

# מטלת מנחה (ממ"ן) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה : פרויקט גמר

מספר השאלות : 1 משקל המטלה : 31 נקודות (חובה)

סמסטר : 2020' מועד אחרון להגשה : 16.08.2020

## קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
  - שליחת מטלות באמצעות דואר אלקטרוני - באישור המנחה בלבד
- הסבר מפורט ב"נוהל הגשת מטלות מנחה"**

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

עליכם לכתוב תוכנת אסמבלר, עבור שפת אסמבלי שתוגדר בהמשך. הפרויקט ייכתב בשפת C.

עליכם להגיש את הפריטים הבאים :

1. קבצי המקור של התוכנית שכתבתם (קבצים בעלי סיומת c או h).
  2. קובץ הרצה (מקומפל ומקושר) עבור מערכת אונוטו.
  3. קובץ makefile. הקימפול חייב להיות עם הקומפיילר gcc והדגלים : -Wall -ansi -pedantic. יש לנפות את כל ההודעות שמוציא הקומפיילר, כך שהתוכנית תתקמפל ללא כל הערות או אזהרות.
  4. דוגמאות הרצה (קלט ופלט) :
- א.** קבצי קלט בשפת אסמבלי, וקבצי הפלט שנוצרו מהפעלת האסמבלר על קבצי קלט אלה. יש להדגים שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האסמבלי.
- ב.** קבצי קלט בשפת אסמבלי המדגימים מגוון רחב של סוגי שגיאות אסמבלי (ולכן לא נוצרים קבצי פלט), ותדפיסי המסך המראים את הודעות השגיאה שמוציא האסמבלר.

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

נזכיר מספר היבטים חשובים של כתיבת קוד טוב :

1. הפשטה של מבני הנתונים : רצוי (ככל האפשר) להפריד בין הגישה למבני הנתונים לבין המימוש של מבני הנתונים. כך, למשל, בעת כתיבת פונקציות לטיפול בטבלה, אין זה מעניינם של המשתמשים בפונקציות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקושרת.
2. קריאות הקוד : יש להשתמש בשמות משמעותיים למשתנים ופונקציות. יש לערוך את הקוד באופן מסודר : הזחות עקביות, שורות ריקות להפרדה בין קטעי קוד, וכד'.
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 טובה 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 ותייעוד ברמה טובה, כמתואר לעיל, אשר משקלם המשותף מגיע עד לכ- 40% ממשקל הפרויקט.

מותר להשתמש בפרויקט בכל מגוון הספריות הסטנדרטיות של שפת C, אבל אין להשתמש בספריות חיצוניות אחרות.

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

מומלץ לקרוא את הגדרת הפרויקט פעם ראשונה ברצף, לקבלת תמונה כללית לגבי הנדרש, ורק לאחר מכן לקרוא שוב בצורה מעמיקה יותר.

### **רקע כללי ומטרת הפרויקט**

כידוע, קיימות שפות תכנות רבות, ומספר גדול של תוכניות, הכתובות בשפות שונות, עשויות לרוץ באותו מחשב עצמו. כיצד "מכיר" המחשב כל כך הרבה שפות? התשובה פשוטה: המחשב מכיר למעשה שפה אחת בלבד: הוראות ונתונים הכתובים בקוד בינארי. קוד זה מאוחסן בגוש בזיכרון, ונראה כמו רצף של ספרות בינאריות. יחידת העיבוד המרכזית - היע"מ (CPU) - יודעת לפרק את הרצף הזה לקטעים קטנים בעלי משמעות: הוראות, מענים ונתונים.

למעשה, זיכרון המחשב כולו הוא אוסף של סיביות, שנוהגים לראותן כמקובצות ליחידות בעלות אורך קבוע (בתים, מילים). לא ניתן להבחין, בעין שאינה מיומנת, בהבדל פיסי כלשהו בין אותו חלק בזיכרון שבו נמצאת תוכנית לבין שאר הזיכרון.

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מגוון פעולות פשוטות, הנקראות **הוראות מכונה**, ולשם כך היא משתמשת באוגרים (registers) הקיימים בתוך היע"מ, ובזיכרון המחשב. **דוגמאות**: העברת מספר מתא בזיכרון לאוגר ביע"מ או בחזרה, הוספת 1 למספר הנמצא באוגר, בדיקה האם מספר המאוחסן באוגר שווה לאפס, חיבור וחסור בין שני אוגרים, וכד'. הוראות המכונה ושילובים שלהן הן המרכיבות תוכנית כפי שהיא טעונה לזיכרון בזמן ריצתה. כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנת), תתורגם בסופו של דבר באמצעות תוכנה מיוחדת לצורה סופית זו.

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

כידוע, לכל שפת תכנות עילית יש מהדר (compiler), או מפרש (interpreter), המתרגם תוכניות מקור לשפת מכונה. האסמבלר משמש בתפקיד דומה עבור שפת אסמבלי.

לכל מודל של יע"מ (כלומר לכל אירגון של מחשב) יש שפת מכונה יעודית משלו, ובהתאם גם שפת אסמבלי יעודית משלו. לפיכך, גם האסמבלר (כלי התרגום) הוא יעודי ושונה לכל יע"מ.

תפקידו של האסמבלר הוא לבנות קובץ המכיל קוד מכונה, מקובץ נתון של תוכנית הכתובה בשפת אסמבלי. זהו השלב הראשון במסלול אותו עוברת התוכנית, עד לקבלת קוד המוכן לריצה על חומרת המחשב. השלבים הבאים הם קישור (linkage) וטעינה (loading), אך בהם לא נעסוק בממ"ן זה.

המשימה בפרויקט זה היא לכתוב אסמבלר (כלומר תוכנית המתרגמת לשפת מכונה), עבור שפת אסמבלי שנגדיר כאן במיוחד לצורך הפרויקט.

**לתשומת לב**: בהסברים הכלליים על אופן עבודת תוכנת האסמבלר, 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 האסמבלר. אין לטעות: עליכם לכתוב את תוכנית האסמבלר בלבד. **אין** לכתוב את תוכניות הקישור והטעינה!!!

## המחשב הדמיוני ושפת האסמבלי

נגדיר עתה את שפת האסמבלי ואת מודל המחשב הדמיוני, עבור פרויקט זה.

הערה: תאור מודל המחשב להלן הוא חלקי בלבד, ככל שנחוץ לביצוע המשימות בפרויקט.

"חומרה":

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

למעבד 8 אוגרים כלליים, בשמות:  $r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7$ . גודלו של כל אוגר הוא 24 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 23. שמות האוגרים נכתבים תמיד עם אות 'r' קטנה.

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

גודל הזיכרון הוא  $2^{21}$  תאים, בכתובות  $0 - 2^{21}-1$ , וכל תא הוא בגודל של 24 סיביות. לתא בזיכרון נקרא גם בשם **"מילה"**. הסיביות בכל מילה ממוספרות כמו באוגר.

מחשב זה עובד רק עם מספרים שלמים חיוביים ושלייליים. אין תמיכה במספרים ממשיים. האריתמטיקה נעשית בשיטת המשלים ל-2 (2's complement). כמו כן יש תמיכה בתווים (characters), המיוצגים בקוד ascii.

מבנה הוראות המכונה:

כל הוראת מכונה במודל שלנו מורכבת מפעולה ואופרנדים. מספר האופרנדים הוא בין 0 ל-2, בהתאם לסוג הפעולה. מבחינת התפקיד של כל אופרנד, נבחין בין אופרנד מקור (source) ואופרנד יעד (destination).

כל הוראת מכונה מקודדת למספר 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 המכונה שבונה האסמבלר, כל מילה תקודד בבסיס הקסאדצימלי (ראו פרטים לגבי קבצי פלט בהמשך).

בכל סוגי הוראות המכונה, **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						מיעון מקור		אוגר מקור		מיעון יעד		אוגר יעד		funct					A	R	E		

במודל המכונה שלנו יש 16 פעולות, בפועל, למרות שניתן לקודד יותר פעולות. כל פעולה מיוצגת בשפת אסמבלי באופן סימבולי על ידי **שם-פעולה**, ובקוד המכונה על ידי קומבינציה ייחודית של ערכי שני שדות במילה הראשונה של ההוראה: **קוד-הפעולה (opcode)**, ו**פונקציה (funct)**.

