נספח

פורמט פקודות מכונה 8086/8

ב-8086/8 גודל יצוג של פקודה היה מבית אחד עד ששה בתים.

פקודות מכונה בגודל בית

בדרך כלל פקודות על אופרנד אוגר 16 ביט אחד או ללא אופרנדים.

יצוג פקודות ללא אופרנדים הם פשוט מספר פקודה. הפקודות הללו הן:

XLAT, LAHF, SAHF, PUSHF, POPF, AAA, DAA, CLC, CMC, STC, CLD, STD, CLI, STI, HLT, IRET, WAIT, RET (ללא אופרנד), INTO, INT3

חלק מהפקודות הפועלות על אופרנד אוגר 16 ביט כללי כמו PUSH, POP , INC ממומשות בפורמט

7 6 5 4 3	2 1 0
OPCODE	REG

כאשר REG הוא קידוד של האוגרים הכלליים מוגדר ע"י

REG	
000	AX
001	CX
010	DX
011	BX
100	SP
101	BP
110	SI
111	DI

הפקודות PUSH sr ו-POP sr כאשר sr הוא אחד מאוגרי הסגמנטים הם מהצורה

7	6	5	4	3	2	1	0
0	0	0	S	R	1	1	1

כאשר SR הוא קידוד של אוגרי הסגמנט, שישמש אותנו גם בהמשך,

ES	0 0
CS	0 1
SS	1 0
DS	1 1

הפקודות IN AL,DX, IN AX,DX, OUT DX,AL, OUT DX,AX ממשמשות בפורמט:

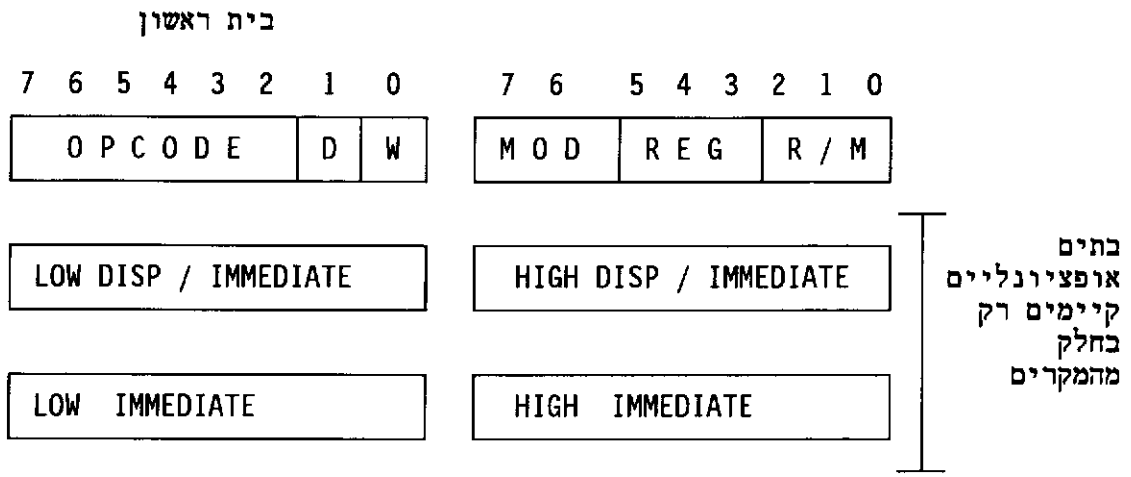
7	6	5	4	3	2	1	0
Opcode7							W

כאשר W == 0 כאשר מדובר בפעולת ביט (AL)

W == 1 כאשר מדובר בפעולת מילה (AX)

פקודות ארוכות יותר מבית אחד.

הפורמט הבא מתאר את מרבית פקודות המכונה של ה-8086/8:



המפתח לפענוח פקודות הארוכות יותר הם שני הבתים הראשונים:

6 הביטים הראשונים מאפינים את סוג הפקודה (ADD, MOV ...)

ביט D (פקודות מכונה 2 אופרנדים בלבד):
אם D==1 שדה ה-REG בבית השני מצין את אופרנד היעד
אם D==0 שדה ה-REG בבית השני מצין את אופרנד מקור

במקרה של פעולות אוגר כללי - אוגר כללי D==1

ביט W: W==0 מדובר הפעולת בית
W==1 מדובר הפעולת מילה

הערות:

- בפקודות אריתמטיות עם קבועים (למשל CMP AX,-6, ADD CX,121) ביט ה-D נקרא S והוא מצין אם להרחיב קבוע 8 ביט ל-16 ביט תוך התחשבות בסימן. S==0 אין התחשבות בסימן. S==1 הרחבה תוך התחשבות בסימן אם W==1.

- בפקודות הזזה הביטיות ביט ה-D נקרא V והוא משמש להבדיל בין גירסאות ההזזה הבודדת של הפקודות הללו לבין הגירסאות המצינות את ההזזה ב-CL, למשל להבחין בין הפקודה SHL AX,1 לבין SHL AX,CL. V==0 הזזה בודדת. V==1 הזזה לפי CL.

- בפקודות LOOPE, LOOPNE ובקידומות REPE, REPNE ביט ה-W נקרא Z והוא משמש לאבחנה של התנאי הנוסף מעבר ל- CX > 0 להסתעפות:

Z == 0 <==> LOOPNE או REPNE

Z == 1 <==> LOOPE או REPE

<u>בית 2 בפקודה</u>

MOD - 2 ביטים המבדילים בין מיעון זכרון ואוגרים ובמקרה של זכרון מצינים כמה בתים של הזזה (Displacement) באים לאחר בית זה.

