Instrukcje AVX

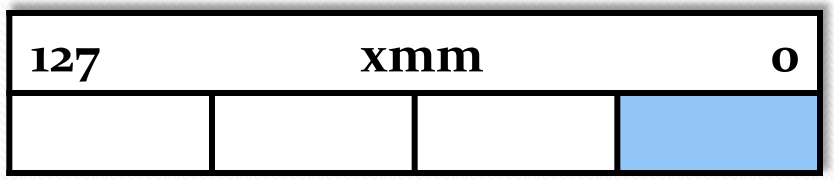
Advanced Vector Extensions

Instrukcje AVX

Część II

liczby zmienno-przecinkowe

Operacje na skalarach



SS skalar / **pojedynczej precyzji**

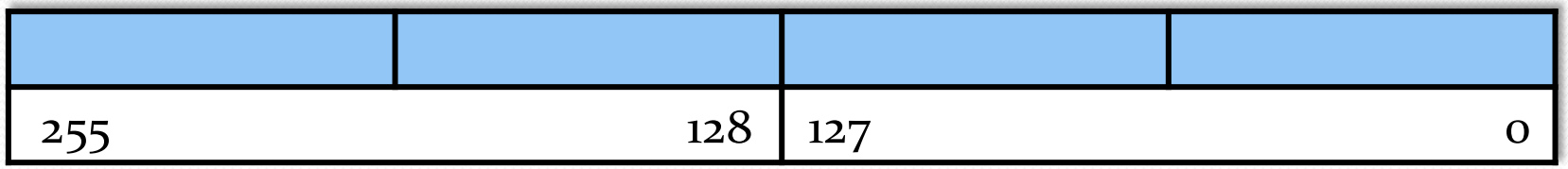
SD skalar / **podwójnej precyzji**

Operacje na wektorach

PS wektor / **pojedynczej precyzji**

PD wektor / **podwójnej precyzji**

PS wektor / **pojedynczej precyzji**



PD wektor / **podwójnej precyzji**

Operacje AVX zmienno-przecinkowe

- Instrukcje przesłania
- Instrukcje arytmetyczne (w tym FMA)
- Instrukcje porównania
- Instrukcje logiczne
- Instrukcje konwersji

Instrukcje przesłania AVX

- Instrukcje przesłania

VMOVS[S/D], VMOV[U/A]P[S/D]

VMOVNTP[S/D], VMOV[H/L]P[S/D]

VMOV[HL/LH]PS, VMOV[D/SH/SL]DUP

VMASKMOVP[S/D]

- Instrukcje konwersji: VUNPACK[H/L]P[S/D]
- Instrukcje wstawiania: VINSERTPS, VINSERTF128
- Instrukcje wyciągania: VEXTRACTPS, VEXTRACTF128

Instrukcje przesłania

Instrukcja przesłania

VMOVSS

1. vmovss xmm1, xmm2, xmm3

Przepisuje z xmm2 do xmm1 liczbę pojedynczej precyzji pozostałe uzupełnia z xmm3.

xmm1[31:0] ← xmm3[31:0]

xmm1[127:32] ← xmm2[127:32}

2. vmovss xmm1, m32

Przepisuje liczbę rzeczywistą pojedynczej precyzji z pamięci m32 do rejestru xmm1.

xmm1 ← m32

xmm1[127:32] ← 0

3. vmovss m32, xmm1

Przesyła liczbę rzeczywistą pojedynczej precyzji (scalar) z xmm1 do pamięci m32.

m32 ← xmm1[31:0]

Bity od 32/128/256 do MSB są zerowane.

Instrukcja przesłania

VMOVSD

1. vmovsd xmm1, xmm2, xmm3

Przepisuje z xmm2 do xmm1 liczbę podwójnej precyzji pozostałe uzupełnia z xmm3.

$$xmm1[63:0] \leftarrow xmm2[63:0]$$
$$xmm1[127:64] \leftarrow xmm3[127:64]$$

2. vmovsd xmm1, m64

Przepisuje liczbę rzeczywistą podwójnej precyzji z pamięci m64 do rejestru xmm1.

$$xmm1[63:0] \leftarrow m64[63:0]$$
$$xmm1[127:64] \leftarrow 0$$

Bity od 64/128/256 do MSB są zerowane.

3. vmovsd m64, xmm1

Przesyła liczbę rzeczywistą pojedynczej precyzji z xmm1 do pamięci m64.

$$m64 \leftarrow xmm1[64:0]$$

Instrukcja przesłania

VMOV[U/A]P[S/D]

vmov[u/a]p[s/d] xmm1, xmm2/m128

vmov[u/a]p [s/d] ymm1, ymm2/m256

Przesyła wektory liczb rzeczywistych pojedynczej/podwójnej precyzji **bez wyrównania / z wyrównaniem** (U – unaligned / A – aligned) z xmm2/ymm2 lub m128/m256 do xmm1/ymm1.

cel ← źródło | xmm1/ymm1 ← xmm2/ymm2 lub m128/m256

vmov[u/a]p[s/d] xmm2/m128, xmm1

vmov[u/a]p[s/d] ymm2/m256, ymm1

Przesyła wektory liczb zmienno-przecinkowych pojedynczej/podwójnej precyzji **bez wyrównania / z wyrównaniem** (U – unaligned / A - aligned) z xmm1/ymm1 do xmm2/ymm2 lub m128/m256.

cel ← źródło | xmm2/ymm2 lub m128/m256 ← xmm1/ymm1

Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania

VMOVNTP[S/D]

vmovntp[s/d] m128, xmm1

vmovntp[s/d] m256, ymm1 (AVX2)

Przesyła wektor liczb rzeczywistych pojedynczej / podwójnej precyzji z rejestru xmm1/ymm1 do pamięci m128/m256.

cel ← źródło

m128 ← xmm1

m256 ← ymm1

NT oznacza (non-temporal hint) przesłanie z pominięciem pamięci podręcznej (cache).

Bity od 128/256 do MSB są zerowane.

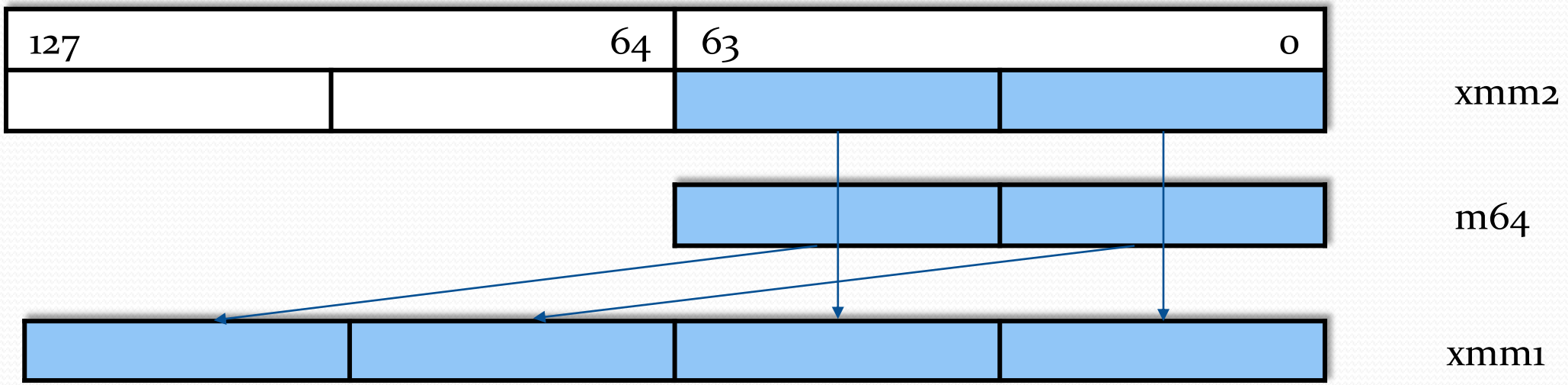
Instrukcja przesłania

VMOVHPS

1. vmovhps xmm1, xmm2, m64

Przesyła **dwie wartości** rzeczywiste pojedynczej precyzji z młodszej połowy xmm2 do młodszej połowy xmm1 oraz dwie wartości rzeczywiste z pamięci m64 do starszej połowy rejestru celu xmm1.

$$xmm1[63:0] \leftarrow xmm2[63:0] \;\&\&\; xmm1[127:64] \leftarrow m64[63:0]$$



Bity od 128/256 do MSB są zerowane.

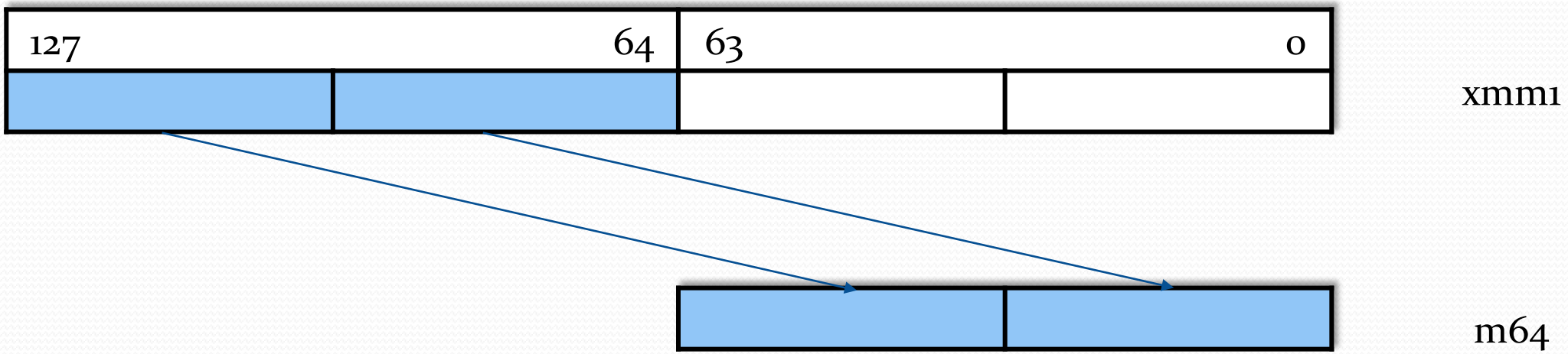
Instrukcja przesłania

VMOVHPS

2. vmovhps m64, xmm1

Przesyła **dwie wartości** rzeczywiste pojedynczej precyzji ze starszej połowy xmm1 do pamięci m64.

m64 ← xmm1[127:64]



Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania

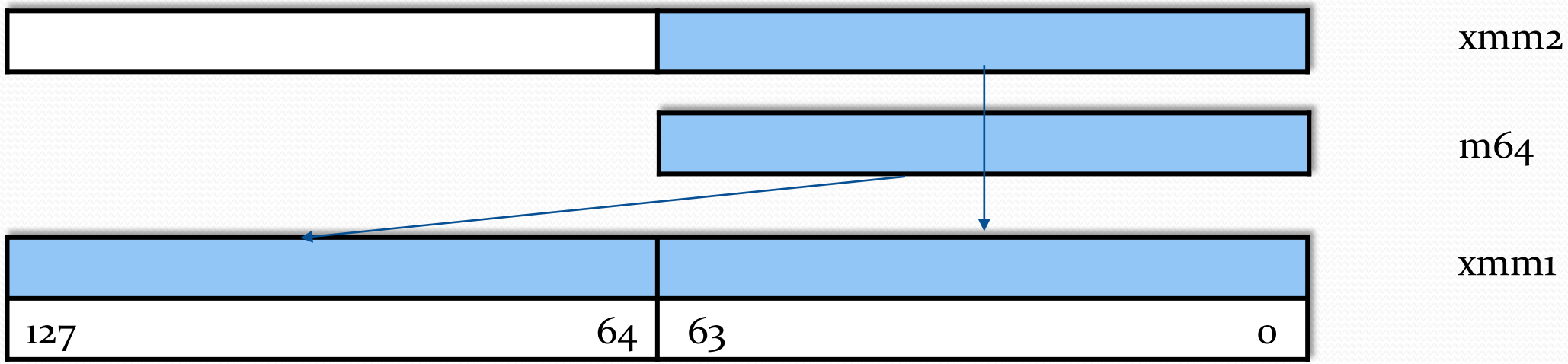
VMOVHPD

1. vmovhpd xmm2, xmm1, m64

Kopiuje wartości liczb rzeczywistych podwójnej precyzji z młodszej połowy rejestru xmm1 oraz z pamięci m64, wynik zapisuje w xmm2.

xmm1[63:0] ← xmm2[63:0] && xmm1[127:64] ← m64[63:0]

.



Bity od 128/256 do MSB są zerowane.

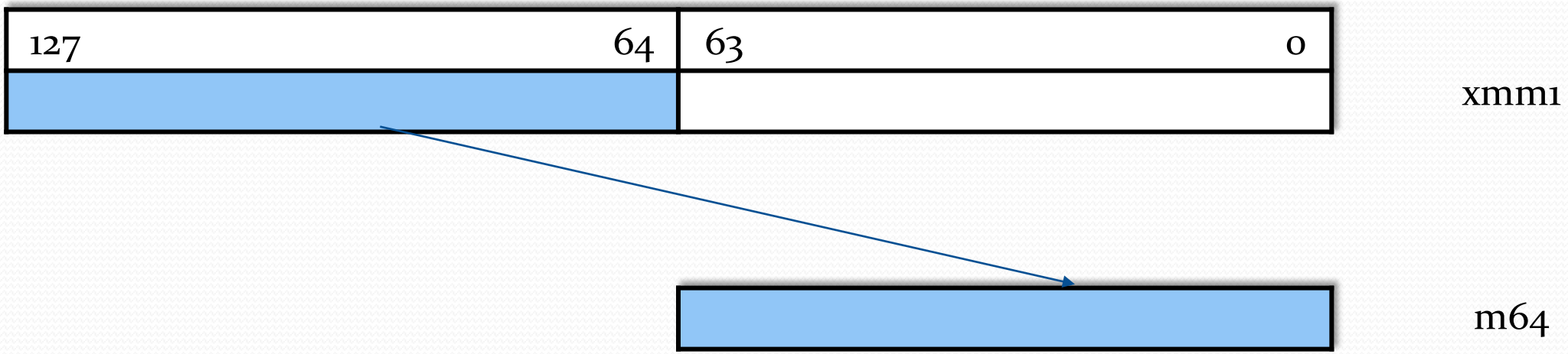
Instrukcja przesłania

VMOVHPD

2. vmovhpd m64, xmm1

Kopiuje liczbę rzeczywistą podwójnej precyzji ze starszej połowy xmm1 do pamięci m64.

m64 ← xmm1[127:64]



Bity od 128/256 do MSB są zerowane.

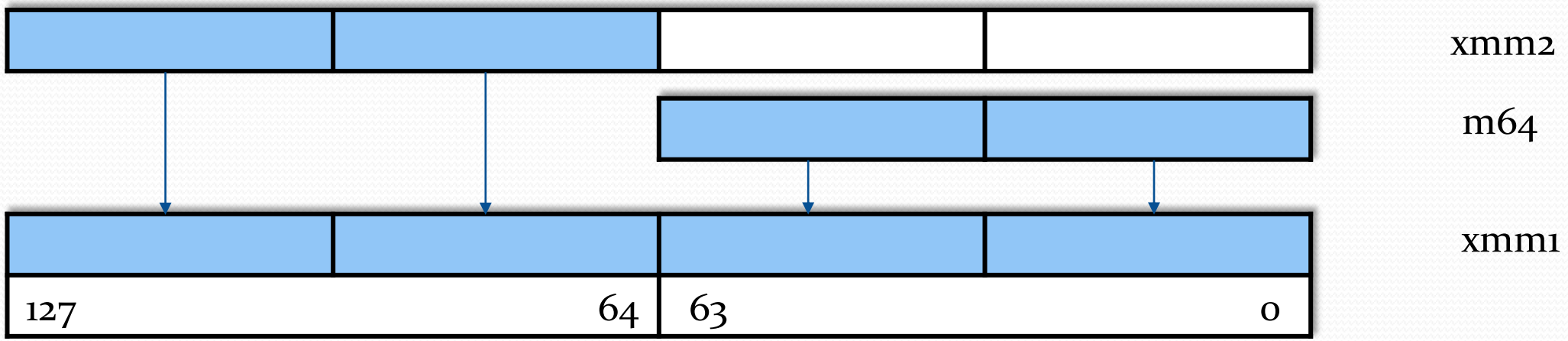
Instrukcja przesłania

VMOVLPS

1. vmovlps xmm1, xmm2, m64

Przepisuje po dwie wartości rzeczywiste pojedynczej precyzji ze starszej połowy rejestru xmm2 do xmm1 oraz z pamięci m64, wynik zapisuje w xmm1.

$$xmm1[63:0] \leftarrow m64[63:0] \;\&\&\; xmm1[127:64] \leftarrow xmm2[127:64]$$



Bity od 128/256 do MSB są zerowane.

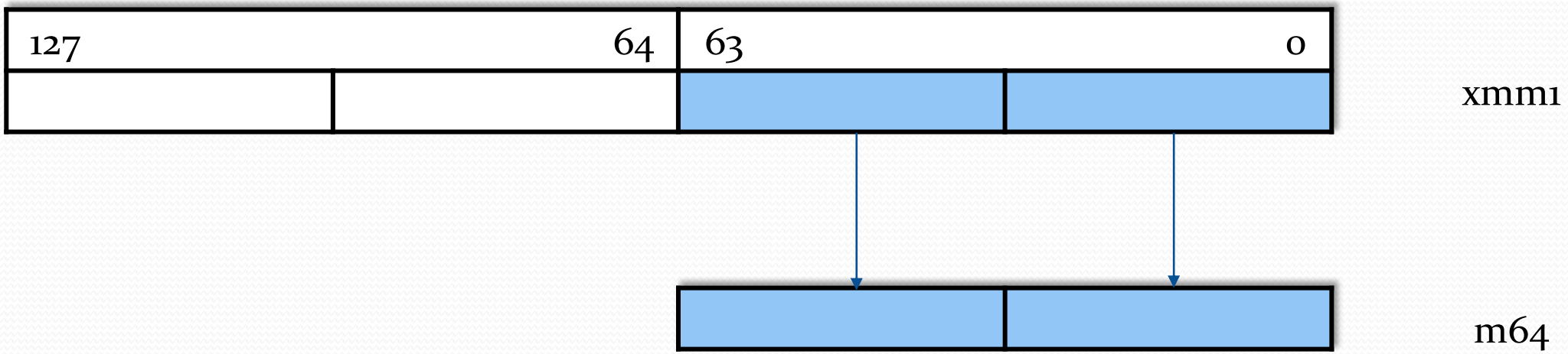
Instrukcja przesłania

VMOVLPS

2. vmovlps m64, xmm1

Przesyła dwie wartości liczb rzeczywistych pojedynczej precyzji z młodszej połowy xmm1 do pamięci m64.

m64[63:0] ← xmm1[63:0]



Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania

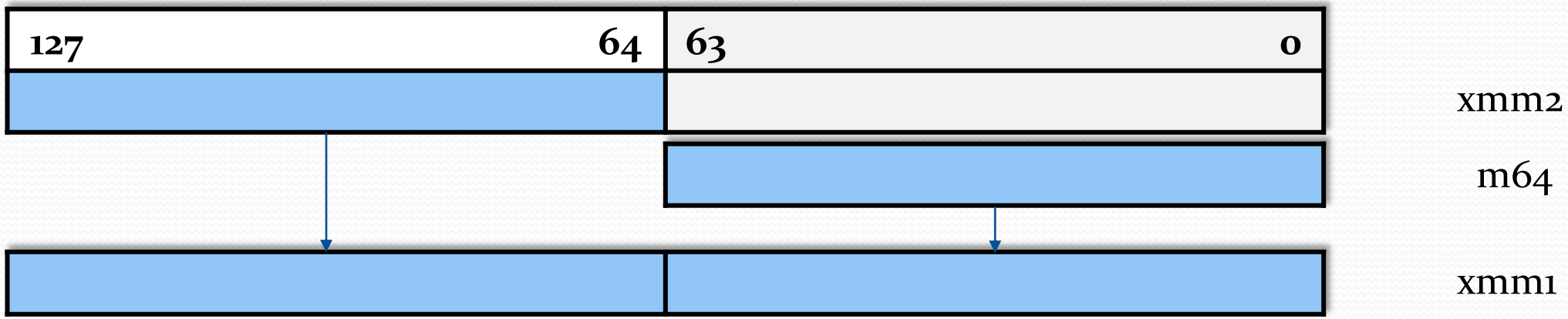
VMOVLPD

1. vmovlpd xmm1, xmm2, m64

Przesyła liczbę rzeczywistą podwójnej precyzji z pamięci m64 oraz starszą **połową** rejestru xmm2, wynik zapisuje w xmm1.

$$xmm1[63:0] \leftarrow m64[63:0] \;\&\&\; xmm1[127:64] \leftarrow xmm2[127:64]$$

.



Bity od 128/256 do MSB są zerowane.

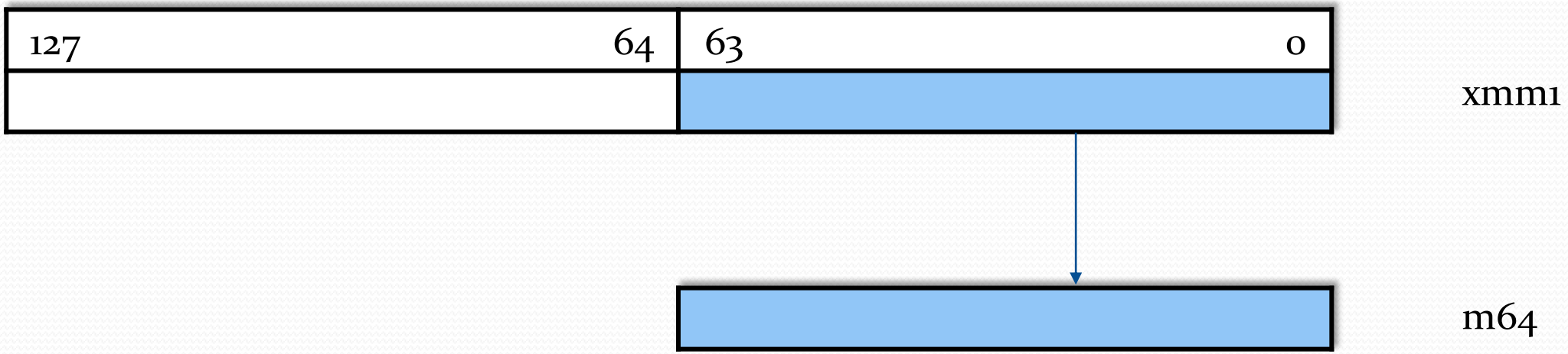
Instrukcja przesłania

VMOVLPD

2. vmovlpd m64, xmm1

Przesyła **liczbę** rzeczywistą podwójnej precyzji z młodszej połowy xmm1 do pamięci m64.

m64[63:0] ← xmm1[63:0]



Bity od 128/256 do MSB są zerowane.

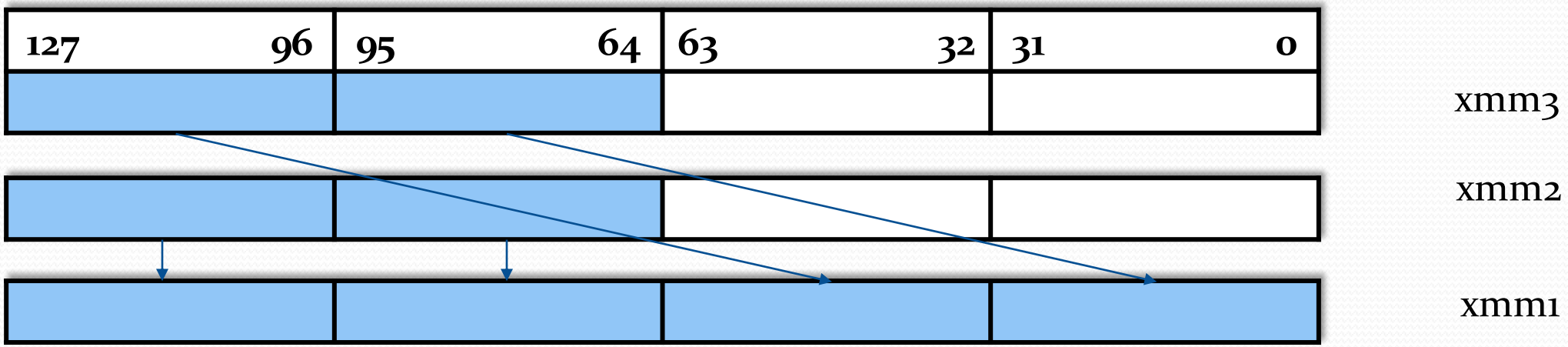
Instrukcja przesłania

VMOVHLPS

vmovhlps xmm1, xmm2, xmm3

Przesyła ze starszej połowy rejestru xmm3 do młodszej połowy xmm1 oraz z starszej połowy rejestru xmm2 do starszej połowy rejestru celu po dwie **liczby rzeczywiste pojedynczej precyzji**, wynik zapisuje w xmm1.

xmm1[63:0] ← xmm3[127:64] && xmm1[127:64] ← xmm2[127:64]



Bity od 128/256 do MSB są zerowane.

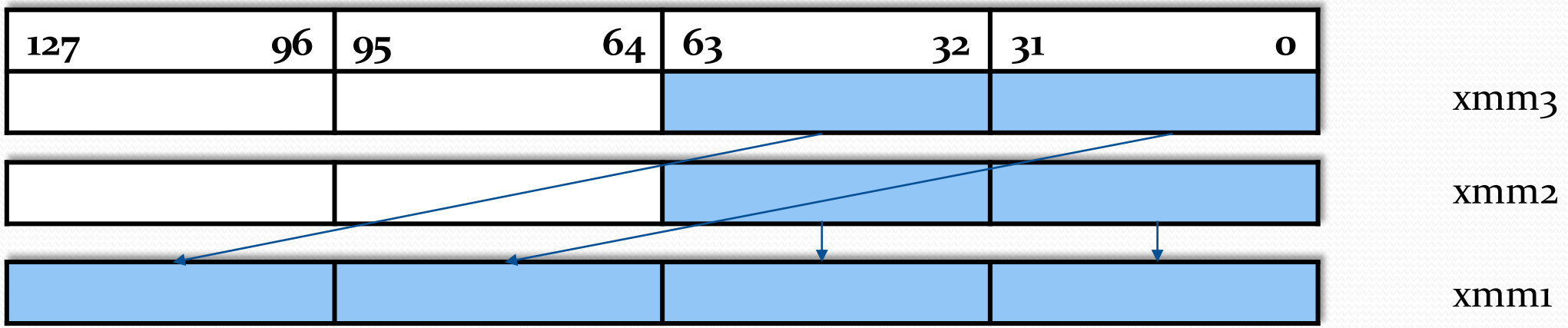
Instrukcja przesłania

VMOVLHPS

vmovlhps xmm1, xmm2, xmm3

Przesyła z młodszej połowy rejestru xmm2 do młodszej połowy rejestru celu oraz ze młoszej połowy rejestru xmm3 do starszej połowy rejestru celu po dwie **liczby rzeczywiste pojedynczej precyzji**, wynik zapisuje w xmm1.

xmm1[63:0] ← xmm2[63:0] && xmm1[127:64] ← xmm3[127:63]



Bity od 128/256 do MSB są zerowane.

