

# מטלת מנהה (ממ"ז) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטרת פרויקט גמר

משקל המטרת : 31 נקודות (חוובה)

מספר השאלות : 1

מועד אחרון להגשה : 16.08.2020

סמסטר : 2020ב'

## קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
- שליחת מטלות באמצעות דואר אלקטרוני - באישור המנהה בלבד  
**הסבר מפורט ב"נווה הגשת מטלות מנהה"**

אחד המטרות העיקריות של הקורס "20465 - מעבדה בתכנות מערכות" היא לאפשר לסטודנטים בקורס להתנסות בכתיבת פרויקט תוכנה גדול, אשר יחקה את פעולתה של אחת מתוכניות המערכת השכיחות.

עליכם לכתוב תוכנת אסמבילר, עברו שפת אסמבלי שתוגדר בהמשך. הפרויקט יכתב בשפת C.

עליכם להגיש את הפריטים הבאים :

1. קבצי המקור של התוכנית שתכתבם (קבצים בעלי הסיומת c או h).
2. קובץ הרצה (מקומפל ומקשור) עברו מערכת אוביונטו.
3. קובץ makefile. הקימפול חייב להיות עם הקומפייל gcc והדגלים : -Wall -ansi -pedantic . יש לנפות את כל ההודעות שモוצאת הקומפיילר, כך שהתוכנית תתקמפל ללא כל העוראות או זהירות.
4. דוגמאות הרצה (קלט ופלט):
  - א. קבצי קלט בשפת אסמבלי, ובכפי הפלט שנוצרו מהפעלת האסמבילר על קבצי קלט אלה. יש להציג שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האסמביל.
  - ב. קבצי קלט בשפת אסמבלי המדגימים מגוון רחב של סוגי שגיאות אסמבלי (ולבן לא נוצרים קבצי פלט), ותדפסי המסקן ותדפסי המסתך.

בשל גודל הפרויקט, עליכם לחלק את התוכנית למספר קבצי מקור, לפי מישיות. יש להקפיד שקוד המקור של התוכנית יעמוד בקריטריונים של בהירות, קריאות וכתיבה נאה ומובנית.

Очיר מספר היבטים חשובים של כתיבת קוד טוב :

1. הפשטה של מבני הנתונים : רצוי (כל האפשר) להפריד בין הגישה למבנה הנתונים לבין השימוש של מבני הנתונים. כך, למשל, בעת כתיבת פונקציות לטיפול בטבלה, אין זה מעניינים של המשתמשים בפונקציות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקוורת.
2. קריאות הקוד : יש להשתמש בשמות שימושיים למשתנים ופונקציות. יש לעורוך את הקוד באופן מסודר : הזוחות עקביות, שורות ריקות להפרדה בין קטעי קוד, ועוד.
3. תיעוד : יש להכניס בקבצי המקור תיעוד תמציתי וברור, שיסביר את תפקידה של כל פונקציה (באמצעות הערות כותרת לכל פונקציה). כמו כן יש להסביר את תפקידם של משתנים חשובים. כמו כן, יש להכניס הערות ברמת פירוט גבוהה בכל הקוד.

הערה: תוכנית "עובדת", דהיינו תוכנית שמבצעת את כל הדרוש ממנה, אינה לכשעצמה ערובה לציון גבוה. כדי לקבל ציון גבוה, על התוכנית לעמוד בקריטריונים של כתיבה ותיעוד ברמה גבוהה, כמפורט לעיל, אשר משקלם המשותף מגע עד לכ- 40% משקל הפROYיקט.

יותר להשתמש בפרויקט בכל מגוון הספריות הסטנדרטיות של שפת C, אבל אין להשתמש בספריות חיצונית אחרות.

מומלץ לעבוד בזוגות. אין לעבוד בצוותים גדולים יותר. **פרויקט שיוגש על ידי שלשה או יותר, לא יבדק ולא יקבל ציון**. הבודרים להגיש יחד את הפROYיקט, יהיו **שייכים** לאותה **קבוצת הנחיה**. הציון יהיה זהה לשני הסטודנטים.

מומלץ לקרוא את הגדרת הפROYיקט פעמיים ראשונה ברכז, לקבלת תמונה כללית לגבי הנדרש, ורק לאחר מכן לקרוא שוב בפעם מעמיקה יותר.

### **רקע כללי ומטרת הפרויקט**

כידוע, קיימות שפות תוכנות רבות, ומספר גדול של תוכניות, הכתובות בשפות שונות, עשויות לזרז באוטומטית. כיצד "מכיר" המחשב כל כך הרבה שפות? התשובה פשוטה: המחשב מכיר למעשה שפה אחת בלבד: הוראות ונתונים הכתובים בקוד ביארוי. קוד זה מאוחסן בגוש בזיכרונו, ונראה כמו רצף של ספרות ביארוי. יחידת העיבוד המרכזי - היע"מ (CPU) - יודעת לפרק את הרצף הזה לקטעים קטנים בעלי משמעות: הוראות, מענים ונתונים.

למעשה, זיכרונו המחשב כולל הוא אוסף של סיביות, שנוהגים לראותן כמקובצות ליחידות בעלות אורך קבוע (בתים, מילימטרים). לא ניתן להבחין, בין שאינה מיומנת, בהבדל פיסי כלשהו בין אותו חלק בזיכרונו שבו נמצאת תוכנית לבין שאר הזיכרונו.

יחידת העיבוד המרכזי (היע"מ) יכולה לבצע מגוון פעולות פשוטות, הנקראות **הוראות מכונה**, ולשם כך היא משתמש באוגרים (registers) הקיימים בתוך היע"מ, ובזיכרונו המחשב. **דוגמאות**: העברת מספר מתא בזיכרונו לאוגר מסוים או בחרורה, הוספה 1 למספר הנמצא באוגר, בדיקה האם האסם מופיע באוגר שווה לאפס, חיבור וחיסור בין שני אוגרים, וכו'. הוראות המכונה ושילובים שלן הן המרכיבות תוכנית מסוימת כפי שהיא טעונה לזכרון בזמן ריצתה. כל תוכנית מקורה (התוכנית כפי שנכתבה בידי המתכנן), תורגמת בסופו של דבר באמצעות תוכינה מיוחדת לזרחה סופית זו.

היע"מ יודע לבצע קוד שנמצא בפורמט של **שפה מכונה**. זהו רצף של ביטים, המהווים קידוד בינאי של סדרת הוראות המכונה המרכיבות את התוכנית. קוד זה אינו קריא למשתמש, ולכן לא נוח לקודד (או לקרוא) תוכניות ישירות בשפת מכונה. **שפה אסטטבלי** (assembly language) היא שפת תוכנות מאפשרת לייצג את הוראות המכונה בצורה סימבולית קלה ונוחה יותר לשימוש. כמובן שיש צורך לתרגם את הייצוג הסימבולי לקוד בשפת מכונה, כדי שהתוכנית תוכל לפעול במחשב. תרגום זה נעשה באמצעות כלי שנקרא **אסטטבלר** (assembler).

כידוע, לכל שפת תוכנות עילית יש מהדר (compiler), או מפרש (interpreter), המתרגמים תוכניות מקור לשפת מכונה. האסטטבלר משמש בתפקיד דומה עבור שפת אסטטבלי.

כל מודל של יע"מ (כלומר לכל אירוגון של מחשב) יש שפת מכונה יעודית משלו, ובהתאם גם שפת אסטטבלית יעודית משלו. לפיכך, גם האסטטבלר (כלי התרגום) הוא יעודית ושונה לכל יע"מ.

taskido של האסטטבלר הוא לבנות קובץ המכיל קוד מכונה, מוקובץ נתן של תוכנית הכתובת בשפת אסטטבלי. זהו השלב הראשון במסלול אותו עברת התוכנית, עד לקבלת קוד המוכן לריצה על חומרת המחשב. **השלבים הבאים הם קישור (linking) וטעינה (loading)**, אך בהם לא נעסק במאין זה.

המשימה בפרויקט זה היא כתוב אסטטבלר (כלומר תוכנית המתרגם לשפת מכונה), עבר שפת אסטטבלית שנדיר כאן במיוחד לצורך הפרויקט.

**لتשומת לב:** בהסבירים הכלליים על אופן עבודה תוכנת האסטטבלר, תהיה מדי פעם התייחסות גם לעבודת שלבי הקישור והטעינה. התייחסויות אלה נועדו על מנת לאפשר לכם להבין את המשך תהליך העיבוד של הפלט של תוכנת האסטטבלר. אין לטעות: עליכם כתוב את תוכנית האסטטבלר בלבד. אין לכתוב את תוכניות הקישור והטעינה!!!

## המחשב הדמיוני ושפת האסטבלי

נדיר עתה את שפת האסטבלי ואת מודל המחשב הדמיוני, עבור פרויקט זה.  
הערה: תאור מודל המחשב להלן הוא חלק בלבד, ככל שנחוץ לביצוע המשימות בפרויקט.

"חומרה":

המחשב בפרויקט מורכב ממעבד (יע"מ), אוגרים (רגיסטרים), ו זיכרון RAM. חלק מהזיכרון משמש כמחסנית (stack).

למעבד 8 אוגרים כלליים, בשמות: 7, 6, 5, 4, 3, 2, 1, 0. גודלו של כל אוגר הוא 24 סיביות. הסיבית ה-0 ישמשת לציון כסיבית מס' 0, והסיבית המשמעותית ביותר במס' 23. שמות האוגרים נכתבים תמיד עם אות 'Z' קטנה.

כמו כן יש במעבד אוגר בשם PSW (program status word), המכיל מספר דגלים המאפיינים את מצב הפעולות במעבד בכל רגע נתון. ראו בהמשך, בתיאור הוראות המוכנה, הסברים לגבי השימוש בדגלים אלו.

גודל הזיכרון הוא  $2^{24}$  תאים, בכתביות 1-2<sup>24</sup>-0, וכל תא הוא בגודל של 24 סיביות. לתא בזיכרון נקרא גם בשם "מילה". הסיביות בכל מילה ממושפרות כמו באוגר.

מחשב זה עובד רק עם מספרים שלמים חיוביים ושליליים. אין תמייה במספרים ממשיים. האריתמטיקה נעשית בשיטת המשלים 2-complement (2'). כמו כן יש תמייה בתווים (characters), המוצגים בקוד ASCII.

