



# Programowanie niskopoziomowe

Programowanie niskopoziomowe (Politechnika Czestochowska)

Rejestry	Windows 32	Windows 64
Możliwe do użycia bez zabezpieczenia (stos)	> <b>GPR</b> : EAX, ECX, EDX > <b>X87</b> : st(0)-st(7) > <b>Rejestry maskujące</b> : K0-K7 > <b>SSE</b> : XMM0-XMM7	> <b>GPR</b> : RAX, RCX, RDX, R8-R11 > <b>X87</b> : st(0)-st(7) > <b>Rejestry maskujące</b> : K0-K7 > <b>SSE</b> : XMM0-XMM5 (bez AVX512), XMM16-XMM31 (jeśli mamy AVX512)
Konieczne zabezpieczenie (stos)	> <b>GPR</b> : EBX, ESI, EDI, EBP	> <b>GPR</b> : RBX, RSI, RDI, RBP, R12-R15 > <b>SSE</b> : XMM6-XMM15
Parametry	> <b>cdecl, stdcall, pascal, GNU C</b> : na stosie > <b>fastcall – Microsoft/GNU</b> : ECX, EDX > <b>fastcall – Borland</b> : EAX, EDX, ECX > <b>thiscall – Microsoft</b> : ECX	> <b>całkowite/wskaźnik</b> : RCX, RDX, R8, R9 > <b>zmiennoprzecinkowe</b> : XMM0-XMM3 > <b>reszta na stosie</b>
Zwracające wartość	> <b>całkowite/wskaźnik</b> : EAX, EDX > <b>zmiennoprzecinkowe</b> : st(0)	> <b>całkowite/wskaźnik</b> : RAX > <b>zmiennoprzecinkowe</b> : XMM0

Przykłady (Windows 64)				
void a(float, int*, int, double, float)				
Arg1	Arg2	Arg3	Arg4	Arg5
XMM0	RDX	R8	XMM3	Stos
void a(float, float, float, double*)				
Arg1	Arg2	Arg3	Arg4	
XMM0	XMM1	XMM2	R9	

Co	Rok	Ficzerzy (pogrubione te które pojawiły się w pytaniach z wykładu)	Co (kont.)	Rok	Ficzerzy
F14 CADC	Czerwiec 1970	> <b>20 bitowy układ</b> z techniką potokową >na potrzeby myśliwca F-14 TOMCAT	Intel 80186	1982	>16 bitowy >większa wydajność, nowe rozkazy, szybszy zegar >niektóre instrukcje 10-20 razy szybciej >wykorzystany głównie w systemach wbudowanych jako mikrokontroler
Intel 4004	Od 1971	>Uznawany za pierwszy mikroprocesor >Maks. Taktowanie – <b>740 kHz</b> >Osobna pamięć dla danych i programu (arch. harwardzka) >46 instrukcji > <b>16x 4 bitowe rejestry</b> >3-poziomowy stos >2300 tranzystorów (10 um)	Intel 80286	1 lutego 1982	>dwa razy bardziej wydajny od 8086 >24 bitowa szyna adresowa (adresacja do 16MB RAM) >nowe instrukcje >nowy tryb adresowania pamięci – <b>tryb chroniony</b>
Intel 8008	Kwiecień 1974	>Pierwszy 8 bitowy mikroprocesor Intela >obudowa DIP18 >8 bitowa magistrala >dostęp do większej ilości RAM >3-4 razy więcej mocy obliczeniowej niż procesory 4-bitowe  >8 bitowa szyna danych >16 bitowa szyna adresowa >słowo 8 bitowe >72 instrukcje >bezpośrednie adresowanie pamięci o poj. Do 64 KB >arytmetyka dwójkowa i dziesiętna kodowana dwójkowo (BCD) >8 rejestrów programowych dostępnych dla programisty – cykl zapisu 2us >zegar zewnętrzny 2-3MHz (podstawowy cykl rozkazowy – 4 takty)	Intel 80386	1986	> <b>32 bitowa magistrala adresowa</b> oraz 32 bitowa magistrala danych >rozszerzenie rejestrów do 32 bitów >nowe tryby adresowania >praca w trybie: rzeczywistym (16 bit), chronionym (32 bit), wirtualnym (32 bit) > <b>dodanie jednostki MMU – stronicowanie pamięci</b>
Intel 8086	<b>1978</b>	>traktowany był jako projekt przejściowy > <b>główny konstruktor: Stephen (lub Steven) Morse</b> >rozszerzenie listy rozkazów >rozszerzenie możliwości adresowania operandów >wprowadzenie segmentacji obszaru pamięci >mechanizmy przyspieszenia pracy >mechanizmy dla pracy wieloprotocessorowej	Intel 80486		>nazwa handlowa i486 >nowe instrukcje (7) > <b>zintegrowany koprotocessor arytmetyczny x87</b> >poprawiony interfejs szyny danych >pięciostopniowy potok >dwukrotnie szybszy od 80386

Co (Pentium)	Rok	Ficzerzy
Pentium	22 marca 1993	>architektura superskalarna >64 bitowa szyna danych >jednostka przewidywania skoku (branch prediction) – skuteczność 80% >przeprojektowany koprocessor (5x-6x wydajniejszy niż w i486)
Pentium Pro	Październik 1995	>podział kodu x86 na mikrorozkazy (microcode) >wykonywanie poza kolejnością >wykonywanie spekulatywne >dodatkowy potok dla prostych instrukcji >instrukcja CMOVcc
Pentium II	7 maja 1997	> <b>instrukcje MMX</b> >poprawiona obsługa programów 16 bitowych >pamięć podręczna 1 poziomu (L1) dla kodu i danych: 16KB
Pentium III		>architektura typu RISC >rozmiar pamięci podręcznej L1 dla kodu: 16 KB > <b>liczba etapów przetwarzania rozkazu (w potoku): 12</b> >liczba jednostek zmiennoprzecinkowych: 1 (z potokowaniem) >liczba jednostek całkowitoliczbowych: 6 potoków >liczba jednostek MMX: 2 > <b>instrukcje SSE (Streaming SIMD Extensions)</b> >możliwość pracy w trybie wieloprocessorowym (do 2 procesorów)
Pentium 4 (wiele wersji)	20 listopada 2000	>architektura NetBurst > <b>instrukcje SSE2, w nowszych wersjach jądra: SSE3</b> >niektóre modele: wielowątkowość HyperThreading >zwiększona pamięć L2 >pojawia się technologia EM64T (2003) >pierwszy procesor dwurdzeniowy

Co (Intel Core)	Ficzy	Co (Intel Core) (kont.)	Ficzy
Intel Core 2	>8 generacja >mikroarchitektura Intel Core <b>&gt;współczynnik IPC: 3,5</b> >wspólna pamięć dla obu rdzeni >EM64T >technologia wirtualizacji >XD bit >ulepszona technologia SpeedStep >wersja czterordzeniowa	Intel Core i7 Kaby Lake (7 gen)	>14nm+ proces >zwiększenie częstotliwości zegara >zwiększenie wydajności wbudowanego GPU >wsparcie Intel Optane Memory storage caching
Intel Core i7 Nehalem (1 gen)	>modułowa budowa >8 rdzeniowy Nehalem: <b>731 milionów tranzystorów</b> >SSE 4.2 >technologia współbieżnej wielowątkowości >dynamiczne zarządzanie zasilaniem >wbudowanie kontrolera RAM >technologia QuickPath	Intel Core i7 Coffee Lake (8 gen)	>14nm++ proces >zwiększenie liczby rdzeni do 6 >zwiększenie częstotliwości zegara >zwiększenie wydajności wbudowanego GPU
Intel Core i7 Sandy Bridge (2 gen)	>32nm proces >6 jednostek wykonawczych >wbudowany układ graficzny <b>&gt;instrukcje AVX</b> >Turbo Boost 2.0 >pamięć cache L3	Intel Core i7 Coffee Lake (9 gen)	>14nm++ proces >zwiększenie liczby rdzeni do 8 >zwiększenie rozmiaru cache L3 >AVX512 w wersji X
Intel Core i7 Ivy Bridge (3 gen)	>22nm proces (tranzystory 3D) >wbudowane GPU Intel HD Graphics <b>&gt;generator liczb losowych</b> >PCI Express 3.0	Intel Core i9 Comet Lake (10 gen)	>zwiększenie maks. Taktowania do 5.3GHz >zwiększenie liczby rdzeni do 10 >zwiększenie rozmiaru cache L3 >zwiększenie wydajności wbudowanego GPU (Gen 9.5)
Intel Core i7 Haswell (4 gen)	>podniesiona wydajność cache >zwiększona wydajność i energooszczędność >8 jednostek wykonawczych <b>&gt;instrukcje AVX2</b> <b>&gt;instrukcje FMA3</b> >rozbudowany układ graficzny >wsparcie dla Direct3D 11.1 i OpenGL 4.0	Intel Core i9 Rocket Lake (11 gen)	>zwiększenie IPC o 10 do 19% >instrukcje AVX512 i DL (Deep Learning) >zwiększenie rozmiaru cache L1 dla danych: 48KiB >nowy układ graficzny – Xe-LP Gen 12 – DisplayPort 1.4a, HDMI 2.0b >PCI Express 4.0
Intel Core i7 Broadwell (5 gen)	>14nm proces >obsługa pamięci DDR3L >układ iGPU Iris Pro 6200 >127 MB pamięci podręcznej eDRAM >wsparcie dla Direct3D 11.1 i OpenGL 4.4	Intel Core i9 Alder Lake (12 gen)	>hybrydowy – wydajne rdzenie dwuwałkowe i rdzenie energooszczędne – do 8P i 8E >10nm proces SuperFin (Intel 7) >zwiększenie IPC >12 jednostek wykonawczych >nowe instrukcje AVX-VNNI >zwiększenie rozmiaru cache L3 >nowy układ graficzny – Xe-LP Gen 12.2 (32-96EU) >PCI Express 5.0
Intel Core i7 Skylake (6 gen)	>14nm proces >obsługa pamięci DDR3L i DDR4 <b>&gt;instrukcje AVX512</b> – 32 rejestry ZMM – 512 bitowe w wersji XEON >wsparcie dla Direct3D 12 >wsparcie dla Thunderbolt 3.0		

Oryginalne przeznaczenie (z pierwszych 8 i R8, R9 itd. możemy jednak korzystać jak chcemy)	Tryb 64 bitowy (EM64T)				
	Rejestr				
	64 bity	Dolne 32 bity	Dolne 16 bitów	Dolne 8 bitów	Górne 8 bitów
Akumulator	RAX	EAX	AX	AL	AH
Wskaźnik na tablicę	RBX	EBX	BX	BL	BH
Licznik	RCX	ECX	CX	CL	CH
Dane	RDX	EDX	DX	DL	DH

