

# מטלת מנהה (ממ"ו) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטרת : פרויקט גמר

משקל המטלת : 61 נקודות (חובה)

מספר שאלות : 1

מועד אחרון להגשה : 20.03.2022

סמסטר : א' 2022

## קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
- שליחת מטלות באמצעות דואר אלקטורי - באישור המנהה בלבד

### הסבר מפורט ב" ניהול הגשת מטלות מנהה"

אחת 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 ל כתוב תוכנת אסמבילר, עבור שפת אסמבלי שתוגדר בהמשך. הפרויקט ייכתב בשפת C.

עליכם להגיש את הפריטים הבאים :

1. קבצי המקור של התוכנית שכתבתם (קבצים בעלי סימות c או h).
2. קובץ הרצה (מוקומפל ומקשור) עבור מערכת אוביונטו.
3. קובץ makefile. הקימפול חייב להיות עם הקומפיילר, כך שהתוכנית תתкомפלט ללא כל הערות או אזהרות.
4. דוגמאות הרצה (קלט ופלט) :
  - א. קובצי קלט בשפת אסמבילר, קובצי הפלט שנוצרו מהפעלת האסמבילר על קבצי קלט אלה. יש להציגים שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האסמבילר.
  - ב. קובצי קלט בשפת אסמבילרי המציגים מגוון רחב של סוגי שגיאות אסמבילר (ולכן נוצרים קבצי פלט), ותדפסי המסקן המראים את ההודעות השגיאה שמוציא האסמבילר.

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

נזכיר מספר היבטים חשובים של כתיבת קוד טוב :

1. הפשטה של בני הנתונים : רצוי (כל האפשר) להפריד בין הגישה למבנה הנתונים לבני הIMPLEMENTATION של בני הנתונים. כך, למשל, בעת כתיבת פונקציות לטיפול בטבלה, אין זה מעניינים של המשמשים בפונקציות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקושרת.
2. קריאות הקוד : יש להשתמש בשמות משמעותיים למשתנים ופונקציות. יש לעורך את הקוד באופן מסודר: הזוחות עקביות, שורות ריקות להפרדה בין קטיעי קוד, וכו'.
3. תייעוד : 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 שمبرכעת את כל הדורש ממנה, אינה לכשעצמה ערובה לצוין גובה. כדי לקובל ציוון גובה, על התוכנית לעמוד בקריטריונים של כתיבה ותיעוד ברמה טוביה, כמתואר לעיל, אשר משקלם המשותף מגע עד לכ- 40% משקל הפרויקט.

モותר 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 ציוון.** חוברים להגיש יחד את הפרויקט, יהיו **שייכים** לאלה **קבוצת הנחיה.** הציוון יהיה לשני הסטודנטים.

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

### פרק כללי ומטרת הפרויקט

כידוע, קיימות שפות תוכנות רבות, ומספר גדול של תוכניות, הכתובות בשפות שונות, עשויות לזרז באוטומאטיות. כיצד "מכיר" המחשב כל צ'ק הרצה שפות? התשובה פשוטה: המחשב מכיר למעשה שפה אחת בלבד: הראות ונתוני הכתובים בקוד בינארי. קוד זה מאוחסן בOSH זיכרונו, ונראה כמו רצף של ספירות ביןarieות. יחידת העיבוד המרכזית - היע"מ (CPU) - יודעת לפרק את הרץ הזה לקטעים קטנים בעלי משמעות: הראות, מענינים ונתוניים.

למעשה, זיכרונו המחשב כולל הוא אוסף של סיביות, שנוהגים לראותן כמקובצות לייחדות בעלות אורך קבוע (בתים, מילימים). לא ניתן להבחין, בעין שאיתנה מיומנת, בהבדל פיסי כלשהו בין אותן חלק בזיכרון שבו נמצאת תוכנית לבין שאר הזיכרון.

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מגוון פעולות פשוטות, הנקראות **הוראות המכונה**, ולשם כך היא משתמשת באוגרים (registers) הקיימים בתוך היע"מ, ובזיכרון המחשב. **דוגמאות:** העברת מספר מתא בזיכרון לאוגר מסוים או בחרזה, הוספה 1 למספר הנמצא באוגר, בדיקה האם מסpter המאוחסן באוגר שווה לאפס, חיבור וחיסור בין שני אוגרים, וכו'. הוראות המכונה ושילובים שלחן הן המרכיבות תוכנית כפי שהיא טעונה לזכרון בזמן ריצתה. כל תוכנית מקור (תוכנית כתובה בידי המתכנת), תתרגם בסופו של דבר באמצעות תוכינה מיוחדת לזרחה סופית זו.

היע"מ יודע לבצע קוד שנמצא בפורמט של **שפת מכונה**. זהו רצף של ביטים, המהווים קידוד ביניاري של סדרת הוראות המכונה המרכיבות את התוכנית. קוד זה אינו קרייא למשתמש, ולכן לא נוח לקודד (או לקרוא) תוכניות ישירות בשפת מכונה. **שפת אסמבלי (assembly language)** היא שפת תוכנות מאפשרת לייצג את הוראות המכונה בזרחה סימבולית קלה ונוחה יותר לשימוש. כמו כן יש צורך לתרגם את הייצוג הסימבולי לקוד בשפת מכונה, כדי שהתוכנית תוכל לרוץ במחשב. תרגום זה נעשה באמצעות כל שינקרה **אסambilר (assembler)**.

כידוע, לכל שפת תוכנות עליית יש מהדר (compiler), או מפרש (interpreter), המתרגם תוכניות מקור לשפת מכונה. האסambilר משמש בתפקיד דומה עבור שפת אסambilי.

כל מודול של יע"מ (כלומר לכל אירוגון של מחשב) יש שפת מכונה יעודית משלו, ובהתאם גם שפת אסambilי יעודית משלו. לפיכך, גם האסambilר (כל התרגומים) הוא יudoוי ושונה לכל יע"מ.

תפקידו של האסambilר הוא לבנות קוד המכיל קוד מכונה, מקובץ נתון של תוכנית הכתובת בשפת אסambilי. זהו השלב הראשון במסלול אותו עוברת התוכנית, עד לקבלת קוד המוקן לריצה על חומרת המחשב. השלבים הבאים הם קישור (linkage) וטעינה (loading), אך בהם לא עוסוק במאין זה.

המשימה בפרויקט זה היא לכתוב אסambilר (כלומר תוכנית המתרגם לשפת מכונה), עבור שפת אסambilי שנגידיר כאן במיוחד לפרויקט.

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

## המחשב הדמיוני ושפת האסטמבי

נדיר עתה את שפת האסטמבי ואת מודל המחשב הדמיוני, עבור פרויקט זה.

הערה : תואר מודל המחשב להלן הוא חלקו בלבד, ככל שנחוץ לביצוע המשימות בפרויקט.

"חומרה":

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

למעבד 16 אוגרים כלליים, בשמות: r0, r1, r2, r3, r4, r5, r6, r7.....r15. הסיבית הכי פחותה משמעותית נמצוא בסייבית מס' 0, והסיבית המשמעותית ביותר במס' 19. שמות האוגרים נכתבם 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 המכונה, הסברים לגבי השימוש בדגלים אלו.

גודל הזיכרון הוא 8192 תאים, בכתבות 0-1918, וכל תא הוא בגודל של 20 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 בשיטות המשלים ל-2 (2's complement). כמו כן יש תמייה בתווים (characters), המוצגים בקוד ASCII.

מבנה הוראת המכונה:

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

כל הוראה מכונה מקודדת במספר מילוט זיכרון רצופות, החל ממילה אחת ועד למקסימום של מיללים, בהתאם לסוג הפעולה (ראו פרטים בהמשך).

בקובץ הפלט המכיל את קוד המכונה שבונה האסטמבלר, כל מילה תקודד "בסיס מיוחד" (ראו פרטים לגבי קבצי פלט בהמשך).

בהוראה מכונה ללא אופרנדים, המבנה של המילה הראשונה (והיחידה) הוא :

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	A	R	E																opcode

ואילו בהוראות עם אופרנדים המבנה של הקידוד יכול לפחות 2 מילוט זיכרון במבנה הבא :

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	A	R	E																opcode
0	A	R	E																מיון אונר יעד
																			מיון אונר מקור funct

מילים נוספות בהתאם לשיטות המיון יעד

במודל המcona שלנו יש 16 פעולות, בפועל, למרות שניינו לקודד יותר פעולות. כל פעולה מיוצגת בשפת אסםביי באפן סימבולי על ידי **שס-פעולה**, ובקוד המcona על ידי קומבינציה ייחודית של ערכי שני שדות במילה הראשונה של הוראה: **קוד-הפעולה (opcode)**, **ופונקציה (funct)**.

להלן טבלת הפעולות:

שם הפעולה	פונקציה (בסיס עשרוני)	קוד הפעולה (בסיס עשרוני)
mov		0
cmp		1
add	10	2
sub	11	2
lea		4
clr	10	5
not	11	5
inc	12	5
dec	13	5
jmp	10	9
bne	11	9
jsr	12	9
red		12
prn		13
rts		14
stop		15

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

להלן מפרט הפעולות בקידוד הראשונה בקוד המcona של כל הוראה.

**שדה opcode:** שדה זה מיוצג ב- 16 סיביות, והוא מכיל בית דולק בודד בהתאם לקוד הפעולה. למשל אם מדובר על קוד פעולה 9, אזי סיבית 9 תקבל ערך של 1.

**סיבית 19:** לא בשימוש, ערכיה קבוע לאפס.

**סיביות 18-16:** עבר קידוד הוראות, סיביות אלה מכילות את סיוג הקידוד (A=Absolute, R=Relocatable, E=External) לכל מילת קידוד של הוראה יש סיוג, בהתאם לסיוג הסיבית המתאימה בשדה ARE תקבל ערך 1.

**סיביות 15-12:** שדה זה, הנקרא funct, מתפרק כאשר מדובר בפעולה שקוד-הפעולה (opcode) שלה משותף לכמה פעולות שונות (כאמור, קוד-פעולה 2, 5 או 9). השדה funct יכול ערך ייחודי לכל פעולה מקובצת הפעולות שיש להן אותו קוד-פעולה. אם קוד-הפעולה משמש לפעולה אחת בלבד, הסיביות של השדה funct יהיו מאופסות.

**סיביות 11-8:** מכילות את מספרו של אוגר המקור במידה ואופרנד המקור הוא אוגר. אם אין בהוראה אופרנד מקור שהוא אוגר, סיביות אלה יהיו מאופסות.