להלן טבלת הפעולות :

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

הערה : שם-הפעולה נכתב תמיד באותיות קטנות. פרטים על מהות הפעולות השונות יובאו בהמשך.

להלן מפרט השדות במילה הראשונה בקוד המכונה של כל הוראה.

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

**סיביות 3-7 :** שדה זה, הנקרא **funct**, מתפקד כאשר מדובר בפעולה שקוד-הפעולה (opcode) שלה משותף לכמה פעולות שונות (כאמור, קודי-פעולה 2, 5 או 9). השדה funct יכול ערך ייחודי לכל פעולה מקבוצת הפעולות שיש להן אותו קוד-פעולה. אם קוד-הפעולה משמש לפעולה 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 יינתן 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. במילה הראשונה של כל הוראה, ערך הסיבית A תמיד 1, ושתי הסיביות האחרות (R,E) מאופסות.

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

## שיטות מיעון:

בשפת האסמבלי שלנו קיימות ארבע שיטות מיעון, המסומנות במספרים 0,1,2,3. השימוש בחלק משיטות המיעון מצריך מילות-מידע נוספות בקוד המכונה של הוראת המכונה, בנוסף למילה הראשונה.

לכל אופרנד של ההוראה נדרשת **לכל 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 היעד.

כל מילת-מידע נוספת של ההוראה מקודדת באחד משלושה סוגים של קידוד. **סיביות 0-2** של כל מילת-מידע הן השדה 'A,R,E', 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) מציינת שקידוד המילה הוא מוחלט (Absolute), ואינו מצריך שינוי בשלבי הקישור והטעינה.
- סיבית 1 (הסיבית R) מציינת שהקידוד הוא של כתובת פנימית הניתנת להזזה (Relocatable), ומצריך שינוי בשלבי הקישור והטעינה.
- סיבית 0 (הסיבית E) מציינת שהקידוד הוא של כתובת חיצונית (External), ומצריך שינוי בשלבי הקישור והטעינה.

הסבר על התפקיד של השדה 'A,R,E' בקוד המכונה יבוא בהמשך. ערך השדה 'A,R,E' הנדרש בכל אחת משיטות המיעון מופיע בתיאור שיטות המיעון להלן.

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	אופן כתיבת האופרנד	דוגמה
0	מיעון מיידי	מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר שלם בשיטת המשלים ל-2, ברוחב של 21 סיביות, השוכן בסיביות 23-3 של המילה.  הסיביות 2-0 של מילת המידע הן השדה A,R,E. במיעון מיידי, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני.	mov #-1, r2  בדוגמה זו האופרנד הראשון של הפקודה (אופרנד המקור) נתון בשיטת מיעון מיידי. ההוראה כותבת את הערך 1- אל אוגר r2.
1	מיעון ישיר	מילת-מידע נוספת של ההוראה מכילה כתובת בזיכרון. המילה בכתובת זו בזיכרון היא האופרנד.  הכתובת מיוצגת כמספר <u>ללא סימן</u> ברוחב של 21 סיביות, בסיביות 23-3 של מילת המידע.  הסיביות 2-0 במילת המידע הן השדה A,R,E. במיעון ישיר, ערך הסיביות האלה תלוי בסוג הכתובת הרשומה בסיביות 23-3. אם זוהי כתובת שמייצגת שורה בקובץ המקור הנוכחי (כתובת פנימית), ערך הסיבית R הוא 1, ושתי הסיביות האחרות מאופסות. ואילו אם זוהי כתובת שמייצגת שורה בקובץ מקור אחר של התוכנית (כתובת חיצונית), ערך הסיבית E הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד הוא <u>תווית</u> שכבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת התווית בתחילת השורה של הנחית 'data'. או 'string', או בתחילת השורה של הוראה, או באמצעות אופרנד של הנחית 'extern'.  התווית מייצגת באופן סימבולי כתובת בזיכרון.	השורה הבאה מגדירה את התווית x: x: .data 23  ההוראה: dec x  מקטינה ב-1 את תוכן 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 שתבצע נמצאת בכתובת next).  הכתובת next תקודד בסיביות 23-3 של מילת המידע הנוספת.

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	אופן כתיבת האופרנד	דוגמה
2	מיעון יחסי	<p>שיטה זו רלוונטית <u>אך</u> <u>ורק</u> להוראות המבצעות קפיצה (הסתעפות) להוראה אחרת. מדובר בקודי-הפעולה הבאים בלבד: jmp, bne, jsr. <u>לא ניתן</u> להשתמש בשיטה זו בהוראות עם קודי-פעולה אחרים.</p> <p>בשיטה זו, יש בקידוד ההוראה מילת מידע נוספת המכילה את מרחק הקפיצה, במילות זיכרון, מכתובת ההוראה הנוכחית (פקודת הקפיצה) אל כתובת ההוראה המבוקשת (ההוראה הבאה לביצוע).</p> <p>מרחק הקפיצה מיוצג כמספר עם סימן בשיטת המשלים ל-2 ברוחב של 21 סיביות, השוכן בסיביות 23-3 של מילת המידע הנוספת. מרחק זה יהיה שלילי במקרה שהקפיצה היא אל הוראה שבכתובת יותר נמוכה, וחיובי במקרה שהקפיצה היא אל הוראה שבכתובת יותר גבוהה.</p> <p>הסיביות 2-0 של מילת המידע הן השדה A,R,E. במיעון יחסי, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.</p>	<p>האופרנד מתחיל בתו &amp; ולאחריו ובצמוד אליו מופיע שם של תווית.</p> <p>התווית מייצגת באופן סימבולי כתובת של <u>הוראה בקובץ המקור הנוכחי של התוכנית</u>.</p> <p>ייתכן שהתווית כבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת התווית בתחילת שורת הוראה.</p> <p>יודגש כי בשיטת מיעון יחסי <u>לא ניתן</u> להשתמש בתווית (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 שתבצע נמצאת בכתובת next).</p> <p>נניח כי ההוראה jmp שבדוגמה נמצאת בכתובת 500 (עשרוני). כמו כן, נניח כי התווית next מוגדרת בקובץ המקור הנוכחי בכתובת 300 (עשרוני).</p> <p>מרחק הקפיצה אל ההוראה בכתובת next הוא 200-, ומרחק זה יקודד בסיביות 23-3 של מילת המידע הנוספת.</p>
3	מיעון אוגר ישיר	<p>האופרנד הוא אוגר. לשיטת מיעון זו אין מילת מידע נוספת. מספרו של האוגר מקודד במילה הראשונה של ההוראה, בשדה המתאים: אוגר מקור/יעד.</p>	האופרנד הוא שם של אוגר.	<p>clr r1</p> <p>בדוגמה זו, ההוראה clr מאפסת את תוכן האוגר r1.</p>

#### מפרט 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 הנוכחית שמתבצעת** (הכוונה תמיד לכתובת המילה הראשונה של ההוראה).

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

#### **קבוצת ההוראות הראשונה:**

אלו הן הוראות הדורשות שני אופרנדים.

