AAA ASCII adjust after addition

Kod hex	Instrukcja	Opis
37	AAA	Koryguj AL po dodawaniu BCD

AAA powoduje korekcję znajdującego się w AL wyniku dodawania dwóch liczb, o ile dodawane są liczby BCD. Dopiero po korekcji wynik będzie liczbą BCD. Należy pamiętać, że AAA samo w sobie nie jest działaniem arytmetycznym, wykonuje jedynie konwersję, gdyż po dodaniu dwóch liczb BCD w wyniku otrzymujemy liczbę binarną, z której AAA dopiero tworzy liczbę BCD.

AAD ASCII adjust AX after division

Kod hex	Instrukcja	Opis
D5 0A	AAD	Koryguj nieupakowany kod BCD przed dzieleniem

AAD wykonuje korekcję dzielnej (licznika) zawartej w AH i w AL przed wykonaniem dzielenia, zakładając, że argumentami dzielenia będą liczby nieupakowane w kodzie BCD.

AAM ASCII adjust AX after multiply

Kod hex	Instrukcja	Opis
D4 0A	AAM	Koryguj nieupakowany kod BCD po mnożeniu

AAM wykonuje korekcję wyniku mnożenia zawartego w AX tylko po instrukcji MUL.

AAS ASCII adjust after subtraction

Kod hex	Instrukcja	Opis
3F	AAS	Koryguj AL po odejmowaniu BCD

Podobnie jak AAA z tym, że korekcja następuje po odejmowaniu.

ADC Add with carry

Kod hex	Instrukcja	Opis
14 *ib*	ADC AL,*i8*	Dodawanie z przeniesieniem *i8* do AL
15 *iw*	ADC AX,*i16*	Dodawanie z przeniesieniem *i16* do AX
15 *id*	ADC EAX,*i32*	Dodawanie z przeniesieniem *i32* do EAX
80 /2 *ib*	ADC *r/m8*,*i8*	=/= *i8* do *r/m8*
81 /2 *iw*	ADC *r/m16*,*i16*	=/= *i16* do *r/m16*
81 /2 *id*	ADC *r/m32*,*i32*	Dodawanie z bezpośrednim CF *i32* do *r/m32*
83 /2 *ib*	ADC *r/m16*,*i8*	=/= rozszerzonego *i8* do *r/m16*
83 /2 *ib*	ADC *r/m32*,*i8*	=/= rozszerzonego *i8* do *r/m32*
10 /*r*	ADC *r/m8*,*r8*	Dodawanie z przeniesieniem *r8* do *r/m8*
11 /*r*	ADC *r/m16*,*r16*	=/= *r16* do *r/m16*
11 /*r*	ADC *r/m32*,*r32*	=/= *r32* do *r/m32*
12 /*r*	ADC *r8*,*r/m8*	=/= *r/m8* do *r8*
13 /*r*	ADC *r16*,*r/m16*	=/= *r/m16* do *r16*
13 /*r*	ADC *r32*,*r/m32*	=/= *r/m32* do *r32*

ADC dodaje operand źródłowy i znacznik przeniesienia CF do operandu docelowego. Wynik dodawania zastępuje wartość operandu docelowego. Jest to dodawanie arytmetyczne, w którym przeniesienie umożliwia wykonywanie dodawania wielu rejestrów czy komórek pamięci. Jeśli wynik nie mieści się w operandzie docelowym, znacznik CF jest ustawiany.

ADD Add

Kod hex	Instrukcja	Opis
04 *ib*	ADD AL,*i8*	Dodawanie *i8* do AL
05 *iw*	ADD AX,*i16*	Dodawanie *i16* do AX
05 *id*	ADD EAX,*i32*	Dodawanie *i32* do EAX
80 /0 *ib*	ADD *r/m8*,*i8*	=/= *i8* do *r/m8*
81 /0 *iw*	ADD *r/m16*,*i16*	=/= *i16* do *r/m16*

81 /0 *id*	ADD *r/m32,i32*	=/= *i32* do *r/m32*
83 /0 *ib*	ADD *r/m16,i8*	=/= rozszerzonego *i8* do *r/m16*
83 /0 *ib*	ADD *r/m32,i8*	=/= rozszerzonego *i8* do *r/m32*
00 /*r*	ADD *r/m8,r8*	=/= *r8* do *r/m8*
01 /*r*	ADD *r/m16,r16*	=/= *r16* do *r/m16*
01 /*r*	ADD *r/m32,r32*	=/= *r32* do *r/m32*
02 /*r*	ADD *r8,r/m8*	=/= *r/m8* do *r8*
03 /*r*	ADD *r16,r/m16*	=/= *r/m16* do *r16*
03 /*r*	ADD *r32,r/m32*	=/= *r/m32* do *r32*

ADD dodaje do siebie operand źródłowy i docelowy, i umieszcza wynik w operandzie docelowym. Jest to dodawanie arytmetyczne, niebiorące pod uwagę stanu znacznika przeniesienia CF. Jeśli wynik nie mieści się w operandzie docelowym, znacznik przeniesienia CF jest ustawiany.

AND **Logical AND**

Kod hex	**Instrukcja**	**Opis**
24 *ib*	AND AL,*i8*	Iloczyn logiczny *i8* i AL
25 *iw*	AND AX,*i16*	Iloczyn logiczny *i16* i AX
25 *id*	AND EAX,*i32*	=/= *i32* i EAX
80 /4 *ib*	AND *r/m8,i8*	=/= *i8* i *r/m8*
81 /4 *iw*	AND *r/m16,i16*	=/= *i16* i *r/m16*
81 /4 *id*	AND *r/m32,i32*	=/= *i32* i *r/m32*
83 /4 *ib*	AND *r/m16,i8*	=/= rozszerzonego *i8* i *r/m16*
83 /4 *ib*	AND *r/m32,i8*	=/= rozszerzonego *i8* i *r/m32*
20 /*r*	AND *r/m8,r8*	=/= *r8* i *r/m8*
21 /*r*	AND *r/m16,r16*	=/= *r16* i *r/m16*
21 /*r*	AND *r/m32,r32*	=/= *r32* i *r/m32*
22 /*r*	AND *r8,r/m8*	=/= *r/m8* i *r8*
23 /*r*	AND *r16,r/m16*	=/= *r/m16* i *r16*
23 /*r*	AND *r32,r/m32*	=/= *r/m32* i *r32*

AND wykonuje operację iloczynu logicznego dwóch operandów. Wynik operacji umieszczany jest w operandzie docelowym. Stan znacznika przeniesienia pomocniczego nie jest zdefiniowany.

BT Bit test

Kod hex	Instrukcja	Opis
0F A3	BT *r/m16,r16*	Zapisanie określonego bitu do znacznika CF
0F A3	BT *r/m32,r32*	=/=
0F BA /4 *ib*	BT *r/m16,i8*	=/=
0F BA /4 *ib*	BT *r/m32,i8*	=/=

BT kopiuje określony bit z pierwszego operandu do znacznika CF, przez co może on być później zbadany lub użyty w inny sposób. Numer bitu, który ma być kopiowany, określany jest przez drugi operand. BT nie modyfikuje wartości żadnego z swoich operandów.

BTC Bit test and complement

Kod hex	Instrukcja	Opis
0F BB	BTC *r/m16,r16*	Zapisanie określonego bitu do znacznika CF i zanegowanie go
0F BB	BTC *r/m32,r32*	=/=
0F BA /7 *ib*	BTC *r/m16,i8*	=/=
0F BA /7 *ib*	BTC *r/m32,i8*	=/=

Działanie podobne jak BT z tym, że bit zapisany w CF jest następnie w źródle negowany.

BTR Bit test and reset

Kod hex	Instrukcja	Opis
0F B3	BTR *r/m16,r16*	Zapisanie określonego bitu do znacznika CF i wyzerowanie go
0F B3	BTR *r/m32,r32*	=/=
0F BA /6 *ib*	BTR *r/m16,i8*	=/=
0F BA /6 *ib*	BTR *r/m32,i8*	=/=

Działanie podobne jak BT z tym, że bit zapisany w CF jest następnie w źródle zerowany.