**סיביות 7-6:** מכילות את מספרה של שיטת המיעון של אופרנד המקור. אם אין בהוראה אופרנד מקור, סיביות אלה יהיו מאופסות. מפרט של שיטות המיעון השונות יינטו בהמשך.

**סיביות 5-2:** מכילות את מספרו של אוגר היעד במידה ואופרנד היעד הוא אוגר. אם אין בהוראה אופרנד יעד שהוא אוגר, סיביות אלה יהיו מאופסות.

**סיביות 0-1:** מכילות את מספרה של שיטת המיעון של אופרנד היעד. אם אין בהוראה אופרנד יעד, סיביות אלה יהיו מאופסות.

### שיטות מייען:

שיטות מייען (addressing modes) הן האופנים השונים בהם ניתן להעביר אופרנדים של הוראות מכונה. בשפת האסמבלי שלנו קיימות ארבע שיטות מייען, המסווגות במספרים 0,1,2,3.

השימוש בשיטות המייען מצריך מילוט-מידע נוספת בקוד המכונה של הוראה, בנוסף למלילים הראשונות. קידוד אופרנד של הוראה עשוי לייצר מילוט זיכרון נוספת בהתאם לסוג שיטת המייען. כאשר בהוראה יש שני אופרנדים, קודם יופיעו מילוט-המידע הנוספות של אופרנד המקור, ולאחריהם מילוט-המידע הנוספות של אופרנד השני.

להלן המפרט של שיטות המייען.

מספר	שיטה המייען	תוכן מילוט-המידע הנוספות	תחביר האופרנד באסמבלי	דוגמאות
0	מייען מיידי (immediate)	מילוט-מידע נוספת של הוראה מכילה את האופרנד עצמו, שהוא מספר שלם בסיס המשלים ל-2, ברוחב של 16 סיביות מילה זו תסוג מסוג A	האופרנד מתחילה בתו '# ולאחריו ובצמוד אליו מופיע מספר שלם בסיס עשרוני.	mov #1, r2 בדוגמה זו האופרנד הראשון של הוראה הראשון (אופרנד המקור) נתנו בשיטת מייען מיידי. הוראה כתובת את הערך 1- אל אוגר 2.
1	מייען ישיר (direct)	2 מילוט-מידע נוספת של הוראה יכilo את כתובת האופרנד מבנה של כתובת בסיס והיסט. <b> כתובת בסיס =</b> הכתובות הקרובות ביותר לכנתובת האופרנד, הקטנה ממנו, ומתחלקת ב-16. למשל, אם כתובת האופרנד היא 36, אז כתובת הבסיס היא 32, מאחר ו- 32 הוא המספר הקרוב ביותר ל- 36 שקטן ממנו ומתחלך ב-16. <b> היסט =</b> המרחק מכתובת הבסיס לכנתובת האופרנד. בדוגמה שניתנה בהסביר לכתובת בסיס, ההיסט יהיה 4. מאחר והמרחק (ההיסט) מ- 32 ל- 36 הוא 4.	האופרנד הוא <u>תוויות</u> שכבר הוגדרה, או שתווגר בהמשך הקובץ. הוגדרה נועשת על ידי כתיבת התווית בתחילת השורה של הנתיב 'data' או '.string', או בתחילת השורה של הוראה, או באמצעות אופרנד של הנתיב 'extern'. התווית מייצגת באופן סימבולי כתובות בזיכרון.	הוראה הבאה מדירה את התווית x : x: .data 23 : הוראה : dec x מקטינה ב-1 את תוכן המילה שכותבת x בזיכרון ("משתנה" x). הכתובת של x מקודדת הכתובת הבא שתתבצע, במילוט-המידע הנוספות, מבנה של כתובת בסיס והיסט. <u>דוגמה נוספת :</u> הוראה jmp next מבצעת קופיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתתבצע נמצאת בכתובת next). הכתובת next מקודדת הכתובת הבא שתתבצע במילוט-המידע הנוספות מבנה של כתובת בסיס והיסט.
.	.	מילוט-המידע הנוספת הראונה תכיל את כתובת הבסיס. ומילוט המידע השני תכיל את היסט. כתובת הבסיס והיסט מיוצגים כמספר <u>לא סימני</u> ברוחב של 16 סיביות. והם יסווגו מסוג R במידה והאופרנד הוא תווית חיונית, כתובת הבסיס והיסט יכilo אפסים, וקידודים אלה יסווגו מסוג E במקרה כזה.	.	.

מספר	שיטת המיעון	תוכן מילוט-המידע הנוספות	תחביר האופרנד באסמלבי	דוגמה
2	מיוען אינדקס	<p>בשיטת זו, יש בקידוד ההוראה 2 מילוט מידע נוספת, המכילה את כתובות התווית בצורת כתובות בסיס והיסט כפי שתואר בשיטת מיוען מס' 1.</p> <p>מספרו של האוגר שצוין בסוגרים המרובעים, ישמר כפי שמופיע בשיטת מיוען מס' 3, בסיביות אוגר המקור/יעד בהתאם לכך אם האופרנד הוא אופרנד מקור/יעד.</p>	<p>האופרנד מתחילה בשם של תווית ולאחריה בסוגרים מרובעים, שם של אוגר שמספרו בין 10 ל- 15 בלבד.</p> <p><b>הפקודה:</b>  <code>mov x[r12], r4</code></p> <p>בדוגמה זו האופרנד הראשון הוא גישה כתובת של X בזיכרון, והחל משם מתקדמים כמוות של מילוט זיכרון נספנות לפי הערך שהוא בזיכרונו <code>r12</code>, ומה שיש באותה כתובות ישמש את המעבד כאופרנד המקור. בדוגמה זו אופרנד זו ישמר באוגר <code>r4</code>.</p>	<p>השורה הבאה מגדרה את התווית X :  <code>x: .data 23,12,34,50</code></p>
3	מיוען אוגר ישיר (register direct)	<p>אין מילוט-מידע נוספת בגין האוגר.</p> <p>מספרו של האוגר ישמר בסיביות של אוגר המקור/יעד בהתאם לכך אם האופרנד הוא אופרנד מקור/יעד.</p>	<p>האופרנד הוא שם של אוגר.</p> <p><b>דוגמה נוספת:</b>  <code>clr r1</code>  <code>clr r1</code> מאפסת את האוגר <code>r1</code>.  <code>clr</code>  <code>mov #1, r2</code></p> <p>האופרנד השני של ההוראה (אופרנד היעד) נתון בשיטת מיוען אוגר ישיר. ההוראה כתובת את הערך המיידי 1- אל אוגר <code>r2</code>.</p>	<p><b>דוגמה נוספת:</b>  <code>clr r1</code>  <code>clr r1</code> מאפסת את האוגר <code>r1</code>.  <code>clr</code>  <code>mov #1, r2</code></p> <p>האופרנד השני של ההוראה (אופרנד היעד) נתון בשיטת מיוען אוגר ישיר. ההוראה כתובת את הערך המיידי 1- אל אוגר <code>r2</code>.</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 הוראה הנוכחית שמתבצעת (הכוונה תמיד לכתובת המילה הראשונה של הוראה).

הוראות המכונה מחלקות לשולש קבוצות, לפי מספר האופרנדים הנדרשים לפועלם.

#### **קבוצת הוראות הראשונה:**

אלן הן הוראות המקבלות שני אופרנדים.

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

הסבר הזוגמה	דוגמה	הפעולה המתבצעת	funct	opcode	הוראה
העתק את תוכן המשתנה A (המילה שבכ坨בת זיכרון) אל אוגר r1.	mov A, r1	מבצעת העתקה של תוכן אופרנד המקור (האופרנד הראשון) אל אופרנד היעד (האופרנד השני).		0	mov
אם תוכן המשתנה A זהה לתוכנו של אוגר r1 אז הדגל Z ("דגל האפס") באוגר הסטטוס (PSW) יודלק, אחרת הדגל יאפס.	cmp A, r1	מבצעת השוואה בין שני האופרנדים. ערך אופרנד היעד (השני) מופחת מערך אופרנד המקור (הראשון), ללא שימוש החיסור מעכנת הדגל בשם Z ("דגל האפס") באוגר הסטטוס (PSW).		1	cmp
אוגר r0 מקבל את תוצאה החיבור של תוכן המשתנה A ותוכנו הנוכחי של r0.	add A, r0	אופרנד היעד (השני) מקבל את תוצאה החיבור של אופרנד המקור (הראשון) והיעד (השני).	10	2	add
אוגר r1 מקבל את תוצאה החיסור של הקבוע 3 מתוכנו הנוכחי של האוגר r1.	sub #3, r1	אופרנד היעד (השני) מקבל את תוצאה החיסור של אופרנד המקור (הראשון) מואופרנד היעד (השני).	11	2	sub
המען שמייצגת התווית HELLO מוצב לאוגר r1.	lea HELLO, r1	lea הוא קיצור (ראשי תיבות) של load effective address מציבה את מען הזיכרון המיוצג על ידי התווית שבאופרנד הראשון (המקור), אל האופרנד השני (היעד).		4	lea

#### **קבוצת הוראות השנייה:**

אלן הן הוראות המקבלות אחד בלבד. אופן הקידוד של האופרנד הוא כמו של אופרנד היעד בהוראה עם שני אופרנדים. השדה של אופרנד המקור (סיביות 2-3) בmäßigת הראשונה בקידוד ההוראה אינו בשימוש, ולפיכך יהיה מאופס.

