

### Rejestry AVX ymm oraz zmm

- Dla rozkazów AVX dedykowano rejestry:
- 16 rejestrów 256 bitowych dla AVX oraz AVX2  
**ymm/15 – ymm/o**
  - 32 rejestry 512 bitowe dla AVX-512  
**zmm/31 – zmm/o**
  - Część instrukcji (większość) operuje na młodszej części rejestrów  
**xmm/[15/31] – xmm/o**



### Rejestry XMM

Przykład: VADDPD  $xmm1 = xmm2 + xmm3/m128$

zmm/7	zmm/6	zmm/5	zmm/4	zmm/3	zmm/2	zmm/1	zmm/0
				ymm/3	ymm/2	ymm/1	ymm/0
				xmm/3	xmm/2	xmm/1	xmm/0
511		255		128	127	64	63
					xmm2/1	xmm2/0	
					+	+	
					xmm3/1 lub m28/1	xmm3/0 lub m28/0	
					=	=	
					xmm1/1	xmm1/0	

### Rejestry YMM

Przykład: VADDPD  $ymm1 = ymm2 + ymm3/m256$

zmm/7	zmm/6	zmm/5	zmm/4	zmm/3	zmm/2	zmm/1	zmm/0
				ymm/3	ymm/2	ymm/1	ymm/0
				xmm/3	xmm/2	xmm/1	xmm/0
511		256	255		128	127	64
					ymm2/1	ymm2/0	
					+	+	
					ymm3/3 lub m256/3	ymm3/2 lub m256/2	xmm3/1 lub m28/1
					=	=	=
					ymm1/3	ymm1/2	xmm1/1

### Rejestry XMM

Przykład: VADDPD  $xmm1 = xmm2 + xmm3/m512$

zmm/7	zmm/6	zmm/5	zmm/4	zmm/3	zmm/2	zmm/1	zmm/0
				ymm/3	ymm/2	ymm/1	ymm/0
				xmm/3	xmm/2	xmm/1	xmm/0
511		256	255		128	127	64
					xmm2/1	xmm2/0	
					+	+	
					xmm3/3 lub m512/3	xmm3/2 lub m512/2	xmm3/1 lub m28/1
					=	=	=
					xmm1/3	xmm1/2	xmm1/1



### 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

### Typy danych dla AVX

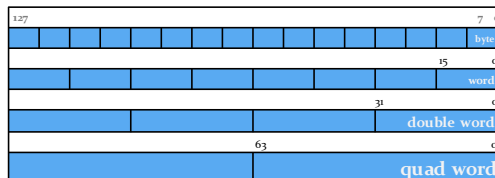
- Liczby zmiennie-przecinkowe (packed i scalar)

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

- Specyfikatory zmiany typu:  
xmmword ptr [adres]  
ymmword ptr [adres]

### Typy danych AVX liczby całkowite

Przykład na rejestrach 128 bitowych **xmm**



### Typy danych AVX

Liczby zmiennie-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 zmiennie-przecinkowych.

## Instrukcje AVX

### Instrukcje typu AVX

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

- nie zawsze działamy na wektorach.
- pominięcie koprocesora.
- gdź liczba danych jest niepodzielna przez liczbę elementów wektora.

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

## 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 **znaczne spowolnienie działania programu/podprogramu**.

## 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/components/ipp.html>

Ekwiwalent np. dla **VADDPD**

```
__m128d  _mm128d_add_pd (__m128d a, __m128d b);
__m256d  _mm256d_add_pd (__m256d a, __m256d b);
__m512d  _mm512d_add_pd (__m512d a, __m512d b);
```

## Systematyka instrukcji AVX

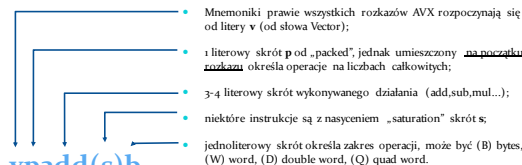
- Dokumentacja firmy Intel wyróżnia 12 kategorii instrukcji AVX.
- Na potrzeby niniejszego wykładu zastosowano podział na instrukcje AVX dotyczące liczb całkowitych, liczb zmiennie-przecinkowych oraz oddzielną grupę FMA, która w AVX operuje wyłącznie na liczbach zmiennie-przecinkowych. W AVX dostępna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

## 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 zmiennie-przecinkowych, w tych ostatnich sytuują się instrukcje FMA.

## Budowa rozkazu AVX

## Budowa rozkazu AVX liczb całkowite



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

### Budowa rozkazu AVX liczbymienno-przecinkowe

- 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 precyzji: double lub single.

**vaddpd**

Rozkaz **vaddpd** wykonuje dodawanie (add) wektorowo/równoległe (p) liczb zmienno-przecinkowych podwójnej precyzji (d).

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

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, 213, 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 precyzji „double” lub „single”;

**vfmadd132pd**

Rozkaz **vfmadd132pd** mnoży (m) i dodaje (add) równoległe/packed (p) liczby zmienno-przecinkowe podwójnej precyzji (d), kolejność operacji arytmetycznych jest oznaczona przez trzy cyfry 132, czyli wg przykładu mnoży 1. i 3. argument, do iloczynu dodaje 2. argument.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

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. 0e0h

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

27

### Instrukcje AVX

dla liczb całkowitych

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

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 niskopoziomse

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
- VPMOVMASKB, VMASKMOVDQU, VPMASKMOV[D/Q]

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

30

### Operacje przestania AVX

- Instrukcje kompresji/rozpakowania:** VPack[S/U]SWB, VPack[S/U]SDW, VPUNPCKHBW, VPUNPCKHWD, VPUNPCKHDQ, VPUNPCKHQDQ, VPUNPCKLBW, VPUNPCKLWD, VPUNPCKLDQ, VPUNPCKLQDQ

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomosc

3

### Operacje przestania AVX

- Instrukcje przetasowania:** VPSHUFb, VPSHUFD, VPSHUFHW, VPSHUFLW
- Instrukcje permutacji:** VPERMD, VPERMQ
- Instrukcje mieszajace:** VPBLENDB, VPBLENdW, VPBLENDD
- Instrukcje rozgłaszania:** VPBROADCASTb, VPBROADCASTW, VPBROADCASTD, VPBROADCASTQ
- Instrukcje zbierania:** VPGATHERDD, VPGATHERQD, VPGATHERDQ, VPGATHERQQ

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomosc

3

### Instrukcja przestania

#### 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
```

VMOVD / VMOVQ przesyła podwójne lub początkowe 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 niskopoziomosc

33

### Instrukcja przestania 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
```

Przesła całą zawartość (128 lub 256 bitów) źródła do celu, jeśli celem 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 przesyłać dane do/z niewyrównanych lokalizacji pamięci należy w instrukcji zamiast (A) użyć (U) (ang. unaligne).

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

Był od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomosc

34

### Instrukcja przestania z wyrównaniem

#### VMOVNTDQ[A]

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

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

Przesła całą zawartość z/do pamięci m128/m256 do/z rejestru xmm1/ymm1. Dla instrukcji w wersji (A) pamięć musi być wyrównana (ang. aligne) do granicy 16 (m128) lub 32 (256) 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).

Był od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomosc

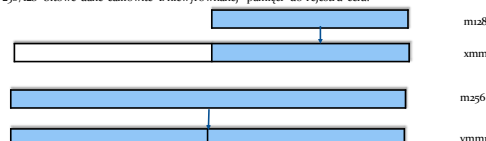
35

### Instrukcja ładowanie danych z pamięci

#### VLDDQU

```
vlddqu xmm1, m128
vlddqu ymm1, m256
```

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



Był od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomosc

36

### Instrukcja przesłania

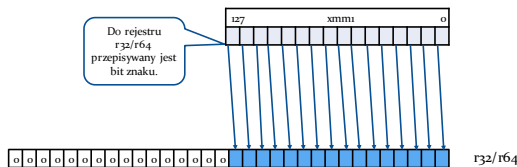
#### VPMOVMASKB

vpmovmaskb reg, xmm1     rejestr ← xmm1  
 vmovmaskb reg, ymm1     rejestr ← ymm1 (AVX2)

Przesyła najstarsze bity (bity znaku) **każdego bajtu** rejestru xmm1/ymm1 po kolei do rejestru r32/r64, dla xmm1 przenosi **16 bitów** do r32, dla ymm1 przenosi **32 bity** do r64, pozostałe starsze bity są zerowane.

