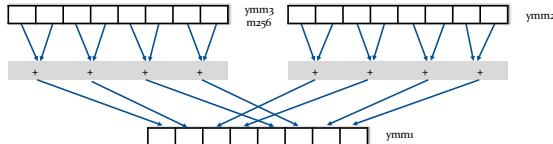
Instrukcja dodawania

VPHADDW / VPHADDD / VPHADDSW

vphaddd xmm1, xmm2, ymm3/m128

vphaddd ymm1, ymm2, ymm3/m256

Horizontalne dodawanie siedmiu słów/podwójnych słów zapisywane wyniku z przeplotem po 64 bity. Jako najmłodsze są zapisywane sumy z rejestru xmm2. Ostatnia w/w instrukcja jest dodawaniem słów z nasyceniem.



(C) KISI d.KIK PCz 2023

Programowanie niko poziomie

103

Instrukcja dodawania - przykład:

```
void vec_avx_add_int
(int* t1, int* t2, int* t3, int n)
{
    __asm {
        cd.
        sub ecx, 32; podwójne słowo
        vmovdqa ymm0, ymmword ptr[esi + ecx];
        vmovdqa ymm1, ymmword ptr[edx + ecx];
        vpadddd ymm2, ymm1, ymm0;
        vmovdqa ymmword ptr[edi + ecx], ymm2;

        jnz petla;
    }
}
```

(C) KISI d.KIK PCz 2023

Programowanie niko poziomie

104

Instrukcje odejmowania

(C) KISI d.KIK PCz 2023

Programowanie niko poziomie

105

Instrukcja odejmowania

VPSUB[B/W/D/Q]

vpsub[b/w/d/q] xmm1, xmm2, xmm3/m128

vpsub[b/w/d/q] ymm1, ymm2, ymm3/m256 (AVX2)

Od wartości bajtu/słowa/podwójnego słowa/poczwórnego słowa z rejestru **xmm2/ymm2** są odejmowane równolegle odpowiednie wartości z rejestru **xmm3/ymm3** lub z pamięci **m128/m256**, wynik jest zapisywany w rejestrze **xmm1/ymm1**.

$$\text{cel}[i] = \text{źródło1}[i] - \text{źródło2}[i]$$

$$\text{xmm1}[i] = \text{xmm2}[i] - \text{xmm3/m128}[i]$$

$$\text{ymm1}[i] = \text{ymm2}[i] - \text{ymm3/m256}[i]$$

Bit od 128/256 do MSB są zeroowane.

(C) KISI d.KIK PCz 2023

Programowanie niko poziomie

106

Instrukcja odejmowania

VPSUBQ

255	192	191	128	127	64	63	0	
ymm3/3	ymm2/2	ymm1/1	xmm1/o					
-	-	-	-	-				
=	=	=	=	=				

ymm2/xmm2

ymm3/3	ymm2/2	xmm1/o

ymm3/xmm3
m256/m128

ymmu	xmm1

ymmu/xmm1

(C) KISI d.KIK PCz 2023

Programowanie niko poziomie

107

Instrukcja odejmowania

VPSUB[U]S[B/W]

vpsub[u]s[b/w] xmm1, xmm2, xmm3/m128

vpsub[u]s[b/w] ymm1, ymm2, ymm3/m256 (AVX2)

Od wartości ze znakiem/ bez znaku/ wektorów bajtów/słów rejestru **xmm2/ymm2** są odejmowane odpowiednie wartości rejestru **xmm3/ymm3** lub pamięci **m128/m256**, wynik jest zapisywany w rejestrze **xmm1/ymm1 z nasyceniem**.

$$\text{cel}[i] = \text{źródło1}[i] - \text{źródło2}[i]$$

$$\text{xmm1}[i] = \text{xmm2}[i] - \text{xmm3/m128}[i]$$

$$\text{ymm1}[i] = \text{ymm2}[i] - \text{ymm3/m256}[i]$$

Bit od 128/256 do MSB są zeroowane.

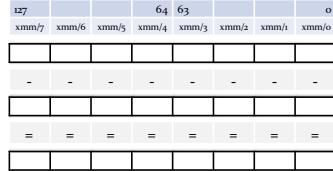
(C) KISI d.KIK PCz 2023

Programowanie niko poziomie

108

Instrukcja odejmowania

VPSUBW



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

109

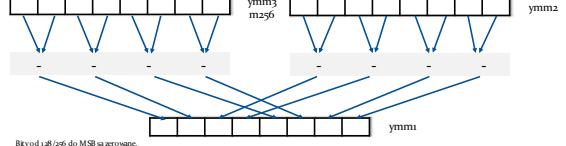
Instrukcja odejmowania

VPHSUBW / VPHSUBD / VPHSUBSW

vphsubd xmm1, xmm2, xmm3/m256

vphsubw ymm1, ymm2, ymm3/m256

Horyzontalne odejmowanie sąsiednich słów/podwójnych słów w ramach 128-bitowych części. **Od młodszego elementu wektora test odejmowany, stoczy, oraz jako najmłodsze** są zapisywane różnice z rejestru xmm1. Ostatnia instrukcja jest odejmowaniem słów z nasyceniem.



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

110

Instrukcja sumowanie modułów różnic

VPSADBW

vpsadbw xmm1, xmm2, xmm3/m128

vpsadbw ymm1, ymm2, ymm3/m256 (AVX2)

Od wartości wektorów bajtów rejestru **xmm2/ymmm2 są odejmowane** odpowiednie wartości rejestru **xmm3/ymmm3** lub pamięci **m128/m256**, następnie obliczane są wartości absolute i ich sumy po 8 elementów, wynik jest zapisywany w rejestrze **xmm1/ymmm1** dla zestawów 8-elementowych

Bity od 128/256 do MSB są zeroowane.

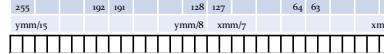
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

111

Instrukcja sumowanie modułów różnic

VPSADBW



ymm2

ymmm3/m256
różnica
abs

ymmm1

112

Instrukcje mnożenia

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

113

Instrukcja mnożenia

VPMULL[W/D]

vpmull[w/d] xmm1, xmm2, xmm3/m128

vpmull[w/d] ymm1, ymm2, ymm3/m256 (AVX2)

Mnożenie wektorów słów/podwójnych słów ze znakiem z rejestru **xmm2/ymmm2** przez odpowiadające im wartości z **xmm3/m128 / ymm3/m256**. **iloczyny** są podwójnymi/poczwórnymi słowami, jednak do rejestru celu **xmm1/ymmm1** są zapisywane **jedynie** młodsze słowa/podwójne słowa **iloczynów**.

$$\text{cel}[i] = \text{lo}(\text{źródło1}[i] * \text{źródło2}[i])$$

$$\text{xmm1}[i] = \text{lo}(\text{xmm2}[i] * \text{xmm3/m128}[i])$$

$$\text{ymmm1}[i] = \text{lo}(\text{ymmm2}[i] * \text{ymmm3/m256}[i])$$

Bity od 128/256 do MSB są zeroowane.

