intel®

Table 2. Instruction Set Summary

Mnemonic and Description	Instruction Code			
DATA TRANSFER				
MOV = Move (|ODITSZAPC| = |--------|)	76543210	76543210	76543210	76543210
Register/Memory to/from Register	100010dw	mod reg r/m		
Immediate to Register/Memory	1100011w	mod 000 r/m	data	data if w = 1
Immediate to Register	1011 w reg	data	data if w = 1	
Memory to Accumulator	1010000w	addr-low	addr-high	
Accumulator to Memory	1010001w	addr-low	addr-high	
Register/Memory to Segment Register	10001110	mod 0 reg r/m		
Segment Register to Register/Memory	10001100	mod 0 reg r/m		
PUSH = Push: (|ODITSZAPC| = |--------|)				
Register/Memory	11111111	mod 110 r/m		
Register	01010 reg			
Segment Register	000 reg 110			
POP = Pop: (|ODITSZAPC| = |--------|)				
Register/Memory	10001111	mod 000 r/m		
Register	01011 reg			
Segment Register	000 reg 111			
XCHG = Exchange: (|ODITSZAPC| = |--------|)				
Register/Memory with Register	1000011w	mod reg r/m		
Register with Accumulator	10010 reg			
IN = Input from: (|ODITSZAPC| = |--------|)				
Fixed Port	1110010w	port		
Variable Port	1110110w			
OUT = Output to: (|ODITSZAPC| = |--------|)				
Fixed Port	1110011w	port		
Variable Port	1110111w			
XLAT = Translate Byte to AL	11010111		(|ODITSZAPC| = |--------|)	
LEA = Load EA to Register	10001101	mod reg r/m	(|ODITSZAPC| = |--------|)	
LDS = Load Pointer to DS	11000101	mod reg r/m	(|ODITSZAPC| = |--------|)	
LES = Load Pointer to ES	11000100	mod reg r/m	(|ODITSZAPC| = |--------|)	
LAHF = Load AH with Flags	10011111	(|ODITSZAPC| = |--------|)		
SAHF = Store AH into Flags	10011110	(|ODITSZAPC| = |----*****|)		
PUSHF = Push Flags	10011100	(|ODITSZAPC| = |--------|)		
POPF = Pop Flags	10011101	(|ODITSZAPC| = |*********|)		

Mnemonic and Description	Instruction Code			
ARITHMETIC	76543210	76543210	76543210	76543210
ADD = Add: (|ODITSZAPC| = |*---*****|)				
Reg./Memory with Register to Either	000000dw	mod reg r/m		
Immediate to Register/Memory	100000sw	mod 000 r/m	data	data if s: w = 01
Immediate to Accumulator	0000010w	data	data if w = 1	
ADC = Add with Carry: (|ODITSZAPC| = |*---*****|)				
Reg./Memory with Register to Either	000100dw	mod reg r/m		
Immediate to Register/Memory	100000sw	mod 010 r/m	data	data if s: w = 01
Immediate to Accumulator	0001010w	data	data if w = 1	
INC = Increment: (|ODITSZAPC| = |*---****-|)				
Register/Memory	1111111w	mod 000 r/m		
Register	01000 reg			
AAA = ASCII Adjust for Add	00110111		(|ODITSZAPC| = |?---??*?*|)	
DAA = Decimal Adjust for Add	00100111		(|ODITSZAPC| = |?---*****|)	
SUB = Subtract: (|ODITSZAPC| = |*---*****|)				
Reg./Memory and Register to Either	001010dw	mod reg r/m		
Immediate from Register/Memory	100000sw	mod 101 r/m	data	data if s w = 01
Immediate from Accumulator	0010110w	data	data if w = 1	
SBB = Subtract with Borrow: (|ODITSZAPC| = |*---*****|)				
Reg./Memory and Register to Either	000110dw	mod reg r/m		
Immediate from Register/Memory	100000sw	mod 011 r/m	data	data if s w = 01
Immediate from Accumulator	0001111w	data	data if w = 1	
DEC = Decrement: (|ODITSZAPC| = |*---****-|)				
Register/memory	1111111w	mod 001 r/m		
Register	01001 reg			
NEG = Change sign	1111011w	mod 011 r/m	(|ODITSZAPC| = |*---*****|)	
CMP = Compare: (|ODITSZAPC| = |*---*****|)				
Register/Memory and Register	001110dw	mod reg r/m		
Immediate with Register/Memory	100000sw	mod 111 r/m	data	data if s w = 01
Immediate with Accumulator	0011110w	data	data if w = 1	
AAS = ASCII Adjust for Subtract	00111111		(|ODITSZAPC| = |?---??*?*|)	
DAS = Decimal Adjust for Subtract	00101111		(|ODITSZAPC| = |?---*****|)	
MUL = Multiply (Unsigned)	1111011w	mod 100 r/m	(|ODITSZAPC| = |*---????*|)	
IMUL = Integer Multiply (Signed)	1111011w	mod 101 r/m	(|ODITSZAPC| = |*---????*|)	
AAM = ASCII Adjust for Multiply	11010100	00001010	(|ODITSZAPC| = |?---**?*?|)	
DIV = Divide (Unsigned)	1111011w	mod 110 r/m	(|ODITSZAPC| = |?---?????|)	
IDIV = Integer Divide (Signed)	1111011w	mod 111 r/m	(|ODITSZAPC| = |?---?????|)	
AAD = ASCII Adjust for Divide	11010101	00001010	(|ODITSZAPC| = |?---**?*?|)	
CBW = Convert Byte to Word	10011000		(|ODITSZAPC| = |--------|)	
CWD = Convert Word to Double Word	10011001		(|ODITSZAPC| = |--------|)	

