# הגדרות משתנים ופקודת mov

### שיעור 3- מערכים

שפת אסמבלי

* אסמבלי היא שפת תכנה שעובדת בצורה הקרובה ביותר לחומרת המחשב. לא ניתן לכתוב תכנה בלי להבין את ארגון המחשב והחומרה.
* נלמד על ארגון המחשב באמצעות מעבד 80x86 תוצרת אינטל. המעבד הראשון יוצר לראשונה בשנת 1978.
* מעבד 80x86 מאורגן על פי ארכיטקטורת פון נוימן ומהווה בסיסי למעבדים הקיימים. כל משפחת המעבדים של אינטל שומרת תאימות לאחור עם מעבדי 80x86.

### ג'ון פון נוימן

התכנון הבסיסי של מחשב נקרא ארכיטקטורה.

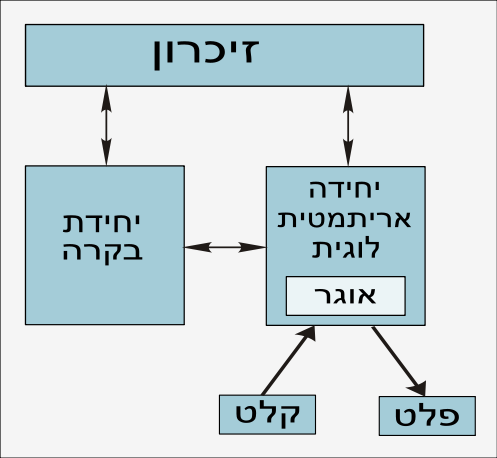
גון פון נוימן היה מתמטיקאי שפיתח את הארכיטקטורה של רוב המחשבים שאנו משתמשים בהם היום.

*ג'ון לואיס פון נוימן - John von Neumann (1903 – 1957)*

*היה מתמטיקאי הונגרי-אמריקאי ממוצא יהודי שהמיר דתו לנצרות קתולית.*

*הוא היה שותף לשניים מההישגים הטכנולוגיים הבולטים של המאה העשרים: פיתוח פצצת האטום ופיתוח המחשב האלקטרוני.*

*זכור גם כיוצרה של תורת המשחקים (יחד עם אוסקר מורגנשטרן), הרים תרומה משמעותית לחקר מכניקת הקוונטים, תורת הקבוצות וענפי מתמטיקה נוספים ושילב בהצלחה רבה פעילות במחקר טהור ובמחקר שימושי, בענפי מדע רבים.*



## ארכיטקטורת פון ניומן

ארכיטקטורת פון נוימן

כוללת שלוש אבני בניין מרכזיות:

יחידת העיבוד המרכזית: Central Processing Unit – CPU

זיכרון: Memory

ויחידות קלט/פלט: Input / Output.

* במעבדים מבוססי ארכיטקטורת VNA, יחידת העיבוד המרכזית מבצעת את כל החישובים. מידע והוראות למעבד (קוד התוכנית) נמצאים בזיכרון עד שהם נדרשים על־ידי המעבד.
* מבחינת המעבד, יחידות הקלט / פלט נראות כמו זיכרון, כיוון שהמעבד יכול לשלוח אליהן מידע ולקרוא מהן מידע.

יחידת העיבוד המרכזית של המעבד מתקשרת עם הזיכרון ועם יחידות הקלט / פלט באמצעות קווי תקשורת שנקראים **Bus**, או בעברית **פסים**. יש סוגים שונים של קווי תקשורת, שלכל אחד יש תפקיד ייחודי.

### פסי המערכת

פסי המערכת: System Buses מחברים את הרכיבים השונים של מכונות

פס הוא למעשה אוסף של קווי חשמל שמעבירים אותות בין הרכיבים השונים.

האותות המועברים הם רמות מתח חשמלי שונות – יש מתח שמייצג 0 ויש מתח שמייצג 1

הוראה בשפת סף: mov ax, 0Ah

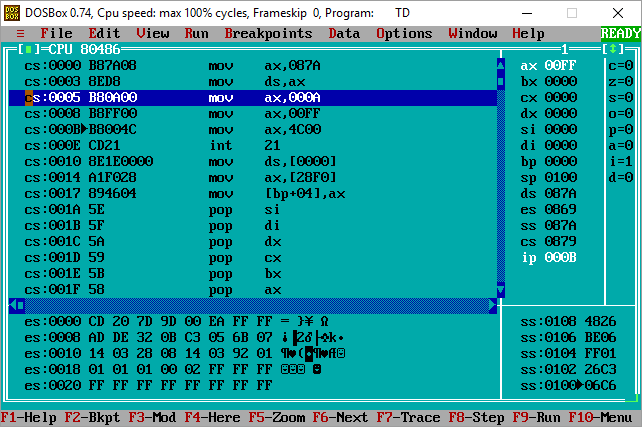
מתורגמת להוראה בשפת מכונה בה קוד הפעולה הוא: B80A00

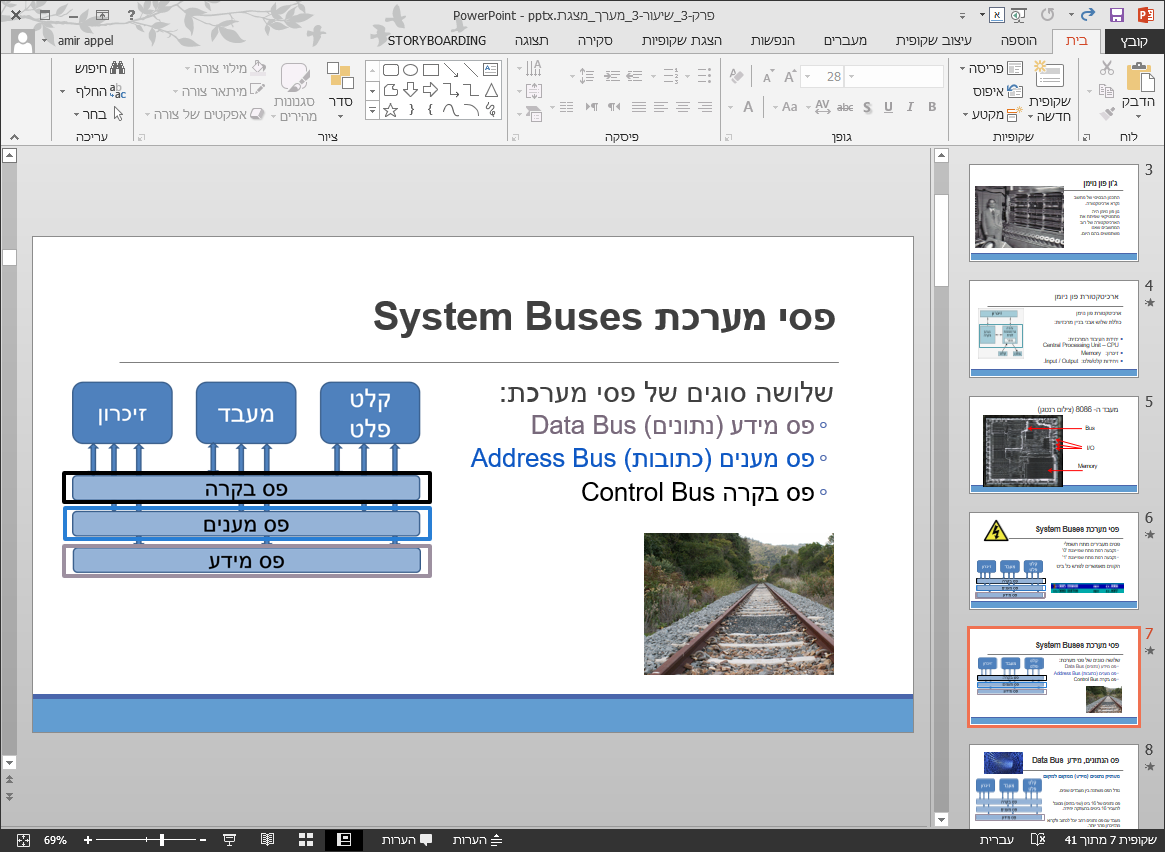
1011 1000 0000 1010 0000 0000

והוראה בשפת: mov ax, 0FFh

מתורגמת להוראה בשפת מכונה בה קוד הפעולה הוא: B8FF00

1011 10001 1111 1111 0000 00000





* שלושה סוגים של פסי מערכת:
  + פס מידע (נתונים) Data Bus
  + פס מענים (כתובות) Address Bus
  + פס בקרה Control Bus

DATA BUS – פס נתונים

פס הנתונים משמש להעברת נתונים בין המרכיבים השונים של המחשב. גודל פס הנתונים משתנה בין משפחות שונות של מעבדים. במחשבים סטנדרטיים פס הנתונים מכיל 16, 32 או 64 קווים .

