

# מטלת מנהה (ממ"ו) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלת : פרויקט גמר

משקל המטלת : 61 נקודות (חויבת)

מספר השאלות : 1

מועד אחרון להגשה : 17.03.2024

סמיטר : א' 2024

## קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
  - שליחת מטלות באמצעות דואר אלקטרוני - **באישור המנהה בלבד**
- הסביר מפורט ב"גղל 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 אSEMBLER, עבור שפת אSEMBLER שתוגדר בהמשך. הפרויקט יכתב בשפת C.

עליכם להגיש את הפריטים הבאים :

1. קבצי המקור של התוכנית שכתבתם (קובציים בעלי סיומת .c או .h).
  2. קובץ הרצה (מקומפל ומקישר) עבור מערכת אוביונטו.
  3. קובץ makefile. הקימפול חייב להיות עם הקומפיילר gcc והדגלים : -Wall -ansi -pedantic . יש לנפות את כל ההודעות שモוציא הקומפיילר, כך שהתוכנית תתקמפל ללא כל העורות או זהירות.
  4. דוגמאות הרצה (קלט ופלט) :
- א.** קובצי קלט בשפת אSEMBLER, וקובצי פלט שנוצרו מהפעלת האSEMBLER על קבצי קלט אלה. יש להציג שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האSEMBLER.
- ב.** קובצי קלט בשפת אSEMBLER המציגים מגוון רחב של סוגי שגיאות אSEMBLER (ולכן לא נוצרים קבצי פלט), ותדפסי המסתן המראים את ה הודעות השגיאה שמוציא האSEMBLER.

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

זכיר מספר היבטים חשובים של כתיבת קוד טוב :

1. הפיטה של מבני הנתונים : רצוי (כל האפשר) להפריד בין הגישה למבנה הנתונים לביו המימוש של מבני הנתונים. כך, למשל, בעת כתיבת פונקציות לטיפול בטבלה, אין זה מעניינים של המשתנים בפונקציות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקושרת.
2. קריאות הקוד : יש להשתמש במסות שימושיות למשתנים ופונקציות. יש לעורך את הקוד באופן מסודר : הזחות עקביות, שורות ריקות להפרדה בין קטעי קוד, ועוד'.
3. תיעוד : יש להכניס בקבצי המקור תיעוד תמציתי וברור, שיסביר את תפקידה של כל פונקציה (באמצעות העורות כותרת לכל פונקציה). כמו כן יש להסביר את תפקדים של משתנים חשובים. כמו כן, יש להכניס העורות ברמת פירוט גבוהה בכל הקוד.

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

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

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

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

### **רקע כללי ומטרת הפרויקט**

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

למעשה, זיכרו המחשב כולל הוא אוסף של סיביות, שנוהגים לראותן כמקובצות ליחידות בעלות אורך קבוע (בתים, מיללים). לא ניתן להבחין, עיין שאינה מיומנת, בהבדל פיסי כלשהו בין אותו חלק בזיכרון שבו נמצאת תוכנית לבין שאר הזיכרו.

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מגוון פעולות פשוטות, הנקראות **הוראות מכונה**, ולשם כך היא משתמשת ברגיסטרים (registers) הקיימים בתחום היע"מ, ובזיכרון המחשב. **דוגמאות**: העברת מספר מתא בזיכרון לרגיסטר ביע"מ או בחזרה, הוספה 1 למספר הנמצא ברגיסטר, בדיקה האם המאוחסן ברגיסטר שווה לאפס, חיבור וחיסור בין שני רגיסטרים, ועוד.

הוראות המכונה ושילובים שלהן הן המרכיבות תוכנית כפי שהיא טעונה לזכרו בזמן ריצתה. כל תוכנית מקור (התוכנית כפי שנכתבה בידי המוכנה), תורגמת בסופו של דבר באמצעות תוכנה מיוחדת לצורה סופית זו.

היע"מ יודיע לבצע קוד שנמצא בפורמט של **שפה מכונה**. זהו רצף של ביטים, המהוויםקידוד ביןארי של סדרת הוראות המכונה המרכיבות את התוכנית. קוד זה אינו קרייא למשתמש, ולכן לא נוח לקובד (או לזרוא) תוכניות ישירות בשפת מכונה. **שפה אסמבלי** (assembly language) היא שפת תוכנות מאפשרת לייצג את הוראות המכונה בצורה סימבולית קליה ונוחה יותר לשימוש. כموון 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 תוכנות עילית יש מהדר (compiler), או מפרש (interpreter), המתרגם תוכניות מקור לשפת מכונה. האסmbלר משמש בתפקיד דומה עבור שפת אסמבלי.

לכל מודל של יע"מ (כלומר לכל אירוגון של מחשב) יש שפת מכונה יייעודית משלו, ובהתאם גם שפת אסמבלי יייעודית משלו. לפיכך, גם האסmbלר (כלי התרגום) הוא יייעודי ושונה לכל יע"מ.

תפקידו של האסmbלר הוא לבנות קובץ המכיל קוד מכונה, מקובץ נתון של תוכנית הכתובת בשפת אסmbלי. זהו השלב הראשון במסלול אותו עברת התוכנית, עד לקבלת קוד המוכן לריצה על חומרת המחשב. **השלבים הבאים הם קישור (linkage) וטעינה (loading)**, אך בהם לא נעסק במאין זה.

המשמעותה בפרויקט זה היא לכתוב אסmbלר (כלומר תוכנית המתרגמת לשפת מכונה), עבר שפת אסmbלי שנדרש כאן במיוחד לצורכי הפרויקט.

**לעניינות לב** : בהסבירים הכלליים על אופן עבודה תוכנית האסmbלר, 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 האסמבילר. אין לטעות: עליהם לכתוב את תוכנית האסמבילר בלבד. אין לכתוב את תוכניות הקישור והטעינה!!!

### המחשב הדמיוני וסתת האסמבלי

נגיד לך את סתת האסמבלי ואת מודל המחשב הדמיוני, עבור פרויקט זה.  
הערה: תיאור מודל המחשב להלן הוא חלקו בלבד, ככל שנחוץ לביצוע המשימות בפרויקט.

"חומרה":

המחשב בפרויקט מורכב ממעבד CPU (יע"מ - יחידת עיבוד מרכזית), אוגרים (רגיסטרים) ו זיכרון RAM. חלק מהזיכרון משמש גם כמחסנית (stack).

למעבד 8 רגיסטרים כללים, בשמות: r7, r6, r5, r4, r3, r2, r1, r0. גודלו של רגיסטר הוא 14 סיביות. הסיבית ה-1 הינה פחות משמעותית מאשר סיבית מס' 0, והסיבית המשמעותית ביותר במס' 13. שמות הרגיסטרים כתבים תמיד עם אות 'z' קטנה.

כמו כן יש במעבד רגיסטר 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 4096 תאים, בכתביות 0-4095 (בסיס עשרוני), וכל תא הוא בגודל של 14 סיביות. לתא בזיכרון נקרא גם בשם "מילה". הערך בכל מילה מסוימת כמו ברגיסטר.

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

### מבנה הוראות מכונה:

כל הוראה מכונה מקודדת במספר מילוט זיכרו רצופות, החל ממילה אחת ועד למקסימום חמישה מילים, בהתאם לשיטות המיעוון בהן נעשה שימוש (ראו בהמשך).

בקובץ הפלט המכיל את קוד המכונה שבודה האסמבילר, כל מילה תקודד בסיס 4 ("מוצפן") המוגדר כדלקמן (ואו הסברים ודוגמאות בהמשך):

בסיס 4 רגיל	0	1	2	3
בסיס 4 מוצפן	*	#	%	!

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

13 12 11 10	9 8 7 6	5 4	3 2	1 0
לא בשימוש	opcode	מייעון אופרנד מקור	מייעון אופרנד יעד	A,R,E

### **סיביות 1-0 (השדה 'E,R,A'):**

במילה הראשונה של הוראה, סיביות אלה תמיד מאופסות (00).

**השדה 'E,R,A' מתווסף לכל אחת מהamilims בקידוד ההוראה** (ראו פירוט שיטות המיעון). שדה זה מצין מהו סוג הקידוד של המילה: קידוד מוחלט (Absolute) , חיצוני (External) או נייח (Relocatable).

הערך 00 משמעו שקידוד המילה הוא מוחלט (ואינו צריך שינוי בשלבי קישור ובעינה). הערך 01 משמעו שהקידוד הוא של כתובת חיצונית (ומצריך שינוי בשלבי קישור ובעינה). הערך 10 משמעו שהקידוד הוא של כתובת פנימית הניתנת להזזה (ומצריך שינוי בקשר ובעינה).

ראו הסבר נוסף על סיביות אלה בהמשך.

**סיביות 3-2:** מקודדות את מספרה של שיטת המיעון של אופרנד היעד (destination operand) . אם אין בהוראה אופרנד יעד, ערכן של סיביות אלה הוא 0.

**סיביות 5-4:** מקודדות את מספרה של שיטת המיעון של אופרנד המקור (source operand) . אם אין בהוראה אופרנד מקור, ערכן של סיביות אלה הוא 0.

**סיביות 9-6:** במילה הראשונה של ההוראה סיביות אלה מהוות את **קוד-הפעולה** (opcode). כל opcode מיוצג בשפת אסמלילי באופן סימבולי על ידי **שם-פעולה**.

בשפה שלנו יש 16 קודים פעולה והם :

שם הפעולה	קוד הפעולה (בסיס עשרוני)
mov	0
cmp	1
add	2
sub	3
not	4
clr	5
lea	6
inc	7
dec	8
jmp	9
bne	10
red	11
prn	12
jsr	13
rts	14
hlt	15

שמות הפעולות נכתבים תמיד באותיות קטנות. פרוטו המשמעות של הפעולות יבוא בהמשך.

