
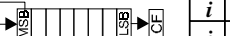
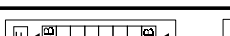
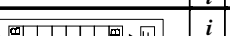
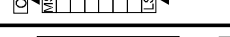


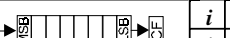


TRANSFER				Flags								
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C
MOV	Move (copy)	MOV Source, Dest	Dest := Source									
XCHG	Exchange	XCHG S <sub>2</sub> , S <sub>1</sub>	S <sub>2</sub> := S <sub>1</sub> , S <sub>1</sub> := S <sub>2</sub>									
STC	Set Carry	STC	CF := 1									1
CLC	Clear Carry	CLC	CF := 0									0
CMC	Complement Carry	CMC	CF := ¬ CF									±
STD	Set Direction	STD	DF := 1 (string op's downwards)		1							
CLD	Clear Direction	CLD	DF := 0 (string op's upwards)		0							
STI	Set Interrupt	STI	IF := 1			1						
CLI	Clear Interrupt	CLI	IF := 0			0						
PUSH	Push onto stack	PUSH Source	DEC SP, [SP] := Source									
PUSHF	Push flags	PUSHF	O, D, I, T, S, Z, A, P, C 286+: also NT, IOPL									
PUSHA	Push all general registers	PUSHA	AX, CX, DX, BX, SP, BP, SI, DI									
POP	Pop from stack	POP Dest	Dest := [SP], INC SP									
POPF	Pop flags	POPF	O, D, I, T, S, Z, A, P, C 286+: also NT, IOPL	±	±	±	±	±	±	±	±	±
POPA	Pop all general registers	POPA	DI, SI, BP, SP, BX, DX, CX, AX									
CBW	Convert byte to word	CBW	AX := AL (signed)									
CWD	Convert word to double	CWD	DX:AX := AX (signed)	±				±	±	±	±	±
CWDE	Conv word extended double	CWDE 386	EAX := AX (signed)									
IN <i>i</i>	Input	IN Dest, Port	AL/AX/EAX := byte/word/double of specified port									
OUT <i>i</i>	Output	OUT Port, Source	Byte/word/double of specified port := AL/AX/EAX									

*i* for more information see instruction specifications      Flags: ±=affected by this instruction ? = undefined after this instruction

ARITHMETIC				Flags								
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C
ADD	Add	ADD Source, Dest	Dest := Dest + Source	±					±	±	±	±
ADC	Add with Carry	ADC Source, Dest	Dest := Dest + Source + CF	±					±	±	±	±
SUB	Subtract	SUB Source, Dest	Dest := Dest - Source	±					±	±	±	±
SBB	Subtract with borrow	SBB Source, Dest	Dest := Dest - ( Source + CF )	±					±	±	±	±
DIV	Divide (unsigned)	DIV Op	Op = byte: AL := AX / Op      AH := Rest	?					?	?	?	?
DIV	Divide (unsigned)	DIV Op	Op = word: AX := DX:AX / Op      DX := Rest	?					?	?	?	?
DIV 386	Divide (unsigned)	DIV Op	Op = doublew.: EAX := EDX:EAX / Op      EDX := Rest	?					?	?	?	?
IDIV	Signed Integer Divide	IDIV Op	Op = byte: AL := AX / Op      AH := Rest	?					?	?	?	?
IDIV	Signed Integer Divide	IDIV Op	Op = word: AX := DX:AX / Op      DX := Rest	?					?	?	?	?
IDIV 386	Signed Integer Divide	IDIV Op	Op = doublew.: EAX := EDX:EAX / Op      EDX := Rest	?					?	?	?	?
MUL	Multiply (unsigned)	MUL Op	Op = byte: AX := AL * Op      if AH=0 ♦	±					?	?	?	±
MUL	Multiply (unsigned)	MUL Op	Op = word: DX:AX := AX * Op      if DX=0 ♦	±					?	?	?	±
MUL 386	Multiply (unsigned)	MUL Op	Op = double: EDX:EAX := EAX * Op      if EDX=0 ♦	±					?	?	?	±
IMUL <i>i</i>	Signed Integer Multiply	IMUL Op	Op = byte: AX := AL * Op      if AL sufficient ♦	±					?	?	?	±
IMUL	Signed Integer Multiply	IMUL Op	Op = word: DX:AX := AX * Op      if AX sufficient ♦	±					?	?	?	±
IMUL 386	Signed Integer Multiply	IMUL Op	Op = double: EDX:EAX := EAX * Op if EAX sufficient ♦	±					?	?	?	±
INC	Increment	INC Op	Op := Op + 1 (Carry not affected !)	±					±	±	±	±
DEC	Decrement	DEC Op	Op := Op - 1 (Carry not affected !)	±					±	±	±	±
CMP	Compare	CMP S <sub>2</sub> , S <sub>1</sub>	S <sub>2</sub> - S <sub>1</sub>	±					±	±	±	±
SAL	Shift arithmetic left (= SHL)	SAL Quantity, Dest		<i>i</i>					±	±	?	±
SAR	Shift arithmetic right	SAR Quantity, Dest		<i>i</i>					±	±	?	±
RCL	Rotate left through Carry	RCL Quantity, Dest		<i>i</i>								±
RCR	Rotate right through Carry	RCR Quantity, Dest		<i>i</i>								±
ROL	Rotate left	ROL Quantity, Dest		<i>i</i>								±
ROR	Rotate right	ROR Quantity, Dest		<i>i</i>								±

