מטלת מנחה (ממ"ן) 14

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרויקט גמר

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

סמסטר: 2019ב' | מועד אחרון להגשה: 12.8.2019

קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
- שליחת מטלות באמצעות דואר אלקטרוני - באישור המנחה בלבד

הסבר מפורט ב"נוהל הגשת מטלות מנחה"

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

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

עליכם להגיש:

1. קבצי המקור של התוכנית שכתבתם (קבצים בעלי סיומת c. או h.).
2. קובץ הרצה (מקומפל ומקושר) עבור מערכת אובונטו.
3. קובץ makefile. יש להשתמש בקומפיילר gcc עם הדגלים: Wall -ansi -pedantic-. יש לנפות את כל ההודעות שמוציא הקומפיילר, כך שהתכנית תתקמפל ללא כל הערות או אזהרות.
4. דוגמאות הרצה (קלט ופלט):

א. קבצי קלט בשפת אסמבלי, וקבצי הפלט שנוצרו מהפעלת האסמבלר על קבצי קלט אלה. יש להדגים שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האסמבלי.

ב. קבצי קלט בשפת אסמבלי המדגימים מגוון רחב של סוגי שגיאות אסמבלי (ולכן לא נוצרים קבצי פלט), ותדפיסי המסך המראים את הודעות השגיאה שמוציא האסמבלר.

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

נזכיר מספר היבטים חשובים של כתיבת קוד טוב:

1. הפשטה של מבני הנתונים: רצוי (ככל האפשר) להפריד בין הגישה למבני הנתונים לבין המימוש של מבני הנתונים. כך, למשל, בעת כתיבת פונקציות לטיפול בטבלה, אין זה מעניינם של המשתמשים בפונקציות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקושרת.

2. קריאות הקוד: יש להשתמש בשמות משמעותיים למשתנים ופונקציות. כמו כן, רצוי להגדיר קבועים רלוונטיים תוך שימוש בהנחית #define, ולהימנע מ"מספרי קסם", שמשמעותם נהירה לכם בלבד. יש לערוך את הקוד באופן מסודר: הזחות עקביות, שורות ריקות להפרדה בין קטעי קוד, וכד'.

3. תיעוד: יש להכניס בקבצי המקור תיעוד תמציתי וברור, שיסביר את תפקידה של כל פונקציה (באמצעות הערות כותרת לכל פונקציה). כמו כן יש להסביר את תפקידם של משתנים חשובים. כמו כן, יש להכניס הערות ברמת פירוט טובה בכל הקוד.

הערה: תוכנית "עובדת", דהיינו תוכנית שמבצעת את הדרוש ממנה, אינה לכשעצמה ערובה לציון גבוה. כדי לקבל ציון גבוה, על התכנית לעמוד בקריטריונים של כתיבה ותיעוד ברמה טובה, כמתואר לעיל, אשר משקלם המשותף מגיע עד לכ- 40% ממשקל הפרויקט.

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

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

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

רקע כללי ומטרת הפרויקט

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

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

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

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

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

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

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

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

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

המחשב הדמיוני ושפת האסמבלי

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

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

"חומרה":

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

למעבד 8 אוגרים כלליים, בשמות: r0, r1, r2, r3, r4, r5, r6, r7.
גודלו של כל אוגר הוא 14 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 13. שמות האוגרים נכתבים תמיד עם אות 'r' קטנה.

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

גודל הזיכרון הוא 4096 תאים, בכתובות 0-4095 (בבסיס עשרוני), וכל תא הוא בגודל של 14 סיביות. לתא בזיכרון נקרא גם בשם "**מילה**". הסיביות בכל מילה ממוספרות כמו באוגר.

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), המיוצגים בקוד ascii.

מבנה הוראת מכונה:

כל הוראת מכונה מקודדת למספר מילות זיכרון רצופות, **החל ממילה אחת ועד למקסימום חמש מילים**, הכל בהתאם לשיטות המיעון בהן נעשה שימוש (ראו בהמשך).

בקובץ הפלט המכיל את קוד המכונה שבונה האסמבלר, כל מילה תקודד בבסיס 4 "מיוחד" המוגדר כדלקמן (ראו הסברים ודוגמאות בהמשך):

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

בכל סוגי 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	מיעון אופרנד מקור	מיעון אופרנד יעד	A,R,E

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

בשפה שלנו יש 16 קודי פעולה והם :

שם הפעולה	קוד הפעולה (בבסיס 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
stop	15

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

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

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

סיביות 13-10: אינן בשימוש וערכן הוא 0 .

סיביות 1-0 (השדה 'A,R,E'): במילה הראשונה של הוראה סיביות אלה תמיד מאופסות (00).

לתשומת לב : **השדה 'A,R,E' מתווסף לכל אחת מהמילים בקידוד ההוראה** (ראו פירוט שיטות המיעון להלן).

<u>שיטות מיעון :</u>

בשפת האסמבלי שלנו קיימות ארבע שיטות מיעון, המסומנות במספרים 0,1,2,3.
השימוש בשיטות מיעון מצריך קידוד של מילות-מידע נוספות בקוד המכונה של כל הוראת מכונה.
כל אופרנד של ההוראה תופס מילה אחת או שתי מילים נוספות, תלוי בשיטת המיעון של האופרנד.

כאשר בהוראה יש שני אופרנדים, קודם יופיעו מילות-המידע הנוספות של האופרנד הראשון (אופרנד המקור), ולאחריהן מילות-המידע הנוספות של האופרנד השני (אופרנד היעד). קיים גם מקרה מיוחד בו קידוד שני האופרנדים נעשה באמצעות מילת-מידע אחת משותפת לשני האופרנדים.

כאמור, בכל מילת-מידע נוספת של ההוראה, סיביות 1-0 הן השדה A,R,E. השדה מציין מהו סוג הקידוד של המילה : קידוד מוחלט (Absolute) , חיצוני (External) או ניתן להזזה (Relocatable).
הערך 00 משמעו שקידוד המילה הוא **מוחלט** (ואינו מצריך שינוי בשלבי הקישור והטעינה).
הערך 01 משמעו שהקידוד הוא של **כתובת חיצונית** (ומצריך שינוי בשלבי הקישור והטעינה).
הערך 10 משמעו שהקידוד הוא של **כתובת פנימית הניתנת להזזה** (ומצריך שינוי בקישור ובטעינה).

אפיון יותר מפורט של תפקיד השדה 'A,R,E' בקוד המכונה מופיע בהמשך.

להלן תיאור שיטות המיעון:

ערך	שיטת מיעון	תוכן המילה נוספת	אופן הכתיבה	דוגמה

0	מיעון מיידי	מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר שלם בשיטת המשלים ל-2, המיוצג ברוחב של 12 סיביות, אליהן מתווספות זוג סיביות של השדה A,R,E (הערך של שדה זה הוא תמיד 00 עבור מיעון מיידי).	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני.
			