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

הסבר הזוגמה	דוגמה	הפעולה המתבצעת	funct	opcode	הוראה
האוגר r2 מקבל את הערך 0.	clr r2	איפוס תוכן האופרנד.	10	5	clr
כל בית באוגר r2 מתחפה.	not r2	היפוך הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0).	11	5	not
תוכן האוגר r2 מוגדל ב-1.	inc r2	הגדלת תוכן האופרנד באחד.	12	5	inc
תוכן המשתנה Count מוקטן ב-1.	dec Count	הקטנת תוכן האופרנד באחד.	13	5	dec
PC $\leftarrow$ PC+addressOf(Line) הכתובת של תווית Line נשמרת לתוכן מצביע התכניתית ולפיכך ההוראה הבאה שתבצע תהיה בمعنى Line.	jmp Line	קפיצה (הסטעפות) בלתי מותנית אל ההוראה שנמצאת בمعنى המיוצג על ידי האופרנד. כלומר, כתוצאה מביצוע ההוראה, מצביע הרווכנית (PC) מקבל את כתובת יעד הקפיצה.	10	9	jmp

הווראה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
bne	9	11	בנ"ה הוא קיצור (אשי תיבות) של: bne (branch if not equal (to zero)) הוראת הסתעפות מותנית. אם ערכו של הדגל Z באוגר הסטטוס (PC) הינו 0, אז מצביע התוכנית יקבל את כתובת התווית Line, ולפיכך תחיה בהאה שתתבצע Line. הוראת cmp נקבע באמצעות .	bne Line	אם ערך הדגל Z באוגר הסטטוס (PSW) הוא 0, אז PC $\leftarrow$ address(Line) מצביע התוכנית יקבל את כתובת התווית Line, ולפיכך הוראה הבאה שתתבצע תחיה במען Line.
jsr	9	12	קריאה לשגרה (סברוטינה). כתובת ההוראה שאחרי הוראת jsr הונכתה (PC+2) נדחפת לתוך המחסנית שביצרו המחשב, ומצביע התוכנית (PC) מקבל את כתובת השגרה. הערה: חוזרת המשגרה מתבצעת באמצעות הוראת tsz, תוך שימוש בכתובת שבמחסנית.	jsr SUBR	push(PC+2) PC $\leftarrow$ address(SUBR) מצביע התוכנית יקבל את כתובת התווית SUBR, ולבסוף, הוראה הבאה ולפיכך, תחיה בהאה שתתבצע Line. SUBR. כתובת החזרה מהשגרה נשמרת במחסנית.
red	12		קריאה שלתו 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	יודפס לפט התו הנמצא באוגר r1

**קבוצת ההוראות השלישית:**  
אלן הוראות ללא אופרנדים. קידוד ההוראה מורכב ממילה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד (סיביות 3-0) במילה הראשונה של ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השויות לקבוצה זו הן : rts, stop

הווראה	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 $\leftarrow$ pop() ההוראה הבאה שתתבצע תהיה זו שאחרי הוראת jsr שקרה לשגרה. התוכנית עוצרת מיידית.
stop	15	עצירת ריצת התוכנית.	stop	

**מבנה תכנית בשפת אסמבלי :**

תכנית בשפת אסמבלי בנויה **ממרקוריים וממשפטים** (statements).

#### **מרקוריים :**

מרקוריים הם קטעי קוד הכלולים בתוכם משפטים. בתוכנית ניתן להגדיר מקרו ולהשתמש בו במקומות שונים בתוכנית. השימוש במרקרו ממוקם מסוים בתוכנית יגרום לפרישת המקרו לאותו מקום.

הגדרת מקרו נעשית באופן הבא : (בדוגמה שם המקרו הוא m1)

```
macro m1
  inc r2
  mov A,r1
endm
```

שימוש במקרו הוא פשוט אזכור שמו.  
למשל, אם בתוכנית במקום כלשהו כתוב:

m1

m1

בדוגמה זו, השתמשנו פעמיים במקרו `m1`, התוכנית לאחר פרישת המקרו תיראה כך:

```
inc r2  
mov A,r1
```

```
inc r2  
mov A,r1
```

#### התוכנית לאחר פרישת המקרו היא התוכנית שהאSEMBLER אמרו לתרגם.

הנחיות לגבי מקרו:

- אין במערכת הגדרות מקרו מוקנות.
- שם של חוראה או הנחיה לא יכול להיות שם של מקרו.
- ניתן להניח שלכל שורת מקרו בקוד המקור קיימת סגירה עם שורת `endm` (אין צורך לבדוק זאת).
- הגדרת מקרו תהיה תמיד לפני הקראיה למקרו נדרש שהקדם-אSEMBLER ייצור קובץ עם הקוד המורחב הכולל פרישה של המקרו של קובץ המקור המקורי בהמשך). "קובץ המקור המורחב" הוא "קובץ מקור" לאחר פרישת המקרו, לעומת "קובץ מקור ראשוני" שהוא קובץ הקלט למערכת, כולל הגדרת המקראים.

#### משפטים:

קובץ מקור בשפת אSEMBLER מורכב משורות המכילות משפטי השפה, כאשר כל המשפט מופיע בשורה נפרדת. כלומר, ההפרדה בין משפט למשפט בקובץ המקור הינה באמצעות התוו 'ז' (שורה חדשה).

אורכה של שורה בקובץ המקור הוא 80 תוים לכל היתר (לא כולל התוו זו).

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

סוג המשפט	הסביר כללי
משפט ריק	זוהי שורה המכילה אך ורק תווים לבנים (whitespace), ככלומר רק את התווים ' ' ו- 't' (רווחים וטאבים). ייתכן ובשורה אין אף תו (למעט התו חא), ככלומר השורה ריקה.
משפט הערה	זוהי שורה בה התו הראשון הינו ';' (נקודה פסיק). על האסמלר להתעלם לוחטין משורה זו.
משפט הנטיה	זהו משפט המניח את האסמלר מה עליו לעשות כשהוא פועל על תוכנית המקור. יש מספר סוגים של משפטי הנטיה. המשפט הנטיה עשוי לגורם להקצת זיכרון ואתחול משתנים של התוכנית, אך הוא אינו מייצר קידוד של הוראות מכונה לביצוע בעת ריצת התוכנית.
משפט הוראה	זהו משפט המיציר קידוד של הוראות מכונה לביצוע בעת ריצת התוכנית. המשפט מורכב ממשם ההוראה (פעולה) שעל המעבד לבצע, והאפרנדים של ההוראה.

כעת נפרט יותר לגבי סוגי המשפטים השונים.

#### משפט הנטיה:

משפט הנטיה הוא בעל המבנה הבא :

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

לאחר מכן מופיע שם הנטיה. לאחר שם הנטיה יופיעו פרמטרים (מספר הפרמטרים בהתאם להנטיה). שם של הנטיה מתחיל בטו ' ' (נקודה) ולאחריו תווים באותיות קטנות (lower case) בלבד.

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

1. הנטיה 'data'.

פרמטרים של הנטיה '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 בכלל 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'. מנחאת האסמלר להקצות מקומות בתמונה הנתונים (data image), אשר בו יארחסו הערכים של הפרמטרים, ולקדם את מונה הנתונים, בהתאם למספר הערכים. אם בתוכנית 'data' מוגדרת תווית, אז תווית זו מקבלת את ערך מון הנתונים (לפני 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 מילימ"ר רצופות שיכילו את המספרים שמופיעים בהנטיה. התווית XYZ מזווהה עם כתובות המילה הראשונה.

אם נכתב בתוכנית את ההוראה :  
mov XYZ, r1

אז בזמן ריצת התוכנית יוכל לאוגר 12 הערך 7.

ואילו הוראה :

```
lea XYZ, r1
```

תכנס לאוגר 12 את ערך התווית XYZ (כלומר הכתובתizi זיהוי בזיכרון מאוחסן הערך 7).

2. הנקה 'string'.

הנקה 'string', פרמטר אחד, שהוא מחרוזת חוקית. תווים המחרוזות מקודדים לפי ערכי ASCII המתאימים, ומוכנסים אל תומנת הנתונים לפי סדרם, כל TWO בamilha נפרדת. בסוף המחרוזת יתווסף התו '0' (הערך המספרי 0), המסמך את סוף המחרוזת. מונה הנתונים של האסמבילר יקודם בהתאם לאורך המחרוזת (בתוספת מקום אחד עבור התו המסיים). אם בשורת ההנחיה מוגדרת תווית, אז תווית זו מקבלת את ערך מונה הנתונים (לפניהם) ומוכנסת אל טבלת הסמלים, דומה כמי שנעשה עבור 'data'. (כלומר ערך התווית יהיה הכתובתizi זיהוי בזיכרון שבתחלת המחרוזת).

לדוגמה, הנקה :

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

מקצת בתומנת הנתונים רצף של 7 מיללים, ומתחילה את המילים לקובץ ASCII של התווים לפי הסדר במחרוזת, ולאחריהם ערך 0 לסימון סוף המחרוזת. התווית STR מזוהה עם כתובות התחלת המחרוזת.

3. הנקה 'entry'.

הנקה 'entry', פרמטר אחד, והוא שם של תווית המוגדרת בקובץ המקור הנוכחי (כלומר תווית שמקבלת את ערכיה בקובץ זה). מטרת הנקה entry היא לאפיין את התווית זו באופן שיאפשר לקוד אסמבילר הנמצא בקבצי מקור אחרים להשתמש בה (כօפרנד של הוראה).

לדוגמה, השורות :

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

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

לשומות לב : תווית המוגדרת בתחלת שורת entry. הינה חסרת משמעות והאסמבילר מועלם מהתווית זו (אפשר שהאסמבילר יוציא הודעה זהה).

4. הנקה 'extern'.

הנקה 'extern', פרמטר אחד, והוא שם של תווית שאינה מוגדרת בקובץ המקור הנוכחי. מטרת הוראה היא להודיע לאסמבילר כי התווית מוגדרת בקובץ מקור אחר, וכי קוד האסמבילר בקובץ הנוכחי עושה בתווית שימוש.

נשים לב כי הנקה זו תואמת להנקה 'entry'. המופיעה בקובץ בו מוגדרת התווית. בשלב הקישור תבוצע התאמה בין ערך התווית, כפי שנקבע בקוד המכונה של הקובץ שהגידר את התווית, לבין קיודן ההוראות המשמשות בתווית בקבצים אחרים (שלב הקישור אינו רלוונטי למימוש זה).

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

```
.extern HELLO
```

**ערה :** לא ניתן להגיד באותו הקובץ את אותה התווית גם כ-entry וגם כ-extern (בדוגמאות לעיל, התווית HELLO).

**لتשומת לב :** תווית המוגדרת בתחילת שורת `extern`. הינה חסרת משמעות והאסמבלר מתעלם מהתווית זו (אפשר שהאסמבלר יוציא הודעה זהה).

#### משפט הוראה :

משפט הוראה מורכב ממחלקיים הבאים :

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

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

שם הפעולה תמיד באותיות קטנות (lower case), והוא אחת מ- 16 הפעולות שפורטו לעיל.  
לאחר שם הפעולה יופיעו אופרנדים, בהתאם לסוג הפעולה. יש להפריד בין שם-הפעולה לבין האופרנד הראשון באמצעות רווחים ו/או טאים (אחד או יותר).

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזה ב' , ' (פסיק). בדומה להנחייה 'data.' , לא **חייבת להיות הצמדה של האופרנדים לפסיק**. כל כמות של רווחים ו/או טאים 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

#### ಅಧಿಕ್ಷಾತ್ವ ಮಂಜುರಿಗೆ ಸಂಪರ್ಕ ಕೊಳ್ಳಲು

תוויות :

תוית היא סמל שנוגדר בתחילת שפט הוראה' או בתחילת הנקיטת `data`. או `string`.  
תוית חוקית מתחילה באות אלפביתית (גדולה או קטנה), ולאחריה סדרה כלשהי של אותיות אלפביתיות (גדולות או קטנות) ו/או ספרות. האורך המаксימלי של תווית הוא 31 תווים.