REG (פקודות 2 אופרנדים בלבד) - 3 ביטים המצינים שדה אוגר.

R/M - 3 ביטים שיחד עם ה-MOD קובעים את שיטת המיעון

MOD ==

00	אין Displacement	זכרון
01	Displacement של בית אחד	
10	Displacement של שני בתים	
11	פעולת אוגר- אוגר	

REG - יחד עם ביט ה-W בוחר את האוגר:

REG	W==0	W==1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

בפקודות MOV שמעורבות בהם אוגרי הסגמנטים, השימוש ב-REG יהיה לפי הפורמט:

7 6	5	4 3	2 1 0
M O D	0	S R	R / M

כאשר SR הוא הקידוד של אוגרי הסגמנטים שתואר קודם.

הוראות שבהם יש אופרנד אחד (למשל INC AL, INC BL, NOT AX, INC WORD PTR [BX]) או אופרנד מתפרש אחד (MUL, DIV) או פקודות 2 אופרנדים שאחד מהם קבוע (למעט כאשר AX או AL הם האופרנד השני) משתמשים בביטים בשדה ה-REG כהשלמה ל-opcode. כלומר יש לו משמעויות אחרות.

R/M - מצין את המיעון יחד עם ה-MOD כמתואר להלן:

				MOD 11	
R/M	MOD==00	MOD==01	MOD==10	W==0	W==1
000	BX+SI	BX+SI+DISP8	BX+SI+DISP16	AL	AX
001	BX+DI	BX+DI+DISP8	BX+DI+DISP16	CL	CX
010	BP+SI	BP+SI+DISP8	BP+SI+DISP16	DL	DX
011	BP+DI	BP+DI+DISP8	BP+DI+DISP16	BL	BX
100	SI	SI+DISP8	SI+DISP16	AH	SP
101	DI	DI+DISP8	DI+DISP16	CH	BP
110	Direct Address	BP+DISP8	BP+DISP16	DH	SI
111	BX	BP+DISP8	BP+DISP16	BH	DI

Direct Address - הכתובת נלקחת ישר משני הבתים אחרי בית שיטת המיון.

התיחסות טהורה ל-BP (למשל MOV [BP],AX) ממומשת כהתיחסות דרך [BP+0] כלומר displacement 0 בגודל בית. ממילא השימוש שהאוגר BP נועד לו מכתיב שכמעט ואין צורך בגרסה הזו.

בתים מקדימים Prefix bytes

לחלק מהפקודות (מכל הפורמטים) ישנם אופציות של בתים מקדימים prefix bytes משני סוגים: Instruction Prefix ו- Segment Override. הדבר יכול להגדיל יצוג של פקודת מכונה בעד 2 בתים. לפיכך הגודל המירבי של פקודת מכונה ב-8086 הוא 8 בתים.

האפשריות של Instruction Prefix הן

```
F3h = 1111 0011b   REP prefix (פקודות מחרוזת בלבד)
F3h = 1111 0011b   REPE / REPZ prefix
                              (פקודות מחרוזת בלבד )
F2h = 1111 0010b   REPNE / REPNZ prefix
                              (פקודות מחרוזת בלבד )
F0h = 1111 0010b   LOCK prefix
```

האפשריות של Segment Override הן

```
26h - 0010 0110b - ES segment override prefix
2Eh - 0010 1110b - CS segment override prefix
36h - 0011 0110b - SS segment override prefix
3Eh - 0011 1110b - DS segment override prefix
```

מיעון בשיטת Segment Override

כאשר בפקודת היתיחסות לזכרון יש Segment Override (למעט DS) מופיע לפני הפקודה בית מיוחד המצין ל-CPU לבחור אוגר סגמנט אחר מברירת המחדל (DS למעט כאשר BP מופיע אז נבחר SS). לאחר מכן תופיע הפקודה כמו קודם.

הערכים של ה-Segment Override נבדלים ב-2 הביטים האמצעיים:

```
                    SR
                   ----
ES:  26h = 0 0 1 0  0 1 1 0        (0 0)
CS:  2Eh = 0 0 1 0  1 1 1 0        (0 1)
SS:  36h = 0 0 1 1  0 1 1 0        (1 0)
DS:  3Eh = 0 0 1 1  1 1 1 0        (1 1)
                   ----
```

פקודות נוספת שאינן לפי הפורמט הנ"ל

פקודות 2 אופרנדים בין קבוע כאפרנד מקור והאוגר AX או AL כאופרנד יעד (למשל MOV AL,9 ADD AX,7 אבל גם פקודות קלט/פלט עם קבוע למשל OUT 60h,AL) הם מהפורמט

7 6 5 4 3 2 1	0	
Opcode7	W	Const

כאשר W משמש להבדיל בין פעולת בית למילה, כמו קודם.
Const יכול להיות בית או מילה, למעט פקודות קלט/פלט (בית בלבד).

פקודות ההסתעפות (עם אופרנד JMP, CALL, JE, INT, RET), למעט הסתעפויות עקיפות, הם לפי הפורמט:

Opcode8	Const

כאשר Const הוא קבוע בגודל בין בית אחד לארבעה.