מבנה הוראות המוכנה:

כל הוראה מוכנה במודול שלנו מורכבת מפעולה ואופרנדים. מספר האופרנדים הוא בין 0 ל-2, בהתאם לסוג הפעולה. מבחינת התפקיד של כל אופרנד, נבחן בין אופרנד מקור (source) ואופרנד יעד (destination).

כל הוראה מוכנה מוקודדת במספר מילות זיכרון רצופות, החל ממילה אחת ועד למקסימום שלוש מילים, בהתאם לשיטת המיעון בה נתון כל אופרנד (ראו פרטים בהמשך).

בקובץ הפלט המכיל את קוד המוכנה שבונה האסטבלי, כל מילה תקודד בסיס הקסדצימלי (ראו פרטים לגבי קבצי פלט בהמשך).

בכל סוג הוראות המוכנה, **המבנה של המילה הראשונה תמיד זהה**.  
מבנה המילה הראשונה בהוראה הוא כדלהלן:

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode				מייעון	מייעון	אוגר מקור	מקור	אוגר יעד	יעד	funct				A	R	E							

במודול המוכנה שלנו יש 16 פעולות, בפועל, למגוון שימושים לקודד יותר פעולות. כל פעולה מיוצגת בשפת אסטבלי באמצעות סימболים על ידי **שם-פעולה**, ובקוד המוכנה על ידי קומבינציה ייחודית של ערכי שני שדות במילה הראשונה של ההוראה: **קוד-הפעולה (opcode)**, ו**פונקציה (funct)**.

להלן טבלת הפעולות:

קוד-הפעולה (בסיסי עשרוני)	funct	שם הפעולה
0		mov
1		cmp
2	1	add
2	2	sub
4		lea
5	1	clr
5	2	not
5	3	inc
5	4	dec
9	1	jmp
9	2	bne
9	3	jsr
12		red
13		prm
14		rts
15		stop

הערה: שס-הפעולה נכתב תמיד באOTTיות קטנות. פרטים על מהות הפעולות השונות יובאו בהמשך.  
להלן מפרט הפעולות בamilha הראשונה בקוד המכונה של כל הוראה.

**סיביות 18-23:** סיביות אלה מכילות את **קוד-הפעולה** (opcode). ישן מספר פעולות עם קוד פעולה זהה (ראו בטבלה לעיל, קוד-פעולה 2, 5 או 9), ומה שבדיל ביניהן הוא השדה funct.

**סיביות 3-7:** שדה זה, הנקרא **funct**, מתייחס כאשר מדובר בפעולת שקוד-הפעולה (opcode) שהינה משותף לכמה פעולות שונות (כאמור, קוד-פעולה 2, 5 או 9). השדה funct יכול ערך ייחודי לכל פעולה מקבוצת הפעולות שיש להן אותו קוד-פעולה. אם קוד-הפעולה משמש ל פעולה אחת בלבד, הסיביות של השדה funct יהיו מאופסות.

**סיביות 16-17:** מכילות את מספרה של שיטת המיעון של אופרנד המקור. אם אין בהוראה אופרנד מקור, סיביות אלה יהיו מאופסות. מפרט של שיטות המיעון השונות ינתן בהמשך.

**סיביות 13-15:** מכילות את מספרו של אוגר המקור, במקרה שאופרנד המקור הוא אוגר. אחרת, סיביות אלה יהיו מאופסות.

**סיביות 11-12:** מכילות את מספרה של שיטת המיעון של אופרנד היעד. אם אין בהוראה אופרנד יעד, סיביות אלה יהיו מאופסות.

**סיביות 8-10:** מכילות את מספרו של אוגר היעד, במקרה שאופרנד היעד הוא אוגר. אחרת סיביות אלה יהיו מאופסות.

**סיביות 0-2 (השדה 'A,R,E'):** אפיון משמעותו של שדה זה בקוד המכונה יובא בהמשך. בamilha הראשונה של כל הוראה, ערך הסיבית A תמיד 1, ושתי הסיביות האחרות (R,E) מאופסות.

לתשומת לב: השדה 'A,R,E' מתייחס לכל אחת מהamilias בקידוד ההוראה (ראו המפרט של שיטות המיעון בהמשך).

## שיטות מייעו:

בשפת האסמבלי שלו קיימות ארבע שיטות מייעו, המסווגות במספרים 0,1,2,3. השימוש בחלק מסוית המיעו מצריך מילת-מידע נוספת בסוף המיעו, בונסף למילה הראשונה.

לכל אופרנד של ההוראה נדרש **כל היותר מילת-מידע אחת נוספת**. כאשר בהוראה יש שני אופרנדים הדורשים מילת-מידע נוספת, קודם תופיע מילת-המידע של אופרנד המקור, ולאחריה מילת-המידע של אופרנד השני.

כל מילת-מידע נוספת של ההוראה מקודדת באחד משלשה סוגי של קידוד. **סיביות 2-0** של כל מילת-מידע הן השדה 'A', המציין מהו סוג הקידוד של המילה. לכל סוג קידוד יש סיבית נפרדת, שערכה 1 אם מילת-המידע נתונה בסוג קידוד זה, ואחרת ערך הסיבית הוא 0.

- סיבית 2 (הסיבית A) מצינית שקידוד המילה הוא מוחלט (Absolute), ומינו מצריך שינוי בשלבי הקישור והטעינה.
- סיבית 1 (הסיבית R) מצינית שהקידוד הוא של כתובות פנימית הנחנית להזזה (Relocatable), ומצריך שינוי בשלבי הקישור והטעינה.
- סיבית 0 (הסיבית E) מצינית שהקידוד הוא של כתובות חיצונית (External), ומצריך שינוי בשלבי הקישור והטעינה.

הסביר על התפקיד של השדה 'E,R,A' בקוד המכוונה יבוא בהמשך.  
ערך השדה 'A,R,E' הנדרש בכל אחת משיטות המיעו מופיע בתיאור שיטות המיעו להלן.

מספר	שיטה המיעו	תוכן מילת-המידע נוספת הנוספת	אופן כתיבת האופרנד	דוגמה
0	מייען מיידי	מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר שלם בסיסי עשרוני. הסיביות 0-2 של מילת המידע הן השדה E. A,R,E. ערך הסיביות <b>A</b> הוא <b>1</b> , ושתי הסיביות <b>האחרות מאופסות</b> .	האופרנד מתחליל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בסיסי עשרוני.	mov #1, r2  בדוגמה זו האופרנד הראשון של הפוקודה הראשון (אופרנד המקור) נתון. בשיטת מייען מיידי. ההוראה כתובה את הערך 1 אל אונר 2.
1	מייען ישיר	מילת-מידע נוספת של ההוראה מכילה כתובות בזיכרון המילה בכתובת זו בזיכרון היא האופרנד. הכתובת מוצגת כמספר <b>לא סימן</b> ברוחב של 21 סיביות, בסיביות 23-3 של מילת המידע. הסיביות 0-2 במלילת המידע הן השדה E. A,R,E. ערך הסיביות האלה תלוי בסוג הכתובת הרשומה בסיביות 23-3. אם זהה כתובות שמייצגת שורה בקובץ המקור הנוכחי (כתבות פנימית), ערך הסיבית R הוא 1, ושתי הסיביות האחרות מאופסות. ואילו אם זהה כתובות שמייצגת שורה בקובץ מקור אחר של התוכנית (כתבות חיצונית), ערך הסיבית E הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד הוא <u>תוויות</u> שכבר הוגדרה, או שתוגדר בהמשך הכתובת התוויתית בתחילת השורה של הנקית 'data'. או השורה הבאה מגדירה את התוויות א':  x: .data 23  ההוראה: dec x  מקטינה ב-1 את תוכן x המילה שבכתובת x בזיכרון (ה"משתנה" x).  <u>דוגמה נוספת:</u> ההוראה jmp next  מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתבוצע נמצאת בכתובת next).  הכתובת next תקודד בסיביות 23-3 של מילת המידע נוספת.	shwraha .data 23  dec x  jmp next

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	אופן כתיבת האופרנד	דוגמה
2	מיון יחסית	<p> שיטה זו רלוונטיות אך ורק להוראות המבצעות קפיצה (הסתעפות) להוראה אחרת. מדובר בקוד-הפעולה הבאים בלבד : jsr, bne, cmp, jmp. לא ניתן להשתמש בשיטה זו בהוראות עם קוד-הפעולה אחרים.</p> <p><b> בשיטה זו, יש בקידוד ההוראה מילת מידע נוספת המכילה את מרחק הקפיצה, במילוי זיכרון, כתובות ההוראה הנוכחיות (פקודת הקפיצה) אל כתובות ההוראה המבוקשות (ההוראה הבאה לביצוע).</b></p> <p> מרחק הקפיצה מיוצג כמספר עם סימן בשיטת המשלימים 2-ברוחב של 21 סיביות, השוכן בסיביות 23-3 של מילת המידע הנוספת. מרחק זה יהיה שלילי במקרה שהקפיצה היא אל הוראה שכנות יותר נמוכה, וחוביי במקרה שהקפיצה היא אל הוראה שכנות יותר גבוהה. הסיביות 0-2 של מילת המידע הן השדה E. A,R,E. במיון יחסית, ערך הסיבית A הוא 1, ושתיי הסיביות האחרות מאופסות.</p>	<p> האופרנד מתחליל בתו &amp; ולאחריו ובצמוד אליו מופיע שם של תווית.</p> <p><b> התווית מייצגת באופן סימבולי כתובות של הוראה בקובץ המקור הנוכחי של התווית.</b></p> <p> ייתכן שהתווית כבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת התווית בתחילת שורת הוראה.</p> <p> יודגש כי בשיטת מיון יחסית לא ניתן להשתמש בכתובת (כתובת) שמוגדרת בקובץ מקור אחר (כתובת חיצונית).</p>	<p>jmp &amp;next</p> <p>בדוגמה זו, ההוראה מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתבוצע נמצאת בכתובת next). נניח כי ההוראה cmp שבדוגמה נמצאת בכתובת 500 (עשרוני). כמו כן, נניח כי התווית next מוגדרת בקובץ המקור הנוכחי בכתובת 300 (עשרוני). מרחק הקפיצה אל ההוראה בכתובת next הוא -200, ומרחיק זה יקודד בסיביות 3-23 של מילת המידע הנוספת.</p>
3	ישיר	האופרנד הוא שם של אוגר.	לשיטות מיון זו אין מילת מידע נוספת. מספרו של האוגר מקודד במילה הראשונה של ההוראה, בשדה המתאים : אוגר מקור/יעד.	<p>clr r1</p> <p>בדוגמה זו, ההוראה clr מספקת את תוכן האוגר r1.</p>

#### מפורט הוראות המכונה:

בתיאור הוראות המכונה נשתמש במונח PC (קיצור של "Program Counter". זהו אוגר פנימי של המעבד (לא אוגר כללי), שמכיל בכל רגע נתון את כתובת הזיכרון בה נמצאת ההוראה הקיימת שמתבצעת (הכוונה תמיד לכתובת המילה הראשונה של ההוראה).

