אסמבלר – סיכום

1. ארכיטקטורות מעבד

1.1. משפחות מעבדים

1.1.1. 8086 (XT) – כ-3600 מילים באוצר הפקודות

1.1.2. 80286 ((AT – שיפור חומרה

1.1.3. 386 – שיפור חומרה

1.1.4. 486 – הוספת מילים לאוצר הפקודות

1.1.5. 586 (PENTIUM) – הוספת מילים לאוצר הפקודות

1.2. complex instruction set compute – CISC

1.2.1. מעבד שמכיר מגוון גדול של פקודות

1.2.2. גודל פקודה משתנה (לכל דגם היה טווח גדלים משתנה)

1.2.3. נדרשים מספר מחזורי FETCH כדי להביא את הפקודות הארוכות

1.3. reduced instruction set compute – RISC

1.3.1. מעבד שמכיר מגוון קטן של פקודות

1.3.2. גודל פקודה קבוע ל-4 בתים (BYTE)

1.3.3. מספיק מחזור FETCH יחיד כדי להביא פקודה למעבד לתחילת טיפול (לכן מוגדרת כארכיטקטורה מהירה)

1.4. השוואת CISC מול RISC

	Complex Instruction Set (CISC)Computer			Reduced Instruction Set (RISC) Computer		Superscalar		
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1973	1978	1989	1987	1991	1993	1996	1996
Number of instructions	208	303	235	69	94	225		
Instruction size (bytes)	2–6	2–57	1–11	4	4	4	4	4
Addressing modes	4	22	11	1	1	2	1	1
Number of general-purpose registers	16	16	8	40 - 520	32	32	40 - 520	32
Control memory size (Kbits)	420	480	246	—	—	—	—	—
Cache size (KBytes)	64	64	8	32	128	16-32	32	64

1.5. מחזורי פס

1.5.1. קריאה מהזיכרון - FETCH

1.5.2. כתיבה אל הזיכרון

1.5.3. קריאה מרכיבי קלט/פלט

1.5.4. כתיבה אל רכיבי קלט/פלט

1.6. מחזור FETCH&EXECUTE

1.6.1. הבאת הוראה מהזיכרון – לפי ערך ה-IP (סה"כ 16 סיביות– 3 בתים)

1.6.2. קידום ערך ה-IP בהתאם לאורך ההוראה (גודל פקודה הוא 2 בייט ולכן הIP מקודם ב-2 בכל צעד)

1.6.3. הכנסת ההוראה לאוגר Instruction Register ופענוחה

1.6.4. הפעלת הפקודה האגורה ב-Instruction Register

1.7. מימוש PIPELINE

1.7.1. יצירת ערוץ הבאה וערוץ ביצוע של פקודות

1.7.2. נועד להגביר את יעילות הביצוע לעבודה "מקבילית"

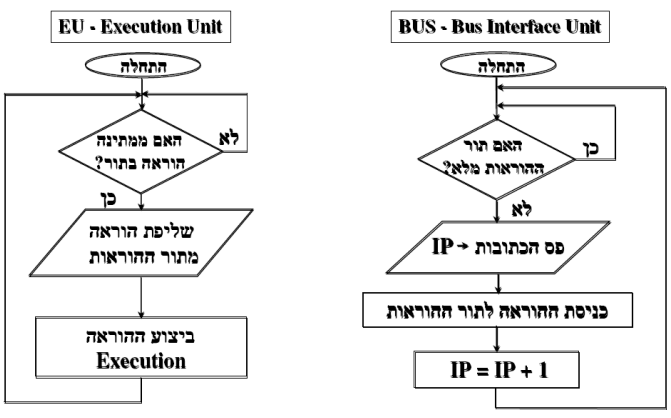
1.7.3. תור ההוראות מאפשר גישה איטית יותר ולכן מעבדים איטיים יותר לא משפיעים על הביצוע

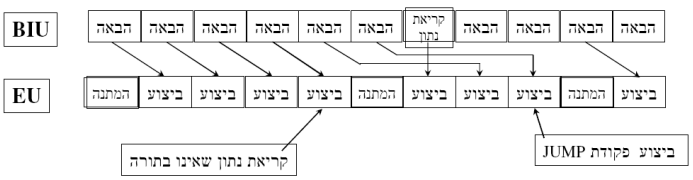
1.7.4. במימוש הPIPELINE יש כשלים:

1.7.4.1. פקודת JMP דורשת לרוקן את התור הישן ולהביא תור חדש של פקודות

1.7.4.2. בפקודה ארוכה (הדורשת מספר מחזורי FETCH) המעבד מבצע המתנה ארוכה בהעברת המידע מהBIU ל EU

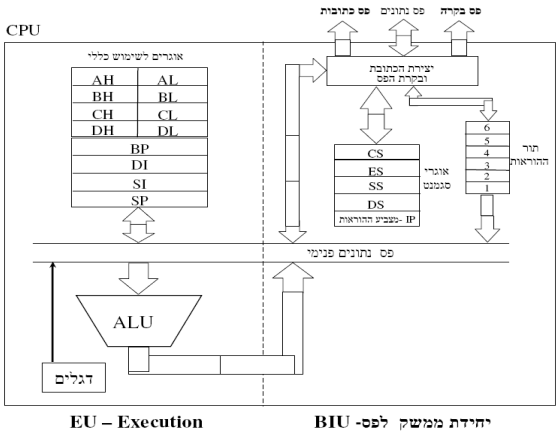
הבאה וביצוע ב - 8086





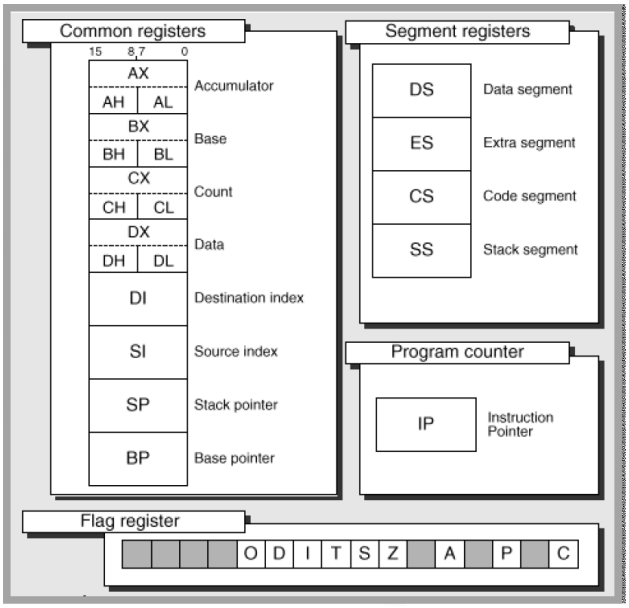
2. רכיבי המעבד

2.1. Bus Interface Unit – BIU
2.1.1. אוסף רכיבי המעבד האחראים על הבאת המידע למעבד
2.2. Execution Unit – EU
2.2.1. אוסף רכיבי המעבד האחראים על חישוב וביצוע הפקודה
2.3. Instruction Register
2.3.1. אוגר את קוד ההוראה מהזיכרון עבור ביצוע הפקודה
2.3.2. אוגר את קוד המידע עבור כתיבה חזרה לזיכרון
2.4. פסי הנתונים – THREE-BUS
2.4.1. פס הכתובות – Address Bus
2.4.1.1. מכיל 16 קווי כתובת A0-A15
2.4.2. פס נתונים – Data Bus
2.4.2.1. מכיל 8 קווי נתון D0-D7
2.4.3. פס בקרה – Control Bus
2.4.3.1. מכיל 4 קווי בקרה
2.4.3.2. עבור 4 סוגי מחזורי הפס
2.5. Random Access Memory – RAM
2.5.1. כל קודי התוכנה והנתונים אגורים בזיכרון זה
2.6. Arithmetic Logic Unit – ALU
2.6.1. רכיב פעולות החישוב שבמעבד



3. <u>מבנה המעבד</u>

3.1. סכימה כללית



3.2. <u>אוגרים</u>

3.2.1. Accumulator – AX

3.2.1.1. רק דרכו ניתן לבצע פעולות קלט/פלט

3.2.1.2. משמש לביצוע הוראות כפל וחילוק

3.2.2. Base – BX

3.2.2.1. יכול לשמש כמצביע לזיכרון

3.2.3. Count – CX

3.2.3.1. יכול לשמש כמונה בביצוע LOOP

3.2.4. Data – DX

3.2.4.1. משמש כמצביע לקלט/פלט

3.2.4.2. מקבל את ערכי הסיביות החל מהסיבית ה-9 ומעלה עבור כפל AX ב16 סיביות אחרות

3.2.5. Destination Index – DI

3.2.5.1. משמש כמצביע יעד לנתון לפי הכתובת DS:DI

3.2.6. Source Index – SI

3.2.6.1. משמש כמצביע מקור הנתון לפי הכתובת ES:SI

3.2.7. Stack Pointer – SP

3.2.8. Base Pointer – BP

3.2.9. Data Segment – DS

3.2.9.1. מצביע המיקום הממשי של תחילת הסגמנט של הנתונים השמורים בזיכרון הRAM

3.2.10. Extra Segment – ES

3.2.10.1. מצביע מיקום נוסף

3.2.10.2. נעזרים בו כאשר לא רוצים לאבד את מיקום אחד הסגמנטים

3.2.10.3. מגיע בדרך כלל יחד עם DI

3.2.11. Code Segment – CS

3.2.11.1. מצביע על המיקום הממשי של תחילת הסגמנט של הקוד

3.2.12. Stack Segment – SS

3.2.12.1. מצביע המיקום הממשי של תחילת הסגמנט של המחסנית

3.2.13. Instruction Pointer – IP

3.2.13.1. מצביע על המיקום היחסי של הקוד לפי המיקום האמיתי CS:IP

3.2.13.2. לפי ההצבעה שלו ההוראות נבחרות לטעינה במעבד ולביצוע

3.3. דגלים – Flag Register

3.3.1. Overflow – O
- 3.3.1.1. כאשר משתנה מסוג SIGNED גדול מדיי או קטן מדיי
- 3.3.1.2. 1 = כאשר התבצע OVERFLOW או UNDERFLOW

3.3.2. Direction – D
- 3.3.2.1. דגל בקר המכתיב כיוון
- 3.3.2.2. 1 = קריאה מכתובת גבוהה לנמוכה ("שמאלה")
- 3.3.2.3. משמש לפקודת קריאה של "מחרוזות"

3.3.3. Interrupt – I
- 3.3.3.1. מגדיר האם תהליך Interrupt יכול להתרחש
- 3.3.3.2. 1 = INT מאופשר

3.3.4. Trap – T
- 3.3.4.1. משמש לצורכי DEBUG
- 3.3.4.2. מפעיל את פעולת ה-Single Step

3.3.5. Sign – S
- 3.3.5.1. מסמן שפעולת החישוב נתנה ערך חיובי או שלילי
- 3.3.5.2. 1 = התקבל ערך שלילי
- 3.3.5.3. פועל על ערכי INTEGER בלבד

3.3.6. Zero – Z
- 3.3.6.1. מסמן שפעולת החישוב נתנה ערך מאופס
- 3.3.6.2. 1 = התקבל ערך מאופס

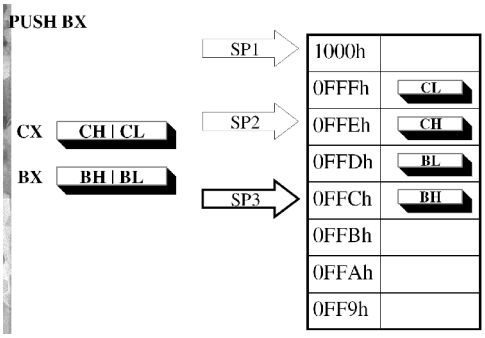
3.3.7. Aux Carry – A
- 3.3.7.1. דומה לדגל CF, רק פועל על החלק הנמוך של האוגרים (DL ,CL ,BL ,AL)

3.3.8. Parity – P
- 3.3.8.1. מסמן שלתוצאה יש מספר זוגי של ערכי הביטים
- 3.3.8.2. 1 = מספר זוגי

3.3.9. Carry – C
- 3.3.9.1. כאשר משתנה מסוג UNSIGNED חורג מגודלו או שנהפך להיות שלילי
- 3.3.9.2. 1 = כאשר התבצע BORROW או CARRY

3.4. המחסנית – Stack

3.4.1. המחסנית בעיקרה נועדה כדי להחזיק את הCS:IP ואת הדגלים בעת מעבר לתת רוטינה, בעת CALL או INT, ולשחזרם לאחר קבלת RET או IRET



