MICRO LOGIC CORP.
HACKENSACK, NJ

8086 & 8088
MICROPROCESSOR INSTANT REFERENCE CARD

MICRO CHART®

Instruction Set

INST	ADDR	HEX	C
AAA	none	37	M2
AAD	none	D5,0A	B1
AAM	none	D4,0A	C1
AAS	none	3F	M2
ADC	r,r	10,X8	D1
ADC	m,r	10,X8	E1
ADC	rr,rr	11,X9	D1
ADC	mm,rr	11,X9	F1
ADC	r,m	12,8X	G1
ADC	rr,mm	13,9X	H1
ADC	r,i	80,X2	A1
ADC	m,i	80,X2	I1
ADC	rr,ii	81,X2	A1
ADC	mm,ii	81,X2	J1
ADC	rr,i	83,X2	A1
ADC	mm,i	83,X2	J1
ADC	AL,i	14	A1
ADC	AX,ii	15	A1
ADD	r,r	00,X8	D1
ADD	m,r	00,X8	E1
ADD	rr,rr	01,X9	D1
ADD	mm,rr	01,X9	F1
ADD	r,m	02,8X	G1
ADD	rr,mm	03,9X	H1
ADD	r,i	80,X0	A1
ADD	m,i	80,X0	I1
ADD	rr,ii	81,X0	A1
ADD	mm,ii	81,X0	J1
ADD	rr,i	83,X0	A1
ADD	mm,i	83,X0	J1
ADD	AL,i	04	A1
ADD	AX,ii	05	A1
AND	r,r	20,X8	D1
AND	m,r	20,X8	E1
AND	rr,rr	21,X9	D1
AND	mm,rr	21,X9	F1
AND	r,m	22,8X	G1
AND	rr,mm	23,9X	H1
AND	r,i	80,X4	A1
AND	m,i	80,X4	I1
AND	rr,ii	81,X4	A1
AND	mm,ii	81,X4	J1
AND	AL,i	24	A1
AND	AX,ii	25	A1
CALL	dd	E8	K1
CALL	rr	FF,X2	L1
CALL	mm	FF,X2	M1
CALL	aaaa	9A	N1
CALL	dw	FF,X3	O1
CBW	none	98	P1
CLC	none	F8	P1
CLD	none	FC	P1
CLI	none	FA	P1
CMC	none	F5	P1
CMP	r,r	38,X8	D1
CMP	m,r	38,X8	G1
CMP	rr,rr	39,X9	D1
CMP	mm,rr	39,X9	H1
CMP	r,m	3A,8X	G1
CMP	rr,mm	3B,9X	H1
CMP	r,i	80,X7	A1
CMP	m,i	80,X7	G1
CMP	rr,ii	81,X7	A1
CMP	mm,ii	81,X7	R1
CMP	rr,i	83,X7	A1
CMP	mm,i	83,X7	R1
CMP	AL,i	3C	A1
CMP	AX,ii	3D	A1
CMPS	byte	A6	S1
CMPS	word	A7	T1
CWD	none	99	U1
DAA	none	27	A1
DAS	none	2F	A1
DEC	m	FE,X1	V1
DEC	mm	FF,X1	W1
DEC	AL	FE,C8	D1
DEC	CL	FE,C9	D1
DEC	DL	FE,CA	D1
DEC	BL	FE,CB	D1
DEC	AH	FE,CC	D1
DEC	CH	FE,CD	D1
DEC	DH	FE,CE	D1
DEC	BH	FE,CF	D1
DEC	AX	48	D1
DEC	CX	49	D1
DEC	DX	4A	D1
DEC	BX	4B	D1
DEC	SP	4C	D1
DEC	BP	4D	D1
DEC	SI	4E	D1
DEC	DI	4F	D1
DIV	r	F6,X6	X1
DIV	m	F6,X6	Y1
DIV	rr	F7,X6	Z1
DIV	mm	F7,X6	D4
ESC	0,rr	D8,XN	P1
ESC	1,rr	D9,XN	P1
ESC	2,rr	DA,XN	P1
ESC	3,rr	DB,XN	P1
ESC	4,rr	DC,XN	P1
ESC	5,rr	DD,XN	P1
ESC	6,rr	DE,XN	P1
ESC	7,rr	DF,XN	P1
ESC	0,mm	D8,XN	B2
ESC	1,mm	D9,XN	B2
ESC	2,mm	DA,XN	B2
ESC	3,mm	DB,XN	B2
ESC	4,mm	DC,XN	B2
ESC	5,mm	DD,XN	B2
ESC	6,mm	DE,XN	B2
ESC	7,mm	DF,XN	B2
HLT	none	F4	P1
IDIV	r	F6,X7	C2
IDIV	m	F6,X7	D2
IDIV	rr	F7,X7	E2
IDIV	mm	F7,X7	A4
IMUL	r	F6,X5	G2
IMUL	m	F6,X5	H2
IMUL	rr	F7,X5	I2
IMUL	mm	F7,X5	B4
IN	AL,i	E4	K2
IN	AX,i	E5	L2
IN	AL,DX	EC	M2
IN	AX,DX	ED	N2
INC	m	FE,X0	V1
INC	mm	FF,X0	W1
INC	AL	FE,C0	D1
INC	CL	FE,C1	D1
INC	DL	FE,C2	D1
INC	BL	FE,C3	D1
INC	AH	FE,C4	D1
INC	CH	FE,C5	D1
INC	DH	FE,C6	D1
INC	BH	FE,C7	D1
INC	AX	40	D1
INC	CX	41	D1
INC	DX	42	D1