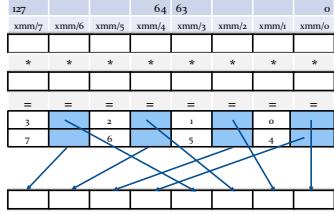
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

114

Instrukcja mnożenia

VPMULLW



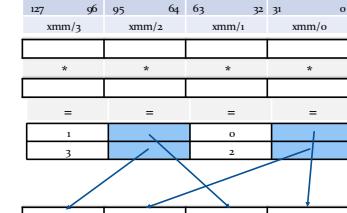
(C) KISI d.KIK PCz 2023

Programowanie niukpoziomowe

115

Instrukcja mnożenia

VPMULLD



ymm2/xmm2

ymm3/xmm3
m256/m256

iloczyny (poczwarne słowa)

ymm1/xmm1

116

(C) KISI d.KIK PCz 2023

Programowanie niukpoziomowe

115

Instrukcja mnożenia

VPMUH[U]W

vpmulh[u]w ymm1, xmm2, xmm3/m128
vpmulh[u]w ymm1, ymm2, ymm3/m256 (AVX2)

Mnożenie wektorów słów bez znaku/u znakiem (U) z rejestru xmm2/ymm2 przez odpowiadające im wartości z xmm3/m128 / ymm3/m256, iloczyny są podwójnymi słowami, jednak do rejestru celu xmm1/ymm1 są zapisywane jedynie starsze słowa, iloczynów:

- cel[i] = hi (źródło1[i] x źródło2[i])
- xmm1[i] = hi (xmm2[i] x xmm3/m128[i])
- ymm1[i] = hi (ymm2[i] x ymm3/m256[i])

Bity od 128 do MSB są zerowane.

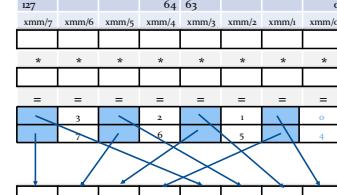
(C) KISI d.KIK PCz 2023

Programowanie niukpoziomowe

118

Instrukcja mnożenia

VPMULHW



xmm2

xmm3/m128

wynik mnożenia
(podwójne słowa)

xmm1

119

(C) KISI d.KIK PCz 2023

Programowanie niukpoziomowe

118

Instrukcja mnożenia

VPMULHRSW

vpmulhsw xmm1, xmm2, xmm3/m128
vpmulhsw ymm1, ymm2, ymm3/m256 (AVX2)

Mały wектор słów ze znakiem i zaokrągleniem, wartości z rejestru xmm1/ymm1 przez wartości z rejestru xmm3/m128 / ymm3/m256, podwójne słowa iloczynów zostają przesunięte w prawo o 14 bitów oraz zostaje dodana jedynka w celu zaokrąglenia wartości. Bity od 1 do 16 są zapisywane w celu.

```
cel[i] = ( (źródło1[i] * źródło2[i] >> 14) + 1 ) >> 1
xmm1[i] = ((xmm2[i] * xmm3/m128[i] >> 14) + 1) >> 1
ymm1[i] = ((ymm2[i] * ymm3/m256[i] >> 14) + 1) >> 1
```

Bity od 128 do MSB są zerowane.

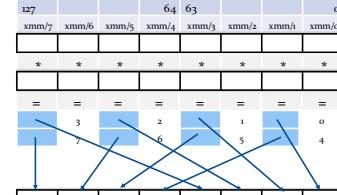
(C) KISI d.KIK PCz 2023

Programowanie niukpoziomowe

120

Instrukcja mnożenia

VPMULHRSW



xmm2

xmm3/m128

(iloczyn[i]>>14 + 1) >> 1

xmm1

121

(C) KISI d.KIK PCz 2023

Programowanie niukpoziomowe

120

Instrukcja mnożenia

VPMUL[U]DQ

vpmul[u]dq xmmi, xmm2, xmm3/m128

vpmul[u]dq ymmi, ymm2, ymm3/m256 (AVX2)

Mnożenie **co drugich** elementów wektora podwójnych słów ze znakiem/bez znaku (U) xmm2/ymm2 **z co drugimi** elementami podwójnych słów ze znakiem ymm3/m128 / ymm3/m256, iloczyny są zapisywane w xmmi/ymmi jako wektor poczwórnego słów **ze znakiem**.

cel[i] = źródło[i][i] * źródło2[i][i]

xmmi[i] = xmm2[2i] * xmm3/m128[2i]

ymmi[i] = ymm2[2i] * ymm3/m256[2i]

Bit od 127/255 do MSB są zeroowane.

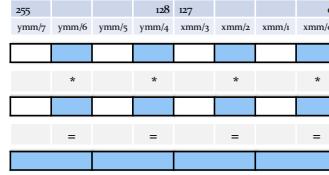
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

122

Instrukcja odejmowania

VPMULDQ



ymm2

ymm3/m256

ymmi

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

123

Instrukcja mnożenia

VPCLMULQDQ

vpclmulqdq xmmi, xmm2, xmm3/m128, imm8 (AVX)

Mnożenie poczwórnego słowa z xmm2 przez poczwórnego słowo z xmm3/m128, iloczyn jest zapisywany w xmmi. Bity imm8[0] i imm8[4] wybierają młodsze lub starsze (o lub i) poczwórne słowa z rejestrów xmm2 i xmm3/m128, które zostaną pomnożone.

```
if imm8[0] = 0|| && imm8[4] = 0||i => cel <- źródło[ o|i] * źródło2[ o|i]
if imm8[0] = o && imm8[4] = o => xmmi <- xmm2[63:0] * xmm3/m128[63:0]
if imm8[0] = o && imm8[4] = i => xmmi <- xmm2[63:0] * xmm3/m128[127:64]
if imm8[0] = 1 && imm8[4] = o => xmmi <- xmm2[127:64] * xmm3/m128[63:0]
if imm8[0] = 1 && imm8[4] = i => xmmi <- xmm2[127:64] * xmm3/m128[127:64]
```

Bit od 127/255 do MSB są zeroowane.

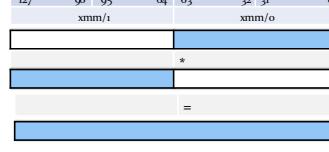
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

124

Instrukcja mnożenia

VPCLMULQDQ



ymm2/xmm2
&& imm8[0] = 0

ymm3/xmm1 lub m256/m128
&& imm8[4] = 1

ymmi/xmmi

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

125

Instrukcje mnożenia z dodawaniem

Dla liczb zmiennoprzecinkowych odpowiednikiem bardziej zaawansowanym są instrukcje FMA

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

126

Instrukcja mnożenia i dodawania

VPMADDWD

vpmaddwd xmmi, xmm2, xmm3/m128

vpmaddwd ymmi, ymm2, ymm3/m256 (AVX2)

Mnoży słowa z rejestru xmm2/ymm2 przez słowa z rejestru xmm3/m128 / ymm3/m256, iloczyny są podwójnymi słowami, następnie kolejne podwójne słowa dodaje horyzontalnie i zapisuje jako podwójne słowa w rejestrze celu xmmi/ymmi.

```
cel[i] = źródło[i][i] * źródło2[i][i] + źródło[i][i+1] * źródło2[i+1][i]
xmmi[i] = xmm2[2i] * xmm3/m128[2i] + xmm2[2i+1] * xmm3/m128[2i+1]
ymmi[i] = ymm2[2i] * ymm3/m256[2i] + ymm2[2i+1] * ymm3/m256[2i+1]
```

Bit od 127/255 do MSB są zeroowane.