			יש גם אפשרות שבמקום המספר יופיע שם של מאקרו שהוגדר בתכנית (ראו פרטים בהמשך).
				mov #-1,r2
				
				בדוגמה זו האופרנד הראשון של הפקודה נתון בשיטת מיעון מיידי. ההוראה כותבת את הערך 1- אל אוגר r2
				
				דוגמה נוספת:
				נתונה הגדרת המאקרו:
				.define size = 8
				
				אזי בהוראה:
				mov #size, r1
				האופרנד הראשון הוא מיידי, כאשר המספר 8 מיוצג באמצעות שם המאקרו size. ההוראה כותבת את הערך 8 אל אוגר r1

1	מיעון ישיר	מילת-מידע נוספת של ההוראה מכילה כתובת של מילה בזיכרון. מילה זו בזיכרון היא האופרנד. הכתובת מיוצגת כמספר ללא סימן ברוחב של 12 סיביות, אליהן מתווספות זוג סיביות של השדה A,R,E (הערך של שדה זה הוא או 01 או 10, תלוי בסוג הכתובת - חיצונית או פנימית).	האופרנד הוא <u>תווית</u> שכבר הוצהרה או שתוצהר בהמשך הקובץ. ההצהרה נעשית על ידי כתיבת תווית בתחילת הנחית 'data.'או 'string.', או בתחילת הוראה של התוכנית, או באמצעות אופרנד של הנחית 'extern.'	נתונה ההגדרה: x: .data 23 אזי ההוראה: dec x מקטינה ב-1 את תוכן המילה שבכתובת x בזיכרון (ה"משתנה" x).

2	מיעון אינדקס קבוע	שיטת מיעון זו משמשת לגישה לאיבר במערך לפי אינדקס. המערך נמצא בזיכרון. כל איבר במערך הוא בגודל מילה.
		
		בשיטת מיעון זו קיימות בקידוד ההוראה שתי מילות-מידע נוספות. המילה הנוספת הראשונה מכילה את כתובת התחלת המערך. המילה הנוספת השניה 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 במיעון ישיר, ובמילת-המידע השניה כמו במיעון מיידי).
			האופרנד מורכב מתווית המציינת את כתובת התחלת המערך, ולאחריה בסוגריים מרובעים האינדקס במערך אליו רוצים לפנות.
			
			האינדקס יכול להינתן ע"י קבוע מספרי, או ע"י שם של מאקרו שהוגדר בעזרת define. (יוסבר בהמשך). האינדקסים במערך מתחילים מ-0.
				נתונה ההגדרה:
				x: .data 23,25,19,30
				
				אזי ההוראה:
				mov x[2],r2
				תעתיק את המספר 19 הנמצא באינדקס 2 במערך x אל האוגר r2.
				
				דוגמה נוספת:
				נתונה הגדרת המערך x לעיל, וכן הגדרת המאקרו:
				.define k=1
				
				אזי ההוראה:
				mov r2,x[k]
				תעתיק את תוכן האוגר r2 אל המילה באינדקס 1 במערך x (נדרס התוכן הקודם 25).

ערך	שיטת מיעון	תוכן המילה נוספת	אופן הכתיבה	דוגמה
3	מיעון אוגר ישיר	האופרנד הוא אוגר. אם האוגר משמש כאופרנד יעד, מילת-מידע נוספת של הפקודה תקודד בסיביות 2-4 את מספרו של האוגר. ואילו אם האוגר משמש כאופרנד מקור, מספר האוגר יקודד בסיביות 5-7 של מילת-המידע. אם בפקודה יש שני אופרנדים ושניהם בשיטת מיעון אוגר ישיר, הם יחלקו מילת-מידע אחת משותפת, כאשר הסיביות 2-4 הן עבור אוגר היעד, והסיביות 5-7 הן עבור אוגר המקור. למילת-המידע מתווספות זוג סיביות של השדה A,R,E (הערך של שדה זה הוא תמיד 00 עבור מיעון אוגר ישיר). סיביות אחרות במילת-המידע שאינן בשימוש יכילו 0.	האופרנד הוא שם של אוגר.	mov r1,r2 בדוגמה זו, ההוראה מעתיקה את תוכן אוגר r1 אל אוגר r2. בדוגמה זו, שני האופרנדים הם בשיטת מיעון אוגר ישיר, ולכן יחלקו מילת-מידע נוספת אחת משותפת.

<u>הערה</u>: מותר להשתמש בתווית עוד לפני שמגדירים אותה, בתנאי שהתווית אכן מוגדרת במקום כלשהו בהמשך הקובץ.

<u>מפרט הוראות המכונה</u>:

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

קבוצת ההוראות הראשונה:
אלו הן הוראות הדורשות שני אופרנדים.

ההוראות השייכות 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 שבכתובת A בזיכרון) אל אוגר r1.
cmp	מבצעת "השוואה" בין שני האופרנדים שלה. אופן ההשוואה: תוכן אופרנד היעד (השני) מופחת מתוכן אופרנד המקור (הראשון), ללא שמירת תוצאת החיסור. פעולת החיסור מעדכנת דגל בשם Z ("דגל האפס") באוגר הסטטוס (PSW).	cmp A, r1	אם תוכן המשתנה A זהה לתוכנו של אוגר r1 אזי דגל האפס, Z, באוגר הסטטוס (PSW) יודלק, אחרת הדגל יאופס.
add	אופרנד היעד (השני) מקבל את תוצאת החיבור של אופרנד המקור (הראשון) והיעד (השני).	add A, r0	אוגר r0 מקבל את תוצאת החיבור של תוכן המשתנה A ותוכנו הנוכחי של r0.

הוראה	הסבר הפעולה	דוגמה	הסבר הדוגמה
sub	אופרנד היעד (השני) מקבל את תוצאת החיסור של אופרנד המקור (הראשון) מאופרנד היעד (השני).	sub #3, r1	אוגר r1 מקבל את תוצאת החיסור של הערך 3 מתוכנו הנוכחי של האוגר r1.
lea	lea הוא קיצור (ראשי תיבות) של load effective address. פעולה זו מציבה את המען בזיכרון המיוצג על ידי התווית שבאופרנד הראשון (המקור), אל אופרנד היעד (האופרנד השני).	lea HELLO, r1	המען שמייצגת התווית HELLO מוצב לאוגר r1.

