פקודות בקרה - תנאים לוגיים

# cmp

פקודות בקרה

פקודות הבקרה יכולות להשוות בין שני ערכים, לקפוץ לנקודה מוגדרת בקוד בעקבות תוצאת ולחזור על פעולה כל עוד מתקיים תנאי מוגדר.

תפקידן של פקודות הבקרה הוא לאפשר קפיצה קדימה או אחורה אל שורות קוד בתכנית.

קפיצה קדימה – דילוג על שורות בתכנית, מאפשרת להריץ קוד כל שהוא רק אם תנאי מתקיים.

קפיצה אחורה – מאפשרת לחזור על הרצה של קוד מספר פעמים.

הוראות הבקרה מתחלקות לשני סוגים:

הוראות בקרה עם תנאי מתקיים (אם התנאי אמת, אז... קפוץ ל....)

והראות בקרה ללא תנאי (קפוץ ל....)

אבני בניין לפקודות בקרה

* יכולת להשוות בין שני ערכים

פקודת cmp

* יכולת "לקפוץ" לנקודה מוגדרת בקוד בעקבות תוצאת ההשוואה

סוגים שונים של פקודות קפיצה:

קפיצה בלתי מותנית jmp

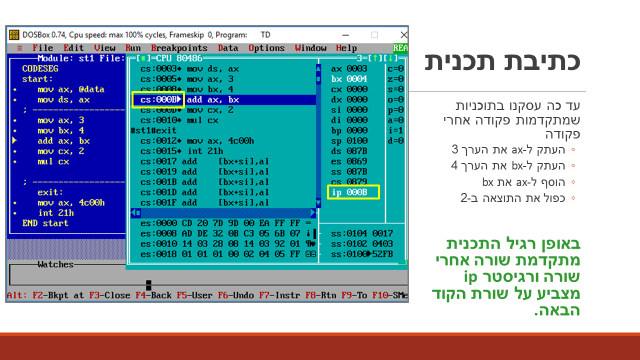
קפיצה מותנית jb, jbe, ja, jg, je, jne ועוד

* יכולת לחזור על פעולה כל עוד מתקיים תנאי מוגדר

פקודת loop

כתיבת תכנית

* עד כה עסקנו בתוכניות שמתקדמות פקודה אחרי פקודה
  + העתק ל-ax את הערך 3
  + העתק ל-bx את הערך 4
  + הוסף ל-ax את bx
  + כפול את התוצאה ב-2
* **באופן רגיל התכנית מתקדמת שורה אחרי שורה ורגיסטר ip מצביע על שורת הקוד הבאה.**



בדוגמה זו, ברגיסטר ax יש את הערך 3

ברגיסטר bx את הערך 4

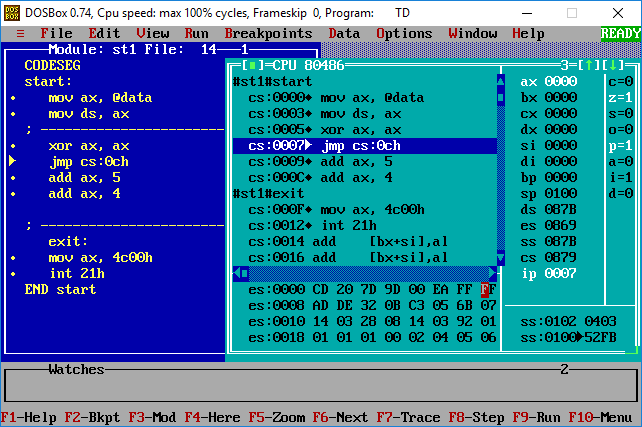
ורגיסטר ip מצביע על 0Bh (11d) הפקודה הבאה לביצוע ⇦ פקודת החיבור

# פקודת קפיצה לא מותנית - jmp

* קיצור של jump
* פקודת jmp שולחת את המעבד, ללא תנאי, לנקודה אחרת בתכנית.
* לאחר ביצוע פקודת ה-jmp ישתנה רגיסטר ה-IP וריצת התכנית תמשך מנקודה זו

בדוגמה שלפנינו ax קיבל את הערך 0 ⇦ רגיסטר ip קיבל את הערך 7 ⇦ הפקודה הבאה לביצוע.