INC	BX	43	D1
INC	SP	44	D1
INC	BP	45	D1
INC	SI	46	D1
INC	DI	47	D1
INT	3	CC	O2
INT	i	CD	P2
INTO	none	CE	Q2
IRET	none	CF	R2
JA	d	77	S2
JAE	d	73	S2
JB	d	72	S2
JBE	d	76	S2
JC	d	72	S2
JCXZ	d	E3	T2
JE	d	74	S2
JG	d	7F	S2
JGE	d	7D	S2
JL	d	7C	S2
JLE	d	7E	S2
JMP	d	EB	U2
JMP	dd	E9	U2
JMP	rr	FF,X4	V2
JMP	mm	FF,X4	W2
JMP	aaaa	EA	U2
JMP	dw	FF,X5	X2
JNA	d	76	S2
JNAE	d	72	S2
JNB	d	73	S2
JNBE	d	77	S2
JNC	d	73	S2
JNE	d	75	S2
JNG	d	7E	S2
JNGE	d	7C	S2
JNL	d	7D	S2
JNLE	d	7F	S2
JNO	d	71	S2
JNP	d	7B	S2
JNS	d	79	S2
JNZ	d	75	S2
JO	d	70	S2
JP	d	7A	S2
JPE	d	7A	S2
JPO	d	7B	S2
JS	d	78	S2
JZ	d	74	S2
LAHF	none	9F	A1
LDS	rr,dw	C5,9X	Y2
LEA	rr,m	8D,9X	Z2
LES	rr,dw	C4,9X	Y2
LOCK	prfx	F0	P1
LODS	byte	AC	B3
LODS	word	AD	C3
LOOP	d	E2	D3
LOOPE	d	E1	T2
LOOPZ	d	E1	T2
LOOPNZ	d	E0	E3
LOOPNE	d	E0	E3
MOV	r,r	88,X8	P1
MOV	m,r	88,X8	G1
MOV	rr,rr	89,X9	P1
MOV	mm,rr	89,X9	H1
MOV	r,m	8A,8X	F3
MOV	rr,mm	8B,9X	G3
MOV	m,i	C6,X0	Q1
MOV	mm,ii	C7,X0	R1
MOV	AL,i	B0	A1
MOV	CL,i	B1	A1
MOV	DL,i	B2	A1
MOV	BL,i	B3	A1
MOV	AH,i	B4	A1
MOV	CH,i	B5	A1
MOV	DH,i	B6	A1
MOV	BH,i	B7	A1
MOV	AX,ii	B8	A1
MOV	CX,ii	B9	A1
MOV	DX,ii	BA	A1
MOV	BX,ii	BB	A1
MOV	SP,ii	BC	A1
MOV	BP,ii	BD	A1
MOV	SI,ii	BE	A1
MOV	DI,ii	BF	A1
MOV	AL,aa	A0	K2
MOV	AX,aa	A1	L2
MOV	aa,AL	A2	K2
MOV	aa,AX	A3	L2
MOV	sr,rr	8E,MX	P1
MOV	sr,mm	8E,MX	G3
MOV	rr,sr	8C,XL	P1
MOV	mm,sr	8C,XL	H1
MOVS	byte	A4	H3
MOVS	word	A5	I3
M0VSB	none	A4	H3
M0VSW	none	A5	I3
MUL	r	F6,X4	J3
MUL	m	F6,X4	K3
MUL	rr	F7,X4	L3
MUL	mm	F7,X4	C4
NEG	r	F6,X3	D1
NEG	m	F6,X3	E1
NEG	rr	F7,X3	D1
NEG	mm	F7,X3	F1
NOP	none	90	D1
NOT	r	F6,X2	D1
NOT	m	F6,X2	E1
NOT	rr	F7,X2	D1
NOT	mm	F7,X2	F1
OR	r,r	08,X8	D1
OR	m,r	08,X8	E1
OR	rr,rr	09,X9	D1
OR	mm,rr	09,X9	F1
OR	r,m	0A,8X	G1
OR	rr,mm	0B,9X	H1
OR	r,i	80,X1	A1
OR	m,i	80,X1	I1
OR	rr,ii	81,X1	A1
OR	mm,ii	81,X1	J1
OR	AL,i	0C	A1
OR	AX,ii	0D	A1
OUT	i,AL	E6	K2
OUT	i,AX	E7	L2
OUT	DX,AL	EE	M2
OUT	DX,AX	EF	N2
POP	mm	8F,X0	N3
POP	AX	58	O3
POP	CX	59	O3
POP	DX	5A	O3
POP	BX	5B	O3
POP	SP	5C	O3
POP	BP	5D	O3
POP	SI	5E	O3
POP	DI	5F	O3
POP	ES	07	O3
POP	CS	0F	O3
POP	SS	17	O3
POP	DS	1F	O3
POPF	none	9D	O3
PUSH	mm	FF,X6	P3
PUSH	AX	50	Q3
PUSH	CX	51	Q3
PUSH	DX	52	Q3
PUSH	BX	53	Q3
PUSH	SP	54	Q3
PUSH	BP	55	Q3
PUSH	SI	56	Q3
PUSH	DI	57	Q3
PUSH	ES	06	R3
PUSH	CS	0E	R3
PUSH	SS	16	R3
PUSH	DS	1E	R3
PUSHF	none	9C	R3
RCL	r,1	D0,X2	P1
RCL	m,1	D0,X2	V1
RCL	rr,1	D1,X2	P1
RCL	mm,1	D1,X2	W1
RCL	r,CL	D2,X2	S3
RCR	m,CL	D2,X2	T3
RCL	rr,CL	D3,X2	S3
RCL	mm,CL	D3,X2	E4
RCR	r,1	D0,X3	P1
RCR	m,1	D0,X3	V1