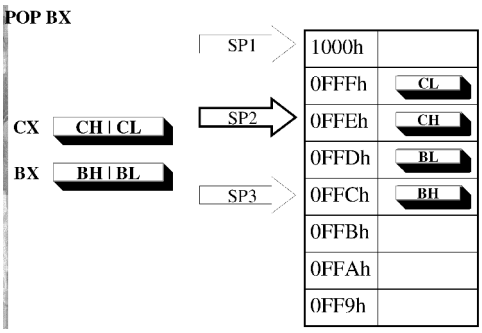
3.4.2. מוגבלת לגודל של 64kBytes

3.4.3. אוגר SP מצביע על ראש המחסנית

3.4.4. קפיצת האוגר תהיה כתלות בכמות המידע הנדחף או הנשלף

3.4.5. דחיפה למחסנית – PUSH
- 3.4.5.1. מתבצע עם אופרנד יחיד, בהתאם לגודלו
- 3.4.5.2. המידע מועתק ממנו למחסנית
- 3.4.5.3. אוגר SP מפחית 2 מערכו, והמידע המוצבע מועתק
- 3.4.5.4. דוגמא: PUSH AX
- 3.4.5.5. בPUSH לאוגר AX, למשל, נדחף קודם AL ואח"כ AH

3.4.6. שליפה מהמחסנית – POP
- 3.4.6.1. מתבצע עם אופרנד יחיד, בהתאם לגודלו
- 3.4.6.2. המידע מועתק מהמחסנית אליו
- 3.4.6.3. המידע מועתק מהמחסנית לאוגר הרלוונטי, ואוגר SP מוסיף 2 לערכו
- 3.4.6.4. דוגמא: POP AX
- 3.4.6.5. בPOP לאוגר AX, למשל, נשלף קודם AH ואח"כ AL



4. **<u>חישוב מיקום כתובת גישה לזיכרון ב-RM</u>**
   4.1. [כתובת היסט ב-16 ביט] + {[16=0000] x [כתובת בסיס ב-16 ביט]} = [כתובת מוחלטת ב-20 ביט]
   4.2. כל רכיב זיכרון מכיל 8 ביט
   4.3. סה"כ גישה ל-1 מגה-בייט של זיכרון
5. **<u>סוגי מיעון:</u>**
   5.1. מיעון ישיר – Direct Addressing
      5.1.1. העברת מידע מאוגר ישירות לזיכרון ולהיפך
      5.1.2. לדוגמא: MOV AX, [a101h]
   5.2. מיעון מיידי – Immediate Addressing
      5.2.1. הגדרת ערך הנתון בגוף ההודעה והעברתו
      5.2.2. לדוגמא: MOV AX, a301h
   5.3. מיעון אוגר – Register Addressing
      5.3.1. העברת מידע בין אוגרים
      5.3.2. לדוגמא: MOV AX, BX
   5.4. מיעון עקיף ע"י אוגר – Indirect Addressing
      5.4.1. הכתובת היחסית של המשתנה שמורה בתוך האוגר, המשמש כמצביע
      5.4.2. אוגר היכול להצביע על נתון בDS: BX, DI, SI
      5.4.3. אוגר היכול להצביע על נתון בSS: BP
      5.4.4. לדוגמא: MOV AX, [BX]
      5.4.5. ניתן לשלב, לדוגמא: MOV AX, [BX+SI+30]
   5.5. מיעון יחסי ע"י אוגר – Array Addressing
      5.5.1. מאפשר לפנות למערך של נתונים המאוחסנים בזיכרון בזה אחר זה
      5.5.2. לדוגמא: MOV AX, vector[BX]

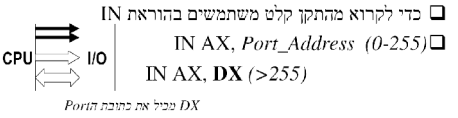
6. קלט/פלט – I/O

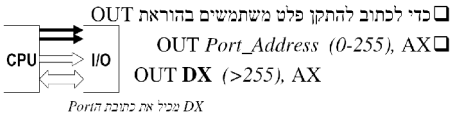
6.1. למעבד גישה ל-256 רכיבי קלט/פלט

6.2. הגישה נעשית דרך PORT ספציפי עבור כל רכיב חומרה

6.3. וקטור הכתובת ל-PORT הוא בגודל 8 סיביות

6.4. כאשר רוצים לגשת ליותר לכתובת הגדולה מ-255, ניתן להשתמש באוגר DX המכיל 16 סיביות כאשר מבצעים IN או OUT





6.5. Programmable Interrupt Controller – PIC

6.5.1. בקר פסיקות

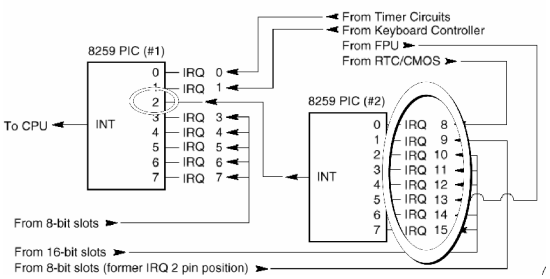
6.5.2. מאפשר להתקני הקלט/פלט לקבל את שירותי המעבד

6.5.3. ה-PIC משתמש ב-INT כדי לקבל את "תשומת לב" המעבד לטפל ברכיב חומרה המחובר אליו

6.5.3.1. ע"י רכיב זה, נמנע הצורך מהמעבד לסרוק את כל ההתקנים כל פעם כדי לבדוק האם אחד מהם דרש INT

6.5.4. PIC אחד יכול לתת מענה ל-8 רכיבי I/O

6.5.5. ע"י חיבור בטור של רכיבי PIC, ניתן לתת מענה ליותר רכיבים



6.5.6. IRQ

6.5.6.1. נקודות חיבור ההתקן ל-PIC

6.5.7. Interrupt Mask Register – IMR

6.5.7.1. אצלו מתקבלות בקשות הפסיקה מרכיבי ה-I/O

6.5.7.2. מכיל את 8 הכניסות מההתקנים

6.5.7.3. סדר הכניסות מגדיר את סדר עדיפויות הטיפול בין הכניסות

6.5.7.4. ניתן למסך (להסתיר) התקן ע"י איפוס הסיבית שמולו

6.5.8. Interrupt Request Register – IRR

6.5.8.1. זוכר את סדר הבקשות שדרשו ההתקנים השונים ועוד לא טופלו

6.5.9. Priority Resolver

6.5.9.1. אוגר בו ניתן להגדיר עדיפויות בין קווי ההתקנים

6.5.10. In Service Register – ISR

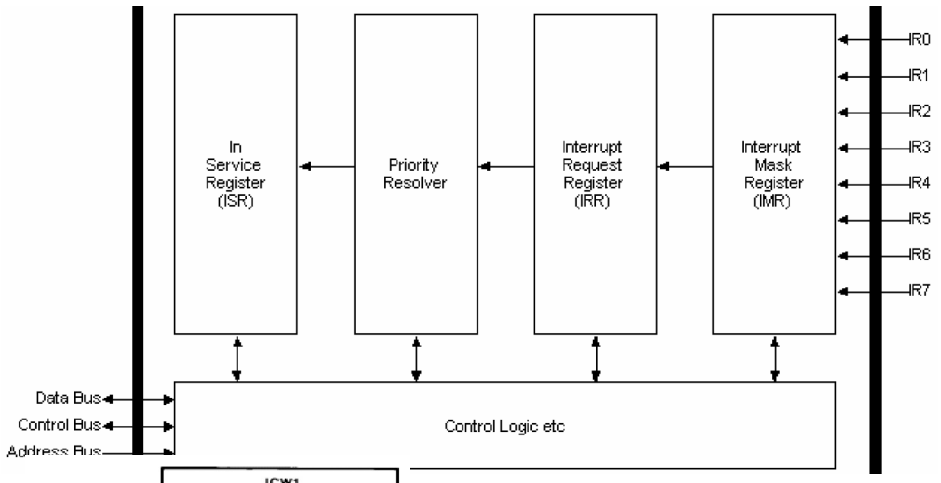
6.5.10.1. מקבל סיבית בודדה (תהיה רק אחת) שמייצגת את ההתקן שידבר עם המעבד

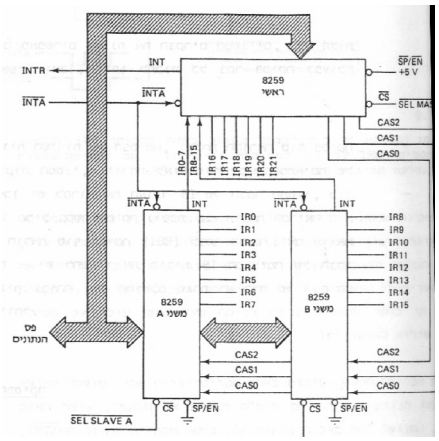
6.5.11. Control Logic

6.5.11.1. מאפשר קינון של הפסיקות (Nested Mode), כדי למנוע מצב שפסיקה בעדיפות נמוכה "תוקעת" את המערכת

6.5.11.2. רכיב זה מאפשר לקבל פסיקות חדשות תוך כדי שפסיקה קיימת מטופלת ע"י המעבד

6.5.11.3. במידה והתקבלה פסיקה חדשה בעלת עדיפות גבוהה יותר, יופסק הטיפול בהתקן הקיים ויעבור לטפל במועדף





6.5.12. Initialization Command Word – ICW

6.5.12.1. מילת תכנות בת 8 סיביות שמאפשרת לקבוע את ערכי ה-PIC

6.5.12.2. קיימות 4 מילות תכנות: ICW1, ICW2, ICW3 , ICW4

6.5.12.3. את המילים בדרך כלל שומרים כמשתנה או כקבוע בקוד

6.5.12.4. עדכון ה-ICW נעשה ע"י הפקודה:

```
MOV AL, ICW2
OUT 20h, AL
```

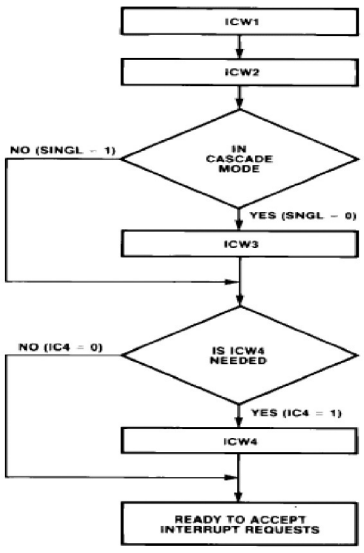
6.5.12.5. במצב בשל בקר פסיקות בודד יש לעדכן לפי סדר את ICW1 , ICW2, ICW4

6.5.12.6. במידה וקיים שרשור, יש לעדכן גם את ICW3

6.5.12.7. באמצעות פקודה זו ניתן להשפיע על ה-IRQ של בקר הפסיקה

6.5.12.8. המעבד מקבל את הערך מה-IRQ

6.5.12.9. המעבד מכפיל את הערך ב-4, ופונה לטבלת הפסיקות (IVT)

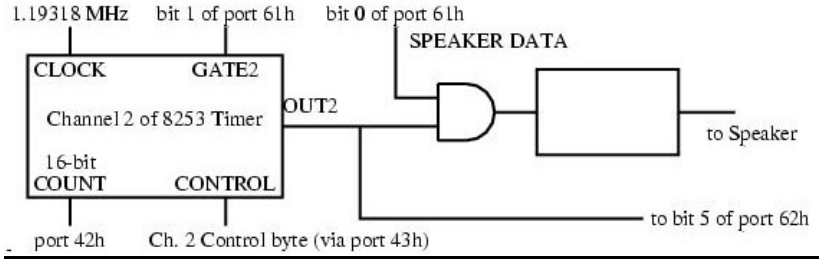


6.5.13. Operation Command Word – OCW

6.5.13.1. תפקידו לעדכן את פעולת הבקר לאחר שעודכנו כל ה-ICW

6.5.13.2. קיימות 3 מילות תכנות מסוג OCW, כל אחת בת 8 סיביות

7. Timer & Speaker



7.1 Timer 8253/4

7.1.1 רכיב שמאפשר לתזמן ארועים

7.1.2 מכיל 3 Counters (OUT0, OUT1, OUT2) בעלי 16 סיביות כל אחד

7.1.2.1 ה-PORT מכיל 8 סיביות ולכן יש לטעון פעמיים בשביל ערך בעל 16 סיביות

7.1.2.2 כתובת PORT של OUT0 הוא 40h

7.1.2.3 כתובת PORT של OUT1 הוא 41h

7.1.2.4 כתובת PORT של OUT2 הוא 42h

7.1.2.5 המונה עובד בקצב של 1193180Hz

7.1.2.6 קצב סיומי מחזור המנייה יכול להגדיר אות שבחיבור ל-Speaker מפיק צליל

7.1.3 מכיל אוגר בקרה – Control Word בן 8 סיביות

7.1.3.1 כתובת PORT אוגר הבקרה הוא 43h

7.1.3.2 לפי הערכים שמצויים בתוך האוגר, נקבע אופן הפעלת ה-Timer ו-3 המונים שלו

7.1.3.3 בכל אתחול אוגר, יש הפנייה למונה מסויים

7.1.3.4 יש להגדיר את ערכי האוגר לפי אתחול המונה

7.1.3.5 את המונה ניתן לאתחל בכל שלב, ואופי פעולתו יוגדר לפי ההגדרות האחרונות שהוטענו

7.1.4 ה-Speaker משתמש ביציאת OUT2

7.1.5 יש תכנן את ה-Counter לעבור ב-Mode 3 (גל ריבועי)

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC — Select Counter:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (See Read Operations)

M — MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW — Read/Write:

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only.
1	0	Read/Write most significant byte only.
1	1	Read/Write least significant byte first, then most significant byte.