הגדרה של **תוית מסתויימת** ב' : ' (נקודתיים). تو זה אינו מהווה חלק מהתוית, אלא רק סימן המציין את סוף ההגדרה. ה' : ' חייב להיות צמוד לתווית (לא רווחים).

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

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hEllo:

x:

He78902:

**لتשומת לב :** מיללים שמורות של שפת האסטמבלי (כלומר שם של פעה או הנחיה, או שם של אוגר) אין יכולות לשמש גם שם של תווית. לדוגמה : הסמלים add, r3 לא יכולים לשמש כתוויות, אבל הסמלים Add, R3, r19, R3 הם תוויות חוקיות.

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

**لتשומת לב :** מותר במשפט הוראה להשתמש באופrnd שהוא סמל שאינו מוגדר כתווית בקובץ הנוכחי, כל עוד הסמל מאופין כחיצוני (באמצעות הנחיה extern). ככלבי בקובץ הנוכחי).

מספר :

מספר חוקי מתחילה בסימן אופציוני : ‘-’ או ‘+’, ולאחריו סדרה של ספרות בסיס عشرוני. לדוגמה : -5, +123, 76 הם מספרים חוקיים. אין תמייה בשפת האסטמבלי שלנוobi ביצוג בסיס אחר מאשר عشرוני, ואין תמייה במספרים שאינם שלמים.

מחרוזות :

מחרוזת חוקית היא סדרת תווי ascii נראים (שניתנים להדפסה), המוקפים במרכאות כפولات (המרכאות אינן נחשבות חלק מהמחרוזות). דוגמה למחרוזת חוקית : “hello world”.

### סימנו המילים בקוד המכונה באמצעות המאפיין ”A,R,E“

האסטמבלר בונה מלכתחילה קוד מכונה שמיועד לטיענה החל מכתובת 100. אולם, לא בכל פעם שהקוד ייטען לזרכו לצורך הרצה, מובטח שאפשר יהיה לטענו אותו החל מכתובת 100. במקרה זה, הקוד המכונה הנוכחי אינו מותאים ויש צורך לתקן אותו. לדוגמה, מילת-המידע של אופrnd בשיטת מיון ישיר לא תהיה נכונה, כי הכתובת השנתנה.

הרעינו הוא להכניס תיקונים נקודתיים בקוד המכונה בכל פעם שייטען לזרכו לצורך הרצה. כך אפשר יהיה לטענו את הקוד בכל פעם למקום אחר, בלי צורך לחזור על הchèק האסטמבל. תיקונים כאלה נעשים בשלב הקישור והטיענה של הקוד (אנו לא מטפלים בכך במילוי זה), אולם על האסטמבלר להוסיף מידע בקוד המכונה שיאפשר להזות את הנקודות בקוד בהן נדרש תיקון.

לצד כל מילה בקוד המכונה, האסטמבלר מוסיף מאפיין שנקרא ”A,R,E“. לכל מילה בקוד, מוצמד שדה המכיל את אחת האותיות A או R או E.

- האות A (קייזר של Absolute) בא להציג שתוכן המילה אינו תלוי במקום בו זיכרונו בו ייטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל, 2 המילים המכילים את קוד ההוראה ואת שיטות המילוי, או מילת-מידע המכילה אופrnd מיידי).
- האות R (קייזר של Relocatable) בא להציג שתוכן המילה תלוי במקומות בו זיכרונו בו ייטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל, מילות-מידע המכילים כתובות של תוויות בצורת כתובות בסיס והיסט).
- האות E (קייזר של External) בא להציג שתוכן המילה תלוי בערכו של סמל שאינו מוגדר בקובץ המקור הנוכחי (למשל, מילת-מידע המכילה ערך של סמל המופיע בהנחיות extern).

**נשים לב** כי רוב המילים בקוד המכונה מאופיניות על ידי האות A. למעשה, רק מילות-המידע הנוספות של שיטות מיון ישיר ושל שיטות מיון אינדקס מאופיניות על ידי האות R או E (תלויה אם האופrnd בקוד האסטמבל) הוא תווית מקומית או סמל חיצוני).

כאשר האסמבילר מקבל כקלט תוכנית בשפת אסמבלי, עליו לטפל תחילה בפרישת המקוראים, ורק לאחר מכן לעבור על התוכנית אליה נפרשו המקוראים. כמובן, פרישת המקוראים תעשה בשלב "קדם אסמבילר", לפני שלב האסמבילר (המtoaר בהמשך).  
אם התוכנית אינה מכילה מקו, תוכנית הפרישה תהיה לתוכנית המקור.

דוגמא לשלב קדם אסמבילר. האסמבילר מקבל את התוכנית הבאה בשפת אסמבלי :

```
; file ps.as
.entry LIST
.extern W

MAIN:    add    r3, LIST
LOOP:    prn   #48
          macro m1
          inc   r6
          mov   r3, W
        endm
          lea    STR, r6
          m1
          sub   r1, r4
          bne   END
          cmp   val1, #-6
          bne   END[r15]
          dec   K
.entry MAIN
          sub   LOOP[r10], r14
END:     stop
STR:     .string "abcd"
LIST:    .data   6, -9
          .data   -100
.entry K
K:       .data   31
.extern val1
```

תחילה האסמבילר עובר על התוכנית ופורש את כל המקוראים הקיימים בה. רק אם תהליך זה מסתיים בהצלחה, ניתן לעבור לשלב הבא. בדוגמה זו, התוכנית לאחר פרישת המקורו תיראה כך :

```

; file ps.am
.entry LIST
.extern W

MAIN:    add    r3, LIST
LOOP:     prn    #48
          lea    STR, r6
          inc    r6
          mov    r3, W
          sub    r1, r4
          bne    END
          cmp    val1, #-6
          bne    END[r15]
          dec    K

.entry MAIN
          sub    LOOP[r10], r14
END:      stop
STR:      .string "abcd"
LIST:     .data   6, -9
          .data   -100
.entry K
K:        .data   31
.extern val1

```

קוד התוכנית, לאחר הפרישה, ישמור בקובץ חדש, כפי שיווסף בהמשך.

### **אלגוריתם שלי של קדם האסמבילר**

נציג להלן אלגוריתם שלי לתחליק קדם האסמבילר. لتשומת לב: אין חובה להשתמש דווקא באלגוריתם זה:

1. קרא את השורה הבאה מקובץ המקור. אם נגמר הקובץ, עבור ל- 9 (סיום).
2. האם השדה הראשון הוא שם מקורי המופיע בטבלת המקור (כגון m1)? אם כן, החלף את שם המקור והעתק במקומו את כל השורות המתאימות מהטבלה לקובץ, חוזר ל- 1. אחרת, המשך.
3. האם השדה הראשון הוא **"macro"** (התחלת הגדרת מקורי)? אם לא, עבור ל- 6.
4. הדלק דגל **"יש macro"**.
5. (קיימת הגדרת מקורי) הכנס לטבלת שורות מקורי את שם המקור (לדוגמה m1).
6. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 9 (סיום). אם דגל **"יש macro"** דולק ולא זההתו תוויות endm הכנס את השורה לטבלת המקור ומחק את השורה הנ"ל מהקובץ. אחרת (לא מקורי) חוזר ל- 1.
7. האם זההתו תוויות endm? אם כן, מחק את התוויות מהקובץ והמשך. אם לא, חוזר ל- 6.
8. כבנה דגל **"יש macro"**. חוזר ל- 1. (סיום שמירת הגדרת מקורי)
9. סיום : שמירת קובץ מקורי פרוש.

cut-after-freeall all-macros is passed to the translation stage for the assembly language, stage of assembly.

### **אסמבילר עם שני מעברים**

בעבר הראשון של האסמבילר, יש לזהות את הסמלים (תוויות) המופיעים בתוכנית, ולהתגצלם לכל סמל ערך מסווני שהוא המعن זיכרנו שהסמל מייצג. בעבר השני, באמצעות ערכי הסמלים, וכן קודי-הפעולה ומספריה האוגרים, בונים את קוד המוכנה.

קוד המכוна של התוכנית (הווראות ונטוונם) נבנה כך שייתאים לטעינה בזיכרון ה

- חל ממען 100 (עשרוני).

התרגום של תוכנית המקור שבדוגמא לקוד בינארי מוצג להלן:

Address (decimal)	Source Code	Machine Code (binary)																			
		19	"	"	"	"	"	"	"	9	8	7	6	5	4	3	2	1	0		
0100	MAIN: add r3, LIST	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0		
0101		0	1	0	0	1	0	1	0	0	0	1	1	1	0	0	0	0	1		
0102		0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0		
0103		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1		
0104	LOOP: prn #48	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
0105		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0106		0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	
0107	lea STR, r6	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0108		0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1
0109		0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0110		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
0111	inc r6	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0112		0	1	0	0	1	1	0	0	0	0	0	0	0	0	1	1	0	1	1	1
0113	mov r3, W	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0114		0	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1
0115		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0116		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0117	sub r1, r4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0118		0	1	0	0	1	0	1	1	0	0	0	1	1	1	0	1	0	0	1	1
0119	bne END	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0120		0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1
0121		0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0122		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0123	cmp vall, #-6	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0124		0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0125		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0126		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0127		0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0
0128	bne END[r15]	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0129		0	1	0	0	1	0	1	1	0	0	0	0	0	0	1	1	1	1	1	0
0130		0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0131		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0132	dec K	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0133		0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1
0134		0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
0135		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
0136	sub LOOP[r10],r14	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0137		0	1	0	0	1	0	1	1	0	1	0	1	0	1	1	1	0	1	0	1
0138		0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0139		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0140	END: stop	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0141	STR: .string "abcd"	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
0142		0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0
0143		0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1
0144		0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0
0145		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0146	LIST: .data 6, -9	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
0147		0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
0148	.data -100	0	1	0	0	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0
0149	K: .data 31	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

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

כדי לבצע המרה לבינארי של אופרנדים כתובים בשיטות מייען המשמשות בסמלים (תוויות), יש ליצור לבנות טבלה המכילה את ערכיו כל הסמלים. אולם בהבדל ממקודים של הפעולות, הידועים מראש, הרי המענינים בזכירון עבור הסמלים בתוכנית אינם ידועים, עד אשר תוכניתה המקור נסרקה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד לעיל, האסמבller אינו יכול לדעת שהסמל END אמור להיות משוויך למן 140 (עשרהוני), והסמל K אמור להיות משוויך למן 149, אלא רק לאחר שנקראו כל שורות התוכנית.