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

ההוראות השייכות 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 ולהיפך: 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
jmp	קפיצה (הסתעפות) בלתי מותנית אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כלומר, בעת ביצוע ההוראה, מצביע התוכנית (PC) יקבל את ערך אופרנד היעד.	jmp LINE	PC ← LINE מצביע התכנית מקבל את המען המיוצג על ידי התווית LINE, ולפיכך הפקודה הבאה שתתבצע תהיה במען זה.
bne	bne הוא קיצור (ראשי תיבות) של: branch if not equal (to zero). זוהי הוראת הסתעפות מותנית. מצביע התוכנית (PC) יקבל את ערך אופרנד היעד אם ערכו של הדגל Z באוגר הסטטוס (PSW) הינו 0. כזכור, הדגל Z נקבע בפקודת cmp.	bne LINE	אם ערך הדגל Z באוגר הסטטוס (PSW) הינו 0, אזי: PC ← LINE
red	קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מהקלט ייכנס לאוגרr1.
prn	הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn r1	התו אשר קוד ה-ascii שלו נמצא באוגר r1 יודפס לפלט הסטנדרטי.
Jsr	קריאה לשגרה (סברוטינה), מצביע התוכנית (PC) הנוכחי נדחף לתוך המחסנית שבזיכרון המחשב, והאופרנד מוכנס ל-PC.	jsr FUNC	push(PC) PC ← FUNC

קבוצת ההוראות השלישית:
אלו הן הוראות ללא אופרנדים. קידוד ההוראה מורכב ממילה אחת בלבד. במקרה זה, השדות של אופרנד המקור ואופרנד היעד אינם רלוונטים (כי אין אופרנדים), ויכילו 0.

ההוראות השייכות לקבוצה זו הן : rts, stop.

הוראה	הסבר הפעולה	דוגמה	הסבר הדוגמה
Rts	חזרה משיגרה. הערך שנמצא בראש המחסנית של המחשב מוצא מן המחסנית, ומוכנס אל מצביע התוכנית (PC).	rts	PC ← pop()
Stop	עצירת ריצת התוכנית.	stop	התכנית עוצרת

<u>מבנה שפת האסמבלי :</u>

שפת האסמבלי בנויה ממשפטים (statements). קובץ בשפת אסמבלי מורכב משורות המכילות משפטים של השפה, כאשר כל משפט מופיע <u>בשורה נפרדת</u>. כלומר התו המפריד בין משפט למשפט בקובץ הינו התו '\n' (שורה חדשה).

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

ישנם חמישה סוגי משפטים (שורות) בשפת האסמבלי, והם :

סוג המשפט	הסבר כללי
משפט ריק	זוהי שורה המכילה אך ורק תווים לבנים (whitespace), כלומר רק את התווים ' ' ו- '\t' (רווחים וטאבים). ייתכן ובשורה אין אף תו (למעט התו \n), כלומר השורה ריקה.
משפט הערה	זוהי שורה בה התו הראשון הינו ';' (נקודה פסיק). על האסמבלר להתעלם לחלוטין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמבלר מה עליו לעשות כשהוא פועל על תכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצאת 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 משם של הוראה שעל המעבד לבצע, ותיאור האופרנדים של ההוראה.
משפט מאקרו	זהו משפט באמצעותו ניתן להגדיר שם סימבולי המייצג קבוע מספרי. במהלך קידוד התוכנית, בכל מקום בקוד בו מופיע השם, הוא יוחלף בקבוע המספרי. משפט זה לכשעצמו אינו מייצר קוד ואינו מקצה זיכרון.

כעת נפרט יותר לגבי סוגי המשפטים השונים.

<u>משפט הנחיה:</u>

משפט הנחיה הוא בעל המבנה הבא :

בתחילת המשפט יכולה להופיע תווית (label). התווית חייבת להיות בתחביר חוקי (התחביר של תווית חוקית יתואר בהמשך). התווית היא אופציונלית.

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

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

יש לשים לב: למילים בקוד המכונה הנוצרות <u>ממשפט הנחיה</u> לא מצורף השדה A,R,E, והקידוד ממלא את כל 14 הסיביות של המילה.

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

1. ההנחיה '.data'

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

.data 7, −57, +17, 9

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

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

כלומר אם נכתוב:

XYZ: .data 7, –57, +17, 9

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

אם נכתוב בתכנית הוראת מכונה:

mov XYZ, r1

אזי בזמן ריצת התכנית יוכנס לאוגר r1 הערך 7.

ואילו ההוראה:

lea XYZ, r1

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

2. ההנחיה '.string'

להנחיה '.string' פרמטר אחד, שהוא מחרוזת חוקית. תווי המחרוזת מקודדים לפי ערכי ה-ascii המתאימים, ומוכנסים אל תמונת הנתונים לפי סדרם, כל תו במילה נפרדת. בסוף המחרוזת יתווסף התו '\0' (הערך המספרי 0), המסמן את סוף המחרוזת. מונה הנתונים של האסמבלר יקודם בהתאם לאורך המחרוזת (בתוספת מקום אחד עבור התו המסיים). אם בשורת ההנחיה מוגדרת תווית, אזי תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום) ומוכנסת אל טבלת הסמלים, בדומה כפי שנעשה עבור '.data' (כלומר ערך התווית יהיה הכתובת בזיכרון שבה מתחילה המחרוזת).

לדוגמה, ההנחיה:

STR: .string "abcdef"

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

3. ההנחיה '.entry'

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

לדוגמה, השורות:

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

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

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

4. ההנחיה '.extern'

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

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

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

```
.extern     HELLO
```

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

<u>משפט הוראה</u>:

משפט הוראה מורכב מהחלקים הבאים, לפי הסדר:
1. תווית אופציונלית.
2. שם הפעולה.
3. אופרנדים בהתאם לסוג הפעולה (בין 0 ל-2 אופרנדים).

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

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

לאחר שם הפעולה יכולים להופיע אופרנדים (אחד או שניים), בהתאם לסוג הפעולה.

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזה בתו ',' (פסיק). בדומה להנחיה '.data', **לא חייבת להיות הצמדה של האופרנדים לפסיק או לשם הפעולה באופן כלשהו**. כל כמות של רווחים ו/או טאבים בין האופרנדים לפסיק, או בין שם הפעולה לאופרנד הראשון, היא חוקית.

למשפט הוראה עם שני אופרנדים המבנה הבא:

תווית-אופציונלית: שם-הפעולה אופרנד-יעד, אופרנד-מקור

לדוגמה:

```
HELLO:          add   r7, B
```

למשפט הוראה עם אופרנד אחד המבנה הבא:

אופרנד שם-הפעולה :תווית-אופציונלית

לדוגמה:

HELLO: bne XYZ

למשפט הוראה ללא אופרנדים המבנה הבא:

שם-הפעולה :תווית-אופציונלית

לדוגמה:

END: stop

<u>משפט מאקרו:</u>

משפט מאקרו הוא בעל המבנה הבא:

.define קבוע-מספרי = שם-המאקרו

לדוגמה:

.define len = 4
.define init = -3

הרעיון הוא לייצג קבוע מספרי באמצעות שם סימבולי. בכל מקום בתוכנית בו מופיע שם של מאקרו, האסמבלר יחליף בקידוד הפקודה לקוד מכונה את השם בקבוע המספרי אליו הוגדר.

