



## Struktura przedmiotu

- Wykład
- Laboratorium

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

2



### Literatura

- Adam Błaszczyk - Win32ASM. Asembler w Windows, Helion 2004
- Randall Hyde - Asembler. Sztuka programowania, Helion 2004
- Stanisław Kruk - Asembler w koprocesorze, Mikom 2003

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

3



### Literatura c.d.

- Ryszard Goczyński, Michał Tuszyński – Mikroprocesory 80286, 80386 i i486, Komputerowa Oficyna Wydawnicza „HELP” 1991
- Michał Tuszyński, Ryszard Goczyński – Koprocesory arytmetyczne 80287 i 80387 oraz jednostka arytmetyki zmiennoprzecinkowej mikroprocesora i486, Komputerowa Oficyna Wydawnicza „HELP” 1992

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

4



### Literatura c.d.

- Eugeniusz Wróbel – Praktyczny kurs asemblera, wyd. II, Helion 2011
- Vlad Pirogow – Asembler, Podręcznik programisty, Helion 2005

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

5



### Literatura c.d.

- Intel® 64 and IA-32 Architectures Software Developer’s Manual
  - Basic Architecture
  - Instruction Set
  - System Programming Guide

[www.intel.com](http://www.intel.com)

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

6

## Narzędzia

- MASM, FASM, ...
- Delphi, Visual Studio, ...

## Języki programowania

- Wysokiego poziomu – Ada, Basic, C, C++, C#, Fortran, Java, Pascal (Delphi), Python, SQL, ...
- Niskiego poziomu – asemblerы – odpowiadają kodowi wykonywanemu przez procesor

## Porównanie

	Języki wysokiego poziomu	Języki niskiego poziomu
Trudność programowania	niiska	duża
Przenośność	duża	mała
Wykorzystanie możliwości procesora i sprzętu	średnie	duże
Programowanie zadań wymagających czasowo	słabe	dobre
Przejrzystość kodu źródłowego	duża	mała
Przejrzystość kodu wynikowego	bardzo mała	duża
Szybkość tworzenia	duża	mała
Szybkość działania	mała	duża
Rozmiar kodu źródłowego	mały	średni
Rozmiar kodu wynikowego	duży	maly
Zajętość pamięci/dysku	duża	mała

## Porównanie

- Iloczyn skalarny

Lokalny	Rozmiar	Zakres	Stan powstania	Takty pizmowane
Wynik	1024	L_0	10000000	33
Delphi	253,322791312028	2796	451	9132
Asembler	253,322791312028	966	2742	
PPV1x1	253,322791312028		865	2706
PPV2x1	253,322791312028		431	1383
PPV4x1	253,322791312028		357	1128
SSE1x2	253,322791312028		430	1271
SSE2x2	253,322791312028		228	675
SSE4x2	253,322791312028		184	543
SBMx2	253,322791312028		298	921
Funkcje w bibliotece DLL				
PPV1x1_L	253,322791312028		866	2742
PPV2x1_L	253,322791312028		433	1383
PPV4x1_L	253,322791312028		358	1124
SSE1x2_L	253,322791312028		434	1282
SSE2x2_L	253,322791312028		225	676
SSE4x2_L	253,322791312028		189	552
Testy ARIK				
AVX1x1_L	253,322791312028		249	762
AVX2x1_L	253,322791312028		167	495
AVX4x1_L	253,322791312028		167	507
AVX1x2_L	253,322791312028		433	1383
AVX2x2_L	253,322791312028		226	681
AVX4x2_L	253,322791312028		187	576

## Porównanie

- Rekonstrukcja obrazu – tomograf komputerowy

Opis programu	Ilość wątków	1 iteracja [ms]	zok iteracji [s]	Przyispeszenie
Oryginalny	1	688	13760 3h49m20s	-
Asembler x64	1	8,1	162 2m42s	84.938
Asembler x64 wielowątkowy i9-7900X 10r 20t	8 10 16 20	1,119047 0,994545 1,017738 0,930187	22,38095 10,89089 20,35476 18,60375	614,809 691,774 676,009 739,636
i9-9980X i8R	16	0,68025	13,6059	1011,392
NVIDIA 1080Ti - 3584r		1,74473	34,8946	394,330
NVIDIA Titan V - 5120r		0,70534	14,091075	975,416

## Producenci procesorów

- AMD
- Cyrix
- IBM
- Intel**
- Nec
- Siemens
- Transmeta
- VIA
- Acorn Computers
- HP
- MOS Technology
- Motorola
- Silicon Graphix
- Zilog
- Texas Instruments
- Samsung

## Trocę historii

- F14 CADC (F-14A Central Air Data Computer)** - mikroprocesor zaprojektowany przez Steve'a Gellera i Raya Holta na potrzeby US Navy do myśliwca F-14 Tomcat.
  - Powstał w czerwcu 1970
  - niezwykle zaawansowany, 20-bitowy układ z techniką potokową
  - istnienie F-14 CADC zostało ujawnione dopiero w 1998 (z powodu tajemnicy wojskowej)

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

13

## Historia c.d.



- Intel 4004** - 4-bitowy mikroprocesor zaprojektowany i produkowany przez firmę Intel od 1971

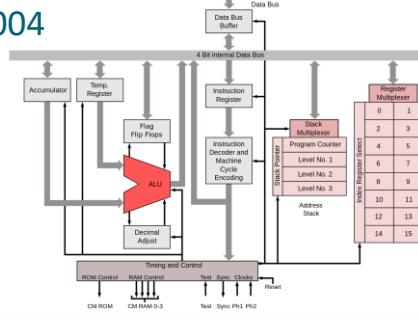
- Powszechnie uznany za pierwszy mikroprocesor
- Maksymalna częstotliwość taktowania - 740 kHz.
- Osobna pamięć dla programu i danych (tzw. "architektura harwardzka").
- 46 instrukcji.
- 16 czterobitowych rejestrów.
- 3-poziomowy stos.
- 2300 tranzystorów (technologia produkcji 10 µm).

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

14

## Historia c.d. – schemat Intel 4004



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

15

## Historia c.d.



- Intel 8008** – pierwszy mikroprocesor 8-bitowy Intel

- obudowa DIP18
- 8-bitowa magistrala
- dostęp do większej ilości RAM
- 3-4 razy więcej mocy obliczeniowej niż procesory 4-bitowe.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

16

## Historia c.d.



### Intel 8080

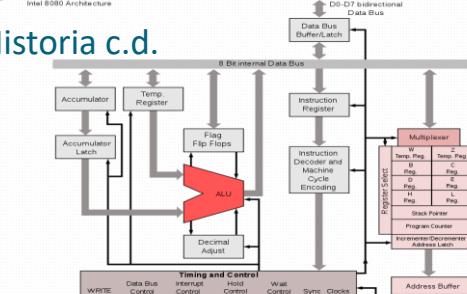
- wyprodukowany przez Intela w kwietniu 1974
- 8-bitowa szyna danych, pamięć adresowana 16-bitową szyną adresową.
- slowo 8-bitowe
- 72 instrukcje
- bezpośrednie adresowanie pamięci o pojemności do 64 KB
- arytmetyka dwójkowa i dziesiątna kodowana dwójkowo (BCD)
- 8 rejestrów programowych dostępnych dla programisty cykl pracy 2µs,
- zegar zewnętrzny o częstotliwości 2-3 MHz (podstawowy cykl rozkazowy – 4 taktów)

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

17

## Historia c.d.



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

18



## Historia c.d.

- Intel 8086 - procesor 16-bitowy wprowadzony w 1978
  - traktowany jako tymczasowy projekt przejściowy. Intel połknął wówczas swoje nadzieje w znacznie bardziej zaawansowanym 32-bitowym układzie 8800 (iAPX 432)
  - Głównym konstruktorem był Stephen Morse, który specjalizował się w oprogramowaniu. "Gdyby szefostwo Intelu chciało, by architektura ta przetrwała wiele generacji i przerodziła się w dzisiejsze procesory, to nigdy nie zleciłiby tego zadania jednej osobie"



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

19



## Historia c.d.

- W 1980 IBM rozpoczyna pracę nad komputerem 5150
- Microsoft ma już gotowy interpreter języka Basic, który działał na układach 8086 i 8088
- IBM 5150 staje się standardem "Pytaliśmy ich, czy chcą mieć komputer od International Business Machines czy od firmy, która swą nazwę wzięła od owocu"



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

20



## Historia c.d.

- rozszerzenie listy rozkazów
- rozszerzenie możliwości adresowania operandów
- wprowadzenie segmentacji obszaru pamięci
- mechanizmy przyspieszenia pracy
- mechanizmy dla pracy wieloprocesorowej

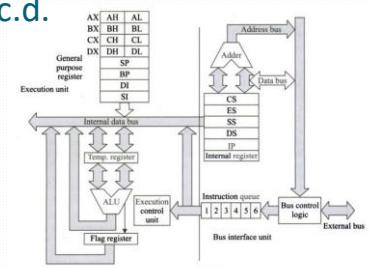
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

21



## Historia c.d.



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

22



## Historia c.d.

- Intel 80186 - procesor opracowany w firmie Intel w 1982.
  - posiadał nieco większą wydajność, kilkanaście nowych rozkazów i szybszy zegar
  - niektóre instrukcje były wykonywane 10-20 razy szybciej.
  - procesor wykorzystywany był głównie w systemach wbudowanych jako mikrokontroler.



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

23



## Historia c.d.

- Intel 80286 - 16-bitowy procesor opracowany przez firmę Intel, pokazany po raz pierwszy 1 lutego 1982
  - mniej więcej dwa razy bardziej wydajny w porównaniu do procesora Intel 8086
  - posiadał 24-bitową szynę adresową mógł adresować aż 16MB pamięci RAM
  - wprowadzono nowe instrukcje,
  - nowy tryb adresowania pamięci (tryb chroniony)



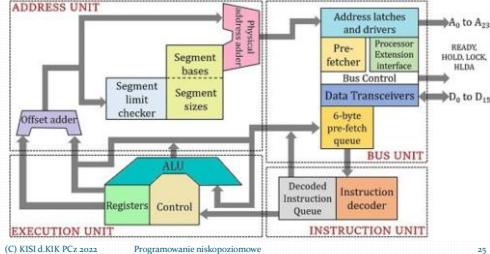
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

24

## Historia c.d.

- Intel 80286 - budowa



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

25

## Historia c.d.

- Intel 80386 - 32-bitowy procesor opracowany przez firmę Intel w 1986

- pierwszy 32-bitowy procesor z rodziną x86. Architektura tego procesora została opracowana jeszcze zanim Intel wypuścił na rynek procesory poprzedniej serii 286, jednak procesor był zbyt wydajny, aby go w tamtym czasie wyprodukować.
- 32-bitowa magistrala adresowa oraz 32-bitowa magistrala danych
- rozszerzone do 32-bitów rejestrów
- nowe tryby adresowania
- praca w trzech trybach: rzeczywistym, chronionym i wirtualnym
- dodanie do procesora jednostki MMU

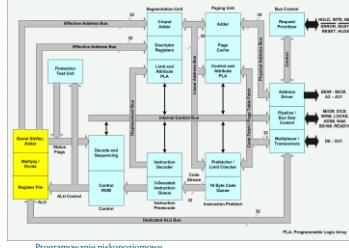
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

26

## Historia c.d.

- Intel 80386 - budowa



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

27

## Historia c.d.

- Intel 80486 - poprawna nazwa handlowa i486

- kilka (7) dodatkowych instrukcji
- cache na dane i instrukcję
- zintegrowany koprocesor arytmetyczny x87
- poprawiony interfejs szyny danych
- zastosowano pięciostopniowy potok.
- usprawnienia spowodowały, że i486 był mniej więcej dwukrotnie szybszy od podobnie taktowanego 80386

(C) KISI d.KIK PCz 2022

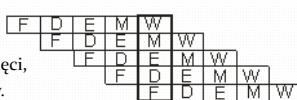
Programowanie niskopoziomowe

28

## Historia c.d.

Wykonywanie instrukcji w potoku. Fazy:

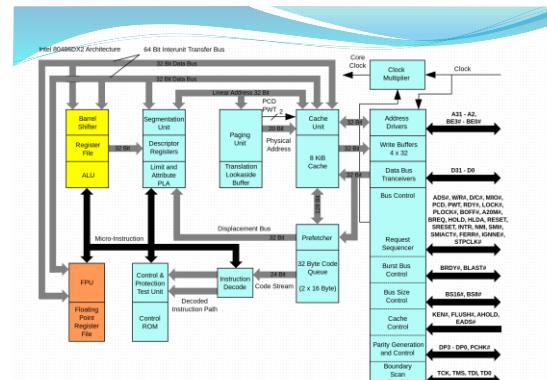
- F - pobieranie instrukcji,
- D - dekodowanie,
- E - wykonanie,
- M - odwołanie do pamięci,
- W - zapisanie wyników.



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

29



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

30

## Historia c.d.

- **Pentium – mikroprocesor się 22 marca 1993**
  - architektura superskalarna (wykonywanie kilku instrukcji w kilku potokach)
  - 64-bitowa szyna danych
  - jednostka *branch prediction* do przewidywania skoków (80% skuteczność)
  - przeprojektowany koprocesor (5-6x wydajniejszy niż w i486)



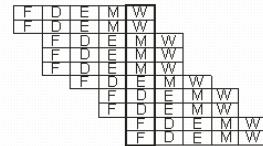
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

31

## Historia c.d.

- **Superskalarność** – cecha procesorów oznaczająca możliwość wykonywania kilku instrukcji (rozkazów maszynowych) jednocześnie, uzyskiwana poprzez zwiększenie jednostek wykonawczych.



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

32

## Historia cd.

- **Pentium Pro - mikroprocesor szóstej generacji należący do rodziny x86 (październik 1995)**
  - Podział kodu x86 na mikrorozkazy
  - Wykonywanie poza kolejnością
  - Wykonywanie spekulatywne
  - Dodatkowy potok ("pipeline") dla prostych instrukcji.



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

33

## Historia cd.

**Mikroprogram, mikro-kod** to program implementujący listę rozkazów procesora. Rozkaz kodu maszynowego jest ciągiem mikrorozkazów (mikroinstrukcji), jest ciągiem operacji sprzętowych wykonywanych wewnętrz procesora. Mikroprogram jest pisany przez konstruktörów procesorów w trakcie ich projektowania.

## Historia cd.

- Wykonywanie poza kolejnością (out-of-order execution)** – zdolność mikroprocesora superskalarnego do zmiany kolejności wykonywania instrukcji, tak, aby maksymalnie wykorzystać jego jednostki wykonawcze (obliczeniowe) i równolegle, w kilku potokach, wykonać jak najwięcej instrukcji. Powoduje to przyśpieszenie wykonywania programów.
- Zmiana kolejności jest możliwa tylko wówczas, gdy instrukcje są od siebie niezależne.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

35

## Historia cd.

**Wykonywanie spekulatywne (speculative execution)** – zdolność mikroprocesorów, przetwarzających rozkazy potokowo, do wykonywania rozkazów znajdujących się po skoku warunkowym, gdy jeszcze nie wiadomo, czy skok ten zostanie wykonany, i które rozkazy zostaną po nim wykonane. Jeśli jednak wybór był niewłaściwy, uzyskane wyniki zostaną pominięte, a potok wyczyszczony.

Wykonywanie spekulatywne występuje zwykle łącznie z mechanizmem **przewidywania skoków** (*branch prediction*).

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

36



## Historia c.d.

- Pentium II – mikroprocesor zaprezentowany 7 maja 1997
  - dodatkowe instrukcje MMX (MultiMedia eXtensions lub Matrix Math eXtensions)
  - poprawiona obsługa programów 16-bitowych
  - cache pierwszego poziomu (L1) dla kodu i danych: 16 kB

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

37



## Historia c.d.

- Pentium III - procesor w 32-bitowej architekturze Intel (IA-32).
  - architektura RISC (Reduced Instruction Set Computers)
  - rozmiar pamięci cache pierwszego poziomu (L1) dla kodu: 16 KB
  - liczba etapów przetwarzania rozkazu (w potoku): 12
  - liczba jednostek zmiennoprzecinkowych: 1 (z potokowaniem)
  - liczba jednostek całkowitoliczbowych: 6 potoków
  - liczba jednostek MMX:2
  - Instrukcje SSE (Streaming SIMD Extensions)
  - możliwość pracy w systemie wieloprocesorowym (do 2 procesorów).

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

38



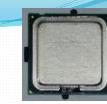
## Historia c.d.

- Pentium 4 – siódma generacja procesorów firmy Intel (od 20 listopada 2000) (wiele wersji)
  - architektura NetBurst
  - instrukcje SSE2, w nowszych wersjach jądra – SSE3
  - niektóre wersje posiadają wbudowaną wielowątkowość (HyperThreading)
  - zwiększoana pamięć poziomu L2
  - pojawia się technologia EM64T (2003)
  - pierwszy procesor dwurdzeniowy

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

39



## Historia c.d.

- Intel Core 2 to ósma generacja mikroprocesorów firmy Intel w architekturze x86
  - mikroarchitektura Intel Core
  - wysoki współczynnik IPC (Instructions Per Cycle) - około 3,5
  - wspólna pamięć cache dla obu rdzeni procesora
  - EM64T,
  - technologia wirtualizacji,
  - XD bit (eXecute Disable – wyłącza możliwość wykonywania instrukcji z oznaczonych stron),
  - ulepszona technologia SpeedStep,
  - wersja czterordzeniowa

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

40



## Historia c.d.

- Intel Core i7
  - modułowa budowa
  - ósmiodzienny Nehalem składa się z 731 milionów tranzystorów
  - SSE 4.2.
  - technologia współbieżnej wielowątkowości
  - dynamiczne zarządzanie zasilaniem
  - wbudowanie kontrolera pamięci RAM
  - technologia Quick-Path

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

41



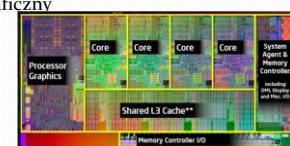
## Historia c.d.

- Intel Core i7 – 2 generacja - Sandy Bridge
  - modułowa budowa
  - 32-nanometrowy proces
  - 6 jednostek wykonawczych
  - wbudowany układ graficzny
  - instrukcje AVX
  - Turbo Boost 2.0
  - pamięć cache L3

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

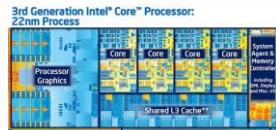
42





## Historia c.d.

- Intel Core i7 - 3 generacja - Ivy Bridge
- modułowa budowa
- 22-nanometrowy proces (tanzystory 3D)
- wbudowany układ graficzny Intel HD Graphics
- instrukcje AVX
- gen. liczb losowych
- PCI Express 3.0



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

43



## Historia c.d.

- Intel Core i7 - 4 generacja - Haswell
- podniesiona wydajność pamięci cache
- zwiększoana wydajność i energooszczędność
- 8 jednostek wykonawczych
- instrukcje AVX2, FMA3
- rozbudowany układ graficzny
- wsparcie Direct3D 11.1 i OpenGL 4.0

(C) KISI d.KIK PCz 2022

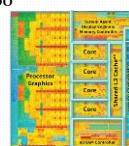
Programowanie niskopoziomowe

44



## Historia c.d.

- Intel Core i7 - 5 generacja - Broadwell
- technologia 14 nm
- zwiększoana wydajność i energooszczędność
- obsługa pamięci DDR3L
- rozbudowany układ graficzny Iris Pro 6200
- 128 MB pamięci podręcznej eDRAM
- wsparcie Direct3D 11.2 i OpenGL 4.4



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

45



## Historia c.d.

- Intel Core i7 - 6 generacja - Skylake
- technologia 14 nm
- obsługa pamięci DDR3L i DDR4
- instrukcje AVX 2
- Instrukcje AVX512 - 32 rejesty ZMM - 512 bitowe - w wersji XEON
- wsparcie Direct3D 12
- wsparcie dla Thunderbolt 3.0

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

46



## Historia c.d.

- Intel Core i7 - 7 generacja - Kaby Lake
- technologia 14 nm+
- zwiększenie częstotliwości zegara
- zwiększenie wydajności układu graficznego
- wsparcie Intel Optane Memory storage caching



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

47



## Historia c.d.

- Intel Core i7 - 8 generacja - Coffee Lake
- technologia 14++ nm
- zwiększenie liczby rdzeni do 6
- zwiększenie częstotliwości zegara
- zwiększenie wydajności układu graficznego

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

48



## Historia c.d.

- Intel Core i9 - 9 generacja - Coffee Lake
- technologia 14++ nm
- zwiększenie liczby rdzeni do 8
- zwiększenie rozmiaru pamięci cache L3
- AVX512 - 32 rejesty ZMM - 512bitowe - w wersji X

(C) KISI d.KIK PCz 2022

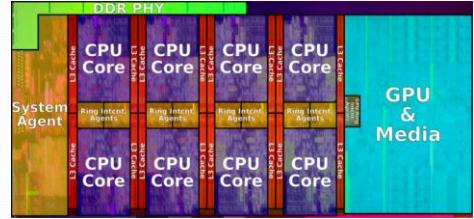
Programowanie niskopoziomowe

49



## Historia c.d.

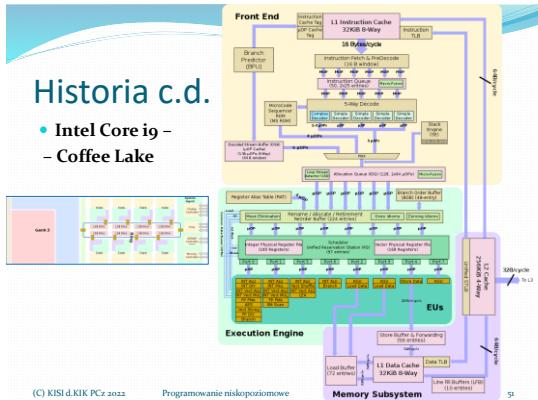
- Intel Core i9 - 9 generacja - Coffee Lake



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

50



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

51



## Historia c.d.

- Intel Core i9 - 10 generacja - Comet Lake
- zwiększenie maksymalnej częstotliwości taktowania do 5,3 GHz
- zwiększenie liczby rdzeni do 10
- zwiększenie rozmiaru pamięci cache L3
- zwiększenie wydajności układu graficznego (Gen 9.5)

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

52



## Historia c.d.

- Intel Core i9 - 11 generacja - Rocket Lake
- zwiększenie IPC o 10 do 19%
- instrukcje AVX512 i DL (Deep Learning)
- zwiększenie rozmiaru pamięci cache L1D - 48KiB
- nowy wydajny układ graficzny (Xe-LP Gen 12), DisplayPort 1.4a, HDMI 2.0b
- PCI Express 4.0

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

53



## Historia c.d.

- Intel Core i9 - 12 generacja - Alder Lake
- procesor hybrydowy - wydajne rdzenie dwuwątkowe i rdzenie energooszczędne - do 8P i 8E
- proces 10 nm SuperFin (Intel 7 proces)
- zwiększenie IPC
- 12 jednostek wykonawczych
- nowe instrukcje AVX-VNNI
- zwiększenie rozmiaru pamięci cache L3
- układ graficzny Xe-LP Gen 12.2 (32-96EU)
- PCI Express 5.0

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

54

## Co dalej?

- Raptor Lake
- Meteor Lake
- Arrow Lake
- Lunar Lake
- Nova Lake

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

55

## Zestawienie

Nazwa procesora	Rok	Maks. częstotliwość taktowania (w momencie wprowadzenia, MHz)	Liczba tranzystorów (mln.) Dane szacunkowe
Intel 8086	1978	8	0,029
Intel 80186	1982	12	0,025
Intel 80286	1982	12,5	0,134
Intel 80386	1985	20	0,275
Intel i486	1989	25	1,2
Pentium	1993	66	3,1
Pentium Pro	1995	200	5,5
Pentium MMX	1995	233	4,5
Pentium II	1997	266	7
Pentium III	1999	500	8,2
Pentium 4	2000	1500	42
Pentium 4 z EM64T	2003	2200	228
Pentium D	2004	3200	230
Intel Core 2	2006	3000	321
Intel Core 2	2008	3400	731
Intel Core i7 2600K	2011	3800	995
Intel Core i7 2700K	2012	3900-3900	1400
Intel Core i7 6700K	2015	4000-4200	1750
Intel Core i9 9900K	2018	3600-5000	>3000
Intel Core i9 10900K	2020	3700-5100	7400
Intel Xeon Phi KNM - 72r	2017	1500-1600	8000

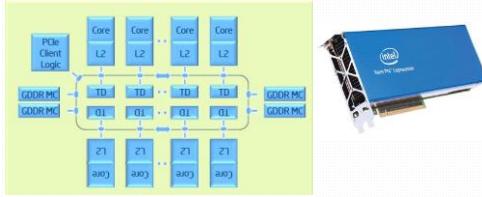
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

56

## Intel Xeon Phi - Knights Corner

- do 61 rdzeni połączonych w dwukierunkowy pierścień

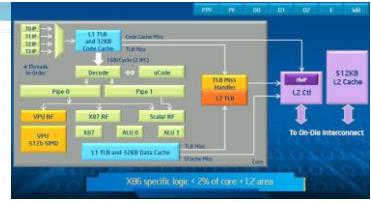


(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

57

## Xeon Phi



- rdzenie wykonują instrukcje SIMD - FMA3 na danych 512 bitowych, co oznacza, że jedna instrukcja przetwarza osiem zestawów danych podwójnej precyzyji albo 16 pojedynczej precyzyji - zwiększa to jeszcze stopień zrównoleglenia wykonywanych operacji.
- każdy rdzeń jest czterowątkowy,
- rdzenie posiadają dużą pamięć podręczną - 512KB.

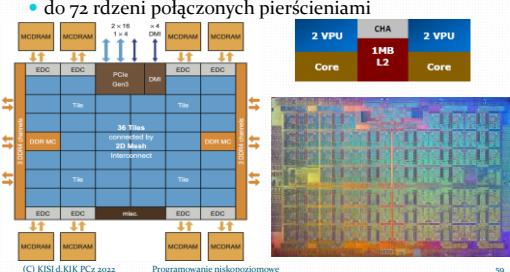
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

58

## Intel Xeon Phi - Knights Landing

- do 72 rdzeni połączonych pierścieniami



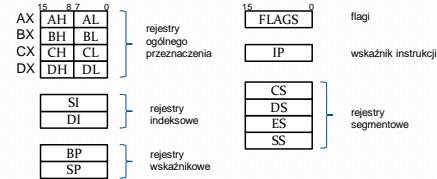
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

59

## Architektura procesora

### Procesor 8086 - rejesty



(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

2

## Rejestry

- **AX (ang. Accumulator)** - jest wykorzystywany głównie do operacji arytmetycznych i logicznych.
- **BX (ang. Base Registers)** - rejestr bazowy, głównie wykorzystywany przy adresowaniu pamięci.
- **CX (ang. Counter Registers)** - rejestr często wykorzystywany jako licznik, np. przy instrukcji LOOP.
- **DX (ang. Data Register)** - rejestr danych, wykorzystywany przy operacjach mnożenia i dzielenia, a także do wysyłania i odbierania danych z portów.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

3

## Rejestry c.d.

- **SI (ang. Source Index)** - rejestr indeksujący pamięć, wskazuje obszar z którego przesyłane są dane. W połączeniu z DS tworzy adres logiczny DS:SI.
- **DI (ang. Destination Index)** - rejestr indeksujący pamięć, wskazuje obszar, do którego przesyłane są dane. W połączeniu z ES, tworzy adres logiczny ES:DI.
- **BP (ang. Base Pointer)** - rejestr stosowany do adresowania pamięci.
- **SP (ang. Stack Pointer)** - wskaźnik stosu.

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

4

## Rejestry c.d.

- **IP (ang. Instruction Pointer)** – zawiera adres aktualnie wykonywanej instrukcji, może być modyfikowany przez rozkazy sterujące pracę programu.
- **FLAGS** – rejestr znaczników.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

5

## Rejestry c.d. - segmentowe

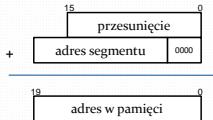
- **CS (ang. Code Segment)** - rejestr informujący o segmencie aktualnie wykonywanego rozkazu. Razem z IP tworzy adres logiczny CS:IP kolejnej instrukcji.
- **DS (ang. Data Segment)** - rejestr informujący o segmencie z danymi.
- **ES (ang. Extra Segment)** - rejestr informujący o segmencie dodatkowym np. przy operacjach przesyłania lańcuchów.
- **SS (ang. Stack Segment)** - rejestr informujący o segmencie stosu.

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

6

## Adres w trybie rzeczywistym

powstaje w wyniku sumowania położenia segmentu i przesunięcia w nim.

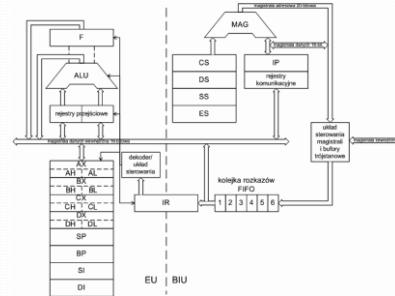


(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

7

## Architektura 8086

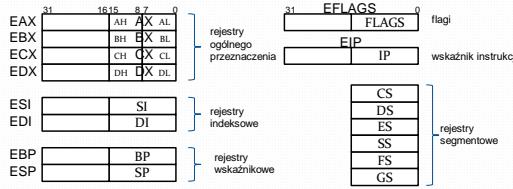


(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

8

## IA32- rejesty



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

9

## Rejestr flag

Rejestr flag w architekturze Intel x86			
bit	Skrot/wartość	opus	kryp
0	CF	flaga przeniesienia (carry)	S
1		zarezerwowany	
2	PE	flaga parzystosci (parity)	S
4	AF	flaga znakowa (adjust)	S
5	ZF	flaga zera (zero)	S
7	SF	flaga znaku (sign)	S
8	TP	flaga umożlwiiająca krokiowe wykonywanie (trap)	X
9	IF	flaga zezwolenia na przerwanie (interrupt enable)	X
10	DF	flaga kierunku przekreślonego (dir)	C
11	OF	flaga przepięcia (overflow)	S
12, 13	IOPL	poziom uprawnieni we/wy (IO privilege level, od .286)	X
14	NT	nested task flag (od .386)	X
16	RF	flaga wznowienia (resume, od .386)	X
17	VM	flaga trybu Virtual 8086 (od .386)	X
18	AC	alignment check (od .486SX)	X
19	VIF	Virtual interrupt flag (od Pentium)	X
20	VIP	Virtual interrupt pending (od Pentium)	X
21	ID	Identification (od Pentium)	X
3..5, 19..21, 22..31	0	zarezerwowany	

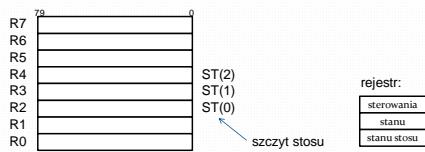
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

10

## Koprocesor

stos rejestrów



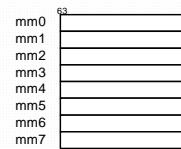
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

11

## Rejesty MMX

Działają na nich instrukcje całkowitoliczbowe SIMD  
Wykorzystują rejesty koprocesora



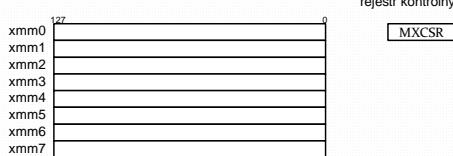
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

12

## Rejestry XMM

Działają na nich instrukcje zmiennoprzecinkowe SIMD

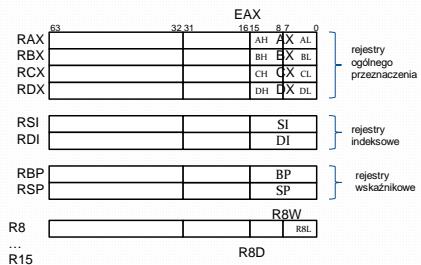


(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

13

## EM64T- rejesty

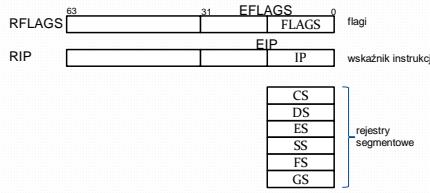


(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

14

## EM64T- rejesty



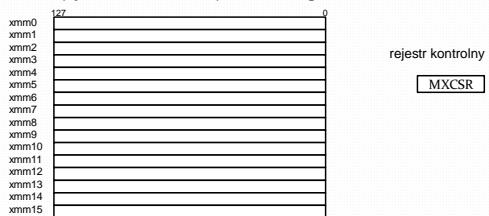
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

15

## EM64T- rejesty XMM

Działają na nich instrukcje zmiennoprzecinkowe SIMD



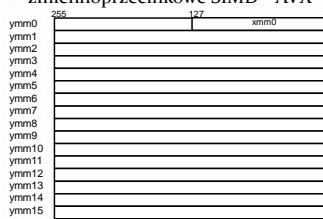
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

16

## AVX- Advanced Vector eXtensions

Rejestry ymm - działają na nich instrukcje zmiennoprzecinkowe SIMD - AVX



(C) KISI d.KIK PCz 2022

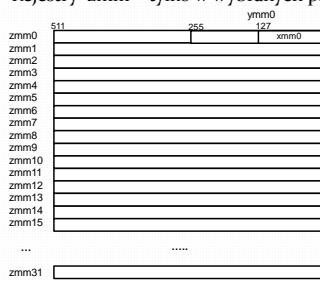
Programowanie niskopoziomowe

17

## AVX512-

## Advanced Vector eXtensions

Rejestry zmm - tylko w wybranych procesorach



(C) KISI d.KIK PCz 2022

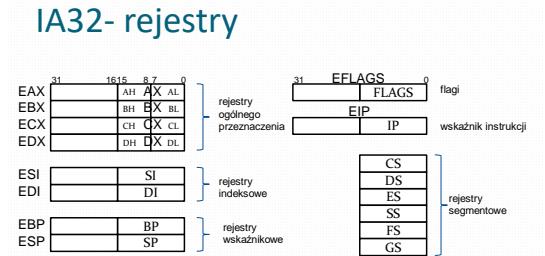
Programowanie niskopoziomowe

18



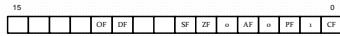


## Tryby adresowania Instrukcje przesyłania



(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

## Rejestr flag



Rejestr flag w architekturze Intel x86			
bit	Skrot/wartosc	Opis	Typ
0	CF	flaga przeniesienia (carry)	S
2	PF	flaga parzystosci (parity)	S
4	AF	flaga wyrównania (adjust)	S
6	ZF	flaga sera (zero)	S
7	SF	flaga znaku (sign)	S
10	DF	flaga kierunku (direction)	C
11	OF	flaga przepełnienia (overflow)	S

- S: Znacznik stanu
- C: Znacznik kontrolny
- X: Znacznik systemowy

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

## Format rozkazów

etykieta:	mnemonik	argumenti, argumentz	:komentarz
etykieta:	mnemonik	cel, źródło	:komentarz
etykieta:	mnemonik		
	mnemonik	argumenti	
	mnemonik	argumenti, argumentz	
Np.:			
	ret		
	pop	eax	
	mov	edx,ecx ;zapamiętaj licznik	
	mov	rax,1001	

(C) KIŚ i d KIK PGz 2022

## Tryby adresowania - rejestrowy

Argumentem instrukcji jest rejestr:

push ebx

mov edx,ebx

inc ecx

dec r9

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

## Tryby adresowania - prosty - natychmiastowy

Argumentem instrukcji jest wartość (zawiera się w kodzie rozkazu):

mov al, 5

mov r10d,32

mov edi, offset tabela

jnz petla

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

## Tryby adresowania - bezpośredni

Argumentem instrukcji jest adres w pamięci (wskaźnik):

```
mov al, [1234ec5fh]
```

```
mov edi, tabela ;pobiera pierwszy element
```

```
mov zmienna, rdx
```

## Tryby adresowania - pośredni - rejestrowy

Argumentem instrukcji jest rejestr – wskaźnik:

```
mov al, [rcx]
```

```
mov edi, [ebx]
```

```
mov [edi], edx
```

## Tryby adresowania - pośredni - bazowy

Argumentem instrukcji jest wskaźnik:

```
mov al, [ebx+5]
```

```
mov edi, [ebx+tablica]
```

```
mov [rbp+8], rdx
```

## Tryby adresowania - pośredni - indeksowy

Argumentem instrukcji jest rejestr – wskaźnik:

```
mov al, [esi]
```

```
mov edi, [esi*4+tablica]
```

```
mov [rdi*8+tablica], rdx
```

## Tryby adresowania - pośredni – bazowo-indeksowy

Argumentem instrukcji jest wskaźnik:

```
mov al, [ebx+esi+3]
```

```
mov edi, [ebx+eax*4]
```

```
mov [rbp+rdi*8+tablica], rdx
```

## Wielkość danych

Można określić wielkość stosowanych danych:

```
mov al, byte ptr [ebx+esi+3]
```

```
mov cx, word ptr [ebx+eax*4]
```

```
mov dword ptr [ebp+edi*4+tablica], edx
```

```
mov qword ptr [rbp+rdi*8+tablica], rdx
```

```
inc byte ptr [ebx+esi+3]
```

```
dec word ptr [ebx+eax*4]
```

```
inc dword ptr [ebp+edi*4+tablica]
```

## Przedrostki segmentowe

Można podać segment do danych:

```
mov al, es:byte ptr [ebx+esi+3]
mov cx, cs:word ptr [ebx+eax*4]
mov ss:[ebp+4], edx
```

Przyporządkowanie rejestrów

```
esp, ebp: ss
eax, ebx, ecx, edx, edi, esi: ds.
eip: cs
```

Analogicznie rejesty 16 i 64 bitowe.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

13

14

## Instrukcje przesyłania

- MOV przesyła dane między rejestrami, pamięcią zamień
- XCHG zamień bajty
- BSWAP wymień i dodaj
- XADD porównaj i wymień
- CMPXCHG porównaj i wymień 8(16) bajtów
- CMPXCHG8(i6)B wyslij na stos
- PUSH zdejmij ze stosu
- POP wyslij na stos flagi
- PUSHF/PUSHFD/PUSHFQ wyslij na stos flagi
- POPF/POPFD/POPFQ zdejmij ze stosu flagi
- PUSHA/PUSHAD wyslij rejestr na stos
- POPA/POPAD zdejmij rejestr ze stosu
- CWD/CDQ/CQO konwertuj word na dword/dword na qword
- CBW/CWDE / CDQE konwertuj byte na word/word na doubleword w rejestrze EAX/ doubleword na quadword w RAX
- MOVSX/MOVSD przeslij i rozszerz znakiem
- MOVZX przeslij i rozszerz zerem

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

14

Wpływ na flagi: -

## Instrukcja MOV

```
mov cel, źródło
```

Przesyła zawartość źródła do miejsca przeznaczenia (cel).

```
mov al, bl
mov [ebp+4], edx
mov zmienna, eax
mov rcx,licznik
mov [ebp+edi*4+tablica], edx
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

15

16

Wpływ na flagi: -

## Instrukcja XCHG

```
xchg cel, źródło
```

Zamienia zawartość źródła i celu.

xchg	ax, zmienna
xchg	ecx, [ebp+4]
xchg	rax, r12

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

16

Wpływ na flagi: -

## Instrukcja BSWAP

```
bswap rejestr
```

Zamienia bajty w argumencie – 32/64 bity.

```
bswap eax
bswap rdx
```

przed

12	c4	7f	de
de	7f	c4	12

po

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

17

18

Wpływ na flagi: OSZAPC

## Instrukcja XADD

```
xadd cel, źródło
```

Zamienia zawartość źródła i celu(8/16/32/64 bity), a ich sumę umieszcza w miejscu przeznaczenia (cel).

xadd	al,bl
xadd	eax,zmienna
xadd	edx,[ebx+esi*4]
xadd	rcx,r8

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

18

Wpływa na flagi: OSZAPC

## Instrukcja CMPXCHG

CMPXCHG arg1, arg2

Działanie:

```
if acc = arg1 then
    arg1 = arg2
else
    acc = arg1
```

acc = al, ax, eax, rx

arg2 - rejestr

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

19

Wpływa na flagi: OSZAPC

## Instrukcja CMPXCHG8(16)B

CMPXCHG8(16)B cel

Działanie:

```
if (E(R)DX:E(R)AX = cel) then
    cel = e(r)cx:e(r)bx
else
    e(r)dx:e(r)ax = cel
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

20

Wpływa na flagi: -

## Instrukcja PUSH

push arg

Przesyła zawartość argumentu na stos.

push	eax
push	rdx
push	ds
push	1234

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

21

Wpływa na flagi: -

## Instrukcja POP

pop cel

Przesyła zawartość stosu do celu.

pop	bx
pop	ecx
pop	rdx
pop	[edx+edi+4]

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

22

Wpływa na flagi: -

## Instrukcja

## PUSHF/PUSHFD/PUSHFQ

pushf/pushfd/pushfq

Przesyła zawartość Flag/Eflag/Rflag na stos.

pushf
pushfd
pushfq

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

23

Wpływa na flagi: OSZAPC

## Instrukcja

## POPF/POPFD/POPFQ

popf/popfd/popfq

Pobiera zawartość Flag/Eflag/Rflag ze stosu.

popf
popfd
popfq

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

24

Wpływa na flagi: -

## Instrukcja PUSHA/PUSHAD

pusha/pushad

Przesyła zawartość di, si, bp, bx, dx, cx, ax / edi, esi, ebp, ebx, edx, ecx, eax na stos.

**Instrukcja nie działa w trybie 64-bitowym.**

```
pusha
pushad
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

25

Wpływa na flagi: -

## Instrukcja POPA/POPAD

popa/popad

Przesyła zawartość stosu do di, si, bp, bx, dx, cx, ax / edi, esi, ebp, ebx, edx, ecx, eax.

**Instrukcja nie działa w trybie 64-bitowym.**

```
popa
popad
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

26

Wpływa na flagi: -

## Instrukcja CWD/CDQ/CQO

CWD/CDQ/CQO

Konwertuje z zachowaniem znaku word na doubleword / doubleword na quadword / quadword na octaword (ax na dx:ax, eax na edx:eax, rax na rdx:rax).

```
cwd
cdq
cqo
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

27

Wpływa na flagi: -

## Instrukcja CBW/CWDE/CDQE

cbw
cwde
cdqe

Konwertuje byte (AL) na word(AX) / word(AX) na doubleword (EAX) / doubleword (EAX) na quadword (RAX) z uwzględnieniem znaku.

```
cbw
cwde
cdqe
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

28

Wpływa na flagi: -

## Instrukcja MOVSX/MOVSDX

movsx cel, źródło

Przesyła zawartość źródła do rejestru celu z uwzględnieniem znaku. Cel posiada 2/4/8 razy więcej bitów.

```
movsx    eax, bl
movsx    cx, al
movsxd   r8, edx ;movsxd tylko dla 32 na 64
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

29

Wpływa na flagi: -

## Instrukcja MOVZX

movzx cel, źródło

Przesyła zawartość źródła do rejestru celu z dopisaniem na starszych bitach zer. Cel posiada 2/4/8 razy więcej bitów. Źródło 8/16 bitów.

```
movzx   eax, bl
movzx   cx, al
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

30

## Przykład

Wypełnienie wartościami od 0 do 255 tabeli bajtów

```

mov  ecx, 256
mov  eax, 0
mov  edi, 0
p1: mov  [edi+tabela], al
inc  edi
inc  eax
dec  ecx
jnz  p1

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

31

## Przykład

Przepisanie wartości integer (32 bity) z tabeli tab1 do tabeli tab2.

```

mov  rcx, 65536
mov  rdi, 0
p1: mov  eax, [tab1+4*rdi]
    mov  [tab2+4*rdi], eax
    inc  rdi
    dec  rcx
    jnz  p1

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

32

## Przykład

Przepisanie wartości int (32 bity) z tabeli tab1 do tabeli tab2 z konwersją na int64. rcx - adres tab1, rdx - adres tab2, r8 - liczba elementów.

```

p1: movsxd    rax, dword ptr[rcx+4*r8-4]
    mov      [rdx+8*r8-8], rax
    dec      r8
    jnz      p1

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

33

## Przykład

Zamiana zawartości zmiennych a i b typu int64.