הוראות המכונה מתחלקות לשולש קבוצות, לפי מספר האופרנדים הדרוש לפעולה.

**קבוצות ההוראות הראשונה:**  
אלן הוראות הדורשות שני אופרנדים.

ההוראות השייכות לקבוצה זו הן : mov, cmp, add, sub, lea

הוֹרָאָה	opcode	funct	הפעולה המתבצעת	דוגמה	הסביר הדוגמה
mov	0		מבצעת העתקה של תוכן אופרנד המקור (האופרנד הראשון) אל אופרנד היעד (האופרנד השני).	mov A, r1	העתק את תוכן המשתנה A (הmillion שבכטובות A בזיכרונו) אל אוגר 1z.
cmp	1		מבצעת השוואה בין שני האופרנדים. ערך אופרנד היעד (השני) מופחת מערך אופרנד המקור (הראשון), ללא שימרת תוצאת החישור. פועלות החישור מעדכנת דגל בשם Z ("דגל האפס") (PSW) יודלק, אחרת הדגל יאפס.	cmp A, r1	אם תוכן המשתנה A זהה לתוכנו של אוגר 1z אז הדגל Z ("דגל האפס") באוגר הסטטוס (PSW) יודלק, אחרת הדגל יאפס.
add	2	1	אופרנד היעד (השני) מקבל את תוצאת החיבור של אופרנד המקור (הראשון) והיעד (השני).	add A, r0	אוגר 0z מקבל את תוצאת החיבור של תוכן המשתנה A ותוכנו הנוכחי של 0z.
sub	2	2	אופרנד היעד (השני) מקבל את תוצאת החישור של אופרנד המקור (הראשון) מאופרנד היעד (השני).	sub #3, r1	אוגר 1z מקבל את תוצאת החישור של הקבוע 3 מתוכנו הנוכחי של האוגר 1z.
lea	4		lea הוא קיצור (ראשי תיבות) של load effective address זו מציבה את המعن זיכרונו המזיג על ידי התוויות שבאופרנד הראשון (המקורה), אל אופרנד היעד (האופרנד השני).	lea HELLO, r1	המען שמייצגת התוויות HELLO מוצב לאוגר 1z.

#### קבוצת ההוראות השניה:

אליהן ההוראות הדורשות אופרנד אחד בלבד. אופן הקידוד של האופרנד הוא כמו של אופרנד היעד בפקודה עם שני אופרנדים. השdotות של אופרנד המקור (סיביות 13-17) במילה הראשונה בקידוד ההוראה אינם בשימוש, ולפיכך יהוו מאופסים.

ההוראות השיקות לקובוצה זו הן : cmr, not, inc, dec, jmp, bne, jsr, red, prm

הוֹרָאָה	opcode	funct	הפעולה המתבצעת	דוגמה	הסביר הדוגמה
clr	5	1	איפוס תוכן האופרנד	clr r2	האוגר 2z מקבל את הערך 0.
not	5	2	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 הפוך ל-1 ולהיפך 1 ל-0).	not r2	כל בית באוגר 2z מתחפה.
inc	5	3	הגדלת תוכן האופרנד באחד.	inc r2	תוכן האוגר 2z מוגדל ב-1.
dec	5	4	הקטנת תוכן האופרנד באחד.	dec Count	תוכן המשתנה Count מוקטן ב-1.
jmp	9	1	קפיצה (הסתעפות) בלתי מותנית אל ההוראה שנמצאת במען המזיג על ידי האופרנד. כלומר, כתוכאה מביצוע ההוראה, מציביע התוכנית (PC) מקבל את כתובות יעד הקפיצה.	&Line	PC $\leftarrow$ PC + distanceTo(Line) מציביע התוכנית מקבל את המعن שמחושב על ידי חיבור המרחק לתוויות Line עם מען ההוראה הנוכחיית, ולפיכך ההוראה הבאה שתבוצע תהיה במען Line.
bne	9	2	mbn הוא קיצור (ראשי תיבות) של branch if not equal (to zero) זהה הוראות הסטעפות מותנית. אם ערכו של הדגל Z באוגר הסטטוס (PSW) היהנו 0, אז מציביע התוכנית (PC) מקבל את כתובות יעד הקפיצה. כזכור, הדגל Z נקבע באמצעות הוראות cmp.	Line	אם ערך הדגל Z באוגר הסטטוס (PSW) הוא 0, או PC $\leftarrow$ address(Line) מציביע התוכנית מקבל את כתובות התוויות Line, ולפיכך ההוראה הבאה שתבוצע תהיה במען Line.

הסבר הדוגמה	דוגמה	הפעולה המתבצעת	funct	opcode	הוראה
push(PC+2) PC $\leftarrow$ address(SUBR)  מצבייע התוכנית יקבל את כתובות התוויות SUBR, לפיכך, ההוראה הבאה SUBR. שתבצע תהיה במען כתובות החזורה מהשורה נשמרת במחסנית.	jsr SUBR	קריאה לשגרה (סברוטינה). כתובות ההוראה שאחרי הוראת זוז הנווכית (PC+2) נדחפת לתוך המחסנית שבזיכרונו (PC) המחשב, ומצבייע התוכנית (PC) מקבל את כתובות השגרה. <u>הערה</u> : חוזרת מהשגרה מתבצעת באמצעות הוראות זוז, תוך שימוש בכתובות שבמחסנית.	3	9	jsr
קוד האsci של התו הנקרא מהקלט ייכנס לאוגר r1.	red r1	קריאה שלתו מהקלט הסטנדרטי (stdin) אל האופרנד.		12	red
יודפס לפט התו הנמצא באופרנד, אל האוגר r1 הנמצא באוגר prn	r1 prn	הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).		13	prn

**קובוצת ההוראות השלישי:**  
אלו הן ההוראות ללא אופרנדים. קידוד ההוראה מורכב ממילה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד (סיביות 17-8) בambilת הראשונה של קידוד ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השינויים לקובץ זו הן : rts, stop

הסבר הדוגמה	דוגמה	הפעולה המתבצעת	opcode	הוראה
PC $\leftarrow$ pop()  ההוראה הבאה שתבוצע תהיה זו שאחרי הוראת jsr שקרה לשגרה.	rts	מתבצעת חוזרת משיגרה. ערך בראש המחסנית של המחשב מושצא מן המחסנית, ומוכנס למצויע התוכנית (PC). <u>הערה</u> : ערך זה נכנס למחסנית בקריאה לשגרה ע"י הוראת jsr	14	rts
התוכנית עצירת התוכנית.	stop	עצירת ריצת התוכנית.	15	stop

#### מבנה תכנית בשפת אסמבלי :

תכנית בשפת אסמבלי בנויה ממשפטים (statements). קובץ מקור בשפת אסמבלי מורכב משורות המכילות ממשפטים של השפה, כאשר כל ממשפט מופיע בשורה נפרדת. כלומר, הפרדה בין ממשפט למשפט בקובץ המקור הינה באמצעות התו 'ת' (שורה חדשה).

אורכה של שורה בקובץ המקור הוא 80 תווים לכל היתר (לא כולל התו ט).

יש ארבעה סוגי משפטיים (שורות בקובץ המקור) בשפת אסמבלי, והם :

סוג המשפט	הסבר כללי
משפט ריק	זהו שורה המכילה אך ורק תווים לבנים (whitespace), כלומר רק את התווים ' ' ו- 't' (רווחים וטאבים). יתכן ובשורה אין אף תו (למעט התו ט), כלומר השורה ריקה.
משפט הערת	זהו שורה בה התו הראשון הינו ';' (נקודה פסיק). על האסמבילר להתעלם לחולותין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמבילר מה לעליו לעשות כשהוא פועל על תוכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגזור להקצת זיכרון ואתחול משתנים של התוכנית, אך הוא אינו מייצר קידוד של הוראות מכונה המיועדות לביצוע בעת ריצת התוכנית.
משפט ההוראה	זהו משפט המ מייצר קידוד של הוראות מכונה לביצוע בעת ריצת התוכנית. המשפט מורכב ממשם של ההוראה שעל המעבד לבצע, ותיאור האופרנדים של ההוראה.

כעת נפרט יותר לגבי סוגי המשפטים השונים.

#### משפט הנטיה:

משפט הנטיה הוא בעל המבנה הבא:

בתחילת המשפט **יכולה להופיע הגדלה של תווית** (label). לתווית יש לחבר חוקי שיתואר בהמשך. התווית היא אופציונלית.

לאחר מכן מופיע שם הנטיה. לאחר שם הנטיה יופיעו פרמטרים (מספר הפרמטרים בהתאם להנטיה).

שם של הנטיה מתחילה בטע'.' (נקודה) ולאחריו תווים באותיות קטנות (lower case) בלבד.

**יש לשיט לב:** למלילים בקוד המכונה הנוצרות **משפט הנטיה לא מצורף השדה A,R,E**, והערך המוגדר על ידי הנטיה מלא את כל 24 הסיביות של המילה.

יש ארבעה סוגים (שמות) של משפטי הנטיה, והם:

##### 1. הנטיה 'data'.

הפרמטרים של הנטיה 'data' הם מספרים שלמים חוקיים (אחד או יותר) המופרדים על ידי הטע'.' (פסיק). לדוגמה:

.data 7, -57, +17, 9

יש לשיט לב שהפסיקים אינם חייבים להיות צמודים למספרים. בין מספר לפסק ובין פסיק למספר יכולים להופיע רווחים וטאים בכל כמות (או בכלל לא), אולם הפסיק חייב להופיע בין המספרים. כמו כן, אסור שיופיע יותר מפסיק אחד בין שני מספרים, וגם לא פסיק אחרி המספר האחרון או לפני המספר הראשון.

