# קודגורו אקסטרים

## או שפת סף כמשחק

## תוכן עניינים


1. הקדמה ..... 2
2. על התחרות ..... 2
3. מנוע המשחק ..... 3
4. התחלה מחויכת ..... 6
5. חזרה לחוקי המשחק ..... 11
6. מרוב עצים רואים יער ..... 12
7. קצת תיאוריה ..... 14
7.1. אוגרים ..... 14
7.2. פקודות ..... 16
8. תרגילי כספות ..... 20
9. קריאת שורדים מתחרים ..... 21
9.1. המפציץ ..... 21
9.2. התותח ..... 22
10. תרגול ..... 25
11. טקטיקות מתקדמות ..... 25
12. תמיכה ..... 27

## 1. הקדמה

קודגורו אקסטרים הוא מנוע משחק על בסיס מעבד 8086 של חברת אינטל®.
חוברת זו (שנכתבת לקראת ועודכנה במהלך ואחרי כיתות הלימוד באורט סינגלובסקי לקראת קודגורו אקסטרים-6) מסבירה כיצד לשחק באופן הדומה לדרך בה ילדים לומדים לדבר – במקום ללמוד דקדוק תחביר ולקרוא את כל המילון לפני אמירת המילה הראשונה – נתחיל בדוגמאות בפרקים הזוגיים ורקע תיאורטי בפרקים האי-זוגיים.
איננו מניחים שום רקע מוקדם, אם כי היכרות עם שפות תכנות יכולה לעזור.

## 2. על התחרות

תחרות "השורד הדיגיטאלי" היא משחק המתרחש בזירה וירטואלית בין מספר תכניות.
כל תכנית מייצגת מתחרה. התכניות נכתבות בשפת אסמבלי 8086 סטנדרטי[1], נטענות כולן למרחב זכרון משותף ("הזירה"), ומורצות במקביל ע"י מנוע המשחק.
מטרת כל מתחרה היא להפטר מהמתחרים היריבים. התכנית המנצחת היא זאת ששורדת אחרונה בזירה.
הפגיעה בתכנית מתחרה נעשית ע"י פגיעה בקוד אותו התכנית מריצה.
כלומר "Fight Club" רק בגרסה הדיגיטלית שלו...

אם אתם חושבים שתוכלו ותרצו להשתתף בתחרות עצמה, הרשמו לאתר בכתובת
http://www.codeguru.co.il/Xtreme
ותקבלו עדכונים לגבי התחרות.

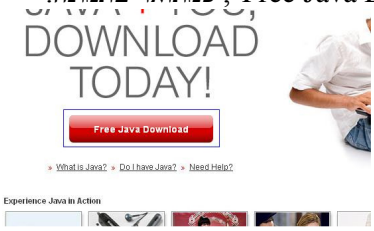
[1] עם שינויים קלים שיתוארו בהמשך.

## 3. מנוע המשחק

מנוע המשחק נכתב בשפת JAVA, ולכן על מנת להריץ את מנוע המשחק, נזדקק לתמיכה בשפת JAVA. על מנת לדעת אם יש למחשב שלכם גרסא מעודכנת, הריצו CMD (Start – run – cmd) וכתבו "java -version" הגרסא הנחוצה היא 1.50 ומעלה. אם אין לכם גרסא מספקת, יש להתקינה.
על מנת לעשות זאת ניכנס לאתר

http://www.java.com

ונלחץ על הכפתור "Free Java Download", כמתואר בתמונה:




לאחר שנוריד ונריץ את תוכנית ההתקנה של Java, יטען המסך הראשון של תוכנית ההתקנה:

על מנת להתחיל בהתקנה נלחץ על כפתור ה "Install", לאחר מכן תוכנית ההתקנה תתקין את מנוע ה Java, תהליך זה עלול להמשך דקה או יותר, בסיום (מוצלח) של ההתקנה יטען לפנינו המסך הבא:

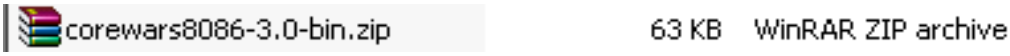
במידה וניתקלתם בבעיה במהלך ההתקנה, ניתן לבקש עזרה בפורום קודגורו אקסטרים.

לאחר מכן נוריד את מנוע המשחק, בלעדיו לא נוכל להריץ את השורדים שלנו. את המנוע של קודגורו אקסטרים נוכל להוריד מאתר:

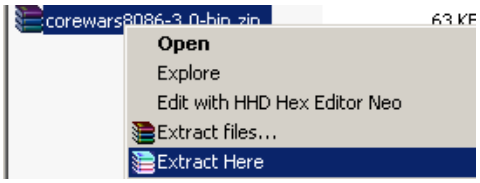
http://codeguru.co.il/Xtreme/tech1.htm

כפי שניתן לראות בתמונה נבחר להוריד את קובץ ה Zip המכיל את קבצי מנוע המשחק:

את קובץ ה Zip ובו קבצי ההתקנה נוריד ונשמור לתיקיה במחשב שלנו:




אם נפתח את ה ZIP ע"י פעולת extract

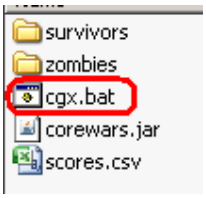



נוכל לראות תיקיה בשם CoreWars8086 :

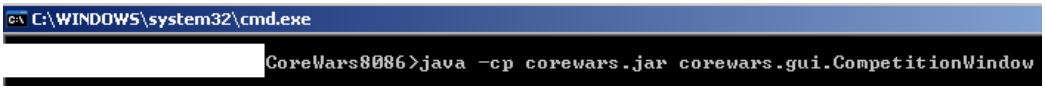
ובה הקבצים הבאים:

| Name | Size | Type | Date Modified |
|---|---|---|---|
| survivors | | File Folder | 30/09/2006 01:15 |
| zombies | | File Folder | 30/09/2006 01:15 |
| cgx.bat | 1 KB | MS-DOS Batch File | 13/01/2006 20:09 |
| corewars.jar | 67 KB | Executable Jar File | 30/09/2006 01:15 |

לאחר מכן יהיה ניתן להריץ את מנוע המשחק ע"י לחיצה כפולה על קובץ ה bat בשם cgx.bat



שיפתח חלון console שאותו אין לסגור ונראה כך:




וגם חלון ובו מנוע המשחק להנאתכם:

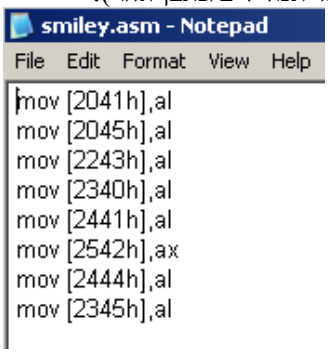
בחלון הזה (ובחלון נוסף שאפשר לפתוח בשם Show Session שמראה לנו את ריצת התוכניות בזירה) קורה כל האקשן.

## 4. התחלה מחויכת

"יום ללא צחוק הוא יום מבוזבז" אמר פעם צ'רלי צ'פלין ולכן נתחיל בכתיבת שורד פשוט שמצייר פרצופון מחייך.
התוכנית של השורד נראית כך:

```
mov [2041h], al
mov [2045h], al
mov [2243h], al
mov [2340h], al
mov [2441h], al
mov [2542h], ax
mov [2444h], al
mov [2345h], al
```

הפקודות שאתם רואים כאן כתובות בשפת אסמבלי 8086.
נעתיק את הפקודות של השורד ונכתוב אותן לתוך notepad (או כל מעבד תמלילים/כתבן אחר):




ונשמור את הקובץ בתור smiley.asm, חשוב לזכור שבחלון ה"שמירה בשם.." נבחר ב-"סוג הקובץ" ב "כל הקבצים *.*", וזאת על מנת שהסיומת "txt." לא תתווסף לסיומת שכבר בחרנו (שהיא "asm.").