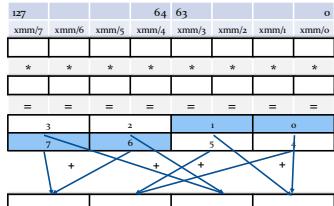
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

127

Instrukcja mnożenia

VPMADDWD



(C) KISI d.KIK PCz 2023

Programowanie nikopoziomowe

xmm2

xmm3/m256

iloczyny

sumuje sąsiednie dwa iloczyny

xmm1

128

Instrukcja mnożenia i dodawania

VPMADDUBSW

vpmadubsw xmm1, xmm2, xmm3/m128

vpmadubsw ymm1, ymm2, ymm3/m256 (AVX2)

Mnoży bajty bez znaku z rejestru xmm2/ymm2 przez bajty z rejestru xmm3/m128 / ymm3/m256, iloczyny są słowne, następnie dwa kolejne słowa dodaje horyzontalnie i zapisuje z nasyceniem, jako słowa w rejestrze celu xmm1/ymm1.

cel[i] = źródło1[2i] * źródło2[2i] + źródło1[2i+1] * źródło2[2i+1]

xmm1[i] = xmm2[2i] * xmm3/m128[2i] + xmm2[2i+1] * xmm3/m128[2i+1]

ymm1[i] = ymm2[2i] * ymm3/m256[2i] + ymm2[2i+1] * ymm3/m256[2i+1]

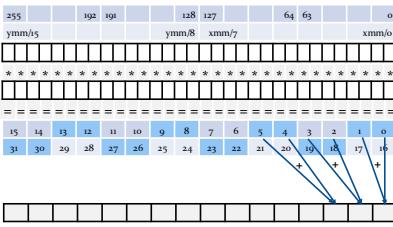
Bity od 128 do MSB są zeroowane.

Programowanie nikopoziomowe

129

Instrukcja sumowanie modułów

VPMADDUBSW



(C) KISI d.KIK PCz 2023

Programowanie nikopoziomowe

ymm2

ymm3/m256

iloczyny

ymmu

129

Operacje arytmetyczne AVX cd.

- Wartość maksymalna:** VPMAXUB, VPMAXUW
VPMAXUD, VPMAXSB, VPMAXSW, VPMAXSD

- Wartość minimalna:** VPMINUB, VPMINUW, VPMINUD
VPMINSB, VPMINSW, VPMINSD, VPHMINPOSUW

- Średnia:** VPAVGB, VPAVGW

- Wartość bezwzględna liczby:** VPABSB, VPABSW, VPABSD

- Negacja / zero / zachowanie:** VPSIGNB, VPSIGNW, VPSIGND

(C) KISI d.KIK PCz 2023

Programowanie nikopoziomowe

130

Instrukcje wartość maksymalna

VPMAX[U/S][B/W/D]

vpmax[u/s][b/w/d] xmm1, xmm2, xmm3/m128

vpmax[u/s][b/w/d] ymm1, ymm2, ymm3/m256 (AVX2)

Porównuje bez znaku/ze znakiem wartości w wektorkach bajtów/słów/podwójnych słów rejestru xmm2/ymm2 z odpowiednimi wartością xmm3/ymm3 lub pamięci m128/m256, wektory wartości maksymalnych są zapisywane w rejestrze xmm1/ymm1.

if źródło1[i] > źródło2[i] then cel[i] = źródło1[i] else cel[i] = źródło2[i]

```
if xmm2/ymm2[i] > xmm3/m128[i] / ymm3/m256[i]
    then xmm1/ymm1[i] = xmm2/ymm2[i]
else xmm1/ymm1[i] = xmm3/m128[i] / ymm3/m256[i]
```

Bity od 128 do MSB są zeroowane.

Programowanie nikopoziomowe

131

(C) KISI d.KIK PCz 2023

Programowanie nikopoziomowe

132

Instrukcja dodawania VPMAX[U/S][B/W/D]

255	ymm/7	ymm/6	ymm/5	ymm/4	xmm/3	xmm/2	xmm/1	xmm/0	o
>	>	>	>	>	>	>	>	>	
=	=	=	=	=	=	=	=	=	
max	max	max	max	max	max	max	max	max	

xmm2
porównania
xmm3/m128
xmm1

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

134

Instrukcja wartość maksymalna VPMIN[U/S][B/W/D]

vpmax[u/s][b/w/d] xmm1, xmm2, xmm3/m128
vpmmin[u/s][b/w/d] ymm1, ymm2, ymm3/m256 (AVX2)
Porównuje bez znaku/ze znakiem wartości w wektorach bajtów/słów/podwójnych słów rejestru xmm2/ymm2 z odpowiednimi wartościami xmm3/ymm3 lub pamięci m128/m256, wektory wartości minimalnych są zapisywane w rejestrze xmm1/ymm1.

```
if źródło1[i] < źródło2[i] then cel[i] = źródło1[i] else cel[i] = źródło2[i]
if xmm2/ymm2[i] < xmm3/m128[i] / ymm3/m256[i]
    then xmm1/ymm1[i] = xmm2/ymm2[i]
else xmm1/ymm1[i] = xmm3/m128[i] / ymm3/m256[i]
```

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

135

Instrukcja dodawania VPMIN[U/S][B/W/D/Q]

255	ymm/7	ymm/6	ymm/5	ymm/4	xmm/3	xmm/2	xmm/1	xmm/0	o
<	<	<	<	<	<	<	<	<	
=	=	=	=	=	=	=	=	=	
min	min	min	min	min	min	min	min	min	

xmm2
porównania
xmm3/m128
xmm1

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

136

Instrukcje wartość średnia

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

137

Instrukcja wartość średnia VPAVG[B/W]

vpavg[b/w] xmm1, xmm2, xmm3/m128

vpavg[b/w] ymm1, ymm2, ymm3/m256 (AVX2)

Zwala średnią dwóch wartości bez znaku z wektorów bajtów/słów, dodaje wektory rejestrów xmm2/ymm2 z odpowiednimi wartościami xmm3/ymm3 lub pamięci m128/m256, sumę zaokrąglającą oraz dzieli przez dwa poprzez przesunięcie bitowe o jeden w prawo, wynik zapisuje w rejestrze xmm1/ymm1.

```
cel[i] = (źródło1[i] + źródło2[i] + i) >> 1
xmm1[i] = (xmm2[i] + xmm3/m128[i] + i) >> 1
ymm1[i] = (ymm2[i] + ymm3/m256[i] + i) >> 1
```