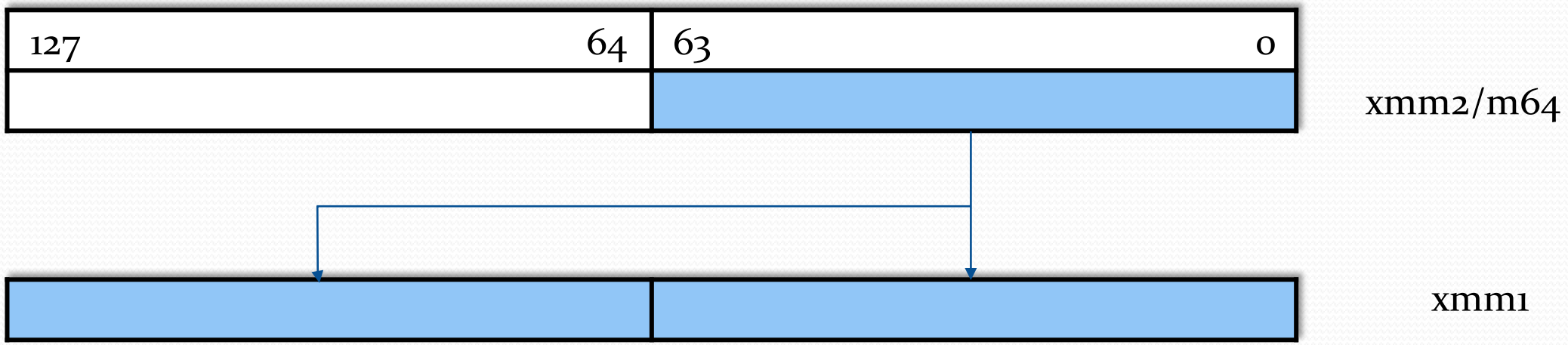
Instrukcja przesłania

VMOVDDUP

vmovddup xmm1, xmm2/m64

Kopiuje liczbę rzeczywistą podwójnej precyzji z rejestru xmm2 lub pamięci m64 do obu części rejestru xmm1.

$$xmm1[63:0] \leftarrow xmm2/m64[63:0] \;\&\&\; xmm1[127:64] \leftarrow xmm2/m64[63:0]$$



Bity od 128/256 do MSB są zerowane.

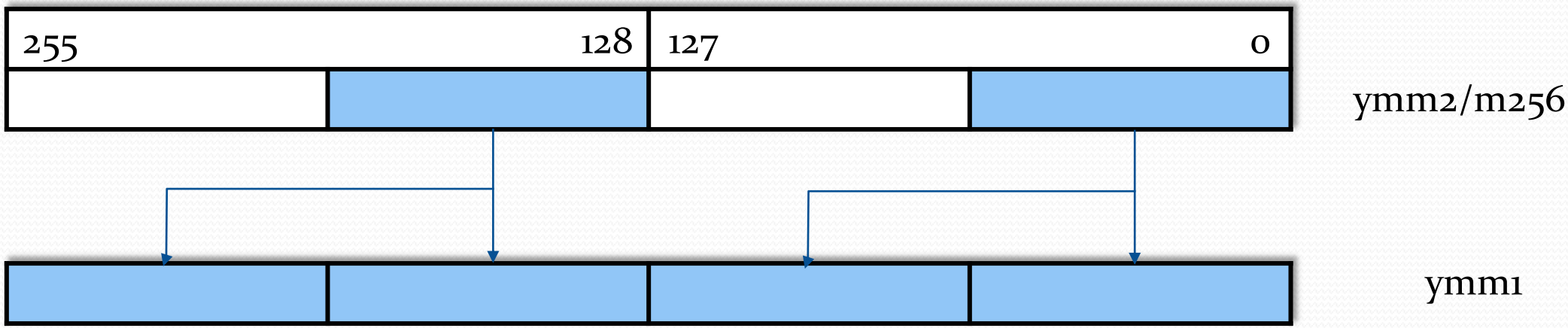
Instrukcja przesłania

VMOVDDUP

vmovddup ymm1, ymm2/m256

Kopiuje liczby rzeczywiste podwójnej precyzji o parzystych indeksach z rejestru ymm2 lub pamięci m256 do ymm1.

$$ymm1[2i] \leftarrow ymm2/m256[2i] \;\&\&\; ymm1[2i+1] \leftarrow ymm2/m256[2i]$$



Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania

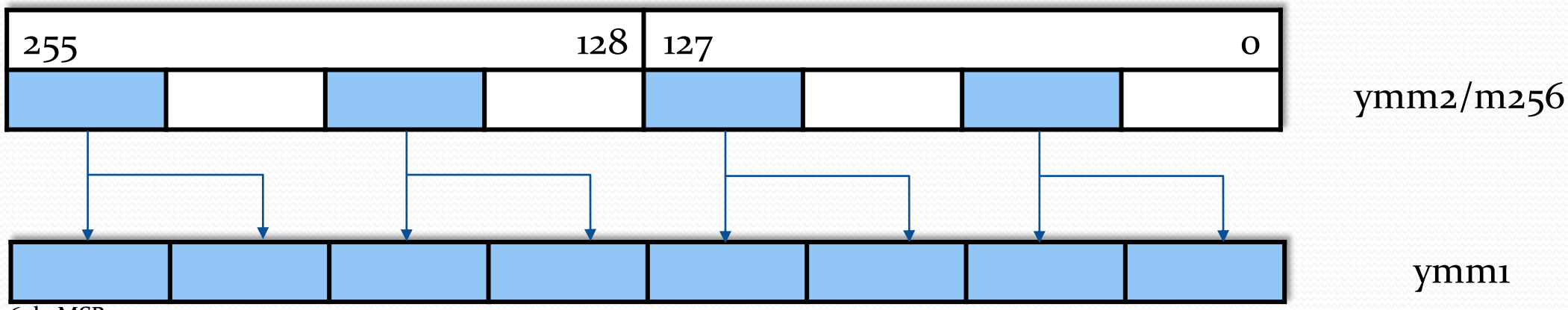
VMOVSHDUP

vmovshdup xmm1, xmm2/m128

vmovshdup ymm1, ymm2/m256

Kopiuje z powieleniem wartości liczb rzeczywistych pojedynczej precyzji o **nieparzystych indeksach** xmm2/ymm2 lub m128/m256 i zapisuje do xmm1/ymm1.

ymm1[2i] ← ymm2/m256[2i+1] && ymm1[2i+1] ← ymm2/m256[2i+1]



Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania

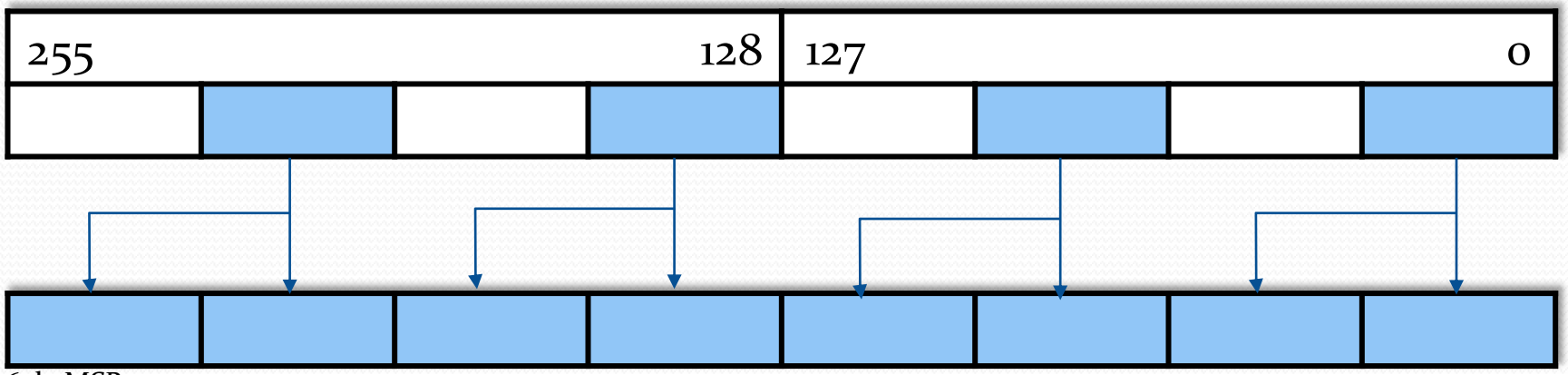
VMOVSLDUP

vmovsldup xmm1, xmm2/m128

vmovsldup ymm1, ymm2/m256

Kopiuje z powieleniem wartości liczb rzeczywistych pojedynczej precyzji o **parzystych indeksach** xmm2/ymm2 lub m128/m256 i zapisuje do xmm1/ymm1.

xmm1[2i] ← xmm2/m128[2i] && ymm1[2i+1] ← ymm2/m256[2i]



Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania warunkowego

VMASKMOVPS (1/4)

vmaskmovps xmm1, xmm2, m128

vmaskmovps ymm1, ymm2, m256

Przesyła liczby rzeczywiste pojedynczej precyzji z pamięci m128/m256 do rejestru celu xmm1/ymm1, 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.

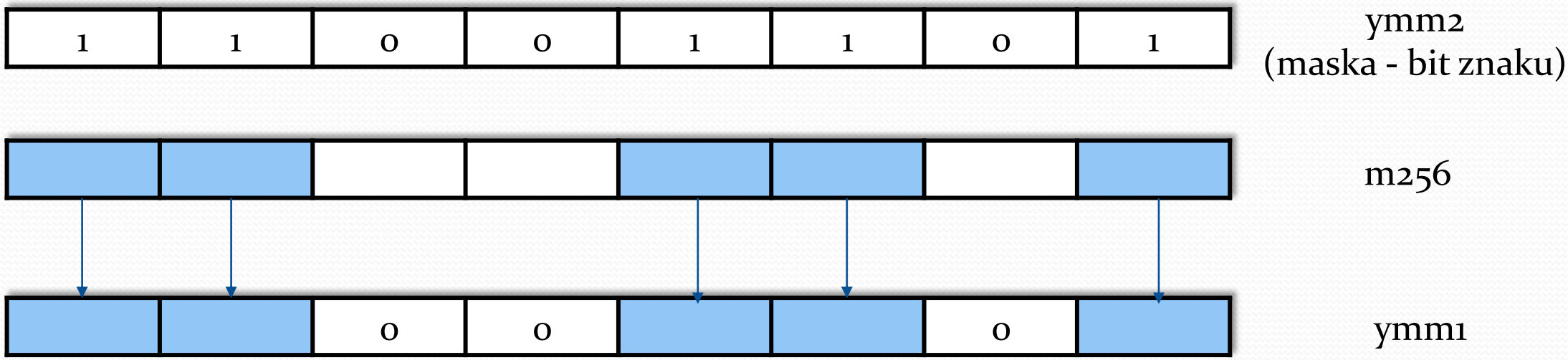
```
if xmm2[i][31] then xmm1[i] ← m128[i]
        else xmm1[i] = 0

if ymm2[i][31] then ymm1[i] ← m256[i]
        else ymm1[i] = 0
```

Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania warunkowego

VMASKMOVPS (2/4)



Model przesłania warunkowego z pamięci

Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania warunkowego

VMASKMOVPS (3/4)

vmaskmovps m128, xmm1, xmm2

vmaskmovps m256, ymm1, ymm2

Przesyła liczby rzeczywiste pojedynczej precyzji z rejestru xmm2/ymm2 do pamięci m128/m256 pod warunkiem, że bit znaku odpowiadających wartości z rejestru maski (drugi operand) xmm1/ymm1 **jest ustawiony na jeden,** w przeciwnym wypadku zapisuje zero.

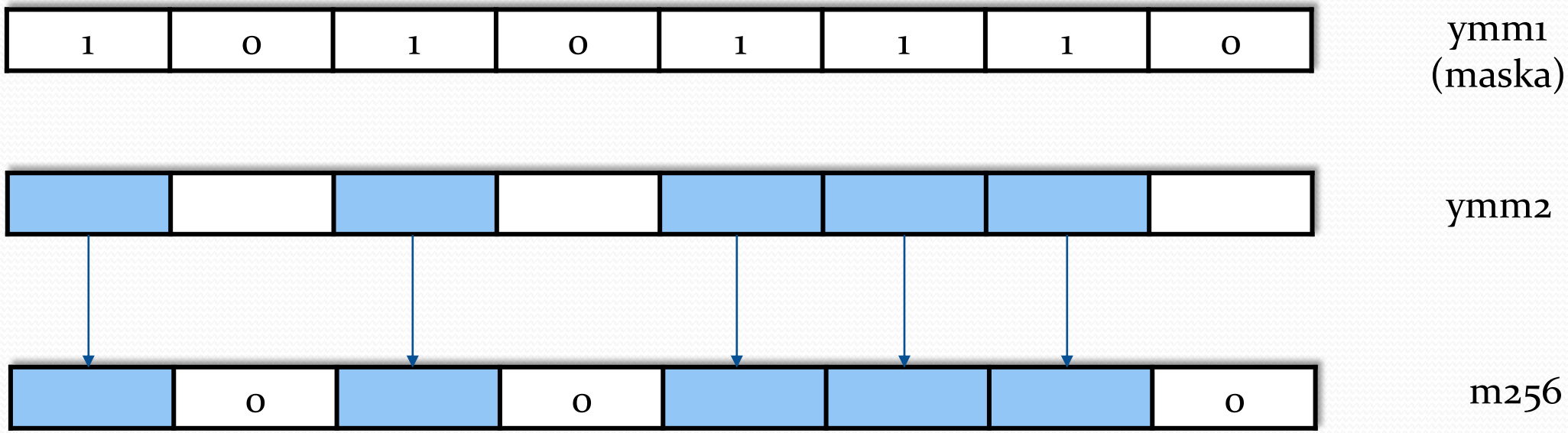
if xmm1[i][31] then m128[i] ← xmm2[i]

if ymm1[i][31] then m256[i] ← ymm2[i]

Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania warunkowego

VMASKMOVPS (4/4)



Model przesłania warunkowego do pamięci

Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania warunkowego

VMASKMOVPD (1/4)

vmaskmovpd xmm1, xmm2, m128

vmaskmovpd ymm1, ymm2, m256

Pobiera kolejne elementy pamięci **podwójnej precyzji z m128/m256**, wynik zapisuje warunkowo w ymm1/xmm1 według **maski** (bit znaku) zdefiniowanej w **ymm2/xmm2.**

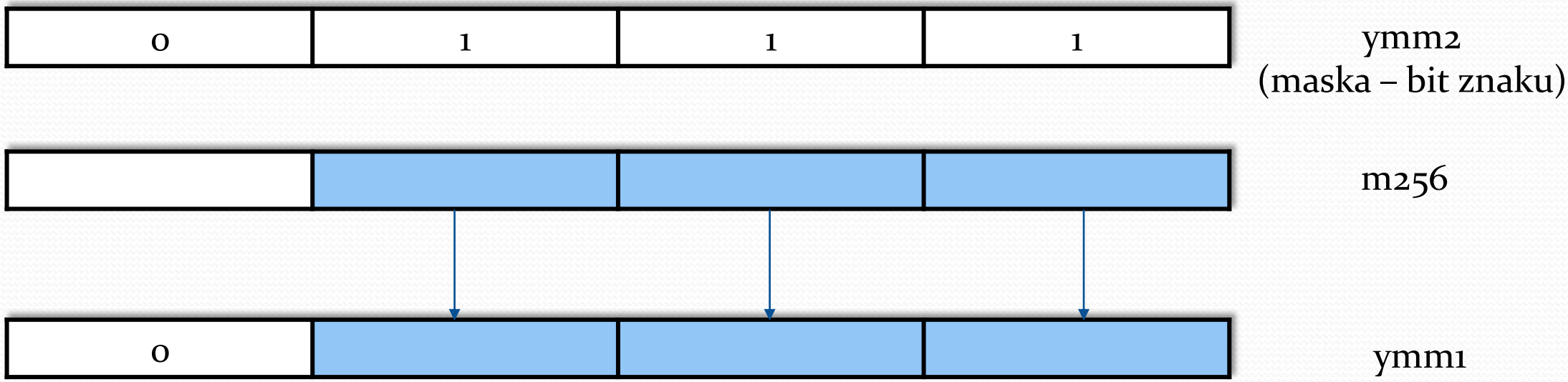
if xmm2[i][63] then xmm1[i] ← m128[i]
else xmm1[i] = 0

if ymm2[i][63] then ymm1[i] ← m256[i]
else ymm1[i] = 0

Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania warunkowego

VMASKMOVPD (2/4)



Model przesłania warunkowego z pamięci

Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania warunkowego

VMASKMOVPD (3/4)

vmaskmovpd m128, xmm1, xmm2

vmaskmovpd m256, ymm1, ymm2

Zapisuje do pamięci m128/m256 kolejne elementy wektora **podwójnej precyzji** z xmm2/ymm2, według maski (bit znaku) zdefiniowanej w ymm1/ xmm1.

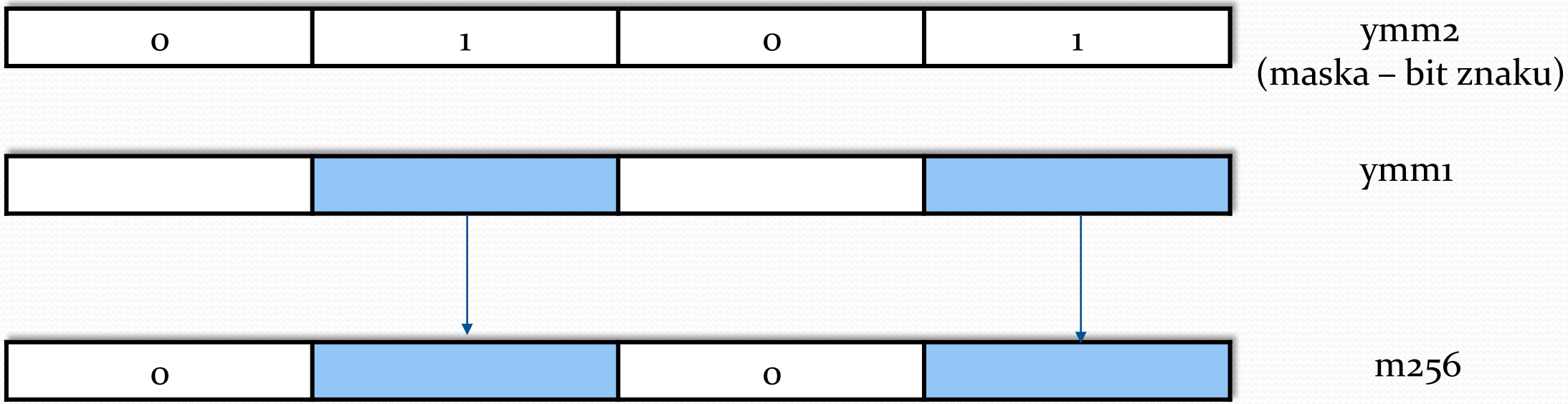
if xmm1[i][63] then m128[i] ← xmm2[i]

if ymm1[i][63] then m256[i] ← ymm2[i]

Bity od 128/256 do MSB są zerowane.

Instrukcja przesłania warunkowego

VMASKMOVPD (4/4)



Model przesłania warunkowego do pamięci

Bity od 128/256 do MSB są zerowane.

Instrukcje dekompresji

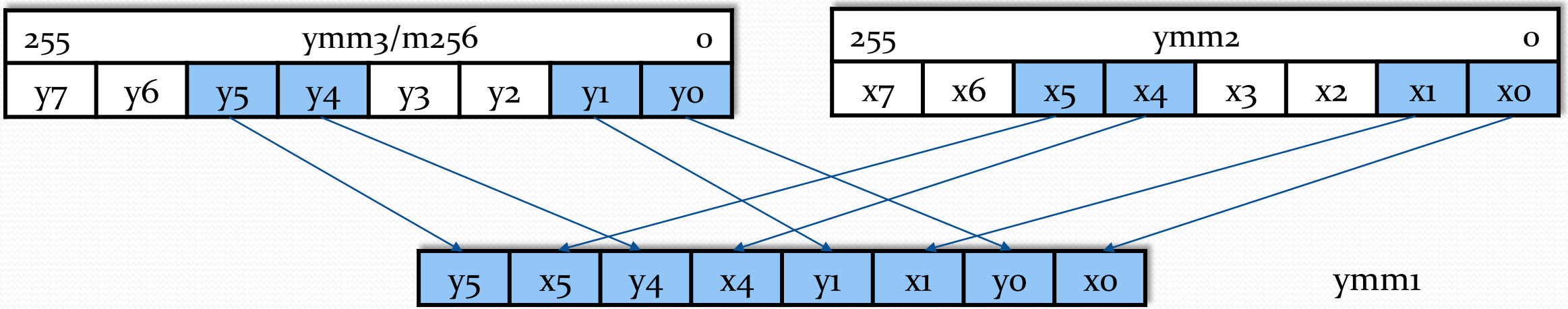
Instrukcja dekompresji

VUNPCKLPS

vunpcklps xmm1, xmm2, xmm3/m128

vunpcklps ymm1, ymm2, ymm3/m256

Przepisuje liczby rzeczywiste pojedynczej precyzji. Młodsze dwie liczby ze 128-bitowych części rejestru xmm2/ymm2 i xmm3/m128 / ymm3/m256 są zapisywane **z przeplotem** jako **liczby rzeczywiste pojedynczej precyzji** do rejestru xmm1/ymm1.



Bity od 128/256 do MSB są zerowane.

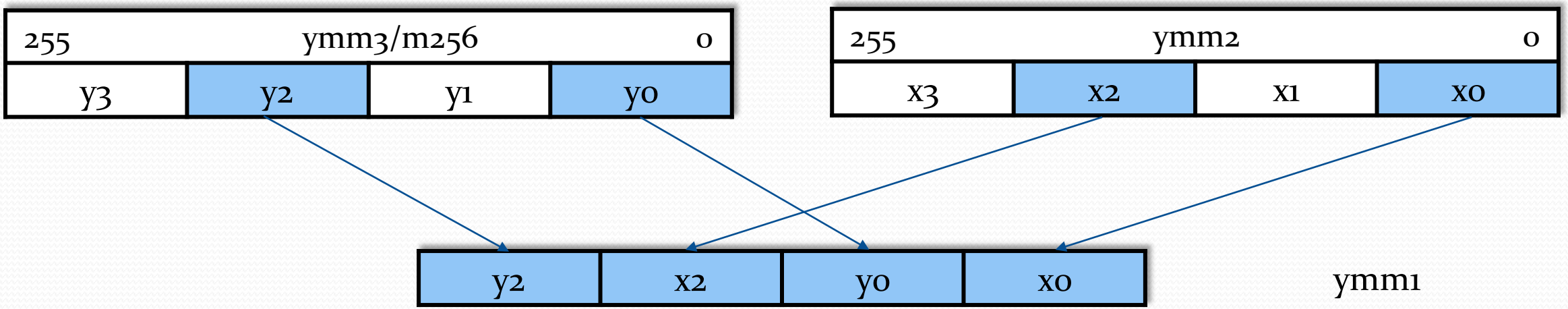
Instrukcja dekompresji

VUNPCKLPD

vunpcklpd xmm1, xmm2, xmm3/m128

vunpcklpd ymm1, ymm2, ymm3/m256

Przepisuje liczby rzeczywiste podwójnej precyzji. Młodsze liczby ze 128-bitowych części rejestru xmm2/ymm2 i xmm3/m128 / ymm3/m256 są zapisywane **z przeplotem** jako **liczby rzeczywiste podwójnej precyzji** do rejestru xmm1/ymm1.



Bity od 128/256 do MSB są zerowane.

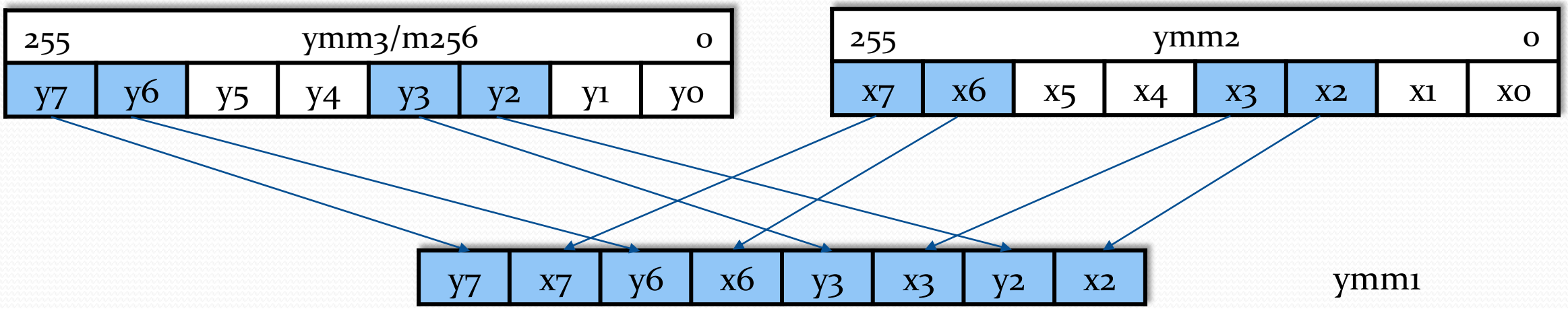
Instrukcja dekompresji

VUNPCKHPS

vunpckhps xmm1, xmm2, xmm3/m128

vunpckhps ymm1, ymm2, ymm3/m256

Przepisuje liczby rzeczywiste pojedynczej precyzji. Starsze dwie liczby ze 128-bitowych części rejestru xmm2/ymm2 i xmm3/m128 / ymm3/m256 są zapisywane **z przeplotem** jako **liczby rzeczywiste pojedynczej precyzji** do rejestru celu xmm1/ymm1.



Bity od 128/256 do MSB są zerowane.