### Instrukcja przesłania

#### VPMOVMASKB



## Instrukcje przesłania

z konwersją

### Instrukcje przesłania z konwersją

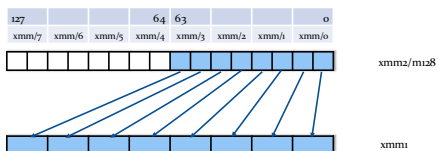
#### VPMOV[S/Z]XB[W/D/Q], VPMOV[S/Z]XW[D/Q] VPMOV[S/Z]XDQ

vpmov[s/z]xbw xmm1, xmm2/m64     vpmov[s/z]xwd xmm1, xmm2/m64  
 vpmov[s/z]xbd xmm1, xmm2/m32     vpmov[s/z]xwq xmm1, xmm2/m32  
 vpmov[s/z]xbq xmm1, xmm2/m16     vpmov[s/z]xdq xmm1, xmm2/m64

Instrukcja zamienia (konwertuje) ze znakiem / bez znaku:  
**hajt** na słowa/podwójne słowa/poczwórne słowa, **slowa** na podwójne słowa/poczwórne słowa,  
**podwójne słowa** na poczwórne słowa przepisując odpowiednio wartości z młodszej części rejestru  
 xmm2/(m64|m32|m16) **do rejestru celu xmm1**.

### Instrukcja przesłania z konwersją

#### VPMOVXSBW



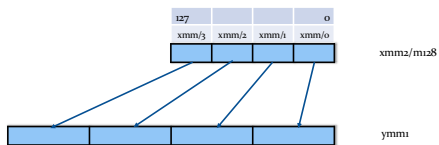
### Instrukcje przesłania z konwersją

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

vpmov[s/z]xbw ymm1, xmm2/m128     vpmov[s/z]xwd ymm1, xmm2/m128  
 vpmov[s/z]xbd ymm1, xmm2/m64     vpmov[s/z]xwq ymm1, xmm2/m64  
 vpmov[s/z]xbq ymm1, xmm2/m32     vpmov[s/z]xdq ymm1, xmm2/m128

Instrukcja **zamienia** (konwertuje) ze znakiem / bez znaku:  
**hajt** na słowa/podwójne słowa/poczwórne słowa, **slowa** na podwójne słowa/poczwórne słowa,  
**podwójne słowa** na poczwórne słowa przepisując odpowiednio wartości z młodszej części z rejestru  
 ymm2/(m128|m64|m32) **do rejestru celu ymm1**, a młodszą część rejestru.

### Instrukcja przesłania z konwersją VPMOVSXDQ



### Instrukcje przesłania - przykład:

```
vmovdqu ymm1,ymmword ptr[rax] ; rdi = int * tab1]
vmovdqu ymm1,ymmword ptr[rax+4*rsi] ; rdi = int * tab1[n]; rax = n

vmovdqu ymm2, ymmword ptr [rsi] ; rsi = int * tab2]
vmovdqu ymm2, ymmword ptr [rsi+4*rcx] ; rsi = int * tab2[m]; rcx = m
```

Dla typu tablicowego ładowanie całego rejestru ymm1/ymm2 z adresu pierwszego (0) elementu tablicy oraz od kolejnych elementów (wielokrotność 4) o wielkości podwójnego słowa.

```
vmovdqa xmm5, xmmword ptr [ebx] ; ebx = unsigned short * a
vpmovzxd ymm5, xmm5 ; konwersja z 16 do 32 bitów
```

Ponieważ obliczenia na 16 bitach mogłyby doprowadzić do przepięnienia (overflow), typ unsigned short konwertujemy na unsigned int 32 bity zachowując przy tym ilość elementów wektora równą 8.

## Instrukcje przesłania warunkowego

### Instrukcja przesłania warunkowego VMSKMOVDQU

vmskmovdqu xmm1, xmm2

Celem jest obszar 16 bajtów pamięci adresowany przez DS / 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

### 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 odpowiadających wartości z rejestru maski (drugi operand) xmm2/ymm2 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
```

## Instrukcje kompresji



### Instrukcja kompresji

#### VPACK[S/U]SWB

`vpack[s/u]swb xmm1, xmm2, xmm3/m128`  
`vpack[s/u]swb ymm1, ymm2, ymm3/m256 (AVX2)`

Konwertuje słowa **ze znakiem** na bajty **ze znakiem (S) / bez znaku(U)**, z rejestru `xmm2 / ymm2` wpisuje do **młodszych** części rejestru `xmm1`, a z `xmm3/m128 / ymm3/ m256` wpisuje do **starszych** części rejestru `xmm1/ ymm1`. Starsza część rejestru `ymm1` jest wypełniana danymi ze starszych części rejestrów `hi ymm2` i `hi ymm3/m256`. Wynik jest zapisywany **ze znakiem(S) / bez znaku(U)** oraz z nasyceniem.

Byte od 128/192 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023

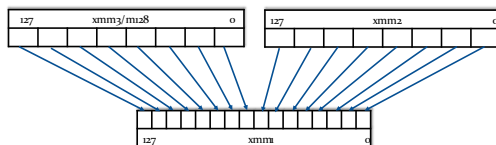
Programowanie niskopoziomosc

49

### Instrukcja kompresji

#### VPACK[S/U]SWB

`vpack[s/u]swb xmm1, xmm2, xmm3/m128`



(C) KISI d.KIK PCz 2023

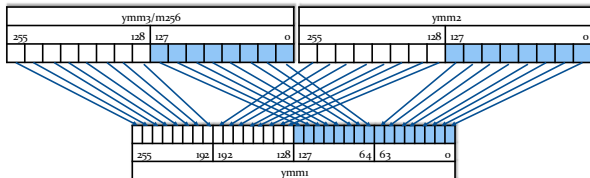
Programowanie niskopoziomosc

90

### Instrukcja kompresji

#### VPACK[S/U]SDW

`vpack[s/u]sdw ymm1, ymm2, ymm3/m256 (AVX2)`



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomosc

9

### Instrukcja kompresji

#### VPACK[S/U]SDW

`vpack[s/u]sdw xmm1, xmm2, xmm3/m128`  
`vpack[s/u]sdw ymm1, ymm2, ymm3/m256 (AVX2)`

Konwertuje podwójne słowa **ze znakiem** na słowa **ze znakiem(S)/bez znaku(U)**, z rejestru `xmm2/ lo ymm2` wpisuje do **młodszych** części rejestru `xmm1/ lo ymm1`, z `xmm3/m128 / lo ymm3/ lo m256` wpisuje do **starszych** części rejestru `edu xmm1/lo ymm1`. Starsza część rejestru `ymm1`, jest wypełniana danymi ze starszych części rejestrów `hi ymm2` oraz `hi ymm3/m256`. Wynik jest zapisywany **ze znakiem(S) / bez znaku(U)** oraz z nasyceniem.

Byte od 128/192 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023

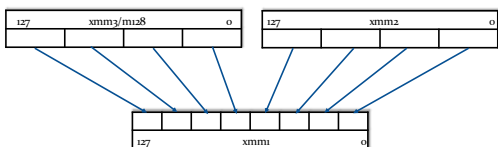
Programowanie niskopoziomosc

51

### Instrukcja kompresji

#### VPACK[S/U]SDW

`vpack[s/u]sdw xmm1, xmm2, xmm3/m128`



(C) KISI d.KIK PCz 2023

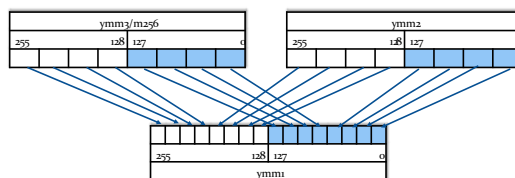
Programowanie niskopoziomosc

53

### Instrukcja kompresji

#### VPACK[S/U]SDW

`vpack[s/u]sdw ymm1, ymm2, ymm3/m256 (AVX2)`



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomosc

54

### Instrukcja kompresji

**VPUNPCKLBW** / VPUNPCKLWD / VPUNPCKLDQ / VPUNPCKLDQD

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 przepлетem do rejestru celu ymm1/xmm1.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 35

### Instrukcja kompresji

**VPUNPCKHBW** / VPUNPCKHWD / VPUNPCKHDQ / VPUNPCKHDQD

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 przepлетem do rejestru celu ymm1/xmm1.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 36

### Instrukcja kompresji

**VPUNPCKLWD** / VPUNPCKLDQ / VPUNPCKLDQD

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 przepлетem do rejestru celu ymm1/xmm1.

Bajty od 128/126 do MSB są zerowane.  
 (C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 37

### Instrukcja kompresji

**VPUNPCKHWD** / VPUNPCKHDQ / VPUNPCKHDQD

vpunpckhwd 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 przepлетem do rejestru celu ymm1/xmm1.

Bajty od 128/126 do MSB są zerowane.  
 (C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 38

### Instrukcja kompresji

**VPUNPCKLDQ** / VPUNPCKLDQD

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

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

Bajty od 128/126 do MSB są zerowane.  
 (C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 39

### Instrukcja kompresji

**VPUNPCKHDQ** / VPUNPCKHDQD

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

Starsze podwójne słowa ze 128 bitowych części rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepлетem do rejestru celu ymm1/xmm1.

Bajty od 128/126 do MSB są zerowane.  
 (C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 40

### Instrukcja kompresji

VPUNPCKLBW / VPUNPCKLWD / VPUNPCKLDQ / VPUNPCKLQDQ

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

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

Bity od 128 (gł) do MSB są zerowane.  
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 64

### Instrukcja kompresji

VPUNPCKHBW / VPUNPCKHWD / VPUNPCKHDQ / VPUNPCKHQDQ

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

Starsze początkowe słowa ze 128 bitowych części rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256 (m128) zapisuje z przplotem do rejestru celu ymm1/xmm1.

Bity od 128 (gł) do MSB są zerowane.  
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 64

## Instrukcje wyłuskiwania / wstawiania

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 65

### Instrukcja wyłuskiwania

VEEXTRACTI128 / VINSERTI128

vextracti128 xmm1/m128, ymm2, imm8 (AVX2)

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

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 64

### Instrukcja wstawiania

VEEXTRACTI128 / 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[0] = 0 przepisuje xmm3/m128 na młodszą część, gdy imm8[0] = 1 na starszą część celu ymm1.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 65

## Instrukcje tasowania

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 66

### Instrukcja tasowania

#### VPSHUFB

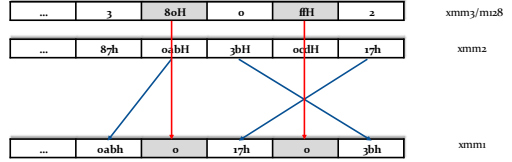
vpshufb xmm1, xmm2, xmm3/m128  
vpshufb ymm1, ymm2, ymm3/m256 (AVX2)

Tasuje bajty z xmm2/ymm2, w zależności od bitu znaku kolejnych bajtów rejestru xmm3/m128 / ymm3/m256. Jeśli bit znaku jest ustawiony odpowiedni bajt rejestru celu xmm1/ymm1 jest zerowany, jeśli bit znaku xmm3/ymm3 / 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 xmm2/ymm2 do właściwego xmm1/ymm1.

