הגדרת משתנים				
1 byte	8 bit	BYTE	DB	Char
2 byte	16 bit	WORD	DW	Int
4 byte	32 bit	DWORD	DD	Long int
6 byte	48 bit	FWORD	DF	
8 byte	64 bit	QWORD	DQ	
10 byte	80 bit	TBYTE	DT	

רשימת אוגרים		
AX - 15-[AH] [AL]-0	Accumulator Register	
BX - 15-[BH] [BL]-0	Base Register	
CX - 15-[CH] [CL]-0	Count Register	
DX - 15-[DH] [DL]-0	Data Register	
SI	Source Index Register	
DI	Destination Index Register	
SP	Stack Pointer Register	
BP	Base Pointer Register	
CS	Code Segment Register	
DS	Data Segment Register	
ES	Extra Segment Register	
SS	Stack Segment Register	
IP	Instruction Pointer	
FLAGS	FLAGS Registers	

אוגר הדגלים - FLAG REGISTER			
0	CF	Carry Flag	דגל נשא
1			
2	PF	Parity Flag	דגל זוגיות
3			
4	AF	Auxiliary Flag	דגל נשא עשרוני
5			
6	ZF	Zero result	תוצאה אפס
7	SF	Sign Flag	תוצאה שלילית
8	TF	Trap Flag	מעבד במצב Single Step
9	IF	Interrupt Enable Flag	אפשר פסיקות
10	DF	Direction Flag	כיוון פעולות מחרוזות
11	OF	Overflow Flag	גלישה אריתמטית
12			
13			
14			
15			

סידור הבתים בזיכרון – LITTLE ENDIAN
בזיכרון הבית הפחות משמעותי נמצא בכתובת הקטנה ביותר והיותר משמעותי בכתובת העוקבת 15[High Byte]8 - 7[Low Byte] 0 31[High Word]16 - 15[Low Word] 0

התייחסות לזיכרון
חישוב כתובת פיזית בזיכרון (ב REAL MODE 8086) : אוגר הסגמנט * 16 + OFFSET
פקודות מכונה רבים לא פועלים על אוגרי הסגמנט מול קבועים , לכתיבה או קריאה מאוגר סגמנט צריך להשתמש באחד מהאוגרים הכללים.
ל CS אי אפשר לכתוב ערכים , אך אפשר לקרוא את ערכו.
ההיסט ב 8086 הוא 16 ביט ומהצורה הבאה : BX SI OR + OR + CONST BP DI
ההיסט ב 386 הוא 32 ביט ומהצורה הבאה : [r32 + n * r32 + const] כאשר: r32 = EAX/EBX/ECX/EDX/ESP/EBP/ESI/EDI n = 1/2/4/8 const = קבוע עד 32 ביט בנוסף שני האוגרים 32 ביט צריכים להיות שונים.
כלל לבחירת אוגר סגמנט בצורה אוטומטית : • אם יש SEGMENT OVERRIDE אז המצוין בפקודה קובע • אחרת אם BP נמצא בין הסוגרים אזי נבחר SS • אחרת נבחר DS

	פקודות מכונה
PTR	המרה : - בכל פעם שאי אפשר לזהות את גודל האופרנד ע"י אחד האופרנדים בפקודה צריך לעשות CASTING לגודל מתאים ע"י PTR לקביעת גודל התייחסות ל 16 ביט החל מהכתובת המוצבת ב BX מבצעים: WORD PTR [BX]
EQU	הגדרת קבוע: Const1 EQU 80 • משנה בזמן אסמבלי את כל הערכים של Const1 ל 80.
HLT	מסיים את עבודת המעבד.
MOV	העברה : - MOV op1,op2 -> op1=op2; • פועלת על – 8 – 16 – 32 BIT (386) • עובד גם על אוגרי הסגמנט כאוגרי מקור ויעד חוץ מ CS שלא יכול לשמש כאוגר יעד , האוגר השני בפקודה עם אוגר סגמנט חייב להיות אוגר כללי 16 ביט או זיכרון. MOV [test] , 'a'
XCHG	החלפה:- MOV op1,op2 -> temp=op2; op2=op1; op1=temp; • פועלת על – 8 – 16 – 32 BIT (386) XCHG AX,BX
	פקודות מחסנית:-
PSUH	דחיפה למחסנית :- PUSH op16 • פועלת על כול אוגרי המעבד חוץ מ FLAGS,IP. • עובדת על זיכרון וקבוע(ב 8086 באופן עקיף). • בכל מקרה הערך הוא 16 ביט. PUSH var1 PUSH [si]
POP	שליפה מהמחסנית :- POP op16 • פועלת על כול אוגרי המעבד חוץ מ CS,FLAGS,IP. • עובדת על זיכרון. • בכל מקרה הערך הוא 16 ביט. POP ES
PUSHF	דחיפת אוגר הדגלים למחסנית:- • ללא אופרנדים. PUSHF
POPF	שליפת אוגר הדגלים מהמחסנית :- • שליפה של 16 ביט מהמחסנית והצבה לאוגר הדגלים.