```

mov  rax, a
xchg rax, b
mov  a, rax

push a
push b
pop  a
pop  b

```

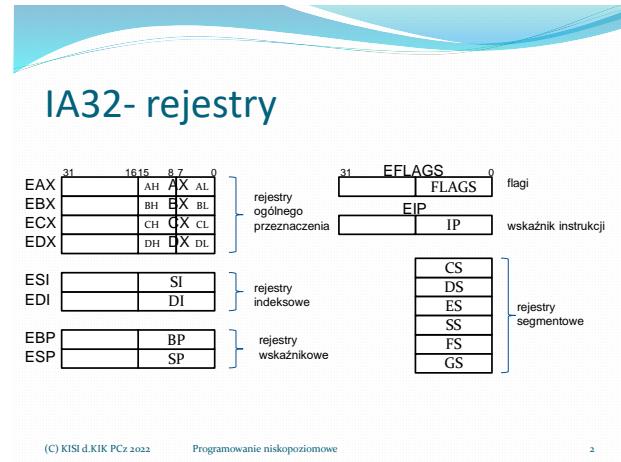
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

34



## Instrukcje arytmetyczne



(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

2

Diagram illustrating the Intel x86 flag register (EFLAGS) and its bit descriptions:

bit	Skrot/wartosc	opis	typ
0	CF	Flaga przeniesienia (carry)	S
2	PF	Flaga parzystosci (parity)	S
4	AF	Flaga wyrównania (adjust)	S
6	ZF	Flaga zera (zero)	S
7	SF	Flaga znaku (sign)	S
10	DF	Flaga kierunku (direction)	C
11	OF	Flaga przepelnienia (overflow)	S

S: Znaczek stanu  
C: Znaczek kontrolny  
X: Znaczek systemowy

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

3

List of arithmetic instructions:

- ADD dodawanie całkowitoliczbowe
- ADC dodawanie z przeniesieniem
- ADCX dodawanie z przeniesieniem bez znaku
- ADOX dodawanie z przeniesieniem bez znaku
- SUB odejmowanie
- SBB odejmowanie z pożyczką
- MUL mnożenie bez znaku
- MULX mnożenie bez znaku
- IMUL mnożenie ze znakiem
- DIV dzielenie bez znaku
- IDIV dzielenie ze znakiem
- INC inkrementacja (zwiększenie)
- DEC dekrementacja (zmniejszenie)
- NEG zmiana znaku
- CMP porównanie

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

4



Wpływa na flagi: OSZAPC

## Instrukcja ADD

add cel, źródło

Dodaje zawartość źródła i celu, sumę umieszcza w miejscu przeznaczenia (cel).

cel := cel + źródło

add eax, zmienna

add edx, [ebx+esi\*4]

add rcx, rbx

**Uwaga:**

Jeśli źródło jest adresowane w trybie prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

5



Przykład – oblicz sumę krawędzi prostokąta

```

    mov  eax, a          ;wczytaj a
    add  eax, b          ;dodaj b
    add  eax, c          ;dodaj c
    add  eax, eax         ;×2
    add  eax, eax         ;×2
  
```

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

6

Wpływa na flagi: OSZAPC

## Instrukcja ADC

adc cel, źródło

Dodaje zawartość źródła, celu i przeniesienia, sumę umieszcza w miejscu przeznaczenia (cel).

cel := cel + źródło + CF

adc eax, zmienna

adc edx, [ebx+esi\*4]

add rcx, rbx

**Uwaga:**

Jeśli źródło jest adresowane w trybie prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

7

;adres zmiennej b

;adres zmiennej a

;najmłodsze podwójne słowo b

;dodajemy do najmłodszego a

;drugie podwójne słowo b

;do drugiego a plus przeniesienie

;trzecie podwójne słowo b

;do trzeciego a plus przeniesienie

;czwarte podwójne słowo b

;do czwartego a plus przeniesienie

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

8

## Przykład – oblicz sumę liczb całkowitych 128-bitowych

```
mov rsi, offset b      ;adres zmiennej b
mov rdi, offset a      ;adres zmiennej a
mov rax, [rsi]          ;najmłodsze poczwórne słowo b
add [rdi], rax          ;dodajemy do najmłodszego a
mov rax, [rsi+8]        ;drugie poczwórne słowo b
adc [rdi+8], rax        ;do drugiego a plus przeniesienie
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

9

Wpływa na flagi: O----  
Wymagane: ADX

## Instrukcja ADOX

adox cel, źródło

Dodaje bez znaku zawartość źródła, celu i flagi przepelenienia, sumę umieszcza w miejscu przeznaczenia (cel – rejestr 32|64 bitowy).

cel := cel + źródło + OF

adox eax, zmienna

adox edx, [ebx+esi\*4]

adox rcx, rbx

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

11

Wpływa na flagi: OSZAPC

Instrukcja SUB

sub cel, źródło

Odejmuję zawartość źródła od celu, różnicę umieszcza w miejscu przeznaczenia (cel).

cel := cel - źródło

sub ecx, zmienna

sub ebx, [ebx+esi\*4]

sub rax, rdx

**Uwaga:**

Jeśli źródło jest adresowane w trybie prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

12

Wpływ na flagi: OSZAPC

## Instrukcja SBB

sbb cel, źródło

Odejmuje zawartość źródła od celu z uwzględnieniem pożyczki, różnicę umieszcza w miejscu przeznaczenia (cel).

cel := cel - (źródło + CF)

sbb edx, zmienna

sbb eax, [ebx+esi\*4]

sbb rax, rdx

**Uwaga:**

Jeśli źródło jest adresowane w trybie prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

13

Wpływ na flagi: OxxxxC

## Instrukcja MUL

mul źródło

Mnoży bez znaku zawartość akumulatora (al, ax, eax, rax) i źródła, wynik umieszcza w miejscu przeznaczenia (ax, dx:ax, edx:eax, rdx:rax). Flagi CF i OF są zerem, jeśli starsza połowa bitów wyniku jest równa zero.

wynik := acc \* źródło

mul zmienna

mul word ptr[ebx + esi\*4]

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

14

## Przykład – oblicz pole prostokąta o bokach a, b

```

mov  eax, a          ;a
mul  b              ;a * b
jc   poza_int       ;jeśli przekroczyony zakres
mov  pole, eax      ;zapisz
...
...
poza_int: ...

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

15

Wpływ na flagi: -----  
Wymaga BM12

## Instrukcja MULX

mulx cel1, cel2, źródło

Mnoży bez znaku zawartość rejestru edx | rdx i źródła, wynik umieszcza w rejestrach celu(cel1:cel2).

cel1:cel2 := e(r)dx \* źródło

mulx ecx, ebx, [tab + esi\*4]

mulx rcx, rbx, rax

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

16

## Instrukcja IMUL -1

imul źródło

Mnoży ze znakiem zawartość akumulatora (al, ax, eax, rax) i źródła, wynik umieszcza w miejscu przeznaczenia (ax, dx:ax, edx:eax, rdx:rax). Flagi CF i OF są zerem, jeśli iloczyn mieści się w młodszej połowie bitów wyniku.

wynik := acc \* źródło

```

imul zmienna
imul ecx

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

17

Wpływ na flagi: OxxxxC

## Instrukcja IMUL -2

imul cel, źródło

Mnoży ze znakiem zawartość rejestru celu (16|32|64 bity) przez źródło, wynik umieszcza w miejscu przeznaczenia (cel). Flagi CF i OF są zerem, jeśli iloczyn mieści się w rejestrze celu.

cel := cel \* źródło

```

imul eax, zmienna
imul rdx, rcx

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

18

Wpływa na flagi: OxxxxC

## Instrukcja IMUL -3

imul cel, źródło1, źródło2

Mnoży ze znakiem zawartość źródła1 (16|32|64 bity) przez źródło2 (stałą), wynik umieszcza w miejscu przeznaczenia (cel). Flagi CF i OF są zerem, jeśli iloczyn mieści się w rejestrze celu.

cel := źródło1 \* źródło2

imul eax, zmienna, 5

imul cx, dx, 77

### Uwaga:

Jeśli źródło2 jest adresowane w trybie prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

19

## Przykład – oblicz sumę krawędzi prostopadłościanu

mov	eax, a	;wczytaj a
add	eax, b	;dodaj b
add	eax, c	;dodaj c
imul	eax, eax, 4	;x4

## Instrukcja DIV

div źródło

Dzieli bez znaku zawartość AX, DX:AX, EDX:EAX, RDX:RAX przez źródło, iloraz umieszcza w AL, AX, EAX, RAX a resztę w AH, DX, EDX, RDX.

div byte ptr zmienna

div ebx

Wpływa na flagi: xxxxxx

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

21

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

22

## Instrukcja IDIV

idiv źródło

Dzieli ze znakiem zawartość AX, DX:AX, EDX:EAX, RDX:RAX przez źródło, iloraz umieszcza w AL, AX, EAX, RAX a resztę w AH, DX, EDX, RDX.

idiv byte ptr zmienna

idiv ebx

Wpływa na flagi: xxxxxx

## Instrukcja INC

inc cel

Zwiększa zawartość celu o 1.

inc zmienna

inc edx

inc rcx

Wpływa na flagi: OSZAP

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

23

## Instrukcja DEC

dec cel

Zmniejsza zawartość celu o 1.

dec zmienna

dec edx

dec r8

Wpływa na flagi: OSZAP

Wpływa na flagi: OSZAPC

## Instrukcja NEG

neg cel

Zmienia znak celu w kodzie U2.

cel := -cel

```
neg ax
neg byte ptr[ebx+esi*4]
neg r11
```

Flaga CF=o, tylko dla argumentu=o.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

25

Wpływa na flagi: OSZAPC

## Instrukcja CMP

cmp źródło1, źródło2

Porównuje zawartość źródła1 i źródła2, wynik nie jest zapamiętywany, tylko są ustawiane flagi.

źródło1-źródło2

```
cmp ax, zmienna
cmp edx, [ebx+esi*4]
cmp rcx, 123
```

Uwaga:

Jeśli źródło2 jest adresowane w trybie prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

26

## Przykład – oblicz sumę kwadratów liczb w tablicy

```
mov eax, 0          ;wartość początkowa sumy
mov esi, eax        ;indeks
mov ebx, offset wektor ;tablica liczb całkowitych
mov ecx, 123         ;licznik
petla: mov edx, [ebx+esi*4]
    imul edx, edx      ;kwadrat liczby
    add eax, edx        ;suma
    inc esi
    dec ecx
    jnz petla           ;wynik w eax
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

27

## Instrukcje arytmetyczne BCD

- DAA korekta upakowanego kodu BCD po dodawaniu  
Decimal adjust after addition
- DAS korekta upakowanego kodu BCD po odejmowaniu  
Decimal adjust after subtraction
- AAA ASCII korekta po dodawaniu  
ASCII adjust after addition
- AAS ASCII korekta po odejmowaniu  
ASCII adjust after subtraction
- AAM ASCII korekta po mnożeniu  
ASCII adjust after multiplication
- AAD ASCII korekta przed dzieleniem  
ASCII adjust before division

!!! Nie działają w trybie 64 bitowym !!!

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

28

## Instrukcja DAA

daa

Korekta upakowanego kodu BCD (w AL) po dodawaniu. Polega na dodaniu 6 najpierw do młodszego półbijta, a potem do starszego, jeśli ich zawartości były większe od 9 lub wystąpiło przeniesienie AF (CF)

daa

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

29

Wpływa na flagi: OSZAPC

## Instrukcja DAS

das

Korekta upakowanego kodu BCD (w AL) po odejmowaniu. Polega na odjęciu 6 najpierw od młodszego półbijta, a potem od starszego, jeśli ich zawartości były większe od 9 lub wystąpiło przeniesienie AF (CF).

das

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

30



## Instrukcja AAA

**aaa**

Korekta nieupakowanego kodu BCD (w AL) po dodawaniu.  
Polega na dodaniu 6 do młodszej półabajta i 1 do AH, jeśli zawartość AL była większa od 9 lub wystąpiło przeniesienie AF.

**aas**



## Instrukcja AAS

**aas**

Korekta nieupakowanego kodu BCD (w AL) po odejmowaniu. Polega na odjęciu 6 od młodszej półabajta i 1 od AH, jeśli zawartość AL. była większa od 9 lub wystąpiło przeniesienie AF.

**aas**

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

31

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

32



## Instrukcja AAM

**aam**

Korekta nieupakowanego kodu BCD (w AX) po mnożeniu.  
Polega na jednoczesnym wykonaniu:

AH=AL div 10

AL:=AL Mod 10.

**aam**



## Instrukcja AAD

**aad**

Korekta nieupakowanego kodu BCD (w AX) przed dzieleniem. Polega na jednoczesnym wykonaniu:

AL:=AH<sup>2</sup>10+AL.

AH=0

**aad**

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

33

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

34



## Przykład – dodawanie liczb BCD

```

mov al, 0
add al, al      ;CF=o
petla: mov al, [esi]    ;pobierz cyfrę źródła
      adc al, ds:[edi]  ;dodaj cyfrę celu z przeniesieniem
      aaa                 ;korekta
      mov ds:[edi], al   ;zapamiętaj cyfrę
      inc esi             ;następna cyfra
      inc edi
      dec ecx
      jnz petla          ;CF nie zmieniło się od AAA!!!
  
```

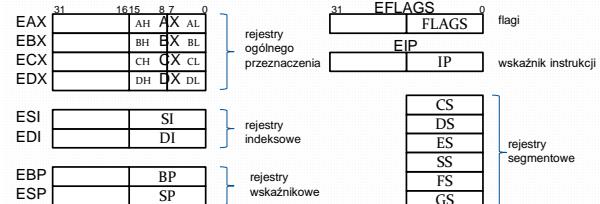
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

35

## Instrukcje logiczne, przesunięć i rotacji

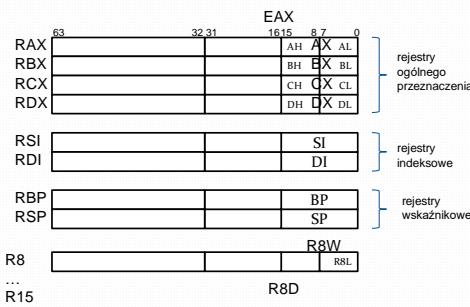
## IA32- rejesty



(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

2

## EM64T- rejesty



(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

3

## Rejestr flag



Rejestr flag w architekturze Intel x86			
bit	Skróty/wartość	opis	typ
0	CF	Flaga przeniesienia (carry)	S
2	PF	Flaga parzystości (parity)	S
4	AF	Flaga wyrównania (adjust)	S
6	ZF	Flaga zera (zero)	S
7	SF	Flaga znaku (sign)	S
10	DF	Flaga kierunku (direction)	C
11	OF	Flaga przepchnięcia (overflow)	S

S: Znacznik stanu  
C: Znacznik kontrolny  
X: Znacznik systemowy

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

4

## Instrukcje logiczne

- AND bitowa funkcja AND
- ANDN bitowa funkcja AND z negacją
- OR bitowa funkcja OR
- XOR bitowa funkcja OR
- NOT bitowa funkcja NOT

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

5

Wpływ na flagi: OSZAPC  
OSZxP0

## Instrukcja AND

and cel, źródło

Wyznacza iloczyn logiczny zawartości źródła i celu (bit po bieżącym), wynik umieszcza w miejscu przeznaczenia (cel).

cel := cel and źródło

and eax, zmienna

and edx, [ebx+esi\*4]

and rax, rdx

Uwaga:  
Jeśli źródło jest adresowane w trybie  
prostym może mieć do 32 bitów.

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

6

Wpływa na flagi: OSZAPC  
OSZxx0  
Wymagane BMI

## Instrukcja ANDN

andn cel, źródło1, źródło2

Wyznacza iloczyn logiczny zanegowanego źródła1 i źródła2 (bit po bicie), wynik umieszcza w miejscu przeznaczenia (cel). Cel i źródło1 są rejestrami 32|64 bitowymi.

cel := (not źródło1) and źródło2

andn edx, eax, zmienna

andn eax, edx, [ebx+esi\*4]

andn r8, rx, rdx

	źródło1	31	0	24	0
	źródło2	0	1	0	0
cel		0	0	1	0

.....

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

7

Wpływa na flagi: OSZAPC  
OSZxP0

## Instrukcja OR

or cel, źródło

Wyznacza sumę logiczną zawartości źródła i celu (bit po bicie), wynik umieszcza w miejscu przeznaczenia (cel).

cel := cel or źródło

or eax, zmienna

or edx, [ebx+esi\*4]

or rax, r9

Uwaga:  
Jeśli źródło jest adresowane w trybie  
prostym może mieć do 32 bitów.

cel	7	0
źródło	1	1
cel	0	1

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

8

Wpływa na flagi: OSZAPC  
OSZxP0

## Instrukcja XOR

xor cel, źródło

Wyznacza, bit po bicie, sumę modulo 2 zawartości źródła i celu wynik umieszcza w miejscu przeznaczenia (cel).

cel := cel xor źródło

xor eax, zmienna

xor edx, [ebx+esi\*4]

xor rax, r9

Uwaga:  
Jeśli źródło jest adresowane w trybie  
prostym może mieć do 32 bitów.

cel	7	0
źródło	1	1
cel	0	1

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

9

Wpływa na flagi: -

## Instrukcja NOT

not cel

Wyznacza negację logiczną zawartości i celu (bit po bicie), wynik umieszcza w miejscu przeznaczenia (cel).

cel := not cel

not eax

not byte ptr [ebx+esi\*4]

not rdx

cel	7	0
	1	1
cel	0	1

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

10

Wpływa na flagi: OSZAPC  
OSZxP0

If ((!a&&b) || (c&&d&&e))  
{...}else {...};

```

mov al, a      mov al, a      mov dl, c
not al        andn al, al, b    mov al, a
and al, b      jnz then_1     and dl, d
mov dl, c      mov dl, c      andn al, al, b
and dl, d      and dl, d     and dl, e
and dl, e      and dl, e     or al, dl
or al, dl      or al, dl     jz else_1
jz else_1      jz else_1     then_1:
then_1:          then_1:

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

11

Wpływa na flagi: OSZAPC  
OSZxP0

## Instrukcje przesunięć i rotacji

- SAR przesunięcie arytmetyczne w prawo
- SHR przesunięcie logiczne w prawo
- SAL przesunięcie arytmetyczne w lewo
- SHL przesunięcie logiczne w lewo
- SARX przesunięcie arytmetyczne w prawo
- SHRX przesunięcie logiczne w prawo
- SHLX przesunięcie logiczne w lewo
- SHRD przesunięcie w prawo double
- SHLD przesunięcie w lewo double
- ROR rotacja w prawo
- ROL rotacja w lewo
- RCR rotacja w prawo przez przeniesienie
- RCL rotacja w lewo przez przeniesienie
- RORX rotacja w prawo

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

12

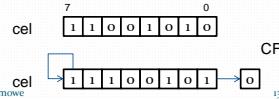


## Instrukcja SAR

**sar cel, ile**

Przesunięcie arytmetyczne celu w prawo o ile bitów. Ile=1, cl lub wartość 0-31|63.

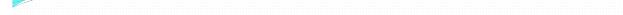
```
sar    eax, 1
sar    [ebx+esi*4], cl
sar    rdx, cl
```



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

13



## Instrukcja SHR

**shr cel, ile**

Przesunięcie logiczne w prawo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
shr    eax, 1
shr    [ebx+esi*4], cl
shr    rdx, cl
```



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

14

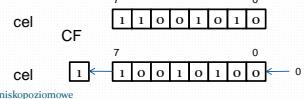


## Instrukcja SAL

**sal cel, ile**

Przesunięcie arytmetyczne celu w lewo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
sal    eax, 1
sal    [ebx+esi*4], cl
sal    rdx, cl
```



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

15

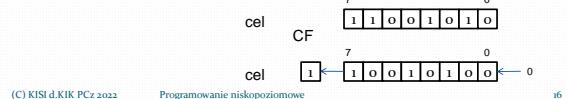


## Instrukcja SHL

**shl cel, ile**

Przesunięcie logiczne celu w lewo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
shl    eax, 1
shl    [ebx+esi*4], cl
shl    rdx, cl
```



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

16



## Przykład – oblicz sumę krawędzi prostokątnego

```
mov    eax, a          ;wczytaj a
add    eax, b          ;dodaj b
add    eax, c          ;dodaj c
sal    eax, 2           ;×4
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

17



## Instrukcja SARX

**sarx cel, źródło, ile**

Przesunięcie arytmetyczne źródła w prawo o ile bitów i zapisanie w celu. Cel i ile są rejestrami 32|64 bitowymi.

```
sarx  eax, zmieniona, edx
sarx  eax, [ebx+esi*4], ecx
sarx  rdx, rax, rcx
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

18



## Instrukcja SHRX

shrx cel, źródło, ile

Przesunięcie logiczne źródła w prawo o ile bitów i zapisanie w celu. Cel i ile są rejestrami 32|64 bitowymi .

```
shrx eax, zmienna, edx
shrx eax, [ebx+esi*4], ecx
shrx rdx, rax, rcx
```

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

19



## Instrukcja SHLX

shlx cel, źródło, ile

Przesunięcie logiczne źródła w lewo ile bitów i zapisanie w celu. Cel i ile są rejestrami 32|64 bitowymi .

```
shlx eax, zmienna, edx
shlx eax, [ebx+esi*4], ecx
shlx rdx, rax, rcx
```

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

20



## Instrukcja SHRD

shrd cel, źródło, ile

Przesunięcie źródła:celu w prawo o ile bitów. Ile=cl lub wartość 0-31|64. Rejestr źródła (16,32,64) pozostaje bez zmian.

```
shrd eax, ecx, 15
shrd [ebx+esi*4], edx, cl
shrd zmienna, rax, cl
```



(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

21

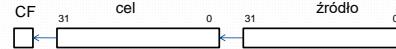


## Instrukcja SHLD

shld cel, źródło, ile

Przesunięcie źródła:celu w lewo o ile bitów. Ile=cl lub wartość 0-31|63. Rejestr źródła (16,32,64) pozostaje bez zmian.

```
shld eax, ecx, 15
shld [ebx+esi*4], edx, cl
shld zmienna, rax, cl
```



(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

22



## Przykład – podziel 256-bitową liczbę a przez 64

```
mov rcx, offset a ;wczytaj adres a
mov rax, [rcx+8] ;wczytaj drugie 64 bity
shrd [rcx], rax, 6 ;przesuń pierwsze
mov rax, [rcx+16] ;wczytaj trzecie 64 bity
shrd [rcx+8], rax, 6 ;przesuń drugie
mov rax, [rcx+24] ;wczytaj czwarte 64 bity
shrd [rcx+16], rax, 6 ;przesuń trzecie
shr rax, 6 ;przesuń czwarte
mov [rcx+24], rax ;zapisz
```

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

23



## Instrukcja ROR

ror cel, ile

Rotacja (obrót) celu w prawo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
ror eax, 1
ror [ebx+esi*4], cl
ror rdx, cl
```

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

24

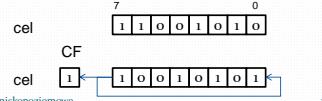


## Instrukcja ROL

**rol cel, ile**

Rotacja (obrót) celu w lewo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
rol    eax, 1
rol    [ebx+esi*4], cl
rol    rdx, cl
```



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

25

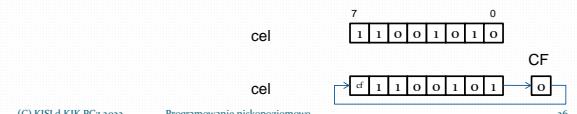
Wpływ na flagi: OSZAPC  
(0x)---C

## Instrukcja RCR

**rcr cel, ile**

Rotacja (obrót) przez przeniesienie celu w prawo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
rcr    eax, 1
rcr    [ebx+esi*4], cl
rcr    rdx, cl
```



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

26

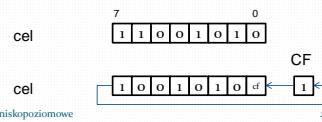


## Instrukcja RCL

**ror cel, ile**

Rotacja (obrót) przez przeniesienie celu w lewo o ile bitów. Ile=1, cl lub wartość 0-31|63.

```
rcl    eax, 1
rcl    [ebx+esi*4], cl
rcl    rdx, cl
```



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

27



## Instrukcja RORX

**rorx cel, źródło, ile**

Rotacja źródła w prawo o ile bitów i zapisanie w celu. Cel jest rejestrem 32|64 bitowym. Ile przyjmuje wartość 0-31|63.

```
rorx  eax, zmienna, 12
rorx  eax, [ebx+esi*4], 7
rorx  rdx, rx, 44
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

28



```
and  eax, 0fffffefffh ;zeruje 12 bit eax
or   ax, 0100oh      ;ustawia 12 bit ax
xor  rax, 0100oh     ;neguje 12 bit rax
and  eax, eax        ;m. in. zeruje CF
xor  eax, eax        ;zeruje eax
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

29



```
and  eax, 070h       ;maska
sar  eax, 4          ;eax - 3bitowa liczba
mov  ah, al          ;zamienia al na jego
and  ax, ofoofh      ;szesnastkową reprezentację
shr  ah, 4            ;ASCII w ah, al.
add  ax, 3030h
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

30

## Przykłady - \*4,25

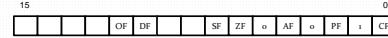
movsx	rax, zmienna	;typu int – 32 bity
mov	rdx, rax	;powiel
sal	rax, 2	;x4
sar	rdx, 2	;/4
adc	rax, rdx	;x4¹/₄
mov	zmienna, eax	;zapisz
 movsx	rdx, eax	; test
cmp	rax, rdx	;porównaj
jnz	blad	;przekroczenie zakresu

## Przykłady

invb	proc	;odwracanie bitów z rcx; rdx liczba odwracanych bitów 1-64	
	xor	rax, rax	;rax:=0
	dec	rdx	;eliminacja zera
	js	@e	
	and	rdx, 3fh	;ograniczenie zakresu
@p:	shr	rcx, 1	;lsb->CF
	rcl	rax, 1	;CF->msb
	dec	rdx	;licznik--
	jns	@p	;następny jeśli nieujemne
@e:	ret		
	invb	endp	

## Instrukcje warunkowe i skoku

### Rejestr flag



Rejestr flag w architekturze Intel x86			
bit	Skrot/wartosc	opis	typ
0	CF	flaga przeniesienia (carry)	S
2	PF	flaga parzystosci (parity)	S
4	AF	flaga wynownia (adjust)	S
6	ZF	flaga zera (zero)	S
7	SF	flaga znaku (sign)	S
10	DF	flaga kierunku (direction)	C
11	OF	flaga przepelnilenia (overflow)	S

S: Znacznik stanu  
C: Znacznik kontrolny  
X: Znacznik systemowy

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

2

## Warunki dotyczące flag

• E/Z	equal/ zero	ZF=1
• NE/NZ	not equal/ not zero	ZF=o
• C	carry	CF=1
• NC	not carry	CF=o
• O	overflow	OF=1
• NO	not overflow	OF=o
• S	sign (negative)	SF=1
• NS	not sign (non-negative)	SF=o
• P/PE	parity/ parity even	PF=1
• NP/PO	not parity/ parity odd	PF=o

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

3

## Warunki porównania liczb

• E/Z	equal/ zero	ZF=1
• NE/NZ	not equal/ not zero	ZF=o

Dla liczb bez znaku:

• A/NBE	above/ not below or equal	CF=o i ZF=o
• AE/NB	above or equal/ not below	CF=o
• B/NAE	below/ not above or equal	CF=1
• BE/NA	below or equal/ not above	CF=1 lub ZF=1

Dla liczb ze znakiem

• G/NLE	greater/ not less or equal	ZF=o i SF=OF
• GE/NL	greater or equal/ not less	SF=OF
• L/NGE	less/ not greater or equal	SF>OF
• LE/NG	less or equal/ not greater	ZF=1 lub SF>OF

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

4

Wpływ na flagi: -

## Instrukcja CMOVcc

CMOVcc cel, źródło

Jeśli jest spełniony warunek cc, przesyła źródło do miejsca przeznaczenia (rejestr 16, 32 lub 64 bitowy). Instrukcja wprowadzona w procesorach rodziny P6!

if cc then cel:=źródło

```

cmovz      eax, zmienna
cmovge    edx, [ebx+esi*4]
cmovna    rax, rdx

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

5

## Instrukcje CMOVcc

• CMOVE/CMOVZ	Przeslij jezeli equal/ zero
• CMOVNE/CMOVNZ	Przeslij jezeli not equal/ not zero
• CMOVA/CMOVNBE	Przeslij jezeli above/ not below or equal
• CMOVAE/CMOVNBE	Przeslij jezeli above or equal/ not below
• CMOVB/CMOVNAE	Przeslij jezeli below/ not above or equal
• CMOVB/CMOVNA	Przeslij jezeli below or equal/ not above
• CMOVG/CMOVNLE	Przeslij jezeli greater/ not less or equal
• CMOVG/CMOVNL	Przeslij jezeli greater or equal/ not less
• CMOVL/CMOVNGE	Przeslij jezeli less/ not greater or equal
• CMOVLE/CMOVNG	Przeslij jezeli less or equal/ not greater
• CMOVC	Przeslij jezeli carry
• CMOVNC	Przeslij jezeli not carry
• CMOVO	Przeslij jezeli overflow
• CMOVNO	Przeslij jezeli not overflow
• CMOVS	Przeslij jezeli sign (negative)
• CMOVNS	Przeslij jezeli not sign (non-negative)
• CMOVP/CMOVPE	Przeslij jezeli parity/ parity even
• CMOVNP/CMOVPO	Przeslij jezeli not parity/ parity odd

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

6

## Przykład

```
MyMax64 proc
movsx  rax, ecx
movsx  rdx, edx
cmp    rax, rdx
cmovl rax, rdx
ret
MyMax64 endp
```

```
function
TForm1.MyMax(x,y:integer):integer;
asm
  mov eax, x
  cmp eax, y
  jnc @@exit
  mov eax, y
@@exit:
end;
```

```
function
TForm1.MyMax2(x,y:integer):integer;
asm
  mov eax, x
  cmp eax, y
  cmovc eax, y ;cmovb eax,y
end;
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

7

## Skoki warunkowe Jcc

- JE/JZ Skocz jeśli equal/zero
- JNE/JNZ Skocz jeśli not equal/not zero
- JA/JNBE Skocz jeśli above/not below or equal/not below
- JA/JNB Skocz jeśli below/not above or equal/not below
- JB/JNAE Skocz jeśli below or equal/not above
- JBE/JNA Skocz jeśli less/less or equal/not above
- JG/JNLE Skocz jeśli greater/not less or equal
- JGE/JNL Skocz jeśli greater or equal/not less
- JL/JNGE Skocz jeśli less/not greater or equal
- JLE/JNG Skocz jeśli less or equal/not greater
- JC Skocz jeśli carry
- JNC Skocz jeśli not carry
- JO Skocz jeśli overflow
- JNO Skocz jeśli not overflow
- JS Skocz jeśli sign (negative)
- JNS Skocz jeśli not sign (non-negative)
- JPO/JNP Skocz jeśli parity odd/not parity
- JPE/JP Skocz jeśli parity even/parity

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

8

## Instrukcja JZ

**jz** przesunięcie

Przeskakuje do podanej etykiety (adres jest względny 16/32bitowy).

EIP := EIP + przesunięcie

**jz** dalej

...

dalej: ...

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

9

Wpływ na flagi: -

## Przykład

```
procedure MySort(tarray of integer;n:integer);
asm
  push edi ;zabezpiecz rejestrę
  push esi
  mov esi, n ;liczba
  dec esi ;esi ostatni element
  mov edx, t ;adres tablicy
  Sortowanie @p: mov edi, esi
  dec edi ;poprzedzający element
  mov eax, [edx+esi*4] ;ostatni do eax
  @w: cmp eax, [edx+edi*4] ;czy >=
  jae @@a
  xchg eax, [edx+edi*4] ;zamień elementy
  mov [edx+esi*4], eax
  @@a: dec edi ;pętla wewnętrzna
  @w: jns @w
  dec esi ;pętla główna
  jnz @p
  pop esi ;przywróć rejestrę
  pop edi
end;
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

10

Wpływ na flagi: -

## Instrukcja JMP

**jmp** adres

Przeskakuje do podanej etykiety (adres jest względny 8/16/32bitowy lub bezwzględny).

EIP := EIP + przesunięcie(adres)

CS := segment(adres); EIP := EIP + przesunięcie(adres)

**jmp** dalej

**jmp** eax

**jmp** [esi]

**jmp** lib1:dalej1

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

12

## Instrukcje sterujące przebiegiem programu +

- **JMP** Skok bezwarunkowy
- **JCXZ/JECXZ/JRCX** Skok jeśli zero w rejestrze CX/ECX/RCX
- **LOOP** Pętla z licznikiem CX/ECX/RCX
- **LOOPZ/LOOPE** Pętla z licznikiem CX/ECX/RCX i zero/equal
- **LOOPNZ/LOOPNE** Pętla z licznikiem CX/ECX/RCX i not zero/not equal
- **CALL** Wywołanie podprogramu
- **RET** Powrót z podprogramu
- **IRET** Powrót z podprogramu obsługi przerwania
- **INT** Przerwanie programowe
- **INTO** Przerwanie przy przekroczeniu zakresu
- **BOUND** sprawdzenie ograniczeń indeksu tablicy
- **ENTER** Wysokopoziomowe wejście do podprogramu – utworzenie ramy stosu
- **LEAVE** Wysokopoziomowe wyjście z podprogramu – usunięcie ramy stosu

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

11



Wpływ na flagi: -

## Instrukcja JCXZ/JECXZ/JRCXZ

JCXZ/JECXZ/JRCXZ przesunięcie

Skok jeśli zero w rejestrze CX/ECX/RCX do podanej etykiety (adres jest wzgórny 16/32/64 bitowy).

EIP := EIP + przesunięcie

petla: ...

jecxz dalej

...

jmp petla

dalej: ...

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

13



## Przykład

Silnia

```
function silnia(n:integer):integer;
asm
  mov  ecx, eax
  dec  ecx
  @p: imul eax, ecx
  dec  ecx
  jnz  @p
end;

function silniai(n:integer):integer;
asm
  mov  ecx, eax
  @p: dec  ecx
  jecxz @e
  imul eax, ecx
  jmp  @p
@e:
end;

function silniaaz(n:integer):integer;
asm
  mov  ecx, eax
  dec  ecx
  @p: imul eax, ecx
  loop @p
end;
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

15



Wpływ na flagi: -

## Instrukcja LOOP

loop przesunięcie

Pętla z licznikiem CX/ECX/RCX. Zmniejsza CX/ECX/RCX o 1 i jeśli nie uzyskano zera przeskakuje do podanej etykiety (adres jest wzgórny 8 bitowy).

EIP := EIP + przesunięcie(adres)

petla: ...

...

loop petla

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

14



## Instrukcja LOOPZ/LOOPE

loopz/loope przesunięcie

Pętla z licznikiem CX/ECX/RCX i zero/equal. Zmniejsza CX/ECX/RCX o 1 i jeśli nie uzyskano zera w CX/ECX/RCX i flaga ZF=1 przeskakuje do podanej etykiety (adres jest wzgórny 8 bitowy).

EIP := EIP + przesunięcie(adres)

petla:...

...

loopz petla

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

17



Wpływ na flagi: -

## Instrukcja LOOPNZ/LOOPNE

loopnz/loopne przesunięcie

Pętla z licznikiem CX/ECX/RCX i nie zero/equal. Zmniejsza CX/ECX/RCX o 1 i jeśli nie uzyskano zera i flaga ZF=0 przeskakuje do podanej etykiety (adres jest wzgórny 8 bitowy).

EIP := EIP + przesunięcie(adres)

petla: ...

...

loopnz petla

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

18

Wpływ na flagi: -

## Instrukcja CALL

call adres

Instrukcja call wywołuje podprogram, wysyła na stos adres powrotu EIP|RIP lub CS:EIP|RIP. Parametr adres wpisuje do EIP|RIP lub CS:EIP|RIP.

```
push e(r)ip; (push cs);
E(R)IP := przesunięcie adres; (CS := selektor segmentu)
```

call procedura

call eax

call [esi]

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

19

Wpływ na flagi: -

## Instrukcja RET

ret (ile)

Instrukcja ret wraca z podprogramu, pobiera ze stosu adres powrotu do EIP|RIP lub CS:EIP|RIP. Jeśli posiada parametr ile, to dodatkowo usuwa ze stosu ile bajtów (niepotrzebne już parametry aktualne wywołania).

```
pop e(r)ip; (pop cs); (e(r)sp:=e(r)sp+ile)
```

ret

ret 6

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

20

## Przykład

Silnia

```
function silnia(n:integer):integer;
asm
  and eax,eax
  cmovz eax,one
  jz @e
  push eax
  dec eax
  call silnia3
  pop edx
  imul eax,edx
@e:
end;

one db 1,0,0,0,0,0,0,0;
silnia4 proc;
  cmp rcx,1
  cmovbe rax,one
  jbe @e
  push rcx
  dec rcx
  call silnia4
  pop rcx
  imul rax,rcx
@e: ret
silnia4 endp;
```

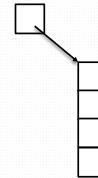
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

21

## Przykład

Tablice - wektory



Zmienna tablicowa jest wskaźnikiem na początek obszaru pamięci, który zawiera tablicę.

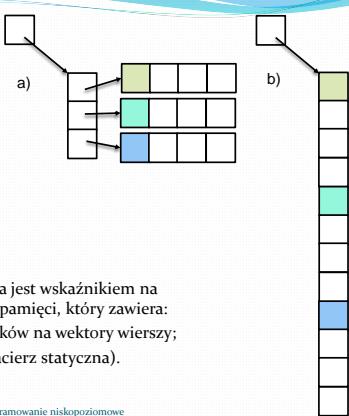
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

22

## Przykład

Tablice – Macierze Dynamiczne i statyczne



Zmienna tablicowa jest wskaźnikiem na początek obszaru pamięci, który zawiera:  
a) wektor wskaźników na wektory wierszy;  
b) całą tablicę (macierz statyczna).

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

23

## Przykład

;rcx – wskaźnik do macierzy dynamicznej  
;rdx – liczba wierszy  
;r8 – liczba kolumn  
suma\_et proc;

```
xor rax, rax ;suma=0
@pz: mov r9, r8 ;liczba kolumn
      mov r10, [rcx+8*rdx-8];adr. wiersza
      @pw: movsx r11, [r10+4*r9-4]
            add rax, r11
            dec r9 ;liczba kolumn-
            jnz @pw
            dec rdx ;liczba wierszy-
            jnz @pz
            ret
suma_et endp;
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

24



## Instrukcja INT

int nr

Wywołuje przerwanie programowe o numerze nr (0-255). Numery z zakresu 0-31 są zarezerwowane. Instrukcja int działa podobnie do instrukcji call, jednak dodatkowo wysyła na stos flagi i wchodzącą do podprogramu część z nich zeruje.

int z1h

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe 25

Wpływ na flagi: -



Wektor Nr	Mnemonik	Opis	Źródło
0	#DE	Błąd dzielenia	Instrukcja DIV i IDIV
1	#DB	Debugger	Dowolne odwołanie do kodu i danych.
2	#NP	Przerwanie NMI	Przerwanie niemaskowne Non-maskable external.
3	#BP	Breakpoint	Instrukcja INT3
4	#OF	Błąd OF	Instrukcja INTO.
5	#BR	BOUND przekroczyony zakres	Instrukcja BOUND.
6	#UD	Błąd kod	Instrukcja UD2 lub zarezerwowany kod.
7	#NM	Urządzenie nie dostępne (Brak koprocesora)	Instrukcje zmiennoprzecinkowe lub WAIT/FWAIT.
8	#DF	Błąd Double	Dowolna instrukcja generująca wyjątek, NMI lub INTR.
9	#MF	CoProcessor Segment	Instrukcje zmiennoprzecinkowe 2
10	#TS	Błędny TSS	Przelaczanie zadań lub dostęp do TSS.
11	#NP	Segment nieobecny	Ladowanie rejestrów segmentowych lub dostęp do segmentów systemowych.
12	#SS	Segment stacji uszkodzony	Odczytywanie/wstawianie rejestrów SS.
13	#GP	Ogólna ochrona	Dowolne odwołanie do pamięci i inne zabezpieczenia.
14	#PF	Błąd strony	Dowolne odwołanie do pamięci.
15		Zarezerwowane	
16	#MF	Błąd zmiennoprzecinkowy (błąd matematyczny)	Instrukcje zmiennoprzecinkowe lub WAIT/FWAIT.
17	#AC	Kontrola wyrownania	Dowolne odwołanie do danych w pamięci.
18	#MC	Kontrola maszyny	Kod błędu (o ile występuje) i źródło zależą od modelu.
19	#XM	Wysokość	Instrukcje zmiennoprzecinkowe SIMDs
20-31		Zarezerwowane	
32-255		Przerwanie maskowne	Przerwanie zewnętrzne INTR lub instrukcje INT n.

1. Instrukcja UD2 została wprowadzona w procesorze Pentium Pro. z. Procesory IA-32 po procesorze Intel®® nie generują tego wyjątku.

2. Wyjątek wprowadzony w procesorze Intel®®. Wyjątek wprowadzony w procesorze Pentium i poprawiony w procesorach rodziny P6.

3. Wyjątek wprowadzony w procesorze Pentium III.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

26



## Instrukcja INTO

into

Wywołuje przerwanie programowe (4) w przypadku ustawienia flagi OF (nadmiaru).

into

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe 27

Wpływ na flagi: -



## Instrukcja IRET/IRETD/IRETQ

iret/iretd/iretq

Instrukcja iret/iretd/iretq wraca z podprogramu obsługi przerwania, pobiera ze stosu adres powrotu do CS:EIP|RIP oraz flagi zachowane przy wywołaniu przerwania.

pop e(r)ip; pop cs; pop (e(r)flags)

iret

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe 28



## Instrukcja BOUND

bound idx, gr

Sprawdza, czy indeks tablicy (wartość 16/32 bitowa ze znakiem) zawarty w rejestrze idx nie przekracza jej granic określonych przez strukturę w pamięci gr złożoną z granicy dolnej i górnej. W przypadku przekroczenia granicy generowany jest wyjątek przekroczenia granicy tablicy.

bound eax, granice1

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe 29

Wpływ na flagi: -



## Instrukcja ENTER

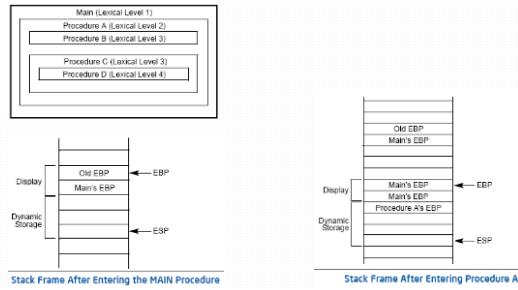
enter storage, level

Tworzy ramę stosu w podprogramie z uwzględnieniem poziomu zagnieżdżenia podprogramów lokalnych (level) i rozmiaru w bajtach zmiennych lokalnych (storage). Na stosie umieszcza wskaźniki ram stosu: podprogramu wywołującego, wszystkich poziomów nadrzednych i własnych.

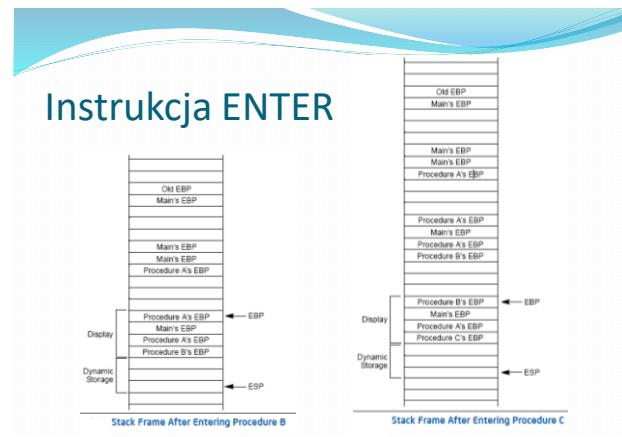
```
PUSH EBP;
FRAME_PTR --> ESP;
IF LEVEL > 0 THEN
    DO (LEVEL - i) times
        EBP --> EBP - 4;
        PUSH Pointer(EBP); (* doubleword wskazywane przez EBP *)
        OD;
        PUSH FRAME_PTR;
    FI;
    EBP --> FRAME_PTR;
    ESP --> ESP - STORAGE;
```

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe 30

## Instrukcja ENTER



(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe



(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

32

## Instrukcja LEAVE

leave

Usuwa ramę stosu (utworzoną instrukcją ENTER) przed wyjściem z podprogramu. Przypisuje do wskaźnika stosu rejestr bazowy, następnie zdejmuję rej. basowy ze stosu.

esp:=ebp; pop ebp

leave

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

## Przykład

```
procedure Sort(var A: array of Integer);
```

```

procedure QuickSort(var A: array of Integer;
var
  Lo, Hi, Mid: TInteger;
begin
  Lo := iLo;
  Hi := iHi;
  Mid := A[Lo + (Hi - Lo) div 2];
  repeat
    while A[Lo] <= Mid do Inc(Lo);
    while A[Hi] >= Mid do Dec(Hi);
    if Lo <= Hi then
      begin
        T := A[Lo];
        A[Lo] := A[Hi];
        A[Hi] := T;
        Inc(Lo);
        Dec(Hi);
      end;
    until Lo > Hi;
    if Hi > iLo then QuickSort(A, iLo, Hi);
    if Lo < iHi then QuickSort(A, Lo, iHi);
  end;
end;

```

```
begin
    QuickSort(A, Low(A), High(A))
end;
```

34

```

procedure mysort(var A:array of integer;iLo,iHi:integer);
asm
  push edi
  push esi
  push ilo
  push ihi
  call @_s
  jmp @_e

@@: enter 0,0           //sortowanie eax-A; [ebp+8]=ihi; [ebp+12]=ilo
  mov esi,[ebp+8]        //esi=ihi
  mov edi,[ebp+12]        //edi=ilo
  mov exes,[eax-1]        //ecx=Mid > A[Lo + Hi] div 2]
  add edx,edi
  sar ecx
  mov exs,[eax+ecx*4]    //mid
  @r:                   //repeat
  cmp [eax+edi*4],ecx   //while A[Lo] < Mid do Inc(Lo)
  jge @_s
  inc edi
  jmp @_r
  @e: cmp [eax+esi*4],ecx //while A[Hi] > Mid do Dec(Hi)
  jle @_s
  dec esi
  jmp @_e