RCR	rr,1	D1,X3	P1
RCR	mm,1	D1,X3	W1
RCR	r,CL	D2,X3	S3
RCR	m,CL	D2,X3	T3
RCR	rr,CL	D3,X3	S3
RCR	mm,CL	D3,X3	E4
REP	prfx	F3	P1
REPE	prfx	F3	P1
REPZ	prfx	F3	P1
REPNE	prfx	F2	P1
REPNZ	prfx	F2	P1
RET	ws	C3	V3
RET	ii ws	C2	W3
RET	as	CB	X3
RET	ii as	CA	Y3
ROL	r,1	D0,X0	P1
ROL	m,1	D0,X0	V1
ROL	rr,1	D1,X0	P1
ROL	mm,1	D1,X0	W1
ROL	r,CL	D2,X0	S3
ROL	m,CL	D2,X0	T3
ROL	rr,CL	D3,X0	S3
ROL	mm,CL	D3,X0	E4
ROR	r,1	D0,X1	P1
ROR	m,1	D0,X1	V1
ROR	rr,1	D1,X1	P1
ROR	mm,1	D1,X1	W1
ROR	r,CL	D2,X1	S3
ROR	m,CL	D2,X1	T3
ROR	rr,CL	D3,X1	S3
ROR	mm,CL	D3,X1	E4
SAHF	none	9E	A1
SAL	r,1	D0,X4	P1
SAL	m,1	D0,X4	V1
SAL	rr,1	D1,X4	P1
SAL	mm,1	D1,X4	W1
SAL	r,CL	D2,X4	S3
SAL	m,CL	D2,X4	T3
SAL	rr,CL	D3,X4	S3
SAL	mm,CL	D3,X4	E4
SAR	r,1	D0,X7	P1
SAR	m,1	D0,X7	V1
SAR	rr,1	D1,X7	P1
SAR	mm,1	D1,X7	W1
SAR	r,CL	D2,X7	S3
SAR	m,CL	D2,X7	T3
SAR	rr,CL	D3,X7	S3
SAR	mm,CL	D3,X7	E4
SBB	r,r	18,X8	D1
SBB	m,r	18,X8	E1
SBB	rr,rr	19,X9	D1
SBB	mm,rr	19,X9	F1
SBB	r,m	1A,8X	G1
SBB	rr,mm	1B,9X	H1
SBB	r,i	80,X3	A1
SBB	m,i	80,X3	I1
SBB	rr,ii	81,X3	A1
SBB	mm,ii	81,X3	J1
SBB	AL,i	1C	A1
SBB	AX,ii	1D	A1
SCAS	byte	AE	U2
SCAS	word	AF	Z3
SEG			
CS:	prfx	2E	P1
DS:	prfx	3E	P1
ES:	prfx	26	P1
SS:	prfx	36	P1
SHL	r,1	D0,X4	P1
SHL	m,1	D0,X4	V1
SHL	rr,1	D1,X4	P1
SHL	mm,1	D1,X4	W1
SHL	r,CL	D2,X4	S3
SHL	m,CL	D2,X4	T3
SHL	rr,CL	D3,X4	S3
SHL	mm,CL	D3,X4	E4
SHR	r,1	D0,X5	P1
SHR	m,1	D0,X5	V1
SHR	rr,1	D1,X5	P1
SHR	mm,1	D1,X5	W1
SHR	r,CL	D2,X5	S3
SHR	m,CL	D2,X5	T3
SHR	rr,CL	D3,X5	S3
SHR	mm,CL	D3,X5	E4
STC	none	F9	P1
STD	none	FD	P1
STI	none	FB	P1
STOS	byte	AA	V2
STOS	word	AB	A2
SUB	r,r	28,X8	D1
SUB	m,r	28,X8	E1
SUB	rr,rr	29,X9	D1
SUB	mm,rr	29,X9	F1
SUB	r,m	2A,8X	G1
SUB	rr,mm	2B,9X	H1
SUB	r,i	80,X5	A1
SUB	m,i	80,X5	I1
SUB	rr,ii	81,X5	A1
SUB	mm,ii	81,X5	J1
SUB	rr,i	83,X5	A1
SUB	mm,i	83,X5	J1
SUB	AL,i	2C	A1
SUB	AX,ii	2D	A1
TEST	r,r	84,8X	D1
TEST	r,m	84,8X	G1
TEST	rr,rr	85,9X	D1
TEST	rr,mm	85,9X	H1
TEST	r,i	F6,X0	U1
TEST	m,i	F6,X0	F2
TEST	rr,ii	F7,X0	U1
TEST	mm,ii	F7,X0	J2
TEST	AL,i	A8	A1
TEST	AX,ii	A9	A1
WAIT	none	9B	M3
XCHG	r,r	86,8X	A1
XCHG	r,m	86,8X	I1
XCHG	rr,rr	87,9X	A1
XCHG	rr,mm	87,9X	J1
XCHG	AX,CX	91	D1
XCHG	AX,DX	92	D1
XCHG	AX,BX	93	D1
XCHG	AX,SP	94	D1
XCHG	AX,BP	95	D1
XCHG	AX,SI	96	D1
XCHG	AX,DI	97	D1
XLAT	byte	D7	V2
XOR	r,r	30,X8	D1
XOR	m,r	30,X8	E1
XOR	rr,rr	31,X9	D1
XOR	mm,rr	31,X9	F1
XOR	r,m	32,8X	G1
XOR	rr,mm	33,9X	H1
XOR	r,i	80,X6	A1
XOR	m,i	80,X6	I1
XOR	rr,ii	81,X6	A1
XOR	mm,ii	81,X6	J1
XOR	AL,i	34	A1
XOR	AX,ii	35	A1