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

הוראה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
mov	0		מבצעת העתקה של תוכן אופרנד המקור (האופרנד הראשון) אל אופרנד היעד (האופרנד השני).	mov A, r1	העתק את תוכן המשתנה A (המילה שבכתובת A בזיכרון) אל אוגר r1.
cmp	1		מבצעת השוואה בין שני האופרנדים. ערך אופרנד היעד (השני) מופחת מערך אופרנד המקור (הראשון), ללא שמירת תוצאת החיסור. פעולת החיסור מעדכנת דגל בשם Z ("דגל האפס") באוגר הסטטוס (PSW).	cmp A, r1	אם תוכן המשתנה A זהה לתוכנו של אוגר r1 אזי הדגל Z ("דגל האפס") באוגר הסטטוס (PSW) יודלק, אחרת הדגל יאופס.
add	2	1	אופרנד היעד (השני) מקבל את תוצאת החיבור של אופרנד המקור (הראשון) והיעד (השני).	add A, r0	אוגר r0 מקבל את תוצאת החיבור של תוכן המשתנה A ותוכנו הנוכחי של r0.
sub	2	2	אופרנד היעד (השני) מקבל את תוצאת החיסור של אופרנד המקור (הראשון) מאופרנד היעד (השני).	sub #3, r1	אוגר r1 מקבל את תוצאת החיסור של הקבוע 3 מתוכנו הנוכחי של האוגר r1.
lea	4		lea הוא קיצור (ראשי תיבות) של load effective address. פעולה זו מציבה את המען בזיכרון המיוצג על ידי התווית שבאופרנד הראשון (המקור), אל אופרנד היעד (האופרנד השני).	lea HELLO, r1	המען שמייצגת התווית HELLO מוצב לאוגר r1.

#### קבוצת ההוראות השניה:

אלו הן הוראות הדורשות אופרנד אחד בלבד. אופן הקידוד של האופרנד הוא כמו של אופרנד היעד בפקודה עם שני אופרנדים. השדות של אופרנד המקור (סיביות 13-17) במילה הראשונה בקידוד ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השייכות לקבוצה זו הן: clr, not, inc, dec, jmp, bne, jsr, red, prn :

הוראה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
clr	5	1	איפוס תוכן האופרנד	clr r2	האוגר r2 מקבל את הערך 0.
not	5	2	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0).	not r2	כל ביט באוגר r2 מתהפך.
inc	5	3	הגדלת תוכן האופרנד באחד.	inc r2	תוכן האוגר r2 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 המיוצג על ידי האופרנד. כלומר, כתוצאה מביצוע ההוראה, מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה.	jmp &Line	$PC \leftarrow PC + \text{distanceTo}(\text{Line})$ מצביע התכנית מקבל את המען שמחושב על ידי חיבור המרחק לתווית Line עם מען ההוראה הנוכחית, ולפיכך ההוראה הבאה שתבצע תהיה במען Line.
bne	9	2	bne הוא קיצור (ראשי תיבות) של: branch if not equal (to zero). זוהי הוראת הסתעפות מותנית. אם ערכו של הדגל Z באוגר הסטטוס (PSW) הינו 0, אזי מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה. כזכור, הדגל Z נקבע באמצעות הוראת cmp.	bne Line	אם ערך הדגל Z באוגר הסטטוס (PSW) הוא 0, אזי $PC \leftarrow \text{address}(\text{Line})$ מצביע התכנית יקבל את כתובת התווית Line, ולפיכך ההוראה הבאה שתבצע תהיה במען Line.

הוראה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
jsr	9	3	קריאה לשגרה (סברוטניה). כתובת ההוראה שאחרי הוראת jsr הנוכחית (PC+2) נדחפת לתוך המחסנית שבזיכרון המחשב, ומצביע התוכנית (PC) מקבל את כתובת השגרה. הערה: חזרה מהשגרה מתבצעת באמצעות הוראת rts, תוך שימוש בכתובת שבמחסנית.	jsr SUBR	push(PC+2) PC ← address(SUBR) מצביע התוכנית יקבל את כתובת התווית SUBR, ולפיכך, ההוראה הבאה שתבצע תהיה במען SUBR. כתובת החזרה מהשגרה נשמרת במחסנית.
red	12		קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מהקלט ייכנס לאוגר r1.
prn	13		הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn r1	ידפס לפלט התו (קוד ascii) הנמצא באוגר r1

### קבוצת ההוראות השלישית:

אלו הן הוראות ללא אופרנדים. קידוד ההוראה מורכב ממילה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד (סיביות 8-17) במילה הראשונה של קידוד ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השייכות לקבוצה זו הן: rts, stop.

הוראה	opcode	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
rts	14	מתבצעת חזרה משיגרה. הערך שבראש המחסנית של המחשב מוצא מן המחסנית, ומוכנס למצביע התוכנית (PC). הערה: ערך זה נכנס למחסנית בקריאה לשגרה ע"י הוראת jsr	rts	PC ← pop() ההוראה הבאה שתבצע תהיה זו שאחרי הוראת jsr שקראה לשגרה.
stop	15	עצירת ריצת התוכנית.	stop	התוכנית עוצרת מיידית.

### מבנה תכנית בשפת אסמבלי:

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

אורכה של שורה בקובץ המקור הוא 80 תווים לכל היותר (לא כולל התו \n).

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

סוג המשפט	הסבר כללי
משפט ריק	זוהי שורה המכילה אך ורק תווים לבנים (whitespace), כלומר רק את התווים ' ' ו- '\t' (רווחים וטאבים). ייתכן ובשורה אין אף תו (למעט התו \n, כלומר השורה ריקה).
משפט הערה	זוהי שורה בה התו הראשון הינו ';' (נקודה פסיק). על האסמבלר להתעלם לחלוטין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמבלר מה עליו לעשות כשהוא פועל על תוכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצאת זיכרון ואתחול משתנים של התוכנית, אך הוא אינו מייצר קידוד של הוראות מכונה המיועדות לביצוע בעת ריצת התוכנית.
משפט הוראה	זהו משפט המייצר קידוד של הוראות מכונה לביצוע בעת ריצת התוכנית. המשפט מורכב משם של הוראה שעל המעבד לבצע, ותיאור האופרנדים של ההוראה.



כעת נפרט יותר לגבי סוגי המשפטים השונים.

### משפט הנחיה:

משפט הנחיה הוא בעל המבנה הבא:

בתחילת המשפט יכולה להופיע הגדרה של תווית (label). לתווית יש תחביר חוקי שיתואר בהמשך. התווית היא אופציונאלית.

לאחר מכן מופיע שם ההנחיה. לאחר שם ההנחיה יופיעו פרמטרים (מספר הפרמטרים בהתאם להנחיה).

שם של הנחיה מתחיל בתו ' ' (נקודה) ולאחריו תווים באותיות קטנות (lower case) בלבד.

**יש לשים לב:** למילים בקוד המכונה הנוצרות ממשפט הנחיה לא מצורף השדה A,R,E, והערך המוגדר על ידי ההנחיה ממלא את כל 24 הסיביות של המילה.

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

#### 1. ההנחיה 'data'.

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

data 7, -57, +17, 9

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

המשפט 'data' מנחה את האסמבלר להקצות מקום בתמונת הנתונים (data image), אשר בו יאוחסנו הערכים של הפרמטרים, ולקדם את מונה הנתונים, בהתאם למספר הערכים. אם בהנחית data מוגדרת תווית, אזי תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום), ומוכנסת אל טבלת הסמלים. דבר זה מאפשר להתייחס אל מקום מסוים בתמונת הנתונים דרך שם התווית (למעשה, זוהי דרך להגדיר שם של משתנה).

כלומר אם נכתוב:

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

אזי יוקצו בתמונת הנתונים ארבע מילים רצופות שיכילו את המספרים שמופיעים בהנחיה. התווית XYZ מזוהה עם כתובת המילה הראשונה.

אם נכתוב בתוכנית את ההוראה:

mov XYZ, r1

אזי בזמן ריצת התוכנית יוכנס לאוגר r1 הערך 7.

ואילו ההוראה:

lea XYZ, r1

תכניס לאוגר r1 את ערך התווית XYZ (כלומר הכתובת בזיכרון בה מאוחסן הערך 7).

#### 2. ההנחיה 'string'.

להנחיה 'string' פרמטר אחד, שהוא מחרוזת חוקית. תווי המחרוזת מקודדים לפי ערכי ה-ascii המתאימים, ומוכנסים אל תמונת הנתונים לפי סדרם, כל תו במילה נפרדת. בסוף המחרוזת

יתווסף התן '0' (הערך המספרי 0), המסמן את סוף המחרוזת. מונה הנתונים של האסמבלר יקודם בהתאם לאורך המחרוזת (בתוספת מקום אחד עבור התו המסיים). אם בשורת ההנחיה 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, ההנחיה:

```
STR: .string "abcdef"
```

מקצה בתמונת הנתונים רצף של 7 מילים, ומאתחלת את המילים לקודי ה-ascii של התווים לפי הסדר במחרוזת, ולאחריהם הערך 0 לסימון סוף מחרוזת. התווית STR מזוהה עם כתובת התחלת המחרוזת.