  @b: cmp edi,esi         //if Lo <= Hi then
  jne @_3
  mov edx,[eax+esi*4]    // [A/Low] <> A[Hi]
  xchd [eax+edi*4]
  jmp @_b

  @_3: kld K1 PCz 2022

```

#### skopoziomowe

## Mnożenie BCD

$$\begin{array}{r}
 222 \\
 5678 \\
 \times \quad 3 \\
 \hline
 5814 \\
 17034 \\
 \hline
 17034
 \end{array}$$

## Mnożenie BCD

```

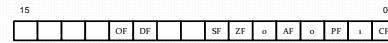
void m_BCD(char*a,char*b,int n, int m, char*w) {
    _asm{
        pushad
        pushf
        mov    ecx, m      //liczba cyfr mnoźnika - m
        mov    edi, b      //adres mnoźnika
        add    edi, ecx
        mov    edx, w      //adres wyniku
        add    edx, ecx
        po_m:
        push   ecx         //zabezpiecz m liczbę cyfr
        //po pozostałych cyfr mnoźnika
        dec    edx         //m do o cyfra wyniku
        dec    edi         //m do o cyfra mnoźnika
        mov    bl, [edi]    //do bl
        mov    ecx, n      //ilość cyfr mnoźnej - m
        mov    esi, a       //adres mnoźnej
        iloczyn:
        mov    al, [esi]    //cyfra mnoźnej - od najstarszej
        mul    bl           //razy cyfra mnoźnika
        aam
        push   ax           //ah - przeniesienie z
        //mnożenia,al - wynik na stos
        inc    esi
        Loop  iloczyn
    }
}

SumaBCD:
    mov    ecx, n      //najmłodsze ze stosu
    pop    ax
    add    al, [edx+ecx] //i dodajemy cyfrę do wyniku
    aaa
    jz    one_n         //jeśli tylko jednocyfrowa mnożna
    SumaBCD:
    mov    bl, ah         //przeniesienie do bl
    pop    ax
    stosu
    add    al, [edx+ecx] //dodajemy kolejną cyfrę wyniku
    aaa
    al,bl             //plus poprzednie przeniesienie
    mov    [edx+ecx], al //i do wyniku
    loop
    one_n:
    mov    [edx], ah     //ostatnie przeniesienie do wyniku
    pop    ecx
    popf
    popad
    } // wynik musi być wyzerowany przed wywołaniem
}

```

# Operacje na znacznikach, bitach i bajtach

## Rejestr flag



Rejestr flag w architekturze Intel x86			
bit	Skrot/wartosc	opis	typ
0	CF	Flaga przeniesienia (carry)	S
2	PF	Flaga parzystosci (parity)	S
4	AF	Flaga wynownania (adjust)	S
6	ZF	Flaga zera (zero)	S
7	SF	Flaga znaku (sign)	S
10	DF	Flaga kierunku (direction)	C
11	OF	Flaga przepelnilenia (overflow)	S

S: Znacznik stanu

C: Znacznik kontrolny

X: Znacznik systemowy

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

2

# Operacje na flagach

- STC Ustawienie CF
- CLC Zerowanie CF
- CMC Zanegowanie CF
- CLD Zerowanie DF – flagi kierunku
- STD Ustawienie DF
- LAHF Przesłanie flag do rejestru AH
- SAHF Przesłanie rejestru AH do flag
- PUSHF/PUSHFD/ Wysłanie flag na stos  
PUSHFQ
- POPF/POPFD/POPFQ Pobranie flag ze stosu
- STI Ustawienie IF – flagi przerwań
- CLI Zerowanie IF

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

3

## Instrukcja STC

Wplywa na flagi: C

stc

Ustawienie flagi CF.

CF := 1

stc

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

4

Wplywa na flagi: C

## Instrukcja CLC

clc

Zerowanie flagi CF.

CF := 0

clc

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

5

Wplywa na flagi: C

## Instrukcja CMC

cmc

Zanegowanie flagi CF.

CF := not CF

cmc

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

6



Wpływa na flagi: D

## Instrukcja STD

std

Ustawienie flagi kierunku DF. Jeżeli DF=1 instrukcje łańcuchowe zmniejszają rejestr ESI lub EDI.

DF := 1

std

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

7



Wpływa na flagi: D

## Instrukcja CLD

cld

Zerowanie flagi kierunku DF. Jeżeli DF=0 instrukcje łańcuchowe zwiększą rejestr ESI lub EDI.

DF := 0

cld

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

8



Wpływa na flagi: -

## Instrukcja LAHF

lahf

Przesłanie flag do rejestru AH

AH := lo(FLAGS)

lahf

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

9



Wpływa na flagi: SZAPC

## Instrukcja SAHF

sahf

Przesłanie rejestru AH do flag. Bity 1,3,5 są ignorowane.

lo(FLAGS) := AH

sahf

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

10



Wpływa na flagi: -

## Instrukcja PUSHF/PUSHFD/PUSHFQ

pushf/pushfd/pushfq

Przesyła zawartość Flag/Eflag/Rflag na stos.

pushf

pushfd

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

11



Wpływa na flagi: OSZAPC

## Instrukcja POPF/POPFD/POPFQ

popf/popfd/popfq

Pobiera zawartość Flag/EFlag ze stosu.

popf

popfd

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

12



## Instrukcja STI

sti

Ustawienie flagi przerwań IF lub VIF. Włącza po następnej instrukcji system przerwań maskowalnych.

IF := 1

sti

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

13

Wpływ na flagi: I

## Instrukcja CLI

cli

Zerowanie flagi przerwań IF lub VIF. Wyłącza system przerwań maskowalnych.

IF := 0

cli

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

14



## Operacje na bitach

- BT Testowanie bitu
- BTS Testowanie bitu z ustawianiem
- BTR Testowanie bitu z zerowaniem
- BTC Testowanie bitu z negacją
- TEST Porównanie logiczne
- BSF Przeszukiwanie bitów w przód
- BSR Przeszukiwanie bitów wstecz
- LZCNT Zlicza zerowe bity od najstarszego
- TZCNT Zlicza zerowe bity od najmłodszego
- BEXTR Wycina ciąg bitów
- BLSI Kopiuje najmłodszy ustawiony bit
- BLSR Zeruje najmłodszy ustawiony bit
- BLMSK Tworzy maskę do bitu=0
- BZHI Zeruje starsze bity

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

15

Wpływ na flagi: OSZAPC  
xxxxxC

## Instrukcja BT

bt baza, nr

Wyznacza wartość bitu nr (rejestr lub wartość) w bazie (rejestr lub zmienna) i umieszcza ją w CF.

CF := bit bazy numer nr

bt zmienna, eax  
bt edx, 12  
bt rcx, 37

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

16



## Instrukcja BTS

bts baza, nr

Wyznacza wartość bitu nr (rejestr lub wartość) w bazie (rejestr lub zmienna) i umieszcza ją w CF. Następnie ustawia badany bit.

CF := bit bazy numer nr

bit bazy numer nr:=1

bts zmienna, eax  
bts edx, 12  
bts rcx, 37

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

17

Wpływ na flagi: OSZAPC  
xxxxxC

## Instrukcja BTR

btr baza, nr

Wyznacza wartość bitu nr (rejestr lub wartość) w bazie (rejestr lub zmienna) i umieszcza ją w CF. Następnie zeruje badany bit.

CF := bit bazy numer nr

bit bazy numer nr := 0

btr zmienna, eax  
btr edx, 12  
btr rcx, 37

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

18

Wpływ na flagi: OSZAPC  
xxxxx

## Instrukcja BTC

**btc** baza, nr

Wyznacza wartość bitu nr (rejestr lub wartość) w bazie (rejestr lub zmienna) i umieszcza ją w CF. Następnie neguje badany bit.

CF := bit bazy numer nr

bit bazy numer nr := not bit bazy numer nr

**btc** zmienna, eax**btc** edx,esi**btc** rcx, 37

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

19

Wpływ na flagi: OSZAPC  
0SZxP0

## Instrukcja TEST

**test** cel, źródło

Wyznacza iloczyn logiczny(bit po bicie) zawartości celu i źródła (rejestr lub wartość), wynik jest pominięty, ustawia flagi.

cel and źródło

**test** eax, zmienna**test** edx, [ebx+esi\*4]

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

20

Wpływ na flagi: OSZAPC  
xxZxx

## Instrukcja BSF

**bsf** cel, źródło

Przeszukiwanie bitów w przód. Szuka w rejestrze lub zmiennej źródła najmłodszego bitu=1, jego indeks umieszcza w rejestrze celu (ZF = 0). Jeśli źródło = 0, wówczas ZF = 1, a cel jest niezdefiniowany

cel := indeks najmłodszego bitu = 1 źródła

**bsf** eax, zmienna**bsf** edx,esi**bsf** rcx, rdx

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

21

Wpływ na flagi: OSZAPC  
xxZxx

## Instrukcja BSR

**bsr** cel, źródło

Przeszukiwanie bitów wstecz. Szuka w rejestrze lub zmiennej źródła najstarszego bitu=1, jego indeks umieszcza w rejestrze celu (ZF = 0). Jeśli źródło = 0, wówczas ZF = 1, a cel jest niezdefiniowany

cel :=indeks najstarszego bitu=1 źródła

**bsr** eax, zmienna**bsr** edx, esi**bsr** rcx, rdx

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

22

Wpływ na flagi: OSZAPC  
xxZxxC  
Wymaga LZCNT

## Instrukcja LZCNT

**lzcnt** cel, źródło

Zlicza starsze (wiodące) zerowe bity źródła (16|32|64) i ilość zapisuje do rejestru celu. Dla celu = 0 ZF = 1. Dla celu = rozmiarowi źródła CF = 1.

cel := liczba wiodących zer w źródle

**lzcnt** eax, zmienna**lzcnt** edx, esi**lzcnt** rcx, rdx

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

23

Wpływ na flagi: OSZAPC  
xxZxxC  
Wymaga BM1

## Instrukcja TZCNT

**tzcnt** cel, źródło

Zlicza od najmłodszego zerowe bity źródła (16|32|64) i ilość zapisuje do rejestru celu. Dla celu = 0 ZF = 1. Dla celu=rozmiarowi źródła CF = 1.

cel := liczba końcowych zer w źródle

**tzcnt** eax, zmienna**tzcnt** edx, esi**tzcnt** rcx, rdx

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

24



## Instrukcja BEXTR

bextr cel, źródło, st\_ile

Wycina z rejestru|zmiennej źródła (32|64) ciąg bitów i umieszcza w rejestrze celu. Początkowy bit określa rejestr st\_ile[7:0], a ilość bitów st\_ile[15:8]. Jeśli cel = o, wówczas ZF = 1.

cel := źródło[start + ile - 1: start]

bextr eax, zmienna, edx

bextr edx, esi, eax

bextr rcx, rdx, rxax

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

25



## Instrukcja BLSI

blsi cel, źródło

Izoluje z rejestru lub zmiennej źródła najmłodszy bit=1 i umieszcza w rejestrze celu (CF = 1). Zeroje pozostałe bity. Jeśli źródło = o, wówczas CF = o, a cel = o.

cel := (-źródło) and źródło

blsi eax, zmienna

blsi edx, esi

blsi rcx, rdx

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

26



## Instrukcja BLSR

blsr cel, źródło

Kopiuje bity z rejestru lub zmiennej źródła (32|64) i umieszcza w rejestrze celu, zeruje najmłodszy bit = 1 (CF = o). Jeśli źródło = o, wówczas CF = 1, a cel = o.

cel := (źródło - 1) and źródło

blsr eax, zmienna

blsr edx, esi

blsr rcx, rdx

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

27



## Instrukcja BLMSMK

blsmask cel, źródło

Ustawia młodsze bity rejestru celu (32|64) na 1 aż do numeru najmłodszego bitu=1 z rejestru lub zmiennej źródła włącznie (CF = o). Zeroje pozostałe bity. Jeśli źródło = o, wówczas CF = 1, a cel = not o.

cel := (źródło - 1) xor źródło

blsmask eax, zmienna

blsmask edx, esi

blsmask rcx, rdx

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

28



## Instrukcja BZHI

bzhi cel, źródło, idx

Kopiuje bity z rejestru lub zmiennej źródła (32|64) do rejestru celu (32|64) i kasuje starsze bity od numeru z rejestru idx (CF = o). Jeśli idx > 31|63, wówczas CF = 1.

cel := źródło; cel[rozmiar - 1: idx] = o

bzhi eax, zmienna, edx

bzhi edx, esi, eax

bzhi rcx, rdx, rxax

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

29



## Instrukcja SETcc

SETcc cel

Jeśli jest spełniony warunek cc, ustawia bajt na 1, w przeciwnym wypadku na 0.

```
if cc then cel := 1
else cel := 0
```

sets al

setge [esi+8]

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

30

## Instrukcje SETcc

- SETE/SETZ              Ustaw bajt jeśli equal/ zero
- SETNE/SETNZ            Ustaw bajt jeśli not equal/ not zero
- SETS                    Ustaw bajt jeśli sign (negative)
- SETNS                   Ustaw bajt jeśli not sign (non-negative)
- SETO                    Ustaw bajt jeśli overflow
- SETNO                  Ustaw bajt jeśli not overflow
- SETPE/SETPP            Ustaw bajt jeśli parity even/ parity
- SETPO/SETNP            Ustaw bajt jeśli parity odd/ not parity
- SETA/SETNBE            Ustaw bajt jeśli above/ not below or equal
- SETAE/SETNB/SETNC    Ustaw bajt jeśli above or equal/ not below/ not carry
- SETB/SETNAE/SETC    Ustaw bajt jeśli below/ not above or equal/ carry
- SETBE/SETNA            Ustaw bajt jeśli below or equal/ not above
- SETG/SETNLE            Ustaw bajt jeśli greater/ not less or equal
- SETGE/SETNL            Ustaw bajt jeśli greater or equal/ not less
- SETL/SETNGE            Ustaw bajt jeśli less/ not greater or equal
- SETLE/SETNG            Ustaw bajt jeśli less or equal/ not greater

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

31

## Przykład – int na bin(string)

```
procedure sab(var s:string;i:integer);
asm
  push ebx
  bsr edx,ecx //w edx nr najstarszej 1
  jnz @i
  mov word ptr [eax],$3001
  jmp @e
@@: inc edx
  mov [eax].dl //długość
  dec edx
  inc eax
@p: bt ecx,edx //testuj
  sete bl
  add bl,$30
  mov [eax].bl //zapisz znak
  inc eax
  dec edx
  jns @p
@e: pop ebx
end;
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

32

## Instrukcja RDPID

rdpid cel

Czyta 32-bitowy identyfikator procesora do rejestru celu.

cel=PID

rdpid eax

rdpid rdx

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

33

## Instrukcja RDTSC

rdtsc

Read Time Stamp Counter. Czyta 64-bitowy licznik do rejestrów EDX: EAX.

EDX: EAX := licznik

rdtsc

## Przykład – funkcja pomiaru czasu

```
function Pomiar(a:integer):integer;
var Cykle_H,Cykle_L:integer;
asm
  rdtsc
  mov Cykle_H,edx
  mov Cykle_L,eax
  ...
  ...
  rdtsc
  sub eax,Cykle_L
  sbb edx,CykleH
  sub EAX,9 ; odliczenie 9?
end;
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

35

## Przykład – funkcja random

```
function MyRandom(a:integer):integer;overload;
asm
  push ebx
  xor ebx,ebx
  imul edx,[ebx+MySeed].so8o88405
  inc edx
  mov [ebx+MySeed].edx,edx
  mul edx
  mov eax,edx
  pop ebx
end;

function MyRandom2(a:integer):integer;overload;
asm
  push eax
  rdtsc
  rot a,3
  imul edx,MySeed,so8o88405
  rot ah,1
  inc edx
  rot ax,a
  mov MySeed,edx
  bswap eax
  xor eax,edx
  pop edx
  mul edx
  mov eax,edx
end;
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

36



## Instrukcja RDRAND

rdrand cel

Czyta 16|32|64-bitową liczbę losową (deterministic random bit generator) do rejestru celu wg normy NIST SP 800-90A. Jeśli CF = 1 wartość jest prawidłowa.

rdrand rax



## Instrukcja RDSEED

rdseed cel

Czyta 32|64-bitową liczbę losową (non-deterministic random bit generator) do rejestru celu wg norm NIST SP 800-90B i NIST SP800-90C. Jeśli CF = 1 wartość jest prawidłowa.

rdseed rax



## Przykład

Oblicz wartość Losową (64)

```
;rcx - wskaźnik do liczby int64
random proc;
    xor    rax, rax ;nieudane
    mov    rdx, 10 ;liczba powtórzeń
    @p:   rdseed r8      ;czytaj liczbę
          jc     @ok
          dec   rdx      ;licznik--
          jnz   @p
          ret
    @ok:  mov    [rcx], r8 ;zapisz wartość
          inc   rax      ;udane
          ret
random endp;
```

# Operacje na łańcuchach

## Operacje na łańcuchach

- MOVS/MOVSB/MOVSW/MOVSD/MOVSQ Prześlij łańcuch/bajtów/słów/podwójnych słów/poczwórnego słów
- CMPS/CMPSB/CMPSW/CMPSD/CMPSQ Porównaj łańcuchy/bajtów/słów/podwójnych słów/poczwórnego słów
- SCAS/SCASB/SCASW/SCASD/SCASQ Skanuj łańcuch/bajtów/słów/podwójnych słów/poczwórnego słów
- LODS/LODSB/LODSW/LODSD/LODSQ Ładuj łańcuch/bajtów/słów/podwójnych słów/poczwórnego słów
- STOS/STOSB/STOSW/STOSD/STOSQ Zapamiętaj łańcuch/bajtów/słów/podwójnych słów/poczwórnego słów
- REP Powtarzaj dopóki ECX nie jest zerem
- REPE/REPZ Powtarzaj dopóki equal/zero
- REPNE/REPNZ Powtarzaj dopóki not equal/not zero

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

2

## Instrukcja MOVS/MOVSB

movs byte ptr [(r|e)di], [(r|e)si]  
movsb

Przesyła bajt z pamięci ds:(r|e)si do pamięci es:(r|e)di.  
Rejestry (r|e)di/(r|e)si są zwiększane/zmniejszane o 1 w zależności od flagi DF (0/1).

$[(r|e)s:edi] := [ds:(r|e)si]$   
 $(r|e)di := (r|e)di \pm 1$   
 $(r|e)si := (r|e)si \pm 1$

movsb

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

3

Wpływ na flagi: -

## Instrukcja MOVS/MOVSD

movs dword ptr [(r|e)di], [(r|e)si]  
movsd

Przesyła podwójne słowo z pamięci ds:esi do pamięci es:edi. Rejestry (r|e)di/(r|e)si są zwiększane/zmniejszane o 4 w zależności od flagi DF (0/1).

$[es:(r|e)di] := [ds:(r|e)si]$   
 $(r|e)di := (r|e)di \pm 4$   
 $(r|e)si := (r|e)si \pm 4$

movsd

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

5

Wpływ na flagi: -

## Instrukcja MOVS/MOVSW

movs word ptr [(r|e)di], [(r|e)si]  
movsw

Przesyła słowo z pamięci ds:(r|e)si do pamięci es:(r|e)di. Rejestry (r|e)di/(r|e)si są zwiększane/zmniejszane o 2 w zależności od flagi DF (0/1).

$[es:(r|e)di] := [ds:(r|e)si]$   
 $(r|e)di := (r|e)di \pm 2$   
 $(r|e)si := (r|e)si \pm 2$

movsw

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

4

Wpływ na flagi: -

## Instrukcja MOVS/MOVSQ

movs qword ptr [(r|e)di], [(r|e)si]  
movsq

Przesyła poczwórne słowo z pamięci ds:(r|e)si do pamięci es:(r|e)di. Rejestry (r|e)di/(r|e)si są zwiększane/zmniejszane o 8 w zależności od flagi DF (0/1).

$[es:(r|e)di] := [ds:(r|e)si]$   
 $(r|e)di := (r|e)di \pm 8$   
 $(r|e)si := (r|e)si \pm 8$

movsq

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

6



## Instrukcja CMPS/CMPSB

**cmps** byte ptr [(r|e)si], [(r|e)di]  
**cmpsb**

Porównuje bajt z pamięci ds:(r|e)si i z pamięci es:(r|e)di.  
 Rejestry (r|e)di/(r|e)si są zwiększone/zmniejszane o 1 w  
 zależności od flagi DF (o/1).

[ds:(r|e)si] - [es:(r|e)di]  
 $(r|e)di := (r|e)di \pm 1$   
 $(r|e)si := (r|e)si \pm 1$

**cmpsb**

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

7



## Instrukcja CMPS/CMPSW

**cmps** word ptr [(r|e)si], [(r|e)di]  
**cmpsw**

Porównuje słowo z pamięci ds:(r|e)si i z pamięci es:(r|e)di.  
 Rejestry (r|e)di/(r|e)si są zwiększone/zmniejszane o 2 w  
 zależności od flagi DF (o/1).

[ds:(r|e)si] - [es:(r|e)di]  
 $(r|e)di := (r|e)di \pm 2$   
 $(r|e)si := (r|e)si \pm 2$

**cmpsw**

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

8



## Instrukcja CMPS/CMPSD

**cmps** dword ptr [(r|e)si], [(r|e)di]  
**cmpsd**

Porównuje podwójne słowo z pamięci ds:(r|e)si i z pamięci  
 es:(r|e)di. Rejestry (r|e)di/(r|e)si są zwiększone/zmniejszane o  
 4 w zależności od flagi DF (o/1).

[ds:(r|e)si] - [es:(r|e)di]  
 $(r|e)di := (r|e)di \pm 4$   
 $(r|e)si := (r|e)si \pm 4$

**cmpsd**

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

9



## Instrukcja CMPS/CMPSQ

**cmps** qword ptr [(r|e)si], [(r|e)di]  
**cmpsq**

Porównuje poczwórne słowo z pamięci ds:(r|e)si i z pamięci  
 es:(r|e)di. Rejestry (r|e)di/(r|e)si są zwiększone/zmniejszane o  
 8 w zależności od flagi DF (o/1).

[ds:(r|e)si] - [es:(r|e)di]  
 $(r|e)di := (r|e)di \pm 8$   
 $(r|e)si := (r|e)si \pm 8$

**cmpsq**

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

10



## Instrukcja SCAS/SCASB

**scas** byte ptr [(r|e)di]  
**scasb**

Porównuje bajt akumulatora AL i pamięci es:(r|e)di. Rejestr  
 (r|e)di jest zwiększany/zmniejszany o 1 w zależności od flagi  
 DF (o/1).

AL - [es:(r|e)di]  
 $(r|e)di := (r|e)di \pm 1$

**scasb**

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

11



## Instrukcja SCAS/SCASW

**scas** word ptr [(r|e)di]  
**scasw**

Porównuje słowo akumulatora AX i pamięci es:(r|e)di. Rejestr  
 (r|e)di jest zwiększany/zmniejszany o 2 w zależności od flagi  
 DF (o/1).

AX-[es:(r|e)di]  
 $(r|e)di := (r|e)di \pm 2$

**scasw**

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

12



## Instrukcja SCAS/SCASD

scas dword ptr [(r|e)di]  
scasd

Porównuje podwójne słowo akumulatora EAX i pamięci es:(r|e)di. Rejestr (r|e)di jest zwiększany/zmniejszany o 4 w zależności od flagi DF (0/1).

EAX - [es:(r|e)di]  
(r|e)di := (r|e)di ± 4

scasd

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

13



## Instrukcja SCAS/SCASQ

scas qword ptr [(r|e)di]  
scasq

Porównuje poczwórne słowo akumulatora RAX i pamięci es:(r|e)di. Rejestr (r|e)di jest zwiększany/zmniejszany o 8 w zależności od flagi DF (0/1).

RAX-[es:(r|e)di]  
(r|e)di := (r|e)di ± 8

scasq

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

14



## Instrukcja LODS/LODSB

lodsb byte ptr [(r|e)si]  
lodsb

Czyta bajt do akumulatora AL z pamięci ds:(r|e)si. Rejestr (r|e)si jest zwiększany/zmniejszany o 1 w zależności od flagi DF (0/1).

AL := [ds:(r|e)si]  
(r|e)si := (r|e)si ± 1

lodsb

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

15



## Instrukcja LODS/LODSW

lodsw word ptr [(r|e)si]  
lodsw

Czyta słowo do akumulatora AX z pamięci ds:(r|e)si. Rejestr (r|e)si jest zwiększany/zmniejszany o 2 w zależności od flagi DF (0/1).

AX = [ds:(r|e)si]  
(r|e)si := (r|e)si ± 2

lodsw

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

16



## Instrukcja LODS/LOSDS

lodsd dword ptr [(r|e)si]  
lodsd

Czyta podwójne słowo do akumulatora EAX z pamięci ds:(r|e)si. Rejestr (r|e)si jest zwiększany/zmniejszany o 4 w zależności od flagi DF (0/1).

EAX = [ds:(r|e)si]  
(r|e)si := (r|e)si ± 4

lodsd

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

17



## Instrukcja LODS/LODSQ

lodsq qword ptr [(r|e)si]  
lodsq

Czyta poczwórne słowo do akumulatora RAX z pamięci ds:(r|e)si. Rejestr (r|e)si jest zwiększany/zmniejszany o 8 w zależności od flagi DF (0/1).

RAX = [ds:(r|e)si]  
(r|e)si := (r|e)si ± 8

lodsq

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

18



## Instrukcja STOS/STOSB

stos byte ptr [(r|e)di]

stosb

Zapisuje bajt z akumulatora AL do pamięci es:(r|e)di. Rejestr (r|e)di jest zwiększany/zmniejszany o 1 w zależności od flagi DF (0/1).

[es:(r|e)di] := AL

(r|e)di:=(r|e)di ± 1

stosb

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

19



## Instrukcja STOS/STOSW

stos word ptr [(r|e)di]

stosw

Zapisuje słowo z akumulatora AX do pamięci es:(r|e)di. Rejestr (r|e)di jest zwiększany/zmniejszany o 2 w zależności od flagi DF (0/1).

[es:(r|e)di] := AX

(r|e)di:=(r|e)di ± 2

stosw

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

20



## Instrukcja STOS/STOSD

stos dword ptr [(r|e)di]

stosd

Zapisuje podwójne słowo z akumulatora EAX do pamięci es:(r|e)di. Rejestr (r|e)di jest zwiększany/zmniejszany o 4 w zależności od flagi DF (0/1).

[es:(r|e)di] := EAX

(r|e)di := (r|e)di ± 4

stosd

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

21



## Instrukcja STOS/STOSQ

stos qword ptr [(r|e)di]

stosq

Zapisuje poczwórne słowo z akumulatora RAX do pamięci es:(r|e)di. Rejestr (r|e)di jest zwiększany/zmniejszany o 8 w zależności od flagi DF (0/1).

[es:(r|e)di]=RAX

(r|e)di:=(r|e)di ±8

stosq

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

22



## Prefiks REP

**REPNZ/REPNE**

**REPZ/REPE**

Powoduje powtórzenie (R|E)CX razy następującej po nim instrukcji łańcuchowej, jeśli spełniony jest warunek (repnz powtarza dopóty ZF = 0, jeśli ZF = 1 powtarzanie jest przerywane itd.). Jeżeli (R|E)CX = 0, to instrukcja nie zostanie wykonana.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

23



## Prefiks REP

**REPNZ/REPNE**

**REPZ/REPE**

rep movsb

rep lodsd

rep stosq

repww cmpsw

repww scasb

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

24

## Przykład

```
mov ecx, 100
mov esi, bufor1
mov edi, bufor2
rep movsb
```

Kopiuje zawartość bufor1 do bufor2.

```
mov rax, 0
mov rcx, 100
mov rdi, bufor
rep ds:stosq
```

Zeruje zawartość bufora (800B).

```
mov al, 77
mov ecx, 100
mov edi, bufor
repnz ds:scasb
```

Szuka wartości 77 w buforze. ZF = 1  
oznacza znalezienie żądanej wartości.

```
mov al, 0
mov rcx, 100
mov rdi, bufor
repz ds:scasb
```

Szuka wartości <>0 w buforze. ZF = 0  
oznacza znalezienie żądanej wartości.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

25

## Operacje na rejestrach segmentowych

- LDS Załadowanie pełnego wskaźnika z użyciem DS
- LES Załadowanie pełnego wskaźnika z użyciem ES
- LFS Załadowanie pełnego wskaźnika z użyciem FS
- LGS Załadowanie pełnego wskaźnika z użyciem GS
- LSS Załadowanie pełnego wskaźnika z użyciem SS

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

26

## Instrukcja LDS

lds cel, źródło

Wczytanie pełnego adresu źródła do pary rejestrów  
ds:cel(32).

ds:cel := wskaźnik do źródła

lds esi, tablica

Wpływ na flagi: -

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

27

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

28

## Instrukcja LES

les cel, źródło

Wczytanie pełnego adresu źródła do pary rejestrów  
es:cel(32).

es:cel := wskaźnik do źródła

les edi, tablica2

Wpływ na flagi: -

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

27

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

28

## Instrukcja LFS

lfs cel, źródło

Wczytanie pełnego adresu źródła do pary rejestrów  
fs:cel.

fs:cel := wskaźnik do źródła

lfs eax, tablica

Wpływ na flagi: -

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

29

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

30

## Instrukcja LGS

lgs cel, źródło

Wczytanie pełnego adresu źródła do pary rejestrów  
gs:cel.

gs:cel := wskaźnik do źródła

lgs eax, tablica

Wpływ na flagi: -

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

29

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

30

Wpływ na flagi: -

## Instrukcja LSS

lss cel, źródło

Wczytanie pełnego adresu źródła do pary rejestrów ss:cel.

ss:cel := wskaźnik do źródła

lss esp, nowy\_stos

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

31

## Inne operacje

- LOCK Powoduje niepodzielne wykonanie następnej instrukcji
- LEA Ładowanie adresu efektywnego
- NOP Nie wykonuje żadnego działania
- UD<sub>2</sub> Instrukcja niezdefiniowana
- XLAT/XLATB Tłumaczenie w oparciu o tablicę translacji
- MOVBE Przesłanie po zamianie kolejności bajtów
- CPUID Identyfikacja procesora

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

32

Wpływ na flagi: -

## Prefiks LOCK

lock

Powoduje wystawienie sygnału LOCK procesora i wykonanie w sposób niepodzielny instrukcji:

add, adc, and, brc, btr, bts, cmpxchg, cmpxch8b, cmpxch16b, dec, inc, neg, not, or, sbb, sub, xor, xadd i xchg,

**jeśli argument celu jest w pamięci.**

lock btr

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

33

## Przykład

mov	zu, i	;inicjalizacja	mov	zu, i
...			...	
xor	ax, ax	;ax=o	...	
@p: lock	xchg	zu, ax ;al<->zu	@p: lock	btr
	test	ax, ax ;czy o	jnc	zu, o ;zajmij
	jz	@p ;akt. czekanie		@p
	...		...	
	...		...	
	...		mov	zu, i ;uwolnij
	mov	zu, i ;oddaj i do zu	...	
	...			

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

34

Wpływ na flagi: -

## Instrukcja LEA

lea cel, źródło

Wczytanie wyznaczonego adresu źródła do rejestru celu.

cel := adres źródła

```
lea eax, [edx+esi*4+12] ; eax = edx + esi * 4 + 12
lea rax, [rdx+rsi*4+12] ; rax = rdx + rsi * 4 + 12
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

35

## Przykład

lea	rax, [rdx]	;kopiuje rejestr
lea	rbx, [rcx + rdx]	;suma rejestrów
lea	rdx, [rax + 10]	;suma rejestrów i stałej
lea	rax, [rax + 1]	;inkrementacja
lea	rax, [rbx+8*rsi + 3]	;suma trzech wartości
lea	rax, [rax + 4*rax]	;mnożenie przez 5

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

36

Wpływa na flagi: -

## Instrukcja NOP

nop

Nic nie robi.

nop

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

37

Wpływa na flagi: -

## Instrukcja UD2

ud2

Generuje wyjątek *instrukcja niezdefiniowana*, nic nie robi, wprowadzona do testów.

ud2

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

38

Wpływa na flagi: -

## Instrukcja XLAT/XLATB

xlat arg

xlatb

Tłumaczenie w oparciu o tablicę translacji.

AL := DS:[(R|E)BX+AL]

xlatb

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

39

Wpływa na flagi: -

## Instrukcja MOVBE

movbe cel, źródło

Przesłanie po zamianie kolejności bajtów. Jeden z argumentów musi być rejestrem (16, 32, 64).

cel := zamień(źródło)

movbe eax, zmienna

przed

12	c4	7f	de
----	----	----	----

po

de	7f	c4	12
----	----	----	----

(C) KISI d.KIK PCz 2022

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

40

## Rejestr flag

Rejestr flag w architekturze Intel x86	bit	Skróć/wartość	opis	typ
o	CF		flaga przeniesienia (carry)	S
i		i	zarezerwowany	
2	PF		flaga parzystości (parity)	S
4	AF		flaga wyrównania (adjust)	S
6	ZF		flaga zera (zero)	S
7	SF		flaga znaku (sign)	S
8	TP		flaga umożliwiająca krokowe wykonanie (trap)	X
9	IF		flaga zezwolenia na przerwania (interrupt enable)	X
10	DF		flaga kierunku (direction)	C
11	OF		flaga przepiętania (overflow)	S
12, 13	IOPL		poziom uprawnień we/wy I/O privilege level, od z86)	X
14	NT		nested task flag (od z86)	X
16	RF		flaga wznowienia (resume, od z86)	X
17	VM		flaga trybu Virtual 8086 (od z86)	X
18	AC		alignment check (od 486SX)	X
19	VIF		Virtual interrupt flag (od Pentium)	X
20	VIP		Virtual interrupt pending (od Pentium)	X
21	ID		identification (od Pentium)	X
3, 5, 15,	o		zarezerwowany	
22-21				

S: Znacznik stanu  
C: Znacznik kontrolny  
X: Znacznik systemowy

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

41

Wpływa na flagi: -

## Instrukcja CPUID

cpuid

Identyfikacja procesora jest możliwa, jeśli bit 21 flaga ID w rejestrze flag może być zmieniana. Na podstawie EAX (czasem też ECX) podaje w EAX,EBX,ECX i EDX różne informacje o procesorze.

cpuid

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

42

## Test ID

```

xor    eax, eax ;o      bts    edx, 21 ;ustaw
pushfd
pop    edx
bts    edx, 21 ;ustaw
push   edx
popfd
pushfd
pop    edx
btr    edx, 21 ;skasuj
rcl    eax, 1 ;oib
rcl    eax, 1 ;ib
je     cpuidOK
push   edx
popfd
pushfd
pop    edx

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

43

## Przykład

```

mov eax, 0
cpuid

```

Zwraca wartość maksymalną dla cpuid oraz identyfikator producenta:

```

eax = max
ebx = 'Genu'
ecx = 'tel'
edx = 'nel'

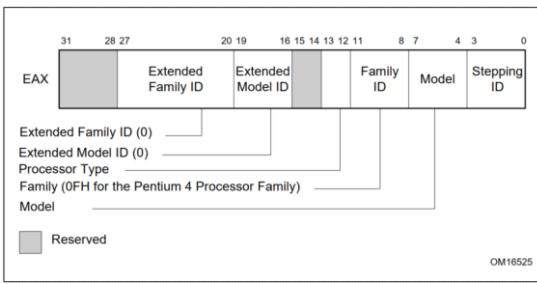
```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

44

## CPUID EAX=1

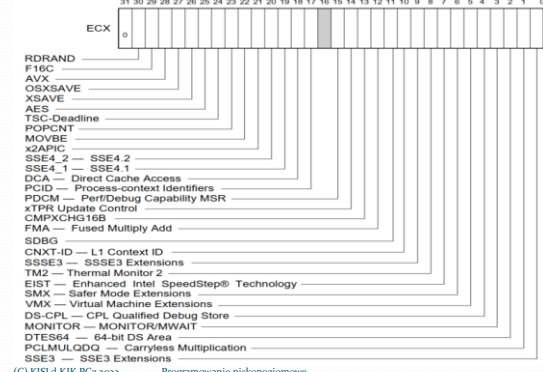


(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

45

## CPUID EAX=1

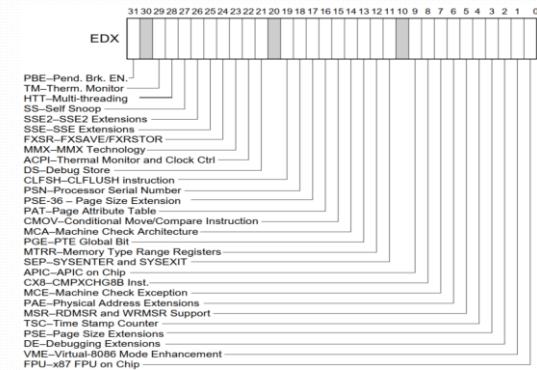


(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

46

## CPUID EAX=1



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

47

## Przykład

Sprawdzenie, czy procesor posiada technologię MMX.

```

mov EAX, 1      ; żądanie informacji o właściwościach
CPUID          ; oFH, oAH instrukcja CPUID
test EDX, 0080000H ; sprawdzenie bitu technologii MMX (bit 23 w EDX)
jnz           ; znaleziono technologię MMX

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

48



Wpływ na flagi: -

## Instrukcja XGETBV

**xgetbv**

Czyta do edx:eax zawartość rozszerzonego rejestrów kontrolnych o indeksie ecx.

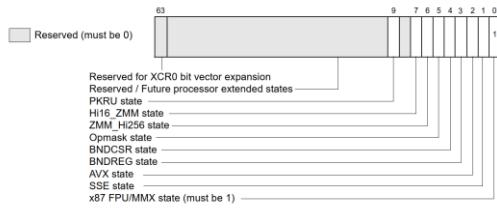
**xgetbv**

## Przykład - Sprawdzenie, czy procesor posiada technologię AVX.

```
supports_AVX proc
    mov    eax, 1
    cpuid
    and   ecx, 01800000H
    cmp   ecx, 01800000H ; sprawdź flagi OSXSAVE i AVX
    jne   not_sup
    ; procesor wspiera inst. AVX i XGETBV jest włączone przez OS
    mov   ecx, 0          ; wybierz rejestr XCR0
    XGETBV                 ; wynik w EDX:EAX
    and   eax, 06H
    cmp   eax, 06H        ; czy OS włączył obsługę XMM i YMM
    jne   not_sup
    mov   eax, 1          ; wspiera AVX
    jmp   done
not_sup: mov   eax, 0          ; nie wspiera AVX
done:   ret
endp
```



## XCR0





## Liczbowe typy danych

Liczby całkowite ze znakiem

15..14	0	słowo ze znakiem
31..30	0	podwójne słowo ze znakiem
63..62	0	poczwórne słowo ze znakiem

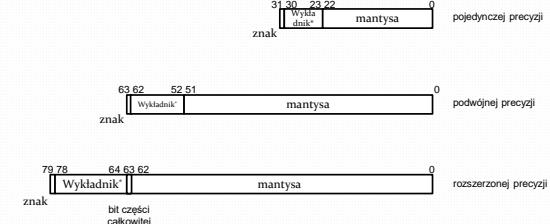
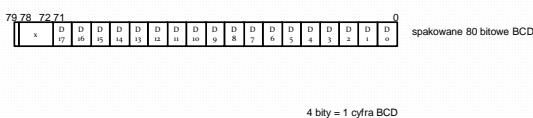
znak

(C) KISI d.KIK PCz 2022      Programowanie niskopoziomowe



## Liczbowe typy danych

Liczby zmiennoprzecinkowe



## Liczby zmiennoprzecinkowe

- liczba =  $(-1)^{\text{znak}} \cdot m \cdot 2^{\text{wykładnik}}$

- Liczba znormalizowana

$$\text{liczbaZ} = 1,01001101010 \cdot 2^{\text{wykładnik}}$$

↑  
bit części całkowitej

- Liczba nieznormalizowana

$$\text{liczbaNZ} = 0,01001101010 \cdot 2^{\text{wykładnik}}$$

↑  
bit części całkowitej

## Liczby zmiennoprzecinkowe

Implementacja:

- liczba =  $(-1)^{\text{znak}} \cdot m \cdot 2^{\text{wykładnik}}$
- wykładnik = przesunięta cecha - przesunięcie
- m = część całkowita + część ułamkowa =
  - = bit części całkowitej + mantysa
  - dla liczb pojedynczej i podwójnej precyzji część całkowita = 1 i nie jest zapamiętywana

Precyza	Przesunięcie
pojedyncza	127
podwójna	1023
rozszerzona	16383

## Wartości

poprawne i niepoprawne zawarte w rejestrach stosu koprocesora.

(C) KISI d.KIK PCz 2022

Klasa	Znak	Przesunięta cecha	Mantysa	
			Część całkowita	Część ułamkowa
Dodatnie niewłaściwe	Poprawne (cońce)	0 11..11	0	11..11
	Alejone (stosne)	0 11..11	0	01..11
Dodatnie zmiennoprzecinkowe	Nieskończoność	0 11..11	0	00..00
	Znormalizowane	0 11..10	1	11..11
Ujemne zmiennoprzecinkowe	0 00..01			00..00
	Nieznormalizowane	0 11..10	0	11..11
Pseudo-znormalizowane	0 00..00	1	11..11	00..00
	0 00..00			00..00
Zero	0 01..00	0	00..00	
	Alejone (stosne)	1 00..00	1	11..11
Ujemne niewłaściwe	Pseudo-znormalizowane	1 00..00		00..00
	Nieznormalizowane	1 11..10	0	11..01
Ujemne niewłaściwe	1 00..01			00..00
	Znormalizowane	1 11..10	1	11..01
Ujemne niewłaściwe	1 00..01			00..00
	Minus nieskończoność	1 11..11	0	00..00
Ujemne niewłaściwe	Alejone (stosne)	1 11..11	0	01..11
	Poprawne (cońce)	1 11..11	0	11..11
Programowanie niskopoziomowe		— 15 bits →		+ 63 bits →

7

## Liczby zmiennoprzecinkowe

	precyza		
	pojedyncza	podwójna	rozszerszona
cyfry znaczące	6	15	18
wartość największa	3,402823466E38	1,7976931348623158E308	1,189731495357231E4932
wartość najmniejsza	1,175494351E-38	2,2250738585072024E-308	3,3621031431120935E-4932

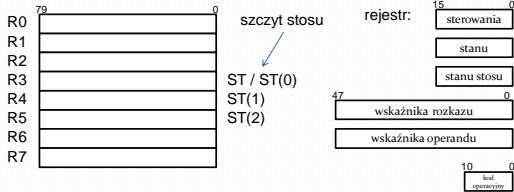
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

8

## Koprocesor - budowa

stos rejestrów



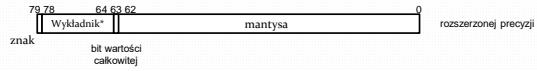
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

9

## Koprocesor

Rejestry Ro, R1, R2, R3, R4, R5, R6, R7



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

10

## Koprocesor

rejestr stanu

15	B	C3	TOP	C2	C1	Co	ES	SF	PE	UE	OE	ZE	DE	IE	0
----	---	----	-----	----	----	----	----	----	----	----	----	----	----	----	---

- B – koprocesor zajęty
- Co-C3 – bity rodzaju wyniku
- TOP – wskaźnik stosu
- ES – znacznik błędu
- SF – znacznik błędu stosu
- PE – błąd niedokładności wyniku
- UE – błąd niedomiaru
- OE – błąd nadmiaru
- ZE – błąd dzielenia przez zero
- DE – błąd zdenormalizowanego argumentu
- IE – błąd niedozwolonej operacji

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

11

## Koprocesor

rejestr sterowania

15		X	RC	PC		PM	UM	OM	ZM	DM	IM	0
----	--	---	----	----	--	----	----	----	----	----	----	---

X – interpretacja nieskończoności (tylko 287)

RC – sterowanie zaokrągleniem: do najbliższej(oo), w dół (oi), w górę (io), obcięcie (ii)

PC – sterowanie dokładnością obliczeń(23 (oo), 53 (io) i 63 (ii) bity)

PM – maskowanie błędu niedokładności wyniku

UM – maskowanie błędu niedomiaru

OM – maskowanie błędu nadmiaru

ZM – maskowanie błędu dzielenia przez zero

DM – maskowanie błędu zdenormalizowanego argumentu

IM – maskowanie błędu niedozwolonej operacji

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

12

## Koprocesor

rejestr stanu zawartości rejestrów stosu

15	TAG(7)	TAG(6)	TAG(5)	TAG(4)	TAG(3)	TAG(2)	TAG(1)	TAG(0)	0
----	--------	--------	--------	--------	--------	--------	--------	--------	---

TAG – pola określają zawartość poszczególnych rejestrów stosu:

oo – liczba prawidłowa

0i – zero

1o – wartość specjalna (nie liczba NaN, nieskończoność...) lub zdenormalizowana

11 – rejestr pusty

## Operacje przesyłania danych

- FLD załadowanie argumentu zmiennoprzecinkowego
- FST zapisanie wartości z wierzchołka stosu
- FSTP zapisanie wartości z wierzchołka stosu i usunięcie go z stosu
- FILD załadowanie liczby całkowitej
- FIST zapisanie liczby całkowitej ze zdjęciem ze stosu
- FISTP zapisanie liczby całkowitej z dodatkiem do stosu
- FBLD załadowanie liczby BCD
- FBSTP zapisanie liczby BCD i zdjęcie jej ze stosu
- FXCH zamiana zawartości rejestrów
- FCMOVE przesłanie warunkowe (jeśli równe)
- FCMOVNE przesłanie warunkowe (jeśli nie równe)
- FCMOVB przesłanie warunkowe (jeśli poniżej)
- FCMOVBE przesłanie warunkowe (jeśli poniżej lub równe)
- FCMOVNB przesłanie warunkowe (jeśli nie poniżej lub równe)
- FCMOVNBE przesłanie warunkowe (jeśli nieuporządkowane)
- FCMOVU przesłanie warunkowe (jeśli uporządkowane)

## Instrukcja FLD

fld źródło

Przesyła liczbę zmiennoprzecinkową z rejestrów st(i) lub z pamięci na wierzchołek stosu.

fpush(źródło)

fld st(3)

fld zmienna

## Instrukcja FST

fst cel

Zapisuje liczbę zmiennoprzecinkową z wierzchołka stosu do rejestrów st(i) lub pamięci.

cel := st

fst st(3)

fst zmienna

## Instrukcja FSTP

fstp cel

Zapisuje liczbę zmiennoprzecinkową z wierzchołka stosu do rejestrów st(i) lub pamięci i zdejmuję ją ze stosu.

cel := st

fpop

fstp st(3)

fstp zmienna

## Instrukcja FILD

fild źródło

Przesyła liczbę całkowitą z pamięci na wierzchołek stosu.

fpush(źródło)

fild zmienna

## Instrukcja FIST

fist cel

Zapisuje liczbę w formacie całkowitym z wierzchołka stosu do pamięci.

cel := int(st)

fist zmienna

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

19

## Instrukcja FISTP

fistp cel

Zapisuje liczbę w formacie całkowitym z wierzchołka stosu do pamięci i zdejmuje ją ze stosu.

cel := int(st)

fpop

fistp zmienna

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

20

## Instrukcja FBLD

fild źródło

Przesyła liczbę całkowitą BCD z pamięci na wierzchołek stosu.

fpush(źródło)

fild zmienna

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

21

## Instrukcja FBSTP

fbstp cel

Zapisuje liczbę w formacie całkowitym BCD z wierzchołka stosu do pamięci i zdejmuje ją ze stosu.

cel := int(st)

fpop

fbstp zmienna

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

22

## Instrukcja FXCH

fxch st(i)

fxch

Zamienia liczbę z wierzchołka stosu z wartością w rejestrze celu. Bez parametru celem jest st(i).

st(i) ⇔ st

fxch st(5)

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

23

## Instrukcja FCMOVcc

FCMOVE	przesłanie warunkowe (jeśli równe, ZF=1)
FCMOVNE	przesłanie warunkowe (jeśli nie równe, ZF=0)
FCMOVB	przesłanie warunkowe (jeśli poniżej, CF=1)
FCMOVBE	przesłanie warunkowe (jeśli poniżej lub równe, CF=1 lub ZF=1)
FCMOVNB	przesłanie warunkowe (jeśli nie poniżej, CF=0)
FCMOVNBE	przesłanie warunkowe (jeśli nie poniżej lub równe, CF=0 i ZF=0)
FCMOVU	przesłanie warunkowe (jeśli nieuporządkowane, PF=1)
FCMOVNU	przesłanie warunkowe (jeśli uporządkowane, PF=0)

fcmovcc st, st(i)

Jeśli jest spełniony warunek cc zapisuje do st wartość rejestru źródła st(i).  
if cc then st := st(i)

fcmovnb st, st(5)

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

24



## Instrukcja FDIV/FDIVP/FIDIV

Dzieli rejestr celu przez wartość źródła. Dla FIDIV źródłem jest liczba całkowita. FDIVP zdejmuje wierzchołek stosu.

$st(cel) := st(cel) / st(\text{źródło})|zmienna$

fdiv st, st(4)

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

31

## Instrukcja FDIVR/FDIVRP/FIDIVR

Dzieli źródło przez rejestr celu. Wynik umieszcza w rejestrze celu. Dla FIDIVR źródłem jest liczba całkowita. FDIVRP zdejmuje wierzchołek stosu.

$st(cel) := st(\text{źródło})|zmienna/st(cel)$

fdivr st, st(3)

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

32

## Przykład

$$y = ax^3 + bx^2 + cx + d$$