מחשב שיש לו פס נתונים של 16 קווים מסוגל להעביר 16 ביטים בבת אחת. [216 = 65,536]

כיוון שכל פניה לזיכרון לוקחת זמן, מעבד שיש לו פס נתונים רחב יותר יוכל לכתוב ולקרוא מהזיכרון בקצב מהיר יותר ולכן יעבוד מהר יותר ממעבד זהה שיש לו פס נתונים עם פחות קווים.

ADDRESS BUS – פס מענים

פס הנתונים מעביר מידע בין אזור מוגדר בזיכרון מיחידת קלט פלט לבין המעבד.

כדי להפריד בין מקומות שונים בזיכרון, לכל בית בזיכרון יש כתובת נפרדת

כאשר התוכנה רוצה לפנות למקום כלשהו בזיכרון, היא מכניסה את כתובת הזיכרון המבוקש לתוך פס המענים.

רכיבים אלקטרוניים ששולטים על הזיכרון מזהים את הכתובת שבפס המענים ודואגים לשלוח למעבד את המידע בכתובת המבוקשת, או לכתוב לזיכרון בכתובת הנ"ל את המידע ששלח המעבד.

כמות הקווים בפס המענים קובעת את גודל מרחב הכתובות שהמעבד יכול לפנות אליו.

לדוגמה, מעבד שיש לו שני קווים בפס המענים **22** יוכל לפנות לארבע כתובות בלבד: 11,01,10,00.

מעבד שיש לו n קווים בפס המענים, יוכל לפנות **2n** כתובות שונות.

למעבד ה־8086, יש 20 קווים בפס המענים. כלומר מרחב הכתובות שלו הוא **220**= 1,048,576.

מעבדים מתקדמים כוללים פס מענים של 32 ביט (4 בתים) , כלומר הם מוגבלים **232** ב־4,294,976,296 בתים – ארבעה ג'יגה בייט.

בשלב מסוים, גם ארבעה גי'גה של מרחב כתובות זיכרון הפכו להיות מגבלה, וכיום מעבדים ומערכות הפעלה כגון Windows 7 תומכים במרחב כתובות בגודל 64 ביט **264** (8 בתים).

**למעבד ה-8086 שני פסי מענים:**

**לזיכרון**: פס מענים ברוחב 20 ביט

**220** = 1,048,576 כתובות

**לקלט פלט:** פס מענים ברוחב 16 ביט

**216** = 65,536 כתובות

CONTROL BUS – פס בקרה

פס הבקרה מכיל קווים חשמליים שתפקידם לעשות סדר בדרך שבה המעבד מתקשר עם יתר הרכיבים.

הסברנו את המנגנון שמאפשר גישה של המעבד לזיכרון באמצעות פס המענים, אך לא הסברנו איך יודעים האם המעבד מבקש לכתוב לזיכרון, או שמא לקרוא מהזיכרון ?

* **שמנו ערכים בפסי המענים והמידע**

אם קו ה־read מכיל אפס, המעבד קורא מהזיכרון.

אם קו ה־write מכיל אפס, המעבד כותב לזיכרון.

* + האם לפנות לזיכרון או ל-I/O?
  + האם לכתוב או לקרוא?
* **פס הבקרה עונה על שאלות אלו**
  + לאן לפנות: קו לזיכרון, קו ל-I/O
  + כיוון ההעתקה: קו קריאה read, קו כתיבה write

Clock Cycle – שליחת הוראה

בכל cycle של השעון נשלחת הוראה על 3 הפסים:

* מאיזו יחידה לקרוא/לכתוב - בקרה
* איזה ערך - מידע
* ולאיזו כתובת - מענים

פעולת המעבד היא מתוזמנת:

The clock frequency was originally limited to 5 MHz, but the last versions were specified for 10 MHz

הרץ (סימון: Hz) הוא יחידת מידה לתדירות במערכת היחידות הבינלאומית. היא נקראת על שמו של הפיזיקאי הגרמני היינריך רודולף הרץ אשר תרם רבות למדע בתחום האלקטרומגנטיות.

הגדרת ההרץ היא Hz = s−1 שמשמעותה "אירוע אחד לשנייה";

100 הרץ משמעותו "100 אירועים לשנייה",

ניתן ליישם את היחידה על כל אירוע מחזורי, למשל: שעון מתקתק ב-1 הרץ.

1 מגה - הרץ (סמל  **MHz**‏) = מיליון פעמים בשנייה

הזיכרון

כמות הכתובות בזיכרון מכונה "גודל הזיכרון"

כל כתובת = **בית אחד**

פס כתובות בגודל n ביטים מאפשר לפנות לזיכרון בגודל **2n**בתים

למעבד ה-8086 זיכרון בגודל 1,048,576 בתים **220 ⇦ MB1**

ניתן לחשוב על הזיכרון כמערך [0..1,048,575]

2^{64}-1 = 18,446,744,073,709,551,615

**28** **=** [**256**](https://he.wikipedia.org/w/index.php?title=256_(%D7%9E%D7%A1%D7%A4%D7%A8)&action=edit&redlink=1)

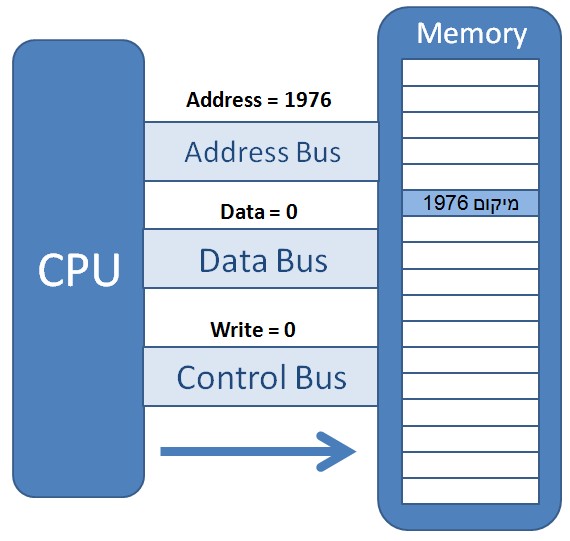
תוספת האופסט מגדילה את הזיכרון בעוד חזקה של 2 עוד 65,536 תאים

או כאשר האופסט הוא 4 ביטים פי 16

**216** **=** [**65,536**](https://he.wikipedia.org/w/index.php?title=65536_(%D7%9E%D7%A1%D7%A4%D7%A8)&action=edit&redlink=1)

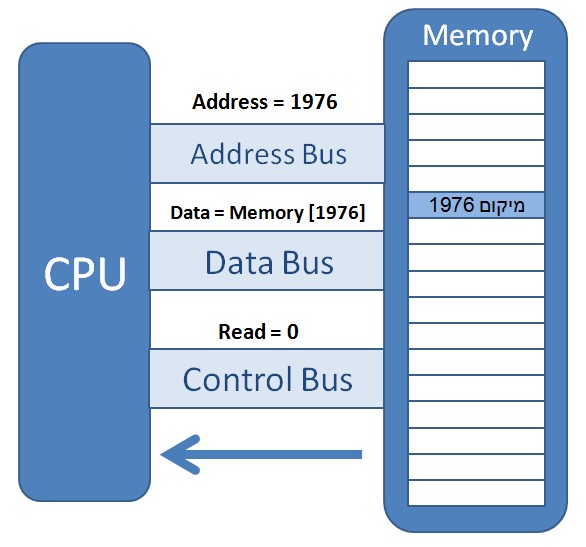
217 = 131,072

**220** **=** **1,048,576**

דוגמה - העתקת הערך '0' לזיכרון

לדוגמה ,כדי להציב במקום ה־1976 במערך את הערך" 0", מתבצעות הפעולות הבאות:

1. המעבד שם את הערך " 0" בפס **המידע**.
2. המעבד שם את הכתובת 1976 בפס **המיעון**.
3. המעבד משנה את קו ה־write בפס הבקרה וקובע את ערכו ל-0 ⇦ הקו "פעיל"



דוגמה- קריאה מהזיכרון

כדי לקרוא את מה שנמצא במקום ה־1976 בזיכרון, מתבצעות הפעולות הבאות:

1. המעבד שם את הכתובת 1976 בפס **המיעון**.
2. המעבד משנה את קו ה - **read** בפס הבקרה וקובע את ערכו ל 0 ⇦ הקו "פעיל"
3. המעבד **קורא** את הערך שבפס המידע.

זיכרון המחשב

זיכרון המחשב הוא המערך של תאים אשר לכל תא יש מספר המסמל את מיקומו.