בפקודות הסתעפות מותנות Const תמיד בית אחד.

פקודה בעלת מבנה מיוחד

ESC -	11011 XXX	MOD YYY R/M	Disp-Low	Disp-High

דוגמאות

הפקודה ADD AX,BX מקודדת בהקסה: 03 C3

בבינארי

Opcode	D	W	MOD	R E G	R/M
0 0 0 0 0 0	1	1	1 1	0 0	0 1 1

הפקודה MOV BX,AX מקודדת בהקסה: 8B D8

בבינארי

Opcode	D	W	MOD	R E G	R/M
1 0 0 0 1 0	1	1	1 1	0 1	1 0 0 0

הפקודה MOV AL,CH מקודדת בהקסה: 8A C5

בבינארי

Opcode	D	W	MOD	R E G	R/M
1 0 0 0 1 0	1	0	1 1	0 0 0 1 0 1	

הפקודה MOV AX,[BX] מקודדת בהקסה: 8B 07

בבינארי

Opcode	D	W	MOD	R E G	R/M
1 0 0 0 1 0	1	1	0 0	0 0 0	0 1 1 1

הפקודה MOV CX,SS:[BX+DI] מקודדת בהקסה: 36 8B 09

בבינארי

Prefix	Opcod	D	W	MOD	R E G	R/M
0 0 1 1 0 1 1 0	1 0 0 0 1 0	1	1	0 0	0 0 0	1 0 0 1

יצירת דוגמאות נוספות

כתוב תוכנית חוקית מבחינת השפה באסמבלי (היא לא חייבת לרוץ) והרץ עליה את tasm עם האופציה /la. תקבל קובץ lst. עם פירוט הקידוד בהקסה.

לדוגמא הפקודה:

```
tasm /la myprog1.asm
```

תיצור קובץ בשם myprog1.lst שאותו תוכל לקרוא בתוכנית editor. ליד הפקודות תראה את פירוט הקידוד הבינארי: היסטי הפקודות ותוכנם.

מספר שורה	היסט	קידוד בהקסה	הפקודה באסמבלי
32	0028	03 C3	ADD AX,BX
33	002A	2B C3	SUB AX,BX
34	002C	8B 90 0080	MOV DX,[BX+SI+80h]
35	0030	8B 90 4000	MOV DX,[BX+SI+4000h]
36	0034	8B 07	MOV AX,[BX]
37	0036	B8 0040	MOV AX,64
38	0039	B9 0080	MOV CX,128
39	003C	8B 46 06	MOV AX,[BP+6]
40	003F	2E: 8B 5D 06	MOV BX,CS:[DI+6]
41	0043	26: 8B 07	MOV AX,ES:[BX]
42	0046	36: 8B 07	MOV AX,SS:[BX]
43	0049	2E: 8B 07	MOV AX,CS:[BX]
44	004C	26: 8B 07	MOV AX,ES:[BX]
45	004F	42	INC DX
46	0050	FE 00	INC BYTE PTR [BX+SI]
47	0052	FF 00	INC WORD PTR [BX+SI]
48	0054	03 14	ADD DX,[SI]
49	0056	03 14	ADD DX,DS:[SI]

פורמט פקודות מכונה 386 ואילך

שפת המכונה של ה-386 הוא באופן כללי הרחבה של שפת המכונה של ה-8086/8. ה-386 מכיר את פקודות המכונה בפורמט הישן ובנוסף יש לו פקודות מכונב נוספות בפורמט דומה עבור האוגרים 32 ביט. הפורמט של רוב הפקודות (כולל בעצם פקודות 8086) הוא

Instruction prefix	Address size prefix	Opreand size prefix	Segment Override
1 או 0	1 או 0	1 או 0	1 או 0
מספר הבתים			

MODR/M byte			
MOD	REG\Opcode	R/M	Opcode
1 או 0			2 או 1
מספר בתים			

SIB (Scale Index Base) byte				
SS	Index	Base	Displacement	Immediate
1 או 0			4 או 0,1,2	4 או 0,1,2
מספר בתים				

לפיכך אפשר להסיק שהאורך המירבי של פקודת מכונה ב-386 ואילך היא 16 בתים.

ערכים אפשריים ל-Instruction Prefix

F3h = 1111 0011b REP prefix (פקודות מחרוזת בלבד)
F3h = 1111 0011b REPE / REPZ prefix
(פקודות מחרוזת בלבד)
F2h = 1111 0010b REPNE / REPNZ prefix
(פקודות מחרוזת בלבד)
F0h = 1111 0010b LOCK prefix

ערכים אפשריים ל- Size prefix

```
66h = 0110 0110b Operand size prefix
67h = 0110 0110b Address size prefix
```

ערכים אפשריים ל- Segment override

```
2Eh - 0010 1110b - CS segment override prefix
36h - 0011 0110b - SS segment override prefix
3Eh - 0011 1110b - DS segment override prefix
26h - 0010 0110b - ES segment override prefix
64h - 0110 0100b - FS segment override prefix
65h - 0110 0101b - GS segment override prefix
```

אם הפקודה מכילה את ה-Operand size prefix פירושו שהפעולה היא בגודל 32 ביט (אוגר EAX, EBX ... או זכרון 32 ביט).
אם הפקודה מכילה את ה-Operand address prefix פירושו שההיסט בחישוב הכתובת הוא 32 ביט (אוגר EAX+4*EBX וכו').