3. ההנחיה 'entry'.

להנחיה 'entry' פרמטר אחד, והוא שם של תווית המוגדרת בקובץ המקור הנוכחי (כלומר תווית שמקבלת את ערכה בקובץ זה). מטרת ההנחיה entry. היא לאפיין את התווית הזו באופן שיאפשר לקוד אסמבלי הנמצא בקבצי מקור אחרים להשתמש בה (כאופרנד של הוראה).

לדוגמה, השורות:

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

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

**לתשומת לב:** תווית המוגדרת בתחילת שורת entry. הינה חסרת משמעות והאסמבלר **מתעלם** מתווית זו (אפשר שהאסמבלר יוציא הודעת אזהרה).

4. ההנחיה 'extern'.

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

נשים לב כי הנחיה זו תואמת להנחית 'entry'. המופיעה בקובץ בו מוגדרת התווית. בשלב הקישור תתבצע התאמה בין ערך התווית, כפי שנקבע בקוד המכונה של הקובץ שהגדיר את התווית, לבין קידוד ההוראות המשתמשות בתווית בקבצים אחרים (שלב הקישור אינו רלוונטי לממ"ן זה).

לדוגמה, משפט ההנחיה 'extern'. התואם למשפט ההנחיה 'entry'. מהדוגמה הקודמת יהיה:

```
extern HELLO
```

**לתשומת לב:** תווית המוגדרת בתחילת שורת extern. הינה חסרת משמעות והאסמבלר **מתעלם** מתווית זו (אפשר שהאסמבלר יוציא הודעת אזהרה).

### משפט הוראה:

משפט הוראה מורכב מהחלקים 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 תמונת הקוד שבונה האסמבלר.

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 בתו ' ', 'פסיק'. 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

#### תווית :

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

**הגדרה של תווית** מסתיימת בתו ' ' (נקודתיים). תו זה אינו מהווה חלק מהתווית, אלא רק סימן המציין את סוף ההגדרה. התו ' ' חייב להיות צמוד לתווית (ללא רווחים).

אסור שאותה תווית תוגדר יותר מפעם אחת (כמובן בשורות שונות). אותיות קטנות וגדולות נחשבות שונות זו מזו.

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hEllo:

x:

He78902:

**לתשומת לב :** מילים שמורות של שפת האסמבלי (כלומר שם של פעולה או הנחיה, או שם של אוגר) אינן יכולות לשמש גם כשם של תווית.

התווית מקבלת את ערכה בהתאם להקשר בו היא מוגדרת. תווית המוגדרת בהנחיות data או string, תקבל את ערך מונה הנתונים (data counter) הנוכחי, בעוד שתווית המוגדרת בשורת הוראה תקבל את ערך מונה ההוראות (instruction counter) הנוכחי.

#### מספר :

מספר חוקי מתחיל בסימן אופציונלי : ' - ' או ' + ' ולאחיו סדרה של ספרות בבסיס עשרוני. לדוגמה : 76, -5, +123 הם מספרים חוקיים. אין תמיכה בשפת אסמבלי בייצוג בבסיס אחר מאשר עשרוני, ואין תמיכה במספרים שאינם שלמים.

## מחרוזות:

מחרוזות חוקית היא סדרת תווי ascii נראים (שניתנים להדפסה), המוקפים במרכאות כפולות (המרכאות אינן נחשבות חלק מהמחרוזת). דוגמה למחרוזת חוקית: "hello world".

## תפקיד השדה A,R,E בקוד המכונה

בכל מילה בקוד המכונה של הוראה (לא של נתונים), האסמבלר מכניס מידע עבור תהליך הקישור והטעינה. זהו השדה A,R,E (שלוש הסיביות הימניות 2,1,0 בהתאמה). המידע ישמש לתיקונים בקוד בכל פעם שייטען לזיכרון לצורך הרצה. האסמבלר בונה מלכתחילה קוד שמיועד לטעינה החל מכתובת 100. התיקונים יאפשרו לטעון את הקוד בכל פעם למקום אחר, בלי צורך לחזור על תהליך האסמבלר.

בכל מילה של ההוראה, בדיוק אחת משלש הסיביות של השדה A,R,E מכילה 1, ושתי הסיביות האחרות מאופסות. מפרט שיטות המיעון שהוצג קודם מציין איזו סיבית תכיל 1 בכל שיטת מעון.

סיבית 'A' (קיצור של Absolute) באה לציין שתוכן המילה אינו תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל מילה המכילה אופרנד מיד).

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

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

## **אסמבלר עם שני מעברים**

כאשר מקבל האסמבלר תוכנית בשפת אסמבלי, עליו לעבור על התוכנית פעמיים. במעבר הראשון, יש לזהות את הסמלים (תוויות) המופיעים בתוכנית, ולתת לכל סמל ערך מספרי שהוא המען בזיכרון שהסמל מייצג. במעבר השני, באמצעות ערכי הסמלים, וכן קודי-הפעולה ומספרי האוגרים, בונים את קוד המכונה.

לדוגמה: האסמבלר 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 המכונה 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 0000101	MAIN: add r3, LIST	First word of instruction Address of label LIST	000010110110100000001100 000000000000001111111010
0000102 0000103	LOOP: prn #48	Immediate value 48	00110100000000000000100 00000000000000110000100
0000104 0000105	lea STR, r6	Address of label STR	000100010001111000000100 000000000000001111010010
0000106	inc r6		000101000001111000011100
0000107 0000108	mov r3, K	Address of label K	000000110110100000000100 00000000000010000010010
0000109	sub r1, r4		000010110011110000010100
0000110 0000111	bne END	Address of label END	001001000000100000010100 000000000000001111001010
0000112 0000113 0000114	cmp K, #-6	Address of label K Immediate value -6	000001010000000000000100 00000000000010000010010 1111111111111111010100
0000115 0000116	bne &END	Distance to label END	001001000001000000010100 000000000000000000110100
0000117 0000118	dec K	Address of label K	000101000000100000100100 00000000000010000010010
0000119 0000120	jmp &LOOP	Distance to label LOOP	001001000001000000001100 111111111111111101111100
0000121	END: stop		001111000000000000000100
0000122	STR: .string "abcd"	Ascii code 'a'	00000000000000001100001
0000123		Ascii code 'b'	00000000000000001100010
0000124		Ascii code 'c'	00000000000000001100011
0000125		Ascii code 'd'	00000000000000001100100
0000126		Ascii code '\0'	000000000000000000000000
0000127 0000128	LIST: .data 6, -9	Integer 6 Integer -9	00000000000000000000110 111111111111111111110111
0000129	.data -100	Integer -100	111111111111111110011100
0000130	K: .data 31	Integer 31	000000000000000000001111

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

כדי לבצע המרה לבינארי של אופרנדים שכתובים בשיטות מיעון המשתמשות בסמלים (תוויות), יש צורך לבנות טבלה המכילה את ערכי כל הסמלים. אולם בהבדל מהקודים של הפעולות, הידועים מראש, הרי המענים בזיכרון עבור הסמלים שבשימוש התוכנית אינם ידועים, עד אשר תוכנית המקור נסרקה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד לעיל, האסמבלר אינו יכול לדעת שהסמל END אמור להיות משויך למען 121 (עשרוני), ושהסמל K אמור להיות משויך למען 130, אלא רק לאחר שנקראו כל שורות התוכנית.

לכן מפרידים את הטיפול של האסמבלר בסמלים לשני שלבים. בשלב הראשון בונים טבלה של כל הסמלים, עם הערכים המספריים המשוויכים להם, ובשלב השני מחליפים את כל הסמלים, המופיעים באופרנדים של ההוראות 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 משויך 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 השני צריכים הערכים של כל הסמלים להיות כבר ידועים.