הפקודה הבאה היא פקודת jmp אשר מדלגת על הפקודה של הוספת 5 ל -ax

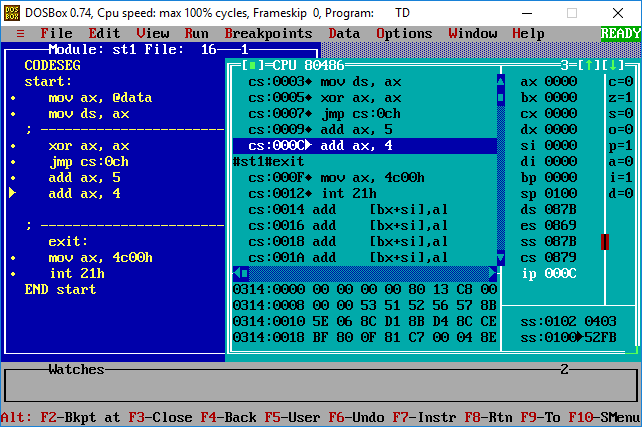


בתכנית שלפנינו דילגנו על הפקודה: add ax, 5

וקפצנו ישר לפקודה של add ax, 4.

רגיסטר ה – IP השתנה למיקום של הפקודה add ax, 4

רגיסטר ip קיבל את הערך 0ch ולכן בסוף התכנית ערך רגיסטר ax = 4.



שימו לב 🎔: שורות הקוד נכתבות בהזחה (באינדנטציה) של Tab בתחילת כל שורה,

והתוויות (labels) נכתבות תמיד בתחילת השורה.

קפיצה near ו-far

DATASEG

address dw 000Ah

CODESEG

mov ax, @data

mov ds, ax

mov ax, 1

jmp [address]

* קפיצה near- בתוך אותו סגמנט
  + אין צורך לציין את הסגמנט
  + לדוגמה:
* קפיצה far –לסגמנט אחר
  + חייבים לציין לאיזה סגמנט קופצים
  + jmp cs:000A
  + שימושית אם יש מסגמנט קוד אחד
  + בפועל לא נשתמש בה- תמיד נגדיר סגמנט קוד אחד

תוויות Labels

ברוב המקרים לא נפנה בפקודת jmp לכתובת בזיכרון.   
כל שינוי בקוד ישנה את מיקום הפקודות ב –code segment ואז המעבר יקפוץ למקום הלא נכון בתכנית. כדי למנוע את הצורך לעדכן את הכתובת של ה"קפיצה" בכל שינוי של הקוד.

**שפת אסמבלי מאפשרת לנו לתת תווית Label לשורה בקוד**

**jmp Start**

**jmp WaitForKey**

**Jmp PrintResult**

**רצוי שלתוויות יהיו שמות בעלי משמעות**

שם של תווית יכול להיות רצף של תווים כל עוד אינו מתחיל בספרה ואינו כולל רווח.  
האסמבלר אינו מבחין באותיות גדולות וקטנות אך תחילת מילה באותיות גדולות ושאר המילה בקטנות יותר קריאה ולכן יותר מומלצת

* מה יהיה ערכו של AX בסוף ריצת הקוד?
* xor ax, ax
* jmp IncAX
* add ax, 3
* IncAX:
* inc ax

AX יקבל את הערך 1 (0+1 = 1)

* מה יהיה ערכו של AX בסוף ריצת הקוד?
* IncAx:

inc ax

* jmp IncAx

יצרנו לולאה אין סופית המגדילה את ערכו של ax ב – 1 בכל פעם.

# פקודת cmp

* קיצור של compare
* מקבלת שני אופרנדים

cmp operand1, operand2

cmp אפורנד יעד, אופרנד מקור

פקודת השוואה משמשת להשוואה בין אופרנדים. הדרך בה המעבד בודק את היחסים בים האופרנדים היא ע"י פעולת חיסור.

* אם נדלק דגל האפס ⇦ האופרנדים שווים.
* הדלקה של דגלים אחרים (גלישה, נשא, סימן) מאפשרת למעבד לבדוק איזה אופרנד גדול יותר.

**איך המעבד מבצע cmp?**

* המעבד מבצע חיסור בין האופרנדים
* בניגוד לחיסור רגיל, התוצאה אינה מועתקת מה-ALU אל אופרנד היעד
* רק הדגלים מושפעים

ALU –יחידה אריתמטית לוגית

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **code** | **cf** | **zf** | **sf** |  | **הסבר** |
| mov ax, 3 |  |  |  |  | פקודת mov אינה משנה דגלים |
| cmp ax, 3 | 0 | 1 | 0 |  | 3-3 = 0 ⇨ zf = 1  המספרים שווים |
| cmp ax, 2 | 0 | 0 | 0 |  | 3-2 = 1 (מספר חיובי)  אופרנד היעד גדול יותר |
| cmp ax, 5 | 1 | 0 | 1 |  | 3-5 = -2 (מספר שלילי) ⇨ cf=1 sf=1  אופרנד המקור גדול יותר |

**דגל הנשא CF**: ביצוע פעולת חיסור בה המחסר (המספר שאותו מחסירים) גדול מהמחוסר (המספר ממנו מתבצע החיסור) דורש שימוש בנשא שלילי.