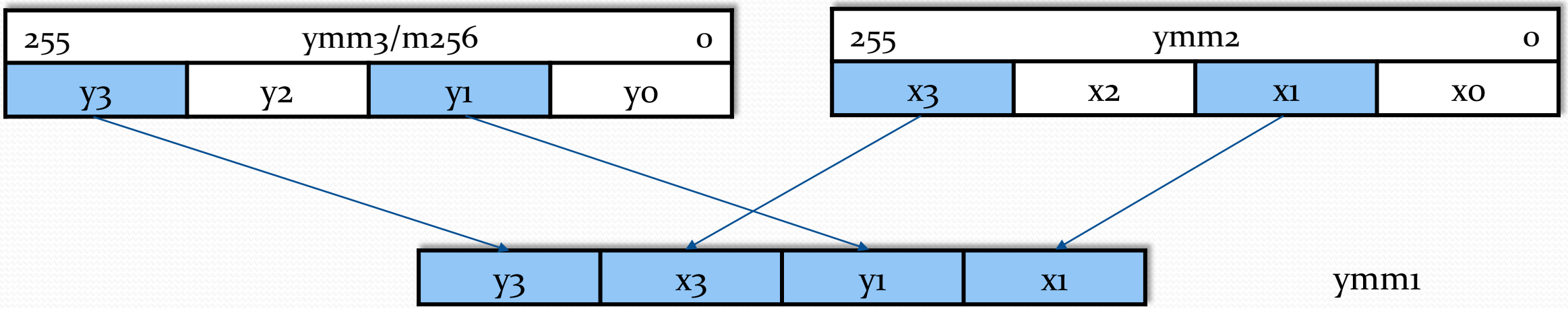
Instrukcja dekompresji

VUNPCKHPD

vunpckhpd xmm1, xmm2, xmm3/m128

vunpckhpd ymm1, ymm2, ymm3/m256

Przepisuje liczby rzeczywiste podwójnej precyzji. Starsze liczby ze 128-bitowych części rejestru xmm2/ymm2 i xmm3/m128 / ymm3/m256 są zapisywane **z przeplotem** jako **liczby rzeczywiste podwójnej precyzji** do rejestru celu xmm1/ymm1.



Bity od 128/256 do MSB są zerowane.

Instrukcje wstawiania

Instrukcja wstawiania

VINSERTPS (1/3)

vinsertps xmm1, xmm2, xmm3/m32, imm8

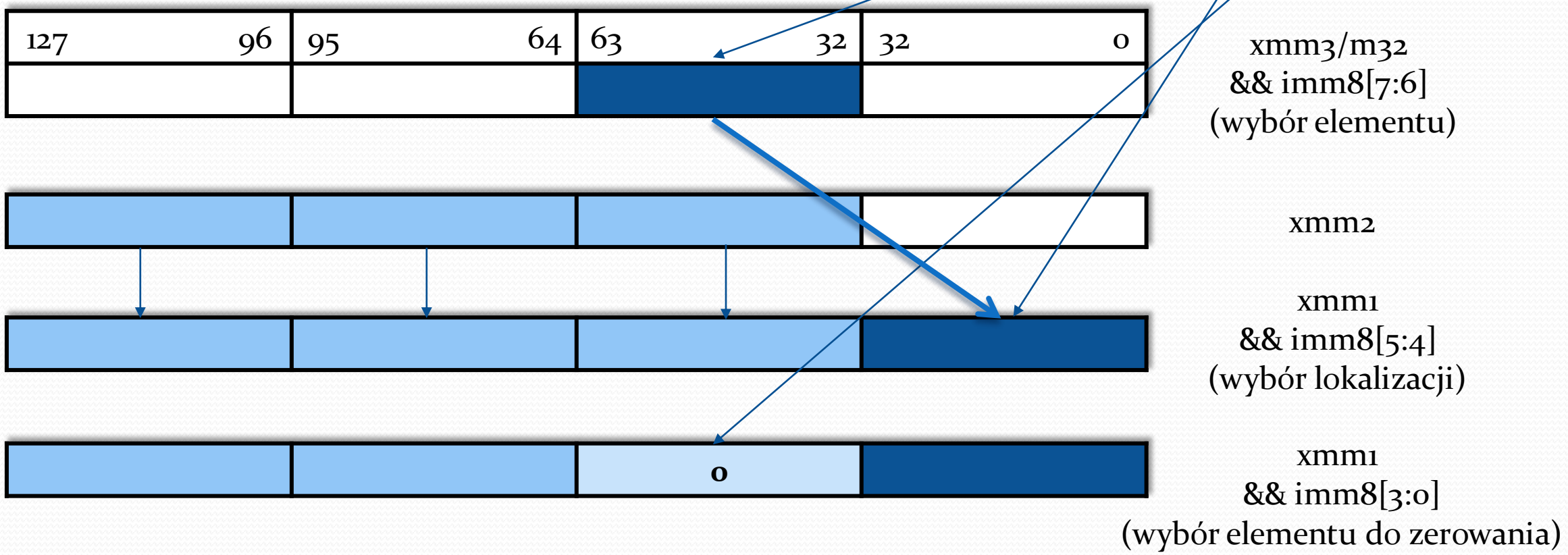
1. Kopiuje zawartość xmm2 do xmm1 oraz

2. Kopiuje element o indeksie imm8[7:6] z xmm3/m32
   do rejestru xmm1 pod index imm8[5:4]

3. Zeruje elementy wektora xmm1 jeśli odpowiadające im bity imm8[3:0] są równe 1

Instrukcja wstawiania

VINSERTPS (3/3)



Bity od 128/256 do MSB są zerowane.

Instrukcja wstawiania

VINSERTF128

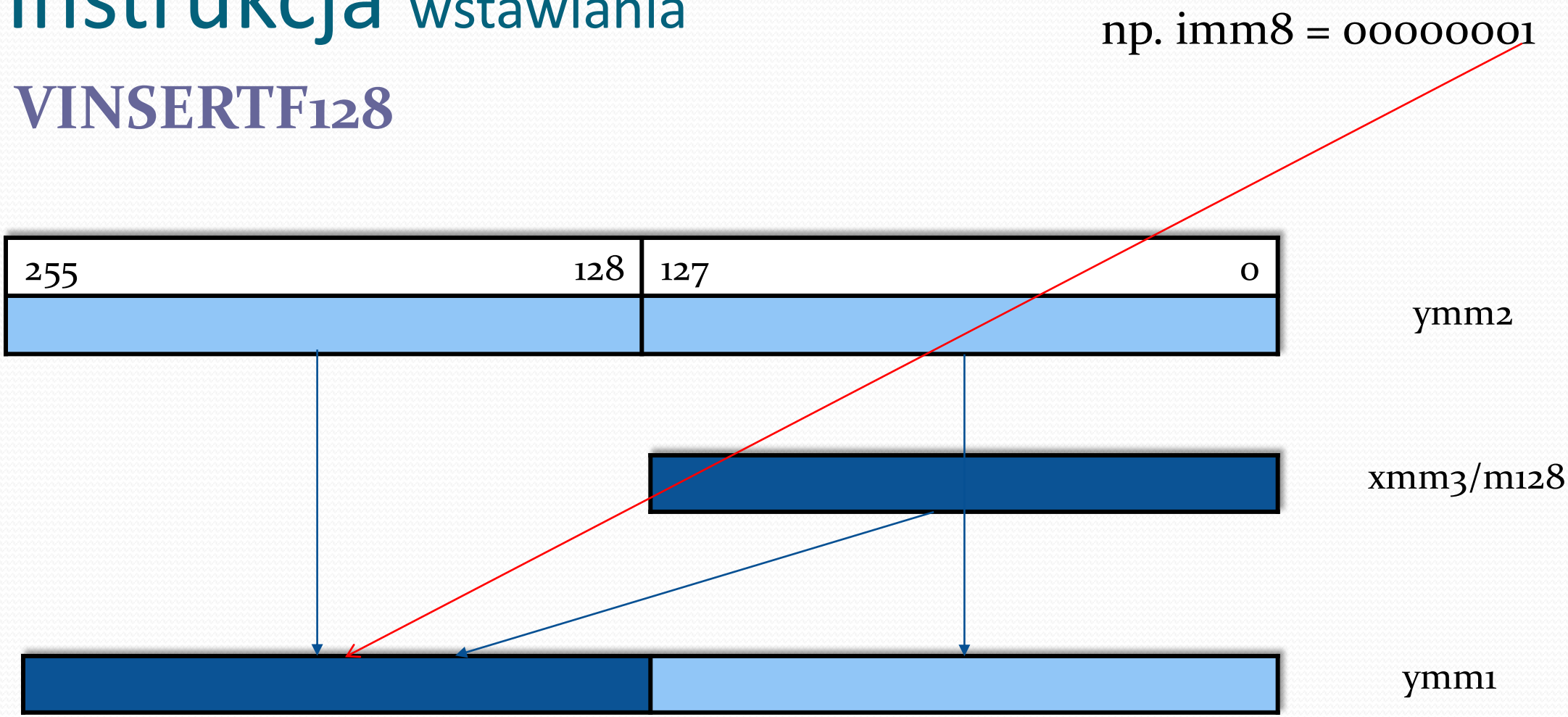
vinsertf128 ymm1, ymm2, xmm3/m128, imm8

Kopiuje połowę rejestru ymm2 oraz cały rejestr xmm3/m128 liczb rzeczywistych zależnie od najmłodszego bitu bajtu sterującego imm8[0].

```
if imm8[0] == 0 then
        ymm1 = ymm2
        ymm1[0] =  xmm3/m128
else
        ymm1 = ymm2
        ymm1[1] =  xmm3/m128
```

Instrukcja wstawiania

VINSERTF128



Instrukcje wybierania

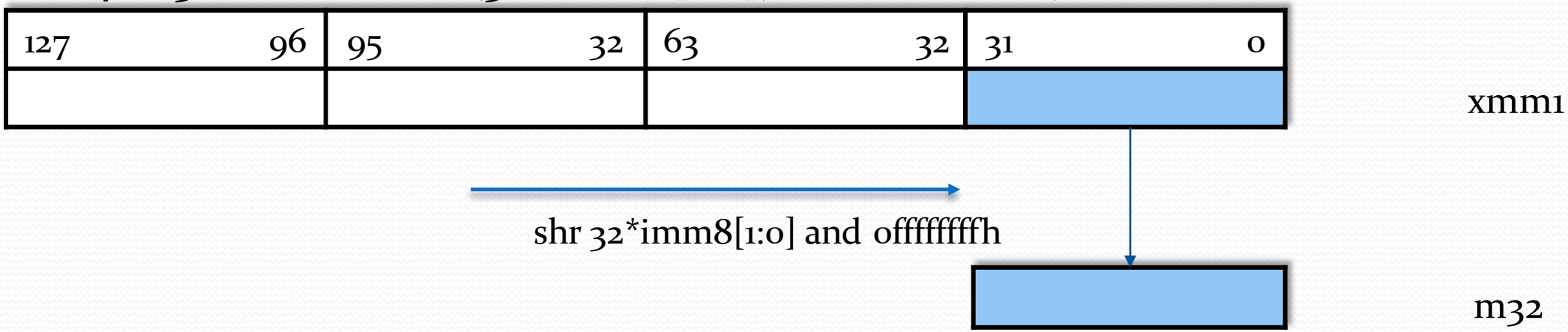
Instrukcja wybierania

VEXTRACTPS

vextractps reg/m32, xmm1, imm8

Wybiera z rejestru xmm1, liczbę rzeczywistą pojedynczej precyzji w oparciu o dwubitową wartość imm8[1:0] stanowiącą offset (wielokrotność przesunięcia bitowego) i **przesyła do rejestru ogólnego przeznaczenia**, (jeśli rejestr ma 64 bity. Wówczas starsza jego część jest zerowana) lub do pamięci.

m64/m32 = xmm1 >> (32*imm8[1:0]) and ffffffffh)



Instrukcja wybierania

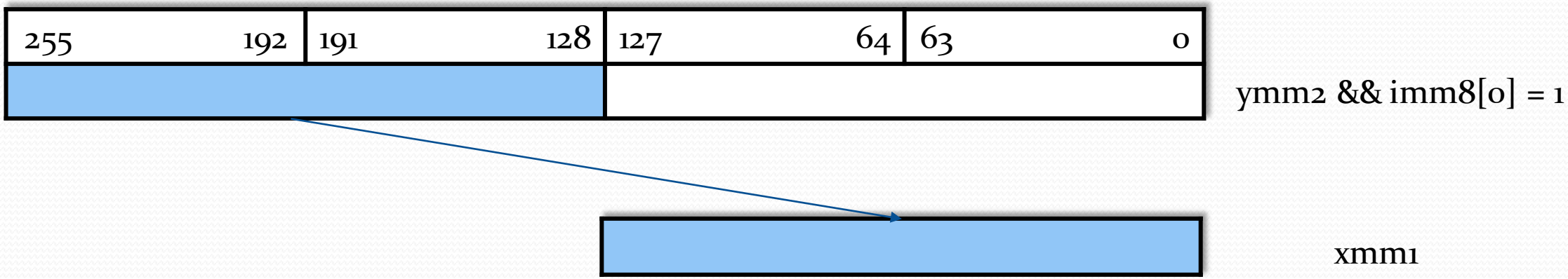
VEXTRACTF128

vextractf128 xmm1/m128, ymm2, imm8

Przesyła połowę rejestru (128 bitów) ymm2 liczb rzeczywistych pojedynczej lub podwójnej precyzji do rejestru xmm1 lub do pamięci m128 według bajtu sterującego imm8.

if imm8[0] = 0 then xmm1/m128 = ymm2[127:0]

else xmm1/m128 = ymm2[255:128]



Operacje przesłania AVX cd.

- Instrukcje mieszające: VBLENDP[S/D], VBLENDVP[S/D]
- Instrukcje rozgłaszania: VBROADCASTS[S/D], VBROADCASTF128
- Instrukcje zbierania: VGATHER[D/Q]P[S/D]
- Instrukcje permutacji: VPERMP[S/D], VPERMILP[S/D], VPREM2F128
- Instrukcje tasowania: VSHUFP[S/D]

Instrukcje mieszające

Instrukcja mieszająca

VBLENDP[S/D]

vblendp[s/d] xmm1, xmm2, xmm3/m128, imm8

vblendp[s/d] ymm1, ymm2, ymm3/m256, imm8

Wybiera komplementarnie elementy wektorów liczb rzeczywistych pojedynczej/podwójnej precyzji, z xmm3/ymm3 lub m128/m256 oraz xmm2/ymm2 według bajtu sterującego imm8, wynik zapisuje w xmm1/ymm1. Kolejne bity imm8 odpowiadają kolejnym 32/64 bitom rejestrów/pamięci i pełnią zadanie przełącznika.

i <0, 7> dla PS lub i <0, 3> dla PD

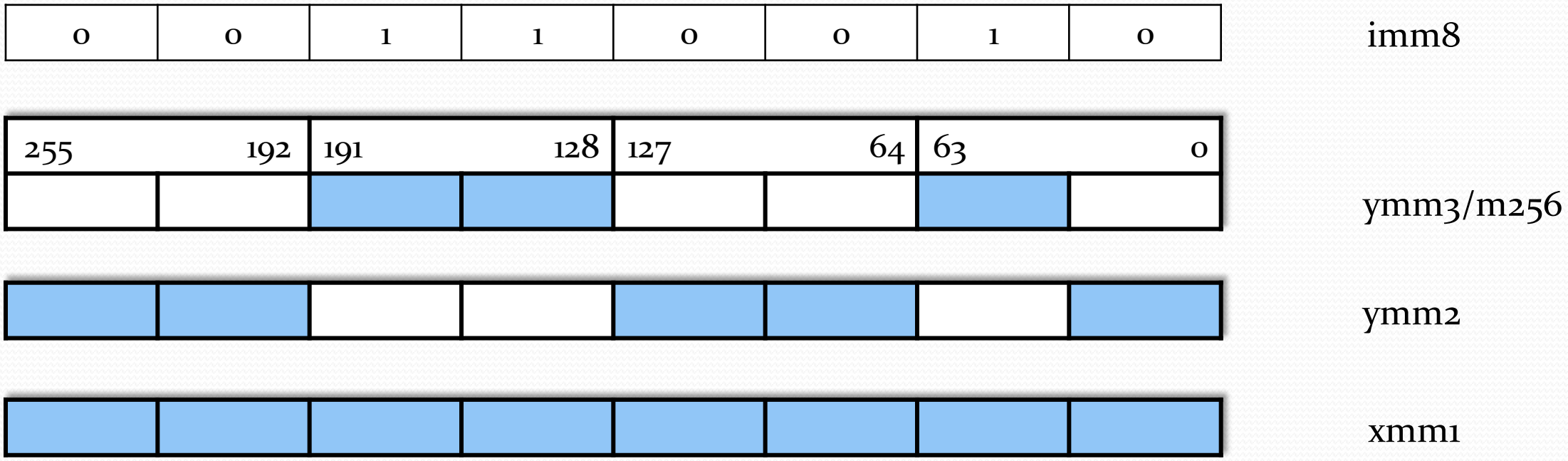
if imm8[i] = 0 then xmm1/ymm1[i] = xmm2/ymm2[i]

else xmm1/ymm1[i] = xmm3/ymm3[i] lub m128/m256[i]

Bity od 128/256 do MSB są zerowane.

Instrukcja mieszająca

VBLENDPS



Instrukcja mieszająca

VBLENDVP[S/D]

vblendvp[s/d] xmm1, xmm2, xmm3/m128, xmm4

vblendvp[s/d] ymm1, ymm2, ymm3/m256, ymm4

Warunkowo kopiuje elementy wektorów liczb rzeczywistych pojedynczej/podwójnej precyzji, z xmm3/ymm3 lub m128/m256 oraz xmm2/ymm2 według maski rejestru xmm4/ymm4, wynik zapisuje w xmm1/ymm1. Maską są odpowiadające poszczególnym wektorom bity znaku xmm4/ymm4.

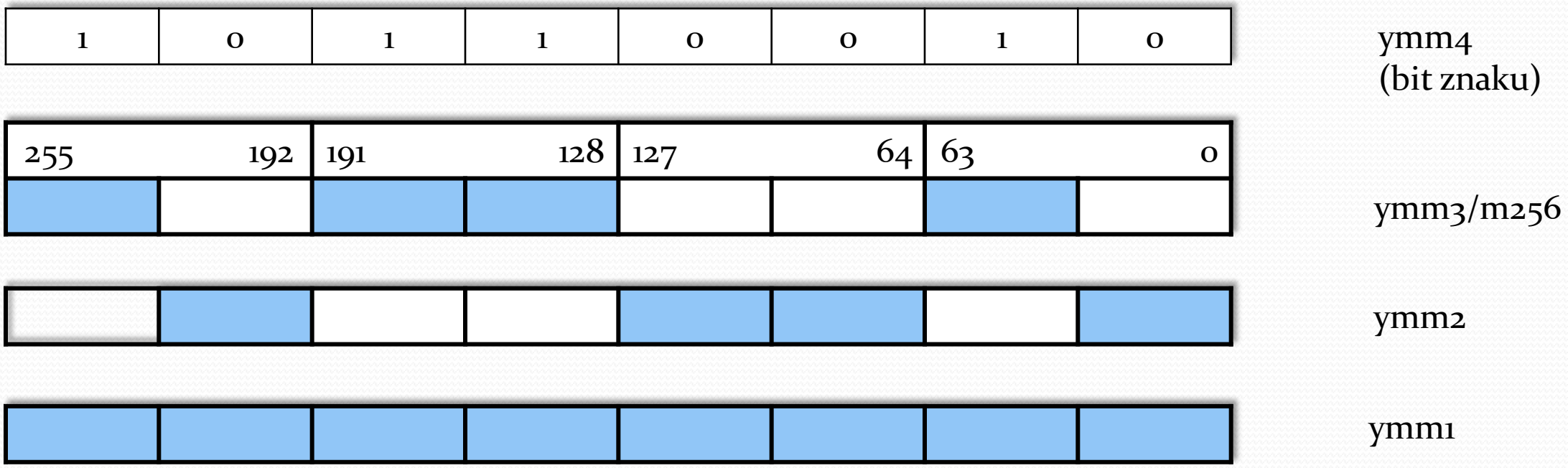
```
if xmm4/ymm4[i][31/63] = 0
        then xmm1/ymm1[i] = xmm2/ymm2[i]
else
        xmm1/ymm1[i] = xmm3/ymm3[i] lub m128/m256[i]
```

Bity od 128/256 do MSB są zerowane.

Instrukcja mieszająca

VBLENDVPS

Maska w rejestrze ymm4



Instrukcje rozgłaszania

Instrukcja rozgłaszania

VBROADCASTS[S/D] / VBROADCASTF128

vbroadcastss xmm1, m32/xmm2

vbroadcastss ymm1, m32/xmm2

Przesyła liczbę rzeczywistą pojedynczej precyzji z pamięci m32 lub najmłodszą z rejestru xmm2 do całego rejestru celu xmm1/ymm1.

vbroadcastsd ymm1, m64

vbroadcastsd ymm1, xmm2

Przesyła liczbę rzeczywistą podwójnej precyzji z pamięci m64 lub najmłodszą z rejestru xmm2 do całego rejestru celu xmm1/ymm1.

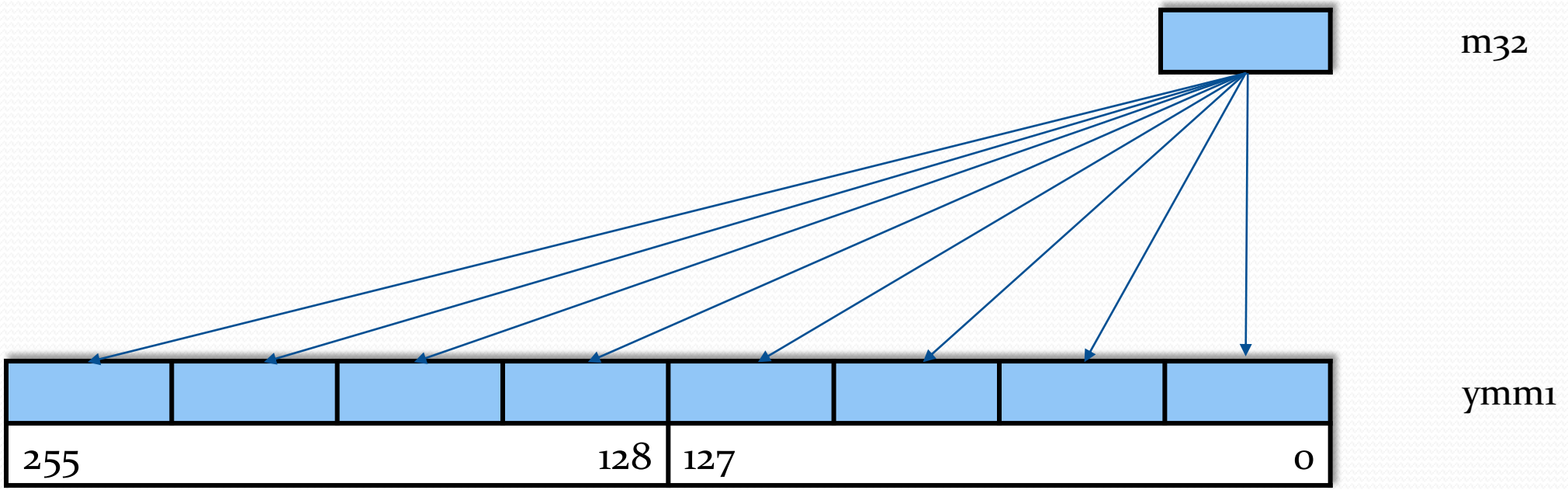
vbroadcastf128 ymm1, m128

Przesyła zawartość pamięci m128 do całego rejestru celu ymm1.

Bity od 128/256 do MSB są zerowane.

Instrukcja rozgłaszania

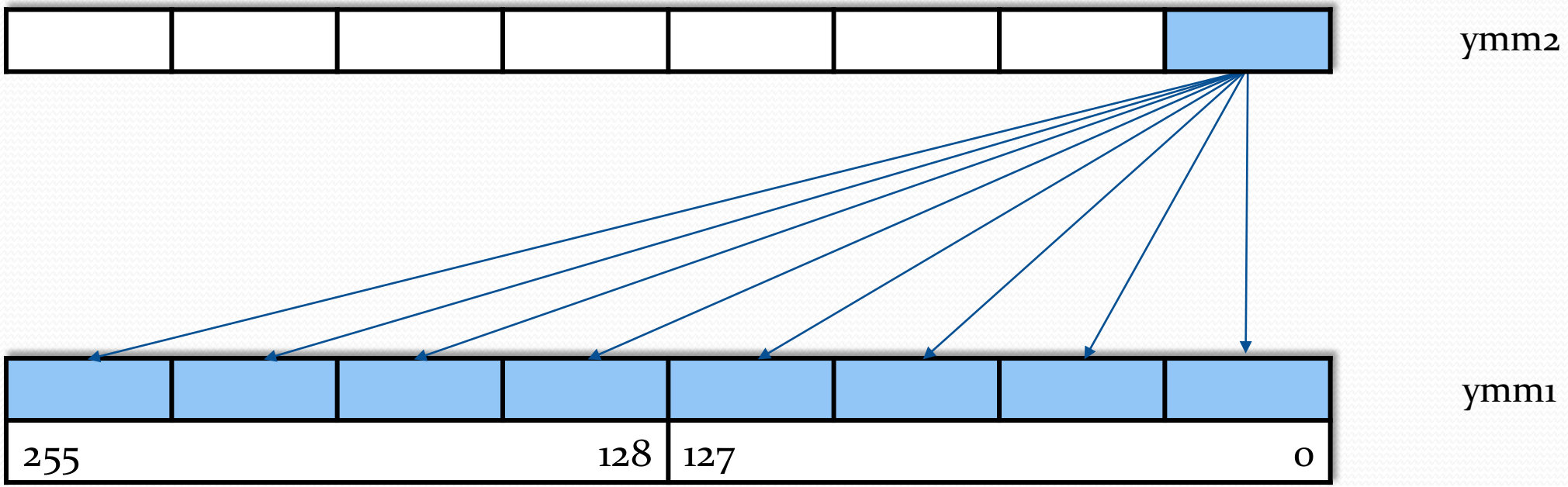
VBRODCASTSS



Bity od 128/256 do MSB są zerowane.

Instrukcja rozgłaszania

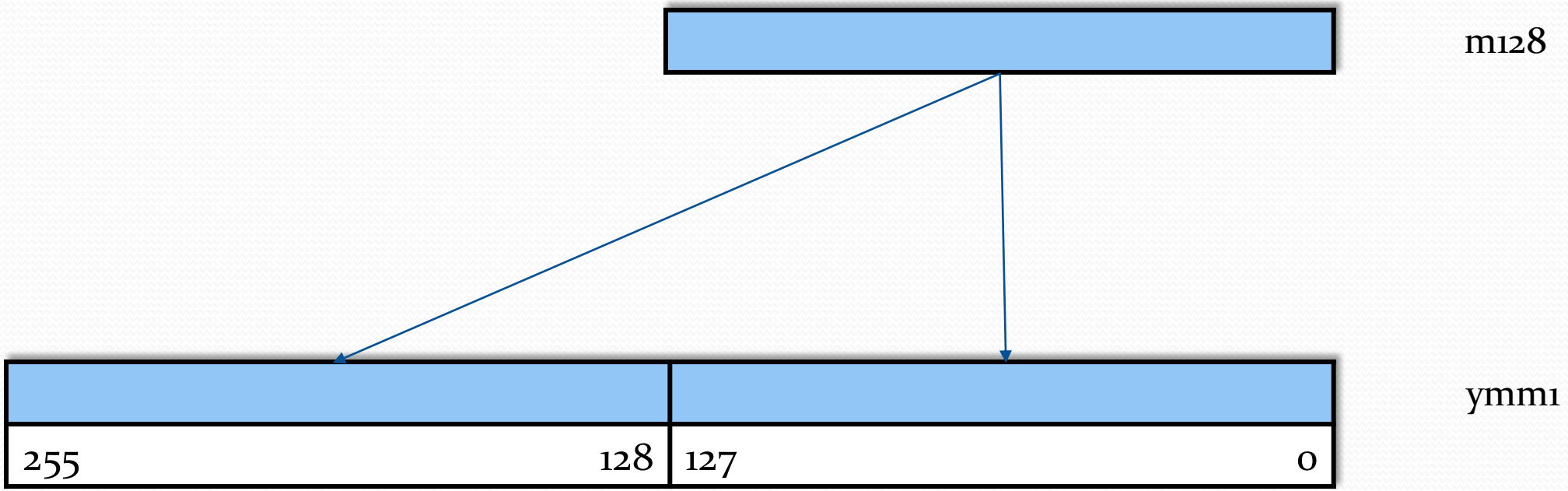
VBRODCASTSS



Bity od 128/256 do MSB są zerowane.