```
i = numer bajtu
if xmm3/m128[i][7] == 1 then xmm1[i] = 0
else { index[3..0] = xmm3/m128[i][3..0]
      xmm1[i] = xmm2[index]
    }
```

### Instrukcja tasowania

#### VPSHUFB



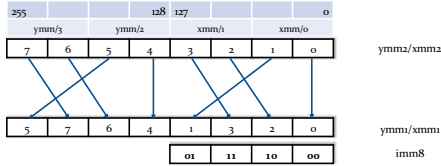
### Instrukcja tasowania

#### VPSHUFD

vpshufd xmm1, xmm2/m128, imm8

vpshufd ymm1, ymm2/m256, imm8 (AVX2)

Tasuje podwójne słowa z rejestru xmm2/ymm2/m128/m256 według dwubitowych wartości bajtu sterującego imm8 (argument kolejności), wynik zapisuje w xmm1/ymm1, tasowanie odbywa się w blokach 128 bitowych.



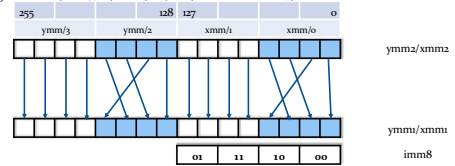
### Instrukcja tasowania

#### VPSHUFLW

vpshufw xmm1, xmm2/m128, imm8 (AVX)

vpshufw ymm1, ymm2/m256, imm8 (AVX2)

Tasuje wektory młodszych słów z rejestru xmm2/ymm2 według dwubitowych wartości bajtu sterującego imm8 (argument kolejności), wynik zapisuje odpowiednio w xmm1/ymm1.



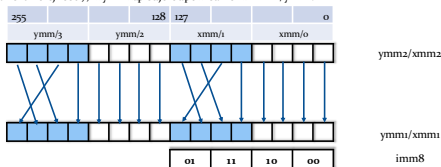
### Instrukcja tasowania

#### VPSHUFBW

vpshufbw xmm1, xmm2/m128, imm8 (AVX)

vpshufbw ymm1, ymm2/m256, imm8 (AVX2)

Tasuje wektory starszych słów z rejestru xmm2/ymm2 według dwubitowych wartości bajtu sterującego imm8 (argument kolejności), wynik zapisuje odpowiednio w xmm1/ymm1.

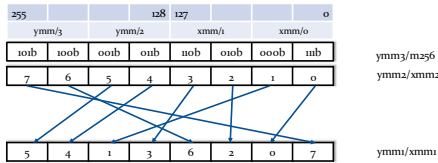


### Instrukcja tasowania

#### VPERMD

vpermd ymm1, ymm2, ymm3/m256 (AVX2)

Wykonuje permutacje wektorów podobnych słów z rejestru ymm3/m256 według porządku podanego w ymm2. najmłodsze 128-bitowe odpowiedniego podobnego słowa rejestru ymm2 wyznacza, z którego miejsca w ymm3/m256 zostanie skopiowane podobne słowo do miejsca pobrania „adresu” (ymm2). Wynik jest zapisywany do ymm1.

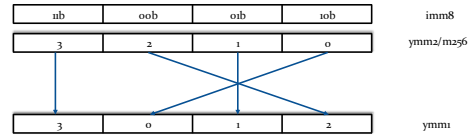


### Instrukcja permutacji

#### VPERMQ

vpemq ymm1, ymm2/m256, imm8 (AVX2)

Wykonuje permutacje wektorów poczwórnych słów z rejestru ymm2/m256 według porządku określonego w imm8. Kolejne dwubitowe pola imm8 określają, spod którego indeksu „adresu” zostaną skopiowane poczwórne słowa z ymm2/m256. Wynik jest zapisywany do ymm1.

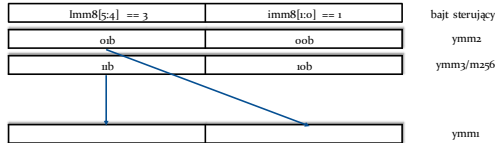


### Instrukcja permutacji

#### VPERM2I128

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

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



### 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]
    vmovdq ymm0, ymmword ptr [eax]
    vmovdq ymm1, ymmword ptr [ecx]
    vperm2i128 ymm2, ymm0, ymm1, 20h
    vperm2i128 ymm4, ymm0, ymm1, 31h
    vmovdq ymm0, ymmword ptr [esi]
    vmovdq ymmword ptr [eax], ymm0
    vmovdq ymmword ptr [esi], ymm1
    vmovdq ymmword ptr [ecx], ymm2
    vmovdq ymmword ptr [edx], ymm3
    pop esi;
}
```

## Instrukcje mieszające

### Instrukcja mieszająca

#### 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 <= 15 - lub <= 31 - indeks bajtu
if źródło[i][7] = 1 => cel[i] = źródło2[i]
else cel[i] = źródło1[i]

if xmm4[i][7] = 1 => xmm1[i] = xmm3/m256[i]
else xmm1[i] = xmm2[i]

if ymm4[i][7] = 1 => ymm1[i] = ymm3/m256[i]
else ymm1[i] = ymm2[i]
```

### Instrukcja mieszająca VPBLENDVB

127	64	63	0
1	1	1	0
0	0	1	0
1	1	0	1
1	0	1	1
0	1	1	0
1	1	0	1
1	1	0	0

xmm4  
bit znaku

xmm3/mmu28

xmm2

xmm1

(C) KISI d.KIK PCz 2023      Programowanie niskopoziomse      79

### Instrukcja mieszająca VPBLENDW

vpblendw xmm1, xmm2, xmm3/m128, imm8 (AVX)  
vpblendw ymm1, ymm2, ymm3/m256, imm8 (AVX2)

W oparciu o bajt kontrolny miesza wektory słów; wybiera elementy wektora z rejestru xmm3/ymm3 lub m28/m256 dla imm8[i] = 1, albo elementy wektora xmm2/ymm2 dla imm8[i] = 0. Dla indeksu 8-15 należy wziąć imm8[i-8]. Wynik zapisuje w xmm1/ymm1.