**סיביות 13-10:** אין בשימוש וערך ה-0.

שיטות מיעון:

בשפה שלנו קיימות ארבע שיטות מיעון, מסומנות במספרים 0,1,2,3. השימוש בשיטות מיעון מצריך קידוד של מילות-מידע נוספים בקוד המכונה של ההוראה. אם בפקודה יש אופרנד אחד, תהייה מילת מידע אחת נוספת. אם בפקודה יש שני אופרנדים, יתכנו שתי מילות-מידע נוספות אחת המשותפת לשני האופרנדים, תלוי בשיטות

המייעון בהו נעשה שימוש (ראו מפרט בהמשך). כאשר בקידוד הפקודה יש שתי מילוט-מידע נוספת, אזי מילת-המידע הראשונה מתיחסת לאופרנד המקור, והשנייה מתיחסת לאופרנד היעד.

בכל מילת-מידע נוספת, סיביות 1-0 הן השדה .A,R,E.

להלן תיאור שיטות המייעון במכונה שלנו:

ערך	שיטת מייעון	תוכן המילה נוספת	אופן הכתיבה	דוגמה
0	מייעון מיידי	מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר שלם בסיס עשרוני. המשלים ל-2, המוצג ברוחב של 12 סיביות, אליו מתווספות זוג סיביות של השדה .A,R,E (הערך של שדה זה הוא תמיד 00 עבור מייעון מיידי).	האופרנד מתחילה בתו # ולאחריו ובצמוד אליו מופיע מספרשלם בסיס עשרוני. יש גם אפשרות שבמקומות מסווגדר בתכנית על ידי define (ראו פרטים בהמשך).	mov #1,r2  בוגמה זו האופרנד הראשון של הפקודה נתון בשיטת מייעון מיידי. ההוראה כתובת את הערך 1- אל אוגר r2  דוגמה נוספת: נתונה הגדרת הקבוע:.define size = 8  אזי ההוראה: mov #size, r1 האופרנד הראשון הוא מיידי, כאשר המספר 8 מוצג באמצעות השם הקבוע size. ההוראה כתובת את הערך 8 אל אוגר r1
1	מייעון ישיר	מילת-מידע נוספת של ההוראה מכילה כתובת של מילה בזיכרון. מילה זו בזיכרון היא האופרנד. הכתובת מוצגת כמספר ללא סימן ברוחב של 12 סיביות, אליו מתווספות זוג סיביות של השדה .A,R,E (הערך של שדה זה הוא או 01 או 10, תלוי בסוג הכתובת - חיונית או פנימית).	האופרנד הוא <u>תוויות</u> שכבר הוצאה או שתוצחר בהמשך הקובץ. ההצעה נעשית על ידי כתיבת תווית בתחילת הנקה 'data'.או 'string', או בתחלת הוראה של הנקה, או באמצעות אופרנד של הנקה '.extern'	נתונה הגדרה: x: .data 23  אזי ההוראה: dec x מקטינה ב-1 את תוכן המילה שבכתבות x בזיכרון ("משתנה" x).

ערך	שיטת מיון	תוכן המילה נוספת	אופן הכתיבה	דוגמה
2	מיון אינדקס קבוע	האופrnd מרכיב מתוויות המציגת את כתובות התחלת המערך, ולאחריה בסוגרים מרובעים האינדקס במערך אליו רוצים לפנות.	נתונה ההגדרה : x: .data 23,25,19,30  איי ההוראה : mov x[2],r2 תעתיק את המספר 19 הנמצא באינדקס 2 במערך x אל הרגיסטר r2.  דוגמה נוספת : נתונה הגדרת המערך x לעיל, וכן הגדרת הקבוע : .define k=1  איי ההוראה : mov r2,x[k] תעתיק את תוכן הרגיסטר r2 אל המילה באינדקס 1 במערך x (נדפס התוכן הקודם .(25	בשיטת מיון זו משמשת לגישה לאייר במערך לפי אינדקס. המערך נמצא בזיכרונו. כל אייר במערך הוא בגודל מילה.  בשיטת מיון זו קיימות בקידוד ההוראה שתி מילות-מידע נוספות. המילה הנוספת הראשונה מכילה את כתובות התחלת המערך. המילה הנוספת השנייה מכילה את האינדקס של האיבר במערך אליו יש לגשת.  הערכים בשתי מילות-המידע הנוספות מיוצגים ברוחב 12 סיביות, אליהן מתווספות זוג סיביות של השדה A,R,E (הערך של שדה זה במילת-המידע הראשונה הוא כמו במיון ישר, ובמילת-המידע השנייה כמו במיון מייד).
3	מיון רגיסטר ישיר	האופrnd הוא שם של רגיסטר.	mov r1,r2  בדוגמה זו, ההוראה מעתקה את תוכן הרגיסטר r1 אל רגיסטר r2.  בדוגמה זו, שני האופrndים הם בשיטת מיון רגיסטר ישר, ולכן יחלקו מילת-מידע נוספת אחת משותפת.	הרגיסטר משמש כאופrnd יעד, מילת-מידע נוספת של הפוקודה תקודד בסיביות 2-4 את מספרו של הרגיסטר. ואילו אם הרגיסטר משמש כאופrnd מקור, מספר הרגיסטר יקודד בסיביות 7-5 של מילת-המידע.  אם בפוקודה יש שני אופrndים ושניהם בשיטת מיון רגיסטר ישר, הם יחלקו מילת-מידע אחד משותפת, כאשר הסיביות 2-4 הן עברו רגיסטר היעד, והסיביות 7-5 הן עברו רגיסטר המקור.  לAMILT-HMIDU MATOVSFOT ZOG SIVIYOT SHL SHEDA A,R,E (הערך של שדה זה הוא תמיד 00 עברו מיון רגיסטר ישר).  SIVIYOT ACHROT BAMILT-HMIDU SHAINEN B SHIMUSH YICHLU 0.

הערה : מותר להתייחס לתווית עוד לפני שמחבירים עליה, בתנאי שהיא אכן מוצחרת במקום כלשהו בקובץ.

#### מפורט הוראות המכונה :

בתיאור הוראות המכונה נשתמש במונח PC (קיצור של "Program Counter".

זהו רגיסטר פנימי של המעבד (לא רגייסטר כללי), שמכיל בכל רגע נתון את כתובות הזיכרונו בה. **نمצת ההוראה הנוכחית שמתבצעת** (הכוונה תמיד לכתובות המילה הראשונה של ההוראה).

הוראות המכונה מתחולקות לשלוש קבוצות, לפי מספר האופרנדים הנדרשים לפעולה.

**קבוצת ההוראות הראשונה:**  
אלו הן ההוראות המקבלות שני אופרנדים.

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

הוראה	הסבר הפעולה	דוגמה	הסבר הדוגמה
mov	מבצעת העתקה של האופרנד הראשון, אופרנד המקור (source) אל האופרנד השני, אופרנד היעד (destination) (בהתאם לשיטת המיעון).	mov A, r1	העתק את תוכן המשתנה A (המילה שבכנתה בזיכרונו) אל רגיסטר r1.
cmp	מבצעת "השוואה" בין שני האופרנדים שלה. אופן ההשוואה: תוכן אופרנד היעד (השני) מופחת מתוכן אופרנד המקור (הראשון), ללא שימרת תוצאה החישור. פעולת החישור מעדכנת דגל בשם Z ("דגל האפס") ברגיסטר הסטטוס (PSW).	cmp A, r1	אם תוכן המשתנה A זהה לתוכנו של רגיסטר r1 אז דגל האפס, Z, ברגיסטר הסטטוס (PSW) יודלק, אחרת הדגל יאפס.
add	אופרנד היעד (השני) מקבל את תוצאה החיבור של תוכן המשתנה A ותוכנו הנוכחי של r0.	add A, r0	רגיסטר 0 יקבל את תוכנת החיבור של ערך 3 מתוכנו הנוכחי של הרגיסטר r1.
sub	אופרנד היעד (השני) מקבל את תוצאה החישור של אופרנד המקור (הראשון) מօפרנד היעד (השני).	sub #3, r1	רגיסטר r1 מקבל את המינוס של ערך 3 מתוכנו הנוכחי של הרגיסטר r1.
lea	lea הוא קיצור (ראשי תיבות) של load effective address מציבה את המعنין בזיכרונו המיצג על ידי התווית שבօפרנד הראשון (המקור), אל אופרנד היעד (השני).	lea HELLO, r1	המען שמייצגת התווית HELLO מוצב לרגיסטר r1.

**קבוצת ההוראות השנייה:**  
אלו הן ההוראות הדורשות אופרנד אחד בלבד. אופן הקידוד של האופרנד הוא כמו של אופרנד היעד בפקודה עם שני אופרנדים. במקרה זה, השדה של אופרנד המקור (סיביות 5-4) במילה הראשונה בקידוד ההוראה הינו חסר משמעות, ולפיכך יכיל 00.

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

הוראה	הסבר הפעולה	דוגמה	הסבר הדוגמה
not	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 וליהיפך : 1 ל-0).	not r2	r2 ← not r2
clr	איפוס תוכן האופרנד.	clr r2	r2 ← 0
inc	הגדלת תוכן האופרנד באחד.	inc r2	r2 ← r2 + 1
dec	הקטנת תוכן האופרנד באחד.	dec C	C ← C - 1

הסבר הדוגמה	דוגמה	הסבר הפעולה	הוראה
PC $\leftarrow$ LINE מצביע התוכנית מקבל את המعن המויצג על ידי התווית LINE, ולפיכך הפוקודה הבאה שתתבצע תהיה בمعنى זה.	jmp LINE	קפיצה (הסתעפות) בלתי מותנית אל ההוראה שנמצאת בمعنى המויצג על ידי האופרנד. כלומר, בעת ביצוע ההוראה, מצביע התוכנית (PC) יקבל את ערך אופרנד היעד.	jmp
אם ערך הדגל Z ברגיסטר הסטטוס (PSW) הינו 0, אז: PC $\leftarrow$ LINE	bne LINE	bne הוא קיצור (ראשי תיבות) של .branch if not equal (to zero) זהה הוראות הסתעפות מותנית. מצביע התוכנית (PC) יקבל את ערך אופרנד היעד אם ערכו של הדגל Z ברגיסטר הסטטוס (PSW) הינו 0. כזכור, הדגל Z נקבע בפקודת .cmp.	bne
קוד ה-ascii של התו הנקרא מהקלט ייכנס לרגיסטר r1.	red r1	קריאה שלתו מהקלט הסטנדרטי (stdin) אל האופרנד.	red
התו אשר קוד ה-ascii שלו נמצא ברגיסטר r1 יודפס לפלט הסטנדרטי.	prn r1	הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn
push(PC) PC $\leftarrow$ FUNC	jsr FUNC	קריאה לשגרה (סברוטינה), מצביע התוכנית (PC) הנוכחי נדחף לתוך המכחסנית שבזיכרון המחשב, ואופרנד מוכנס ל-PC.	jsr

**קובוצת ההוראות השלישייה:** קידוד ההוראה מורכב ממילה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד במילה הראשונה של ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השייכות לקבוצה זו הן : .rts, hlt

הסבר הדוגמה	דוגמה	הסבר הפעולה	הוראה
PC $\leftarrow$ pop()	rts	זרה משגרה. הערך שנמצא בראש המכחסנית של המחשב מזוכה מן המכחסנית, ומוכנס אל מצביע התוכנית (PC).	rts
התוכנית עצרת	hlt	עצירת ריצת התוכנית.	hlt

מבנה שפת האסמלבי :

תכנית בשפת אסמלבי בנוייה **מماקרואים וממשפטים** (statements).