קידוד אוגרים

השדות REG, INDEX ו-BASE הינם קודים של אוגרים, וגם R/M הוא כזה בתנאים מסוימים. בפקודות שמתקבלים בירושה מה-8086 הם קודים של אוגרים 8 ו-16 ביט. במידה ובפקודה מופיעה ה- Operand size prefix או ה- Address size prefix, השדות הללו יקודדו את האוגרים 32 ביט. זה קצת משתנה לפי אם מדובר ביצוג יעד או אפשרות של היסט, אבל באופן עקרוני הקידוד של האוגרים 32 ביט הינו

אוגר	קוד
EAX	000
ECX	001
EDX	010
EBX	011
ESP	100
EBP	101
ESI	110
EDI	111

בית ה-SIB

הבית הזה מופיע כאשר בחישוב היסט 32 ביט יש scale (2,4,8 factor) ליד אוגר האינדקס ו/או כאשר בחישוב ההיסט הזה יש 2 אוגרים 32 ביט.

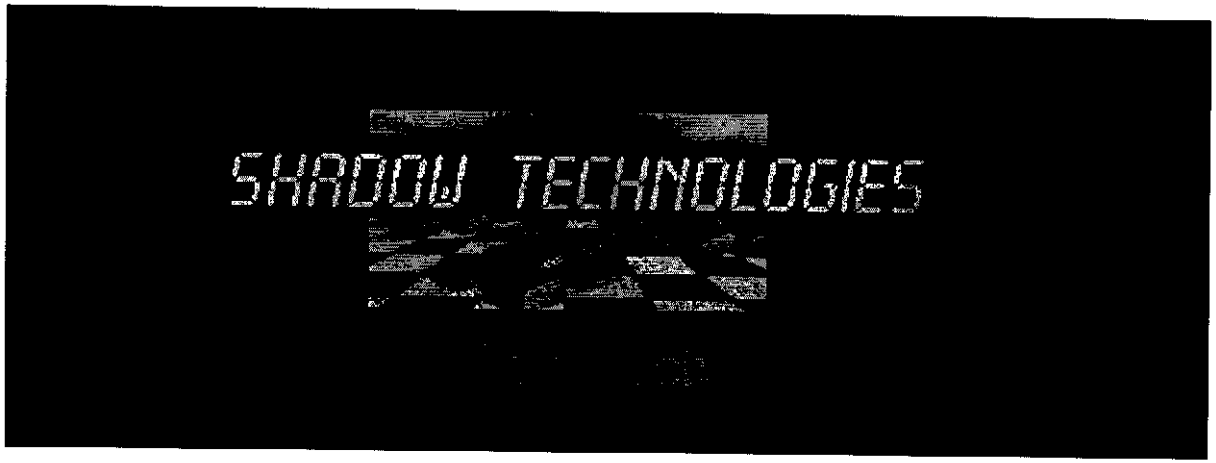
שדה ה-Base הוא קידוד של אוגר האינדקס לפי הטבלה שלעיל למעט 101 שיש לו שלושה פירושים שונים (שניים מהם כוללים את ה-EBP)

בתלות ב-MOD.

שדה ה-Index הוא קידוד של אוגר האינדקס לפי הטבלה שלעיל למעט 100 שאינו נחשב לקידוד של ESP (שאיננו יכול להיות אוגר אינדקס, רק בסיס).

שדה ה-SS הוא מגדיר למעשה ה-scale factor, הוא החזקה של 2 של המקדם של המכפלה של אוגר האינדקס

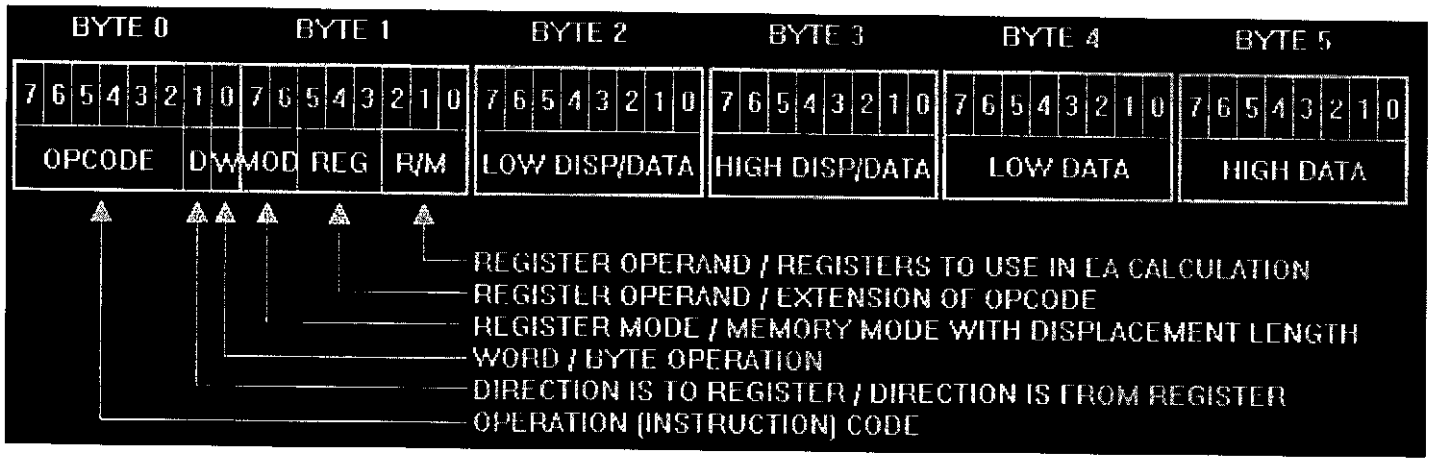
SS
00 - 1*,
01 - 2*,
10 - 4*,
11 - 8*.