המילה השמורה '.define' נתונה באותיות קטנות בלבד.
התחביר של שם המאקרו זהה לתחביר של תווית. אסור להגדיר את אותו שם מאקרו יותר מפעם אחת. כמו כן אותו סמל לא יכול לשמש הן כשם של מאקרו והן כתווית באותה תכנית.
מילים שמורות של שפת האסמבלי (שם של אוגר, שם של הוראת מכונה או שם של הנחיה) אינן יכולות לשמש שם של מאקרו.
הקבוע המספרי הוא שלם בבסיס עשרוני.
בין שם המאקרו לקבוע מפריד התו '='. מותרים תווים לבנים בשני צידי התו.

מאקרו חייב להיות מוגדר <u>לפני השימוש הראשון</u> בו.
<u>אסור להגדיר תווית</u> בשורה שהיא משפט מאקרו.

ניתן להשתמש בשם המאקרו בכל מקום בתכנית האסמבלי בו יכול להופיע קבוע מספרי, כלומר: **אינדקס בשיטת מיעון אינדקס ישיר**, או **ערך בשיטת מיעון מידי**, או **אופרנד של הנחית data**.

לדוגמה, בהינתן הגדרות המאקרו לעיל, אזי ההוראה:

mov x[len], r3

תעתיק את האיבר באינדקס 4 במערך x אל אוגר r3.

וההוראה:

mov #init, r2

תציב את הערך המידי 3- אל האוגר r2

כמו כן, ההנחיה:

.data len

תקצה מילה בזיכרון עם ערך התחלתי 4.

אפיון השדות במשפטים של שפת האסמבלי

תווית:

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

הגדרה של תווית מסתיימת בתו ':' (נקודתיים). תו זה אינו מהווה חלק מהתווית, אלא רק סימן המציין את סוף ההגדרה.

אסור שאותה תווית תוגדר יותר מפעם אחת (כמובן בשורות שונות). אותיות קטנות וגדולות נחשבות שונות זו מזו. כמו כן, אסור שאותו סמל ישמש הן כתווית והן כשם של מאקרו.

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hEllo:
x:
He78902:

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

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

מספר:

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

מחרוזת:

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

תפקיד השדה A,R,E בקוד המכונה

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

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

האות 'A' (קיצור של absolute) באה לציין שתוכן המילה אינו תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התכנית בעת ביצועה (למשל מילה המכילה אופרנד מידי).
במקרה זה שתי הסיביות הימניות יכילו את הערך 00.

האות 'R' (קיצור של relocatable) באה לציין שתוכן המילה תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התכנית בעת ביצועה (למשל מילה המכילה כתובת של תווית המוגדרת בקובץ המקור).
במקרה זה שתי הסיביות הימניות יכילו את הערך 10.

האות 'E' (קיצור של external) באה לציין שתוכן המילה תלוי בערכו של סמל חיצוני (external) (למשל מילה המכילה כתובת של תווית חיצונית, כלומר תווית שאינה מוגדרת בקובץ המקור הנוכחי). במקרה זה שתי הסיביות הימניות יכילו את הערך 01.

אסמבלר עם שני מעברים

כאשר מקבל האסמבלר קוד בשפת אסמבלי לתרגום, עליו לבצע שתי משימות עיקריות לצורך הכנת הקוד הבינארי: הראשונה היא לזהות ולתרגם את שמות הפעולות, והשנייה היא לקבוע מענים לכל הסמלים המופיעים בתוכנית.

לדוגמה: האסמבלר מקבל את הקוד הבא:

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

עליו להחליף את שמות הפעולות mov, jmp, prn, sub, cmp, inc, bne, stop בקוד הבינארי השקול להם במודל המחשב שהגדרנו.

כמו כן, על האסמבלר להחליף את הסמלים K,STR, LIST, L1, MAIN, LOOP, END במענים של המקומות בזיכרון שם נמצאים כל נתון או הוראה בהתאמה.

בנוסף, על האסמבלר להחליף את שמות המאקרוים len ו- sz בקבועים המספריים שהם ערכי המאקרוים בהתאמה, בכל מקום בו הם מופיעים.

אנו מניחים שקוד המכונה של התכנית (הוראות ונתונים) ייבנה כך שיתאים לטעינה בזיכרון החל ממען 100 (בבסיס עשרוני). מתקבל התרגום הבא לקוד מכונה בינארי:

Decimal Address	**Source Code**	**Explanation**	**Binary Machine Code**
0100 **0101** **0102** **0103**	MAIN: mov r3, LIST[sz]	First word of instruction Source register 3 Address of label LIST (integer array) Value of macro sz (index 2)	**00000000111000** **00000001100000** **00001000010010** **00000000001000**
0104 **0105**	LOOP: jmp L1	 Address of label L1	**00001001000100** **00000111100010**
0106 **0107**	prn #-5	 Immediate value -5	**00001100000000** **11111111101100**
0108 **0109** **0110** **0111** **0112**	mov STR[5], STR[2]	 Address of label STR (string) Index 5 Address of label STR Index 2	**00000000101000** **00000111110010** **00000000010100** **00000111110010** **00000000001000**

Decimal Address	Source Code	Explanation	Binary Machine Code
0113 **0114**	sub r1, r4	 Source register 1 and target register 4	**00000011111100** **00000000110000**
0115 **0116** **0117**	cmp r3, #sz	 Source register 3 Value of macro sz (immediate #2)	**00000001110000** **00000001100000** **00000000001000**
0118 **0119**	bne END	 Address of label END	**00001010000100** **00000111110010**
0120 **0121**	L1: inc K	 Address of label K (integer)	**00000111000100** **00001000011110**
0122 **0123**	bne LOOP	 Address of label LOOP	**00001010000100** **00000110100010**
0124	END: stop		**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 **0133** **0134**	LIST: .data 6, -9, len	Integer 6 (first in array of 3 words) Integer -9 Value of macro len (integer 4)	**00000000000110** **11111111110111** **00000000000100**
0135	K: .data 22	Integer 22 (single word)	**00000000010110**

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

כדי לעשות המרה לבינארי של אופרנדים שהם מענים סמליים (תוויות), יש צורך לבנות טבלה דומה. אולם בהבדל מהקודים של הפעולות, הידועים מראש, הרי המענים בזיכרון עבור הסמלים שבשימוש התכנית אינם ידועים, עד אשר תוכנית המקור נסרקה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד לעיל, האסמבלר אינו יכול לדעת שהסמל END אמור להיות משויך למען 124 (עשרוני), ושהסמל K אמור להיות משויך למען 135, אלא רק לאחר שנקראו כל שורות התכנית.