לתשומת לב : תפקיד האסמבלר, על שני המעברים שלו, לתרגם קובץ מקור לקוד בשפת מכונה. בגמר פעולת האסמבלר, התוכנית טרם מוכנה לטעינה לזיכרון לצורך ביצוע. קוד המכונה חייב לעבור לשלבי הקישור/טעינה, ורק לאחר מכן לשלב הביצוע (שלבים אלה אינם חלק מהממ"ן).

### המעבר הראשון

במעבר הראשון נדרשים כללים כדי לקבוע איזה מען ישויד לכל סמל. העיקרון הבסיסי הוא לספור את המקומות בזיכרון, אותם תופסות ההוראות. אם כל הוראה תיטען בזיכרון למקום העוקב להוראה הקודמת, תציין ספירה כזאת את מען ההוראה הבאה. הספירה נעשית על ידי האסמבלר ומוחזקת במונה ההוראות (IC). ערכו ההתחלתי של IC הוא 100 (עשרוני), ולכן קוד המכונה של ההוראה הראשונה נבנה כך שייטען לזיכרון החל ממען 100. ה-IC מתעדכן בכל שורת הוראה המקצה מקום בזיכרון. לאחר שהאסמבלר קובע מהו אורך ההוראה, ה-IC מוגדל במספר התאים (מילים) הנתפסים על ידי ההוראה, וכך הוא מצביע על התא הפנוי הבא.

כאמור, כדי לקודד את ההוראות בשפת מכונה, מחזיק האסמבלר טבלה, שיש בה קידוד מתאים לכל שם פעולה. בזמן התרגום מחליף האסמבלר כל שם פעולה בקידוד שלה. כמו כן, כל אופרנד מוחלף בקידוד מתאים, אך פעולת החלפה זו אינה כה פשוטה. ההוראות משתמשות בשיטות מיעון מגוונות לאופרנדים. אותה פעולה יכולה לקבל משמעויות שונות, בכל אחת משיטות המיעון, ולכן יתאימו לה קידודים שונים לפי שיטות המיעון. לדוגמה, פעולת ההזזה mov יכולה להתייחס להעתקת תוכן תא זיכרון לאוגר, או להעתקת תוכן אוגר לאוגר אחר, וכן הלאה. לכל אפשרות כזאת של mov עשוי להתאים קידוד שונה.

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

כאשר נתקל האסמבלר בתווית המופיעה בתחילת השורה, הוא יודע שלפניו הגדרה של תווית, ואז הוא משייד לה מען – תוכנו הנוכחי של ה-IC. כך מקבלות כל התוויות את מעניהן בעת ההגדרה. תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנוסף לשם התווית גם את המען ומאפיינים נוספים. כאשר תהיה התייחסות לתווית באופרנד של הוראה כלשהי, יוכל האסמבלר לשלוף את המען המתאים מטבלת הסמלים.