BTS Bit test and set

Kod hex	Instrukcja	Opis
0F AB	BTS *r/m16,r16*	Zapisanie określonego bitu do znacznika CF i ustawienie go
0F AB	BTS *r/m32,r32*	=/=
0F BA /5 *ib*	BTS *r/m16,i8*	=/=
0F BA /5 *ib*	BTS *r/m32,i8*	=/=

Działanie podobne jak BT z tym, że bit zapisany w CF jest następnie w źródle ustawiany (przyjmuje wartość 1).

CALL Call procedure

Kod hex	Instrukcja	Opis
E8 *cw*	CALL *rel16*	Wywołanie bliskie, relatywne
E8 *cd*	CALL *rel32*	=/=
FF /2	CALL *r/m16*	Wywołanie bliskie, niebezpośrednie, do adresu zawartego w *r/m16*
FF /2	CALL *r/m32*	=/= do adresu zawartego w *r/m32*
9A *cd*	CALL *ptr16:16*	Wywołanie dalekie, absolutne
9A *cp*	CALL *ptr16:32*	=/=
FF /3	CALL *m16:16*	Wywołanie dalekie, niebezpośrednie, do adresu zawartego w *m16:16*
FF /3	CALL *m16:32*	=/= do adresu zawartego w *m16:32*

CALL przekazuje sterowanie do procedury znajdującej się pod wskazanym adresem. Przed przekazaniem sterowania instrukcja ta odkłada na stos adres instrukcji znajdującej się bezpośrednio po niej. Adres ten pozwala na powrót z procedury przy pomocy instrukcji RET.

CBW/CWDE Convert byte to word/Convert word to double word

Kod hex	Instrukcja	Opis
98	CBW	Konwersja z AL do AX
98	CWDE	=/= z AX do EAX

Instrukcja CBW konwertuje bajt z rejestru AL na słowo w rejestrze AX (CWDE konwertuje słowo z AX na podwójne słowo w EAX).

CLC Clear carry flag

Kod hex	Instrukcja	Opis
F8	CLC	Zerowanie znacznika przeniesienia

CLC zeruje znacznik przeniesienia CF. Instrukcja ta wykonywana jest w sytuacjach, kiedy CF musi być wyzerowany przed rozpoczęciem jakichś operacji, np. przed wykonaniem instrukcji obrotów RCL i RCR.

CLD Clear direction flaga

Kod hex	Instrukcja	Opis
FC	CLD	Zerowanie znacznika kierunku

CLD zeruje znacznik kierunku. Stan tego znacznika wpływa na działanie instrukcji łańcuchowych, takich jak STOS, SCAS czy MOV.

CLI Clear interrupt flag

Kod hex	Instrukcja	Opis
FA	CLI	Zerowanie znacznika zezwolenia na przerwanie

Instrukcja CLI zeruje znacznik zezwolenia na przerwanie IF.

CMC Complement carry flag

Kod hex	Instrukcja	Opis
F5	CMC	Negacja znacznika przeniesienia

Instrukcja CMC odwraca znacznik przeniesienia CF.

CMP Compare two operands

Kod hex	Instrukcja	Opis
3C *ib*	CMP AL,*i8*	Porównanie *i8* z AL
3D *iw*	CMP AX,*i16*	=/= *i16* z AX
3D *id*	CMP EAX,*i32*	=/= *i32* z EAX
80 /7 *ib*	CMP *r/m8*,*i8*	=/= *i8* z *r/m8*
81 /7 *iw*	CMP *r/m16*,*i16*	=/= *i16* z *r/m16*
81 /7 *id*	CMP *r/m32*,*i32*	=/= *i32* z *r/m32*
83 /7 *ib*	CMP *r/m16*,*i8*	=/= rozszerzonego *i8* z *r/m16*
83 /7 *ib*	CMP *r/m32*,*i8*	=/= rozszerzonego *i8* z *r/m32*
38 /*r*	CMP *r/m8*,*r8*	=/= *r8* z *r/m8*
39 /*r*	CMP *r/m16*,*r16*	=/= *r16* z *r/m16*
39 /*r*	CMP *r/m32*,*r32*	=/= *r32* z *r/m32*
3A /*r*	CMP *r8*,*r/m8*	=/= *r/m8* z *r8*
3B /*r*	CMP *r16*,*r/m16*	=/= *r/m16* z *r16*
3B /*r*	CMP *r32*,*r/m32*	=/= *r/m32* z *r32*

CMP porównuje wartości swoich operandów i w zależności od wyniku porównania zmienia stan znaczników. Zazwyczaj następną instrukcją po CMP jest instrukcja skoku warunkowego, np. JE czy JNE.

CMPS/CMPSB/CMPSW/CMPSD Compare string operands

Kod hex	Instrukcja	Opis
A6	CMPS *m8*,*m8*	Porównanie bajtów ES:[(E)DI] z [(E)SI]
A7	CMPS *m16*,*m16*	=/= słów ES:[(E)DI] z [(E)SI]
A7	CMPS *m32*,*m32*	=/= podwójnych słów ES[(E)DI] z [(E)SI]
A6	CMPSB	Porównanie bajtów ES:[(E)DI] z DS:[SI]
A7	CMPSW	=/= słów ES:[(E)DI] z DS:[SI]
A7	CMPSD	=/= podwójnych słów ES:[(E)DI] z DS:[SI]

CMPS porównuje bajt, słowo lub podwójne słowo wskazane przez rejestr źródłowy (ES:[(E)DI]) z bajtem, słowem lub podwójnym słowem wskazanym przez rejestr docelowy ([(E)SI] czy DS:[SI]).

CWD Convert word to doubleword

Kod hex	Instrukcja	Opis
99	CWD	DX:AX←rozszerzenie znaku AX

CWD dokonuje konwersji słowa w rejestrze AX, rozszerzając go do podwójnego słowa w rejestrach DX:AX (DX zawiera bardziej znaczącą część).

DAA Decimal adjust AL after addition

Kod hex	Instrukcja	Opis
27	DAA	Korekcja upakowanego kodu BCD po dodawaniu

DAA wykonuje korekcję danych zawartych w rejestrze AL, będących wynikiem dodawania upakowanych liczb w kodzie BCD.

DAS Decimal adjust AL after subtraction

Kod hex	Instrukcja	Opis
2F	DAS	Korekcja upakowanego kodu BCD po odejmowaniu

DAS wykonuje to samo, co instrukcja DAA z tym, że w rejestrze AL znajduje się wynik odejmowania upakowanych liczb w kodzie BCD.

DEC Decrement by 1

Kod hex	Instrukcja	Opis
FE /1	DEC *r/m8*	Zmniejszenie *r/m8* o 1
FF /1	DEC *r/m16*	=/= *r/m16* o 1
FF /1	DEC *r/m32*	=/= *r/m32* o 1
48+*rw*	DEC *r16*	=/= *r16* o 1
48+*rd*	DEC *r32*	=/= *r32* o 1

DEC mniejsza wartość operandu o 1.

DIV Unsigned divide

Kod hex	Instrukcja	Opis
F6 /6	DIV AL,*r/m8*	Dzielna w AX
F7 /6	DIV AX,*r/m16*	=/= w EAX
F7 /6	DIV EAX,*r/m32*	=/= w EDX:EAX

Instrukcja DIV wykonuje dzielenie z resztą zawartości ustalonego rejestru przez operand.

HLT Halt

Kod hex	Instrukcja	Opis
F4	HLT	Zatrzymanie

Instrukcja HLT wprowadza procesor w stan zatrzymania, dalsze rozkazy nie są realizowane. Powrót procesora do normalnej pracy może nastąpić po jego zresetowaniu lub wykonaniu przerwania priorytetowego.

IDIV Signed division

Kod hex	Instrukcja	Opis
F6 /7	IDIV AL,*r/m8*	Dzielenie znakowe, dzielna w AX
F7 /7	IDIV AX,*r/m16*	=/= w EAX
F7 /7	IDIV EAX,*r/m32*	=/= w EDX:EAX

Podobnie jak DIV z tym, że brany jest pod uwagę również znak.