Cycle Codes

A1	4	A2	11(15)	A3	unused
B1	60		15	B3	12
C1	83	B2	8(12)+	C3	12(16)
D1	3		12+		16
E1	16+	C2	101-112	D3	5:17
F1	16(24)+	D2	107-118+	E3	5:19
	24+	E2	165-184	F3	8+
G1	9+	F2	11+	G3	8(12)+
H1	9(13)+	G2	80-98		12+
	13+	H2	86-104+	H3	18
I1	17+	I2	128-154	I3	18(26)
J1	17(25)+	J2	11(15)+		26
	25+		15+	J3	70-77
K1	19	K2	10	K3	76-83+
	23	L2	10(14)	L3	118-133
L1	16		14	M3	4+5N
	20	M2	8	N3	17(21)+
M1	21(25)+	N2	8(12)		25+
	29+		12	O3	8
N1	28	O2	52		12
	36		72	P3	16(20)+
O1	37(45)+	P2	51		24+
	53+		71	Q3	11
P1	2	Q2	4:53		15
Q1	10+		4:73	R3	10
R1	10(14)+	R2	32		14
	14+		44	S3	8+4N
S1	22	S2	4:16	T3	20+4N+
T1	22(30)	T2	6:18	U3	N/A
	30	U2	15	V3	16
U1	5	V2	11		20
V1	15+	W2	18(22)+	W3	20
W1	15(23)+		22+		24
	23+	X2	24(32)+	X3	26
X1	80-90		32+		34
Y1	86-96+	Y2	16(24)+	Y3	25
Z1	144-162		24+		33
		Z2	2+	Z3	15(19)
A4	171-190(175-194)+				19
B4	134-160(138-164)+				
C4	124-139(128-143)+				
D4	128-143+				
E4	154-172(158-176)+				
F4	158-176+				
G4	20(28)+4N+				
H4	28+4N+				
I4	9+17N				
J4	9+17(25)N				
K4	9+25N				
L4	9+10N				
M4	9+10(14)N				
N4	9+14N				
P4	9+22N				
	9+22(30)N				
	9+30N				
	9+15N				
	9+15(19)N				
	9+19N				
	9+13N				
	9+13(17)N				
	9+17N				

Example

After reading "About the Tables", usage of the tables can be verified by assembly and disassembly of:

1)	1406	ADC	AL,6
2)	09D1	OR	CX,DX
3)	C6440708	MOV	(SI+7),8
4)	D1C6	ROL	SI
5)	D104	LP ROL	(SI)
6)	72FC	JC	LP
7)	D50A	AAD	

The following notes help avoid difficulty (when converting to hex) and correspond to the lines above:

1) Use "AL,i" - not "r,i"
2) Use "rr,rr" - not "r,r". Read about Second Byte Table to convert X9.
3) Parentheses indicate mem pntr and form is "m,i". Form of first operand is "(SI+d),".
4) Use "SI" from reg part of section X.
5) Use "(SI)" from mem part of section X.
6) Read about "Relative Jumps".
7) Special case for disassembly.

Hex and Decimal Conversion

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	2
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	3
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	4
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	5
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	6
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	7
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	8
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	9
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	A
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	B
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	C
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	D
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	E
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Memory Locations

00000 - 00003	Type 0 interrupt pointer for divide-error
00004 - 00007	Type 1 interrupt pointer for single-step
00008 - 0000B	Type 2 interrupt pointer for Non-Mask-Int
0000C - 0000F	Type 3 interrupt pointer for 1-byte- inst
00010 - 00013	Type 4 interrupt pointer for INTO inst
00014 - 0007F	Type 5 thru 31 interrupt pointers reserved for Intel products
00080 - 003FF	Type 32 thru 255 available interrupt pointers (or general memory use)
00400 - FFFEF	Main memory space
FFFF0 - FFFFB	CPU jumps to code here upon Reset
FFFFC - FFFFF	Reserved for Intel products

ASCII

LSD \ MSD	0	1	2	3	4	5	6	7
	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	!	n	~
F	1111	SI	US	/	?	O	←	o	DEL

Unused

Second Byte Table

T		X	0	1	2	3	4	5	6	7	N
7	S	(BX+SI)	00	08	10	18	20	28	30	38	
8	U	(BX+DI)	01	09	11	19	21	29	31	39	
8	M	(BP+SI)	02	0A	12	1A	22	2A	32	3A	
7		(BP+DI)	03	0B	13	1B	23	2B	33	3B	
5	P	(SI)	04	0C	14	1C	24	2C	34	3C	
5	N	(DI)	05	0D	15	1D	25	2D	35	3D	
6	T	(dd)	06	0E	16	1E	26	2E	36	3E	
5	S	(BX)	07	0F	17	1F	27	2F	37	3F	
11		(BX+SI+d)	40	48	50	58	60	68	70	78	
12	T	(BX+DI+d)	41	49	51	59	61	69	71	79	
12	O	(BP+SI+d)	42	4A	52	5A	62	6A	72	7A	
11		(BP+DI+d)	43	4B	53	5B	63	6B	73	7B	
9	D	(SI+d)	44	4C	54	5C	64	6C	74	7C	
9	A	(DI+d)	45	4D	55	5D	65	6D	75	7D	
9	T	(BP+d)	46	4E	56	5E	66	6E	76	7E	
9	A	(BX+d)	47	4F	57	5F	67	6F	77	7F	
11		(BX+SI+dd)	80	88	90	98	A0	A8	B0	B8	
12	I	(BX+DI+dd)	81	89	91	99	A1	A9	B1	B9	
12	N	(BP+SI+dd)	82	8A	92	9A	A2	AA	B2	BA	
11		(BP+DI+dd)	83	8B	93	9B	A3	AB	B3	BB	
9	M	(SI+dd)	84	8C	94	9C	A4	AC	B4	BC	
9	E	(DI+dd)	85	8D	95	9D	A5	AD	B5	BD	
9	M	(BP+dd)	86	8E	96	9E	A6	AE	B6	BE	
9		(BX+dd)	87	8F	97	9F	A7	AF	B7	BF	
0	R	AX or AL	C0	C8	D0	D8	E0	E8	F0	F8	
0	E	CX or CL	C1	C9	D1	D9	E1	E9	F1	F9	
0	G	DX or DL	C2	CA	D2	DA	E2	EA	F2	FA	
0		BX or BL	C3	CB	D3	DB	E3	EB	F3	FB	
0	D	SP or AH	C4	CC	D4	DC	E4	EC	F4	FC	
0	A	BP or CH	C5	CD	D5	DD	E5	ED	F5	FD	
0	T	SI or DH	C6	CE	D6	DE	E6	EE	F6	FE	
0	A	DI or BH	C7	CF	D7	DF	E7	EF	F7	FF	