הכתובת הראשונה בזיכרון היא 00000h והאחרונה FFFFh (216 = 65,536 כתובות)

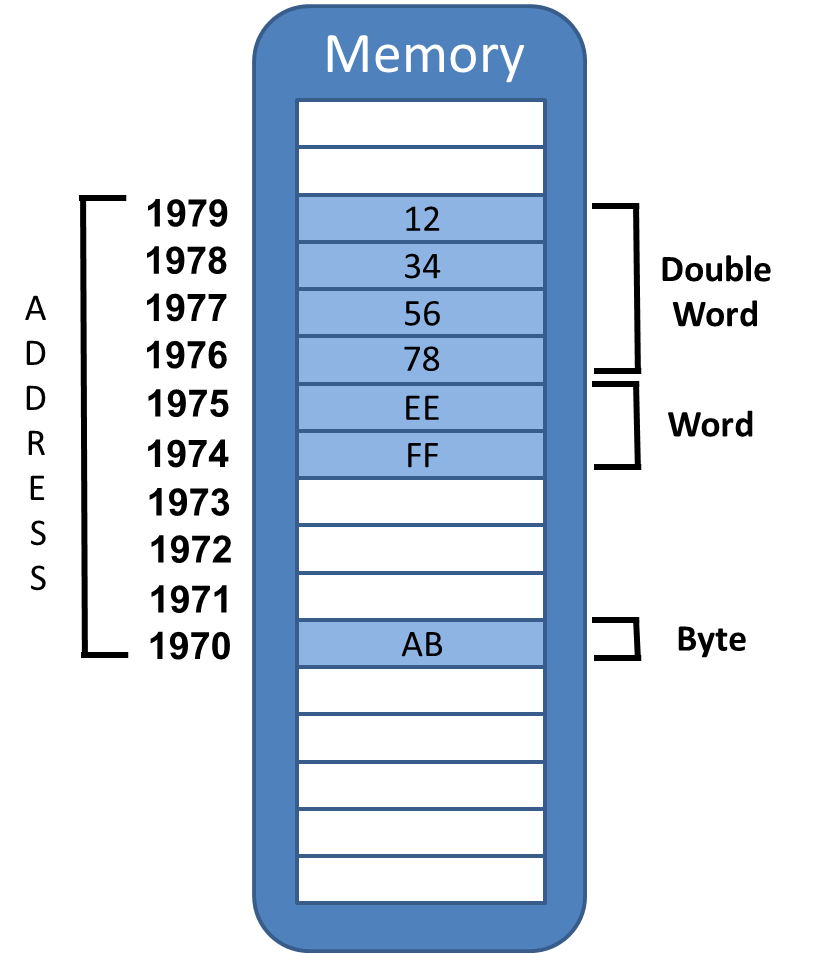
גודל כתובת בזיכרון הוא בית אחד.

כיצד ייכתב מידע בזיכרון שהוא בגודל של מילה word = 2 בתים (2 byte).

Low order byte ייכתב בכתובת הנמוכה יותר

High order byte ייכתב בכתובת הגבוהה יותר.

דוגמה- העתקת ערכים לזיכרון



לתא שבכתובת 1970 העתקנו את הערך **0ABh**

לתא שבכתובת 1974 העתקנו את הערך **0EEFFh**

נתחיל לכתוב את ה – Low order word - FF - 1974

נמשיך לכתוב את ה – High order word – EE - 1975

לתא שבכתובת 1976 העתקנו את הערך **12345678h**

Low order Double word

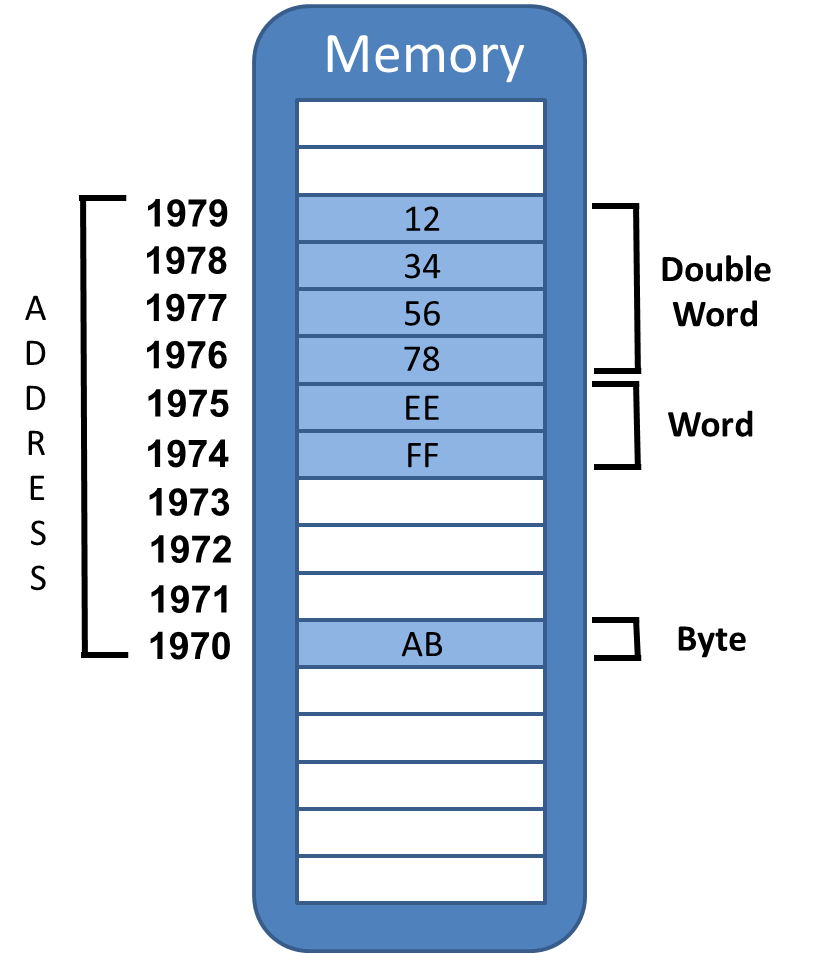
נתחיל לכתוב את ה – Low order word - 78 - 1976

נמשיך לכתוב את ה – High order word – 56 - 1977

High order Double word

נתחיל לכתוב את ה – Low order word - 34 - 1978

נמשיך לכתוב את ה – High order word – 12 - 1979



דוגמה- קריאה מהזיכרון

קריאת byte מכתובת 1974: 0FFh

קריאת word מכתובת 1975: 78EEh

נתחיל את הקריאה מתא 1975 Low order word,

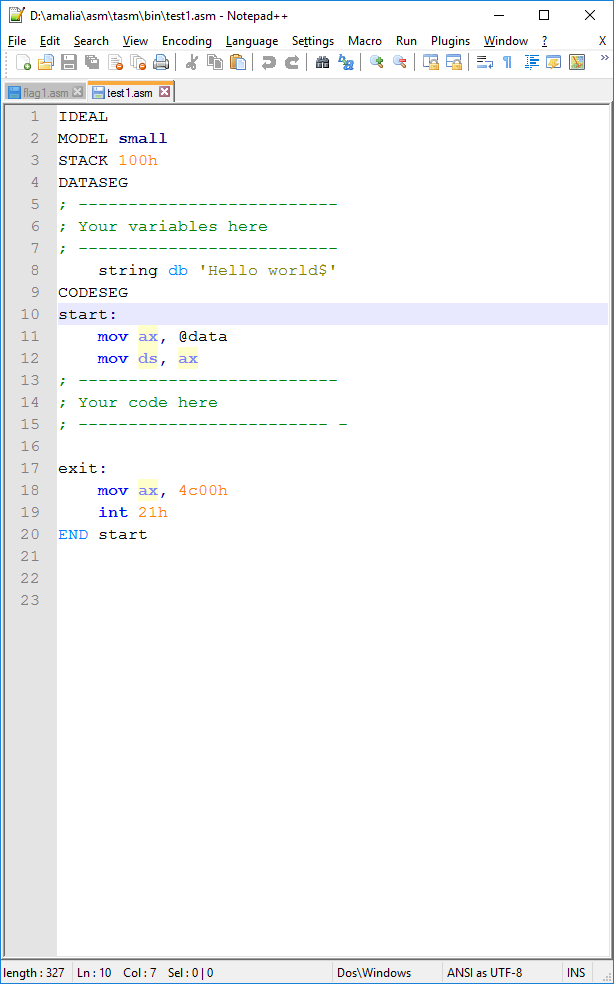
נמשיך בכתובת 1976 High order word

78EEh

אפשרי לקרוא word מכתובת 1970?

כן! י תקבל 0ABh ועוד בית עם "זבל" שנמצא בכתובת 1971

סגמנטים – Segments

זיכרון המחשב מחולק למקטעים.

code segment - הוא מקטע המכיל העתק של אוסף הפקודות של התוכנית

data segment – הוא מקטע השומר את הנתונים

בתחילת כל תכנית כותבים את הפקודה של העברת מיקום תחילת מקטע הנתונים לרגיסטר ds.

מכתובת זו המעבד יכתוב נתונים לזיכרון

במעבדי אינטל, הפנייה לתא בזיכרון נעשית בשיטה הנקראת מיעון מקטעים  
segmented) (addressing.

