שיעור 2 – פקודת בסיסיות ומשתנים

נושאי שיעור זה (כמו גם רוב נושאים האסמבלי) מופיעים בספר "מבוא למערכות מחשב ואסמבלי" של האוניברסיטה הפתוחה. הספר הוא בעברית וקל מאוד לקריאה. כדאי מאוד להיעזר בו. הנושאים של שיעור זה נמצאים בעיקר בפרקים 4,5,6.

מפת הזיכרון

אם גודל כל האוגרים במעבד הוא 16 ביט, אז באופן עקרוני היינו מצפים שכמות הזיכרון שהמעבד יכול "לראות" הוא 2^{16} בתים (כי יכולות להיות רק 2^{16} כתובות זיכרון שונות שאוגר יכול להכיל) שהם 64 קילובייט. ברם, מעבד ה-8086 משתמש במעין "תעלול" שמאפשר לו לראות ולהשתמש בזיכרון בגודל 1 מגבייט שהם 2^{20} בתים. במבט ראשון זה נראה בלתי אפשרי, כי איך אוגר בגודל 16 ביט יכול להכיל כתובת בגודל 20 ביט.

הפתרון נעוץ בכך שהזיכרון מחולק למקטעים (Segments) בגודל 64KB כל אחד. כל כתובת זיכרון הנקראת **כתובת פיזית** (Physical Address) בגודל 20 ביטים מורכבת בעצם משתי כתובות:

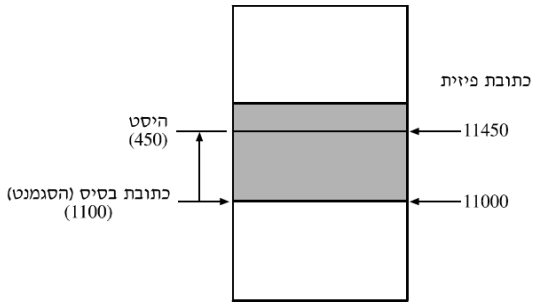
- **כתובת הבסיס** (Base Address)– זוהי הכתובת של תחילת המקטע
- **כתובתיחסית/היסט**(Relative Address או Offset) –זהו המרחק מתחילת המקטע

גם כתובת הבסיס וגם כתובת ההיסט נשמרות באוגרים בגודל 16 ביט, כאשר כתובת הבסיס נשמרת באחד מבין אוגרי מקטע (Segment Register)–CS, DS, ES, SS, וכתובת ההיסט נשמרת בתוך "אוגר רגיל" (כמו BX, SP וכד').
כיצד אם כן בונים כתובת בת 20 ביטים מתוך שתי כתובות בגודל 16 ביטים? ע"י הזזה:

תחילה מוסיפים לכתובת הבסיס 4 סיביות מימין: 0000 (שהן הספרה ההקסדצימאלית0h). תוספת זו שקולה להכפלת כתובת הבסיס ב-16. לאחר מכן מוסיפים לתוצאה שהתקבלה את ההיסט.

לדוגמה, נתבונן באיור 4.3, בו כתובת הבסיס היא 1100h והכתובת היחסית היא 450h.
המעבד מפרש את כתובת הבסיס כ-11000h והוא מוסיף לה את הכתובת היחסית 450h
ובהתאם:

```
  11000h
+  450h
  11450h   כתובת פיזית
```



איור 4.3
הקשר בין הכתובת הפיזית 11450h והכתובת היחסית 450h

מקטעים סטנדרטיים

לרוב נהוג לחלק את הזיכרון למקטעים על פי תפקיד חלק זה בזיכרון. כלומר בד"כ מרחב הזיכרון של התכנית יכיל לפחות 3 מקטעים עיקריים, שלכל אחד מהם תפקיד מוגדר. על כל אחד מהם יצביע אחד מבין אוגרי המקטע באופן הבא:

- אוגר המקטע CS יצביע על ה-Code Segment (מקטע הקוד) – זהו החלק בזיכרון שמכיל את התכנית עצמה (את הפקודות עצמן). זאת אומרת אנחנו יכולים לומר שהכתובת של הפקודה שהמעבד הולך לבצע בכל שלב מורכבת מכתובת המקטע CS והכתובת היחסית בתוך המקטע IP.
- אוגר המקטע DS יצביע על ה-Data Segment (מקטע המידע) – זהו החלק בזיכרון שמכיל את המשתנים בהם משתמשת התכנית.
- אוגר המקטע SS יצביע על ה-Stack Segment (מקטע המחסנית) – זהו החלק בזיכרון שמכיל את המחסנית, עליה נלמד בשיעורים הבאים.