IMUL Signed multiply

Kod hex	Instrukcja	Opis
F6 /5	IMUL *r/m8*	AX←AL**r/m8*
F7 /5	IMUL *r/m16*	DX:AX←AX**r/m16*
F7 /5	IMUL *r/m32*	EDX:EAX←EAX**r/m32*
0F AF /*r*	IMUL *r16,r/m16*	*r16←r16*r/m16*
0F AF /*r*	IMUL *r32,r/m32*	*r32←r32*r/m32*
6B /*r ib*	IMUL *r16,r/m16,i8*	*r16←r/m16*rozszerzone i8*
6B /*r ib*	IMUL *r32,r/m32,i8*	*r32←r/m32*rozszerzone i8*
6B /*r ib*	IMUL *r16,i8*	*r16←r16*rozszerzone i8*
6B /*r ib*	IMUL *r32,i8*	*r32←r32*rozzerzone i8*
69 /*r iw*	IMUL *r16,r/m16,i16*	*r16←r/m16*i16*
69 /*r id*	IMUL *r32,r/m32,i32*	*r32←r/m32*i32*
69 /*r iw*	IMUL *r16,i16*	*r16←r16*i16*
69 /*r id*	IMUL *r32,i32*	*r32←r32*i32*

IMUL mnoży swój operand przez zawartość rejestru AL, AX lub EAX (zawsze któryś z tych dwóch rejestrów, w zależności od operandu), wynik mnożenia natomiast umieszczany jest odpowiednio w AX, DX:AX lub EDX:EAX.

IN Input from port

Kod hex	Instrukcja	Opis
E4 *ib*	IN AL,*i8*	Bajt wejściowy bezpośrednio do AL
E5 *ib*	IN AX,*i8*	Bajt wejściowy bezpośrednio do AX
E5 *ib*	IN EAX,*i8*	Bajt wejściowy bezpośrednio do EAX
EC	IN AL,DX	Bajt wejściowy z portu DX do AL
ED	IN AX,DX	Słowo wejściowe z portu DX do AX
ED	IN EAX,DX	Podwójne słowo wejściowe z portu DX do EAX

Instrkcja IN przenosi bajt danych lub słowo danych z portu ponumerowanego przez pierwszy operand do rejestru (AL, AX lub EAX) określonego przez pierwszy operand.

INC Increment by 1

Kod hex	Instrukcja	Opis
FE /0	INC *r/m8*	Zwiększenie *r/m8* o 1
FF /0	INC *r/m16*	=/= *r/m16* o 1
FF /0	INC *r/m32*	=/= *r/m32* o 1
40+*rw*	INC *r16*	=/= *r16* o 1
40+*rd*	INC *r32*	=/= *r32* o 1

Zwiększa wartość operandu o 1.

INT Call to interrupt procedure

Kod hex	Instrukcja	Opis
CC	INT 3	Przerwanie 3 – pułapka dla debugger'a
CD *ib*	INT *i8*	Przerwanie o numerze wskazanym przez *i8*
CE	INTO	Przerwanie 4 – jeśli flaga przepełnienia = 1

INT wywołuje przerwanie programowe jednego z 256 wektorów znajdujących się w pierwszym kilobajcie pamięci. Numer wektora podawany jest w operandzie. Do powrotu z przerwania używa się przeważnie instrukcji IRET.

IRET Interrupt return

Kod hex	Instrukcja	Opis
CF	IRET	Powrót z przerwania (16-bitowy operand)
CF	IRETD	=/= (32-bitowy operand)

IRET musi być używane do powrotu z procedury obsługi przerwania wywoływanej przez instrukcję INT, czy też przerwania sprzętowego. IRET pobiera ze stosu adres powrotu, a następnie zapamiętane tam znaczniki. Dlatego też zmieniane jest ustawienie wszystkich znaczników.

J? Jump if condition is met

Kod hex	Instrukcja	Opis
77 *cb*	JA *rel8*	Skok krótki, jeśli powyżej (CF=0 i ZF=0)
73 *cb*	JAE *rel8*	Skok krótki, jeśli powyżej lub równe (CF=0)
72 *cb*	JB *rel8*	Skok krótki, jeśli poniżej (CF=1)
76 *cb*	JBE *rel8*	Skok krótki, jeśli poniżej lub równe (CF=1 lub ZF=1)
72 *cb*	JC *rel8*	Skok krótki, jeśli przeniesienie (CF=1)
74 *cb*	JE *rel8*	Skok krótki, jeśli równe (ZF=1)
7F *cb*	JG *rel8*	Skok krótki, jeśli większe (ZF=0 i SF=OF)
7D *cb*	JGE *rel8*	Skok krótki, jeśli większe lub równe (SF=OF)
7C *cb*	JL *rel8*	Skok krótki, jeśli mniejsze (SF<>OF)
7E *cb*	JLE *rel8*	Skok krótki, jeśli mniejsze lub równe (ZF=1 lub SF<>OF)
76 *cb*	JNA *rel8*	Skok krótki, jeśli nie powyżej (CF=1 lub ZF=1)
72 *cb*	JNAE *rel8*	Skok krótki, jeśli nie powyżej lub równe (CF=1)
73 *cb*	JNB *rel8*	Skok krótki, jeśli nie poniżej (CF=0)
77 *cb*	JNBE *rel8*	Skok krótki, jeśli nie poniżej lub równe (CF=0 i ZF=0)
73 *cb*	JNC *rel8*	Skok krótki, jeśli nie przeniesienie (CF=0)
75 *cb*	JNE *rel8*	Skok krótki, jeśli nie równe (ZF=0)
7E *cb*	JNG *rel8*	Skok krótki, jeśli nie większe (ZF=1 lub SF<>OF)
7C *cb*	JNGE *rel8*	Skok krótki, jeśli nie większe lub równe (SF<>OF)
7D *cb*	JNL *rel8*	Skok krótki, jeśli nie mniejsze (SF=OF)
7F *cb*	JNLE *rel8*	Skok krótki, jeśli nie mniejsze lub równe (ZF=0 i SF=OF)
71 *cb*	JNO *rel8*	Skok krótki, jeśli nie przepełnienie (OF=0)
7B *cb*	JNP *rel8*	Skok krótki, jeśli nie parzyste (PF=0)
79 *cb*	JNS *rel8*	Skok krótki, jeśli nie znak (SF=0)
75 *cb*	JNZ *rel8*	Skok krótki, jeśli nie zero (ZF=0)
70 *cb*	JO *rel8*	Skok krótki, jeśli przepełnienie (OF=1)
7A *cb*	JP *rel8*	Skok krótki, jeśli parzyste (PF=1)
7A *cb*	JPE *rel8*	Skok krótki, jeśli parzystość parzysta (PF=1)
7B *cb*	JPO *rel8*	Skok krótki, jeśli parzystość nieparzysta (PF=0)
78 *cb*	JS *rel8*	Skok krótki, jeśli znak (SF=1)
74 *cb*	JZ *rel8*	Skok krótki, jeśli zero (ZF=1)
0F 87 *cw/cd*	JA *rel16/32*	Skok bliski, jeśli powyżej (CF=0 I ZF=0)
0F 83 *cw/cd*	JAE *rel16/32*	Skok bliski, jeśli powyżej lub równe (CF=0)
0F 82 *cw/cd*	JB *rel16/32*	Skok bliski, jeśli poniżej (CF=1)
0F 86 *cw/cd*	JBE *rel16/32*	Skok bliski, jeśli poniżej lub równe (CF=1 lub ZF=1)
0F 82 *cw/cd*	JC *rel16/32*	Skok bliski, jeśli przeniesienie (CF=1)
0F 84 *cw/cd*	JE *rel16/32*	Skok bliski, jeśli równe (ZF=1)
0F 84 *cw/cd*	JZ *rel16/32*	Skok bliski, jeśli zero (ZF=1)