Instrukcja rozgłaszania

VBRODCASTF128



Bity od 128/256 do MSB są zerowane.

Instrukcje zbierania

Instrukcja zbierania

VGATHER[D/Q]P[S/D]

Dotyczy typów danych

Dotyczy adresów

vgather[d/q]p[s/d] xmm1, vm[32/64]x, xmm3 (AVX2)
vgather[d/q]p[s/d] ymm1, vm[32/64]y, ymm3 (AVX2)

Instrukcja kompletuje wektor xmm1/ymm1 używając **adresów w postaci** podwójnych/poczwórnych słów zdefiniowanych w vm32[x/y]/vm64[x/y] używając jako indeksów podwójnych/poczwórnych słów zapisanych w xmm2/ymm2 do wskazanej lokalizacji pamięci, skąd pobierane są liczby rzeczywiste pojedynczej/podwójnej precyzji.

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.

Instrukcja zbierania

VGATHER[D/Q]P[S/D]

vgather[d/q]p[s/d] xmm1, vm[32/64]x, xmm3 (AVX2)
vgather[d/q]p[s/d] ymm1, vm[32/64]y, ymm3 (AVX2)

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

adres_bazowy – adres danych, określa rejestr GPR ma zostać użyty

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

Bity od 128/256 do MSB są zerowane.

Instrukcja zbierania

VGATHER[D/Q]P[S/D]

vgather[d/q]p[s/d] xmm1, vm[32/64]x, xmm3 (AVX2)
vgather[d/q]p[s/d] ymm1, vm[32/64]y, ymm3 (AVX2)

Adresowanie cd.

W opisie instrukcji vm32x wskazuje wektor czterech 32-bitowych indeksów zapisanych w xmm2, vm32y wektor ośmiu 32- bitowych indeksów dla ymm2.

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

Bity od 128/256 do MSB są zerowane.

Instrukcja zbierania

VGATHER[D/Q]P[S/D]

vgather[d/q]p[s/d] xmm1, vm[32/64]x, xmm3 (AVX2)
vgather[d/q]p[s/d] ymm1, vm[32/64]y, ymm3 (AVX2)

Działanie instrukcji gather:

Pobiera z pamięci o wskazanej lokalizacji określonej tu jako adres_fizyczny liczby pojedynczej/podwójnej precyzji 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])]

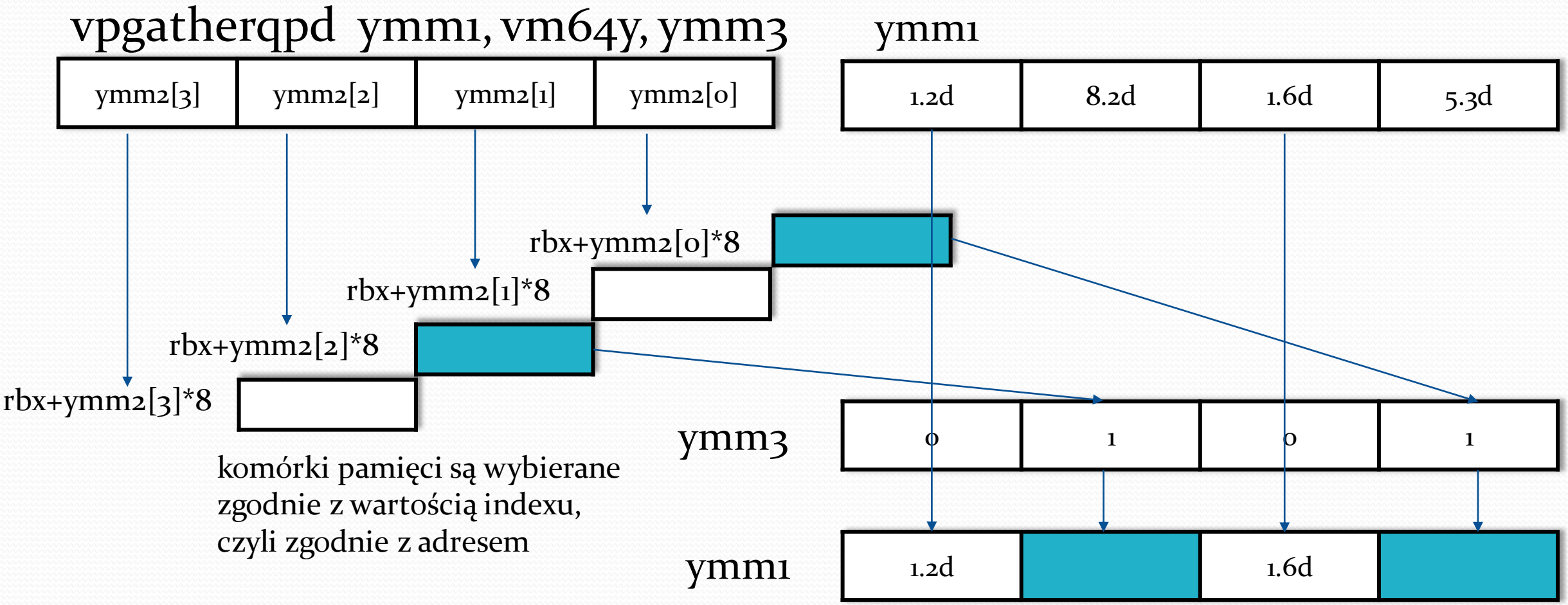
if ymm3[i][63/31] then ymm1[i] ← [adres_fizyczny(ymm2[i])]

Bity od 128/256 do MSB są zerowane.

Instrukcja zbierania (przykład) AVX2 VPGATHERQPD

vpgatherqpd ymm1, [rbx+ymm2*8], ymm3

vm64y = [rbx+ymm2*8]



Instrukcje permutacji

Instrukcja permutacji

VPERMP[S/D]

vpermps ymm1, ymm2, ymm3/m256

Wybiera liczby rzeczywiste pojedynczej precyzji z ymm3/m256 według wskazań ymm2 (trzy najmłodsze bity każdego elementu wektora stanowią indeks), wynik zapisuje w ymm1.

ymm1[i] = ymm3/m256[ymm2[i][2:0]]

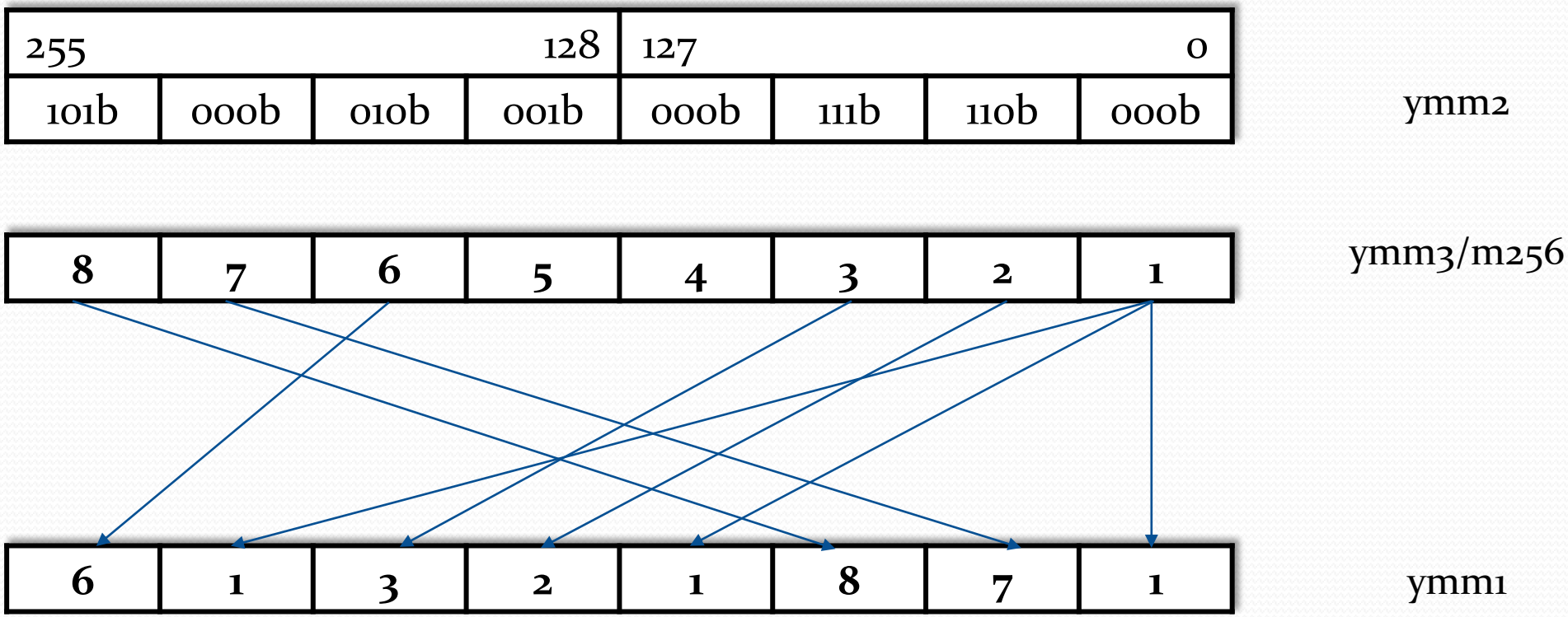
vpermpd ymm1, ymm2/m256, imm8

Wybiera liczby rzeczywiste podwójnej precyzji z ymm2/m256 według bajtu sterującego imm8, (kolejne dwa bity), wynik zapisuje w ymm1.

ymm1[i] = ymm2/m256[imm8[2i+1:2i]]

Bity od 128/256 do MSB są zerowane.

Instrukcja permutacji VPERMPS

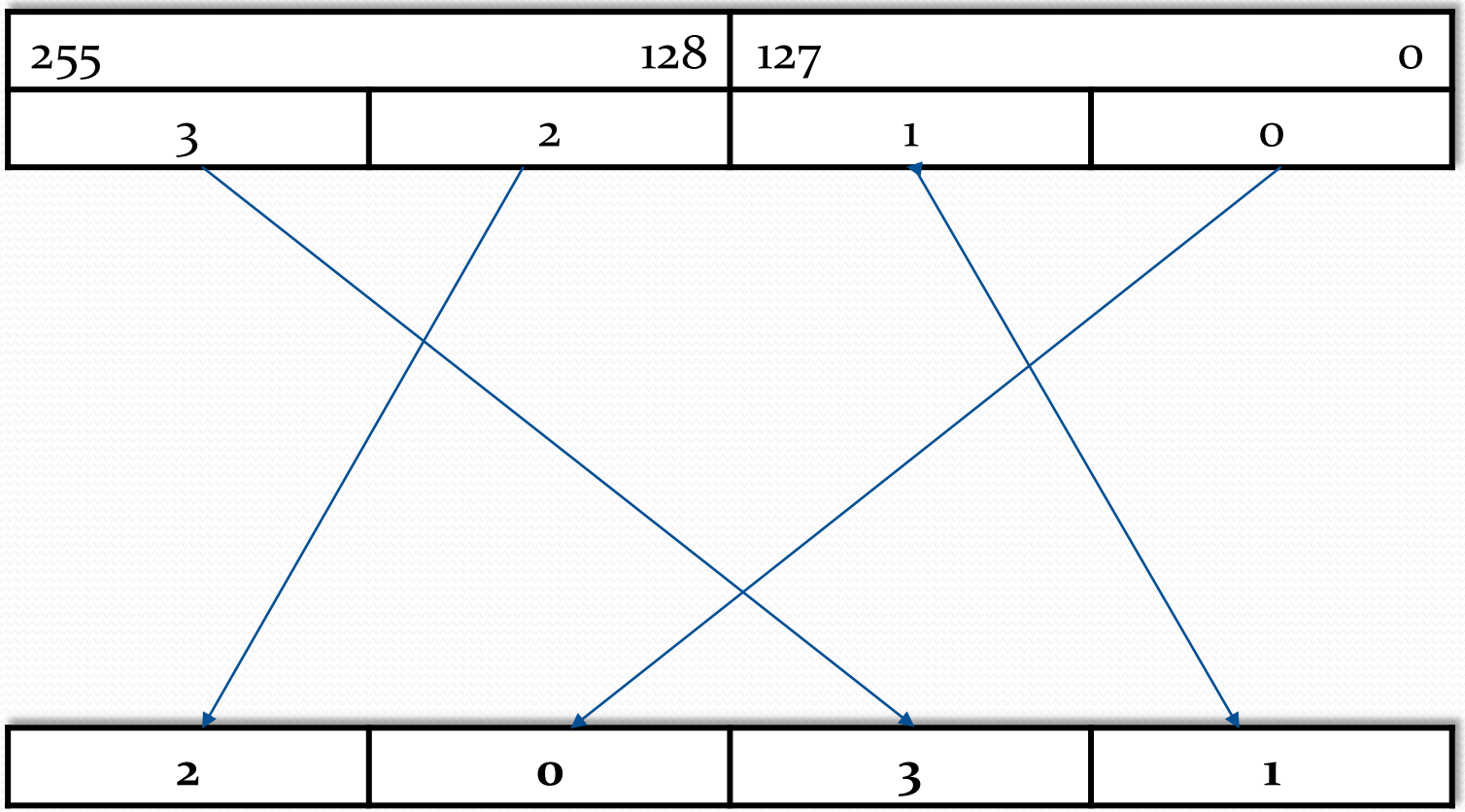


Bity od 128/256 do MSB są zerowane.

Instrukcja permutacji VPERMPD

np. imm8 = 10 00 11 01

Indeks dla kolejnych elementów wektora licząc od elementu zerowego



ymm2

ymm1

Przykład dla imm8 = 10001101 zamienia wektor 0123 na wektor 1302

Instrukcja permutacji

VPERMILPS

vpermilps xmm1, xmm2, xmm3/m128
vpermilps xmm1, xmm2/m128, imm8
vpermilps ymm1, ymm2, ymm3/m256
vpermilps ymm1, ymm2/m256, imm8

Wybiera liczby rzeczywiste pojedynczej precyzji z xmm2/ymm2 (/m128 /m256) według wskazania odpowiadających dwóch najmłodszych bitów xmm3/m128 lub odpowiednich bitów imm8, wynik zapisuje w xmm1/ymm1.

xmm1[i] = xmm2[xmm3/m128[i][1:0]] lub xmm2[imm8[2i+1:2i]]
if i>3 ymm1[i] = ymm2[ymm3/m256[i][1:0]] lub ymm2[4+imm8[2i-7:2i-8]]

Bity od 128/256 do MSB są zerowane.

Instrukcja permutacji

VPERMILPD

vpermilpd xmm1, xmm2, xmm3/m128
vpermilpd xmm1, xmm2/m128, imm8
vpermilpd ymm1, ymm2, ymm3/m256
vpermilpd ymm1, ymm2/m256, imm8

Wybiera liczby rzeczywiste podwójnej precyzji z xmm2/ymm2 (/m128 /m256) według wskazania każdego **drugiego bitu(o indeksie 1)** z xmm3/m128, wynik zapisuje w xmm1/ymm1.

xmm1[i] = xmm2[xmm3/m128[i][1]] lub xmm2[imm8[i]]
if i>1 ymm1[i] = xmm2[2+ymm3/m256[i][1]] lub ymm2[2+imm8[i]]

Bity od 128/256 do MSB są zerowane.

Instrukcja permutacji

VPERM2F128

vperm2f128 ymm1, ymm2, ymm3/m256, imm8

Wybiera odpowiednio 128 bitów liczb rzeczywistych z ymm2 oraz ymm3/m256 według wskazania bajtu sterującego imm8, wynik zapisuje w xmm1/ymm1.

```
if imm8[1:0] = 0 then ymm1[127:0] = ymm2[127:0]
if imm8[1:0] = 1  then ymm1[127:0] = ymm2[255:128]
if imm8[1:0] = 2 then ymm1[127:0] = ymm3/m256[127:0]
if imm8[1:0] = 3 then ymm1[127:0] = ymm3/m256[255:128]

if imm8[5:4] = 0 then ymm1[255:128] = ymm2[127:0]
if imm8[5:4] = 1  then ymm1[255:128] = ymm2[255:128]
if imm8[5:4] = 2 then ymm1[255:128] = ymm3/m256[127:0]
if imm8[5:4] = 3 then ymm1[255:128] = ymm3/m256[255:128]
```

Bity od 128/256 do MSB są zerowane.

imm8[3] => ymm1[127:0] ← 0
imm8[7] => ymm1[255:128] ← 0

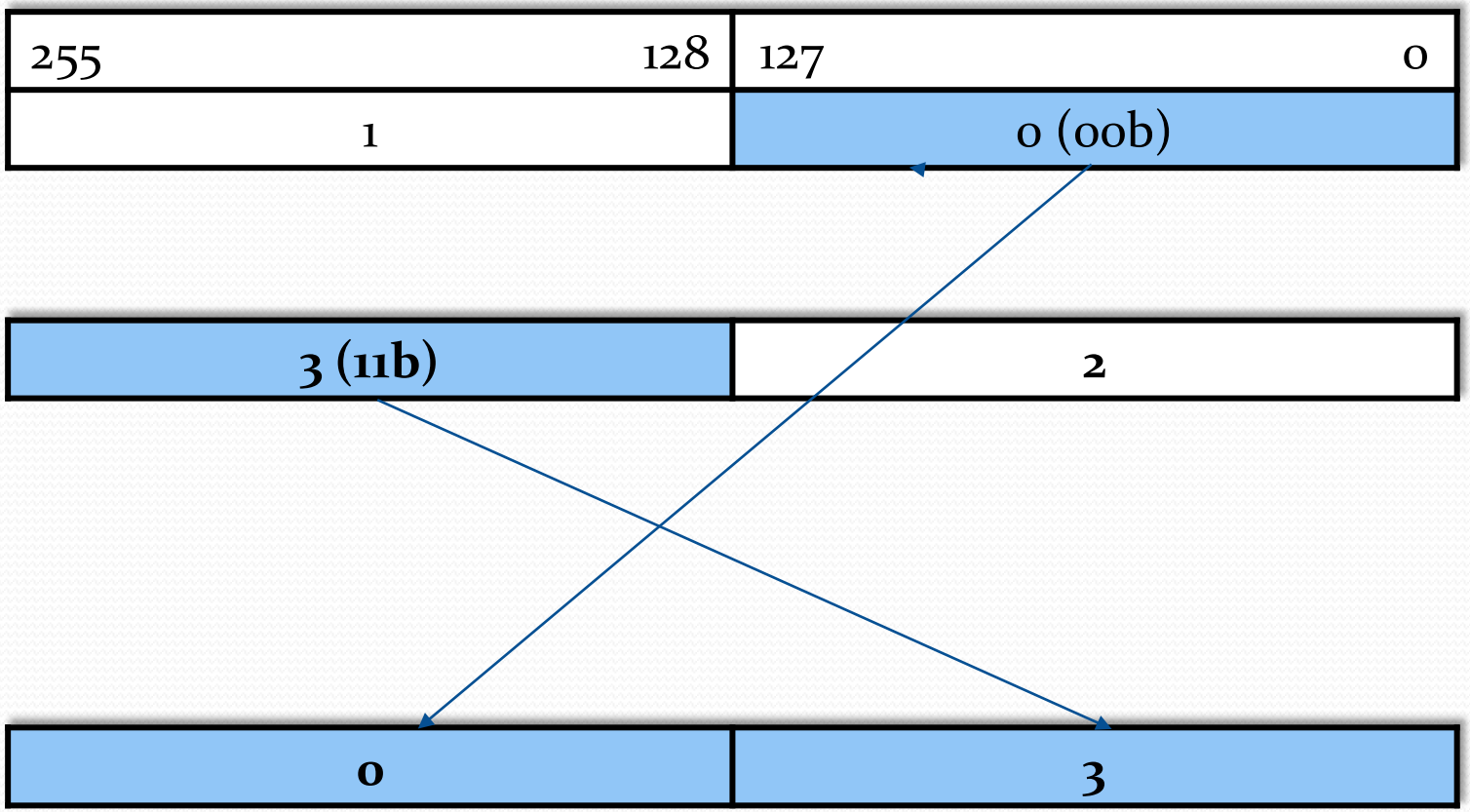
Instrukcja permutacji VPERM2F128

Bit zerowania

Bit zerowania

np. imm8 = **0** 0 **00** **0** 0 **11**

Wartość odpowiada elementowi źródła (3) a usytuowanie (0) elementowi celu



Instrukcje tasowania

Instrukcja tasowania

VSHUFPS

vshufps xmm1, xmm2, xmm3/m256, imm8

vshufps ymm1, ymm2, ymm3/m256, imm8

Wybiera liczby rzeczywiste pojedynczej precyzji z xmm2/ymm2 oraz xmm3/ymm3 lub m128.m256 po dwie z każdego źródła według bajtu sterującego imm8 i zapisuje w xmm1/ymm1 po dwie wartości przeplatając źródła pochodzenia.

if i=(0,1,4,5) then s=2 else s=3

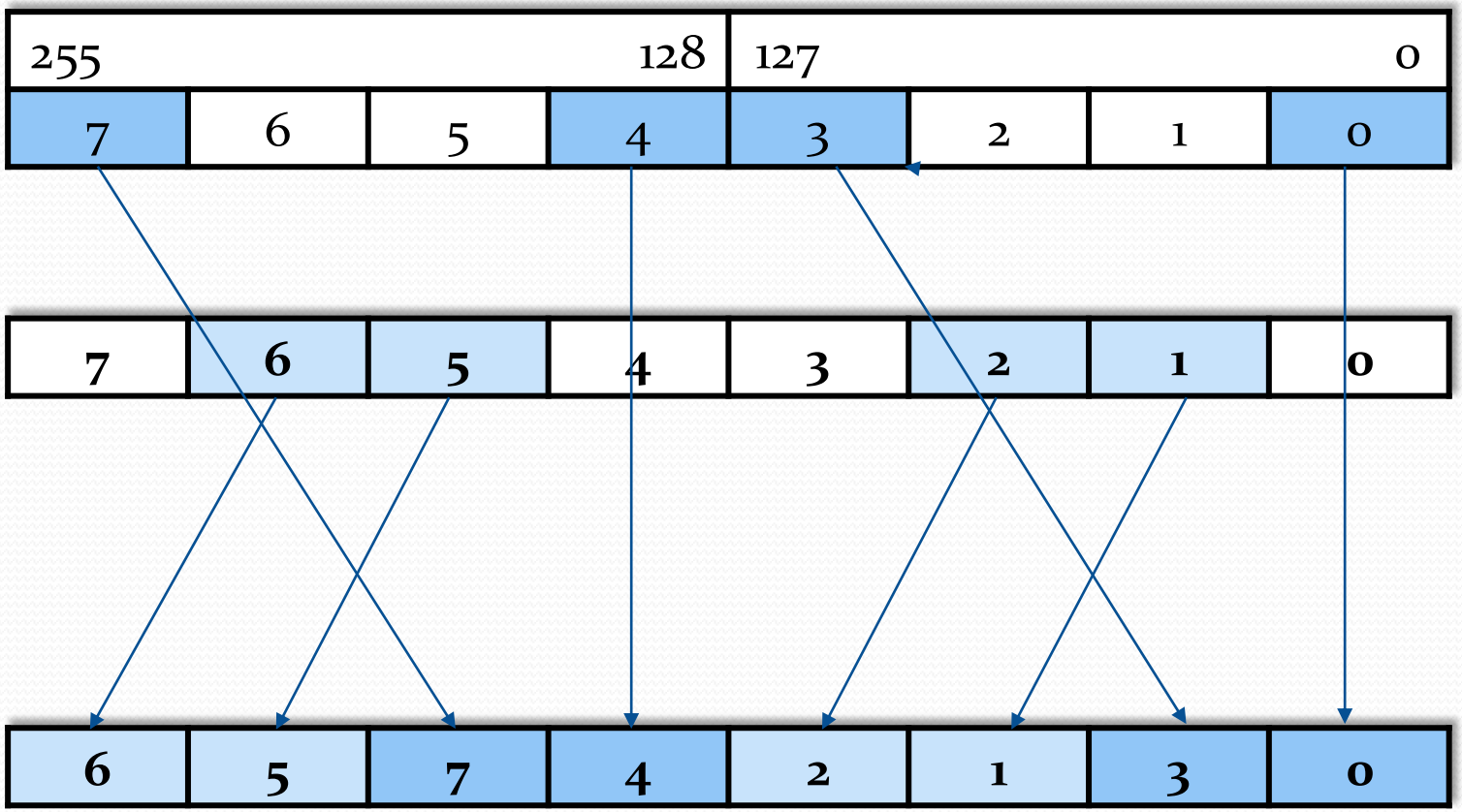
xmm[i] = xmms[imm8[2i+1:2i]] i=(0..3)

ymm[i] = ymms[imm8[2(i-4)+1:2(i-4)]] i=(4..7)

Bity od 128/256 do MSB są zerowane.

Instrukcja tasowania VSHUFPS

np. imm8 = 10 **01 11 00**



Bajt koduje połowę rejestru, drugą powiela według tej samej sekwencji.

Wartości [3:2],[1:0] określają elementy źródła 1, wartości [7:6],[5:4] określają elementy źródła 2, usytuowanie dwóch bitów determinuje element celu.

Instrukcja tasowania

VSHUFPD

vshufpd ymm1, ymm2, ymm3/m256, imm8

Tasuje liczby rzeczywiste podwójnej precyzji z xmm2/ymm2 oraz xmm3/ymm3 lub m128 według bajtu sterującego imm8. Wynik zapisuje w xmm1/ymm1 przeplatając źródła pochodzenia.