לאחר שכתבנו את התוכנית עלינו לתרגם אותה לשפה שהמחשב שלנו מבין, מכיוון שמחשבים (עוד) לא מבינים שפת אנוש אנחנו צריכים לתרגם את הפקודות משפת אנוש לשפת מכונה.
דוגמה לתוכנה כזאת היא nasm, המסוגלת לתרגם את התוכנית משפת אנוש (הטקסט שכתבנו לקובץ smiley.asm ב notepad) לשפת מכונה.
את התוכנה ניתן להוריד, חינם, מהאתר http://www.nasm.us.
נכון לספטמבר 2011 הגרסא העדכנית ביותר היא:
http://www.nasm.us/pub/nasm/releasebuilds/2.09.10/win32

לאחר הלחיצה על הקישור נוכל להגיע לאתר שנראה כך:

**Index of /pub/nasm/releasebuilds/2.09.10/win32**

| Name | Last modified | Size | Description |
|---|---|---|---|
| Parent Directory | | - | |
| nasm-2.09.10-installer.exe | 15-Jul-2011 15:05 | 724K | |
| nasm-2.09.10-win32.zip | 15-Jul-2011 15:05 | 468K | |

*Apache/2.2.17 (Fedora) Server at www.nasm.us Port 80*

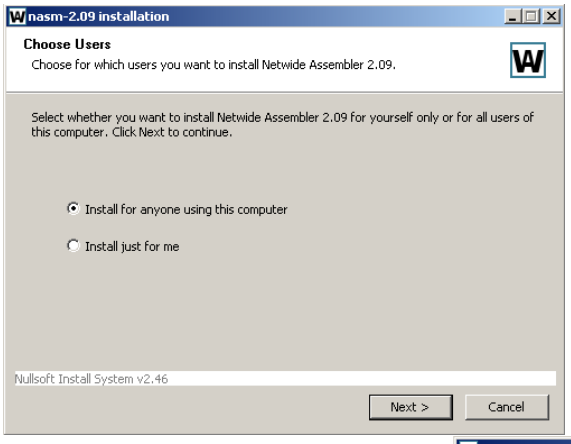
ובו יש קובץ התקנה המסומן במסגרת אדומה.

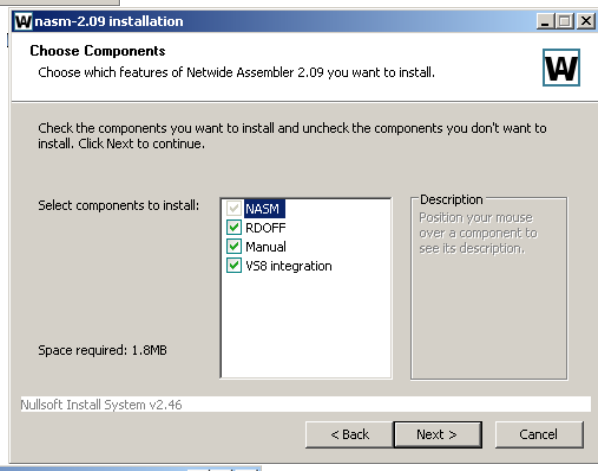
נוריד את קובץ ההתקנה לתיקיה במחשב:

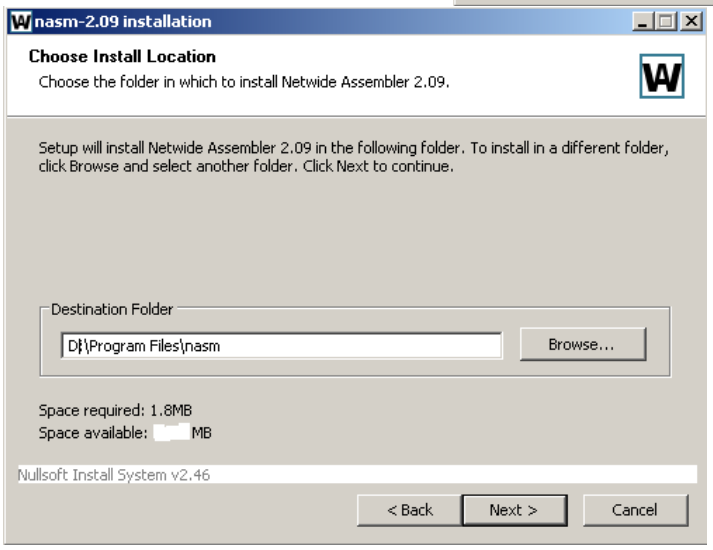
| Name | Size | Type |
| --- | --- | --- |
| nasm-2.09-installer.exe | 719 KB | Application |

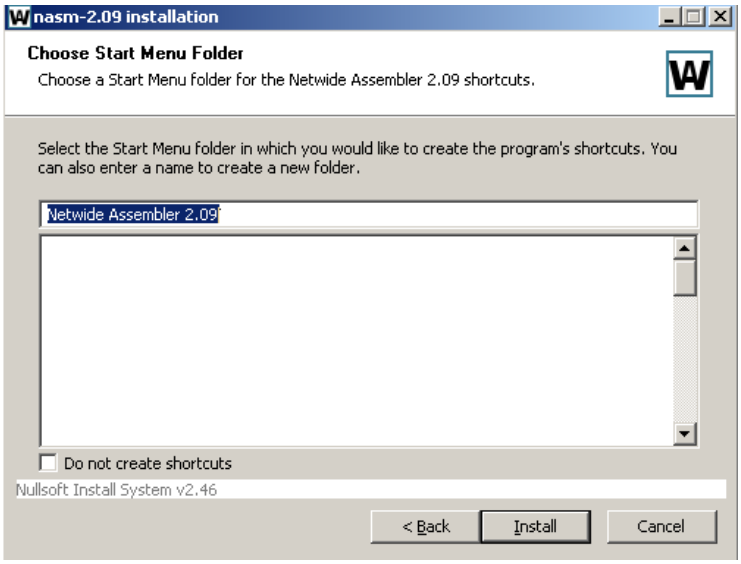
]

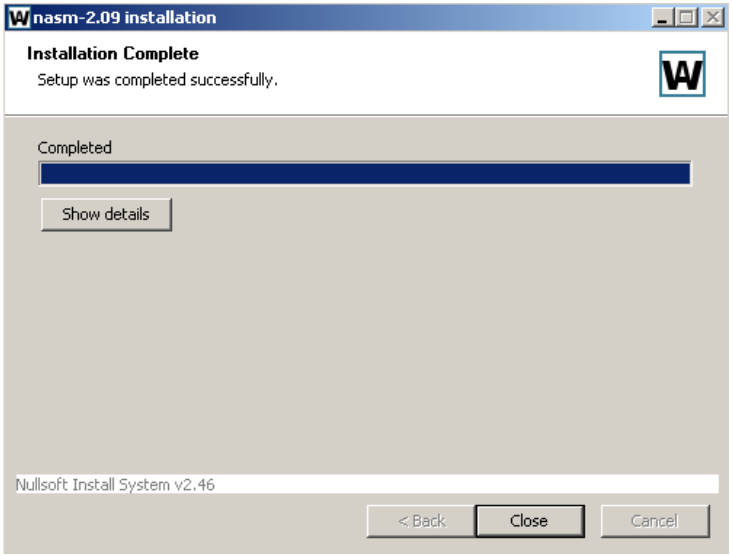
נלחץ על קובץ ההתקנה ונמשיך באשף ההתקנה לפי ההגדרות הבאות:






עכשיו, לאחר שההתקנה הסתיימה בהצלחה נוכל להריץ את nasm דרך הבעזרת קיצור דרך בשם nasm הנוצר על שולחן העבודה שלנו, על מנת להפעיל אותו יש ללחוץ עליו פעמיים.
על מנת לתרגם (לקמפל בשפה מקצועית) נקליד את הפקודה (מתוך החלון השחור שנפתח, כמובן):
nasm smiley.asm

כפי שניתן לראות בתמונה:

אם לא טעיתם בהקלדה, הרצת הפקודה תיצור קובץ בשם smiley.

| Name | Size | Type |
|---|---|---|
| contrib | | File Folder |
| rdoff | | File Folder |
| LICENSE | 2 KB | File |
| nasm-2.09-installer.exe | 719 KB | Application |
| nasm.exe | 573 KB | Application |
| nasm.ico | 3 KB | Icon |
| nasmdoc.pdf | 687 KB | Adobe Acrobat Document |
| nasmpath.bat | 1 KB | MS-DOS Batch File |
| ndisasm.exe | 260 KB | Application |
| Uninstall.exe | 63 KB | Application |
| smiley.asm | 1 KB | Assembler Source |
| smiley | 1 KB | File |

קובץ ובו פקודות האסמבלי
קובץ שנוצר לאחר התרגום

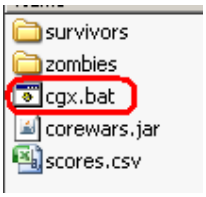
אם אתם מקבלים הודעת שגיאה, עברו שוב על הקובץ וודאו שלא טעיתם.
תכנות היא לעיתים משימה סיזיפית משום שמחשבים לא עושים את מה שאנחנו רוצים שהם יעשו אלא את מה שאנחנו אומרים להם לעשות; וכשזה לא אותו הדבר מתחילות צרות...