```

fld    d      ;d          ;wersja zoptymalizowana
fld    x      ;x; d
fld    st     ;x; x; d
fmul  st, st(i) pxx; x; d
fld    st(i)  ;px; x; x; d
fmul  st, st(i) pxx; xx; x; d
fmul  a      ;axxx; xx; x; d
faddp st(3), st pxx; x; axxx+d
fmul  b      ;b*xx; x; axxx+d
faddp st(2), st ;x; axxx+b*x+d
fmul  c      ;x*x; axxx+b*x+d
fadd  ;axxx+b*x+cx+d
fstp  y

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

33

## Przykład

$y = \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix}$	$y = \begin{bmatrix} x_1 & \dots & x_n \end{bmatrix}$
--	---

```

mov  ecx, n
mov  esi, x
mov  edi, z
fld  [esi]  ;x
fld  [edi]  ;z; x
fmul [edi]  ;$:=x*z
dec  ecx
add  esi, 8
add  edi, 8
fld  [esi]  ;x; s
fmul [edi]  ;x*z; s
fadd [edi]  ;$:=s+x*z
dec  ecx
jnzb @i

```

$\begin{array}{ll} mov & esi, x \\ mov & edi, y \\ fld & [esi] \end{array}$	$;x$
$\begin{array}{ll} mov & ecx, n \\ mov & [edi] \\ fmul & [edi] \end{array}$	$;s:=x^*z$
$\begin{array}{ll} dec & dec \\ add & add \\ add & add \\ fld & [esi] \\ fmul & [edi] \end{array}$	$;x$
$\begin{array}{ll} dec & dec \\ fadd & fadd \\ dec & dec \\ jnz & @i \end{array}$	$;s:=x^*z$

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

34

## Przykład

$$y = \begin{bmatrix} z_1 \\ \vdots \\ z_n \end{bmatrix}$$

```

mov  ecx,n
fld  zero
mov  esi, x
mov  edi, z
@i: fld  qword ptr[esi+8*ecx-8]  ;x; s
      qword ptr[edi+8*ecx-8]  ;x*z; s
      fadd  ;$:=s+x*z
      dec   ecx
      jnz  @i

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

35

## Instrukcja FPREM/FPREM1

Oblicza resztę z dzielenia  $st/st(1)$ , wynik umieszcza w rejestrze  $st$ . Wynik jest dokładny. Jeśli  $st/st(1)$  jest zbyt duży ( $c2=1$ ) w  $st$  umieszczona zostaje częściowa reszta i trzeba powtórzyć instrukcję. Zakres reszty:

FPREM  $<-|st(1)|, |st(1)|>$

FPREM1  $<-|st(1)/2|, |st(1)/2|>$

$Q := \text{int}|\text{round}(st/st(1))|$

$st := st - Q * st(1)$

fprem

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

36

## Instrukcja FABS

Oblicza wartość bezwzględną liczby z wierzchołka stosu.

`st := |st|`

`fabs`

## Instrukcja FCHS

Zmienia znak liczby na wierzchołku stosu.

`st := -st`

`fchs`

## Instrukcja FRNDINT

Zaokrąglą liczbę na wierzchołku stosu.

`st := round(st)`

`frndint`

## Instrukcja FSCALE

Skalowanie przez potęgę 2. Do wykładnika st dodaje część całkowitą st(i).

`st := st * 2int(st(i))`

`fscale`

## Instrukcja FSQRT

Oblicza pierwiastek kwadratowy z liczby na wierzchołku stosu.

`st := sqrt(st)`

`fsqrt`

## Instrukcja FXTRACT

Oblicza wykładnik i mantysę liczby z rejestru st.

`st := wykladnik(st)`

`fpush(mantysa(st))`

`fxtract`

## Przykład

Przekątna prostokąta  $c = \sqrt{a^2 + b^2}$

```
fld a          ;a
fmul st, st   ;a*a
fld b          ;b, a*a
fmul st, st   ;b + b, a*a
fadd          ;a*a + b*b
fsqrt          ;c
fstp c
```

## Przykład

$$y = \frac{\sqrt{|a-b|}}{a+b}$$

```
fld a          ;a
fld st          ;a, a
fld b          ;b, a, a
fadd st(2), st ;b, a, a+b
fsubp st(i), st ;a-b, a+b
fabs            ;|a-b|, (a+b)
fsqrt           ;sqrt(|a-b|), (a+b)
fdivr          ;sqrt(|a-b|)/(a+b)
fst y
```

## Operacje ładowania stałych

- FLD1 zapisanie +1.0 na wierzchołku stosu
- FLDZ zapisanie +0.0 na wierzchołku stosu
- FLDPI zapisanie  $\pi$  na wierzchołku stosu
- FLDL2E zapisanie  $\log_2 e$  na wierzchołku stosu
- FLDLN2 zapisanie  $\log_2 (ln2)$  na wierzchołku stosu
- FLDL2T zapisanie  $\log_{10} 2$  na wierzchołku stosu
- FLDLG2 zapisanie  $\log_{10} 2$  na wierzchołku stosu

Instrukcje zapisują stałe na wierzchołku stosu (st).

## Przykład

$$y = 2\pi r$$

fldpi          ;pi	fldpi          ;pi
fld r          ;r, pi	fld r          ;r, pi
fmul          ;pi*r	fmul st, st   ;pi*r
fadd st, st   ;2*pi*r	fmul          ;pi*r
fstp y	fstp y

$$y = \pi r^2$$

## Operacje funkcji przestępnych

- FSIN Oblicza sinus
- FCOS Oblicza cosinus
- FSINCOS Oblicza sinus i cosinus
- FPTAN Oblicza (częściowy) tangens
- FPATAN Oblicza (częściowy) arcus tangens
- F2XM1 Oblicza  $2^x - 1$
- FYL2X Oblicza  $y * \log_2 x$
- FYL2XP1 Oblicza  $y * \log_2(x+1)$

## Instrukcja FSIN

Oblicza sinus liczby zawartej w st(0) i wynik umieszcza w st(0). Jeśli st nie zawiera się w  $<-2^{63}, 2^{63}>$ , wówczas flaga C2 jest ustawiana. Argument można zredukować instrukcją FPREM z dzielnikiem  $2\pi$ .

$st := \sin(st)$

fsin

## Instrukcja FCOS

Oblicza cosinus liczby zawartej w st(0) i wynik umieszcza w st(0). Jeśli st nie zawiera się w  $<-2^{63}, 2^{63}>$ , wówczas flaga C2 jest ustawiana. Argument można zredukować instrukcją FPREM z dzielnikiem  $2\pi$ .

`st := cos(st)`

`fcos`

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

49

## Instrukcja FSINCOS

Oblicza sinus i cosinus liczby zawartej w st(0) i wynik umieszcza w st(0) i na wierzchołku stosu. Jeśli st nie zawiera się w  $<-2^{63}, 2^{63}>$ , wówczas flaga C2 jest ustawiana. Argument można zredukować instrukcją FPREM z dzielnikiem  $2\pi$ .

`temp := cos(st)`

`st := sin(st)`

`fpush(temp)`

`fsincos`

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

50

## Instrukcja FPTAN

Oblicza tangens liczby zawartej w st(0) i wynik umieszcza w st(0) i 1.0 na wierzchołku stosu. Jeśli st nie zawiera się w  $<-2^{63}, 2^{63}>$ , wówczas flaga C2 jest ustawiana. Argument można zredukować instrukcją FPREM z dzielnikiem  $2\pi$ .

`st := tan(st)`

`fpush(1.0)`

`fptan`

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

51

## Instrukcja FPATAN

Oblicza arcus tangens (kąt) ilorazu st(1)/st i wynik umieszcza w st(1), a st zdejmuje z wierzchołka stosu.

`st(1) := arctg(st(1)/st)`

`fpop`

`fpatan`

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

52

## Instrukcja F2XM1

Oblicza  $2^{\text{st}} - 1$ . st musi być w przedziale  $<-1,1>$ .

`st:=2st - 1`

`f2xm1`

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

53

## Instrukcja FYL2X

Oblicza  $y * \log_2 x$ . st > 0

`st(1) := st(1) * log2st`

`fpop`

`fyl2x`

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

54

## Instrukcja FYL2XP1

Oblicza  $y \cdot \log_2 x$ . Liczba w rejestrze st musi spełniać

$$-(1 - \sqrt{2}/2) < st < (1 - \sqrt{2}/2)$$

```
st(1) := st(1)* log2(st+1)
fpop
```

fyl2xp1

## Przykład

```
a = x^y = 2^{y \cdot \log_2 x}
fld y          y
fld x          ;x; y
fyl2x         y * log2x
fxm1          ;2^(y * log2x) - 1    !!!!!!!
fld1          ;; z^(y * log2x) - 1
fadd          ;a
fstp a
```

## Przykład

```
a = e^x = 2^{x \cdot \log_2 e}
fld x          ;x
fldl2e        ; log2e; x
fmul          ;x * log2e
fld st          ;x * log2e; x * log2e
frndint        ;round(x * log2e); x * log2e
fsub st(i), st ;round(x * log2e); x * log2e - round(x * log2e)
fxch st(i)      ;x * log2e - round(x * log2e); round(x * log2e)
fxm1          ;2^(x * log2e - round(x * log2e)) - 1; round(x * log2e)
fld1          ;1,2^(x * log2e - round(x * log2e)) - 1; round(x * log2e)
fadd          ;2^(x * log2e - round(x * log2e)); round(x * log2e)
fscale        ;2^(x * log2e - round(x * log2e)) * 2^(round(x * log2e) - round(x * log2e))
                ;2^(x * log2e - round(x * log2e)) + round(x * log2e) = 2^(x * log2e); round(x * log2e)
fstp st(i)      ;2^(x * log2e - round(x * log2e) + round(x * log2e)) = 2^(x * log2e)
fstp a
```

## Przykład

```
a = log_b x = log2 x / log2 b
fldi          ;1
fld x          ;x; 1
fyl2x         log2x
fld1          ;; log2x
fld b          ;b; 1; log2x
fyl2x         log2b; log2x
fddiv        ;a
fst a
```

## Operacje porównania

- FCOM porównanie liczb zmiennoprzecinkowych
- FCOMP porównanie liczb zmiennoprzecinkowych i zdjęcie ze stosu
- FCOMPP porównanie liczb zmiennoprzecinkowych i podwójne zdjęcie ze stosu
- FUCOM nieporządkowane porównanie liczb zmiennoprzecinkowych
- FUCOMP nieporządkowane porównanie liczb zmiennoprzecinkowych i zdjęcie ze stosu
- FUCOMPP nieporządkowane porównanie liczb zmiennoprzecinkowych i podwójne zdjęcie ze stosu
- FICOM porównanie z liczbą całkowitą
- FICOMP porównanie z liczbą całkowitą i zdjęcie ze stosu
- FCOMI porównanie liczb zmiennoprzecinkowych i ustawienie EFLAGS
- FUCOMI nieporządkowane porównanie liczb zmiennoprzecinkowych i ustawienie EFLAGS
- FCOMIP porównanie liczb zmiennoprzecinkowych, ustawienie EFLAGS i zdjęcie ze stosu
- FUCOMIP nieporządkowane porównanie liczb zmiennoprzecinkowych, ustawienie EFLAGS i zdjęcie ze stosu
- FTST porównanie z liczbą o.o
- FXAM sprawdzenie liczby zmiennoprzecinkowej

## Instrukcja FCOM/FCOMP/FCOMPP

Wpływ na flagi: C3 C2 C0

fcom/fcomp źródło  
fcompp

Porównanie liczb zmiennoprzecinkowych st i źródła. Źródłem może być rejestr st(i) lub zmienna. Jeśli źródło nie jest podane, to jest nim st(1). Fcomp zdejmuję liczbę z wierzchołka stosu, fcompp zdejmuję dwie liczby.

st(o) <=>? źródło

fpop ;dla fcomp,fcompp  
fpop ;dla fcompp

fcom st(3)

## Stan flag po porównaniu

Flagi koprocesora C<sub>3</sub>, C<sub>2</sub>, Co odpowiadają flagom ZF, PF i CF procesora.

relacja	flagi	C <sub>3</sub>	C <sub>2</sub>	Co
	ZF	PF	CF	
st(o)>zródło	o	o	o	
st(o)<zródło	o	o	1	
st(o)=zródło	1	o	o	
nieuporządkowane*	1	1	1	

\*flagi nie są ustawiane, jeśli wystąpi niezamaskowany wyjątek #IA

## Instrukcja FUCOM/FUCOMP/FUCOMPP

fucom/fucomp st(i)  
fucompp

Porównanie liczb zmiennoprzecinkowych st i st(i). Jeśli rejestr nie jest podany, to jest nim st(i). Fucomp zdejmuje liczbę z wierzchołka stosu, fucompp zdejmuje dwie liczby. Nie zgłasza wyjątku #IA dla nieliczb pasywnych.

st(o) <=>? st(i)  
fpop ;dla fucomp,fucomp  
fpop ;dla fucompp

fucomp st(7)

## Instrukcja FICOM/FICOMP

ficom/ficomp zmienna

Porównanie st z liczbą całkowitą w pamięci (16/32). Ficomp zdejmuje liczbę z wierzchołka stosu.

st(o) <=>? zmienna  
fpop ;dla ficomp

ficom liczba\_c

## Instrukcja FCOMI/FCOMIP/FUCOMI/FUCOMIP

fcomi/fcomip st(i)  
fucomi/fucomip st(i)

Porównanie st z liczbą w st(i). Fcompi/fucomip zdejmuje liczbę z wierzchołka stosu. Ustawia flagi: ZF, PF i CF. Fucomi i fucomip nie zgłasza wyjątku #IA dla nieliczb pasywnych

st(o) <=>? st(i)  
fpop ;dla fcomip/fucomip

fucomi st(4)

## Instrukcja FTST

ftst

Porównanie liczb zmiennoprzecinkowych st i o.o.

st(o) <=>? o.o

ftst

## Instrukcja FXAM

fxam

Sprawdza liczbę na wierzchołku stosu. C<sub>1</sub> = znak liczby.

znaczenie	flagi	C <sub>3</sub>	C <sub>2</sub>	Co
	ZF	PF	CF	
nieznormalizowana, pseudo (nie)liczba	o	o	o	
nieliczba	o	o	1	
znormalizowana	o	1	o	
nieskończoność	o	1	1	
zero	1	o	o	
rejestr pusty	1	o	1	
zdenormalizowana	1	1	o	

## Operacje sterowania koprocesorem

• FINCSTP	zwiększenie rejestru wskaźnika stosu koprocesora
• FDECSTP	zmniejszenie rejestru wskaźnika stosu koprocesora
• FFREE	zwolnienie rejestru zmiennoprzecinkowego
• FINIT	inicjalizacja koprocesora po sprawdzeniu zgłoszenia błędu numerycznego
• FNINIT	inicjalizacja koprocesora bez sprawdzenia zgłoszenia błędu numerycznego
• FCLEX	zerowanie flag błędów numerycznych po sprawdzeniu zgłoszenia błędu numerycznego
• FNCLX	zerowanie flag błędów numerycznych bez sprawdzenia zgłoszenia błędu numerycznego
• FSTCW	zapamiętanie rejestru sterowania po sprawdzeniu zgłoszenia błędu numerycznego
• FNSTCW	zapamiętanie rejestru sterowania bez sprawdzenia zgłoszenia błędu numerycznego
• FLDCW	wczytywanie flag błędów numerycznych
• FSTENV	inicjalizacja środowiska koprocesora po sprawdzeniu zgłoszenia błędu numerycznego
• FNSTENV	inicjalizacja środowiska koprocesora bez sprawdzenia zgłoszenia błędu numerycznego
• FLDENV	wczytywanie środowiska koprocesora
• FSAVE	zapamiętanie zawartości koprocesora po sprawdzeniu zgłoszenia błędu numerycznego
• FNSAVE	zapamiętanie zawartości koprocesora bez sprawdzenia zgłoszenia błędu numerycznego
• FRSTOR	wczytywanie zawartości koprocesora
• FSTSW	zapamiętanie rejestru stanu po sprawdzeniu zgłoszenia błędu numerycznego
• FNSTSW	zapamiętanie rejestru stanu bez sprawdzenia zgłoszenia błędu numerycznego
• WAIT/FWAIT	czekanie na koprocesor
• FNOP	nic robi

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

67

## Instrukcja FINCSTP

fincstp

Zwiększenie rejestru wskaźnika stosu koprocesora. Nie usuwa liczbę ze stosu.

top := (top + 1) mod 8

fincstp

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

68

## Instrukcja FDECSTP

fdecstp

Zmniejszenie rejestru wskaźnika stosu koprocesora.

top := (top - 1) mod 8

fdecstp

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

69

## Instrukcja FFREE

ffree st(i)

Zwolnienie rejestru koprocesora. Rejestr wskaźnika stosu koprocesora nie jest zmieniany.

tag(i) := 11 b

ffree st(3)

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

70

## Instrukcja FINIT/FNINIT

finit/fninit

Inicjalizacja koprocesora. Fninit inicjalizacja koprocesora bez sprawdzenia zgłoszenia błędu numerycznego.

finit

rejestr stanu = 0  
 rejestr stanu zawartości rejestrów stosu = offff h  
 rejestr sterowania = 37f h

X - interpretacja niskopoziomowej (tylko 28)  
 RC - sterowanie zaokrągleniem do najbliższej jednostki (0), w dół (0), w góry (1)  
 PC - sterowanie dokładnością obliczeń (23 (oo), 53 (oo) i 63 (u) bity)  
 PM - maskowanie błędu niedokładności wyniku  
 UM - maskowanie błędu underflow  
 OM - maskowanie błędu overflow  
 ZM - maskowanie błędu dzielenia przez zero  
 DM - maskowanie błędu ulożenia/lub uzupełniania argumentu  
 BM - maskowanie błędu modyfikacji/rozszerszenia operacji

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

71

## Instrukcja FCLEX/FNCLX

fclex/fnclx

Zerowanie flag błędów numerycznych. Fnclx - bez sprawdzenia zgłoszenia błędu numerycznego.

fnclx

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

72

## Instrukcja FSTCW/FNSTCW

`fstcw/fnstcw cel`

Zapamiętanie rejestru sterowania. Cel jest dwubajtową zmienną albo rejestrem AX. Fnstcw - bez sprawdzenia zgłoszenia błędu numerycznego.

`fnstcw`

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

73

## Instrukcja FLDCW

`fldcw źródło`

Wczytanie rejestru sterowania. Źródło jest dwubajtową zmienną.

`fldcw zmienna`

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

74

## Instrukcja FSTENV/FNSTENV

`fstenv/fnstenv cel`

Zapamiętanie środowiska koprocesora . Cel jest 14/28 bajtowym obszarem (tryb 16/32 bitowy). Fnstenv - bez sprawdzenia zgłoszenia błędu numerycznego.

`fnstenv fsrodowisko`

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

75

## Instrukcja FLDENV

`fldenv źródło`

Wczytanie środowiska koprocesora . Źródło jest 14/28 bajtowym obszarem (tryb 16/32 bitowy).

`fldenv fsrodowisko`

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

76

## Instrukcja FSAVE/FNSAVE

`fsave/fnsave cel`

Zapamiętanie zawartości koprocesora (środowisko + rejesty zmiennoprzecinkowe). Cel jest 94/108 bajtowym obszarem (tryb 16/32 bitowy). Fnsave - bez sprawdzenia zgłoszenia błędu numerycznego.

`fnsave fstan`

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

77

## Instrukcja FRSTOR

`frstor źródło`

Wczytanie zawartości koprocesora . Źródło jest 94/108 bajtowym obszarem (tryb 16/32 bitowy).

`frstor fstan`

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

78

## Instrukcja FSTSW/FNSTSW

fstsw/fnstsw cel

Zapamiętanie rejestru stanu. Cel jest dwubajtową zmienną albo rejestrem AX. Fnstsw - bez sprawdzenia zgłoszenia błędu numerycznego.

fnstsw

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

79

## Instrukcja WAIT/FWAIT

wait/fwait

Czekanie przez procesor na gotowość koprocesora (na zakończenie wykonywania instrukcji).

fwait

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

80

## Instrukcja FNOP

fnop

Nic nie robi.

fnop

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

81

## Przykład

$$\sqrt{\Delta} = \sqrt{b^2 - 4ac}$$

fld b	;b	fld b	;b
fld st	;b; b	fld st	;b; b
fmul	;bb	fmul	;bb
fld a	;a; bb	fld a	;a; bb
fmul c	;ac; bb	fmul c	;ac; bb
fadd st, st	;aa; bb	fadd st, st	;aa; bb
fadd st, st	;4ac; bb	fadd st, st	;4ac; bb
isub	;bb-4ac	isub	;bb-4ac
fist	porównanie z o,o	fildz	;o; bb-4ac
fstsw ax	;ax-stan	fcomip st, st(i)	porównanie z o,o
sahf	;stan do flag	jp blad	flagi już ustawione
jp blad	jz rowne	jp blad	jz rowne
jz rowne	je wieksze	je wieksze	je wieksze
jc mniejsze	mniejsze; ....	mniejsze; ....	wieksze; ....
wieksze; ....			

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

82

## Przykład - maksimum

32 bity

adresy 3 zmiennych na stosie

```

mov    eax,a
mov    edx,b
fld    qword ptr[eax]    ;a
fld    qword ptr[edx]    ;b,a
fcomi st(i)
fcmovb st,st(i)        ;max,a
fcmovb st,st(i)        ;max,a
fstp   st(i)            ;max
mov    eax,m
fstp   qword ptr[eax]
ret

```

z wartości na stosie

```

fld    a                ;a
fld    b                ;b,a
fcomi st(i)            ;b,a
fcmovb st,st(i)        ;max,a
fstp   st(i)            ;max
ret

```

64 bity

adresy 3 zmiennych w rejestrach