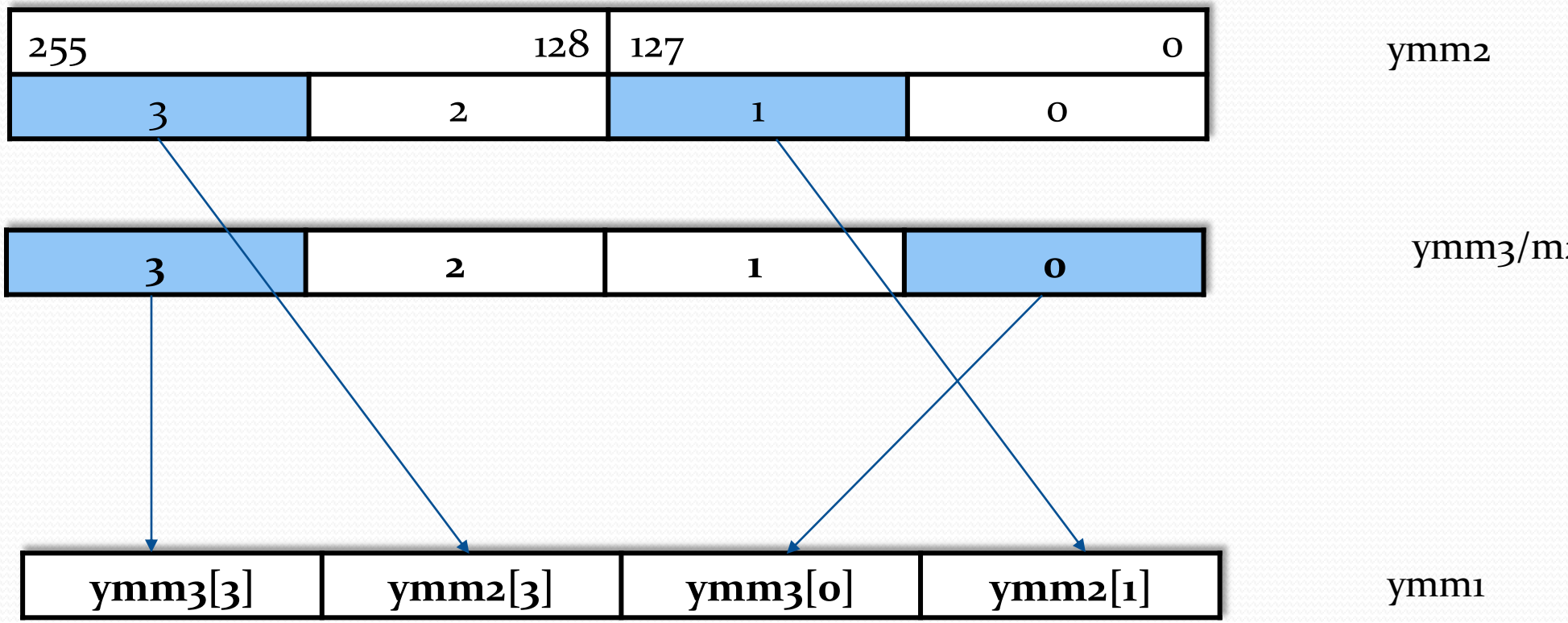
```
xmm1[0] = xmm2[imm8[0]]
xmm1[1] = xmm3[imm8[1]]
ymm1[2] = ymm2[2+imm8[2]]
ymm1[3] = ymm3[2+imm8[3]]
```

Bity od 128/256 do MSB są zerowane.

Instrukcja tasowania

VSHUFPD

np. imm8 = 0000 1 1 0 1



Operacje arytmetyczne AVX

Operacje arytmetyczne AVX

- Instrukcje dodawania: VADDS[S/D], VADDP[S/D], VHADDP[S/D]
- Instrukcje odejmowania: VSUBS[S/D], VSUBP[S/D], VHSUBP[S/D]
- Instrukcje dodawania i odejmowania: VADDSUBP[S/D]
- Instrukcje mnożenia: VMULS[S/D], VMULP[S/D]
- Instrukcje mnożenia z sekwencyjnym dodawaniem: VDPP[S/D]
- Instrukcje dzielenia: VDIVS[S/D], VDIVP[S/D]

Instrukcje dodawania

Instrukcja dodawania
VADDS[S/D], VADDP[S/D]

vadds[s/d] xmm1, xmm2, xmm3/m32/m64

vaddp[s/d] xmm1, xmm2, xmm3/m128

vaddp[s/d] ymm1, ymm2, ymm3/m256

Dodaje skalary/wektory liczb rzeczywistych pojedynczej/podwójnej precyzji z rejestru xmm2/ymm2 i xmm3/ymm3 lub m32/m64/m128/m256, wynik zapisuje w xmm1/ymm1. Dla skalarów pozostałe elementy są przepisywane ze źródła 1.

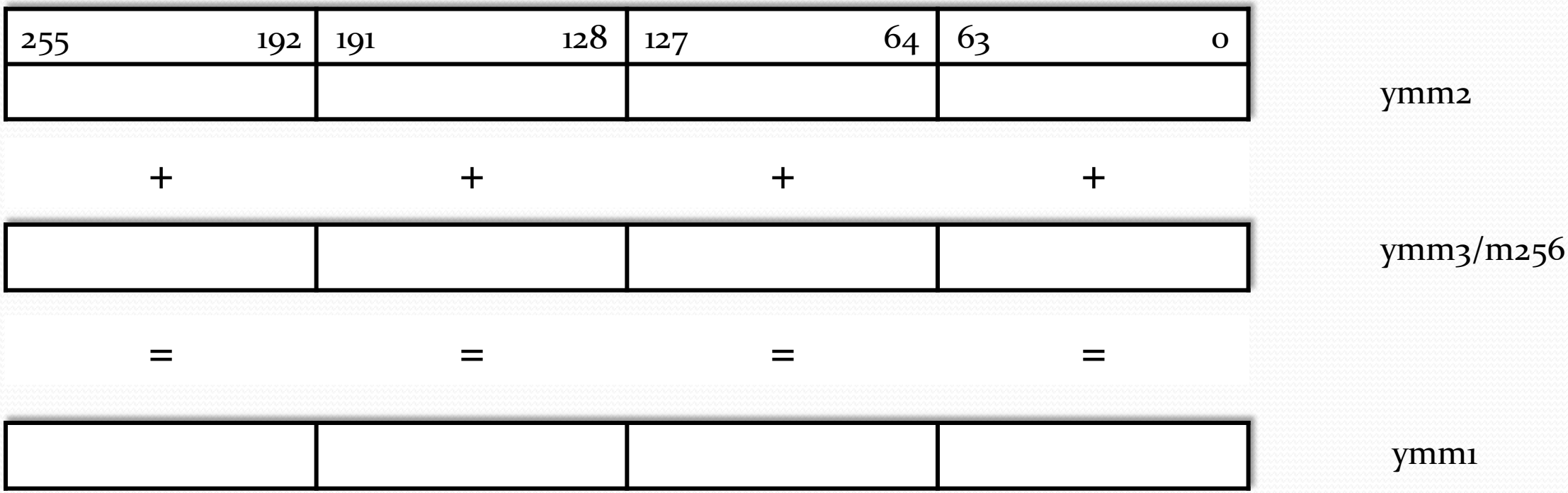
xmm1[i] = xmm2[i] + xmm3/m128[i]

ymm1[i] = ymm2[i] + ymm3/m256[i]

Bity od 128/256 do MSB są zerowane.

Instrukcja dodawania

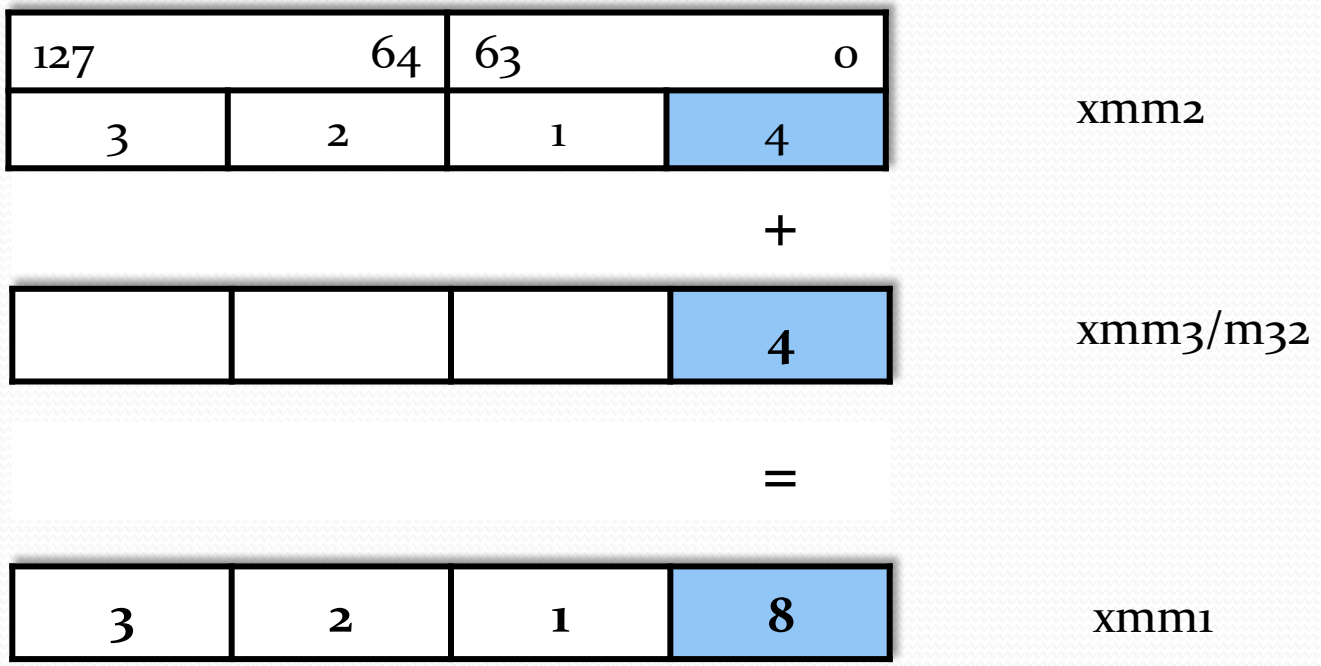
VADDPD



Bity od 128/256 do MSB są zerowane.

Instrukcja dodawania

VADDSS



Bity od 128/256 do MSB są zerowane.

Instrukcja dodawania

VHADDPS

vhaddps xmm1, xmm2, xmm3/m128

vhaddps ymm1, ymm2, ymm3/m256

Horyzontalne dodawanie sąsiednich liczb rzeczywistych pojedynczej precyzji i zapisywanie wyniku z przeplotem co 64 bity.

ymm1/xmm1[31:0] = ymm2/xmm2[63:32] + ymm2/xmm2[31:0]
ymm1/xmm1[63:32] = ymm2/xmm2[127:96] + ymm2/xmm2[95:64]
ymm1/xmm1[95:64] = ymm3/xmm3[63:32] + ymm3/xmm3[31:0]
ymm1/xmm1[127:96] = ymm3/xmm3[127:96] + ymm3/xmm3[95:64]

ymm1[159:128] = ymm2[191:160] + ymm2[159:128]
ymm1[191:160] = ymm2[255:224] + ymm2[223:192]
ymm1[223:192] = ymm3[191:160] + ymm3[159:128]
ymm1[255:224] = ymm3[255:224] + ymm3[223:192]

Bity od 128/256 do MSB są zerowane.

Instrukcja dodawania

VHADDPD

vhaddpd xmm1, xmm2, xmm3/m128

vhaddpd ymm1, ymm2, ymm3/m256

Horyzontalne dodawanie sąsiednich liczb rzeczywistych podwójnej precyzji i zapisywanie wyniku z Przeplotem co 64 bity.

xmm1[0] = xmm2[1] + xmm2[0]

xmm1[1] = xmm3[1] + xmm3[0]

ymm1[2] = ymm2[3] + ymm2[2]

ymm1[3] = ymm3[3] + ymm 3[2]

Bity od 128/256 do MSB są zerowane.

Instrukcje odejmowania

Instrukcja odejmowania

VSUBS[S/D], VSUBP[S/D]

vsubs[s/d] xmm1, xmm2, xmm3/m32/m64

vsubp[s/d] xmm1, xmm2, xmm3/m128

vsubp[s/d] ymm1, ymm2, ymm3/m256

Od zawartości rejestru xmm2/ymm2 **odejmuje** liczby rzeczywiste pojedynczej/podwójnej precyzji odpowiednio z xmm3/ymm3 lub m32/m64/m128/m256, wynik zapisuje w xmm1/ymm1. Dla skalarów pozostałe elementy są przepisywane ze źródła 1.

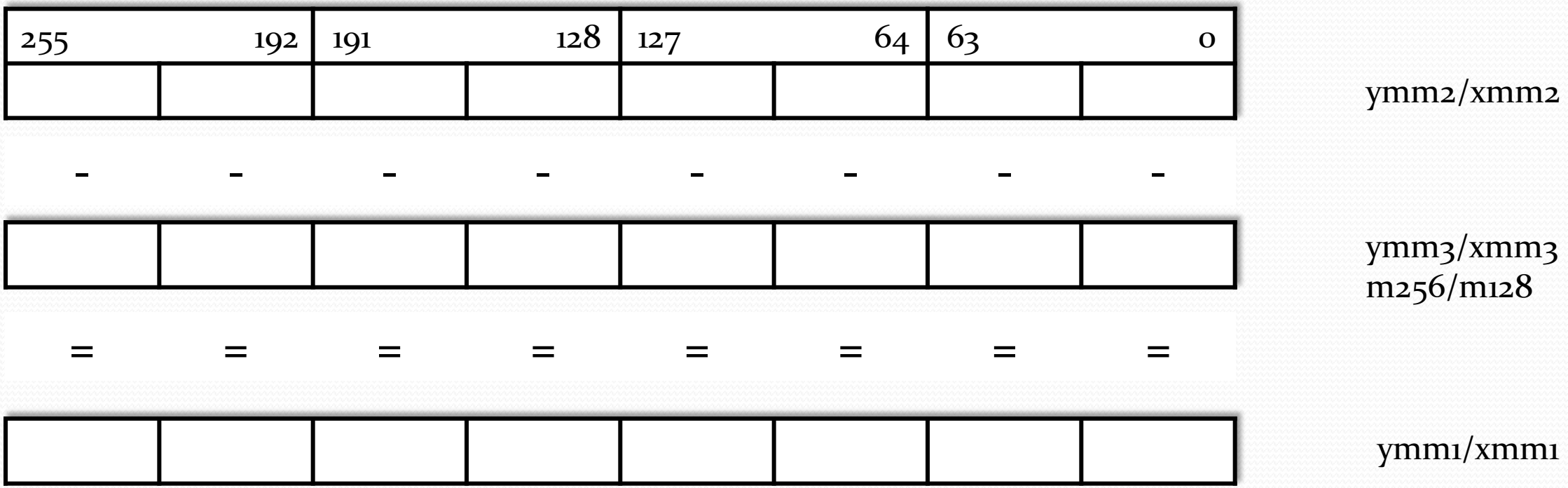
xmm1[i] = xmm2[i] – xmm3/m128[i]

ymm1[i] = ymm2[i] – ymm3/m256[i]

Bity od 128/256 do MSB są zerowane.

Instrukcja odejmowania

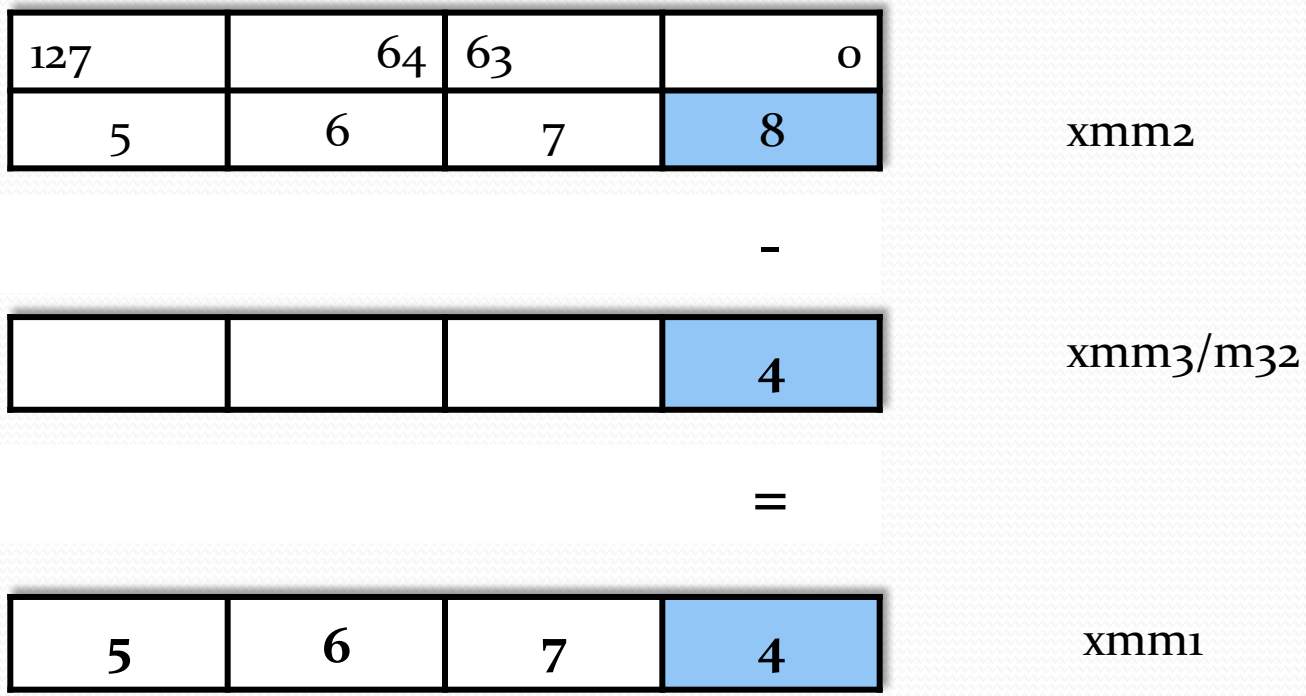
VSUBPS



Bity od 128/256 do MSB są zerowane.

Instrukcja odejmowania

VSUBSS



Bity od 128/256 do MSB są zerowane.

Instrukcja odejmowania horyzontalnego

VHSUBPS

vhsubps xmm1, xmm2, xmm3/m128

vhsubps ymm1, ymm2, ymm3/m256

Horyzontalne odejmowanie sąsiednich liczb rzeczywistych pojedynczej precyzji i zapisywanie wyniku z przeplotem co 64 bity.

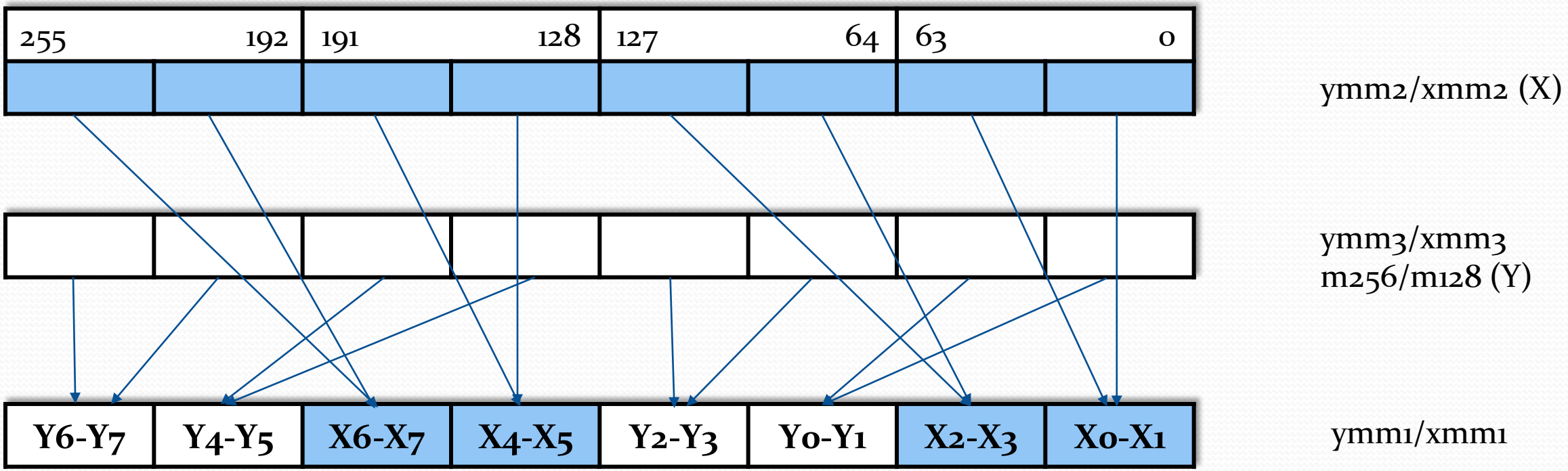
xmm1[0] = xmm2[0] - xmm2[1]
xmm1[1] = xmm2[2] - xmm2[3]
xmm1[2] = xmm3[0] - xmm3[1]
xmm1[3] = xmm3[2] - xmm3[3]

ymm1[4] = ymm2[4] - ymm2[5]
ymm1[5] = ymm2[6] - ymm2[7]
ymm1[6] = ymm3[4] - ymm3[5]
ymm1[7] = ymm3[6] - ymm3[7]

Bity od 128/256 do MSB są zerowane.

Instrukcja odejmowania

VHSUBPS



Bity od 128/256 do MSB są zerowane.

Instrukcja odejmowania horyzontalnego

VHSUBPD

vhsubpd xmm1, xmm2, xmm3/m128

vhsubpd ymm1, ymm2, ymm3/m256

Horyzontalne odejmuje sąsiednie liczby rzeczywiste podwójnej precyzji i zapisuje wynik z przeplotem z przeplotem co 64 bity.

xmm1[0] = xmm2[0] - xmm2[1]

xmm1[1] = xmm3[0] - xmm3[1]

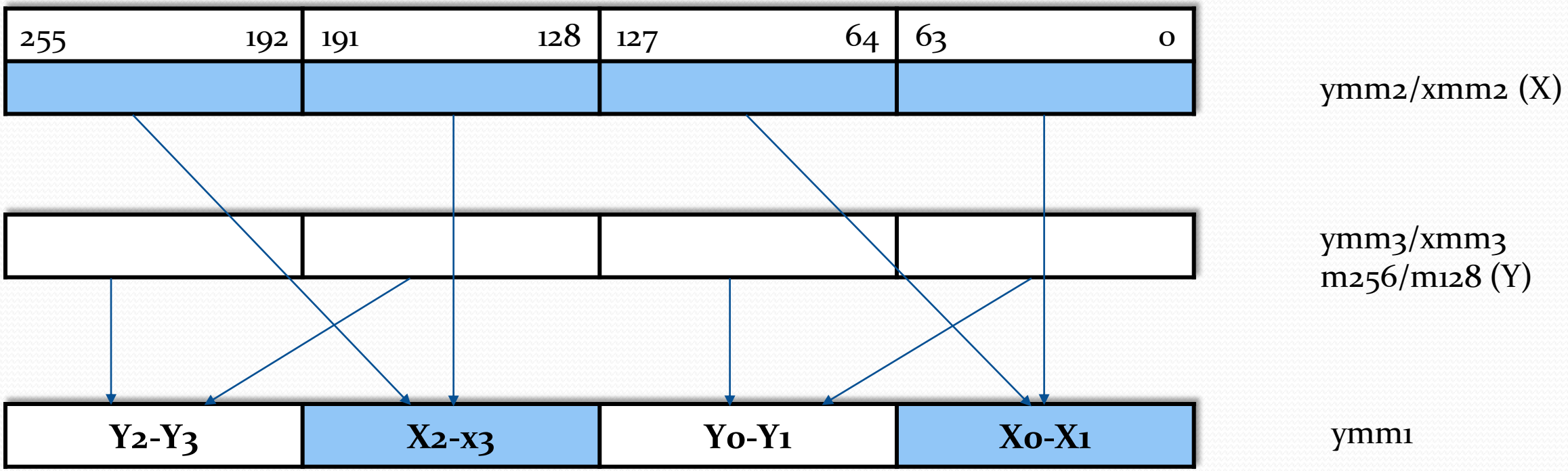
ymm1[2] = ymm2[2] - ymm2[3]

ymm1[3] = ymm3[2] - ymm3[3]

Bity od 128/256 do MSB są zerowane.

Instrukcja dodawania

VHSUBPD



Bity od 128/256 do MSB są zerowane.

Instrukcja odejmowanie macierzy – przykład:

```
void mtx2_avx_sub_float
(float** t1, float** t2, float** t3, int n,
int m){ __asm {
push esi;
push edi;
mov eax, n;

petlaN:
mov esi, t1;
mov esi, dword ptr[esi + eax * 4 - 4];
mov edx, t2;
mov edx, dword ptr[edx + eax * 4 - 4];
mov edi, t3;
mov edi, dword ptr[edi + eax * 4 - 4];
```

```
mov ecx, m;
shl ecx, 2; mnożenie przez 4
petlaM:
sub ecx, 32; ps
vmovups ymm0, ymmword ptr[esi + ecx];
vmovups ymm1, ymmword ptr[edx + ecx];
vsubps ymm2, ymm1, ymm0;
vmovups ymmword ptr[edi + ecx], ymm2;
jnz petlaM;

dec eax;

jnz petlaN;
pop edi;
pop esi;
}
}
```

Instrukcje dodawania i odejmowania

Instrukcja dodawania i odejmowania

VADDSUBP[S/D]

vaddsubp[s/d] xmm1, xmm2, xmm3/m128

vaddsubp[s/d] ymm1, ymm2, ymm3/m256

Naprzemiennie odejmuje i dodaje wektory liczb rzeczywistych pojedynczej/podwójnej precyzji od/do zawartości rejestru xmm2/ymm2 **odejmuje** / **dodaje** odpowiadające wartości xmm3/ymm3 lub m128/m256, wynik zapisuje w xmm1/ymm1.

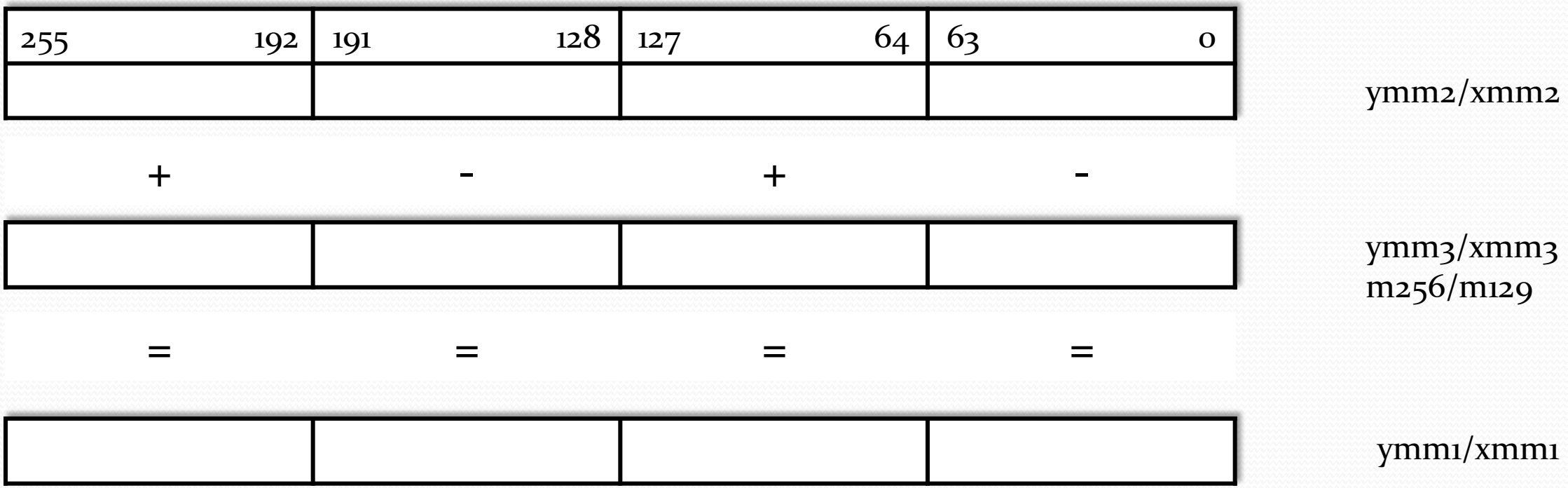
$$ymm1[2i] = ymm2[2i] - ymm3/m256[2i]$$

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

Bity od 128/256 do MSB są zerowane.