Mnemonic and Description	Instruction Code			
LOGIC	76543210	76543210	76543210	76543210
NOT = Invert	1111011w	mod 010 r/m	(|ODITSZAPC| = |--------|)	
SHL/SAL = Shift Logical/Arithmetic Left	110100vw	mod 100 r/m	(|ODITSZAPC| = |*---**?**|)	
SHR = Shift Logical Right	110100vw	mod 101 r/m	(|ODITSZAPC| = |*---**?**|)	
SAR = Shift Arithmetic Right	110100vw	mod 111 r/m	(|ODITSZAPC| = |*---**?**|)	
ROL = Rotate Left	110100vw	mod 000 r/m	(|ODITSZAPC| = |-------- |)	
ROR = Rotate Right	110100vw	mod 001 r/m	(|ODITSZAPC| = |*-------*|)	
RCL = Rotate Through Carry Flag Left	110100vw	mod 010 r/m	(|ODITSZAPC| = |*-------*|)	
RCR = Rotate Through Carry Right	110100vw	mod 011 r/m	(|ODITSZAPC| = |*-------*|)	
AND = And: (|ODITSZAPC| = |*---**?**|)				
Reg./Memory and Register to Either	001000dw	mod reg r/m		
Immediate to Register/Memory	1000000w	mod 100 r/m	data	data if w = 1
Immediate to Accumulator	0010010w	data	data if w = 1	
TEST = And Function to Flags, No Result: (|ODITSZAPC| = |*---**?**|)				
Register/Memory and Register	1000010w	mod reg r/m		
Immediate Data and Register/Memory	1111011w	mod 000 r/m	data	data if w = 1
Immediate Data and Accumulator	1010100w	data	data if w = 1	
OR = Or: (|ODITSZAPC| = |*---**?**|)				
Reg./Memory and Register to Either	000010dw	mod reg r/m		
Immediate to Register/Memory	1000000w	mod 001 r/m	data	data if w = 1
Immediate to Accumulator	0000110w	data	data if w = 1	
XOR = Exclusive or: (|ODITSZAPC| = |*---**?**|)				
Reg./Memory and Register to Either	001100dw	mod reg r/m		
Immediate to Register/Memory	1000000w	mod 110 r/m	data	data if w = 1
Immediate to Accumulator	0011010w	data	data if w = 1	
STRING MANIPULATION				
REP = Repeat	1111001z	(|ODITSZAPC| = |--------|)		
MOVS = Move Byte/Word	1010010w	(|ODITSZAPC| = |--------|)		
CMPS = Compare Byte/Word	1010011w	(|ODITSZAPC| = |*---*****|)		
SCAS = Scan Byte/Word	1010111w	(|ODITSZAPC| = |*---*****|)		
LODS = Load Byte/Wd to AL/AX	1010110w	(|ODITSZAPC| = |--------|)		
STOS = Stor Byte/Wd from AL/A	1010101w	(|ODITSZAPC| = |--------|)		
CONTROL TRANSFER				
CALL = Call: (|ODITSZAPC| = |--------|)				
Direct within Segment	11101000	disp-low	disp-high	
Indirect within Segment	11111111	mod 010 r/m		
Direct Intersegment	10011010	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	11111111	mod 011 r/m		