AL	CL	DL	BL	AH	CH	DH	BH	8
AX	CX	DX	BX	SP	BP	SI	DI	9
ES	CS	SS	DS	—	—	—	—	L
ES	—	SS	DS	—	—	—	—	M

INC xx	DEC xx	CALL (2)	CALL (3)	JMP (2)	JMP (3)	PUSH xx			K
INC x	DEC x								J
TEST xx,ii		NOT xx	NEG xx	MUL xx	IMUL xx	DIV xx	IDIV xx		I
TEST x,i		NOT x	NEG x	MUL x	IMUL x	DIV x	IDIV x		H
ROL xx,CL	ROR xx,CL	RCL xx,CL	RCR xx,CL	SHL/SAL xx,CL	SHR xx,CL		SAR xx,CL		G
ROL x,CL	ROR x,CL	RCL x,CL	RCR x,CL	SHL/SAL x,CL	SHR x,CL		SAR x,CL		F
ROL xx	ROR xx	RCL xx	RCR xx	SHL/SAL xx	SHR xx		SAR xx		E
ROL x	ROR x	RCL x	RCR x	SHL/SAL x	SHR x		SAR x		D
ADD xx,ii	OR xx,i	ADC xx,i	SBB xx,i	AND xx,i	SUB xx,i	XOR xx,i	CMP xx,i		C
ADD x,i	OR x,i	ADC x,i	SBB x,i	AND x,i	SUB x,i	XOR x,i	CMP x,i		B
ADD xx,ii	OR xx,ii	ADC xx,ii	SBB xx,ii	AND xx,i	SUB xx,i	XOR xx,i	CMP xx,i		A

(2) = 'near' transfer indirect via word in reg or mem.
(3) = 'far' transfer indirect via double word in mem.

Pinouts

8086

GND	1	40	VCC
AD14	2	39	AD15
AD13	3	38	A16/S3
AD12	4	37	A17/S4
AD11	5	36	A18/S5
AD10	6	35	A19/S6
AD9	7	34	BHE*/S7
AD8	8	33	MN/MX*
AD7	9	32	RD*
AD6	10	31	HOLD (RQ*/GT0*)
AD5	11	30	HLDA (RQ*/GT1*)
AD4	12	29	WR* (LOCK*)
AD3	13	28	M/IO* (S2*)
AD2	14	27	DT/R* (S1*)
AD1	15	26	DEN* (S0*)
AD0	16	25	ALE (QS0)
NMI	17	24	INTA* (QS1)
INTR	18	23	TEST*
CLK	19	22	READY
GND	20	21	RESET

On 8088 AD8 to AD15 are A8 to A15; pin 28 is IO/M* (S2*); pin 34 is SS0 (HIGH). Max mode is in parenthesis. '*' means active low.

DEALER, EDUCATION & OEM INQUIRIES WELCOME

OR FITNESS FOR PURPOSE.

MICRO LOGIC CORP.
HACKENSACK, NJ

8086 & 8088

MICROPROCESSOR INSTANT REFERENCE CARD

MICRO CHART®

Hex to Instruction Conversion

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	ADD X8 byte	ADD X9 word	ADD 8X byte	ADD 9X word	ADD AL,i	ADD AX,ii	PUSH ES	POP ES	OR X8 byte	OR X9 word	OR 8X byte	OR 9X word	OR AL,i	OR AX,ii	PUSH CS	POP CS	0
1	ADC X8 byte	ADC X9 word	ADC 8X byte	ADC 9X word	ADC AL,i	ADC AX,ii	PUSH SS	POP SS	SBB X8 byte	SBB X9 word	SBB 8X byte	SBB 9X word	SBB AL,i	SBB AX,ii	PUSH DS	POP DS	1
2	AND X8 byte	AND X9 word	AND 8X byte	AND 9X word	AND AL,i	AND AX,ii	SEG =ES	DAA	SUB X8 byte	SUB X9 word	SUB 8X byte	SUB 9X word	SUB AL,i	SUB AX,ii	SEG =CS	DAS	2
3	XOR X8 byte	XOR X9 word	XOR 8X byte	XOR 9X word	XOR AL,i	XOR AX,ii	SEG =SS	AAA	CMP X8 byte	CMP X9 word	CMP 8X byte	CMP 9X word	CMP AL,i	CMP AX,ii	SEG =DS	AAS	3
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI	4
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI	5
6																	6
7	JO	JNO	JC/JB JNAE	JNC/JNB JAE	JE JZ	JNE JNZ	JBE JNA	JNBE JA	JS	JNS	JP JPE	JNP JPO	JL JNGE	JNL JGE	JLE JNG	JNLE JG	7
8	AX	BX	AX	CX	TEST 8X byte	TEST 9X word	XCHG 8X byte	XCHG 9X word	MOV X8 byte	MOV X9 word	MOV 8X byte	MOV 9X word	MOV XL	LEA 9X rr,m	MOV MX	POP OX	8
9	NOP	XCHG AX,CX	XCHG AX,DX	XCHG AX,BX	XCHG AX,SP	XCHG AX,BP	XCHG AX,SI	XCHG AX,DI	CBW	CWD	CALL aaaa	WAIT	PUSHF	POPF	SAHF	LAHF	9
A	MOV AL,aa	MOV AX,aa	MOV aa,AL	MOV aa,AX	MOVS byte	MOVS word	CMPS byte	CMPS word	TEST AL,i	TEST AX,ii	STOS byte	STOS word	LODS byte	LODS word	SCAS byte	SCAS word	A
B	MOV AL,i	MOV CL,i	MOV DL,i	MOV BL,i	MOV AH,i	MOV CH,i	MOV DH,i	MOV BH,i	MOV AX,ii	MOV CX,ii	MOV DX,ii	MOV BX,ii	MOV SP,ii	MOV BP,ii	MOV SI,ii	MOV DI,ii	B
C			RET ii near	RET	LES 9X rr,dw	LDS 9X rr,dw	MOV 0X x,i	MOV 0X xx,ii			RET ii far	RET far	INT 3	INT i	INTO	IRET	C
D	DX	EX	FX	GX	AAM (D4,0A)	AAD (D5,0A)		XLAT	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7	D
E	LOOPNZ LOOPNE	LOOPZ LOOPE	LOOP	JCXZ	IN AL,i	IN AX,i	OUT i,AL	OUT i,AX	CALL dd	JMP dd	JMP aaaa	JMP d	IN AL,DX	IN AX,DX	OUT DX,AL	OUT DX,AX	E
F	LOCK		REPNE REPNZ	REP REPE REPZ	HLT	CMC	HX	IX	CLC	STC	CLI	STI	CLD	STD	JX	KX	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	