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

במעבר הראשון נבנית טבלת סמלים בזיכרון, ובה לכל סמל שבתכנית המקור משויך ערך מספרי, שהוא מען בזיכרון או ערך קבוע של מאקרו. בדוגמה לעיל, טבלת הסמלים לאחר מעבר ראשון היא:

סמל	ערך (בבסיס 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. בתחילת המעבר השני צריכים הערכים של הסמלים להיות כבר ידועים.

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

המעבר הראשון

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

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

על האסמבלר לסרוק את שורת ההוראה בשלמותה, ולהחליט לגבי הקידוד לפי האופרנדים. בדרך כלל מתחלק הקידוד לשדה של שם הפעולה, ושדות נוספים המכילים מידע לגבי שיטות המיעון.

במחשב שלנו קיימת גמישות לגבי שיטת המיעון של כל אחד מהאופרנדים בנפרד. <u>הערה</u>: דבר זה לא מחייב לגבי כל מחשב. ישנם מחשבים בהם, למשל, כל הפקודות הן בעלות אופרנד יחיד (והפעולות מתבצעות על אופרנד זה ועל אוגר קבוע). יש גם מחשבים עם פקודות של שלשה אופרנדים (כאשר האופרנד השלישי משמש לאחסון תוצאת הפעולה), ועוד אפשרויות אחרות.

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

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

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

כאשר מגיע האסמבלר לשורת ההסתעפות (bne A), הוא טרם נתקל בהגדרת התווית A וכמובן לא נתן לה מען, ולכן אינו יכול להחליף את הסמל A (האופרנד של ההוראה bne) במענו בזיכרון. נראה בהמשך כיצד נפתרת בעיה זו.

בכל מקרה, תמיד אפשר לבנות במעבר הראשון את הקוד הבינארי המלא של המילה הראשונה של כל הוראה, וכן את הקוד הבינארי של כל הנתונים (המתקבלים מההנחיות data., string.).

המעבר השני

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

לשם כך עובר האסמבלר שנית על כל קובץ המקור, ומעדכן את קוד המכונה של האופרנדים המשתמשים בסמלים, באמצעות ערכי הסמלים מטבלת הסמלים.
זהו המעבר השני, ובסופו תהיה התוכנית מתורגמת בשלמותה לקוד מכונה.

הפרדת הוראות ונתונים

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

אחת הסכנות הטמונות באי הפרדת ההוראות מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית בתכנית, לנסות "לבצע" את הנתונים כאילו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום תופעה 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.

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

גילוי שגיאות בתכנית המקור

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

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

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

על כל הודעת שגיאה לציין גם את מספר השורה בתכנית המקור בה זוהתה השגיאה.

לתשומת לב: האסמבלר אינו עוצר את פעולתו אחרי שנמצאה השגיאה הראשונה, אלא ממשיך לעבור על הקלט כדי לגלות שגיאות נוספות, ככל שישנן. כמובן שאין כל טעם לייצר את קבצי הפלט אם נתגלו שגיאות (ממילא אי אפשר להשלים את קוד המכונה).

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

שם פעולה	שיטות מיעון חוקיות עבור אופרנד מקור	שיטות מיעון חוקיות עבור אופרנד יעד
mov	0,1,2,3	1,2,3
cmp	0,1,2,3	0,1,2,3
add	0,1,2,3	1,2,3
sub	0,1,2,3	1,2,3
not	אין אופרנד מקור	1,2,3
clr	אין אופרנד מקור	1,2,3
lea	1,2	1,2,3
inc	אין אופרנד מקור	1,2,3

שם פעולה	שיטות מיעון חוקיות עבור אופרנד מקור	שיטות מיעון חוקיות עבור אופרנד יעד
dec	אין אופרנד מקור	1,2,3
jmp	אין אופרנד מקור	1,3
bne	אין אופרנד מקור	1,3
red	אין אופרנד מקור	1,2,3
prn	אין אופרנד מקור	0,1,2,3
jsr	אין אופרנד מקור	1,3
rts	אין אופרנד מקור	אין אופרנד יעד
stop	אין אופרנד מקור	אין אופרנד יעד

אלגוריתם שלדי של האסמבלר

לחידוד ההבנה של תהליך העבודה של האסמבלר, נציג להלן אלגוריתם שלדי למעבר הראשון ולמעבר השני. לתשומת לב: אין חובה להשתמש דווקא באלגוריתם זה.

אנו מחלקים את קוד המכונה לשני אזורים, אזור ההוראות (code) ואזור הנתונים (data). לכל אזור יש מונה משלו, ונסמנם IC (מונה ההוראות - Instruction-Counter) ו-DC (מונה הנתונים - Data-Counter). **נבנה את קוד המכונה כך שיתאים לטעינה לזיכרון החל מכתובת 100**.

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

בכל מעבר מתחילים לקרוא את קובץ המקור מהתחלה.

מעבר ראשון

1. אתחל DC ← 0, IC ← 0.
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-16.
3. האם זוהי הגדרת מאקרו? אם לא, עבור ל-5.
4. הכנס את שם המאקרו לטבלת הסמלים עם המאפיין macro. ערכו יהיה כפי שמופיע בהגדרה. (אם הסמל כבר נמצא בטבלה, יש להודיע על שגיאה). חזור ל-2.
5. האם השדה הראשון בשורה הוא סמל? אם לא, עבור ל-7.
6. הדלק דגל "יש הגדרת סמל".
7. האם זוהי הנחיה לאחסון נתונים, כלומר, האם הנחית data. או string.? אם לא, עבור ל-10.
8. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם המאפיין data. ערכו יהיה DC. (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
9. זהה את סוג הנתונים, קודד אותם בזיכרון, ועדכן את מונה הנתונים DC בהתאם לאורכם. אם זוהי הנחית data., ויש בה נתון שהוא סמל, בדוק שהסמל מופיע בטבלה עם המאפיין macro, והשתמש בערכו. (אם הסמל אינו בטבלה או לא מאופיין כ- macro, יש להודיע על שגיאה). חזור ל-2.
10. האם זו הנחית extern. או הנחית entry. ? אם לא, עבור ל-12.
11. האם זוהי הנחית extern. ? אם כן, הכנס כל סמל (אחד או יותר) המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים ללא ערך, עם המאפיין external. חזור ל-2.
12. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם המאפיין code. ערכו יהיה IC+100 (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
13. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא – הודע על שגיאה בשם ההוראה.
14. נתח את מבנה האופרנדים של ההוראה וחשב את L. בנה כעת את הקוד הבינארי של המילה הראשונה של ההוראה.
15. עדכן IC ← IC + L, וחזור ל-2.
16. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר הראשון, עצור כאן.
17. עדכן בטבלת הסמלים את ערכו של כל סמל המאופיין כ- data , ע"י הוספת הערך IC+100 (ראה הסבר בהמשך).
18. התחל מעבר שני.