0F 8F *cw/cd*	JG *rel16/32*	Skok bliski, jeśli większe (ZF=0 i SF=OF)
0F 8D *cw/cd*	JGE *rel16/32*	Skok bliski, jeśli większe lub równe (SF=OF)
0F 8C *cw/cd*	JL *rel16/32*	Skok bliski, jeśli mniejsze (SF<>OF)
0F 8E *cw/cd*	JLE *rel16/32*	Skok bliski, jeśli mniejsze lub równe (ZF=1 lub SF<>OF)
0F 86 *cw/cd*	JNA *rel16/32*	Skok bliski, jeśli nie powyżej (CF=1 lub ZF=1)
0F 82 *cw/cd*	JNAE *rel16/32*	Skok bliski, jeśli nie powyżej lub równe (CF=1)
0F 83 *cw/cd*	JNB *rel16/32*	Skok bliski, jeśli nie poniżej (CF=0)
0F 87 *cw/cd*	JNBE *rel16/32*	Skok bliski, jeśli nie poniżej lub równe (CF=0 i ZF=0)
0F 83 *cw/cd*	JNC *rel16/32*	Skok bliski, jeśli nie przeniesienie (CF=0)
0F 85 *cw/cd*	JNE *rel16/32*	Skok bliski, jeśli nie równe (ZF=0)
0F 8E *cw/cd*	JNG *rel16/32*	Skok bliski, jeśli nie większe (ZF=1 lub SF<>OF)
0F 8C *cw/cd*	JNGE *rel16/32*	Skok bliski, jeśli nie większe lub równe (SF<>OF)
0F 8D *cw/cd*	JNL *rel16/32*	Skok bliski, jeśli nie mniejsze (SF=OF)
0F 8F *cw/cd*	JNLE *rel16/32*	Skok bliski, jeśli nie mniejsze lub równe (ZF=0 i SF=OF)
0F 81 *cw/cd*	JNO *rel16/32*	Skok bliski, jeśli nie przepełnienie (OF=0)
0F 8B *cw/cd*	JNP *rel16/32*	Skok bliski, jeśli nie parzyste (PF=0)
0F 89 *cw/cd*	JNS *rel16/32*	Skok bliski, jeśli nie znak (SF=0)
0F 85 *cw/cd*	JNZ *rel16/32*	Skok bliski, jeśli nie zero (ZF=0)
0F 80 *cw/cd*	JO *rel16/32*	Skok bliski, jeśli przepełnienie (OF=1)
0F 8A *cw/cd*	JP *rel16/32*	Skok bliski, jeśli parzyste (PF=1)
0F 8A *cw/cd*	JPE *rel16/32*	Skok bliski, jeśli parzystość parzysta (PF=1)
0F 8B *cw/cd*	JPO *rel16/32*	Skok bliski, jeśli parzystość nieparzysta (PF=0)
0F 88 *cw/cd*	JS *rel16/32*	Skok bliski, jeśli znak (SF=1)
0F 84 *cw/cd*	JZ *rel16/32*	Skok bliski, jeśli zero (ZF=1)

Wszystkie powyższe instrukcje wykonują krótki skok (do adresu położonego do 128 bajtów wstecz lub 127 bajtów w przód), jeśli spełniony jest odpowiedni warunek.

JCXZ/JECXZ Jump if CX/ECX zero

Kod hex	**Instrukcja**	**Opis**
E3 *cb*	JCXZ *rel8*	Krótki skok, jeśli CX=0
E3 *cb*	JECXZ *rel8*	Krótki skok, jeśli ECX=0

Wiele instrukcji używa rejestru CX jako licznika. Instrukcja JCXZ umożliwia nam sprawdzenie wartości tego rejestru i skok, jeśli jest ona równa zero.

JMP Jump

Kod hex	Instrukcja	Opis
EB *cb*	JMP *rel8*	Skok krótki
E9 *cw*	JMP *rel16*	Skok bliski, przemieszczenie względne do następnej instrukcji
E9 *cd*	JMP *rel32*	=/=
FF /4	JMP *r/m16*	Skok bliski, pośredni, do adresu podanego w *r/m16*
FF /4	JMP *r/m32*	=/= do adresu podanego w *r/m32*
EA *cd*	JMP *ptr16:16*	Skok daleki, do adresu podanego w operandzie
EA *cp*	JMP *ptr16:32*	=/=
FF /5	JMP *m16:16*	Skok daleki, pośredni, do adresu podanego w *m16:16*
FF /5	JMP *m16:32*	=/= do adresu podanego w *m16:32*

Instrukcja JMP przekazuje bezwarunkowo sterowanie do miejsca opisanego przez operand. Może to być etykieta (bliska i daleka), adres przechowywany w rejestrze ogólnego przeznaczenia (bliski) bądź zawartość komórek pamięci (bliski lub daleki). JMP nie zmienia stanu znaczników, nie odkłada adresu powrotnego na stos.

LAHF Load status flags into AH register

Kod hex	Instrukcja	Opis
9F	LAHF	Ładowanie do AH flagi: SF, ZF, xx, AF, xx, PF, xx, CF

LAHF przenosi dolny bajt słowa flagi do rejestru AH. Bity z MSB do LSB są bitami: znaku, zera, nieokreślonymi, pomocniczymi, nieokreślonymi, parzystości, nieokreślonymi i przeniesienia.

LEA Load effective address

Kod hex	Instrukcja	Opis
8D /*r*	LEA *r16,m*	Zapisanie adresu efektywnego dla *m* w *r16*
8D /*r*	LEA *r32,m*	Zapisanie adresu efektywnego dla *m* w *r32*

LEA pobiera z operandu źródła adres przesunięcia w segmencie (offset) danej i ładuje ten adres do operandu źródłowego. Operand źródłowy musi być rejestrem.

LEAVE High level procedure exit

Kod hex	Instrukcja	Opis
C9	LEAVE	Umieszczenie SP w BP, zdjęcie BP
C9	LEAVE	=/= ESP w EBP, zdjęcie EBP

Instrukcja LEAVE wykonuje czynność odwrotną do tej, którą wykonuje ENTER.

LODS/LODSB/LODSW/LODSD Load string

Kod hex	Instrukcja	Opis
AC	LODS *m8*	Ładowanie bajtu [(E)SI] do AL
AD	LODS *m16*	=/= słowa [(E)SI] do AX
AD	LODS *m32*	=/= podwójnego słowa [(E)SI] do EAX
AC	LODSB	=/= bajtu DS:[(E)SI] do AL
AD	LODSW	=/= słowa DS:[(E)SI] do AX
AD	LODSD	=/= podwójnego słowa DS:[(E)SI] do EAX

LODS umieszcza dane wskazane przez rejestr [(E)SI] w rejestrze AL, AX lub EAX. Zawartość rejestru jest tak modyfikowana, by wskazywała na następny element łańcucha.

LOOP/LOOPcc Loop according to ECX counter

Kod hex	Instrukcja	Opis
E2 *cb*	LOOP *rel8*	Zmniejszenie licznika i krótki skok, jeśli licznik<>0
E1 *cb*	LOOPE *rel8*	Zmniejszenie licznika i krótki skok, jeśli licznik<>0 i ZF=1
E0 *cb*	LOOPNE *rel8*	Zmniejszenie licznika i krótki skok, jeśli licznik<>0 i ZF=0

LOOP jest instrukcją łączącą w sobie operację zmniejszenia licznika, sprawdzenia czy osiągnął on zero i skoku warunkowego. Działa jak kombinacja instrukcji: DEC CX; CMP CX,0; JZ <etykieta>. Licznik musi uprzednio być załadowany liczbą powtórzeń pętli.

LOOPZ Loop while CX>0 and ZF=1

Kod hex	Instrukcja	Opis
E1 *cb*	LOOPZ *rel8*	Zmniejszenie licznika i krótki skok, jeśli CX≠0 i ZF=1

LOOPZ zmniejsza CX, a następnie wykonuje skok do miejsca wskazanego przez operand, o ile CX nie jest równe zero i jednocześnie wskaźnik zera ZF=1. W przeciwnym przypadku, (jeśli choć jeden z tych warunków nie jest spełniony) wykonywana będzie następna instrukcja.

LOOPNZ Loop while CX>0 and ZF=0

Kod hex	Instrukcja	Opis
E0 *cb*	LOOPNZ *rel8*	Zmniejszenie licznika i krótki skok, jeśli CX≠0 i ZF=0

LOOPNZ zmniejsza CX, a następnie wykonuje skok do miejsca wskazanego przez operand, o ile CX nie jest równe zero i jednocześnie wskaźnik zera ZF=0.

W przeciwnym przypadku, (jeśli choć jeden z tych warunków nie jest spełniony) wykonywana będzie następna instrukcja.