ماקרואים :

ماקרואים הם קטעי קוד הכלולים בתוכם משפטיים. בתוכנית ניתן להגדיר ماקרו ולהשתמש בו במקומות שונים בתוכנית. השימוש בmacro ממקום מסוים בתוכנית יגרום לפרישת המacro לאותו מקום.

הגדרותmacro נעשית באופן הבא : (בדוגמה שם המacro הוא m\_mcr)

```
mcr m_mcr  
inc r2  
mov A,r1  
endmcr
```

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

```
.  
. .  
m_mcr  
. .  
m_mcr  
. .
```

בדוגמה זו, השתמשנו פעמיים במאקרו `m_mcr`, התוכנית לאחר פרישת המאקרו תיראה כך:

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

### התוכנית לאחר פרישת המאקרו היא התוכנית שהאסטמבלר אמר לתרגם.

#### הנחות והניחות לגבי מאקרו:

- אין במערכת הגדרות מאקרו מקוונות (אין צורך לבדוק זאת).
- שם של הוראה או הנחה לא יכול להיות שם של מאקרו.
- ניתן להניח שלכל שורה מאקרו בקוד המקור קיימת סגירה עם שורת `endmcr` (אין צורך לבדוק זאת).
- הגדרת מאקרו תהיה תמיד לפני הקריאה למאקרו (אין צורך לבדוק זאת).
- נדרש שהקדם-אסטמבלר ייצור קובץ עם הקוד המורחב הכלול פרישה של המאקרו (הרחבת של קובץ המקור המקורי בהמשך). "קובץ המקור המורחב" הוא "קובץ מקור" לאחר פרישת המאקרו, לעומת "קובץ מקור ראשון" שהוא קובץ הקלט למערכת, כולל הגדרת המאקרוים.

#### משפטים:

קובץ מקור בשפת אסטמבלי מורכב משורות המכילות משפטיים של השפה, כאשר כל המשפט מופיע בשורה נפרדת. כלומר, ההפרדה בין משפט בקובץ המקור הינה באמצעות התו '`\n`' (שורה חדשה).

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

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

סוג המשפט	הסבר כללי
משפט ריק	זהוי שורה המכילה א' ורק תווים לבנים (whitespace), ככלומר רק את התווים ' ' ו- 't' (רווחים וטאים). ייתכן ובשורה אין אףתו (למעט התו ח), ככלומר השורה ריקה.
משפט הערה	זהוי שורה בה התו הראשון הינו ';' (נקודה פסיק). על האסמבול להעתלם לחלווטין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמבול מה עליו לעשות כשהוא פועל על תכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי מיציר להקצת זיכרון ואתחול משתנים של התכנית, אך הוא אינו מייצר קידוד של הוראות מכונה המיועדות לביצוע בעת ריצת התכנית.
משפט הוראה	זהו משפט המייצר קידוד של הוראות מכונה לביצוע בעת ריצת התכנית. המשפט מורכב ממשם של הוראה שעל המעבד לבצע, ותיאור האופרנדים של ההוראה.
משפט הגדרת קבוע	זהו משפט באמצעותו ניתן להגדיר שם סימבולי המייצג קבוע מספרי. במהלך קידוד התוכנית, בכל מקום בו מופיע השם, הוא יוחלף בקבוע המספר. <b>משפט זה לשעצמו אינו מייצר קוד ואינו מקצה זיכרון.</b>

cut נפרט יותר לגבי סוגי המשפטים השונים.

#### משפט הנחיה:

משפט הנחיה הוא בעל המבנה הבא :

בתחלת המשפט יכולה להופיע הגדרה של תווית (label). לתווית יש תחבר חוקי, שיתואר בהמשך. התווית היא אופציונלית.

לאחר מכן מופיע שם הנחיה. לאחר שם הנחיה יופיעו פרמטרים (מספר הפרמטרים בהתאם לנחיה). שם של הנחיה מתחילה בתו ',' (נקודה) ולאחריו תווים באותיות קטנות (lower case) בלבד.

**יש לשים לב: למיללים בקוד המכונה הנוצרות משפט הנחיה לא מצורפות זוג סיביות E,R,A, והקידוד מלא את כל 14 הסיביות של המילה.**

יש ארבעה סוגי משפטי הנחיה, והם :

1. הנחיה 'data'.

הפרמטרים של הנחיה '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 וטאים בכל כמות (או בכלל לא), אולם הפסיק חייב להופיע בין המספרים. כמו כן, אסור שיוופיע יותר מפסיק אחד בין שני מספרים, וגם לא פסיק אחרி המספר האחרון או לפני המספר הראשון.

המשפט 'data', מנהה את האסמבול להקצות מקומות בתמונות הנתונים (data image), אשר בו יוחסנו הערכים של הפרמטרים, ולקדם את מונח הנתונים, בהתאם למספר הערכים. אם בהנחתת.data. מוגדרת תווית, אז תווית זו מקבלת את ערך מונח הנתונים (לפניה הקידום), ומוכנסת אל טבלת הסמלים. דבר זה מאפשר להתיחס אל מקום מסוים בתמונות הנתונים דרך שם התווית (למעשה, זהה ידך להגדיר שם של משתנה).

כלומר אם נכתב :

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

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

אם נכתב בתכנית את הוראה :

mov XYZ, r1

אזי בזמן ריצת התכנית יוכנס לרגיסטר r1 ערך 7.

ואילו הוראה :

lea XYZ, r1

תוכניס לרגיסטר r1 את ערך התווית XYZ (כלומר הכתובת זיכרנו בה מאוחסן הערך 7).

2. ההנחיה 'string'

להנחיה 'string' פרמטר אחד, שהוא מחוזות חוקית. תווים המחרוזת מקודדים לפי ערכי ascii המתאים, ומוכנסים אל תמונה הנתונים לפי סדרם, כל تو במילה נפרדת. בסוף המחרוזת יתוסיף התו '0' (הערך המספרי 0), המסמן את סוף המחרוזת. מונה הנתונים של האסמלר 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 למזה שנעשה עבור '.data' (כלומר ערך התווית יהיה הכתובת זיכרנו שבה מתחילה המחרוזת).

לדוגמה, ההנחיה :

STR: .string "abcdef"

מקצת בתמונה הנתונים רצף של 7 מילימ, ומאחריהם ערך 0 לסימון סוף מחוזות. התווית STR מזוהה עם כתובת הסדר במחוזות, ולאחריהם ערך 0 לסימון סוף מחוזות. התווית STR מזוהה עם כתובת התחלת המחרוזת.

3. ההנחיה 'entry'

להנחיה 'entry'. פרמטר 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 אסמלר הנמצא בקובץ מקור אחרים להשתמש בה (אופrnd של הוראה).

לדוגמה, השורות :

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

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

לתשומת לב : תווית המוגדרת בתחילת שורת entry. אינה חסרת משמעות והאסמלר מתעלם מתווית זו (אפשר שהאסמלר יוציא הודעה זהה).

4. הנקיה 'extern'.

להנchia 'extern'. פרט והוא שם של תווית שאינה מוגדרת בקובץ המקור הנוכחי. מטרת ההוראה היא להודיע לאסמלר כי התווית מוגדרת בקובץ אחר, וכי קוד האסמלר בקובץ הנוכחי עושה בתווית שימוש.

שים לב כי הנchia זו תואמת להנchia 'entry'. המיפוי בקובץ בו מוגדרת התווית. בשלב הקישור התבצע התאמה בין ערך התווית, כפי שנקבע בקובץ המקורי את התווית, לבין קידוד ההוראות המשמשות בתווית בקבצים אחרים (שלב הקישור אינו רלוונטי לም"ז זה).

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

```
.extern HELLO
```

הערה: לא ניתן להגדיר באותו קובץ את אותה התווית גם כ-entry וגם כ-extern (בדוגמאות לעיל, התווית HELLO).

**لتשומת לב:** תווית המוגדרת בתחילת שורת extern. הינה חסרת משמעות והאסמלר מטעלים מתווית זו (אפשר שהאסמלר יוציא הודעה זהה).

#### משפט הוראה:

משפט הוראה מורכב מחלקים הבאים:

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

אם מוגדרת תווית בהוראה, אז היא תוכנס אל טבלת הסמלים. ערך התווית יהיה מען המילה הראשונה של הוראה בתוך תומנת הקוד שבונה האסמלר.

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 ט-abs (אחד או יותר).

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזה בטו' ' (פסיק). בדומה להנchia 'data', לא **חייבת להיות הצמדה של האופרנדים לפסיק**. כל כמות של רווחים ו/או ט-abs 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: hlt

### משפט הגדרת קבוע:

משפט הגדרת קבוע הוא בעל המבנה הבא :

.define קבוע-מספרי = שם-הקבוע

לדוגמה :

```
.define len = 4  
.define init = -3
```

הרעינו הוא ליצג קבוע מספרי באמצעות שם סימבולי. בכל מקום בתוכנית בו מופיע שם של קבוע, האסמלר יחליף בקידוד הפקודה לקוד מכונה את השם בקבוע המספרי אליו הוגדר.