המשפט 'data' מנהה את האSEMBLER להקצota מקום בתמונה הנתונים (data image), אשר בו יוחסנו הערכים של הפרמטרים, ולאחר מכן הנתונים, בהתאם למספר הערכים. אם ב衲ית data מוגדרת תווית, אז תווית זו מקבלת את ערך מונה הנתונים (לפניהם הקידום), ומוכנסת אל טבלת הסמלים. דבר זה מאפשר להתייחס אל מקום מסוים בתמונה הנתונים **דרך** שם התווית (למעשה, זהה דרך להגדיר שם של משתנה).

כלומר אם נכתב:

XYZ: .data 7, -57, +17, 9

אז יוקצו בתמונה הנתונים ארבע מיללים רצופות שיכילו את המספרים שמופיעים בהנטיה. התווית XYZ מזוהה עם כתובות המילה הראשונה.

אם נכתב בתוכנית את הוראה:

mov XYZ, r1

אז בזמן ריצת התוכנית יוכנס לאוגר 1ז הערך 7.

ואילו הוראה:

lea XYZ, r1

תכנס לאוגר 1ז את ערך התווית XYZ (כלומר הכתובת בזיכרון בה מאוחSEN הערך 7).

##### 2. הנטיה 'string'.

ה衲יה 'string' פרמטר אחד, שהוא מחروفות חוקית. תוווי המחרוזות מקודדים לפי ערכי-ascii המתאימים, ומוכנסים אל תמונה הנתונים לפי סדרם, כל תוו במילה נפרד. בסוף המחרוזת

יתווסף התו '0' (הערך המספרי 0), המסמן את סוף המחרוזת. מונה הנתונים של האסמבילר יקודם בהתאם לאורך המחרוזת (בתוספת מקום אחד עבור התו המסיים). אם בשורת הינה מוגדרת תווית, אזי תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום) ומוכנסת אל טבלת הסמלים, בדומה כפי שנעשה עבור 'data'. (כלומר ערך התווית יהיה הכתובת בזיכרון שבה מתחילה המחרוזת).

לדוגמה, הינה:

STR: .string "abcdef"

מקצה בתמונה הנתונים רצף של 7 מילימ', ומאתחלת את המילימ' לקוד ascii של התווים לפי הסדר במחרוזת, ולאחריהם הערך 0 לסימון סוף מחרוזת. התווית STR מזוהה עם כתובות התחלה המחרוזת.

3. הינה 'entry'.

הינה 'entry'. פרטט אחד, והוא שם של תווית המוגדרת בקובץ המקור הנוכחי (כלומר תווית שמקבלת את ערכיה בקובץ זה). מטרת הינה היא לאפיין את התווית הזו באופן שיאפשר לקרוא אסמבלי הנמצא בקבצי מקור אחרים להשתמש בה (אופרנד של הוראה).

לדוגמה, השורות:

```
HELLO: .entry HELLO
        add #1, r1
```

מודיעות לאסמבילר שאפשר להתייחס בקובץ אחר לתווית HELLO המוגדרת בקובץ הנוכחי.

لتשומת לב: תווית המוגדרת בתחילת שורת entry. הינה חסרת משמעות והאסמבילר מתעלם מהתווית זו (אפשר שהאסמבילר יוציא הודעה אזהרה).

4. הינה 'extern'.

הינה 'extern'. פרטט אחד, והוא שם של תווית שאינה מוגדרת בקובץ המקור הנוכחי. מטרת ההוראה היא להודיע לאסמבילר כי התווית מוגדרת בקובץ מקור אחר, וכי קוד האסמבלי בקובץ הנוכחי עושה בתווית שימוש.

נשים לב כי הינה זו תואמת להינה 'entry'. המופיע בקובץ בו מוגדרת התווית. בשלב הקישור התבכע התאמה בין ערך התווית, כפי שנקבע בקוד המכונה של הקובץ שהגדיר את התווית, לבין קידוד ההוראות המשמשות בתווית בקבצים אחרים (שלב הקישור אינו רלוונטי לממין זה).

לדוגמה, משפט הינה 'extern'. התואם לשפט הינה 'entry'. מהדוגמה הקודמת יהיה:

.extern HELLO

لتשומת לב: תווית המוגדרת בתחילת שורת extern. הינה חסרת משמעות והאסמבילר מתעלם מהתווית זו (אפשר שהאסמבילר יוציא הודעה אזהרה).

משפט הוראה:

משפט הוראה מורכב מחלקים הבאים:

1. תווית אופציונלית.
2. שם הפעולה.
3. אופרנדים, בהתאם לסוג הפעולה (בין 0 ל-2 אופרנדים).

אם מוגדרת תווית בשורת ההוראה, אזי היא תוכנס אל טבלת הסמלים. ערך התווית יהיה מען המילה הראשונה של ההוראה בתוך תומנת הקוד שבונה האסמבילר.

שם הפעולה תמיד באותיות קטנות (lower case), והוא אחת מ- 16 הפעולות שפורטו לעיל.

לאחר שם הפעולה יופיעו האופרנדים, בהתאם לסוג הפעולה. יש להפריד בין שם-הפעולה לבין האופרנד הראשון באמצעות רווחים ו/או טאבים (אחד או יותר).

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזה בפסיק. בדומה להנחיה 'data.', לא **חייבת להיות הצמדה של האופרנדים לפסיק**. כל כמות של רווחים ו/או טאבים משני צידי הפסיק היא חוקית.

למשפט הוראה עם שני אופרנדים המבנה הבא :

label: opcode source-operand, target-operand

לדוגמה :

HELLO: add r7, B

למשפט הוראה עם אופרנד אחד המבנה הבא :

label: opcode target-operand

לדוגמה :

HELLO: bne &XYZ

למשפט הוראה ללא אופרנדים המבנה הבא :

label: opcode

לדוגמה :

END: stop

### אפיון השדות במשפטים של שפת האסמבלי

תוויות :

תוויות חוקיות מתחילה באות אלפביתיות (גדולה או קטנה), ולאחריה סדרה כלשהי של אותיות אלפביתיות (גדולות או קטנות) ו/או ספרות. האורך המקסימלי של תוויות הוא 31 תווים.

הגדרה של תוויות מסוימות בתו ': (נקודותים).תו זה אינו מהו חלק מהתוויות, אלא רק סימן המציין את סוף ההגדרה. התו ': חייב להיות צמוד לתוויות (לא רווחים).

אסור שאויה תוויות תוגדר יותר מפעם אחת (כموון בשורות שונות).אותיות קטנות וגדולות נחשבות שונות זו מזו.

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hHello:

X:

He78902:

**להשומת לב : מיללים שמורות של שפת האסמבלי** (כלומר שם של פעולה או הנחיה, או שם של אוגר) **אין יכולות לשמש גם כשם של תוויות**

התוויות מקבלת את ערכיה בהתאם להקשר בו היא מוגדרת.תוויות המוגדרת בהנחיות data. או **countstr.**, קיבל את ערך מונה הנתונים (data counter) הנוichi, בעוד שתוויות המוגדרת בשורת הוראה תקבל את ערך מונה ההוראות (instruction counter) הנוichi.

מספר :

מספר חוקי מתחילה בסימן אופציוני: '-' או '+' ולא חייב סדרה של ספרות בסיס עשרוני. דוגמה: -5, 76, +123 הם מספרים חוקיים. אין תמייה בשפת אסמבלי בייצוג בסיס אחר מאשר עשרוני, ואין תמייה במספרים שאינם שלמים.

## מחuzeות:

מחuzeות חוקיות היא סדרת תוויי `ascii` נראים (שניתנים להדפסה), המוקפים במרקאות כפולות (המרקאות אינן נחשבות חלק מהמחuzeות). דוגמה למחuzeות חוקית: "hello world".

### תפקיד השזה E,R,A בקוד המכונה

בכל מילה בקוד המכונה של הורהה (לא של נתונים), האסמבילר מכניס מידע עבור תהליך הקישור והטיענה. זהו השדה E,R,A (שלוש הסיביות הימניות 2,1,0 בהתחילה). המידע ישמש לתיקונים בקוד בכל פעם שייטען לזכור הרצה. האסמבילר בונה מלכתחילה קוד שמיועד לטיענה החל מכתובת 100. התיקונים יאפשרו לטען את הקוד בכל פעם למקום אחר, בלי צורך לחזור על תהליך האסמבלי.

בכל מילה של הורהה, בדיק אחת משלש הסיביות של השדה E,R,A מכילה 1, ושתי הסיביות האחרות מאופסות. מפרט שיטות המיען שהווצג קודם מציין איזו סיבית תכיל 1 בכל שיטת מעון.

סיבית 'A' (קייזור של Absolute) באה לציין שתוכן המילה אינו תלוי במקום בו זיכרנו בו יטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל מילה המכילה אופרנד מיידי).

סיבית 'R' (קייזור של Relocatable) באה לציין שתוכן המילה תלוי במקום בו יטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל מילה המכילה כתובת של תווית שמוגדרת בקובץ המקור הנוכחי).

סיבית 'E' (קייזור של External) באה לציין שתוכן המילה תלוי בערכו של סמל חיצוני (External). (למשל מילה המכילה כתובת של תווית שמוגדרת בקובץ מקור אחר).

### **אסמבילר עם שני מעברים**

כאשר מקבל האסמבילר תוכנית בשפת אסמבלי, עליו לעבור על התוכנית פעמים. במעבר הראשון, יש לזרות את הסמלים (תוויות) המופיעים בתוכנית, ולתת לכל סמל ערך מסווג שהוא המען בזיכרון שהסמל מייצג. במעבר השני, באמצעות ערכי הסמלים, וכן קוד-הפעולה ומספריו האוגרים, בונים את הקוד המכונה.

לדוגמה: האסמבילר מקבל את התוכנית הבאה בשפת אסמבלי :

```

MAIN:    add   r3, LIST
LOOP:    prn   #48
          lea   STR, r6
          inc   r6
          mov   r3, K
          sub   r1, r4
          bne   END
          cmp   K, #-6
          bne   &END
          dec   K
          jmp   &LOOP
END:     stop
STR:    .string "abcd"
LIST:   .data  6, -9
          .data -100
K:      .data  31

```

קוד המכוונה של התוכנית (הוראות ונתונים) נבנה כך שיתאים לטעינה בזיכרון החל ממען 100 (עשרוני).