MOV Move

Kod hex	Instrukcja	Opis
88 /*r*	MOV *r/m8,r8*	Kopiowanie *r8* do *r/m8*
89 /*r*	MOV *r/m16,r16*	=/= *r16* do *r/m16*
89 /*r*	MOV *r/m32,r32*	=/= *r32* do *r/m32*
8A /*r*	MOV *r8,r/m8*	=/= *r/m8* do *r8*
8B /*r*	MOV *r16,r/m16*	=/= *r/m16* do *r16*
8B /*r*	MOV *r32,r/m32*	=/= *r/m32* do *r32*
8C /*r*	MOV *r/m16*,Sreg	=/= rejestru segmentowego do *r/m16*
8E /*r*	MOV Sreg,*r/m16*	=/= *r/m16* do rejestru segmentowego
A0	MOV AL,*moffs8*	=/= bajtu od adresu (seg:off) do AL
A1	MOV AX,*moffs16*	=/= słowa od adresu (seg:off) do AX
A1	MOV EAX,*moffs32*	=/= podwójnego słowa od adresu (seg:off) do EAX
A2	MOV *moffs8*,AL	=/= z AL do (seg:off)
A3	MOV *moffs16*,AX	=/= z AX do (seg:off)
A3	MOV *moffs32*,EAX	=/= z EAX do (seg:off)
B0+*rb*	MOV *reg8,i8*	=/= *i8* do *reg8*
B8+*rw*	MOV *reg16,i16*	=/= *i16* do *reg16*
B8+*rd*	MOV *reg32,i32*	=/= *i32* do *reg32*
C6 /0	MOV *r/m8,i8*	=/= *i8* do *r/m8*
C7 /0	MOV *r/m16,i16*	=/= *i16* do *r/m16*
C7 /0	MOV *r/m32,32*	=/= *i32* do *r/m32*

Jest to chyba najczęściej używana instrukcja. Kopiuje ona prawy operand do lewego.

MOVS/MOVSB/MOVSW/MOVSD Move data from string to string

Kod hex	Instrukcja	Opis
A4	MOVS *m8,m8*	Ładowanie bajtu [(E)SI]→ES:[(E)DI]
A5	MOVS *m16,m16*	Ładowanie słowa [(E)SI]→ES:[(E)DI]
A5	MOVS *m32,m32*	Ładowanie p. słowa [(E)SI]→ES:[(E)DI]
A4	MOVSB	Ładowanie bajtu DS:[(E)SI]→ES:[(E)DI]
A5	MOVSW	Ładowanie słowa DS:[(E)SI]→ES:[(E)DI]
A5	MOVSD	Ładowanie p. słowa DS:[(E)SI]→ES:[(E)DI]

MOVS kopiuje bajt, słowo lub podwójne słowo od miejsca wskazanego przez [(E)SI] do miejsca wskazanego przez ES:[(E)DI].

MOVSX Move with Sign-Extension

Kod hex	Instrukcja	Opis
0F BE /*r*	MOVSX *r16,r/m8*	Ładowanie bajtu do słowa z rozszerzeniem znaku
0F BE /*r*	MOVSX *r32,r/m8*	Ładowanie bajtu do podwójnego słowa z rozszerzeniem znaku
0F BE /*r*	MOVSX *r32,r/m16*	Ładowanie słowa do podwójnego słowa z rozszerzeniem znaku

MOVSX kopiuje bajt lub słowo i zapisuje je do rejestru słowa lub podwójnego słowa, rozszerzając je z zachowaniem znaku.

MOVZX Move with Zero-Extension

Kod hex	Instrukcja	Opis
0F B6 /*r*	MOVZX *r16,r/m8*	Ładowanie bajtu do słowa z rozszerzeniem zerami
0F B6 /*r*	MOVZX *r32,r/m8*	Ładowanie bajtu do podwójnego słowa z rozszerzeniem zerami
0F B6 /*r*	MOVZX *r32,r/m16*	Ładowanie słowa do podwójnego słowa z rozszerzeniem zerami

MOVZX działa podobnie jak MOVSX z tym że uzupełnianie następuje zerami.

MUL Unsigned multiply

Kod hex	Instrukcja	Opis
F6 /4	MUL AL,*r/m8*	Mnożenie bez znaku (AX←AL**r/m8*)
F7 /4	MUL AX,*r/m16*	=/= (EAX←AX**r/m16*)
F7 /4	MUL EAX,*r/m32*	=/= (EDX:EAX←EAX**r/m32*)

MUL wykonuje mnożenie dwóch operandów bez znaku. Wynik zależy od wielkości tych operandów.

NEG Negate

Kod hex	Instrukcja	Opis
F6 /3	NEG *r/m8*	Negacja *r/m8*, uzupełnia do dwóch
F7 /3	NEG *r/m16*	*=/= r/m16*
F7 /3	NEG *r/m32*	*=/= r/m32*

Jest to asemblerowy odpowiednik mnożenia przez -1, czyli zmiany liczby na przeciwną co do znaku. Nie jest to negacja każdego bitu operandu, jak to robi instrukcja NOT. Proces ten nazywa się także tworzeniem dopełnienia do 2.

NOP No operation

Kod hex	Instrukcja	Opis
90	NOP	Nic nie robi

Najprostsza do zrozumienia instrukcja procesora. Nie robi ona nic. Jej zadaniem jest zapełnienie pamięci pomiędzy innymi instrukcjami. Procesor pobiera kod tej instrukcji z pamięci i ignoruje go.

NOT Logical NOT

Kod hex	Instrukcja	Opis
F6 /2	NOT *r/m8*	Odwracanie wszystkich bitów *r/m8*
F7 /2	NOT *r/m16*	*=/= r/m16*
F7 /2	NOT *r/m32*	*=/= r/m32*

NOT odwraca wszystkie bity operandu – z 0 robi 1 i odwrotnie. Jest to operacja negacji logicznej, zwana także dopełnieniem do 1.

OR Logical OR

Kod hex	Instrukcja	Opis
0C *ib*	OR AL,*i8*	AL ⊕ *i8*
0D *iw*	OR AX,*i16*	AX ⊕ *i16*
0D *id*	OR EAX,*i32*	EAX ⊕ *i32*
80 /1 *ib*	OR *r/m8*,*i8*	*i8* ⊕ *r/m8*
81 /1 *iw*	OR *r/m16*,*i16*	*i16* ⊕ *r/m16*
81 /1 *id*	OR *r/m32*,*i32*	*i32* ⊕ *r/m32*
83 /1 *ib*	OR *r/m16*,*i8*	Rozszerzone *i8* ⊕ *r/m16*
83 /1 *ib*	OR *r/m32*,*i8*	=/= *i8* ⊕ *r/m32*
08 /*r*	OR *r/m8*,*r8*	*r8* ⊕ *r/m8*
09 /*r*	OR *r/m16*,*r16*	*r16* ⊕ *r/m16*
09 /*r*	OR *r/m32*,*r32*	*r32* ⊕ *r/m32*
0A /*r*	OR *r8*,*r/m8*	*r/m8* ⊕ *r8*
0B /*r*	OR *r16*,*r/m16*	*r/m16* ⊕ *r16*
0B /*r*	OR *r32*,*r/m32*	*r/m32* ⊕ *r32*

Instrukcja OR wykonuje operację sumy logicznej swoich dwóch operandów bit po bicie. Wynik zastępuje operand docelowy.

OUT Output to port

Kod hex	Instrukcja	Opis
E6 *ib*	OUT *i8*,AL	Bajt z AL bezpośrednio do nr portu
E7 *ib*	OUT *i8*,AX	Słowo z AX bezpośrednio do nr portu
E7 *ib*	OUT *i8*,EAX	P. słowo z EAX bezpośrednio do nr portu
EE	OUT DX,AL	Bajt z AL do portu o nr w DX
EF	OUT DX,AX	Słowo z AX do portu o nr w DX
EF	OUT DX,EAX	P. słowo z EAX do portu o nr w DX

OUT przenosi zawartość AL, AX lub EAX do portu wskazanego przez pierwszy operand.

POP Pop a value from the stack

Kod hex	Instrukcja	Opis
8F /0	POP *m16*	Ściągnięcie z wierzchołka stosu do *m16*
8F /0	POP *m32*	=/= do *m32*
58+*rw*	POP *r16*	=/= do *r16*

58+*rd*	POP *r32*	=/= do *r32*
1F	POP DS	=/= do DS
07	POP ES	=/= do ES
17	POP SS	=/= do SS
0F A1	POP FS	=/= do FS
0F A9	POP GS	=/= do GS

POP ściąga z wierzchołka stosu słowo i zapisuje go do operandu. Niemożliwe jest zdjęcie ze stosu liczby 1-bajtowej. Można pobrać albo słowo (2 bajty) albo nic.