8086/8088 Assembly Language Programming

Assembly/Machine Language Coding

Machine Language Encoding Derivatives



MOD=11			EFFECTIVE ADDRESS CALCULATION			
R/M	W=0	W=1	R/M	MOD=00	MOD=01	MOD=10
000	AL	AX	000	(BX) + (SI)	(BX) + (SI) + D8	(BX) + (SI) + D16
001	CL	CX	001	(BX) + (DI)	(BX) + (DI) + D8	(BX) + (DI) +D16
010	DL	DX	010	(BP) + (SI)	(BP) + (SI) + D8	(BP) + (SI) + D16
011	BL	BX	011	(BP) + (DI)	(BP) + (DI) + D8	(BP) + (DI) + D16
100	AH	SP	100	(SI)	(SI) + D8	(SI) + D16
101	CH	BP	101	(DI)	(DI) + D8	(DI) + D16
110	DH	SI	110	DIRECT ADDRESS	(BP) + D8	(BP) + D16
111	BH	DI	111	(BX)	(BX) + D8	(BX) + D16

REG	W=0	W=1
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

MOD	EXPLANATION
00	Memory Mode, no displacement follows*
01	Memory Mode, 8 bit displacement follows
10	Memory Mode, 16-bit displacement follows
11	Register Mode (no displacement)

*Except when R/M=110, then 16-bit displacement follows

Register	SR
ES	00
CS	01
SS	10
DS	11

Field	Value	Function
S	0	No sign extend
	1	Sign extended 8-bit immediate data to 16 bits if W=1
V	0	Shift / rotate count is one
	1	Shift / rotate count is specified in the CL register
Z	0	Repeat / loop while zero flag is clear
	1	Repeat / loop while zero flag is set

DATA TRANSFER

MOV = Move

Register/Memory to/from register	100010dw|mod reg r/m|Disp-Lo|Disp-Hi
Immediate to register/meory	1100011w|mod 000 r/m|Disp-Lo|Disp-Hi|Data|Data if w=1
Immediate to register	1011wreg|Data|Data if w=1
Memory to accumulator	1010000w|Addr-Lo|Addr-Hi
Accumulator to memory	1010001w|Addr-Lo|Addr-Hi
Register/Memory to segment register	10001110|mod 0 SR r/m|Disp-Lo|Disp-Hi
Segment Register to register/Memory	10001100|mod 0 SR r/m|Disp-Lo|Disp-Hi

PUSH = Push

Register/Memory	11111111|mod 110 r/m|Disp-Lo|Disp-Hi
Register	01010reg
Segment register	000SR110

POP = Pop

Register/Memory	11111111|mod 110 r/m|Disp-Lo|Disp-Hi
Register	01011reg
Segment register	000SR111

XCHG = Exchange

Register/memory with register	1000011w|mod reg r/m|Disp-Lo|Disp-Hi
Register with accumulator	10010reg

IN = Input from

Fixed Port	1110010w|Data-8
Variable Port	1110110w

Out = Ouput to

Fixed port	1110011w|Data-8
Variable port	1110111w

XLAT = Table lookup to al	11010111
LEA = Load EA to register	10001101|mod reg r/m|Disp-Lo|Disp-Hi
LDS = Load Pointer to DS	11000101|mod reg r/m|Disp-Lo|Disp-Hi
LES = Load Pointer to ES	11000100|mod reg r/m|Disp-Lo|Disp-Hi
LAHF = Load AH with Flags	10011111
SAHF = Store AH into Flags	10011110
PUSHF = Push flags	10011100
POPF = Pop flags	10011101

ARITHMETIC

Add = add

Reg/memory with register to either	000000dw|mod reg r/m|Disp-Lo|Disp-Hi
Immediate to register/memory	100000sw|mod 000 r/m|Disp-Lo|Disp-Hi|Data|Data if s:w=01
Immediate to accumulator	0001010w|Data|Data if w=1

ADC = Add with carry

Reg/Memory with register to either	000100dw|mod reg r/m|Disp-Lo|Disp-Hi
Immediate to register/memory	100000sw|mod 010 r/m|Disp-Lo|Disp-Hi|Data if s:w=01
Immediate to accumulator	0001010w|Data|Data if w=1

INC = Increment

Register/Memory	1111111w|mod 000 r/m|Disp-Lo|Disp-Hi
Register	01000reg|

AAA = Ascii adjust for add	00110111
DAA = Decimal adjust for add	00100111

SUB = Subtract

Reg/memory and register to either	001010dw|mod reg r/m|Disp-Lo|Disp-Hi
Immediate from register/memory	100000sw|mod 101 r/m|Disp-Lo|Disp-Hi|Data|Data if s:w=01
Immediate from accumulator	0010110w|Data|Data if w=1

SBB = Subtract with borrow

Reg/memory and register to either	000110dw|mod reg r/m|Disp-Lo|Disp-Hi
Immediate from register/memory	100000sw|mod 011 r/m|Disp-Lo|Disp-Hi
Immediate from accumulator	0001110w|Data|Data if w=1