	• ללא אופרנדים. POPF
PUSHDF	דחיפת אוגר הדגלים למחסנית (32 ביט)
POPDF	שליפת אוגר הדגלים מהמחסנית (32 ביט)
ADD	חיבור :- ADD op1,op2 -> op1=op1 + op2; • פועלת על – 8 – 16 – 32 BIT (386). ADD AX,var1
ADC	חיבור (עם נשא) :- ADC op1,op2 -> op1=op1 + op2 + carry; • פועלת על – 8 – 16 – 32 BIT (386). ; 32 Bit Sum using 16 Bit registers Var1 DD 71234 Var2 DD 45678 MOV AX,WORD PTR Var2 ADD WORD PTR Var1 , AX MOV AX , WORD PTR Var2+2 ADC WORD PTR Var1+2,AX
INC	קידום אופרנד ב 1:- INC op1 -> op1=op1 +1; • פועלת על – 8 – 16 – 32 BIT (386). • פקודת אופרנד אחד.
SUB	חיסור : - SUB op1,op2 -> op1=op1 - op2; • פועלת על – 8 – 16 – 32 BIT (386). SUB AX,var1
SBB	חיסור (עם נשא) :- SBB op1,op2 -> op1=op1 - op2 - borrow; • פועלת על – 8 – 16 – 32 BIT (386). ; 32 Bit SUB using 16 Bit registers Var1 DD 71234 Var2 DD 45678 MOV AX,WORD PTR Var2 SUB WORD PTR Var1 , AX MOV AX , WORD PTR Var2+2 SBB WORD PTR Var1+2,AX
DEC	חיסור 1 מהאופרנד :- INC op1 -> op1=op1 - 1;

	• פועלת על – 8 – 16 – 32 BIT (386).
NEG	היפוך סימן: NEG op1 -> op1=op1*(-1)
MUL	כפל מספרים חסרי סימן:- MUL op(8b) -> AX= AL*op(8b) MUL op(16b) -> DX:AX = AX * op(16b) [DX= MSB] MUL op(32b) -> EDX:EAX = EAX * op(32b) • פועלת על – 8 – 16 – 32 BIT (386). • לא עובד על קבועים , רק אוגר או זכרון. • פקודת אופרנד אחד. • צריך לדאוג לערכי האופרנדים המעורבים בפקודה לפני הביצוע(לאפס).
IMUL	כפל מתחשב בסימן: - • אותם התנאים כמו MUL אך עם התחשבות בסימן . • יש מקרה מיוחד ב 386 – פקודת שני אופרנדים של אוגרים 32 ביט ורק אוגרים. IMUL ESI,EDI -> ESI=ESI*EDI
DIV	חילוק בלי סימן:- DIV op(8b) -> AL= AX / op(8b) | AH= AX mod op(8b) DIV op(16b) -> AX = DX:AX/op(16b) | DX= DX:AX mod op(8b) DIV op(32b) -> EAX = EDX:EAX/op(32b) | EDX= EDX:EAX mod op(32b) • פועלת על – 8 – 16 – 32 BIT (386). • לא עובד על קבועים , רק אוגר או זכרון. • פקודת אופרנד אחד. • צריך לדאוג לערכי האופרנדים המעורבים בפקודה לפני הביצוע (לאפס).
IDIV	חילוק מתחשב בסימן :- • אותם התנאים כמו DIV אבל עם התחשבות בסימן. • כדי לאפס אופרנדים עליונים שאין להם צורך לפני ביצוע הפקודה אך הם מעורבים בה צריך לבצע התמרה המקום איפוס(במקרה של חסר סימן) , הדבר מתבצע ע"י פקודות ההתמרה המיוחדות.
CBW	התמרה מ BYTE ל WORD (AL ל AX) :
CWD	התמרה מ WORD ל DWORD (AX ל DX:AX)
CDQ	התמרה מ DWORD ל QWORD (EAX ל EDX:EAX)
CWDE	התמרה מ WORD ל DWORD (AX ל EAX)
CMP	השוואה : - • פועלת על – 8 – 16 – 32 BIT (386). • פועל על אוגר\קבוע\זכרון. • זהה ל SUB אך בלי שמירת הערך – רק שינוי באוגר הדגלים. CMP AX,BX
	פקודות ביטים :-
AND	פקודת ביטים (LOGICAL AND):