```

i < 0, 7> lub <0, 15> - indeks słowa
if imm8[i modulo 8] = 1 then cel[i] = źródło1[i]
else cel[i] = źródło2[i]

if imm8[i] = 1 then xmm1[i] = xmm3/mu28[i]
else xmm1[i] = xmm2[i]

if imm8[i modulo 8] = 1 then ymm1[i] = ymm3/m256[i]
else ymm1[i] = ymm2[i]
    
```

Bty od 128/256 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023      Programowanie niskopoziomse      80

### Instrukcja mieszająca VPBLENDW

255	128	127	0
1	0	1	0
0	1	1	0
1	1	0	1
1	0	1	1
0	1	1	0
1	1	0	1
1	1	0	0

imm8

xmm3/mmu28

xmm2

xmm1

(C) KISI d.KIK PCz 2023      Programowanie niskopoziomse      81

### Instrukcja mieszająca VPBLEND

vpblendd 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 ymm1.

```

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

if imm8[i] = 1 then ymm1[i] = ymm3/m256[i]
else ymm1[i] = ymm2[i]
    
```

Bty od 128/256 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023      Programowanie niskopoziomse      82

### Instrukcja mieszająca VPBLEND

255	128	127	0
ymm7/6	ymm5/4	xmm3/2	xmm1/0
1	0	0	0
0	0	1	1
1	1	1	1
0	1	1	0
1	1	0	1
1	1	0	0

imm8

ymm3/xmm3

ymm2/xmm2

ymm1/xmm1

(C) KISI d.KIK PCz 2023      Programowanie niskopoziomse      83

## Instrukcje rozgłaszające

(C) KISI d.KIK PCz 2023      Programowanie niskopoziomse      84

### Instrukcja rozgłaszania

#### VPBROADCAST[B/W/D/Q]

vpbroadcast[b/w/d/q] xmm1, xmm2/m8/m16/m32/m64 (AVX2)  
 vpbroadcast[b/w/d/q] ymm1, ymm2/m8/m16/m32/m64 (AVX2)

Rozgłasza tę samą wartość 8/16/32/64 bitową ze źródła xmm2/ymm2 lub pamięci m8/m16/m32/m64 do wszystkich elementów wektora rejestru celu xmm1/ymm1. Jeśli operand źródłowy jest rejestrem wartość rozgłaszana w najmłodszym elemencie wektora. Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 85

### Instrukcje zbierania

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 86

### Instrukcja zbierania

#### VPGATHER[D/Q][D/Q]

vpgatherdd xmm1, vm32x, xmm3  
 vpgatherqd xmm1, vm64x, xmm3

vpgatherdd ymm1, vm32y, ymm3  
 vpgatherqd ymm1, vm64y, ymm3

vpgatherdq xmm1, vm32x, xmm3  
 vpgatherqq xmm1, vm64x, xmm3

vpgatherdq ymm1, vm32y, ymm3  
 vpgatherqq ymm1, vm64y, ymm3

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 xmm1/ymm1 używając adresów w postaci podwójnych/poczwórnych słów zdefiniowanych w vm32{v/y}/vm64{v/y} używając jako indeksów podwójnych/poczwórnych słów zapisanych w xmm2/ymm2 do wskazanej lokalizacji pamięci skąd pobierane są wartości podwójnego/poczwórnego słowa.

Pobierane z pamięci wartości są zapisywane do rejestru celu xmm1/ymm1 tylko wówczas gdy najstarsze bity odpowiednich elementów wektora maski xmm3/ymm3 są równe 1.

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 87

### 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 (z 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 niskopoziomse 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 niskopoziomse 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/poczwórnych słów i zapisuje je do rejestru celu ymm1/xmm1 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 xmm1[i] ← [adres\_fizyczny(xmm2[i])]**

**if ymm3[i][63/31] then ymm1[i] ← [adres\_fizyczny(ymm2[i])]**

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 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/poczwórnych słów z rejestru **xmm2/ymm2** są dodawane równoległe odpowiednie wartości z rejestru **xmm3/ymm3** lub z pamięci **m128/m256**, wynik jest zapisywany w rejestrze **xmm1/ymm1**.

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

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

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

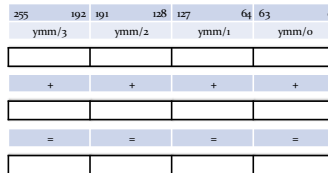
Bez od 128/192 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

97

### Instrukcja dodawania

#### VPADDQ



xmm2/xmm2  
  
ymm13/xmm13  
m256/m128  
  
ymm1/xmm1

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

98

### Instrukcja dodawania

#### VPADDS[B/W]

vpaddsb xmm1, xmm2, xmm3/m128

vpaddsb ymm1, ymm2, ymm3/m256 (AVX2)

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

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

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

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

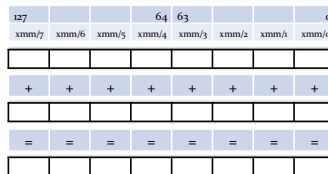
Bez od 128/192 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

99

### Instrukcja dodawania

#### VPADDSW



xmm2  
  
xmm3/m128  
  
xmm1

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

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 **nasyceciem** w rejestrze **xmm1/ymm1**.

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

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

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

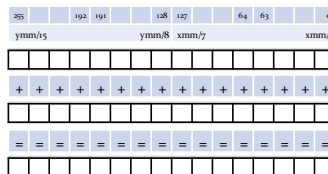
Bez od 128/192 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

101

### Instrukcja dodawania

#### VPADDUSW



xmm2  
  
xmm3/m128  
  
xmm1

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

102

### Instrukcja dodawania

VPHADDW / VPHADDD / VPHADDSW

**vphadd xmm1, xmm2, ymm3/m128**  
**vphadd ymm1, ymm2, ymm3/m256**

Horizontalne dodawanie sąsiednich słów/podwójnych słów i zapisywanie wyniku z przepięciem po 64 bity. Jako najmłodszą są zapisywane sumy z rejestru xmm2. Ostatnia w/w instrukcja jest dodawaniem słów z nasyceniem.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 103

### Instrukcja dodawania - przykład:

```
void vec_avx_add_int
(int* t1, int* t2, int* t3, int n)
__asm {
    push esi;
    push edi;
    mov ecx, n;
    shl ecx, 2;
    mov esi, t1; adres tablicy t1
    mov edx, t2; adres tablicy t2
    mov edi, t3; adres tablicy t3
    od.
    petla:
    sub ecx, 32; podwójne słowo
    vmovdq ymm0, ymmword ptr[esi + ecx];
    vmovdq ymm1, ymmword ptr[edx + ecx];
    vphadd ymm2, ymm1, ymm0;
    vmovdq ymmword ptr[edi + ecx], ymm2;
    jnz petla;
    pop edi;
    pop esi;
}
```

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 104

## Instrukcje odejmowania

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 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 równoległe odpowiednie wartości z rejestru **xmm3/ymm3** lub z pamięci **m128/m256**, wynik jest zapisywany w rejestrze **xmm1/ymm1**.

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

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

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

Bty od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 106

### Instrukcja odejmowania

VPSUBQ

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

ymm2/xmm2  
 ymm3/xmm3  
 m256/m128  
 ymm1/xmm1

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 107

### Instrukcja odejmowania

VPSUB[U]S[B/W]

vpsub[us][b/w] xmm1, xmm2, xmm3/m128  
 vpsub[us][b/w] ymm1, ymm2, ymm3/m256 (AVX2)

Od wartości ze znakiem/ bez znaku (U) 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.

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

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

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

Bty od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 108

### Instrukcja odejmowanie

#### VPSUBSW

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

xmm2

xmm3/m128

xmm1

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 109

### Instrukcja odejmowania

#### VPHSUBW / VPHSUBD / VPHSUBSW

vphsubd xmm1, xmm2, xmm3/m128  
vphsubd ymm1, ymm2, ymm3/m256

Horizontálne odejmowanie sąsiednich słów/podwójnych słów w ramach 128-bitowych części. Od młodszego elementu wektora jest odejmowany starszy, oraz jako najmłodsze są zapisywane różnice z rejestru xmm2. Ostatnia instrukcja jest odejmowaniem słów z nasyceniem

Był od 128/196 do MSB są zerowane.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 110

### Instrukcja sumowanie modułów różnic

#### VPSADBW

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

Od wartości wektorów bajtów rejestru xmm2/ymm2 są **odejmowane** odpowiednie wartości rejestru xmm3/ymm3 lub pamięci m128/m256, następnie obliczane są wartości absolutne i ich sumy po 8 elementów, wynik jest zapisywany w rejestrze xmm1/ymm1 dla zestawów 8-elementowych

Był od 128/196 do MSB są zerowane.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 111

### Instrukcja sumowanie modułów różnic

#### VPSADBW

255	192				191	128				127	64				63	0			
ymm/15					ymm/8	xmm/7									xmm/0				

ymm2

ymm3/m256

różnica

abs

ymm1

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 112

## Instrukcje mnożenia

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 113

### Instrukcja mnożenia

#### VPMULL[W/D]

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

Mnożenie wektorów słów/podwójnych słów ze znakiem z rejestru xmm2/ymm2 przez odpowiadające im wartości z xmm3/m128 / ymm3/m256, **iloczyny są podwójnymi/poczwórnymi słowami** jednak do rejestru celu xmm1/ymm1 są zapisywane **tylko młodsze słowa/podwójne słowa iloczynów**

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

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

$$\text{ymm1}[i] = \text{lo}(\text{ymm2}[i] * \text{ymm3}/\text{m256}[i])$$

Był od 128/196 do MSB są zerowane.

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 114

### Instrukcja mnożenia

#### VPMULLW

127				64	63			0
xmm/7	xmm/6	xmm/5	xmm/4	xmm/3	xmm/2	xmm/1	xmm/0	
*	*	*	*	*	*	*	*	*
=	=	=	=	=	=	=	=	=
3	2	1	0					
7	6	5	4					

xmm2  
xmm3/m128  
wynik mnożenia (podwójne słowa)  
xmm1

(C) KISI d.KIK PCz 2023 Programowanie nisko-poziomie 115

### Instrukcja mnożenia

#### VPMULLD

127	96	95	64	63	32	31	0
xmm/3	xmm/2	xmm/1	xmm/0				
*	*	*	*				
=	=	=	=				
1	0						
3	2						

ymm2/xmm2  
ymm3/xmm3  
m256/m128  
iloczyn (poczwórne słowa)  
ymm1/xmm1

(C) KISI d.KIK PCz 2023 Programowanie nisko-poziomie 116

### Instrukcja mnożenia

#### VPMUH[U]W

vpmulh[u]w xmm1, xmm2, xmm3/m128  
vpmulh[u]w ymm1, ymm2, ymm3/m256 (AVX2)

Mnożenie wektorów słów bez znaku/ze znakiem (U) z rejestru xmm2/ymm2 przez odpowiadające im wartości z xmm3/m28 / ymm3/m256, **iloczyny są podwójnymi słowami**, jednak do rejestru celu xmm1/ymm1 są zapisywane **tylko starsze słowa**, iloczynów.

$cel[i] = hi(\text{źródło1}[i] \times \text{źródło2}[i])$   
 $xmm1[i] = hi(xmm2[i] \times xmm3/m128[i])$   
 $ymm1[i] = hi(ymm2[i] \times ymm3/m256[i])$

Bity od 128/256 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023 Programowanie nisko-poziomie 118

### Instrukcja mnożenia

#### VPMULHW

127			64	63			0
xmm/7	xmm/6	xmm/5	xmm/4	xmm/3	xmm/2	xmm/1	xmm/0
*	*	*	*	*	*	*	*
=	=	=	=	=	=	=	=
3	2	1	0				
7	6	5	4				

xmm2  
xmm3/m128  
wynik mnożenia (podwójne słowa)  
xmm1

(C) KISI d.KIK PCz 2023 Programowanie nisko-poziomie 119

### Instrukcja mnożenia

#### VPMULHRWS

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

Mnoży wektory słów ze znakiem ze skalowaniem i zaokrągleniem, wartości z rejestru xmm2/ymm2 przez wartości z rejestru xmm3/m28 / 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.

$cel[i] = ((\text{źródło1}[i] * \text{źródło2}[i] \gg 14) + 1) \gg 1$   
 $xmm1[i] = ((xmm2[i] * xmm3/m128[i] \gg 14) + 1) \gg 1$   
 $ymm1[i] = ((ymm2[i] * ymm3/m256[i] \gg 14) + 1) \gg 1$

Bity od 128/256 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023 Programowanie nisko-poziomie 120

### Instrukcja mnożenia

#### VPMULHRWS

127			64	63			0
xmm/7	xmm/6	xmm/5	xmm/4	xmm/3	xmm/2	xmm/1	xmm/0
*	*	*	*	*	*	*	*
=	=	=	=	=	=	=	=
3	2	1	0				
7	6	5	4				

ymm2  
xmm3/m128  
(Iloczyn[i] >>14 + 1) >> 1  
xmm1

(C) KISI d.KIK PCz 2023 Programowanie nisko-poziomie 121

### Instrukcja mnożenia

#### VPMUL[U]DQ

vpmul[uldq] xmm1, xmm2, xmm3/m128

vpmul[uldq] ymm1, 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 xmm3/m128 / ymm3/m256, iloczyny są zapisywane w xmm1/ymm1 jako wektor począwszy od słów ze znakiem.

$$cel[i] = \text{źródło1}[2i] * \text{źródło2}[2i]$$

$$xmm1[i] = xmm2[2i] * xmm3/m128[2i]$$

$$ymm1[i] = ymm2[2i] * ymm3/m256[2i]$$

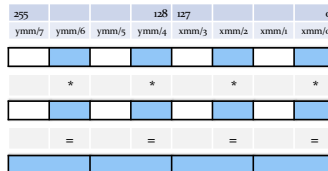
Źródło 128/196 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomose

122

### Instrukcja odejmowanie

#### VPMULDQ



ymm2

ymm3/m256

ymm1

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomose

123

### Instrukcja mnożenia

#### VPCLMULQDQ

vpclmulqdq xmm1, xmm2, xmm3/m128, imm8 (AVX)

Mnożenie począwszy od słowa z xmm2 przez począwszy od słowa z xmm3/m128, iloczyn jest zapisywany w xmm1. Bity imm8[0] i imm8[4] wybierają młodsze lub starsze (0 lub 1) począwszy od słowa z rejestrów xmm2 i xmm3/m128, które zostaną pomnożone.

if imm8[0] = 0 | 1 && imm8[4] = 0 | 1 => cel <- źródło1[ o | 1 ] \* źródło2[ o | 1 ]

if imm8[0] = 0 && imm8[4] = 0 => xmm1 <- xmm2[63:0] \* xmm3/m128[63:0]

if imm8[0] = 0 && imm8[4] = 1 => xmm1 <- xmm2[127:64] \* xmm3/m128[127:64]

if imm8[0] = 1 && imm8[4] = 0 => xmm1 <- xmm2[127:64] \* xmm3/m128[63:0]

if imm8[0] = 1 && imm8[4] = 1 => xmm1 <- xmm2[127:64] \* xmm3/m128[127:64]

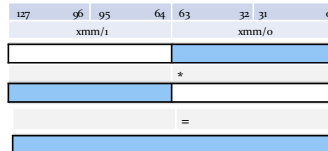
Źródło 128/196 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomose

124

### Instrukcja mnożenia

#### VPCLMULQDQ



ymm2/xmm2  
&& imm8[0] = 0

ymm3/xmm3 lub m256/m128  
&& imm8[4] = 1

ymm1/xmm1

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomose

125

## Instrukcje mnożenia z dodawaniem

Dla liczb zmienno-przecinkowych odpowiednikiem bardziej zaawansowanym są instrukcje FMA

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomose

126

### Instrukcja mnożenia i dodawania

#### VPMADDWD

vpmaddwd xmm1, xmm2, xmm3/m128

vpmaddwd ymm1, 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 xmm1/ymm1.

$$cel[i] = \text{źródło1}[2i] * \text{źródło2}[2i] + \text{źródło1}[2i+1] * \text{źródło2}[2i+1]$$

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

$$ymm1[i] = ymm2[2i] * ymm3/m128[2i] + ymm2[2i+1] * ymm3/m128[2i+1]$$

Źródło 128/196 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023

Programowanie niskopoziomose

127

### Instrukcja mnożenia VPMADDWD

127					64	63					0				
xmm1/7		xmm1/6		xmm1/5		xmm1/4		xmm1/3		xmm1/2		xmm1/1		xmm1/0	
xmm2															
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
xmm3/m128															
=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=
3		2		1		0									
7	6	5	4	3	2	1	0								
iloczynny															
sumuje sąsiednie dwa iloczynny															
xmm1															

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 128

### Instrukcja mnożenia i dodawania VPMADDUBSW

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

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

$$cel[i] = \text{źródło1}[2i] * \text{źródło2}[2i] + \text{źródło1}[2i+1] * \text{źródło2}[2i+1]$$

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

$$ymm1[i] = ymm2[2i] * ymm3/m128[2i] + ymm2[2i+1] * ymm3/m128[2i+1]$$

Bty od 128/196 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 129

### Instrukcja sumowanie modułów VPMADDUBSW

255					192	191					128	127					64	63					0
ymm1/5						ymm1/8		xmm1/7						ymm1/0						ymm1/0			
ymm2																							
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	
ymm3/m256																							
=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16								
iloczynny																							
ymm1																							

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 130

### 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:** VPABS, VPABSW, VPABSD
- **Negacja / zero / zachowanie:** VPSIGNB, VPSIGNW, VPSIGND

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 131

## Instrukcje wartość maksymalna

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 132

### Instrukcja wartość maksymalna VPMAX[U/S][B/W/D]

vpmx[u/s][b/w/d] xmm1, xmm2, xmm3/m128  
vpmx[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 maksymalnych są zapisywane w rejestrze xmm1/ymm1.

$$\text{if } \text{źródło1}[i] > \text{źródło2}[i] \text{ then } cel[i] = \text{źródło1}[i] \text{ else } cel[i] = \text{źródło2}[i]$$

$$\text{if } xmm2/ymm2[i] > xmm3/ymm3[m128[i] / ymm3/m256[i] \text{ then } xmm1/ymm1[i] = xmm2/ymm2[i]$$

$$\text{else } xmm1/ymm1[i] = xmm3/ymm3[m128[i] / ymm3/m256[i]$$

Bty od 128/196 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomose 133

### Instrukcja dodawania

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

255				128	127				0
ymm/7	ymm/6	ymm/5	ymm/4	xmm/3	xmm/2	xmm/1	xmm/0		
>	>	>	>	>	>	>	>	>	>
=	=	=	=	=	=	=	=	=	=
max	max	max	max	max	max	max	max	max	max

xmm2  
porównania  
xmm3/m128  
xmm1

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 134

### Instrukcja wartość maksymalna

#### VPMIN[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 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/ymm3[i] / ymm3/m256[i] **then** xmm1/ymm1[i] = xmm2/ymm2[i] **else** xmm1/ymm1[i] = xmm3/ymm3[i] / ymm3/m256[i]

Był od 128/256 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 135

### Instrukcja dodawania

#### VPMIN[U/S][B/W/D/Q]

255				128	127				0
ymm/7	ymm/6	ymm/5	ymm/4	xmm/3	xmm/2	xmm/1	xmm/0		
<	<	<	<	<	<	<	<	<	<
=	=	=	=	=	=	=	=	=	=
min	min	min	min	min	min	min	min	min	min

xmm2  
porównania  
xmm3/m128  
xmm1

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 136

### Instrukcje wartość średnia

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 137

### Instrukcja wartość średnia

#### VPAVG[B/W]

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

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

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

Był od 128/256 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 138

### Instrukcja dodawania

#### VPAVG[B/W]

127				64	63				0
xmm/7	xmm/6	xmm/5	xmm/4	xmm/3	xmm/2	xmm/1	xmm/0		
+	+	+	+	+	+	+	+	+	+
=	=	=	=	=	=	=	=	=	=
+1	+1	+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	avg

xmm2  
xmm3/m128  
xmm1

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 139

## Instrukcje wartości bezwzględnej

(C) KISI d.KIK PCz 2023 Programowanie nisko poziomowe 140

### 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(źródło[i])
xmm1[i] = abs(xmm2[i])
ymm1[i] = abs(ymm2[i])
```