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

138

Instrukcja dodawania VPAVG[B/W]

127	xmm/7	xmm/6	xmm/5	xmm/4	xmm/3	xmm/2	xmm/1	xmm/0	o
+	+	+	+	+	+	+	+	+	
=	=	=	=	=	=	=	=	=	
+1	+1	+1	+1	+1	+1	+1	+1	+1	
>>1	>>1	>>1	>>1	>>1	>>1	>>1	>>1	>>1	
avg	avg	avg	avg	avg	avg	avg	avg	avg	

xmm2

xmm3/m128

xmm1

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

139



Instrukcja wartość bezwzględna VPABS[B/W/D]

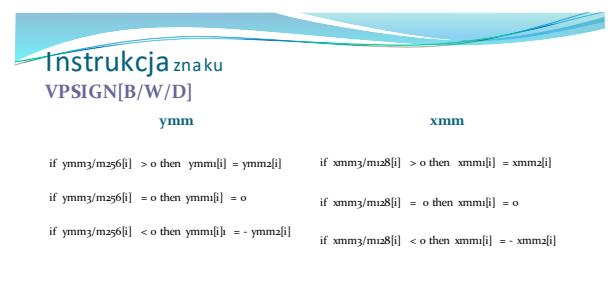
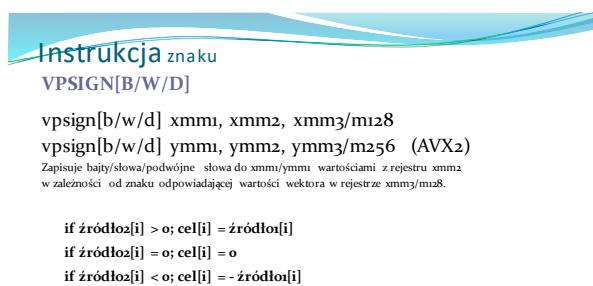
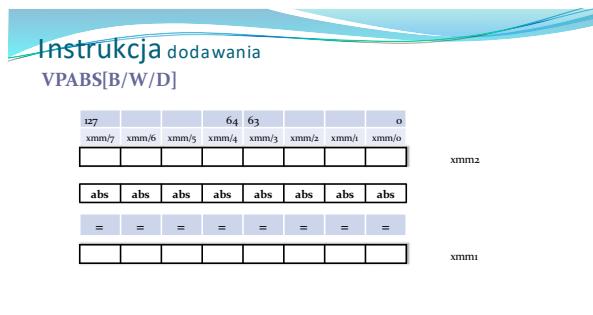
vpabsb xmm1, xmm2
vpabsb ymm1, ymm2 (AVX2)

Oblicza wartość bezwzględną od wartości bajtów/słów/podwójnych słów rejestru xmm2/ymm2, wynik zapisuje w rejestrze xmm1/ymm1 **baz znaku**.

```
cel[i] = abs(źródlo[i])
xmmi[i] = abs(xmm2[i])
ymmi[i] = abs(ymm2[i])
```

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomowe 141



Instrukcja dodawania

VPSIGN[B/W/D]

127			64	63			o
xmm/7	xmm/6	xmm/5	xmm/4	xmm/3	xmm/2	xmm/1	xmm/0
8	7	6	5	4	3	2	1
-4	5	0	-9	2	6	0	-3

xmm2
xmm3/m128
odczytanie bitu znaku

-8	7	0	-5	4	3	0	-1

xmm

Operacje porównania AVX

- **Porównanie liczb:** VPCMPEQB, VPCMPEQW
VPCMPEQD, VPCMPGTB, VPCMPGTW
VPCMPTD, VPCMPGTQ

- **Porównanie ciągów:**
VPCMPESTRI, VPCMPISTRI, VPCMPESTRM
VPCMPISTRM

Instrukcje porównania

Instrukcja porównania liczb

VPCMPEQ[B/W/D/Q]

127			64	63			o
xmm/7	xmm/6	xmm/5	xmm/4	xmm/3	xmm/2	xmm/1	xmm/0
2	4	9	7	6	5	3	2
EQ							
5	0	1	7	6	4	3	2
=	=	=	=	=	=	=	=
o	o	o	-4	-4	o	-4	-4

xmm3/m128
xmm2
xmm1

Instrukcja porównania liczb

VPCMPGT[B/W/D/Q]

vpcmpgt[b/w/d] xmm1, xmm2, xmm3/m128

vpcmpgt[b/w/d] ymm1, ymm2, ymm3/m128 (AVX2)

Porównuje wartości współrzędnych wektorów bajtów/słów/podwójkowych słów/poczwórkowych słów z rejestru xmm3/mym3 lub pamięci m128/mz6 z odpowiednimi wartościami współrzędnych rejestru xmm1/ymm1, jeśli wartości są równe, odpowiednie współrzędne rejestru celu xmm1/ymm1 są ustawiane na -i, jeśli nie na o.

if źródło1[i] = źródło2[i] then cel[i] = -i else cel[i] = o;
if xmm3/m128[i] = xmm2[i] then xmm1[i] = -i else xmm1[i] = o;
if ymm3/m128[i] = ymm2[i] then ymm1[i] = -i else ymm1[i] = o;

Instrukcja porównania liczb

VPCMPGT[B/W/D/Q]

127	64	63	0	xmm/7	xmm/6	xmm/5	xmm/4	xmm/3	xmm/2	xmm/1	xmm/0
7	2	3	4	6	8	9	-1				
GT	GT	GT	GT	GT	GT	GT	GT				
0	3	4	6	5	7	0	1				
=	=	=	=	=	=	=	=				
0	-1	-1	-1	0	0	0	0				

xmm3/m128

xmm2

xmm1

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

152

Instrukcje porównania ciągów znakowych

• Operacje porównania ciągów znakowych porównują w istocie liczby całkowite.

• Instrukcje te można podzielić na porównujące ciągi znakowe o, ustalonej (znanej) w rejestrach [R/E]AX i [R/E]DX oraz nieznanej, długości.

• Instrukcje CMP na wyjściu tworzą indeks lub maskę, ale wynik porównania jest zapisywany w [R/E]CX/xmmo (brak w wywołaniu instrukcji).