מבנה הפקודות

הפקודות בשפת אסמבלי נראות למשל כך:

```
mov ax,1234h
```

מספר מושגים:

- פקודה/הוראה (Instruction) - זוהי הפקודה כולה שמתורגמת ע"י האסמבלר לפקודה אחת בשפת המכונה.
- אופקוד(OpCode)– זהו החלק של "שם הפקודה". לדוגמא mov או add. לפעמים מתייחסים במילה Opcode דווקא לייצוג הבינארי של שם הפקודה.
- אופרנד (Operand) – אלו הם הערכים עליהם מבצעים את הפעולה. לדוגמא: ax או 1234h.

מספר האופרנדים:

לפקודה בודדת יכולים להיות 0 או יותר אופרנדים. למשל:

- הפקודה cld (שמאפסת את אחד הדגלים) לא מקבל אף אופרנד.
- הפקודה inc (שמעלה את ערך של ax ב-1) מקבלת אופרנד אחד. לדוגמא: inc ax
- הפקודה mov שראינו מקבלת שני אופרנדים.

כאשר יש לפקודה שני אופרנדים, מפרידים ביניהם עם פסיק.

שמירת התוצאה:

1. הזכרנו כאן למשל פקודת חיבור (Add) שמקבל שני אופרנדים ומחברת ביניהם. אבל לא אמרנו איפה נשמרת התוצאה. התשובה היא שב-x86 הנוהג הוא שהאופרנד הראשון הוא גם היעד של הפקודה (destination).כלומר הפקודה add ax,bx בעצם מבצעת ax = ax + bx, כלומר שומרת את תוצאת הפעולה באופרנד הראשון.
2. שימו לב שזה אומר שהאופרנד הראשון (בפקודה שיש לה תוצאה) לא יכול להיות ערך מיידי (Immediate Value). לדוגמא: אי אפשר לכתוב Add 7,ax שכן המעבד לא יכול לשמור את התוצאה במספר הקבוע 7 (בניגוד לפקודה החוקית add ax,7 שם התוצאה נשמרת ב-ax).

סדר הפעולות:

בפעולות בהן לשני האופרנדים תפקיד שונה, האופרנד הראשון הוא בד"כ האופרנד הראשי. לדוגמא:

1. בפעולת חיסור sub (substract), האופרנד הראשון הוא המחוסר והשני הוא המחסר. כלומר הפקודה sub ax,bx עושה את הפעולה ax = ax-bx.
2. בפעולת השמה mov האופרנד הראשון הוא היעד. כלומר movax,bx מבצע ax=bx– השמה של הערך בתוך bx לתוך ax. הערך בתוך bx לא משתנה. הערך "הישן" של ax הולך לאיבוד.

סוגי אופרנדים:

קיימים מספר סוגי אופרנדים שחשוב מאוד להכיר ולהבין אותם:

- ערך מיידי (Immediate Value) – זהו בעצם ערך קבוע המקודד בפקודה עצמה, למשל בפקודה add ax,7 המספר 7 הוא אופרנד מסוג ערך מיידי.
- אוגר –במקרה זה הפקודה משתמש בתוכן של האוגר המדובר. לדוגמא בפקודה add ax,7 האופרנד הראשון הוא אוגר, והפקודה תוסיף לתוכן האוגר ax את הערך 7.
- כתובת זיכרון ישירה – במקרה זה הפקודה תיקח את הערך מתוך הזיכרון בכתובת אותה מציין האופרנד. ציון הכתובת מתבצע באופן בו למדנו – כלומר היא מורכבת מכתובת בסיס (בצורה של אוגר מקטע) וכתובת יחסית (Offset) שכתובה בפקודה. לדוגמא הפקודה add ds:[100h],7 תוסיף את הערך 7 לערך שנמצא בתא בכתובת היחסית 100h במקטע עליו מצביע אוגר המקטע ds.
- כתובת זיכרון בלתי ישירה – גם במקרה זה האופרנד מציין כתובת בזיכרון, רק שכעת ערך ההיסט לא מופיע ישירות בפקודה אלא נלקח מתוך אוגר בזמן ריצה. למשל: הפקודה add ds:[bx],7 תוסיף 7 לתא בזיכרון שנמצא בכתובת הבאה:
  - כתובת הבסיס של המקטע נמצא באוגר ds
  - ההיסט (הכתובת היחסית) נמצא באוגר bx.