Instrukcja dodawania i odejmowania

VADDSUBPD



Bity od 128/256 do MSB są zerowane.

Instrukcje mnożenia

Instrukcja mnożenia

VMULS[S/D], VMULP[S/D]

vmuls[s/d] xmm1, xmm2, xmm3/m32/m64

vmulp[s/d] xmm1, xmm2, xmm3/m128

vmulp[s/d] ymm1, ymm2, ymm3/m256

Mnoży liczby rzeczywiste pojedynczej/podwójnej precyzji z rejestru xmm2/ymm2 odpowiednio przez xmm3/ymm3 lub m32/m64/m128/m256, wynik zapisuje w xmm1/ymm1. Dla skalarów pozostałe elementy pochodzą ze źródła 1.

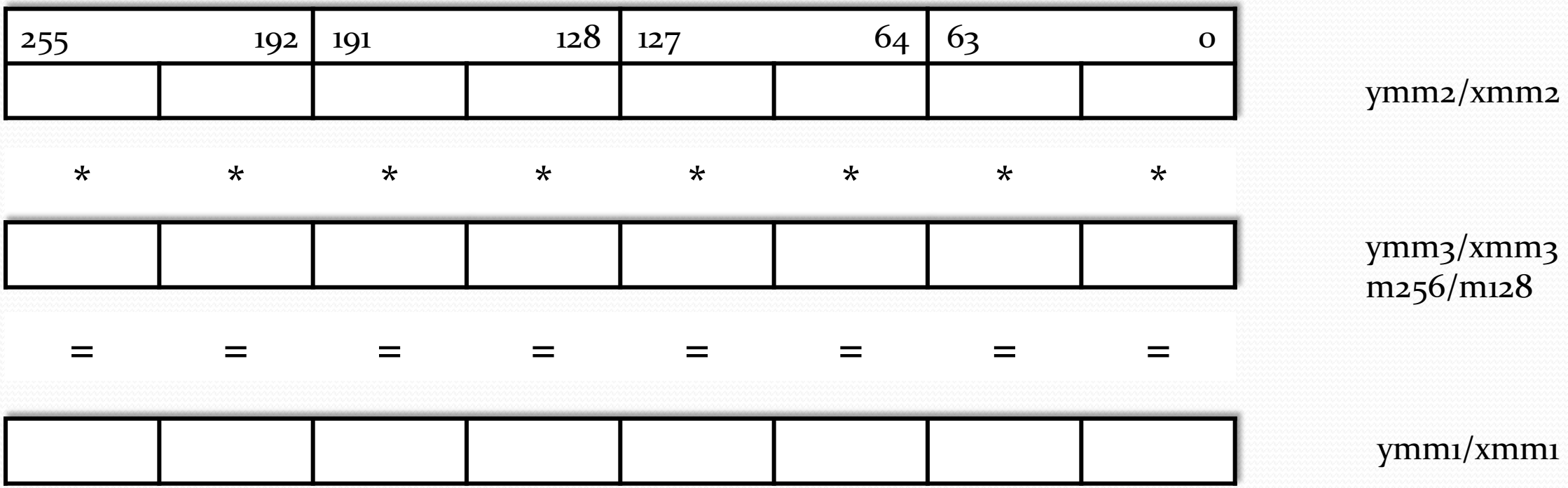
xmm1[i] = xmm2[i] * xmm3/m128[i]

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

Bity od 128/256 do MSB są zerowane.

Instrukcja mnożenia

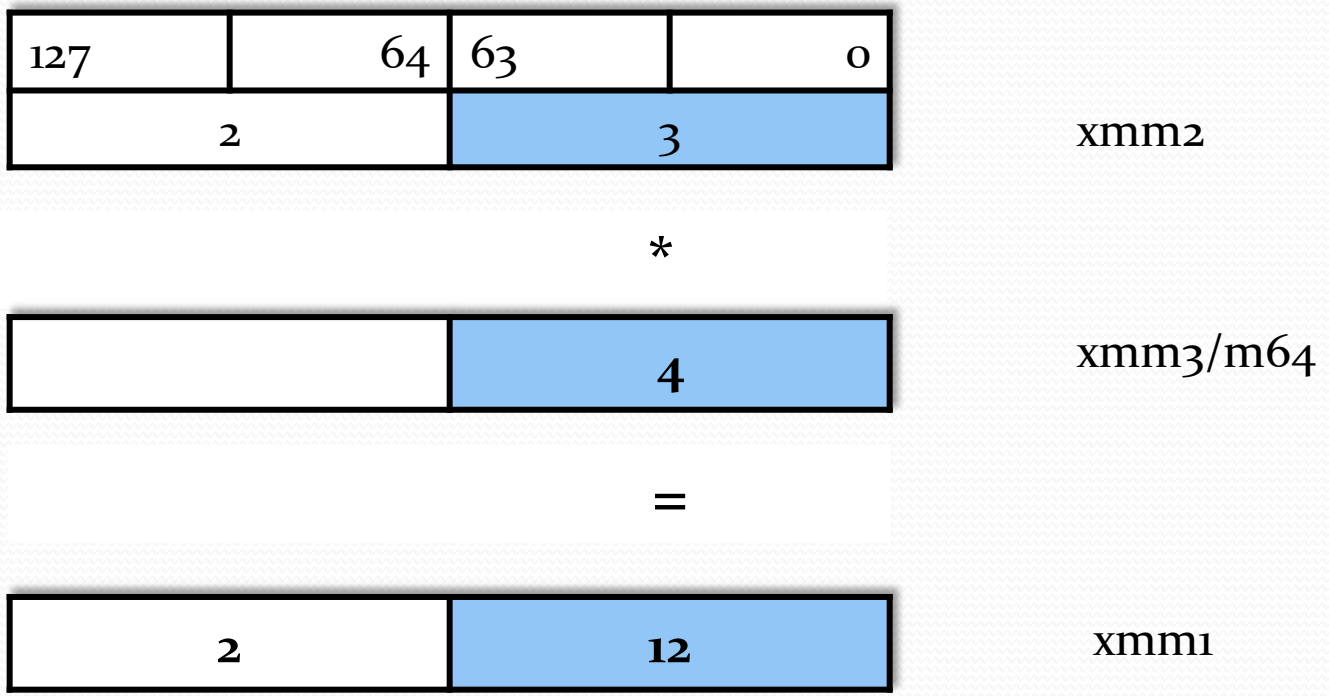
VMULPS



Bity od 128/256 do MSB są zerowane.

Instrukcja mnożenia

VMULSD



Bity od 128/256 do MSB są zerowane.

Instrukcja przykład: iloczyn skalarny

```
double iloczynVektor(double* tab1,
double *tab2, int n) {
double sum = 0;
double zerod = 0;
__asm {

push esi
push edi
vbroadcastsd ymm0, zerod;
mov ecx, n;
mov esi, tab1;
mov edi, tab2;
shl ecx, 3

p2:
sub ecx, 32
vmovupd ymm1, ymmword ptr[esi + ecx]
vmulpd ymm1, ymm1, ymmword ptr[edi + ecx]
vaddpd ymm0, ymm0, ymm1
jnz p2

vperm2f128 ymm1, ymm0, ymm0, 1; lo 1=hi 0
vaddpd ymm0, ymm0, ymm1
vpermilpd ymm1, ymm0, 1
vaddpd ymm0, ymm0, ymm1
vmovsd sum, xmm0;

pop edi
pop esi

} return sum; }
```

Instrukcja mnożenia , dodawania przykład:

```
.code
WIELOMIAN PROC public

push rbx
mov rax, rcx    ; wskaźnik na x w rcx
mov rbx, rdx    ; wskaźnik na wynik w rdx

movsxd r8, r8d ; movsxd r8, ilość w r8d

sub r8, 8 ; -1 bo zliczanie od 0
loop:
        vmovups ymm0, [rax+r8*4] ; x
        vmulps ymm1, ymm0, ymm0 ; x^2

        vmulps ymm2, ymm1, ymm0 ; x^3

        vaddps ymm0, ymm0, ymm1 ; x+x^2
        vaddps ymm0, ymm0, ymm2 ; x+x^2+x^3
        vmovups [rbx+r8*4], ymm0    ; wynik

        sub r8, 8 ; dekrementacja
        jge loop

        vzeroupper
        pop rbx

        ret
WIELOMIAN ENDP

end
```

Instrukcja mnożenia , dodawania przykład:

```
double wielomian_v1(double a, double b,
double c, double d, double x)
{
double wynik = 0;
__asm
{
vmovsd xmm0, a;
vmovsd xmm1, b;
vmovsd xmm2, c;
vmovsd xmm3, d;
vmovsd xmm4, x;
vmulsd xmm0, xmm0, xmm4     ; ax
vmulsd xmm2, xmm2, xmm4     ; cx
vmulsd xmm4, xmm4, xmm4     ; xx
vmulsd xmm0, xmm0, xmm4     ; axxx
vaddsd xmm0, xmm0, xmm3     ; axxx+d
vmulsd xmm1, xmm1, xmm4     ; bxx
vaddsd xmm0, xmm0, xmm2     ; axxx+cx+d
vaddsd xmm0, xmm0, xmm1     ; axxx+bxx+cx+d
vmovsd wynik, xmm0;

}
return wynik;
}
```

Instrukcja iloczyn skalarny

VDPPS

vdpps xmm1, xmm2, xmm3/m128, imm8

vdpps ymm1, ymm2, ymm3/m256, imm8

Oblicza iloczyn skalarny wektorów mnożąc warunkowo elementy rejestru xmm2 przez xmm3/mem, a następnie warunkowo (zależnie od ustawień imm8[3...0]) zapisuje wynik lub 0 w xmm1.

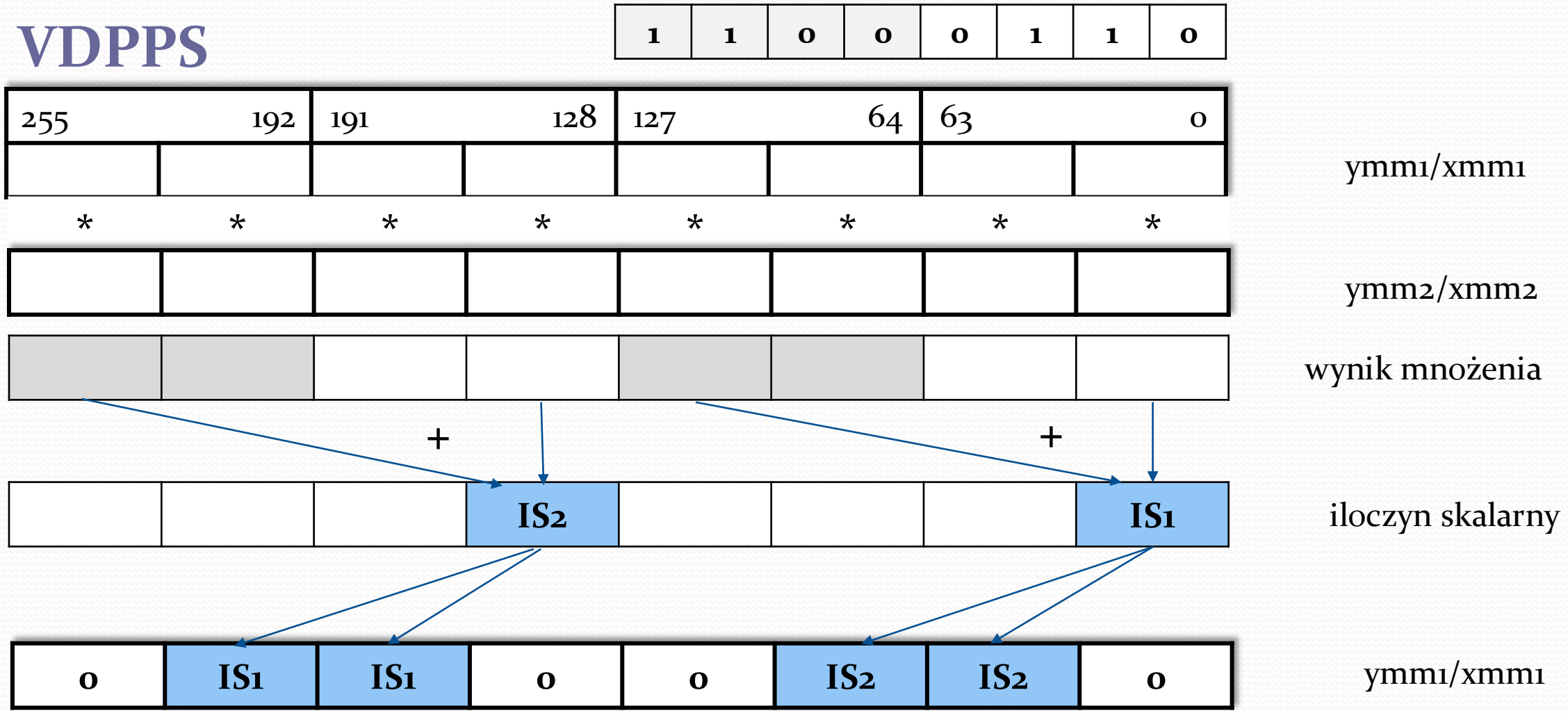
```
is = 0
if imm8[i+4] then
        is += xmm2[i]*xmm3/m128[i]

if imm8[i]
        xmm1[i] =  is
else
        xmm1[i] = 0;
```

```
is1 = 0; is2 = 0
if imm8[i+4] then
        is1 += xmm2[i]*xmm3/m128[i]
        is2 += ymm2[i+4]*ymm3/m256[i+4]
if imm8[i]
        xmm1[i] =  is1; ymm1[i+4] =  is2
else
        xmm1[i] =  0; ymm1[i+4] =  0
```

Bity od 128/256 do MSB są zerowane.

Instrukcja iloczyn skalarny



Instrukcja iloczyn skalarny

VDPPD

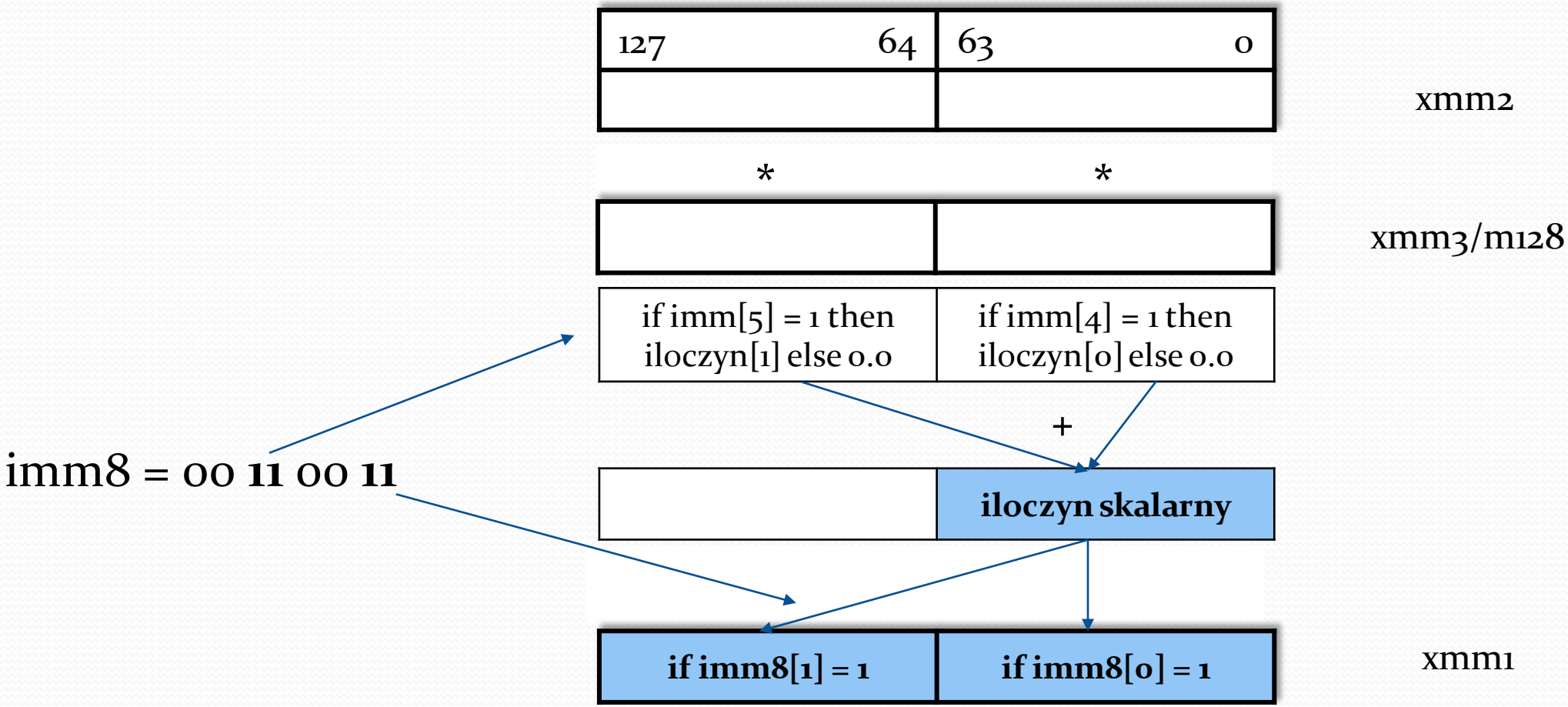
vdppd xmm1,xmm2, xmm3/m128, imm8 (tylko na xmm)

Oblicza iloczyn skalarny wektorów mnoży warunkowo elementy rejestru xmm2 przez xmm3/m128, a następnie sumuje iloczyny ustalając iloczyn skalarny, wynik, w zależności od bajtu sterującego zapisuje w podanych w imm8[1,0] lokalizacjach xmm1.

```
is = 0
if imm8[i+4] then
        is += xmm2[i]*xmm3/m128[i]
if imm8[i]=1 then
        xmm1[i] =  is
else
        xmm[i] = 0
```

Bity od 128/256 do MSB są zerowane.

Instrukcja iloczyn skalarny
VDPPD (tylko dla xmm)



Instrukcje dzielenia

Instrukcja dzielenia

VDIVS[S/D]

vdivs[s/d] xmm1, xmm2, xmm3/m32/m64

Dzieli liczby (skalary) rzeczywiste pojedynczej/podwójnej precyzji z rejestru xmm2/ymm2 odpowiednio przez xmm3/ymm3 lub m32/m64, wynik zapisuje w xmm1/ymm1. Pozostałe elementy przepisuje z xmm2.

xmm1[0] = xmm2[0] / xmm3[0]|m32|m64

Bity od 128/256 do MSB są zerowane.

Instrukcja dzielenia

VDIVP[S/D]

vdivp[s/d] xmm1, xmm2, xmm3/m128

vdivp[s/d] ymm1, ymm2, ymm3/m256

Dzieli wektory liczb rzeczywistych pojedynczej/podwójnej precyzji z rejestru xmm2/ymm2 odpowiednio przez xmm3/ymm3 lub m32/m64/m128/m256, wynik zapisuje w xmm1/ymm1.

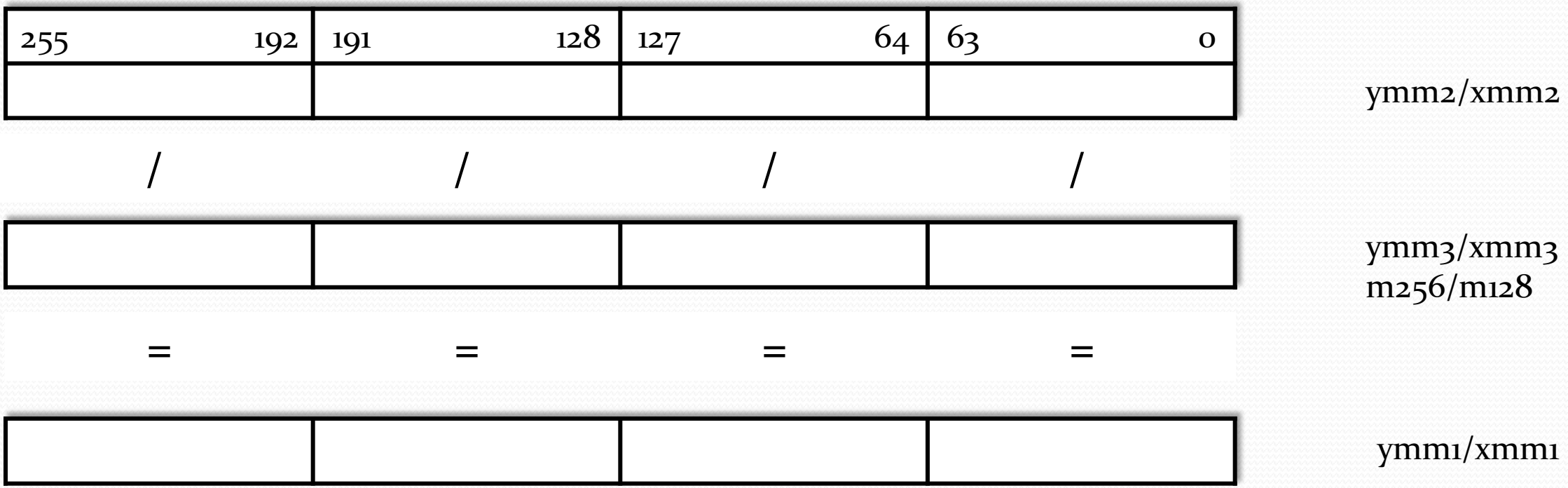
xmm1[i] = xmm2[i] / xmm3/m128[i]

ymm1[i] = ymm2[i] / ymm3/m256[i]

Bity od 128/256 do MSB są zerowane.

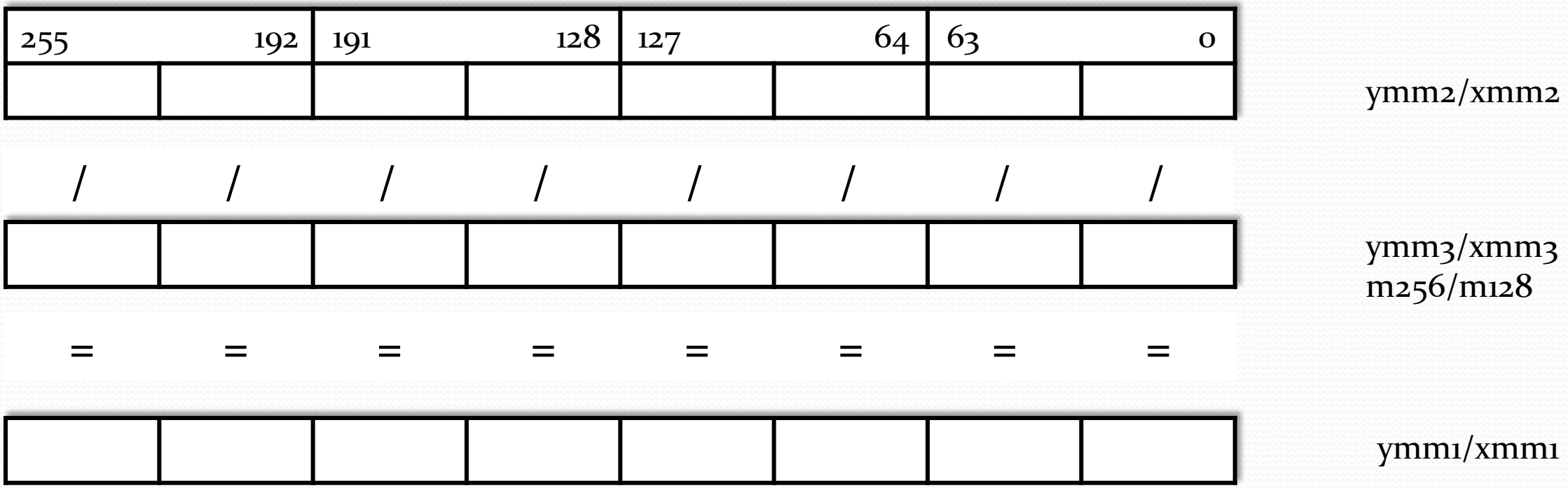
Instrukcja dzielenia

VDIVPD



Instrukcja mnożenia

VDIVPS



Bity od 128/256 do MSB są zerowane.

