Instructions for 382N

Opcode	Mnemonic	Description
04 *ib*	ADD AL,*imm8*	Add *imm8* to AL
05 *iw*	ADD AX,*imm16*	Add *imm16* to AX
05 *id*	ADD EAX,*imm32*	Add *imm32* to EAX
80 /0 *ib*	ADD *r/m8*,*imm8*	Add *imm8* to *r/m8*
81 /0 *iw*	ADD *r/m16*,*imm16*	Add *imm16* to *r/m16*
81 /0 *id*	ADD *r/m32*,*imm32*	Add *imm32* to *r/m32*
83 /0 *ib*	ADD *r/m16*,*imm8*	Add sign-extended *imm8* to *r/m16*
83 /0 *ib*	ADD *r/m32*,*imm8*	Add sign-extended *imm8* to *r/m32*
00 /*r*	ADD *r/m8*,*r8*	Add *r8* to *r/m8*
01 /*r*	ADD *r/m16*,*r16*	Add *r16* to *r/m16*
01 /*r*	ADD *r/m32*,*r32*	Add *r32* to *r/m32*
02 /*r*	ADD *r8*,*r/m8*	Add *r/m8* to *r8*
03 /*r*	ADD *r16*,*r/m16*	Add *r/m16* to *r16*
03 /*r*	ADD *r32*,*r/m32*	Add *r/m32* to *r32*
24 *ib*	AND AL,*imm8*	AL AND *imm8*
25 *iw*	AND AX,*imm16*	AX AND *imm16*
25 *id*	AND EAX,*imm32*	EAX AND *imm32*
80 /4 *ib*	AND *r/m8*,*imm8*	*r/m8* AND *imm8*
81 /4 *iw*	AND *r/m16*,*imm16*	*r/m16* AND *imm16*
81 /4 *id*	AND *r/m32*,*imm32*	*r/m32* AND *imm32*
83 /4 *ib*	AND *r/m16*,*imm8*	*r/m16* AND *imm8* (sign-extended)
83 /4 *ib*	AND *r/m32*,*imm8*	*r/m32* AND *imm8* (sign-extended)
20 /*r*	AND *r/m8*,*r8*	*r/m8* AND *r8*
21 /*r*	AND *r/m16*,*r16*	*r/m16* AND *r16*
21 /*r*	AND *r/m32*,*r32*	*r/m32* AND *r32*
22 /*r*	AND *r8*,*r/m8*	*r8* AND *r/m8*
23 /*r*	AND *r16*,*r/m16*	*r16* AND *r/m16*
23 /*r*	AND *r32*,*r/m32*	*r32* AND *r/m32*
E8 *cw*	CALL *rel16*	Call near, relative, displacement relative to next instruction
E8 *cd*	CALL *rel32*	Call near, relative, displacement relative to next instruction
FF /2	CALL *r/m16*	Call near, absolute indirect, address given in *r/m16*
FF /2	CALL *r/m32*	Call near, absolute indirect, address given in *r/m32*
9A *cp*	CALL *ptr16:32*	Call far, absolute, address given in operand
FC	CLD	Clear DF flag
0F 42 /*r*	CMOVC *r16*, *r/m16*	Move if carry (CF=1)

0F 42 */r*	CMOVC *r32*, *r/m32*	Move if carry (CF=1)
0F B0 */r*	CMPXCHG *r/m8*,*r8*	Compare AL with *r/m8*. If equal, *r/m8* = *r8* and ZF=1. Else, ZF=0, AL=*r/m8*.
0F B1 */r*	CMPXCHG *r/m16*,*r16*	Compare AX with *r/m16*. If equal, *r/m16* = *r16* and ZF=1. Else, ZF=0, AX=*r/m16*.
0F B1 */r*	CMPXCHG *r/m32*,*r32*	Compare EAX with *r/m32*. If equal, *r/m32* = *r32* and ZF=1. Else, ZF=0, EAX=*r/m32*.