Aktualna ramka stosu	RBP	EBP	BP	BPL	
Wierzchołek stosu	RSP	ESP	SP	SPL	
Indeks źródła	RSI	ESI	RSI	SIL	
Indeks celu	RDI	EDI	RDI	DIL	

Licznik rozkazów	RIP	EIP	IP		
------------------	-----	-----	----	--	--

Dowolne	R8	R8w	R8d	R8b	
	R9	R9w	R9d	R9b	
	R10	R10w	R10d	R10b	
	R11	R11w	R11d	R11b	
	R12	R12w	R12d	R12b	
	R13	R13w	R13d	R13b	
	R14	R14w	R14d	R14b	
	R15	R15w	R15d	R15b	

Flagi	RFLAGS	EFLAGS	FLAGS		
-------	--------	--------	-------	--	--

Seg. z instr. prog.			CS		
Seg. ze stosem prog.			SS		
Seg. dane			DS		
Seg. na dane #2			ES		
Seg. na dane #3			FS		
Seg. na dane #4			GS		

Tryb 32 bitowy (IA-32)			
Rejestr			
32 bity	Dolne 16 bitów	Dolne 8 bitów	Górne 8 bitów
EAX	AX	AL	AH
EBX	BX	BL	BH
ECX	CX	CL	CH
EDX	DX	DL	DH

EBP	BP		
ESP	SP		
ESI	RSI		
EDI	RDI		

EIP	IP		
-----	----	--	--


EFLAGS	FLAGS		
--------	-------	--	--

	CS		
	SS		
	DS		
	ES		
	FS		
	GS		

Przyporządkowanie  
rejestrów do  
przedrostków seg.  
(wykład str. 3)

DS
DS
DS
DS

SS
SS
DS
DS, ES

CS
----


--


Bajt	(tylko AX, BX, CX, DX)			
7	RAX	EAX	AX	AH
6				
5				
4				
3				
2				
1				
0				

Bajt	Tylko R8-R15)			
7	R8	R8d	R8w	R8b
6				
5				
4				
3				
2				
1				
0				

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol	Przykład 1
Instrukcje podstawowe								
Cel, źródło								
MOV	MOV cel, źródło	Reg, Reg	Cel = źródło		---	Move		
XCHG	XCHG cel, źródło	Reg, Mem	Cel ↔ źródło (zamienia miejscami) Domyślnie posiada prefiks LOCK, jeśli jeden z arg. Jest w pamięci			Exchange	Zamień miejscami	
XADD	XADD cel, źródło	Mem, Reg	1. cel ↔ źródło 2. cel += źródło		OSZAPC	Exchange and add	Zamień miejscami i dodaj	
CMPXCHG	CMPXCHG cel, źródło	Reg/Mem, Stała	if acc == cel then Cel = źródło Else Acc = cel End Acc – al/ax/eax/rax			Compare and exchange	Porównaj i zamień	
Cel								
BSWAP	BSWAP cel	Reg/Mem (32/64 bit)	Zamiana bajtów		---	Byte swap		(32 bit) Przed: 41 e1 b5 59 Po: 59 b5 e1 41 Nie zamienia kolejności bitów w bajtach
CMPXCHG8B	CMPXCHG8B cel	Mem (obszar 8 bajtów)	if edx:eax/rdx:rax == cel then Cel = ecx:ebx/rcx:rbx Else edx:eax/rdx:rax = cel End		OSZAPC	Compare and exchange 8 bytes	Porównaj i zamień 8 bajtów	
CMPXCHG16B	CMPXCHG16B cel	Mem (obszar 16 bajtów)	Edx, eax, ecx, ebx dla w. 8 bajtów Rdx, rax, rcx, rbx dla w. 16 bajtów			Compare and exchange 16 bytes	Porównaj i zamień 16 bajtów	

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol	Przykład 1
Instrukcje operacji na stosie								
Źródło/cel								
PUSH	PUSH źródło	Reg/Mem (16/32/64 bit)	źródło na stos		---		Wrzuć na stos	
POP	POP cel		Cel = wierzchołek stosu				Zdejmij ze stosu	
PUSHF	PUSHF	---	rejestr FLAGS na stos		---	Push flags (FLAGS)	Wrzuć na stos flagi (rejestr FLAGS)	
PUSHFD	PUSHFD		rejestr EFLAGS na stos			Push flags dword (EFLAGS)	Wrzuć na stos podwójne słowo flag (rejestr EFLAGS)	
PUSHFQ	PUSHFQ		rejestr RFLAGS na stos			Push flags qword (RFLAGS)	Wrzuć na stos poczwórne słowo flag (rejestr RFLAGS)	
POPF	POPF		FLAGS = wierzchołek stosu		0SZAPC	Pop flags	Zdejmij ze stosu flagi i wrzuć do FLAGS	
POPFD	POPFD		EFLAGS = wierzchołek stosu			Pop flags dword (EFLAGS)	Zdejmij ze stosu podwójne słowo flag i wrzuć do EFLAGS	
POPFQ	POPFQ		RFLAGS = wierzchołek stosu			Pop flags qword (RFLAGS)	Zdejmij ze stosu poczwórne słowo flag i wrzuć do RFLAGS	
PUSHA	PUSHA		ax, cx, dx, bx, sp, bp, si, di na stos	Nie działają w trybie 64 bitowym	---	Push all GPR registers	Wrzuć na stos wszystkie rejestry ogólnego przeznaczenia (16 bit)	
PUSHAD	PUSHAD		eax, ecx, edx, ebx, esp, ebp, esi, edi na stos			Push all GPR registers dword	Wrzuć na stos wszystkie rejestry ogólnego przeznaczenia (32 bit)	
POPA	POPA		ax, cx, dx, bx, sp, bp, si, di = zawartość stosu			Pop all GPR registers	Zdejmij ze stosu wszystkie rejestry ogólnego przeznaczenia (16 bit)	
POPAD	POPAD		eax, ecx, edx, ebx, esp, ebp, esi, edi = zawartość stosu			Pop all GPR registers dword	Zdejmij ze stosu wszystkie rejestry ogólnego przeznaczenia (32 bity)	



Mnemonic	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol	Przykład 1
Instrukcje konwersji								
CWD	CWD	---	rozszerz ax na dx:ax z uwzgl. Znaku (16 → 32)	Rozszerza na dwa rejestry, używane przed dzieleniem	---	Convert <b>w</b> ord to <b>d</b> word with sign extension		
CDQ	CDQ		rozszerz eax na edx:eax z uwzgl. Znaku (32 → 64)			Convert <b>d</b> word to <b>q</b> word with sign extension		
CQO	CQO		rozszerz rax na rdx:rax z uwzgl. Znaku (64 → 128)			Convert <b>q</b> word to <b>o</b> ctaword with sign extension		
CBW	CBW		rozszerz al na ax z uwzgl. Znaku(8 → 16)	Rozszerza rejestr akumulatora		Convert <b>b</b> yte to <b>w</b> ord with sign extension		
CWDE	CWDE		rozszerz ax na eax z uwzgl. Znaku(16 → 32)			Convert <b>w</b> ord to <b>d</b> word with sign extension		
CDQE	CDQE		rozszerz eax na rax z uwzgl. Znaku(32 → 64)			Convert <b>d</b> word to <b>q</b> word with sign extension		
Cel, źródło								
MOVSX	MOVSX cel, źródło	Reg (16/32 bit), Reg/Mem (8/16 bit)	Cel = źródło, rozszerz z uwzgl. Znaku		---	<b>M</b> ove with sign extension		
MOVSXD	MOVSXD cel, źródło	Reg (64 bit), Reg/Mem (32 bit)	Cel = źródło, rozszerz z uwzgl. Znaku Tylko 32 na 64 bit			<b>M</b> ove with sign extension <b>d</b> word		
MOVZX	MOVZX cel, źródło	Reg (16/32/64 bit), Reg/Mem (8/16 bit)	Cel = źródło, rozszerz zerami			<b>M</b> ove with zero extend		

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol	
Instrukcje dodawania i odejmowania								
Cel, źródło								
ADD	ADD cel, źródło	Reg, Reg	Cel += źródło		OSZAPC	Add	Dodaj	
ADC	ADC cel, źródło		Cel += źródło + CF			Add with carry	Dodaj z przeniesieniem bez znaku (CF)	
ADCX	ADCX cel, źródło	Reg, Mem	Cel += źródło + CF	Pozwalają na dwa niezależne toki operacji dodawania z przeniesieniem (carry-chain) liczb wielobitowych (128b, 256b itd.)	—C	Add with carry without affecting flags	Dodaj z przeniesieniem bez znaku (CF)	
ADOX	ADOX cel, źródło	Mem, Reg	Cel += źródło + OF	Wymaga ADX	0—	Add with overflow without affecting flags	Dodaj z przeniesieniem <b>bez</b> znaku (OF)	
SUB	SUB cel, źródło	Reg/Mem, Stała	Cel -= źródło		OSZAPC	Subtract	Odejmij	
SBB	SBB cel, źródło		Cel = cel – (źródło + CF)			Subtract with borrow	Odejmij z pożyczką	
Instrukcje mnożenia i dzielenia								
Źródło								
MUL	MUL źródło	Reg/Mem	ax/dx:ax/edx:eax/rdx:rax = al/ax/eax/rax * źródło Nie uwzględnia znaku OF=0 i CF=0 jeśli starsza połowa bitów jest równa zero (wynik mieści się w akumulatorze)		0xxxxC	Unsigned multiply	Pomnóż bez znaku	
Cel1, cel2, źródło								
MULX	MULX cel1, cel2, źródło	Reg, Reg, Reg/Mem (wszystkie 32/64 bit)	cel1:cel2 = edx/rdx * źródło	Wymaga BMI2		Unsigned multiply without affecting flags	Pomnóż bez znaku, bez zmiany flag	
Źródło								
IMUL	IMUL źródło	Reg/Mem	al/ax/eax/rax *= źródło Uwzględnia znak OF=0 i CF=0, jeśli iloczyn mieści się w akumulatorze		0xxxxC	Signed multiply	Pomnóż ze znakiem	
	Cel, źródło							
	IMUL cel, źródło	Reg, Reg/Mem (wszystkie 16/32/64 bit)	Cel *= źródło Uwzględnia znak OF=0 i CF=0, jeśli iloczyn mieści się w celu					
	Cel, źródło1, źródło2							
	IMUL cel, źródło1, źródło2	Reg, Reg/Mem, Stała (wszystkie 16/32/64 bit)	Cel = źródło1 * źródło2 (stała) Uwzględnia znak OF=0 i CF=0, jeśli iloczyn mieści się w celu					
Źródło								
DIV	DIV źródło	Reg/Mem	AL = AX / źródło, AH = AX % źródło LUB AX = DX:AX / źródło, DX = DX:AX % źródło LUB EAX = EDX:EAX / źródło, EDX = EDX:EAX % źródło LUB RAX = RDX:RAX / źródło, RDX = RDX:RAX % źródło Nie uwzględnia znaku	Oblicza jednocześnie iloraz i resztę xxxxx		Unsigned divide	Podziel bez znaku	
IDIV	IDIV źródło		To samo co wyżej, ale uwzględnia znak			Signed divide	Podziel ze znakiem	