בשיטה זו, מרחב הזיכרון הליניארי מחולק בצורה לוגית למקטעים (segments),

מקטע בגודל ארבע בתים (28\*28\*28\*28 ) ⇦ 232 ⇦ 4,294,967,296

כל מקטע בזיכרון מתחיל בכתובת פיזית שהיא כפולה של 16 בתים, כתובת כזו נקראת פיסקה (paragraph). כל מקטע כולל מרחב כתובות של 16bit (מ-0 עד 65,535 או FFFFh)

כלומר, אפשר להתחיל מקטע מן הכתובת 16×0 שהיא הכתובת - 00000h ,

או מן הכתובת 16×1 שהיא הכתובת 00010h ,

או מן הכתובת 16×2 שהיא הכתובת 00020h , וכך הלאה.

בכתובות אלה הספרה הפחות משמעותית היא תמיד 0. לכתובת של תחילת מקטע קוראים גם כתובת הבסיס.

כלומר, כדי לציין כתובת של תא במקטע מסוים, מספיקות 16 סיביות.

לכתובת זו קוראים כתובת יחסית ( Relative address ),

מפני שהיא יחסית לכתובת ההתחלה של המקטע והיא מציינת את המרחק ממנה (ולכן היא תמיד חיובית).

כל מקטע כולל מרחב כתובות של 16bit (מ-0 עד 65,535 או FFFFh)

לכתובת הזו יש שמות נוספים:

כתובת אפקטיבית ( Effective addres ),

כתובת לוגית ( Logical address )

והיסט ( Offset ) שם המצביע על כך שהכתובת מבטאת מרחק בבתים מתחילת המקטע.

המעבד 8086 מקצה לתכנית ארבעה מקטעים (בגודל עד 64KB ) ולכל אחד מהמקטעים יש שם וייעוד שונה.

**מקטע הקוד** ( Code Segment ) – בו נשמרת התכנית שיש להריץ. בדרך-כלל זהו אזור רציף

של כתובות המכילות את הוראות התכנית בשפת מכונה; לאזור הזה המעבד פונה בשלב

ההבאה של מחזור ההבאה-ביצוע, כדי לקרוא את ההוראה שצריך לבצע.

**מקטע הנתונים** ( Data Segment ) – באזור הזה מאוחסנים המשתנים. המעבד פונה לאזור

הזה בשלב הביצוע של מחזור ההבאה-ביצוע לצורך קריאה וכתיבה של נתונים שיש לעבד.

**מקטע המחסנית** ( Stack Segment ) משמש לניהול פרוצדורות (שגרות) וכאמצעי זיכרון

לאחסון נתונים.

**המקטע הנוסף** ( External Segment ) הוא מקטע נתונים נוסף, המשמש בדרך-כלל לאחסון

של נתונים המשותפים לכמה משימות.

כתובת הבסיס של כל מקטע, מאוחסנת באחד מארבעת אוגרי המקטעים: CS, DS, SS

ו- ES .

כאשר אנו רוצים להריץ את התכנית, מערכת ההפעלה מקצה את המקטעים הדרושים

ובהתאם מאותחלים גם אוגרי המקטע בכתובות הבסיס המתאימות.

השיטה שנבחרה היא לחלק את הזיכרון למקטעים קטנים יותר – סגמנטים - segments.

כל כתובת בזיכרון ניתנת לביטוי על-ידי מספר ה-segment שלה, וההיסט - offset מתחילת הסגמנט.

הצורה המקובלת לרישום של כתובת בזיכרון היא: Segment:offset

במשפחת80x86, פניה לזיכרון מתבצעת על ידי שילוב של שני רגיסטרים בני 16 ביטים: הרגיסטר הראשון מחזיק את מיקום תחילת הסגמנט בזיכרון. הרגיסטר השני מחזיק את האופסט של הזיכרון מתחילת הסגמנט.

הגודל המקסימלי של האופסט קובע את הגודל המקסימלי של הסגמנט.

במעבדים בהם אנו נדון, בעלי אופסט של 16 ביטים ,גודל של סגמנט לא יכול להיות יותר מאשר 64K, 216 כתובות, כאשר הכתובות בכל סגמנט הן בין 0000h ל -0FFFFh (מ-0 עד 65,535).

כדי להגיע לכתובת של מיקום תחילת סגמנט, כופלים את הסגמנט ב־16.

כתוצאה מכך יוצא שסגמנטים תמיד מתחילים בכתובת שהיא כפולה של 16 בתים.

לדוגמה:

סגמנט מספר 0002h מתחיל 2\*16 = 32 בתים לאחר תחילת הזיכרון.

סגמנט מספר 0011h מתחיל 272 בתים לאחר תחילת הזיכרון. 1116 1016 = 11016 (17\*16=272)

כדי להגיע לכתובת כלשהי בזיכרון, מוסיפים לכתובת תחילת הסגמנט את האופסט.

לדוגמה, נניח שהסגמנט הוא 3DD6h והאופסט הוא 12h.   
הכתובת בזיכרון תירשם כך – 3DD6h:0012h

הכתובת בזיכרון תחושב כך: 3DD60h + 0012h = 3DD72h

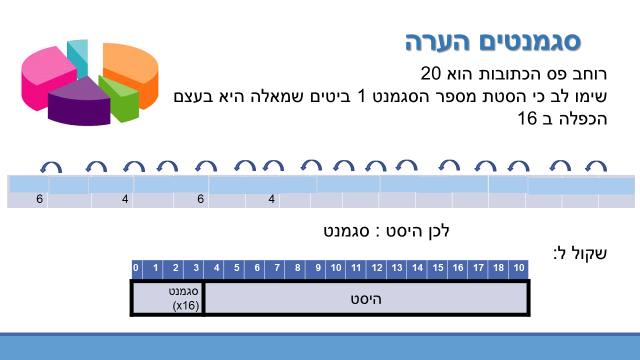
שימו לב לכך שתוספת ה - 0 מצד ימין באבר הראשון נועדה לכפול את כתובת הסגמנט ב־16, כדי להגיע למיקום תחילת הסגמנט.



087Ah ⇨ 2,170d

segment: 087Ah offset:00 ⇨ מיקום תחילת הנתונים

[01] ⇨ 087Ah + 1 = 087Bh , [16] ⇨ 088Ah = 087Ah + 10h (2,186d)



### משתנים

משתנה הוא תא זיכרון או יותר אשר ניתן לשמור בו נתונים.

כמו [קופסה] לשמירת מידע.

כדי להקל על הפנייה לתא הזיכרון אנו נותנים לו שם.

לדוגמה אם הכתובת ds:00 תיקרא age, ניתן יהיה לכתוב:

mov al, [age]

[סוגריים מרובעים] מציינים את הערך שנמצא במיקום בזיכרון המחשב

במקרה זה age ⇨ ds:00

[age] מצביע ⇦ על הערך שנמצא בזיכרון במיקום [00] בסגמנט הנתונים

כיצד נשמרים המשתנים בזיכרון?

אנו מצהירים בתחילת התכנית על המשתנים בהם נשתמש בתכנית כדי להקצות להם מקום ב – data segment.

הקצאת הכתובות היא לפי בתים עוקבים. בית למשנה מסוג Byte, שני בתים למשתנה מטיפוס word וארבע בתים למשתנה מטיפוס double word.

במעבדים של אינטל, מוקצים כמה בתים עוקבים למשתנים מטיפוס בית, שני בתים עוקבים למשתנה מטיפוס מילה, ולטיפוס מילה כפולה מוקצים ארבעה בתים עוקבים.

אחסון המשתנים ב – data segment מתבצע בשיטה הנקראת little endian,   
בשיטה זו הבית הפחות משמעותי מאוחסן בכתובת הנמוכה והבית המשמעותי בכתובת הגבוה. (מי שמופיע ראשון הוא הקצה הקטן – אינדיאני קטן).

אחסון המשתנים ב – data segment מתבצע בשיטה הנקראת little-endian . לפי שיטה זו, הבית הפחות משמעותי מאוחסן בכתובת הנמוכה, והבית המשמעותי של המילה מאוחסן בכתובת העוקבת.

השם little-endian , דהיינו: מי שמופיע ראשון הוא "הקצה הקטן"  
השם נבחר בהומור, שכן הוא מזכיר "אינדיאני קטן" – little Indian.