POPA Pop all general-purpose register

Kod hex	Instrukcja	Opis
61	POPA	Ściągnięcie DI, SI, BP, SP, BX, DX, CX i AX ze stosu

POPA zdejmuje ze stosu wszystkie rejestry ogólnego przeznaczenia. Rejestry ściągane są w kolejności podanej w powyższej tabeli.

POPF Pop stacks into EFLAGS register

Kod hex	Instrukcja	Opis
9D	POPF	Ściągnięcie z wierzchołka stosu i kopiowanie do dolnych 16 bitów rejestru EFLAG

POPF zdejmuje ze stosu słowo (dwa bajty) i kopiuje je do dolnych 16 bitów rejestru EFLAG. POPF w połączeniu z PUSHF są najczęściej używane w procedurach obsługi przerwań.

PUSH Push operand onto the stack

Kod hex	Instrukcja	Opis
FF /6	PUSH *r/m16*	Odkładanie na stos *r/m16*
FF /6	PUSH *r/m32*	=/= *r/m32*

50+*rw*	PUSH *r16*	=/= *r16*
50+*rd*	PUSH *r32*	=/= *r32*
6A	PUSH *i8*	=/= *i8*
68	PUSH *i16*	=/= *i16*
68	PUSH *i32*	=/= *i32*
0E	PUSH CS	=/= CS
16	PUSH SS	=/= SS
1E	PUSH ES	=/= ES
0F A0	PUSH FS	=/= FS
0F A8	PUSH GS	=/= GS

PUSH odkłada na stos operand. Niemożliwe jest odłożenie na stos liczby 1-bajtowej. Można odłożyć albo słowo (2 bajty) albo nic.

PUSHA Push all general-purpose register

Kod hex	Instrukcja	Opis
60	PUSHA	Odkładanie AX, CX, DX, BX, SP, BP, SI i DI na stos

PUSHA odkłada na stos wszystkie rejestry ogólnego przeznaczenia. Są one wkładane w kolejności podanej w powyższej tabeli.

PUSHF Push EFLAGS register onto the stack

Kod hex	Instrukcja	Opis
9C	PUSHF	Odkładanie na stos dolne 16 bitów rejestru EFLAG

PUSHF odkłada na stos zawartość rejestru znaczników (2 bajty).

RET Return from procedure

Kod hex	Instrukcja	Opis
C3	RET	Powrót bliski
CB	RET	Powrót daleki, równe przywileje
CB	RET	Powrót daleki, mniejsze przywileje,

		przełączenie stosu
C2 *iw*	RET *i16*	Powrót bliski, zdjęcie *i16* bajtów parametrów
CA *iw*	RET *i16*	Powrót daleki, równe przywileje, zdjęcie *i16* bajtów
CA *iw*	RET *i16*	Powrót daleki, mniejsze przywileje, zdjęcie *i16* bajtów

Istnieją dwa rodzaje powrotów: bliski (near) i daleki (far). Powrót bliski występuje w obrębie aktualnego segmentu kodu. Powrót daleki występuje, gdy sterowanie zostaje przekazane do innego segmentu.

ROL Rotate left

Kod hex	Instrukcja	Opis
D0 /0	ROL *r*/*m8*,1	Obrót 8 bitów *r*/*m8* jeden raz w lewo
D2 /0	ROL *r*/*m8*,CL	Obrót 8 bitów *r*/*m8* CL-razy w lewo
C0 /0 *ib*	ROL *r*/*m8*,*i8*	Obrót 8 bitów *r*/*m8* *i8*-razy w lewo
D1 /0	ROL *r*/*m16*,1	Obrót 16 bitów *r*/*m16* jeden raz w lewo
D3 /0	ROL *r*/*m16*,CL	Obrót 16 bitów *r*/*m16* CL-razy w lewo
C1 /0 *ib*	ROL *r*/*m16*,*i8*	Obrót 16 bitów *r*/*m16* *i8*-razy w lewo
D1 /0	ROL *r*/*m32*,1	Obrót 32 bitów *r*/*m32* jeden raz w lewo
D3 /0	ROL *r*/*m32*,CL	Obrót 32 bitów *r*/*m32* CL-razy w lewo
C1 /0 *ib*	ROL *r*/*m32*,*i8*	Obrót 32 bitów *r*/*m32* *i8*-razy w lewo

ROL obraca bity operandu przeznaczenia w lewo, to znaczy ku bardziej znaczącym bitom. Obrót jest przesunięciem, przy czym to, co wychodzi z jednej strony rejestru, wchodzi do niego z drugiej. Ilość pozycji, o jaką ma być obrócony rejestr, może być podana jako stała lub w rejestrze CL.

ROR Rotate right

Kod hex	Instrukcja	Opis
D0 /1	ROR *r*/*m8*,1	Obrót 8 bitów *r*/*m8* jeden raz w prawo
D2 /1	ROR *r*/*m8*,CL	Obrót 8 bitów *r*/*m8* CL-razy w prawo
C0 /1 *ib*	ROR *r*/*m8*,*i8*	Obrót 8 bitów *r*/*m8* *i8*-razy w lewo
D1 /1	ROR *r*/*m16*,1	Obrót 16 bitów *r*/*m16* jeden raz w prawo
D3 /1	ROR *r*/*m16*,CL	Obrót 16 bitów *r*/*m16* CL-razy w prawo
C1 /1 *ib*	ROR *r*/*m16*,*i8*	Obrót 16 bitów *r*/*m16* *i8*-razy w prawo
D1 /1	ROR *r*/*m32*,1	Obrót 32 bitów *r*/*m32* jeden raz w prawo

D3 /1	ROR *r/m32*,CL	Obrót 32 bitów *r/m32* CL-razy w prawo
C1 /1 *ib*	ROR *r/m32,i8*	Obrót 32 bitów *r/m32 i8*-razy w prawo

ROR obraca bity operandu przeznaczenia w prawo, to znaczy ku mniej znaczącym bitom. Ilość pozycji, o jaką ma być obrócony rejestr, może być podana jako stała lub w rejestrze CL.

SBB Integer subtraction with borrow

Kod hex	**Instrukcja**	**Opis**
1C *ib*	SBB AL,*i8*	Odejmowanie z pożyczką *i8* od AL
1D *iw*	SBB AX,*i16*	=/= *i16* od AX
1D *id*	SBB EAX,*i32*	=/= *i32* od EAX
80 /3 *ib*	SBB *r/m8,i8*	=/= *i8* od *r/m8*
81 /3 *iw*	SBB *r/m16,i16*	=/= *i16* od *r/m16*
81 /3 *id*	SBB *r/m32,i32*	=/= *i32* od *r/m32*
83 /3 *ib*	SBB *r/m16,i8*	=/= *i8* od *r/m16*
83 /3 *ib*	SBB *r/m32,i8*	=/= *i8* od *r/m32*
18 /*r*	SBB *r/m8,r8*	=/= *r8* od *r/m8*
19 /*r*	SBB *r/m16,r16*	=/= *r16* od *r/m16*
19 /*r*	SBB *r/m32,r32*	=/= *r32* od *r/m32*
1A /*r*	SBB *r8,r/m8*	=/= *r/m8* od *r8*
1B /*r*	SBB *r16,r/m16*	=/= *r/m16* od *r16*
1B /*r*	SBB *r32,r/m32*	=/= *r/m32* od *r32*

SBB jest instrukcją wykonującą odejmowanie z pożyczką. Operand źródłowy jest odejmowany od operandu docelowego. Od wyniku odejmowana jest jeszcze zawartość wskaźnika przeniesienia CF.