Mnemonic	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Inne instrukcje							
Cel							
INC	INC cel	Reg/Mem	cel++		OSZAPx	Increment	Inkrementuj
DEC	DEC cel		cel--			Decrement	Dekrementuj
NEG	NEG cel		Cel = -cel (w kodzie U2)		OSZAPC	Negate U2	Zaneguj
Źródło1, źródło2							
CMP	CMP źródło1, źródło2	Reg, Mem	Porównaj źródło1 i źródło2 poprzez operację: <u>Źródło1 – źródło2</u> , odrzuć wynik, ustaw flagi		OSZAPC	Compare	Porównaj
		Mem, Reg					
		Reg, Reg					
Instrukcje BCD							
DAA	DAA	---	PDF str. 26-27	Nie działają w trybie 64 bitowym	OSZAPC	Decimal adjust after addition	Korekta <b>upakowanego</b> BCD po dodawaniu
DAS	DAS					Decimal adjust after subtraction	Korekta <b>upakowanego</b> BCD po odejmowaniu
AAA	AAA					ASCII adjust after addition	Korekta nieupakowanego BCD po dodawaniu
AAS	AAS					ASCII adjust after subtraction	Korekta nieupakowanego BCD po odejmowaniu
AAM	AAM					ASCII adjust after multiplication	Korekta nieupakowanego BCD po mnożeniu
AAD	AAD					ASCII adjust before division	Korekta nieupakowanego BCD <b>przed</b> dzieleniem

Mnemonik	Składnia	Operandy	Co robi	Zmieniane flagi	Uwagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje logiczne							

*Cel, źródło*

AND	AND cel, źródło	Reg, Reg	Cel = cel and źródło	0SZxP0			
OR	OR cel, źródło	Reg, Mem	Cel = cel or źródło				
XOR	XOR cel, źródło	Mem, Reg	Cel = cel xor źródło				

*Cel*

NOT	NOT cel	Reg/Mem	Cel = not cel	---			
-----	---------	---------	---------------	-----	--	--	--

*Cel, źródło1, źródło2*

ANDN	ANDN cel, źródło1, źródło2	Reg, Reg, Reg/Mem	Cel = (not źródło1) and źródło2	0SZxx0	Wymaga BMI1		
------	----------------------------	-------------------	---------------------------------	--------	-------------	--	--

### Instrukcje przesunięć

*Cel, ile*

SAR	SAR cel, ile	Reg/Mem, stała 1	Cel = cel >> ile (ze znakiem)	(0x)SZxPC	Flaga OF jest zmieniana tylko dla przesunięć o 1 bit w lewo/prawo. Jak zmieniana jest flaga CF: wykłady merged, strona 30	Shift arithmetic right	
SHR	SHR cel, ile	Reg/Mem, rejestr CL	Cel = cel >> ile (bez znaku)			Shift logical right	
SAL	SAL cel, ile	Reg/Mem, stała 0-31 lub 63	Cel = cel << ile (ze znakiem)			Shift arithmetic left	
SHL	SHL cel, ile		Cel = cel << ile (bez znaku)			Shift logical left	

*Cel, źródło, ile*

SARX	SARX cel, źródło, ile	Reg (32/64 bit), Reg/Mem, Reg (32/64 bit)	Cel = źródło >> ile (ze znakiem)	---	Wymaga BMI2	Shift arithmetic right, without affecting flags	
SHRX	SHRX cel, źródło, ile		Cel = źródło >> ile (bez znaku)			Shift logical right, without affecting flags	
SHLX	SHLX cel, źródło, ile		Cel = źródło << ile (bez znaku)			Shift logical left, without affecting flags	
SHRD	SHRD cel źródło, ile	Reg/Mem, Reg (16/32/64 bit), stała 0-31 lub 63	Przesunięcie źródła: celu o ile bitów w prawo Źródło zostaje bez zmian	(0x)SZxPC	OF i CF: patrz wyżej	Shift right with double precision	
SHLD	SHLD cel, źródło, ile	Reg/Mem, Reg (16/32/64 bit), rejestr CL	To samo co wyżej, ale przesuwu w lewo			Shift left with double precision	

### Instrukcje rotacji

*Cel, ile*

ROR	ROR cel, ile	Reg/Mem, stała 1	Obrazki – PDF str. 31-32	(0x)SZxPC	Flaga OF jest zmieniana tylko dla rotacji o 1 bit w lewo/prawo. Jak zmieniana jest flaga CF: połączone wykłady, strona 31-32	Rotate right	
ROL	ROL cel, ile	Reg/Mem, rejestr CL				Rotate left	
RCR	RCR cel, ile	Reg/Mem, stała 0-31 lub 63				Rotate through carry right	
RCL	RCL cel, ile					Rotate through carry left	
<i>Cel, źródło, ile</i>				---	Wymaga BMI2	Rotate right, without affecting flags	
RORX	RORX cel, źródło, ile	Reg (32/64 bit), Reg/Mem, stała 0-31 lub 63					

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje warunkowe							
CMOVcc	CMOVcc cel, źródło	Reg, Reg (oba 16/32/64 bit)	Cel = źródło, jeśli warunek jest spełniony		---	Conditional <b>move</b> if <b>cc</b> (condition code)	Warunkowe przesłanie jeśli warunek cc jest spełniony
		Reg, Mem (oba 16/32/64 bit)					
		Mem, Reg (oba 16/32/64 bit)					
Jcc	Jcc cel	Stała/Reg/Mem (wartość to adres względny) / etykieta	Skocz do adresu cel, jeśli warunek jest spełniony			Jump if <b>cc</b> (condition code)	Warunkowy skok jeśli warunek cc jest spełniony
Instrukcja skoku warunkowego – Jcc – możliwe warianty							

Cel

JE/JZ	JE/JZ cel	Stala (16/32 bit) Reg (16/32 bit) Mem (16/32 bit) Etykieta (Wartości interpretowane są jako adres względny)	Skocz jeśli ZF==1 (A == B)	Porównania dla liczb bez znaku (flagi CF, ZF)	---	Jump if equal/zero		
JNE/JNZ	JNE/JNZ cel		Skocz jeśli ZF==0 (A != B)				Jump if not equal/not zero	
JA/JNBE	JA/JNBE cel		Skocz jeśli CF==0 i ZF==0 (A > B) lub !(A <= B)				Jump if above / not below or equal	
JAE/JNB	JAE/JNB cel		Skocz jeśli CF==0 (A >= B) lub !(A < B)				Jump if above or equal/not below	
JB/JNAE	JB/JNAE cel		Skocz jeśli CF==1 (A < B) lub !(A >= B)			Jump if below / not above or equal		
JBE/JNA	JBE/JNA cel		Skocz jeśli CF==1 lub ZF==1 (A <= B) lub !(A > B)			Jump if below or equal / not above		
JG/JNLE	JG/JNLE cel		Skocz jeśli ZF==0 i SF==OF (A > B) lub !(A <= B)	Porównania dla liczb ze znakiem (flagi ZF, SF, OF)		Jump if greater / not less or equal		
JGE/JNL	JGE/JNL cel		Skocz jeśli SF==OF (A >= B) lub !(A < B)			Jump if greater or equal / not less		
JL/JNGE	JL/JNGE cel		Skocz jeśli SF!=OF (A < B) lub !(A >= B)			Jump if less / not greater or equal		
JLE/JNG	JLE/JNG cel		Skocz jeśli ZF==1 lub SF != OF (A <= B) lub !(A > B)			Jump if less or equal / not greater		
JC	JC cel			Skocz jeśli CF==1		Jump if carry		
JNC	JNC cel			Skocz jeśli CF==0		Jump if not carry		
JO	JO cel			Skocz jeśli OF==1		Jump if overflow		
JNO	JNO cel			Skocz jeśli OF==0		Jump if not overflow		
JS	JS cel			Skocz jeśli SF==1		Jump if sign		
JNS	JNS cel			Skocz jeśli SF==0		Jump if not sign		
JPO/JNP	JPO/JNP cel		Skocz jeśli PF==1	Jump if parity odd / not parity				
JPE/JP	JPE/JP cel		Skocz jeśli PF==0	Jump if parity even / parity				
JCXZ	JCXZ cel	Stala	Skocz jeśli CX==0	Warunki możliwe do zastosowania <b>tylko</b> dla skoku warunkowego (Jcc). SETcc i CMOVcc nie mogą z nich korzystać	Jump if cx is zero			
JECXZ	JECXZ cel	Reg	Skocz jeśli ECX==0		Jump if ecx is zero			
JRCXZ	JRCXZ cel	Mem (Wszystkie 16/32/64 bit) Etykieta (Wartości interpretowane są jako adres względny)	Skocz jeśli RCX==0		Jump if rcx is zero			

#### Instrukcja przesłania warunkowego – CMOVcc – możliwe warianty

Np. CMOVE, CMOVLE itd.. wszystkie jak powyżej (zaznaczone na czerwono), z wyjątkiem CXZ, ECXZ, RCXZ