המילה השמורה 'define', נוטה באוטיות קטנות בלבד. התחריר של שם הקבוע זהה לתחריר של תווית. אסור להגדיר את אותו שם קבוע יותר מפעם אחת. כמו כן אותו סמל לא יכול לשמש חן כשם של קבוע וחן כתווית באותה תנינית. מילים שמורות של שפת האסמלרי (שם של רגייסטר, שם של הוראת מכונה או שם של הנקודה) אינן יכולות לשמש שם של קבוע. בין שם הקבוע לבין הקבוע המספרי מפheid התו '='. מוגדרים תווים לבנים בשני צידי התו.

הקבוע חייב להיות מוגדר לפני השימוש הראשוני בו.  
אסור להגדיר תווית בשורה שהיא משפט הגדרת קבוע.

ניתן להשתמש בשם הקבוע בכל מקום בתוכנית האסמלרי בו יכול להופיע קבוע מספרי, ככלمر :  
**אינדקס בשיטת מייעון אינדקס ישיר, או ערך בשיטת מייעון מיידי, או אופרנד של הנקודה .data.**

לדוגמה, בהינתן הגדרות קבוע לעיל, אזי ההוראה :  
mov x[len], r3  
תעתיק את האיבר באינדקס 4 במערך x אל אונגר r3.

וההוראה :  
mov #init, r2  
תציב את הערך המיידי 3- אל האונגר r2

כמו כן, הנקודה :  
.data len  
תקacha מילה בזיכרון עם ערך התחלתי 4.

### אפיון השדות במשפטים של שפת האסמלרי

תווות:

תוויות היא סמל שמוגדר בתחילת משפט הוראה , או בתחילת הנקודה .data. או string. תוויות חוקית מתחילה באות אלפביתית (גדולה או קטנה), ולאחריה סדרה כלשהי של אותיות אלפביתיות (גדולות או קטנות) ו/או ספרות. האורך המаксימלי של תוויות הוא 31 תווים.

הגדרה של תווית מסוימת בtau : (נקודותים). tau זה אינו מהווה חלק מהתווית, אלא רק סימן המציין את סוף ההגדירה. התו '=' חייב להיות צמוד לתווית (לא רווחים).

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

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hEllo:

X:

He78902:

**لتשומת לב :** מיללים שמורות של שפט האסמבלי (כלומר שם של פעולה או הנחיה, או שם של הgiстיר) אינן יכולות לשמש גם שם של תווית. כמו כן, אסור שאותו סמל ישמש הן כתווית והן שם של מקאו או של קבוע.

התווית מקבלת את ערכיה בהתאם להקשר בו היא מוגדרת. תווית המוגדרת בהנחיות `data string`, מקבלת ערך מונה הנקנים (data counter) הנוכחי, בעוד שתווית המוגדרת בשורת הוראה מקבלת ערך מונה ההוראות (instruction counter) הנוכחי.

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

מספר:

מספר חוקי מתחילה בסימן אופציונלי: ‘-’, ‘+’, ולאחריו סדרה כלשהי של ספרות בסיס עשרוני. דוגמה: -76, +123 הם מספרים חוקיים. אין תמיכה בשפט האסמבלי שלו ביצוג בסיס אחר מאשר עשרוני, ואין תמיכה במספרים שאינם שלמים.

מחרוות:

מחרוות חוקית היא סדרת תווים ascii נראים (שניתנים להדפסה), המוקפים במרכאות כפولات (המרכאות אינן נחשבות חלק מהמחרוות). דוגמה למחרוות חוקית: “hello world”.

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

בכל מילה בקוד המכונה של הוראה (לא של נתונים), האסמבller מכניס מידע עboro תהליך הקישור והטעינה. זהו השדה A,R,E . המידע ישתמש לתיקונים בקוד בכל פעם שייטען לזכרון לצורך הרצה. האסמבller בונה מלכתחילה קוד שמיועד לטיענה החל מכתובת התחלה. התיקונים יאפשרו לטען את הקוד בכל פעם למקום אחר, בלי צורך לחזור על תהליך האסמבלי.

שתי הסיבות בשדה E,A,R יכilio את אחד הערכים הבינאריים: 00, 10, או 01. המשמעות של כל ערך מפורטת להלן.

האות 'A' (קייזר של absolute) בא להציגו שתוכן המילה אינו תלוי במקום בו זיכרו בו יייטען בפועל קוד המכונה של התכנית בעת ביצועה (למשל מילה המכילה אופrnd מיידי). במקרה זה שתי הסיבות הימניות יכilio את הערך 00.

האות 'R' (קייזר של relocatable) בא להציגו שתוכן המילה תלוי במקומות בו זיכרו בו יייטען בפועל קוד המכונה של התכנית בעת ביצועה (למשל מילה המכילה כתובת של תווית המוגדרת בקובץ המקור). במקרה זה שתי הסיבות הימניות יכilio את הערך 10.

האות 'E' (קייזר של external) בא להציגו שתוכן המילה תלוי בערכו של סמל חייצוני (external). (למשל מילה המכילה כתובת של תווית חייצונית, ככלומר תווית שאינה מוגדרת בקובץ המקור). במקרה זה שתי הסיבות הימניות יכilio את הערך 01.

כאשר האסמבller מקבל כקלט תוכנית בשפט אסמבלי, עליו לטפל תחילתה בפרישת המאקרוואים, ורק לאחר מכן לעבור על התוכנית אליה נפרשו המאקרוואים. ככלומר, פרישת המאקרוואים תעשה בשלב "קדם אסמבלי", לפני שלב האסמבller (המתואר בהמשך). אם התוכנית אינה מכילה מקרו, תוכנית הפרישה תהיה זהה לתוכנית המקור.

דוגמה לשלב קדם אסמבller. האסמבller מקבל את התוכנית הבאה בשפט אסמבלי :

```

.define sz = 2
MAIN:      mov   r3, LIST[sz]
LOOP:       jmp   L1
            mcr  m_mcr
            cmp   r3, #sz
            bne   END
            endmcr
            prn   #-5
            mov   STR[5], STR[2]
            sub   r1, r4
            m_mcr
L1:        inc   K
            bne   LOOP
END:        hlt
.define len = 4
STR:        .string "abcdef"
LIST:       .data  6, -9, len
K:          .data  22

```

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

```

.define sz = 2
MAIN:      mov   r3, LIST[sz]
LOOP:       jmp   L1
            prn   #-5
            mov   STR[5], STR[2]
            sub   r1, r4
            cmp   r3, #sz
            bne   END
L1:        inc   K
            bne   LOOP
END:        hlt
.define len = 4
STR:        .string "abcdef"
LIST:       .data  6, -9, len
K:          .data  22

```

קוד התוכנית, לאחר הפרישה, ישמר בקובץ חדש, כפי שIOSCAR המשיך.

### **אלגוריתם שלדי של קדם האסמבילר**

נזכיר להלן אלגוריתם שלדי לתהליך קדם האסמבילר. لتשומת לב: אין חובה להשתמש דווקא באלגוריתם זה:

- .1. קרא את השורה הבאה מקובץ המקור. אם נגמר הקובץ, עברו ל- 9 (סיום).
- .2. האם השדה הראשון הוא שם מאקרו המופיע בטבלת המאקרו (כגון `m_mcr`)? אם כן, החלף את שם המאקרו והעתק במקומו את כל השורות המתאימות מהטבלה לקובץ, חוזר ל- 1. אחרת, המשך.
- .3. האם השדה הראשון הוא `mcr` (התחלת הגדרת מאקרו)? אם לא, עברו ל- 6.
- .4. הדלק דגל "`יש mcr`".
- .5. (קיימת הגדרת מאקרו) הכנס לטבלת שורות מאקרו את שם המאקרו (לדוגמה `m_mcr`).
- .6. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עברו ל- 9 (סיום).

- אם דגל "יש mcr" דולק ולא זזהה תווית **endmcr** הכנס את השורה לטבלת המאקרו  
ומחק את השורה הניל מהקובץ. אחרת (לא מאקרו) חוזר ל-1.
- .7. האם זזהה תווית **endmcr**? אם כן, מחק את התווית מהקובץ והמשך. אם לא, חוזר ל-6.
  - .8. כבה דגל "יש mcr". חוזר ל-1. (סיום שמירת הגדרת מאקרו).
  - .9. סיום: שמירת קובץ מאקרו פרוש.

### אסמבלר עם שני מעברים

במעבר הראשון של האסմבלר, יש להזיהות את הסמלים (תוויות) המופיעים בתוכנית, ולתת לכל סמל ערך מסווני שהוא המعنן בזיכרונו שהסמל מייצג. במעבר השני, באמצעות ערכי הסמלים, וכן קודי-הפעולה ומספריו הרגיסטרים, בונים את קוד המכונה.

עליו להחליף את שמות הפעולות `mov, jmp, prn, sub, cmp, inc, bne, hlt` בקוד הבינארי השקלול להם במודל המחשב שהגדנו.

כמו כן, על האסմבלר להחליף את הסמלים K,STR, LIST, L1, MAIN, LOOP, END במשמעותם של המיקומות בזיכרון שם נמצאים כל נתון או הוראה בהתאם.

בנוסף, על האסמבלר להחליף את קבועים שהוגדרו על ידי `define` (בדוגמה הקבועים הם `len` ו-`sz`), קבועים המספריים שהם ערכי הקבועים בהתאם, בכל מקום בו הם מופיעים.