```

fld    qword ptr[rcx]    ;a
fld    qword ptr[rdx]    ;b,a
fcomi st(1)
fcmovb st,st(1)         ;max,a
fstp   st(1)             ;max
fstp   qword ptr[r8]
ret

```

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

83

# Instrukcje typu SIMD

# Instrukcje typu SIMD

Single Instruction Multiple Data - przetwarzanych jest wiele strumieni danych przez jeden wykonywany program – cecha tzw. komputerów wektorowych.

Instrukcje SIMD dzieli się na:

- MMX (MultiMedia eXtensions lub Matrix Math eXtensions) - liczby całkowite,
- SSE (Streaming SIMD Extensions) - liczby zmiennoprzecinkowe.

# Instrukcje typu MMX

# MMX

- wprowadzone w 1997 przez Intela dla procesorów Pentium MMX.
- Przykłady zastosowań:
  - wyświetlanie grafiki trójwymiarowej: przekształcenia geometryczne, cieniowanie, teksturowanie;
  - dekodowanie obrazów JPEG i PNG;
  - dekodowanie i kodowanie filmów MPEG (m.in. wyznaczanie transformat DCT i IDCT);
  - filtrowanie sygnałów: obrazów statycznych, filmów, dźwięku;
  - wyświetlanie grafiki dwuwymiarowej (blue box, maskowanie, przezroczystość);
  - wyznaczanie transformat: Haara, FFT.

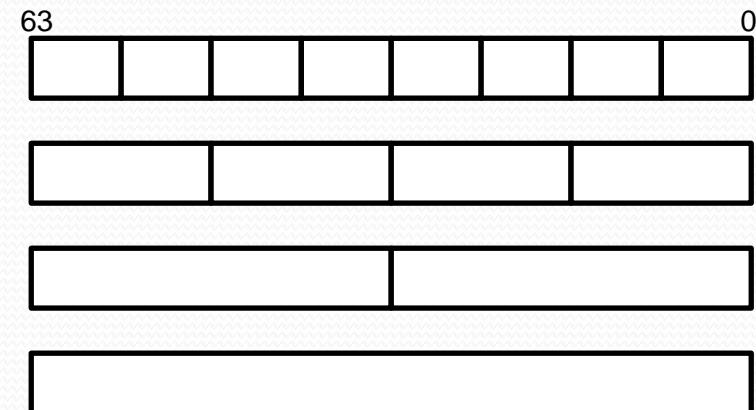
# Rejestry MMX

- 8 rejestrów 64 bitowych oznaczanych jako MM<sub>0</sub>, ..., MM<sub>7</sub>,
- wykorzystują rejesty koprocessora – młodsze 64 bity (mantysa),
- odczyt i zapis wartości, powoduje **wzięcie w użycie zawartości wszystkich rejestrów koprocessora**, nie można mieszać obliczeń MMX z obliczeniami w koprocessorze, po zakończeniu obliczeń MMX należy zwolnić rejesty.

# Typy danych

MMX wprowadził nowe wektorowe (macierzowe lub tablicowe) typy danych (ang. packed, czyli dosłownie spakowane, upakowane). „Spakowanie” polega traktowaniu danych 64-bitowych jako składających z odrębnych elementów o tej samej wielkości:

- $8 \times 8$  bitów (packed byte),
- $4 \times 16$  bitów (packed word),
- $2 \times 32$  bity (packed dword),
- $1 \times 64$  bity (quad word).



# Budowa rozkazów

- Mnemoniki prawie wszystkich rozkazów MMX rozpoczynają się od litery **p** (od słowa packed);
- 3-4 literowy skrót wykonywanego działania (np. add, sub, mul);
- litera **s** lub **u** określa, że działanie jest wykonywane na liczbach ze znakiem (signed) lub bez znaku (unsigned);
- litera **s** oznacza operację wykonywaną z nasyceniem;
- rozmiar elementu wektora: **b** - bajt (8 bitów), **w** - słowo (16 bitów), **d** - podwójne słowo (32 bity).

**paddusb**

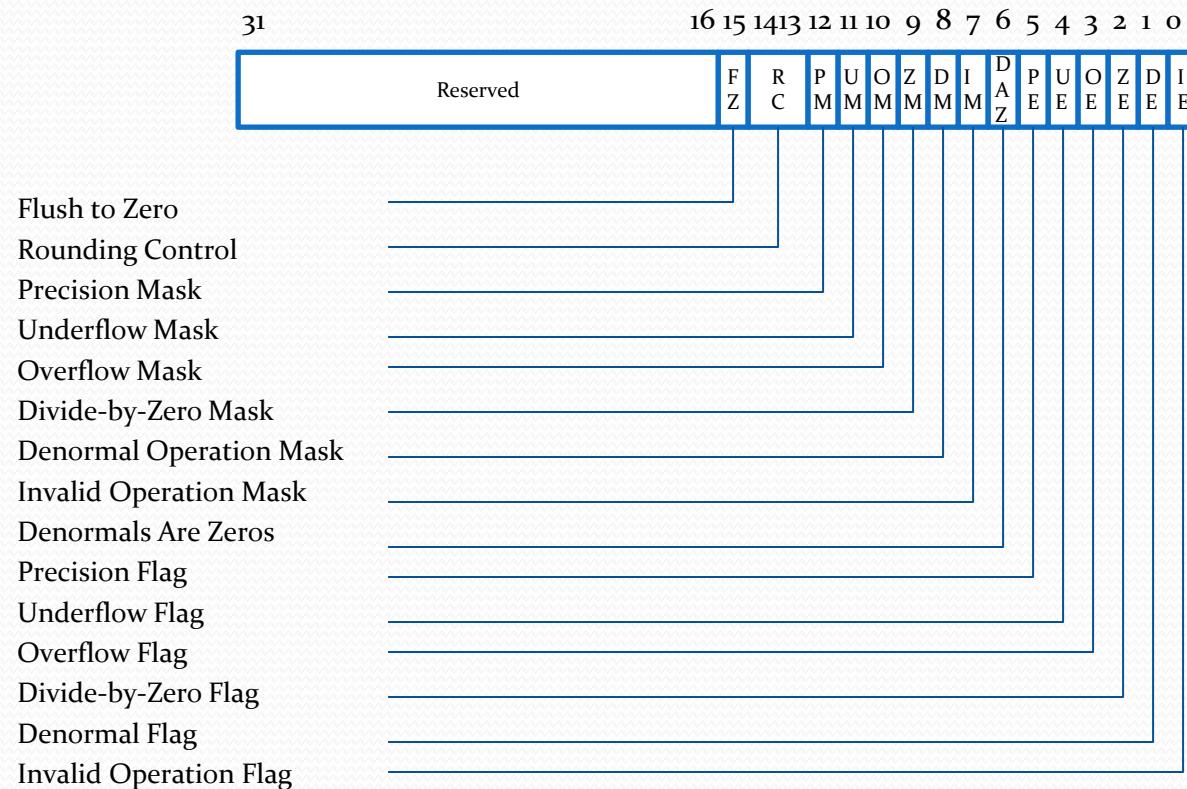
- Rozkaz **paddusb** wykonuje równoległe (p) dodawanie (add) bez znaku (u) z nasyceniem (s) liczb o rozmiarze bajtu (b)

# Nasycenie

Zakresy liczb (dla 8 bitów):

- bez znaku o ...  $2^n - 1$  (0 ... 255)
- ze znakiem –  $2^{n-1} \dots 2^{n-1} - 1$  (-128 ... 127)
- Jeśli wynik jest mniejszy od najmniejszej liczby z zakresu jest ustawiany na tę liczbę.
- Jeśli wynik jest większy od największej liczby z zakresu jest ustawiany na tę liczbę.
- Dla operacji bez nasycenia wynik jest obcinany do odpowiedniej ilości bitów.

# Rejestr MXCSR



# Operacje przesyłania

- MOVD        przesyłanie podwójnego słowa
- MOVQ        przesyłanie poczwórnego słowa

movd/movq cel, źródło

Przesyłanie podwójnego/poczwórnego słowa do/z rejestru mmx. Przy zapisie podwójnego słowa bity 32-63 rejestru mmx wypełniane są zerami.

# Operacje konwersji

- **PACKSSWB** pakowanie z nasyceniem słów ze znakiem do bajtów
- **PACKSSDW** pakowanie z nasyceniem podwójnych słów ze znakiem do słów
- **PACKUSWB** pakowanie z nasyceniem słów bez znaku do bajtów
- **PUNPCKHBW** rozpakowanie z przeplotem starszych bajtów
- **PUNPCKHWD** rozpakowanie z przeplotem starszych słów
- **PUNPCKHDQ** rozpakowanie z przeplotem starszych podwójnych słów
- **PUNPCKLBW** rozpakowanie z przeplotem młodszych bajtów
- **PUNPCKLWD** rozpakowanie z przeplotem młodszych słów
- **PUNPCKLDQ** rozpakowanie z przeplotem młodszych podwójnych słów

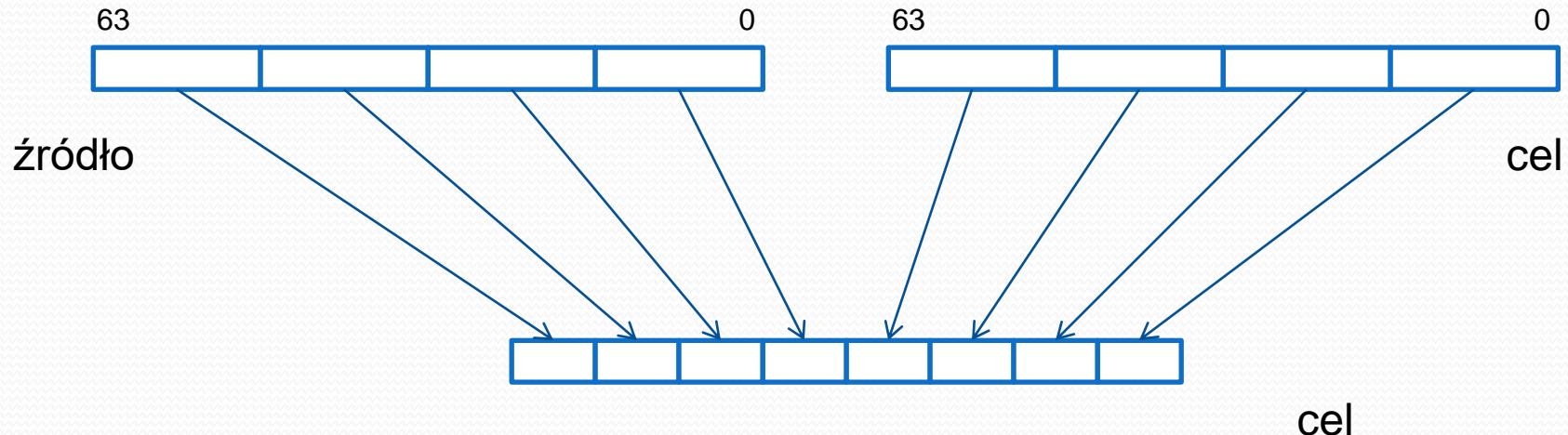
# Instrukcja

## PACKSSWB/PACKUSWB

PACKSSWB cel, źródło

PACKUSWB cel, źródło

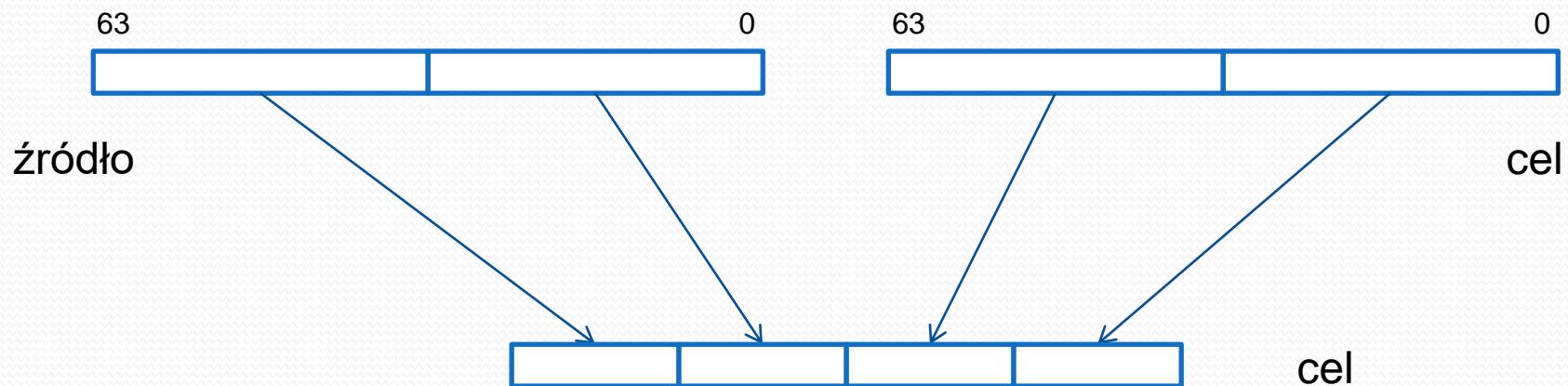
Pakowanie z nasyceniem słów ze znakiem/bez znaku do bajtów. Cel musi być rejestrem mmx.



# Instrukcja PACKSSDW

PACKSSDW cel, źródło

Pakowanie z nasyceniem podwójnych słów ze znakiem do słów. Cel musi być rejestrem mmx.



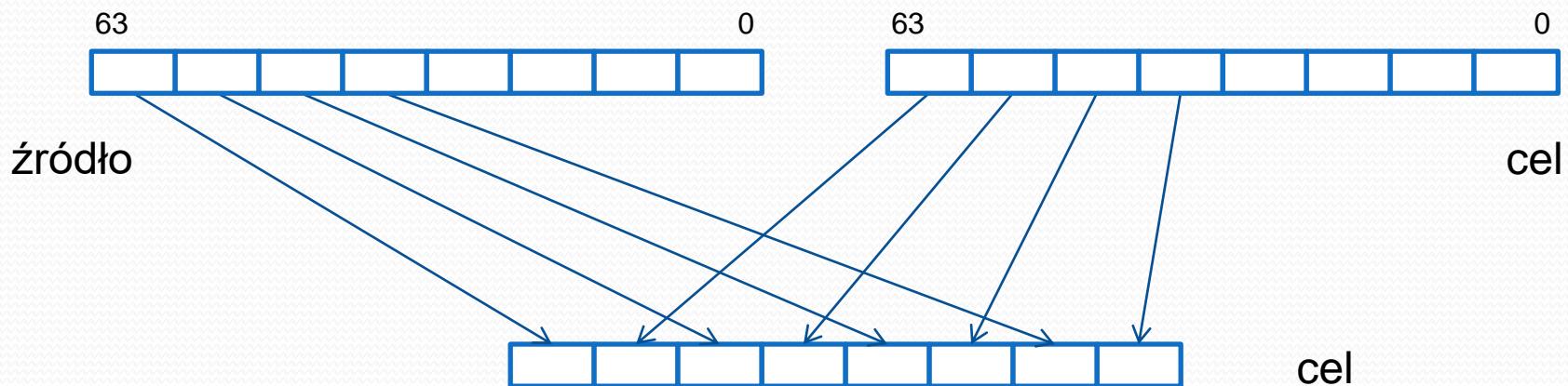
# Instrukcje PUNPCKHBW

## PUNPCKHWD PUNPCKHDQ

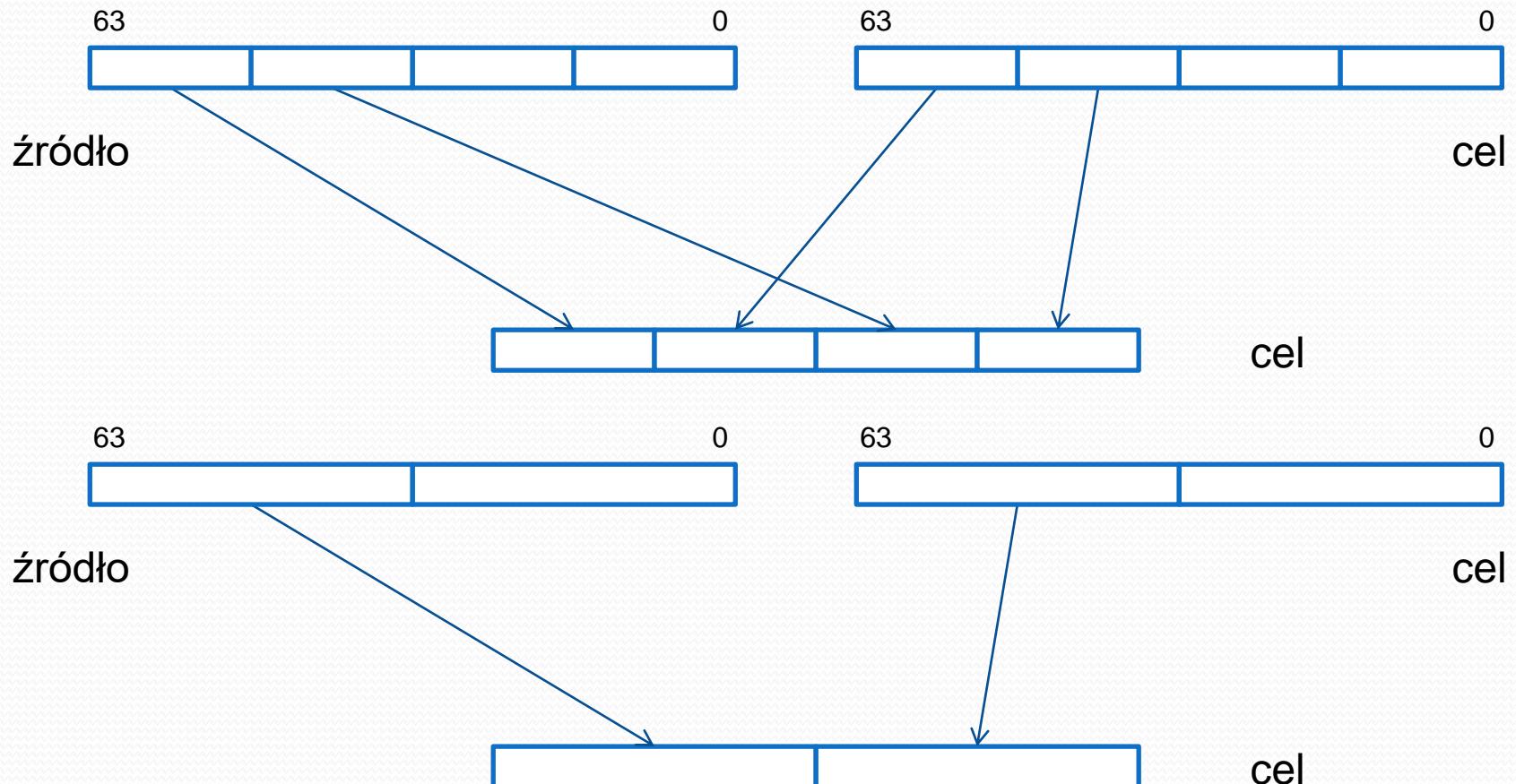
PUNPCKHBW/PUNPCKHWD/PUNPCKHDQ cel, źródło

Rozpakowanie z przeplotem starszych bajtów/słów/  
podwójnych słów bez znaku. Cel musi być rejestrem mmx.

Jeśli źródło = o to następuje konwersja do słów, podwójnych i  
poczwórnego słowa.



# Instrukcje PUNPCKHWD PUNPCKHDQ

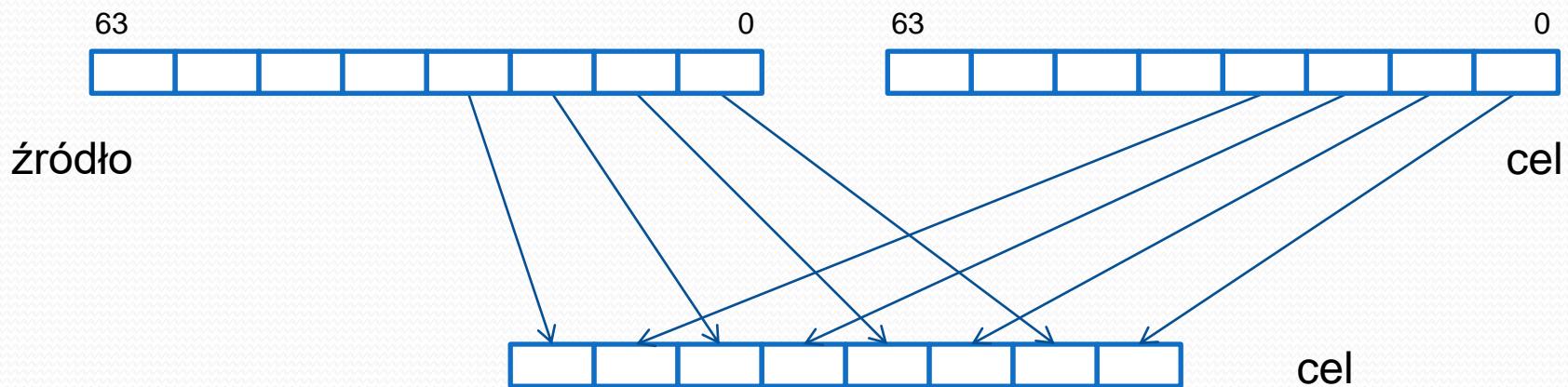


# Instrukcje PUNPCKLBW

## PUNPCKLWD PUNPCKLDQ

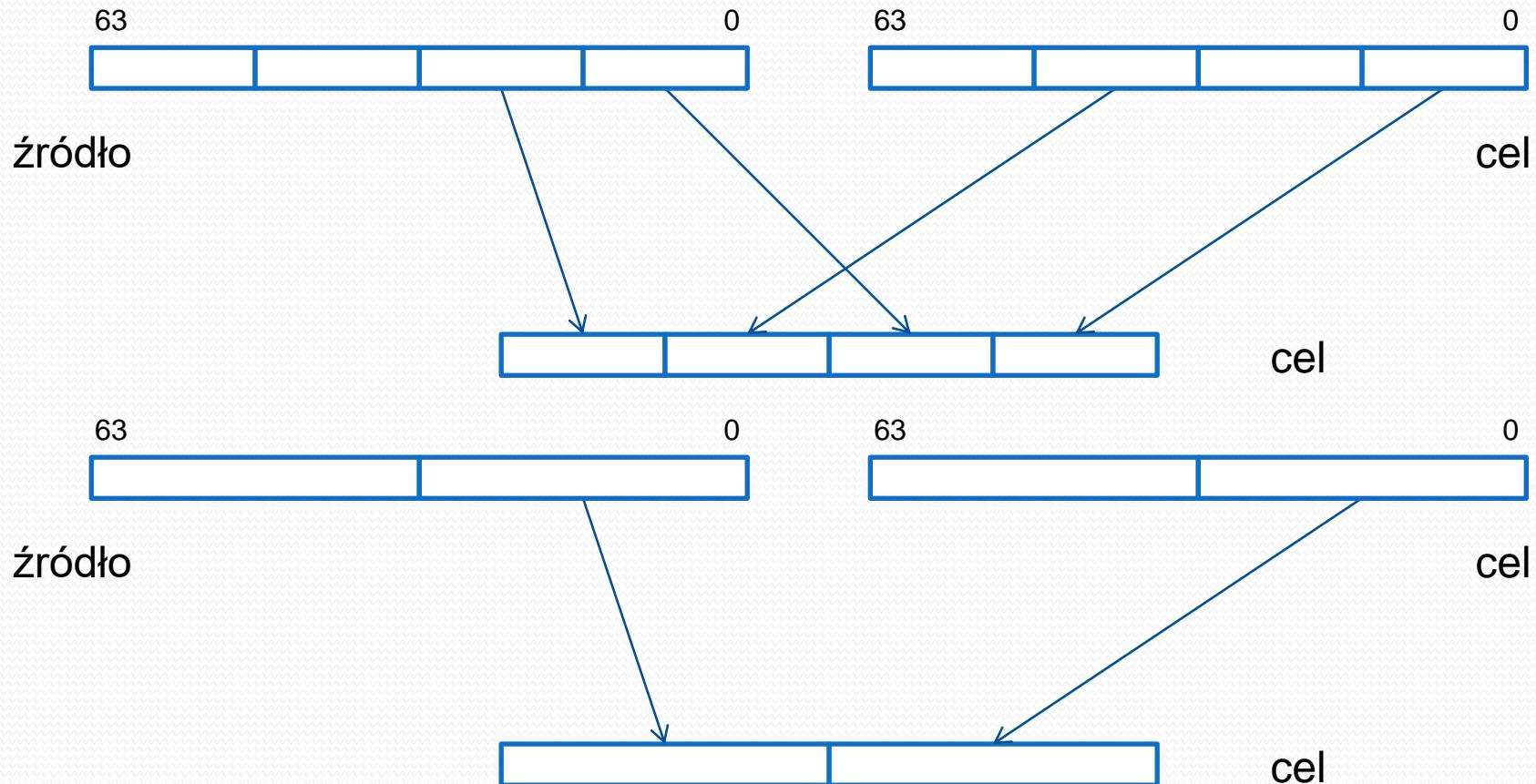
PUNPCKLBW/PUNPCKLWD/PUNPCKLDQ cel, źródło

Rozpakowanie z przeplotem młodszych bajtów/słów/  
podwójnych słów bez znaku. Cel musi być rejestrem mmx.  
Jeśli źródło = o to następuje konwersja do słów, podwójnych i  
poczwórnego słowa.



# Instrukcje

## PUNPCKLWD PUNPCKLDQ



# Operacje arytmetyczne

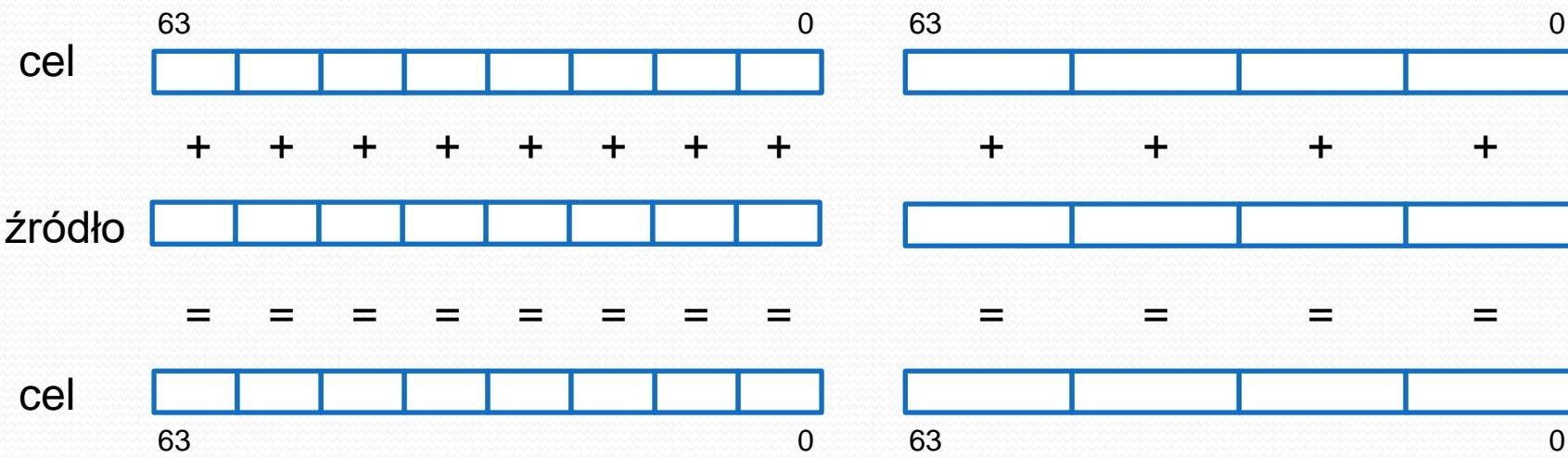
- PADDB                dodawanie wektorów bajtów
- PADDW              dodawanie wektorów słów
- PADDD              dodawanie wektorów podwójnych słów
- PADDSB             dodawanie z nasyceniem wektorów bajtów ze znakiem
- PADDSW             dodawanie z nasyceniem wektorów słów ze znakiem
- PADDUSB            dodawanie z nasyceniem wektorów bajtów bez znaku
- PADDUSW            dodawanie z nasyceniem wektorów słów bez znaku

# Instrukcje

## PADDB PADDW PADDD

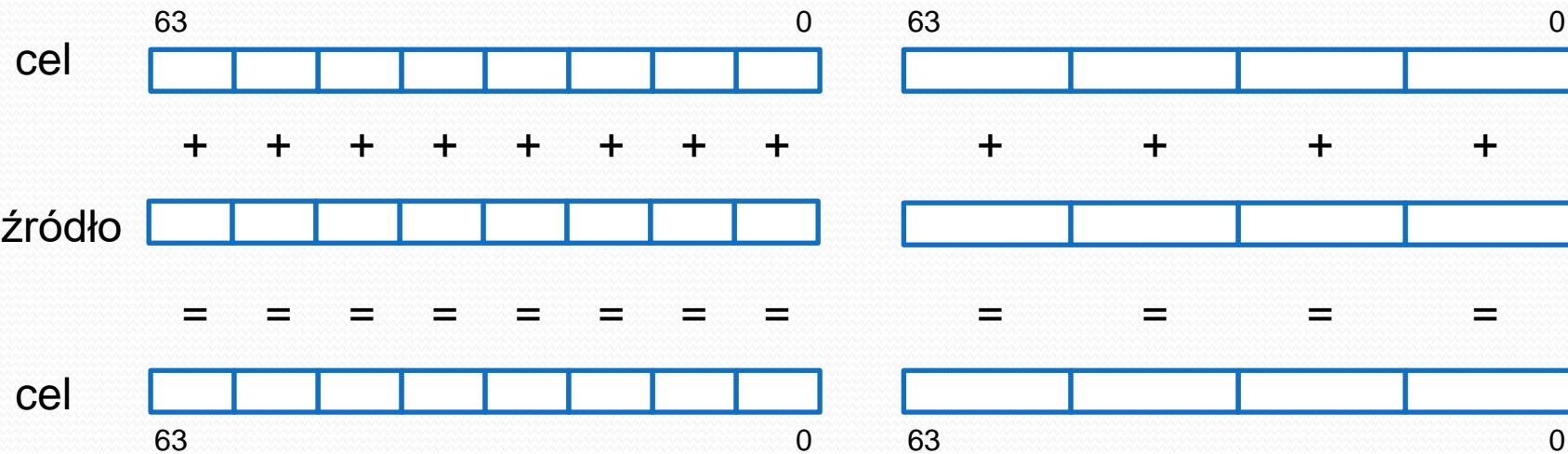
PADDB/PADDW/PADDD cel, źródło

Dodawanie wektorów bajtów/słów/podwójnych słów. Cel musi być rejestrem mmx. Wynik nie uwzględnia przeniesienia.



# Instrukcje PADDSB PADDSW PADDUSB PADDUSW

PADDSB/PADDSW/PADDUSB/PADDUSW cel, źródło  
Dodawanie z nasyceniem wektorów bajtów/słów ze  
znakiem/bez znaku. Cel musi być rejestrem mmx.



# Operacje arytmetyczne

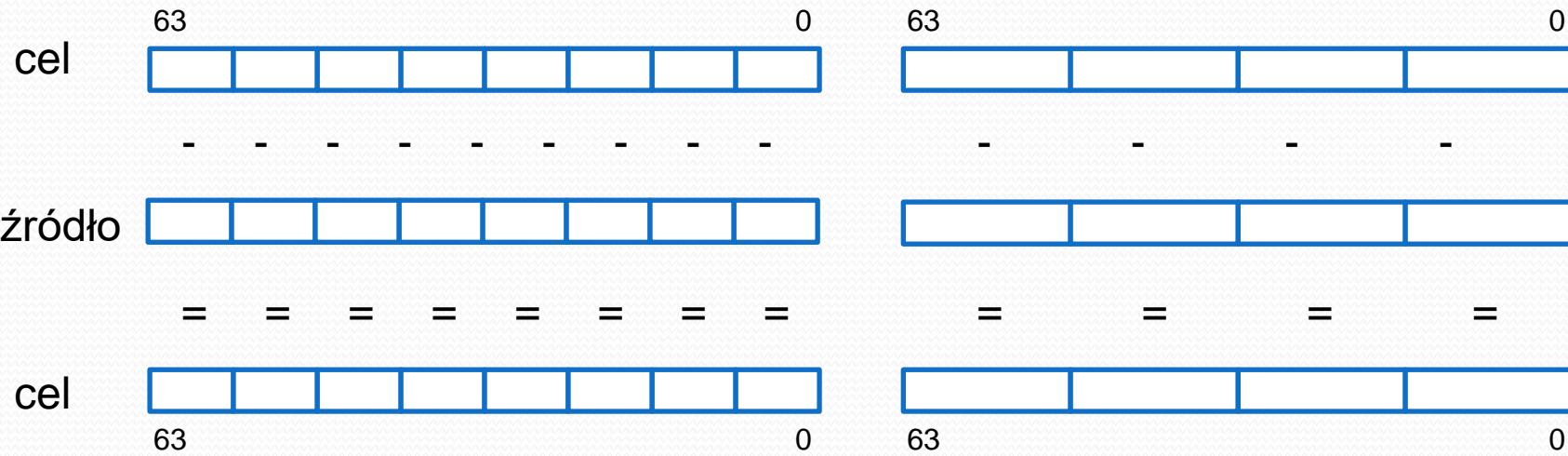
- PSUBB               odejmowanie wektorów bajtów
- PSUBW               odejmowanie wektorów słów
- PSUBD               odejmowanie wektorów podwójnych słów
- PSUBSB              odejmowanie z nasyceniem wektorów bajtów ze znakiem
- PSUBSW              odejmowanie z nasyceniem wektorów słów ze znakiem
- PSUBUSB             odejmowanie z nasyceniem wektorów bajtów bez znaku
- PSUBUSW            odejmowanie z nasyceniem wektorów słów bez znaku

# Instrukcje

## PSUBB PSUBW PSUBD

PSUBB/PSUBW/PSUBD      cel, źródło

Odejmowanie wektorów bajtów/słów/podwójnych słów. Cel musi być rejestrem mmx. Wynik nie uwzględnia przeniesienia.

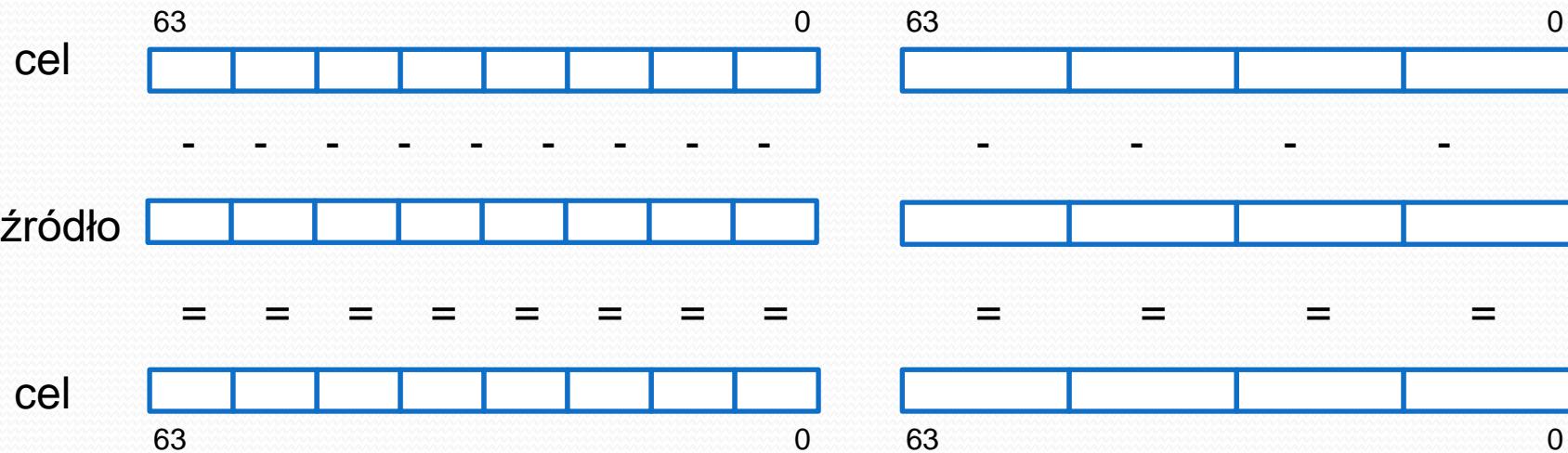


# Instrukcje PSUBSB PSUBSW

## PSUBUSB PSUBUSW

PSUBSB/PSUBSW/PSUBUSB/PSUBUSW cel, źródło

Odejmowanie z nasyceniem wektorów bajtów/słów ze znakiem/bez znaku. Cel musi być rejestrem mmx.



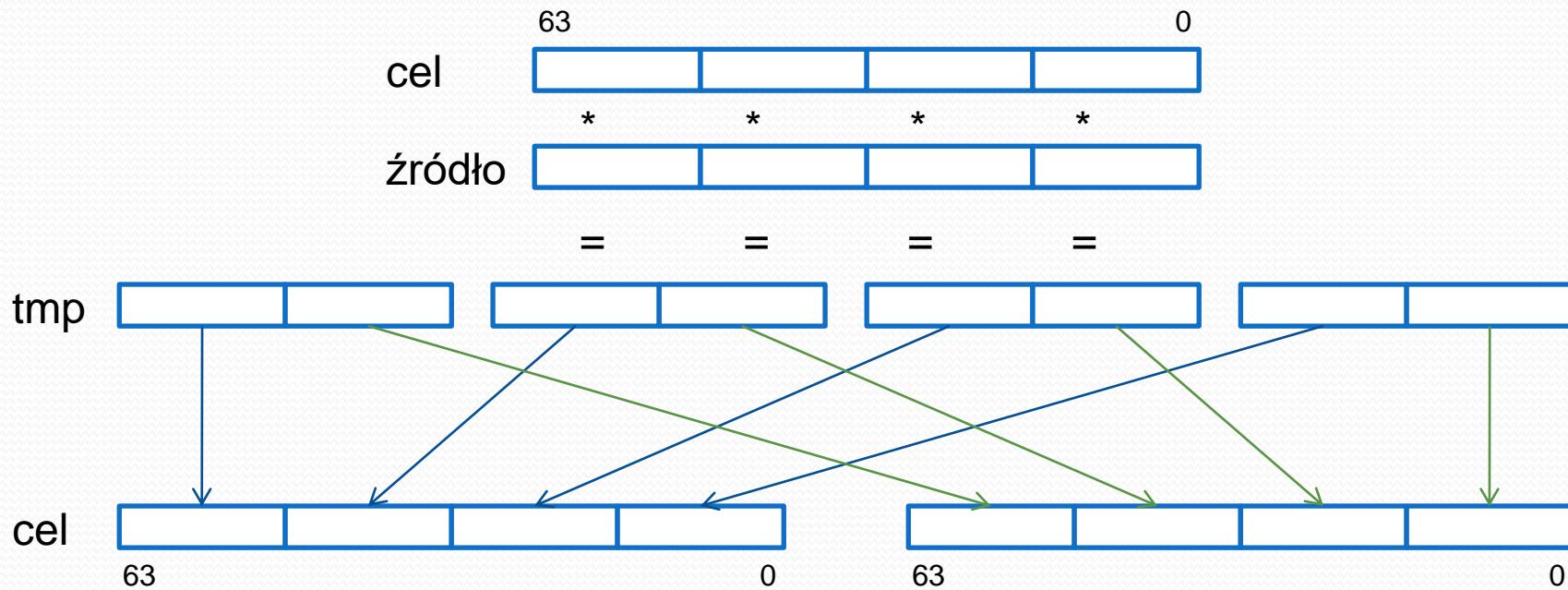
# Operacje arytmetyczne

- **PMULHW**      mnożenie wektorów słów i zapamiętanie starszych słów wyniku
- **PMULLW**      mnożenie wektorów słów i zapamiętanie młodszych słów wyniku
- **PMADDWD**      mnożenie i dodawanie wektorów słów

# Instrukcje PMULHW PMULLW

## PMULHW/PMULLW cel, źródło

Mnożenie wektorów słów i zapamiętanie starszych/młodszych słów wyniku. Cel musi być rejestrem mmx.

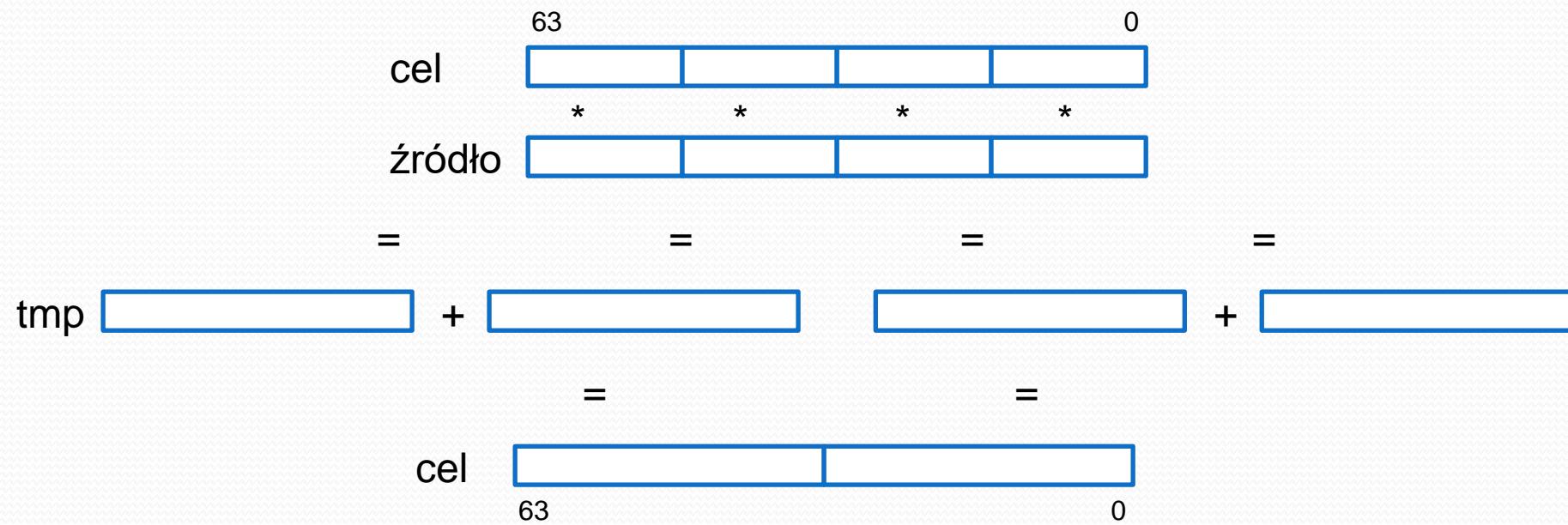


# Instrukcje PMADDWD

PMADDWD

cel, źródło

Mnożenie i dodawanie wektorów słów. Cel musi być rejestrem mmx.



# Operacje porównania

- PCMPEQB sprawdzenie równości wektorów bajtów
- PCMPEQW sprawdzenie równości wektorów słów
- PCMPEQD sprawdzenie równości wektorów podwójnych słów
- PCMPGTB sprawdzenie większości wektorów bajtów ze znakiem
- PCMPGTW sprawdzenie większości wektorów słów ze znakiem
- PCMPGTD sprawdzenie większości wektorów podwójnych słów ze znakiem

# Instrukcje PCMPEQB PCMPEQW PCMPEQD

PCMPEQB/PCMPEQW/PCMPEQD cel, źródło

Sprawdzenie równości składowych wektorów bajtów/słów/podwójnych słów. Cel musi być rejestrem mmx.

```
if cel[i] = źródło[i] then cel[i] := offh/offffh/offfffffh  
else cel[i] := 0
```

# Instrukcje PCMPGTB PCMPGTW PCMPGTD

PCMPGTB/PCMPGTW/PCMPGTD cel, źródło  
Sprawdzenie relacji większości składowych wektorów bajtów/słów/podwójnych słów ze znakiem. Cel musi być rejestrem mmx.

```
if cel[i] > źródło[i] then cel[i] := offh/offffh/offfffffh  
else cel[i] := 0
```

# Operacje logiczne

- PAND              bitowy iloczyn logiczny
- PANDN             bitowy iloczyn logiczny z negacją
- POR                bitowa suma logiczna
- PXOR              bitowa suma modulo 2

# Instrukcje

## PAND PANDN POR PXOR

PAND/PANDN/POR/PXOR                    cel, źródło

Obliczenie bitowego iloczynu logicznego/bitowego iloczynu logicznego z negacją/bitowej sumy logicznej/bitowej sumy modulo 2. Cel musi być rejestrem mmx.

cel := cel and źródło

cel := (not cel) and źródło

cel := cel or źródło

cel := cel xor źródło

# Operacje przesunięć

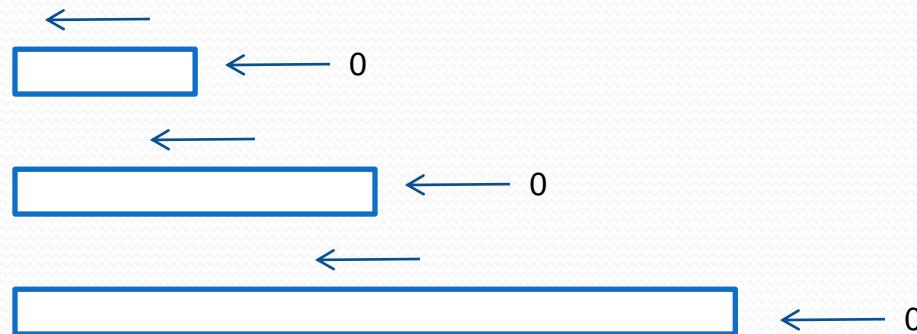
- PSLLW logiczne przesunięcie w lewo wektora słów
- PSLLD logiczne przesunięcie w lewo wektora podwójnych słów
- PSLLQ logiczne przesunięcie w lewo wektora poczwórnego słów
- PSRLW logiczne przesunięcie w prawo wektora słów
- PSRLD logiczne przesunięcie w prawo wektora podwójnych słów
- PSRLQ logiczne przesunięcie w prawo wektora poczwórnego słów
- PSRAW arytmetyczne przesunięcie w prawo wektora słów
- PSRAD arytmetyczne przesunięcie w prawo wektora podwójnych słów

# Instrukcje

## PSLLW PSLLD PSLLQ

PSLLW/PSLLD/PSLLQ      cel, ile

Logiczne przesunięcie w lewo elementów wektora  
słów/podwójnych słów/poczwórnego słów. Cel musi być  
rejestrem mmx, ile jest rejestrem mmx, zmienną lub stałą.  
Młodsze bity wypełniane są zerami.

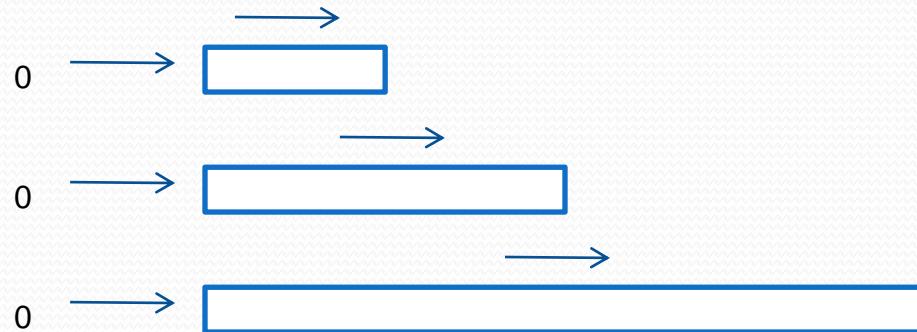


# Instrukcje

## PSRLW PSRLD PSRLQ

PSRLW/PSRLD/PSRLQ      cel, ile

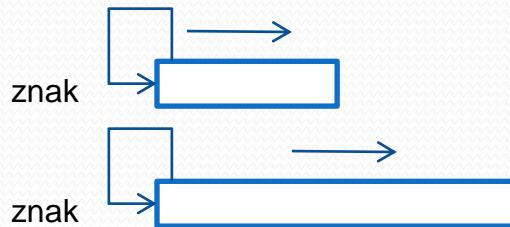
Logiczne przesunięcie w prawo elementów wektora  
słów/podwójnych słów/poczwórnego słów. Cel musi być  
rejestrem mmx, ile jest rejestrem mmx, zmienną lub stałą.  
Starsze bity wypełniane są zerami.



# Instrukcje PSRAW PSRAD

PSRAW/PSRAD      cel, ile

Arytmetyczne przesunięcie w prawo elementów wektora słów/podwójnych słów. Cel musi być rejestrem mmx, ile jest rejestrem mmx, zmienną lub stałą. Starsze bity wypełniane są znakiem.



# Operacje sterujące

- FXSAVE zapisanie stanu x87 FPU i rejestrów SIMD
- FXRSTOR wczytanie stanu x87 FPU i rejestrów SIMD
- EMMS zwalnia wszystkie rejesty koprocessora
- LDMXCSR wczytanie rejestru MXCSR
- STMXCSR zapisanie rejestru MXCSR

# Instrukcje

## FXSAVE

## FXRSTOR

FXSAVE/FXRSTOR cel512

zapisanie/wczytanie stanu

x87 FPU i rejestrów SIMD

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																			
FPU IP				FOP		FTW		FSW		FCW		0																						
MXCSR_MASK		MXCSR				FPU DP								16																				
Reserved				ST0/MM0								32																						
Reserved				ST1/MM1								48																						
Reserved				ST2/MM2								64																						
Reserved				ST3/MM3								80																						
Reserved				ST4/MM4								96																						
Reserved				ST5/MM5								112																						
Reserved				ST6/MM6								128																						
Reserved				ST7/MM7								144																						
XMM0																160																		
XMM1																176																		
XMM2																192																		
XMM3																208																		
XMM4																224																		
XMM5																240																		
XMM6																256																		
XMM7																272																		
XMM8																288																		
XMM9																304																		
XMM10																320																		
XMM11																336																		
XMM12																352																		
XMM13																368																		
XMM14																384																		
XMM15																400																		
Reserved																416																		
Reserved																432																		
Reserved																448																		
Reserved																464																		
Reserved																480																		
Reserved																496																		

# Instrukcja EMMS

EMMS

Zwalnia wszystkie rejesty koprocessora wpisując do pól TAG[i] rejestru stanu zawartości rejestrów stosu wartość 11b (rejestr pusty). Wszystkie instrukcje MMX wpisują do pól TAG[i] o, co oznacza liczbę prawidłową!

# Instrukcje LDMXCSR STMXCSR

LDMXCSR/STMXCSR zmienna

Wczytanie/zapisanie zawartości rejestru MXCSR.

# Operacje MMX wprowadzone z SSE

- PAVGB oblicza średnią z elementów wektorów bajtów bez znaku
- PAVGW oblicza średnią z elementów wektorów słów bez znaku
- PEXTRW wydobycie słowa
- PINSRW wstawienie słowa
- PMAXUB oblicza maksimum z elementów wektorów bajtów bez znaku
- PMAXSW oblicza maksimum z elementów wektorów słów ze znakiem
- PMINUB oblicza minimum z elementów wektorów bajtów bez znaku
- PMINSW oblicza minimum z elementów wektorów słów ze znakiem
- PMOVMSKB przesłanie maski bajtów
- PMULHUW mnożenie wektorów słów bez znaku i zapamiętanie starszych słów wyniku
- PSADBW oblicza sumę wartości bezwzględnych różnic
- PSHUFW tasuje słowa w rejestrze MMX

# Instrukcje PAVGB PAVGW

PAVGB/PAVGW cel, źródło

Oblicza średnią z elementów wektorów bajtów/słów bez znaku. Cel jest rejestrem MMX.

e\_cel := (e\_cel + e\_źródło + 1) >> 1

# Instrukcje PEXTRW PINSRW

PEXTRW cel, źródło, numer

PINSRW cel, źródło, numer

Wydobycie/wstawienie słowa o podanym numerze z/do rejestru MMX do/z rejestru ogólnego przeznaczenia lub pamięci.

# Instrukcje PMAXUB PMAXSW PMINUB PMINSW

PMAXUB/PMAXSW/PMINUB/PMINSW cel, źródło

Oblicza maksimum/minimum z elementów wektorów bajtów  
bez znaku/słów ze znakiem.

e\_cel := e\_cel max/min e\_źródło

# Instrukcja PMOVMSKB

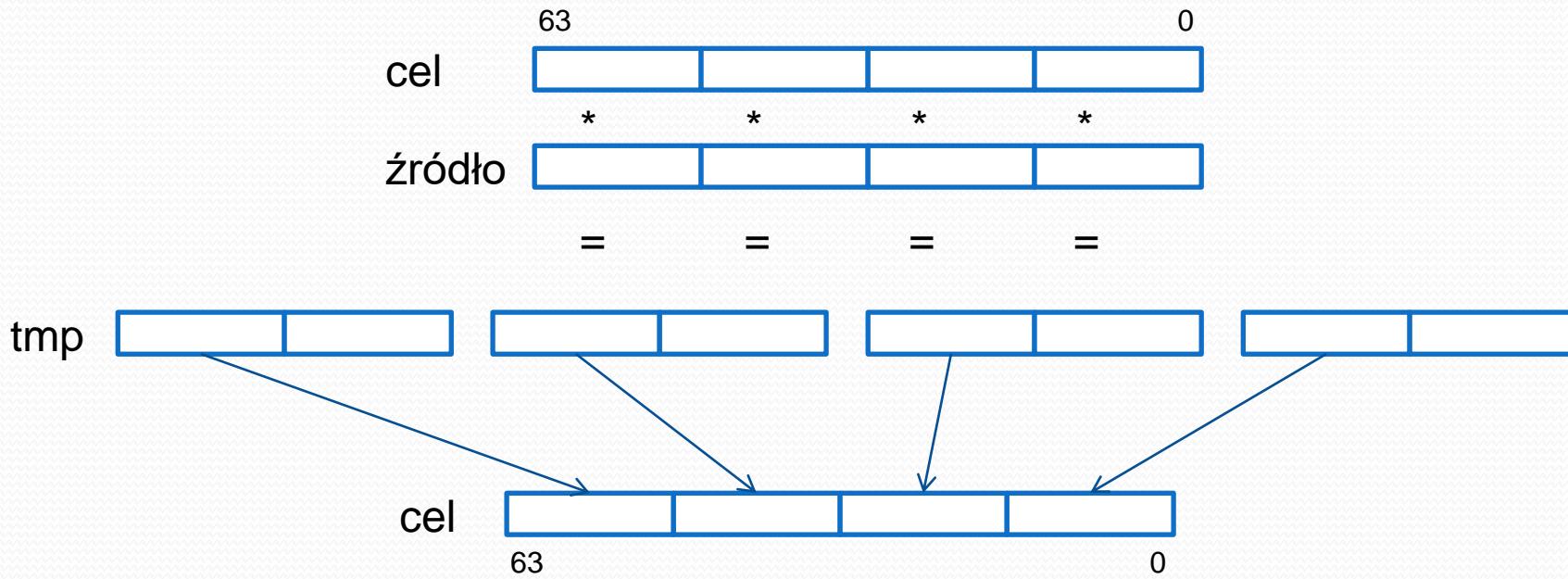
PMOVMSKB cel, źródło

Przesłanie maski bajtów. Celem jest rejestr ogólnego przeznaczenia. Najstarsze bity elementów wektora z rejestru MMX wpisywane są na bity o ..7 celu. Stosowane w celu określenia znaku lub sprawdzenia wyniku porównania.

# Instrukcje PMULHUW

PMULHUW cel, źródło

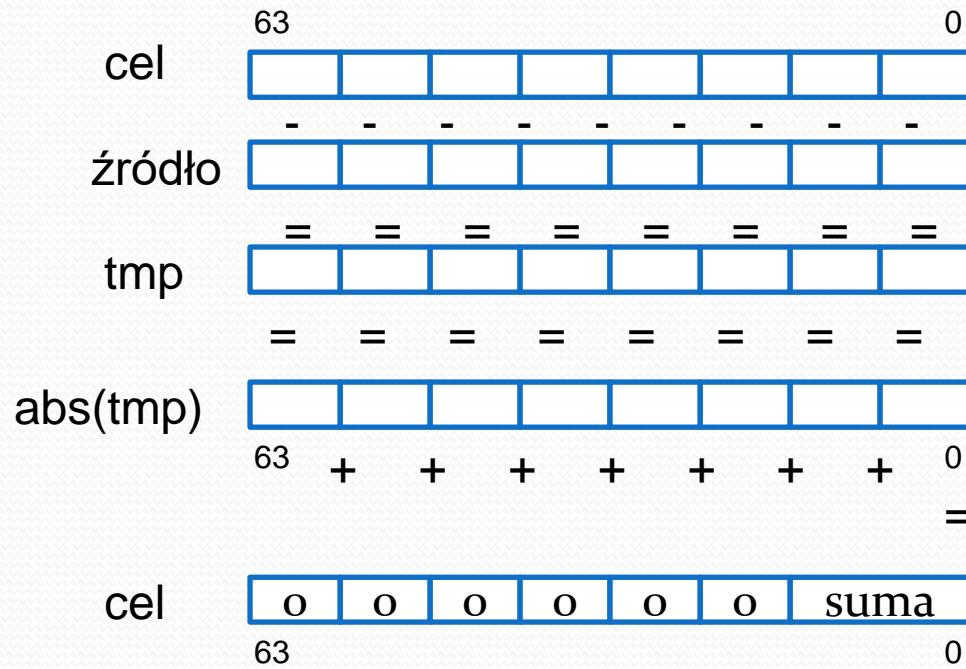
Mnożenie wektorów słów bez znaku i zapamiętanie starszych słów wyniku. Cel musi być rejestrem mmx.



# Instrukcja PSADBW

PSADBW      cel, źródło

Oblicza sumę wartości bezwzględnych różnic.

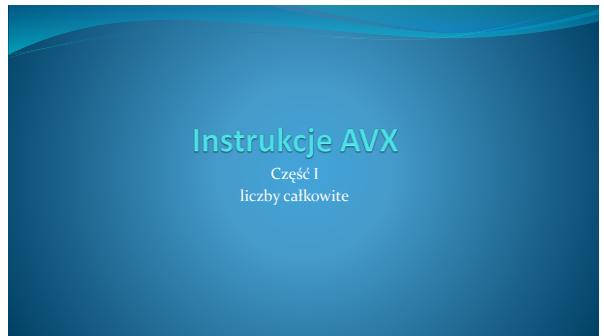
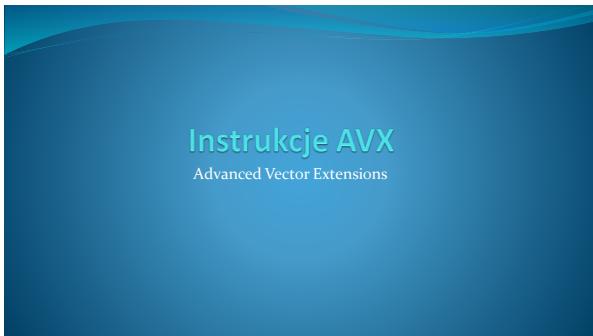


# Instrukcja PSHUFW

PSHUFW cel, źródło, kolejność

Tasuje słowa w rejestrze celu MMX. Źródłem jest rejestr MMX lub pamięć. Bity 7,6; 5,4 ;3,2 i 1,0 stałej kolejność określają numer słowa w źródle, które zostanie umieszczone jako 3,2,1 i 0 w rejestrze celu np.

dla źródła= **0123 4567 89ab cdefh** i kolejności 00 01 10 11b  
będzie: cel=**0cdef 89ab 4567 0123h**



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

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



Instrukcje AVX  
są sposobem zrównoleglenia programów  
na poziomie danych

(C) KISI d.KIK PCz

## Rejestry AVX ymm oraz zmm

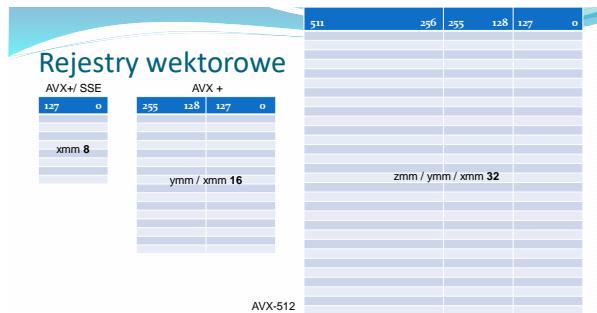
Dla rozkazów AVX dedykowano rejesty:

- 16 rejestrów 256 bitowych dla AVX oraz AVX2  
**ymm/<sub>15</sub> - ymm/o**
  - 32 rejesty 512 bitowe dla AVX-512  
**zmm/<sub>31</sub> - zmm/o**
  - Część instrukcji (większość) operuje na młodszej części rejestrów  
**xmm/[<sub>15</sub>-<sub>31</sub>] - xmm/o**

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

7



Rejestry XMM

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

zmm $\backslash$ /7	zmm $\backslash$ /6	zmm $\backslash$ /5	zmm $\backslash$ /4	zmm $\backslash$ /3	zmm $\backslash$ /2	zmm $\backslash$ /1	zmm $\backslash$ /0
				ymm $\backslash$ /3	ymm $\backslash$ /2	ymm $\backslash$ /1	ymm $\backslash$ /0
511				256	255	128	63
						127	64
					xmm2 $\backslash$ /1	xmm2 $\backslash$ /0	
						+	+
					xmm3 $\backslash$ /3 lub m128 $\backslash$ /1	xmm3 $\backslash$ /0 lub m128 $\backslash$ /0	
					=	=	
					xmm1 $\backslash$ /1	xmm1 $\backslash$ /0	

(C) KISI d.KIK PCz 2022

#### Programowanie niskopoziomowe

9

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/2	ymm/0
						xmm/1	xmm/0
511			236	255	128	127	64
				ymmm/3	ymmm/2	xmmm/1	xmmm/0
				+ ymmm3/3 lub m236/3	+ ymmm3/2 lub m236/2	+ xmmm3/1 lub m238/1	+ xmmm3/0 lub m238/0
				=	=	=	=
				ymmm/3	ymmm/2	xmmm/1	xmmm/0

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

10

Reistry XMM

Przykład: VADDPD       $xmm_1 = xmm_2 + xmm_3 / m128$

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

11

## Typy daných AVX

# Typy danych AVX

## Typy danych dla AVX

- Liczby całkowite (packed)

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

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

13

## Typy danych dla AVX

- Liczby zmienno-przecinkowe (packed i scalar)

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

- Specyfikatory zmiany typu:

xmmword ptr [adres]  
ymmword ptr [adres]

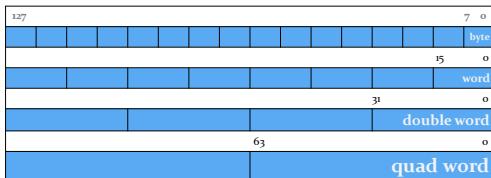
(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

14

## Typy danych AVX liczby całkowite

### Przykład na rejestrach 128 bitowych xmm



(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

15

## Typy danych AVX

### Liczby zmienno-przecinkowe

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

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

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

16

## Instrukcje AVX

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

17

## Instrukcje typy AVX

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

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

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

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

18

## Instrukcje typy 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 znaczące spowolnienie działania programu/podprogramu.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

19

## Instrukcje typy 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>

Ekwivalent np. dla **VADDPD**

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

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

20

## Systematyka instrukcji AVX

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

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

21

## Systematyka instrukcji AVX

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

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

22

## Budowa rozkazu AVX

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

23

## Budowa rozkazu AVX liczby całkowite

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

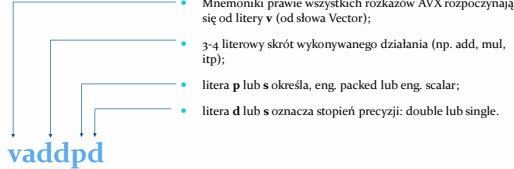
Rozkaz **vpaddb** wykonuje dodawanie (add) wektorowo/równolegle (p) liczb całkowitych w zakresie 8 bitów (b) i ewentualnie z nasyceniem

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

24

## Budowa rozkazu AVX liczby zmienno-przecinkowe



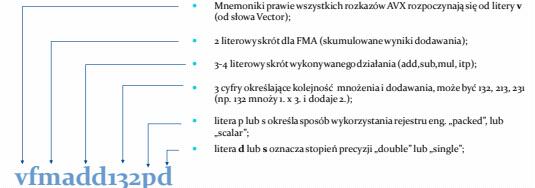
Rozkaz **vaddpd** wykonuje dodawanie (add) wektorowo/równolegle (p) liczb zmienno-przecinkowych podwójnej precyzji (d).

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

25

## Budowa rozkazu AVX operacje FMA



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

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

26

## Bajt sterujący imm8

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

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

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

np. 0ech

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

27

## Instrukcje AVX

dla liczb całkowitych

## Instrukcje AVX dla liczb całkowitych

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

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

28

## Operacje przesłania AVX

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

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

29

## Operacje przesłania 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 2022

Programowanie niekompresyjne

31

## Operacje przesłania AVX

- Instrukcje przetasowania:** VPSHUFB, VPSHUFWD, VPSHUFHW, VPSHUFLW
- Instrukcje permutacji:** VPERMD, VPERMQ
- Instrukcje mieszające:** VPBLENDB, VPBLENDW, VPBLENDWD
- Instrukcje rozgłaszania:** VPBROADCASTB, VPBROADCASTW, VPBROADCASTD, VPBROADCASTQ
- Instrukcje zbierania:** VPGATHERDD, VPGATHERQD, VPGATHERDQ, VPGATHERQQ

(C) KISI d.KIK PCz 2022

Programowanie niekompresyjne

32

## Instrukcja przesłania VMOV[D/Q]

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

VMOVD / VMOVQ przesyła podwójne lub poczwórne 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 2022

Programowanie niekompresyjne

33

## Instrukcja przesłania z wyrównaniem VMOVDQ[A/U]

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

Przesyła całą zawartość (128 lub 256 bitów) źródła do celu, jeśli cellem lub źródłem jest pamięć, wówczas w wersji (A) dane w pamięci muszą być wyrównane 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).

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

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompresyjne

34

## Instrukcja przesłania z wyrównaniem VMOVNTDQ[A]

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

Przesyła całą zawartość z/do pamięci m128/m256 do/z rejestru 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).

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

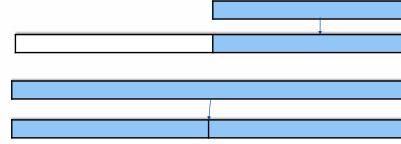
Programowanie niekompresyjne

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.



m128  
xmm1  
m256  
ymm1

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompresyjne

36

## Instrukcja przesłania VPMOVMSKB

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

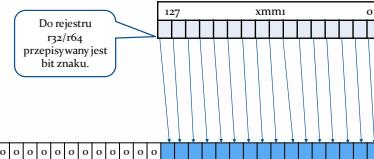
Przesyła najstarsze bity (bity znaku) **każdego bajtu** rejestrów xmmi/ymmi po kolejno do rejestrów r32/r64. Dla xmm przenosi 16 bitów do r32, dla ymm przenosi 32 bity do r64, pozostałe starsze bity są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

37

## Instrukcja przesłania VPMOVMSKB



r32/r64

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

38

## Instrukcje przesłania z konwersją

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

39

## Instrukcje przesłania z konwersją

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

vpmov[s/z]xbw xmmi, xmmm2/m64	vpmov[s/z]xwd xmmi, xmmm2/m64
vpmov[s/z]xbd xmmi, xmmm2/m32	vpmov[s/z]xwq xmmi, xmmm2/m32
vpmov[s/z]xbq xmmi, xmmm2/m16	vpmov[s/z]xdq xmmi, xmmm2/m16

Instrukcja zamienia (konwertuje) ze znakiem / bez znaku:

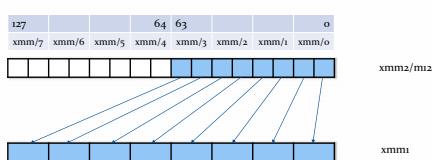
**bajty** na słowa/podwójne słowa/poczwórne słowa, **słowa** na podwójne słowa/poczwórne słowa, **podwójne słowa** na poczwórne słowa przepisując odpowiednio wartości z młodszej części rejestrów xmmm2/(m64||m32||m16) do rejestrów celu **xmmi**.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

40

## Instrukcja przesłania z konwersją VPMOVSBW



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

41

## Instrukcje przesłania z konwersją

VPMOVSBW[W/D/Q], VPMOVSXW[D/Q], VPMOVSXDQ

vpmov[s/z]xbw ymmi, ymmm2/m128	vpmov[s/z]xwd ymmi, ymmm2/m128
vpmov[s/z]xbd ymmi, ymmm2/m64	vpmov[s/z]xwq ymmi, ymmm2/m64
vpmov[s/z]xbq ymmi, ymmm2/m32	vpmov[s/z]xdq ymmi, ymmm2/m32

Instrukcja zamienia (konwertuje) ze znakiem / bez znaku:

**bajty** na słowa/podwójne słowa/poczwórne słowa, **słowa** na podwójne słowa/poczwórne słowa, **podwójne słowa** na poczwórne słowa przepisując odpowiednio wartości z młodszej części rejestrów ymmm2/(m128||m64||m32) do rejestrów celu **ymmi**, na młodszą część rejestrów.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

42



(C) KISI d.KIK PCz 2022

Programowanie niekompozycyjne

43

**Instrukcje przesłania - przykład:**

```

vmovdqu ymm1, ymmword ptr [rdi]           ; rdi = int * tab1]
vmovdqu ymm1, ymmword ptr [rdi+rax]         ; rdi = int * tab1[n]; rax = n
vmovdqu ymm2, ymmword ptr [rsi]              ; rsi = int * tab2]
vmovdqu ymm2, ymmword ptr [rsi+rcx]          ; rsi = int * tab2[m]; rcx=m

```

Ustalenie typu tablicowego dla całego rejestru ymm1/ymm2 czyli ładowanie (loading) adresu pierwszego elementu tablicy (komórki pamięci) oraz kolejnych n adresów (wielokrotność 4) o wielkości podwójnego słowa.