מספר הערות על אופרנדים מסוג כתובת:

1. שימו לב שהדרך לכתוב אופרנדים שמציינים כתובות היא באמצעות **[ ] (סוגריים מרובעים)**. הסוגריים המרובעים אומרים לאסמבלי שאנחנו רוצים להשתמש בתוכן תא הזיכרון המדובר. כלומר הסוגריים המרובעים מזכירים את אופרטור ה-* בשפת C. לדוגמא:
   - הפקודה mov bx,7 תשים את הערך 7 באוגר bx. לא יהיה שום שינוי בזיכרון.
   - הפקודה mov ds:[bx],7 תשים את הערך 7 בתא הזיכרון שנמצא בהיסט bx ביחס למקטע ds. לא יהיה שום שינוי בערך האוגר bx (או בערך האוגר ds) בעצמו. לדוגמא, אם האוגר bx הכיל את הערך 12h והאוגר ds הכיל את הערך 120h, אז הפקודה תשים את הערך 7 בתא בכתובת 1212h (זכרו שיש להוסיף 4 ביטים/ספרה הקסדצימלית 0 לאוגר המקטע).
2. פעמים רבות כאשר מציינים כתובת, ניתן להשמיט את כתיבת אוגר המקטע. זה לא אומר שבפועל לא יתבצע התהליך של בניית הכתובת מתוך כתובת הבסיס וההיסט, אלא רק שהאסמבלר יכניס בעצמו לפקודה את אוגר המקטע ה-default-יבי עבור כתובת זו. אוגר המקטע ה-default-יבי נקבע לפי האוגר שמשמש לציון הכתובת היחסית (ולכן זה אפשרי רק כאשר משתמשים בכתובת זיכרון בלתי ישירה, כלומר משתמש באוגר לציון הכתובת היחסית ולא כותבים את הכתובת היחסית בצורה ישירה). לדוגמא:
   - כאשר כותבים mov [bx],7 האסמבלר משתמש באוגר המקטע ה-default-יבי עבור bx שהוא ds (ה-data segment).
   - כאשר כותבים mov [sp],8 האסמבלר משתמש באוגר המקטע ה-default-יבי עבור sp שהוא ss (ה-stack segment).

גודל האופרנד:

שימו לב שגודל האופרנד יכול להיות בית אחד או שני בתים. לפעמים זה ברור מההקשר ולפעמים צריך לציין את זה באופן מפורש בפקודה. לדוגמא:

1. בפקודה movax,bx גודל שני האופרנדים הוא 2 בתים/16 ביט.
2. בפקודה moval,bl גודל שני האופרנדים הוא בית אחד/8 ביט (להזכירכם, הסיומות h,l משמשות לגישה ישירה לבתים של האוגרים הכלליים).
3. הפקודה moval,bx היא <u>לא חוקית</u>, כי יש חוסר התאמה בין הגדלים של שני האופרנדים.
4. בפקודה mov ds:[bx],ax גודל האופרנדים הוא 2 בתים/16 ביט. כלומר הפקודה מבצעת השמה של 16 ביט מתוך האוגר ax לזיכרון.
5. בפקודה mov ds:[bx],al גודל האופרנדים הוא בית אחד/8 ביטים. כלומר הפקודה מבצעת השמה של 8 ביט מתוך הבית התחתון של האוגר ax לזיכרון. שימו לב שהמעבד עדיין משתמש בכל 16 הביט של האוגרים bx ו-ds כדי לבנות את הכתובת בזיכרון שאליה מתבצעת פעולת ההשמה.
6. הפקודה mov ds:[bx],5 היא <u>דו-משמעית ולכן לא חוקית.</u> הסיבה היא שהאסמבלר לא יכול לנחש אם הכוונה שלנו היא לעשות השמה בגודל 16 ביט (כמו בדוגמא 4 הנ"ל) או בגודל 8 ביטים (כמו בדוגמא 5 הנ"ל). למה בכלל יש הבדל? כי השמה של 16 ביט מסתכלת על היעד בזיכרון בתור משתנה בגודל 16 ביט (שני בתים) ולכן אם אנחנו עושים השמה של הערך 5 לתוך המשתנה הזה, הפקודה תאפס את כל הבית העליון. ואילו השמה של 8 ביטים מסתכלת על היעד בזיכרון בתור משתנה בגודל 8 ביטים (בית בודד), ולכן השמה של הערך 5 לתוך המשתנה הזה לא תשנה שום בית נוסף.

לכן כדי לפתור את דו-המשמעות, אנחנו צריכים לציין באופן מפורש את גודל האופרנד באופן הבא:

- אם גודל האופרנד הוא בית אחד, נכתוב: mov **byte ptr** ds:[bx],5
- אם גודל האופרנד הוא שני בתים, נכתוב: mov **word ptr** ds:[bx],5

בציור: נניח שאוגר המקטע ds מכיל את הכתובת 0, האוגר bx מכיל את הכתובת 101h, והזיכרון <u>לפני</u> הפעולה נראה כך:

כתובת	תוכן
100h	143
101h	'A'
102h	7
103h	0

אז אם נבצע mov **byte ptr** ds:[bx],5 הזיכרון <u>אחרי</u> הפעולה ייראה כך:

כתובת	תוכן
100h	143
101h	5
102h	7
103h	0

ולעומת זאת אילו נבצע mov **word ptr** ds:[bx],5 הזיכרון <u>אחרי</u> הפעולה ייראה כך:

כתובת	תוכן
100h	143
101h	5
102h	0
103h	0

זאת אומרת המעבד התייחס לתא בכתובת 101h בתור משתנה בגודל <u>שני בתים</u>, ולכן השמה של הערך 5 לתוך המשתנה <u>מאפסת את הבית העליון של המשתנה.</u>

הפקודות עצמן

בשלב זה, אתם בטח שואלים את עצמכם "אוקיי, אבל מה עם הפקודות עצמן? איך אני עושה כל מיני פעולות?". אנחנו לא רוצים לעבור על המון פקודות וללמוד מה הן עושות (זה יהיה מאוד מייגע). הדרך הטובה והנכונה ללמוד פקודות ספציפיות היא להיעזר במדריך הפקודות של המעבד, שם מתועדות כל הפקודות. לשימושכם מדריך מקוצר של הפקודות של מעבד 8086 בקובץ text מצורף.

Intel8086.txt

כמו כן, כמובן שיש לרשותכם גם את האינטרנט ואת הספר של האוניברסיטה הפתוחה. בינתיים שווה שתסתכלו במדריך הפקודות ותבינו את הפקודות:

add, sub, mul, inc, dec, mov, shl, shr, not, xor, or, and, neg

הסבר על פקודות אלה מופיע גם בפרק 5 של הספר של האוניברסיטה הפתוחה.

בשיעורים הבאים נלמד נושאים מתקדמים יותר באסמבלי ועל הדרך נלמד גם עוד פקודות מתקדמות.

משתנים וזיכרון

רקע – משתנים בשפת C

כדי להבין טוב יותר את סוגי המשתנים באסמבלי, ראשית ניזכר בשלושת הסוגים המרכזיים של משתנים בשפת C (שימו לב, אין כאן הכוונה לטיפוסים של המשתנים, אלא למאפיינים שונים של הזיכרון בו הוא משתמש וזמן חייו):

1. **<u>משתנה גלובאלי</u>** (נקרא גם משתנה סטאטי)
דוגמא:

```
int num;
int main()
{
 …
}
```

א. **מתי הוא חי?** זהו משתנה ש"חי" לאורך כל זמן ריצת התכנית – מהרגע שהיא נטענת ועד לרגע שהיא מסתיימת.