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje skoku							
JMP	JMP cel	Stała/Reg/Mem (8/16/32 bit wszystkie) (Adres jest względny lub bezwzględny)	Skok bezwarunkowy		---		
LOOP	LOOP cel	Stała/Reg/Mem (8 bit) (Adres jest względny)	Skacz dopóki CX/ECX/RCX != 0				
LOOPZ/LOOPE	LOOPZ/LOOPE cel		To samo co LOOP + musi być: ZF=1				
LOOPNZ/LOOPNE	LOOPNZ/LOOPNE cel		To samo co LOOP + musi być: ZF=0				
Instrukcje związane z wywoływaniem podprogramów							
Brak/Adres/nr/ile							
CALL	CALL adres	Stała Reg Mem (wszystkie 16/32/64 bit) Etykieta (Wartości interpretowane są jako adres względny)	Wywołuje podprogram, wysyła na stos adres powrotu EIP/RIP lub CS:EIP/RIP Adres wpisuje do EIP/RIP lub CS:EIP/RIP		---		
RET	RET		Powrót z procedury, pobiera ze stosu adres powrotu do EIP/RIP lub CS:EIP/RIP			return	
	RET ile	Stała (8 bit), wartości 0-255	To samo co wyżej + usuwa <i>ile</i> bajtów ze stosu				
INT	INT nr	Stała (8 bit), wartości 0-255	Wywołuje przerwanie programowe o numerze <i>nr</i> Wysyła na stos flagi i część z nich (wchodząc do podprogramu) zeruje			interrupt	
INTO	INTO		Wywołuje przerwanie programowe nr 4 jeśli OF=1			interrupt if overflow	
IRET	IRET					Interrupt return	
IRETD	IRETD		Wraca z podprogramu obsługi przerwania Pobiera ze stosu adres powrotu do CS:EIP/RIP oraz flagi			Interrupt return dword	
IRETQ	IRETQ					Interrupt return qword	
BOUND	BOUND idx, gr	Reg, Mem (wszystkie 16/32 bit)	Sprawdza czy indeks w rejestrze <i>idx</i> nie przekracza zakresu, zdefiniowanego przez <i>gr</i> (pamięć). Generuje wyjątek przekroczenia granicy tablicy jeśli zakres jest przekroczony			bounds test	
Brak/rozmiar, poziom							
ENTER	ENTER rozmiar, poziom	Stała (16 bit), Stała (8 bit)	PDF str. 38-39				
LEAVE	LEAVE						

Mnemonik	Składnia	Operandy	Co robi	Zmieniane flagi	Uwagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje manipulacji flagami							
STC	STC	---	CF = 1	xxxxxC		Set carry	Ustaw flagę przeniesienia
CLC	CLC		CF = 0	xxxxxC		Clear carry	Wyczyść flagę przeniesienia
CMC	CMC		CF = not CF	xxxxxC		Complement carry	Zaneguj flagę przeniesienia
STD	STD		DF = 1	D		Set direction	Ustaw flagę kierunku
CLD	CLD		DF = 0	D		Clear direction	Wyczyść flagę kierunku
LAHF	LAHF		AH ← rejestr FLAGS	xSZAPC		Load to AH from flags (FLAGS)	Załaduj do rejestru AH rejestr FLAGS
SAHF	SAHF		rejestr FLAGS ← AH. Bity 1, 3 i 5 są ignorowane	xSZAPC		Store from AH to flags (FLAGS)	Zapisz do rejestru FLAGS rejestr AH
STI	STI		IF = 1 lub VIF = 1. Włącza przerwania maskowalne	I		Set interrupt	Ustaw flagę przerwań maskowalnych
CLI	CLI		IF = 0 lub VIF = 0. Wylłącza przerwania maskowalne	I		Clear interrupt	Wyczyść flagę przerwań maskowalnych
Instrukcje manipulacji bitami – testowanie i zmienianie							

*Baza, nr*

BT	BT baza, nr		CF = wartość bitu wartości <i>baza</i> na pozycji <i>nr</i>	xxxxxC		Bit test	
BTS	BTS baza, nr	Reg/Mem, Reg (wszystkie 16/32/64 bit)	CF = wartość bitu wartości <i>baza</i> na pozycji <i>nr</i> Ustaw bit na tej pozycji na <b>1</b>			Bit test & set	
BTR	BTR baza, nr	Reg/Mem (16/32/64 bit), Stała	CF = wartość bitu wartości <i>baza</i> na pozycji <i>nr</i> Ustaw bit na tej pozycji na <b>0</b>			Bit test & reset	
BTC	BTC baza, nr		CF = wartość bitu wartości <i>baza</i> na pozycji <i>nr</i> Ustaw bit na tej pozycji na <b>przeciwny do obecnego</b>			Bit test & complement	

*Arg1, arg2*

TEST	TEST arg1, arg2	Reg, Reg	arg1 and arg2. Odrzuca wynik, ustawia flagi	0SZxP0		Logical compare	Porównaj logicznie
------	-----------------	----------	---	--------	--	-----------------	--------------------

*Cel*

SETcc	SETcc cel	Reg/Mem (8 bit)	Jeśli cc to cel = 1 else cc = 0 Wpisuje do celu 1 jeśli warunek cc jest spełniony, wpisuje 0 w przeciwnym razie	---	Za cc (warunek) można podstawić wszystkie możliwe warunki (patrz arkusz Instrukcje warunkowe skoku) oprócz tych specyficznych dla Jcc (skok warunkowy)	conditional set byte	
-------	-----------	-----------------	--	-----	--	----------------------	--

Mnemonik	Składnia	Operandy	Co robi	Zmieniane flagi	Uwagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje manipulacji bitami – przeszukiwanie							

*Cel, źródło*

BSF	BSF cel, źródło	Reg, Reg	Szuka <b>najmłodszego</b> bitu = 1. Indeks umieszcza w <i>celu</i> (ZF=0) Jeśli <i>źródło</i> == 0 to ZF=1, a cel jest niezdefiniowany	xxZxxx	Działa jak TZCNT, z wyjątkiem 0 (TZCNT – 32, BSF – niezdefiniowane)	Bit scan forward	
BSR	BSR cel, źródło	Reg, Mem Mem, Reg	Szuka <b>najstarszego</b> bitu = 1. Indeks umieszcza w <i>celu</i> (ZF=0) Jeśli <i>źródło</i> == 0 to ZF=1, a cel jest niezdefiniowany			Bit scan reverse	
LZCNT	LZCNT cel, źródło	Reg, Reg/Mem (wszystkie 16/32/64 bit)	Zlicza liczbę bitów = 0, aż do napotkania bitu = 1 Wartość umieszcza w <i>celu</i> Zaczyna od <b>lewej</b> strony (najstarsze)	xxZxxC		Leading zero count	
TZCNT	TZCNT cel, źródło		Zlicza liczbę bitów = 0 aż do napotkania bitu = 1 Wartość umieszcza w <i>celu</i> Zaczyna od <b>prawej</b> strony (najmłodsze)			Trailing zero count	

#### Instrukcje manipulacji bitami – modyfikowanie ciągów bitów

*Cel, źródło, st\_ile*

BEXTR	BEXTR, cel, źródło st_ile	Reg, Reg/Mem, Reg (wszystkie 32/64 bit)	Wycina z <i>źródła</i> ciąg bitów i umieszcza w <i>celu</i> bity <i>st_ile</i> [7:0] – pozycja startowa, bity <i>st_ile</i> [15:8] – długość  Cel := źródło[start + ile – 1:start]	0xZxx0		Bit extract	
-------	------------------------------	--	--	--------	--	-------------	--

*Cel, źródło*

BLSI	BLSI cel, źródło		Izoluje najmłodszy bit = 1, (CF=1) i umieszcza w <i>celu</i> Jeśli <i>źródło</i> = 0 to CF=0, a <i>cel</i> = 0 Zeruje pozostałe bity  Cel := (~źródło) and źródło	0SZxxC		Bit lowest set isolate	
BLSR	BLSR cel, źródło	Reg, Reg/Mem (wszystkie 32/64 bit)	Izoluje najmłodszy bit = 1, (CF=1), umieszcza w <i>celu</i> , zeruje go w <i>źródle</i> Jeśli <i>źródło</i> == 0 to CF=0, a <i>cel</i> = 0  Cel := (źródło – 1) and źródło	0SZxxC		Bit lowest set reset	
BLSMK	BLSMK cel, źródło		Ustawia młodsze bity w <i>celu</i> na 1 aż do napotkania najmłodszego bitu = 1 w <i>źródle</i> (CF=0). Zeruje pozostałe. Jeśli <i>źródło</i> == 0 to CF=1, a <i>cel</i> = not 0  Cel := (źródło – 1) xor źródło	0S0xxC		Bit lowest set mask	

*Cel, źródło, idx*

BZHI	BZHI cel, źródło, idx	Reg, Reg/Mem, Reg (wszystkie 32/64 bit)	Kopiuje bity z <i>źródła</i> do <i>celu</i> . Kasuje starsze bity od <i>idx</i> (CF=0) Jeśli <i>idx</i> > 31 63 to CF=1  Cel := źródło; cel[rozmiar – 1:idx] = 0	0SZxxC		Bit zero high starting at index	
------	--------------------------	--	---	--------	--	---------------------------------	--



Mnemonik	Składnia	Operandy	Co robi	Zmieniane flagi	Uwagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje inne							
<i>Cel</i>							
RDPID	RDPID cel	Reg (32/64 bit)	Czyta 32 bitowy id procesora Cel=PID	---		Read processor identifier	
RDTSC	RDTSC	---	Czyta 64 bitowy licznik (cykle zegara CPU od czasu startu) EDX:EAX = licznik	---		Read time stamp counter	
<i>Cel</i>							
RDRAND	RDRAND cel	Reg (16/32/64 bit)	Czyta liczbę losową do celu wg normy NIST SP 800-90A Jeśli CF=1 wartość jest prawidłowa	0000C		Read random	
RDSEED	RDSEED cel		Czyta liczbę losową do celu wg normy NIST SP 800-90B i NIST SP800-90C Jeśli CF=1 wartość jest prawidłowa			Read random (seed)	