התרגום של תוכנית תכנית המקור שבדוגמה לקוד בינארי מוצג להלן:

Address (decimal)	Source Code	Explanation	Binary Machine Code
0000100	MAIN: add r3, LIST	First word of instruction Address of label LIST	000010110110100000001100 0000000000000000111111010
0000101			
0000102	LOOP: prn #48	Immediate value 48	0011010000000000000000100 0000000000000000110000100
0000103			
0000104	lea STR, r6	Address of label STR	00010001000111000000100 00000000000000001111010010
0000105			
0000106	inc r6		0001010000011100000011100
0000107	mov r3, K	Address of label K	000000110110100000000100 000000000000000010000010010
0000108			
0000109	sub r1, r4		0000101100111100000010100
0000110	bne END	Address of label END	0010010000001000000010100 00000000000000001111001010
0000111			
0000112	cmp K, #-6	Address of label K Immediate value -6	0000010100000000000000100 000000000000000010000010010 1111111111111111111111010100
0000113			
0000114			
0000115	bne &END	Distance to label END	0010010000010000000010100 0000000000000000000000110100
0000116			
0000117	dec K		00010100000010000000100100 000000000000000010000010010
0000118			
0000119	jmp &LOOP	Distance to label LOOP	0010010000010000000001100 111111111111111111111101111100
0000120			
0000121	END: stop		0011110000000000000000000100
0000122	STR: .string "abcd"	Ascii code 'a'	00000000000000000000001100001
0000123		Ascii code 'b'	00000000000000000000001100010
0000124		Ascii code 'c'	00000000000000000000001100011
0000125		Ascii code 'd'	00000000000000000000001100100
0000126		Ascii code '\0'	00000000000000000000000000000000
0000127	LIST: .data 6, -9	Integer 6 Integer -9	00000000000000000000000000000010 111111111111111111111111011111
0000128			
0000129	.data -100	Integer -100	111111111111111111111111011100
0000130	K: .data 31	Integer 31	00000000000000000000000000000011111

האסמבלר מחזיק טבלה שבה רשומים כל שמות הפעולה של ההוראות והקודים הבינאריים (opcode, funct) המתאימים להם, וכן שמות הפעולות ניתנים להמרת לבינארי בקלות. כאשר נקרא שם פעולה, אפשר פשוט לעיין בטבלה ולמצוא את הקידוד הבינארי.

כדי לבצע המרה לבינארי של אופרנדים שכטובים בשיטות מיעון המשמשות בסמלים (תוויות), יש ליצור לבנות טבלה המכילה את ערכיו כל הסמלים. אולם בהבדל מהקודים של הפעולות, הידועים מראש, הרי המענינים בזיכרון עברו הסמלים בשימוש התוכנית אינם ידועים, עד אשר תוכנית המקור נסרקה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד לעיל, האסמבלר איינו יכול לדעת שהסמל END אמרור להיות משוויך למםן 121 (עשרוני), והסמל K אמרור להיות משוויך למםן 130, אלא רק לאחר שנקרוו כל שורות התוכנית.

לכן מפרידים את הטיפול של האסמבלר בסמלים לשני שלבים. בשלב הראשון בונים טבלה של כל הסמלים, עם הערכים המספריים המשווים להם, ובשלב השני מחליפים את כל הסמלים, המופיעים באופרנדים של הוראות התוכנית, בערכיהם המספריים. הביצוע של שני שלבים אלה כרוך בשתי סריקות (הנקראות " מעברים ") של קובץ המקור.

במעבר הראשון נבנית טבלת סמלים בזיכרון, ובה לכל סמל שבתוכנית המקור משוויך ערך מספרי, שהוא מען בזיכרון.

עבור הדוגמה לעיל, טבלת הסמלים היא כדלקמן:

סמל	ערך (בבסיס עשרוני)
MAIN	100
LOOP	102
END	121
STR	122
LIST	127
K	130

במעבר השני נעשית ההמרה של קוד המקור לקוד מכונה. בתחילת המעבר שני צרכיים הערכים של כל הסמלים להיות כבר ידועים.

لتשומת לב: תפקיד האסטובלר, על שני המעברים שלו, לתרגם קובץ מקור לקוד בשפת מכונה. בגמר פעולה האסטובלר, התוכנית טרם מוכנה לטעינה ל זיכרון לצורך ביצוע. קוד המכונה חייב בעבר שלבי הקישור/טעינה, ורק לאחר מכן לשלב הביצוע (שלבים אלה אינם חלק מההמ"ז).

### המעבר הראשון

במעבר הראשון נדרשים כללים כדי לקבוע איזה מען ישוק לכל סמל. העיקרונות הבסיסיים הם לספור את המיקומות בזיכרון, אותן תופסות ההוראות. אם כל הוראהティיען בזיכרון למקום העוקב להוראה הקודמת, תציג ספריה כזאת את מען ההוראה הבאה. הספריה נעשית על ידי האסטובלר ומוחזקת במונה ההוראות (IC). ערכו ההתחלתי של IC הוא 100 (עשרוני), וכן קוד המכונה של ההוראה הראשונה נבנה כך שיכירון החל ממטען 100. ה-IC מתעדכן בכל שורת הוראה המקצת מקום בזיכרון. לאחר שהאסטובלר קובע מהו אורך ההוראה, ה-IC מוגדל במספר התאים (מיילים) הנתפסים על ידי ההוראה, וכך הוא מציבע על התא הפנוי הבא.

כאמור, כדי לקודד את ההוראות בשפת מכונה, מחזק האסטובלר טבלה, שיש בה קידוד מתאים לכל שם פעולה. בזמן התרגום מחליף האסטובלר כל שם פעולה בקידוד שלה. כמו כן, כל אופרנד מוחלף בקידוד מתאים, אך פעולה החלפה זו אינה כה פשוטה. ההוראות משתמשות בשיטות מייען מגוונות לאופרנדים. אותה פעולה יכולה לקבל שימושיות שונות, בכל אחת משיטות המייען, וכן יתאים לה קידודים שונים לפי שיטות המייען. לדוגמה, פעולה ההזזה Zus תסמן יכולה להתיחס להעתיקת תוכן תא זיכרון לאוגר, או להעתיקת תוכן אוגר לאוגר אחר, וכן הלאה. לכל אפשרות כזו שוטן עשוי להתאים קידוד שונה.

על האסטובלר לסרוק את שורת ההוראה בשלמותה, ולהחליט לגבי הקידוד לפי האופרנדים. בדרך כלל מתחלק הקידוד לשדה של שם הפעולה, ושדות נוספים המכילים מידע לגבי שיטות המייען. כל השדות ביחיד דורותים מילה אחת או יותר בקוד המכונה.

כאשר נתקל האסטובלר בתווית המופיעה בתחילת ההוראה, הוא יודע שלפניו הגדרה של תווית, והוא משייך לה מען – תוכנו הנוכחי של IC. כך מקבלות כל התוויות את מענייהן בעת ההגדירה. תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנוסף לשם התווית גם את המען ומאפיינים נוספים. כאשר תהייה התייחסות לתווית באופרנד של ההוראה בלבד, יוכל האסטובלר לשולף את המען המתאים מטבלת הסמלים.

הוראה יכולה להתיחס גם לסמל שטרם הוגדר עד כה בתוכנית, אלא יוגדר רק בהמשך התוכנית. להלן, לדוגמה, הוראות הסתעפות למען שמוגדר על ידי התווית A שמופיעה רק בהמשך הקוד:

A: .....  
 bne A  
 .  
 .  
 .

כאשר מגיע האסטובלר לשורת ההסתעפות (Aorph), הוא טרם נתקל בהגדרת התווית A וכמוון לא יודע את המען המשויך לתווית. לכן האסטובלר לא יכול לבנות את הקידוד הבינארי של האופרנד A. נראה בהמשך כיצד נפתרת בעיה זו.

בכל מקרה, תמיד אפשר לבנות מעבר הראשון את הקידוד הבינארי המלא של המילה הראשונה של כל הוראה, את הקידוד הבינארי של מילט-המידע הנוסף של אופרנד מיידי, וכן את הקידוד הבינארי של כל הנתונים (המתקבלים מההנחיות `data`, `string`).

## המעבר השני

ראינו שבמעבר הראשון, האסמלר אינו יכול לבנות קוד המכונה של אופרנדים המשמשים בסמלים שעדיין לא הוגדרו. רק לאחר שהאסמלר עבר על כל התוכנית, כך שכל הסמלים נכנסו כבר לטבלת הסמלים, יוכל האסמלר להשלים את קוד המכונה של כל האופרנדים.

לשם כך מבצע האסמלר מעבר נוסף (מעבר שני) על כל קובץ המקוור, ומעדכן את קוד המכונה של האופרנדים המשמשים בסמלים, באמצעות ערכיו הסמלים מטבלת הסמלים.  
בסוף המעבר השני, תהיה התוכנית מתורגמת בשלמותה לקוד מכונה.

## הפרדת הוראות ונתונים

בתוכנית מבחןים בשני סוגים של תוכן: הוראות ונתונים. יש לארכן את קוד המכונה כך שתתיה הפרדה בין הנתונים וההוראות. הפרדת ההוראות והנתונים לקטועים שונים בזיכרון היא שיטה עדיפה על פני הצמדה של הגדרות הנתונים להוראות המשמשות בהן.

אחת הסכנות הטමונות באירוע ההראות מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית בתוכנית, לנסות "לבצע" את הנתונים כאילו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום לתופעה כזו היא הסתעפות לא נכונה. התוכנית כמובן לא תעבור נכון, אך לרוב הנזק הוא יותר חמוץ, כי נוצרת חריגת חומרה ברגע שהמעבד מבצע פעולה שאינה חוקית.

האסמלר שלנו חייב להפריד, בקוד המכונה שהוא מיציר, בין קטע הנתונים לבין ההוראות. כלומר בקובץ הפלט (בקוד המכונה) תהיה הפרדה של הוראות ונתונים לשני קטיעים נפרדים, אם כי בקובץ הקלט אין חובה שתהייה הפרדה כזו. בהמשך מתואר אלגוריתם של האסמלר, ובו פרטים כיצד לבצע את ההפרדה.

## גילוי שגיאות בתוכנית המקור

האסמלר אמר לגלות ולדוח על שגיאות בתחריב של תוכנית המקור, כגון פעולה שאינה קיימת, מספר אופרנדים שגוי, סוג אופרנד שאין מתאים לפעולה, שם אוגר לא קיים, ועוד שגיאות אחרות. כמו כן מודיא האסמלר שכלי סמל מוגדר פעמי אחת בדיקות.