	• פועלת על – 8 – 16 – 32 (386) BIT. AND AX,BX ; AX[0] = 1 – if (AX[0]=1 AND BX[0]=1)
OR	פקודת ביטים (LOGICAL OR): • פועלת על – 8 – 16 – 32 (386) BIT. OR AX,BX ; AX[0] = 1 – if (AX[0]=1 OR BX[0]=1)
XOR	פקודת ביטים (LOGICAL XOR): • פועלת על – 8 – 16 – 32 (386) BIT. XOR AX,BX ; AX[0] = 1– if (AX[0]=1 OR BX[0]=1) but not both
TEST	בדיקה : - • מבצע AND אך בלי שמירת התוצאה , רק משנה באוגר הדגלים.
NOT	פקודת ביטים (LOGICAL NOT): • פועלת על – 8 – 16 – 32 (386) BIT. • היפוך ביטים. NOT AX ; AX[0] = 1– if (AX[0]=0)
	פקודות הזזה וסיבוב:
SHL / SHR	הזזה לוגית: SHL/R op1,op2 - shift op1 , op2 times. • OP1 יכול להיות 8\16\32 ביט. • OP2 או קבוע או CL. • SHL : בהזזה (נניח ב 1 שמאלה) הביט המשמעותי ביותר נכנס ל CF (דגל נשא) ומשם לאיבוד , לביט הנמוך ביותר נכנס 0. • SHR : בהזזה ימינה (נניח ב 1) המשמעותי ביותר מתאפס והנמוך ביותר נאבד.
SAL / SAR	הזזה אריתמטית : SAL/R op1,op2 - shift op1 , op2 times. • OP1 יכול להיות 8\16\32 ביט. • OP2 או קבוע או CL. • SAL : זהה ל SHL • SAR : זהה ל SHR אך משמר סימן , הביט המשמעותי ביותר הוא שנדחף לביטים העליונים בהזזה.
ROL / ROR	סיבוב : ROL/R op1,op2 - ROLL op1 , op2 times. • OP1 יכול להיות 8\16\32 ביט. • OP2 או קבוע או CL. • ROR : כמו SHR אך הביט המשמעותי פחות נכנס למשמעותי יותר ול CF. • ROL – כמו ROR כך בכיוון שמאלה
RCL / RCL	סיבוב (CF נחשב בתוך האופרנד)