נניח שקטע הקוד לעיל (הוראות ונתונים) ייטען בזיכרון החל ממען 100 (בבסיס 10). במקרה זה קיבל את ה"תרגום" הבא:

Decimal Address	Source Code	Explanation	Binary Machine Code
0100	MAIN: mov r3, LIST[sz]	First word of instruction Source register 3 Address of label LIST (integer array) Value of constant sz (index 2)	00000000111000 00000001100000 00001000010010 00000000001000
0104	LOOP: jmp L1		00001001000100
0105		Address of label L1	00000111100010
0106	prn # -5		00001100000000 11111111101100
0108	mov STR[5], STR[2]		00000000101000
0109		Address of label STR (string)	00000111110010
0110		Index 5	000000000010100
0111		Address of label STR	00000111110010
0112		Index 2	00000000001000
0113	sub r1, r4		00000011111100
0114		Source register 1 and target register 4	00000000110000
0115	cmp r3, #sz		000000001110000
0116		Source register 3	000000001100000
0117		Value of constant sz (immediate #2)	00000000001000
0118	bne END		00001010000100
0119		Address of label END	00000111110010
0120	L1: inc K		00000111000100
0121		Address of label K (integer)	00001000011110
0122	bne LOOP		00001010000100
0123		Address of label LOOP	00000110100010
0124	END: hlt		00001111000000
0125	STR: .string "abcdef"	Ascii code 'a'	000000001100001
0126		Ascii code 'b'	000000001100010
0127		Ascii code 'c'	000000001100011
0128		Ascii code 'd'	000000001100100
0129		Ascii code 'e'	000000001100101

Decimal Address	Source Code	Explanation	Binary Machine Code
0130		Ascii code 'f'	00000001100110
0131		Ascii code '\0' (end of string)	00000000000000
0132	LIST: .data 6, -9, len	Integer 6 (first in array of 3 words)	00000000000010
0133		Integer -9	1111111110111
0134		Value of constant len (integer 4)	000000000000100
0135	K: .data 22	Integer 22 (single word)	0000000010110

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

כדי לבצע המרה לבינארי של אופרנדים שכותבים בשיטות מייען המשמשות בסמלים (תוויות), יש צורך לבנות טבלה המכילה את ערכיו כל הסמלים. אולם בהבדל מהקודים של הפעולות, הידיעות מראש, הרוי המעניינים בזכרון עבור הסמלים שבשימה התוכנית אינם ידועים, עד אשר תוכנית המקור נסקרה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד לעיל, האסמבלר אינו יכול לדעת שהסמל END משוויך למן 124 (עשרוני), והסמל K משוויך למן 135, אלא רק לאחר שנקרוו כל שורות התוכנית.

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

במעבר הראשון נבנית טבלת סמלים בזכרון, ובה לכל סמל שבתוכנית המקור משוויך ערך מסווני, שהוא מופיע בזיכרון או ערך קבוע, שהוגדר על ידי define. בדוגמה לעיל, טבלת הסמלים לאחר מעבר ראשון היא:

סמל	ערך (בסיס עשרוני)
SZ	2
MAIN	100
LOOP	104
L1	120
END	124
len	4
STR	125
LIST	132
K	135

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

لتשומת לב: תפקיד האסמבלר, על שני המעברים שלו, לתרגם קובץ מקור לקוד בשפת מכונה. בגמר פעולות האסמבלר, התוכנית טרם מוכנה לטיעינה בזכרון לצורך ביצוע. קוד המכונה חייב לעבור שלבי הקישור/טיעינה, ורק לאחר מכן לשלב הביצוע (שלבים אלה **אין** חלק מהמ Ма"ז).

### המעבר הראשון

במעבר הראשון נדרשים כלליים כדי לקבוע איזה מeon ישוויך לכל סמל. העיקרונו הבסיסי הוא לספור את המיקומות בזכרון, ואתם תופסות הוראות. אם כל הוראה תיתען בזכרון למקום העוקב להוראה הקודמת, תציין ספרירה כזאת את מeon ההוראה הבאה. הספרירה נעשית על ידי האסמבלר ומוחזקת במבנה ההוראות (IC). ערכו ההתחלתי של IC הוא 100 (עשרוני), ולכן קוד המכונה של ההוראה הראשונה נבנה כך שייתען בזכרון החל מeon 100. IC מעדכן בכל שורת

הורהה המקצה מקום בזיכרונו. לאחר שהאסטמבלר קובע מהו אורך ההוראה, ה-CI מוגדל במספר התאים (מיילים) הנתפסים על ידי ההוראה, וכך הוא מצביע על התא הפנוי הבא.

כאמור, כדי לקודד את ההוראות בשפת מכונה, מחזיק האסטמבלר טבלה, שיש בה קוד מתאים לכל שם פעולה. בזמן התרגומים מחליף האסטמבלר כל שם פעולה בקוד שלו, וכן כל אופרנד מוחלף בקידוד מתאים, אך פועלות החלפה זו אינה כה פשוטה. ההוראות משתמשות בשיטות מייענו מגוונות לאופרנדים. אותה פעולה יכולה לקבל שימושויות שונות, בכל אחת משיטות המייענו, וכן יכולות לתמוך תא זיכרונו לרегистר, או להעתיקת תוכן רегистר לרегистר אחר, וכן הלאה. ככל אפשרות צואת שלptom עשויי להתאים קידוד שונה.

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

כאשר נתקל האסטמבלר בתווית המופיעה בתחילת השורה, הוא יודע שלפניו הגדרה של תווית, ואז הוא משייך לה מען – תוכנו הנוכחי של CI. כך מקבלות כל התוויות את מענהן בעת ההגדירה. תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנוסף לשם התווית גם את המعنוי ומאפייניהם נוספים. כאשר תהיה התיאחסות לתווית באופרנד של הורהה כלשהי, יוכל האסטמבלר לשולף את המعنוי המתאים מטבלת הסמלים.

הוראה יכולה להתיחס גם לסמל שטרם הוגדר עד כה בתכנית, אלא יוגדר רק בהמשך התכנית. להלן דוגמה, הוראת הסתעפות למען שמוגדר על ידי התווית A שמשמעותו רק בהמשך הקוד:

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

כאשר מגיע האסטמבלר לשורת ההסתעפות (A bne), הוא טרם נתקל בהגדרת התווית A ומובן לא יודע את המعنוי המשויך לתווית. לכן האסטמבלר לא יוכל לבנות את הקידוד הבינאי של האופרנד A. נראה בהמשך כיצד נפתרת בעיה זו.

בכל מקרה, תמיד אפשר לבנות במעבר הראשון את הקידוד הבינאי המלא של המילה הראשונה של כל הורהה, את הקידוד הבינאי של מילת-המידע נוספת מיידי, או רегистר, וכן את הקידוד הבינאי של כל הנזונים (המתקבלים מההנחיות ..string).

## המעבר השני

ראינו שבמעבר הראשון, האסטמבלר אינו יכול לבנות את קוד המכונה של אופרנדים המשמשים בסמלים שעדיין לא הוגדרו. רק לאחר שהאסטמבלר עבר על כל התכנית, כך שככל הסמלים נכנסו כבר לטבלת הסמלים, יוכל האסטמבלר להשלים את קוד המכונה של כל האופרנדים.

לשם כך מבצע האסטמבלר מעבר נוסף (מעבר שני) על כל קבוע המקור, וمعدכן את קוד המכונה של האופרנדים המשמשים בסמלים, באמצעות ערכי הסמלים מטבלת הסמלים. בסוף המעבר השני, תהיה התוכנית מטורגת בשלמותה לקוד מכונה.

## הפרדת הוראות ונתוניים

בתכנית מבחןים בשני סוגים של תוכן: הוראות ונתוניים. יש לארגן את קוד המכונה כך שתהייה הפרדה בין הנתוניים וההוראות. הפרדת ההוראות והנתוניים לקטעים שונים בזיכרון היא שיטה עדיפה על פני הצמדה של הגדרות הנתוניים להוראות המשמשות בהן.

אחד הסכנות הטමונות באירוע הפרדת ההוראות מהנתוניים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית בתכנית, לנסוטו "לבצע" את הנתוניים כאלו היו הוראות חוקיות. למשל, שגיאה

שיכולה לגרום לתופעה כזו הסתעפות לא נכונה. התכנית כמובן לא תעבוד נכון, אך לרוב הנזק הוא יותר חמור, כי נוצרת חריגת חומרה ברגע שהמעבד מבצע פעולה שאינה חוקית.

האסטמבלר שלנו חייב להפריד, בקוד המכונה שהוא מיצר, בין קטע הנתונים לקטע ההוראות. בלומר בקובץ הפלט (בקוד המכונה) תהיה הפרדה של הוראות ונתוני לשני קטיעים נפרדים, ואילו בקובץ הקלט אין חובה שתהיה הפרדה כזו. בהמשך מתואר אלגוריתם של האסטמבלר, ובו פרטים כיצד לבצע את ההפרדה.

### גilio שגיאות בתוכנית המקור

הנחת המטלה היא שאין שגיאות בהגדירות המקורי, ולכן שלב קדם האסטמבלר אינו מכיל שלב גilio שגיאות, לעומת זאת האסטמבלר-Amor לגנות ולדוח על שגיאות בתחביר של תוכנית המקור, כגון פעולה שאינה קיימת, מספר אופרנדים שגוי, סוג אופרנד שאינו מותאים לפעולה, שם וגייסטר לא קיימים, ועוד שגיאות אחרות. כמו כן מודוא האסטמבלר שככל סמל מוגדר פעם אחת בדיקוק.

מכאן, שככל שגיאה המתגללה על ידי האסטמבלר נגרמת (בדרך כלל) על ידי שורת קלט מסויימת.

לדוגמה, אם מופיעים שני אופרנדים בהוראה שאמור להיות בה רק אופרנד יחיד, האסטמבלר ייתן הودעת שגיאה בנוסח "יוטר מדי אופרנדים".

הערה: אם יש שגיאה בקוד האסטמבלி בגוף מקארו, הרי שגיאה זו יכולה להופיע ולהתגלות שוב ושוב, בכל מקום בו נפרש המקארו. נשים לב שכאשר האסטמבלר בודק שגיאות, כבר לא ניתן לזהות שזה קוד שנפרש מקארו, כך שלא ניתן לחסוך גilio שגיאה כפוליטים.

האסטמבלר ידפיס את 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: האסטמבלר אין עוצר את פעולתו אחרי שנמצאה השגיאה הראשונה, אלא ממשיך לעבור על הקלט כדי לגנות שגיאות נוספות, ככל שישן. כמו כן שאין כל טעם לייצר את קבצי הפלט אם נתגלו שגיאות (ممילא אי אפשר להשלים את קוד המכונה).