DEC = Decrement

Register/memory	1111111w|mod 001 r/m|Disp-Lo|Disp-Hi
Register	01001reg

NEG = Change sign	1111011w|mod 0111 r/m|Disp-Lo|Disp-Hi

CMP = Compare

Register/memory and register	001110dw|mod reg r/m|Disp-Lo|Disp-Hi
Immediate with register/memory	100000sw|mod 111 r/m|Disp-Lo|Disp-Hi|Data|Data if s:w=1
Immediate with accumulator	0011110w|Data

AAS = Ascii adjust for subtract	00111111
DAS = Decimal adjust for subtract	00101111
MUL = Multiply(unsigned)	1111011w|mod 100 r/m|Disp-Lo|Disp-Hi
IMUL = Integer multiply(signed)	1111011w|mod 101 r/m|Disp-lo|Disp-Hi
AAM = Ascii adjust for multiply	11010100|00001010|Disp-Lo|Disp-Hi
DIV = Divide(unsigned)	1111011w|mod 110 r/m|Disp-Lo|Disp-Hi
IDIV = Integer Divide(Signed)	1111011w|mod 111 r/m|Disp-Lo|Disp-Hi
AAD = Ascii Adjust for divide	11010101|00001010|Disp-Lo|Disp-Hi
CBW = Convert byte to word	10011000
CWD = Convert word to doubleword	10011001

LOGIC

NOT = Invert	1111011w|mod 010 r/m|Disp-Lo|Disp-Hi
SHL/SAL = Shift logical/arithmetic left	110100vw|mod 100 r/m|Disp-Lo|Disp-Hi
SHR = Shift logical right	110100vw|mod 101 r/m|Disp-Lo|Disp-Hi
SAR = Shift arithmetic right	110100vw|mod 111 r/m|Disp-Lo|Disp-Hi

```
ROL      = Rotate left                  110100vw|mod 000 r/m|Disp-Lo|Disp-Hi
ROR      = Rotate right                 110100vw|mod 001 r/m|Disp-Lo|Disp-Hi
RCL      = Rotate through carryf left   110100vw|mod 010 r/m|Disp-Lo|Disp-Hi
RCR      = Rotate through carryf right  110100vw|mod 011 r/m|Disp-Lo|Disp-Hi

AND      = Boolean AND

Reg/memory with register to either      001000dw|mod reg r/m|Disp-Lo|Disp-Hi
Immediate to register/memory            1000000w|mod 100 r/m|Disp-Lo|Disp-Hi|Data|Data if w=1
Immediate to accumulator                0010010w|Data|Data if w=1

TEST     = And function to flags no result

Register/memory and register            000100dw|mod reg r/m|Disp-Lo|Disp-Hi
Immediate data and register memory      1111011w|mod 000 r/m|Disp-Lo|Disp-Hi|Data|Data if w=1
Immediate and accumulator               1010100w|Data

OR = Boolean OR

Reg/Memory and register to either       000010dw|mod reg r/m|Disp-Lo|Disp-Hi
Immediate to register memory            0011010w|mod 001 r/m|Disp-Lo|Disp-Hi
Immediate to accumulator                1010100w|Data|Data if w=1

XOR = Exclusive Boolean OR

Reg/memory and register to either       001100dw|mod reg r/m|Disp-Lo|Disp-Hi
Immediate to register/memory            0011010w|Data|Disp-Lo|Disp-Hi|Data|Data if w=1
Immediate to accumulator                0011010w|Data|Data if w=1
```

STRING MANIPULATION

```
REP  = Repeat                           1111001z
MOVS = Move byte/word                   1010010w
CMPS = Compare byte/word                1010011w
SCAS = Scan byte/word                   1010111w
LODS = Load byte/word to AL/AX          1010110w
STDS = Store byte/word from AL/AX       1010101w
```

CONTROL TRANSFER

```
CALL = Call
Direct within segment                    11101000|IP-INC-LO|IP-INC-HI
Indirect within segment                  11111111|mod 010 r/m|Disp-Lo|Disp-Hi
Direct intersegment                      10011010|IP-Lo|IP-Hi|CS-LO|CS-HI
Indirect intersegment                    11111111|mod 011 r/m|Disp-Lo|Disp-Hi

RET = Return from call

Within segment                           11000011
Within segment adding immediate to SP    11000010|Data-Lo|Data-Hi
Intersegment                             11001011
Intersegment adding immediate to SP      11001010|Data-Lo|Data-Hi

JE/JZ    = Jump equal/zero               01110100|IP-INC8
JL/JNGE  = Jump less/not greater-equ     01111100|IP-INC8
JLE/JNG  = Jump less or equ/not greater  01111110|IP-INC8
JB/JNAE  = Jump below/not above equal    01110010|IP-INC8
JBE/JNA  = Jump below equal/not above    01110110|IP-INC8
JP/JPE   = Jump parity/parity even       01111010|IP-INC8
JO       = Jump on overflow              01110000|IP-INC8
JS       = Jump on sign                  01111000|IP-INC8
JNE/JNZ  = Jump not equal/not zero       01110101|IP-INC8
JNL/JGE  = Jump not less/greater or equ  01111101|IP-INC8
JNLE/JG  = Jump not less equal/greater   01111111|IP-INC8
JNB/JAE  = Jump not below/above or equ   01110011|IP-INC8
JNBE/JA  = Jump not below or equ/above   01110111|IP-INC8
JNP/JPO  = Jump not parity/parity odd    01111011|IP-INC8
JNO      = Jump not overflow             01110001|IP-INC8
JNS      = Jump not sign                 01111001|IP-INC8
JCXZ     = Jump on cx zero               11100011|IP-INC8

JMP = Unconditional Jump

Direct within segment                    11101001|IP-INC-LO|IP-INC-HI
Direct within segment-short              11101011|IP-INC8
Indirect within segment                  11111111|mod 100 r/m|Disp-Lo|Disp-Hi
Direct intersegment                      11101010|IP-LO|IP-HI|CS-LO|CS-HI
Indirect intersegment                    11111111|mod 101 r/m|Disp-Lo|Disp-Hi

LOOP          = Loop CX times            11100010|IP-INC8
LOOPZ/LOOPE   = Loop while zero/equ      11100001|IP-INC8
LOOPNZ/LOOPNE = Loop not zero/equ        11100000|IP-INC8
```