Mnemonic and Description	Instruction Code			
JMP = Unconditional Jump: (|--------|)	76543210	76543210	76543210	
Direct within Segment	11101001	disp-low	disp-high	
Direct within Segment-Short	11101011	disp		
Indirect within Segment	11111111	mod 100 r/m		
Direct Intersegment	11101010	offset-low	offset-high	
		seg-low	seg-high	
Indirect Intersegment	11111111	mod 101 r/m		
RET = Return from CALL: (|ODITSZAPC| = |--------|)				
Within Segment	11000011			
Within Seg Adding Immed to SP	11000010	data-low	data-high	
Intersegment	11001011			
Intersegment Adding Immediate to SP	11001010	data-low	data-high	
JE/JZ = Jump on Equal/Zero	01110100	disp	(|ODITSZAPC| = |--------|)	
JL/JNGE = Jump on Less/Not Greater or Equal	01111100	disp		
JLE/JNG = Jump on Less or Equal/ Not Greater	01111110	disp		
JB/JNAE = Jump on Below/Not Above or Equal	01110010	disp		
JBE/JNA = Jump on Below or Equal/ Not Above	01110110	disp		
JP/JPE = Jump on Parity/Parity Even	01111010	disp		
JO = Jump on Overflow	01110000	disp		
JS = Jump on Sign	01111000	disp		
JNE/JNZ = Jump on Not Equal/Not Zero	01110101	disp		
JNL/JGE = Jump on Not Less/Greater or Equal	01111101	disp		
JNLE/JG = Jump on Not Less or Equal/ Greater	01111111	disp		
JNB/JAE = Jump on Not Below/Above or Equal	01110011	disp		
JNBE/JA = Jump on Not Below or Equal/Above	01110111	disp		
JNP/JPO = Jump on Not Par/Par Odd	01111011	disp		
JNO = Jump on Not Overflow	01110001	disp		
JNS = Jump on Not Sign	01111001	disp		
LOOP = Loop CX Times	11100010	disp	(|ODITSZAPC| = |--------|)	
LOOPZ/LOOPE = Loop While Zero/Equal	11100001	disp		
LOOPNZ/LOOPNE = Loop While Not Zero/Equal	11100000	disp		
JCXZ = Jump on CX Zero	11100011	disp	(|ODITSZAPC| = |--------|)	
INT = Interrupt: (|ODITSZAPC| = |--00-----|)				
Type Specified	11001101	type		
Type 3	11001100			
INTO = Interrupt on Overflow	11001110	(|ODITSZAPC| = |--**-----|)		
IRET = Interrupt Return	11001111	(|ODITSZAPC| = |*********|)		

Mnemonic and Description	Instruction Code	
	76543210	76543210
PROCESSOR CONTROL		
CLC = Clear Carry	11111000	(|ODITSZAPC| = |--------0|)
CMC = Complement Carry	11110101	(|ODITSZAPC| = |--------*|)
STC = Set Carry	11111001	(|ODITSZAPC| = |--------1|)
CLD = Clear Direction	11111100	(|ODITSZAPC| = |-0-------|)
STD = Set Direction	11111101	(|ODITSZAPC| = |-1-------|)
CLI = Clear Interrupt	11111010	(|ODITSZAPC| = |--0------|)
STI = Set Interrupt	11111011	(|ODITSZAPC| = |--1------|)
HLT = Halt	11110100	(|ODITSZAPC| = |---------|)
WAIT = Wait	10011011	(|ODITSZAPC| = |---------|)
ESC = Escape (to External Device)	11011xxx	mod x x x r/m (|ODITSZAPC| = |---------|)
LOCK = Bus Lock Prefix	11110000	(|ODITSZAPC| = |---------|)