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

```

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

כאשר מגיע האסמבלר לשורת ההסתעפות (bne A), הוא טרם נתקל בהגדרת התווית A וכמובן לא יודע את המען המשויד לתווית. לכן האסמבלר לא יכול לבנות את הקידוד הבינארי של האופרנד &A. נראה בהמשך כיצד נפתרת בעיה זו.

בכל מקרה, תמיד אפשר לבנות במעבר הראשון את הקידוד הבינארי המלא של המילה הראשונה של כל הוראה, את הקידוד הבינארי של מילת-המידע הנוספת של אופרנד מייד, וכן את הקידוד הבינארי של כל הנתונים (המתקבלים מההנחות .data, .string).

### המעבר השני

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

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

### הפרדת הוראות ונתונים

בתוכנית מבחינים בשני סוגים של תוכן: הוראות ונתונים. יש לארגן את קוד המכונה כך שתהיה הפרדה בין הנתונים וההוראות. הפרדת ההוראות והנתונים לקטעים שונים בזיכרון היא שיטה עדיפה על פני הצמדה של הגדרות הנתונים להוראות המשתמשות בהן.

אחת הסכנות הטמונות באי הפרדת ההוראות מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית בתוכנית, לנסות "לבצע" את הנתונים כאילו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום תופעה כזו היא הסתעפות לא נכונה. התוכנית כמובן לא תעבוד נכון, אך לרוב הנזק הוא יותר חמור, כי נוצרת חריגת חומרה ברגע שהמעבד מבצע פעולה שאינה חוקית.

האסמבלר שלנו חייב להפריד, בקוד המכונה שהוא מיצר, בין קטע הנתונים לקטע ההוראות. כלומר **בקובץ הפלט (בקוד המכונה) תהיה הפרדה של הוראות ונתונים לשני קטעים נפרדים**, אם כי **בקובץ הקלט אין חובה שתהיה הפרדה כזו**. בהמשך מתואר אלגוריתם של האסמבלר, ובו פרטים כיצד לבצע את ההפרדה.

### גילוי שגיאות בתוכנית המקור

האסמבלר אמור לגלות ולדווח על שגיאות בתחביר של תוכנית המקור, כגון פעולה שאינה קיימת, מספר אופרנדים שגוי, סוג אופרנד שאינו מתאים לפעולה, שם אוגר לא קיים, ועוד שגיאות אחרות. כמו כן מוודא האסמבלר שכל סמל מוגדר פעם אחת בדיוק.

מכאן, שכל שגיאה המתגלה על ידי האסמבלר נגרמת (בדרך כלל) על ידי שורת קלט מסוימת.

לדוגמה, אם מופיעים שני אופרנדים בהוראה שאמור להיות בה רק אופרנד יחיד, האסמבלר ייתן הודעת שגיאה בנוסח "יותר מדי אופרנדים".

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

לתשומת לב: האסמבלר אינו עוצר את פעולתו אחרי שנמצאה השגיאה הראשונה, אלא ממשיך לעבור על הקלט כדי לגלות שגיאות נוספות, ככל שישנן. כמובן שאין כל טעם לייצר את קבצי הפלט אם נתגלו שגיאות (ממילא 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 מיעון חוקיות עבור אופרנד היעד	שיטות מיעון 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 האסמבלר

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

האסמבלר מתחזק שני מערכים, שייקראו להלן תמונת ההוראות (code) ותמונת הנתונים (data). מערכים אלו נותנים למעשה תמונה של זיכרון המכונה (כל איבר במערך הוא בגודל מילה של המכונה, כלומר 24 סיביות). במערך ההוראות בונה האסמבלר את הקידוד של ההוראות המכונה שנקראו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמבלר את קידוד הנתונים שנקראו מקובץ המקור (שורות הנחיה מסוג 'data' ו-'string').

האסמבלר משתמש בשני מונים, שנקראים IC (מונה ההוראות - Instruction-Counter), ו-DC (מונה הנתונים - Data-Counter). מונים אלו מצביעים על המקום הבא הפנוי במערך ההוראות ובמערך הנתונים, בהתאמה. בכל פעם כשמתחיל האסמבלר לעבור 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 לזיכרון (לצורך ריצה) החל מכתובת 100.

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

במעבר הראשון האסמבלר בונה את טבלת הסמלים ואת השלד של תמונת הזיכרון (הוראות ונתונים).

האסמבלר קורא את קובץ המקור שורה אחר שורה, ופועל בהתאם לסוג השורה (הוראה, הנחיה, או שורה ריקה/הערה).

1. שורה ריקה או שורת הערה: האסמבלר מתעלם מהשורה ועובר לשורה הבאה.

2. שורת הוראה:

האסמבלר מנתח את השורה ומפענח מהי ההוראה, ומחן שיטות המיעון של האופרנדים. מספר האופרנדים נקבע בהתאם להוראה שנמצאה. שיטות המיעון נקבעות בהתאם לתחביר של כל אופרנד, כפי שהוסבר לעיל במפרט שיטות המיעון. למשל, התו # מציין מיעון מידי, תווית מציינת מיעון ישיר, שם של אוגר מציין מיעון אוגר ישיר, וכד'.



אם האסמבלר מוצא בשורת ההוראה גם הגדרה של תווית, אזי התווית מוכנסת אל טבלת הסמלים. ערך התווית הוא IC, והמאפיין הוא code.

כעת האסמבלר קובע לכל אופרנד את ערכו באופן הבא :

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

האסמבלר מכניס למערך ההוראות, בכניסה עליה מצביע מונה ההוראות IC, את קוד המילה הראשונה של ההוראה (בפורמט קידוד כפי שתואר קודם). מילה זו מכילה את קוד הפעולה, את מספרי שיטות המיעון של אופרנד המקור והיעד, ואת מספרי האוגרים של אופרנד המקור והיעד במקרה של שיטת מיעון אוגר ישיר. ה- IC מקודם ב-1.

נזכור שכאשר יש רק אופרנד אחד (כלומר אין אופרנד מקור), הסיביות של שיטת המיעון של אופרנד המקור יכולו 0. בדומה, אם זוהי הוראה ללא אופרנדים (rts, stop), אזי הסיביות של שיטות המיעון של שני האופרנדים יכולו 0. כמו כן, אם שיטת המיעון אינה אוגר ישיר, הסיביות של מספר האוגר הרלוונטי יכולו 0.

אם זוהי הוראה עם אופרנדים (אחד או שניים), האסמבלר "משריין" מקום במערך ההוראות עבור מילות-המידע הנוספות הנדרשות בהוראה זו, ככל שנדרשות, ומקדם את IC בהתאם. כאשר אופרנד הוא בשיטת מיעון מיידי, האסמבלר מקודד גם את המילה הנוספת המתאימה במערך ההוראות. ואילו בשיטת מיעון ישיר או יחסי, מילת המידע הנוספת במערך ההוראות נשארת ללא קידוד בשלב זה.

3. שורת הנחיה :

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

I. 'data'.

האסמבלר קורא את רשימת המספרים, המופיעה לאחר 'data', מכניס כל מספר אל מערך הנתונים, ומקדם את מצביע הנתונים DC ב-1 עבור כל מספר שהוכנס.

אם בשורה 'data' מוגדרת גם תווית, אזי התווית מוכנסת לטבלת הסמלים. ערך התווית הוא ערך מונה הנתונים DC שלפני הכנסת המספרים למערך. המאפיין של התווית הוא data.

II. 'string'.

הטיפול ב-'string' דומה ל-'data', אלא שקודי ה-ascii של התווים הם אלו המוכנסים אל מערך הנתונים (כל תו במילה נפרדת). לבסוף מוכנס למערך הנתונים הערך 0 (המציין סוף מחרוזת). 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'.

זוהי הנחיה לאסמבלר לאפיין את התווית הנתונה כאופרנד כ- entry בטבלת הסמלים. בעת הפקת קבצי הפלט (ראו בהמשך), התווית תירשם בקובץ ה-entries. לתשומת לב: זה לא נחשב כשגיאה אם בקובץ המקור מופיעה יותר מהנחיית entry. אחת עם אותה תווית כאופרנד. המופעים הנוספים אינם מוסיפים דבר, אך גם אינם מפריעים.

#### IV. 'extern'.

זוהי הצהרה על סמל (תווית) המוגדר בקובץ מקור אחר, והקובץ הנוכחי עושה בו שימוש. האסמבלר מכניס את הסמל המופיע כאופרנד לטבלת הסמלים, עם הערך 0 (הערך האמיתי לא ידוע, וייקבע רק בשלב הקישור), ועם המאפיין external. לא ידוע באיזה קובץ נמצאת הגדרת הסמל, ואין זה רלוונטי עבור האסמבלר.

לתשומת לב: זה לא נחשב כשגיאה אם בקובץ המקור מופיעה יותר מהנחיית extern. אחת עם אותה תווית כאופרנד. המופעים הנוספים אינם מוסיפים 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, האסמבלר מעדכן בטבלת הסמלים כל סמל המאופיין כ- data, על ידי הוספת  $IC + 100$  (עשרוני) לערכו של הסמל. הסיבה לכך היא שבתמונה הכוללת של קוד המכונה, תמונת הנתונים מופרדת מתמונת הוראות, וכל הנתונים נדרשים להופיע בקוד המכונה אחרי כל ההוראות. סמל מסוג data הוא תווית בתמונת הנתונים, והעדכון מוסיף לערך הסמל (כלומר לכתובתו בזיכרון) את האורך הכולל של תמונת ההוראות, בתוספת כתובת התחלת הטעינה של הקוד, שהיא 100.

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

במעבר השני, האסמבלר משלים באמצעות טבלת הסמלים את קידוד כל המילים במערך ההוראות שטרם קודדו במעבר הראשון. במודל המכונה שלנו אלו הן מילות-מידע נוספות של פקודות, אשר מקודדות אופרנד בשיטת מיעון ישיר או יחסי. האופרנד מכיל סמל שמוגדר כפנימי או כחיצוני, ולכן בשדה ה- A,R,E הסיבית R או הסיבית E, בהתאמה, תהיה 1 (ראו גם מפרט שיטות המיעון לעיל).

#### אלגוריתם שלדי של האסמבלר

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

לתשומת לב: אין חובה להשתמש דווקא באלגוריתם זה.

כאמור, אנו מחלקים את תמונת קוד המכונה לשני חלקים: תמונת ההוראות (code), ותמונת הנתונים (data). לכל חלק נתחזק מונה נפרד: IC (מונה ההוראות) ו-DC (מונה הנתונים).

#### נבנה את קוד המכונה כך שיתאים לטעינה לזיכרון החל מכתובת 100.

בכל מעבר מתחילים לקרוא את קובץ המקור מההתחלה.

#### מעבר ראשון

1. אתחל  $DC \leftarrow 0$ ,  $IC \leftarrow 100$ .
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-17.
3. האם השדה הראשון בשורה הוא סמל? אם לא, עבור ל-5.
4. הדלק דגל "יש הגדרת סמל".
5. האם זוהי הנחיה לאחסון נתונים, כלומר, האם הנחית data או string? אם לא, עבור ל-8.
6. אם יש הגדרת סמל (תווית), 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, קודד אותם בתמונת הנתונים, והגדל את מונה הנתונים DC על ידי הוספת האורך הכולל של הנתונים שהוגדרו בשורה הנוכחית. חזור ל-2.
8. האם זו הנחית extern או הנחית entry? אם לא, עבור ל-11.
9. אם זוהי הנחית entry. חזור ל-2 (ההנחיה תטופל במעבר השני).
10. אם זו הנחית extern, הכנס את הסמל המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים עם הערך 0, ועם המאפיין external. חזור ל-2.

11. זוהי שורת הוראה. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם המאפיין code. ערכו של הסמל יהיה IC (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
12. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא, אז הודע על שגיאה בשם ההוראה.
13. נתח את מבנה האופרנדים של ההוראה, וחשב מהו מספר המילים הכולל שתופסת ההוראה בקוד המכונה (נקרא למספר זה L).
14. בנה כעת את הקוד הבינארי של המילה הראשונה של ההוראה, ושל כל מילת-מידע נוספת המקודדת באופרנד במיעון מידי.
15. שמור את הערכים IC ו-L יחד עם נתוני קוד המכונה של ההוראה.
16. עדכן  $IC \leftarrow IC + L$ , וחזור ל-2.
17. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר הראשון, עצור כאן.
18. שמור את הערכים הסופיים של IC ושל DC (נקרא להם ICF ו-DCF). נשתמש בהם לבניית קבצי הפלט, אחרי המעבר השני.
19. עדכן בטבלת הסמלים את ערכו של כל סמל המאופיין 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 נגמר קובץ המקור, עבור ל-7.
2. אם השדה הראשון בשורה הוא סמל (תווית), דלג עליו.
3. האם זוהי הנחית data או string? אם כן, חזור ל-1.
4. האם זוהי הנחית entry? אם לא, עבור ל-6.
5. הוסף בטבלת הסמלים את המאפיין entry לסמל המופיע באופרנד של ההנחיה (אם הסמל לא נמצא בטבלת הסמלים, יש להודיע על שגיאה). חזור ל-1.
6. השלם את הקידוד הבינארי של מילות-המידע של האופרנדים, בהתאם לשיטות המיעון שבשימוש. 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 מילת-המידע הרלוונטית לרשימת מילות-מידע שמתייחסות לסמל חיצוני. לפי הצורך, לחישוב הקידוד והכתובות, אפשר להיעזר בערכים IC ו-L של ההוראה, כפי שנשמרו במעבר הראשון. חזור ל-1.
7. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר השני, עצור כאן.
8. בנה את קבצי הפלט (פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו קודם, ונציג את הקוד הבינארי שמתקבל 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

```

נבצע מעבר ראשון על הקוד לעיל, ונבנה את טבלת הסמלים. כמו כן, נשלים במעבר זה את הקידוד של כל תמונת הנתונים, ושל המילה הראשונה של כל הוראה. כמו כן, נקודד מילות-מידע נוספות של כל הוראה, ככל שקידוד זה אינו תלוי בערך של סמל. את מילות-המידע שעדיין לא ניתן לקודד במעבר הראשון נסמן ב "בדוגמה להלן".