	• בדומה ל ROR\L אך ערך ה CF נחשב כאילו בתוך האופרנד.
	פקודות מחרוזות : * בכל הפקודות האלה צריך לאתחל או DS:SI או ES:DI (או שניהם) לתחילתו או סופו של מערך. * הפקודות ידאגו לקדם או להפחית את המצבעים DI\SI ב 1. * קידום או הפחתה נקבעים ע"י הדגל DF והוא נקבע ע"י הפקודות הבאות. - CLD - DF=0 - קידום. - STD - DF=1 - הפחתה.
LODS (B/W/D)	טעינה ל AX מהמחרוזת:- LODSB -> MOV AL,DS:[SI] ; SI = (+/- 1) = (DF*-2 + 1) LODSW -> MOV AX,DS:[SI] ; SI = (+/- 1) =2*(DF*-2 + 1) LODSD -> MOV EAX,DS:[SI] ; SI = (+/- 1) =4*(DF*-2 + 1)
STOS (B/W/D)	טעינה מאוגר AX למחרוזת :- STOSB -> MOV ES:[DI],AL ; SI = (+/- 1) = (DF*-2 + 1) STOSW -> MOV ES:[DI],AX ; SI = (+/- 1) =2*(DF*-2 + 1) STOSD -> MOV ES:[DI],EAX ; SI = (+/- 1) =4*(DF*-2 + 1)
MOVS (B/W/D)	העתקה ממחרוזת למחרוזת : - העתקה מ DS:SI ל ES:DI [לא להתבלבל הפירוט למטה לדוגמה בלבד במציאות אין תמיכה להעברה מזיכרון לזיכרון] * MOVSB -> MOV ES:[DI],DS:[SI] ; SI&DI += (+/- 1) = (DF*-2 + 1) * MOVSW -> MOV ES:[DI],DS:[SI] ; SI&DI += (+/- 1) = (DF*-2 + 1) * MOVSD -> MOV ES:[DI],DS:[SI] ; SI&DI += (+/- 1) = (DF*-2 + 1)
REP	לולאה לפקודת מחרוזת: - REP STOS(B/W/D) | LODS (B/W/D) | MOVS (B/W/D) גורם לפקודת מחרוזת לחזור על עצמה עד ש CX מגיע ל 0 , בכל חזרה CX מופחת ב 1 .
SCAS (B/W/D)	השוואה בין אוגר AX ל מחרוזת :- • משווה בין EAX\AX\AL בהתאם לסיומת – לבין הזיכרון במחרוזת במקום ES:DI. • מתבצע קידום או הפחתה ל DI.
CMPS (B/W/D)	השוואה בין מחרוזת למחרוזת :- • שתי המחרוזות נמצאות ב ES:DI ו DS:SI. • מתבצע קידום או הפחתה ל SI/DI.

REPE/NE	לולאה לפקודת מחרוזת עם התניה : - REPE SCAS (B/W/D) | CMPS (B/W/D) • REPE – חוזר על הפקודה כל עוד יש שוויון. • REPNE – חוזר על הפקודה כל עוד יש אי שוויון.
CALL	קריאה לפונקציה :- שתי גרסאות לפקודה • NEAR CALL : (הסתעפות באותו סגמנט) דוחף את IP למחסנית אחר כך מבצע הסתעפות לכתובת שבאופרנד. • FAR CALL : (הסתעפות גם לסגמנטים אחרים) דוחף את CS למחסנית אחר כך דוחף את IP למחסנית אחרף כך מבצע הסתעפות לכתובת שבאופרנד.
RET	פקודת חזרה מפונקציה :- שתי גרסאות לפקודה • NEAR RET : (הסתעפות באותו סגמנט) שלוף מילה מהמחסנית ומעביר ל IP , (כלומר מסתעף למילה שנשלפה) • FAR RET : (הסתעפות גם לסגמנטים אחרים) שלוף שתי מלים מהמחסנית הראשונה מועברת ל IP והשנייה ל CS , כלומר מסתעף לכתובת המלאה שנשלפה.
LEA	חישוב כתובת דינמי : • מקבל אוגר או ליבם באופרנד המקור ומחזיר את ההיסט שלו באוגר היעד. • מאפשר לחשב כתובות בזמן ריצה.

C	ASSEMPLY
	תכנות מבני

If (AX==BX)
{
[CODE]
}
	CMP AX,BX
	JNE skip [תנאי הפוך]
	[CODE]
	skip:

If (AX == BX)
{
[CODE - 1]
}
Else
{
[CODE - 2]
}
	CMP AX,BX
	JNE else [תנאי הפוך]
	[CODE - 1]
	JMP endif
	else:
	[CODE - 2]
	endif:

If ((AX == BX) && (CX == DX))
{
[CODE]
 	
}
	CMP AX,BX
	JNE skipcode [תנאי הפוך]
	CMP CX,DX
	JNE skipcode [תנאי הפוך]
	[CODE]
	skipcode:

If ((AX == BX) || (CX == DX))
{
[CODE]
 	
}
	CMP AX,BX
	JE doCode [תנאי ישיר]
	CMP CX,DX
	JE doCode [תנאי ישיר]
	JMP skipCode
	doCode:
	[CODE]
	skipCode:

While (AX == BX)
{
[CODE]
}
	JMP test
	code:
	[CODE]
	test:
	CMP AX,BX
	JE code [תנאי ישיר]

do { **[CODE]** } while (AX == BX)	codeBody: **[CODE]** CMP AX,BX JE codeBody [תנאי ישיר]

For(DX=0; DX < N ; DX++)
{
[CODE]
 	