בציון כתובת של משתנה, אנו רושמים רק את כתובת הבית התחתון

מחרוזת

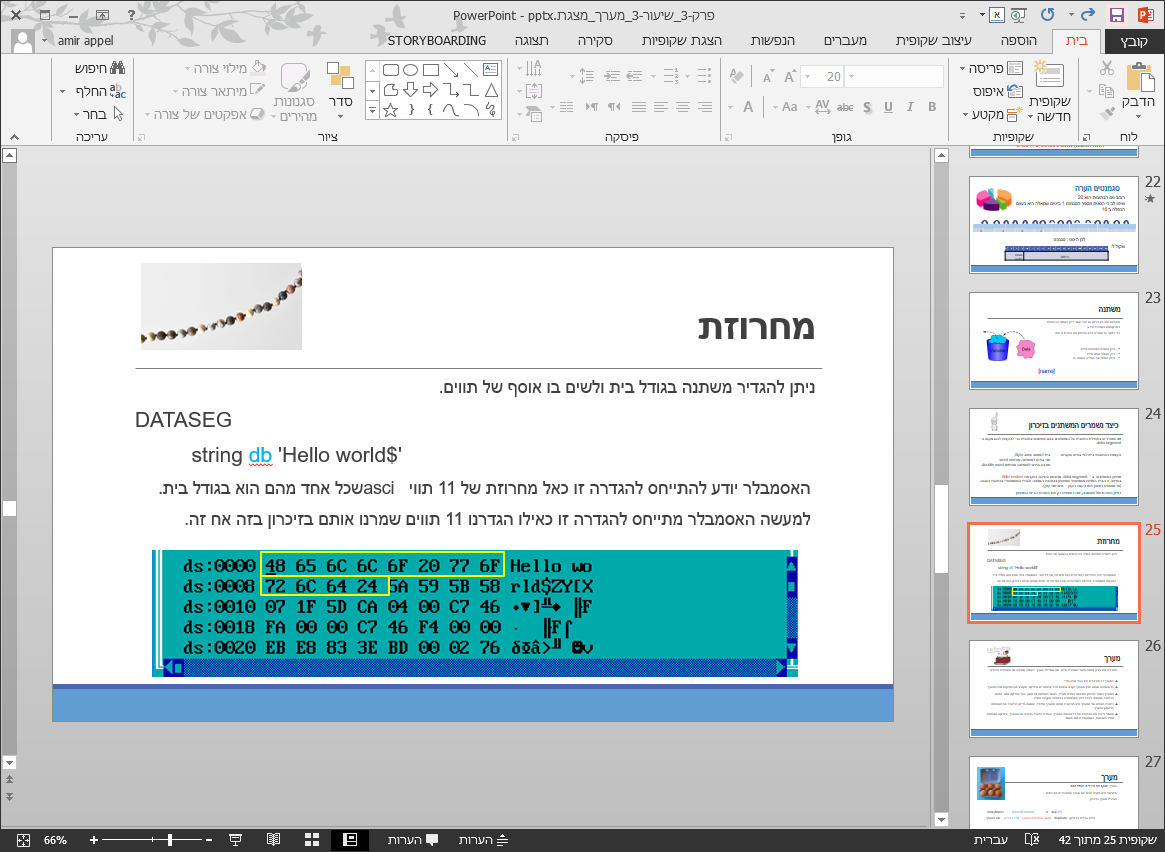
ניתן להגדיר משתנה בגודל בית ולשים בו אוסף של תווים.

DATASEG

string db 'Hello world'

האסמבלר יודע להתייחס להגדרה זו כאל מחרוזת של 11 תווי asci שכל אחד מהם הוא בגודל בית.

למעשה האסמבלר מתייחס להגדרה זו כאילו הגדרנו 11 תווים שמרנו אותם בזיכרון בזה אחר זה.



מערך

מערכים הם צורה נפוצה מאוד לשמירת מידע. מה שמייחד מערך לעומת שמירה של משתנים בזיכרון:

* במערך כל האיברים הם בעלי אותו גודל.
* כל משתנה שהוא חלק ממערך נקרא אלמנט ולכל אלמנט יש אינדקס, שקובע מה המיקום שלו במערך.
* המערך נשמר בזיכרון המחשב בצורה טורית, כאשר האלמנט הראשון, בעל אינדקס אפס, נמצא בכתובת הנמוכה ביותר ויתר האלמנטים בכתובות עוקבות אחריו .
* כתובת הבסיס של המערך היא הכתובת ממנה המערך מתחיל, ששווה בדיוק לכתובת של האלמנט הראשון במערך.
* אפשר לדעת מה הכתובת של כל אלמנט במערך, בעזרת כתובת הבסיס של המערך, אינדקס האלמנט וגודל האלמנט, באמצעות חישוב פשוט:

מערך

מערך: אוסף של איברים בגודל זהה.

מחרוזת היא מקרה פרטי של מערך שהאיברים הם תווים.

**הגדרת מערך בזיכרון:**

ArrayName SizeOfElement N dup (?)

(ללא ערכים בזיכרון) duplicate מספר אלמנטים במערך גודל בזיכרון שם המערך

הצהרה על מערכים

evenNumbers db 2, 4, 6, 8, 10

מערך של 5 מספרים זוגיים.

oddNumbers db 1, 3, 5, 7, 9, 11

מערך של 6 מספרים אי זוגיים.

points db 4 dup (0)

מערך ריק של 4 מספרים (שטרם קיבלו ערך).

letters db 7 dup ('A')

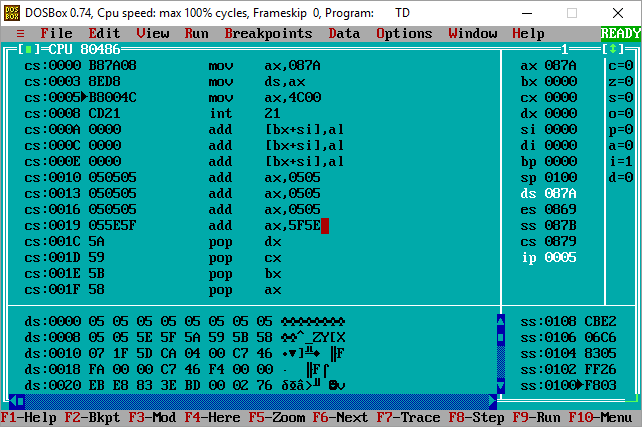
מערך של 7 אותיות 'A'.

שכפול ערכים במערך

dup הוא קיצור של המילה duplicate שכפול.

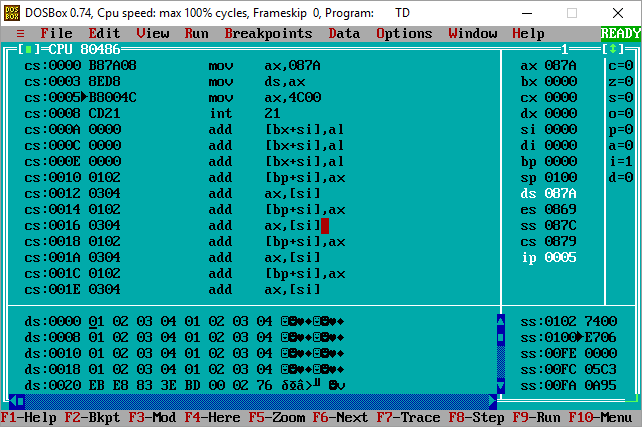
ArrayOfTenFives db 10 dup (5)

ייצור מערך של 10 איברים, כל ערך בגודל בית, ערכו של כל בית 5.



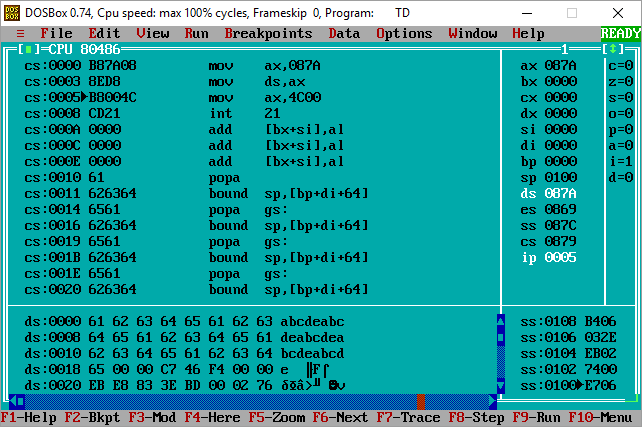
ArrayOf1234 db 8 dup (1, 2, 3, 4)

ייצור מערך של 32 איברים (4\*8), כל ערך בגודל בית את המספרים (4 3 2 1)

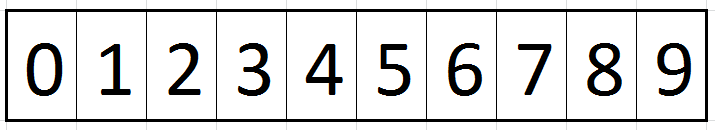


ArrayLetters db 5 ('a', 'b', 'c', 'd', 'e')

ייצור מערך של 25 איברים (5\*5), כל ערך בגודל בית, המכיל את תווי (a b c d e) ASCII



איברים במערך



לכל איבר במערך יש אינדקס

האיבר הראשון במערך הוא באינדקס 0

האיבר השני במערך הוא באינדקס 1 וכו'

תרגום אופרנד לכתובת בזיכרון

האסמבלר ממיר את האופרנדים הבאים (בגודל 16 ביט) לכתובות בזיכרון (בגודל 20 ביט)?

mov [1], ax

mov [var], ax

mov [bx], ax

האופרנדים מייצגים אופסט בלבד

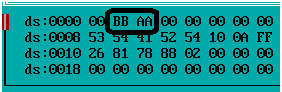
האסמבלר מניח שאנחנו מתייחסים לסגמנט ds

כלומר [01] הוא למעשה [ds:01]

mov ax, 0AABBh

הערך AABB מועתק לכתובת [01] ול-[02] ב – ds (במקטע הנתונים)

mov [01], ax



ובמערך:

[שם המשתנה] מבציע על מיקום תחילת המערך בסגמנט הנתונים ds.