F4	HLT	Halt
FE /0	INC *r/m8*	Increment *r/m* byte by 1
FF /0	INC *r/m16*	Increment *r/m* word by 1
FF /0	INC *r/m32*	Increment *r/m* doubleword by 1
40+ *rw*	INC *r16*	Increment word register by 1
40+ *rd*	INC *r32*	Increment doubleword register by 1
CF	IREtd	Interrupt return (32-bit operand size)
77 *cb*	JNBE *rel8*	Jump short if not below or equal (CF=0 and ZF=0)
75 *cb*	JNE *rel8*	Jump short if not equal (ZF=0)
0F 87 *cw*	JNBE *rel16*	Jump near if not below or equal (CF=0 and ZF=0)
0F 87 *cd*	JNBE *rel32*	Jump near if not below or equal (CF=0 and ZF=0)
0F 85 *cw*	JNE *rel16*	Jump near if not equal (ZF=0)
0F 85 *cd*	JNE *rel32*	Jump near if not equal (ZF=0)
EB *cb*	JMP *rel8*	Jump short, relative, displacement relative to next instruction
E9 *cw*	JMP *rel16*	Jump near, relative, displacement relative to next instruction
E9 *cd*	JMP *rel32*	Jump near, relative, displacement relative to next instruction
FF /4	JMP *r/m16*	Jump near, absolute indirect, address given in *r/m16*
FF /4	JMP *r/m32*	Jump near, absolute indirect, address given in *r/m32*
EA *cd*	JMP *ptr16:16*	Jump far, absolute, address given in operand
EA *cp*	JMP *ptr16:32*	Jump far, absolute, address given in operand
88 */r*	MOV *r/m8*,*r8*	Move *r8* to *r/m8*
89 */r*	MOV *r/m16*,*r16*	Move *r16* to *r/m16*
89 */r*	MOV *r/m32*,*r32*	Move *r32* to *r/m32*
8A */r*	MOV *r8*,*r/m8*	Move *r/m8* to *r8*
8B */r*	MOV *r16*,*r/m16*	Move *r/m16* to *r16*
8B */r*	MOV *r32*,*r/m32*	Move *r/m32* to *r32*
8C */r*	MOV *r/m16*,*Sreg*	Move segment register to *r/m16*
8E */r*	MOV *Sreg*,*r/m16*	Move *r/m16* to segment register
B0+ *rb*	MOV *r8*,*imm8*	Move *imm8* to *r8*
B8+ *rw*	MOV *r16*,*imm16*	Move *imm16* to *r16*
B8+ *rd*	MOV *r32*,*imm32*	Move *imm32* to *r32*
C6 /0	MOV *r/m8*,*imm8*	Move *imm8* to *r/m8*
C7 /0	MOV *r/m16*,*imm16*	Move *imm16* to *r/m16*

C7 /0	MOV *r/m32*,*imm32*	Move *imm32* to *r/m32*
0F 6F /*r*	MOVQ *mm*, *mm/m64*	Move quadword from *mm/m64* to *mm*.
0F 7F /*r*	MOVQ *mm/m64*, *mm*	Move quadword from *mm* to *mm/m64*.
A4	MOVS *m8*, *m8*	Move byte at address DS:(E)SI to address ES:(E)DI
A5	MOVS *m16*, *m16*	Move word at address DS:(E)SI to address ES:(E)DI
A5	MOVS *m32*, *m32*	Move doubleword at address DS:(E)SI to address ES:(E)DI
F6 /2	NOT *r/m8*	Reverse each bit of *r/m8*
F7 /2	NOT *r/m16*	Reverse each bit of *r/m16*
F7 /2	NOT *r/m32*	Reverse each bit of *r/m32*
0F FD /*r*	PADDW *mm*, *mm/m64*	Add packed word integers from *mm/m64* and *mm*.
0F FE /*r*	PADDD *mm*, *mm/m64*	Add packed doubleword integers from *mm/m64* and *mm*.
0F ED /*r*	PADDSW *mm*, *mm/m64*	Add packed signed word integers from *mm/m64* and *mm* and saturate the results.