מעבר שני

1. אתחל IC ← 0
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 9.
3. אם השדה הראשון הוא סמל, דלג עליו.
4. האם זוהי הנחית data. או string. או extern. ? אם כן, חזור ל-2.
5. האם זוהי הנחית entry. ? אם לא, עבור ל-7.
6. סמן בטבלת הסמלים את הסמלים המתאימים במאפיין entry. חזור ל-2.
7. השלם את קידוד האופרנדים החל מהמילה השניה בקוד הבינארי של ההוראה, בהתאם לשיטת המיעון. אם אופרנד מכיל סמל, מצא את הערך בטבלת הסמלים (אם הסמל לא נמצא בטבלה, יש להודיע על שגיאה).
8. עדכן IC ← IC + L, וחזור ל-2.
9. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר השני, עצור כאן.
10. בנה את קבצי הפלט (פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שהצגנו קודם:

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

נבצע עתה מעבר ראשון על הקוד הנתון. נבנה את טבלת הסמלים. כמו כן, נבצע במעבר זה גם את קידוד כל הנתונים, וקידוד של המילה הראשונה של כל הוראה. כמו כן ניתן לקודד מילים נוספות של כל ההוראה, כל עוד קידוד זה אינו תלוי בכתובת של תווית. את החלקים שעדיין לא מקודדים במעבר זה, נשאיר כמות שהם (מסומנים ב- ? בדוגמה להלן).

נשים לב שקוד המכונה נבנה לטעינה לזיכרון החל מהמען 100 (בבסיס עשרוני).

Decimal Address	**Source Code**	**Explanation**	**Binary Machine Code**
0100 **0101** **0102** **0103**	MAIN: mov r3, LIST[sz]	First word of instruction Source register 3 Address of label LIST (integer array) Value of macro sz (index 2)	**00000000111000** **00000001100000** **?** **00000000001000**
0104 **0105**	LOOP: jmp L1	 Address of label L1	**00001001000100** **?**
0106 **0107**	prn #-5	 Immediate value -5	**00001100000000** **11111111101100**
0108 **0109** **0110** **0111** **0112**	mov STR[5], STR[2]	 Address of label STR (string) Index 5 Address of label STR Index 2	**00000000101000** **?** **00000000010100** **?** **00000000001000**

Decimal Address	Source Code	Explanation	Binary Machine Code
0113 **0114**	sub r1, r4	Source register 1 and target register 4	**00000011111100** **00000000110000**
0115 **0116** **0117**	cmp r3, #sz	Source register 3 Value of macro sz (immediate #2)	**00000001110000** **00000001100000** **00000000001000**
0118 **0119**	bne END	Address of label END	**00001010000100** **?**
0120 **0121**	L1: inc K	Address of label K (integer)	**00000111000100** **?**
0122 **0123**	bne LOOP	Address of label LOOP	**00001010000100** **?**
0124	END: stop		**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 **0133** **0134**	LIST: .data 6, -9, len	Integer 6 (first in array of 3 words) Integer -9 Value of macro len (integer 4)	**00000000000110** **11111111110111** **00000000000100**
0135	K: .data 22	Integer 22 (single word)	**00000000010110**

טבלת הסמלים:

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

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

Decimal Address	Source Code	Binary Machine Code
0100 **0101** **0102** **0103**	MAIN: mov r3, LIST[sz]	**00000000111000** **00000001100000** **00001000010010** **00000000001000**
0104 **0105**	LOOP: jmp L1	**00001001000100** **00000111100010**
0106 **0107**	prn #-5	**00001100000000** **11111111101100**

Decimal Address	Source Code	Binary Machine Code
0108 0109 0110 0111 0112	mov STR[5], STR[2]	00000000101000 00000111110010 00000000010100 00000111110010 00000000001000
0113 0114	sub r1, r4	00000011111100 00000000110000
0115 0116 0117	cmp r3, #sz	00000001110000 00000001100000 00000000001000
0118 0119	bne END	00001010000100 00000111110010
0120 0121	L1: inc K	00000111000100 00001000011110
0122 0123	bne LOOP	00001010000100 00000110100010
0124	END: stop	00001111000000
0125	STR: .string "abcdef"	00000001100001
0126		00000001100010
0127		00000001100011
0128		00000001100100
0129		00000001100101
0130		00000001100110
0131		00000000000000
0132 0133 0134	LIST: .data 6, -9, len	00000000000110 11111111110111 00000000000100
0135	K: .data 22	00000000010110

לאחר סיום עבודת האסמבלר, קבצי הפלט מועברים להמשך עיבוד בשלבי הקישור והטעינה.
לא נדון כאן באופן עבודת שלבי הקישור/טעינה (כאמור, אלה אינם למימוש בפרויקט זה).

קבצי קלט ופלט של האסמבלר

בהפעלה של האסמבלר, יש להעביר אליו באמצעות ארגומנטים של שורת הפקודה (command line arguments) רשימה של שמות קבצי מקור (אחד או יותר). אלו הם קבצי טקסט, ובהם תכניות בתחביר של שפת האסמבלי, שהוגדרה למעלה. האסמבלר פועל על כל קובץ מקור בנפרד, ויוצר עבורו קובץ מטרה (object) נפרד המכיל את קוד המכונה. כמו כן האסמבלר יוצר קובץ externals עבור כל קובץ מקור בו יש הצהרות על סמלים חיצוניים, וכן קובץ entries עבור כל קובץ מקור בו יש הצהרות על סמלים כנקודות כניסה.

שמות קבצי המקור חייבים להיות עם הסיומת ".as". למשל, השמות x.as , y.as, ו-hello.as הם שמות חוקיים. העברת שמות הקבצים הללו כארגומנטים לאסמבלר נעשית <u>ללא ציון הסיומת</u>.
לדוגמה: נניח שתוכנית האסמבלר שלנו נקראת assembler, אזי שורת הפקודה הבאה:

assembler x y hello

תפעיל את האסמבלר על הקבצים: x.as, y.as, hello.as.

האסמבלר מגדיר שמות לקבצי הפלט המבוססים על שם קובץ הקלט, כפי שהופיע בשורת הפקודה, בתוספת סיומת שונה. הסיומת ".ob" עבור קובץ ה-object, הסיומת ".ent" עבור קובץ ה-entries, והסיומת ".ext" עבור קובץ ה-externals.
לדוגמה, בהפעלת האסמבלר באמצעות שורת הפקודה: assembler x
יווצר קובץ הפלט x.ob, וכן קבצי הפלט x.ext ו- x.ent ככל שיש entries/externals בקובץ המקור.

<u>מבנה כל קובץ פלט יתואר בהמשך</u>.

אופן פעולת האסמבלר

נרחיב כאן על אופן פעולת האסמבלר, בנוסף לאלגוריתם השלדי שניתן לעיל.