BCD:

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

7.2 רכיב 8255

7.2.1 רכיב המכיל אוגר בן 8 סיביות שתפקידו ע"י שערים לוגיים לאפשר הפעלת רכיבים שונים במחשב (מקלדת, שעון, מסך...)

7.2.2 כתובת PORT האוגר היא 61h

7.2.3 סיבית 0:

7.2.3.1 1 = אפשור תקשורת בין ה-Timer ל-Speaker

7.2.4 סיבית 1:

7.2.4.1 1 = אפשור יציאת אות מה-OUT2

7.3 הפעלת הרמקול

7.3.1 שליחת ערך 182 ל-43h (טעינת אוגר הבקרה ב-Timer)

7.3.2 הכנת AX עם ערך הצליל המבוקש

7.3.3 שליחת AL ל-42h

7.3.4 שליחת AH ל-42h

7.3.5 קבלת ערך האוגר של 8255 ע"י IN AL, 61h

7.3.6 שינוי הסיבית 0 ו-1 לערכים "1" ע"י ביצוע 00000011

7.3.7 פתיחת השערים ע"י ביצוע OUT 61h, AL

7.3.8 ביצוע השהייה בקוד לקבלת משך צפצוף רצוי

7.3.9 קבלת ערך האוגר של 8255 ע"י IN AL, 61h

7.3.10 שינוי הסיבית 0 ו-1 לערכים "1" ע"י ביצוע 11111100

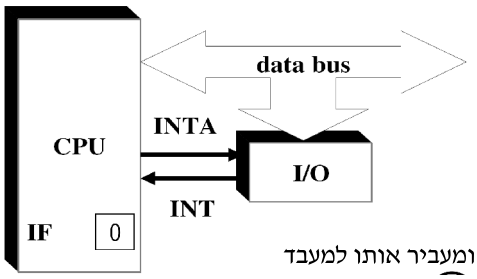
7.3.11 סגירת השערים ע"י ביצוע OUT 61h, AL

Note	Frequency	Frequency #			
C	130.81	9121			
C#	138.59	8609	D#	311.13	3834
D	146.83	8126	E	329.63	3619
D#	155.56	7670	F	349.23	3416
E	164.81	7239	F#	369.99	3224
F	174.61	6833	G	391.00	3043
F#	185.00	6449	G#	415.30	2873
G	196.00	6087	A	440.00	2711
G#	207.65	5746	A#	466.16	2559
A	220.00	5423	B	493.88	2415
A#	233.08	5119	C	523.25	2280
B	246.94	4831	C#	554.37	2152
Middle C	261.63	4560	D	587.33	2031
C#	277.18	4304	D#	622.25	1917
D	293.66	4063	E	659.26	1809

8. פסיקות בחומרה ב-Real Mode

8.1 פסיקה
- 8.1.1 בקשה מהמעבד להשהות את ביצוע התכנית הנוכחית, ולעבור לטיפול בתכנית הנקראת "שגרת טיפול בפסיקה" (ISR)
- 8.1.2 הפסיקה יכולה להתבצע עבור חומרה והן עבור תוכנה

8.2 ISR – Interrupt Subroutine Request
- 8.2.1 בקבלת פסיקה, המעבד מופנה לטפל ברוטינה אחרת



8.3 INT
- 8.3.1 פס בקשה של התקן חומרה חיצוני לקבל פסיקה מהמעבד
- 8.3.2 רוב הזמן הקו מדווח 1, אך ברגע הבקשה הוא עובר רגעית ל-0

8.4 INTA
- 8.4.1 פס המאפשר להתקן חומרה לקבל אישור מהמעבד להפריע בתכנית
- 8.4.2 המעבד שולח אישור להתקן ע"י מעבר של פעמיים מערך 1 ל-0.
- 8.4.3 בירידה השנייה, פס הנתונים, המכיל 16 קווי DATA, דוגם את ערך ההתקן ומעביר אותו למעבד

8.5 IF
- 8.5.1 דגל המגדיר אם לקבל את הבקשה לפסיקה או לא
- 8.5.2 1 = המעבד יקבל את הפסיקה. ומאפס את הדגל כדי לא לאפשר פסיקה נוספת.
- 8.5.3 0 = המעבד לא יקבל את הפסיקה
- 8.5.4 אפשר תכניתית לשנות את הדגל ע"י פקודת CLI (שיאפס) או STI (ירים את הדגל ל1)

8.6 IRET
- 8.6.1 פקודת חזרה משגרת הטיפול בפסיקה

8.7 פסיקת NMI – Non-Maskable Interrupt
- 8.7.1 פסיקה המשמשת למצבי חירום בהם המערכת מגלה בעייה באספקת כוח
- 8.7.2 זוהי פסיקת חומרה שלא ניתן לחסום ע"י דגל
- 8.7.3 אפשר לגשת תוכניתית ולהפעיל את הפסיקה כרצוננו

8.8 פסיקת Devide Error
- 8.8.1 פסיקה המתרחשת בעת ביצוע חלוקה ב-0, מתבצעת פסיקה והמעבד מוציא הודעה

8.9 פסיקת Single Step
- 8.9.1 פסיקה הגורמת לביצוע עצירה של פעולת המעבד אחרי כל הוראה
- 8.9.2 משמשת בעת עבודה עם תוכנת דיבוג כלשהי

8.10 פסיקת into
- 8.10.1 פסיקת תוכנה הבודקת את דגל ה-OVERFLOW
- 8.10.2 מתרחשת כאשר OF=1

8.11 פסיקת INTO
- 8.11.1 פסיקת תוכנה המתרחשת כתוצאה מחלוקה ב-0 הגורמת ל-OVERFLOW באוגר
- 8.11.2 מתרחשת כאשר מבוצעת פעולה אריתמטית שגוייה בין מספרים מסומנים

8.12 פסיקה חיצונית – External Interrupt
- 8.12.1 פסיקה הנגרמת ע"י כל רכיב חומרה חיצוני למעבד (עכבר, מקלדת, טיימר ...)

8.13 פסיקה פנימית – Internal Interrupt
- 8.13.1 פסיקה המתרחשת בתוך המעבד ונגרמת משגיאה או מהוראת INT (קוד הקורא ל-INT, חלוקה ב-0, INTO ...)

8.14 פסיקת תוכנה – Software Interrupt
- 8.14.1 פסיקה הנגרמת מקריאה לביצוע INT

8.15 BIOS
- 8.15.1 רכיב זכרון מסוג ROM
- 8.15.2 מנהל את רכיבי החומרה
- 8.15.3 מכיל את גרעין מערכת ההפעלה, רגע לפני תהליך העלאת מערכת ההפעלה

8.16. IVT – Interrupt Vector Table

8.16.1. זוהי טבלת הפסיקות שבזיכרון המעבד

8.16.2. המעבד מקבל וקטור בן 8 סיביות (INT xxh)

8.16.3. את הווקטור,המעבד מכפיל ב-4 כדי להגיע לתא המתאים ב-IVT

8.16.4. משם הוא שואב את ה-CS וה-IP שמפנים אותו לביצוע רוטינת הפסיקה המתבקשת

8.16.5. ב-Real Mode, הטבלה נמצאת החל מכתובת 0

8.16.6. ב-Protected Mode, הטבלה נמצאת במקום חסוי בזכרון, כדי למנוע גישה ישירה לטבלה

8.16.7. קיימים 256 ווקטורי פסיקות בטבלה (לכן גודלה 1Kbyte)

8.16.8. 5 הוקטורים הראשונים זהים בין כל דגמי המעבדים מ-8086 עד PENTIUM

8.16.9. 32 הוקטורים הראשונים הם לשימושי INTEL

8.16.10. 224 וקטורים אחרונים פנויים לשימוש המשתמש

8.16.11. רשימת סוגי הפסיקות הקיימים:

Type	Function	Comment
0	Divide Error	Processor - zero or overflow
1	Single Step (DEBUG)	Processor - TF=1
2	Nonmaskable Interrupt Pin	Processor - NMI Signal
3	Breakpoint	Processor - Similar to Sing Step
4	Arithmetic Overflow	Processor - into
5	Print Screen Key	BIOS - Key Depressed
6	Invalid Opcode	Processor - Invalid Opcode
7	Coprocessor Not Present	Processor - no FPU
8	Time Signal	BIOS - From RT Chip (AT - IRQ0)
9	Keyboard Service	BIOS - Gen Service (AT - IRQ1)
A - F	Originally Bus Ops (IBM PC)	BIOS - (AT - IRQ2-7)
10	Video Service Request	BIOS - Accesses Video Driver
11	Equipment Check	BIOS - Diagnostic
12	Memory Size	BIOS - DOS Memory
13	Disk Service Request	BIOS - Accesses Disk Driver
14	Serial Port Service Request	BIOS - Accesses Serial Port Drvr
15	Miscellaneous	BIOS - Cassette, etc.
16	Keyboard Service Request	BIOS - Accesses KB Driver
17	Parallel Port LPT Service	BIOS - Printer Driver
18	ROM BASIC	BIOS - BASIC Interpreter in ROM
19	Reboot	BIOS - Bootstrap
1A	Clock Service	BIOS - Time of Day from BIOS
1B	Control-Break Handler	BIOS - Keyboard Break
1C	User Timer Service	BIOS - Timer Tick
1D	Pointer to Video Parm Table	BIOS - Video Initialization
1E	Pointer to Disk Parm Table	BIOS - Disk Subsystem Init.
1F	Pointer to Graphics Fonts	BIOS - CGA Graphics Fonts
20	Program Terminate	DOS - Clear Memory, etc.
21	Function Call	DOS - Transfer Control
22	Terminate Address	DOS - program Terminate handler
23	Control-C Handler	DOS - For OS Use
24	Fatal Error Handler	DOS - Critical Error
25	Absolute Disk Read	DOS - Disk Read
26	Absolute Disk Write	DOS - Disk Write
27	Terminate	DOS - TSR Usage
28	Idle Signal	DOS - Idle
2F	Print Spool	DOS - Cassette, etc.
70-77	Hardware Interrupts in AT Bios	DOS - (AT - IRQs 8-15)

8.17. HALT

8.17.1. פקודה הגורמת למעבד לעצור מעבודתו ולא לבצע לכ פקודה או לקבל פקודות חדשות

8.17.2. יציאה ממצב זה יעשה ע"י אחת מ-2 האפשרויות:

8.17.2.1. פסיקת INT 19h (REBOOT)

8.17.2.2. תתרחש פסיקה חיצונית כלשהי

8.18. REBOOT

8.18.1. אתחול מערכת ההפעלה

8.18.2. נקראת ע"י INT 19h

8.18.3. וקטור הפסיקה מפנה לכתובת ffff:0000 את CS:IP

8.18.4. האוגרים DS, SS, ES מתאפסים

8.18.5. אוגר הדגלים מתאפס

8.19. <u>תהליך הטיפול בפסיקה:</u>

8.19.1. תוכנית מתבצעת ע"י המעבד לפי הקוד המוצבע ע"י CS:IP

8.19.2. התקן מבקש פסיקה דרך קו INT

8.19.3. המעבד מאפס את הדגלים IF, TF

8.19.4. נדחפים למחסנית לפי הסדר: IP, CS, אוגר הדגלים

8.19.5. אישור ביצוע הפסיקה מוחזר להתקן דרך קו INTA

8.19.6. ההתקן מספק את קוד הפסיקה INT דרך פס הנתונים

8.19.6.1. ההתקן מזדהה דרך קוד זה

8.19.6.2. הקוד INT מכיל 8 סיביות (מוגדר בתוך ה- INT xxh)

8.19.6.3. ניתן להגדיר עד 256 התקנים

8.19.6.4. הקוד מפנה את המעבד לטבלת IVT

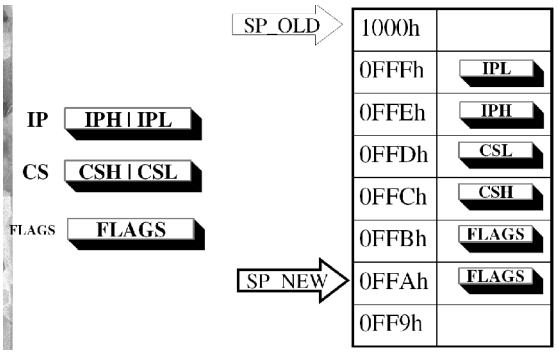
8.19.7. המעבד טוען את וקטור הפסיקה לתוך CS:IP

8.19.8. ביצוע שגרת הפסיקה (ISR)