Instrukcja dzielenia przykład:

```
.code
dzielw proc public
; wektor liczb 1 w rcx; wektor liczb 2 w rdx; wektor liczb wynikowych w r8; liczba elementów w r9
        shl r9, 2    ; r9 rozmiar danych w bajtach
loopd:
        sub r9, 32
        vmovups ymm0, [rcx+r9]
        vmovups ymm1, [rdx+r9]
        vdivps ymm0, ymm0, ymm1
        vmovups [r8+r9], ymm0
        jnz loopd
        ret
dzielw endp
end
```

Operacje arytmetyczne AVX

- Instrukcje maksimum: VMAX[S/P][S/D]
- Instrukcje minimum: VMIN[S/P][S/D]
- Instrukcje zaokrąglenia: VROUND[S/P][S/D]
- Instrukcje odwróconej wartości przybliżonej: VRCPS[S/S]
- Instrukcje odwrócony pierwiastek przybliżony: VRSQRT[S/P]S
- Instrukcje pierwiastkowania: VSQRT[S/P][S/D]

Instrukcja
wartość maksymalna

Instrukcja wartość maksymalna

VMAXS[S/D], VMAXP[S/D]

vmaxs[s/d] xmm1, xmm2, xmm3/m32/m64

vmaxp[s/d] xmm1, xmm2, xmm3/m128

vmaxp[s/d] ymm1, ymm2, ymm3/m256

Zapisuje wartość maksymalną z porównania liczb rzeczywistych pojedynczej/podwójnej precyzji z rejestrów xmm2/ymm2 i xmm3/ymm3 lub m32/m64/m128/m256 do xmm1/ymm1. Dla skalarów pozostałe elementy pochodzą z xmm2.

if xmm2/ymm2[i] > xmm3/ymm3 lub m32/m64/m128/m256[i]

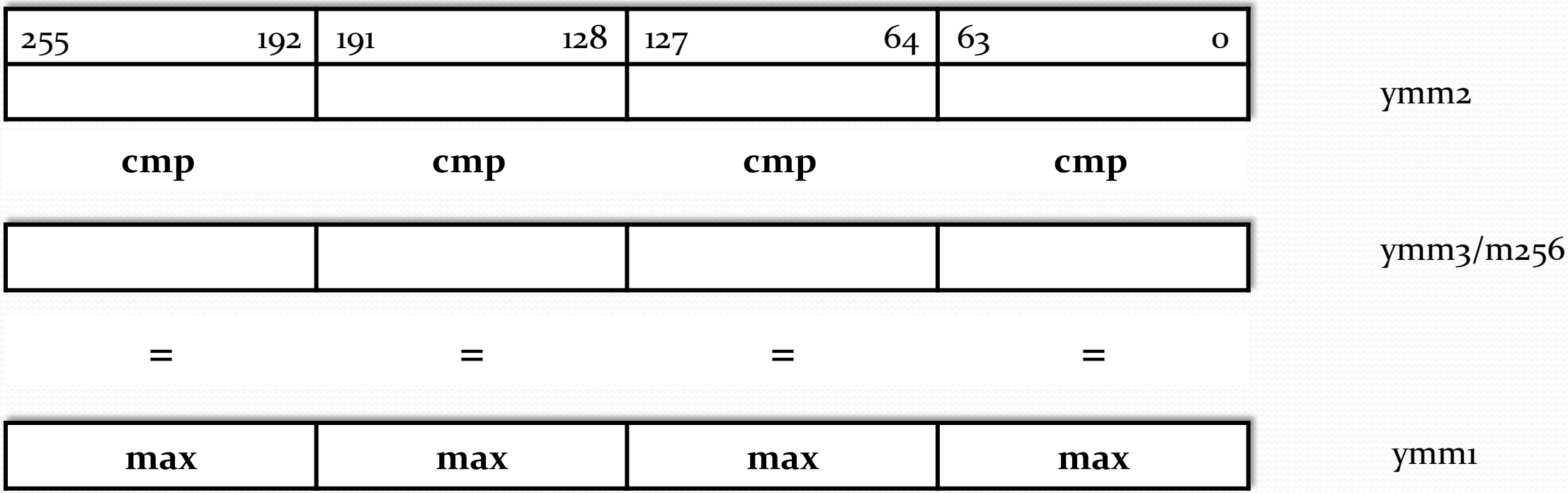
then xmm1/ymm1[i] = xmm2/ymm2[i]

else xmm1/ymm1[i] = xmm3/ymm3 lub m32/m64/m128/m256[i]

Bity od 128/256 do MSB są zerowane.

Instrukcja wartość maksymalana

VMAXPD



Bity od 128/256 do MSB są zerowane.

Instrukcja
wartość minimalna

Instrukcja wartość minimalna

VMIN[S/P][S/D]

vmins[s/d] xmm1, xmm2, xmm3/m32/m64

vminp[s/d] xmm1, xmm2, xmm3/m128

vminp[s/d] ymm1, ymm2, ymm3/m256

Zwraca wartość minimalną z liczb rzeczywistych pojedynczej/podwójnej precyzji odpowiednio skalary/wektory dla rejestrów xmm2/ymm2 i xmm3/ymm3 lub m32/m64/m128/m256, wynik zapisuje do xmm1/ymm1. Dla skalarów pozostałe elementy pochodzą z xmm2.

if xmm2/ymm2[i] < xmm3/ymm3 lub m32/m64/m128/m256[i]

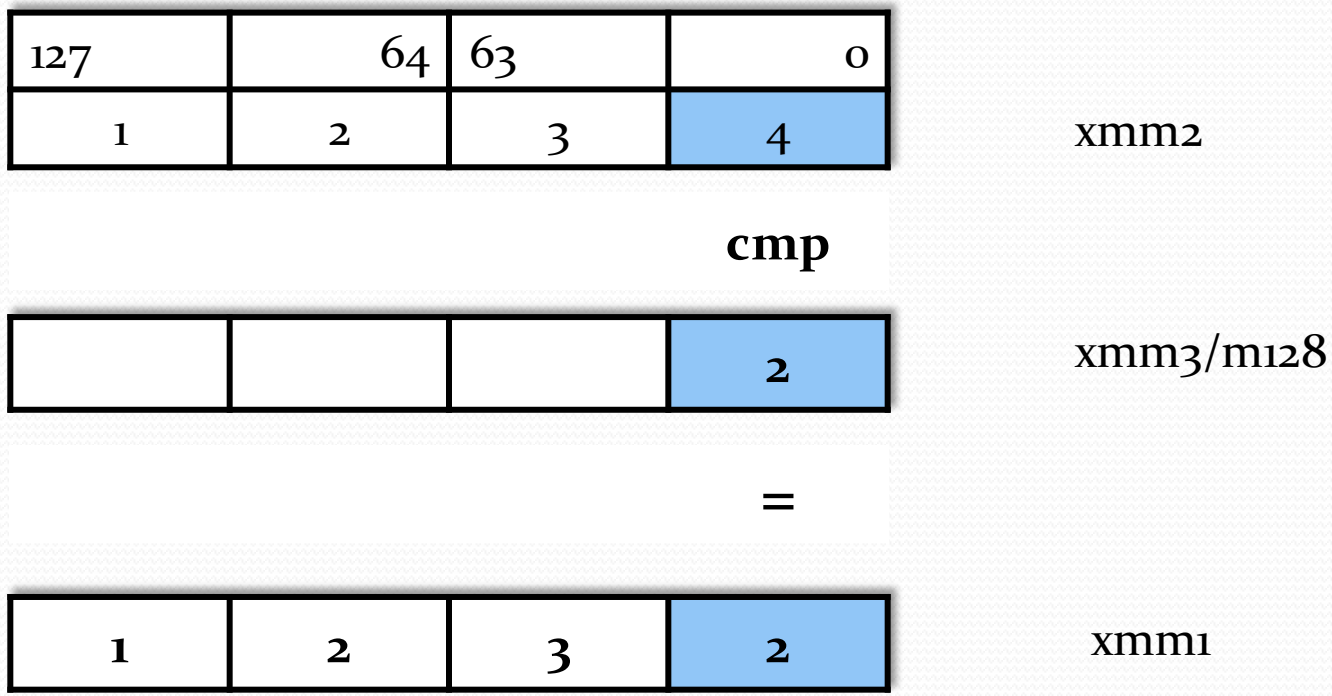
then xmm1/ymm1[i] = xmm2/ymm2[i]

else xmm1/ymm1[i] = xmm3/ymm3 lub m32/m64/m128/m256[i]

Bity od 128/256 do MSB są zerowane.

Instrukcja wartość minimalna

VMINSS



Bity od 128/256 do MSB są zerowane.

Instrukcje
wartość przybliżona

Instrukcja zaokrąglenia

VROUND[S/P][S/D]

vroundss xmm1, xmm2, xmm3/m32, imm8

vroundsd xmm1, xmm2, xmm3/m64, imm8

Zaokrągla najmłodszą liczbę rzeczywistą pojedynczej/podwójnej precyzji z xmm3 lub m32/m64 do wartości podwójnego/poczwórnego słowa (integer), wynik zapisuje w xmm1 jako liczbę rzeczywistą pojedynczej precyzji (z przecinkiem i zerami po nim), pozostałe elementy pochodzą z xmm2, sposób zaokrąglenia jest zdeterminowany bajtem sterującym imm8.

vroundp[s/d] xmm1, xmm2/m128, imm8

vroundp[s/d] ymm1, ymm2/m256, imm8

Zaokrągla wektor liczb rzeczywistych pojedynczej/podwójnej precyzji z xmm2/ymm2 lub m128/m256 do liczb całkowitych podwójnych/poczwórnych słów, wynik zapisuje jako liczby rzeczywiste w xmm1/ymm1, zaokrąglenie odbywa się według bajtu sterującego imm8.

Bity od 128/256 do MSB są zerowane.

Instrukcja zaokrąglenia

VROUND[S/P][S/D]

np. imm8 = 0000 **0 0 00**

P Precision Mask; 0. normal 1. inexact (niedokładny)

RS Rounding select (wybór zaokrąglenia) 1. MXCSR.RC 0. imm8.RC

RC Rounding mode (sposób zaokrąglenia)

Precision:
if imm8[3] = 0;

if imm8[3] = 1
then ustawienie precyzji jest ignorowane;

RC Rounding mode:
- 00 - Zaokrąglij do najbliższej (parzystej)
- 01 - Zaokrąglij w dół (w kierunku $-\infty$)
- 10 – Zaokrąglij w górę (w kierunku $+\infty$)
- 11 - Zaokrąglij do zera (obetnij)

Bity od 128/256 do MSB są zerowane.

Instrukcja wartości odwrotności przybliżonej

VRCP[SS/PS]

vrcpss xmm1, xmm2, xmm3/m32

Oblicza **przybliżoną** wartość **odwrotności** liczby rzeczywistej pojedynczej precyzji z xmm3/m32 i wynik umieszcza w xmm1, dodatkowo przepisuje starsze elementy xmm2 do xmm1.

(Relative Error ≤ 1,5 *2^-12.)

xmm1[31:0] ← 1.0/xmm3/m32[31:0]

xmm1[127:32] ← xmm12[127:32]

vrcpps xmm1, xmm2/m128

vrcpps ymm1, ymm2/m256

Oblicza przybliżoną wartość odwrotności elementów wektora liczb rzeczywistych pojedynczej precyzji z xmm2/ymm2 lub m128/m256 , wynik umieszcza w xmm1/ymm1. (Relative Error ≤ 1,5 *2^-12.)

xmm1[i] ← 1.0/xmm3/m128[i]

ymm1[i] ← 1.0/ymm3/m256[i]

Bity od 128/256 do MSB są zerowane.

Instrukcja wartości odwrotności przybliżonej pierwiastka

VRSQRTSS / VRSQRTPS

vrsqrtss xmm1, xmm2, xmm3/m32

Oblicza **przybliżoną odwrotność pierwiastka** z liczby rzeczywistej pojedynczej precyzji, wynik umieszcza w xmm1, dodatkowo przepisuje starsze elementy xmm2 do xmm1.(Relative Error ≤ 1,5 *2^-12.)

xmm1[31:0] ← 1.0/sqrt(xmm3/m32[31:0])
xmm1[127:32] ← xmm12[127:32]

vrsqrtps xmm1, xmm2/m128

vrsqrtps ymm1, ymm2/m256

Oblicza **przybliżoną odwrotność pierwiastka** z elementów wektora liczb rzeczywistych pojedynczej precyzji xmm2/ymm2 lub m128/m256, wynik umieszcza w xmm1/ymm1. (Relative Error ≤ 1,5 *2^-12.)

xmm1[i] ← 1.0/sqrt(xmm3/m128[i])
ymm1[i] ← 1.0/sqrt(ymm3/m256[i])

Bity od 128/256 do MSB są zerowane.

Instrukcje pierwiastkowania

Instrukcja wartości odwrotności przybliżonej pierwiastka

VSQRT[S/P][S/D]

vsqrtss xmm1, xmm2, xmm3/m32

vsqrtsd xmm1, xmm2, xmm3/m64

Oblicza wartość pierwiastka kwadratowego z liczby rzeczywistej pojedynczej/podwójnej precyzji xmm3/m32, wynik umieszcza w xmm1, dodatkowo przepisuje starsze elementy xmm2 do xmm1.

xmm1[0] ← sqrt(xmm3[0]/m32/m64)

xmm1[i] ← xmm2[i]

vsqrtp[s/d] xmm1, xmm2/m128

vsqrtp[s/d] ymm1, ymm2/m256

Oblicza wartość pierwiastka kwadratowego z elementów wektora liczb rzeczywistych pojedynczej /podwójnej precyzji xmm2/ymm2 lub m128/m256, wynik zapisuje w xmm1/ymm1.

xmm1[i] ← sqrt(xmm3/m128[i])

ymm1[i] ← sqrt(ymm3/m256[i])

Bity od 128/256 do MSB są zerowane.

Instrukcje FMA
Fused Multiply Add

Operacje arytmetyczne FMA

- Instrukcje mnożenie i dodawanie: VFMADD[123/213/231][S/P][S/D]
- Instrukcje mnożenie i odejmowanie: VFMSUB[123/213/231][S/P][S/D]
- Instrukcje mnożenie i odejmowanie z dodawaniem: VFMADDSUB[123/213/231]P[S/D]
- Instrukcje mnożenie i odejmowanie z dodawaniem: VFMSUBADD[123/213/231]P[S/D]
- Instrukcje mnożenie z negacją i dodawanie:
- VFNMADD[123/213/231][S/P][S/D]
- Instrukcje mnożenie z negacją i odejmowanie: VFNMSUB[123/213/231][S/P][S/D]

Instrukcje FMA

VFMADD[132/213/231][S/P][S/D]

vfmadd[132/231/231]S[S/D] xmm1, xmm2, xmm3/m32/m64

vfmadd[132/231/231]P[S/D] xmm1, xmm2, xmm3/m128

vfmadd[132/231/231]P[S/D] ymm1, ymm2, ymm3/m256

Mnoży skalary / wektory liczb rzeczywistych pojedynczej / podwójnej precyzji dwóch rejestrów i **dodaje** odpowiednie wartości trzeciego rejestru w zależności od podanej kolejności, pierwsze dwie cyfry oznaczają czynniki iloczynu, trzecia cyfra jest składnikiem sumy, wynik zapisuje w xmm1/ymm1.

132

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

231

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

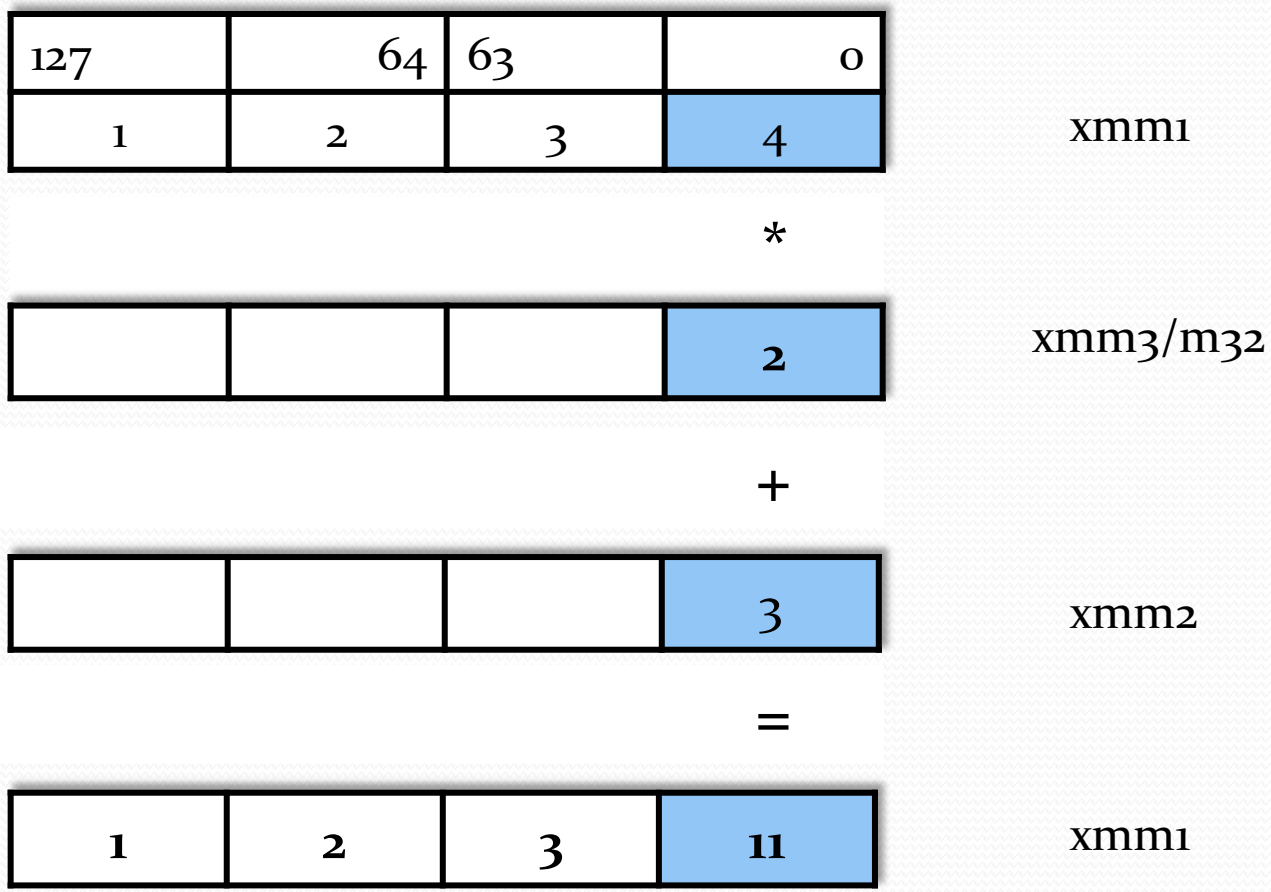
213

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

Bity od 128/256 do MSB są zerowane.

Instrukcje FMA

vfmadd132ss xmm1, xmm2, xmm3/m32



Instrukcje FMA

VFMSUB[132/213/231][S/P][S/D]

vfmsub[132/231/231]**S**[S/D] xmm1, xmm2, xmm3/m32/m64

vfmsub[132/231/231]**P**[S/D] xmm1, xmm2, xmm3/m128

vfmsub[132/231/231]**P**[S/D] ymm1, ymm2, ymm3/m256

Mnoży skalary / wektory liczb rzeczywistych pojedynczej / podwójnej precyzji dwóch rejestrów i **odejmuje** odpowiednie wartości trzeciego rejestru w zależności od podanej kolejności, pierwsze dwie cyfry oznaczają czynniki iloczynu, trzecia cyfra jest elementem różnicy, wynik zapisuje w xmm1/ymm1.

132

ymm1[i] = ymm1[i] * ymm3/m256[i] - ymm2[i]

231

ymm1[i] = ymm2[i] * ymm3/m256[i] - ymm1[i]

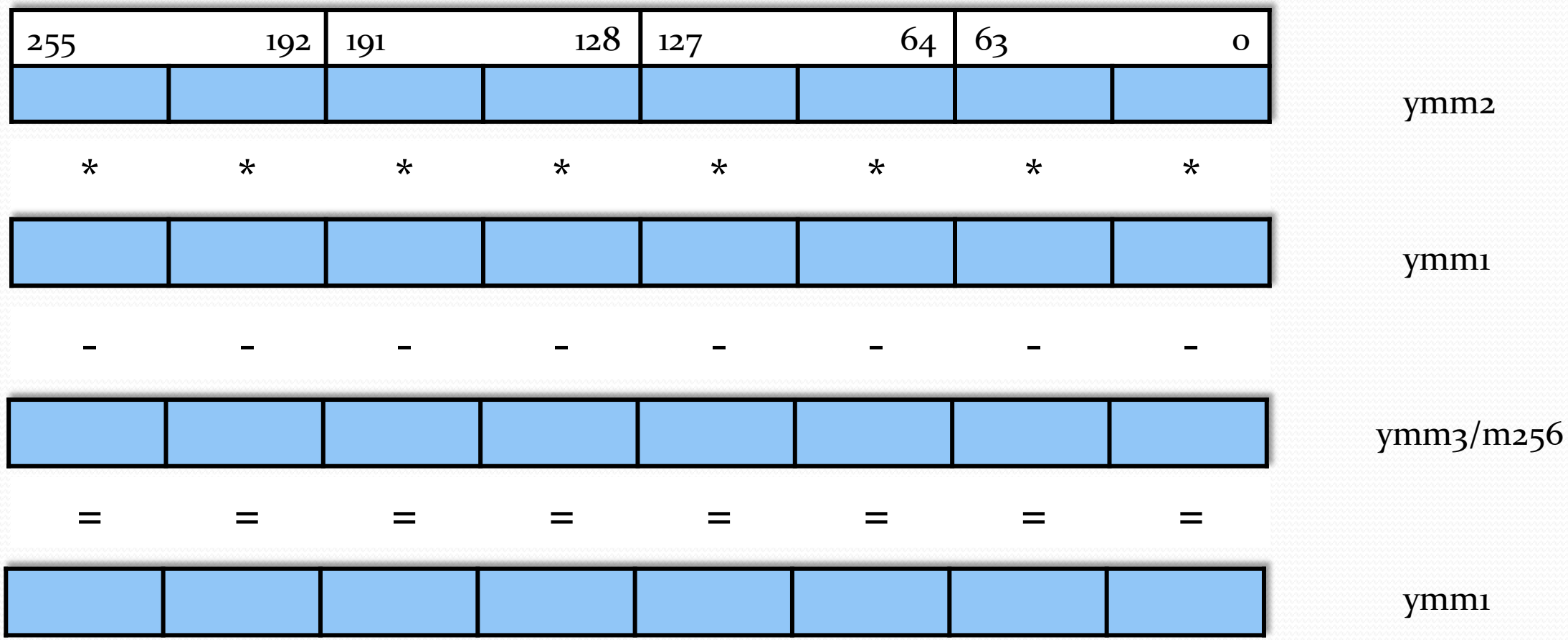
213

ymm1[i] = ymm2[i] * ymm1[i] - ymm3/m256[i]

Bity od 128/256 do MSB są zerowane.

Instrukcje FMA

vfmsub213ps ymm1, ymm2, ymm3/m256



Instrukcje FMA

VFMADDSUB[132/213/231]P[S/D]

vfmaddsub[132/231/231]p[s/d] xmm1, xmm2, xmm3/m128

vfmaddsub[132/231/231]p[s/d] ymm1, ymm2, ymm3/m256

Mnoży wektory liczb rzeczywistych pojedynczej / podwójnej precyzji dwóch rejestrów i naprzemiennie **odejmuje i dodaje** odpowiednie wartości trzeciego rejestru w zależności od podanej kolejności, pierwsze dwie cyfry oznaczają czynniki iloczynu, trzecia cyfra jest elementem różnicy/sumy, wynik zapisuje w xmm1/ymm1.

132

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

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

231

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

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

213

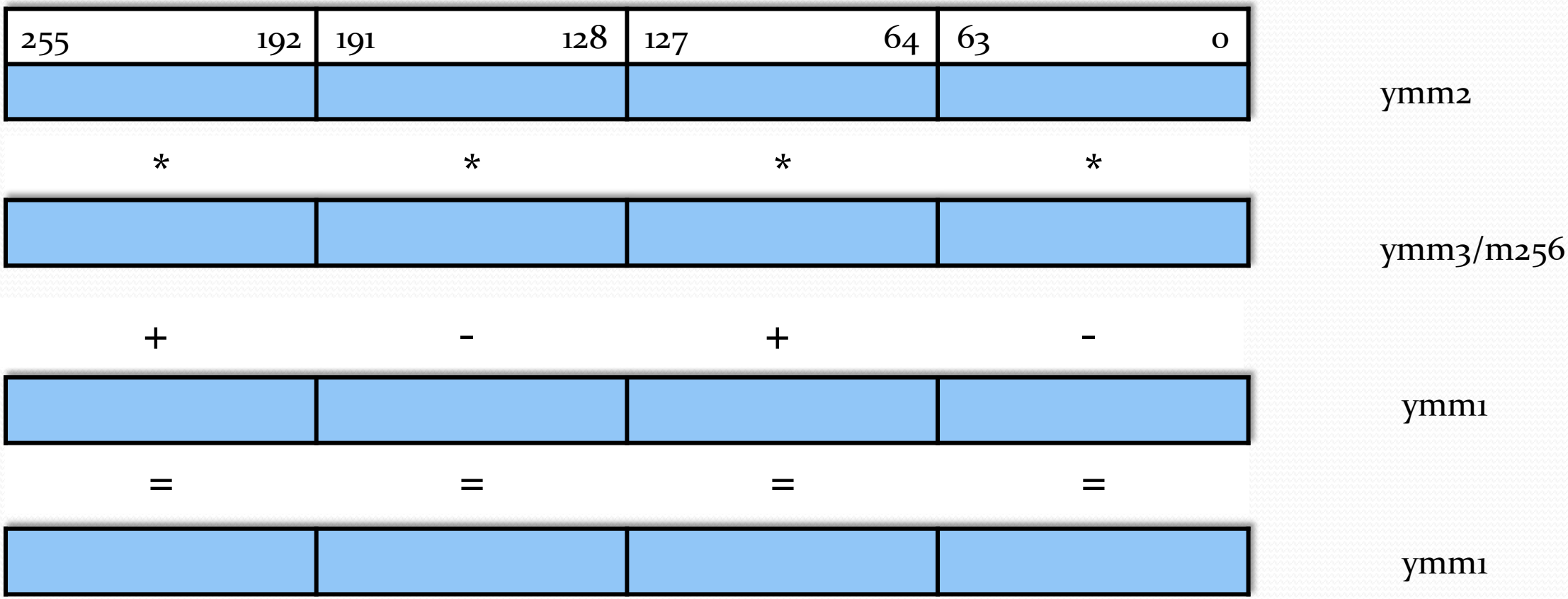
ymm1[2i] = ymm2[2i] * ymm1[2i] – ymm3/m256[2i]

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

Bity od 128/256 do MSB są zerowane.

Instrukcja FMA

vfmaddsub231pd ymm1, ymm2, ymm3/m256



Instrukcje FMA

VFMSUBADD[132/213/231]P[S/D]

vfmsubadd[132/231/231]p[s/d] xmm1, xmm2, xmm3/m128

vfmsubadd[132/231/231]p[s/d] ymm1, ymm2, ymm3/m256

Mnoży wektory liczb rzeczywistych pojedynczej / podwójnej precyzji dwóch rejestrów i naprzemiennie **dodaje i odejmuje** odpowiednie wartości trzeciego rejestru w zależności od podanej kolejności, pierwsze dwie cyfry oznaczają czynniki iloczynu, trzecia cyfra jest elementem sumy/różnicy, wynik zapisuje w xmm1/ymm1.