• W instrukcjach tego typu istotne zadanie pełni bajt sterujący imm8, gdzie można zdefiniować to złożone i wieloetapowe porównanie i pełni on w istocie funkcję algorytmu instrukcji

• Instrukcje porównania jako nieliczne w AVX ustawiają flagi

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomowe

153

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

152

Instrukcja porównania ciągów znakowych

VPCMP[E/I]STR[I/M]

Bajt sterujący imm8 (1/4)

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

Typ danych na wejściu

imm8[1:0] = [ob] bajty bez znaku, [ib] słowa bez znaku,
[ob] bajty ze znakiem, [ib] słowa ze znakiem

Operacja (sposób porównania)

imm8[3:2] = [ob] porównanie arytmetyczne, czy w ciągu występują podane bajty/słowa,
[ob] porównanie arytmetyczne większe lub równe dla parzystych elementów wektora
wektora lub mniejsze lub równe dla nieparzystych elementów wektora
[ob] porównuje arytmetycznie, czy odpowiadające sobie wartości są równe
[ib] porównuje arytmetycznie, czy równy (w kolejności) }

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

154

Instrukcja porównania ciągów znakowych

VPCMP[E/I]STR[I/M]

Bajt sterujący imm8 (2/4)

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

Polaryzacja (nadawanie znaku)

imm8[5:4] = [ob] pozytywna polaryzacja (bez zmiany znaku),
[ob] negatywna polaryzacja (ze zmianą znaku),
[ob] stosowanie maski (bez zmiany znaku),
[ib] stosowanie maski (ze zmianą maski) dla elementów nieważnych w reg/mem/sq
przepisywane, w przeciwnym wypadku nieważne są negowane }

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

155

Instrukcja porównania ciągów znakowych

VPCMP[E/I]STR[I/M]

Bajt sterujący imm8 (3/4)

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

Wynik zwracany w postaci indeksu lub maski jest tworzony etapami:

1. logiczne porównanie każdy z każdym (bajt z bajtem / słowo ze słowem)
2. intermediate result 1 (pośrednie zagregowane wyniki porównania - prawdziwe)
3. intermediate result 2 etap poprzedni jest negowany logicznie
4. wzorcany test albo najbardziej/najmniej znaczący bit porównania pkt. 3 (index) albocale porównanie z pkt. 3 w opcji rozszerzenia zerami lub rozszerzenia do bajtu/słowa (maska)

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

156

Instrukcja porównania ciągów znakowych

VPCMP[E/I]STR[I/M]

Bajt sterujący imm8 (4/4)

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

Wszelkie Wyjątki

Dla indeksu (I) = wynik do ECX/RCX
imm8[6] = [ob] z wyniku jest pobierany najmłodszy bit
[ib] z wyniku jest pobierany najstarszy bit }

Dla maski (M) = wynik do xmm0
imm8[6] = [ob] zwracany wynik uzupełniany jest zerami,
[ib] wynik jest rozszerzany do bajtu/słowa (z samymi zerami lub jedynkami) }

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

157

Instrukcja porównania ciągów znakowych VPCMPE[I]STR[I/M]

Instrukcje porównania ciągów znakowych operują na rejestrach **xmm** oraz w sposób określony przez bajt sterujący **imm8**, który jest częścią kodującą instrukcji.

Instrukcje ustawiają flagi arytmetyczne ZF, CF, SF, OF, AF, PF (wyjątek w AVX), jednak znaczenia flag zostały przeciżone z ich zwykłego znaczenia celem dostarczenia dodatkowych informacji o relacji pomiędzy dwoma wejściami.

Instrukcje typu PCMPeSTR wykonywają porównania arytmetyczne między wszystkimi możliwymi parami bajtów lub słów, po jednym z każdego wektora źródłowego. Wartości logiczne tych porównań są następnie agregowane w celu uzyskania wyniku końcowego.

(C) KISI d.KIK PCz 2023

Programowanie nikopozitomowe

158

Instrukcja porównania ciągów znakowych VPCMPE[I]STR[I/M]

Algorytm instrukcji definiowany w bajcie sterującym:

- Ustawienie źródła
- Operacje porównania i agregacji (wyniki pośrednie)
- Polaryzacja
- Wybór wyjścia dla wyniku końcowego

Bajt kontrolny określa spodziewany wynik i kontroluje następujące atrybuty:

- format danych bajt/słowo, ze znakiem/bez znaku imm8[]
- koduje tryb operacji porównania
- określa przetwarzanie pośrednie
- określa operację tworzenia wyjścia zależnie, czy index, czy maska.

Zatem instrukcje porównujące ciągi znakowe bajtów lub słów, **wynikiem jest:**

- maska w xmm0 lub index w [R/E]CX

• ustawione flagi

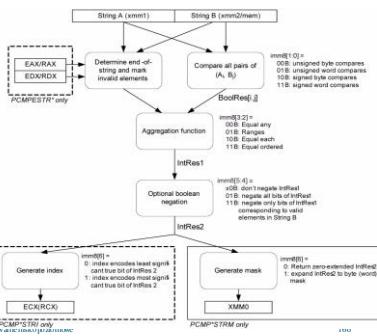
(C) KISI d.KIK PCz 2023

Programowanie nikopozitomowe

159

Instrukcja porównania ciągów znakowych VPCMPE[I]STR[I/M]

VPCMPE[I]STR[I/M]



(C) KISI d.KIK PCz 2023

Programowanie nikopozitomowe

160

Instrukcja porównania ciągów znakowych VPCMPESTRI

vpcmpestri xmm1, xmm2/m128, imm8 (AVX)

Porównuje lańuchy o ustalonej długości z rejestrów xmm1 i xmm2/m128, na wyjściu tworzy index, wynik zapisuje w rejestrze ogólnego przeznaczenia ECX

- E (Explicit) – oznacza lańuchy o ustalonej długości
- I (Index) – oznacza, że instrukcja tworzy na wyjściu index

Operand 1	Operand 2	Długość 1	Długość 2	Wynik
xmm1	xmm2/m128	[R/E]AX	[R/E]DX	ECX

CFlag - Reset if IntRes2 is equal to zero, set otherwise

ZFlag - Set if absolute-value of EDX is < 16 (8), reset otherwise

SFlag - Set if absolute-value of EAX is < 16 (8), reset otherwise

OFlag - IntRes2[0]

AFlag - Reset

PFlag - Reset

(C) KISI d.KIK PCz 2023

Programowanie nikopozitomowe

161

Instrukcja porównania ciągów znakowych VPCMPISTRI

vcpmpistri xmm1, xmm2/m128, imm8 (AVX)

I (Implicit) - oznacza lańuch o nieoznaczonej długości.

I (Index) - oznacza, że instrukcja tworzy na wyjściu index

Porównuje lańuchy o nieustalonej długości z rejestrów xmm1 i xmm2/m128, na wyjściu tworzy index, wynik zapisuje w rejestrze ogólnego przeznaczenia ECX.

Operand 1	Operand 2	Wynik
xmm1	xmm2/m128	ECX

CFlag - Reset if IntRes2 is equal to zero, set otherwise

ZFlag - Set if any byte/word of xmm2/mem128 is null, reset otherwise

SFlag - Set if any byte/word of xmm1 is null, reset otherwise

OFlag - IntRes2[0]

AFlag - Reset

PFlag - Reset

(C) KISI d.KIK PCz 2023

Programowanie nikopozitomowe

162

Instrukcja porównania ciągów znakowych VPCMPESTRM

vcpmestrm xmm1, xmm2/m128, imm8 (AVX)

E (Explicit) - oznacza lańuchy o ustalonej długości

M (Mask) - oznacza, że instrukcja tworzy na wyjściu maskę

Porównuje lańuchy o ustalonej długości z rejestrów xmm1 i xmm2/m128, na wyjściu tworzy maskę, wynik zapisuje w rejestrze xmm0. (nie jest umieszczany w definicji).