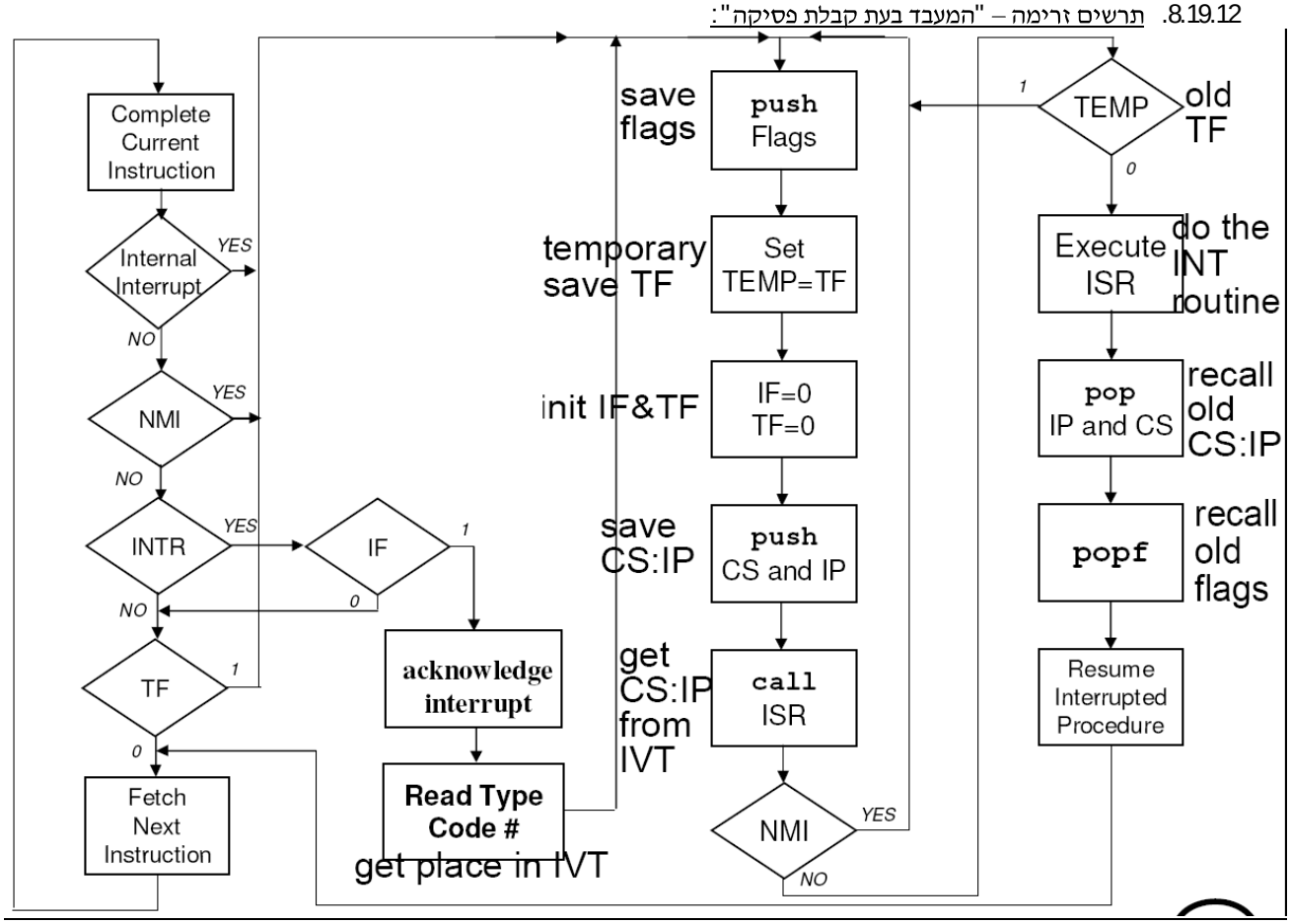
8.19.9. קבלת IRET מתוך שגרת הפסיקה

8.19.10. נשלפים מהמחסנית לפי הסדר: אוגר הדגלים, CS, IP

8.19.11. המעבד חוזר לבצע את התוכנית לפי הקוד המוצבע ע"י CS:IP



8.19.12. <u>תרשים זרימה – "המעבד בעת קבלת פסיקה":</u>



9. Protected Mode – PM

9.1. מצב בו מערכת ההפעלה מבצעת הגנה על הזיכרון

9.2. ההגנה מבוצעת בחלקה ע"י חומרת המחשב

9.3. ה-PM מקיים את עקרון ה-Multy Tasking

9.3.1. המעבד מנהל מספר Tasks במקביל ולא ידרוס זיכרון של Task אחר

9.4. מבנה אוגר ב-PM

15- 3	2	1 - 0
Selector	TI	RPL/DPL

9.4.1. Selector

9.4.1.1. אוגרי הסגמנט אינם מכילים את ההצבעה לסגמנט עצמו, אלא הצבעה לאחד מ-8k ה-Descriptors השונים

9.4.1.2. 13 סיביות Selector. בוחר את אחד מ-8000 ה-Descriptors. (2^13=8k)

9.4.2. 1 סיבית TI (בחירת סוג DT). GDT = 0. LDT = 1

9.4.3. 2 סיביות Request/Code Privilege Level – RPL/CPL

9.4.3.1. רמת הרשאה/זכות לפנות ל-DT ולקבל את כתובת הזכרון הנדרשת

9.4.3.2. 00 – הרמה הגבוהה ביותר. מאפשר גישה לכל Descriptor

9.4.3.3. 11 – הרמה הנמוכה ביותר. לא מאפשר לגשת לכל מיני מקומות. למשל, סגמנט מערכת ההפעלה

9.4.3.4. במידה ורמת ה-RPL מתאימה או גבוהה מהרמה הנדרשת, הפנייה ל-DT מתקבלת, אחרת – נדחית

9.4.3.5. ה-RPL שייך לאוגרים DS, SS, ES ...

9.4.3.6. ה-CPL שייך לאוגר CS

9.5. Descriptor Table – DT

9.5.1. טבלה המכילה את אוסף ההצבעות הקונקרטיות לזיכרון

9.5.2. הגישה נעשית ע"י אוגר שמצביע למקום הרלוונטי ב-DT

9.5.2.1. לדוגמא:

9.5.2.1.1. כדי לבצע: MOV AX, [BX]

9.5.2.1.2. צריך לספק לאוגר DS את הכתובת המתאימה ב-DT

9.5.2.1.3. לשאוב משם את הכתובת הנדרשת שבזיכרון ולפנות לזיכרון שבאותה הכתובת

9.5.3. ה-DT הוא בגודל 8Kbyte, ומחולק ל-8192 יחידות

9.5.4. כל יחידה (Descriptor) בגודל 8 בתים (סה"כ ניתן לנהל 8000 סגמנטי זיכרון)

9.5.5. קיימות 2 טבלאות DT במעבד (GDT, LDT) – סה"כ ניתן לנהל 16000 סגמנטי זיכרון

9.6. Global DT – GDT

9.6.1. טבלה כללית / טבלת מערכת

9.6.2. מכיל הגדרות סגמנטים המתאימות לכל תכנית

9.7. Local DT – LDT

9.7.1. טבלה מקומית / טבלת אפליקציה

9.7.2. מכיל הגדרות סגמנטים הספציפיים ליישומים מסויימים

9.8. GDT Register – GDTR

9.8.1. אוגר המצביע לבסיס טבלת ה-GDT

9.9. LDT Descriptor – LDTD

9.9.1. Descriptor הנמצא ב-GDT ומכיל הצבעה לטבלת ה-LDT

9.10. LDT Register – LDTR

9.10.1. אוגר המצביע ל-LDTD, כדי לקבל הפנייה ל-LDT

9.11. Basic Segment Address – BSA

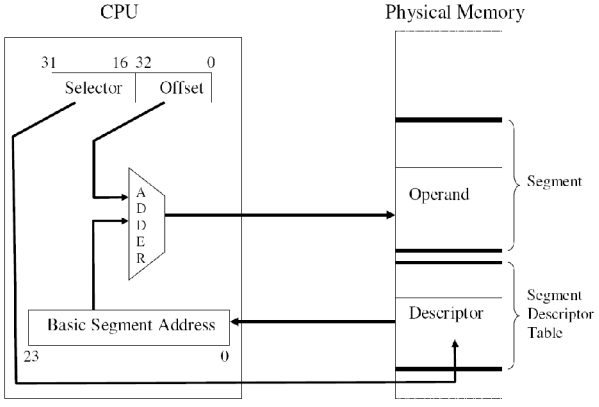
9.11.1. אוגר המקבל מה-DT את הכתובת האמיתית בזיכרון של בסיס הסגמנט הרצוי

9.12. גישה לזיכרון ב-PM

9.12.1. אוגר Selector (למשל, DS, או כל אוגר אחר הפונה ל-DT) פונה ל-Descriptor במיקום מסוים ב-DT

9.12.2. ה-Descriptor מספק את הכתובת לבסיס הסגמנט ומעביר אותה ל-BSA

9.12.3. אוגר ה-Offset (למשל, BX) יחד עם ה-BSA מגדירים מיקום מדויק בזיכרון



386-Pentium III Descriptor

Base B31-B24	G	D	0	AV	Limit L19-L16
Access rights	Base(B23-B16)				
Base (B15-B0)					
Limit (L15-L0)					

80286 descriptor

7	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	6
5	Access rights	Base (B23 – B16)	4
3	Base (B15-B0)		2
1	Limit (L15-L0)		0

9.13.1. זוהי יחידה ב-DT, בגודל 8 בתים (64 סיביות)

9.13.2. Base Address

9.13.2.1. מציין את כתובת תחילת הסגמנט

9.13.2.2. ב-8086, הפועל ב-RM, מוקצות 20 סיביות, גישה ישירה ל-1MByte זכרון

9.13.2.3. ב-286, הפועל ב-PM, מוקצות 24 סיביות, גישה עקיפה ל-16MByte זיכרון

9.13.2.4. ב-386 והלאה, הפועל ב-PM, מוקצות 32 סיביות, גישה עקיפה ל-4GByte זיכרון

9.13.3. Segment Limit

9.13.3.1. מציין את גבול הסגמנט

9.13.3.2. ה-Limit מגדיר את ה-Offset המירבי המותר בסגמנט הספציפי

9.13.3.3. ב-8086, הפועל ב-RM, מוקצות 16 סיביות, גישה לכתובת יחסית עד גודל 64KByte

9.13.3.4. ב-286, הפועל ב-PM, מוקצות 16 סיביות, גישה לכתובת יחסית עד גודל 64KByte

9.13.3.5. ב-386 והלאה, הפועל ב-PM, מוקצות 20 סיביות, גישה לכתובת יחסית עד גודל 1MByte כאשר ניגשים לבית ספציפי או 4MByte כאשר מבצעים דילוג של 4 בתים בין כתובת לכתובת

9.13.4. Granularity – G

9.13.4.1. 1 סיבית

9.13.4.2. מגדיר את גודל הדילוג של פנייה לסגמנט בין מקום אחד בזכרון לשכנו

9.13.4.3. בכך מאפשר גישה לזיכרון בגודל 1GByte או 4GByte

9.13.4.4. G =0

9.13.4.4.1. גודל הדילוג יהיה 1 בית

9.13.4.4.2. טווח הכתובות : 00000h – fffffh

9.13.4.5. G =1

9.13.4.5.1. גודל הדילוג יהיה 4 בתים

9.13.4.5.2. טווח הכתובות : 00000XXXh – fffffXXXh

9.13.5. D/B

9.13.5.1. תומך את אופן עבודת המעבד ב-RM או ב-PM

9.13.5.2. 0 = עבודה עם אוגרים בגודל 16 סיביות

9.13.5.3. 1 = עבודה עם אוגרים בגודל 32 סיביות

9.13.5.4. לפי הדגל, המעבד ידע לקחת את גודל האופרנד הנדרש

9.13.5.5. בחומרת CISC אורך קודי המכונה משתנה מהוראה להוראה, וב-RISC הוא קבוע. המחשב צריך לדעת לפי הפקודה, ולפי הדגל, לכמה מידע להתייחס.

9.13.5.6. לדוגמא, בפקודה : B8 90909090

9.13.5.6.1. ה-OP Code מסוג B8, הוא פקודת מכונה המבצעת MOV וצורכת 1Byte

9.13.5.6.2. 9090h – צורך 2 בתים לזיכרון

9.13.5.6.3. 90909090h – צורך 4 בתים לזיכרון

9.13.5.6.4. אם D/B = 0, יתבצע : MOV AX, 9090h, צורך סה"כ 3Bytes זיכרון

9.13.5.6.5. אם D/B = 1, יתבצע : MOV EAX, 90909090h, צורך סה"כ 5Bytes זיכרון

9.13.6. AVL

9.13.6.1. סיבית המיועדת לשימוש של המערכת

9.13.7. O

9.13.7.1. סיבית חסויה לשימוש INTEL

9.13.8. Access Rights

9.13.8.1. 8 סיביות

7	6-5	4	3	2	1	0
P	DPL	S	E	ED/C	R/W	A

9.13.8.2. A

9.13.8.2.1. 0 = לא קיימת גישה לסגמנט

9.13.8.2.2. 1 = קיימת גישה לסגמנט

9.13.8.3. S

9.13.8.3.1. 0 = ה-Descriptor שייך למערכת הפעלה

9.13.8.3.2. 1 = ה-Descriptor מצביע על אזור ב-CS או ב-DS

9.13.8.4. DPL – Descriptor Privilege Level

9.13.8.4.1. 00 – הרמה הגבוהה ביותר.

9.13.8.4.2. 11 – הרמה הנמוכה ביותר.