**דגל הסמן** **SF**: ערכו של דגל הסימן יהיה 1 אם הביט העליון (השמאלי ביותר) של התוצאה הנשמרת באופרנד היעד הוא - מספר שלילי . אחרת ערכו של דגל הסימן יהיה 0 - מספר חיובי, כאשר מתייחסים אליו כמספר מסומן Signed.

|  |  |  |  |
| --- | --- | --- | --- |
| **הפקודה** | **דגל האפס – ZF**  (1 כאשר התוצאה היא אפס) | **דגל הסימן- SF**  (1 כאשר הביט השמאלי של התוצאה הוא 1) | **דגל הנשא- CF**  (1 כאשר המחסר גדול מהמחוסר בייצוג unsigned) |
| mov ax, 3 | פקודת mov אינה משפיעה על מצב הדגלים | | |
| cmp ax, 3 | 1 | 0 | 0 |
| cmp ax, 2 | 0 | 0 | 0 |
| cmp ax, 4 | 0 | 1 | 1 |

צורת כתיבה חוקית של פקודת cmp.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **תוצאה** |  | **דוגמה** |  | **הפקודה** |
| שינוי מצב הדגלים בהתאם  ליחס בין האופרנדים | cmp | al, bl | cmp | register, register |
| cmp | al, [byteVar] | cmp | register, [memory] |
| cmp | [WordVar], cx | cmp | [memory], register |
| cmp | ax, 5 | cmp | register, constant |
| cmp | [WordVar], 5 | cmp | [memory], constant |

קפיצות מותנות

קפיצות מותנות מאפשרות לנו לבנות תכניות שיש בהן קבלת החלטות, לוגיקה וטיפול במצבים שונים ⇦ "אם מתקיים... אז בצע..."

קפיצות מותנות עובדות בדרך הבאה:

* לפני פקודת הקפיצה, מבצעים בדיקה השוואתית של שני אופרנדים באמצעות פקודת cmp.
* תוצאת פעולת ה־cmp תהיה קביעת הדגלים לפי היחס בין האופרנדים.
* פקודת הקפיצה בודקת אם דגל כלשהו, או כמה דגלים, מקיימים תנאי מוגדר. לדוגמה, האם דגל האפס שווה ל ־1 .
* אם התנאי מתקיים, הקוד קופץ לכתובת שהוגדרה על-ידי המשתמש. כתובת זו תצוין באמצעות label.
* אם התנאי לא מתקיים, המעבד ממשיך את ביצוע הפקודות לפי הסדר - כלומר, עובר לפקודה הבאה שלאחר פקודת הקפיצה.

השוואה בין שני מספרים

בביצוע פעולת השוואה בין שני אופרנדים. אם האופרנד הראשון גדול מהשני, בצע קפיצה

ערך האופרנד הראשון ⇦00000001b

ערך האופרנד השני ⇦ 10000001b

האם המעבד יבצע קפיצה?

הערך 10000001b יכול לייצג שני מספרים:

12910, כמספר unsigned

12710-, כמספר signed

כדי שהמעבד יידע אם לבצע קפיצה, אנחנו צריכים להורות לו מהו סוג ההשוואה המבוקש

התשובה אם המעבד יקפוץ או לא תלויה בפקודת הקפיצה!

צורת כתיבה חוקית של פקודת cmp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| תוצאה |  | דוגמה |  | הפקודה |
| שינוי מצב הדגלים בהתאם  ליחס בין האופרנדים | cmp | al, bl | cmp | register, register |
| cmp | al, [byteVar] | cmp | register, [memory] |
| cmp | [WordVar], cx | cmp | [memory], register |
| cmp | ax, 5 | cmp | register, constant |
| cmp | [WordVar], 5 | cmp | [memory], constant |