לכן מפרידים את הטיפול של האסמבller 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 בזכירון, ובה לכל סמל שבתוכנית המקור משוויך ערך מסווני, שהוא מן בזכירון .

במעבר השני נעשית המירה של קוד המקור לקוד מכונה. בתחלת המעבר השני צריכים הערכים של כל הסמלים להיות כבר ידועים

עבור הדוגמה, טבלת הסמלים נתונה להלן. לכל סמל יש בטבלה גם מאפיינים (attributes) שיוסברו בהמשך. אין חשיבות לסדר השורות בטבלה (כאן הטבלה לפי הסדר בו הוגדרו הסמלים בתוכנית).

Symbol	Value (decimal)	Base address	offset	Attributes
W	0	0	0	external
MAIN	100	96	4	code, entry
LOOP	104	96	8	code
END	140	128	12	code
STR	141	128	13	data
LIST	146	144	2	data, entry
K	149	144	5	data, entry
val1	0	0	0	external

لتשומת לב : תפקיד האסמבller, על שני המעברים שלו, לתרגם קובץ מקור לקוד בשפת מכונה. בגמר פעולות האסמבller, התוכנית טרם מוכנה לטעינה לזכירון לצורך ביצוע. קוד המכונה חייב לעבור לשבי הקשר/טעינה, ורק לאחר מכן לשלב הביצוע (שלבים אלה אינם חלק מההמם"ז).

### המעבר הראשון

במעבר הראשון נדרשים כלליים כדי לקבוע איזה מן ישוויך לכל סמל. העיקרונו הבסיסי הוא למספר את המיקומות בזכירון, אותן תופסות ההוראות. אם כל הוראה תופיען בזכירון במקום העקב להוראה הקודמת, תציגו ספירה כזו את מעו ההוראה הבאה. הספירה נעשית על ידי האסמבller ומוחזקת במבנה ההוראות (IC). ערכו החתולתי של IC הוא 100 (עשרהוני), וכן קוד המכונה של ההוראה הראשונה נבנה כך שיופיען בזכירון החל ממען 100. ה-IC מתעדכן בכל שורה ההוראה המקצתה מקום בזכירון. לאחר שהאסמבller קובע מהו אורך ההוראה, ה-IC מוגדל במספר התאים (מילימ) הנתפסים על ידי ההוראה, וכך הוא מצבע על התא הפוני הבא.

כאמור, כדי לקודד את ההוראות בשפת מכונה, מחזיק האסמבller טבלה, שיש בה קידוד מתאים לכל שם פעולה. בזמן התרוגום מחליף האסמבller כל שם פעולה בקידוד שלה. כמו כן, כל אופרנד מוחלף בקידוד מותאים, אך פועלות החלפה זו אינה כה פשוטה. ההוראות משתמשות בשיטות מייען, מיעון מגוונות לאופרנדים. אותה פעולה יכולה לקבל שימושויות שונות, בכל אחת משתמשות במישות המיעון, וכן יתאיםו לה קידודים שונים לפי שיטות המיעון. לדוגמה, פועלות ההזזה Zus או יכולה להתיחס

להעתיקת תוכן תא זיכרון לאוגר, או להעתיקת תוכן אוגר לאוגר אחר, וכן הלאה. ככל אפשרות זאת של **750m** עשוי להתאים קידוד שונה.

על האסטבלר לסרוק את שורת ההוראה בשלמותה, ולהחליט לגבי הקידוד לפי האופרנדים. בדרך כלל מתחלק הקידוד לשדה של שם הפעלה, וסדרות נוכפים המכילים מידע לגבי שיטות המיען. כל השדות ביחד דורשים מילה אחת או יותר בקוד המכוונה.

כאשר נתקל האסטבלר בתוויות המופיעות בתחילת השורה, הוא יודע שלפניו הגדרה של תווית, ואז הוא משיך לה מען – תוכנו הנוכחי של IC. כך מקבלות כל התוויות את מענהן בעקבות ההגדרה. תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנוסף לשם התווית גם את המען ומאפיינים נוספים. כאשר תהיה התיקוסות לתווית באופרנד של הוראה כלשהי, יוכל האסטבלר לשלוות את המען המתאים מטבלת הסמלים.

ההוראה יכולה להתיחס גם לסדר טרם הוגדר עד כה בתוכנית, אלא יוגדר רק בהמשך התוכנית. להלן, לדוגמה, הוראות הסתעפות מען שמוגדר על ידי התווית A שמשמעותו רק בהמשך הקוד :

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

כאשר מגיע האסטבלר לשורת הסתעפות (A bne), הוא טרם נתקל בהגדרת התווית A וכמוון לא יודע את המען המשויך לתווית. לכן האסטבלר 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 ההוראה, את הקידוד הבינארי של מילת-המידע הנוספת של אופרנד מיידי, או אוגר, וכן את הקידוד הבינארי של כל הנתונים (המתקבלים מההנחיות `.string`, `.data`, `.string`).

## המעבר השני

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

לשומך מבצע האסטבלר מעבר נוסף (מעבר שני) על כל קובץ המקור, ומעדכן את קוד המכוונה של האופרנדים המשמשים בסמלים, באמצעות ערכי הטבלת הסמלים. בסוף המעבר השני, תהיה התוכנית מתורגמת בשלמותה לקוד מכונה.

## הפרצת הוראות ונתונים

בתוכנית מבחנים שני סוגי של תוכן: הוראות ונתונים. יש לארגן את קוד המכוונה כך שתתיה הפרדה בין הנתונים וההוראות. הפרצת ההוראות והנתונים לקטעים שונים בזיכרון היא שיטה עדיפה על פני הצמדה של הגדרות הנתונים להוראות המשמשות בהן.

אחת הסכנות הטමונות באירוע הפרדת ההוראות מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית בתוכנית, לנסוטו "לבצע" את הנתונים אליו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום לתופעה כזו היא הסתעפות לא נוכנה. התוכנית כموון לא תעבור נכוון, אך לרוב הנזק הוא יותר חמור, כי נוצרת חריגת חומרה ברגע שהמעבד מבצע פעולה שאינה חוקית.

האסטבלר שלנו חייב **להפריד**, בקוד המכוונה שהוא מיצר, בין קטוע הנתונים לקטוע ההוראות. **כלומר בקובץ הפלט (בקוד המכוונה) תהיה הפרדה של הוראות ונתונים לשני קטיעים נפרדים,** אם כי **בקובץ הקלט אין חובה שתהייה הפרדה כזו.** בהמשך מתוואר אלגוריתם של האסטבלר, ובו פרטים כיצד לבצע את ההפרדה.

## גילוי שגיאות בתוכנית המקור

כפי שהוסבר לעל, הנחת המטלה היא שאין שגיאות בהגדירות המקור, ולכן שלב קדם לאסמבלי אין מכיל שלב גילוי שגיאות, לעומת זאת האסמבלי אמר לגלות ולדוח על שגיאות בתחריב של תוכנית המקור, כגון פעליה שאינה קיימת, מספר אופרנדים שני, סוג אופרנד שאין מתאים לפעליה, שם אונגר לא קיים, ועוד שגיאות אחרות. כמו כן מודיא האסמבלי שכל סמל מוגדר פעמי אחת בדיק.

מכאן, שככל שגיאה המתגלה על ידי האסמבלי נגרמת (בדרכם כלל) על ידי שורת קלט מסויימת. לדוגמה, אם מופיעים שני אופרנדים בהוראה שאמור להיות בה רק אופרנד אחד, האסמבלי ייתן הודעת שגיאה בנוסח "יותר מדי אופרנדים".

הערה: אם יש שגיאה בקוד האסמבלי בגורף מקורו, הרי שגיאה זו יכולה להופיע ולהתגלושוב ושוב, בכל מקום בו נפרש המקורו. נשים לב לכך שהאסמבלי בודק שגיאות, כבר לא ניתן להזוזה שזה קוד שנפרש מקורו, כך שלא ניתן לחסוך גילוי שגיאה כפולית.

האסמבלי ידפיס את ה הודעות השגיאה אל הפלט הסטנדרטי `stdout`. בכל הודעת שגיאה יש לציין גם את מספר השורה בקובץ המקור בה זוהתה השגיאה (מנין השורות בקובץ מתחילה ב-1).

لتשומת לב: האסמבלי אינו עובד את פעולתו אחרי שנמצאה השגיאה הראשונה, אלא ממשיך לעبور על הקלט כדי לגלוות שגיאות נוספות, ככל שישן. כMOVן שאין כל טעם לייצר את קבצי הפלט אם נתגלו שגיאות (ممילא אי אפשר להשלים את קוד המכונה).

הטבלה הבאה מפרטת מהן של שיטות המיענו 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
1,2,3	0,1,2,3	mov		0
0,1,2,3	0,1,2,3	cmp		1
1,2,3	0,1,2,3	add	10	2
1,2,3	0,1,2,3	sub	11	2
1,2,3	1,2	lea		4
1,2,3	אין אופרנד מקור	clr	10	5
1,2,3	אין אופרנד מקור	not	11	5
1,2,3	אין אופרנד מקור	inc	12	5
1,2,3	אין אופרנד מקור	dec	13	5
1,2	אין אופרנד מקור	jmp	10	9
1,2	אין אופרנד מקור	bne	11	9
1,2	אין אופרנד מקור	jsr	12	9
1,2,3	אין אופרנד מקור	red		12
0,1,2,3	אין אופרנד מקור	prn		13
אין אופרנד יעד	אין אופרנד מקור	rts		14
אין אופרנד יעד	אין אופרנד מקור	stop		15

## תהליך העבודה של האסמבלי

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

האסמבלי מתחזק שני מערכיים, שייקראו להלן תMOVנות ההוראות (code) ותMOVנות הנתונים (data). מערכיים אלו נתונים למשהה תMOVנה של זיכרון המכונה (כל איבר בזיכרון הוא בגודל מילה של המכונה, כולם 24 סיביות). בזיכרון ההוראות בונה האסמבלי את הקידוד של הוראות המכונה שנקרו בו מהלך המעבר על קובץ המקור. בזיכרון הנתונים מכניס האסמבלי את קידוד הנתונים שנקרו מקובץ המקור (שורות הינה מסוג `data`. ו- `'string'`).

האסמבלר משתמש בשני מונחים, שנקראים IC (מונה ההוראות - Instruction-Counter) ו- DC (Data-Counter). מונחים אלו מצביעים על המיקום הבא הפנוי במערך ההוראות ובמערך הנתונים, בהתאם. בכל פעם שמתחליל האסמבלר עבר על קובץ מקור, המונה IC מקבל ערך תחלי 100, והמונה DC מקבל ערך תחלי 0. הערך התחלתי IC=100 נקבע כדי שקוד המכמה של התוכנית יתאים לטעינה ליזכרון (לצורך ריצה) החל McMOTOBET 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 נשמרות בטבלה שם הסמל, ערכו המספרי, ומאפיינים נוספים (אחד או יותר), כגון המיקום בתמונה הזיכרון (entry) אוernal (או code).