Operand 1	Operand 2	Długość 1	Długość 2	Wynik
xmm1	xmm2/m128	[R/E]AX	[R/E]DX	xmm0

CFlag - Reset if IntRes2 is equal to zero, set otherwise

ZFlag - Set if absolute-value of EDX is < 16 (8), reset otherwise

SFlag - Set if absolute-value of EAX is < 16 (8), reset otherwise

OFlag - IntRes2[0]

AFlag - Reset

PFlag - Reset

(C) KISI d.KIK PCz 2023

Programowanie nikopozitomowe

163

Instrukcja porównania ciągów znakowych VPCMPISTRM

vpcmipstrm xmm1, xmm2/m128, imm8 (AVX)

I (Implicit) - oznacza łańcuch o nieoznaczonej długości.

M (Mask) - oznacza, że instrukcja tworzy na wyjściu maskę

Porównuje łańcuchy o nie ustalonej długości z rejestru xmm1 i xmm2/m128, na wyjściu tworzy maskę wynik zapisując w rejestrze **xmmo** (nie jest umieszcany w definicji).

Operand 1	Operand 2	Wynik
xmm1	xmm2/m128	xmmo

CFlag - Reset if IntRes2 is equal to zero, set otherwise

ZFlag - Set if any byte/word of xmm2/mm128 is null, reset otherwise

SFlag - Set if any byte/word of xmm1 is null, reset otherwise

OFlag - IntRes[0]

AFlag - Reset

PFlag - Reset

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

164

Operacje przesunięć AVX liczby całkowite

• Przesunięcie w lewo:

VPSLL[W/D/Q]

VPSLLDQ

VPSRLV[D/Q]

VPSRA[W/D/Q]

• Przesunięcie w prawo:

VPSRL[W/D/Q]

VPSRLDQ

VPSRAV[W/D/Q]

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

165

Instrukcje przesunięć bitowych

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

166

Instrukcja przesunięcia w prawo VPSRL[W/D/Q]

vpsrl[w/d/q] xmm1, xmm2, xmm3/m128 lub imm8

vpsrl[w/d/q] ymm1, ymm2, ymm3/m256 lub imm8 (AVX2)

Przesuwalogiczne słowa/podwójne słowa /poczwórne słowa w prawo logicznie (cale) z rejestru xmm2/ymm2 o wartość wskazaną przez rejestr xmm3/ymm3 lub m256. Podczas przesunięcia starsze bity są zerowane. Jeśli wartość licznika jest większa niż 15 dla słów, 31 dla podwójnych słów, 63 dla poczwórnego słowa, wówczas wszystkie bity są zerowane.

cel[i] = źródło1[i] >> źródło2[o] lub imm8

xmm1[i] = xmm2[i] >> xmm3/m128[o] lub imm8

ymmi[i] = ymm2[i] >> ymm3/m256[o] lub imm8

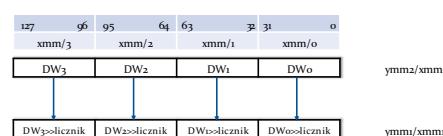
Bit od 128 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

167

Instrukcja przesunięcia w prawo VPSRLD



Licznik może być zapisany w rejestrze **xmm3/m128** (ymm3/m256)
albo w bajcie sterującym **imm8**

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

168

Instrukcja przesunięcia logicznego w prawo VPSRLDQ

vpsrldq xmm1, xmm2, imm8

vpsrldq ymm1, ymm2, imm8 (AVX2)

Przesuwa logicznie w prawo podwójne poczwórne słowo z rejestru xmm2/ymm2 o liczbe bajtów określonej przez imm8. Podczas przesunięcia starsze bity są zerowane.

cel[i] = źródło1[i] >> imm8

xmm1[i] = xmm2[i] >> imm8

ymmi[i] = ymm2[i] >> imm8

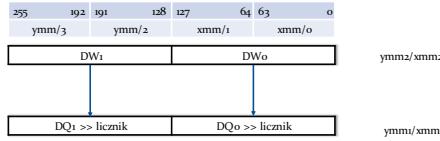
Bit od 128 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

169

Instrukcja dodawania VPSRDQ



Licznik jest określony w **imm8**.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

170

Instrukcja przesunięcia w prawo VPSRLV[D/Q]

vpsrlv[d/q] xmm1, xmm2, xmm3/m128 (AVX2)
vpsrlv[d/q] ymm1, ymm2, ymm3/m256 (AVX2)

Przesuwa **logicznie w prawo** bity podwójne słowa/poczwórne słowa z rejestru xmm2/ymm2 o **liczbę bitów** wskazaną przez odpowiednie elementy rejestru xmm3/ymm3 lub m128/m256. Podczas przesunięcia starsze bity są zerowane. Jeśli wartość licznika jest większa niż 31 dla podwójnych słów, 63 dla poczwórnego słów, wówczas wszystkie bity są zerowane.

cel[i] = źródło1[i] >> źródło2[i]
xmm1[i] = xmm2[i] >> xmm3/m128[i]
ymm1[i] = ymm2[i] >> ymm3/m256[i]

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

171

Instrukcja przesunięcia arytmetycznego w prawo VPSRA[W/D]

vpsra[w/d] xmm1, xmm2, xmm3/m128 lub imm8
vpsra[w/d] ymm1, ymm2, ymm3/m256 lub imm8 (AVX2)

Przesuwa **arytmetycznie w prawo z powielaniem bitu znaku** słowa/podwójne słowa z rejestru xmm2/ymm2 (cale) określona przez wartość zapisaną w rejestrze xmm3/ymm3 lub m128/m256 lub przez bajt sterujący imm8. **Starsze bity są ustawiane na bit znaku**. Jeśli wartość licznika jest większa niż 15 dla słów, 31 dla podwójnych słów, 63 dla poczwórnego słów, wówczas wszystkie bity są ustawiane na bit znaku.

cel[i] = źródło1[i] >> źródło2[0] lub źródło2[1]
xmm1[i] = xmm2[i] >> xmm3/m128[0] lub imm8
ymm1[i] = ymm2[i] >> ymm3/m256[0] lub imm8

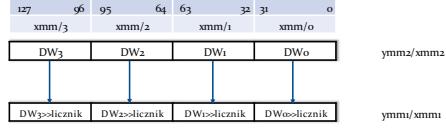
Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

172

Instrukcja przesunięcia arytmetycznego w prawo VPSRAD



Licznik może być zapisany w rejestrze **xmm3/m128** (ymm3/m256)
albo w bajcie sterującym **imm8**

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

173

Instrukcja przesunięcia arytmetycznego w prawo VPSRAVD

vpsravd xmm1, xmm2, xmm3/m128 (AVX2)
vpsravd ymm1, ymm2, ymm3/m256 (AVX2)

Przesuwa **arytmetycznie w prawo z powielaniem bitu znaku** słowa/podwójne słowa z rejestru xmm2/ymm2 o **liczbę bitów** określona przez wartość zapisaną w rejestrze xmm3/ymm3 lub m128/m256. **Starsze bity są ustawiane na bit znaku**. Jeśli wartość licznika jest większa niż 15 dla słów, 31 dla podwójnych słów, wówczas wszystkie bity są ustawiane na bit znaku.