Address (decimal)	Source Code	Explanation	Binary Machine Code
0000100 0000101	MAIN: add r3, LIST	First word of instruction Address of label LIST	000010110110100000001100 ?
0000102 0000103	LOOP: prn #48	Immediate value 48	00110100000000000000100 000000000000000110000100
0000104 0000105	lea STR, r6	Address of label STR	000100010001111000000100 ?
0000106	inc r6		000101000001111000011100
0000107 0000108	mov r3, K	Address of label K	000000110110100000000100 ?
0000109	sub r1, r4		000010110011110000010100
0000110 0000111	bne END	Address of label END	001001000000100000010100 ?
0000112 0000113 0000114	cmp K, #-6	Address of label K Immediate value -6	000001010000000000000100 ? 1111111111111111010100
0000115 0000116	bne &END	Distance to label END	001001000001000000010100 ?
0000117 0000118	dec K	Address of label K	000101000000100000100100 ?
0000119 0000120	jmp &LOOP	Distance to label LOOP	001001000001000000001100 ?
0000121	END: stop		001111000000000000000100
0000122	STR: .string "abcd"	Ascii code 'a'	000000000000000001100001
0000123		Ascii code 'b'	000000000000000001100010
0000124		Ascii code 'c'	000000000000000001100011
0000125		Ascii code 'd'	000000000000000001100100
0000126		Ascii code '\0'	000000000000000000000000
0000127 0000128	LIST: .data 6, -9	Integer 6 Integer -9	000000000000000000000110 111111111111111111110111
0000129	.data -100	Integer -100	111111111111111110011100
0000130	K: .data 31	Integer 31	000000000000000000011111

טבלת הסמלים אחרי מעבר ראשון היא :

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

נבצע עתה את המעבר השני. נשלים באמצעות טבלת הסמלים את הקידוד החסר במילים המסומנות "?". הקוד הבינארי בצורתו הסופית כאן זהה לקוד שהוצג בתחילת הנושא "אסמבלר עם שני מעברים".

הערה : כאמור, האסמבלר בונה קוד מכונה כך שיתאים לטעינה לזיכרון החל מכתובת 100 (עשרוני). אם הטעינה בפועל (לצורך הרצת התוכנית) תהיה לכתובת אחרת, יידרשו תיקונים בקוד הבינארי בשלב הטעינה, שיוכנסו בעזרת מידע נוסף שהאסמבלר מכין בקבצי הפלט (ראו בהמשך).

Address (decimal)	Source Code	Explanation	Binary Machine Code
0000100 0000101	MAIN: add r3, LIST	First word of instruction Address of label LIST	000010110110100000001100 000000000000001111111010
0000102 0000103	LOOP: prn #48	Immediate value 48	00110100000000000000100 00000000000000110000100
0000104 0000105	lea STR, r6	Address of label STR	000100010001111000000100 000000000000001111010010
0000106	inc r6		000101000001111000011100
0000107 0000108	mov r3, K	Address of label K	000000110110100000000100 00000000000010000010010
0000109	sub r1, r4		000010110011110000010100
0000110 0000111	bne END	Address of label END	001001000000100000010100 000000000000001111001010
0000112 0000113 0000114	cmp K, #-6	Address of label K Immediate value -6	000001010000000000000100 00000000000010000010010 1111111111111111010100
0000115 0000116	bne &END	Distance to label END	001001000001000000010100 000000000000000000110100
0000117 0000118	dec K	Address of label K	000101000000100000100100 00000000000010000010010
0000119 0000120	jmp &LOOP	Distance to label LOOP	001001000001000000001100 111111111111111101111100
0000121	END: stop		001111000000000000000100
0000122	STR: .string "abcd"	Ascii code 'a'	000000000000000001100001
0000123		Ascii code 'b'	000000000000000001100010
0000124		Ascii code 'c'	000000000000000001100011
0000125		Ascii code 'd'	000000000000000001100100
0000126		Ascii code '\0'	000000000000000000000000
0000127 0000128	LIST: .data 6, -9	Integer 6 Integer -9	000000000000000000000110 111111111111111111110111
0000129	.data -100	Integer -100	11111111111111111110011100
0000130	K: .data 31	Integer 31	000000000000000000011111

בסוף המעבר השני, אם לא נתגלו שגיאות, האסמבלר בונה את קבצי הפלט (ראו בהמשך), שמכילים את הקוד הבינארי ומידע נוסף עבור שלבי הקישור והטעינה. כאמור, שלבי הקישור והטעינה אינם למימוש בפרויקט זה, ולא נדון בהם כאן.

### קבצי קלט ופלט של האסמבלר

בהפעלה של האסמבלר, יש להעביר אליו באמצעות ארגומנטים של שורת הפקודה (command line arguments) רשימה של שמות קבצי מקור (אחד או יותר). אלו הם קבצי טקסט, ובהם תוכניות בתחביר של שפת האסמבלי שהוגדרה בממ"ן זה.

האסמבלר פועל על כל קובץ מקור בנפרד, ויוצר עבורו קבצי פלט כדלקמן:

- קובץ object, המכיל את קוד המכונה.
  - קובץ externals, ובו פרטים על כל המקומות (הכתובות) בקוד המכונה בהם יש מילת-מידע שמקודדת ערך של סמל חיצוני (סמל שהוגדר באמצעות ההנחיה extern, ומאופיין בטבלת הסמלים כ- external).
  - קובץ entries, ובו פרטים על כל סמל שמוצהר כנקודת כניסה (סמל שהופיע כאופרנד של הנחיית entry, ומאופיין בטבלת הסמלים כ- entry).
- אם אין בקובץ המקור אף הנחיית extern, האסמבלר לא יוצר את קובץ הפלט מסוג externals. אם אין בקובץ המקור אף הנחיית entry, האסמבלר לא יוצר את קובץ הפלט מסוג entries.

שמות קבצי המקור חייבים להיות עם הסיומת ".as". למשל, השמות x.as, y.as, ו-hello.as הם שמות חוקיים. העברת שמות הקבצים הללו כארגומנטים לאסמבלר נעשית ללא ציון הסיומת.

לדוגמה: נניח שתוכנית האסמבלר שלנו נקראת assembler, אזי שורת הפקודה הבאה:

```
assembler x y hello
```

תריך את האסמבלר 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" עבור קובץ ה-object, הסיומת ".ent" עבור קובץ ה-entries, והסיומת ".ext" עבור קובץ ה-externals.

לדוגמה, בהפעלת האסמבלר באמצעות שורת הפקודה: assembler x יוצר קובץ פלט x.ob, וכן קבצי פלט x.ext ו-x.ent ככל שיש הנחיות entry או extern. בקובץ המקור. נציג כעת את הפורמטים של קבצי הפלט. דוגמאות יובאו בהמשך.

### פורמט קובץ ה-object

קובץ זה מכיל את תמונת הזיכרון של קוד המכונה, בשני חלקים: תמונת ההוראות ראשונה, ואחריה ובצמוד תמונת הנתונים.