```

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

```

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

(C) KISI d.KIK PCz 2022

Programowanie niekompozycyjne

44



(C) KISI d.KIK PCz 2022

Programowanie niekompozycyjne

45

**Instrukcja przesłania warunkowego  
VMASKMOVDQU**

**vmaskmovdqu xmm1, xmm2**

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

**xmm2 = maska**

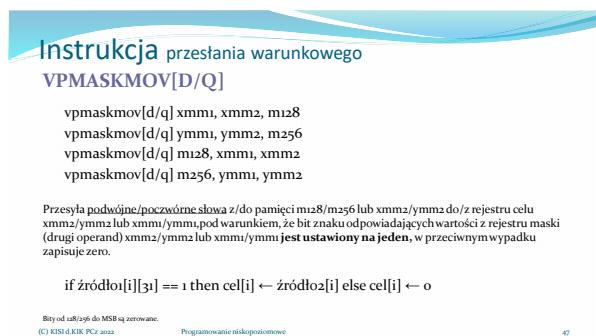
**if xmm2[i][7] = 1 then m128[i] = xmm1[i]**

i – jest numerem bajtu

(C) KISI d.KIK PCz 2022

Programowanie niekompozycyjne

46



Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompozycyjne

47



(C) KISI d.KIK PCz 2022

Programowanie niekompozycyjne

48

## Instrukcja kompresji

**VPACK[S/U]SWB**

vpack[s/u]swb xmmi, xmm2, xmm3/m128  
vpack[s/u]swb ymmi, ymm2, ymm3/m256 (AVX2)

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

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

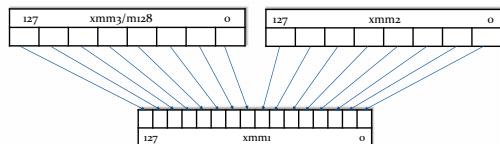
Programowanie niekropozitowne

49

## Instrukcja kompresji

**VPACK[S/U]SWB**

vpack[s/u]swb xmmi, xmm2, xmm3/m128



(C) KISI d.KIK PCz 2022

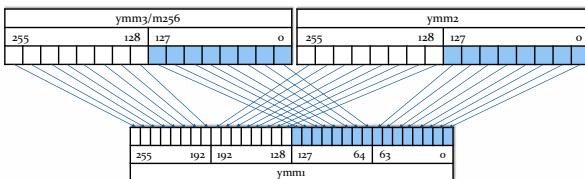
Programowanie niekropozitowne

50

## Instrukcja kompresji

**VPACK[S/U]SWB**

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



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

51

## Instrukcja kompresji

**VPACK[S/U]SDW**

vpack[s/u]sdw xmmi, xmm2, xmm3/m128  
vpack[s/u]sdw ymmi, ymm2, ymm3/m256 (AVX2)

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

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

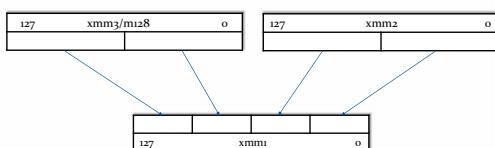
Programowanie niekropozitowne

52

## Instrukcja kompresji

**VPACK[S/U]SDW**

vpack[s/u]sdw xmmi, xmm2, xmm3/m128



(C) KISI d.KIK PCz 2022

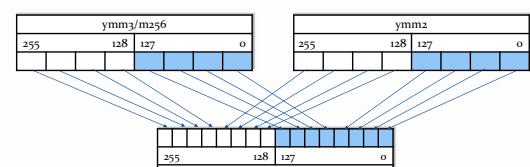
Programowanie niekropozitowne

53

## Instrukcja kompresji

**VPACK[S/U]SDW**

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



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

54

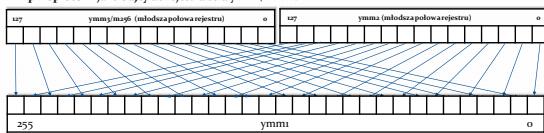
## Instrukcja kompresji

**VPUNPCKLBW** / VPUNPCKLWD / VPUNPCKLDQ / VPUNPCKLQDQ

vpunpcklbw xmmi, xmm2, xmm3/m128

vpunpcklbw ymmi, ymm2, ymm3/m256 (AVX2)

Bajty z młodszej polowy rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepłotem jako bajty do rejestru celu ymmi/ymmi.



(C) KISI d.KIK PCz 2022

Programowanie niekopiowalne

55

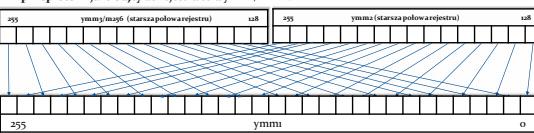
## Instrukcja kompresji

**VPUNPCKHBW** / VPUNPCKHWD / VPUNPCKHDQ / VPUNPCKHQDQ

vpunpckhbw xmmi, xmm2, xmm3/m128

vpunpckhbw ymmi, ymm2, ymm3/m256 (AVX2)

Bajty ze starszej polowy rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepłotem jako bajty do rejestru celu ymmi/ymmi.



(C) KISI d.KIK PCz 2022

Programowanie niekopiowalne

56

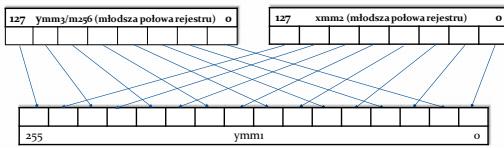
## Instrukcja kompresji

**VPUNPCKLBW / VPUNPCKLWD / VPUNPCKLDQ / VPUNPCKLQDQ**

vpunpcklwd xmmi, xmm2, xmm3/m128

vpunpcklwd ymmi, ymm2, ymm3/m256 (AVX2)

Słowa ze młodziej polowy rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepłotem jako słowa do rejestru celu ymmi/xmmi.



Bit od 128/129 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekopiowalne

57

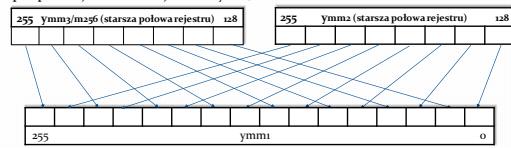
## Instrukcja kompresji

**VPUNPCKHWB / VPUNPCKHWD / VPUNPCKHDQ / VPUNPCKHQDQ**

vpunpckhwd xmmi, xmm2, xmm3/m128

vpunpckhwd ymmi, ymm2, ymm3/m256 (AVX2)

Słowa ze starszej polowy rejestru ymm2 (xmm2) oraz ymm3 (xmm3)/m256(m128) zapisuje z przepłotem jako słowa do rejestru celu ymmi/xmmi.



Bit od 128/129 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekopiowalne

58

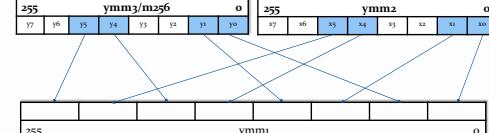
## Instrukcja kompresji

**VPUNPCKLBW / VPUNPCKLWD / VPUNPCKLDQ / VPUNPCKLQDQ**

vpunpckldq xmmi, xmm2, xmm3/m128

vpunpcklddq ymmi, ymm2, ymm3/m256 (AVX2)

Przepisuje podwójne słowa. Pary młodszych podwójnych słów z rejestru xmm2/ymm2 i xmm3/m128 / ymm3/m256 są zapisywane z przepłotem jako podwójne słowa do rejestru xmmi/ymmi.



Bit od 128/129 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekopiowalne

59

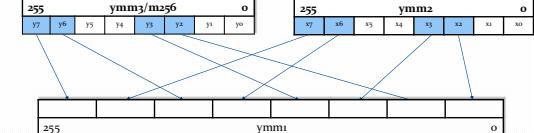
## Instrukcja kompresji

**VPUNPCKHWB / VPUNPCKHWD / VPUNPCKHDQ / VPUNPCKHQDQ**

vpunpckhwd xmmi, xmm2, xmm3/m128

vpunpckhwd ymmi, ymm2, ymm3/m256 (AVX2)

Przepisuje podwójne słowa. Pary starszych podwójnych słów z rejestru xmm2/ymm2 i xmm3/m128 / ymm3/m256 są zapisywane z przepłotem jako podwójne słowa do rejestru xmmi/ymmi.



Bit od 128/129 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekopiowalne

60

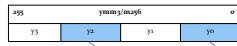
## Instrukcja kompresji

VPUNPCKLBW / VPUNPCKLWD / VPUNPCKLDQ / VPUNPCKLQDQ

vpunpcklqdq xmm1, xmm2, xmm3/m128

vpunpcklqdq ymm1, ymm2, ymm3/m256 (AVX2)

Przepisuje młodsze poczwórne słowa z rejestru xmmz/ymmz oraz młodsze z rejestru xmm3/m128/ymm3/m256 z przeplatem do rejestru xmmy/ymmu.



Bity od 128/196 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie indeksopozycjowe

61



Bity od 128/196 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie indeksopozycjowe

62

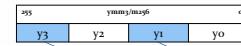
## Instrukcja kompresji

VPUNPCKHBW / VPUNPCKHWD / VPUNPCKHDQ / VPUNPCKHQDQ

vpunpckhqdq xmm1, xmm2, xmm3/m128

vpunpckhqdq ymm1, ymm2, ymm3/m256 (AVX2)

Przepisuje młodsze poczwórne słowa z rejestru xmmz/ymmz oraz młodsze z rejestru xmm3/m128/ymm3/m256 z przeplatem do rejestru xmmy/ymmu.

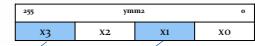


Bity od 128/196 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie indeksopozycjowe

63



Bity od 128/196 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie indeksopozycjowe

64

## Instrukcje wytłuskiwania / wstawiania

(C) KISI d.KIK PCz 2022

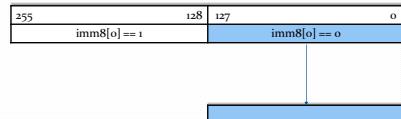
Programowanie indeksopozycjowe

65

## Instrukcja wytłuskiwania VEXTRACTH128 / VINSETH128

vextractiiz8 xmm1/m128, ymm2, imm8 (AVX2)

Przepisuje 128 bitów z rejestru ymm do rejestru xmm lub muż. Jeśli zerowy bitu bajtu sterującego imm8[0] = 0 przepisuje xmm3/m128 na mloszą częśc, gdy imm8[0] = 1 na starszą częśc rejestru ymm2.



ymm2

xmm1/m128

(C) KISI d.KIK PCz 2022

Programowanie indeksopozycjowe

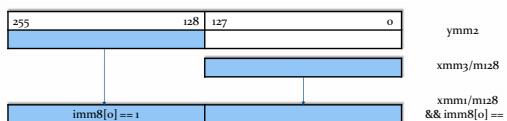
66

## Instrukcja wstawiania

VINSETTH128 / VINSERTI128

vinsertiiz8 ymm1, ymm2, xmm1/m128, imm8 (AVX)

Przepisuje cały reżestr ymm2 do ymmi następnie przepisuje reżestr xmm3 lub muż również do rejestru celu ymmi zależnie od ustawienia bitu bajtu sterującego imm8[0] = 0 przepisuje xmm3/m128 na mloszą częśc, gdy imm8[0] = 1 na starszą częśc celu ymmi.



(C) KISI d.KIK PCz 2022

Programowanie indeksopozycjowe

67

## Instrukcje tasowania

(C) KISI d.KIK PCz 2022

Programowanie indeksopozycjowe

68

## Instrukcja tasowania VPSHUFB

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

Tasuje bajty z xmm3/ymm3/m128, w zależności od bitu znaku kolejnych bajtów rejestru xmm3/m128 / ymm3/m256 jeśli bit znaku jest ustawiony odpowiedni bajt rejestru całego xmm1/ymm1 jest zerowany, jeśli bit znaku xmm3/ymm3/m256 nie jest ustawiony wówczas z takiego bajtu jest tworzony 4 bajty lub 5 (ymm) bitowy indeks wskazujący numer bajtu, który ma być przepisany z xmm1/ymm2 do właściwego xmm1/ymm1.

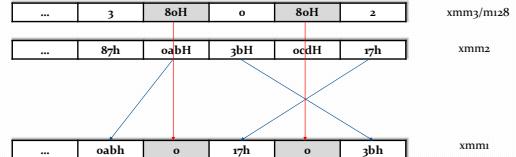
```
i = numer bajtu
if xmm3/m128[i][7] == 1 then xmm1[i] = 0
else { index = xmm3/m128[i]
        xmm1[i] = xmm2[index]
    }
```

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

67

## Instrukcja tasowania VPSHUFB



xmm3/m128

xmm2

xmm1

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

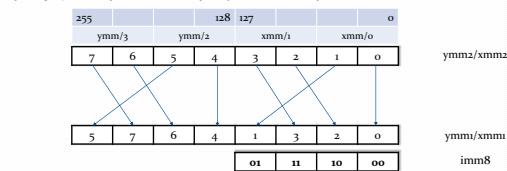
68

## Instrukcja tasowania VPSHUFD

`vpshufd xmm1, xmm2 imm8`

`vpshufd ymm1, ymm2, imm8 (AVX2)`

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



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

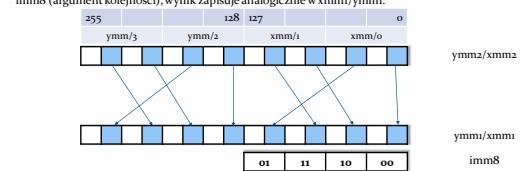
69

## Instrukcja tasowania VPSHUFLW

`vpshuflw xmm1, xmm2 imm8 (AVX)`

`vpshuflw ymm1, ymm2, imm8 (AVX2)`

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



ymm2/xmm2

ymm1/xmm1

imm8

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

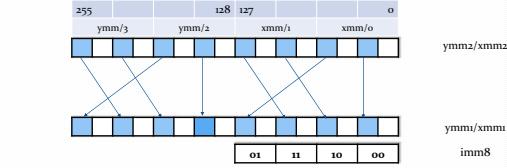
70

## Instrukcja tasowania VPSHUFHW

`vpshufhw xmm1, xmm2 imm8 (AVX)`

`vpshufhw ymm1, ymm2, imm8 (AVX2)`

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



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

71

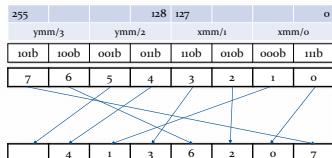
## Instrukcje permutacji

## Instrukcja tasowania

### VPERMD

**vpermd** ymm1, ymm2, ymm3/m256 (AVX2)

Wykonyuje permutację wektorów podwójnych słów z rejestru **ymm3/m256** według porządku podanego w **ymm2**, najmłodsze 2 bity odwoływanej podwójnej słowa rejestru ymm3 wskazują, z którego miejsca w ymm3/m256 zostanie skopiowane podwójne słwo do miejsca położenia „adresu” (ymm1). Wynik jest zapisywany w **ymm1**.



(C) KISI d.KIK PCz 2022

Programowanie niekoprozessowe

73

ymm3/m256  
ymm2/xmm2

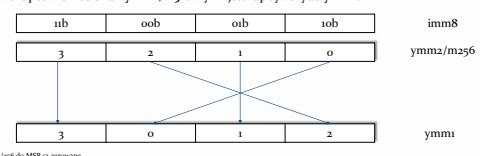
ymmi/xmmi

## Instrukcja permutacji

### VPERMQ

**vpermq** ymm1, ymm2/m256, imm8 (AVX2)

Wykonyuje permutację wektorów poczwórnego słowa z rejestru **ymm2/m256** według porządku określonego w **imm8**, kolejne dwubitowe pola **imm8[0:1]** oznaczają, z spod którego indeksu „adresu” zostaną skopiowane poczwórne słowa z **ymm2/m256**. Wynik jest zapisywany do **ymm1**.



(C) KISI d.KIK PCz 2022

Programowanie niekoprozessowe

74

imm8  
ymm2/m256

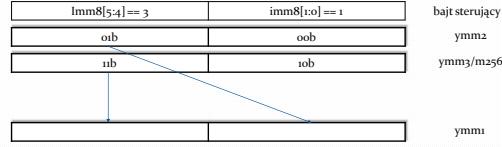
ymmu

## Instrukcja permutacji

### VPERM2I128

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

Wykonyuje permutację dwóch wektorów 128-bitowych z rejestrów **ymm2** oraz **ymm3/m256**, bajt sterujący **imm8** odpowiada za sposób przepisania, pola **imm8[5:4]** i **imm8[0:1]** są indeksami wskazującymi skład należą pobrać starszą i młodszą część rejestru celu, bity **imm8[7] = 1** i **imm8[3] = 1** powodują wykorzystanie starszej i młodszej części.



(C) KISI d.KIK PCz 2022

Programowanie niekoprozessowe

75

## Instrukcja mieszająca

### VPBLENDVB

**vpblendvb** xmm1, xmm2, xmm3/m256, xmm4 (AVX)

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

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

```
i < 15> lub <0, 32> - ilość bajtów
if źródło2[i]>7 && i <= cel[i] => cel[i] = źródło2[i]
else cel[i] = źródło2[i]

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

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

Bit od 128/256 do MSB są zerowane.

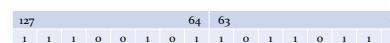
(C) KISI d.KIK PCz 2022

Programowanie niekoprozessowe

77

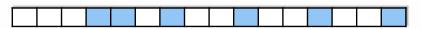
## Instrukcja mieszająca

### VPBLENDVB



xmm4 bit znaku

xmm3/mmm128



xmm2

xmm1

(C) KISI d.KIK PCz 2022

Programowanie niekoprozessowe

78

## Instrukcja mieszająca VPBLENDW

Miesza wektory słów. W oparciu o bajt kontrolny wybiera z rejestru xmm3/ymm3 lub m128/m256 elementy wektora dla imm8[i] = 1, w przypadku wartości indeksu 8-15 należy odjąć 8 aby odwołać się do bitu z bajtu kontrolnego. Elementy wektora dla których imm8 = 0 są uzupełniane odpowiednim elementem xmm2/ymm2. Wynik zapisuje w xmmi/ymmi.

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

if imm8[i] = 1 then xmmi[i] = xmm3/m128[i]
else xmmi[i] = xmm2[i]

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

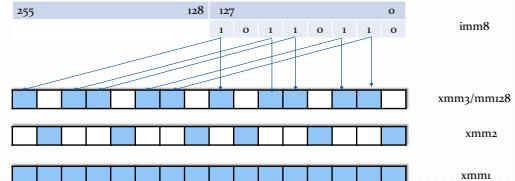
Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

79

## Instrukcja mieszająca VPBLENDW



(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

80

## Instrukcja mieszająca VPBLENDD

vblendd ymmi, 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 ymmi.

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

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

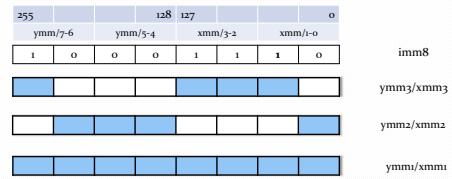
Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

81

## Instrukcja mieszająca VPBLENDD



(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

82

## Instrukcje rozgłaszające

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

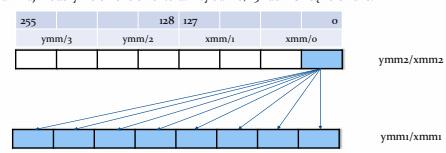
83

## Instrukcja rozgłaszańska VPBROADCAST[B/W/Q]

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

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

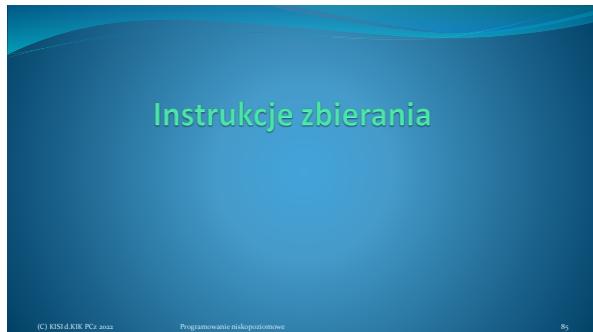
Rozgłasza te samą wartość 8/16/32/64 bitową ze źródła xmm2/ymm2 lub pamięci m8/m16/m32/m64 do wszystkich elementów wektora rejestru celu xmmi/ymmi. Jeśli operand źródłowy jest rejestrem wartość rozgłoszana w najmłodszym elemencie wektora. Bity od 128/256 do MSB są zerowane.



(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

84



(C) KISI d.KIK PCz 2022

Programowanie niekropozitomowe

85

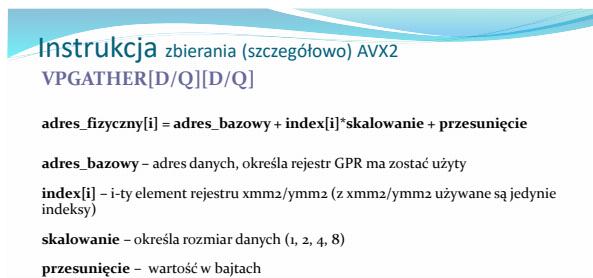


Bity od 10h do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitomowe

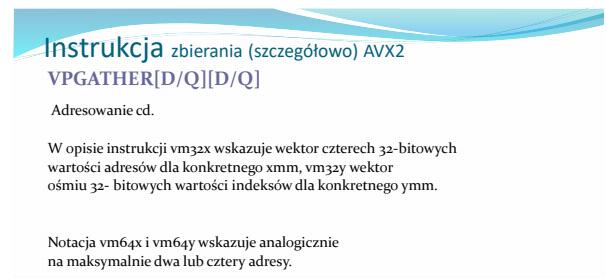
86



(C) KISI d.KIK PCz 2022

Programowanie niekropozitomowe

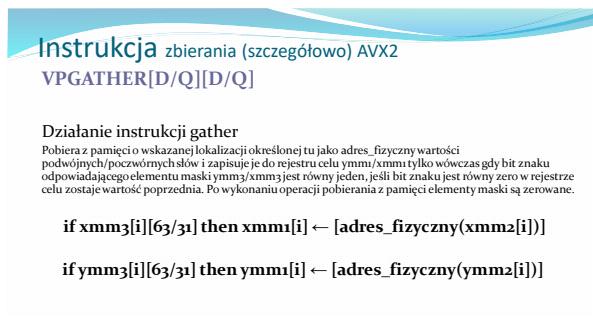
87



(C) KISI d.KIK PCz 2022

Programowanie niekropozitomowe

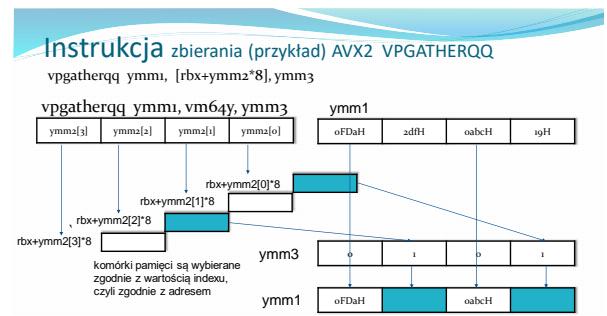
88



(C) KISI d.KIK PCz 2022

Programowanie niekropozitomowe

89



(C) KISI d.KIK PCz 2022

Programowanie niekropozitomowe

90

## Instrukcja zbierania (ogólnie) AVX2 VPGATHER[D/Q][D/Q]

Wyjście[i] = Wejście[Index[i]] Gather AVX2  
 Wyjście[Index[i]] = Wejście[i] Scather AVX-512

a0	a1	a2	a3	a4	a5	a6	a7
6	4	7	0	1	3	2	5
a6	a4	a7	a0	a1	a3	a2	a5

Model instrukcji Gather

Wejście[i]  
dane z pamięci

Index[i]

Wyjście[i]  
rejestr celu

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      91

## Instrukcja zbierania (ogólnie) AVX2 VPGATHER[D/Q][D/Q]

Różnica pomiędzy instrukcjami  
*gather* a *blend/perm/shuf*

Instrukcje *blend/perm/shuf* operują na rejestrach,  
zatem najpierw należy załadować dane  
z pamięci do rejestru ymm/xmm.

Instrukcja *gather* pobiera dane bezpośrednio z pamięci,  
jednak wcześniej trzeba przygotować rejestr indeksów  
(rejestr porządku).

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      92

## Operacje arytmetyczne

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      93

## Operacje arytmetyczne AVX

- Dodawanie:** VPADD, VPADDW, VPADD, VPADD, VPADD  
VPADDSB, VPADDSW, VPADDUS, VPADDUSW  
VPADDW, VPADD, VPADDWS
- Odejmowanie:** VPSUBB, VPSUBW, VPSUBD, VPSUBQ  
VPSUBSB, VPSUBSW, VPSUBSB, VPSUBSW  
VPSUBW, VPHSUBD, VPHSUBW, VPSADB
- Mnożenie:** VPMULLW, VPMULLD, VPMULHUW  
VPMULHW, VPMULHRSW, VPMULDQ, VPMILUDQ  
VPCMULQDQ
- Mnożenie i dodawanie:** VPMADDWD, VPMADDUBSW

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      94

## Instrukcje dodawania

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      95

## Instrukcja dodawania VPADD[B/W/D/Q]

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

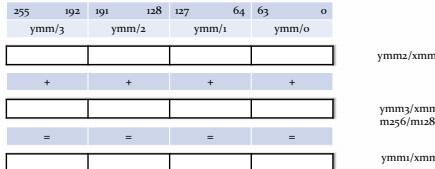
Do wartości bajtów/słów/podwójnych słów/poczwórnego słowa z rejestru **xmm2/ymm2** są dodawane równolegle odpowiednie wartości z rejestru **xmm3/ymm3** lub z pamięci **m128/m256**, wynik jest zapisywany w rejestrze **xmm1/ymm1**.

cel = źródło1 + źródło2  
 $xmm1 = xmm2 + xmm3/m128$   
 $ymm1 = ymm2 + ymm3/m256$

Bity od 128-256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      96

## Instrukcja dodawania VPADDQ



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

97

## Instrukcja dodawania VPADDS[B/W]

vpaddsb xmm1, xmm2, xmm3/m128  
vpaddsd 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 nasyceniem w rejestrze xmm1/ymm1.

cel = źródło1 + źródło2

xmm1 = xmm2 + xmm3/m128

ymm1 = ymm2 + ymm3/m256

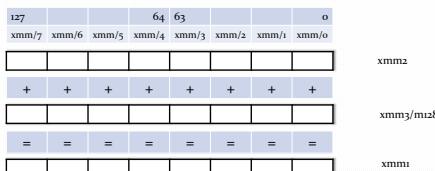
Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

98

## Instrukcja dodawania VPADDSW



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

99

## Instrukcja dodawania VPADDUS[B/W]

vpaddusb xmm1, xmm2, xmm3/m128  
vpaddud 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 nasyceniem w rejestrze xmm1/ymm1.

cel = źródło1 + źródło2

xmm1 = xmm2 + xmm3/m128

ymm1 = ymm2 + ymm3/m256

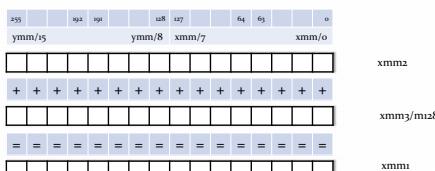
Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

100

## Instrukcja dodawania VPADDUSW



(C) KISI d.KIK PCz 2022

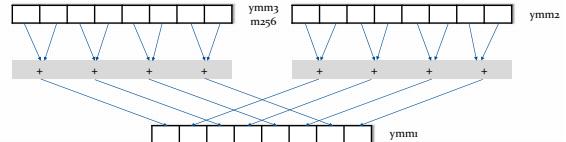
Programowanie niekropozitowane

101

## Instrukcja dodawania VPHADDW / VPHADD / VPHADDSW

vphaddd ymm1, ymm2, ymm3/m256

Horyzontalne dodawanie sąsiednich słów/podwójnych słów i zapisywanie wyniku z przepłotem. Jako najmłodsze są zapisywane sumy z rejestru xmm2/ymm2 (operand 2). Ostatnia w/w instrukcja jest dodawaniem słów z nasyceniem.



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

102

## Instrukcje odejmowania

**VPSUB[B/W/D/Q]**

255	192	191	128	127	64	63	0
ymm3		ymm2	xmm1	xmm3/m128	xmm2	xmm3/m256	xmm/o
-	-	-	-	-	-	-	-
=	=	=	=	=	=	=	=

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      103

## Instrukcja odejmowania

**VPSUB[B/W/D/Q]**

vpsub[b/w/d/q] xmml, xmml, xmml3/m128  
vpsub[b/w/d/q] ymm1, ymm1, ymm3/m256 (AVX2)

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

cel = źródło1 - źródło2  
xmm1 = xmm2 - xmm3/m128  
ymm1 = ymm2 - ymm3/m256

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      104

## Instrukcja odejmowania

**VPSUBQ**

255	192	191	128	127	64	63	0
ymm3/3		ymm2/2	xmm1/1	xmm3/m128	xmm2	xmm3/m256	xmm/o
-	-	-	-	-	-	-	-
=	=	=	=	=	=	=	=

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      105

## Instrukcja odejmowania

**VPADD[U]S[B/W]**

vpsub[u]s[b/w] xmml, xmml, xmml3/m128  
vpsub[u]s[b/w] ymm1, ymm1, 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**.

cel = źródło1 - źródło2  
xmm1 = xmm2 - xmm3/m128  
ymm1 = ymm2 - ymm3/m256

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      106

## Instrukcja odejmowania

**VPSUBSW**

127	64	63	0				
xmm7	xmm6	xmm5	xmm4	xmm3	xmm2	xmm1	xmm0
-	-	-	-	-	-	-	-
=	=	=	=	=	=	=	=

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      107

## Instrukcja odejmowania

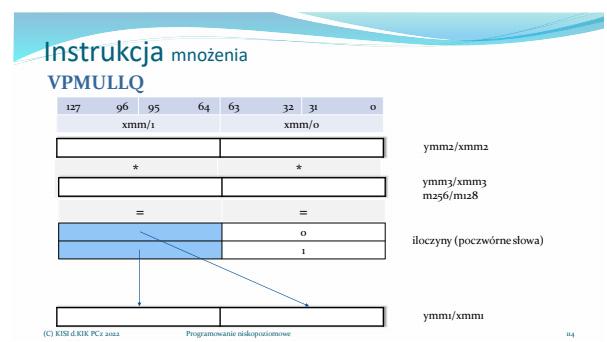
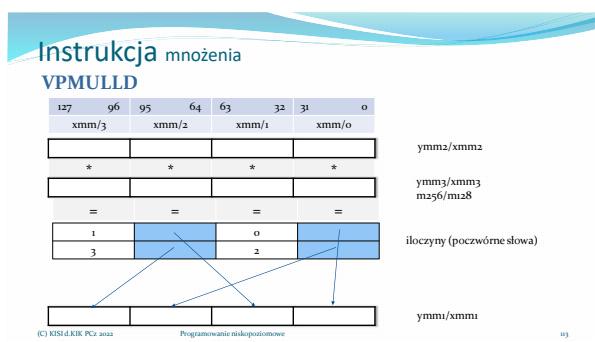
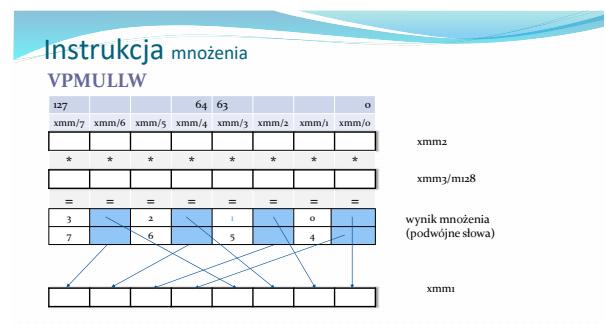
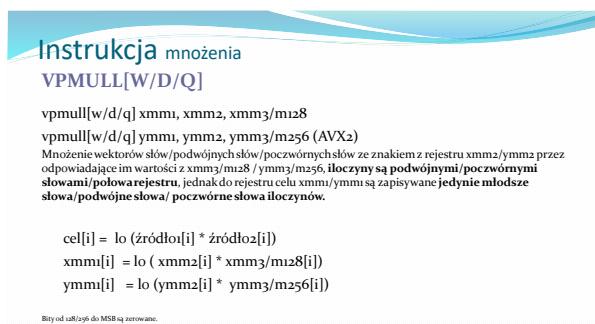
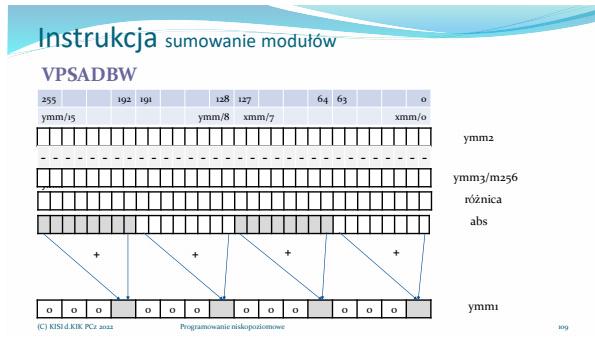
**VPHSUBW / VPHSUBD / VPHSUBSW**

vphsubd ymm1, ymm2, ymm3/m256

**Horyzontalne odejmowanie** sąsiednich słów/podwójnych słów i zapisywanie wyniku z przeplatem. **Od m�odszego elementu wektora jest odejmowany starszy, oraz jako najmłodsze** są zapisywane różnice z rejestru xmm2/ymm2 (operand 2). Ostatnia instrukcja jest odejmowaniem słów z nasyceniem.

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      108



## Instrukcja mnożenia VPMUH[U]W

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

Mnożenie wektorów słów bez znaku/ze znakiem (U) z rejestrów xmm2/ymm2 przez odpowiadające im wartości z xmm3/m128 / ymm3/m256. **iloczyny są podwójnymi**, jednak do rejestrów celu xmmi/ymmi są zapisywane **jedynie starsze słowa**, iloczynów.

cel[i] = hi (źródło[i] x źródło2[i])

xmmi[i] = hi (xmm2[i] x xmm3/m128[i])

ymmi[i] = hi (ymm2[i] x ymm3/m256[i])

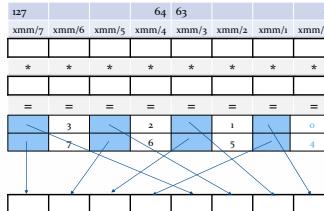
Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekoprozbowe

105

## Instrukcja mnożenia VPMULHW



xmm2

xmm3/m128

wynik mnożenia  
(podwójne słowa)

xmmi

(C) KISI d.KIK PCz 2022

Programowanie niekoprozbowe

106

## Instrukcja mnożenia VPMULHRSW

vpmulhrsw xmmi, xmm2, xmm3/m128  
vpmulhrsw ymmi, ymm2, ymm3/m256 (AVX2)

Mnożenie wektorów słów ze znakiem ze skalowaniem i zaokrągleniem, wartości z rejestrów xmm2/ymm2 przez wartości z rejestrów xmm3/m128 / ymm3/m256, podwójne słowa iloczynów zostają przesunięte prawo o 14 bitów oraz zostaje dodana jedynka w celu zaokrąglenia wartości. Bity od 1 do 16 są zapisywane w celu.

cel[i] = ((źródło[i] \* źródło2[i] >> 14) + 1) >> 1

xmmi[i] = ((xmm2[i] \* xmm3/m128[i] >> 14) + 1) >> 1

ymmi[i] = ((ymm2[i] \* ymm3/m256[i] >> 14) + 1) >> 1

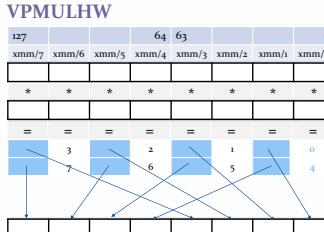
Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekoprozbowe

107

## Instrukcja mnożenia VPMULHW



xmm2

xmm3/m128

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

xmmi

(C) KISI d.KIK PCz 2022

Programowanie niekoprozbowe

108

## Instrukcja mnożenia VPMUL[U]DQ

vpmul[u]dq xmmi, xmm2, xmm3/m128  
vpmul[u]dq ymmi, ymm2, ymm3/m256 (AVX2)

Mnożenie **co drugim** 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 xmmi/ymmi jako wektor poczwórnego słów ze znakiem.

cel[i] = źródło[i] \* źródło2[i]

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

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

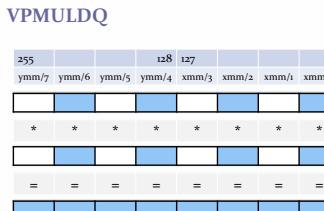
Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekoprozbowe

109

## Instrukcja odejmowanie VPMULDQ



xmm2

xmm3/m128

xmmi

(C) KISI d.KIK PCz 2022

Programowanie niekoprozbowe

110

## Instrukcja mnożenia VPCLMULQDQ

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

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

```
if imm8[0] = 0 || 1 && imm8[4] = 0 || 1 => cel <- źródło1[0 || 1] * źródło2[0 || 1]
if imm8[0] = 0 = 0 && imm8[4] = 0 => xmm1 <- xmm2[63:0] * xmm3/m128[63:0]
if imm8[0] = 0 && imm8[4] = 1 => xmm1 <- xmm2[63:0] * 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]
```

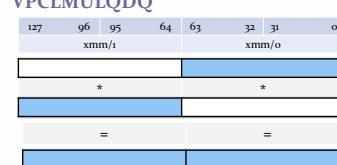
Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

121

## Instrukcja mnożenia VPCLMULQDQ



yymm2/xmm2  
&& imm8[0] = 0

yymm3/xmm3 lub m256/m128  
&& imm8[4] = 1

yymm1/xmm1

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

122

## Instrukcje mnożenia z dodawaniem

Dla liczb zmiennoprzecinkowych  
odpowiednikiem bardziej zaawansowanym  
są instrukcje FMA

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

123

## 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] = źródło1[2i] * źródło2[2i] + źródło1[2i+1] * źródło2[2i+1]
xmm1[i] = xmm2[2i] * xmm3/m128[2i] + xmm2[2i+1] * xmm3/m128[2i+1]
ymm1[i] = ymm2[2i] * ymm3/m128[2i] + ymm2[2i+1] * ymm3/m128[2i+1]
```

Bit od 128/256 do MSB są zerowane.

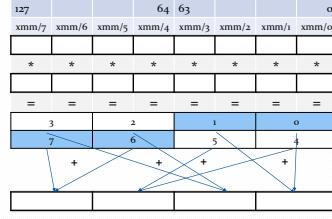
(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

124

## Instrukcja mnożenia

### VPMADDWD



xmm2

xmm3/m128

iloczyny

sumuje sąsiednie dwa iloczyny

xmm1

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

125

## 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, iloczyny są słowami, następnie dwa kolejne słowa dodaje horyzontalnie i zapisuje z nasyceniem, jako słowa w rejestrze celu xmm1/ymm1.

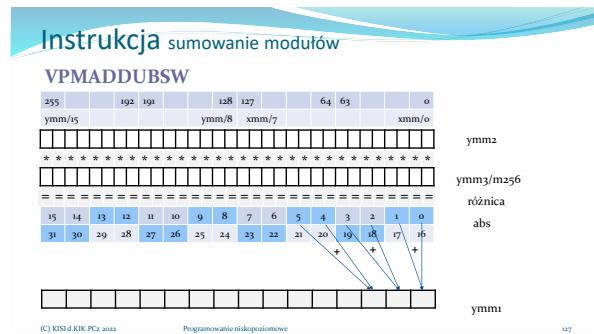
```
cel[i] = źródło1[2i] * źródło2[2i] + źródło1[2i+1] * źródło2[2i+1]
xmm1[i] = xmm2[2i] * xmm3/m128[2i] + xmm2[2i+1] * xmm3/m128[2i+1]
ymm1[i] = ymm2[2i] * ymm3/m128[2i] + ymm2[2i+1] * ymm3/m128[2i+1]
```

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

126



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

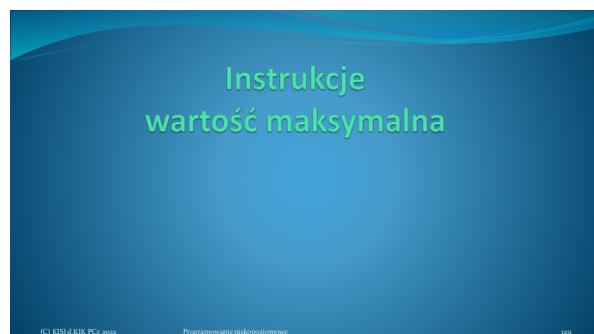
127



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

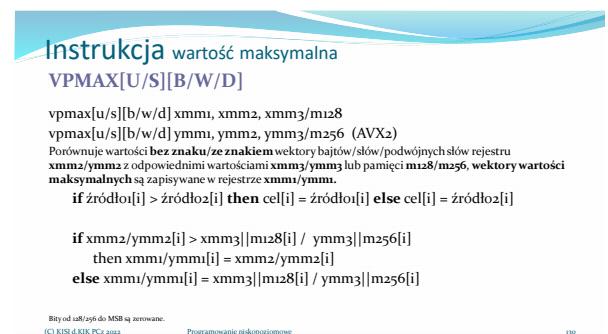
128



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

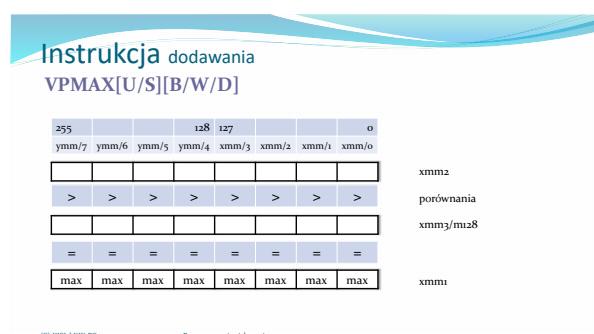
129



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

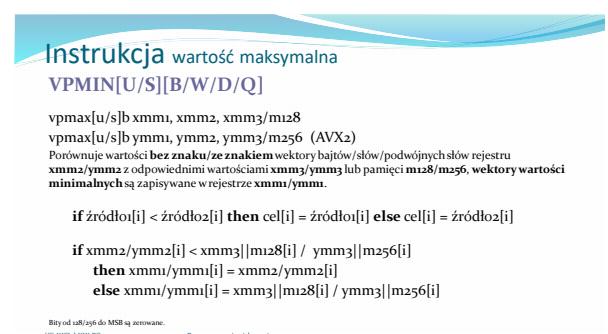
130



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

131



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowane

132

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

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

xmm2

porównania

xmm3/m128

xmm1

(C) KISI d.KIK PCz 2022

Programowanie niekompozycyjne

133

## Instrukcje wartość średnia

(C) KISI d.KIK PCz 2022

Programowanie niekompozycyjne

134

## Instrukcja wartość średnia VPAVG[B/W]

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

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

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

cell[i] = (źródło1[i] + źródło2[i] + 1) &gt;&gt; 1

xmm1[i] = (xmm2[i] + xmm3/m128[i] + 1) &gt;&gt; 1

ymm1[i] = (ymm2[i] + ymm3/m256[i] + 1) &gt;&gt; 1

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompozycyjne

135

## Instrukcja dodawania VPAVG[B/W]

vpavg[b/w]

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

xmm2

xmm3/m128

xmm1

(C) KISI d.KIK PCz 2022

Programowanie niekompozycyjne

136

## Instrukcje wartości bezwzględnej

(C) KISI d.KIK PCz 2022

Programowanie niekompozycyjne

137

## 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**.

cell[i] = abs(źródło[i])

xmm1[i] = abs(xmm2[i])

ymm1[i] = abs(ymm2[i])

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompozycyjne

138

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

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

xmm2

| abs |
|-----|-----|-----|-----|-----|-----|-----|-----|
| =   | =   | =   | =   | =   | =   | =   | =   |
|     |     |     |     |     |     |     |     |

xmm1

(C) KISI d.KIK PCz 2022

Programowanie niekopiowane

139

## Instrukcje znaku pozostawienie / zerowanie / negacja

(C) KISI d.KIK PCz 2022

Programowanie niekopiowane

140

## 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 wartością z rejestru xmm2 w zależności od znaku odpowiadającej wartości wektora w rejestrze xmm3/m128.