NOTES:
AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive;
Less = less positive (more negative) signed values
if d = 1 then "to" reg; if d = 0 then "from" reg
if w = 1 then word instruction; if w = 0 then byte instruction
if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16 bits, disp-high is absent
if mod = 10 then DISP = disp-high; disp-low
if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP
DISP follows 2nd byte of instruction (before data if required)
*except if mod = 00 and r/m = 110 then EA = disp-high; disp-low.

if s w = 01 then 16 bits of immediate data form the operand
if s w = 11 then an immediate data byte is sign extended to form the 16-bit operand
if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
x = don't care
z is used for string primitives for comparison with ZF FLAG

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)	Segment
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

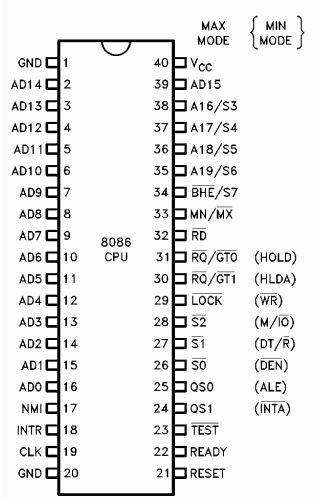
Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:
FLAGS = X:X:X:X:(OF):(DF):(IF):(TF):(SF):(ZF):X:(AF):X:(PF):X:(CF)

The codes used for 'Affected Flags' column are: |-*01? | Unaffected/Affected/Reset/Set/Unknown

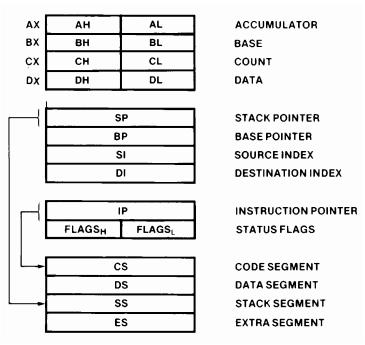
NOTES:

Tabulated and Edited by Dr. Mohammed Hawa, University of Jordan.
Version 1.1, June, 2005.

Pin Diagram

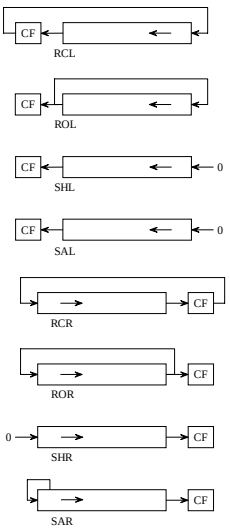


8086 Registers



Intentionally Blank

Rotate Instructions



Conditional Jump Instructions

Instruction	Description	Condition	Aliases	Opposite
JC	Jump if carry	Carry = 1	JB, JNAE	JNC
JNC	Jump if no carry	Carry = 0	JNB, JAE	JC
JZ	Jump if zero	Zero = 1	JE	JNZ
JNZ	Jump if not zero	Zero = 0	JNE	JZ
JS	Jump if sign	Sign = 1	-	JNS
JNS	Jump if no sign	Sign = 0	-	JS
JO	Jump if overflow	Overflow = 1	-	JNO
JNO	Jump if no overflow	Overflow = 0	-	JO
JP	Jump if parity	Parity = 1	JPE	JNP
JPE	Jump if parity even	Parity = 1	JP	JPO
JNP	Jump if no parity	Parity = 0	JPO	JP
JPO	Jump if parity odd	Parity = 0	JNP	JPE

Unsigned Comparisons

Instruction	Description	Condition	Aliases	Opposite
JA	Jump if above (>)	Carry = 0, Zero = 0	JNBE	JNA
JNBE	Jump if not below nor equal (not <=)	Carry = 0, Zero = 0	JA	JBE
JAE	Jump if above or equal (>=)	Carry = 0	JNC, JNB	JNAE
JNB	Jump if not below (not <)	Carry = 0	JNC, JAE	JB
JB	Jump if below (<)	Carry = 1	JC, JNAE	JNB
JNAE	Jump if not above nor equal (not >=)	Carry = 1	JC, JB	JAE
JBE	Jump if below or equal (<=)	Carry = 1 or Zero = 1	JNA	JNBE
JNA	Jump if not above (not >)	Carry = 1 or Zero = 1	JBE	JA
JE	Jump if equal (=)	Zero = 1	JZ	JNE
JNE	Jump if not equal ()	Zero = 0	JNZ	JE