SET? Set byte on condition

Kod hex	**Instrukcja**	**Opis**
0F 97	SETA *r/m8*	Ustawienie bajtu jeśli powyżej (CF=0 i ZF=0)
0F 93	SETAE *r/m8*	Ustawienie bajtu jeśli powyżej lub równe (CF=0)
0F 92	SETB *r/m8*	Ustawienie bajtu jeśli poniżej (CF=1)
0F 96	SETBE *r/m8*	Ustawienie bajtu jeśli poniżej lub równe (CF=1 lub ZF=1)
0F 92	SETC *r/m8*	Ustawienie bajtu jeśli przeniesienie (CF=1)
0F 94	SETE *r/m8*	Ustawienie bajtu jeśli równe (ZF=1)

0F 9F	SETG *r/m8*	Ustawienie bajtu jeśli większe (ZF=0 i SF=OF)
0F 9D	SETGE *r/m8*	Ustawienie bajtu jeśli większe lub równe (SF=OF)
0F 9C	SETL *r/m8*	Ustawienie bajtu jeśli mniejsze (SF<>OF)
0F 9E	SETLE *r/m8*	Ustawienie bajtu jeśli mniejsze lub równe (ZF=1 lub SF<>OF)
0F 96	SETNA *r/m8*	Ustawienie bajtu jeśli nie powyżej (CF=1 lub ZF=1)
0F 92	SETNAE *r/m8*	Ustawienie bajtu jeśli nie powyżej lub równe (CF=1)
0F 93	SETNB *r/m8*	Ustawienie bajtu jeśli nie poniżej (CF=0)
0F 97	SETNBE *r/m8*	Ustawienie bajtu jeśli nie poniżej lub równe (CF=0 i ZF=0)
0F 93	SETNC *r/m8*	Ustawienie bajtu jeśli nie przeniesienie (CF=0)
0F 95	SETNE *r/m8*	Ustawienie bajtu jeśli nie równe (ZF=0)
0F 9E	SETNG *r/m8*	Ustawienie bajtu jeśli nie większe (ZF=1 lub SF<>OF)
0F 9C	SETNGE *r/m8*	Ustawienie bajtu jeśli nie większe lub równe (SF<>OF)
0F 9D	SETNL *r/m8*	Ustawienie bajtu jeśli nie mniejsze (SF=OF)
0F 9F	SETNLE *r/m8*	Ustawienie bajtu jeśli nie mniejsze lub równe (ZF=0 i SF=OF)
0F 91	SETNO *r/m8*	Ustawienie bajtu jeśli nie przepełnienie (OF=0)
0F 9B	SETNP *r/m8*	Ustawienie bajtu jeśli nie parzyste (PF=0)
0F 99	SETNS *r/m8*	Ustawienie bajtu jeśli nie znak (SF=0)
0F 95	SETNZ *r/m8*	Ustawienie bajtu jeśli nie zero (ZF=0)
0F 90	SETO *r/m8*	Ustawienie bajtu jeśli przepełnienie (OF=1)
0F 9A	SETP *r/m8*	Ustawienie bajtu jeśli parzyste (PF=1)
0F 9A	SETPE *r/m8*	Ustawienie bajtu jeśli parzystość parzysta (PF=1)
0F 9B	SETPO *r/m8*	Ustawienie bajtu jeśli parzystość nieparzysta (PF=0)
0F 98	SETS *r/m8*	Ustawienie bajtu jeśli znak (SF=1)
0F 94	SETZ *r/m8*	Ustawienie bajtu jeśli zero (ZF=1)

Jeśli warunek określony przez instrukcje jest spełniony, instrukcja SET? Nadaje bajtowi wskazanemu przez operand wartość 1.

SHL Shift left

Kod hex	**Instrukcja**	**Opis**
D0 /4	SHL *r/m8*,1	Przesunięcie 8 bitów *r/m8* jeden raz w lewo
D2 /4	SHL *r/m8*,CL	Przesunięcie 8 bitów *r/m8* CL-razy w lewo

C0 /4 *ib*	SHL *r*/*m8*,*i8*	Przesunięcie 8 bitów *r*/*m8 i8*-razy w lewo
D1 /4	SHL *r*/*m16*,1	Przesunięcie 16 bitów *r*/*m16* jeden raz w lewo
D3 /4	SHL *r*/*m16*,CL	Przesunięcie 16 bitów *r*/*m16* CL-razy w lewo
C1 /4 *ib*	SHL *r*/*m16*,*i8*	Przesunięcie 16 bitów *r*/*m16 i8*-razy w lewo
D1 /4	SHL *r*/*m32*,1	Przesunięcie 32 bitów *r*/*m32* jeden raz w lewo
D3 /4	SHL *r*/*m32*,CL	Przesunięcie 32 bitów *r*/*m32* CL-razy w lewo
C1 /4 *ib*	SHL *r*/*m32*,*i8*	Przesunięcie 32 bitów *r*/*m32 i8*-razy w lewo

SHL przesuwa swój pierwszy operand w lewo o liczbę podaną w drugim operandzie, lub w rejestrze CL. Operację tę można porównać z mnożeniem binarnym przez 2 (tyle razy ile jest podane w drugim operandzie lub rejestrze CL).

SHR Shift right

Kod hex	**Instrukcja**	**Opis**
D0 /5	SHR *r*/*m8*,1	Przesunięcie 8 bitów *r*/*m8* jeden raz w prawo
D2 /5	SHR *r*/*m8*,CL	Przesunięcie 8 bitów *r*/*m8* CL-razy w prawo
C0 /5	SHR *r*/*m8*,*i8*	Przesunięcie 8 bitów *r*/*m8 i8*-razy w prawo
D1 /5	SHR *r*/*m16*,1	Przesunięcie 16 bitów *r*/*m16* jeden raz w prawo
D3 /5	SHR *r*/*m16*,CL	Przesunięcie 16 bitów *r*/*m16* CL-razy w prawo
C1 /5 *ib*	SHR *r*/*m16*,*i8*	Przesunięcie 16 bitów *r*/*m16 i8*-razy w prawo
D1 /5	SHR *r*/*m32*,1	Przesunięcie 32 bitów *r*/*m32* jeden raz w prawo
D3 /5	SHR *r*/*m32*,CL	Przesunięcie 32 bitów *r*/*m32* CL-razy w prawo
C1 /5 *ib*	SHR *r*/*m32*,*i8*	Przesunięcie 32 bitów *r*/*m32 i8*-razy w prawo

Podobnie jak SHL z tym, że bity przesuwane są w prawo (podobnie jak przy dzieleniu przez 2).

STC Set carry flag

Kod hex	**Instrukcja**	**Opis**
F9	STC	Ustawienie flagi przeniesienia CF

STC ustawia znacznik przeniesienia CF na 1.

STD Set direction flag

Kod hex	Instrukcja	Opis
FD	STD	Ustawienie flagi kierunku DF

STD ustawia znacznik kierunku na 1. Przydatne jest to przy pewnych zadaniach wymagających ustawionego znacznika DF, gdyż wpływa na kierunek wykonywania instrukcji łańcuchowych. DF zerujemy instrukcją CLD.

STI Set interruption flag

Kod hex	Instrukcja	Opis
FB	STI	Ustawienie flagi przerwania IF

STI ustawia flagę przerwania IF. Po jej ustawieniu, tuz po zakończeniu wykonywania następnej instrukcji maskowalne przerwania sprzętowe będą przyjmowane, aż do ponownego wyzerowania flagi IF.

STOS/STOSB/STOSW/STOSD Store string

Kod hex	Instrukcja	Opis
AA	STOS *m8*	Zapamiętanie AL pod adresem ES:[(E)DI]
AB	STOS *m16*	Zapamiętanie AX pod adresem ES:[(E)DI]
AB	STOS *m32*	Zapamiętanie EAX pod adresem ES:[(E)DI]
AA	STOSB	Odpowiednik STOS *m8*
AB	STOSW	=/= STOS *m16*
AB	STOSD	=/= STOS *m32*

Instrukcja STOS zapamiętuje zawartość AL – operacje 8 bitowe, AX - operacje 16 bitowe, lub EAX – operacje 32 bitowe pod adresem wskazywanym przez parę rejestrów ES:[(E)DI].

SUB Subtract

Kod hex	Instrukcja	Opis
2C *ib*	SUB AL,*i8*	Odejmowanie *i8* od AL
2D *iw*	SUB AX,*i16*	=/= *i16* od AX
2D *id*	SUB EAX,*i32*	=/= *i32* od EAX
80 /5 *ib*	SUB *r/m8*,*i8*	=/= *i8* od *r/m8*
81 /5 *iw*	SUB *r/m16*,*i16*	=/= *i16* od *r/m16*
81 /5 *id*	SUB *r/m32*,*i32*	=/= *i32* od *r/m32*
83 /5 *ib*	SUB *r/m16*,*i8*	=/= rozszerzonego *i8* od *r/m16*
83 /5 *ib*	SUB *r/m32*,*i8*	=/= rozszerzonego *i8* od *r/m32*
28 /*r*	SUB *r/m8*,*r8*	=/= *r8* od *r/m8*
29 /*r*	SUB *r/m16*,*r16*	=/= *r16* od *r/m16*
29 /*r*	SUB *r/m32*,*r32*	=/= *r32* od *r/m32*
2A /*r*	SUB *r8*,*r/m8*	=/= *r/m8* od *r8*
2B /*r*	SUB *r16*,*r/m16*	=/= *r/m16* od *r16*
2B /*r*	SUB *r32*,*r/m32*	=/= *r/m32* od *r32*

SUB wykonuje odejmowanie bez pożyczki. Od operandu docelowego odejmowany jest operand źródłowy. Przy wyniku działania ujemnym ustawiany jest znacznik CF.