```
if źródło2[i] > o; cel[i] = źródło1[i]
if źródło2[i] = o; cel[i] = o
if źródło2[i] < o; cel[i] = - źródło1[i]
```

Bity od 255 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekopiowane

141

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

**xmm**

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

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

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

xmm2

xmm3/m256  
odczytanie bitu znakuxmm3/m128  
wyznaczenie funkcji  
signum

xmm1

(C) KISI d.KIK PCz 2022

Programowanie niekopiowane

143

## Operacje porównania AVX

- Porównanie liczb:** VPCMPEQB, VPCMPEQW  
 VPCMPEQD, VPCMPEQD, VPCMPEQD, VPCMPEQD  
 VPCMPEQD, VPCMPEQD, VPCMPEQD
- Porównanie ciągów:**  
 VPCMPESTRI, VPCMPESTRI, VPCMPESTRI, VPCMPESTRI  
 VPCMPESTRI, VPCMPESTRI

## Instrukcje porównania

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      145

### Instrukcja porównania liczb VPCMPEQ[B/W/D]

vpcmpeq[b/w/d] xmmi, xmmt, xmm3/m128  
vpcmoeq[b/w/d] ymmi, ymm2, ymm3/m256 (AVX2)

Porównuje wartości współrzędnych wektorów bajtów/słów/podwójnych słów z rejestru xmm3/ymm3 lub pamięci m128/m256 z odpowiednimi wartościami współrzędnych rejestru xmm2/ymm2, jeśli wartości są równe, odpowiednio współrzędne rejestru celu xmmi/ymmi 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 xmmi[i] = -1 else xmmt[i] = 0;
if ymm3/m256[i] = ymm2[i] then ymmi[i] = -1 else ymm2[i] = 0;
```

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      146

## Instrukcja porównania liczb VPCMPEQ[B/W/D]

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      147

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

xmm3/m128

xmm2

xmmi

### Instrukcja porównania liczb VPCMPGT[B/W/D/Q]

vpcmpgt[b/w/d] xmmi, xmmt, xmm3/m128  
vpcmpgt[b/w/d] ymmi, ymm2, ymm3/m256 (AVX2)

Porównuje wartości współrzędnych wektorów bajtów/słów/poczwórnego słowa/poczwórnego słowa 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 odpowiednio współrzędne rejestru celu xmmi/ymmi są ustawiane 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] => xmmi[i] = -1 else xmmt[i] = 0
if ymm2[i] > ymm3/m256[i] then ymmi[i] = -1 else ymm2[i] = 0
```

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      148

## Instrukcja porównania liczb VPCMPGT[B/W/D/Q]

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      149

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

xmm3/m128

xmm2

xmmi

### Instrukcje porównania ciągów znakowych

- Operacje porównania ciągów znakowych porównują w istocie liczby całkowite.
- Instrukcje te można podzielić na porównujące ciągi znakowe o, ustalonej (znanej) w rejestrach [R/E]AX i [R/E]DX oraz nieznanej, długości.
- Instrukcje CMP na wyjściu tworzą indeks lub maskę ale wynik porównania jest zapisywany w [R/E]CX/xmmo (brak w definicji 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 2022      Programowanie niekompromisowe      150

## 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] = {**oob** bajty bez znaku, **orb** słowa bez znaku,  
**rb** bajty ze znakiem, **nb** słowa ze znakiem}

#### Operacja(sposób porównania)

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

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

19

## 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] = {**oob** pozytywna polaryzacja (bez zmiany znaku),  
**ob** negatywna polaryzacja (ze zmianą znaku),  
**rb** stosowanie maski (bez zmiany znaku),  
**nb** stosowanie maski (ze zmianą maski) dla elementów nieważnych w reg/mem i są przepisywane, w przeciwnym wypadku nieważne są negowane }

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

193

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

### Bajt sterujący imm8 (3/4)

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

#### Wynik zwracany w postaci indeksu lub maski jest tworzony etapami:

1. logiczne porównanie każdy z każdym (bajt z bajtem / słowo ze słowem)
2. intermediate result 1 (pośrednie zagrégowane wyniki porównania - prawdziwe)
3. intermediate result 2 etap poprzedni jest negowany logicznie
4. zwracany jest albo najbardziej/najmniej znaczący bit 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 2022

Programowanie niekompromisowe

193

## 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 ECX/RCX

imm8[6] = {ob z wyniku jest pobierany najmłodszy bit

ib z wyniku jest pobierany najstarszy bit }

Dla maski (M) = wynik do xmm0

imm8[6] = {ob zwracany wynik uzupełniany jest zerami,

ib wynik jest rozszerzany do bajtu/słowa (z samymi zerami lub jedynkami) }

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

194

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

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

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

Instrukcje typu PCMPxSTRx 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 2022

Programowanie niekompromisowe

195

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

Algorytm instrukcji definiowany w bajcie sterującym:

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

Bajt kontrolny określa spodziewany wynik i kontroluje następujące atrybuty:

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

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

- maska w xmm0 lub index w [R/E]AX
- ustawione flagi.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

196

## Instrukcja porównania ciągów znakowych VPCMPESTRI

vpcmpestri xmmi, xmm2/m128, imm8 (AVX)

Porównuje **łańcuchy o ustalonej długości** z rejestrów xmmi 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
xmmi	xmm2/m128	[R/E]AX	[R/E]DX	ECX

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

197

## Instrukcja porównania ciągów znakowych VPCMPISTRI

vpcmpistri xmmi, xmm2/m128, imm8 (AVX)

I (Implicit) - oznacza łańcuch o nieoznaczonej długości.  
I (Index) – oznacza, że instrukcja tworzy na wyjściu index

Porównuje **łańcuchy o nie ustalonej długości** z rejestrów xmmi i xmm2/m128, na wyjściu tworzy index, wynik zapisuje w rejestrze podstawowego przeznaczenia ECX.

Operand 1	Operand 2	Wynik
xmmi	xmm2/m128	ECX

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

198

## Instrukcja porównania ciągów znakowych VPCMPESTRM

vpcmpestrm xmmi, 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 rejestrów xmmi i xmm2/m128, na wyjściu tworzy maskę, wynik zapisuje w rejestrze **xmmo**. (nie jest umieszczany w definicji).

Operand 1	Operand 2	Długość 1	Długość 2	Wynik
xmmi	xmm2/m128	[R/E]AX	[R/E]DX	xmmo

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

199

## Instrukcja porównania ciągów znakowych VPCMPISTRM

vpcmpistrm xmmi, xmm2/m128, imm8 (AVX)

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

Porównuje **łańcuchy o nie ustalonej długości** z rejestrów xmmi i xmm2/m128, na wyjściu tworzy maskę, wynik zapisuje w rejestrze **xmmo** (nie jest umieszczany w definicji).

Operand 1	Operand 2	Wynik
xmmi	xmm2/m128	xmmo

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

200

## Operacje przesunięć AVX liczby 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 2022

Programowanie niekompromisowe

201

## Instrukcje przesunięć bitowych

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

`vpsrl[w/d/q] xmmi, xmm2, xmm3/m128 lub imm8  
vpsrl[w/d/q] ymmi, ymm2, ymm3/m256 lub imm8 (AVX2)`

Przesuwa **słowa/podwójne słowa/poczwórne słowa w prawo logicznie (cale)** 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 słów, 31 dla podwójnych słów, 63 dla poczwórnego słów, wówczas wszystkie bity są zerowane.

`cel[i] = źródło[i] >> źródło2[i] lub imm8  
xmmi[i] = xmm2[i] >> xmm3/m128[i] lub imm8  
ymmi[i] = ymm2[i] >> ymm3/m256[i] lub imm8`

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

053

## Instrukcja przesunięcia w prawo VPSRLV[D/Q]

`vpsrlv[d/q] xmmi, xmm2, xmm3/m128 lub imm8  
vpsrlv[d/q] ymmi, ymm2, ymm3/m256 lub imm8 (AVX2)`

Przesuwa **logicznie w prawo** bity podwójne słowa/poczwórne słowa z rejestru `xmm2/ymm2` o liczbie **bitów** 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 poczwórnego słów, wówczas wszystkie bity są zerowane.

`cel[i] = źródło[i] >> źródło2[i]  
xmmi[i] = xmm2[i] >> xmm3/m128[i]  
ymmi[i] = ymm2[i] >> ymm3/m256[i]`

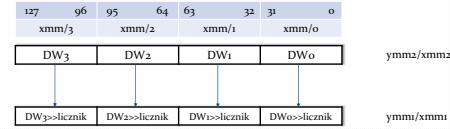
Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

054

## Instrukcja przesunięcia w prawo VPSRLD



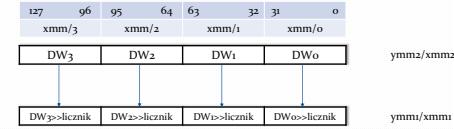
**Licznik** może być zapisany w rejestrze **xmm3/m128** (`ymm3/m256`) albo w bajcie sterującym **imm8**

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

055

## Instrukcja przesunięcia w prawo VPSRLD



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

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

056

## Instrukcja przesunięcia logicznego w prawo VPSRLDQ

`vpsrldq xmmi, xmm2, imm8  
vpsrldq ymmi, ymm2, imm8 (AVX2)`

Przesuwa **logicznie w prawo** podwójne poczwórne słowo (xmm) z rejestru `xmm2/ymm2` o **liczbę bajtów**, określzoną przez rejestr `xmm3/ymm3` lub `m128/m256`. Podczas przesunięcia **starsze bity są zerowane**.

`cel[i] = źródło[i] >> imm8  
xmmi[i] = xmm2[i] >> imm8  
ymmi[i] = ymm2[i] >> imm8`

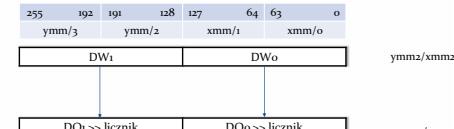
Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

057

## Instrukcja dodawania VPSRDQ



**Licznik** jest określony w **imm8**.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

058

## Instrukcja przesunięcia arytmetycznego w prawo VPSRA[W/D/Q]

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

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

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

`cel[i] = źródło[i] >> źródło2[i] lub źródło2`

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

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

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

079

## Instrukcja przesunięcia arytmetycznego w prawo VPSRAV[W/D /Q]

`vpsrav[w/d/q] xmm1, xmm2, xmm3/m128`

`vpsrav[w/d/q] ymm1, ymm2, ymm3/m256 (AVX2)`

Przesuwa arytmetyczne w prawo z powielaniem bitu znaku słowa/podwójne słowa/poczwórne słowa z rejestru xmm2/ymm2 o liczbę bitów określonych przez wartość zapisaną w rejestrze xmm3/ymm3 lub m128/m256. Starsze bity są ustawiane na bit znaku. Jeśli wartość licznika jest większa niż 15 dla słów, 31 dla podwójnych słów, 63 dla poczwórnich słów, wówczas wszystkie bity są ustawiane na bit znaku.

`cel[i] = źródło[i] >> źródło2[i]`

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

`ymmi[i] = ymm2[i] >> ymm3/m256[i]`

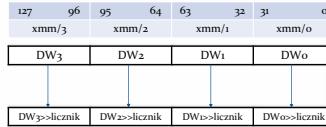
Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

079

## Instrukcja przesunięcia arytmetycznego w prawo VPSRAD



ymm2/xmm2

ymmi/xmmi

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

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

079

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

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

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

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

Przesuwa logicznie w lewo słowo/podwójne słowo/poczwórne słowo (w całości) z rejestru xmm2/ymm2 o wartość określzoną 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 poczwórnich słów, wówczas wszystkie bity danego elementu są zerowane.

`cel[i] = źródło[i] << źródło2[i] lub imm8`

`xmm1[i] = xmm2[i] << xmm3/m128[i] lub imm8`

`ymmi[i] = ymm2[i] << ymm3/m256[i] lub imm8`

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

079

## Instrukcja przesunięcia logicznego w lewo VPSLDD

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

`vpslly[w/d/q] xmm1, xmm2, xmm3/m128`

`vpslly[w/d/q] ymm1, ymm2, ymm3/m256`

Przesuwa logicznie w lewo słowo/podwójne słowo/poczwórne słowo z rejestru xmm2/ymm2 o liczbę bitów określzoną przez rejestr xmm3/ymm3 lub m128/m256. Młodsze bity są zerowane. Jeśli wartość licznika jest większa niż 15 dla słów, większa niż 31 dla podwójnych słów, większa niż 63 dla poczwórnich słów, wówczas wszystkie bity danego elementu są zerowane.

`cel[i] = źródło[i] << źródło2[i]`

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

`ymmi[i] = ymm2[i] << ymm3/m256[i]`

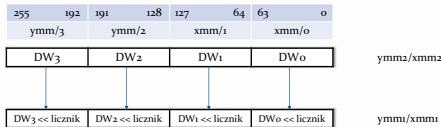
Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

079

## Instrukcja przesunięcia arytmetycznego w lewo VPSLLVQ



Licznik jest określony w **xmm3/ymm3 lub m128/m256**

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

175

## Operacje logiczne / inne AVX liczby całkowite

- Instrukcje logiczne:  
VPAND, VPANDN, VPOR, VPXOR

- Instrukcje zerowania:  
VZEROALL, VZEROUPPER

- Instrukcje dodatkowe:  
VLDMXCSR / VSTMXCSR

- Instrukcja wyrównywania:  
VPALIGNR

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

176

## Instrukcje logiczne

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

177

## Instrukcje logiczne koniunkcja VPAND / VPANDN

vpand x<sub>mm1</sub>, x<sub>mm2</sub>, x<sub>mm3/m128</sub>  
vpand y<sub>mm1</sub>, y<sub>mm2</sub>, y<sub>mm3/m256</sub> (AVX<sub>2</sub>)

Oblicza iloczyn logiczny bit po bicie dla wszystkich bitów rejestrów x<sub>mm2/ymm2</sub> oraz x<sub>mm3/ymm3</sub> lub m<sub>128/m256</sub>, wynik zapisuje w x<sub>mm1/ymm1</sub>.

cel[i] = źródło1[i] and źródło2[i]

vpandn x<sub>mm1</sub>, x<sub>mm2</sub>, x<sub>mm3/m128</sub> (AVX)  
vpandn y<sub>mm1</sub>, y<sub>mm2</sub>, y<sub>mm3/m256</sub> (AVX<sub>2</sub>)

Oblicza iloczyn logiczny bit po bicie operendu x<sub>mm3/ymm3</sub> lub m<sub>128/m256</sub> oraz negacji operendu x<sub>mm2/ymm2</sub>, wynik zapisuje w x<sub>mm1/ymm1</sub>.

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

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

178

## Instrukcje logiczne alternatywa VPOR / VPXOR

vpor x<sub>mm1</sub>, x<sub>mm2</sub>, x<sub>mm3/m128</sub>  
vpor y<sub>mm1</sub>, y<sub>mm2</sub>, y<sub>mm3/m256</sub> (AVX<sub>2</sub>)

vxpor x<sub>mm1</sub>, x<sub>mm2</sub>, x<sub>mm3/m128</sub> (AVX)  
vxpor y<sub>mm1</sub>, y<sub>mm2</sub>, y<sub>mm3/m256</sub> (AVX<sub>2</sub>)

Oblicza sumę logiczną bit po bicie dla wszystkich bitów rejestrów x<sub>mm2/ymm2</sub> oraz x<sub>mm3/ymm3</sub> lub m<sub>128/m256</sub>, wynik zapisuje w x<sub>mm1/ymm1</sub>.

cel[i] = źródło1[i] or źródło2[i]

cel[i] = źródło1[i] xor źródło2[i]

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

179

## Instrukcje dodatkowe

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

180

## Instrukcje dodatkowe VZEROALL / VZEROUNDER

### vzeroall (AVX)

Zeruje bity wszystkie rejestrów ymm/o - ymm<sub>15</sub>/o

### vzeroupper (AVX)

Zeruje bity od 128 do ostatniego rejestrów ymm/o - ymm<sub>15</sub>/zmm/o - zmm<sub>15</sub>/z

W trybie 32 bitowym zeruje tylko pierwsze 8 rejestrów.

Bitы от 128/256 до MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

86

## Instrukcje dodatkowe VLDMXCSR / VSTMXCSR

### vldmxcsr m32 (AVX)

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

### vstmxcsr m32 (AVX)

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

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

Programowanie niekropozitowne

87

## Instrukcja łączenia dodatkowe VPALIGNR

### vpaligr xmm1, xmm2, xmm3/m128, imm8 (AVX)

### vpaligr ymm1, ymm2, ymm3/m256, imm8 (AVX2)

Łączy (konkatenacją) rejesty źródła xmm2/ymm2 i xmm3/ymm3 lub m128/m256, na podstawie bajtu sterującego przesunięcia 128 bitowe części i zapisuje młodsze 128 części do rejestrów 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

Bitы от 128/256 do MSB są zerowane.

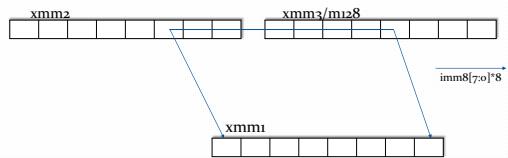
(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

88

## Instrukcja łączenia dodatkowe VPALIGNR

### Sposób łączenia rejestrów xmm



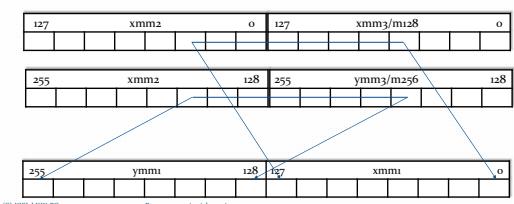
(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

89

## Instrukcja łączenia dodatkowe VPALIGNR

### Sposób łączenia rejestrów xmm



89

## Instrukcje szyfrujące

## Operacje szyfrujące AVX

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

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

87

## 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 2022

Programowanie niskopoziomowe

88

## 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 2022

Programowanie niskopoziomowe

89

## Instrukcja szyfrująca AES (Advanced Encryption Standard)

### VAESENC

```
vaesenc xmm1, xmm1, xmm3/m128
```

Szyfruje jedną rundę (jednokrotne) dane (blok danych) całego rejestru xmm2 (128 bitów) z wykorzystaniem 128 bitowego klucza zapinanego 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 2022

Programowanie niskopoziomowe

90

## Instrukcja szyfrująca AES (Advanced Encryption Standard)

### VAESENCLAST

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

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

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

Bity od 128/196 do MSB nie są modyfikowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

91

## Instrukcja szyfrująca AES (Advanced Encryption Standard)

### VAESDEC

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

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

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

Bity od 128/196 do MSB nie są modyfikowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

92

## Instrukcja szyfrująca AES (Advanced Encryption Standard)

### VAESDECLAST

vaesdeclast xmm1, xmm2, xmm3/m128

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

cel = aes(źródło1) xor źródło2 (key)

xmm1 = aes(xmm2) xor xmm3/m128

Bity od 128/256 do MSB nie są modyfikowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

193

## 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)

Bity od 128/256 do MSB nie są modyfikowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

194

## 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

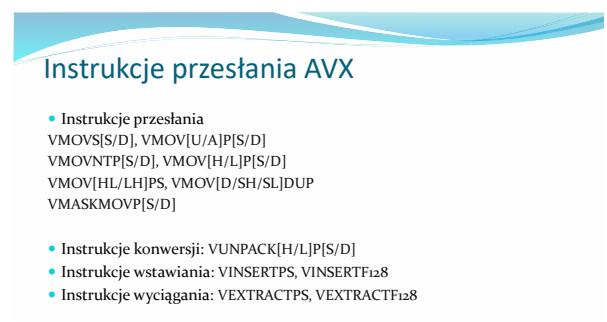
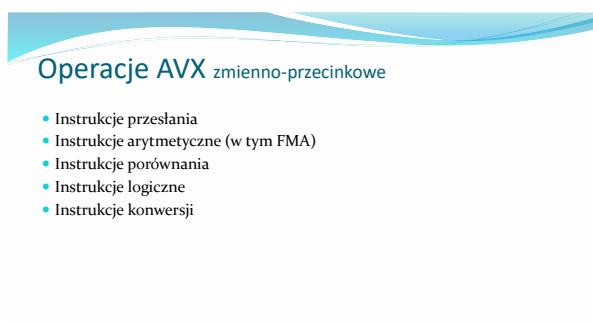
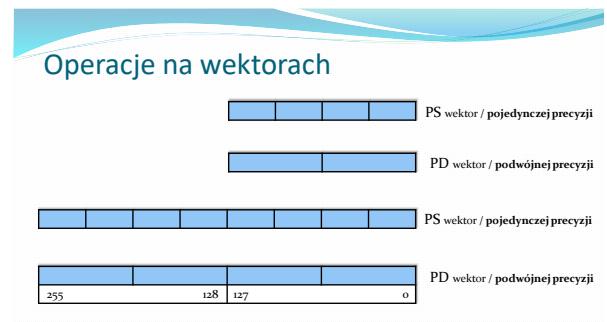
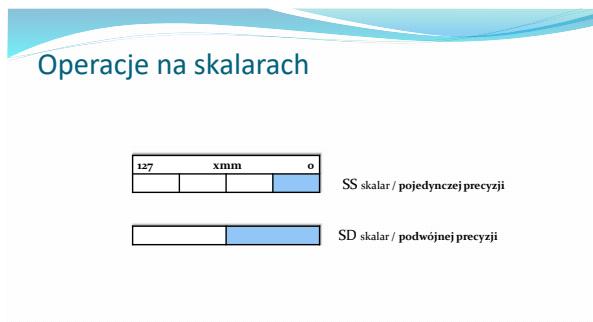
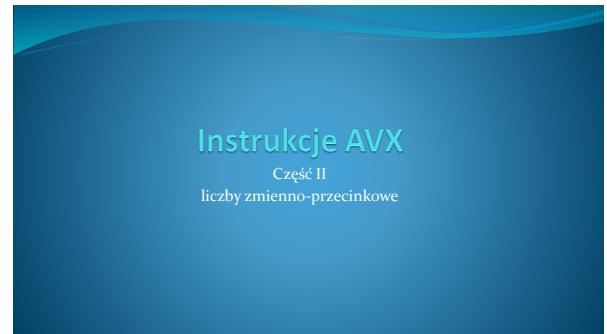
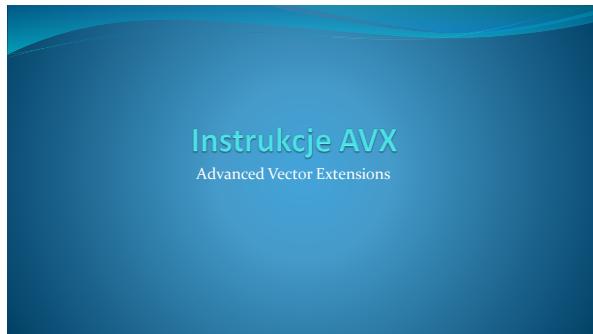
xmm1 = szyfrowanie(xmm2/m128), imm8

Bity od 128/256 do MSB nie są modyfikowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

195





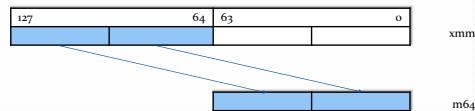
## Instrukcja przesłania

### VMOVHPS

2. vmovhps m64, xmm1

Przesyła dwie wartości rzeczywiste pojedynczej precyzji ze starszej polowy rejestru xmm1 do pamięci m64.

$m64 \leftarrow xmm1[127:64]$



(C) KISI d.KIK PCz 2022

Programowanie niekropotomowe

13

## Instrukcja przesłania

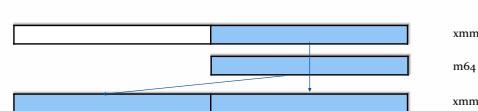
### VMOVHPD

1. vmovehdp xm1, xm2, m64

Kopiuje wartości liczb rzeczywistych podwójnej precyzji z młodszej polowy rejestru xm1 oraz z

pamięci m64, wynik zapisuje w xm2.

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



(C) KISI d.KIK PCz 2022

Programowanie niekropotomowe

14

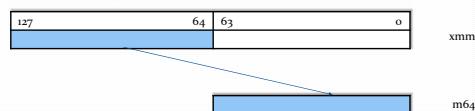
## Instrukcja przesłania

### VMOVHPD

2. vmovehdp m64, xm1

Kopiuje liczbę rzeczywistą podwójnej precyzji ze starszej polowy rejestru xm1 do pamięci m64.

$m64 \leftarrow xmm1[127:64]$



(C) KISI d.KIK PCz 2022

Programowanie niekropotomowe

15

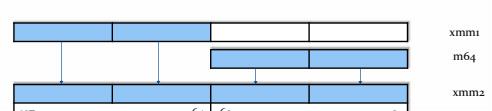
## Instrukcja przesłania

### VMOVLPS

1. vmoveplps xm1, xm2, m64

Przepisuje po dwie wartości rzeczywiste podwójnej precyzji ze starszej polowy rejestru xm1 do xmmin2 oraz pamięci m64, wynik zapisuje w xm2.

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



(C) KISI d.KIK PCz 2022

Programowanie niekropotomowe

16

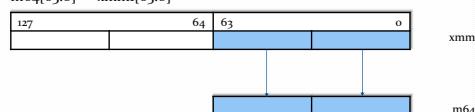
## Instrukcja przesłania

### VMOVLPS

2. vmoveplps m64, xm1

Przesyła dwie wartości liczb rzeczywistych pojedynczej precyzji z młodszej polowy rejestru xm1 do pamięci m64.

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



(C) KISI d.KIK PCz 2022

Programowanie niekropotomowe

17

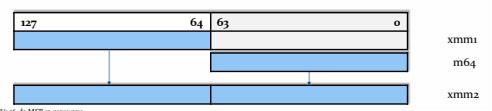
## Instrukcja przesłania

### VMOVLPD

1. vmoveplpd xm1, xm2, m64

Przesyła liczbę rzeczywistą podwójnej precyzji z pamięci m64 oraz starszą polową rejestru xm1, wynik zapisuje w xm2.

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



(C) KISI d.KIK PCz 2022

Programowanie niekropotomowe

18

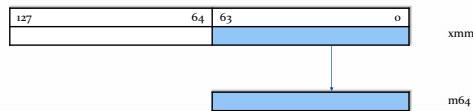
## Instrukcja przesłania

### VMOVLPD

2. vmovlpd m64, xmmi

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

$m64[63:0] \leftarrow xmmi[63:0]$



Bity od 128:96 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

19

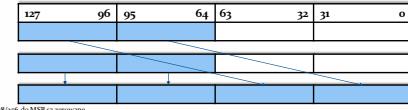
## Instrukcja przesłania

### VMOVHLPS

vmovhlps xmmi, xmm2, xmmj

Przesyła ze starszej połowy rejestru xmm3/mu28 do młodszej połowy xmmi oraz z starszej połowy rejestru xmm2 do starszej połowy rejestru celu po dwie liczby rzeczywiste pojedynczej precyzji, wynik zapisuje w xmmi.

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



Bity od 128:96 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

20

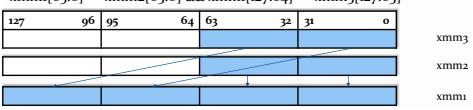
## Instrukcja przesłania

### VMOVLHPS

vmovlhps xmni, xmm2, xmni

Przesyła młodszą połowę rejestru xmm2 do młodszej połowy rejestru celu oraz ze młodszą połowy rejestru xmni/mu28 do starszej połowy rejestru celu po dwie liczby rzeczywiste pojedynczej precyzji , wynik zapisuje w xmni.

$xmni[63:0] \leftarrow xmm2[63:0] \&& xmni[127:64] \leftarrow xmni[127:63]$



Bity od 128:96 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

21

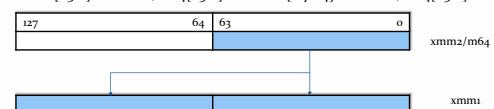
## Instrukcja przesłania

### VMOVDUP

vmovddup xmni, xmm2/m64

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

$xmni[63:0] \leftarrow xmni[63:0] \&& xmni[127:64] \leftarrow xmni[127:64]$



Bity od 128:96 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

22

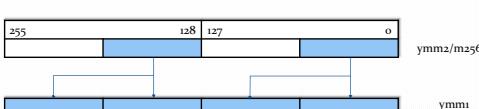
## Instrukcja przesłania

### VMOVDDUP

vmovddup ymmi, ymm2/m256

Kopiuje liczbę rzeczywistą podwójnej precyzji o parzystych indeksach z rejestru ymm2 lub pamięci m256 do ymmi.

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



Bity od 128:96 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

23

## Instrukcja przesłania

### VMOVSHDUP

vmovshdup xmni, xmm2/m128

vmovshdup ymmi, ymm2/m256

Kopiuje z powielaniem wartości liczb rzeczywistych pojedynczej precyzji o nieparzystych indeksach ymm2/ymmi lub ymm2/m256 i zapisuje do xmni/ymmi.

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



Bity od 128:96 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

24

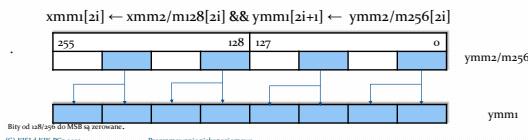
## Instrukcja przesłania

### VMOVSLDUP

vmovsldup xmm1, xmm2/m128

vmovsldup ymm1, ymm2/m256

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



(C) KISI d.KIK PCz 2022

Programowanie niekropozitowe

25

## 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[adres+i\*4]

else xmm1[i] = 0

if ymm2[i][31] then ymm1[i] ← m256[adres+i\*4]

else ymm1[i] = 0

Bity od 128/256 do MSB są zerowane.

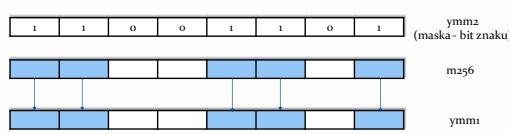
(C) KISI d.KIK PCz 2022

Programowanie niekropozitowe

26

## Instrukcja przesłania warunkowego

### VMASKMOVPS (2/4)



Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowe

27

## Instrukcja przesłania warunkowego

### VMASKMOVPS (3/4)

vmaskmovps m128, xmm1, xmm2

vmaskmovps m256, ymm1, ymm2

Przesyła liczby rzeczywiste pojedynczej precyzji z rejestru xmm1/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[adres+i\*4] ← xmm2[i]

if ymm1[i][31] then m256[adres+i\*4] ← ymm2[i]

Bity od 128/256 do MSB są zerowane.

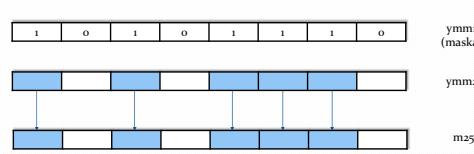
(C) KISI d.KIK PCz 2022

Programowanie niekropozitowe

28

## Instrukcja przesłania warunkowego

### VMASKMOVPS (4/4)



Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowe

29

## 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[adres+i\*8]  
else xmm1[i] = 0

if ymm2[i][63] then ymm1[i] ← m256[adres+i\*8]  
else ymm1[i] = 0

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowe

30

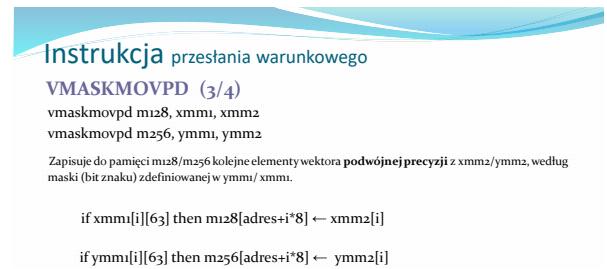


Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

31

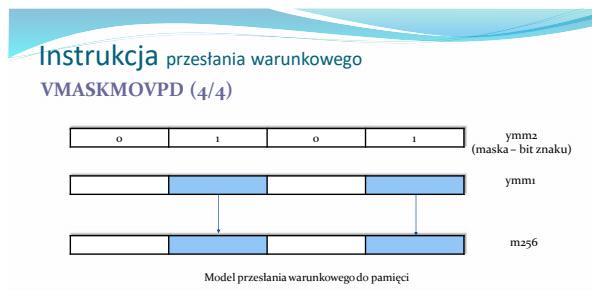


Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

32

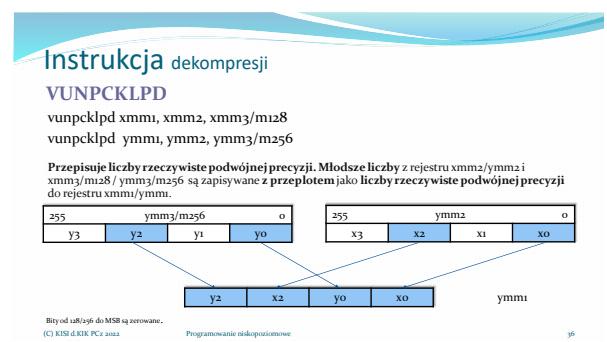
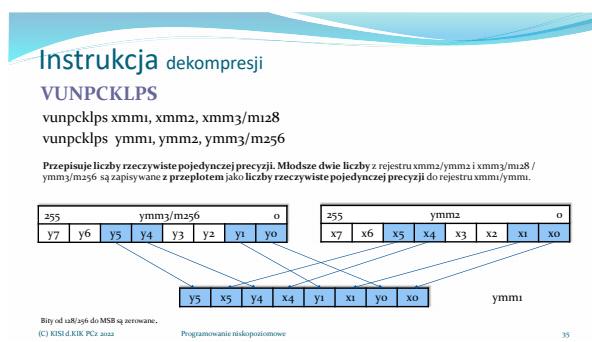


Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekropozitowne

33

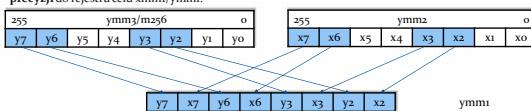


## Instrukcja dekompresji

### VUNPCKHPS

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

Przepisuje liczby rzeczywiste pojedynczej precyzyji. Starsze dwie liczby z rejestru xmm2/ymm2 i xmm3/m128 / ymm3/m256 są zapisywane z przepłotem jako liczby rzeczywiste pojedynczej precyzyji do rejestru celu xmm1/ymm1.



Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekopiowalne

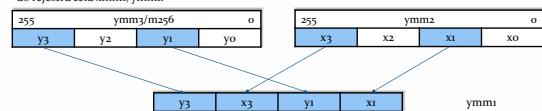
37

## Instrukcja dekompresji

### VUNPCKHPD

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

Przepisuje liczby rzeczywiste podwójnej precyzyji. Starsze liczby z rejestru xmm2/ymm2 i xmm3/m128 / ymm3/m256 są zapisywane z przepłotem jako liczby rzeczywiste podwójnej precyzyji do rejestru celu xmm1/ymm1.



Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekopiowalne

38

## Instrukcje wstawiania

## Instrukcja wstawiania

### VINSERTPS (1/2)

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

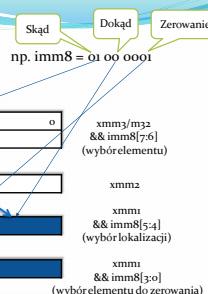
(C) KISI d.KIK PCz 2022

Programowanie niekopiowalne

40

## Instrukcja wstawiania

### VINSERTPS (3/3)



Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekopiowalne

41

## 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].