את הקובץ בלי סיומת ה asm (במקרה שלנו smiley) נצטרך לשים בתיקית ה survivors כדי שיוכל להשתתף במשחק. זוהי בעצם התוכנית המהודרת (מקומפלת) אותה יכול מנוע המשחק להריץ.

כשהמשחק מתחיל המנוע פונה לתיקיית survivors ומטעין לזירה את השורדים שנמצאים בתיקיה זו.
העתיקו את התוצאה פעמיים בשמות שונים אל מדריך השורדים (שוב, יש להקליד את הפקודות הללו מתוך חלון ה-nasm-shell):

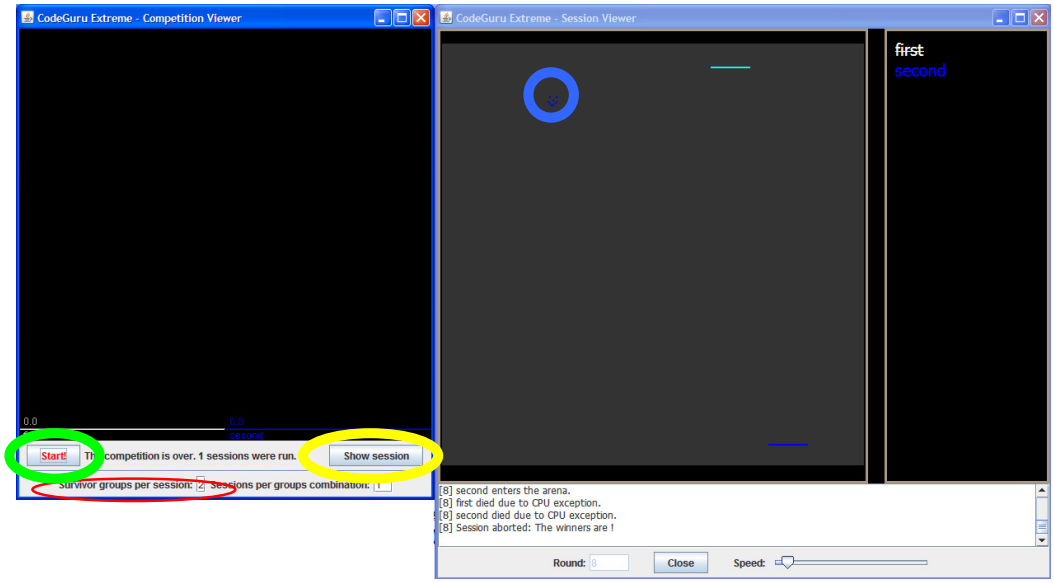
```
copy smiley survivors\first
copy smiley survivors\second
```

הריצו את המנוע המשחק ע"י לחיצה על קובץ cgx.bat:



כוונו את מספר השורדים (survivors group per session) ל-2 במקום ארבעה – ראו אליפסה אדומה בציור למטה;
בקשו להראות את הזירה (Show session) – ראו אליפסה צהובה בציור למטה;
והריצו (start) – ראו אליפסה ירוקה בציור למטה.

חדי העין יוכלו להבחין מייד בפרצופון מחייך – אותו בחרנו לסמן באליפסה כחולה:




כל הכבוד – הרגע הרצתם את תוכניתכם הראשונה!

## 5. חזרה לחוקי המשחק

בעצם מה שעשינו בפרק הקודם היה לכתוב תוכנית פשוטה בשפת סף (אסמבלי).
מנוע המשחק צובע נקודות (פיקסלים) על המסך בהתאם לפקודות התוכנית.
בתוכנית החייכן לעיל כל פקודה מציירת נקודה (פרט לאחת הפקודות שמציירת שתי נקודות, נגיע לכך בהמשך).
מנוע המשחק מריץ מספר תוכניות במקביל.
כדי לפשט את הדוגמא הרצנו את ה"שורד" (תוכנת החייכן) עם תוכנה נוספת שלא עושה דבר.

אפשר לראות כיצד הופיע החייכן (סמילי):

| | 40 | 41 | 42 | 43 | 44 | 45 | 46 |
|---|---|---|---|---|---|---|---|
| 20 | | 2041 | | | | 2045 | |
| 21 | | | | | | | |
| 22 | | | | 2243 | | | |
| 23 | 2340 | | | | | 2345 | |
| 24 | | 2441 | | | 2444 | | |
| 25 | | | 2542 | 2542 | | | |

לוח המשחק מכיל 256 שורות (00 עד FF בהקסה), כל שורה 256 פיקסלים.
סה"כ יש 65,536 נקודות על המסך, כל אחת מהן יש כתובת של ארבע ספרות הקסהדצימליות.

ניתן להתייחס לפי נתונים אלו אל הכתובות בלוח המשחק בצורה הבאה:

| | **00** | **01** | **..** | **..** | **FE** | **FF** |
|---|---|---|---|---|---|---|
| **00** | 0000 | 0001 | .. | .. | 00FE | 00FF |
| **01** | 0100 | 0101 | .. | .. | 01FE | 01FF |
| **:** | : | : | | | : | : |
| **:** | : | : | | | : | : |
| **FE** | FE00 | FF01 | .. | .. | FEFE | FEFF |
| **FF** | FF00 | FF01 | .. | .. | FFFE | FFFF |

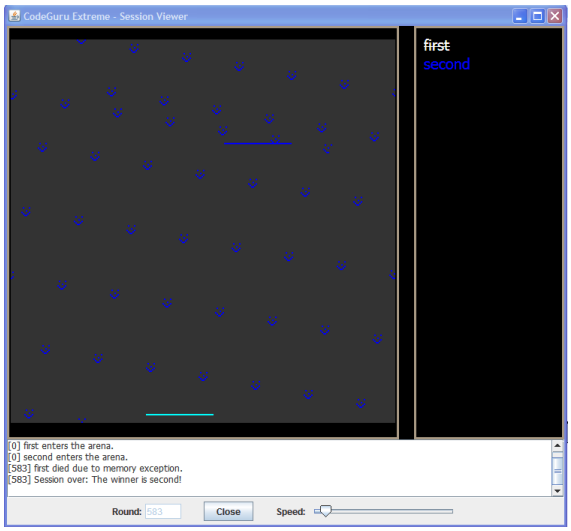
עכשיו ציירו על נייר משבצות תמונה אחרת (עץ, בית, שמש, השם שלכם, קבוצת כדורגל אהובה – כל העולה על דעתכם) וכתבו את התוכנית המתאימה שתצייר את אותו ציור על המסך.