}
	MOV DX,0
	JMP nextTest
	codeBody:
	[CODE]
	INC DX

	nextTest: CMP DX,N JL codeBody

For (CX = N ; CX > 0 ; CX --)
{
[CODE]
 	
}
	MOV CX,N
	JCXZ skip
	codeStart:
	[CODE]
	LOOP codeStart
	skip:

פונקציות – הגדרה + התייחסות התאמה לקריאה מ TC

הגדרת פונקציה באסמבלר

```
==================================================================
[NAME]    PROC    [TYPE]

      [ CODE ]

[NAME]   ENDP
==================================================================
```

כאשר:

[NAME] = שם הפונקציה
[TYPE] =NEAR/FAR – סוג - CALL / RET קובע את התנהגות

קריאה לפונקציה מ TURBO C :

- דבר ראשון הפונקציה צריכה לשמור את ערכי האוגרים DI,SI, SS,DS,CS, SP,BP (כמובן אם בוצע בהם שימוש)
- כתיבת הערכים ושחרורם באחריות הפונקציה הקוראת.
- הפרמטרים נדחפים מימן לשמאל.
- החזרת ערכים מתבצעת דרך AX אם מדובר ב 16 ביט DX:AX אם מדובר ב 32 ביט.

```
==================================================================
[NAME]    PROC    [TYPE]
PUSH BP                          להקצאת מקום למשתנים לוקליים
MOV BP,SP

      [ CODE ]

POP BP
[NAME]   ENDP
==================================================================
```

מצב המחסנית אחרי קריאת TC לרוטינה : idiv_mod(int NUM,int DNUM,int *Q int *Rem)

ערך ישן של BP	SP-> BP
כתובת לחזרה IP	BP + 2
תוכן NUM	BP + 4
תוכן DNUM	BP + 6
כתובת Q	BP + 8
כתובת REM	BP + 10

הקצאת שטח למשתנים לוקליים מתבצע ע"י הזזת SP

```
SUB   SP,K
```

וההגעה אליהם מתבצעת ע"י BP שנשמר בתחילת הפונקציה.
בסוף הפונקציה צריך להחזיר את SP ל BP שנשמר – סוף הפונקציה יראה :

```
MOV  SP,BP
POP BP
RET
Myfunc ENDP
```

מראה המחסנית בריצה פונקציה עם משתנים לוקליים ופרמטרים מועברים :
מוגדר בגוף הפונקציה

```
Int NUM,Q,REM;
```

REM Q NUM	משתנים לוקליים	<----------SP
	OLD BP	<----------BP
	OLD IP	
	פרמטרים	

לגישה לפונקציות מקובץ לקובץ (ASM)
בקובץ הקורא צריך לרשום

```
EXTRN     [PROC_NAME]:[TYPE],[PROC2…..
```

בקובץ הנקר צריך לרשום

```
PUBLIC    [PROC_NAME]
```

אם קובץ C קורא לרוטינת אסמבלר

```
extern int proc ( int I, int *p) ..
```

מצבי הזיכרון של TC	
Tiny	מודל הזיכרון הקטן ביותר , כל אוגרי הסגמנט מצביעים לאותו מקום , בסך בכל כל התוכנה הינה K64 , כל ההסתעפויות מסוג NEAR.
Small	התוכנה פרוסה על גבי שני סגמנטים של K64 לכל אחד , סגמנט ל CODE וסגמנט ל DATA , CS מצביע לסגמנט ה CODE, DS/ES/SS כולם מצביעים לסגמנט ה DATA, ההסתעפויות תמיד NEAR .
Medium	הקוד יכול להיות ביותר מסגמנט אחד (עד 1 MB) לעומת המידע שכולו נמצא בסגמנט אחד K64 , ההסתעפויות הם FAR.
Compact	ההפך ל MEDIUM מצביעי המידע הם FAR , משתמש ביותר מסגמנט מידע (עד MB1) אך הקוד נמצא בסגמנט אחד של K64 .
Large	התוכנה יכולה להכיל יותר מסגמנט קוד ויותר מסגמנט מידע (עד 1 MB) , מצביעים והסתעפויות FAR.
Huge	כמו ה LARGE אך מבטל את ההגבלה של TC לכך שהמידע הסטטי יהיה לכל היותר K64 ומאפשר לשמור יותר מ K64 של STATIC DATA .