מכאן, שכלי שגיאה המתגלה על ידי האסמלר נגרמת (בדרך כלל) על ידי שורת קלט מסויימת.

לדוגמה, אם מופיעים שני אופרנדים בהוראה שאמור להיות בה רק אופרנד ייחיד, האסמלר ייתן הודעת שגיאה בנושא "יוטר מדי אופרנדים".

האסמלר ידפיס את הודעות השגיאה אל הפלט הסטנדרטי `stdout`. בכל הודעת שגיאה יש לציין גם את מספר השורה בקובץ המקור בה זוהתה השגיאה.

לתשומתך: האסמלר אינו עוצר את פעולתו אחרי שנמצאה השגיאה הראשונה, אלא ממשיך לעبور על הקלט כדי לגלוות שגיאות נוספות, ככל שיישன. כמובן שאין כל טעם לייצר את קבצי הפלט אם נתגלו שגיאות (ممילא אי אפשר להשלים את קוד המכונה).

הטבלה הבאה מפרטת מהן של שיטות המיעון החוקיות, עבור אופרנד המקור ואופרנד היעד של ההוראות השונות הקיימות בשפה הנתונה:

שיטות מייען חוקיות עבור אופרנד היעד	שיטות מייען חוקיות עבור אופרנד המקור	שם ההוראה	funct	Opcode
1,3	0,1,3	mov		0
0,1,3	0,1,3	cmp		1
1,3	0,1,3	add	1	2
1,3	0,1,3	sub	2	2
1,3	1	lea		4
1,3	אין אופרנד מקור	clr	1	5
1,3	אין אופרנד מקור	not	2	5
1,3	אין אופרנד מקור	inc	3	5
1,3	אין אופרנד מקור	dec	4	5
1,2	אין אופרנד מקור	jmp	1	9
1,2	אין אופרנד מקור	bne	2	9
1,2	אין אופרנד מקור	jsr	3	9
1,3	אין אופרנד מקור	red		12
0,1,3	אין אופרנד מקור	prn		13
אין אופרנד יעד	אין אופרנד מקור	rts		14
אין אופרנד יעד	אין אופרנד מקור	stop		15

### תהליך העבודה של האסטמבלר

נתאר כעת את אופן העבודה של האסטמבלר. בהמשך, יוצג אלגוריתם שלדי למעבר ראשון ושני.

האסטמבלר מתחזק שני מערכיים, שינקראו להלן תומנות ההוראות (code) ותומנות הנתונים (data). מערכיים אלו נותנים למשעה תומנה של זיכרון המכונה (כל איבר בזיכרון הוא בגודל מילה של המכונה, כולם 24 סיביות). במערך ההוראות בונה האסטמבלר את הקידוד של הוראות המכונה שנקראו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסטמבלר את קידוד הנתונים שנקראו מקובץ המקור (שורות הначיה מסוג 'data', ו-'string').

האסטמבלר משתמש בשני מונחים, שינקראים IC (מונה ההוראות - DC (Instruction-Counter), ו- (Data-Counter). מונחים אלו מצבעים על המקום הבא הפנוי במערך ההוראות ובמערך הנתונים, בהתאם. בכל פעם כשמתחליל האסטמבלר לעבר על קובץ מקור, המונה IC מקבל ערך התחלתי 100, והמונה DC מקבל ערך התחלתי 0. הערך ההתחלתי IC=100 נקבע כדי שקוד המכונה של התוכנית יתאים לטעינה לזכרון (לצורך ריצה) החל מכתובת 100.

בנוסך, מתחזק האסטמבלר טבלה, אשר בה נאפסות כל התוויות בהן נתקל האסטמבלר במהלך המעבר על קובץ המקור. לטבלה זו קוראים טבלת-הסמלים (symbol-table). לכל סמל נשמרם בטבלה שם הסמל, ערכו המספרי, ומאפיינים שונים, כגון המיקום (code או data), וסוג הסמל (entry או external).

במעבר הראשון האסטמבלר בונה את טבלת הסמלים ואת השלד של תומנות הזיכרון (הוראות ונתונים).

האסטמבלר קורא את קובץ המקור שורה אחר שורה, ופועל בהתאם לשוג השורה (הוראה, הначיה, או שורה ריקה/הערה).

1. שורה ריקה או שורת הערה : האסטמבלר מתעלם מהשורה וועבר לשורה הבאה.

2. שורת הוראה :

האסטמבלר מנתח את השורה ומפענח מהי ההוראה, ומהן שיטות המייען של האופרנדים. מספר האופרנדים נקבע בהתאם להוראה שנמצאה. שיטות המייען נקבעות בהתאם לתחביר של כל אופרנד, כפי שהסביר לעיל במפרט שיטות המייען. למשל, התו '# מצין מיעון מיידי, תווית מצינית מיעון ישיר, שם של אוגר מצין מיעון אוגר ישיר, וככ'.

אם האסמבller מוצא בשורת ההוראה גם הגדלה של תווית, אז התווית מוכנסת אל טבלת הסמלים. ערך התווית הוא IC, והמאפיין הוא code.

כעת האסמבller קובע לכל אופרנד את ערכו באופן הבא :

- אם זה אוגר – האופרנד הוא מספר האוגר.
- אם זו תווית (מייעון ישר) – האופרנד הוא ערך התווית כפי שמופיע בטבלת הסמלים (ויתכן והסמל טרם נמצא בטבלת הסמלים, במידה והוא יוגדר רק בהמשך התוכנית).
- אם זה התו # ואחריו מספר (מייעון מיידי) – האופרנד הוא המספר עצמו.
- אם זו שיטת מייעון אחרת – ערכו של האופרנד נקבע לפי המפרט של שיטת המייעון (ראו תאור שיטות המייעון לעיל)

האסמבller מכניס למערך ההוראות, בכינסה עליה מצבעו מונה ההוראות IC, את קוד המילה הראשונה של ההוראה (בפורמט קידוד כפי שתואר קודם). מילה זו מכילה את קוד הפעולה, את מספרי שיטות המייעון של אופרנד המקור והיעד, ואת מספרי האוגרים של אופרנד המקור והיעד במקרה של שיטת מייעון אוגר ישר. ה-IC מקודם ב-1.

נזכיר שכאשר יש רק אופרנד אחד (כלומר אין אופרנד מקור), הסיביות של שיטת המייעון של אופרנד המקור יכilo 0. בדומה, אם זה ההוראה ללא אופרנדים (ts, stop), אז הסיביות של שיטות המייעון של שני האופרנדים יכilo 0. כמו כן, אם שיטת המייעון אינה אוגר ישר, הסיביות של מספר האוגר הרלוונטי יכilo 0.

אם זה ההוראה עם אופרנדים (אחד או שניים), האסמבller "משרין" מקום למערך ההוראות עבור מילוט-הميدע הנוספות הנדרשות בהוראה זו, ככל שנדרשות, ומקדם את IC בהתאם. כאשר אופרנד הוא בשיטת מייעון מיידי, האסמבller מקודד גם את המילה הנוספת המתאימה למערך ההוראות. ואילו בשיטת מייעון ישר או יחס, מילת המידע הנוספת למערך ההוראות נשארת ללא קידוד בשלב זה.

### 3. שורת הנחיה :

כאשר האסמבller קורא בקובץ המקור שורת הנחיה, הוא פועל בהתאם לסוג הנחיה, באופן הבא :

I. 'data'.  
האסמבller קורא את רשימת המספרים, המופיעה לאחר 'data.', מכניס כל מספר אל מערך הנתונים, ומקדם את מצבי הענתונים DC ב-1 עבור כל מספר שהוכנס.

אם בשורה 'data.' מוגדרת גם תווית, אז התווית מוכנסת לטבלת הסמלים. ערך התווית הוא ערך מונה הענתונים DC שלפניהם הכנסת המספרים למערך. המאפיין של התווית הוא .data.

II. 'string'.  
הטיפול ב-'string' דומה ל-'data.', אלא שקובדי ascii של התווים הם אלו המוכנסים אל מערך הענתונים (כל تو במילה נפרדת). לבסוף מוכנס למערך הענתונים ערך 0 (המצין סוף מחוזות). המונה DC מקודם באורך המחרוזת + 1 (גם התו המסיים את המחרוזת תופס מקום).

הטיפול בתווית המוגדרת בהנחיה string. זהה לטיפול הנעשה בהנחיה 'data.'

III. 'entry'.  
זהו הינה הנחיה לאסמבller לאפיין את התווית הענתונה כאופרנד כ-entry בטבלת הסמלים. בעת הפיקת קבצי הפלט (ראו בהמשך), התווית תירשם בקובץ ה-entries.  
لتשומת לב: זה לא נחשב כשגיאה אם בקובץ המקור מופיעה יותר מהנחיית עטוף. אחת עם אותה תווית כאופרנד. המופעים הנוספים אינם מושיפים דבר, אך גם אינם מפריעים.

#### IV. *extern*

זהי הערה על סמל (תוית) המוגדר בקובץ מקור אחר, והקובץ הנוכחי עושה בו שימוש.  
האSEMBLER מכניס את הסמל המופיע כאופרנד לטבלת הסמלים, עם הערך 0 (הערך האמיטי לא ידוע, וייקבע רק בשלב הקישור), ועם המאפיין *external*. לא ידוע באיזה קובץ נמצאת הגדרת הסמל, ואין זה רלוונטי עבור האSEMBLER.

לתשומת לב: זה לא נחשב כשייניה אם בקובץ המקור מופיע יותר מהנהנית *extern*. אחת עם אותה תווית כאופרנד. המופיעים הנוספים אינם מושפעים דבר, אך גם אינם מפריעים.

לתשומת לב: באופרנד של הוראה או של הנחית *entry*, מותר להשתמש בסמל אשר יוגדר בהמשך הקובץ (אם באופן ישיר על ידי הגדרת תווית, ואם באופן עקיף על ידי הנחית *extern*).

בסוף המעבר הראשון, האSEMBLER מעדכן בטבלת הסמלים כל סמל המופיע כ-*data*, על ידי הוספת (100 + IC) לערך של הסמל. הסיבה לכך היא שבתמונה הכלולות של קוד המכונה, תמונה הנתונים מופרדת מתמונה הוראות, וכל הנתונים נדרש להופיע בקוד המכונה אחרי כל ההוראות. סמל מסוג *data* הוא תווית בתמונה הנתונים, והעדכון מוסיף לערך הסמל (כלומר לכתובתו בזיכרון) את האורך הכלול של תמונה ההוראות, בתוספת כתובת התחלה הטעינה של הקוד, שהיא 100.