## 6. מרוב עצים רואים יער

חייכן בודד קל יותר לצייר ידנית. אבל הכוח של שפת תכנות הוא האפשרות לשכפל.
ראו כיצד מתמלא הלוח כולו בחייכנים על ידי לולאה:

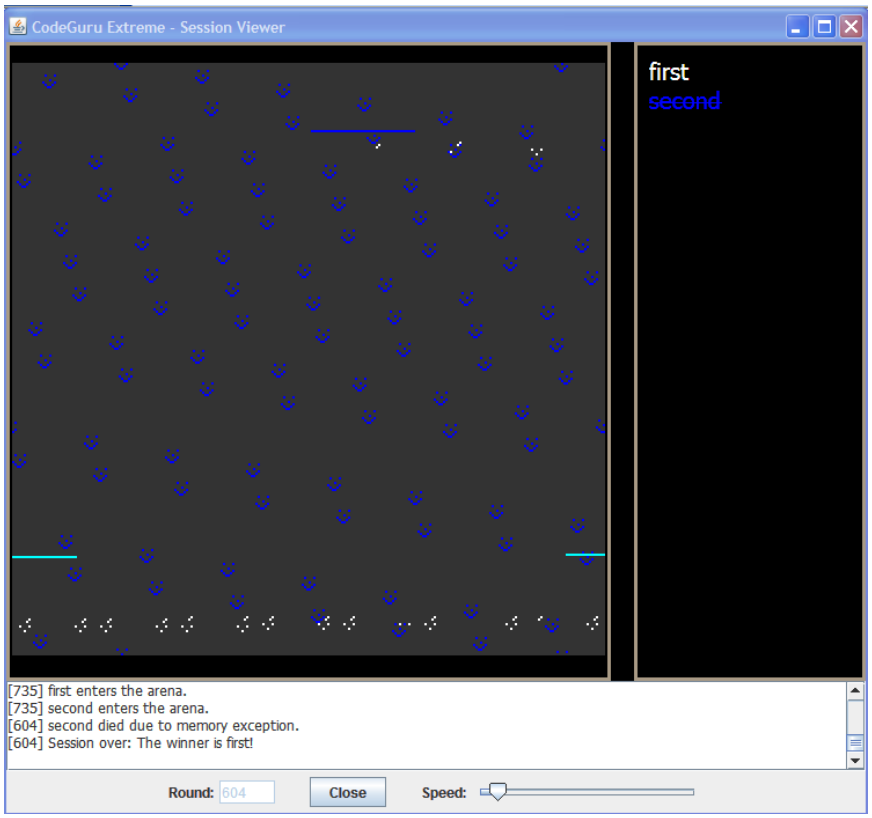
```
mov     cx, 100
l:
  call    smile
  add     bx, 623h
  loop    l
smile:
  mov     [bx+2041h], al
  mov     [bx+2045h], al
  mov     [bx+2243h], al
  mov     [bx+2340h], al
  mov     [bx+2441h], al
  mov     [bx+2542h], ax
  mov     [bx+2444h], al
  mov     [bx+2345h], al
  ret
```

והרי לנו יער של חייכנים:




שחקו עם הפרמטרים, וראו מה יוצא.

שימו לב שמדי פעם נופלות שגיאות שגורמות לכל מני צורות שונות ומשונות להופיע. למשל:




עוד נגיע לכך בהמשך; בנתיים רק דעו שאם לא קיבלתם את התוצאה המבוקשת, אתם יכולים לנסות שוב.

# 7. קצת תיאוריה

## 7.1. אוגרים

ועכשיו, כמובטח מפרק אי-זוגי, נתן קצת הסבר מסודר מה בעצם אנחנו עושים.
המרכיב הבסיסי של המידע במחשב הוא סיבית (bit בלעז). זוהי ספרה בינארית (יכולה לקבל שני ערכים אפס או אחד, הסיבה לכך מגיע מתחום החשמל, אך לא נרחיב עליה כעת).
שמונה סיביות מרכיבים בית (byte), דוגמאות להמחשה:

- [11111111]
- [10101011]
- [10001001]
- [10001]

ניתן לראות כי כמעט כל הדוגמאות מכילות שמונה סיביות כל אחת, הסיביות הללו יכולות להיות כבויות (כלומר להכיל את הסיפרה 0) או דולקות (להכיל את הסיפרה 1), הדוגמה האחרונה לא מכילה 8 סיביות, אך עדיין יכולה לייצג בית, הסיבה היא ששלושת הסיביות האחרונות החסרות, אשר משלימות את הדוגמה לבית (8 סיביות) מכילות אפסים, ולכן אין לכתוב אותם, למעשה הדוגמה האחרונה מקבילה לדוגמה הזו:

- [00010001] = [10001]

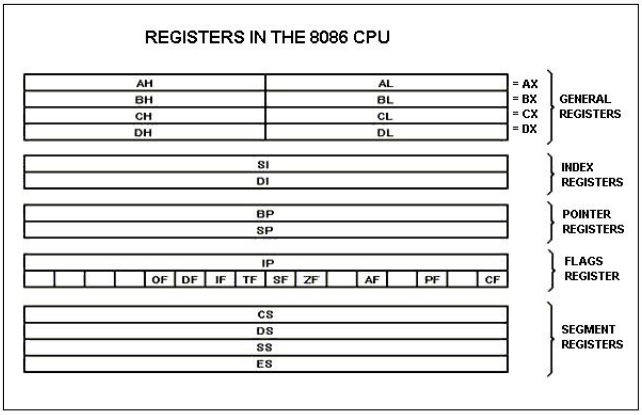
חשוב מאוד לזכור זאת על מנת למנוע בלבול מיותר.

מרחב הזיכרון של קודגורו אקסטרים הוא 64 קילובייט (64 כפול 1024, ז"א 65,536 בתים) המיוצג על המסך כ-65,536 פיקסלים (נקודות) בריבוע 256 על 256.

בנוסף, שיטה הספירה שבה נשתמש עלולה להיות שונה משיטת הספירה שהנך מכיר\ה, עד היום נהגנו לספור בשיטה העשרונית, הקרויה גם ספירה בבסיס 10 (האם אתם יכולים לשער למה?),
בתכנות (ולאו דווקא באסמבלי) נהוג לספור בשיטה ההקסה-דצימלית.
שיטה זו נקראת גם ספירה בבסיס 16, מכיוון שלהבדיל מהשיטה העשרונית, לאחר הסיפרה 9 תגיע הספרה a, ולאחר מכן f,e,d,c,b.
ניתן לקרוא על השיטה הזו בקישור הבא:

http://he.wikipedia.org/wiki/בסיס_הקסדצימלי

שפת המכונה עצמה כאמור מורכבת מאופ-קודים (באנגלית: opcodes), אלו תרגום של תוכנית האסמבלי שלנו בשפה שהמחשב יבין, כל פקודה שכתבנו בתוכנית שלנו מתורגמת לפקודה שהמעבד יבין.
פקודות אלו יטענו לזיכרון וירוצו אחת אחרי השניה עד שהתוכנית תופסק מסיבות מסויימות (לדוגמה, סיום התוכנית).
במעבד 8086 יש מספר אוגרים (בלעז רגיסטרים):



1. **האוגרים הכללים:**
   a. AX – אוגר כללי לחישובים (פקודת הכפל, למשל, עובדת רק עליו)
   b. BX – האוגר הכללי היחיד היכול לשמש כהזחה (offset) לכתובות
   c. CX – אוגר כללי בעל פקודות מיוחדות ללולאות
   d. DX – אוגר כללי היכול להתחבר לאוגר AX לקבלת ערך של 32 ביט
2. **אוגרי מחרוזות:**
   a. SI – משמש כ-Source Index לפקודות הזזה
   b. DI – משמש כ-Destination Index לפקודת הזזה
3. **אוגרי מחסנית:**
   a. BP – Base Pointer למחסנית ועוד
   b. SP – מצביע למחסנית, Stack Pointer
4. **אוגר הפקודות והדגלים:**
   a. IP – מצביע לפקודה המתבצעת ברגע זה Instruction Pointer
   b. Flags – תשעה ביטי דגל (נשא, זוגיות, נשא משני, אפס, סימן, דיבאג, פסיקה, כיוון וגלישה)
5. **אוגרי מקטע:**
   a. CS – אוגר מקטע המצביע לקוד
   b. DS – אוגר מקטע המצביע למידע
   c. ES – אוגר מקטע נוסף המצביע למידע
   d. SS – אוגר מקטע המצביע למחסנית

גודלם של אוגרים אלו הוא 16 סיביות, או 2 בתים, והם משמשים לאחסון מידע פרטי של התוכנית שלנו. חשוב להדגיש כי המידע הזה אינו משותף לשאר השורדים ואינו ניתן לשינוי על ידם, להבדיל מאזור הזיכרון המשותף בו נמצאים השורדים אשר נגיש אחד לשני, אוגרים אלו ישמשו אותנו בעת ריצת התוכנית.
לארבעת האוגרים הראשונים (לשימוש כללי) ניתן להתייחס כאל אוגרים בגודל של 8 סיביות, או בית אחד.
לדוגמה, לשמונת הסיביות הראשונות (נקראות גם הנמוכות) שבאוגר ax, נתייחס כאל al, ואל שמונת הסיביות שיבואו אחריהם (הנקראות הגבוהות) נתייחס כ ah.
אותו הדבר חל על שאר האוגרים, כלומר bx יכול להתייחס כ bl/bh, האוגר cx כ cl/ch, והאוגר dx כ dl/dh.
חשוב להבין כי אם נכתוב את הספרה 0fh לתוך הרגיסטר al, אזי תוכן הרגיסטר ax ישתנה כך ששמונת הסיביות התחתונות שלו יכילו את הספרה 0fh.