```

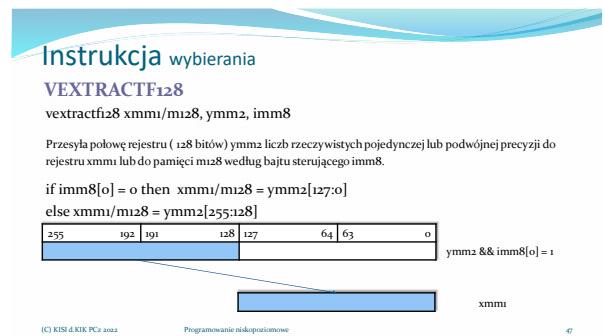
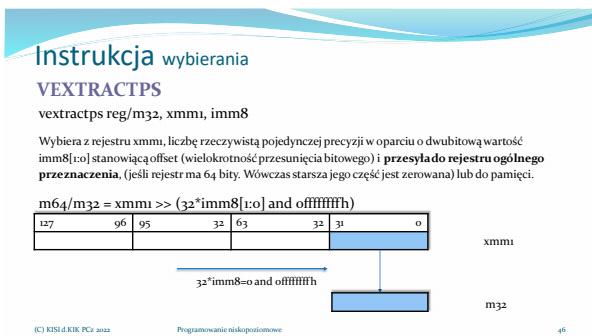
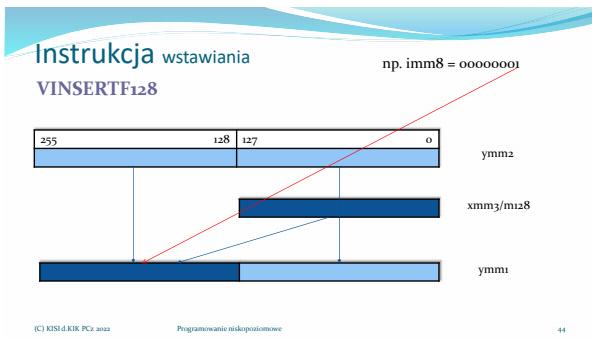
if imm8[0] == 0 then
    ymm1[i] = ymm2[i]
    lo ymm1[i] = xmm3/m128[i]
else
    ymm1[i] = ymm2[i]
    hi ymm1[i] = xmm3/m128[i]

```

(C) KISI d.KIK PCz 2022

Programowanie niekopiowalne

43



## 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 pełni zadanie przeróżnika.

i <0, > dla PS lub i <0, > 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.

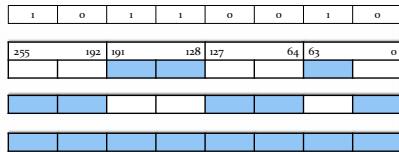
(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

50

## Instrukcja mieszająca

### VBLENDPS



imm8

ymm3/m256

ymm2

xmm1

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

51

## 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.

(C) KISI d.KIK PCz 2022

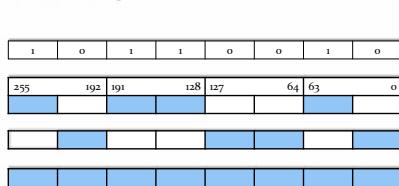
Programowanie niekompromisowe

52

## Instrukcja mieszająca

### VBLENDVP[S]

Maska w rejestrze ymm4



ymm4  
(bit znaku)

ymm3/m256

ymm2

ymm1

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

53

## Instrukcje rozgłaszania

## Instrukcja rozgłaszania

### VBROADCASTS[S/D] / VBROADCASTF128

vbroadcastss xmm1, m32/xmm2

vbroadcastst ss 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.

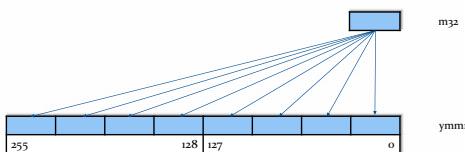
Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

55

## Instrukcja rozgłaszania VBROADCASTSS



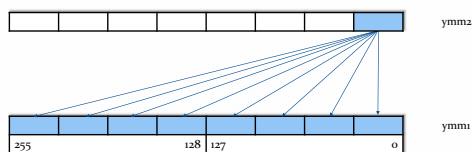
Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie indeksopozycyjne

56

## Instrukcja rozgłaszania VBROADCASTSS



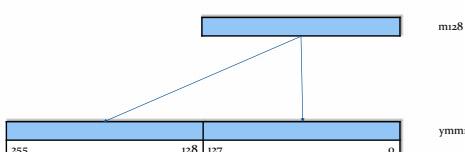
Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie indeksopozycyjne

57

## Instrukcja rozgłaszania VBROADCASTF128



Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie indeksopozycyjne

58

## Instrukcje zbierania

### Instrukcja zbierania VGATHER[D/Q]P[S/D]

Dotyczy typów danych

Dotyczy adresów

Instrukcja kompletuje wektor xmni/ymmi używając **adresów w postaci podwójnych/poczwórnich** słów zdefiniowanych w  $\text{vm32[x/y]/vm64[x/y]}$  używając jako indeksów podwójnych/poczwórnich słów zapisanych w  $\text{xmna/ymma}$  do wskazanej lokalizacji pamięci, skład pobierane są liczby rzeczywiste pojedynczej/podwójnej precyzji.

Pobierane z pamięci wartości są zapisywane do rejestru celu xmni/ymmi tylko wówczas gdy najstarsze bity odpowiednich elementów wektora maski xmnm3/ymmm3 są równe 1.

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie indeksopozycyjne

60

### Instrukcja zbierania VGATHER[D/Q]P[S/D]

$\text{vgather}[d/q/p/s/d] \text{ xmni, vm}[32/64]x, \text{xmm3 (AVX2)}$   
 $\text{vgather}[d/q/p/s/d] \text{ ymmi, vm}[32/64]y, \text{ymm3 (AVX2)}$

**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 xmnm3/ymmm3 (z xmnm3/ymmm3 używane są jedynie indeksy)

**skalowanie** - określa rozmiar danych (1, 2, 4, 8)

**przesunięcie** - wartość w bajtach

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie indeksopozycyjne

61

## Instrukcja zbierania

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

`vgather[d/q/p/s/d] xmmi, vm[32/64]x, xmm3 (AVX2)`  
`vgather[d/q/p/s/d] ymmi, 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.

Bit od 128/256 do MSB są zerowane.

Programowanie niekopiowalne

62

## Instrukcja zbierania

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

`vgather[d/q/p/s/d] xmmi, vm[32/64]x, xmm3 (AVX2)`  
`vgather[d/q/p/s/d] ymmi, 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 ją do rejestru celu ymmi/vmmi tylko wówczas gdy bit znaku odpowiadającego elementowi maski ymm3/vmm3 jest równy jeden, jeśli bit znaku jest równy zero w rejestrze celu wartość zostaje poprzednia. Po wykonaniu operacji pobierania z pamięci elementy maski są zerowane.

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

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

Bit od 128/256 do MSB są zerowane.

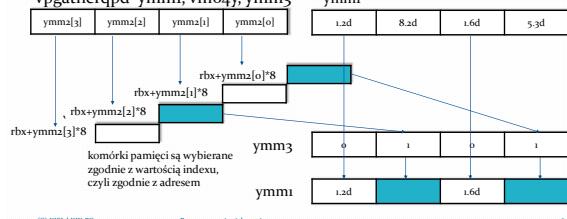
Programowanie niekopiowalne

63

## Instrukcja zbierania (przykład) AVX2 VPGATHERQPD

`vpgatherqd ymm1, [rbx+ymm2*8], ymm3      mv64y = [rbx+ymm2*8]`

`vpgatherqd ymm1, vm64y, ymm3`



(C) KISI d.KIK PCz 2022

Programowanie niekopiowalne

64

## 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 offset, przesunięcie), wynik zapisuje w ymm1.

`ymmi[i] = ymm3/m256 >> ymm2[i][2:0]*32`

`vpermpd ymm1, ymm2/m256, imm8`

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

`ymmi[i] = ymm2/m256 >> imm8[2i+1:2i]*64`

Bit od 128/256 do MSB są zerowane.

Programowanie niekopiowalne

66

## Instrukcja permutacji

### VPERMPS

255	128	127	0
10b	ooob	oiob	ooob

ymmm2

H	G	F	E	D	C	B	A
---	---	---	---	---	---	---	---

ymmm3/m256

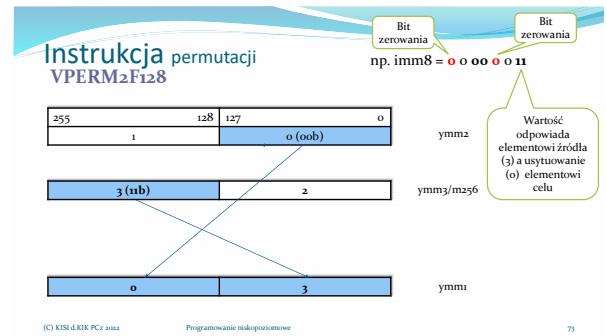
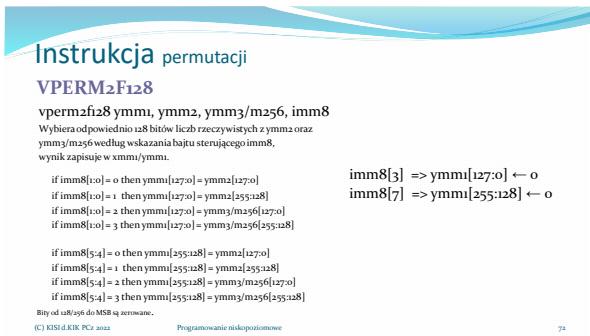
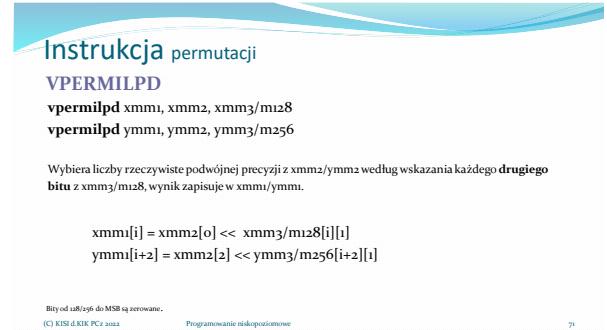
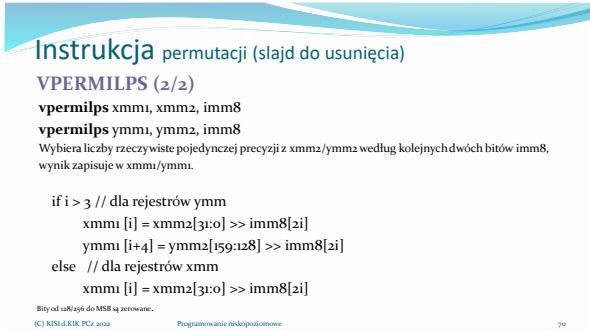
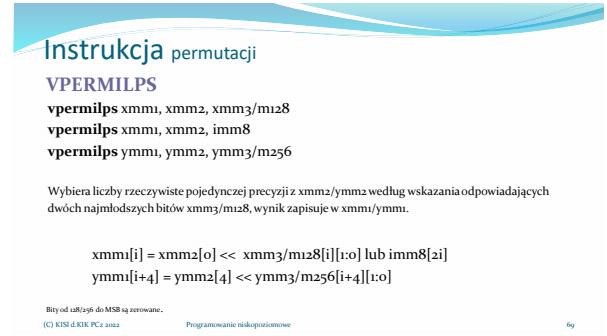
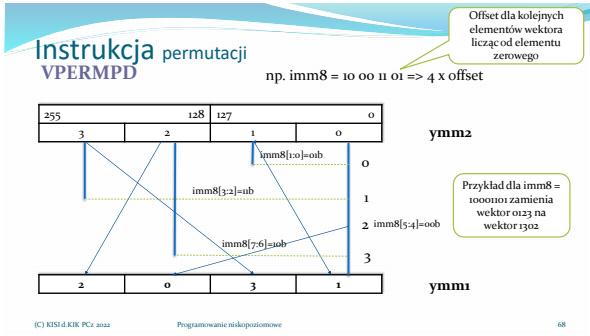
F	A	C	B	A	H	G	A
---	---	---	---	---	---	---	---

ymmi

Bit od 128/256 do MSB są zerowane.

Programowanie niekopiowalne

67



## Instrukcje tasowania

## Instrukcja tasowania

VSHUFPS

vshufps xmm1, xmm2, xmm3/m256, imm8  
vshufps ymm1, ymm2, ymm3/m256, imm8

Wybiera liczby rzeczywiste pojedynczej precyzyji z xmm2/ymm2 oraz xmm3/ymm3 lub m128/m256 po dwie z każdego źródła według bajtu sterującego imm8 i zapisuje w xmml/ymml po dwie wartości przepłatając źródła pochodzenia.

```

if i=(0..1,4..5) then s=2 else s=3
xmm[i] = xmms[imm8[2*i+1:2*i]]                                i=(0..3)
ymm[i] = ymms[imm8[2*(i-4)+1:2*(i-4)]]                         i=(4..7)

```

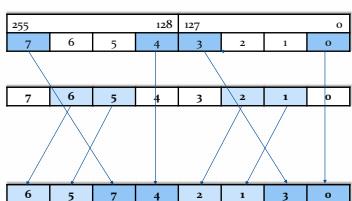
Bity od 128/256 do MSB są zerowane.

Programowanie niskopoziomowe

5

# Instrukcja tasowania VSHUFPS

pp. imm8 = 10.81.11.89



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

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

(C) KISIL KIK PGs 2001

[Programación dinámica](#)

## Instrukcja tasowania

VSHUFDPD

vshufpd ymm1, ymm2, ymm3/m256, imm8

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

```

xmmi[0] = xmm2[imm8[0]]
xmmi[1] = xmm3[imm8[1]]
ymmi[2] = ymm2[2+imm8[2]
ymmi[3] = ymm3[2+imm8[3]

```

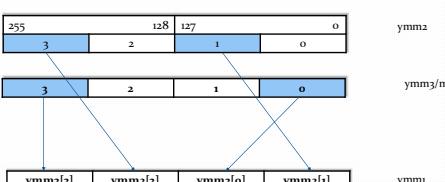
Bit y od w8/zs6 do MSB sa zerowan

Rzeczywistość niskopaliomowa

3

# Instrukcja tasowania VSHUERD

pp\_jmm8 - 00000112



## 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]

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

80

## Instrukcje dodawania

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

vaddss[s/d] xmms1, xmms2, xmms3/m32/m64

vaddp[s/d] xmms1, xmms2, xmms3/m128  
vaddp[s/d] ymm1, ymm2, ymm3/m256

Dodaje skalary/wektory liczb rzeczywistych pojedynczej/podwójnej precyzji z rejestru xmms2/ymm2 i xmms3/ymm3 lub m32/m64/m128/m256, wynik zapisuje w xmms1/ymm1.

$$\begin{aligned} \text{xmms1}[i] &= \text{xmms2}[i] + \text{xmms3}/\text{m128}[i] \\ \text{ymm1}[i] &= \text{ymm2}[i] + \text{ymm3}/\text{m256}[i] \end{aligned}$$

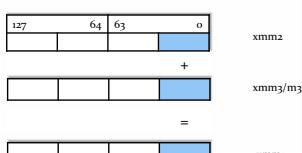
Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

81

### Instrukcja dodawania VADDSS

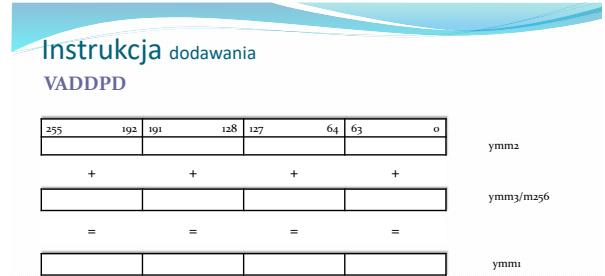


Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

84



Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

83

### Instrukcja dodawania VHADDP

vhaddps xmms1, xmms2, xmms3/m128

vhaddps ymm1, ymm2, ymm3/m256

Horyzontalne dodawanie sąsiednich liczb rzeczywistych pojedynczej precyzji i zapisywanie wyniku z przepłotem.

$$\begin{aligned} \text{ymms1}/\text{xmms1}[3:0] &= \text{ymms2}/\text{xmms2}[63:32] + \text{ymms2}/\text{xmms2}[3:0] \\ \text{ymms1}/\text{xmms1}[63:32] &= \text{ymms2}/\text{xmms2}[127:96] + \text{ymms2}/\text{xmms2}[95:64] \\ \text{ymms1}/\text{xmms1}[95:64] &= \text{ymms3}/\text{xmms3}[63:32] + \text{ymms3}/\text{xmms3}[3:0] \\ \text{ymms1}/\text{xmms1}[127:96] &= \text{ymms3}/\text{xmms3}[127:96] + \text{ymms3}/\text{xmms3}[95:64] \\ \text{ymms1}[159:128] &= \text{ymms2}[191:160] + \text{ymms2}[159:128] \\ \text{ymms1}[191:160] &= \text{ymms2}[255:224] + \text{ymms2}[223:192] \\ \text{ymms1}[223:192] &= \text{ymms3}[191:160] + \text{ymms3}[159:128] \\ \text{ymms1}[255:224] &= \text{ymms3}[255:224] + \text{ymms3}[223:192] \end{aligned}$$

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

85

## Instrukcja dodawania

### VHADDPD

vhaddpd xmm1, xmm2, xmm3/m128

vhaddpd ymm1, ymm2, ymm3/m256

Horyzontalne dodawanie sąsiednich liczb rzeczywistych podwójnej precyzji i zapisywanie wyniku z przepłotem.

$$\text{xmmi}[o] = \text{xmm2}[1] + \text{xmm2}[o]$$

$$\text{xmmi}[1] = \text{xmm3}[1] + \text{xmm3}[o]$$

$$\text{ymmi}[2] = \text{ymm2}[3] + \text{ymm2}[2]$$

$$\text{ymmi}[3] = \text{ymm3}[3] + \text{ymm3}[2]$$

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

86

## Instrukcje odejmowania

## Instrukcja odejmowania

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

vsub[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 **odejmuję** liczby rzeczywiste pojedynczej/podwójnej precyzji odpowiednio z xmm3/ymm3 lub m32/m64/m128/m256, wynik zapisuje w xmmi/ymmi.

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

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

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

88

## Instrukcja odejmowania

### VSUBPS

255	192	191	128	127	64	63	0
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
=	=	=	=	=	=	=	=

yymm2/xmm2

yymm3/xmm3  
m256/m128

ymmi/xmmi

Bit od 128/256 do MSB są zerowane.

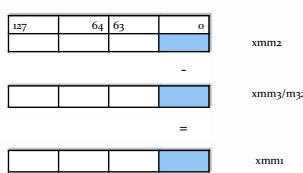
(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

89

## Instrukcja odejmowania

### VSUBSS



Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

90

## Instrukcja odejmowania horyzontalnego

### VHSUBP[S/D]

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

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

Horyzontalne odejmowanie sąsiednich liczb rzeczywistych pojedynczej/podwójnej precyzji i zapisywanie wyniku z przepłotem.

$$\text{xmmi}[i+1] = \text{xmm2}[2i] - \text{xmm2}[2i+1]$$

$$\text{xmmi}[i] = \text{xmm3}[2i] - \text{xmm3}[2i+1]$$

$$\text{ymmi}[i+1+2] = \text{ymm2}[2i+2] - \text{ymm2}[2i+1+2]$$

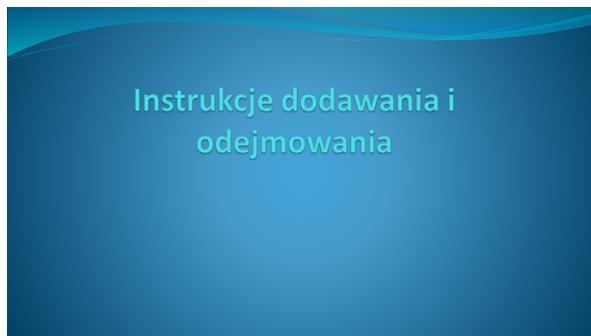
$$\text{ymmi}[i+2] = \text{ymm3}[2i+2] - \text{ymm3}[2i+1+2]$$

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

91



**Instrukcja dodawania i odejmowania**

**VADDSSUBP[S/D]**

vaddsubp[s/d] xmml, xmml, xmml3/m128  
vaddsubp[s/d] ymm1, ymm1, ymm3/m256

Naprzemienne odejmuję i dodaje wektory liczb rzeczywistych pojedynczej/podwójnej precyzji od/do zawartości rejestru xmml/ymml **odejmuję / dodaje** odpowiadającą wartości xmml3/ymml3 lub m128/m256, wynik zapisuje w xmml/ymml.

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

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      93

**Instrukcja dodawania i odejmowania**

**VADDSSUBPD**

255	192	191	128	127	64	63	0
+	-		+	-			
=	=	=	=	=			

ymml2/xmml2  
ymml3/xmml3  
m256/m128  
ymml/xmml

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      94



**Instrukcja mnożenia**

**VMULPS**

vmulps[s/d] xmml, xmml, xmml3/m32/m64  
vmulps[s/d] xmml, xmml, xmml3/m128  
vmulps[s/d] ymm1, ymm1, ymm3/m256

Mnoży liczby rzeczywiste pojedynczej/podwójnej precyzji z rejestru xmml/ymml odpowiednio przez xmml3/ymml3 lub m32/m64/m128/m256, wynik zapisuje w xmml/ymml.

xmml[i] = xmml2[i] \* xmml3/m128[i]  
ymml[i] = ymm2[i] \* ymm3/m256[i]

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      96

**Instrukcja mnożenia**

**VMULPS**

255	192	191	128	127	64	63	0
*	*	*	*	*	*	*	*
=	=	=	=	=	=	=	=

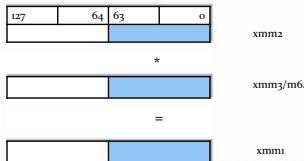
ymml2/xmml2  
ymml3/xmml3  
m256/m128  
ymml/xmml

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022      Programowanie niekompromisowe      97

## Instrukcja mnożenia VMULSD

VMULSD



98

## Instrukcja iloczyn skalarny VDPPS

vdpps xmmi, xmm2, xmm3/m128, imm8  
vdpps ymmi, ymm2, ymm3/m256, imm8  
Oblicza iloczyn skalarny wektorów mnożąc warunkowo elementy rejestru xmmi przez xmm2, a następnie warunkowo (zależnie od ustawień imm8[3...0]), wynik zapisuje w xmmi.

```
is = 0
if imm8[i+4] then
    is += xmm2[i]*xmm3/m128[i]
    is2 += ymm2[i+4]*ymm3/m256[i+4]
```

```
if imm8[i]
    xmmi[i] = is
```

```
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]
```

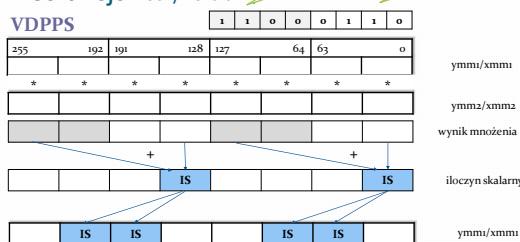
```
xmmi[i] = is1
ymmi[i+4] = is2
```

Bity od 128/256 do MSB są zerowane.

Programowanie niekompromisowe

99

## Instrukcja iloczyn skalarny VDPPS



100

## Instrukcja iloczyn skalarny VDPD

vdpd xmmi,xmm2, xmm3/m128, imm8 (tylko na xmm)  
Oblicza iloczyn skalarny wektorów mnożąc 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 xmmi.

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

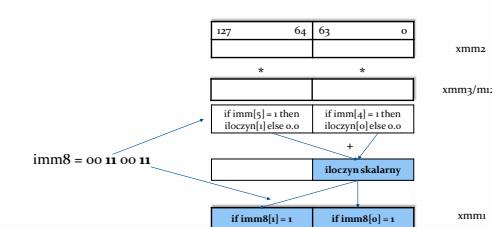
```
if imm8[i]=1 then
    xmmi[i] = is
```

Bity od 128/256 do MSB są zerowane.

Programowanie niekompromisowe

101

## Instrukcja iloczyn skalarny VDPPD (tylko dla xmm)



102

## Instrukcje dzielenia

## Instrukcja dzielenia

### VDIVP[S/D]

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

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

Dzieli liczby rzeczywiste pojedynczej/podwójnej precyzji z rejestru xmm2/ymm2 odpowiednio przez xmm3/ymm3 lub m32/m64/m128/m256, wynik zapisuje w xmm/ymm.

$$\text{xmm}_3[i] = \text{xmm}_2[i] / \text{xmm}_3/m128[i]$$

$$\text{ymm}_3[i] = \text{ymm}_2[i] / \text{ymm}_3/m256[i]$$

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022 Programowanie niekompromisowe

104

## Instrukcja dzielenia

### VDIVPD

255	192	191	128	127	64	63	0
/	/	/	/	/	/	/	/
=	=	=	=	=	=	=	=

ymm2/xmm2

ymm3/xmm3  
m256/m128

ymm1/xmm1

(C) KISI d.KIK PCz 2022 Programowanie niekompromisowe

105

## Instrukcja mnożenia

### VDIVPS

255	192	191	128	127	64	63	0
/	/	/	/	/	/	/	/
=	=	=	=	=	=	=	=

ymm2/xmm2

ymm3/xmm3  
m256/m128

ymm1/xmm1

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022 Programowanie niekompromisowe

106

## 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]

(C) KISI d.KIK PCz 2022 Programowanie niekompromisowe

107

## Instrukcja wartość maksymalna

### wartość maksymalna

### Instrukcja wartość maksymalna

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

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

vmaxp[s/d] ymm1, ymm2, ymm3/m128

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

Zapisuje wartość maksymalną z porównania liczb rzeczywistych pojedynczej/podwójnej precyzji z rejestrow xmm2/ymm2 i xmm3/ymm3 lub m32/m64/m128/m256, wynik zapisuje do xmm1/ymm1.

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]

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022 Programowanie niekompromisowe

109

## Instrukcja wartość maksymalna

**VMAXPD**

255	192	191	128	127	64	63	0
cmp	cmp	cmp	cmp				ymm2
=	=	=	=				ymm3/m256
max	max	max	max				ymmu

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

100

## Instrukcja wartość minimalna

127	64	63	0
			ymm2
=			
			ymm3/m256

## 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.

```
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.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

101

## Instrukcja wartość minimalna

**VMINSS**

127	64	63	0
			xmm2
=			
			xmm3/m128

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

102

## Instrukcje wartość przybliżona

## Instrukcja zaokrąglenia

**VROUND[S/P][S/D]**

vroundss xmm1, xmm2, xmm3/m32, imm8  
 vroundsd xmm1, xmm2, xmm3/m64, imm8

Zaokrąga 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), ponadto bity od 33 są przepisywane z xmm2, sposób zaokrąglania jest zdeterminowany bajtem sterującym imm8.

```
vrounddp[s/d] xmm1, xmm2/m128, imm8
vrounddp[s/d] ymm1, ymm2/m256, imm8
```

Zaokrąga wektor liczb rzeczywistych pojedynczej/podwójnej precyzji z xmm2/ymm2 lub m128/m256 Do liczby całkowitej podwójnego/poczwórnego słowa, wynik zapisuje jako liczbę rzeczywistą xmm1/ymm1, zaokrąglenie odbywa się według bajtu sterującego imm8.

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

103

## Instrukcja zaokrąglenia

### VROUND[S/P][S/D]

vroundss xmmi, xmm2, xmm3/m32, imm8

vroundsd xmmi, xmm2, xmm3/m64, imm8

Zaokrąga najmłodszą liczbę rzeczywistą pojedynczej/podwójnej precyzji z xmm3 lub m32/m64 do wartości podwójnego/poczwórnego słowa (integer), wynik zapisuje w xmmi jako liczbę rzeczywistą pojedynczej precyzji (z przecinkiem zerami po nim), ponadto bity od 33 są przepisywane z xmm2, sposób zaokrąglenia jest zdeterminowany bajtem sterującym imm8.

vroundps xmmi, xmm2/m128, imm8

vroundpd ymmi, ymm2/m256, imm8

Zaokrąga wektor liczb rzeczywistych pojedynczej/podwójnej precyzji z xmm2/ymm2 lub m128/m256 do liczby całkowitej podwójnego/poczwórnego słowa, wynik zapisuje jako liczbę rzeczywistą w xmmi/ymmi, zaokrąglanie odbywa się według bajtu sterującego imm8.

Bit od 128/56 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekooperacyjne

16

Bit od 128/56 do MSB są zerowane.

(C) KISI d.KIK PCz 2022 Programowanie niekooperacyjne 17

Bit od 128/56 do MSB są zerowane.

(C) KISI d.KIK PCz 2022 Programowanie niekooperacyjne 17

## Instrukcja wartości odwrotności przybliżonej

### VRCP[SS/PS]

vrccpss xmmi, xmm2, xmm3/m32

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

(Relative Error  $\leq 1,5 \cdot 2^{-12}$ )

```
xmmi[31:0] ← 1.0/xmm3/m32[31:0]
xmmi[127:32] ← xmmi2[127:32]
```

vrccpps xmmi, xmm2/m128

vrccpps ymmi, ymm2/m256

Oblicza przybliżoną wartość odwrotności elementów wektora liczb rzeczywistych pojedynczej precyzji z xmm2/ymm2 lub m128/m256, wynik umieszcza w xmmi/ymmi. (Relative Error  $\leq 1,5 \cdot 2^{-12}$ )

```
xmmi[i] ← 1.0/xmm3/m32[i]
ymmi[i] ← 1.0/ymm3/m256[i]
```

Bit od 128/56 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekooperacyjne

18

Bit od 128/56 do MSB są zerowane.

(C) KISI d.KIK PCz 2022 Programowanie niekooperacyjne 19

Bit od 128/56 do MSB są zerowane.

(C) KISI d.KIK PCz 2022 Programowanie niekooperacyjne 19

## Instrukcje pierwiastkowania

## Instrukcja zaokrąglenia

### VROUND[S/P][S/D]

np. imm8 = 0000 0 0 00

P Precision Mask; o. normal i. inexact (niedokładny)

RS Rounding select (wybór zaokrąglenia) i. MXCSR.RC o. imm8.RC

RC Rounding mode (sposób zaokrąglenia)

RC Rounding mode:  
 oo - Zaokrąglaj do najbliższej (parzystej)  
 oi - Zaokrąglaj w dół (w kierunku -oo)  
 o - Zaokrąglaj w górę ( w kierunku +oo)  
 ui - Zaokrąglaj do zera (obemij)

Bit od 128/56 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekooperacyjne

17

Precision:  
 if imm8[3] = 0;  
 if imm8[3] = 1  
 then ustawianie precyzji jest ignorowane;

## Instrukcja wartości odwrotności przybliżonej pierwiastka

### VRSQRTSS / VRSQRTPS

vrsqrts xmmi, xmm2, xmm3/m32

Oblicza przybliżoną odwrotność pierwiastka z liczby rzeczywistej pojedynczej precyzji, wynik umieszcza w xmmi, dodatkowo przepisuje starsze elementy xmm2 do xmmi. (Relative Error  $\leq 1,5 \cdot 2^{-12}$ )

```
xmmi[31:0] ← 1.0/sqrt(xmm3/m32[31:0])
xmmi[127:32] ← xmmi2[127:32]
```

vrsqrtps xmmi, xmm2/m128

vrsqrtps ymmi, ymm2/m256

Oblicza przybliżoną odwrotność pierwiastka z elementów wektora liczb rzeczywistych pojedynczej precyzji xmm2/ymm2 lub m128/m256, wynik umieszcza w xmmi/ymmi. (Relative Error  $\leq 1,5 \cdot 2^{-12}$ )

```
xmmi[i] ← 1.0/sqrt(xmm3/m32[i])
ymmi[i] ← 1.0/sqrt(ymm3/m256[i])
```

Bit od 128/56 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekooperacyjne

19

## Instrukcja wartości odwrotności przybliżonej pierwiastka

### VSQRT[S/P/S]

vsqrts xmmi, xmm2, xmm3/m32

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

```
xmmi[31:0] ← sqrt(xmm3/m32[31:0])
xmmi[127:32] ← xmmi2[127:32]
```

vsqrtps xmmi, xmm2/m128

vsqrtps ymmi, ymm2/m256

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

```
xmmi[i] ← sqrt(xmm3/m128[i])
ymmi[i] ← sqrt(ymm3/m256[i])
```

Bit od 128/56 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekooperacyjne

20

## 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:** VFNMNSUB[123/213/231][S/P][S/D]

(C) KISI d.KIK PCz 2022 Programowanie niskopoziomowe

123

## 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 precyzyji 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 xmmi/ymmi.

132  
 $y_{mmi}[i] = y_{mmi}[i] * y_{mm3/m256}[i] + y_{mm2}[i]$

231  
 $y_{mmi}[i] = y_{mm2}[i] * y_{mm3/m256}[i] + y_{mmi}[i]$

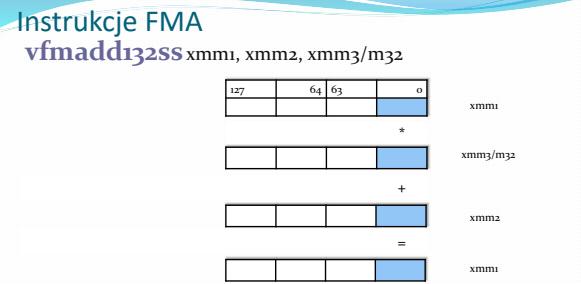
213  
 $y_{mmi}[i] = y_{mm2}[i] * y_{mmi}[i] + y_{mm3/m256}[i]$

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

124



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

125

## 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 precyzyji dwóch rejestrów i **odejmuję** odpowiednie wartości trzeciego rejestru w zależności od podanej kolejności, pierwsze dwie cyfry oznaczają czynniki iloczynu, trzecia cyfra jest składnikiem sumy, wynik zapisuje w xmmi/ymmi.

132  
 $y_{mmi}[i] = y_{mmi}[i] * y_{mm3/m256}[i] - y_{mm2}[i]$

231  
 $y_{mmi}[i] = y_{mm2}[i] * y_{mm3/m256}[i] - y_{mmi}[i]$

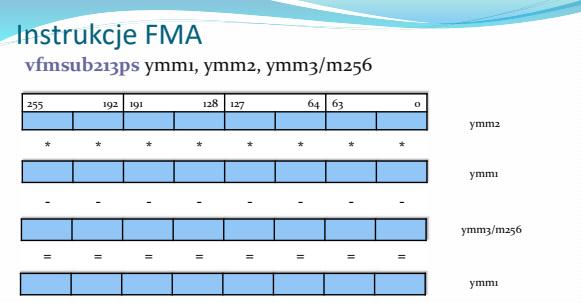
213  
 $y_{mmi}[i] = y_{mm2}[i] * y_{mmi}[i] - y_{mm3/m256}[i]$

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

126



(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

127

## Instrukcje FMA

### VFMADDSSUB[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 precyzyji dwóch rejestrów i naprzemienne **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[si] = ymm1[si] * ymm2/m128[si] - ymm1[si]
    ymm1[si+1] = ymm1[si+1] * ymm2/m128[si+1] + ymm2[si+1]

231
    ymm1[si] = ymm1[si] * ymm2/m256[si] - ymm1[si]
    ymm1[si+1] = ymm2[si+1] * ymm3/m256[si+1] + ymm1[si+1]

213
    ymm1[si] = ymm1[si] * ymm2[si] - ymm3/m256[si]
    ymm1[si+1] = ymm2[si+1] * ymm3[si+1] + ymm3/m256[si+1]

```

Bitы od 128:256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

128

Programowanie niekompromisowe

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

128

## 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 precyzyji dwóch rejestrów i naprzemienne **dodaje i odejmuje** odpowiednie wartości trzeciego rejestru w zależności od podanej kolejności, pierwsze dwie cyfry oznaczają czynniki iloczynu, trzecia cyfra jest elementem sumy różnicowej, wynik zapisuje w xmm1/ymm1.

```

132
    ymm1[si] = ymm1[si] * ymm3/m128[si] + ymm2[si]
    ymm1[si+1] = ymm1[si+1] * ymm3/m128[si+1] - ymm2[si+1]

231
    ymm1[si] = ymm2[si] * ymm3/m256[si] + ymm1[si]
    ymm1[si+1] = ymm2[si+1] * ymm3/m256[si+1] - ymm1[si+1]

213
    ymm1[si] = ymm2[si] * ymm3[si] + ymm3/m256[si]
    ymm1[si+1] = ymm2[si+1] * ymm3[si+1] - ymm3/m256[si+1]

```

Bitы od 128:256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

129

Programowanie niekompromisowe

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

## 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/m32/m28

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

Mnoży skalary / wektory liczb rzeczywistych pojedynczej / podwójnej precyzyji dwóch rejestrów, zmienia znak iloczynu 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] = -ymmu[i] * ymm3/m256[i] + ymm2[i]

231
    ymm1[i] = -ymmu[i] * ymm3/m256[i] + ymm1[i]

213
    ymm1[i] = -ymmu[i] * ymm1[i] + ymm3/m256[i]

```

Bitы od 128:256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

132

Programowanie niekompromisowe

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

## Instrukcja FMA

### vfmaddsub231pd ymm1, ymm2, ymm3/m256

255	192	191	128	127	64	63	0
*	*	*	*	*	*	*	*
+	-		+	-			
=	=		=	=			

ymmm2

ymm3/m256

ymmu

ymmu

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

129

## Instrukcja FMA

### vfmsubadd231pd ymm1, ymm2, ymm3/m256

255	192	191	128	127	64	63	0
*	*	*	*	*	*	*	*
-	+		-	+			
=	=		=	=			

ymmm2

ymm3/m256

ymmu

ymmu

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

130

## Instrukcje FMA

### vfnmadd132ss ymm1, ymm2, ymm3/m256

127	64	63	0
			neg
*			*
+			

xmm1

xmm3/m32

xmm2

xmm1

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

131

## 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/m32/m64

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

Mnozy skalary / wektory liczb rzeczywistych pojedynczej / podwójnej precyzji dwóch rejestrów, zmienia znak iloczynów i odaje mu odpowiednie wartości trzeciego rejestrów w zależności od podanej kolejności, pierwsze dwie cyfry oznaczają czynniki iloczynu, trzecia cyfra jest składnikiem sumy, wynik zapisuje w xmm/ymm.

$$\text{132} \quad \text{yymm1[i]} = -\text{yymm1[i]} * \text{yymm3/m256[i]} - \text{yymm2[i]}$$

$$\text{231} \quad \text{yymm1[i]} = -\text{yymm2[i]} * \text{yymm3/m256[i]} - \text{yymm1[i]}$$

$$\text{233} \quad \text{yymm1[i]} = -\text{yymm2[i]} * \text{yymm1[i]} - \text{yymm3/m256[i]}$$

Bitы od 128-256 do MSB są zerowane.

Programowanie niekompromisowe

134

## Instrukcje FMA

**vfnmsub213ps ymm1, ymm2, ymm3/m256**

255	192	191	128	127	64	63	0
neg							

ymmm2

*	*	*	*	*	*	*	*
-	-	-	-	-	-	-	-

ymmm

=	=	=	=	=	=	=	=
-	-	-	-	-	-	-	-

yymm3/m256


yymm


yymm


yymm

## Operacje porównania

## 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/m32/m64/m128/m256 wedlug funktora zapisanego na bitach imm8[4:0], łącznie 32 funktry wynik jako liczbę całkowitą -1 lub 0 zapisuje w xmmi/ymmi.

Bitы od 128-256 do MSB są zerowane.

Programowanie niekompromisowe

135

## Operacje porównania:

- Instrukcje porównania:**

**VCMP[S/P][S/D]**

**VCOMIS[S/D]**

**VUCOMIS[S/D]**

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

137

funktor	imask	opis	>			<			=			Unord.			NaN or QNaN			
			>H	<H	=H	>L	<L	=L	>N	<N	=N	>Z	<Z	=Z	>S	<S	=S	
EQ_OQ (EQ)	0H	Equal (ordered, non-signaling)	t	t	t	t	t	t	No	No	No	t	t	t	t	t	No	
LT_OQ (LT)	1H	Less-than (ordered, non-signaling)	t	t	t	t	t	t	Yes	EQ_OQ	10H	Equal (ordered, non-signaling)	f	f	f	t	Yes	Yes
LE_OQ (LE)	2H	Less-than-or-equal (ordered, non-signaling)	t	t	t	t	t	t	Yes	LT_OQ	11H	Less-than (ordered, non-signaling)	f	t	f	f	No	No
UNORD_Q (UNORD)	3H	Unordered (non-signaling)	t	t	t	t	t	t	No	UNORD_S	13H	Unordered (signaling)	f	t	f	t	No	No
NEQ_OQ (NEQ)	4H	Not-equal (ordered, non-signaling)	t	t	f	t	t	No	NEQ_US	14H	Not-equal (unordered, signaling)	f	t	f	t	Yes	Yes	
NLT_US (NLT)	5H	Not-less-than (ordered, non-signaling)	t	f	t	t	t	Yes	NLT_UQ	15H	Not-less-than (unordered, non-signaling)	t	f	t	t	No	No	
NLE_US (NLE)	6H	Not-less-than-or-equal (ordered, non-signaling)	t	f	f	t	t	Yes	NLE_UQ	16H	Not-less-than-or-equal (unordered, non-signaling)	t	f	f	t	No	No	
ORD_Q (ORD)	7H	Ordered (non-signaling)	t	t	t	t	t	No	ORD_S	17H	Ordered (signaling)	t	t	t	f	Yes	Yes	
EQ_OQ	8H	Equal (unordered, non-signaling)	t	f	t	t	t	No	EQ_US	18H	Equal (unordered, signaling)	f	t	t	t	Yes	Yes	
NGE_US (NGE)	9H	Not-greater-than-unordered (ordered, signaling)	t	t	f	t	t	Yes	NGE_UQ	19H	Not-greater-than-or-equal (unordered, non-signaling)	f	t	f	t	No	No	
NGT_US (NGT)	AH	Not-greater-than (unordered, non-signaling)	t	t	t	t	t	Yes	NGT_UQ	1AH	Not-greater-than (ordered, non-signaling)	t	t	t	t	No	No	
FALSE_OQ (FALSE)	BH	False (ordered, non-signaling)	t	f	t	t	t	No	FALSE_US	1BH	False (unordered, signaling)	f	t	f	t	Yes	Yes	
NEQ_OQ (NEQ)	CH	Not-equal (ordered, non-signaling)	t	t	f	t	t	No	NEQ_US	1CH	Not-equal (unordered, signaling)	t	t	f	t	Yes	Yes	
GE_OQ (GE)	DH	Greater-than-or-equal (ordered, non-signaling)	t	t	t	t	t	Yes	GE_OQ	1DH	Greater-than-or-equal (unordered, non-signaling)	t	t	t	f	No	No	
GT_OQ (GT)	EH	Greater-than (ordered, non-signaling)	t	t	f	t	t	Yes	GT_US	1EH	Greater-than (unordered, non-signaling)	t	t	f	t	No	No	

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

139

**Instrukcje porównania VCMP[S/P][S/D]**

#	Pseudo-opis	Implementacja
1.	VCMPEDQS[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_0
2.	VCMPITNS[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_1
3.	VCMPLES[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_2
4.	VCMPUNORDS[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_3
5.	VCMPNEQQS[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_4
6.	VCMPNITS[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_5
7.	VCMPNLEQS[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_6
8.	VCMPORDS[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_7
9.	VCMPEQ_UQS[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_8
10.	VCMPNGES[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_9
11.	VCMPNITS[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_10
12.	VCMPFALSES[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_11
13.	VCMPNEQQS[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_12
14.	VCMPGENS[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_13
15.	VCMPGTQS[P/S/D] reg, reg, reg	VCMP[S/P][S/D] reg, reg, reg, reg3_14

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

140

**Instrukcje porównania****VCOMIS[S/D] / VUCOMIS[S/D]**

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

Porównuje pojedyncze liczby rzeczywiste pojedynczej/podwójnej precyzyji z rejestr xm2 lub pamięci m32/m64 i xmmi, wynikiem jest ustawienie odpowiednich flag.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

141

**Instrukcje porównania****VCOMIS[S/D] / VUCOMIS[S/D]**

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

Porównuje pojedyncze liczby rzeczywiste pojedynczej/podwójnej precyzyji z rejestr xm2 lub pamięci m32/m64 i xmmi, wynikiem jest ustawienie odpowiednich flag.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

142

**Operacje logiczne****Operacje logiczne**

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

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

144

**Instrukcje logiczne koniunkcja****VANDP[S/D] / VANDNP[S/D]**

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

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

Zwraca prawdę (1) lub fałsz (0) dla koniunkcji wektorów liczb rzeczywistych pojedynczej/podwójnej precyzyji z xmm2/ymm2 i xmm3/ymm3 lub m128/m256, wynik zapisuje w xmmi/ymmi.

cel[i] = źródło1[i] and źródło2[i]  
 vandp[s/d] xmm1, xmm2, xmm3/m128  
 vandp[s/d] ymm1, ymm2, ymm3/m256  
 Zwraca prawdę (1) lub fałsz (0) dla koniunkcji z negacją wektorów liczb rzeczywistych pojedynczej/podwójnej precyzyji z xmm2/ymm2 i xmm3/ymm3 lub m128/m256, wynik zapisuje w xmmi/ymmi.  
 cel[i] = (not źródło1[i]) and źródło2[i]

Bity od 128/256 do MSB są zerowane.

Programowanie niskopoziomowe

145

## Instrukcje logiczne alternatywa

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

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

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

Zwraca prawdę (1) lub falsz (0) dla alternatywy wektorów liczb rzeczywistych pojedynczej/podwójnej precyzji rejestru xmm2/ymm2 (źródło 1) i xmm3/ymm3 lub m128/m256 (źródło 2), wynik zapisuje w xmm1/ymm1.

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

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

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

Zwraca prawdę (1) lub falsz (0) dla alternatywy wykluczającej wektorów liczb rzeczywistych pojedynczej/podwójnej precyzji rejestru xmm2/ymm2 (źródło 1) i xmm3/ymm3 lub m128/m256 (źródło 2), wynik zapisuje w xmm1/ymm1.

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

146

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

## Instrukcje testowania

### VTESTP[S/D]

vtestp[s/d] xmml, xmm2/m128

vtestp[s/d] ymm1, ymm2/m256

Wykonuje logiczne koniunkcję (AND) na bitach znaku liczb rzeczywistych pojedynczej / podwójnej precyzji z rejestru xmml/ymm1 i xmm2/ymm2 lub m1128/m256, wynikiem jest ustalenie flagi ZF, jeśli wszystkie bity znaku = 0 => ZF = 1 else ZF = 0

oraz

wykonuje logiczne koniunkcję z zaprzeczeniem (AND NOT) na bitach znaku liczb rzeczywistych pojedynczej / podwójnej precyzji z rejestru xmml/ymm1 i xmm2/ymm2 lub m1128/m256, wynikiem jest ustalenie flagi CF, jeśli wszystkie bity znaku = 0 => CF = 1 else CF = 0

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

147

## Operacje konwersji



## Instrukcje konwersji całkowite na rzeczywiste

### VCVTQD2P[S/D]

vcvtqd2p[s/d] xmml, xmm2/m128

vcvtqd2p[s/d] ymm1, ymm2/m256

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

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

150

## Operacje logiczne

- Calkowite na rzeczywiste:

VCVTQD2[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 transakcją)

- Rzeczywiste na rzeczywiste:

VCVTSD2SS, VCVTSS2SD

VCVTPD2PS, VCVTPS2PD

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

149

## Instrukcje konwersji całkowite na rzeczywiste

### VCVTSI2S[S/D]

vcvtsi2ss xmml, xmm2, reg/m32

vcvtsi2sd xmml, xmm2, reg/m64

Konwertuje pojedyncze podwójne słwo ze znakiem z rejestru ogólnego przeznaczenia lub m32/m64 na jedną liczbę rzeczywistą pojedynczej/podwójnej precyzji, wynik zapisuje do xmm1/ymm1, bity [127:64/31] są przepisywane z xmm2/ymm2

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niekompromisowe

151

## Instrukcje konwersji rzeczywiste na całkowite

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

`vcvt[t]ps2dq xmmi, xmm2/m128`

`vcvt[t]ps2dq ymmi, 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 `xmmi/ymmi` używając tanszakcji [T] z zaokrągleniem w kierunku zera.

`vcvt[t]pd2dq xmmi, xmm2/m128`

`vcvt[t]pd2dq ymmi, ymm2/m256`

Konwertuje cztery/dwa pojedyncze słowa ze znakiem z `xmm2/ymm2` lub `m128/m256` na cztery/dwa podwójne słowa ze znakiem, wynik zapisuje w `xmmi/ymmi` używając tanszakcji [T] z zaokrągleniem w kierunku zera.

Bity od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

152

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

153

## Instrukcje konwersji rzeczywiste na rzeczywiste

### **VCVTSD2SS / VCVTSS2SD**

`vcvtsd2ss xmmi, xmm2, xmm3/m64`

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

`vcvtss2sd xmmi, xmm2, xmm3/m32`

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

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

154

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

155

## Instrukcje konwersji rzeczywiste na rzeczywiste

### **VCVTPD2PS / VCVTPS2PD**

`vcvtpd2ps xmmi, xmm2/m128`

`vcvtpd2ps xmmi, 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 xmmi, xmm2/m64`

`vcvtps2pd ymmi, xmm2/m128`

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

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

156

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

155

## Instrukcje konwersji rzeczywiste na całkowite

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

`vcvt[t]ss2si reg32, xmmi/m32`

`vcvt[t]ss2si reg64, ymmi/m64`

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

`vcvt[t]sd2si reg32, xmmi/m64`

`vcvt[t]sd2si reg64, ymmi/m64`

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

Instrukcja z T oznacza konwersję z transzakcją z zaokrągleniem w kierunku zera.

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

153

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

154

## Instrukcje konwersji rzeczywiste na rzeczywiste

### **VCVTSD2SS / VCVTSS2SD**

`vcvtsd2ss xmmi, xmm2, xmm3/m64`

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

`vcvtss2sd xmmi, xmm2, xmm3/m32`

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

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

155

Bit od 128/256 do MSB są zerowane.

(C) KISI d.KIK PCz 2022

Programowanie niskopoziomowe

154