Mnemonik	Składnia	Przykład 1	Przykład 2
Instrukcje manipulacji flagami			
STC	STC		
CLC	CLC		
CMC	CMC		
STD	STD		
CLD	CLD		
LAHF	LAHF		
SAHF	SAHF		
STI	STI		
CLI	CLI		
Instrukcje manipulacji bitami – testowanie i zmienianie			
BT	BT baza, nr	Baza: 0001 1101 1111 1101 1101 1001 1100 0000 Nr = 4: CF = 0 ^Nr	
BTS	BTS baza, nr	Baza przed: 0001 1101 1111 1101 1101 1001 1100 0000 Baza po: 0001 1101 1111 1101 1101 1001 1101 0000 Nr = 4: CF = 0 ^Nr	
BTR	BTR baza, nr	Baza przed: 0001 1101 1111 1101 1101 1001 1100 0000 Baza po: 0001 1101 1111 1101 1101 1001 1000 0000 Nr = 4: CF = 0 ^Nr	Baza przed: 0001 1101 1111 1101 1101 1001 1100 0000 Baza po: 0001 1101 1111 1101 1101 1001 0000 0000 Nr = 6: CF = 1 ^Nr
BTC	BTC baza, nr	Baza przed: 0001 1101 1111 1101 1101 1001 1100 0000 Baza po: 0001 1101 1111 1101 1101 1001 1100 0001 Nr = 0: CF = 0 Nr^	
TEST	TEST arg1, arg2		
SETcc	SETcc cel		
Instrukcje manipulacji bitami – przeszukiwanie			
BSF	BSF cel, źródło	Źródło: 0001 1101 1111 1101 1101 1001 1100 0000 Cel: 6 ^ ZF=0	
BSR	BSR cel, źródło	Źródło: 0001 1101 1111 1101 1101 1001 1100 0000 Cel: 28 ^ ZF=0	
LZCNT	LZCNT cel, źródło	Źródło: 0001 1101 1111 1101 1101 1001 1100 0000 Cel: 3 ^^ ZF=0	
TZCNT	TZCNT cel, źródło	Źródło: 0001 1101 1111 1101 1101 1001 1100 0000 Cel: 6 ^^ ^^ ZF=0	

Mnemonic	Składnia	Przykład 1	Przykład 2
<b>Instrukcje manipulacji bitami – modyfikowanie ciągów bitów</b>			
BEXTR	BEXTR, cel, źródło st_ile	Źródło przed: 0001 1101 1111 1101 1101 1001 1100 0000 ^ st_ile[7:0] = 4, st_ile[15:8] = 7 Cel po:         0000 0000 0000 0000 0000 0000 0001 1100 ^ ^ ^ ^ ^ ^ ^ ^	
BLSI	BLSI cel, źródło	Źródło przed: 0001 1101 1111 1101 1101 1001 1100 0000 ^ Cel po:         0000 0000 0000 0000 0000 0000 0100 0000 ^	
BLSR	BLSR cel, źródło	Źródło przed: 0001 1101 1111 1101 1101 1001 1100 0000 ^ Źródło po:     0001 1101 1111 1101 1101 1001 1000 0000 Cel po:         0000 0000 0000 0000 0000 0000 0100 0000	
BLSMSK	BLSMSK cel, źródło	Cel przed:     0000 0000 0000 0000 0000 0000 1000 0101 Źródło przed: 0001 1101 1111 1101 1101 1001 1100 0000 ^ Cel po:         0000 0000 0000 0000 0000 0000 0111 1111 Źródło po:     0001 1101 1111 1101 1101 1001 1100 0000	
BZHI	BZHI cel, źródło, idx	Źródło przed: 0001 1101 1111 1101 1101 1001 1100 0000 ^ Idx = 8 Cel po:         0000 0000 0000 0000 0000 0000 1100 0000 ^ ^ ^ ^ ^ ^ ^ ^	
<b>Instrukcje inne</b>			
RDPID	RDPID cel		
RDTSO	RDTSO		
RDRAND	RDRAND cel		
RDSEED	RDSEED cel		

Mnemonik	Składnia	Operandy	Co robi	Zmieniane flagi	Uwagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcja przenoszenia – MOVS							
MOVS	MOVS <rozmiar> [(r e)di], [(r e)si]	Mem (di/edi/rdi), Mem (si/esi/rsi)	Przesyła z <b>adresu</b> SI/ESI/RSI do do <b>adresu</b> DI/EDI/RDI	---		Move string byte/word/dword/qword	
MOVSB	MOVSB	---	Rozmiar przesyłanego fragmentu zależy od instrukcji (MOVSB, MOVSQ itd.)			Move string byte	
MOVSW	MOVSW		Jeśli DF==0, to <b>zwiększa</b> oba rejestry o wartość rozmiaru danych (w bajtach) np. dla MOVSD będzie to edi+=4, esi+=4			Move string word	
MOVSD	MOVSD		Jeśli DF==1, to <b>zmniejsza</b> oba rejestry o wartość rozmiaru danych (w bajtach) np. dla MOVSD będzie to edi-=4, esi-=4			Move string dword	
MOVSQ	MOVSQ					Move string qword	
Instrukcja porównywania – CMPS							
CMPS	CMPS <rozmiar> [(r e)si], [(r e)di]	Mem (adres w SI/ESI/RSI), Mem (adres w DI/EDI/RDI)	Porównuje spod <b>pamięci</b> SI/ESI/RSI i DI/EDI/RDI Ustawia flagi	OSZAPC		Compare string byte/word/dword/qword	
CMPSB	CMPSB	---	Rozmiar porównywanego fragmentu zależy od instrukcji (CMPSB, CMPSQ itd.)			Compare string byte	
CMPSW	CMPSW		Jeśli DF==0, to <b>zwiększa</b> oba rejestry o wartość rozmiaru danych (w bajtach) np. dla CMPSD będzie to edi+=4, esi+=4			Compare string word	
CMPSD	CMPSD		Jeśli DF==1, to <b>zmniejsza</b> oba rejestry o wartość rozmiaru danych (w bajtach) np. dla CMPSD będzie to edi-=4, esi-=4			Compare string dword	
CMPSQ	CMPSQ					Compare string qword	
Instrukcja porównywania z akumulatorem – SCAS							
SCAS	SCAS <rozmiar> [(r e)di]	Mem (adres w DI/EDI/RDI)	Porównuje AL/AX/EAX/RAX i wartość spod adresu DI/EDI/RDI Ustawia flagi	OSZAPC		Scan string byte/word/dword/qword	
SCASB	SCASB	---	Rozmiar porównywanego fragmentu zależy od instrukcji (SCASB, SCASQ itd.)			Scan string byte	
SCASW	SCASW		Jeśli DF==0, to <b>zwiększa</b> oba rejestry o wartość rozmiaru danych (w bajtach) np. dla SCASD będzie to edi+=4, esi+=4			Scan string word	
SCASD	SCASD		Jeśli DF==1, to <b>zmniejsza</b> oba rejestry o wartość rozmiaru danych (w bajtach) np. dla SCASD będzie to edi-=4, esi-=4			Scan string dword	
SCASQ	SCASQ					Scan string qword	

Mnemonik	Składnia	Operandy	Co robi	Zmieniane flagi	Uwagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcja ładowania do akumulatora – LODS							
LODS	LODS <rozmiar> [(r e)si]	Mem (adres w SI/ESI/RSI)	Ładuje do AL/AX/EAX/RAX wartość spod adresu SI/ESI/RSI	---		Load string byte/word/dword/qword	
LODSB	LODSB	---	Rozmiar ładowanego fragmentu zależy od instrukcji (LODSB, LODSQ itd.)			Load string byte	
LODSW	LODSW		Jeśli DF==0, to <b>zwiększa</b> oba rejestry o wartość rozmiaru danych (w bajtach) np. dla LODSD będzie to edi+=4, esi+=4			Load string word	
LODSD	LODSD		Jeśli DF==1, to <b>zmniejsza</b> oba rejestry o wartość rozmiaru danych (w bajtach) np. dla LODSD będzie to edi-=4, esi-=4			Load string dword	
LODSQ	LODSQ					Load string qword	
Instrukcja zapisu z akumulatora – STOS							
STOS	STOS <rozmiar> [(r e)di]	Mem (adres w DI/EDI/RDI)	Zapisuje spod AL/AX/EAX/RAX do adresu DI/EDI/RDI	---		Store string byte/word/dword/qword	
STOSB	STOSB	---	Rozmiar zapisywanego fragmentu zależy od instrukcji (STOSB, STOSQ itd.)			Store string byte	
STOSW	STOSW		Jeśli DF==0, to <b>zwiększa</b> oba rejestry o wartość rozmiaru danych (w bajtach) np. dla STOSD będzie to edi+=4, esi+=4			Store string word	
STOSD	STOSD		Jeśli DF==1, to <b>zmniejsza</b> oba rejestry o wartość rozmiaru danych (w bajtach) np. dla STOSD będzie to edi-=4, esi-=4			Store string dword	
STOSQ	STOSQ					Store string qword	
Prefiks REP, REPNZ/REPNE, REPZ/REPE							
REP	Powoduje powtórzenie CX/ECX/RCX razy następującej po niej instrukcji łańcuchowej. Jeśli CX/ECX/RCX==0 to nie zostanie wykonana			---			
REPNZ/REPNE	Jak wyżej + może przerwać wcześniej, gdy ZF nie już == 0 (czyli ZF==1)						
REPZ/REPE	Jak wyżej + może przerwać wcześniej gdy ZF nie już == 1 (czyli ZF==0)						

Mnemonik	Składnia	Operandy	Co robi	Zmieniane flagi	Uwagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje ładowania do rejestrów segmentowych							
Cel, źródło							
LDS	LDS cel, źródło	Reg (16/32 bit), Mem (16/32 bit)	Wczytuje do pary rejestrów DS:cel wskaźnik do źródła	---		load far pointer, <b>d</b> ata segment	
LES	LES cel, źródło		Wczytuje do pary rejestrów ES:cel wskaźnik do źródła			load far pointer, extra segment	
LFS	LFS cel, źródło		Wczytuje do pary rejestrów FS:cel wskaźnik do źródła			load far pointer, <b>F</b> segment	
LGS	LGS cel, źródło		Wczytuje do pary rejestrów GS:cel wskaźnik do źródła			load far pointer, <b>G</b> segment	
LSS	LSS cel, źródło		Wczytuje do pary rejestrów SS:cel wskaźnik do źródła			load far pointer, stack segment	
Prefiks LOCK							
LOCK	Powoduje niepodzielne wykonanie instrukcji Można zastosować tylko do następujących instrukcji:						
przesyłania:	XADD, XCHG, CMPXCHG, CMPXCHG8B, CMPXCHG16B				XCHG posiada już domyślnie prefiks LOCK		
arytmetyczne:	ADD, ADC, INC, DEC, NOT, SUB, SBB						
logiczne	AND, NOT, OR, XOR						
Inne instrukcje							
Cel, źródło							
LEA	LEA cel, źródło	Reg, Mem	Ładuje <b>adres źródła</b> (nie wartość spod adresu) do <i>celu</i>	---		Load <b>e</b> ffective <b>a</b> ddress	Załaduj efektywny adres
MOVBE	MOVBE cel, źródło	Reg, Reg Reg, Mem Mem, Reg (wszystkie 16/32/64 bit)	Cel = zamierniKolejnośćBajtów(źródło) Działa jak instrukcje MOV + BSWAP			Move byte <b>e</b> xchange	
CPUID	CPUID	---	Na podstawie EAX (czasem ECX) podaje informacje o procesorze w EAX, EBX, ECD, EDX Identyfikacja procesora jest możliwa jeśli bit 21 może być zmieniany w rejestrze flag	---			
XGETBV	XGETBV		EDX:EAX = zawartość rozszerzonego rejestru kontrolnego o indeksie ECX				
XLAT	XLAT		Tłumaczenie w oparciu o tablicę translacji			Look-up table translation	
XLATB	XLATB		AL = DS:[BX/EBX/RBX+AL]			Look-up table translation <b>b</b> yte	
NOP	NOP		Nic			No <b>o</b> peration	
UD2	UD2		Wyrzuca wyjątek <i>instrukcja niezdefiniowana</i> Wprowadzona do testów			Undefined instruction	