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

שיטות מיעון חוקיות עבור אופרנד יעד	שיטות מיעון חוקיות עבור אופרנד מקור	שם פעולה
1,2,3	0,1,2,3	<code>mov</code>
0,1,2,3	0,1,2,3	<code>cmp</code>
1,2,3	0,1,2,3	<code>add</code>
1,2,3	0,1,2,3	<code>sub</code>
1,2,3	אין אופרנד מקור	<code>not</code>
1,2,3	אין אופרנד מקור	<code>clr</code>
1,2,3	1,2	<code>lea</code>
1,2,3	אין אופרנד מקור	<code>inc</code>
1,2,3	אין אופרנד מקור	<code>dec</code>
1,3	אין אופרנד מקור	<code>jmp</code>
1,3	אין אופרנד מקור	<code>bne</code>
1,2,3	אין אופרנד מקור	<code>red</code>
0,1,2,3	אין אופרנד מקור	<code>prn</code>
1,3	אין אופרנד מקור	<code>jsr</code>
אין אופרנד יעד	אין אופרנד מקור	<code>rts</code>
אין אופרנד יעד	אין אופרנד מקור	<code>hlt</code>

## אלגוריתם שלדי של האסטבלר

לחידוד ההבנה של תהליכי העבודה של האסטבלר, נציג להלן אלגוריתם שלדי למעבר הראשון ולמעבר השני. لتשומת לב: אין חובה להשתמש דווקא באלגוריתם זה.

אנו מחלקים את קוד המכונה לשני אזורים, אזור ההוראות (code) ואזור הנתונים (data). לכל אזור יש מונה משלה, ונסמנם IC (МОНА ההוראות - Instruction-Counter) ו-DC (МОНА הנתונים - Data-Counter). בניית את קוד המכונה כך שיתאים לטיענה לזכרון החל מכתובת 100.

כמו כן, נסמן ב- L את מספר המיללים שתופס קוד המכונה של הוראה נתונה.

בכל מעבר מתחילה לקרוא את קובץ המקור מהתחלתה.

### מעבר ראשון

- .1.  $IC \leftarrow 0$ ,  $DC \leftarrow 0$ .
- .2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עברו ל-16.
- .3. האם זהה הגדרת define (קבוע)? אם לא, עברו ל-5.
- .4. הכנס את שם הקבוע לטבלת הסמלים עם המאפיין mdefine. ערכו יהיה כפי שמופיע בהגדרת. (אם הסמל כבר נמצא בטבלה, יש להזדיע על שגיאה). חוזר ל-2.
- .5. האם השדה הראשון בשורה הוא סמל? אם לא, עברו ל-7.
- .6. הדלק דגל "יש גדרת סמל".
- .7. האם זהה הינה לאחסון נתונים, כמו למשל, האם הינה data.string או ?.string? אם לא, עברו ל-.
- .10. אם יש הגדרת סמל (תוויות), הכנס אותו לטבלת הסמלים עם המאפיין data. ערכו יהיה DC. (אם הסמל כבר נמצא בטבלה יש להזדיע על שגיאה).
- .9. זהה את סוג הנתונים, קודו אותם בזיכרון, ועדכן את מונה הנתונים DC בהתאם לאורכם. אם זהה הינה.data., ויש בה נתון שהוא סמל, בדוק שהסמל מופיע בטבלה עם המאפיין mdefine, והשתמש בערכו. (אם הסמל אינו בטבלה או לא מאופיין כ- mdefine יש להזדיע על שגיאה). חוזר ל-2.
- .10. האם זו הינה entry.או הינה ?.entry.? אם לא, עברו ל-12.
- .11. האם זהה הינה extern.? אם כן, הכנס כל סמל (אחד או יותר) המופיע כאופrnd של ההנעה לתוך טבלת הסמלים ללא ערך, עם המאפיין external. ערכו יהיה IC+100.
- .12. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם המאפיין code. ערכו יהיה IC+100 (אם הסמל כבר נמצא בטבלה יש להזדיע על שגיאה).
- .13. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא – הודיע על שגיאת שם הוראה.
- .14. נתח את מבנה האופrndים של ההוראה וחשב את L. בנה כתעת את הקוד הבינארי של המילה הראשונה של ההוראה.
- .15. עדכן  $IC \leftarrow IC + 2$ , וחזר ל-2.
- .16. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר הראשון, עצור כאן.
- .17. עדכן בטבלת הסמלים את ערכו של כל סמל המופיעים כ- data, ע"י הוספת הערך IC+100 (ראה הסבר בהמשך).
- .18. התחל מעבר שני.

### מעבר שני

- .1.  $IC \leftarrow 0$ .
- .2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עברו ל-9.
- .3. אם השדה הראשון הוא סמל, דרג עליו.
- .4. האם זהה הינה ?.entry. או .string.? אם כן, חוזר ל-2.
- .5. האם זהה הינה entry.? אם לא, עברו ל-7.

6. סמן בטבלת הסמלים את הסמלים המותאים במאפיין entry. חזרה ל-2.
7. השלם את קידוד האופרנדים החל מהמילה השנייה בקוד הבינארי של הוראה, בהתאם לשיטת המיעון. אם אופרנד מכיל סמל, מצא את הערך בטבלת הסמלים (אם הסמל לא נמצא בטבלה, יש להודיע על שגיאה).
8. עדכן  $L \leftarrow IC + 1$ , וחזור ל-2.
9. קובץ המקור נקרא בשלמותו. אם נמצא שגיאות במעבר השני, עוזר כאן.
10. בנה את קבצי הפלט (פרטים נוספים בהמשך).

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

```
.define sz = 2
MAIN:    mov   r3, LIST[sz]
LOOP:     jmp   L1
          prn   #-5
          mov   STR[5], STR[2]
          sub   r1, r4
          cmp   r3, #sz
          bne   END
L1:       inc   K
          bne   LOOP
END:      hlt

.define len = 4
STR:      .string "abcdef"
LIST:    .data  6, -9, len
K:        .data  22
```

נבצע עתה מעבר ראשון על הקוד הנוכחי. נבנה את טבלת הסמלים. כמו כן, נבצע במעבר זה גם את קידוד כל הנתונים, וקידוד המילה הראשונה של כל הוראה. את החלקים שעדיין לא מתרגמים במעבר זה, נשאיר כמותם שהם (מסומנים ב- ? בדוגמה להלן).

אנו מניחים שהקוד ייטע ל זיכרון החל מהמ عن 100 (בסיס דצימלי).

Decimal Address	Source Code	Explanation	Binary Machine Code
0100	MAIN: mov r3, LIST[sz]	First word of instruction	00000000111000
0101		Source register 3	0000000011000000
0102		Address of label LIST (integer array)	?
0103		Value of sz (index 2)	000000000001000
0104	LOOP: jmp L1		00001001000100
0105		Address of label L1	?
0106	prn #-5		000011000000000
0107		Immediate value -5	1111111101100
0108	mov STR[5], STR[2]		00000000101000
0109		Address of label STR (string)	?
0110		Index 5	00000000010100
0111		Address of label STR	?
0112		Index 2	000000000001000
0113	sub r1, r4		00000011111100
0114		Source register 1 and target register 4	00000000110000
0115	cmp r3, #sz		000000001110000
0116		Source register 3	0000000011000000
0117		Value of sz (immediate #2)	000000000001000
0118	bne END		00001010000100
0119		Address of label END	?

Decimal Address	Source Code	Explanation	Binary Machine Code
0120	L1: inc K		00000111000100
0121		Address of label K (integer)	?
0122	bne LOOP		00001010000100
0123		Address of label LOOP	?
0124	END: hlt		00001111000000
0125	STR: .string "abcdef"	Ascii code 'a'	00000001100001
0126		Ascii code 'b'	00000001100010
0127		Ascii code 'c'	00000001100011
0128		Ascii code 'd'	00000001100100
0129		Ascii code 'e'	00000001100101
0130		Ascii code 'f'	00000001100110
0131		Ascii code '\0' (end of string)	00000000000000
0132	LIST: .data 6, -9, len	Integer 6 (first in array of 3 words)	00000000000010
0133		Integer -9	1111111110111
0134		Value of len (integer 4)	000000000000100
0135	K: .data 22	Integer 22 (single word)	00000000010110

טבלת הסמלים :

סמל	מיפויים	ערך (בסיס עשרוני)
sz	mdefine	2
MAIN	code	100
LOOP	code	104
L1	code	120
END	code	124
len	mdefine	4
STR	data	125
LIST	data	132
K	data	135

נבע עתה את המעבר השני. נשלים את הקידוד החסר באמצעות טבלת הסמלים, ונרשום את הקוד בצורתו הסופית :

Decimal Address	Source Code	Binary Machine Code
0100	MAIN: mov r3, LIST[sz]	00000000111000
0101		00000001100000
0102		00001000010010
0103		000000000001000
0104	LOOP: jmp L1	00001001000100
0105		00000111100010
0106	prn #5	000011000000000
0107		11111111101100
0108	mov STR[5], STR[2]	00000000101000
0109		00000111110010
0110		000000000010100
0111		00000111110010
0112		000000000010000
0113	sub r1, r4	00000011111100
0114		00000000110000

Decimal Address	Source Code	Binary Machine Code
0115	cmp r3, #sz	00000001110000
0116		00000001100000
0117		00000000001000
0118	bne END	00001010000100
0119		00000111110010
0120	L1: inc K	00000111000100
0121		00001000011110
0122	bne LOOP	00001010000100
0123		00000110100010
0124	END: hlt	00001111000000
0125	STR: .string "abcdef"	00000001100001
0126		00000001100010
0127		00000001100011
0128		00000001100100
0129		00000001100101
0130		00000001100110
0131		00000000000000
0132	LIST: .data 6, -9, len	00000000000010
0133		1111111110111
0134		000000000000100
0135	K: .data 22	00000000010110

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

### קבצי קלט ופלט של האסמבלייר

בפעולת של האסמבלייר, 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 האסמבלי שהוגדרה בມמ"ז זה.