פסיקות - INTERRUPTS

Interrupt Vector

- ויקטור הפסיקות – 0-255 נמצא בראש הזיכרון של המחשב בגודל K1
- כתובות נשמרים באופן מלא עבור פסיקה מס K , ה OFFSET נמצא במקום 4*K,4*K+1 , ו SEGMENT נמצא במקום 2+K*4 , 3+K*4 .

מנגנון מימוש פסיקה :

1. ה CPU גומר את הפקודה הנוכחית.
2. מיד לאחר מכן דוחף למחסנית את אוגר הדגלים.
3. מאפס את ה IF.
4. מאפס את ה TP.
5. דוחף את כתובת הפקודה הבאה לביצוע , ה CS לאחר מכן ה IP.

פקודות הסתעפות

CLI	איפוס ה IF , 0=IF
STI	הדלקת ה IF , 1=IF
PUSHF POP AX AND AX,1111111011111111B PUSH AX POPF	איפוס ה TF , 0=TF
PUSHF POP AX OR AX,100000000B PUSH AX POPF	הדלקת ה TF , 1=TF

רשימת פסיקות חלקית

CODE	NAME	TYPE	Holds By
0	Divide Overflow	Exception	DOS
1	Single Step	Hardware	BIOS
5	Print Screen	Hardware	BIOS
8	Timer	Hardware	BIOS
9	Keyboard	Hardware	BIOS
16(10h)	Video	Software	BIOS
19(13h)	Floppy	Software	BIOS
22(16h)	Keyboard	Software	BIOS
28(18h)	Ctrl-Break	Hardware	BIOS
33(21h)	Function Request	Software	DOS

מימוש פונקצית טיפול בפסיקה

במימוש פונקציה לטיפול בפסיקה צריך לדאוג

- לשמור את **כל** האוגרים כולל הכללים .
- במקום שבו רוצים לשנות את כתובת הפסיקה בווקטור הפסיקות לכתובת החדשה צריך לדאוג לכיבוי מנגנון הפסיקות בזמן שינוי כתובת הפסיקה.

פקודת המקלדת 16h

תוכנית ניהול המקלדת, מקצה חוצץ של 20 לחיצות מקלדת .
התוכנית מבצעת את עבודתה לפי הפרמטר AH,

התנהגות התוכנה לפי ה AH היא :

AH	התנהגות
0h	• קוראת לחיצה מהמקלדת – ומבטלת אותה , בחזרה מהתוכנית ה AL יהיה קוד ה ASCII ו AH יהיה ה SCAN CODE. • אם אין מידע בחוצץ התוכנה ממתינה זמן בלתי מוגבל עד להגעת מידע (לחיצה).
01h	• התוכנה חוזרת מיד ואינה ממתינה למידע גם כאשר אין מידע בחוצץ. • התוכנה מדליקה את ZF אם לא היה מידע בחוצץ אחרת ZF אחרי החזרה כבוי. • התוכנה אינה מבטלת את הלחיצה , קריאה נוספת לתוכנה תחזיר אותה לחיצה.
02h	• מחזירה את סטאטוס המקלדת ב AL ,
03h	• קביעת זמן רענון.
05h	• הדמית לחיצת מקלדת . • המידע עבור ההדמיה נלקח מ CX , ASCCI = CL , SCAN=CH.

אסמבלר מותנה :

מבנה כללי

```
IFx
        [CODE]

ELSE
        [CODE – 2]

ENDIF
```

IF [ביטוי]	מבצע אסמבלי אם [הביטוי] <> 0
IFE [ביטוי]	מבצע אסמבלי אם [הביטוי] = 0
IFB [סמל]	מבצע אסמבלי אם [סמל] = blank
IFNB [סמל]	מבצע אסמבלי אם [סמל] <> blank
IFDEF [סמל]	מבצע אסמבלי אם ה[סמל] מוגדר
IFNDEF [סמל]	מבצע אסמבלי אם ה[סמל] לא מוגדר
IF1	לבצע במעבר אסמבלי ראשון
IF2	לבצע במעבר אסמבלי שני
IFIDN<arg1,arg2>	מבצע אסמבלי אם arg1 זהה ל arg2
IFDIF<arg1,arg2>	מבצע אסמבלי אם arg1 שונה ל arg2
ELSE	לבצע במקרה שהתנאי נכשל
ENDIF	סגור את חלק התנאי

מאקרו

```
[NAME]  MACRO  para1,para2,…
LOCAL [LABLE]
.
[LABLE] :

ENDM
```