Miscellaneous Notes

COMPATIBILITY: The 8086 and 8088 are 100% compatible in machine and assembly languages.

SEGMENTS: Memory segments are 64K byte sections of the full megabyte space. Four segments are assigned to code, stack, data, and extra data. Their physical location is given by their respective registers (CS, SS, DS, ES) times 16. Locations within a segment are specified by a 16-bit offset (or logical) address relative to the beginning of the segment.

ALIGNMENT: On both processors, words can start at even or odd addresses. However, on the 8086, each load or store of an odd aligned word adds 4 cycles to execution time. 8086 programs should at least align the stack.

DESTINATION (,) SOURCE: Instructions that take data from some "source" and put a result at some "destination" are written in the form: MNEMONIC DESTINATION,SOURCE

BYTE ORDER: Two byte and two word values, displacements, and addresses in code, stack, jump-table, and data areas are stored with Least significant half at Lower address.

RELATIVE JUMPS: The destination address of a relative jump is the sum of the signed displacement and the address of the first byte of the next instruction.

STRING POINTERS: For string operations, while SI points into the DATA segment, note that DI points into the EXTRA segment.

BP FOR STACK: When register BP is specified in an instruction, the variable is assumed to reside in the STACK segment.

RESERVED PORTS: Ports 00F8H thru 00FFH of the 64K I/O locations are reserved for Intel products.

INTERRUPT NOTES: When a segment register and another value must be updated together without the possibility of an intervening interrupt (e.g. SS and SP), the segment register should be changed first and followed immediately by the instruction that updates the other value. (Interrupts are not recognized immediately after a move to segment register, POP segment register, or prefix instruction.) Interrupts are accepted and handled properly during repeated string operations provided additional prefixes are not used (and assuming there are no algorithmic conflicts with string data). The NMI and INTR interrupt lines are respectively edge and level triggered.

RESET: A hardware Reset sets CS=FFFF, DS=SS=ES=0000, FLAGS=0, and starts executing code at location FFFF0.

ROTATES AND SHIFTS: All single-bit rotates and shifts set OF=1 if the MSB (sign bit) is changed by the operation. If the sign bit retains its original value, OF is cleared. OF is undefined after multi-bit operations.

PARITY FLAG: The parity flag reflects the parity of only the low order 8 bits of results. (Flag is set if even number of one-bits, cleared if odd.)

BCD TERMS: Packed BCD and Unpacked BCD have respectively two and one binary coded decimal digits per byte. Unpacked BCD + 30H yields ASCII.

LOGICAL INSTRUCTIONS: AND, OR, TEST, and XOR instructions clear the OF and CF flags.

SEGMENT OVERRIDE EXCEPTIONS: A segment override prefix can be attached to instructions (placed just before the opcode byte) to cause data to be accessed at any of the three alternatives to the default segment except for: stack operations; string destinations; and instruction fetches.

DERIVATION: This card is based on Intel publications.

About the Tables

FLAG CODES TABLE: In the FLAG CODES table, 'U' indicates that the flag becomes undefined. Otherwise the listed flag is affected according to the operation.

INSTRUCTION DESCRIPTION TABLE: The single letter column corresponds to the leftmost column of the FLAG CODES table.

HEX COLUMN OF INSTRUCTION SET: Non-HEX values for the second byte refer to sections of the SECOND BYTE TABLE (see below). Following the listed opcode byte(s) go an immediate displacement or address if applicable and finally immediate data if applicable.

'C' COLUMN OF INSTRUCTION SET: These codes refer to the CYCLE CODES table.

CYCLE CODES TABLE: Listed numbers are instruction execution times in CPU cycles. When 8086 and 8088 times differ, the 8086 time is given first and the 8088 is given on the next line. A '+' terminator indicates to add calculation time for the effective address per section 'T' of the SECOND BYTE TABLE. For the 8086, the number in parenthesis applies when word data is at an odd address. 8086 times assume the stack at an even address. A '–' indicates a min to max range. X:Y are times for FAILURE:SUCCESS. Note that several factors can increase execution time over the figures shown. A series of fast executing instructions can drain the instruction queue and increase execution time; and instruction prefetch can conflict with memory data access also increasing execution time. The actual time for a code sequence is claimed to typically be within 5-10% of the theoretical time although in special cases it can be much more. For the 8086, instruction alignment can affect speed in some cases but usually not substantially.

SECOND BYTE TABLE: This table allows conversion to and from hex of the second byte (excluding prefixes) of an instruction. The table is referred to by other parts of this card in such forms as X9, 9X, X0, MX, KX, etc. X9, for example, directs you to find the first operand of the instruction being converted in section X, and the second operand in section 9. (Section 9 is located below number matrix.) The machine code is then found at the intersection. X0 sends you to X for the ONLY operand and across to section 0 for the machine code. For disassembly, first find the machine code in the number matrix and determine the instruction from the associated points in the indicated sections. When assembling, make sure register values are taken from the bottom part of section X while register pointers are taken from the upper parts.