טבלת הסמלים מכילה כתעת את כל הערכים הנחוצים להשלמת תמונה הזיכרון (למעט ערכים של סמלים חיצוניים).

במעבר השני, האSEMBLER משלים באמצעות טבלת הסמלים את קידוד כל המילים במערך ההוראות שטרם קידדו במעבר הראשון. במודל המכונה שלנו אלו הן מילוט-מידע נוספת של פקודות, אשר מקודדות אופרנד בשיטת מיון ישיר או יחסית. האופרנד מכיל סמל שמווגדר כפנימי או חיצוני, ולבן בשדה ה-*A,R,E* הסיבית *R* או הסיבית *E*, בהתאם, תהיה 1 (ראו גם מפרט שיטות המיון לעיל).

#### אלגוריתם שלדי של האSEMBLER

לחידוד ההבנה של תהליך העבודה של האSEMBLER, נציג להלן אלגוריתם שלדי למעבר הראשון ולמעבר השני.

לתשומת לב: אין חובה להשתמש דוקא באלגוריתם זה.

כאמור, אנו מחלקים את תמונה קוד המכונה לשני חלקים: תמונה ההוראות (*code*), ותמונה הנתונים (*data*). לכל חלק נתזק מונה נפרד: IC (מונה ההוראות) ו-DC (מונה הנתונים).

**בנייה את קוד המכונה כך שייתאים לטעינה לזכרון החל מכתובת 100.**

בכל מעבר מתחילה לקרוא את קובץ המקור מההתחלתה.

#### מעבר ראשון

1. **אתחל**  $100 \leftarrow 0, IC \leftarrow DC \leftarrow 0$ .
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עברו ל-17.
3. האם השדה הראשון בשורה הוא סמל? אם לא, עברו ל-5.
4. הדלק דגל "יש הגדרת סמל".
5. האם זוהי הנחית לאחסון נתונים, כלומר, האם הנחית *data*. או *string*? אם לא, עברו ל-8.
6. אם יש הגדרת סמל (תוית), הכנס אותו לטבלת הסמלים עם המאפיין *data*. ערך הסמל יהיה DC. (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
7. זהה את סוג הנתונים, קודו אותם בתמונה הנתונים, והגדיל את מונה הנתונים DC על ידי הוספת האורך הכלול של הנתונים שהוגדרו בשורה הנוכחית. חוזר ל-2.
8. האם זו הנחית *extern*. או הנחית *entry*? אם לא, עברו ל-11.
9. אם זוהי הנחית *entry*. חוזר ל-2 (הנחהיה תטפל במעבר השני).
10. האם זו הנחית *extern*, הכנס את הסמל המופיע כאופרנד של ההנחה לתוכן טבלת הסמלים עם הערך 0, ועם המאפיין *extern*. חוזר ל-2.

11. זהה שורת הוראה. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם המאפיין `.code`.
12. ערכו של הסמל יהיה IC (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
13. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא, אז הודיע על שגיאה בשם ההוראה.
14. נתח את מבנה האופרנדים של ההוראה, וחשב מהו מספר המיללים הכלול בתופסת ההוראה בקוד המכונה (נקרא למספר זה L).
15. שומר את הערכיהם IC → L יחד עם נתוני קוד המכונה של ההוראה.
16. עדכן L → IC + IC ← IC, וזורר ל-2.
17. קובץ המקור נקרא בשנותו. אם נמצא שגיאות במעבר הראשוני, עזרו כאן.
18. שומר את הערכיהם הסופיים של IC ושל DC (נקרא להם ICF ו-DCE). השתמש בהם לבניית קבצי הפלט, אחרי המעבר השני.
19. עדכן בטבלת הסמלים את ערכו של כל סמל המופיע כ-`data`, ע"י הוספת הערך ICF (ראה הסבר לכך בהמשך).
20. התחל מעבר שני.

#### **מעבר שני**

1. קרא את השורה הבאה מקובץ המקור. אם גמר קובץ המקור, עברו ל-7.
2. אם השדרה הראשונית בשורה הוא סמל (תוויות), דלג עליו.
3. האם זהה הנחיתת `data`? או `extern string`? אם כן, זורר ל-1.
4. האם זהה הנחיתת `entry`? אם לא, עברו ל-6.
5. הוסף בטבלת הסמלים את המאפיין `entry` למופיעים הסמל המופיע כאופרנד של ההנחיתת (אם הסמל לא נמצא בטבלת הסמלים, יש להודיע על שגיאה). זורר ל-1.
6. השלים את הקידוד הבינארי של מילוט-המידע של האופרנדים, בהתאם לשיטות המיעון שהסבירו. לכל אופרנד בקוד המקור המכיל סמל, מצא את ערכו של הסמל בטבלת הסמלים (אם הסמל לא נמצא בטבלה, יש להודיע על שגיאה). **אם הסמל מאופיין external**, הוסף את **כתובת מילוט-המידע הרלוונטי לרישימת מילוט-המידע שמתיחסות לסמלי חיצוני**. לפי הצורך, לחישוב הקידוד והכתובות, אפשר להיעזר בערכיהם IC ו-L של ההוראה, כפי שנשמרו במעבר הראשוני. זורר ל-1.
7. קובץ המקור נקרא בשנותו. אם נמצא שגיאות במעבר השני, עזרו כאן.
8. בנה את קבצי הפלט (פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו קודם, ונציג את הקוד הבינארי שמתתקבל במעבר ראשון ובמעבר שני.

```

MAIN:    add   r3, LIST
LOOP:    prn   #48
          lea    STR, r6
          inc   r6
          mov    r3,K
          sub   r1, r4
          bne   END
          cmp   K, #-6
          bne   &END
          dec   K
          jmp   &LOOP
END:     stop
STR:    .string "abcd"
LIST:   .data  6, -9
          .data -100
K:      .data  31

```

בוצע מעבר ראשון על הקוד לעיל, ובנייה את טבלת הסמלים. כמו כן, נשלים במעבר זה את הקידוד של כל תMOVות הנתונים, ושל המילה הראושונה של כל הוראה. כמו כן, נקודד מילוט-מידע נוספים של כל הוראה, ככל שקידוד זה אינו תלוי בערך של סמל. את מילוט-המידע שעדין לא ניתן לקודד במעבר הראשון נסמן ב "?" בדוגמה להלן.

Address (decimal)	Source Code	Explanation	Binary Machine Code
0000100	MAIN: add r3, LIST	First word of instruction	000010110110100000001100
0000101		Address of label LIST	?
0000102	LOOP: prn #48		001101000000000000000000100
0000103		Immediate value 48	0000000000000000110000100
0000104	lea STR, r6		000100010001111000000100
0000105		Address of label STR	?
0000106	inc r6		000101000001111000011100
0000107	mov r3, K		0000000110110100000000100
0000108		Address of label K	?
0000109	sub r1, r4		000010110011110000010100
0000110	bne END		001001000000100000010100
0000111		Address of label END	?
0000112	cmp K, #-6		0000010100000000000000100
0000113		Address of label K	?
0000114		Immediate value -6	1111111111111111010100
0000115	bne &END		00100100000100000010100
0000116		Distance to label END	?
0000117	dec K		000101000001000000100100
0000118		Address of label K	?
0000119	jmp &LOOP		00100100000100000001100
0000120		Distance to label LOOP	?
0000121	END: stop		00111100000000000000000000100
0000122	STR: string "abcd"	Ascii code 'a'	00000000000000000000001100001
0000123		Ascii code 'b'	00000000000000000000001100010
0000124		Ascii code 'c'	00000000000000000000001100011
0000125		Ascii code 'd'	00000000000000000000001100100
0000126		Ascii code '\0'	0000000000000000000000000000000
0000127	LIST: .data 6, -9	Integer 6	0000000000000000000000000000110
0000128		Integer -9	1111111111111111111111110111
0000129	.data -100	Integer -100	1111111111111111111111110011100
0000130	K: .data 31	Integer 31	000000000000000000000000000000011111

טבלת הסמלים אחרי מעבר ראשון היא :

סמל	ערך (בסיס עשרוני)	איפיון הסמל
MAIN	100	code
LOOP	102	code
END	121	code
STR	122	data
LIST	127	data
K	130	data

בוצע עתה את המעבר השני. נשלים באמצעות טבלת הסמלים את הקידוד החסר במיללים המסומנים "?" . הקוד הבינארי בצוותו הסופית וכך זהה לקוד שהוצע בתחלת הנושא "אסמבלר עם שני מעברים".

הערה : כאמור, האסמבלר בונה קוד מכונה כך שייתאים לטעינה לזכרו החל מכתובת 100 (עשרוני).

אם הטעינה בפועל (לצורך רכזת התוכנית) תהיה לכתובות אחרות, יידרשו תיקונים בקוד הבינארי בשלב הטעינה, שיוכנסו בעזרת מידע נוסף שהאסמבller מכין בקבצי הפלט (ראו בהמשך).

בסוף המעבר השני, אם לא נתגלו שגיאות, האסטブル בונה את קבצי הפלט (ראו בהמשך), שמכילים את הקוד הבינארי ומידע נוסף עבור שלבי הקישור והטיענה. כאמור, שלבי הקישור והטיענה אינם לミושם בפרקיקת זה, ולא נדנו בהם כאן.

## קבצי קלט ופלט של האסמבולר

בhfulla של האסmbler, יש להעיבר אליו באמצעות ARGUMENTS של שורת הפוקודה (command line arguments) רשיימה של שמות קבצי מקור אחד או יותר. אלם קבצי טקסט, ובם תוכניות בבחירה של שפת האסmbler שהוגדרה במ"ז.

האסטמבלר פועל על כל קובייה מקור בונפרד. ויוצר עבورو קבازي פלט כדלקמן:

- קובץ `object`, המכיל את קוד המוכונה.
  - קובץ `externals`, ובו פרטים על כל המקומיות (הכתובות) בקוד המוכונה בהם יש מילת-מידע שמקודדת ערך של סמל חזוני (סמל שהוגדר באמצעות הvariable `extern`), ומופיע בטבלת הסמלים כ-`external`.
  - קובץ `entries`, ובו פרטים על כל סמל שימושר נקבעת כניסה (סמל שהופיע כאופרנד של הvariable `entry`. ומופיע בטבלת הסמלים כ-`entry`).