132

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

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

231

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

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

213

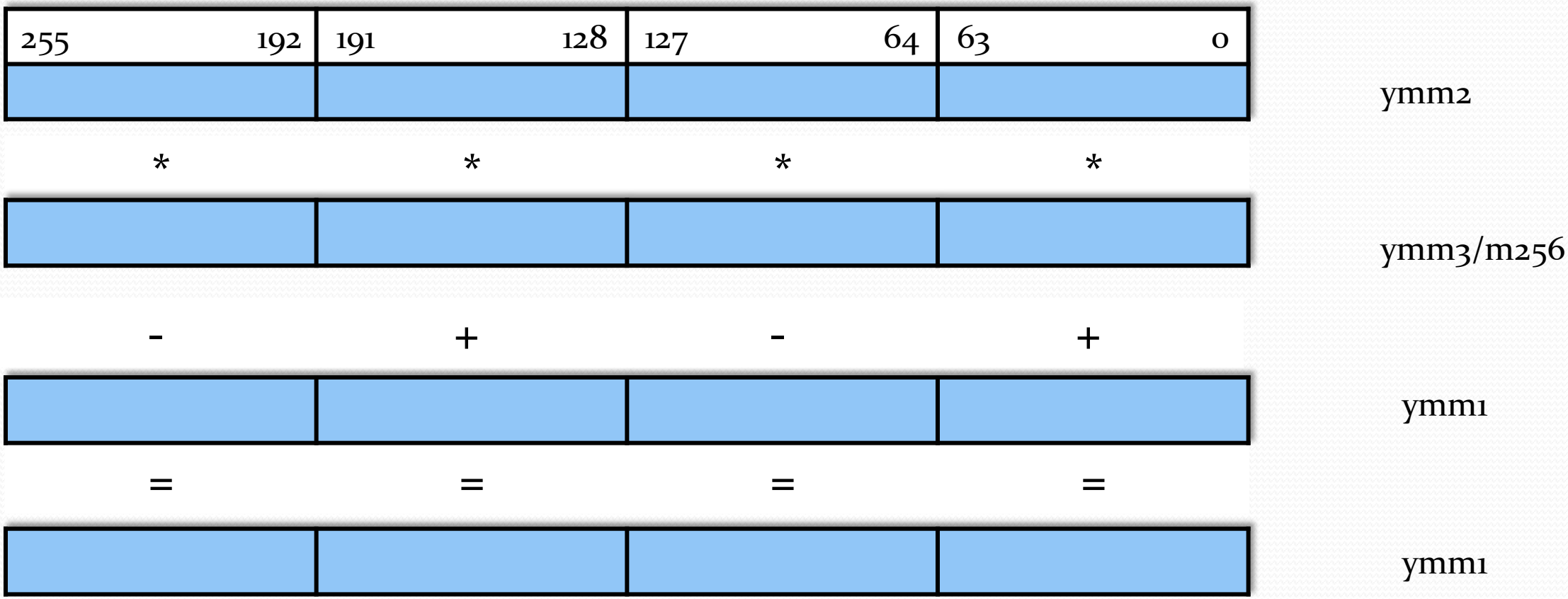
ymm1[2i] = ymm2[2i] * ymm1[2i] + ymm3/m256[2i]

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

Bity od 128/256 do MSB są zerowane.

Instrukcja FMA

vfmsubadd231pd ymm1, ymm2, ymm3/m256



Instrukcje FMA

VFNMADD[132/213/231][S/P][S/D]

vfnmadd[132/231/231]S[S/D] xmm1, xmm2, xmm3/m32/m64

vfnmadd[132/231/231]P[S/D] xmm1, xmm2, xmm3/m128

vfnmadd[132/231/231]P[S/D] ymm1, ymm2, ymm3/m256

Mnoży skalary / wektory liczb rzeczywistych pojedynczej / podwójnej precyzji dwóch rejestrów, **zmienia znak iloczynów** i **dodaje** odpowiednie wartości trzeciego rejestru w zależności od podanej kolejności, pierwsze dwie cyfry oznaczają czynniki iloczynu, trzecia cyfra jest składnikiem sumy, wynik zapisuje w xmm1/ymm1.

132

ymm1[i] = -ymm1[i] * ymm3/m256[i] + ymm2[i]

231

ymm1[i] = -ymm2[i] * ymm3/m256[i] + ymm1[i]

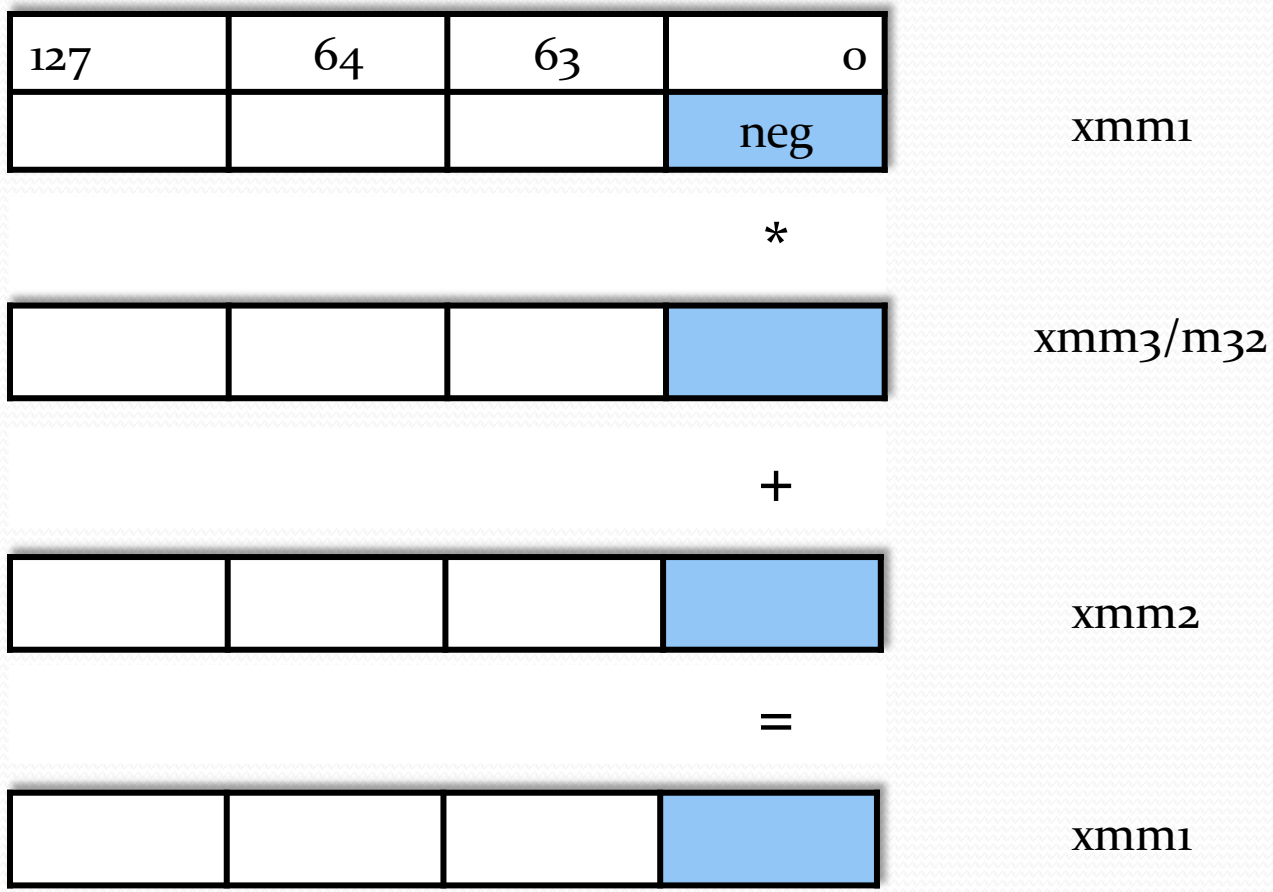
213

ymm1[i] = -ymm2[i] * ymm1[i] + ymm3/m256[i]

Bity od 128/256 do MSB są zerowane.

Instrukcje FMA

vfnmadd132ss ymm1, ymm2, ymm3/m256



Instrukcje FMA

VFNMSUB[132/213/231][S/P][S/D]

vfnmsub[132/231/231]**S**[S/D] xmm1, xmm2, xmm3/m32/m64

vfnmsub[132/231/231]**P**[S/D] xmm1, xmm2, xmm3/m128

vfnmsub[132/231/231]**P**[S/D] ymm1, ymm2, ymm3/m256

Mnoży skalary / wektory liczb rzeczywistych pojedynczej / podwójnej precyzji dwóch rejestrów, **zmienia znak iloczynów** i **odejmuje** odpowiednie wartości trzeciego rejestru w zależności od podanej kolejności, pierwsze dwie cyfry oznaczają czynniki iloczynu, trzecia cyfra jest elementem różnicy, wynik zapisuje w xmm1/ymm1.

132

ymm1[i] = -ymm1[i] * ymm3/m256[i] - ymm2[i]

231

ymm1[i] = -ymm2[i] * ymm3/m256[i] - ymm1[i]

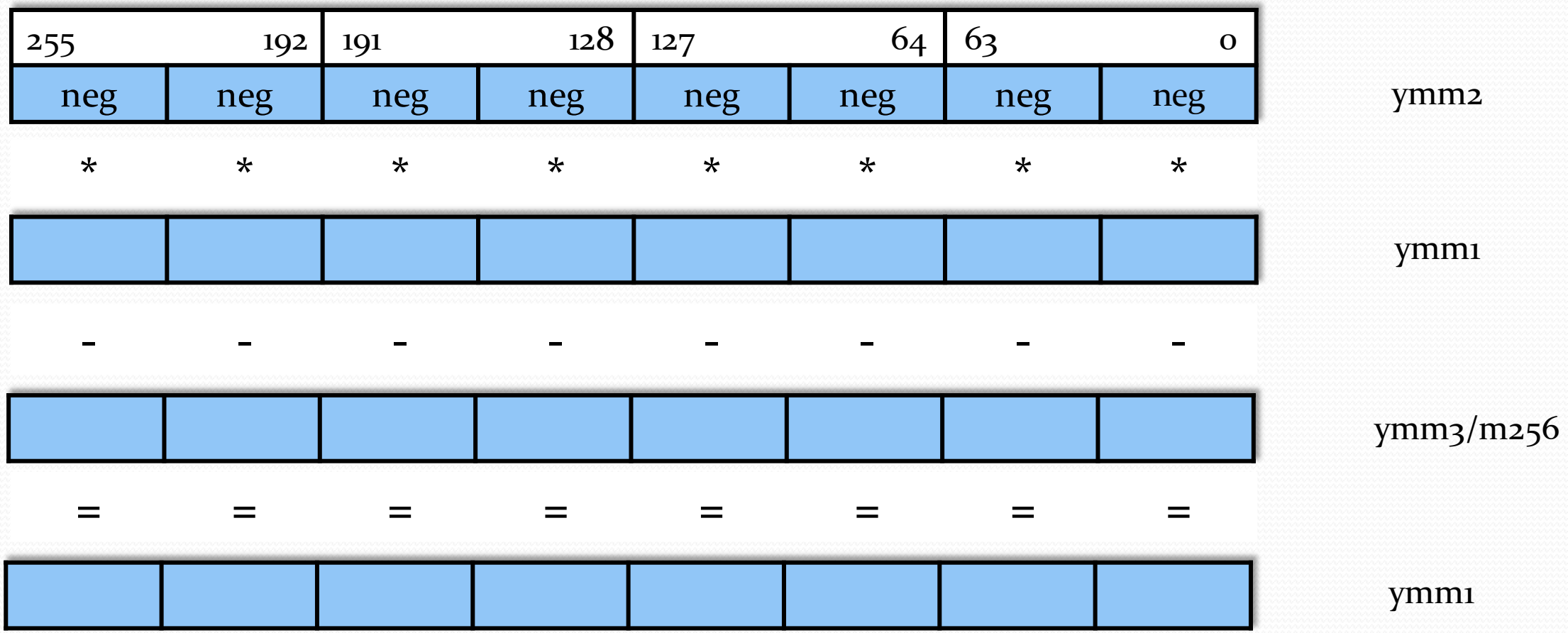
213

ymm1[i] = -ymm2[i] * ymm1[i] - ymm3/m256[i]

Bity od 128/256 do MSB są zerowane.

Instrukcje FMA

vfnmsub213ps ymm1, ymm2, ymm3/m256



Operacje porównania

Operacje porównania:

- Instrukcje porównania:

VCMP[S/P][S/D]

VCOMIS[S/D]

VUCOMIS[S/D]

Instrukcje porównania

VCMPS[S/D], VCMPP[S/D]

vcmps[s/d] xmm1, xmm2 xmm3/m32/m64, imm8

vcmpp[s/d] xmm1, xmm2 xmm3/m128, imm8

vcmpp[s/d] ymm1, ymm2, ymm3/m256, imm8

Porównuje skalary/wektory liczb rzeczywistych pojedynczej/podwójnej precyzji xmm2/ymm2 i xmm3/ymm3 lub m32/m64/m128/m256 według funktora zapisanego na bitach imm8[4:0] (łącznie 32 funktory), wynik **jako liczbę całkowitą** -1 lub 0 zapisuje w xmm1/ymm1. Dla skalarów pozostałe elementy są kopiowane ze źródła1.

Bity od 128/256 do MSB są zerowane.

Instrukcje porównania VCMPP[S/D]

funktor	imm8	opis	>	<	=	Unordered	#IA on QNAN
EQ_OQ (EQ)	0H	Equal (ordered, non-signaling)	f	f	t	f	No
LT_OS (LT)	1H	Less-than (ordered, signaling)	f	t	f	f	Yes
LE_OS (LE)	2H	Less-than-or-equal (ordered, signaling)	f	t	t	f	Yes
UNORD_Q (UNORD)	3H	Unordered (no-signaling)	f	f	f	t	No
NEQ_UQ (NEQ)	4H	Not-equal (unordered, non-signaling	t	t	f	t	No
NLT_US (NLT)	5H	Not-less-than (unordered, signaling)	t	f	t	t	Yes
NLE_US (NLE)	6H	Not-less-than-or-equal (unordered, signaling)	t	f	f	t	Yes
ORD_Q (ORD)	7H	Ordered (non-signaling)	t	t	t	f	No
EQ_UQ	8H	Equal (unordered, non-signaling)	f	f	t	t	No
NGE_US (NGE)	9H	Not-greater-than-orequal (unordered, signaling)	f	t	f	t	Yes
NGT_US (NGT)	AH	Not-greater-than (unordered, signaling)	f	t	t	t	Yes
FALSE_OQ (FALSE)	BH	False (ordered, non-signaling)	f	f	f	f	No
NEQ_OQ	CH	Not-equal (ordered, non-signaling)	t	t	f	f	No
GE_OS (GE)	DH	Greater-than-or-equal (ordered, signaling)	t	f	t	f	Yes
GT_OS (GT)	EH	Greater-than (ordered, signaling)	t	f	f	f	Yes
TRUE_UQ(TRUE)	FH	True (unordered, nonsignaling)	t	t	t	t	No
EQ_OS	10H	Equal (ordered, signaling)	f	f	t	f	Yes
LT_OQ	11H	Less-than (ordered, nonsignaling)	f	t	f	f	No
LE_OQ	12H	Less-than-or-equal (ordered, nonsignaling)	f	t	t	f	No
UNORD_S	13H	Unordered (signaling)	f	f	f	t	Yes
NEQ_US	14H	Not-equal (unordered, signaling)	t	t	f	t	Yes
NLT_UQ	15H	Not-less-than (unordered, nonsignaling)	t	f	t	t	No
NLE_UQ	16H	Not-less-than-or-equal (unordered, nonsignaling)	t	f	f	t	No
ORD_S	17H	Ordered (signaling)	t	t	t	f	Yes
EQ_US	18H	Equal (unordered, signaling)	f	f	t	t	Yes
NGE_UQ	19H	Not-greater-than-or-equal (unordered, nonsignaling)	f	t	f	t	No
NGT_UQ	1AH	Not-greater-than (unordered, nonsignaling)	f	t	t	t	No
FALSE_OS	1BH	False (ordered, signaling)	f	f	f	f	Yes
NEQ_OS	1CH	Not-equal (ordered, signaling)	t	t	f	f	Yes
GE_OQ	1DH	Greater-than-or-equal (ordered, nonsignaling)	t	f	t	f	No
GT_OQ	1EH	Greater-than (ordered, nonsignaling)	t	f	f	f	No
TRUE_US	1FH	True (unordered,signaling)	t	t	t	t	Yes

Instrukcje porównania VCMP[S/P][S/D]

Lp.	Pseudo-operacja	Implementacja
1.	VCMP**EQ**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 0*
2.	VCMP**LT**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 1*
3.	VCMP**LE**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 2*
4.	VCMP**UNORD**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 3*
5.	VCMP**NEQ**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 4*
6.	VCMP**NLT**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 5*
7.	VCMP**NLE**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 6*
8.	VCMP**ORD**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 7*
9.	VCMP**EQ_UQ**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 8*
10.	VCMP**NGE**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 9*
11.	VCMP**NGT**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 0AH*
12.	VCMPFALSE[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 0BH*
13.	VCMP**NEQ_OQ**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 0CH*
14.	VCMP**GE**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 0DH*
15.	VCMP**GT**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 0EH*

Lp.	Pseudo-operacja	Implementacja
16.	VCMP**TRUE**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, **0FH***
27.	VCMP**EQ_OS**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 10H*
18.	VCMP**LT_OQ**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 11H*
19.	VCMP**LE_OQ**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 12H*
20.	VCMP**UNORD_S**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 13H*
21.	VCMPN**EQ_US**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 14H*
22.	VCMP**NLT_UQ**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 15H*
23.	VCMP**NLE_UQ**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 16H*
24.	VCMP**ORD_S**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 17H*
25.	VCMP**EQ_US**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 18H*
26.	VCMP**NGE_UQ**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 19H*
27.	VCMP**NGT_UQ**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 1AH*
28.	VCMP**FALSE_OS**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 1BH*
29.	VCMP**NEQ_OS**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 1CH*
30	VCMP**GE_OQ**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 1DH*
31	VCMP**GT_OQ**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 1EH*
32	VCMP**TRUE_US**[S/P][S/D] *reg1, reg2, reg3*	VCMP[S/P][S/D] *reg1, reg2, reg3, 1FH*

Instrukcje porównania

VCOMIS[S/D] / VUCOMIS[S/D]

vcomis[s/d] xmm1, xmm2/m32/m64

Porównuje pojedyncze liczby rzeczywiste pojedynczej/podwójnej precyzji z rejestru xmm2 lub pamięci m32/m64 i xmm1 wynikiem jest **ustawienie odpowiednich flag procesora.** Instrukcja VCOMISD sygnalizuje wyjątek nieprawidłowej operacji zmiennoprzecinkowej SIMD (#I) gdy operandem źródłowym jest QNaN lub SNaN.

Instrukcje porównania

VCOMIS[S/D] / VUCOMIS[S/D]

vcomis[s/d] xmm1, xmm2/m32/m64

Porównuje pojedyncze liczby rzeczywiste pojedynczej/podwójnej precyzji z rejestru xmm2 lub pamięci m32/m64 i xmm1, wynikiem jest ustawienie odpowiednich flag procesora. Instrukcja VUCOMISD sygnalizuje wyjątek nieprawidłowej operacji tylko wtedy, gdy operandem źródłowym jest SNaN.

Operacje logiczne

Operacje logiczne

- Koniunkcja: VANDP[S/D]
- Koniunkcja z zaprzeczeniem: VANDNP[S/D]
- Alternatywa: VORP[S/D]
- Alternatywa wykluczająca: VXORP[S/D]
- Instrukcja Test: VTESTP[S/D]

Instrukcje logiczne koniunkcja

VANDP[S/D] / VANDNP[S/D]

vandp[s/d] xmm1, xmm2, xmm3/m128

vandp[s/d] ymm1, ymm2, ymm3/m256

Oblicza bitową **koniunkcję wektorów** liczb rzeczywistych pojedynczej/podwójnej precyzji z xmm2/ymm12 i xmm3/ymm3 lub m128/m256, wynik zapisuje w xmm1/ymm1.

cel[i] = źródło1[i] and źródło2[i]

vandnp[s/d] xmm1, xmm2, xmm3/m128

vandnp[s/d] ymm1, ymm2, ymm3/m256

Oblicza bitową **koniunkcję z negacją wektorów** liczb rzeczywistych pojedynczej /podwójnej precyzji z xmm2/ymm12 i xmm3/ymm3 lub m128/m256, wynik zapisuje w xmm1/ymm1.

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

Bity od 128/256 do MSB są zerowane.

Instrukcje logiczne alternatywa

VORP[S/D] / VXORP[S/D]

vorp[s/d] xmm1, xmm2, xmm3/m128

vorp[s/d] ymm1, ymm2, ymm3/m256

Oblicza bitową **alternatywę wektorów liczb rzeczywistych pojedynczej/podwójnej precyzji** rejestru xmm2/ymm2 (źródło 1) i xmm3/ymm3 lub m128/m256 (źródło 2), wynik zapisuje w w xmm1/ymm1.

cel[i] = źródło1[i] or źródło2[i]

vxorp[s/d] xmm1, xmm2, xmm3/m128

vxorp[s/d] ymm1, ymm2, ymm3/m256

Oblicza bitową **alternatywę wykluczającej wektorów liczb rzeczywistych pojedynczej/podwójnej precyzji** rejestru xmm2/ymm2 (źródło 1) i xmm3/ymm3 lub m128/m256 (źródło 2), wynik zapisuje w w xmm1/ymm1.

cel[i] = źródło1[i] xor źródło2[i]

Bity od 128/256 do MSB są zerowane.

Instrukcje testowania

VTESTP[S/D]

vtestp[s/d] xmm1, xmm2/m128

vtestp[s/d] ymm1, ymm2/m256

Wykonuje **logicznie koniunkcję (AND) na bitach znaku** liczb rzeczywistych pojedynczej / podwójnej precyzji z rejestru xmm1/ymm1 i xmm2/ymm2 lub m1128/m256, wynikiem jest ustawienie flagi ZF, jeśli wszystkie bity znaku = 0 => ZF = 1 lub ZF = 0 w przeciwnym przypadku

oraz jednocześnie

wykonuje **logicznie koniunkcję z zaprzeczeniem (AND NOT)(nie cel i źródło) na bitach znaku** liczb rzeczywistych pojedynczej / podwójnej precyzji z rejestru xmm1/ymm1 i xmm2/ymm2 lub m1128/m256, wynikiem jest ustawienie flagi CF, jeśli wszystkie bity znaku = 0 => CF=1, jeśli nie to CF=0.

Operacje konwersji

Operacje logiczne

- Całkowite na rzecywiste:

VCVTDQ2[PS/PD],VCVTSI2[SS/SD]

- Rzeczywiste na całkowite:

VCVT[PS/PD]2DQ ,VCVT[SS/SD]2SI

VCVTT[PS/PD]2DQ, VCVTT[SS/SD]2SI (z tranzakcją)

- Rzeczywiste na rzeczywiste:

VCVTSD2SS, VCVTSS2SD

VCVTPD2PS, VCVTPS2PD

Instrukcje konwersji całkowite na rzeczywiste

VCVTDQ2P[S/D]

vcvtdq2ps xmm1, xmm2/m128

vcvtdq2ps ymm1, ymm2/m256

vcvtdq2pd xmm1, xmm2/m64

vcvtdq2pd ymm1, xmm2/m128

Konwertuje dwa/cztery/osiem podwójnych słów ze znakiem z xmm2/ymm2 lub m128/m256 na dwie/cztery/osiem liczb rzeczywistych pojedynczej/podwójnej precyzji, wynik zapisuje do rejestru celu xmm1/ymm1. Konwertuje zawsze dwa/dwa, cztery/cztery, osiem/osiem.

Bity od 128/256 do MSB są zerowane.

Instrukcje konwersji całkowite na rzeczywiste

VCVTSI2S[S/D]

vcvtsi2ss xmm1, xmm2, reg/m32

vcvtsi2sd xmm1, xmm2, reg/m64

Konwertuje **pojedyncze podwójne słowo** ze znakiem z rejestru ogólnego przeznaczenia lub m32/m64 **na jedną liczbę rzeczywistą** pojedynczej/podwójnej precyzji, wynik zapisuje do xmm1/ymm1, bity [127:64/32] są przepisywane z xmm2/ymm2.

Bity od 128/256 do MSB są zerowane.

Instrukcje konwersji rzeczywiste na całkowite

VCVT[T]P[S/D]2DQ

vcvt[t]ps2dq xmm1, xmm2/m128

vcvt[t]ps2dq ymm1, ymm2/m256

Konwertuje cztery/osiem pojedynczych słów ze znakiem z xmm2/ymm2 lub m128/m256 na cztery/osiem podwójnych słów ze znakiem, wynik zapisuje w xmm1/ymm1.

vcvt[t]pd2dq xmm1, xmm2/m128

vcvt[t]pd2dq ymm1, ymm2/m256

Konwertuje dwa/cztery pojedyncze słowa ze znakiem z xmm2/ymm2 lub m128/m256 na dwa/cztery podwójne słowa ze znakiem, wynik zapisuje w xmm1/ymm1.

Zwrócona wartość jest zaokrąglana zgodnie z bitami kontrolnymi zaokrąglania w rejestrze MXCSR lub dla [T] obcięta kierunku zera.

Bity od 128/256 do MSB są zerowane.

Instrukcje konwersji rzeczywiste na całkowite

VCVT[T]S[S/D]2SI

vcvt[t]ss2si reg32, xmm1/m32

vcvt[t]ss2si reg64, xmm1/m64

Konwertuje liczbę pojedynczej precyzji z xmm1 lub pomięci m32/m64 na podwójne/poczwórne słowo ze znakiem, wynik zapisuje w rejestrze ogólnego przeznaczenia r32/r64.

vcvt[t]sd2si reg32, xmm1/m64

vcvt[t]sd2si reg64, xmm1/m64

Konwertuje liczbę podwójnej precyzji z xmm1 lub m32/m64 na podwójne/poczwórne słowo ze znakiem, wynik zapisuje w rejestrze ogólnego przeznaczenia r32/r64.

Instrukcja z T oznacza konwersję z obcięciem w kierunku zera.

Bity od 128/256 do MSB są zerowane.

Instrukcje konwersji rzeczywiste na rzeczywiste

VCVTSD2SS / VCVTSS2SD

vcvtsd2ss xmm1, xmm2, xmm3/m64

Konwertuje liczbę podwójnej precyzji z xmm3/m64 na liczbę pojedynczej precyzji, wynik umieszcza w xmm1, starsze bity xmm1 są uzupełniane z xmm2.

vcvtss2sd xmm1, xmm2, xmm3/m32

Konwertuje liczbę pojedynczej precyzji z xmm3/m32 na liczbę podwójnej precyzji, wynik umieszcza w xmm1, starsze bity xmm1 są uzupełniane z xmm2.

Bity od 128/256 do MSB są zerowane.

Instrukcje konwersji rzeczywiste na rzeczywiste

VCVTPD2PS / VCVTPS2PD

vcvtpd2ps xmm1, xmm2/m128

vcvtpd2ps xmm1, ymm2/m256

Konwertuje wektor liczb rzeczywistych podwójnej precyzji na wektor liczb rzeczywistych pojedynczej precyzji. Konwertuje dwa na dwa elementy lub cztery na cztery elementy.

vcvtps2pd xmm1, xmm2/m64

vcvtps2pd ymm1, xmm2/m128

Konwertuje wektor liczb rzeczywistych pojedynczej precyzji na wektor liczb rzeczywistych podwójnej precyzji. Konwertuje dwa na dwa elementy lub cztery na cztery elementy.

Bity od 128/256 do MSB są zerowane.