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

- קובץ am, המכיל את קובץ המקור לאחר שלב קדם האסמבלייר (לאחר פרישת המאקרואים)
- קובץ object, המכיל את קוד המוכונה.
- קובץ externals, ובו פרטיים על כל המKENOTOT (הכוNOTOT) בקוד המוכונה בהם יש מילת- מידע שמקודדת ערך של סמל שהוצעו בchiezoni (סמל שהופיע כאופרנד של הנחיתת ..extern, ומואופיין בטבלת הסמלים כ- external).
- קובץ entries, ובו פרטיים על כל סמל שימושה כניסה (סמל שהופיע כאופרנד של הנחיתת entry, ומואופיין בטבלת הסמלים כ- entry).

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

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

לדוגמא : נניח שתוכנית האסמבלייר שלנו נקראת assembler, אז שורת הפקודה הבאה :

```
assembler x y hello
```

תורץ את האסמבלייר על הקבצים : x.as, y.as, hello.as

שמות קבצי הפלט מבוססים על שם קובץ הקלט, כפי שהופיע בשורת הפקודה, בתוספת סימנת `object`, שמתאימה : הסימנת `"am."` עבר קובץ לאחר פרישת מאקרו, הסימנת `"ob."` עבר קובץ ה-`entries`, והסימנת `"ext."` עבר קובץ ה-`externals`.

לדוגמא, בהפעלת האסמלר באמצעות שורת הפקודה : `x assembler` ייווצר קובץ פלט `.o.x`, וכן קבצי פלט `.ext.x` ו- `.ent.x` ככל שיש הנחיות `.extern`. או `.as`. בקובץ המקור. אם אין מאקרו בקובץ המקור, אז קובץ `"am."` יהיה זהה לקובץ `"as."`.

### **אופן פעולה האסמלר**

נרחיב כאן על אופן פעולה האסמלר, בנוסף לאלגוריתם השודי שניתנו לעיל.

האסמלר מחזיק שני מערכים, שיקראו להן מערך הוראות ומערך הנתונים. מערכים אלו נתונים למשזה תמונה של זיכרון המcona (גודל כל כניסה בזיכרון זהה לגודלה של מילת מכונה : 14 סיביות). במערך ההוראות מכניס האסמלר את הקידוד של הוראות המכונה שנקרוו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמלר את קידוד הנתונים שנקרוו מקובץ המקור (שורות מסווג `.data`).

לאסמלר יש שני מונחים : מונה ההוראות (IC) ומונה הנתונים (DC). מונחים אלו מצביעים על המקום הבא הפניו במערכות לעיל, בהתאם. כשהתחל האסמלר לעבר על קובץ מקור, שני מונחים אלו מואפסים.

בנוסף יש לאסמלר טבלה, אשר בה נאספות כל התוויות בהן נתקל האסמלר במהלך המעבר על הקובץ. לטבלה זו קוראים טבלת סמלים (symbol-table). לכל סמל (תווית) נשמרם שמו, ערכו, ומאפיינים שונים שצינו קודם, כגון המיקום (`data` או `code`) או אופן העדכוון (`relocatable` או `external`).

האסמלר קורא את קובץ המקור שורה אחר שורה, מחליט מהו סוג השורה (הערה, קבוע, הוראה, הנחיה, או שורה ריקה) ופועל בהתאם.

1. שורה ריקה או שורת הערה : האסמלר מתעלם מהשורה וועבר לשורה הבאה.
2. שורת קבוע : האסמלר מכניס את שם הקבוע לטבלת הסמלים עם המאפיין `.mdefine`.
3. שורת הוראה :

האסמלר מוצא מהי הפעולה, ומהן שיטות המיעון של האופרנדים. (מספר האופרנדים אותם הוא מփש נקבע בהתאם להוראה אותה הוא מצא). האסמלר קובע לכל אופרנד את ערכו באופן הבא :

- אם זה רגיסטר – האופרנד הוא מספר הרגייסטר.
- אם זו תווית (מייען ישיר) – האופרנד הוא ערך התווית כפי שמופיע בטבלת הסמלים (ייתכן והסמל טרם נמצא בטבלת הסמלים).
- אם זה התו # ואחריו מספר או שם של קבוע `define` (מייען מיד) – האופרנד הוא המספר עצמו.
- אם זו שיטת מייען אחרת – ערכו של האופרנד נקבע לפי המפרט של שיטת המייען (ראו תיאור שיטות המייען לעיל)

קביעת שיטת המייען נעשית בהתאם לתחביר של האופרנד, כפי שהסביר לעיל בהגדרת שיטות המייען. למשל, מספר מצין מייען מיידי, תווית מצינית מייען ישיר וכד'.

לאחר שהאסמלר ניתח את השורה והחליט לגבי הפעולה, שיטת מייען אופרנד המקור (אם יש), ושיטת מייען אופרנד היעד (אם יש), הוא פועל באופן הבא :

אם זהה פועלה בעלת שני אופרנדים, אזי האסטבלר מכניס לערך ההוראות, במקום עליו מצביע מונה ההוראות IC, את קוד המילה הראשונה של ההוראה (בשיטת היצוג של הוראות המכונה כפי שתואר קודם לעיל). מילה זו מכילה את קוד הפעולה, ואת שיטות המיעון. בנוסח "משרין" האסטבלר מักם במערך עבור המילים הנוספות הנדרשות עבור הוראה זו, ומגדיל את מונה ההוראות בהתאם. אם אחד או שני אופרנדים הם בשיטת מיעון רגיستر או מיידי, האסטבלר מוקודד-cut את המילים הנוספות הרלוונטיות במערך ההוראות.

אם זהה פועלה בעלת אופרנד אחד בלבד, ככלומר אין אופרנד מקור, אזי הקידוד הינו זהה לעיל, פרט לשיביות של שיטת המיעון של אופרנד המקור במילה הראשונה, אשר יכילה תמיד 0, מכיוון שאין רלוונטיות לפועלה.

אם זהה פועלה ללא אופרנדים אזי תקודד רק המילה הראשונה (והיחידה). השיביות של שיטות המיעון של שני אופרנדים יכילה 0.

אם בשורת ההוראה קיימת תווית, אזי התווית מוכנסת אל טבלת הסמלים תחת השם המתאים, ערך התווית הוא ערך מונה ההוראות לפני קידוד ההוראה.

#### 4. שורת הנחיה :

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

##### I. '.data'

האסטבלר קורא את רשימת המספרים, המופיעה לאחר '.data', מכניס כל מספר אל מערך הנתונים, ומקדם את מצביע הנתונים DC באחד עבור כל מספר שהוכנס. נשים לב שם של קבוע define יכול לשמש במקום מספר.

אם בשורה '.data' יש תווית, אזי תווית זו מוכנסת לטבלת הסמלים. היא מקבלת את הערך של מונה הנתונים DC שלפני הכנסת המספרים למערך הנתונים. הטיפוס של התווית הוא relocatable, וכך מסומן שההגדרה ניתנה בחלק הנתונים.

בסוף המעבר הראשוני, ערך התווית יעדכן בטבלת הסמלים על ידי הוספה ה-IC (כלומר הוספה האורך הכלול של קידוד כל ההוראות). הסיבה לכך היא שבתמונה קוד המכונה, הנתונים מופרדים מההוראות, וכל הנתונים יופיעו אחרי כל ההוראות (ראו תאור קבצי הפלט בהמשך).

##### II. '.string'

הטיפול ב-'string' דומה ל-'data', אלא שקודוי ascii של התווים הם אלו המוכנסים אל מערך הנתונים (כל TWO בכינסה נפרדת). לאחר מכן מוכנס הערך 0 (המציע סוף מחוזות) אל מערך הנתונים. מונה הנתונים מוקדם באורך המחרוזת + 1 (גם האפס בסוף המחרוזות וטופס מקום).

הטיפול בתווית המוגדרת בשורה זו זהה לטיפול הנעשה בהנחיה 'data'.

##### III. '.entry'

זהה בקשה לאסטבלר להכניס את התווית המופיעה כאופרנד של '.entry'. אל קובץ ה-entries האסטבלר רושם את הבקשה ובסיום העבודה, התווית הניל תירשם בקובץ ה-entries.

##### IV. '.extern'

זהה הצהרה על סמל (תווית) המוגדר בקובץ אחר, ואשר קטע האסטבלר בקובץ הנווכי עושה בו שימוש. האסטבלר מכניס את הסמל אל טבלת הסמלים. ערכו הוא 0 (הערך האמתי לא ידוע, ויקבע רק בשלב הקישור), וטיפוסו הוא external. לא ידוע באיזה קובץ נמצאת הגדרת הסמל (וגם אין זה משנה עבור האסטבלר).

יש לשים לב: בהוראה או בהנחיה אפשר להשתמש בשם של סמל אשר הצהרה עליו ניתנת בהמשך הקובץ (אם באופן ישיר על ידי הגדרת תווית, ואם באופן עקיף על ידי הנחיתות external).

בסוף המעבר הראשון, האסםבלר מערכן בטבלת הסמלים כל סמל המופיע כ-`data`, על ידי הוספת `IC+100` (עשרוני) לערכו של הסמל. הסיבה לכך היא שבתמונה הכללת של קוד המכונה, הנתונים מופרדים מהhorאות, וכל הנתונים נדרשים להופיע אחרי כל horאות. סמל מסווג `data` הוא למעשה תווית באזור הנתונים, והעדכו מושך לערך הסמל (כלומר כתובתו בזיכרון) את האורך הכלל של קיזוז כל horאות, בתוספת כתובות התחלת הטעינה של הקיזוז, שהוא 100.

טבלת הסמלים מכילה כעת את כל הערכים הנחוצים להשלמת הקיזוז (למעט ערכים של סמלים חייצוניים).

במעבר השני, האסםבלר מקודד באמצעות טבלת הסמלים את כל המיללים במערך horאות שטרם קודדו במעבר הראשון. אלו הן מיללים שצרכות להכיל כתובות של תוויות (שדה ה-`E`, `R`, `A`, `01` או `10` במיילים אלה יהיה `10` או `01`).

## פורמט קובץ ה- object

האSEMBLER בונה את תMOVות זיכרונו המוכנה כך שקידוד ההוראה הראשונה מקובץ האSEMBLER ייכנס למן 100 (בבסיס עשרוני) בזיכרונו, קידוד ההוראה השנייה יכנס מען העוקב אחרי ההוראה הראשונה (תלו依 במספר המילים של ההוראה הראשונה), וכך הלאה עד לההוראה الأخيرة.