ריכוז פקודות jmp

|  |  |  |
| --- | --- | --- |
| כל פקודת jmp תחיל ב – J  N - not / E - equal  משמעות הפקודה | כל פקודת jmp תחיל ב – J  G - greater / L - less  מספרים Signed | כל פקודת jmp תחיל ב – J  A – above / B - below  מספרים Unsigmed |
| קפוץ אם האופרנד הראשון גדול מהשני | JG – Jump if Greater | JA – Jump if above |
| קפוץ אם האופרנד הראשון קטן מהשני | JL – Jump if Less | JB – Jump if Below |
| קפוץ אם האופרנד הראשון והשני שווים | JE – Jump Equal | |
| קפוץ אם האופרנד הראשון והשני שונים | JNE – Jump Not Equal | |
| קפוץ אם האופרנד הראשון גדול או שווה לאופרנד השני | JGE – Jump if Greater or Equal | JAE – Jump if Above or Equal |
| קפוץ אם האופרנד הראשון קטן או שווה לאופרנד השני | JLE – Jump if Less or Equal | JBE – Jump if Below or Equal |

# תרגילים

# פקודות בקרה - תנאים לוגיים

## cmp

## bg5

בדיקת האם הקשנו על מקש q:

|  |  |
| --- | --- |
| הפסיקה של המתנה למקש שומרת את ערך המקש לריגסטר al.  ההוראה cmp משווה האם הערך השמור ברגיסטר al = לאות q  je  א הערך שווה (=)  קופצים לתווית end\_game:  ממשיכים בפקודות על פי הסדר הרגיל. | ; waits for character   * mov ah, 0h * int 16h   ; check if user asks to quit   * cmp al, 'q' * je end\_game   . . . . . . .  end\_game:  . . . . . . . |

שמרו את התכנית שיצרתם בשיעור הקודם בשם חדש והמשיכו לעבוד בקובץ החדש.

## האם לצאת מהתכנית Quit?

בתכנית שכתבנו בשיעור קודם הזזנו את השחקן על המסך, בעזרת פקודות חיבור וחיסר.

לפני הפסיקה המחזירה את המסך לתצוגת טקסט ויציאה מהתכנית, הוסיפו תווית

end\_game:

end\_game:

; text mode

mov ax, 2h

int 10h

חזרו אחורה לתזוזה של השחקן בהוראות שלפני התזוזה האחרונה של השחקן.  
(לפני התזוזה של השחקן האדום).

נוסיף לפני קובץ פקודת אלו את התנאי שיבדוק מהו ערך המקש שהקשנו לאחר הפסיקה של המתנה למקש:

; waits for character

mov ah, 0h

int 16h

; check if user asks to quit

cmp al, 'q'

je end\_game

כדי שנדע באיזה מקום במשחק יש לנו את הבדיקה של האם הקשנו על מקש 'q'

נצבע את השחקן בפסיקה המציירת את השחקן לפני בדיקה זו בצבע לבן – 15.

שמרו והריצו.

בדקו את ההרצה פעם אחת, כאשר אתם רואים את השחקן הלבן, הקישו על מקש q ⇦ יציאה מהתכנית.

בדקו שוב את ההרצה, כאשר אתם רואים את השחקן הלבן, הקישו על כל מקש אחר ⇦ התנית מציגה עוד תזוזה של השחקן, ממתינים למקש ויוצאים מהתכנית.

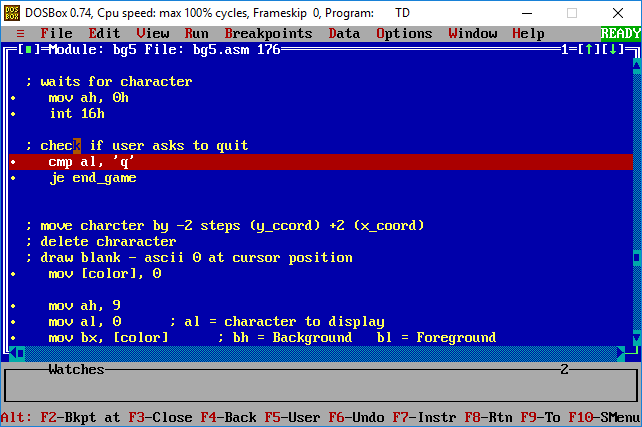
הריצו שוב את התכנית עם ה – TD,

רדו עם הסמן לשורה שבה רשומה פקודת ההשוואה.

cmp al, 'q'

בחרו עם העכבר בתפריט העליון במילה Breakpoints

השורה תהפוך לשורה אדומה.



זהו סימן לתכנית לעצור בהרצה בפקודה זו.