כדי להגיע למיקום במערך יש להוסיף את האינדקס של האיבר במערך.

DATASEG

array db 0AAh, 0BBh, 0CCh, 0ddh, 0EEh, 0FFh

CODESEG

AA ⇨ ds:00

BB ⇨ ds:01

CC ⇨ ds:02

DD ⇨ ds:03

EE ⇨ ds:04

FF ⇨ ds:05

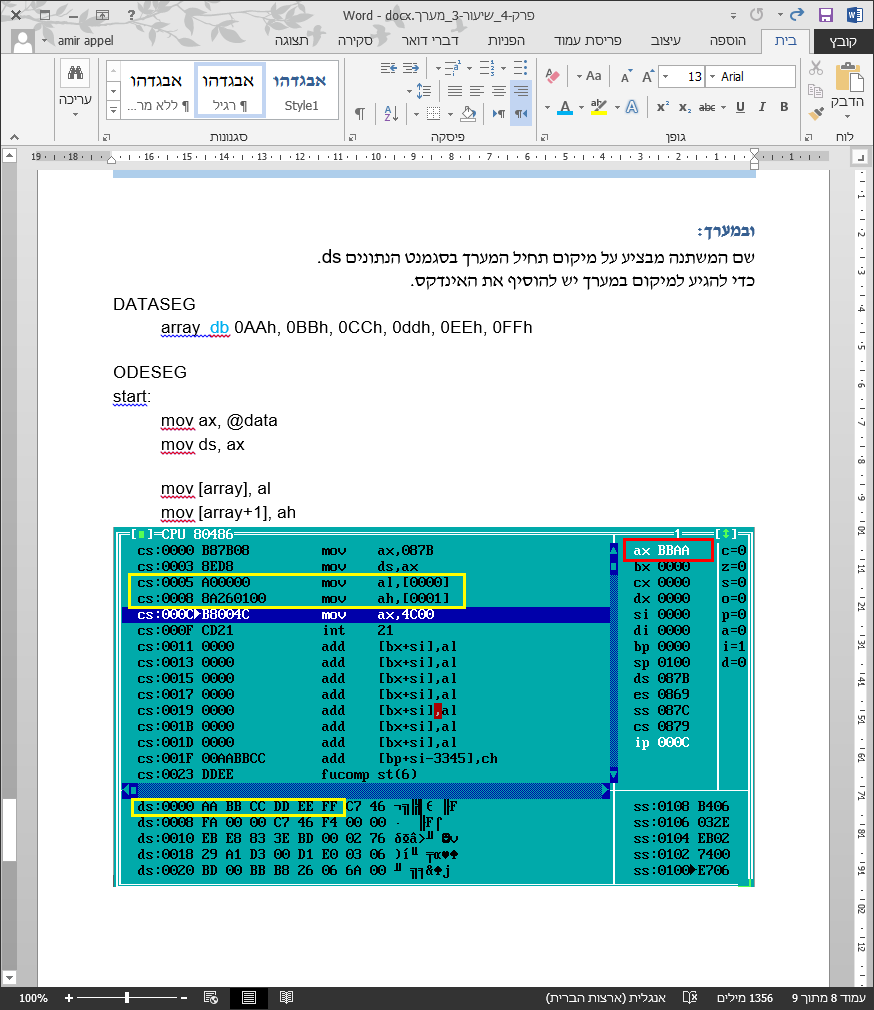
start:

mov ax, @data

mov ds, ax

mov al, [array]

mov ah, [array+1]



שימו לב 🎔 AA הועתק לכתובת הנמוכה של ax ⇦ al

BB הועתק לכתוב הגבוה של ax ⇦ ah

העתקה אל מערך

DATASEG

array db 6 dup (0)

העתקת הקבוע AAh לתוך האיבר הראשון במערך. אינדקס - 0

**mov [array],** **0AAh**

האסמבלר יתרגם את [[array מכתובת יחסית לכתובת קבועה

CODESEG

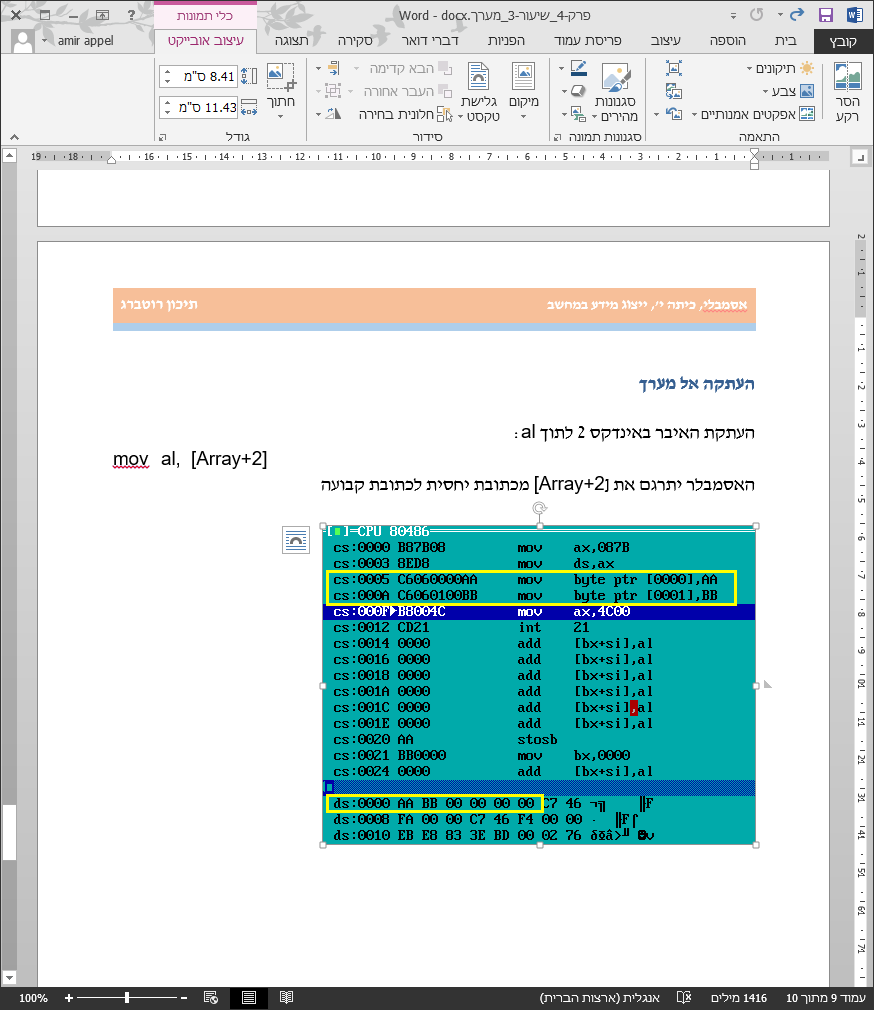
start:

mov ax, @data

mov ds, ax

mov [array], 0AAh

mov [array+1], 0BBh



פקודת offset

פקודת ה – offset מעתיקה לרגיסטר bx את הכתובת של תחילת המערך, האיבר הראשון במערך ⇦ אינדקס 0.

דרך נפוצה לטיפול במערכים היא להעתיק ל-bx את כתובת תחילת המערך:

mov bx, offset array

לרגיסטר bx תועתק כתובת תחילת המערך,   
כתובת האיבר הראשון במערך ⇦ האיבר שנמצא באינדקס 0.

העתקת האיבר באינדקס 2 לתוך al:

mov al, [bx+2]