מיד לאחר קידוד ההוראה الأخيرة, מכניםים לתMOVת הזיכרונו את קידוד הנתונים שנבנו על ידי ההנחיות 'data'.<sup>string</sup>. הנתונים יוכנסו בסדר בו הם מופיעים בקובץ המקור. אופרנד של ההוראה שמתייחס לSAMPLE שהוגדר באותו קובץ, יקודד כך שיציביע על המיקום המתאים בתMOVת הזיכרונו שבונה האSEMBLER.

נשים לב שהMOVתים מופיעים בתMOVת הזיכרונו אחרי ההוראות. זהה הסיבה בגללה יש לעדכן בטבלת SAMPLEים, בסוף המעבר הראשון, את ערכי SAMPLEים המגדירים נתונים (SAMPLEים מסווג .data).

עקרונית, קובץ object מכיל את תMOVת הזיכרונו שתוארה כאן. קובץ object מורכב משורות של טקסט כדלקמן :

השורה הראשונה בקובץ ה-object היא "cotratt", המכילה שני מספרים (בבסיס עשרוני) : האורך הכלול של SAMPLE ההוראות (זיכרונו) ואחריו האורך הכלול של SAMPLE הנתונים (SAMPLE זיכרונו). בין שני המספרים יש רווח אחד.

השורות הבאות בקובץ מכילות את תMOVת הזיכרונו. בכל שורה שני ערכים : כתובות של מילה בזיכרונו, ותוכן המילה. הכתובת תירשם בסיס עשרוני בארכע ספרות (כולל APPSים מוביילים). תוכן המילה יירשם בסיס 4 "mozfun" (ראה להלן) בשבע ספרות (כולל APPSים מוביילים). בין שני הערכים בכל שורה יפריד רווח אחד.

בסיס 4 רגיל	0	1	2	3
בסיס 4 מוצפן	*	#	%	!

קובץ object לדוגמה, כפי שאמור להיבנות על ידי האSEMBLER, נמצא בהמשך.

## פורמט קובץ ה- entries

קובץ entries בניית MOVות טקסט. כל שורה מכילה שם של SAMPLE שהוגדר כ-entry ואת ערכו, כפי שנמצא בטבלת SAMPLEים. הערכים מיוצגים בסיס עשרוני.

## פורמט קובץ ה- externals

קובץ externals בניית MOVות טקסט. כל שורה מכילה שם של SAMPLE שהוגדר כ-external, וכותבות בקוד המוכנה בה יש קידוד המתייחס לSAMPLE זה. כМОון שיתיכן ויש מספר כתובות בקוד המוכנה בהם מתיחסים לאותו SAMPLE חיצוני. ככל התייחסות כזו תהיה שורה נפרדת בקובץ externals. הכתובות מיוצגות בסיס עשרוני.

מדגים את קבצי הפלט שמייצר האSEMBLER עבור קובץ מקור בשם as.ps שהודם קודם לכן. התוכנית לאחר שלב פרישת המאקרו תיראה כך :

```
; file ps.as
```

```
.entry LIST
.extern W
.define sz = 2
MAIN:      mov    r3, LIST[sz]
```

```

LOOP:    jmp   W
        prn   #-5
        mov   STR[5], STR[2]
        sub   r1, r4
        cmp   K, #sz
        bne   W
L1:      inc   L3
.entry LOOP
        bne   LOOP
END:     hlt
.define len = 4
STR:     .string "abcdef"
LIST:    .data   6, -9, len
K:       .data   22
.extern L3

```

להלן טבלת הקידוד המלא הבינארי שמתקיים מקובץ המקור, ולאחריה הפורמטים קבצי הפלט השונים.

Decimal Address	Source Code	Binary Machine Code
0100	MAIN: mov r3, LIST[sz]	000000000111000
0101		000000001100000
0102		00001000010010
0103		000000000001000
0104	LOOP: jmp W	00001001000100
0105		0000000000000001
0106	prn #-5	000011000000000
0107		11111111101100
0108	mov STR[5], STR[2]	00000000101000
0109		00000111110110
0110		00000000010100
0111		00000111110110
0112		000000000001000
0113	sub r1, r4	00000011111100
0114		000000000110000
0115	cmp K, #sz	000000001010000
0116		00001000011110
0117		000000000001000
0118	bne W	00001010000100
0119		0000000000000001
0120	L1: inc L3	00000111000100
0121		0000000000000001
0122	bne LOOP	00001010000100
0123		00000110100010
0124	END: hlt	000011110000000
0125	STR: .string "abcdef"	000000001100001
0126		000000001100010
0127		000000001100011
0128		000000001100100
0129		000000001100101
0130		000000001100110
0131		0000000000000000

Decimal Address	Source Code	Binary Machine Code
0132	LIST: .data 6, -9, len	0000000000000110
0133		1111111110111
0134		0000000000000100
0135	K: .data 22	00000000010110

להלן תוכן קבצי הפלט של הדוגמה.

הקובץ ps.o

```

25 11
0100 * * * * ! % *
0101 * * * # % * *
0102 * * % * # * %
0103 * * * * * % *
0104 * * % # * # *
0105 * * * * * #
0106 * * ! * * * *
0107 ! ! ! ! % ! *
0108 * * * * % ! *
0109 * * # ! ! # %
0110 * * * * # # *
0111 * * # ! ! # %
0112 * * * * * % *
0113 * * * ! ! ! *
0114 * * * * ! * *
0115 * * * # # *
0116 * * % * # ! %
0117 * * * * * # *
0118 * * % % * # *
0119 * * * * * #
0120 * * # ! * # *
0121 * * * * * #
0122 * * % % * # *
0123 * * # % % * %
0124 * * ! ! * * *
0125 * * * # % * #
0126 * * * # % * %
0127 * * * # % * !
0128 * * * # % # *
0129 * * * # % # #
0130 * * * # % # %
0131 * * * * * *
0132 * * * * * # %
0133 ! ! ! ! ! # !
0134 * * * * * # *
0135 * * * * # %
```

הקובץ ps.ent

```

LOOP 0104
LIST 0132
```

כל המספרים בבסיס עשרוני

הקובץ ps.ext

```

W 0105
W 0119
L3 0121
```

כל המספרים בbasis עשרוני

לתשומת לב : אם בקובץ המקור אין הנקיות `extern`. אז לא ייווצר קובץ `ext`. בדומה, אם אין בקובץ המקור הנקיות `entry`, לא ייווצר קובץ `ent`. **אין לייצור קובץ ext או ent שנשאר ריק**.

הערה : אין חשיבות לסדר השורות בקבצים מסווג `ent` או `ext`. כל שורה עומדת בפני עצמה.

## סיכום והנחיות כלליות

- אורך התוכנית, הניתנת כקלט לאסטבלר אינו ידוע מראש, ולכן אורך התוכנית המתורגמת אינו אמור להיות צפוי מראש. אולם כדי להקל בימוש האסטבלר, ניתן להניח גודל מקסימלי. לפיכך יש אפשרות לשימוש במערכים לאחסן תMOVת קוד המוגנה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים וטבלת המאקרו), יש למש באופן ייעיל וחסכוני (למשל באמצעות רשיימה מקושרת והקצת זיכרון דינמי).
- השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הקלט, למעט הסיומות. למשל, אם קובץ הקלט הוא `prog.as` אז קבצי הפלט שייצרו הם : `prog.ob`, `prog.ext`, `prog.ent`.
- מתכונת הפעלת האסטבלר צריכה להיות כפי הנדרש בממ"ז, ללא שינויים כלשהם. ככלומר, ממשך המשתמש יהיה אך ורק באמצעות שורת הפוקודה. בפרט, שמות קבצי המקור יועברו לתכנית האסטבלר כארגומנטים בשורת הפוקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גורפיים למיניהם, ועוד'.
- יש להקפיד לחלק את מימוש האסטבלר למספר מודולים (קבצים בשפט C) לפי משימות. אין לרוץ משימות מסוימים שונים במודול יחיד. מומלץ לחלק למודולים כגון : מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבטיס, ניתוח תחבירי של שורה), טבלת הסמלים, מפתח הזיכרון, טבלאות קבועות (קוד הפעולה, שיטות המיעון החוקיקות לכל פעולה, ועוד').
- יש להקפיד ולהתעד את המימוש באופן מלא וברור, באמצעות הערות מפורחות בקוד.
- יש לאפשר תווים לבנים עודפים בקובץ הקלט בשפט אסטבלרי. למשל, אם בשורת הוראה יש שני אופrndים המופרדים בפסיק, אזי לפני ואחריו הפסיק מותר להיות רווחים וטאבים בכל 경우에는. בדומה, גם לפני ואחריו שם הפעולה. מוגדרות גם שורות ריקות. האסטבלר יתעלם מתחומים לבנים מיוחדים (כלומר ידלג עליהם).
- הקלט (קוד האסטבלרי) עלול להכיל שגיאות תחביריות. על האסטבלר לגלות ולדוח על כל השורות השגויות בקלט. אין לעצור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להדפיס למסך הודעות מפורטות ככל הניתן, כדי שאפשר יהיה להבין מה והיכן כל שגיאה. כמובן שם קובץ קלט מכיל שגיאות, אין טעם להפיק עבورو את קבצי הפלט (`ob`, `ext`, `ent`).

**תמ ונשלם פרק ההסברים והגדרת הפרויקט.**

- בשאלות ניתן לפנות לקבוצת הדיוון באתר הקורס, ועל כל אחד מהמנהלים בשעות הקבלה שלهما.**
- לזהירותם, באפשרותם של כל סטודנט לפנות לכל מנהה, לאו דזוקא למנהה 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 להופיע בכלם.
- לתשומתיכם : לא תיתן דחיה בהגשת הממ"ז, פרט למקרים מיוחדים כגון מילואים או אשפוז. במקרים אלו יש לבקש ולקבל אישור **מראש מנהלה** הקבוצה.

**ב ה צ ל ח ה !**