ב. **מתי נקבע גודל הקצאת הזיכרון?** גודל הקצאת הזיכרון נקבע *<u>בזמן קומפילציה</u>*.

ג. **כמה עותקים יש למשתנה**? למשתנה עותק אחד ויחיד.

ד. **באיזה זיכרון הוא נמצא?** המשתנה נמצא בזיכרון <u>בו נמצאת התכנית עצמה</u>.בד"כ תכניות מחולקות למקטעים (sections): code section, data section וכד'. המשתנים הגלובליים נמצאים בד"כ ב- code section. כאשר התכנית נטענת לזיכרון, מערכת ההפעלה מקצה זיכרון למקטעים השונים, וביניהם גם למשתנים הגלובאליים. **שימו לב!** המשתנים הגלובליים **לא יושבים במחסנית או בערימה.**

2. **<u>משתנה מקומי</u>**

דוגמא:

```
void foo()
{
        int num;
        …
}
```

א. **מתי הוא חי?**זהו משתנה שחי רק בזמן החיים של התחום (scope) בו הוא מוגדר. התחום של משתנה יהיה בד"כ פונקצייה, והמשתנה יווצר בעת הכניסה לפונקצייה ויהרס לאחר סיומה.

ב. **מתי נקבע גודל הקצאת הזיכרון?** גודל הקצאת הזיכרון נקבע *<u>בזמן קומפילציה</u>*.

ג. **כמה עותקים יש למשתנה**? למשתנה יכולים להיות מספר עותקים שחיים בו זמנית, בהתאם למספר העותקים הרצים של התחום בו הוא מוגדר. למשל אם זהו מתשנה מקומי של פונקצייה רקורסיבית, אז מספר העותקים שלו יהיה על פי העומק של הרקורסיה ברגע נתון.

ד. **באיזה זיכרון הוא נמצא?** המשתנה מוגדר במחסנית של התכנית. כאשר תכנית מופעלת ע"י מערכת ההפעלה, היא מקבל מקום עבור המחסנית שלה, ובזמן ריצת התכנית, התכנית משתמשת במקום זה עבור המשתנים המקומיים. (אנחנו נלמד יותר על המחסנית בשיעור הבא).

3. **<u>משתנה דינאמי</u>**

דוגמא:

```
void foo(int size)
{
        int* p = (int*)malloc(size);
}
```

א. **מתי הוא חי?** זהו משתנה שחי מהרגע בוא הוא מוקצה דינאמית ע"י הפונקצייה malloc, ועד הרגע בו הוא משוחרר ע"י הפונקצייה free.

ה. **מתי נקבע גודל הקצאת הזיכרון?** גודל הקצאת הזיכרון נקבע *<u>בזמן ריצה</u>*.

ו. **כמה עותקים יש למשתנה**? נוצר עותק עבור כל קריאה בזמן ריצה לפונקצייה malloc (למשל אם נקרא לפונקצייה 10 פעמים בלולאה, יווצרו 10 משתנים שונים).

ז. **באיזה זיכרון הוא נמצא?** המשתנה מוגדר בערימה (heap) של התכנית. בקצרה ניתן לומר שזהו איזור זיכרון המנוהל ע"י מערכת ההפעלה.

משתנים בשפת אסמבלי

בקורס שלנו לא נשתמש במשתנים דינאמיים (למרות שזה אפשרי), אלא נתמקד בשני סוגי המשתנים האחרים –

א. בשיעור היום נלמד על משתנים **גלובאליים**

ב. בשיעור הבא נלמד על המחסנית ועל משתנים **מקומיים**

משתנים גלובאליים

המאפיינים של המשתנים הגלובאליים בשפת C, אותם תיארנו לעיל, תקפים גם לגבי שפת אסמבלי. כלומר משתנים גלובאליים הם משתנים בעלי **עותק יחיד**, הם חיים **לאורך כל זמן ריצת התכנית,** והם יושבים **בזיכרון של התכנית עצמה**.