9.13.8.4.3. מגדיר את רמת ההרשאה המינימלית הנדרשת לקבל אישור גישה לזיכרון

9.13.8.4.4. מתבצעת השוואה אל מול הRPL של האוגר Selector הפונה לDescriptor

9.13.8.5. P

9.13.8.5.1. 0 = ה-Descriptor לא מוגדר ולא בשימוש

9.13.8.5.2. 1 = ה-Descriptor מוגדר ובשימוש

9.13.8.6. E

9.13.8.6.1. 0 = ה-Descriptor מטפל בסגמנט ה-DATA

9.13.8.6.1.1. R/W

9.13.8.6.1.1.1. מתייחס ל-W

9.13.8.6.1.1.2. 0 = ניתן לגשת אך לא ניתן לכתוב לזיכרון.

9.13.8.6.1.1.3. 1 = ניתן לגשת וגם ניתן לכתוב לזיכרון.

9.13.8.6.1.2. ED/C

9.13.8.6.1.2.1. מתייחס ל-ED

9.13.8.6.1.2.2. 0 = כיוון הסגמנט הוא כאשר ערכי הכתובות עולים . טיפול ב-DATA רגיל.

9.13.8.6.1.2.3. 1 = כיוון הסגמנט הוא כאשר ערכי הכתובות יורדים . טיפול ב-STACK.

9.13.8.6.2. 1 = ה-Descriptor מטפל בסגמנט ה-CODE

9.13.8.6.2.1. R/W

9.13.8.6.2.1.1. מתייחס ל-R

9.13.8.6.2.1.2. 0 = מאפשר להתעלם מרמת ההרשאות – גישה חופשית לקוד.

9.13.8.6.2.1.3. 1 = כופה בדיקה אל מול רמת הרשאה.

9.13.8.6.2.2. ED/C

9.13.8.6.2.2.1. מתייחס ל-C

9.13.8.6.2.2.2. 0 = לא קיימת גישה לסגמנט

9.13.8.6.2.2.3. 1 = קיימת גישה לסגמנט

9.14. Flat Model

9.14.1. כאשר משתמשים בסגמנט יחיד בכל מרחב הזיכרון

9.15. Virtual Model

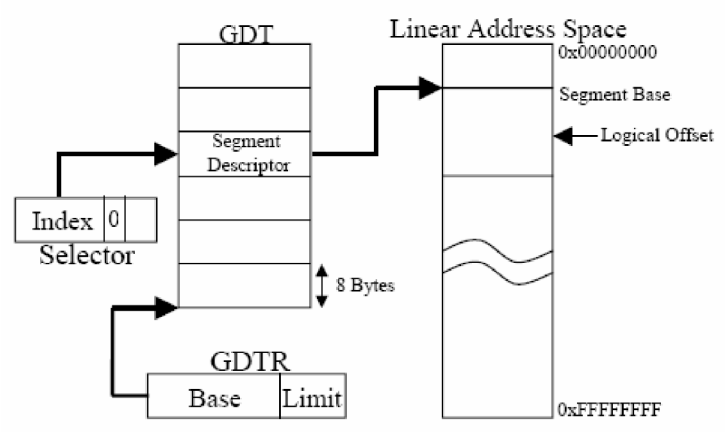
9.15.1. נקרא גם Virtual Memory

9.15.2. ניהול הקצאת הזיכרון ע"י מערכת ההפעלה היא ע"י וירטואליזציה של החומרה

9.15.3. הניהול נעשה ב-Protected Mode

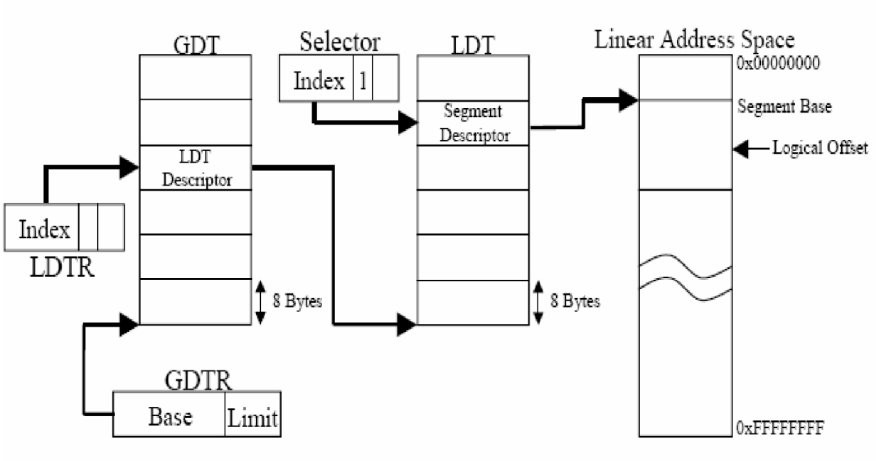
9.15.4. הגישה לזיכרון תעשה תמיד דרך Descriptor

9.16. גישה ל-GDT



9.16.1. ה-GDTR מכוון לבסיס ה-GDT
9.16.2. ע"י Selector ניתן לגשת לטבלת ה-GDT, ולפי הרשאה ניתן לקבל גישה לזיכרון לצורך ביצוע הפעולה

9.17. גישה ל-LDT



9.17.1. ה-GDTR מכוון לבסיס ה-GDT
9.17.2. ע"י LDTR ניתן לגשת לטבלת ה-GDT אל ה-LDTD
9.17.3. שם לקבל את המידע לגבי המיקום של ה-LDT
9.17.4. ע"י Selector ניתן לגשת לטבלת ה-LDT, ולפי הרשאה ניתן לקבל גישה לזיכרון לצורך ביצוע הפעולה

9.18. האוגרים ב-PM

9.18.1. מידע על סטאטוס האוגרים CS, SS, DS, ES, FS, GS מוצגים באופן חלקי למשתמש
9.18.2. החלק הגלוי מציג את המידע לגבי ה-Segment Selector
9.18.3. המידע החסוי מכיל את הנתונים לגבי כתובת הבסיס, ה- CPL / RPL, ומידע הגישה לטבלאות ה-DT

9.19. אתחול מחשב ב-PM

9.19.1. צור GDT
9.19.2. צור IDT
9.19.3. חסום פסיקות – בזמן הגדרת ה-GDT וה-IDT, כל תהליך הקריאה לזיכרון עוד לא מוגדר
9.19.4. טען את ה-GDTR ל-GDT
9.19.5. טען את ה-IDTR ל-IDT
9.19.6. שנה את הסיבית PE (Protected Enabled) ל-1 באוגר ה-IDT
9.19.7. טען את CS ואת EIP באמצעות ה-GDT
9.19.8. טען את DS ואת SS ע"י ה-Segment Selector

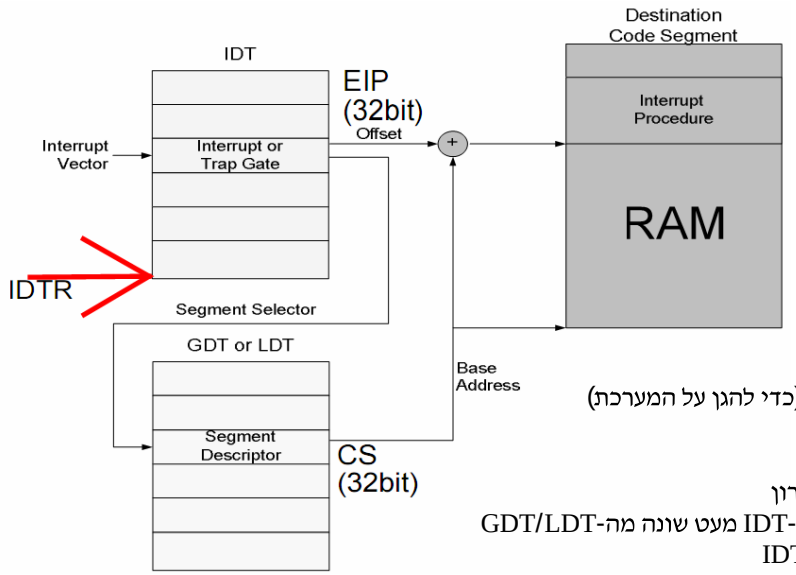
9.20. ההוראות ברמת הרשאה הכי גבוהה

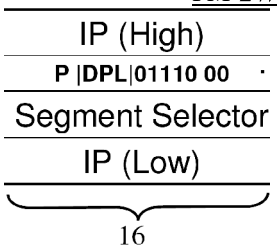
9.20.1. זוהי רשימת הוראות שאין למשתמש הרשאה לבצע אותן
9.20.2. ניתן לבצע רק ברמת הרשאה גבוהות כאשר עובדים בסביבת PM

- **LGDT** – Load GDT register
- **LLDT** – Load LDT register
- **LTR** – Load task register
- **LIDT** – Load IDT register
- **MOV CR** – Load and store control registers
- **LMSW** – Load machine status word
- **CLTS** – Clear task-switch flag in register CR0

- **MOV DR** – Load and store debug registers
- **INVD** – Invalidate cache without write-back
- **WBINVD** – Invalidate cache with write-back
- **INVLPG** – Invalidate TLB entry
- **HLT** – Halt processor
- **RDMSR/WRMSR** – Read/Write Model-Specific Registers
- **RDPMC** – Read Performance-Monitoring Counters
- **RDTSC** – Read Time-Stamp Counter

9.21. פסיקות חומרה ב-PM





9.21.1. Interrupt Descriptor Table – IDT
9.21.1.1. טבלה הדומה ל-IVT
9.21.1.2. הטבלה ממוקמת במקום לא ידוע בזיכרון (כדי להגן על המערכת)
9.21.1.3. גודל וקטור בטבלה הוא 8 בתים
9.21.2. IDT Register – IDTR
9.21.2.1. אוגר המצביע על מיקום בסיס הטבלה בזכרון
9.21.3. הפסיקה עובדת באופן דומה, אך מבנה המידע ב-IDT מעט שונה מה-GDT/LDT
9.21.4. קיים במעבד אוגר IDTR המצביע על טבלת ה-IDT
9.21.5. גישה לאוגר ע"י פקודת LIDT היא ברמת הרשאה הכי גבוהה ולכן חסומה למשתמש
9.21.6. כל יחידת פסיקה היא בגודל 8 בתים
9.21.7. הטבלה יכולה להיות ממוקמת בכל מקום בזיכרון (הטבלה בגודל 256 יחידות פסיקה)
9.21.8. IRETD
9.21.8.1. בסיום פסיקה – שולף את מידע הDescriptor (שמכיל את ECS), את EIP, ואת אוגר הדגלים (EFLAGS)
9.21.8.2. תזכורת:
9.21.8.2.1. IRET – שולף את IP, CS, ואת FLAGS
9.21.8.2.2. RET – שולף את IP ואת CS

9.22. Swap File
9.22.1. מידע שיושב ב-HD ומהווה מחסן זמני לזיכרון המחשב
9.22.2. כך ניתן להוסיף זיכרון שישמש את המעבד מעבר לכמות הזיכרון שה-RAM מסוגל להחזיק
9.22.3. קובץ ה-Swap יושב בתוך ה-Virtual Memory

9.23. Paging – שיטת העימוד
9.23.1. שיטה בה מחלקים את הזיכרון למקטעים שווים
9.23.2. כל מקטע נקרא page
9.23.3. במערכות PC גודל ה-Page הוא 4K-8K בתים
9.23.4. מערכת ההפעלה מטעינה מהדיסק הקשיח חלקים לעמודים האלה
9.23.5. קיים סיכוי שקוד ריצה שאמור ללכת למעבד, יהיה ברגע נתון טעון על הדיסק הקשיח ולא בזיכרון ה-RAM
9.23.6. החומרה מנהלת את הקריאה למקטעי הזיכרון השונים
9.23.7. יתרונות וחסרונות השיטה:
9.23.7.1. פרוצדורות באורכים משתנים
9.23.7.1.1. כמות הקוד, גדלי מחסניות משתנים, כמות נתונים משתנה לכל תכנית, גורמים לכך שלא ניתן לתת סדר גודל קבוע כדי להקצות משאב זיכרון.
9.23.7.2. בעיה של פרגמנטציה
9.23.7.2.1. בגלל עקרון הMulty-Tasking, אין ציפוף טוב לזיכרון ולכן הניצול של מרחב הזיכרון אינו מיטבי
9.23.7.3. זמן מעבר בין סגמנטים
9.23.7.3.1. תהליך יכול להיות מחולק למספר מקטעים לא מסודרים ברציפות בזיכרון
9.23.7.3.2. קפיצה מרובה בין סגמנטים יכולה להאט את מהירות התכנית
9.23.7.4. זמן ביצוע Swap
9.23.7.4.1. תהליך הטענת מידע מזיכרון ה-RAM ל-HD ומשיכה של מידע מה-HD למקום שהתפנה לוקח זמן רב
9.23.7.4.2. תהליך זה מאט את מהירות התכנית

10. הבדלים בין Real Mode לבין Protected Mode

10.1. שימוש באוגרים השונים

10.1.1. RM

16-bit Default Segment + Offset address combinations:

Segment	Offset	Special Purpose
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX, DI, SI, an 8- or 16-bit number	Data address
ES	DI for string ops	String destination address

10.1.2. PM

32-bit Default Segment + Offset address combinations:

Segment	Offset	Special Purpose
CS	EIP	Instruction address
SS	ESP or EBP	Stack address
DS	EBX, EDI, ESI, EAX, ECX, EDX, an 8- or 32-bit number	Data address
ES	DI for string ops	String destination address
FS	No default	General address (386+)
GS	No default	General address (386+)

10.2. מרחב הזיכרון

10.2.1. RM

10.2.1.1. גודל אוגר – 16 סיביות
10.2.1.2. מרחב זיכרון של עד 1MByte
10.2.1.3. גודל סגמנט מקסימאלי הוא 64Kbyte
10.2.1.4. ההצבעה לזיכרון היא ישירות ע"י הסגמנט המתאים
10.2.1.5. ניתן לבצע חפיפה בין הסגמנטים
10.2.1.6. אחריות ביצוע LINK ו-LOAD של תכנית היא על מערכת ההפעלה
10.2.1.7. אחריות הקצאת הסגמנטים עבור המחסנית, הקוד ואיתחול האוגרים השונים היא על מערכת ההפעלה
10.2.1.8. model Small – הנחייה לקומפיילר לאפשר חפיפה בין הסגמנטים CS ל-DS
10.2.1.9. הזזת מיקום הסגמנט נעשה ע"י שינוי ערך האוגר המתאים

10.2.2. PM

10.2.2.1. גודל אוגר – 32 סיביות
10.2.2.2. מרחב זיכרון של עד 4GByte
10.2.2.3. גודל סגמנט מקסימאלי הוא כל הזיכרון
10.2.2.4. ההצבעה לזיכרון נעשית באופן עקיף ע"י הצבעה למקום בטבלת ה-DT (ומשם למקום הנכון בזיכרון)
10.2.2.5. ניתן לשלוט על הרשאת הגישה לכל סגמנט
10.2.2.6. מנגנון מסובך לתחזוקה שוטפת מבינת תרגום הכתובות
10.2.2.7. קשה ומורכב לכתוב תוכנה, כיוון שאין גישה ישירה לזיכרון
10.2.2.8. קשה לנפות שגיאות, כיוון שאין גישה ישירה לזיכרון

11. **<u>מקוד אסמבלי להרצת תכנית</u>**

11.1. עריכה בעורך אסמבלר

11.1.1. מתקבל קובץ ASCII (FILE.asm)

11.2. קומפילציה

11.2.1. מתקבל קובץ בינארי (FILE.obj)

11.2.2. מתרחש תהליך של ה-LINKER

11.2.3. מתקבל קובץ HEADER בינארי (FILE.exe)

11.3. הרצה

11.3.1. ה-LOADER טוען את הקוד למעבד ע"י העברת מידע בינארי

11.3.2. המעבד מריץ את התכנית

11.4. <u>HEADER</u>

11.4.1. קטע קוד שמאפשר להעביר את הבקרה ממערכת ההפעלה לתכנית

11.4.2. דואג לחבר מספר תוכניות מסוג ASM

11.5. <u>UnAssembly / DisAssembly</u>

11.5.1. פקודה לביצוע Reverse Engineering ההופכת קוד בינארי לקוד המקור

11.6. <u>קובץ LST</u>

11.6.1. קובץ המציג תמונה משותפת של :

11.6.1.1. הקוד

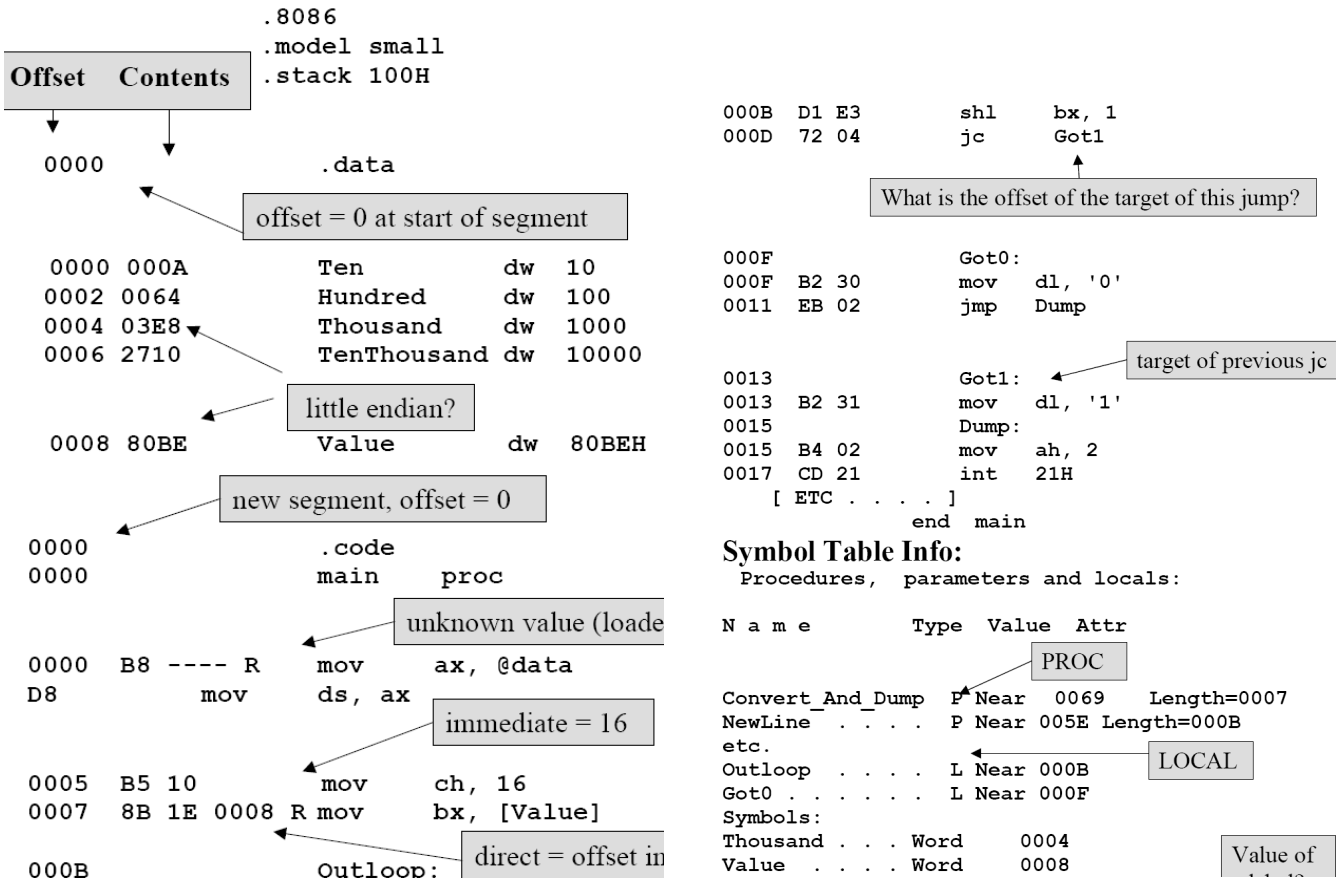
11.6.1.1.1. כתובת ה-Offset (IP) בתוך הסגמנט

11.6.1.1.2. שפת המכונה

11.6.1.1.3. תכנית האסמבלר

11.6.1.2. טבלת משתני המערכת ותגיות

11.6.1.2.1. שם התגית/משתנה, סוג והכתובת היחסית בסגמנט



12. **<u>תצוגת קוד אסמבלר</u>**

12.1. לדוגמא:

EB8:0107 BB0104 MOV BX, 0401

12.2. EB8:0107 – מייצג את כתובת הקוד בזיכרון הטעון ב- CS:IP

12.3. BB – מייצג צירוף לוגי שמנחה את המשך הפקודה

12.4. 01 – מייצג את הערכים הנמוכים 0401

12.5. 04 – מייצג את הערכים הגבוהים של 0401

12.6. MOV BX, 0401 – מייצג את הפקודה באסמבלר

13. **<u>גדלי המשתנים בקוד</u>**

13.1. DD = 32bit, DW = 16bit, DB = 8bit

14. <u>מבנה כללי של תכנית אסמבלר (פורמט model small.)</u>

```
.model small     ; מגדיר גודל מקטע נתונים לסגמנט שיהיה בגודל מירבי של 64 ק"ב
#MAKE_EXE#
;****Stack Segment*****************************************************************
.stack  dw 100h  ;הגדרת המחסנית וגודלה
;****Data Segment******************************************************************
.data  ;הגדרת סגמנט הנתונים
Five equ 5d   ;הגדרת קבוע בתכנית
Message db 'Hello', 13, 10, '$'  ;הגדרת מחרוזת. יש לשים בסוף $. 13 מגדיר "שורה חדשה". 10 מגדיר "החזרת סמן לתחילת השורה
;***** Code Segment ***************************************************************
.code  ;הגדרת סגמנט הקוד
START:  ;הגדרת תגית תחילת התכנית הראשית
pusha ;save all registers
pushf ;save all flags

   Mov ax, @data ;init DS on the data segment
   Mov ds, ax
   Mov ax, 0a000h ;init ES on beginning of the screen
   Mov es, ax
…
…
popf ;resume all flags
popa ;resume all registers
;-------------end program, returns the control to the operating system----------
Endprogram:
   mov ax, 4c00h
   int 21h  ;החזרת השליטה על המעבד אל מערכת ההפעלה
;*************************************End of the Main Program**********************
end  ;הנחייה למהדר על סיום הקוד
```

15. <u>מבנה כללי של תכנית אסמבלר (פורמט אחר)</u>

```
assume cs:cod_seg, ds:dat_seg, ss:st_seg  ;מתן שמות לאוגרים
;****Data Segment******************************************************************
dat_seg  segment  ;הגדרה למהדר לתחילת הסגמנט הנתונים
…
dat_seg ends  ;הגדרה למהדר על סיום סגמנט הנתונים
;****Stack Segment*****************************************************************
st_seg  segment  ;הגדרה למהדר לתחילת סגמנט המחסנית
    DW  10 dup(?)  ;הגדרת מחסנית בעלת 10 תאים בגודל 2 בתים כל מילה. ? מגדיר שכל תא יקבל ערך ברמ"ח
st_seg ends  ;הגדרה למהדר על סיום סגמנט המחסנית
;***** Code Segment ***************************************************************
cod_seg  segment  ;הגדרה למהדר לתחילת סגמנט הקוד
START:  ;הגדרת תגית תחילת התכנית הראשית
pusha ;save all registers
pushf ;save all flags

   Mov ax, dat_seg  ;init DS on the data segment
   Mov ds, ax
   Mov ax, 0a000h ;init ES on beginning of the screen
   Mov es, ax
…
…
popf ;resume all flags
popa ;resume all registers
;-------------end program, returns the control to the operating system----------
Endprogram:
   mov ax, 4c00h
   int 21h  ;החזרת השליטה על המעבד אל מערכת ההפעלה
;*************************************End of the Main Program**********************
cod_seg ends  ;הגדרה למהדר על סיום סגמנט הקוד
end cs:start, ds:dat_seg, ss:st_seg  ;הנחייה למהדר על סיום עבודתו
```

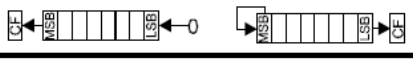
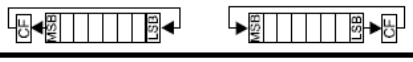
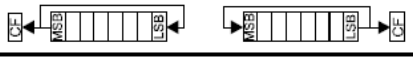
Intel Assembler 80186 and higher

CodeTable 1/2

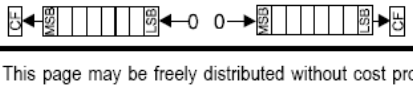
© 1996-2003 by Roger Jegerlehner, Switzerland
V 2.3 English. Also available in Spanish

TRANSFER				Flags								
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C
MOV	Move (copy)	MOV Dest,Source	Dest:=Source									
XCHG	Exchange	XCHG Op1,Op2	Op1:=Op2 , Op2:=Op1									
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*	Input	IN Dest, Port	AL/AX/EAX := byte/word/double of specified port									
OUT *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 Dest,Source	Dest:=Dest+Source	±				±	±	±	±	±
ADC	Add with Carry	ADC Dest,Source	Dest:=Dest+Source+CF	±				±	±	±	±	±
SUB	Subtract	SUB Dest,Source	Dest:=Dest-Source	±				±	±	±	±	±
SBB	Subtract with borrow	SBB Dest,Source	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*	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 Op1,Op2	Op1-Op2	±				±	±	±	±	±
SAL	Shift arithmetic left (≡ SHL)	SAL Op,Quantity		*i*				±	±	?	±	±
SAR	Shift arithmetic right	SAR Op,Quantity		*i*				±	±	?	±	±
RCL	Rotate left through Carry	RCL Op,Quantity		*i*								±
RCR	Rotate right through Carry	RCR Op,Quantity		*i*								±
ROL	Rotate left	ROL Op,Quantity		*i*								±
ROR	Rotate right	ROR Op,Quantity		*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 Dest,Source	Dest:=Dest∧Source	0				±	±	?	±	0
OR	Logical or	OR Dest,Source	Dest:=Dest∨Source	0				±	±	?	±	0
XOR	Logical exclusive or	XOR Dest,Source	Dest:=Dest (exor) Source	0				±	±	?	±	0
SHL	Shift logical left (≡ SAL)	SHL Op,Quantity		*i*				±	±	?	±	±
SHR	Shift logical right	SHR Op,Quantity		*i*				±	±	?	±	±