HEX TO INSTRUCTION TABLE: To convert from hex to an instruction, scan down for the first digit (MSD) and across for the second. Two-character codes (upper case) in the table refer to sections of the SECOND BYTE TABLE but only when they appear on the first of the two lines of an entry. On the second line, two-character codes refer to registers.

ADDRESSING COLUMN OF INSTRUCTION SET:

r	=	byte register
rr	=	word register
i	=	immediate byte value
ii	=	immediate word value
d	=	immediate signed byte displacement
dd	=	immediate signed word displacement
aa	=	immediate two byte address (offset from segment start) (address can be of byte or word)
aaaa	=	immediate four byte address (2 byte offset followed by 2 byte segment address/16)
m	=	memory byte specified by memory pointers of section X of SECOND BYTE TABLE
mm	=	memory word specified by memory pointers of section X of SECOND BYTE TABLE. (With CALL or JMP instructions memory has 2 byte offset from segment start of point to go to.)
x	=	reg or mem byte
xx	=	reg or mem word
sr	=	segment register
dw	=	memory double-word specified by memory pointers of section X of SECOND BYTE TABLE. (With CALL or JMP instructions memory has 2 byte offset from segment start followed by 2 byte segment address/16 of point to go to.)
ws	=	within segment
as	=	another segment
()	=	data in mem

Where 'byte' or 'word' is listed, the assembler may require a dummy reference to labels.

Instruction Description

Flag Codes

A	A C OU PU SU ZU
B	AU CU OU P S Z
C	A C O P S Z
D	AU C O P S Z
E	EVERY FLAG
F	NO OTHERS
G	A O P S Z
H	C O
I	I T
J	A C P S Z
K	AU CU OU PU SU ZU
L	AU C O PU SU ZU
M	A C OU P S Z
N	NONE

Flags

A = Aux carry flag
C = Carry flag
D = Direction flag
I = Interrupt enable
O = Overflow flag
P = Parity flag
S = Sign flag
T = Trap flag
Z = Zero flag