האסמבלר מחזיק שני מערכים, שייקראו להלן מערך ההוראות ומערך הנתונים. מערכים אלו נותנים למעשה תמונה של זיכרון המכונה (כל איבר במערך הוא בגודל מילה של המכונה, כלומר 14 סיביות). במערך ההוראות מכניס האסמבלר את הקידוד של הוראות המכונה שנקראו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמבלר את קידוד הנתונים שנקראו מקובץ המקור (שורות הנחיה מסוג 'data.' ו- 'string.').

לאסמבלר יש שני מונים: מונה ההוראות (IC) ומונה הנתונים (DC). מונים אלו מצביעים על המקום הבא הפנוי במערכים לעיל, בהתאמה. כשמתחיל האסמבלר לעבור על קובץ מקור, שני מונים אלו מאופסים.

בנוסף יש לאסמבלר טבלה, אשר בה נאספות כל התוויות בהן נתקל האסמבלר במהלך המעבר על הקובץ. לטבלה זו קוראים טבלת סמלים (symbol-table). לכל סמל נשמרים שמו, ערכו המספרי, ומאפיינים שונים שצוינו קודם, כגון המיקום (data או code או macro), או אופן העדכון (external או relocatable).

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

1. שורה ריקה או שורת הערה: האסמבלר מתעלם מהשורה ועובר לשורה הבאה.
2. שורת מאקרו: האסמבלר מכניס את שם המאקרו לטבלת הסמלים עם המאפיין macro.
3. שורת הוראה:

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

אם האסמבלר גילה בשורת ההוראה גם הגדרה של תווית, אזי התווית מוכנסת אל טבלת הסמלים. ערך התווית הוא IC+100, והמאפיינים הם code ו- relocatable.

כעת האסמבלר קובע לכל אופרנד את ערכו באופן הבא:

- אם זה אוגר – האופרנד הוא מספר האוגר.
- אם זו תווית (מיעון ישיר) – האופרנד הוא ערך התווית כפי שמופיע בטבלת הסמלים (ייתכן והסמל טרם נמצא בטבלת הסמלים).
- אם זה התו # ואחריו מספר או שם של מאקרו (מיעון מידי) – האופרנד הוא המספר עצמו.
- אם זו שיטת מיעון אחרת – ערכו של האופרנד נקבע לפי המפרט של שיטת המיעון (ראה תאור שיטות המיעון לעיל)

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

אם זוהי הוראה בעלת אופרנדים (אחד או שניים), האסמבלר "משריין" מקום במערך ההוראות עבור מילות-המידע הנוספות הנדרשות בהוראה זו, ומקדם את IC בהתאם. כאשר אחד או שני האופרנדים הם בשיטת מיעון אוגר-ישיר או מיידי, האסמבלר מקודד גם את המילים הנוספות הרלוונטיות במערך ההוראות.

נזכור שאם יש רק אופרנד אחד, כלומר אין אופרנד מקור, הסיביות של שיטת המיעון של אופרנד המקור יכילו תמיד 0, מכיוון שאינן רלוונטיות. בדומה, אם זוהי הוראה ללא אופרנדים (rts, stop), אזי הסיביות של שיטות המיעון של שני האופרנדים יכילו 0.

4. שורת הנחיה:

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

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

אם בשורה 'data.' מוגדרת גם תווית, אזי תווית זו מוכנסת לטבלת הסמלים. ערך התווית הוא ערך מונה הנתונים DC <u>שלפני</u> הכנסת המספרים למערך. המאפיינים של התווית הם data ו- relocatable.

II. 'string.'
הטיפול ב-'string.' דומה ל- 'data.', אלא שקודי ה-ascii של התווים הם אלו המוכנסים אל מערך הנתונים (כל תו בכניסה נפרדת). לאחר מכן מוכנס הערך 0 (המציין סוף מחרוזת) אל מערך הנתונים. מונה הנתונים מקודם באורך המחרוזת + 1 (גם האפס בסוף המחרוזת תופס מקום).

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

III. 'entry.'
זוהי בקשה לאסמבלר להכניס את התווית המופיעה כאופרנד של 'entry.' אל קובץ ה-entries.
האסמבלר רושם את הבקשה ובסיום העבודה, התווית הנ"ל תירשם בקובץ ה-entries.

IV. 'extern.'
זוהי הצהרה על סמל (תווית) המוגדר בקובץ אחר, ואשר קטע האסמבלי בקובץ הנוכחי עושה בו שימוש. האסמבלר מכניס את הסמל אל טבלת הסמלים, עם הערך 0 (הערך האמיתי לא ידוע, וייקבע רק בשלב הקישור), והמאפיין external. לא ידוע באיזה קובץ נמצאת הגדרת הסמל (וגם אין זה משנה עבור האסמבלר).

<u>יש לשים לב:</u> בהוראה או בהנחיה אפשר להשתמש בשם של סמל אשר ההצהרה עליו ניתנת בהמשך הקובץ (אם באופן ישיר על ידי הגדרת תווית, ואם באופן עקיף על ידי הנחית extern).

<u>בסוף המעבר הראשון</u>, האסמבלר מעדכן בטבלת הסמלים כל סמל המאופיין כ- data, על ידי הוספת IC+100 (עשרוני) לערכו של הסמל. הסיבה לכך היא שבתמונה הכוללת של קוד המכונה, הנתונים מופרדים מההוראות, וכל הנתונים נדרשים להופיע אחרי כל ההוראות. סמל מסוג data הוא למעשה תווית באזור הנתונים, והעדכון מוסיף לערך הסמל (כלומר לכתובתו בזיכרון) את האורך הכולל של קידוד כל ההוראות, בתוספת כתובת התחלת הטעינה של הקוד, שהיא 100.

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

במעבר השני, האסמבלר מקודד באמצעות טבלת הסמלים את כל המילים במערך ההוראות שטרם קודדו במעבר הראשון. אלו הן מילים שצריכות להכיל כתובות של תוויות (שדה ה- A,R,E במילים אלה יהיה 10 או 01).

פורמט קובץ ה- object

כזכור, האסמבלר בונה את תמונת הזיכרון של התכנית כך שקידוד ההוראה הראשונה מקובץ האסמבלי ייכנס למען 100 (בבסיס עשרוני) בזיכרון, קידוד ההוראה השניה יכנס למען העוקב אחרי ההוראה הראשונה (תלוי במספר המילים של ההוראה הראשונה), וכך הלאה עד להוראה האחרונה.

<u>אחרי ההוראה האחרונה</u>, מכניסים לתמונת הזיכרון את קידוד איזור הנתונים, שנבנה על ידי ההנחיות 'string. 'data.'. זוהי הסיבה בגללה יש לעדכן בטבלת הסמלים, בסוף המעבר הראשון, את ערכי הסמלים עם המאפיין data.

נשים לב שבגמר המעבר השני, אופרנד של הוראה המתייחס לסמל שהוגדר באותו קובץ, כבר מקודד כך שיצביע על המקום המתאים בתמונת הזיכרון שבונה האסמבלר.