XCHG Exchange register/memory with register

Kod hex	Instrukcja	Opis
90+*rw*	XCHG AX,*r16*	Zamiana *r16* z AX
90+*rw*	XCHG *r16*,AX	=/= AX z *r16*
90+*rd*	XCHG EAX,*r32*	=/= *r32* z EAX
90+*rd*	XCHG *r32*,EAX	=/= EAX z *r32*
86 /*r*	XCHG *r/m8*,*r8*	=/= *r8* z *r/m8*
86 /*r*	XCHG *r8*,*r/m8*	=/= *r/m8* z *r8*
87 /*r*	XCHG *r/m16*,*r16*	=/= *r16* z *r/m16*
87 /*r*	XCHG *r16*,*r/m16*	=/= *r/m16* z *r16*
87 /*r*	XCHG *r/m32*,*r32*	=/= *r32* z *r/m32*
87 /*r*	XCHG *r32*,*r/m32*	=/= *r/m32* z *r32*

XCHG wymienia zawartość dwóch swoich operandów, dlatego też jeden rejestr nie może być dwoma operandami, np. XCHG AX,AX, gdyż nie ma to logicznego sensu.

XOR Logical exclusive OR

Kod hex	Instrukcja	Opis
34 *ib*	XOR AL,*i8*	AL XOR *i8*
35 *iw*	XOR AX,*i16*	AX XOR *i16*
35 *id*	XOR EAX,*i32*	EAX XOR *i32*
80 /6 *ib*	XOR *r/m8*,*i8*	*r/m8* XOR *i8*
81 /6 *iw*	XOR *r/m16*,*i16*	*r/m16* XOR *i16*
81 /6 *id*	XOR *r/m32*,*i32*	*r/m32* XOR *i32*
83 /6 *ib*	XOR *r/m16*,*i8*	*r/m16* XOR rozszerzone *i8*
83 /6 *ib*	XOR *r/m32*,*i8*	*r/m32* XOR rozszerzone *i8*
30 /*r*	XOR *r/m8*,*r8*	*r/m8* XOR *r8*
31 /*r*	XOR *r/m16*,*r16*	*r/m16* XOR *r16*
31 /*r*	XOR *r/m32*,*r32*	*r/m32* XOR *r32*
32 /*r*	XOR *r8*,*r/m8*	*r8* XOR *r/m8*
33 /*r*	XOR *r16*,*r/m16*	*r16* XOR *r/m16*
33 /*r*	XOR *r32*,*r/m32*	*r32* XOR *r/m32*

Instrukcja XOR wykonuje operację logiczną EXOR (suma modulo 2) na pojedynczych bitach obu operandów. Wynik kopiowany jest do operandu przeznaczenia.

Oznaczenia stosowane w opisie instrukcji:

Kolumna **kod hex**:

- /digit: cyfra między 0 a 7 wskazuje, że bajt ModR/M używa tylko r/m (rejestr lub pamięć). W polu reg jest cyfra, która pozwala na rozszerzenie kodu instrukcji.
- /r: wskazuje, że bajt ModR/M instrukcji zawiera zarówno operand rejestru, jak i operand r/m.
- *cb*, *cw*, *cd*, *cp*: odpowiednio 1, 2 i 4-bajtowe wartości następujące po kod hex, które są użyte do określenia offsetu kodu i (jeśli to możliwe) nowej wartości kodu dla rejestru segmentu kodu.

- *ib*, *iw*, *id*: odpowiednio 1, 2 i 4-bajtowe bezpośrednie operandy do instrukcji, które następują po kod hex, bajt ModR/M lub bajt indeksująco-skalujący.

+*rb*, +*rw*, +*rd*: kod rejestru o wartościach od 0 do 7

rb	**rw**	**rd**
AL=0 CL=1 DL=2 BL=3	AX=0 CX=1 DX=2 BX=3	EAX=0 ECX=1 EDX=2 EBX=3
rb	**rw**	**rd**
AH=4 CH=5 DH=6 BH=7	SP=4 BP=5 SI=6 DI=7	ESP=4 EBP=5 ESI=6 EDI=7

Kolumna **Instrukcja**:

- *rel8*: adresy względne w obszarze 128 bajtów przed końcem i 127 bajtów za końcem instrukcji.
- *rel16*, *rel32*: adresy względne wewnątrz tego samego segmentu kodu jak asemblowana instrukcja; stosuje się odpowiednio – rel16 do instrukcji z 16-bitowymi operandami, rel32 – 32-bitowymi.
- *ptr16:16*, *ptr16:32*: daleki wskaźnik. Notacja 16:16 wskazuje, która wartość wskaźnika ma 2 części. Wartość na lewo od dwukropka wskazuje segment kodu. Po prawej zaś stronie umieszczona jest wartość wskazująca offset kodu docelowego.
- *r8*: 1 bajt rejestrów AL, CL, DL, BL, AH, CH, DH i BH.
- *r16*: 1 słowo rejestrów AX, CX, DX, BX, SP, BP, SI i DI.

- *r32*: 1 z podwójnych słów rejestrów EAX, ECX, EDX, EBX, ESP, EBP, ESI i EDI.

- *i8*: natychmiastowa wartość bajtu. Jest to liczba ze znakiem pomiędzy -128 a +127 włącznie. Dla instrukcji, w której i8 jest połączona ze słowem lub podwójnym słowem operandu, wartość natychmiastowa (bezpośrednia) jest rozszerzoną formą znakową słowa lub podwójnego słowa. Górny bajt słowa jest wypełniony najwyższym bitem wartości natychmiastowej.

- *i16*: natychmiastowa wartość słowa dla instrukcji z operandem 16-bitowym. Mieści się w zakresie od -32768 do +32767.

- *i32*: natychmiastowa wartość podwójnego słowa dla instrukcji z operandem 32-bitowym. Mieści się w zakresie od +2147483647 do -2147483648.

- *r/m8*: 1-bajtowy operand, który stanowi zawartość rejestru (AL, BL, CL, DL, AH, BH, CH, DH) albo jest bajtem z pamięci.

- *r/m16*: słowo rejestru lub operand pamięci dla instrukcji z operandem 16-bitowym. Tymi rejestrami są AX, BX, CX, DX, SP, BP, SI, DI.

- *r/m32*: podwójne słowo rejestru lub operand pamięci dla instrukcji z operandem 32-bitowym. Tymi rejestrami są EAX, EBX, ECX, EDX, ESP, EBP, ESI, EDI.

- *r/m64*: poczwórne słowo rejestru lub operand pamięci dla instrukcji z operandem 64-bitowym.

- *m*: 16- lub 32-bitowy operand pamięci.

- *m8*: adresowany bajt pamięci poprzez DS:(E)SI lub ES:(E)DI; używane tylko w instrukcjach łańcuchowych.
- *m16*: adresowane słowo pamięci poprzez DS:(E)SI lub ES:(E)DI.
- *m32*: adresowane podwójne słowo pamięci poprzez DS:(E)SI lub ES:(E)DI.
- *m16*:*16*, *m16*:*32*: zawieranie operandu pamięci dalekiego wskaźnika składającego się z dwóch liczb – po lewej wskaźnik selektora segmentu, po prawej offsetu.
- *moffs8*, *moffs16*, *moffs32*: prosta zmienna pamięci typu BYTE, WORD, DWORD używana w instrukcji MOV. Liczba obok słowa moffs wskazuje rozmiar, który jest określony przez atrybut rozmiaru adresu instrukcji.
- Sreg: rejestr segmentowy. Wartości bitów wyznaczających segmenty rejestrowe: ES=0, CS=1, SS=2, DS=3, FS=4, GS=5.