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol	
Instrukcje przesyłania danych								
Źródło								
FLD	FLD źródło	st(i)/Mem (32/64 bit)	Ładuje na wierzchołek stosu z pamięci float/double lub st(i)		---	Load		
Cel								
FST	FST cel	st(i)/Mem (32/64 bit)	Zapisuje wartość wierzchołka stosu do pamięci jako float/double lub do st(i)			Store		
FSTP	FSTP cel		To samo co wyżej + zdejmuje wartość w st ze stosu			Store & pop from stack		
Źródło								
FILD	FILD źródło	Mem (16/32/64 bit)	Konwertuje (na format x87) i ładuje na wierzchołek stosu liczbę całkowitą			Integer load		
Cel								
FIST	FIST cel	Mem (16/32/64 bit)	Zapisuje wartość wierzchołka stosu do pamięci jak liczbę całkowitą (ucina wartość ułamkową)			Integer store		
FISTP	FISTP cel		To samo co wyżej + zdejmuje wartość w st ze stosu			Integer store & pop from stack		
Źródło								
FBLD	FBLD źródło	Mem (obszar 10 bajtów)	Konwertuje (na format x87) i ładuje na wierzchołek stosu 80 bitową liczbę BCD		BCD load			
Cel								
FBSTP	FBSTP cel	Mem (obszar 10 bajtów)	Zapisuje wartość wierzchołka stosu do pamięci jak 80 bitową liczbę BCD Zdejmuje wartość w st ze stosu		BCD store & pop from stack			
FXCH	FXCH st(i)	st(i)	Zamienia miejscami wartość w st i st(i)		Exchange			
	FXCH	---	Zamienia miejscami wartość w st i st(1)					
FCMOVcc	FCMOVcc st, st(i)	st, st(i)	Przesyła wartość st(i) do st (wierzchołek) jeśli warunek cc jest spełniony		Conditional move if cc(condition code)	Warunkowe przesłanie jeśli warunek cc jest spełniony		

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
----------	----------	----------	---------	-------	-----------------	-------------------------------------	-------------------------------------

### Instrukcja przesłania warunkowego – FCMOVcc – możliwe warianty

FCMOV <del>E</del>	FCMOV <del>E</del> st, st(i)	st, st(i)	Patrz FCMOVcc, robi to jeśli st == st(i)	Instrukcje FCMOVcc mają wersje tylko z literami E i B tj. nie ma np. porównania Above Equal + porównania FCOM są wykonywane jak dla liczb bez znaku	---	Conditional <b>move</b> if equal	
FCMOV <del>NE</del>	FCMOV <del>NE</del> st, st(i)		Patrz FCMOVcc, robi to jeśli st ≠ st(i)			Conditional <b>move</b> if not equal	
FCMOV <del>B</del>	FCMOV <del>B</del> st, st(i)		Patrz FCMOVcc, robi to jeśli St < st(i)			Conditional <b>move</b> if below	
FCMOV <del>BE</del>	FCMOV <del>BE</del> st, st(i)		Patrz FCMOVcc, robi to jeśli St ≤ st(i)			Conditional <b>move</b> if below or equal	
FCMOV <del>NB</del>	FCMOV <del>NB</del> st, st(i)		Patrz FCMOVcc, robi to jeśli !(st < st(i)), czyli st ≥ st(i)			Conditional <b>move</b> if not below	
FCMOV <del>NBE</del>	FCMOV <del>NBE</del> st, st(i)		Patrz FCMOVcc, robi to jeśli !(st ≤ st(i)), czyli st > st(i)			Conditional <b>move</b> if not below or equal	

### Tryby adresowania w instrukcjach arytmetycznych

Stosowe: <i>Fop</i> Operandy niejawnie st(1) i st	Rejestrowe 1: <i>Fop</i> st, st(i) Rejestrowe 2: <i>Fop</i> st(i), st Operandy st, st(i) lub st(i), st	Rejestrowe ze zdjęciem ze stosu: <i>FopP</i> st(i), st Operandy st(i), st	Z argumentem całkowitym w pamięci: <i>Flop</i> źródło Operandy: niejawnie st, pamięć
--	--	--	---

### Instrukcje arytmetyczne – FADD

Brak / Cel i źródło / Źródło

FADD	FADD	---	St(1) = st(1) + st Zdejmij wartość w st	---		
	FADD st(i), st	st(i), st	St(i) = st(i) + st			
	FADD st, st(i)	st, st(i)	St = st + st(i)			
	FADD źródło	Mem (32/64 bit) (float/double)	St = st + źródło			
FADDP	FADDP st(i), st	st(i), st	St(i) = st(i) + st Zdejmij wartość w st			
FIADD	FIADD źródło	Mem (16/32 bit)(liczba całkowita)	St = st + źródło			



Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje arytmetyczne – FSUB							

Brak / Cel i źródło / Źródło

FSUB	FSUB	---	$St(1) = st(1) - st$ Zdejmij wartość w st		---		
	FSUB st(i), st	st(i), st	$St(i) = st(i) - st$				
	FSUB st, st(i)	st, st(i)	$St = st - st(i)$				
	FSUB źródło	Mem (32/64 bit) (float/double)	$St = st - \text{źródło}$				
FSUBP	FSUBP st(i), st	st(i), st	$St(i) = st(i) - st$ Zdejmij wartość w st				
FISUB	FISUB źródło	Mem (16/32 bit)(liczba całkowita)	$St = st - \text{źródło}$				

#### Instrukcje arytmetyczne – FSUBR

Brak / Cel i źródło / Źródło

FSUBR	FSUBR	---	$St(1) = st - st(1)$ Zdejmij wartość w st	W operacji <b>zamienia</b> miejscami argumenty, w stosunku do tego jak „normalnie” by było obliczone np. Normalnie a i b: a-b FSUBR a i b: b-a	---		
	FSUBR st(i), st	st(i), st	$St(i) = st - st(i)$				
	FSUBR st, st(i)	st, st(i)	$St = st(i) - st$				
	FSUBR źródło	Mem (32/64 bit) (float/double)	$St = \text{źródło} - st$				
FSUBRP	FSUBRP st(i), st	st(i), st	$St(i) = st - st(i)$ Zdejmij wartość w st				
FISUBR	FISUBR źródło	Mem (16/32 bit)(liczba całkowita)	$St = \text{źródło} - st$				

#### Instrukcje arytmetyczne – FMUL

Brak / Cel i źródło / Źródło

FMUL	FMUL	---	$St(1) = st(1) * st$ Zdejmij wartość w st		---		
	FMUL st(i), st	st(i), st	$St(i) = st(i) * st$				
	FMUL st, st(i)	st, st(i)	$St = st * st(i)$				
	FMUL źródło	Mem (32/64 bit) (float/double)	$St = st * \text{źródło}$				
FMULP	FMULP st(i), st	st(i), st	$St(i) = st(i) * st$ Zdejmij wartość w st				
FIMUL	FIMUL źródło	Mem (16/32 bit)(liczba całkowita)	$St = st * \text{źródło}$				

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje arytmetyczne – FDIV							

Brak / Cel i źródło / Źródło

FDIV	FDIV	---	$St(1) = st(1) / st$ Zdejmij wartość w st		---		
	FDIV st(i), st	st(i), st	$St(i) = st(i) / st$				
	FDIV st, st(i)	st, st(i)	$St = st / st(i)$				
	FDIV źródło	Mem (32/64 bit) (float/double)	$St = st / \text{źródło}$				
FDIVP	FDIVP st(i), st	st(i), st	$St(i) = st(i) / st$ Zdejmij wartość w st				
FIDIV	FIDIV źródło	Mem (16/32 bit)(liczba całkowita)	$St = st / \text{źródło}$				

### Instrukcje arytmetyczne – FDIVR

Brak / Cel i źródło / Źródło

FDIVR	FDIVR	---	$St(1) = st / st(1)$ Zdejmij wartość w st	W operacji <b>zamienia</b> miejscami argumenty, w stosunku do tego jak „normalnie” by było obliczone np. Normalnie a i b: a/b FDIVR a i b: b/a	---		
	FDIVR st(i), st	st(i), st	$St(i) = st / st(i)$				
	FDIVR st, st(i)	st, st(i)	$St = st(i) / st$				
	FDIVR źródło	Mem (32/64 bit) (float/double)	$St = \text{źródło} / st$				
FDIVRP	FDIVRP st(i), st	st(i), st	$St(i) = st / st(i)$ Zdejmij wartość w st				
FIDIVR	FIDIVR źródło	Mem (16/32 bit)(liczba całkowita)	$St = \text{źródło} / st$				

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje inne #1							
FPREM	FPREM	---	St = st % st(1) (reszta z dzielenia) Jeśli wartość jest zbyt duża, to C2=1 i instrukcję należy powtórzyć Zakres reszty: <- st(1) ,  st(1) >		C2 (flagi x87)		
FPREM1	FPREM1		St = st % st(1) (reszta z dzielenia) Jeśli wartość jest zbyt duża, to C2=1 i instrukcję należy powtórzyć Zakres reszty: <- st(1)/2 ,  st(1)/2 >				
FABS	FABS		St =  st		---		
FCHS	FCHS		St = - st				
FRNDINT	FRNDINT		St = round(st)				
FSCALE	FSCALE		St = st*2^int(st(1)) Skaluje przez potęgę 2				
FSQRT	FSQRT		St = sqrt(st)				
FXTRACT	FXTRACT		Oblicza wykładnik i mantysę z st Stan stosu: Temp = st 0. mantysa(temp) 1. wykładnik(temp)				
Instrukcje ładowania stałych							
FLD1	FLD1	---	St = +1.0		---	Load 1	
FLDZ	FLDZ		St = +0.0			Load zero	
FLDPI	FLDPI		St = Pi			Load Pi	
FLDL2E	FLDL2E		St = Log <sub>2</sub> (e)			Load Log 2 E	
FLDLN2	FLDLN2		St = Log <sub>e</sub> (2) (ln(2))			Load Ln 2	
FLDL2T	FLDL2T		St = Log <sub>2</sub> (10)				
FLDLG2	FLDLG2		St = Log <sub>10</sub> (2)				