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

השורה הראשונה בקובץ ה- object היא "כותרת", המכילה שני מספרים (בבסיס עשרוני): הראשון הוא האורך הכולל של קטע ההוראות (במילות זיכרון), והשני הוא האורך הכולל של קטע הנתונים (במילות זיכרון). בין שני המספרים יש רווח אחד.

השורות הבאות בקובץ מכילות את תמונת הזיכרון. בכל שורה שני ערכים: כתובת של מילה בזיכרון, ותוכן המילה. הכתובת תירשם בבסיס עשרוני בארבע ספרות (כולל אפסים מובילים). תוכן המילה יירשם בבסיס 4 "מיוחד" (ראה להלן) בשבע ספרות (כולל אפסים מובילים). בין שני הערכים בכל שורה יפריד רווח אחד.

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

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

פורמט קובץ ה-entries

קובץ ה-entries בנוי משורות טקסט. כל שורה מכילה שם של סמל שהוגדר כ- entry ואת ערכו, כפי שנמצא בטבלת הסמלים. הערכים מיוצגים בבסיס עשרוני.

פורמט קובץ ה- externals

קובץ ה-externals בנוי אף הוא משורות טקסט. כל שורה מכילה שם של סמל שהוגדר external, וכתובת בקוד המכונה בה יש קידוד של אופרנד המתייחס לסמל זה. כמובן שייתכן ויש מספר כתובות בקוד המכונה בהם מתייחסים לאותו סמל חיצוני. לכל התייחסות כזו תהיה שורה נפרדת בקובץ ה-externals. הכתובות מיוצגות בבסיס עשרוני.

<u>נדגים את קבצי הפלט שמייצר האסמבלר עבור קובץ מקור בשם ps.as הנתון להלן.</u>

```
; 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:            stop
.define len = 4
STR:            .string  "abcdef"
LIST:           .data   6, -9, len
K:              .data   22
.extern L3
```

להלן הקידוד הבינארי המלא (תמונת הזיכרון) של קובץ המקור, כפי שנבנה במעבר הראשון והשני.

Decimal Address	**Source Code**	**Binary Machine Code**
0100 **0101** **0102** **0103**	MAIN: mov r3, LIST[sz]	**00000000111000** **00000001100000** **00001000010010** **00000000001000**
0104 **0105**	LOOP: jmp W	**00001001000100** **00000000000001**
0106 **0107**	prn #-5	**00001100000000** **11111111101100**
0108 **0109** **0110** **0111** **0112**	mov STR[5], STR[2]	**00000000101000** **00000111110110** **00000000010100** **00000111110110** **00000000001000**
0113 **0114**	sub r1, r4	**00000011111100** **00000000110000**
0115 **0116** **0117**	cmp K, #sz	**00000001010000** **00001000011110** **00000000001000**
0118 **0119**	bne W	**00001010000100** **00000000000001**
0120 **0121**	L1: inc L3	**00000111000100** **00000000000001**
0122 **0123**	bne LOOP	**00001010000100** **00000110100010**
0124	END: stop	**00001111000000**
0125	STR: .string "abcdef"	**00000001100001**
0126		**00000001100010**
0127		**00000001100011**
0128		**00000001100100**
0129		**00000001100101**
0130		**00000001100110**
0131		**00000000000000**
0132 **0133** **0134**	LIST: .data 6, -9, len	**00000000000110** **11111111110111** **00000000000100**
0135	K: .data 22	**00000000010110**

<u>להלן תוכן קבצי הפלט של הדוגמה.</u>

<u>הקובץ ps.ob:</u>

```
   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  ****##%
```

<u>הקובץ ps.ent:</u>

כל המספרים בבסיס עשרוני

```
LOOP  0104
LIST  0132
```

<u>הקובץ ps.ext:</u>

כל המספרים בבסיס עשרוני

```
W     0105
W     0119
L3    0121
```

<u>לתשומת לב</u>: אם בקובץ המקור אין הנחיות extern. אזי לא ייווצר קובץ ext. בדומה, אם אין בקובץ המקור הנחיות entry., לא ייווצר קובץ ent. **<u>אין ליצור קובץ ext או ent שנשאר ריק</u>**.

<u>הערה</u>: אין חשיבות לסדר השורות בקבצים מסוג ent. או ext. כל שורה עומדת בפני עצמה.

סיכום והנחיות כלליות

- גודל תוכנית המקור הניתנת כקלט לאסמבלר אינו ידוע מראש, ולכן גם גודלו של קוד המכונה אינו צפוי מראש. אולם בכדי להקל במימוש האסמבלר, מותר להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים לאכסון תמונת קוד המכונה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים), יש לממשל באופן יעיל וחסכוני (למשל באמצעות רשימה מקושרת והקצאת זיכרון דינאמי).

- השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הקלט, למעט הסיומות. למשל, אם קובץ הקלט הוא prog.as אזי קבצי הפלט שיווצרו הם : prog.ob, prog.ext, prog.ent

- מתכונת הפעלת האסמבלר צריכה להיות כפי הנדרש בממ״ן, ללא שינויים כלשהם. כלומר, ממשק המשתמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו לתכנית האסמבלר כארגומנטים בשורת הפקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גרפיים למיניהם, וכד׳.

- יש להקפיד לחלק את מימוש האסמבלר למספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסוגים שונים במודול יחיד. מומלץ לחלק למודולים כגון : מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחבירי של שורה), טבלת הסמלים, מפת הזיכרון, טבלאות קבועות (קודי הפעולה, שיטות המיעון החוקיות לכל פעולה, וכד׳).

- יש להקפיד ולתעד את המימוש באופן מלא וברור, באמצעות הערות מפורטות בקוד.

- יש לאפשר תווים לבנים עודפים בקובץ הקלט בשפת אסמבלי. למשל, אם בשורת הוראה יש שני אופרנדים המופרדים בפסיק, אזי לפני ואחרי הפסיק מותר שיהיו רווחים וטאבים בכל כמות. בדומה, גם לפני ואחרי שם הפעולה. מותרות גם שורות ריקות. האסמבלר יתעלם מתווים לבנים מיותרים (כלומר ידלג עליהם).

- הקלט (קוד האסמבלי) עלול להכיל שגיאות תחביריות. על האסמבלר לגלות ולדווח על כל השורות השגויות בקלט. אין לעצור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להדפיס למסך הודעות מפורטות ככל הניתן, כדי שאפשר יהיה להבין מה והיכן כל שגיאה. כמובן שאם קובץ קלט מכיל שגיאות, אין טעם להפיק עבורו את קבצי הפלט (ob, ext, ent).

תם ונשלם פרק ההסברים והגדרת הפרויקט.

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

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

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

ב ה צ ל ח ה !