## 7.2. פקודות

פקודות בהם השתמשנו בתוכנית:

**label**
Label, טוב, זו היא לא ממש פקודה, אלא הגדרת אזור בקוד (סימניה), נגדיר אותו בעזרת מילה שנבחר (במקרה שלנו, smily או l) ונצמיד לה את התו ":" (כמובן, ללא המרכאות), נשתמש ב label זה בהמשך.

**mov**
הפקודה mov (קיצור של המילה האנגלית move) תשמש אותנו כאשר נרצה להעתיק ערך מסויים ממקום א' למקום ב', כאשר מקום ב' יכול להיות רגיסטר או כתובת זיכרון ממקום א', שיכול להיות גם כן אוגר או כתובת זיכרון, אבל יכול להיות גם כן ערך קבוע (ראו שורה ראשונה בתוכנית).
על מנת לציין שאנחנו משתמשים בכתובת זיכרון ולא בערך קבוע, נקיף את הכתובת בסוגריים מרובעיים (ראו שימוש ב mov בתוך אזור ה :smiley)

**call**
הפקודה call, תפקידה היא לקפוץ אל הlabel שהגדרנו, היא מקבלת כפרמטר את שם הlabel (במקרה שלנו smily).
על מנת לחזור מתוך label שנקרא על ידי call, נשתמש בפקודה ret.

**add**
הפקודה add תפקידה הוא לבצע חיבור של ערך מסויים עם ערך אחר, במקרה שלנו, הערך 623h(זוכרים מה מייצג h? סימון הקסה-דצימלי) יתווסף לערך שנמצא ברגיסטר bx.

**loop**
הפקודה loop מקבלת כפרמטר את הlabel אליו ניקפוץ בעת ביצוע הלולאה, כמו כן מונה הלולאה (מספר הפעמים שהיא תבוצע) נשמר ב cx, במקרה שלנו 100.

בטבלה רצ"ב צרפנו רשימה מלאה של כל פקודות המעבד 8086. לכל אחת מהפקודות הכחולות קיים קישור כדוגמת http://home.comcast.net/~fbui/intel_m.html#mul לקבלת תיאור.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| STC | CMPSW | JAE | JNC | JZ | MUL | REPNZ | STD |
| MOVSW | CWD | JB | JNE | LAHF | NEG | REPZ | STI |
| JS | JNBE | JBE | JNG | LDS | NOP | RET | STOSB |
| CMPSB | JA | JC | JNGE | LEA | NOT | RETF | STOSW |
| ADC | DEC | JCXZ | JNL | LES | OR | ROL | SUB |
| ADD | DIV | JE | JNLE | LODSB | SHR | ROR | TEST |
| AND | CMP | JG | JNO | LODSW | POP | MOVSB | XCHG |
| CALL | JNB | JGE | JNP | LOOP | POPF | JPO | XLAT |
| CBW | IRET | JL | JNS | LOOPE | PUSH | SAR | XOR |
| CLC | JPE | JLE | JNZ | LOOPNE | PUSHF | SBB | SHL |
| CLD | INC | JMP | JO | LOOPNZ | RCL | SCASB | REP |
| CLI | INT* | JNA | JP | LOOPZ | RCR | SCASW | MOV |
| CMC | JNAE | | | | | | |

*הפקודה INT מקבלת פרמטרים מסויימים היצוינו תחת הקטגוריה "טכניקות מתקדמות".

הסיבה שביער החייכנים בחרנו באוגר BX היא שהוא אוגר הכללי היחיד שמסוגל למען כתובת בצורה שרצינו. אפשר להעביר קבוע לאוגר; אפשר להעביר אוגר לכתובת קבועה בזיכרון; אפשר להעביר קבוע לכתובת שאוגר מצביע עליה; אבל אי אפשר להעביר קבוע לכתובת קבועה בזיכרון.
כדי להבין טוב יותר מה אפשר ומה לא, רצ"ב טבלה של כל צורות המיעון האפשריות

**8086 ADDRESS MODES**

| TYPE | INSTRUCTION | SOURCE | ADDRESS GENERATION | DESTINATION |
|---|---|---|---|---|
| 1) REGISTER | MOV AX, BX | REGISTER BX | | REGISTER AX |
| 2) IMMEDIATE | MOV CH,3AH | DATA 3AH | | REGISTER CH |
| 3) DIRECT | MOV [1234], AX | REGISTER AX | (DS x 10H) + DISPLACEMENT<br>10000H + 1234 | MEMORY 11234H |
| 4) REGISTER INDIRECT | MOV [BX], CL | REGISTER CL | (DS x10H) + BX<br>10000H + 0300H | MEMORY 10300H |
| 5) BASE PLUS INDEX | MOV [BX + SI], BP | REGISTER BP | (DS x 10H) + BX + SI<br>10000H + 0300H + 0200H | MEMORY 10500H |
| 6) REGISTER RELATIVE | MOV CL, [BX +4] | MEMORY 10304H | (DS x 10H) + BX + 4<br>10000H + 0300H + 4 | REGISTER CL |
| 7) BASE RELATIVE PLUS INDEX | MOV ARRAY [BX + SI], DX | REGISTER DX | (DS x 10H) + ARRAY + BX + SI<br>10000H+1000H+0300H+0200H | MEMORY 11500H |

ASSUME: BX = 0300H; SI = 0200H; ARRAY = 1000H; DS = 1000H.

## 7.3. המחסנית

המחסנית היא אזור זיכרון פרטי (בצורת ברירת המחדל) המשמש לאחסון נתונים החיוניים לריצת התוכנית שלנו.
המחסנית היא מיבנה נתונים אשר עובד צורת Last-In-First-Out (בקיצור LIFO), ואפשר להבין זאת בעזרת מחשבה על מחסנית אמיתית, כאשר הכדור הראשון אשר ידחף למחסנית יהיה הכדור האחרון שיצא ממנה, לעומת הכדור האחרון שידחף למחסנית והיה הכדור הראשון שיצא ממנה.

המחסנית תאחסן את הנתונים אשר נשמרו בה בזיכרון המחשב, וכמו שלמדנו עד כה, אנחנו נתבונן בזיכרון המחשב בצורת מיעון שונה במקצת ממה שהכרנו עד כה, אנחנו נעבוד בצורת segment:offset, כלומר, יהיה מקטע (segment) למחסנית, והיסט (offset) בו המחסנית תתחיל.

סגמנט המחסנית נשמר בגריסטר ss (קיצור ל-stack segment), חשוב לציין כי מנוע המשחק של קודגורו אקסטרים יאתחל לנו את הרגיסטר הזה לסגמנט פרטי אשר שורדים אחרים (שאינם מקבוצתנו, ועל קבוצות נדבר בהמשך) לא יוכלו לכתוב.

נקודת ההתחלה של המחסנית בתוך הסגמנט תקבע על ידי רגיסטר ההסט, שם הרגיסטר הזה הוא bp (קיצור ל base pointer), גם רגיסטר זה יאותחל אוטומטית על ידי מנוע המשחק.