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

174

Instrukcja przesunięcia logicznego w lewo VPSLL[W/D/Q]

vpsll[w/d/q] xmm1, xmm2, xmm3/m128 lub imm8
vpsll[w/d/q] ymm1, ymm2, ymm3/m256 lub imm8 (AVX2)

Przesuwa **logicznie w lewo** słowo/podwójne słowo/poczwórne słowo (w całości) z rejestru xmm2/ymm2 o **wartość określona przez rejestr xmm3/ymm3 lub m128/m256 lub przez bajt sterujący imm8**. **Młodsze bity są zerowane**. Jeśli wartość licznika jest większa niż 31 dla słów, większa niż 63 dla podwójnych słów, większa niż 127 dla poczwórnego słów, wówczas bity danego elementu są zerowane.

cel[i] = źródło1[i] << źródło2[0] lub imm8
xmm1[i] = xmm2[i] << xmm3/m128[0] lub imm8
ymm1[i] = ymm2[i] << ymm3/m256[0] lub imm8

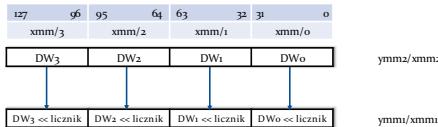
Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

175

Instrukcja przesunięcia logicznego w lewo VPSLLD



Licznik może być zapisany w rejestrze **xmm3/mu28** (yymm3/m256)
albo w bajcie sterującym **imm8**

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

176

Instrukcja przesunięcia logicznego w lewo VPSLLV[D/Q]

vpsllv[d/q] xmm1, xmm2, xmm3/m128
vpsllv[d/q] ymm1, ymm2, ymm3/m256

Przesuwa logicznie w lewo podwójne słwo/poczwórnego słwo z rejestru ymm3/ymm2 o liczbę bitów określona przez rejestr ymm3/ymm2 lub mu28/m256. **Młodsze bity są zerowane.** Jeśli wartość licznika jest większa niż 3 dla podwójnych słów, większa niż 63 dla poczwórkowych słów, wówczas wszystkie bity danego elementu są zerowane.

cel[i] = źródło1[i] << źródło2[i]
xmm1[i] = xmm2[i] << xmm3/m128[i]
ymm1[i] = ymm2[i] << ymm3/m256[i]

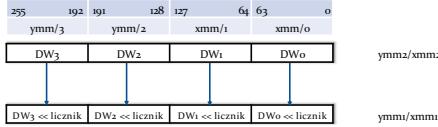
Bit od 128 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

177

Instrukcja przesunięcia arytmetycznego w lewo VPSLLVQ



Liczniki są określone w **xmm3/ymm3** lub **mu28/m256**

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

178

Operacje logiczne/inne AVX liczby całkowite

- **Instrukcje logiczne:**
VPAND, VPANDN, VPOR, VPXOR
- **Instrukcje zerowania:**
VZEROALL, VZEROUPPER
- **Instrukcje dodatkowe:**
VLDMXCSR / VSTMXCSR
- **Instrukcja wyrównywania:**
VPALIGNR

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

179

Instrukcje logiczne

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

180

Instrukcje logiczne koniunkcja VPAND / VPANDN

vpand xmm1, xmm2, xmm3/m128
vpand ymm1, ymm2, ymm3/m256 (AVX2)

vpandn xmm1, xmm2, xmm3/m128 (AVX)
vpandn ymm1, ymm2, ymm3/m256 (AVX2)

Oblicza iloczyn logiczny bit po bicie dla wszystkich bitów rejestrów xmm2/ymm2 oraz xmm3/ymm3 lub mu28/m256 oraz negacji xmm3/ymm3 lub mu28/m256, wynik zapisuje w xmm1/ymm1.

cel[i] = źródło1[i] and źródło2[i]

Bit od 128 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

Bit od 128 do MSB są zerowane.

181

Instrukcje logiczne alternatywa

VPOR / VPXOR

vpor xmm1, xmm2, xmm3/m128
vpor ymm1, ymm2, ymm3/m256 (AVX2)

Oblicza sumę logiczną bit po bicie dla wszystkich bitów rejestrów xmm2/ymm2, oraz xmm3/ymm3 lub m128/m256, wynik zapisuje w xmm1/ymm1.

cel[i] = źródło1[i] or źródło2[i]

vpxor xmm1, xmm2, xmm3/m128 (AVX)
vpxor ymm1, ymm2, ymm3/m256 (AVX2)

Oblicza alternatywę wykluczającą bit po bicie dla wszystkich bitów rejestrów xmm2/ymm2, oraz xmm3/ymm3 lub m128/m256, wynik zapisuje w xmm1/ymm1.

cel[i] = źródło1[i] xor źródło2[i]

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

i82

Instrukcje dodatkowe

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

i83

Instrukcje dodatkowe

VZEROALL / VZEROUNDER

• vzeroall (AVX)

Zeruje wszystkie rejesty ymm/o - ymm/15

• vzeroupper (AVX)

Zeruje bity od 128, do ostatniego rejestrów ymm/o - ymm/15 / zmm/o - zmm/15.

W trybie 32-bitowym zeruje tylko pierwsze 8 rejestrów.

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

i84

Instrukcje dodatkowe

VLDMXCSR / VSTMXCSR

• vldmxcsr m32 (AVX)

Ładuje zawartość operandy źródłowego m32 do rejestru kontrolnego i statusu (MXCSR Control and Status Register), jest to ładowanie ustawień.