Bryod 128/256 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023 Programowanie nisko poziomowe 141

### Instrukcja dodawania VPABS[B/W/D]

127					64	63					0
xmm1/7	xmm/6	xmm/5	xmm/4	xmm/3	xmm/2	xmm/1	xmm/0				
abs	abs	abs	abs	abs	abs	abs	abs				
=	=	=	=	=	=	=	=				

xmm2

xmm1

(C) KISI d.KIK PCz 2023 Programowanie nisko poziomowe 142

## Instrukcje znaku pozostawienie / zerowanie / negacja

(C) KISI d.KIK PCz 2023 Programowanie nisko poziomowe 143

### Instrukcja znaku VPSIGN[B/W/D]

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

Zapisuje bajty/słowa/podwójne słowa do xmm1/ymm1 wartościami z rejestru xmm2 w zależności od znaku odpowiadającej wartości wektora w rejestrze xmm3/m128.

```
if źródło[i] > 0; cel[i] = źródło[i]
if źródło[i] = 0; cel[i] = 0
if źródło[i] < 0; cel[i] = -źródło[i]
```

Bryod 128/256 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023 Programowanie nisko poziomowe 144

### Instrukcja znaku VPSIGN[B/W/D]

<p><b>ymm</b></p> <p>if ymm3/m256[i] &gt; 0 then ymm1[i] = ymm2[i] if ymm3/m256[i] = 0 then ymm1[i] = 0 if ymm3/m256[i] &lt; 0 then ymm1[i] = - ymm2[i]</p>	<p><b>xmm</b></p> <p>if xmm3/m128[i] &gt; 0 then xmm1[i] = xmm2[i] if xmm3/m128[i] = 0 then xmm1[i] = 0 if xmm3/m128[i] &lt; 0 then xmm1[i] = - xmm2[i]</p>
---	---

(C) KISI d.KIK PCz 2023 Programowanie nisko poziomowe 145



### Instrukcja dodawania

#### VPSIGN[B/W/D]

127	64			63	0		
xmm7	xmm6	xmm5	xmm4	xmm3	xmm2	xmm1	xmm0
8	7	6	5	4	3	2	1
-4	5	0	-9	2	6	0	-3
-8	7	0	-5	4	3	0	-1

xmm2  
xmm3/m128  
odczytanie bitu znaku  
  
xmm1

### Operacje porównania AVX

- **Porównanie liczb:** VPCMPEQB, VPCMPEQW, VPCMPEQD, VPCMPGTB, VPCMPGTW, VPCMPGTD, VPCMPGTQ
- **Porównanie ciągów:** VPCMPSTR, VPCMPISTR, VPCMPSTRM, VPCMPISTRM

## Instrukcje porównania

### Instrukcja porównania liczb

#### VPCMPEQ[B/W/D/Q]

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

Porównuje wartości współrzędnych wektorów bajtów/słów/podwójnych słów/poczwórnych słów z rejestru xmm3/ymm3 lub pamięci m128/m256 z odpowiednimi wartościami współrzędnych rejestru xmm2/ymm2, jeśli wartości są równe, odpowiednie współrzędne rejestru celu xmm1/ymm1 są ustawiane na -1, jeśli nie na 0.

**if źródło1[i] = źródło2[i] then cel[i] = -1 else cel[i] = 0;**  
**if xmm3/m128[i] = xmm2[i] then xmm1[i] = -1 else xmm1[i] = 0;**  
**if ymm3/m256[i] = ymm2[i] then ymm1[i] = -1 else ymm1[i] = 0;**

Byt od 128/128 do MSB są zerowane.