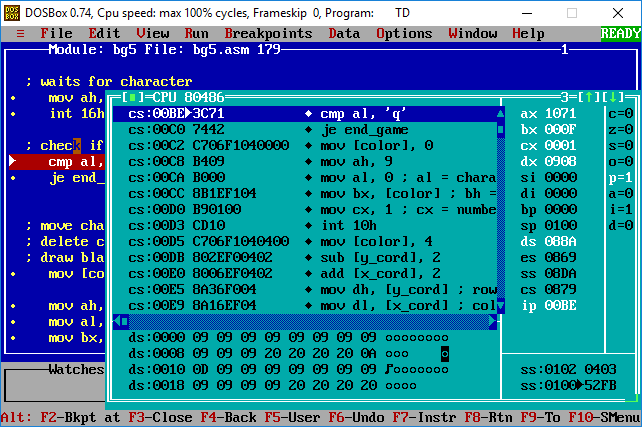
נריץ את התכנית עם F9.

כאשר נגיע לשחקן הלבן נקיש על q במקלדת

נחזור למסך ה – TD

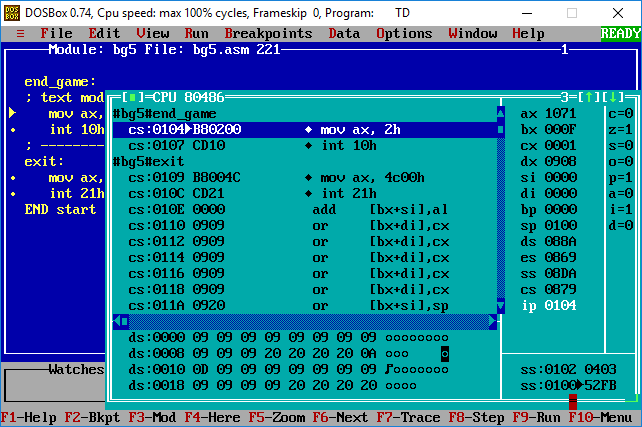
נצפנה בתצוגת המעבד, Alt + v ⇨ C

שימו לב לרגיסטר ה – IP ⇦ הוא קיבל את הערך של הפקודה לביצוע (פקודת ההשוואה)



נריץ עת הפקודה ע"י F8 ⇦ נעבור לפקודת הקפיצה ורגיסטר IP קפץ בשני ערכים

נריץ את פקודת הקפיצה ⇦ שימו לב לרגיסטר IP הוא קיבל את הערך של מיקום התווית end\_game



## קפיצה רחוקה

עכשיו ננסה להוסיף את הפקודה של אפשרות יציאה מהתכנית בתחילת המשחק.

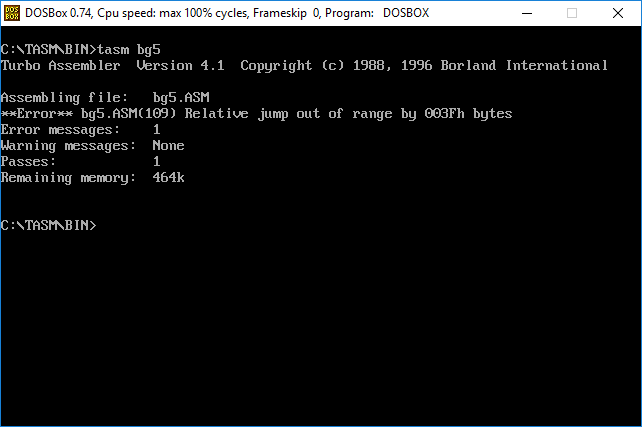
אחרי ציור לוח המשחק והופעת השחקן הראשונה.

הוסיפו לאחר הפסיקה של הקשה על מקש רווח את הבדיקה האם הקשנו על מקש q?

שמרו והריצו.