Signed Comparisons

Instruction	Description	Condition	Aliases	Opposite
JG	Jump if greater (>)	Sign = Overflow or Zero = 0	JNLE	JNG
JNLE	Jump if not less than nor equal (not <=)	Sign = Overflow or Zero = 0	JG	JLE
JGE	Jump if greater than or equal (>=)	Sign = Overflow	JNL	JGE
JNL	Jump if not less than (not <)	Sign = Overflow	JGE	JL
JL	Jump if less than (<)	Sign Overflow	JNGE	JNL
JNGE	Jump if not greater nor equal (not >=)	Sign Overflow	JL	JGE
JLE	Jump if less than or equal (<=)	Sign Overflow or Zero = 1	JNG	JNLE
JNG	Jump if not greater than (not >)	Sign Overflow or Zero = 1	JLE	JG
JE	Jump if equal (=)	Zero = 1	JZ	JNE
JNE	Jump if not equal ()	Zero = 0	JNZ	JE

8086 Flags Register

```
|ODITSZAPC|    Overflow Flag, Direction Flag, Interrupt Flag, Trap Flag,
               Sign Flag, Zero Flag, Auxilary carry Flag, Parity Flag,
               Carry Flag
```

ASCII Character Set

b_4 b_3 b_2 b_1	H_1 / H_0	0	1	2	3	4	5	6	7
	b_7	0	0	0	0	1	1	1	1
	b_6	0	0	1	1	0	0	1	1
	b_5	0	1	0	1	0	1	0	1
0 0 0 0	0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2	STX	DC2	"	2	B	R	b	r
0 0 1 1	3	ETX	DC3	#	3	C	S	c	s
0 1 0 0	4	EOT	DC4	$	4	D	T	d	t
0 1 0 1	5	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6	ACK	SYN	&	6	F	V	f	v
0 1 1 1	7	BEL	ETB	'	7	G	W	g	w
1 0 0 0	8	BS	CAN	(	8	H	X	h	x
1 0 0 1	9	HT	EM	)	9	I	Y	i	y
1 0 1 0	A	LF	SUB	*	:	J	Z	j	z
1 0 1 1	B	V	ESC	+	;	K	[	k	{
1 1 0 0	C	FF	FS	,	<	L	\	l	|
1 1 0 1	D	CR	GS	-	=	M	]	m	}
1 1 1 0	E	SO	RS	.	>	N	^	n	~
1 1 1 1	F	SI	US	/	?	O	-	o	DEL

Intentionally Blank

Assembler Directives

ALIGN	Align to word boundary
ASSUME sr:sy(,...)	Assume segment register name(s)
ASSUME NOTHING	Remove all former assumptions
DB e(,...)	Define Byte(s)
DBS e	Define Byte Storage
DD e(,...)	Define Double Word(s)
DDS e	Define Double Word Storage
DW e(,...)	Define Word(s)
DWS e	Define Word Storage
EXT (sr:)sy(t)	External(s)(t=ABS/BYTE/DWORD/FAR/NEAR/WORD)
LABEL t	Label (t=BYTE/DWORD/FAR/NEAR/WORD)
PROC t	Procedure (t=FAR/NEAR, default NEAR)
ABS	Absolute value of operand
BYTE	Byte type operation
DWORD	Double Word operation
FAR	IP and CS registers altered
HIGH	High-order 8 bits of 16-bit value
LENGTH	Number of basic units
LOW	Low-order 8 bit of 16-bit value
NEAR	Only IP register need be altered
OFFSET	Offset portion of an address
PTR	Create a variable or label
SEG	Segment of address
SHORT	One byte for a JMP operation
SIZE	Number of bytes defined by statement
THIS	Create a variable/label of specified type
TYPE	Number of bytes in the unit defined
WORD	Word operation