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje funkcji przestępnych							
FSIN	FSIN	---	St = sin(st) Jeśli st poza zakresem to C2=1. Argument można zredukować FPREM z dzielnikiem 2Pi  <i>Zakres argumentu: &lt;-2^63, 2^63&gt;</i>		C2 (flagi <b>x87</b> )	Sine	
FCOS	FCOS		St = cos(st) Jeśli st poza zakresem to C2=1. Argument można zredukować FPREM z dzielnikiem 2Pi  <i>Zakres argumentu: &lt;-2^63, 2^63&gt;</i>			Cosine	
FSINCOS	FSINCOS		Sin(st) i Cos(st) jednocześnie Stan stosu: Temp = st 0. cos(temp) 1. sin(temp)  Jeśli st poza zakresem to C2=1. Argument można zredukować FPREM z dzielnikiem 2Pi  <i>Zakres argumentu: &lt;-2^63, 2^63&gt;</i>			Sine & cosine	
FPTAN	FPTAN		Tan(st). Wrzuca dodatkowo 1.0 na wierzchołek Stan stosu: Temp = tan(st) 0. 1.0 1. tan(temp)  <i>Zakres argumentu: &lt;-2^63, 2^63&gt;</i>			Partial <b>tan</b> gent	
FPATAN	FPATAN		St(1) = arctan(st(1)/st). Zdejmuje st ze stosu  <i>Zakres argumentów: R (dowolny)</i>		---	Partial arcus <b>tan</b> gent	
F2XM1	F2XM1		St = (2^st) – 1  <i>Zakres argumentu: &lt;-1, 1&gt;</i>	Pamiętać o tym że dodatkowo odejmuje 1 od wyniku!		2 to x power, <b>minus 1</b>	
FYL2X	FYL2X		St = y*log <sub>2</sub> (x), gdzie x==st, y==st(1)  <i>Zakres argumentu x: (0, inf)</i> <i>Zakres argumentu y: R</i>			Y * Log 2 X	
FYL2XP1	FYL2XP1		St = y*log <sub>2</sub> (x + 1), gdzie x==st, y==st(1)  <i>Zakres argumentu x: (-1, inf)</i> <i>Zakres argumentu y: R</i>			Y * Log 2 X plus 1	

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje porównywania							
Brak / Źródło							
FCOM	FCOM źródło	st(i)/Mem (32/64 bit)	Porównuje st i źródło. Ustawia flagi.		C3, C2, C0 (flagi x87)	Compare	
	FCOM	---	Porównuje st i st(1). Ustawia flagi.				
FCOMP	FCOMP źródło	st(i)/Mem (32/64 bit)	Porównuje st i źródło. Ustawia flagi. Zdejmuje st ze stosu			Compare & pop from stack	
	FCOMP	---	Porównuje st i st(1). Ustawia flagi. Zdejmuje st ze stosu				
FCOMPP	FCOMPP	---	Porównuje st i st(1). Ustawia flagi. Zdejmuje st i st(1) ze stosu			Compare & pop twice (pop pop) from stack	
FUCOM	FUCOM źródło	st(i)/Mem (32/64 bit)	To samo co FCOM, <b>ale</b> nie zgłasza wyjątku #IA dla nieliczb pasywnych (QNaN)			Unordered compare	
	FUCOM						
FUCOMP	FUCOMP źródło	st(i)/Mem (32/64 bit)	To samo co FCOMP, <b>ale</b> nie zgłasza wyjątku #IA dla nieliczb pasywnych (QNaN)			Unordered compare & pop from stack	
	FUCOMP	---					
FUCOMPP	FUCOMPP	---	To samo co FCOMPP, <b>ale</b> nie zgłasza wyjątku #IA dla nieliczb pasywnych (QNaN)			Unordered compare & pop twice (pop pop) from stack	
FICOM	FICOM źródło	Mem (16/32 bit) (liczba całkowita)	Porównuje st i liczbę całkowitą (źródło) w pamięci. Ustawia flagi		Integer compare		
FICOMP	FICOMP źródło		Porównuje st i liczbę całkowitą (źródło) w pamięci. Ustawia flagi. Zdejmuje st ze stosu			Integer compare & pop from stack	
FCOMI	FCOMI st(i)	st(i)	To samo co FCOM, <b>ale</b> ustawia <b>flagi x86</b>	Wersje z FCOM i FUCOM z I na końcu (ustawiają flagi x86) nie mają opcjonalnego operandu (wtedy byłoby to st(1)) + akceptują tylko rejestry koprocesora	ZF, PF, CF (flagi x86) xxZxPC	Compare & set x86 flags	
FCOMIP	FCOMIP st(i)		To samo co FCOMP, <b>ale</b> ustawia <b>flagi x86</b>			Compare & set x86 flags & pop from stack	
FUCOMI	FUCOMI st(i)		To samo co FUCOM, <b>ale</b> ustawia <b>flagi x86</b>			Unordered compare & set x86 flags	
FUCOMIP	FUCOMIP st(i)		To samo co FUCOMP, <b>ale</b> ustawia <b>flagi x86</b>			Unordered compare & set x86 flags & pop from stack	
FTST	FTST	---	Porównuje st i 0.0. Ustawia flagi		C3, C2, C0 (flagi x87)	Test	
FXAM	FXAM		Sprawdza typ liczby w st. Ustawia flagi		C3, C2, C1, C0 (flagi x87)	Examine	

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Rozwinięcie skrótu / Operacja – ang	Rozwinięcie skrótu / Operacja – pol
Instrukcje sterowania koprocesorem							
FINCSTP	FINCSTP		Zwiększa wskaźnik stosu koprocesora o 1 Top := (top + 1) mod 8		---	Increment stack top pointer	
FDECSTP	FDECSTP		Zmniejsza wskaźnik stosu koprocesora o 1 Top := (top – 1) mod 8			Decrement stack top pointer	
		Cel					
FFREE	FFREE st(i)	st(i)	Zwalnia rejestr. Rejestr wskaźnika stosu nie jest zmieniany tag(i) := 11 b				
FINIT	FINIT		Inicjalizacja koprocesora				
FNINIT	FNINIT		To co wyżej, ale nie sprawdza zgłoszenia błędu numerycznego				
FCLEX	FCLEX		Zerowanie flag błędów numerycznych			Clear exceptions	
FNCLEX	FNCLEX		To co wyżej, ale nie sprawdza zgłoszenia błędu numerycznego				
		Cel/źródło					
FSTCW	FSTCW cel	Mem (obszar 2 bajtów)	Zapamiętanie rejestru sterowania			Store control word	
FNSTCW	FNSTCW cel	Rejestr AX	To co wyżej, ale nie sprawdza zgłoszenia błędu numerycznego				
FLDCW	FLDCW źródło	Mem (obszar 2 bajtów)	Wczytanie rejestru sterowania			Load control word	
FSTENV	FSTENV cel	Mem (obszar 14 bajtów) (tryb 16 bit)	Zapamiętanie środowiska koprocesora			Store environment	
FNSTENV	FNSTENV cel	Mem (obszar 28 bajtów) (tryb 32 bit)	To co wyżej, ale nie sprawdza zgłoszenia błędu numerycznego				
FLDENV	FLDENV źródło	Mem (obszar 14 bajtów) (tryb 16 bit) Mem (obszar 28 bajtów) (tryb 32 bit)	Wczytanie środowiska koprocesora			Load environment	
FSAVE	FSAVE cel	Mem (obszar 94 bajtów) (tryb 16 bit)	Zapamiętanie zawartości koprocesora (środowisko + rejestry st(i))				
FNSAVE	FNSAVE cel	Mem (obszar 108 bajtów) (tryb 32 bit)	To co wyżej, ale nie sprawdza zgłoszenia błędu numerycznego				
FRSTOR	FRSTOR źródło	Mem (obszar 94 bajtów) (tryb 16 bit) Mem (obszar 108 bajtów) (tryb 32 bit)	Wczytanie zawartości koprocesora			Restore	
FSTSW	FSTSW cel	Mem (obszar 2 bajtów)	Zapamiętanie rejestru stanu				
FNSTSW	FNSTSW cel	Rejestr AX	To co wyżej, ale nie sprawdza zgłoszenia błędu numerycznego			Store state word	
WAIT/FWAIT	WAIT/FWAIT		Czekanie przez procesor na gotowość koprocesora				
FNOP	FNOP		Nic				

Mnemonic	Składnia	Przykład – operacje	Przykład – stos/pamięć przed operacją	Przykład – stos/pamięć po operacji
Instrukcje przesyłania danych				
FLD	FLD źródło	FLD qword ptr [rdx]	(pusty stos)	0. fromDouble([rdx])
FST	FST cel	FST dword ptr [rdx]	0. 35.414	0. 35.414 Pamięć: [rdx] = toFloat(st(0)) [rdx] == 35.414f
FSTP	FSTP cel	FST dword ptr [rdx]	0. 35.414	(pusty stos) Pamięć: [rdx] = toFloat(st(0)) [rdx] == 35.414f
FILD	FILD źródło	FILD word ptr [rdx]	(pusty stos)	0. fromInt16([rdx])
FIST	FIST cel	FIST dword ptr [rdx]	0. 35.414	0. 35.414 Pamięć: [rdx] = toInt32(st(0)) [rdx] == 35.000
FISTP	FISTP cel	FISTP dword ptr [rdx]	0. 35.414	(pusty stos) Pamięć: [rdx] = toInt32(st(0)) [rdx] == 35.000
FBLD	FBLD źródło	FBLD [rdx]	(pusty stos)	0. fromBCD([rdx])
FBSTP	FBSTP cel	FBSTP [rdx]	0. 35.414	(pusty stos) Pamięć: [rdx] = toBCD(st(0)) [rdx] == 35.414 (w formie BCD :P)
FXCH	FXCH st(i)	FXCH st(2)	0. x 1. y 2. z	0. z 1. y 2. x
	FXCH	FXCH	0. x 1. y 2. z	0. y 1. x 2. z
FCMOVcc	FCMOVcc st, st(i)	FCOM st(2) FCMOVbe st, st(2) (54.9 <= 42?)	0. 54.9 1. 341 2. 42	0. 42 1. 341 2. 42