אם אין בקובץ המקור אף הנקיטת `extern`, האסמבולר לא יוצר את קובץ הפלט מסוג `externals`.  
אם אין בקובץ המקור אף הנקיטת `entry`, האסמבולר לא יוצר את קובץ הפלט מסוג `entries`.

שמות קבצי המקור חייבים להיות עם הסיומת "as". למשל, השמות x.as, y.as, ו-x.as הם שמות חוקיים. העברת שמות הקבצים הללו כארוגמנטים לאסSEMBLER נעשית לא ציון הסיומת.

לדוגמה: נניח שתוכנית האסSEMBLER שלנו נקראת assembler, אז שורת הפקודה הבאה:

assembler x y hello

תרץ את האסMBELLER על הקבצים: x.as, y.as, hello.as

שמות קבצי הפלט מבוססים על שם קובץ הקלט, כפי שהופיע בשורת הפקודה, בתוספת סיומת מתאימה: הסיומת "ob." עבר קובץ ה-object, הסיומת "ent." עבר קובץ ה-entries, והסיומת "ext." עבר קובץ ה-externals.

לדוגמה, בהפעלת האסMBELLER באמצעות שורת הפקודה: assembler x.y.ext.x.o. יוצר קובץ פلت ob.x, וכן קבצי פلت ext.x.o. ו-ext.entry. או .extern. בקובץ המקור. נציג כעת את הפורמטים של קבצי הפלט. דוגמאות יובאו בהמשך.

#### **פורמט קובץ ה-object**

קובץ זה מכיל את תМОונת הזיכרון של קוד המכוּנה, בשני חלקים: תМОונת ההוֹראות ראשונה, ואחריה ובצמוד תМОונת הנתונים.

כזכור, האסMBELLER מקודד את ההוֹראות כך שתМОונת ההוֹראות תתאים לטעינה **חחל מכתובה 100** (עשורי) בזיכרון. נשים לב שרך בסוף המעבר הראשוני יודעים מהו הגולל הכללי של תМОונת ההוֹראות. מכיוון שתМОונת הנתונים נמצאת אחרי תМОונת ההוֹראות, גודל תМОונת ההוֹראות משפיע על הכתובות בתМОונת הנתונים. זו הסיבה שבגללה היה צריך לעדכן בטבלת הסמלים, בסוף המעבר הראשוני, את ערכי הסמלים המאופיינים כ-data (כזכור, בערך 19 הוסףנו לכל סמל כזה את הערך ICF). במעבר השני, בהשלמת הקידוד של מילוט-המידע, משתמשים בערכים המעודכנים של הסמלים, המותאמים למבנה המלא והסופי של תМОונת הזיכרון.

כעת האסMBELLER יכול לכתוב את תМОונת הזיכרון בשלמותה לתוך קובץ פلت (קובץ ה-object).

השורה הראשונה בקובץ ה-object היא "cotratt", המכילה שני מספרים (בבסיס עשרוני): הראשון הוא האורך הכללי של תМОונת ההוֹראות (ב밀ות זיכרון), והשני הוא האורך הכללי של תМОונת הנתונים (ב밀ות זיכרון). בין שני המספרים מפריד רווח אחד. כזכור, במעבר הראשוני, בערך 18, נשמרו הערכים ICF ו-IDF. האורך הכללי של תМОונת ההוֹראות הוא 100-ICF, והאורך הכללי של תМОונת הנתונים הוא IDF.

התוצאות הבאות בקובץ מכילות את תМОונת הזיכרון. בכל שורה שני שדות: כתובות של מילה בזיכרון, ותוכן המילה. הכתובות תירשם בסיסי **עשורי שבע ספרות** (כולל אפסים מוביילים). תוכן המילה יירשם בסיסי **הקסאדיימלי ב-6 ספרות** (כולל אפסים מוביילים). בין שני השדות בשורה יש רווח אחד.

#### **פורמט קובץ ה-entries**

קובץ entries בניית משורות טקסט, שורה אחת לכל סמל שמאופיין בטבלת הסמלים כ-entry. בשורה מופיע שם הסמל, ולאחריו ערכו כפי שנקבע בטבלת הסמלים (בסיסי עשרוני בשבע ספרות, כולל אפסים מוביילים). בין שני השדות בשורה יש רווח אחד. **אין חשיבות לסדר השורות**, כי כל שורה עומדת בפני עצמה.

#### **פורמט קובץ ה-externals**

קובץ externals בניית אף הוא משורות טקסט, שורה לכל כתובות בקוד המכוּנה בה יש מילת מדע המתואחשת לסמל שמאופיין כ-external. כזכור, רשיימה של מילוט-מידע אלה נבנתה במעבר השני (בערך 6).

כל שורה בקובץ `externals` מכילה את שם הסמל החיצוני, ולאחריו הכתובת של מילט-המידע (בבסיס עשרוניسبע ספרות, כולל אפסים מובילים). בין שני השדות בשורה יש רווח אחד. אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.

תשומת לב: ייתכן ויש מספר כתובות בקובץ המכונה בהן מילות-המידע מתאפיחות לאותו סמל חיצוני. לכל כתובות כזו תהיה שורה נפרדת בקובץ `externals`.

נדגים את הפלט שמייצר האסמבולר עבור קובץ מקור בשם `ps.as` הנתון להלן.

```
; file ps.as

.entry LIST
.extern W
MAIN:    add   r3, LIST
LOOP:     prn   #48
          lea   W, r6
          inc   r6
          mov   r3, K
          sub   r1, r4
          bne   END
          cmp   K, #-6
          bne   &END
          dec   W
.entry MAIN
          jmp   &LOOP
          add   L3, L3
END:      stop

STR:      .string "abcd"
LIST:     .data  6, -9
          .data  -100
K:        .data  31
.extern L3
```

להלן היקוד הבינארי המלא (תמונה הזיכרון) של קובץ המקור, כפי שנבנה מעבר הראשוני והשני.

טבלת הסמלים הסופית בגמר המעבר השני היא :

סמל	ערך (בסיס עשרוני)	איפיון הסמל
W	0	external
MAIN	100	code, entry
LOOP	102	code
END	124	code
STR	125	data
LIST	130	data, entry
K	133	data
L3	0	external

להלן תוכן קבצי הפלט של הדוגמה.

הקובץ **ps.o**

```
25 9
0000100 0b680c
0000101 000412
0000102 340004
0000103 000184
0000104 111e04
0000105 000001
0000106 141e1c
0000107 036804
0000108 00042a
0000109 0b3c14
0000110 240814
0000111 0003e2
0000112 050004
0000113 00042a
0000114 fffffd4
0000115 241014
0000116 00004c
0000117 140824
0000118 000001
0000119 24100c
0000120 ffff7c
0000121 09080c
0000122 000001
0000123 000001
0000124 3C0004
0000125 000061
0000126 000062
0000127 000063
0000128 000064
0000129 000000
0000130 000006
0000131 ffffff7
0000132 ffff9c
0000133 00001f
```

הקובץ **ps.ent**

MAIN 0000100  
LIST 0000130

W 0000105  
W 0000118  
L3 0000122  
L3 0000123

## סיכום והנחיות כלליות

- גודל תוכנית המקור הניתנת כקלט לאסטבלר אינו ידוע מראש, ולכן גם גודלו של קוד המבונה אינו צפוי מראש. אולם בכך להקל בימוש האסטבלר, מותר להניח גודל מסוימלי. לפיכך יש אפשרות להשתמש במערכות לאחסן תמונות קוד המבונה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים), יש למשם באופן ייעיל וחסכוני (למשל באמצעות רשימה מקושרת והקצאת זיכרון דינמי).
- השמות של קבועי הפלט צריכים להיות תואמים לשם קבוע הפלט, למעט הסיומות. למשל, אם קבוע הקלט הוא `as` אז קבוע קבוע הפלט שיווצרו הם : `mt`, `prog.ob`, `prog.ext`, `prog.ent`.
- מתכונת הפעלת האסטבלר צריכה להיות כפי הנדרש בממ"ן, ללא שינויים כלשהם. ככלומר, ממשך המשמש יהיה אך ורק באמצעות שורת הפוקודה. בפרט, שמורות קבועי המבוקר יועברו לתוכנית האסטבלר כארגומנטים (אחד או יותר) בשורת הפוקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גרפיים למיניהם, ועוד.
- יש להקפיד לחלק את שימוש האסטבלר למספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסוימים ביחד במודול אחד. מומלץ לחלק למודולים כגון : מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחבירי של שורה), טבלת הסמלים, מפת הזיכרון, טבלאות קבועות (קוד הפעולה, שיטות המיעון החוקיקות לכל פעולה, ועוד).
- יש להקפיד ולתעד את השימוש באופן מלא וברור, באמצעות העורות מפורטות בקוד.
- יש לאפשר תווים לבנים עדפים בקובץ הקלט בשפת אסטבלרי. למשל, אם בשורת הוראה יש שני אופרנדים המופרדים בפסיק, אזי לפני ואחרי הפסיק מותר שייהיו רווחים וטאים בכל 경우에는. בדומה, גם לפני ואחרי שם הפעולה. מותר גם שורות ריקות. האסטבלר יתעלם מהתווים לבנים מיותרים (כלומר ידלג עליהם).
- הקלט (קוד האסטבלר) עלול להכיל שגיאות תחביריות. על האסטבלר לגלות ולדוח על כל השורות השגויות בקלט. אין לעצור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להדפיס למסך הודעות מפורטות ככל הנitin, כדי שאפשר יהיה להבין מה והיכן כל שגיאה. כמובן שם קובץ קלט מכיל שגיאות, אין טעם להפיק עבورو את קבוע הפלט (`ob`, `ext`, `ent`).

גם ונשלם פרק ההסבירים והגדרת הפרויקט.

**בשאלות ניתן לפנות לקובצת הדיוון באתר הקורס, ועל כל אחד מהמנחים בשעות הקבלה שלהם.**

להזיכרים, באפשרותו של כל סטודנט לפנות לכל מנהה, לאו דווקא למנהל הקבוצה שלו, לקבלת עזרה. שוב מומלץ לכל אלה שטרם בדקו את התכנים באתר הקורס לעשות זאת. נשאלות באתר זה הרובה שאלות בנושא חומר הלימוד והממי"נים, והתשובות יכולות להועיל לכם.

لتשומתיכם : לא ניתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילאים או מחלה ממושכת. במקרים אלו יש לבקש ולקבל אישור מראש מצוות הקורס.

**בהצלחה !**