בשפת אסמבלי, על מנת לידחוף נתונים למחסנים נשתמש בפקודה push, פקודה זה יכולה לקבל כאופרנד קבוע (לדוגמה, ABCDh), רגיסטר (כולם, לדוגמה, ax) וערך של כתובת.
חשוב לציין שגודל הערך חייב להיות WORD (כלומר, 16 סיביות), לפעמים האסמבלר יחייב אותנו להוסיף את מאפיין word לפני האופרנד, לדוגמה:

```
push ax
push word [bx]
```

בפקודה הראשונה המהדר יודע שמדובר במילה אך בפקודה השנייה הוא צריך את עזרתנו כדי להבין האם נרצה לדחוף למחסנית מילה או בית בודד; אחרת ניתקל בשגיאת קימפול.

על מנת לחלץ נתונים מהמחסנית נשתמש בפקודה pop, הפקודה הזו מקבלת כאופרנד רגיסטר או כתובת.
פקודה זו תמחק מהמחסנית את הערך האחרון שנדחף אליה ותכתוב אותו ברגיסטר או הכתובת שנבחרה.
כמובן שלאחר הרצת פקודה זו, הערך הקודם שהיה ברגיסטר או בכתובת שנבחרה, ימחק.

דוגמה כללית לאפשרויות שימוש במחסנית.

```
mov ax, 01100h
mov word [0ABCDh], 0ABCDh

push ax             ; pushing the value 01100h into the stack.
push word [0ABCDh]  ; pushing the value 0ABCDh into the stack.
push 0DEADh         ; pushing the value 0DEADh into the stack.

pop word [0ABCDh]   ; address 0ABCDh value is now 0DEADh.
pop ax              ; register ax value is now 0ABCDh.
pop cx              ; register cx value is now 01100h.
```

שימו לב שהדוגמא המחוקה בשורה השישית איננה חוקית במעבד 8086 (היא הוכנסה לשימוש רק במעבדי 80186 ואילך) ולכן איננה חוקית בקודגורו אקסטרים.

כאשר נדחוף למחסנית ערך בן 16 סיביות, היא תתחיל בכתובת [ss:bp] ותיגמר בכתובת [ss:bp+2], או במילים אחרות [ss:sp], הרגיסטר sp (קיצור ל-stack pointer) יצביע תמיד לראש המחסנית חיסור של bp עם sp יוכל להגיד לנו את גודל המחסנית הנתונה.
לצורך הדגמה:

```
push ax
push ax
; sp = bp + 4.
```

```
pop ax
pop ax
; sp = bp.
```

## 8. תרגילי כספות

אחר שציירנו קצת, נעבור לתרגילי כספות.
מקור השם הוא בתחרות הכספות של מכון ויצמן - בתחרות זו כל צוות מתכנן כספת המתבססת על עקרון פיזיקלי שמאפשר לפרוץ אותה (קל ולא מעניין לתכנן כספת שקשה לפתוח).
הבעיה המעניינת היא כיצד לתכנן כספת שניתן לפרוץ במהירות אך קשה לגלות זאת.
מידע על התחרות ניתן למצוא בקישור הבא:
http://www.weizmann.ac.il/zemed/activities.php?cat=0&id=571&act=large

נתחיל בדוגמא:
כיצד מנצחים את השורד הבא:

```
loop:
  mov     ax, [1234h]
  cmp     ax, 5678h
  jnz     loop
```

השורד הנ"ל קורא את תא הזיכרון 1234h ומשווה (CoMPare) את ערכו ל 5678h; אם הוא אינו שווה (Jump Not Zero)– הוא קופץ לתחילת התוכנית בלולאה אינסופית.
ברגע שכתובת הזיכרון 1234h יופיע הערך 5678h השורד בעצם "יתאבד" (ימשיך לבצע את הפקודה הבאה שלא כתבנו – ומנוע המשחק מאתחל את כל הזיכרון לפקודות לא חוקיות.

ומה עם השורד הבא:

```
loop:
  mov     ax, [1234h]
  add     ax, 6789h
  cmp     ax, 5678h
  jnz     loop
```

כאמור – אין בעיה לכתוב שורד שקשה לפגוע בו, למשל

```
loop:
  jmp     loop
```

התוכל לכתוב שורד דומה שאפשר לנצח? הביאו את השורדים שכתבתם לשיעור הבא ונבחן אותם יחדיו.

# 9. קריאת שורדים מתחרים

בתחרות עצמה לא תקבלו את קוד המקור של השורדים המתחרים, אלא את הגרסא המקומפלת שרצה במנוע. כדי להבין מול מה אתם מתמודדים עליכם לבצע תהליך של הנדסה הפוכה ( Reverse engineering) של השורדים. אחת הדרכים לעשות זאת היא על ידי הפקודה debug. זוהי פקודה שקיימת בכל מערכות ההפעלה החל מ DOS הישן והטוב. מריצים אותה על ידי

debug filename

שימו לב שמכיוון שהתוכנית כה עתיקה, היא תומכת רק בשמות 8.3 (עד שמונה תווים שם, נקודה, ועד שלושה תווים סיומת).

- עזרה ניתן לקבל על ידי ? (סימן שאלה).
- הפקודה U (Unassemble) מאפשרת להפוך את המידע של השורד לפקודות. זה שימושי כמובן להבנת שורדים מתחרים אבל גם כדי לפתח שורדים משלכם.
- הפקודה T (sTep) מריצה פקודה אחת.
- הפקודה R (Registers) מראה את מצב כל האוגרים.
- ואפשר גם לעדכן על ידי E (Edit), W (Write).
- ועל מנת לצאת – הפקודה Q (Quit).

דרך נוחה להפוך שורד לאסמבלי היא להריץ

```
debug sored > sored.asm
u
q
```

שימו לב שבזמן שתכתבו את הפקודות לא יציג המסך כלום, אל דאגה – בסיום יווצר הקובץ sored.asm אותו תוכלו לערוך (למחוק את החלקים המיותרים בתחילת כל שורה, לתקן דברים קטנים כגון הוספת h לקבועים וכד' כדי לקבל קובץ אסמבלי חוקי.

אפשר ורצוי להריץ את השורדים (גם שורדים שכתבתם) ב-debug לפני הרצתם במנוע המשחק.
חשוב להבין מה הם הבדלים בין המנוע ל-debug:

1. אוגרים
   - א. במנוע AX מכיל את כתובת הטעינה – ושאר האוגרים מאופסים.
   - ב. ב-debug CX מכיל את אורך השורד – ושאר האוגרים מאופסים.
2. אוגרי מקטע
   - א. במנוע DS ו-CS מצביעים על הזירה; ES מצביע על המקטע המשותף, ו-SS מצביע על המחסנית האישית.
   - ב. ב-debug כל האוגרים מצביעים לאותה הכתובת.
3. פקודות
   - א. המנוע מריץ רק את הפקודות החוקיות (8086, לא כולל פסיקות וגישה מחוץ לזירה).
   - ב. ב-debug אפשר לראות (Unassemble) רק פקודות 8086, אבל הרצה (sTep, Go) תריץ גם פקודות אחרות בלי להתלונן.

ועכשיו נחקור שני שורדים לדוגמא

## 9.1. המפציץ

מפציץ הוא מטוס קרב המטיל פצצות על מטרות קרקע (נעשה בהם שימוש בעיקר במלחמת העולם השניה).
במקרה שלנו זהו שורד שמפציץ את זירת המשחק בבית (byte) עם ערך 0xCC החל מתחילת הזיכרון (כתובת 0000).

מכיוון שהשורד מתחיל לכתוב בכתובת אפס, והוא נטען בכתובת מקרית, הרי שבסיכוי סביר הוא יפגע בעצמו לפני שיפגע באחרים. מה יקרה במקרה זה?

**מדוע בחרנו בערך 0xCC להפציץ איתו?**
בחרנו ב byte של 0xCC מאחר והוא מסמל " opcode רע" (INT3), ולכן אם נצליח לפגוע בקוד של שורד אחר בנקודה מתאימה, כנראה שנצליח להרוג אותו - ואין בקוד גורו אקסטרים "אי המתים" - כאן אם מועצת השבט אמרה את דברה זה סופי ומוחלט .. (לפחות עד הסיבוב הבא..).