• vstmxcsr m32 (AVX)

Przesyła zawartość rejestru kontrolnego i statusu (MXCSR Control and Status Register) do operandy źródłowego m32, jest to kopowanie ustawień.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

i85

Instrukcja łączenia dodatkowe

VPALIGNR

vpalignr xmm1, xmm2, xmm3/m28, imm8 (AVX)

vpalignr ymm1, ymm2, ymm3/m256, imm8 (AVX2)

Łączy (konkatenacja) rejesty źródła xmm1/ymm1 z xmm2/ymm2 lub m128/m256, na podstawie bajtu sterującego przesuna 128-bitową częścią o imm8*8 i zapisuje 128-bitową częścią do rejestru celu xmm1/ymm1.

cel = (źródło1+źródło2) >> źródło2

xmm1 = (xmm2+xmm3/m128) >> imm8[7:0]*8

hi ymm1 = (hi ymm2 + hi ymm3/m256) >> imm8[7:0]*8

lo ymm1 = (lo ymm2 + lo ymm3/m256) >> imm8[7:0]*8

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

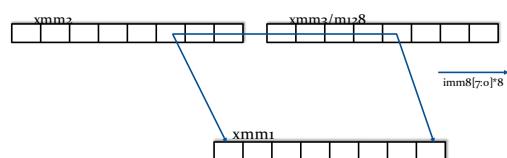
Programowanie niskopoziomowe

i86

Instrukcja łączenia dodatkowe

VPALIGNR

Sposób łączenia rejestrów xmm



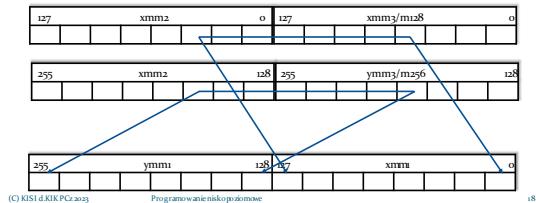
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomowe

i87

Instrukcja łączenia dodatkowe VPALIGNR

Sposób łączenia rejestrów xmm



(C) KISI d.KIK PCz 2023 Programowanie niskopoziomowe 188

Instrukcje szyfrujące

Operacje szyfrujące AVX

- Instrukcje szyfrujące
 - VAESENC, VAESENCLAST
 - VAESDEC, VAESDECLAST
 - VAESIMC, VAESKEYGENASSIST

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomowe 190

Instrukcje szyfrujące

Algorytm AES (Advanced Encryption Standard) (1/2)

Instrukcje typu AVX udostępniają szyfrowanie danych z zastosowaniem algorytmu AES jedynie w wersji 128 bitowej.

Algorytm AES występuje w trzech wariantach

- AES-128 używa klucza 128 bitowego (możliwe 10 rund szyfrowania)
 - AES-192 używa klucza 192 bitowego (możliwe 12 rund szyfrowania)
 - AES-256 używa klucza 256 bitowego (możliwe 14 rund szyfrowania)
- to jednak podstawową jednostką do zaszyfrowania/odszyfrowania jest blok danych 128 bitowy wykorzystując odpowiednio klucz 128, 192, 256 bitowy

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomowe 191

Instrukcje szyfrujące

Algorytm AES (Advanced Encryption Standard) (2/2)

AES jest algorytmem symetrycznym to znaczy że ten sam klucz jest stosowany do zaszyfrowania i odszyfrowania danych.

Dane podlegają trzem rodzajom przekształceń:

- podstawianie (substitution),
 - transponowanie (transposition)
 - mieszanie (mixing)
- po czym następuje zestawienie przekształconych danych (alternatywa wykluczająca) z kluczem.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomowe 192

Instrukcja szyfrująca AES (Advanced Encryption Standard)

VAESENC

`vaeenc xmm1, xmm2, xmm3/m128`

Szyfruje jedną rundę (jednokrotne) dane (blok danych) całego rejestru xmm2 (128 bitów) z wykorzystaniem 128 bitowego klucza symetrycznego (ten sam klucz do szyfrowania i odszyfrowania) zapisanego w rejestrze xmm3/m128, zaszyfrowane dane zapisuje w rejestrze xmm1.

```
[cel = aes(źródło1) xor źródło2 (key)]
for (unsigned int i = 0; i < [1-9]; i++)
{
    xmm1 = aes(xmm2) xor xmm3/m128
}
```

Bit od 127/256 do MSB nie są modyfikowane.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomowe 193

Instrukcja szyfrująca AES (Advanced Encryption Standard)

VAESENCLAST

```
vaesenclast xmm1, xmm2, xmm3/m128
```

Szyfruje jedną ale ostatnią rundę dane (blok danych) całego rejestru xmm2 (128 bitów) z wykorzystaniem 128 bitowego klucza zapisanego w rejestrze xmm3/m128, zaszyfrowane dane zapisuje w rejestrze xmm1.

```
cel = aes(źródło1) xor źródło2 (key)
xmm1 = aes(xmm2) xor xmm3/m128
```

Bity od 128/256 do MSB nie są modyfikowane.
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomowe 194

Instrukcja szyfrująca AES (Advanced Encryption Standard)

VAESDEC

```
vaesdec xmm1, xmm2, xmm3/m128
```

Odszyfruje jedną rundę (jednokrotne) blok danych całego rejestru xmm2 (128 bitów) z wykorzystaniem 128 bitowego klucza wcześniej użytego do zaszyfrowania zapisanego w rejestrze xmm3/m128, odszyfrowane dane zapisuje w rejestrze xmm1.

```
for (unsigned int i = 0; i < [1-9]; i++)
{
    [cel = aes(źródło1) xor źródło2 (key)]
    xmm1 = aes(xmm2) xor xmm3/m128
}
```

Bity od 128/256 do MSB nie są modyfikowane.
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomowe 195

Instrukcja szyfrująca AES (Advanced Encryption Standard)

VAESDECLAST

```
vaesdeclast xmm1, xmm2, xmm3/m128
```

Odszyfruje jedną ale ostatnią rundę dane (blok danych) całego rejestru xmm2 (128 bitów) z wykorzystaniem 128 bitowego klucza zapisanego w rejestrze xmm3/m128, odszyfrowane dane zapisuje w rejestrze xmm1.

```
cel = aes(źródło1) xor źródło2 (key)
xmm1 = aes(xmm2) xor xmm3/m128
```

Bity od 128/256 do MSB nie są modyfikowane.
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomowe 196

Instrukcja szyfrująca AES (Advanced Encryption Standard)

VAESIMC

```
vaesimc xmm1, xmm2/m128
```

Dokonuje **przekształcenia** 128 bitowego **klucza** zapisanego w xmm2/m128 poprzez odwróconą funkcję mieszaną kolumn InvMixColumns(), wynik zapisuje w xmm1.

Funkcja InvMixColumns() jest odwrotnością funkcji MixColumns().

```
cel = InvMixColumn(źródło-key)
xmm1 = InvMixColumn(xmm2/m128)
```

Bity od 128/256 do MSB nie są modyfikowane.
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomowe 197

Instrukcja szyfrująca AES (Advanced Encryption Standard)

VAESKEYGENASSIST

```
vaeskeygenassist xmm1, xmm2/m128 , imm8
```

Arysuje w rozszerzeniu **klucza**, poprzez obliczanie kroków w kierunku wygenerowania nowego klucza do zaszyfrowania, używając RoundConstant (pełni funkcję klucza klucza) ma sposób zdefiniowany w bajcie sterującym imm8, wynik zapisuje w rejestrze celu xmm1.

```
cel = szyfrowanie(źródło-key), źródło2
xmm1 = szyfrowanie(xmm2/m128), imm8
```

Bity od 128/256 do MSB nie są modyfikowane.
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomowe 198