Download latest version free of charge from www.jegerlehner.ch/intel This page may be freely distributed without cost provided it is not changed. All rights reserved

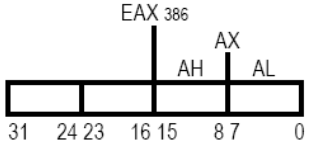
© 1996-2003 by Roger Jegerlehner, Switzerland
V 2.3 English. Also available in Spanish

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 Dest,Source	Dest := address of Source									
INT	Interrupt	INT Nr	interrupts current program, runs spec. int-program			0	0					

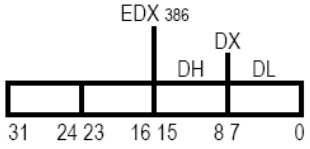
JUMPS (flags remain unchanged)							
Name	Comment	Code	Operation	Name	Comment	Code	Operation
CALL	Call subroutine	CALL Proc		RET	Return from subroutine	RET	
JMP	Jump	JMP Dest					
JE	Jump if Equal	JE Dest	(≡ JZ)	JNE	Jump if not Equal	JNE Dest	(≡ JNZ)
JZ	Jump if Zero	JZ Dest	(≡ JE)	JNZ	Jump if not Zero	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)
JAE	Jump if Above or Equal	JAE 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	

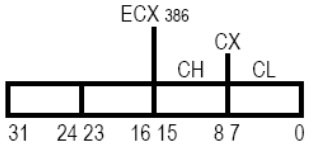
General Registers:



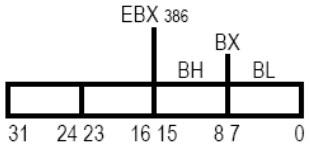
Accumulator



Data mul, div, IO



Count loop, shift



BaseX data ptr

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: | - | - | - | - | O | D | I | T | S | Z | - | A | - | P | - | C |

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

Download latest version free of charge from www.jegerlehner.ch/intel This page may be freely distributed without cost provided it is not changed. All rights reserved

17. טבלת ASCII

Character Name	Char	Code	Decimal	Binary	Hex
Null	NUL	Ctrl @	0	00000000	00
Start of Heading	SOH	Ctrl A	1	00000001	01
Start of Text	STX	Ctrl B	2	00000010	02
End of Text	ETX	Ctrl C	3	00000011	03
End of Transmit	EOT	Ctrl D	4	00000100	04
Enquiry	ENQ	Ctrl E	5	00000101	05
Acknowledge	ACK	Ctrl F	6	00000110	06
Bell	BEL	Ctrl G	7	00000111	07
Back Space	BS	Ctrl H	8	00001000	08
Horizontal Tab	TAB	Ctrl I	9	00001001	09
Line Feed	LF	Ctrl J	10	00001010	0A
Vertical Tab	VT	Ctrl K	11	00001011	0B
Form Feed	FF	Ctrl L	12	00001100	0C
Carriage Return	CR	Ctrl M	13	00001101	0D
Shift Out	SO	Ctrl N	14	00001110	0E
Shift In	SI	Ctrl O	15	00001111	0F
Data Line Escape	DLE	Ctrl P	16	00010000	10
Device Control 1	DC1	Ctrl Q	17	00010001	11
Device Control 2	DC2	Ctrl R	18	00010010	12
Device Control 3	DC3	Ctrl S	19	00010011	13
Device Control 4	DC4	Ctrl T	20	00010100	14
Negative Acknowledge	NAK	Ctrl U	21	00010101	15
Synchronous Idle	SYN	Ctrl V	22	00010110	16
End of Transmit Block	ETB	Ctrl W	23	00010111	17
Cancel	CAN	Ctrl X	24	00011000	18
End of Medium	EM	Ctrl Y	25	00011001	19
Substitute	SUB	Ctrl Z	26	00011010	1A
Escape	ESC	Ctrl [	27	00011011	1B
File Separator	FS	Ctrl \	28	00011100	1C
Group Separator	GS	Ctrl]	29	00011101	1D
Record Separator	RS	Ctrl ^	30	00011110	1E
Unit Separator	US	Ctrl _	31	00011111	1F
Space			32	00100000	20
Exclamation Point	!	Shift 1	33	00100001	21
Double Quote	"	Shift ‘	34	00100010	22
Pound/Number Sign	#	Shift 3	35	00100011	23
Dollar Sign	$	Shift 4	36	00100100	24
Percent Sign	%	Shift 5	37	00100101	25
Ampersand	&	Shift 7	38	00100110	26
Single Quote	‘	‘	39	00100111	27
Left Parenthesis	(	Shift 9	40	00101000	28
Right Parenthesis	)	Shift 0	41	00101001	29
Asterisk	*	Shift 8	42	00101010	2A
Plus Sign	+	Shift =	43	00101011	2B
Comma	,	,	44	00101100	2C
Hyphen / Minus Sign	-	-	45	00101101	2D
Period	.	.	46	00101110	2E
Forward Slash	/	/	47	00101111	2F
Zero Digit	0	0	48	00110000	30
One Digit	1	1	49	00110001	31
Two Digit	2	2	50	00110010	32
Three Digit	3	3	51	00110011	33
Four Digit	4	4	52	00110100	34
Five Digit	5	5	53	00110101	35
Six Digit	6	6	54	00110110	36
Seven Digit	7	7	55	00110111	37
Eight Digit	8	8	56	00111000	38
Nine Digit	9	9	57	00111001	39
Colon	:	Shift ;	58	00111010	3A
Semicolon	;	;	59	00111011	3B
Less-Than Sign	<	Shift ,	60	00111100	3C
Equals Sign	=	=	61	00111101	3D
Greater-Than Sign	>	Shift .	62	00111110	3E
Question Mark	?	Shift /	63	00111111	3F
At Sign	@	Shift 2	64	01000000	40
Capital A	A	Shift A	65	01000001	41
Capital B	B	Shift B	66	01000010	42
Capital C	C	Shift C	67	01000011	43
Capital D	D	Shift D	68	01000100	44
Capital E	E	Shift E	69	01000101	45
Capital F	F	Shift F	70	01000110	46
Capital G	G	Shift G	71	01000111	47
Capital H	H	Shift H	72	01001000	48
Capital I	I	Shift I	73	01001001	49
Capital J	J	Shift J	74	01001010	4A
Capital K	K	Shift K	75	01001011	4B
Capital L	L	Shift L	76	01001100	4C
Capital M	M	Shift M	77	01001101	4D
Capital N	N	Shift N	78	01001110	4E
Capital O	O	Shift O	79	01001111	4F
Capital P	P	Shift P	80	01010000	50

Capital Q	Q	Shift Q	81	01010001	51
Capital R	R	Shift R	82	01010010	52
Capital S	S	Shift S	83	01010011	53
Capital T	T	Shift T	84	01010100	54
Capital U	U	Shift U	85	01010101	55
Capital V	V	Shift V	86	01010110	56
Capital W	W	Shift W	87	01010111	57
Capital X	X	Shift X	88	01011000	58
Capital Y	Y	Shift Y	89	01011001	59
Capital Z	Z	Shift Z	90	01011010	5A
Left Bracket	[	[	91	01011011	5B
Backward Slash	\	\	92	01011100	5C
Right Bracket	]	]	93	01011101	5D
Caret	^	Shift 6	94	01011110	5E
Underscore	_	Shift -	95	01011111	5F
Back Quote	`	`	96	01100000	60
Lower-case A	a	A	97	01100001	61
Lower-case B	b	B	98	01100010	62
Lower-case C	c	C	99	01100011	63
Lower-case D	d	D	100	01100100	64
Lower-case E	e	E	101	01100101	65
Lower-case F	f	F	102	01100110	66
Lower-case G	g	G	103	01100111	67
Lower-case H	h	H	104	01101000	68

Lower-case I	I	I	105	01101001	69
Lower-case J	j	J	106	01101010	6A
Lower-case K	k	K	107	01101011	6B
Lower-case L	l	L	108	01101100	6C
Lower-case M	m	M	109	01101101	6D
Lower-case N	n	N	110	01101110	6E
Lower-case O	o	O	111	01101111	6F
Lower-case P	p	P	112	01110000	70
Lower-case Q	q	Q	113	01110001	71
Lower-case R	r	R	114	01110010	72
Lower-case S	s	S	115	01110011	73
Lower-case T	t	T	116	01110100	74
Lower-case U	u	U	117	01110101	75
Lower-case V	v	V	118	01110110	76
Lower-case W	w	W	119	01110111	77
Lower-case X	x	X	120	01111000	78
Lower-case Y	y	Y	121	01111001	79
Lower-case Z	z	Z	122	01111010	7A
Left Brace	{	Shift [	123	01111011	7B
Vertical Bar	|	Shift \	124	01111100	7C
Right Brace	}	Shift]	125	01111101	7D
Tilde	~	Shift `	126	01111110	7E
Delta	Δ		127	01111111	7F

18. <u>מסך טקסט</u>

18.1. תא טקסט

18.1.1. קיימות 80 עמודות ו-25 שורות (סה"כ 2000 תאי תצוגה)

18.1.2. כל תא מורכב מ-2 בתים

18.1.2.1. הראשון – מגדיר את התו שיופיע בתא

18.1.2.2. השני – את פורמט התצוגה (צבע, רקע, הבהוב/הדגשה) של התו

משמעות ה bit setting	MSB				מספר ה bit			LSB
	7	6	5	4	3	2	1	0
תו רגיל (normal foreground)	0	X	X	X	X	X	X	X
תו מהבהב/מודגש (blinking/Bold)	1	X	X	X	X	X	X	X
צבע הרקע – **b**ackground	X	**b**	**b**	**b**	X	X	X	X
צבע התו - **f**oreground	X	X	X	X	X	**f**	**f**	**f**
הדגש התו - **B**old	X	X	X	X	**B**	X	X	X

18.2. לתחזוקת המסך נדרש זיכרון של 4KByte = 80 * 25 * 2

18.3. הכתובת האבסולוטית של תחילת המסך היא B800:0000h

18.4. הגדרת מסך TEXT

```
MOV AX, 03h
INT 10h
```

19. <u>מסך גרפי</u>

19.1. אזור זיכרון המסך מתחיל בכתובת 0A000:0000h

```
INT 10, 0 : set video mode (clear screen)
     AH = 0
     AL = 00: text 40*25 16 color (mono)
          01: text 40*25 16 color
          02: text 80*25 16 color (mono)
          03: text 80*25 16 color
          04: CGA 320*200 4 color
          05: CGA 320*200 4 color (m)
          06: CGA 640*200 2 color
          07: MDA monochrome text 80*25
          08: PCjr
          09: PCjr
          0A: PCjr
          0B: reserved
          0C: reserved
          0D: EGA 320*200 16 color
          0E: EGA 640*200 16 color
          0F: EGA 640*350 mono
          10: EGA 640*350 16 color
          11: VGA 640*480 16 color
          12: VGA 640*480 16 color
          13: VGA 320*200 256 color
     NB: add 80h to mode to prevent screen clr
```

19.2. תא צבע

19.2.1. סך הפיקסלים שניתן להגדיר לתצוגה היא כתלות בסוג הגדרת המסך (כמות הפיקסלים לרוחב ולגובה)

19.2.2. ב-256 צבעים, כל פיקסל מיוצג ע"י 1Byte

19.2.3. במסך מונו (שחור/לבן)

19.2.3.1. כל Byte מייצג 8 פיקסלים (כל סיבית מייצגת פיקסל – כבוי/דלוק)

19.2.3.2. שורה במסך מוגדרת ע"י 80 בתים

19.3. הדפסת טקסט למסך גרפי

19.3.1. המסך הגרפי מחולק לתאים בהם גודל כל תו מודפס לוקח 8*8 פיקסלים

19.3.2. ההדפסה נעשית לפי הפינה השמאלית העליונה המוגדרת בערכי DH (שורה) ו-DL (עמודה)

19.3.2.1. בתחילה ממקמים את סמן הטקסט

```
MOV DL, column
MOV DH, row
MOV AH, 2h
INT 10h
```

19.3.2.2. אחר כך מדפיסים את המחרוזת

```
MOV DX, offset String
MOV AH, 9h
INT 21h
```

19.4. שינוי ערך הפיקסל בנקודה מסוימת

19.4.1. הגישה נעשית למיקום place (16 סיביות) כאשר זיכרון המסך מסודר באופן טורי

19.4.2. עם שינוי ערך לצבע color (8 סיביות)

```
MOV DI, place
MOV byte ptr[DI], color  /  MOV byte Dptr[DI], color
```