*i* for more information see instruction specifications      ♦ then CF := 0, OF := 0 else CF := 1, OF := 1

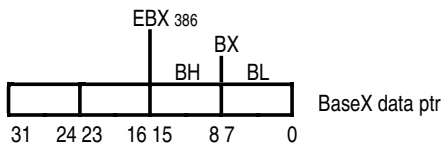
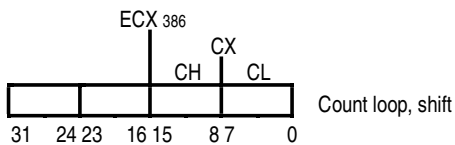
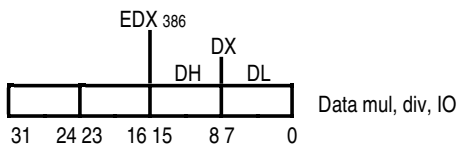
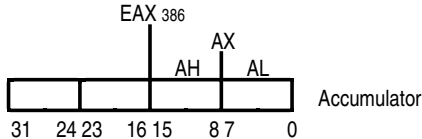
LOGIC				Flags								
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C
NEG	Negate (two-complement)	NEG Op	Op := 0 - Op    if Op = 0 then CF := 0 else CF := 1	±					±	±	±	±
NOT	Invert each bit	NOT Op	Op := ¬ Op (invert each bit)									
AND	Logical and	AND Source, Dest	Dest := Dest ∧ Source	0					±	±	?	0
OR	Logical or	OR Source, Dest	Dest := Dest ∨ Source	0					±	±	?	0
XOR	Logical exclusive or	XOR Source, Dest	Dest := Dest (exor) Source	0					±	±	?	0
SHL	Shift logical left (= SAL)	SHL Quantity, Dest		<i>i</i>					±	±	?	±
SHR	Shift logical right	SHR Quantity, Dest		<i>i</i>					±	±	?	±

MISC				FLAGS								
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C
NOP	No operation	NOP	No Operation									
LEA	Load effective address	LEA Source, Dest	Dest := address of Source									
INT	Interrupt	INT Nr	Interrupts current program, runs spec. int-program			0	1					