INT	= Interupt	
Type specified		11001101|Data-8
INT3		11001100
INTO	= Interupt on overflow	11001110
IRET	= Interupt return	11001111

PROCESSOR CONTROL

CLC	= Clear carry	11111000
CMC	= Complement carry	11110101
STC	= Set carry	11111001
CLD	= Clear direction	11111100
STD	= Set direction	11111101
CLI	= Clear interupt	11111010
STI	= Set interupt	11111011
HLT	= Halt	11110100
WAIT	= wait	10011011
ESC	= Escape to external device	11011xxx|mod yyy r/m|Disp-Lo|Disp-Hi
LOCK	= Buss lock prefix	11110000
SEGMENT	= Segment overide prefix	001SR110

Table 26-2. 16-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) /digit (Opcode) REG =			AL AX EAX 0 000	CL CX ECX 1 001	DL DX EDX 2 010	BL BX EBX 3 011	AH SP ESP 4 100	CH BP EBP 5 101	DH SI ESI 6 110	BH DI EDI 7 111
Effective Address	Mod	R/M	ModR/M Values in Hexadecimal							
[BX + SI]	00	000	00	08	10	18	20	28	30	38
[BX + DI]		001	01	09	11	19	21	29	31	39
[BP + SI]		010	02	0A	12	1A	22	2A	32	3A
[BP + DI]		011	03	0B	13	1B	23	2B	33	3B
[SI]		100	04	0C	14	1C	24	2C	34	3C
[DI]		101	05	0D	15	1D	25	2D	35	3D
disp16		110	06	0E	16	1E	26	2E	36	3E
[BX]		111	07	0F	17	1F	27	2F	37	3F
[BX + SI] + disp8	01	000	40	48	50	58	60	68	70	78
[BX + DI] + disp8		001	41	49	51	59	61	69	71	79
[BP + SI] + disp8		010	42	4A	52	5A	62	6A	72	7A
[BP + DI] + disp8		011	43	4B	53	5B	63	6B	73	7B
[SI] + disp8		100	44	4C	54	5C	64	6C	74	7C
[DI] + disp8		101	45	4D	55	5D	65	6D	75	7D
[BP] + disp8		110	46	4E	56	5E	66	6E	76	7E
[BX] + disp8		111	47	4F	57	5F	67	6F	77	7F
[BX + SI] + disp16	10	000	80	88	90	98	A0	A8	B0	B8
[BX + DI] + disp16		001	81	89	91	99	A1	A9	B1	B9
[BP + SI] + disp16		010	82	8A	92	9A	A2	AA	B2	BA
[BP + DI] + disp16		011	83	8B	93	9B	A3	AB	B3	BB
[SI] + disp16		100	84	8C	94	9C	A4	AC	B4	BC
[DI] + disp16		101	85	8D	95	9D	A5	AD	B5	BD
[BP] + disp16		110	86	8E	96	9E	A6	AE	B6	BE
[BX] + disp16		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL		001	C1	C9	D1	D9	EQ	E9	F1	F9
EDX/DX/DL		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AH		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH		111	C7	CF	D7	DF	E7	EF	F7	FF

NOTES: **disp8** denotes an 8-bit displacement following the ModR/M byte, to be sign-extended and added to the index. **disp16** denotes a 16-bit displacement following the ModR/M byte, to be added to the index. Default segment register is SS for the effective addresses containing a BP index, DS for other effective addresses.