נבחן את קוד האסמבלי של שורד זה:

```
; 1. initialization
      mov   al, 0CCh
      mov   bx, 0

; 2. bombing loop
bomb:
      mov   [bx], al
      inc   bx
      jmp   bomb
```

חלק האתחול מורכב משתי פקודות:
`mov al, 0CCh` – אנו מכניסים לאוגר AL את המידע בו נרצה להפציץ ובמקרה זה ה opcode הזדוני שלנו 0xCCh. בכל פעם שנבצע הפצצה נשתמש בתוכנו של אוגר זה.
`mov bx, 0` – אנו מכניסים לאוגר BX את הכתובת ההתחלתית שלנו, ובמקרה זה 0000.
בכל פעם שנבצע הפצצה אנו נקדם את ערכו של אוגר הכתובת BX, כדי להתקדם לתא הבא בזירה.

חלק הלולאה מורכב מהפקודות הבאות:
`bomb:` - פקודת label שמציינת שכאן מתחילה הלולאה, בכל חזרה על הלולאה נקפוץ למיקום זה ונבצע את הפקודות שאחריה.
`mov [bx], al` – אנו מעבירים את ערכו של AL לכתובת עליה BX מצביע.
`inc bx` – אנו מקדמים את BX לכתובת הבאה וזאת כדי שנוכל להפציץ מקום אחר בלוח.
`jmp bomb` – אנו מבצעים קפיצה ל label בשם bomb כדי לבצע חזרה נוספת על הפקודות.

## 9.2. התותח

תותח הוא כלי נשק חם הנועד להמטיר פגזים על אויב הנמצא במרחק ממנו.
במקרה שלנו זהו שורד מעט מתוחכם יותר משורד ה"מפציץ".
במקרה שלנו זהו שורד שמפציץ את זירת המשחק, במעגלים, בבית (byte) עם ערך 0xCC.

השורד אינו מפציץ את כל לוח המשחק ברצף כמו המפציץ שסקרנו קודם, אלא מפציץ עם קפיצות – כלומר יש רווח בין מקומות שאנו מפציצים אותם.

**אם אנחנו מדלגים על תאים בדרך, האם אין מצב שנפספס קוד של שורד יריב?**
התשובה היא לא. אנו מניחים שהחלק הפגיע בקוד שורד היריב יתפרס על יותר מ byte אחד ולכן עדיין נפגע בו; ואם לא בסיבוב הראשון, הרי שבסיבובים הבאים.

**האם לא נפגע בקוד של השורד שלנו כמו שקרה בשורד ה"מפציץ"?**
התשובה היא לא. השורד הזה יותר מתוחכם משורד המפציץ בתכונה נוספת – הוא נמנע מלהפציץ את הקוד שלו.
הוא עושה זאת ע"י כך שערכו של ה offset הראשון יהיה המקום לאחר הקוד שלו עצמו, ובמידה ולאחר סיבוב כשהוא חוזר לאזור בו הקוד שלו נמצא הוא קופץ ומדלג על הקוד שלו עצמו.

נבחן את קוד האסמבלי של שורד זה:

```
; 1. initialization
start:
     mov  bx, ax
     add  bx, (end - start)
     mov  al, 0CCh

; 2. bombing loop
bomb:
     mov  [bx], al
     add  bx, 8
     jmp  bomb
end:
```

חלק האתחול מורכב הפקודות הבאות:
`start:` – פקודת label שמציינת את תחילת קטע האתחול. אנו נשתמש ב label זה גם כדי לציין את מקום תחילת השורד שלנו בזיכרון.
`mov bx, ax` – בעת טעינת השורד לזירה, האוגר AX מכיל את הכתובת אליה השורד נטען. אנו מעבירים ערך זה לאוגר BX.
`add bx, (end - start)` – אנו מוסיפים לאוגר BX את אורך קטע הקוד שתופס השורד; לאחר הוספה זו אוגר BX יכיל את חישוב ה offset הראשון אחרי הקוד של השורד – לפני שעדכנו את BX היה בו את המיקום של השורד והוספנו את גודל השורד מכאן שקבלנו עכשיו את המיקום לאחר קוד השורד. שימו לב שלמרות שהפקודה נראית מסובכת, הרי שהאסמבלר מחשב את ההפרש בזמן האסמבלי ובשפת המכונה זו תוספת קבוע בלבד.
`mov al, 0CCh` – אנו מכניסים לאוגר AL את המידע בו נרצה להפציץ ,ובמקרה זה ה opcode הזדוני שלנו 0xCC. בכל פעם שנבצע הפצצה נשתמש בתוכנו של אוגר זה.

חלק הלולאה מורכב מהפקודות הבאות:
`bomb:` – פקודת label שמציינת שכאן מתחילה הלולאה; בכל חזרה של הלולאה נקפוץ למיקום זה ונבצע את הפקודות שאחריה.

`mov [bx], al` – אנו מעבירים את ערכו של AL לכתובת עליה האוגר BX מצביע.
`add bx, 8` – אנו מעדכנים את ערכו של אוגר BX המחזיק את כתובת ההפצצה ע"י כך שאנו מוסיפים לו 8.
מדוע שמונה בתים? מכיוון שזה יותר גדול מגודל החלק הרלוונטי של הקוד של השורד שלנו (שבעה בתים, איך לחשב זאת נראה אח"כ) וגם מתחלק בגודל לוח המשחק, כך שאחרי לולאה אחת סביב כל הכתובות נחזור בדיוק לאותה הכתובת וכך לא נדרוך על עצמנו גם בלולאות הבאות.
`jmp bomb` – אנו מבצעים קפיצה ל label בשם bomb כדי לבצע חזרה נוספת על הפקודות.
`end:` – פקודת label שנשתמש בה כדי לציין את סוף קוד השורד בזיכרון.

שלד לפרק תיאוריה נוסף

אחרי ששחקנו בשורדים אמיתיים; קראנו איך הם עובדים ויצרנו גרסאות שלהם; הגיע הזמן לרדת שלב ולהבין קצת יותר באמת למה מתרגם קוד האסמבלי שאנו כותבים ומדוע ולמה הוא פגיע.

הבה ניקח את התותח כדוגמא. נשתמש בתוכנה debug של DOS על מנת לראות מה הם הבתים אליהם היא מתרגמת:

```
0100 89C3          MOV     BX,AX
0102 83C30E        ADD     BX,+0E
0105 B0CC          MOV     AL,CC
0107 8807          MOV     [BX],AL
0109 83C308        ADD     BX,+08
010C EBF9          JMP     0107
```

הפקודה הראשונה הפכה לשני הבתים 89 ו-C3. אם נדרוך עליה בבית CC היא תהפוך לפקודה אחרת: אם הביית הראשון יכתב, נקבל Int3 המקווה; אבל אם הביית השני יכתב נקבל פקודה אחרת. במקרה שלנו mov sp,cx. מה יקרה עם השורד יפגע בכתובת 106? למעשה זה לא ממש משנה משום שהשורד שלנו מבצע את הפקודה דנן כפקודה ראשונה ויותר לא חוזר אליה – גם הפקודה תפגע, התותח אפילו לא ישים לב. לעומת זאת שבעת הבתים שבין 107 ל 10D הם החלק הפגיע.

פקודות קצרות יותר עוזרות להקטין את החלק הפגיע. כך, למשל, הפקודה

mov ax,0000

תופסת שלושה בתים ולעומתה הפקודה

sub ax,ax

תופסת רק שניים אבל מבצעת כמעט בדיוק אותו הדבר (ההבדל היחיד הוא בדגלים, אם זה חשוב).

חשוב גם להבין שפגיעה בפקודה מרובת בתים עשויה לגרום לתופעות אחרות מהמצופה. כך, למשל, אנו יכולים לכתוב 0CCCCh ולפגוע בשורד מתחרה, ובכל זאת לא לגרום לא להריץ פקודה לא חוקית. למשל אם נפגע בדיוק בפקודה mov ax,0000 בבתים השני והשלישי שלה ונהפוך אותה לפקודה החוקית mov ax,0cccch.