בשפת C קיימים טיפוסים רבים (int, char, double), ושימוש נכון בטיפוסים נאכף ע"י הקומפיילר. בשפת אסמבלי ה"טיפוס" של משתנה הוא פשוט הגודל שלו בבתים. כלומר יש לנו משתנה מסוג byte (שגודלו בית אחד), משתנה מסוג word (שגודלו 2 בתים) ומשתנה מסוג dword (שגודלו 4 בתים).כמו כן, ניתן להגדיר מערך של משתנים מכל אחד מהגדלים.

התחביר של הגדרת משתנה גלובאלי בשפת אסמבלי הוא:

```
<var name><data type><init value>
```

כאשר:

- var name הוא שם המשתנה (שחייב להתחיל באות a-zA-Z, יכול להיות רצף אותיות ומספרים, **ולא יכול** להכיל קו תחתון underscore _).
- data type הוא אחד מבין

2	מספר בתים	שם
בית – byte	1	DB
מילה – word	2	DW
מילה כפולה – Double Word	4	DD
מילה מרובעת – Quad Word	8	DQ

- init value הוא ערך התחלתי של המשתנה (הערך אותו הוא יקבל בתחילת פעולת התכנית).

<u>דוגמאות</u>:

1. `myvar DB 'a'`–משתנה מטיפוס byte בגודל בית אחד. הערך ההתחלתי שלו יהיה התו 'a'.
2. `Num1 DW 1645`– משתנה מטיפוס word בגודל שני בתים. הערך ההתחלתי שלו יהיה המספר 1645.

<u>הערות</u>:

1. כדי ליצור משתנה לא מאותחל, במקום לכתוב ערך התחלתי נכתוב סימן שאלה. לדוגמא:
   `numseconds DD ?` –משתנה מטיפוס dword בגודל 4 בתים. המשתנה לא מאותחל.
2. הערך ההתחלתי יכול להיות מספר בבסיס 2/10/16 באופן הבא:
   a. בסיס 10 – פשוט כותבים את המספר
   b. בסיס 2 – כותבים מספר בינארי ובסוף כותבים את האות b. לדוגמא: `bin DB 01010111b`
   c. בסיס 16 – כותבים את המספר בהקסדצימלי, <u>מוסיפים 0 בהתחלה (ה-0 הוא לא חלק מהמספר)</u>, ובסוף מוסיפים h. לדוגמא: `hexnum DW 0A134h`

שימוש במשתנים

כאשר נרצה להשתמש במשתנה, נוכל להשתמש ישירות בשם שלו. לדוגמא:

```
num1 dw 10
num2 dw 8
num3 dw ?

mov ax,num1
add ax,num2
mov num3,ax
```

נקודה חשובה מאוד! זיכרו שבשיעור הקודם דיברנו על סוגי אופרנדים – אוגרים, ערכים מיידיים (קבועים) וכתובות זיכרון. שימו לב ש"משתנים" הם **לא** סוג נוסף של אופרנד, אלא רק **"שם יפה לכתובת זיכרון"**. כלומר כאשר האסמבלר עובר על התכנית ובונה ממנה את קובץ ההרצה (למשל קובץ exe), הוא מחליף את שמות המשתנים בכתובות שלהם בתוך התכנית. זאת אומרת האסמבלר יהפוך את השורה mov ax,num1 למשהו כמו mov ax,ds:[120h].

שימוש בכתובות של משתנים

לפעמים נרצה להשתמש בכתובת של משתנה גלובאלי שהגדרנו בתכנית. בשביל זה קיימת ההוראה offset. שימו לב ש-offset היא לא פקודה של המעבד, אלא *הוראה* לאסמבלר עצמו (כלומר "ביצוע ההוראה" מתרחש בזמן "קומפילציה" של התכנית ולא בזמן ריצה). לדוגמא:

```
mov ax,offset num1
```

כאשר האסמבלר "יקמפל" את התכנית, הוא יחליף את offset num1 **בכתובת** של המשתנה num1 בזיכרון (שוב שימו לב שזה אפשרי רק כי המשתנה הוא גלובלי והכתובת שלו נקבעת בזמן "קומפילציה"). שורה זו בעצם תתורגם ע"י האסמבלר למשהו כמו:

```
mov ax,120h
```

כאשר 120h היא הכתובת של המשתנה num1 בתכנית.