Registers

AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

SP	STACK POINTER
BP	BASE POINTER
SI	SOURCE INDEX
DI	DESTINATION INDEX

IP	INSTRUCTION PNTR
F	----ODITSZ-A-P-C

CS	CODE SEGMENT
DS	DATA SEGMENT
SS	STACK SEGMENT
ES	EXTRA SEGMENT

Intentionally Blank

AAA	A	ASCII adjust for addition— Restores AL to unpacked BCD after addition of unpacked BCD number (byte add only)
AAD	B	ASCII adjust for division — Modifies AL *before* dividing unpacked BCD numbers so that result will be unpacked BCD (div by byte only) (AL=AH*OAH+AL;AH=0) AAD followed by DIV only produces unpacked BCD quotients between 0-9. Quotients above 9 will be in normal binary. For unpacked BCD quotients between 00-99 use: AAD; DIV; MOV REMAH; AAM
AAM	B	ASCII adjust for multiplication — Restores AH and AL to unpacked BCD after multiplication of unpacked BCD numbers (byte mul only)
AAS	A	ASCII adjust for subtraction — Restores AL to unpacked BCD after subtraction of unpacked BCD number (byte sub only)
ADC	C	Add with carry
ADD	C	Add
AND	D	Logical AND (clears CF, OF)
CALL	N	Call procedure (pushes return addr)
CBW	N	Convert byte to word — Extends sign bit of AL throughout AH
CLC	F	Clear carry flag
CLD	F	Clear direction flag — Prepares for auto increment of SI and DI during string-op
CLI	F	Clear interrupt-enable flag — disables interrupts (except NMI and software interrupts)
CMC	F	Complement carry flag
CMP	C	Compare destination with source — JG, for example, will jump if destination is greater than source
CMPS	C	Compare string — Compares bytes or words pointed to by DI and SI and updates DI and SI by 1 or 2 accordingly - JG, for example, will jump if destination is greater than source
CWD	N	Convert word to double-word — Extends sign bit of AX throughout DX
DAA	M	Decimal adjust for addition — Restores AL to packed BCD after addition of packed BCD number (byte add only)
DAS	M	Decimal adjust for subtraction — Restores AL to packed BCD after subtraction of packed BCD number (byte sub only)
DEC	G	Decrement by one
DIV	K	Divide unsigned — (AL=AX/source; AH=rem) or (AX = DX:AX / source; DX = rem) Type 0 interrupt if div by 0 or quotient too large
ESC	N	Escape — for instructions to coprocessor
HALT	N	Halt and wait for interrupt
IDIV	K	Divide signed — see DIV
IMUL	L	Multiply signed — (AX = AL * source) or (DX:AX = AX * source) CF and OF are set when upper half of result has significant bits.
IN	N	Input from port
INC	G	Increment by one
INT	I	Interrupt — Activates 1 of 256 interrupt routines by software (clears IF, TF) (As with hardware int, flags are saved on stack)
INTO	N	Interrupt if overflow — If OF=1 then INT 4 (clears IF, TF if successful)
IRET	E	Interrupt return - Returns from interrupt procedure whether activated by hardware or software (flags are restored from stack)
JA	N	Jump if above - unsigned
JAE	N	Jump if above or equal - unsigned
JB	N	Jump if below - unsigned
JBE	N	Jump if below or equal - unsigned
JC	N	Jump if carry - If CF=1
JCXZ	N	Jump if CX register zero
JE	N	Jump if equal - If ZF=1
JG	N	Jump if greater - signed
JGE	N	Jump if greater or equal - signed
JL	N	Jump if less - signed
JLE	N	Jump if less or equal - signed
JMP	N	Jump unconditionally
JNA	N	Jump if not above - unsigned
JNAE	N	Jump if not above nor equal - unsigned
JNB	N	Jump if not below - unsigned
JNBE	N	Jump if not below nor equal - unsigned
JNC	N	Jump if no carry - If CF=0
JNE	N	Jump if not equal - If ZF=0
JNG	N	Jump if not greater - signed
JNGE	N	Jump if not greater nor equal - signed
JNL	N	Jump if not less - signed
JNLE	N	Jump if not less nor equal - signed
JNO	N	Jump if no overflow - If OF=0
JNP	N	Jump if not parity - If PF=0
JNS	N	Jump if not sign - If SF=0
JNZ	N	Jump if not zero - If ZF=0
JO	N	Jump if overflow - If OF=1
JP	N	Jump if parity - If PF=1
JPE	N	Jump if parity even - If PF=1
JPO	N	Jump if parity odd - If PF=0
JS	N	Jump if sign - If SF=1
JZ	N	Jump if zero - If ZF=1
LAHF	N	Load AH from low byte of flags
LDS	N	Load pointer using DS - A double word pointer located in memory is moved into a register (first word) and register DS (second word)
LEA	N	Load effective address - The address (offset from beginning of segment) of the source operand (as opposed to its value) is loaded into a register
LES	N	Load pointer using ES - Similar to LDS
LOCK	N	Lock bus - A prefix causing CPU to assert LOCK signal during execution of prefixed instruction
LODS	N	Load string - Loads byte or word pointed to by SI into AL or AX and updates SI by 1 or 2 accordingly
LOOP	N	Loop - Decrement CX by one and jump if CX not zero
LOOPE	N	Loop while equal - Decrement CX and jump if CX not zero and ZF=1
LOOPNE	N	Loop while not equal - Same as LOOPNZ
LOOPNZ	N	Loop while not zero - Decrement CX and jump if CX not zero and ZF=0
LOOPZ	N	Loop while zero - Same as LOOPE
MOV	N	Move - Moves to destination from source
MOVS	N	Move string - Moves byte or word pointed to by SI to location pointed to by DI and updates pointers by 1 or 2 accordingly
MOVSB	N	Move string byte - See MOVS
MOVSW	N	Move string word - See MOVS
MUL	L	Multiply unsigned - See IMUL
NEG	C	Negate - two's complement (multiply by -1)
NOP	N	No operation
NOT	N	Logical NOT - one's complement
OR	D	Logical OR (clears CF, OF)
OUT	N	Output to port
POP	N	Pop word from stack - SP increases after access
POPF	E	Pop flags from stack
PUSH	N	Push word onto stack - SP decreases first
PUSHF	N	Push flags onto stack
RCL	H	Rotate thru carry left - by 1 or by CL
RCR	H	Rotate thru carry right - by 1 or by CL
REP	N	Repeat prefix - See below
RET	N	Return from procedure - Not for use with interrupt procedures - Optional pop-value (usually even #) is added to SP to dump passed parameters
ROL	H	Rotate left - by 1 or by CL - CF = LSB of result
ROR	H	Rotate right - by 1 or by CL - CF = MSB of result
SAHF	J	Store AH into low byte of flags
SAL	D	Shift arithmetic left - zero fill - by 1 or by CL — CF = last bit shifted out
SAR	D	Shift arithmetic right - sign extension - by 1 or by CL - CF = last bit shifted out - note that negative numbers are rounded differently from IDIV by 2
SBB	C	Subtract with borrow - destination minus source
SCAS	C	Scan string - Compares AL or AX with byte or word pointed to by DI and updates DI by 1 or 2 accordingly - JG, for example, will jump if AL or AX is greater than string element
SHL	D	Shift logical left - Same as SAL
SHR	D	Shift logical right - zero fill - by 1 or by CL - CF = last bit shifted out
STC	F	Set carry flag
STD	F	Set direction flag - Prepares for auto decrement of SI and DI during string-op
STI	F	Set interrupt-enable flag - enables interrupts after next instruction
STOS	N	Store string - Stores AL or AX into location pointed to by DI and updates DI by 1 or 2 accordingly
SUB	C	Subtract - destination minus source
TEST	D	Test by logical AND - affects only flags (also clears CF, OF)
WAIT	N	CPU enters WAIT state
XCHG	N	Exchange - switches contents of destination and source
XLAT	N	Translate - Replaces AL with a byte from a table pointed to by BX. AL initially gives the position in the table. The first element is at position 0.
XOR	D	Logical Exclusive-OR - Differing bits yield one - Like bits yield zero (clears CF, OF)

REPEAT INSTRUCTIONS

REP LODS	Repeats LODS CX times (1) - N4 cycles for bytes, P4 for words
REP MOVS	Repeats MOVS CX times (1) - F4 cycles for bytes, G4 for words
REP STOS	Repeats STOS CX times (1) - H4 cycles for bytes, I4 for words
REPE CMPS	Repeats CMPS until strings mismatch but not more than CX times (1) - J4 cycles for bytes, K4 for words
REPE SCAS	Repeats SCAS until AL or AX mismatches string but not more than CX times (1) - L4 cycles for bytes, M4 for words
REPNE CMPS	Repeats CMPS until strings match but not more than CX times (1) - J4 cycles for bytes, K4 for words
REPNE SCAS	Repeats SCAS until AL or AX matches string but not more than CX times (1) - L4 cycles for bytes, M4 for words
REPNZ CMPS	Same as REPNE CMPS
REPNZ SCAS	Same as REPNE SCAS
REPZ CMPS	Same as REPE CMPS
REPZ SCAS	Same as REPE SCAS

(1) CX is decremented each time. DI (and SI) end up pointing one position past end of string(s) or past first point of match/mismatch. ZF does not have to be setup before using these instructions.