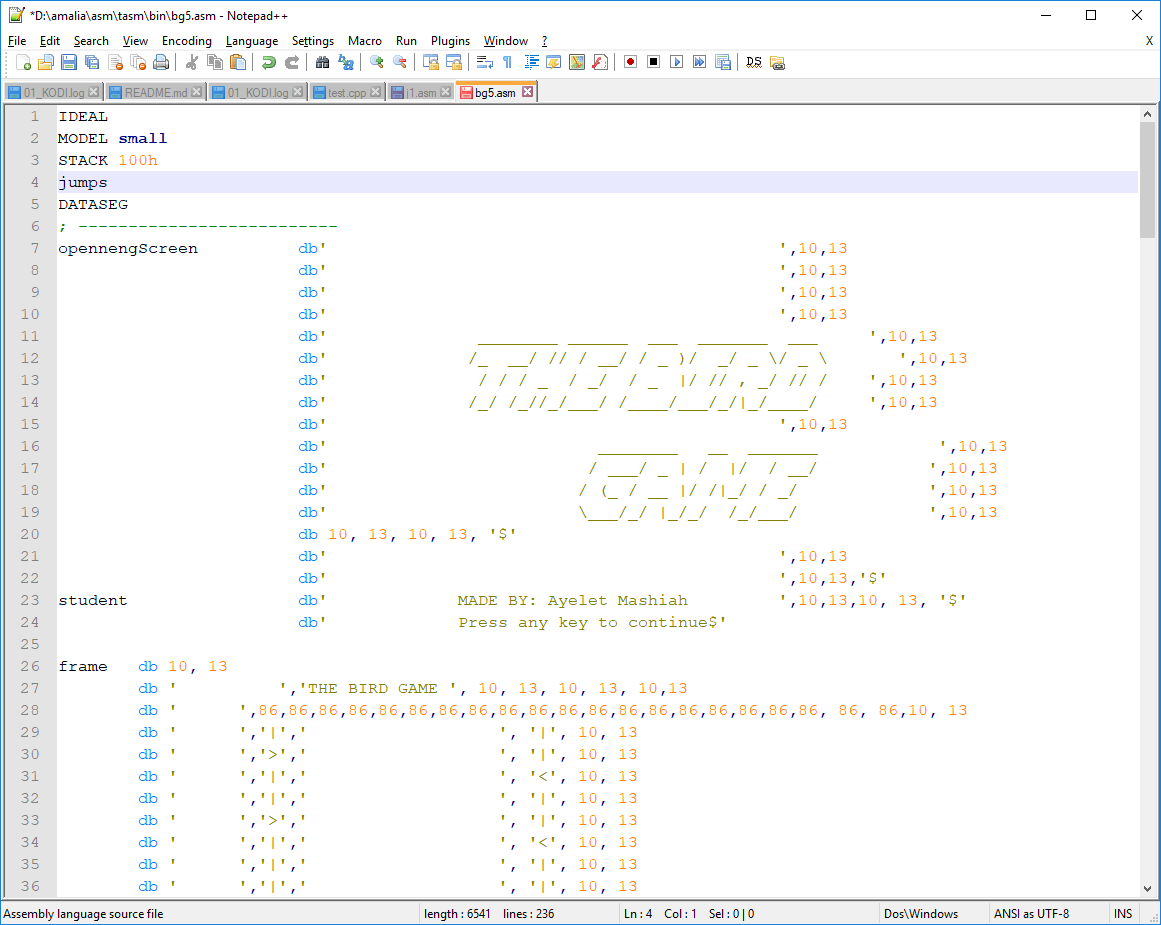
בהרצה של tasm כאשר הופכים את הקובץ לשפת מכונה, קיבלו הודעת שגיאה.

Relative jump out of range



הודעה זו אומרת לנו שהקפיצה שהאסמבלר צריך לעשות היא של שורות רבות מידי.

כדי למנוע בעיה זו נוסיף בתחילת הקובץ את הפקודה jmps המאפשרת לאסמבלי ליצור קפיצות גדולות.



הוסיפו את הפקודה

הריצו את התכנית.



בהמשך נלמד לכתוב את הקוד כך שלא תיווצר בעייה של long jump.

כרגע נשתמש בקפיצה לתווית "ביניים"

נשנה את שם התווית בתנאי לשם תווית הביניים

; check if user asks to quit

cmp al, 'q'

je jump1

נמקם בתכנית את תווית הביניים שלנו

מתווית הביניים נקפוץ לתווית ⇦ end\_game

כדי שתווית הביניים לא תפריע למהלך הפקודות שכבר יצרנו לפני תווית זו נעשה קפיצה לתווית אחרת שתדלג על שתי פקודות אלו.

**help\_jump:**

במהלך רגיל של הפקודות שנגיע לתווית

help\_jump נדלג על שתי ההוראות הבאות ונקפוץ לתווית: end\_help\_jump

jmp end\_help\_jump

**jump1:**

jmp end\_game

אם קפצנו לתווית jump1

נקפוץ לתווית: end\_game

**end\_help\_jump:**

נסו לשתול את "קפיצת העזר" אחרי ההדפסה הראשונה של השחקן הראשון למסך.

לפני הפסיקה של המתנה למקש.

בדקו האם הצלחתם?