כזכור, האסמבלר מקודד את ההוראות כך שתמונת ההוראות תתאים לטעינה החל מכתובת 100 (עשרוני) בזיכרון. נשים לב שרק בסוף המעבר הראשון יודעים מהו הגודל הכולל של תמונת ההוראות. מכיוון שתמונת הנתונים נמצאת אחרי תמונת ההוראות, גודל תמונת ההוראות משפיע על הכתובות בתמונת הנתונים. זו הסיבה שבגללה היה צורך לעדכן בטבלת הסמלים, בסוף המעבר הראשון, את ערכי הסמלים המאופיינים כ-data (כזכור, בצעד 19 הוספנו לכל סמל כזה את הערך ICF). במעבר השני, בהשלמת הקידוד של מילות-המידע, משתמשים בערכים המעודכנים של הסמלים, המותאמים למבנה המלא והסופי של תמונת הזיכרון.

כעת האסמבלר יכול לכתוב את תמונת הזיכרון בשלמותה לתוך קובץ פלט (קובץ ה-object).

השורה הראשונה בקובץ ה-object היא "כותרת", המכילה שני מספרים (בבסיס עשרוני): הראשון הוא האורך הכולל של תמונת ההוראות (במילות זיכרון), והשני הוא האורך הכולל של תמונת הנתונים (במילות זיכרון). בין שני המספרים מפריד רווח אחד. כזכור, במעבר הראשון, בצעד 18, נשמרו הערכים ICF ו-IDF. האורך הכולל של תמונת ההוראות הוא ICF-100, והאורך הכולל של תמונת הנתונים הוא IDF.

השורות הבאות בקובץ מכילות את תמונת הזיכרון. בכל שורה שני שדות: כתובת של מילה בזיכרון, ותוכן המילה. הכתובת תירשם בבסיס עשרוני בשבע ספרות (כולל אפסים מובילים). תוכן המילה יירשם בבסיס הקסאדצימלי ב-6 ספרות (כולל אפסים מובילים). בין שני השדות בשורה יש רווח אחד.

### פורמט קובץ ה-entries

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

### פורמט קובץ ה-externals

קובץ ה-externals בנוי אף הוא משורות טקסט, שורה לכל כתובת בקוד המכונה בה יש מילת מידע המתייחסת לסמל שמאופיין כ-external. כזכור, רשימה של מילות-מידע אלה נבנתה במעבר השני (צעד 6).

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

לתשומת לב: ייתכן ויש מספר כתובות בקוד המכונה בהן מילות-המידע מתייחסות לאותו סמל חיצוני. לכל כתובת כזו תהיה שורה נפרדת בקובץ ה-externals.

נדגים את הפלט שמייצר האסמבלר 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 הקידוד הבינארי המלא (תמונת הזיכרון) של קובץ המקור, כפי שנבנה במעבר הראשון והשני.

Address (decimal)	Source Code	Explanation	Binary Machine Code
0000100 0000101	MAIN: add r3, LIST	First word of instruction Address of label LIST	000010110110100000001100 000000000000010000010010
0000102 0000103	LOOP: prn #48	Immediate value 48	00110100000000000000100 000000000000000110000100
0000104 0000105	lea W, r6	Address of extern label W	000100010001111000000100 000000000000000000000001
0000106	inc r6		000101000001111000011100
0000107 0000108	mov r3, K	Address of label K	000000110110100000000100 000000000000010000101010
0000109	sub r1, r4		000010110011110000010100
0000110 0000111	bne END	Address of label END	001001000000100000010100 0000000000000001111100010
0000112 0000113 0000114	cmp K, #-6	Address of label K Immediate value -6	000001010000000000000100 000000000000010000101010 1111111111111111010100
0000115 0000116	bne &END	Distance to label END	001001000001000000010100 000000000000000001001100
0000117 0000118	dec W	Address of extern label W	000101000000100000100100 000000000000000000000001
0000119 0000120	jmp &LOOP	Distance to label LOOP	001001000001000000001100 111111111111111101111100
0000121 0000122 0000123	add L3, L3	Address of extern label L3 Address of extern label L3	0000100100001000000001100 0000000000000000000000001 0000000000000000000000001
0000124	END: stop		00111100000000000000000100
0000125	STR: .string "abcd"	Ascii code 'a'	000000000000000001100001
0000126		Ascii code 'b'	000000000000000001100010
0000127		Ascii code 'c'	000000000000000001100011
0000128		Ascii code 'd'	000000000000000001100100
0000129		Ascii code '\0'	0000000000000000000000000
0000130 0000131	LIST: .data 6, -9	Integer 6 Integer -9	0000000000000000000000110 111111111111111111110111
0000132	.data -100	Integer -100	11111111111111110011100
0000133	K: .data 31	Integer 31	000000000000000000011111

טבלת הסמלים הסופית בגמר המעבר השני היא :

סמל	ערך (בבסיס 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.ob :

```
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 fffff7
0000132 ffff9c
0000133 00001f
```

הקובץ ps.ent :

```
MAIN 0000100
LIST 0000130
```

הקובץ ps.ext :

```
W 0000105
W 0000118
L3 0000122
L3 0000123
```

## סיכום והנחיות כלליות

- גודל תוכנית המקור הניתנת כקלט לאסמבלר אינו ידוע מראש, ולכן גם גודלו של קוד המכונה אינו צפוי מראש. אולם בכדי להקל במימוש האסמבלר, מותר להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים לאחסון תמונת קוד המכונה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים), יש לממש באופן יעיל וחסכוני (למשל באמצעות רשימה מקושרת והקצאת זיכרון דינאמי).
- השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הקלט, למעט הסיומות. למשל, אם קובץ הקלט הוא prog.as אזי קבצי הפלט שיווצרו הם : prog.ob, prog.ext, prog.ent
- מתכונת הפעלת האסמבלר צריכה להיות כפי הנדרש בממ"ן, ללא שינויים כלשהם. כלומר, ממשיק המשתמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו לתוכנית האסמבלר כארגומנטים (אחד או יותר) בשורת הפקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גרפיים למיניהם, וכד'.
- יש להקפיד לחלק את מימוש האסמבלר למספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסוגים שונים במודול יחיד. מומלץ לחלק למודולים כגון : מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחבירי של שורה), טבלת הסמלים, מפת הזיכרון, טבלאות קבועות (קודי הפעולה, שיטות המיעון החוקיות לכל פעולה, וכד').
- יש להקפיד ולתעד את המימוש באופן מלא וברור, באמצעות הערות מפורטות בקוד.
- יש לאפשר תווים לבנים עודפים בקובץ הקלט בשפת אסמבלי. למשל, אם בשורת הוראה יש שני אופרנדים המופרדים בפסיק, אזי לפני ואחרי הפסיק מותר שיהיו רווחים וטאבים בכל כמות. בדומה, גם לפני ואחרי שם הפעולה. מותרות גם שורות ריקות. האסמבלר יתעלם מתווים לבנים מיותרים (כלומר ידלג עליהם).
- הקלט (קוד האסמבלי) עלול להכיל שגיאות תחביריות. על האסמבלר לגלות ולדווח על כל השורות השגויות בקלט. אין לעצור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להדפיס למסך הודעות מפורטות ככל הניתן, כדי שאפשר יהיה להבין מה והיכן כל שגיאה. כמובן שאם קובץ קלט מכיל שגיאות, אין טעם להפיק עבורו את קבצי הפלט (ob, ext, ent).

תם ונשלם פרק ההסברים והגדרת הפרויקט.

### **בשאלות ניתן לפנות לקבוצת הדיון באתר הקורס, ואל כל אחד מהמנחים בשעות הקבלה שלהם.**

להזכירכם, באפשרותו של כל סטודנט לפנות לכל מנחה, לאו דווקא למנחה הקבוצה שלו, לקבלת עזרה. שוב מומלץ לכל אלה שטרם בדקו את התכנים באתר הקורס לעשות זאת. נשאלות באתר זה הרבה שאלות בנושא חומר הלימוד והממ"נים, והתשובות יכולות להועיל לכולם.

לתשומת לבכם : לא תינתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או מחלה ממושכת. במקרים אלו יש לבקש ולקבל אישור מראש מצוות הקורס.

**ב ה צ ל ח ה !**