במעבר הראשון האסמבלר בונה את טבלת הסמלים ואת השילד של תמונה 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 וועבר לשורה הבאה.

2. שורת הוראה :

האסמבלר מנתח את השורה ומפענח מהי ההוראה, ומahan 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 '#' מצין מעון מיידי, תווית מצינית מעון שיר, שם של אוגר מצין מעון אוגר שיר, וכו'.

אם האסמבלר מוצא בשורת ההוראה גם הגדרה של תווית, אזו התווית (הסמל) המוגדרת מוכנסת לטבלת הסמלים. ערך הסמל בטבלה הוא IC, והמאפיין הוא code.

כעת האסמבלר קבוע לכל אופרנד את ערכו באופן הבא :

- אם זה אוגר – האופרנד הוא מספר האוגר.
- אם זו תווית (מעון שיר) – 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 (מעון מיידי) – האופרנד הוא המספר עצמו.
- אם זו שיטת מעון אחרת – ערכו של האופרנד נקבע לפי המפרט של שיטת המיעון (ראו תאור שיטות המיעון לעיל).

האסמבלר מכניס למערך ההוראות, בכניסה אליה מצביע מונה ההוראות IC, את הקוד הבינארי המלא של המילה הראשונה של ההוראה (בפורמט קידוד כפי שתואר קודם). מילה זו מכילה את קוד הפעולה וה-funct, ואת מספרי שיטות המיעון של אופרנד המקור והיעד. IC מקודם ב-1.

זכור שכאשר יש רק אופרנד אחד (כלומר אין אופרנד מקור), הסיביות של שיטת המיעון של אופרנד המקור יכilo 0. בדומה, אם זהה הוראה ללא אופרנדים (ts, stop), אזו הסיביות של שיטות המיעון של שני האופרנדים יכilo 0.

אם זהה הוראה עם אופרנדים (אחד או שניים), האסמבלר "משרין" מקומות במערך ההוראות עבור מיליה-המידע הנוספת הנדרשות בהוראה זו, ככל שנדרשות, ומקדם את IC בהתאם. כאשר אופרנד הוא בשיטת מעון מיידי או אוגר שיר, האסמבלר מקודד גם את המילה הנוספת המתאימה במערך ההוראות. ואילו בשיטת מעון שיר או יחס, מיליה 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'

האSEMBLER קורא את רשימת המספרים, המופיעעה לאחר '.data', מכניס כל מספר אל מערך הנתונים (בקידוד ביארי), ומקדם את מצביע הנתונים DC ב-1 עבר כל מספר שהוכנס.

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

## II. '.string'

הטיפול ב'.string' דומה ל'.data', אלא שקובדי ascii של התווים הם אלו המוכנסים אל מערך הנתונים (כלתו במילה נפרצת). לבסוף מוכנס למערך הנתונים הערך 0 (המצין 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'

זהה הנחיה לאSEMBLER לאפיון את התווית הנתונה כאופrnd c-entry בטבלת הסמלים. בעת הפuktת קבצי הפלט (ראו בהמשך), התווית תירשם בקובץ entries.

لتשומת לב: זה לא נחשב כשגיאה אם בקובץ המקור מופיעות יותר מהנחיה entry. אחת עם אותן תוויות כאופrnd. המופיעים הנוספים אינם מושפעים דבר, אך גם אינם מפריעים.

## IV. '.extern'

זהה הצהרה על סמל (תווית) המוגדר בקובץ מקור אחר, והקובץ המקורי עושה בו שימוש. האSEMBLER מכניס את הסמל המופיע כאופrnd לטבלת הסמלים, עם הערך 0 (הערך האמיינטי לא ידוע, ויקבע רק בשלב הקישור), ועם המאפיין external. לא ידוע באיזה קובץ נמצאת הגדרת הסמל, ואין זה רלוונטי עבור האSEMBLER.

لتשומת לב: זה לא נחשב כשגיאה אם בקובץ המקור מופיעות יותר מהנחיה extern. אחת עם אותן תוויות כאופrnd. המופיעים הנוספים אינם מושפעים דבר, אך גם אינם מפריעים.

لتשומת לב: באופrnd של הוראה או של הנחיה entry, מותר להשתמש בסמל אשר יוגדר בהמשך הקובץ (אם באופן ישיר על ידי הגדרת תווית, ואם באופן עקיף על ידי הנחיה extern).

בסוף המעבר הראשו, האSEMBLER מעדכן בטבלת הסמלים כל סמל המופיע כ-.data, על ידי הוספת (100) IC + (עשרוני) לערכו של הסמל. הסיבה לכך היא שבתמונה הכלולת של קוד המוכונה, תמונה נתוניות מופרצת מתמונה הוראות, וכל הנתונים נדרשים להופיע בקוד המוכונה אחרי כל ההוראות. סמל מסווג data הוא תווית בתמונה הנתונים, והעדכו מוסיף לערך הסמל (כלומר כתובתו בזיכרון) את האורך הכלול של תמונה ההוראות, בתוספת כתובת התחלה הטעינה של הקוד, שהיא 100.

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

במעבר השני, האSEMBLER משלים באמצעות טבלת הסמלים את קידוד כל המילים בערך ההוראות שטרם קודדו במעבר הראשון. במודול המכונה שלו אלו הן מילוט-מידע נוספת של ההוראות, אשר מקודדות אופrnd בשיטת מייען ישיר או יחסי.

## אלגוריתם שלי של האSEMBLER

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

لتשומת לב: אין חובה להשתמש דווקא באלגוריתם זה.

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

نبנה את קוד המכונה כך שיתאים לטעינה לזכרון החל מכתובת 100.

בכל מעבר מתחילה לקרוא את קובץ המקור מההתחלתה.

### מעבר ראשון

- .1. אתחל  $100 \leftarrow IC, DC \leftarrow 0$ .
- .2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עברו ל-7.
- .3. האם השדה הראשון בשורה הוא תווית? אם לא, עברו ל-5.
- .4. הדלק דגל "יש הגדרת סמל".
- .5. האם זהה הנחיה לאחסון נתונים, כמו למשל, `char data = 'A'`? אם לא, עברו ל-8.
- .6. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם המאפיין `data`. ערך הסמל יהיה בסיס והיסטט. (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה, יש להודיע על שגיאה).
- .7. זהה את סוג הנתונים, קודו אותם בתמונה הנתונים, והגדל את מונה הנתונים `DC` על ידי הוספת האורך הכלול של הנתונים שהוגדרו בשורה הנוכחית. חזור ל-2.
- .8. האם זו הנחיה `extern`? אם לא, עברו ל-11.
- .9. אם זהה הנחיה `entry`, חזור ל-2 (ההנחיה תטופל במעבר השני).
- .10. אם זו הנחיה `extern`, הכנס את הסמל המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים עם פעמיים הערך 0 (בסיס והיסטט), ועם המאפיין `external`. (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה ללא המאפיין `external`, יש להודיע על שגיאה). חזור ל-2.
- .11. זהה שורת הוראה. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם המאפיין `code`. ערכו של הסמל יהיה בסיס והיסטט (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה, יש להודיע על שגיאה).
- .12. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא, אז הודיע על שגיאה בשם ההוראה.
- .13. נתח את מבנה האופרנדים של ההוראה, וחשב מהו מספר המילים הכוללתו לפחות אחת מההוראה בקוד המכונה (נקרא למספר זה  $L$ ).
- .14. בנה כעת את הקוד הבינארי של המילה הראשונה של ההוראה, ושל כל מילת- מידע נוספת המוקדדת אופרנד במיעון מיידי. אפשר לקוד גם את המילה השנייה בקוד ההוראה (אם קיימות).
- .15. שמר את הערכים  $IC$  ו-  $L$  יחד עם תנאי קוד המכונה של ההוראה.
- .16. עדכן  $IC \leftarrow IC + L$ , וחזור ל-2.
- .17. קובץ המקור נקרא בשלמותו. אם נמצא שגיאות במעבר הראשון, עצור כאן.
- .18. שמר את הערכים הסופיים של  $IC$  ושל  $DC$  (נקרא להם `ICF` ו- `DCF`). השתמש בהם לבניית קבצי הפלט, אחרי המעבר השני.
- .19. עדכן בטבלת הסמלים את ערכו של כל סמל המופיע כ- `data`, ע"י שימוש בערך `ICF` באופן הבא: ראשית יש לחשב את ערכו המלא המעודכן של הסמל בהינתן `ICF`-והבסיס+היסטט הנוכחיים (ככל שהוא מופיע כך בטבלה), ואז לחשב בסיס+היסטט חדשם.
- .20. התחל מעבר שני.

### מעבר שני

- .1. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עברו ל-7.
- .2. אם השדה הראשון בשורה הוא סמל (תווית), דרג לעילו.
- .3. האם זהה הנחיה `data`? אם כן, חזור ל-1.
- .4. האם זהה הנחיה `entry`? אם לא, עברו ל-6.
- .5. הוסף בטבלת הסמלים את המאפיין `entry` למאפייני הסמל המופיע כאופרנד של ההנחיה (אם הסמל לא נמצא בטבלת הסמלים, יש להודיע על שגיאה). חזור ל-1.
- .6. השלים את הקידוד הבינארי של מילוט-המידע של האופרנדים, בהתאם לשיטות המיעון בשימוש. 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 הרלוונטי לשיממת מילוט-המידע שמתיחסות לסמל חיצוני. לפי הatzroch, לחישוב הקידוד והכתובות, אפשר להיעזר בערכים `IC` ו- `L` של ההוראה, כפי שנשמרו במעבר

- הראשון. חוזר ל- 1. לתשומת לב : יש להשלים שתי מילוט מידע לכל אופרנד. כמו כן, אם מדובר בסמל חיצוני, יש לשום בקובץ ext את הכתובות של שתי מילוט המידע. לפי הגדרת השפה, כתובות מילת החיסט תהיה תמיד עוקבת לכתובות מילת הבסיס (דוגמת קובץ ext בהמשך) .
- .7. קובץ המקור נקרא שלמדו. אם נמצאו שגיאות מעבר השני, עזרו כאן.
- .8. בנה את קבצי הפלט (פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו לעיל **(לאחר שלב פרישת המקראים)**, ונציג את הקוד הבינארי שמקבל במעבר ראשון ובמעבר שני. להלן שוב תכנית הדוגמה.

```
; file ps.as
.entry LIST
.extern W
MAIN:    add    r3, LIST
LOOP:     prn   #48
          lea    STR, r6
          inc    r6
          mov    r3, W
          sub    r1, r4
          bne   END
          cmp    val1, #-6
          bne   END[r15]
          dec    K
.entry MAIN
          sub    LOOP[r10], r14
END:      stop
STR:      .string "abcd"
LIST:     .data   6, -9
          .data   -100
.entry K
K:        .data   31
.extern val1
```

נבצע מעבר ראשון על הקוד לעיל, וنبנה את טבלת הסמלים. כמו כן, נשלים במעבר זה את הקידוד של כל תומנות הנתונים, ושל המילה הראשונה של כל הוראה (ניסי לבייש לקודד גם את המילה השנייה). כמו כן, נקודד מילוט-מידע נוספת של כל הוראה, ככל שקידוד זה אינו תלוי בערך של סמל. את מילוט-המידע שעדינו לא ניתן לקודד במעבר הראשון נסמן ב "?" בדוגמה להלן.