20. פסיקות שונות

20.1. הגדרת מסך TEXT

```
MOV AX, 03h
INT 10h
```

20.2. הדפסת פיקסל למסך

20.2.1. DX מספק את מספר השורה

20.2.2. CX מספק את מספר העמודה

20.2.3. AL מספק את צבע הפיקסל

```
MOV AH, 0Ch
INT 10h
```

20.3. קריאת פיקסל מהמסך

20.3.1. DX מספק את מספר השורה

20.3.2. CX מספק את מספר העמודה

20.3.3. AL מקבל את צבע הפיקסל

```
MOV AH, 0Dh
INT 10h
```

20.4. מיקום סמן טקסט במסך גרפי

20.4.1. DH מספק את מספר השורה

20.4.2. DL מספק את מספר העמודה

20.4.3. BH=0, הכוונה לעמוד וידאו ראשון

20.4.4. הדפסת המחרוזת נעשית כמו במסך טקסט

```
MOV BH, 0h
MOV AH, 2h
INT 10h
```

20.5. סיום תוכנית

20.5.1. החזרת השליטה למערכת ההפעלה

```
MOV AH, 4Ch
INT 21h
```

20.6. קליטת תו בודד מלוח המקשים

20.6.1. ערך התו נשמר ב- AL

```
MOV AH, 1h
INT 21h
```

20.6.2. ואפשר גם

```
MOV AH, 8h
INT 21h
```

20.7. הדפסת תו בודד למסך

20.7.1. ערך התו נלקח מ- DL

```
MOV AH, 2h
INT 21h
```

20.8. הדפסת מחרוזת שלמה למסך

20.8.1. כתובת המחרוזת נמצאת ב- DX

20.8.2. יש לוודא שבסוף המחרוזת טעון הערך $, המציין את סוף מחרוזת

```
MOV AH, 9h
INT 21h
```

20.9. קליטת מחרוזת מהמקלדת

20.9.1. כתובת המחרוזת נמצאת ב- DX

20.9.2. קליטת התווים נעשית עד הקשת ENTER ע"י המשתמש

20.9.2.1. יש לשמור מספיק מקום במחרוזת לכל התווים

20.9.2.2. ערך הASCCI של ENTER הוא 13d=0Dh, שמשמעותו היא "עבור לשורה חדשה"

20.9.2.3. הערך 10d=0Ah, משמעותו – החזר את הסמן לתחילת השורה

20.9.3. ה-Byte הראשון במחרוזת מציין את מספר התווים המקסימאלי של המחרוזת

20.9.3.1. יש לבצע בדיקה שתמנע הקלדה הגדולה מגודל המחרוזת בכתיבת התכנית

20.9.3.2. אם הוקשו יותר תווים מהיכולת לקלוט, המחרוזת לא תתעדכן

20.9.4. ה-Byte השני במחרוזת מציין את מספר התווים שהוקלדו ונטענו לתוך המחרוזת

20.9.4.1. הפסיקה עצמה מנהלת את שינוי ערך כמות התווים שהוכנסו

```
MOV AH, 0ah
INT 21h
```

20.10. קבלת נתוני עכבר

20.10.1. DX מקבל את מספר השורה

20.10.2. CX מקבל את מספר העמודה

20.10.3. BX מקבל את מצב הלחצנים בעכבר

20.10.3.1. לחצן שמאלי – סיבית 0

20.10.3.2. לחצן ימני – סיבית 1

20.10.3.3. ערך 1 = לחיצה

```
INT 33h
```

20.11. התאמת רזולוציית המסך למיקום עכבר

20.11.1. התאמת הרוחב

```
MOV AX, 07h
MOV CX, 0h
MOV DX, screenWidth
INT 33h
```

20.11.2. התאמת הרוחב

```
MOV AX, 07h
MOV CX, 0h
MOV DX, screenHeight
INT 33h
```

20.12. הדפסת תו בודד למסך???

20.12.1. ערך התו נלקח מ- AL

```
MOV AH, 0eh
MOV BH, 0
INT 10h
```

20.13. שמירת ה-X בייטים "הבאים" בזיכרון

20.13.1. כמות Xbytes הבייטים מוגדרת ב- DX

```
MOV DX, Xbytes
INT 27h
```

21. הוראות קלט/פלט שונות

21.1. תו מקלדת נלחץ

21.1.1. סיבית ה-LSB מייצגת אם מקש במקלדת לחוץ

21.1.2. 1 = מקש לחוץ

```
IN AL, 64h
```

21.2. קורא את תו המקלדת שנלחץ

21.2.1. קוד התו שנלחץ נשמר ב-AL

21.2.2. לכל סוג מקלדת רשימת קודים משלה (פיזור מקשים שונה)

```
IN AL, 60h
```

21.3. פקודת סיום פסיקה ל-PIC

21.3.1. ה-20h הימני הוא מילת בקרה ל-PIC שמשמעותה End Of Interrupt = EOI

21.3.2. במידה והמתכנת "יוזם" פסיקה, עליו להודי ל-PIC שהפסיקה הסתיימה

21.3.3. כלומר – להעביר ל-0 את הסיבית ב-ISR שאפשרה לחומרה לתקשר עם המעבד

```
OUT 20h, 20h
```

22. פקודות חשובות

22.1 LOOP

22.1.1. נועד כדי להחזיר אותנו לביצוע חוזר של מקבץ שורות קוד

22.1.2. לדוגמא: LOOP loopStart

22.1.3. קופץ בקוד למיקום של התגית loopStart

22.1.4. CX יגדיר את כמות הפעמים לביצוע החוזר, כאשר יתאפס התכנית תמשיך לעבור על שאר הקוד

22.1.5. הוראת הLOOP מורכבת מ-4 בתים, 2 בתים לביצוע פקודת הקפיצה ו-2 בתים לגודל וכיוון הקפיצה. סה"כ גודל הקפיצה המותנית יהיה עד 8 סיביות (256 מקומות בזכרון, כלומר – עד 128 פקודות, כאשר כל פקודה בגודל 2 בתים). הקפיצה המותנית מתייחסת לדגלים CF, AF, OF כדי להחליט על גודל וכיוון הקפיצה

22.2 JMP - Jump

22.2.1. נועד כדי לדלג למיקום אחר בקוד

22.2.2. דוגמא: JMP jumpPoint

22.2.3. קופץ בקוד למיקום של התגית jumpPoint

22.2.4. ממשיך מנקודה זו לבצע את הפקודות

22.2.5. גודל הקפיצה מותנית ל128 בייט בזיכרון (מאותם שיקולים כמו בLOOP)

22.3 RET - CALL

22.3.1. נועד כדי לקרוא לרוטינה מסויימת, להפעיל אותה ולחזור חזרה לאותה נקודה לאחר ביצוע

22.3.2. דוגמא: CALL routine

22.3.3. מכניס למחסנית את מיקום עזיבת הקוד הראשי המוצבע ע"י IP

22.3.4. קופץ בקוד למיקום של התגית routine

22.3.5. ממשיך מנקודה זו לבצע את הפקודות

22.3.6. כאשר מגיע לפקודה RET – שולף מהמחסנית לתוך IP את מיקום עזיבת הקוד הראשי, וחוזר לשם

22.3.7. גודל הקפיצה אינו מוגבל וניתן להגיע לכל מקום בCS

22.4 CMP - Compare

22.4.1. פקודת השוואה בין 2 משתנים

22.4.2. דוגמא: CMP AX, BX

22.4.3. בפקודה מתבצעת פעולת AX=AX – BX, כלומר האופרנד השמאלי משתנה לאחר הפעולה

22.4.4. במידה והאופרנד השמאלי מתאפס, ZF משנה ערכו ל-1

22.5 CLD – Clear Direction

22.5.1. פקודה לשינוי ערך DF להיות 0

22.5.2. במצב זה, פעולות כגון הדפסת מחרוזת, האינקדס יקודם במילה בכל איטרציה

22.6 STD – Set Direction

22.6.1. פקודה לשינוי ערך DF להיות 0

22.6.2. במצב זה, פעולות כגון הדפסת מחרוזת, האינקדס יקודם במילה בכל איטרציה

22.7 IN

22.7.1. קריאה מהתקן קלט

22.7.2. יש לציין את אחד מ-256 כתובת קלט/פלט

22.7.3. דוגמא: IN AX, 63h (במקום ערך ניתן גם להשתמש באוגר)

22.8 OUT

22.8.1. כתיבה להתקן פלט

22.8.2. יש לציין את אחד מ-256 כתובת קלט/פלט

22.8.3. דוגמא: OUT 61h, AX (במקום ערך ניתן גם להשתמש באוגר)

22.9 LEA - Load Effective Address

22.9.1. טוען לאוגר שבחרנו את כתובת המשתנה שבחרנו

22.9.2. אם נטען ל-DX אז הגישה תהיה דרך: DS:DX

22.10 MOV DX, offset Var

22.10.1. טוען ל-DX את כתובת המשתנה Var

22.11 CLI – Clear I

22.11.1. IF=0

22.11.2. המעבד לא יקבל בקשות לפסיקה

22.12 STI – Set I

22.12.1. IF=1

22.12.2. המעבד יכול לקבל בקשות לפסיקה

22.13. העתקת מחרוזות

22.13.1. לפי המיקום, מתבצעת העתקה של תו מהמקור אל היעד וקידום של המצביעים

22.13.2. ע"י הגדרת CX ומתן הוראת REP לפני הפקודה, ניתן לבצע לולאה שתעתיק כמות תווים רצויה מהמחרוזת

פקודות להעתקת מחרוזות:

סוג הפעולה	הוראה	העתקה של	מ..	ל..	קידום
העתקה מזיכרון לאוגר Load String	LODSB	byte	DS:[SI]	AL	אם DF==0 ← SI+=1 אם DF==1 ← SI-=1
	LODSW	word	DS:[SI]	AX	אם DF==0 ← SI+=2 אם DF==1 ← SI-=2
העתקה מאוגר לזיכרון Store String	STOSB	byte	AL	ES:[DI]	אם DF==0 ← DI+=1 אם DF==1 ← DI-=1
* אתחול ES	STOSW	word	AX	ES:[DI]	אם DF==0 ← DI+=2 אם DF==1 ← DI-=2
העתקה מזיכרון לזיכרון Move String	MOVSB	byte	DS:[SI]	ES:[DI]	אם DF==0 ← DI+=1 SI+=1 אם DF==1 ← DI-=1 SI-=1
* אתחול ES	MOVSW	word	DS:[SI]	ES:[DI]	אם DF==0 ← DI+=2 SI+=2 אם DF==1 ← DI-=2 SI-=2
קידומת לביצוע חזרה על העתקה Repeat	REP xxx xxx – הוראת העתקה	חזרה על פעולת העתקה CX פעמים (עד ש CX מתאפס)			CX-=1

*יש לאתחל את ES שיצביע על סגמנט הנתונים

22.14. השוואת מחרוזות

22.14.1. לפי המיקום, מתבצעת העתקה של תו מהמקור אל היעד וקידום של המצביעים

סוג הפעולה	הוראה	השוואה של	הנמצא ב..	ל..	קידום
השוואה Compare String	CMPSB	Byte	DS:[SI]	ES:[DI]	אם DF==0 ← DI+=1 SI+=1 אם DF==1 ← DI-=1 SI-=1
	CMPSW	word	DS:[SI]	ES:[DI]	אם DF==0 ← DI+=2 SI+=2 אם DF==1 ← DI-=2 SI-=2
חיפוש Scan String	SCASB	Byte	AL	ES:[DI]	אם DF==0 ← DI+=1 אם DF==1 ← DI-=1
	SCASW	word	AX	ES:[DI]	אם DF==0 ← DI+=2 אם DF==1 ← DI-=2
קידומת לחזרה על השוואה כל עוד האיברים שווים Repeat while equal	REPE xxx xxx – הוראת השוואה	יש להכניס ל CX את מספר ה bytes/words שרוצים להשוות. הבדיקה נמשכת כל עוד CX!=0 וגם (zero flag) ZF== 1			CX=CX-1
קידומת לחזרה על השוואה כל עוד האיברים שונים Repeat while not equal	REPNE xxx xxx – הוראת השוואה	הבדיקה נמשכת כל עוד CX!=0 וגם (zero flag) ZF== 0			CX=CX-1

23. תכנית ה-DEBUG

23.1. תכנית ה-DEBUG הכלולה ב-MS-DOS היא תוכנית מוניטור המאפשרת להקליד פקודות אסמבלר של ה-8086 ישירות לזיכרון ולהריצן החל מכתובת מסויימת בזיכרון (ועד ל...) עם אפשרויות ניפוי בסיסיות.

23.2. הפקודות העיקריות הן :

23.2.1. R – קריאת ערכי האוגרים

23.2.2. D - הצגת מרחב כתובות בזיכרון

23.2.3. E – הקלדת ערכי HEX ישירות לזיכרון

23.2.4. T - trace הרצה בצעדים של תכניות/פקודות עם הצגת מצב אוגרי המעבד בכל צעד

23.2.5. G - הרצה ברצף רגילה של תכניות

23.2.6. A - כתיבה בשפת אסמבלר (ברמת פקודה בלבד ללא תוויות שמיות)

23.2.7. U - תרגום קוד מכונה לשפת אסמבלר

23.2.8. ? - אינדקס לכל הפקודות האפשריות ב-DEBUG

23.3. הפעלת תוכנת ה- DEBUG נשעית באמצעות הרצת הפקודה DEBUG ב-RUN (הפעלה) של Window (גישה מתפריט START) .