JUMPS (flags remain unchanged)							
Name	Comment	Code	Operation	Name	Comment	Code	Operation
CALL	Call subroutine	CALL Proc		RET	Return from subroutine	RET	
JMP	Jump	JMP Dest					
JE	Jump if Equal	JE Dest	(= JZ)	JNE	Jump if not Equal	JNE Dest	(= JNZ)
JZ	Jump if Zero	JZ Dest	(= JE)	JNZ	Jump if not Zero	JNZ Dest	(= JNE)
JCXZ	Jump if CX Zero	JCXZ Dest		JECXZ	Jump if ECX Zero	JECXZ Dest	386
JP	Jump if Parity (Parity Even)	JP Dest	(= JPE)	JNP	Jump if no Parity (Parity Odd)	JNP Dest	(= JPO)
JPE	Jump if Parity Even	JPE Dest	(= JP)	JPO	Jump if Parity Odd	JPO Dest	(= JNP)

JUMPS Unsigned (Cardinal)				JUMPS Signed (Integer)			
JA	Jump if Above	JA Dest	(= JNBE)	JG	Jump if Greater	JG Dest	(= JNLE)
JAЕ	Jump if Above or Equal	JAЕ Dest	(= JNB = JNC)	JGE	Jump if Greater or Equal	JGE Dest	(= JNL)
JB	Jump if Below	JB Dest	(= JNAE = JC)	JL	Jump if Less	JL Dest	(= JNGE)
JBE	Jump if Below or Equal	JBE Dest	(= JNA)	JLE	Jump if Less or Equal	JLE Dest	(= JNG)
JNA	Jump if not Above	JNA Dest	(= JBE)	JNG	Jump if not Greater	JNG Dest	(= JLE)
JNAE	Jump if not Above or Equal	JNAE Dest	(= JB = JC)	JNGE	Jump if not Greater or Equal	JNGE Dest	(= JL)
JNB	Jump if not Below	JNB Dest	(= JAE = JNC)	JNL	Jump if not Less	JNL Dest	(= JGE)
JNBE	Jump if not Below or Equal	JNBE Dest	(= JA)	JNLE	Jump if not Less or Equal	JNLE Dest	(= JG)
JC	Jump if Carry	JC Dest		JO	Jump if Overflow	JO Dest	
JNC	Jump if no Carry	JNC Dest		JNO	Jump if no Overflow	JNO Dest	
				JS	Jump if Sign (= negative)	JS Dest	
				JNS	Jump if no Sign (= positive)	JNS Dest	

General Registers:



Example:

```

.DOSSEG ; Demo program
.MODEL SMALL
.STACK 1024
Two EQU 2 ; Const
.DATA
VarB DB ? ; define Byte, any value
VarW DW 1010b ; define Word, binary
VarW2 DW 257 ; define Word, decimal
VarD DD 0AFFFFh ; define Doubleword, hex
S DB "Hello !", 0 ; define String
.CODE
main: MOV AX, DGROUP ; resolved by linker
MOV DS, AX ; init datasegment reg
MOV [VarB], 42 ; init VarB
MOV [VarD], -7 ; set VarD
MOV BX, Offset[S] ; addr of "H" of "Hello !"
MOV AX, [VarW] ; get value into accumulator
ADD AX, [VarW2] ; add VarW2 to AX
MOV [VarW2], AX ; store AX in VarW2
MOV AX, 4C00h ; back to system
INT 21h
END main
    
```



Flags:

**Control Flags** (how instructions are carried out):  
 D: Direction 1 = string op's process down from high to low address  
 I: Interrupt whether interrupts can occur. 1= enabled  
 T: Trap single step for debugging

**Status Flags** (result of operations):

C: Carry result of unsigned op. is too large or below zero. 1 = carry/borrow  
 O: Overflow result of signed op. is too large or small. 1 = overflow/underflow  
 S: Sign sign of result. Reasonable for Integer only. 1 = neg. / 0 = pos.  
 Z: Zero result of operation is zero. 1 = zero  
 A: Aux. carry similar to Carry but restricted to the low nibble only  
 P: Parity 1 = result has even number of set bits