DATASEG

array db 0AAh, 0BBh, 0CCh, 0DDh, 0EEh, 0FFh

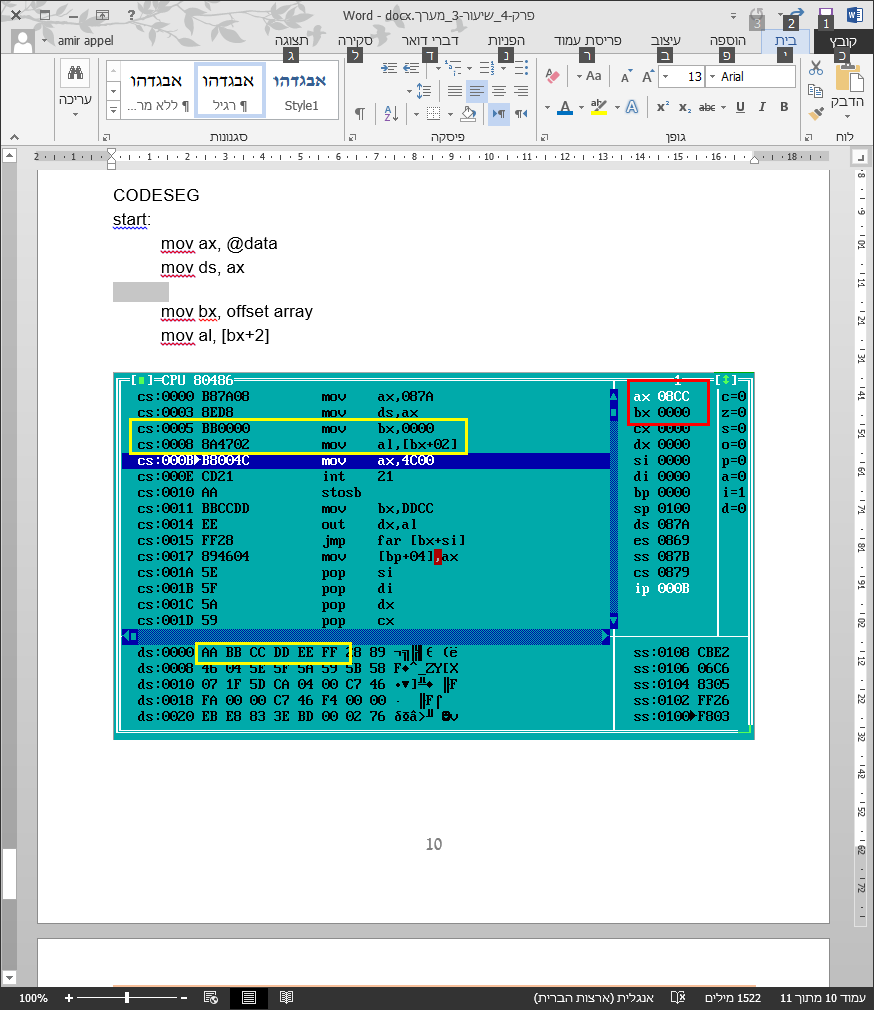
CODESEG

start:

mov ax, @data

mov ds, ax

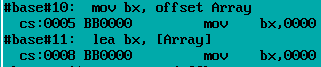
mov bx, offset array

 mov al, [bx+2]

פקודת LEA

LEA: Load Effective Address

כמו פקודת offset, התרגום הוא לאותו קוד מכונה



ההנחיה word ptr

אם נרצה להעתיק לרגיסטר ax את האיבר השלישי במערך.

כאשר נריץ את התכנית ב - turbo debugger נקבל הודעת שגיאה שהגודל של האופרנדים אינו מתאים.

DATASEG

array db 0AAh, 0BBh, 0CCh, 0DDh, 0EEh

CODESEG

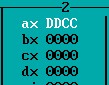
…

mov ax, [array+2]



הפקודה - **byte ptr / word ptr**

מודיעה לאסמבלר לבצע העתקה של שני בתים לרגיסטר

DATASEG

array db 0AAh, 0BBh, 0CCh, 0DDh, 0EEh

CODESEG

…

mov ax, [**word ptr** array+2]

אופרנד המקור הוא בגודל בית, שכן ציינו בהצהרה על המערך db.  
אופרנד היעד ax הוא בגודל מילה – הרגיסטר הוא אופרנד המכיל 16 בתים.

אי לכך, עלינו ליידע את האסמבלר שאנחנו מנסים לבצע העתקה של שני בתים – ובכך להימנע מהשגיאה. ax, [word ptr array+2] mov

הודענו לאסמבלר להתייחס אל המיקום בזיכרון לא בתור byte בודד, אלא בתור word - שני בתים. לתוך ax תועתק כמות זיכרון בגודל word. מיקום תחילת הזיכרון הוא בכתובת Array+2. רגיסטר al קיבלת את הכתובת הנמוכה ורגיסטר ah קיבל את הכתובת הגבוה.

byte ptr / word ptr – המשך

* mov [bx], 5

פקודה זו חוקית. אך אם נריץ אותה האסמבלר יכתוב לנו הודעת שגויה של type override.

ניתן להשתמש בזיכרון בגדלים שונים כדי לשמור את הערך 5 (בית, מילה, בית כפול)

באיזה גודל זיכרון להשתמש?

צריך לתת הנחיה לאסמבלר באיזה גודל בית להשתמש.

mov [byte ptr bx], 5

mov [word ptr bx], 5

הפקודה byte ptr מגדירה להקצות בית אחד לשמירת הערך.

הפקודה word ptr מגדירה להקצות מילה לשמירת הערך.

type override אזהרת

mov [bx], 5

פקודה זו חוקית אך נתונה ליותר מפרשנות אחת:  
את הערך 5 אפשר לשמור במשתנה בגודל בית אחד, מילה, מילה כפולה... האסמבלר צריך לדעת כמה בתים להקצות לטובת שמירת הערך 5.

האסמלבר יחזיר לנו אזהרת type override.



הפתרון הוא פשוט להורות לאסמבלר **כמה בתים להקצות בזיכרון** לשמירת המידע.

mov [byte ptr bx], 5 ; לגשת לבית אחד **בזיכרון**

mov [word ptr bx], 5 ; לגשת לשני בתים **בזיכרון**

mov [byte ptr bx], 5 ⇨ [הכתבות בזיכרון] = 00000101 ≈ 0h5

mov [word ptr bx], 5 ⇨ [הכתבות בזיכרון] = 000h5

תרגול

**רשימת פסיקות - מחסן פקודות**

|  |  |
| --- | --- |
| פקודה להדפסת מחרוזת  (שם משתנה המחרוזת msg) | mov dx, offset msg  mov ah, 9h  int 21h |
| פקודה להדפסת תו על המסך | mov dl, 'x' ; print x  mov ah, 2  int 21h |
| פקודה למעבר שורה | mov dl, 0ah  mov ah, 2h  int 21h |
| פקודה לקליטת מספר מהמשתמש ושמירתו לרגיסטר al | mov ah, 1h  int 21h  sub al, '0' |
| פקודה להדפסת ערך השמור ברגיסטר **al** בתרגום הערך למספר בבסיס 10 על המסך  Ten הוא שם משתנה המכיל את הערך 10  ten db 10 | mov ah, 0  div [ten]  add ax, '00'  mov dx, ax  mov ah, 2h  int 21h  mov dl, dh  int 21h |

**לולאה:**

mov cx, 5

SomeLabel:

; פקודות לביצוע

loop SomeLabel

מערכים:

הצהרה על משתנים נעשית בסגמנט הנתונים – Data Segment.

name type value

num db 5

# הצרה על מערך

מערך הוא אוסף של איברים בעלי גודל זהה. לכל איבר במערך שי אינדקס המצביע על מיקומו בערך.

הצהרה על מערך

name type value1, value2, value3

array1 db 2, 4, 6

array2 db 5 dup (0)

יוצר מערך של 5 איברים בגודל byte המאותחלים בערך 0

array3 dw 4 dup (?)

יוצר מערך של 4 איברים בגודל word שאין להם ערכים התחלתיים

array4 db 6 dup ('X')

יוצר מערך של 6 איברים בגודל byte המכילים את האות X.

לכל איבר במערך יש אינדקס המצביע על מיקומו במערך והכתובת שלו:

array1 db 2, 4, 6

|  |  |  |  |
| --- | --- | --- | --- |
| כתובת בזיכרון | array1 | array1 + 1 | array1 + 2 |
| אינדקס במערך | 0 | 1 | 2 |
| ערך | 2 | 4 | 6 |

מצביע bx

bx הוא רגיסטר מיוחד. הוא משמש בעיקר כמצביע לכתובת בזיכרון.

****פקודת offset****

פקודת ה – offset מעתיקה לרגיסטר bx את הכתובת של תחילת המערך,   
האיבר הראשון במערך ⇦ אינדקס 0.

דרך נפוצה לטיפול במערכים היא להעתיק ל-bx את כתובת תחילת המערך:

mov bx, offset array

לרגיסטר bx תועתק כתובת תחילת המערך,   
כתובת האיבר הראשון במערך ⇦ האיבר שנמצא באינדקס 0.

mov al, [bx+2]

העתקת האיבר באינדקס 2 של המערך לתוך al:

תרגילים

1. הגדירו מערך של 8 בתים שמאותחלים לערך 5 (הספרה 5).

הגדירו מערך של 8 מילים שמאותחלות לערך '5' (התו – 5).

הגדירו מערך של 8 בתים שמאותחל לערך 'e'

השוו את תמונת הזיכרון בשלושת המערכים!

הריצו את התכנית ב – TD

tasm /zi file\_name

tlink /v file\_name

td file\_name

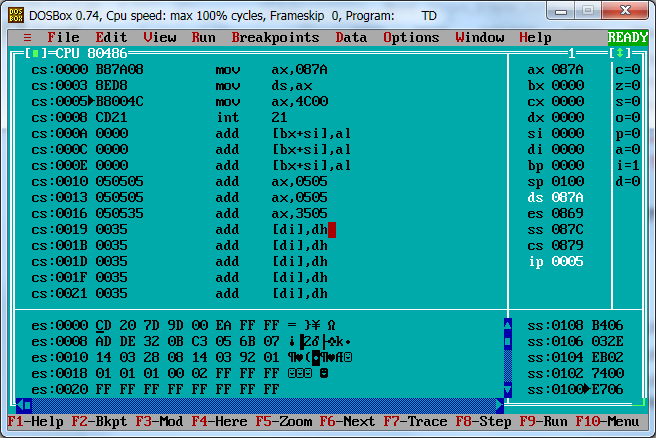
בדקו ב – td את התצוגה של ה – cpu

הריצו את שתי השורות הראשונות של הקוד.

mov ax, @data

mov ds, ax

בעזרת המקש F8.