### Instrukcja porównania liczb

#### VPCMPEQ[B/W/D/Q]

127	64			63	0		
xmm7	xmm6	xmm5	xmm4	xmm3	xmm2	xmm1	xmm0
2	4	9	7	6	5	3	2
EQ	EQ	EQ	EQ	EQ	EQ	EQ	EQ
5	0	1	7	6	4	3	2
=	=	=	=	=	=	=	=
0	0	0	-1	-1	0	-1	-1

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/m256 (AVX2)

Porównuje wartości współrzędnych wektorów bajtów/słów/podwójnych słów/poczwórnych słów z rejestru xmm3/ymm3 lub pamięci m128/m256 z odpowiednimi wartościami współrzędnych rejestru xmm2/ymm2, jeśli wartości xmm2/ymm2 są **większe niż** wartości xmm3/ymm3 lub m128/m256 wówczas odpowiednie współrzędne rejestru celu xmm1/ymm1 są ustawiane są na -1, w przeciwnym wypadku na 0.

**if źródło1[i] > źródło2[i] then cel[i] = -1 else cel[i] = 0**  
**if xmm2[i] > xmm3/m128[i] => xmm1[i] = -1 else xmm1[i] = 0**  
**if ymm2[i] > ymm3/m256[i] then ymm1[i] = -1 else ymm1[i] = 0**

Byt od 128/128 do MSB są zerowane.

### 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	xmm/0		
7	2	3	4	6	8	9	-1		xmm3/mi28	
GT	GT	GT	GT	GT	GT	GT	GT			
0	3	4	6	5	7	0	1		xmm2	
=	=	=	=	=	=	=	=			
0	-1	-1	-1	0	0	0	-1		xmm1	

(C) KISI d.KIK PCz 2023

Programowanie mikroprocesora

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 nieznannej, 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 mikroprocesora

153

### 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] = {00b bajty bez znaku, 01b słowa bez znaku, 10b bajty ze znakiem, 11b słowa ze znakiem}

##### Operacja (sposób porównania)

imm8[3:2] = {00b porównanie arytmetyczne, czy w ciągu występują podane bajty/słowa, 01b porównanie arytmetyczne większe lub równe dla parzystych elementów wektora lub mniejsze lub równe dla nieparzystych elementów wektora 10b porównuje arytmetycznie, czy odpowiadające sobie wartości są równe 11b porównuje arytmetycznie, czy równe (w kolejności) }

(C) KISI d.KIK PCz 2023

Programowanie mikroprocesora

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] = {00b pozytywna polaryzacja (bez zmiany znaku), 01b negatywna polaryzacja (ze zmianą znaku), 10b stosowanie maski (bez zmiany znaku), 11b stosowanie maski (ze zmianą maski) dla elementów nieważnych w reg/memu są przepisywane, w przeciwnym wypadku nieważne są negowane }

(C) KISI d.KIK PCz 2023

Programowanie mikroprocesora

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:

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

(C) KISI d.KIK PCz 2023

Programowanie mikroprocesora

156

### Instrukcja porównania ciągów znakowych

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

##### Bajt sterujący imm8 (4/4)

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

##### Wybór Wyjścia

Dla indeksu (I) = wynik do ECK/RCX

imm8[6] = {0b z asynku jest pobierany najmłodszy bit  
1b z asynku jest pobierany najstarszy bit }

Dla maski (M) = wynik do xmm0

imm8[6] = {0b zwracany wynik uzupełniany jest zerami,  
1b asynku jest rozszerzany do bajtu/słowa (z samymi zerami lub jedynkami) }

(C) KISI d.KIK PCz 2023

Programowanie mikroprocesora

157

### Instrukcja porównania ciągów znakowych VPCMP[E/I]STR[I/M]

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

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

Instrukcje typu PCMPSTR wykonują 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 niskopoziomse

158

### Instrukcja porównania ciągów znakowych VPCMP[E/I]STR[I/M]

#### Algotym instrukcji definiowany w bajcie sterującym:

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

Bajty kontroly 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 porównują ciągi znakowe bajtów lub słów, **wynikiem jest:**

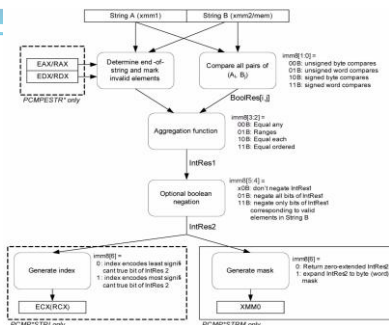
- **maska w xmm0 lub index w R/E/CX**

**ustawione flagi**

Programowanie niskopoziomse

159

### Instrukcja porównania ciągów znakowych VPCMP[E/I]STR[I/M]



(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

160

### Instrukcja porównania ciągów znakowych VPCMPESTRI

vpcmpestri xmm1, xmm2/m128, imm8 (AVX)

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

**E (Explicit)** - oznacza łańcuchy 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 (B), reset otherwise

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

OFlag - IntRes2[0]

AFlag - Reset

PFlag - Reset

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

161

### Instrukcja porównania ciągów znakowych VPCMPISTRI

vpcmpestri xmm1, xmm2/m128, imm8 (AVX)

**I (Implicit)** - oznacza łańcuch o nieoznaczonej długości.

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

**Porównuje łańcuchy o nieustalonej długości** z rejestru **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 niskopoziomse

162

### Instrukcja porównania ciągów znakowych VPCMPESTRM

vpcmpestrm xmm1, xmm2/m128, imm8 (AVX)

**E (Explicit)** - oznacza łańcuchy o ustalonej długości

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

**Porównuje łańcuchy o ustalonej długości** z rejestru **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 (B), reset otherwise

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

OFlag - IntRes2[0]

AFlag - Reset

PFlag - Reset

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomse

163

### Instrukcja porównania ciągów znakowych VPCMPISTRM

vpcmpistm xmm1, xmm2/m128, imm8 (AVX)

I (Implicit) - oznacza łańcuch o nieoznaczonej długości.  
 M (Mask) - oznacza, że instrukcja tworzy na wyjściu maskę

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

Operand 1	Operand 2	Wynik
xmm1	xmm2/m128	xmm0

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  
 DFlag - IntRes2[0]  
 AFlag - Reset  
 PFlag - Reset

(C) KISI d.KIK PCz 2023 Programowanie mikroprocesora 164

### Operacje przesunięć AVX liczb całkowite

- Przesunięcie w lewo:**
  - VPSLL[W/D/Q]
  - VPSLLDQ
  - VPSLLV[W/D/Q]
- Przesunięcie w prawo:**
  - VPSRL[W/D/Q]
  - VPSRLDQ
  - VPSRLV[D/Q]
  - VPSRA[W/D/Q]
  - VPSRAV[W/D/Q]

(C) KISI d.KIK PCz 2023 Programowanie mikroprocesora 165

## Instrukcje przesunięć bitowych

(C) KISI d.KIK PCz 2023 Programowanie mikroprocesora 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)

Przesuwalogenicznie słowa/podwójne słowa /poczwórne słowa w prawo logicznie (całe) z rejestru xmm2/ymm2 o wartość wskazaną przez rejestr xmm3/ymm3 lub m128/m256. Podczas przesunięcia starsze bity są zerowane. Jeśli wartość licznika jest większa niż 31 dla podwójnych słów, 63 dla poczwórnych słów, wówczas wszystkie bity są zerowane.

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

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023 Programowanie mikroprocesora 167

### Instrukcja przesunięcia w prawo VPSRLD

127	96	64	32	0
xmm1/3	xmm1/2	xmm1/1	xmm1/0	
DW <sub>3</sub>	DW <sub>2</sub>	DW <sub>1</sub>	DW <sub>0</sub>	ymm2/xmm2
DW <sub>3</sub> -licznik	DW <sub>2</sub> -licznik	DW <sub>1</sub> -licznik	DW <sub>0</sub> -licznik	ymm1/xmm1

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

(C) KISI d.KIK PCz 2023 Programowanie mikroprocesora 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 liczbę bajtów określoną przez imm8. Podczas przesunięcia starsze bity są zerowane.

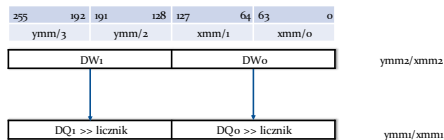
cel[i] = źródło1[i] >> imm8  
 xmm1[i] = xmm2[i] >> imm8  
 ymm1[i] = ymm2[i] >> imm8

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2023 Programowanie mikroprocesora 169

## Instrukcja dodawania

## VPSRDQ



Licznik jest określony w imm8.