# 10. תרגול

מה עושה השורד הבא

...

# 11. טקטיקות מתקדמות

## 11.1 הפצצה כבדה

11.1.1 סוג זה של הפצצה יכתוב לכתובת הזיכרון המתחילה בערכם של es:di ול 255 הכתובות שלאחר מכן את הערכים הנמצאים באוגרים dx:ax (סה"כ יכתבו 256 בתים).

סדר הכתיבה הוא al ראשון, ah שני, dl שלישי ו- dh רביעי ואחרון, כיוון הכתיבה יקבע על ידי דגל ה-"כיוון" (direction flag) באוגר הדגלים. ערכם של es:di משתנה בהתאם לאחר הקריאה.

הקריאה לסוג זה של הפצצה יעשה על ידי האופקוד הבא (נקרא גם פסיקה):

```
int 86h
```

**חשוב:** יש לעדכן את האוגר es לאותו ערך הנמצא באוגר cs, במידה ולא יעשה השורד יפסל בעקבות שגיאת memory exception לאחר הקריאה לפסיקה.

11.1.2 דוגמא לשימוש בקוד:

```
; heavy bombing starts at 0:0 to 0:255.
push cs
pop es          ; set es as the code segment (main memory map)

xor di,di       ; di is our offset, xoring it with himself
                ; will set him to zero

mov ax, 0cccch  ; set invalid opcodes in ax.
mov dx, ax      ; ... and copy it to dx.
```

```
int 86h          ; execute interrupt 0x86.

jmp $            ; will keep us doing nothing & alive ;-)
```

שורד זה "יפציץ" את השורה הראשונה של מוניטור התחרות, ניתן לשחק בכיוון הכתיבה על ידי שינוי דגל ה-"כיוון", על ידי הפקודות CLD ו- STD.

### 11.2. הפצצה חכמה

11.2.1. הפצצה חכמה היא סוג נוסף של טקטיקת לחימה שהתווספה למשחק, הפצצה זו תחפש רצף של ארבע בתים (אשר יכולים להיות ארבע בתים של השורד המותקף) על גבי הסגמנט (64 קילו בית) הראשון ותחליף אותו עם רצף אחר של ארבע בתים, אשר יכול להיות רצף של פקודות לא חוקיות (0CCh) או הוראת קפיצה (jmp) לאזור הקוד שלנו, על מנת להרוויח את זמן המעבד של השורד המותקף, חשוב לזכור כי ההתקפה תחול על כול שורד, גם על שלנו, בהנחה וארבעת הבתים שחיפשנו ימצאו בקוד השורד, לכן יש לבחור רצף בתים יחודי לשורד אותו נרצה לתקוף.
על מנת להפעיל את סוג הפצצה זה נשתמש בפסיקה מספר 0x87, על ידי קריאה לפקודה:

```
int 87h
```

לפני הקריאה לפסיקה נאתחל את אוגר ה- es לאותו הערך המופיע באוגר ה cs.
יש גם לאתחל את di למספר (offset) ממנו נרצה להתחיל לחפש מתוך אותו סגמנט.
לאחר מכן יש לכתוב לתוך האוגרים ax:dx את ארבעת הבתים אותם נרצה לחפש בסגמנט, ובאוגרים bx:cx את ארבעת הבתים אשר יוחלפו לאחר מציאה.

חשוב לדעת כי לאחר הקריאה ערך es:di לא ישתנה, כך שלא ניתן לדעת את מיקום המופע של ארבעת הבתים שנמצאו, אם נמצאו בכלל בסגמנט.

8.2.2. דוגמא לשימוש בקוד:

```
push cs
pop es           ; set es as the code segment(main memory map)

xor di,di        ; di is our offset, xoring it with himself
                 ; will set him to zero

mov ax, 04187h   ; ax:dx is the byte string to find.
mov dx, 08820h

mov cx, 0cccch ; cx:bx is the byte string to replace with.
mov bx,cx

int 87h          ; execute interrupt 0x87.

jmp $            ; will keep us doing nothing & alive ;-)
```

**<u>הערות:</u>** ארבעת הבתים הנמצאים באוגרים ax:dx (הפקודה הרביעית והחמישית) הם אופקודים הנמצאים בדוגמת הקוד שהצגנו בפרק 6 "מרוב עצים רואים יער", הריצו שורד זה יחד עם דוגמאת הקוד ותראו כיצד השורד השני, המצוי בפרק 6, מפסיד מהר מאוד.

### 11.3. האצת שורד

11.3.1. במצב ריצה נורמאלי של מנוע המשחק, לכל שורד מוקצה חתיכת זמן ריצה זהה לכל שאר השורדים, לצורך העניין, חתיכת הזמן הזו זהה לזמן הרצת אופקוד בודד בקוד השורד, חשוב לזכור כי מנוע המשחק כל האופקודים הם בעלי אותו זמן ריצה.
האצת שורד היא שיטה מיוחדת הנוספה למנוע המשחק, המאפשר לשורד מסויים, מנצל שיטה זו, להריץ יותר מאופקוד בודד בחתיכת הזמן שהוקצתה לו, ובכח להשיג עליונות על שאר השורדים שאין מנצלים את השיטה, הוא יכול להשתמש בחתיכת הזמן מוגדלת

על מנת לתקוף שטחים גדולים יותר באזור המשחק, כמו כן על מנת לנסות טקטיקות לחימה אחרות.
קביעת רמת האצת השורד נעשת על ידי אוגר ווירטואלי בשם Energy, ערכו ההתחלתי הוא אפס (כלומר אין האצה) וכל 5 סיבובים במשחק (כלומר, 5 אופקודים שהשורד שלנו מריץ) יוחסר 1 מערכו של אוגר זה, אם ערכו הוא 0, לא יוחסר דבר (כלומר ישאר אפס).

הגדלת על אוגר ה Energy נעשת על ידי קריאה לאופקוד WAIT בדיוק ארבע פעמים, בצורה רציפה, מנוע המשחק יריץ את ארבעת האופקודים הללו בזמן של אופקוד אחד, חשוב לדעת כי קריאה לאופקוד WAIT פחות מארבע פעמים תגרום לשגיאה בקוד השורד, והוא יפסל.

כל רביעיית WAIT תגדיל את ערך האוגר Energy ב-1, מהירות השורד תחושב על ידי הנוסחא הבא:

Speed = log2(energy) + 1

### 11.4 שיתוף פעולה בין שורדים

11.4.1 מנוע המשחק מאפשר שיתוף פעולה בין שני שורדים, ולכן כל קבוצה תוכל לשלוח לתחרות שני שורדים אשר יוכלו לשתף פעולה על מנת לנצח את שאר השורדים בתחרות.
על מנת שמנוע המשחק ידע שמדובר באותה קבוצה של שורדים, יש להוסיף את הספרה 1 לסוף שמו של השורד הראשון, ואת הספרה 2 לסוף שמו של השורד השני, לדוגמה:

Rocky1
Rocky2

לכל שורד בקבוצה אוגרים, מחסנית אישית וזמן ריצה פרטיים, אך מובטח כי השורדים יטענו תמיד יחדיו לזירה (במקומות אקראיים), וכמו כן, ניקוד השורדים מאותה קבוצה יסוכם יחדיו בגרף הניקוד.
זו גם הסיבה שאין לקרוא לשורד בודד בשם המסתיים בספרה – מנוע המשחק יחפש את השורד השני של הקבוצה ו"יעוף" (אתם מוזמנים לתקן זאת בקוד המקור של המנוע).

לכל קבוצה מוגדר בלוק זיכרון משותף בגודל 1024 בתים, וחסום מפני שאר השורדים שמחוץ לקבוצה, הנועד להעברת הודעת בין השורדים, כתובת הבלוק הזה היא es:0000.

## 12. תמיכה

חשוב לציין כי בכל מקרה של שאלה\בעיה אתם מוזמנים לפנות ל
support@codeguru.co.il

כמו כן אפשר להעזר בפורומים שלנו בקישור
http://codeguru.co.il/cs/forums/5/showforum.aspx