בעזרת המקש Tab עברו לאזור של ה – DataSeg

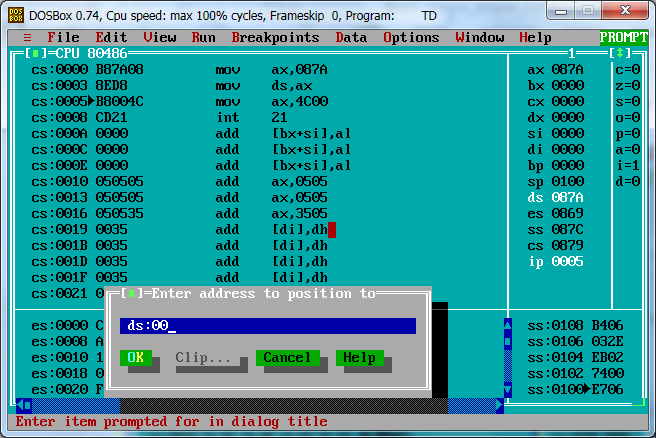
ראו שמסגרת מקטע הנתונים נצבעה בכחול

הקישו על מקש

Ctrl + g

בחלון שנפתח

הקישו ds:00 והקישו Enter.

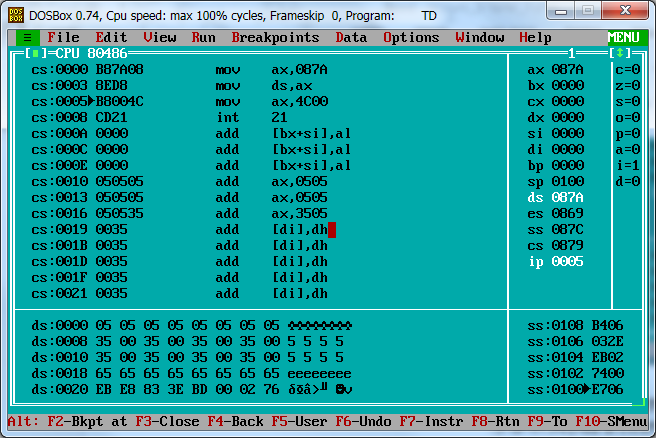


תוכלו לראות את הנתונים במקטע הזיכרון.

ראו את השורה הראשונה - 8 תאי זיכרון המכילים את הערך 5

שתי השורות הבאות- 4 מילים בכל שורה מכלים את התו – 5

ובשורה הרביעית – 8 תאי זיכרון בהם התו e.



DATASEG

; --------------------------

arr1 db 8 dup (5)

arr2 dw 8 dup ('5')

arr3 db 8 dup ('e')

CODESEG

start:

mov ax, @data

mov ds, ax

; --------------------------

; Your code here

; --------------------------

exit:

mov ax, 4c00h

int 21h

END start

1. הצהירו על מערך בן 4 איברים.

השמו בו ערכים בסדר עולה מ 1 עד 4.

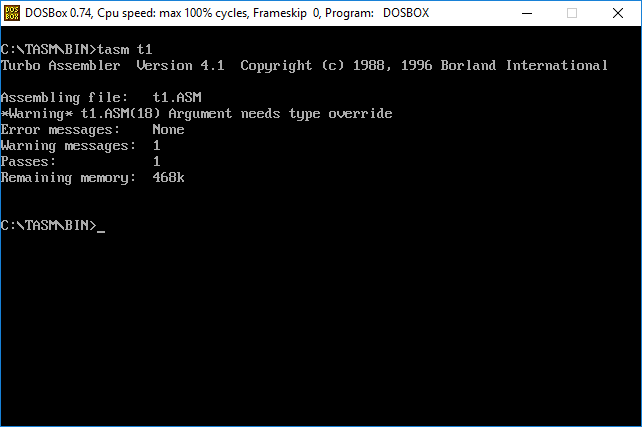
הגדילו כל תא ב – 10h. הפקודה לחיבור שני אופרנדים היא: add destination, source

כאשר רוצים להוסיף למשתנה במערך, שפנינו לכתובת שלו בעזרת רגיסטר [bx], מספר קבוע.

add [bx], 10h

בהרצת התוכנית נקבל הודעת שגיאה.

Argument needs type override



הודעה זו היא מאחר והאסמבלי לא יודע האם התא שהפנינו אליו את הוא בגודל מילה, בית או מילה כפולה. לכן יש להודיע לו לאיזה גודל משתנה אנו רוצים לפנות וזאת נעשה ע"י פקודת:

byte ptr / word ptr

add [byte ptr bx], 10h

בדקו את התכנית ב – TD

הוסיפו לתכנית פסיקה להדפסת ערך השמור ברגיסטר al ופסיקה המדפיסה למסך רווח והציגו את המערך החדש למסך.

DATASEG

; --------------------------

arr1 db 1, 2, 3, 4

ten db 10

CODESEG

;print al register

proc print\_al

mov ah, 0

div [ten]

add ax, '00'

start:

mov ax, @data

mov ds, ax

; --------------------------

mov cx, 4

mov bx, offset arr1

add\_loop:

add [byte ptr bx], 10h

mov al, [bx]

call print\_al

call print\_L

inc bx

loop add\_loop

mov dx, ax

mov ah, 2h

int 21h

mov dl, dh

int 21h

ret

endp print\_al

proc print\_L

mov dl, ' '

mov ah, 2

int 21h

ret

endp print\_L

1. הצהירו על מערך של 5 אותיות.

השמו בו את האותיות: 'A',B','C','D', 'E',.

הפרש בייצוג של אותיות "גדולות" ו"קטנות" הוא 32. הגדילו כל אות ב - 32 והדפיסו את המערך.

שימו לב 🎔, הפסיקה של הדפסת תו **שונה** מהדפסה של מספר השמור ברגיסטר al.

DATASEG

; --------------------------

; Your variables here

array1 db 'A,'B','C','D', 'E'

בתחילת הלולאה רגיסטר cx = 5

רגיסטר bx מצביע על תא בכתובת [00] (תחילת המערך)

הערכים בזיכרון הם של אותיות "גדולות" =ABCDE.

נוסיף לכל אות אסקי 32 ונציג את התוצאה למסך.   
רצוי ליצור פעולה חדשה להדפת תווים.

CODESEG

proc print\_L

mov dl, ' '

mov ah, 2

int 21h

ret

endp print\_L

proc print\_tav

mov cx, 5

mov bx, offset arr1

printLoop:

add [byte ptr bx], 32

mov al, [bx]

call print\_tav

call print\_L

inc bx

loop printLoop

mov dl, al

mov ah, 2

int 21h

ret

endp print\_tav

start:

mov ax, @data

mov ds, ax

1. כתבו תכנית המחשבת את סכום האיברים במערך.  
   צרו מערך של חמישה בתים וקבעו להם ערכים, סכמו את האיברים במערך והציגו את התוצאה.   
   **הערה**: שימו לב שסכום האיברים במערך לא יעלה על 255.

DATASEG

; --------------------------

; Your variables here

array db 5 dup (5)

ten db 10

CODESEG

start:

mov ax, @data

mov ds, ax

; --------------------------

; Your code here

mov bx, offset array

mov cx, 5

mov al, 0

sum:

mov ah, [bx]

add al, ah

inc bx

loop sum

**call print\_al**

1. כתבו תכנית בה מערך של מספרים, בעל 8 תאים.

החליפו את המיקום של כל זוג מספרים:  
הערך של אינדקס 0 יוחלף עם הערך של אינדקס 1 וכך הלאה.  
הציגו את המערך למסך.

לדוגמה במערך: 1 2 3 4 5 6 7 8

נקבל : 2 1 4 3 5 6 5 8 7

DATASEG

; --------------------------

ten db 10

arr db 1, 2, 3, 4, 5, 6, 7, 8

CODESEG

proc print\_al

mov ah, 0

div [ten]

add ax, '00'

mov dx, ax

mov ah, 2h

int 21h

mov dl, dh

int 21h

ret

endp print\_al

proc print\_L

mov dl, ' '

mov ah, 2

int 21h

ret

endp print\_L

start:

mov ax, @data

mov ds, ax

; --------------------------

mov bx, offset arr

mov cx, 4

change:

mov al, [byte ptr bx]

mov ah, [byte ptr bx+1]

mov [byte ptr bx], ah

mov [byte ptr bx+1], al

add bx, 2

loop change

mov cx, 8

mov bx, offset arr

print\_arr:

mov al, [byte ptr bx]

call print\_al

call print\_L

inc bx

loop print\_arr