58+ *rw*	POP *r16*	Pop top of stack into *r16*; increment stack pointer
58+ *rd*	POP *r32*	Pop top of stack into *r32*; increment stack pointer
1F	POP DS	Pop top of stack into DS; increment stack pointer
07	POP ES	Pop top of stack into ES; increment stack pointer
17	POP SS	Pop top of stack into SS; increment stack pointer
0F A1	POP FS	Pop top of stack into FS; increment stack pointer
0F A9	POP GS	Pop top of stack into GS; increment stack pointer
0F 70 /*r ib*	PSHUFW *mm1*, *mm2*/*mm64*, *imm8*	Shuffle the words in *mm2*/*m64* based on the encoding in *imm8* and store result in *mm1*.
FF /6	PUSH *r/m16*	Push *r/m16*
FF /6	PUSH *r/m32*	Push *r/m32*
50+*rw*	PUSH *r16*	Push *r16*
50+*rd*	PUSH *r32*	Push *r32*
6A	PUSH *imm8*	Push *imm8*
68	PUSH *imm16*	Push *imm16*
68	PUSH *imm32*	Push *imm32*
0E	PUSH CS	Push CS
16	PUSH SS	Push SS
1E	PUSH DS	Push DS
06	PUSH ES	Push ES
0F A0	PUSH FS	Push FS
0F A8	PUSH GS	Push GS
F3 A4	REP MOVS *m8*,*m8*	Move (E)CX bytes from DS:[(E)SI] to ES:[(E)DI]
F3 A5	REP MOVS *m16*,*m16*	Move (E)CX words from DS:[(E)SI] to ES:[(E)DI]
F3 A5	REP MOVS *m32*,*m32*	Move (E)CX doublewords from DS:[(E)SI] to ES:[(E)DI]
C3	RET	Near return to calling procedure

CB	RET	Far return to calling procedure
C2 *iw*	RET *imm16*	Near return to calling procedure and pop *imm16* bytes from stack
CA *iw*	RET *imm16*	Far return to calling procedure and pop *imm16* bytes from stack
D0 /4	SAL *r/m8*,1	Multiply *r/m8* by 2, once
D2 /4	SAL *r/m8*,CL	Multiply *r/m8* by 2, CL times
C0 /4 *ib*	SAL *r/m8*,*imm8*	Multiply *r/m8* by 2, *imm8* times
D1 /4	SAL *r/m16*,1	Multiply *r/m16* by 2, once
D3 /4	SAL *r/m16*,CL	Multiply *r/m16* by 2, CL times
C1 /4 *ib*	SAL *r/m16*,*imm8*	Multiply *r/m16* by 2, *imm8* times
D1 /4	SAL *r/m32*,1	Multiply *r/m32* by 2, once
D3 /4	SAL *r/m32*,CL	Multiply *r/m32* by 2, CL times
C1 /4 *ib*	SAL *r/m32*,*imm8*	Multiply *r/m32* by 2, *imm8* times
D0 /7	SAR *r/m8*,1	Signed divide *r/m8* by 2, once
D2 /7	SAR *r/m8*,CL	Signed divide *r/m8* by 2, CL times
C0 /7 *ib*	SAR *r/m8*,*imm8*	Signed divide *r/m8* by 2, *imm8* times
D1 /7	SAR *r/m16*,1	Signed divide *r/m16* by 2, once
D3 /7	SAR *r/m16*,CL	Signed divide *r/m16* by 2, CL times
C1 /7 *ib*	SAR *r/m16*,*imm8*	Signed divide *r/m16* by 2, *imm8* times
D1 /7	SAR *r/m32*,1	Signed divide *r/m32* by 2, once
D3 /7	SAR *r/m32*,CL	Signed divide *r/m32* by 2, CL times
C1 /7 *ib*	SAR *r/m32*,*imm8*	Signed divide *r/m32* by 2, *imm8* times
FD	STD	Set DF flag
90+*rw*	XCHG AX,*r16*	Exchange *r16* with AX
90+*rd*	XCHG EAX,*r32*	Exchange *r32* with EAX
86 /*r*	XCHG *r/m8*,*r8*	Exchange *r8* with *r/m8*
87 /*r*	XCHG *r/m16*,*r16*	Exchange *r16* with *r/m16*
87 /*r*	XCHG *r/m32*,*r32*	Exchange *r32* with *r/m32*