Address (decimal)	Source Code	Machine Code (binary)																		
		19	"	"	"	"	"	"	"	"	"	9	8	7	6	5	4	3	2	1
0100	MAIN: add r3, LIST	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0101		0	1	0	0	1	0	1	0	0	0	1	1	1	1	0	0	0	0	1
0102		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0103		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0104	LOOP: prn #48	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0105		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0106		0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0107	lea STR, r6	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0108		0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0
0109		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0110		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0111	inc r6	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0112		0	1	0	0	1	1	0	0	0	0	0	0	0	0	1	1	0	1	1
0113	mov r3, W	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0114		0	1	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1
0115		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0116		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0117		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0118	bne END	0	1	0	0	1	0	1	1	0	0	0	1	1	1	0	1	0	0	1
0119		0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0120		0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	1

Address (decimal)	Source Code	Machine Code (binary)																			
		19	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	..	
0121		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	
0122		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	
0123	cmp val1, #-6	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0124		0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0125		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0126		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0127		0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	
0128	bne END[r15]	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0129		0	1	0	0	1	0	1	1	0	0	0	0	0	0	1	1	1	1	0	
0130		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0131		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0132	dec K	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0133		0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	1
0134		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0135		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0136	sub LOOP[r10],r14	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0137		0	1	0	0	1	0	1	1	1	0	1	0	1	1	1	0	1	1	0	1
0138		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0139		?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
0140	END: stop	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0141	STR: .string "abcd"	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
0142		0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
0143		0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
0144		0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0
0145		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0146	LIST: .data 6, -9	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
0147		0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
0148	.data -100	0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0
0149	K: .data 31	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

טבלת הסמלים אחרי מעבר ראשון היא :

Symbol	Value (decimal)	Base address	offset	Attributes
W	0	0	0	external
MAIN	100	96	4	code, entry
LOOP	104	96	8	code
END	140	128	12	code
STR	141	128	13	data
LIST	146	144	2	data, entry
K	149	144	5	data, entry
val1	0	0	0	external

בוצע עתה את המעבר השני. נשלים באמצעות טבלת הסמלים את הקידוד החסר בambilים המסומנים ???. הקוד הבינארי בצוותו הסופית כאן זהה לקוד שהוזג בתחילת הנושא **"אסמבלר עם שני מעברים."**

הערה : כאמור, האסmbלר בונה קוד מכונה כך שיתאים לטיעינה לזכרון החל מכתובת 100 (עשרה). אם הטיעינה בפועל (לצורך הרצאת התוכנית) תהיה לכמתות אחרות, יידרשו תיוקנים בקוד הבינארי בשלב הטיעינה, שיוכנסו בעורת מידע נוסף שהאסmbלר מכין בקבצי הפלט (ראו בהמשך).

Address (decimal)	Source Code	Machine Code (binary)																
		19	"	"	"	"	"	"	9	8	7	6	5	4	3	2	1	0
0100	MAIN: add r3, LIST	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0101		0	1	0	0	1	0	0	0	1	1	1	0	0	0	0	1	
0102		0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0
0103		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0104	LOOP: prn #48	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0105		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0106		0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0107	lea STR, r6	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0108		0	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1
0109		0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0110		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0111	inc r6	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0112		0	1	0	0	1	1	0	0	0	0	0	0	0	1	1	0	1
0113	mov r3, W	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0114		0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1
0115		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0116		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0117	sub r1, r4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0118		0	1	0	0	1	0	1	0	0	0	1	1	1	0	1	0	1
0119	bne END	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0120		0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1
0121		0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0122		0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
0123	cmp vall, #-6	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0124		0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0125		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0126		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0127		0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	1	0
0128	bne END[r15]	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0129		0	1	0	0	1	0	1	1	0	0	0	0	0	1	1	1	1
0130		0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0131		0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
0132	dec K	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0133		0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1
0134		0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0
0135		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0136	sub LOOP[r10],r14	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0137		0	1	0	0	1	0	1	1	0	1	0	1	0	1	1	0	1
0138		0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0139		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0140	END: stop	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0141	STR: .string "abcd"	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0142		0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
0143		0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1
0144		0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0
0145		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0146	LIST: .data 6, -9	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0147		0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1
0148	.data -100	0	1	0	0	1	1	1	1	1	1	1	0	0	1	1	1	0
0149	K: .data 31	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

טבלת הסמלים אחרי מעבר שני היא :

Symbol	Value (decimal)	Base address	offset	Attributes
W	0	0	0	external
MAIN	100	96	4	code, entry
LOOP	104	96	8	code
END	140	128	12	code
STR	141	128	13	data
LIST	146	144	2	data, entry
K	149	144	5	data, entry
val1	0	0	0	external

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

### קבצי קלט ופלט של האסמבLER

בפעולת של האסמבLER, יש להעביר אליו באמצעות ארגומנטים של שורת הפקודה (command line arguments) רשימה של שמות קבצי מקור (אחד או יותר). אלו הם קבצי טקסט, ובهم תוכניות בתחביר של שפת האסמבLER שהוגדרה במיל'ין זה.

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

- קובץ .am, המכיל את קובץ המקור לאחר שלב קדם האסמבLER (לאחר פרישת המקרואים).
- קובץ .object, המכיל את קוד המכמה.
- קובץ .externals, ובו פרטים על כל המקבומות (התובות) בקוד המכמה בהם יש מילת-מידע שמוקודדת ערך של סמל שהזכר חיצוני (סמל שהופיע כאופרנד של הנחיה .extern, ומופיע בטבלת הסמלים .external).
- קובץ .entries, ובו פרטים על כל סמל שימוש נקודות כניסה (סמל שהופיע כאופרנד של הנחיה .entry, ומופיע בטבלת הסמלים .entry).

אם אין בקובץ המקור אף הנחיה .extern, האסמבLER לא יוצר את קובץ הפלט מסווג .externals. אם אין בקובץ המקור אף הנחיה .entry, האסמבLER לא יוצר את קובץ הפלט מסווג .entries.

שמות קבצי המקור חייבים להיות עם הסיומת ".as". למשל, השמות .x, .y.as, .as.y והם שמות חוקיים. העברת שמות הקבצים הללו כארוגמנטים לאסמבLER נעשית לא ציון הסיומת.

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

assembler x y hello

תրץ את האסמבLER על הקבצים : .x.as, .y.as, hello.as

שמות קבצי הפלט מבוטסים על שם קובץ הקלט, כפי שהופיע בשורת הפקודה, בתוספת סיומת .object. מתאימה : הסיומת ".am". עברו קובץ לאחר פרישת מקשו, הסיומת ".ob". עברו קובץ ה-.externals. הסיומת ".ent". עברו קובץ ה-.ext, והסיומת ".am". עברו קובץ ה-.as.

לדוגמה, בפעולת האסמבLER באמצעות שורת הפקודה : assembler x .y.as .x.ext .ob. יוצר קובץ פלט .ob.x, וכן קבצי פלט .ext.x. וכל שיש הנחיות .entry.או .extern. בקובץ המקור. אם אין מקשו בקובץ המקור, אז קובץ ".am". יהיה זהה לקובץ ".as".

ນציג כעת את הפורמטים של קבצי הפלט. דוגמאות יובאו בהמשך.

### **פורמט קובץ ה-object**

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

כזכור, האסטבלר מקודד את ההוראות כך שתמונה ההוראות תתאים לטעינה החל מכתובת 100 (עשרות) בזיכרון. נשים לב שرك המ עבר הראשו יודיעים מהו הגודל הכללי של תמונה ההוראות. מכיוון שתמונה הנתונים נמצאת אחרי תמונה ההוראות, גודל תמונה ההוראות משפיע על הכתובות בתמונה הנתונים. זו הסיבה שבגללה היה צריך לעמוד בטבלת הסמלים, בסוף המ עבר הראשו, את ערכי הסמלים המאופיינים כ-data (כזכור, באלוירית השודי שהציג עיל, בצד 19, הוסיף לכל סמל נזה את הערך ICF). במעבר השני, בהשלמת הקידוז שלAMILOT-המידע, משתמשים בערכים המעודכנים של הסמלים, המותאמים למבנה המלא והסופי של תמונה הזיכרון.

כעת האסטבלר יכול לכתוב את תמונה הזיכרון בשלהמota לתוכן קובץ פלט (קובץ ה-object).

השורה הראשונה בקובץ ה-object היא "cotratt", המכילה שני מספרים (בבסיס עשרוני) : הראשון הוא האורך הכללי של תמונה ההוראות (בAMILOT זיכרון), והשני הוא האורך הכללי של תמונה הנתונים (AMILOT זיכרון). בין שני המספרים מפריד רווח אחד. כזכור, במעבר הראשון, בצד 19, נשמרו ערכי הסמלים תוך שימוש ב-ICF.

השורות הבאות בקובץ מכילות את תמונה הזיכרון. בכל שורה זוג שדות : כתובות של מילה בזיכרון, ותוכן המילה. הכתובות תירישם בסיס עשרוני באربع ספרות (כולל אפסים מוביילים). תוכן המילה יירשם **בבסיס "מיוחד"** ב-3 ספרות (כולל אפסים מוביילים). בין השדות בשורה יש רווח אחד.