Mnemonik	Składnia	Przykład – operacje	Przykład – stos/pamięć przed operacją	Przykład – stos/pamięć po operacji
Instrukcje funkcji przestępnych				
FSIN	FSIN			
FCOS	FCOS			
FSINCOS	FSINCOS	FSINCOS	0. x	0. cos(x) 1. sin(x)
FPTAN	FPTAN		0. x	0. 1.0 1. tan(x)
FPATAN	FPATAN		0. x 1. y	0. arctan(y/x)
F2XM1	F2XM1	F2XM1	0. 0.5	0. (2 <sup>0.5</sup> ) - 1
FYL2X	FYL2X		0. x 1. y	0. y*log <sub>2</sub> (x) 1. y
FYL2XP1	FYL2XP1		0. x 1. y	0. y*log <sub>2</sub> (x + 1) 1. y



Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Budowa mnemonika
Instrukcje przesyłania						
		Cel, źródło			---	
MOVD	MOVD cel, źródło	MM, MM/Mem/Reg (32 bit)	PDF, str. 80	Bity 32-63 są wypełniane zerami		MOV_D
MOVQ	MOVQ cel, źródło	MM, MM/Mem/Reg (64 bit)				MOV_Q
Instrukcje konwersji – pakowanie						
PACKSSWB	PACKSSWB cel, źródło	MM, MM/Mem	PDF, str. 81-83	Pakowanie jest tylko z nasyceniem	---	PACK_S_S_W_B
PACKSSDW	PACKSSDW cel, źródło					PACK_S_S_D_W
PACKUSWB	PACKUSWB cel, źródło					PACK_U_S_W_B
Instrukcje konwersji – rozpakowanie z przeplotem						
PUNPCKHBW	PUNPCKHBW cel, źródło	MM, MM/Mem	PDF, str. 84-87	Rozpakowanie młodsze/starsze warianty: >B → W, >W → D, >D → Q	---	PUNPCK_H_B_W
PUNPCKHWD	PUNPCKHWD cel, źródło					PUNPCK_H_W_D
PUNPCKHDQ	PUNPCKHDQ cel, źródło					PUNPCK_H_D_Q
PUNPCKLBW	PUNPCKLBW cel, źródło					PUNPCK_L_B_W
PUNPCKLWD	PUNPCKLWD cel, źródło					PUNPCK_L_W_D
PUNPCKLDQ	PUNPCKLDQ cel, źródło					PUNPCK_L_D_Q
Instrukcje arytmetyczne – dodawanie						
PADDB	PADDB cel, źródło	MM, MM/Mem	PDF, str. 88-90	>ze znakiem, bez nasycenia: <b>B, W, D</b> >ze znakiem i nasyceniem: <b>B, W</b> >bez znaku i z nasyceniem: <b>B, W</b> >nie uwzględnia przeniesienia	---	P_ADD_B
PADDW	PADDW cel, źródło					P_ADD_W
PADDD	PADDD cel, źródło					P_ADD_D
PADDSB	PADDSB cel, źródło					P_ADD_S_B
PADDSW	PADDSW cel, źródło					P_ADD_S_W
PADDUSB	PADDUSB cel, źródło					P_ADD_U_S_B
PADDUSW	PADDUSW cel, źródło					P_ADD_U_S_W

Rozwinięcie skrótów / Operacja – ang	Rozwinięcie skrótów / Operacja – pol
<b>Move dword</b>	
<b>Move qword</b>	
<b>Pack signed with saturation words to bytes</b>	
<b>Pack signed with saturation dwords to words</b>	
<b>Pack unsigned with saturation words to bytes</b>	
<b>Unpack interleaved high bytes to words</b>	
<b>Unpack interleaved high words to dwords</b>	
<b>Unpack interleaved high dwords to qword</b>	
<b>Unpack interleaved low bytes to words</b>	
<b>Unpack interleaved low words to dwords</b>	
<b>Unpack interleaved low dwords to qword</b>	
<b>Add bytes</b>	
<b>Add words</b>	
<b>Add dwords</b>	
<b>Add with saturation bytes</b>	
<b>Add with saturation words</b>	
<b>Add unsigned with saturation bytes</b>	
<b>Add unsigned with saturation bytes</b>	

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Budowa mnemonika
Instrukcje arytmetyczne – odejmowanie						
PSUBB	PSUBB cel, źródło	MM, MM/Mem	PDF, str. 91-93	>ze znakiem, bez nasycenia: <b>B, W, D</b> >ze znakiem i nasyceniem: <b>B, W</b> >bez znaku i z nasyceniem: <b>B, W</b> >nie uwzględnia przeniesienia	---	P_SUB_B
PSUBW	PSUBW cel, źródło					P_SUB_W
PSUBD	PSUBD cel, źródło					P_SUB_D
PSUBSB	PSUBSB cel, źródło					P_SUB_S_B
PSUBSW	PSUBSW cel, źródło					P_SUB_S_W
PSUBUSB	PSUBUSB cel, źródło					P_SUB_U_S_B
PSUBUSW	PSUBUSW cel, źródło					P_SUB_U_S_W
Instrukcje arytmetyczne – mnożenie						
PMULHW	PMULHW cel, źródło	MM, MM/Mem	PDF, str. 94-96		---	P_MUL_H_W
PMULLW	PMULLW cel, źródło			P_MUL_L_W		
PMADDWD	PMADDWD cel, źródło			P_M_ADD_W_D		
Instrukcje porównywania						
PCMPEQB	PCMPEQB cel, źródło	MM, MM/Mem	PDF, str. 97-99	>Tylko warianty równy (A==B) i większy niż (A > B) >wynik porównania (0/1) jest zapisywany w rejestrze celu, dla każdego elementu	---	P_CMP_EQ_B
PCMPEQW	PCMPEQW cel, źródło					P_CMP_EQ_W
PCMPEQD	PCMPEQD cel, źródło					P_CMP_EQ_D
PCMPGTB	PCMPGTB cel, źródło					P_CMP_GT_B
PCMPGTW	PCMPGTW cel, źródło					P_CMP_GT_W
PCMPGTD	PCMPGTD cel, źródło					P_CMP_GT_D

Rozwinięcie skrót / Operacja – ang	Rozwinięcie skrót / Operacja – pol
<b>Sub</b> bytes	
<b>Sub</b> words	
<b>Sub</b> dwords	
<b>Sub</b> with saturation bytes	
<b>Sub</b> with saturation words	
<b>Sub</b> unsigned with saturation bytes	
<b>Sub</b> unsigned with saturation bytes	
<b>Multiply</b> , keep high words	
<b>Multiply</b> , keep low words	
<b>Multiply</b> and <b>add</b> words to dwords	
<b>Compare</b> equal bytes	
<b>Compare</b> equal words	
<b>Compare</b> equal dwords	
<b>Compare</b> greater than bytes	
<b>Compare</b> greater than words	
<b>Compare</b> greater than dwords	

Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Budowa mnemonika
Instrukcje logiczne						
PAND	PAND cel, źródło	MM, MM/Mem	PDF, str. 100-101		---	P_AND
PANDN	PANDN cel, źródło					P_ANDN
POR	POR cel, źródło					P_OR
PXOR	PXOR cel, źródło					P_XOR
Instrukcje przesunięć bitowych						
		Cel, ile			---	
PSLLW	PSLLW cel, ile	MM, Mem/stała	PDF, str. 102-105	Logiczne – lewo/prawo W, D, Q		P_S_L_L_W
PSLLD	PSLLD cel, ile					P_S_L_L_D
PSLLQ	PSLLQ cel, ile					P_S_L_L_Q
PSRLW	PSRLW cel, ile					P_S_R_L_W
PSRLD	PSRLD cel, ile					P_S_R_L_D
PSRLQ	PSRLQ cel, ile					P_S_R_L_Q
PSRAW	PSRAW cel, ile			Arytmetyczne – tylko w prawo W, D		P_S_R_A_W
PSRAD	PSRAD cel, ile					P_S_R_A_D
Instrukcje sterujące						
		Brak/Cel/źródło			---	
FXSAVE	FXSAVE cel	Mem (obszar 512 bajtów)	PDF, str. 106-109			FX_SAVE
FXRSTOR	FXRSTOR źródło					FX_RSTOR
EMMS	EMMS			E_MM_S		
LDMXCSR	LDMXCSR cel	Mem (obszar 4 bajtów)				LD_MXCSR
STMXCSR	STMXCSR źródło					ST_MXCSR



Mnemonik	Składnia	Operandy	Co robi	Uwagi	Zmieniane flagi	Budowa mnemonika
Instrukcje wprowadzone z SSE – średnia						
		Cel, źródło			---	
PAVGB	PAVGB cel, źródło	MM, MM/Mem	PDF, str. 111	To samo co: (a + b + 1)>>1		P_AVG_B
PAVGW	PAVGW cel, źródło					P_AVG_W
Instrukcje wprowadzone z SSE – wydobycie/wstawienie						
		Cel, źródło, numer			---	
PEXTRW	PEXTRW cel, źródło, numer	MM, MM/Mem, stała	PDF, str. 112			P_EXTR_W
PINSRW	PINSRW cel, źródło, numer					P_INSR_W
Instrukcje wprowadzone z SSE – min/max						
		Cel, źródło			---	
PMAXUB	PMAXUB cel, źródło	MM, MM/Mem	PDF, str. 113			P_MAX_U_B
PMAXSW	PMAXSW cel, źródło					P_MAX_S_W
PMINUB	PMINUB cel, źródło					P_MIN_U_B
PMINSW	PMINSW cel, źródło					P_MIN_S_W
Instrukcje wprowadzone z SSE – inne						
		Cel, źródło			---	
PMOVBMSKB	PMOVBMSKB cel, źródło	Reg, MM	PDF, str. 114-117	Stosowane w celu określenia znaku lub sprawdzenia wyniku porównania		P_MOV_MSK_B
PMULHUW	PMULHUW cel, źródło	MM, MM/Mem				P_MUL_H_U_W
PSADBW	PSADBW cel, źródło	MM, MM/Mem				P_S_AD_B_W
		Cel, źródło, kolejność				
PSHUFW	PSHUFW cel, źródło, kolejność	MM, MM/Mem, stała				P_SHUF_W

Rozwinięcie skrót / Operacja – ang	Rozwinięcie skrót / Operacja – pol
Average bytes	
Average words	
Extract word	
Insert word	
Max of unsigned bytes	
Max of signed words	
Min of unsigned bytes	
Min of signed words	
Move masked bytes	
Multiply, keep high, unsigned words	
Sum of absolute differences bytes to words	
Shuffle words	