<u>דוגמא נוספת:</u>

```
temp dw 123

mov ax,temp
mov bx,offset temp
```

תכנית זו תכניס לתוך ax את ה**תוכן** של המשתנה temp (כלומר את הערך 123) ותכניס לתוך האוגר bx את **כתובת** המשתנה temp בזיכרון.

<u>הערה:</u> אם אתם זוכרים, אז בשיעור הקודם דיברנו על כך שכתובות באסמבלי x86 מורכבות מכתובת ממקטע ומהיסט בתוך המקטע. למען הדיוק, ההוראה offset מתייחסת רק להיסט של המשתנה ביחס לתחילת המקטע (ומכאן גם שמה). אולם מכיוון שבתכנית שלנו, כל המקטעים קבועים ומתחילים באותו מקום, אנחנו מתעלמים מפרט זה למען הפשטות, ומתייחס להיסט בתור הכתובת כולה.

קבועים equ

בדומה להוראת #define בשפת C, המאפשרת להגדיר קבועים בזמן קומפילציה (או ליתר דיוק בזמן preprocessing), בשפת אסמבלי ניתן להגדיר קבועים ע"י שימוש בהוראה equ באופן הבא:

```
NUMPLAYERS EQU 3

mov ax, NUMPLAYERS
```

שימו לב שבניגוד למשתנים, שימוש ב-equ <u>לא גורם להקצאת זיכרון בכלל,</u> אלא רק מצהיר על קבוע שהאסמבלר מחליף בערך שלו בזמן ה"קומפילציה".

<u>דוגמא:</u> כדי להבין טוב יותר למה הכוונה בזה ש-equ לא גורם להקצאת זכרון בכלל, נסתכל על הדוגמא הבא, ונניח שתחילת התכנית בכתובת 100h בזיכרון.

```
num1 dw 34

num2 dd 300

CHAR.VALUE EQU 'C'

char db CHAR.VALUE
```

אם תחילת התכנית היא בכתובת 100h בזיכרון, היכן בזיכרון יישבו כל אחד מהמשתנים?

- המשתנה num1 נמצא ממש בתחילת התכנית, ולכן הוא יהיה בכתובת 100h בזיכרון והוא יתפוס 2 בתים.
- המשתנה num2 יופיע מיד אחרי המשתנה num1 (שתפס 2 בתים) ולכן הוא יהיה בכתובת 102h, ויתפוס 4 בתים.
- CHAR.VALUE הוא לא משתנה אלא קבוע, ולכן הוא לא יהיה בזיכרון בכלל (אלא רק ישמש את האסמבלי בזמן "קומפילציה")
- המשתנה char יופיע מיד אחרי המשתנה num2, ולכן הוא יהיה בכתובת 106h.

מערכים

ניתן להצהיר על מערכים במספר דרכים:

1. כתיבה של מספר ערכים התחלתיים מופרדים ע"י פסיקים תביא להגדרת מספר משתנים (מערך). לדוגמא:

   numarray dw 1,2,3,4,5

   תביא להגדרת מערך של wordים בן 5 תאים (ולכן גודלו הוא $5 \times 2 = 10$ בתים), שיאותחל בערכים 1,2,3,4,5.
2. שימוש במילה dup שמביאה לשכפול הערך ההתחלתי ויצירת מערך. לדוגמא:

   numarray dw 6 dup(0A123h)

   תביא להגדרת מערך של wordים בן 6 תאים (ולכן גודלו הוא $6 \times 2 = 12$ בתים), שיאותחל בערכים 0A123h בכל אחד מהתאים במערך.
3. שימוש במילה dup בתוספת הערך ההתחלתי ? תביא להגדרת מערך לא מאותחל. לדוגמא:

   numarray dw 100 dup(?)

   תביא להגדרת מערך <u>לא מאותחל</u> של wordים בן 100 תאים (ולכן גודלו הוא $100 \times 2 = 200$ בתים).