#### **"בסיס מיוחד"**

כל שורה בתמונה הזיכרון היא באורך 20 סיביות, החל מסיבית 0 (מיימין) ועד לסייבית 19 (משמאלי). נחלק את 20 הסיביות ל-5 קבוצות בניו 4 סיביות בכל קבוצה כך :

- סיביות 19-16 יקראו קבוצה A
- סיביות 12-15 יקראו קבוצה B
- סיביות 8-11 יקראו קבוצה C
- סיביות 4-7 יקראו קבוצה D
- סיביות 0-3 יקראו קבוצה E

כל 4 סיביות של קבוצה מסוימת יומרו בספרה הקסדצימלית וייכתו לקובץ ה-OB באופן הבא :  
תילה ייכתב שם הקבוצה באות גדולה, ולאחריו הייצוג הקסדצימלי של סיביות הקבוצה. כך  
"יכתו כל הסיביות מכל הקבוצות עם הפרדה של תו מקף (-) בין קבוצה לקבוצה.  
לדוגמה, נסתכל על 20 הסיביות הבאות :

0	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

הם ייכתו לקובץ OB כך :

A4-B0-C3-Dc-E1

### **פורמט קובץ ה-entries**

קובץ ה-entries בניית טקסט, שורה אחת לכל סמל שמאופיין בטבלת הסמלים כ-entry. בשורה מופיע שם הסמל, ולאחריו כתובות הבסיס שלו וההיסטוריה, כפי שנקבע בטבלת הסמלים (בבסיס עשרוני באربع ספרות, כולל אפסים מוביילים). אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה. בין השורות בשורה יש פסיק אחד.

### **פורמט קובץ ה- *externals***

קובץ ה-*externals* בניו אף הוא משורות טקסט, שורה לכל כתובות בקובץ המcona בה יש מילת מידע המתיחסת לסמל שماופיין כ-*external*. כזכור, רשיימה של מילות-מידע אלה נבנתה בעבר השני (צעד 6 באלגוריתם השלב).

כל שורה בקובץ ה-*externals* מכילה את שם הסמל החיצוני, ולאחריו המילה BASE ולאחריה הכתובות של מילת-המידע בקובץ המכונה בה נדרשת כתובות הבסיס (בבסיס עשרוני באربע ספרות, כולל אפסים מוביילים). ולאחר מכן, שורה נוספת המכילה את שם הסמל החיצוני, ולאחריו המילה OFFSET ולאחריה הכתובות של מילת-המידע בקובץ המכונה בה נדרש היחסט (OFFSET) (בבסיס עשרוני באربע ספרות, כולל אפסים מוביילים).

בין השורות בשורה יש רווח אחד. **אין חשיבות לסדר השורות**, כי כל שורה עומדת בפני עצמה.

**لتשומת לב:** ניתן ויש מספר כתובות בקובץ המכונה בהן מילות-המידע מתיחסות לאותו סמל חיצוני. לכל כתובות כזו תהיה נפרדת בקובץ ה-*externals*.

**נדגמים את הפלט שמייצר האסמבולר עבור קובץ מקור בשם *ps.as* שהודגש קודם לכן.**

התוכנית לאחר שלב פרישת המקור תיראה כך:

```
; file ps.am
.entry LIST
.extern W

MAIN:      add    r3, LIST
LOOP:       prn   #48
            lea    STR, r6
            inc    r6
            mov    r3, W
            sub   r1, r4
            bne   END
            cmp   val1, #-6
            bne   END[r15]
            dec    K

.entry MAIN
            sub   LOOP[r10], r14
END:        stop
STR:        .string "abcd"
LIST:       .data  6, -9
            .data  -100
.entry K
K:          .data  31
.extern val1
```

להלן הקידוד הבינארי המלא (תמונת הזיכרון) של קובץ המקור, בגמר המעבר השני.

Address (decimal)	Source Code	Machine Code (binary)																
		19	"	"	"	"	"	"	9	8	7	6	5	4	3	2	1	0
0100	MAIN: add r3, LIST	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0101		0	1	0	0	1	0	1	0	0	0	1	1	1	0	0	0	1
0102		0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0
0103		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0104	LOOP: prn #48	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0105		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0106		0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0107	lea STR, r6	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0108		0	1	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1
0109		0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0110		0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
0111	inc r6	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0112		0	1	0	0	1	1	0	0	0	0	0	0	0	1	1	0	1
0113	mov r3, W	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0114		0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1
0115		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0116		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0117	sub r1, r4	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0118		0	1	0	0	1	0	1	1	0	0	0	1	1	1	0	1	1
0119	bne END	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0120		0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1
0121		0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0122		0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
0123	cmp vall, #-6	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0124		0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0125		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0126		0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0127		0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	1	0
0128	bne END[r15]	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0129		0	1	0	0	1	0	1	1	0	0	0	0	0	1	1	1	1
0130		0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0131		0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0
0132	dec K	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0133		0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1
0134		0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0
0135		0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0136	sub LOOP[r10],r14	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0137		0	1	0	0	1	0	1	1	1	0	1	0	1	1	1	0	1
0138		0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0139		0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0140	END: stop	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0141	STR: .string "abcd"	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
0142		0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1
0143		0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1
0144		0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1
0145		0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0146	LIST: .data 6, -9	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
0147		0	1	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1
0148	.data -100	0	1	0	0	1	1	1	1	1	1	1	1	0	0	1	1	1
0149	K: .data 31	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

טבלת הסמלים בסיום המעבר השני היא :

Symbol	Value (decimal)	Base address	offset	Attributes
W	0	0	0	external
MAIN	100	96	4	code, entry
LOOP	104	96	8	code
END	140	128	12	code
STR	141	128	13	data
LIST	146	144	2	data, entry
K	149	144	5	data, entry
val1	0	0	0	external

להלן תוכן קבצי הפלט של הדוגמה.

:ps.ob הקובץ

41	9
----	---

```

0100 A4-B0-C0-D0-E4
0101 A4-Ba-C3-Dc-E1
0102 A2-B0-C0-D9-E0
0103 A2-B0-C0-D0-E2
0104 A4-B2-C0-D0-E0
0105 A4-B0-C0-D0-E0
0106 A4-B0-C0-D3-E0
0107 A4-B0-C0-D1-E0
0108 A4-B0-C0-D5-Eb
0109 A2-B0-C0-D8-E0
0110 A2-B0-C0-D0-Ed
0111 A4-B0-C0-D2-E0
0112 A4-Bc-C0-D1-Eb
0113 A4-B0-C0-D0-E1
0114 A4-B0-C3-Dc-E1
0115 A1-B0-C0-D0-E0
0116 A1-B0-C0-D0-E0
0117 A4-B0-C0-D0-E4
0118 A4-Bb-C1-Dd-E3
0119 A4-B0-C2-D0-E0
0120 A4-Bb-C0-D0-E1
0121 A2-B0-C0-D8-E0
0122 A2-B0-C0-D0-Ec
0123 A4-B0-C0-D0-E2
0124 A4-B0-C0-D4-E0
0125 A1-B0-C0-D0-E0
0126 A1-B0-C0-D0-E0
0127 A4-Bf-Cf-Df-Ea
0128 A4-B0-C2-D0-E0
0129 A4-Bb-C0-D3-Ee

```

0130 A2-B0-C0-D8-E0  
0131 A2-B0-C0-D0-Ec  
0132 A4-B0-C0-D2-E0  
0133 A4-Bd-C0-D0-E1  
0134 A2-B0-C0-D9-E0  
0135 A2-B0-C0-D0-E5  
0136 A4-B0-C0-D0-E4  
0137 A4-Bb-Ca-Db-Eb  
0138 A2-B0-C0-D6-E0  
0139 A2-B0-C0-D0-E8  
0140 A4-B8-C0-D0-E0  
0141 A4-B0-C0-D6-E1  
0142 A4-B0-C0-D6-E2  
0143 A4-B0-C0-D6-E3  
0144 A4-B0-C0-D6-E4  
0145 A4-B0-C0-D0-E0  
0146 A4-B0-C0-D0-E6  
0147 A4-Bf-Cf-Df-E7  
0148 A4-Bf-Cf-D9-Ec  
0149 A4-B0-C0-D1-Ef

:ps.ent הקובץ

MAIN, 96, 4  
LIST, 144, 2  
K, 144, 5

:ps.ext הקובץ

W BASE 115  
W OFFSET 116  
  
vall BASE 125  
vall OFFSET 126

## סיכום והנחיות כלליות

- גודל תוכנית המקור הניתנת כקלט לאסטמבלר אינו ידוע מראש, וכך גם גודלו של קוד המכמה אינו צפוי מראש. אולם כדי להקל בימוש האסטמבלר, מותר להניחס גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכות לאחסון תומנות קוד המכמה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים וטבלת המקורי), יש למשם באופן עילית וחסכוני (למשל באמצעות רשימה מקושרת והקצתה זיכרון דינמי).
- השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הפלט, למעט הסיומות. למשל, אם קובץ הפלט הוא asprog או קבצי הפלט שייצרו הם : prog.ob, prog.ext, prog.ent.
- מתכונת הפעלת האסטמבלר צריכה להיות כפי הנדרש בממ"ן, ללא שינוים כלשהם. ככלומר, משחק המשתמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמורות קבצי המקור יועברו לתוכנית האסטמבלר כארגומנטים (אחד או יותר) בשורת הפקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גרפיים למיניהם, וכו'.
- יש להזכיר לחלק את שימוש האסטמבלר במספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסווגים שונים במודול יחיד. מומלץ לחלק למודולים כגון : מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחרيري של שורה), טבלת הסמלים, מפת הזיכרון, טבלאות קבועות (קובדי הפעלה, שיטות המיעון החוקיות לכל פעולה, וכו').
- יש להזכיר ולתעד את השימוש באופן מלא וברור, באמצעות העורות מפורטות בקוד.
- יש לאפשר תווים לבנים עודפים בקובץ הפלט בשפת אסטמבלר. למשל, אם בשורת הוראה יש שני אופרנדים המופרדים בפסיק, אז לפניהם ואחרי הפסיק מותר שייחיו רווחים וטאבים בכל מקום. בדומה, גם לפניהם ואחריהם שם הפעולה. מותירות גם שורות ריקות. האסטמבלר יתעלם מהתווים לבנים מיוחדים (כלומר ידלג עליהם).
- הקלט (קוד האסטמבלר) עלול להכיל שגיאות תחביריות. על האסטמבלר לגלות ולזוזוח על כל השורות השגויות בקלט. אין לעצור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להציג לפחות הודעות מפורטות מכל הנitin, כדי שאפשר יהיה להבין מה והיכן כל שגיאה. כמובן שגם קובלט מכיל שגיאות, אין טעם להפיק עבورو את קבצי הפלט (ob, ext, ent).

תס ונסלים פרק ההסבירים והגדرات הפרויקט.

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

lezicircums, באפשרותו של כל סטודנט לנפות לכל מנהה, לאו דווקא למנהל 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 לכם.

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

**ב ה צ ל ח ה !**