(C) KISI d.KIK PCz 2023

Programowanie mikroprocesora

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 poczwórnych słów, wówczas wszystkie bity są zerowane.

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

Bit od 128/196 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie mikroprocesora

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 powieleniem bitu znaku** słowa/podwójne słowa z rejestru xmm2/ymm2 (**całe**) określoną 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 poczwórnych słów, wówczas wszystkie bity są ustawiane na bit znaku.

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

Bit od 128/196 do MSB są zerowane.

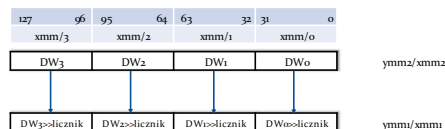
(C) KISI d.KIK PCz 2023

Programowanie mikroprocesora

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 mikroprocesora

173

## Instrukcja przesunięcia arytmetycznego w prawo

## VPSRAVD

vpsravd xmm1, xmm2, xmm3/m128 (AVX2)

vpsravd ymm1, ymm2, ymm3/m256 (AVX2)

Przesuwa **arytmetycznie w prawo z powieleniem bitu znaku** słowa/podwójne słowa z rejestru xmm2/ymm2 o liczbę bitów określoną przez wartości zapisane 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.

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

Bit od 128/196 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie mikroprocesora

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 słowa (w całości) z rejestru xmm2/ymm2 o wartość określoną przez rejestr xmm3/ymm3 lub m128/m256 lub przez bajt sterujący imm8. **Młodsze bity są zerowane**. Jeśli wartość licznika jest większa niż 15 dla słów, większa niż 31 dla podwójnych słów, większa niż 63 dla poczwórnych 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/m128[0] lub imm8
```

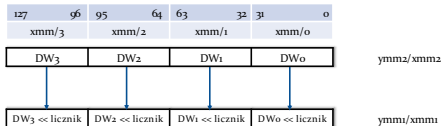
Bit od 128/196 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie mikroprocesora

175

### Instrukcja przesunięcia logicznego w lewo VPSLLD



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

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomose

176

### Instrukcja przesunięcia logicznego w lewo VPSLLV[D/Q]

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

Przesuwa **logicznie w lewo** podwójne słowo/poczwórne słowo z rejestru xmm2/ymm2 o liczbę bitów określoną przez rejestr xmm3/ymm3 lub m128/m256. **Młodsze bity są zerowane**. Jeśli wartość licznika jest większa niż 31 dla podwójnych słów, większa niż 63 dla poczwórnych 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/m128[i]
```

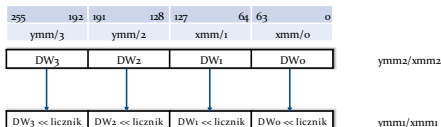
Byte od 128/196 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomose

177

### Instrukcja przesunięcia arytmetycznego w lewo VPSLLVQ



Liczniki są określone w **xmm3/ymm3** lub **m128/m256**

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomose

178

### Operacje logiczne / inne AVX liczyby 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 niskopoziomose

179

## Instrukcje logiczne

### 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 m128/m256, wynik zapisuje w xmm1.

Oblicza **iloczyn logiczny bit po bicie** operandów xmm3/ymm3 lub m128/m256 oraz negacji operandu xmm2/ymm2, wynik zapisuje w xmm1/ymm1.

```
cel[i] = źródło1[i] and źródło2[i]
```

```
cel[i] = (not źródło1[i]) and źródło2[i]
```

Byte od 128/196 do MSB są zerowane.

(C) KISI d.KIK PCz 2023

Programowanie niskopoziomose

Byte od 128/256 do MSB są zerowane.

181

### Instrukcje logiczne alternatywa

#### VPOR / VPXOR

vp<sub>or</sub> xmm1, xmm2, xmm3/m128  
vp<sub>or</sub> ymm1, ymm2, ymm3/m256 (AVX2)

vp<sub>xor</sub> xmm1, xmm2, xmm3/m128 (AVX)  
vp<sub>xor</sub> 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.

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] or źródło2[i]  
cel[i] = źródło1[i] xor źródło2[i]

Bity od 128/196 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023      Programowanie niskopoziomse      Bity od 128/256 do MSB są zerowane.      182

## Instrukcje dodatkowe

(C) KISI d.KIK PCz 2023      Programowanie niskopoziomse      183

### Instrukcje dodatkowe

#### VZEROALL / VZERoupper

- vzeroall (AVX)**  
Zeruje wszystkie rejestry ymm/0 - ymm/15
- vzeroupper (AVX)**  
Zeruje bity od 128. do ostatniego rejestrów ymm/0 - ymm/15 / zmm/0 - zmm/15.

W trybie 32 bitowym zeruje tylko pierwsze 8 rejestrów.

Bity od 128/196 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023      Programowanie niskopoziomse      184

### Instrukcje dodatkowe

#### VLDMXCSR / VSTMXCSR

- vldmxcsr m32 (AVX)**  
Ładuje zawartość operandu źródłowego m32 do rejestru kontrolnego i statusu (MXCSR Control and Status Register), jest to **ładowanie ustawień**.
- vstmxcscr m32 (AVX)**  
Przesyła zawartość rejestru kontrolnego i statusu (MXCSR Control and Status Register) do operandu źródłowego m32, jest to **kopiowanie ustawień**.

(C) KISI d.KIK PCz 2023      Programowanie niskopoziomse      185

### Instrukcja łączenia dodatkowe

#### VPALIGNR

vp<sub>alignr</sub> xmm1, xmm2, xmm3/m128, imm8 (AVX)  
vp<sub>alignr</sub> ymm1, ymm2, ymm3/m256, imm8 (AVX2)

Łączy (konkatenacja) rejestry źródła xmm2/ymm2 i xmm3/ymm3 lub m128/m256, na podstawie bajtu sterującego przesuwają 128 bitowe części o imm8\*8 i zapisuje 128 bitowe części do rejestru celu xmm1/ymm1.

cel = (źródło1+źródło2) >> źródło3  
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

Bity od 128/196 do MSB są zerowane.  
(C) KISI d.KIK PCz 2023      Programowanie niskopoziomse      186

### Instrukcja łączenia dodatkowe

#### VPALIGNR

Sposób łączenia rejestrów xmm

(C) KISI d.KIK PCz 2023      Programowanie niskopoziomse      187

### Instrukcja łączenia dodatkowe

#### VPALIGNR

Sposób łączenia rejestrów xmm

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

## Instrukcje szyfrujące

(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse 189

### Operacje szyfrujące AVX

- Instrukcje szyfrujące

VAEENC, VAEENCLAST  
VAESDEC, VAESDECLAST  
VAESIMC, VAESKEYGENASSIST

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

### Instrukcja szyfrująca AES (Advanced Encryption Standard)

#### VAEENC

vaeenc xmm1, xmm2, xmm3/m128

Szyfruje jedną rundą (jednokrotnie) 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
}

```

Bity od 128/196 do MSB nie są modyfikowane.

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



## Instrukcja szyfrująca AES (Advanced Encryption Standard)

## VAESENCLEAST

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

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

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

Byte od 128/196 do MSB nie są modyfikowane.  
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse

194

## Instrukcja szyfrująca AES (Advanced Encryption Standard)

## VAESDEC

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

Odszyfrowuje jedną rundą (jednokrotnie) blok danych całego rejestru xmm2 (128 bitów) z wykorzystaniem 128 bitowego klucza wcześniej użytego do zaszyfrowania zapianego 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
}
```

Byte od 128/196 do MSB nie są modyfikowane.  
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse

195

## Instrukcja szyfrująca AES (Advanced Encryption Standard)

## VAESDECLAST

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

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

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

Byte od 128/196 do MSB nie są modyfikowane.  
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse

196

## Instrukcja szyfrująca AES (Advanced Encryption Standard)

## VAESIMC

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

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

Funkcja InvMixColumns() jest odwrotnością funkcji MixColumns().

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

Byte od 128/196 do MSB nie są modyfikowane.  
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse

197

## Instrukcja szyfrująca AES (Advanced Encryption Standard)

## VAESKEYGENASSIST

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

Asystuje 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(źródło1-key), źródło2
xmm1 = szyfrowanie(xmm2/m128), imm8
```

Byte od 128/196 do MSB nie są modyfikowane.  
(C) KISI d.KIK PCz 2023 Programowanie niskopoziomse

198