Table 26-3. 32-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) /digit (Opcode) REG =			AL AX EAX 0 000	CL CX ECX 1 001	DL DX EDX 2 010	BL BX EBX 3 011	AH SP ESP 4 100	CH BP EBP 5 101	DH SI ESI 6 110	BH DI EDI 7 111
Effective Address	Mod R/M		ModR/M Values in Hexadecimal							
[EAX]	00	000	00	08	10	18	20	28	30	38
[ECX]		001	01	09	11	19	21	29	31	39
[EDX]		010	02	0A	12	1A	22	2A	32	3A
[EBX]		011	03	0B	13	1B	23	2B	33	3B
[--][--][1]		100	04	0C	14	1C	24	2C	34	3C
disp32		101	05	0D	15	1D	25	2D	35	3D
[ESI]		110	06	0E	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F
disp8[EAX]	01	000	40	48	50	58	60	68	70	78
disp8[ECX]		001	41	49	51	59	61	69	71	79
disp8[EDX]		010	42	4A	52	5A	62	6A	72	7A
disp8[EBX];		011	43	4B	53	5B	63	6B	73	7B
disp8[--][--]		100	44	4C	54	5C	64	6C	74	7C
disp8[EBP]		101	45	4D	55	5D	65	6D	75	7D
disp8[ESI]		110	46	4E	56	5E	66	6E	76	7E
disp8[EDI]		111	47	4F	57	5F	67	6F	77	7F
disp32[EAX]	10	000	80	88	90	98	A0	A8	B0	B8
disp32[ECX]		001	81	89	91	99	A1	A9	B1	B9
disp32[EDX]		010	82	8A	92	9A	A2	AA	B2	BA
disp32[EBX]		011	83	8B	93	9B	A3	AB	B3	BB
disp32[--][--]		100	84	8C	94	9C	A4	AC	B4	BC
disp32[EBP]		101	85	8D	95	9D	A5	AD	B5	BD
disp32[ESI]		110	86	8E	96	9E	A6	AE	B6	BE
disp32[EDI]		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX/DX/DL		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AH		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH		111	C7	CF	D7	DF	E7	EF	F7	FF

NOTES: [1]**[--][--]** means a SIB follows the ModR/M byte.
[2]**disp8** denotes an 8-bit displacement following the SIB byte, to be sign-extended and added to the index. **disp32** denotes a 32-bit displacement following the SIB byte, to be added to the index.

/M Byte

CH	DH	BH
BP	SI	DI
EBP	ESI	EDI
5	6	7
101	110	111

decimal

28	30	38
29	31	39
2A	32	3A
2B	33	3B
2C	34	3C
2D	35	3D
2E	36	3E
2F	37	3F
68	70	78
69	71	79
6A	72	7A
6B	73	7B
6C	74	7C
6D	75	7D
6E	76	7E
6F	77	7F
A8	B0	B8
A9	B1	B9
AA	B2	BA
AB	B3	BB
AC	B4	BC
AD	B5	BD
AE	B6	BE
AF	B7	BF
E8	F0	F8
E9	F1	F9
EA	F2	FA
EB	F3	FB
EC	F4	FC
ED	F5	FD
EE	F6	FE
EF	F7	FF

gn-extended and added to
, to be added to the index.

Table 26-4. 32-Bit Addressing Forms with the SIB Byte

r32 Base = Base =			EAX 0 000	ECX 1 001	EDX 2 010	EBX 3 011	ESP 4 100	[*] 5 101	ESI 6 110	EDI 7 111
Scaled Index	SS Index		SIB Values in Hexadecimal							
[EAX]	00	000	00	01	02	03	04	05	06	07
[ECX]		001	08	09	0A	0B	0C	0D	0E	0F
[EDX]		010	10	11	12	13	14	15	16	17
[EBX]		011	18	19	1A	1B	1C	1D	1E	1F
none		100	20	21	22	23	24	25	26	27
[EBP]		101	28	29	2A	2B	2C	2D	2E	2F
[ESI]		110	30	31	32	33	34	35	36	37
[EDI]		111	38	39	3A	3B	3C	3D	3E	3F
[EAX*2]	01	000	40	41	42	43	44	45	46	47
[ECX*2]		001	48	49	4A	4B	4C	4D	4E	4F
[ECX*2]		010	50	51	52	53	54	55	56	57
[EBX*2]		011	58	59	5A	5B	5C	5D	5E	5F
none		100	60	61	62	63	64	65	66	67
[EBP*2]		101	68	69	6A	6B	6C	6D	6E	6F
[ESI*2]		110	70	71	72	73	74	75	76	77
[EDI*2]		111	78	79	7A	7B	7C	7D	7E	7F
[EAX*4]	10	000	80	81	82	83	84	85	86	87
[ECX*4]		001	88	89	8A	8B	8C	8D	8E	8F
[EDX*4]		010	90	91	92	93	94	95	96	97
[EBX*4]		011	98	89	9A	9B	9C	9D	9E	9F
none		100	A0	A1	A2	A3	A4	A5	A6	A7
[EBP*4]		101	A8	A9	AA	AB	AC	AD	AE	AF
[ESI*4]		110	B0	B1	B2	B3	B4	B5	B6	B7
[EDI*4]		111	B8	B9	BA	BB	BC	BD	BE	BF
[EAX*8]	11	000	C0	C1	C2	C3	C4	C5	C6	C7
[ECX*8]		001	C8	C9	CA	CB	CC	CD	CE	CF
[EDX*8]		010	D0	D1	D2	D3	D4	D5	D6	D7
[EBX*8]		011	D8	D9	DA	DB	DC	DD	DE	DF
none		100	E0	E1	E2	E3	E4	E5	E6	E7
[EBP*8]		101	E8	E9	EA	EB	EC	ED	EE	EF
[ESI*8]		110	F0	F1	F2	F3	F4	F5	F6	F7
[EDI*8]		111	F8	F9	FA	FB	FC	FD	FE	FF

NOTES: [*] means a disp32 with no base if MOD is 00, [EBP] otherwise. This provides the following addressing modes:

disp32[index] (MOD = 00)
disp8[EBP][index] (MOD = 01)
disp32[EBP][index] (MOD = 10)