

מטלת מנחה (ממ"ן) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה : פרויקט גמר

מספר השאלות : 1 משקל המטלה : 31 נקודות

סמסטר : 2017' מועד אחרון להגשה : 20.8.2017

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, לשפת אסמבלי שתוגדר בהמשך. הפרויקט ייכתב בשפת C. אין להוסיף ספריות חיצוניות: ניתן להשתמש רק בספריות, מתוך הספרייה הסטנדרטית.

עליכם להגיש :

1. קבצי המקור של התוכנית שכתבת (קבצים בעלי סיומת c או h).
2. קבצי הרצה.
3. הגדרת סביבת העבודה (MAKEFILE). יש לקמפל עם הדגלים : -Wall -ansi -pedantic ולנפות את כל ההערות שמוציא הקומפיילר, כך שהתכנית תתקמפל ללא הערות.
4. דוגמאות של קבצי קלט, וקבצי הפלט שנוצרו על ידי הפעלת האסמבלר על קבצי קלט אלה.
5. דוגמאות של הפעלת האסמבלר על קבצי קלט המכילים מגוון של שגיאות אסמבלי (ולכן לא נוצרים קבצי פלט). יש לצרף את תדפיסי המסך המראים את הודעות השגיאה שמוציא האסמבלר.

בשל גודל הפרויקט, עליכם לחלק את התוכנית למספר קבצי מקור. יש להקפיד שהקוד הנמצא בתוכניות המקור יעמוד בקריטריונים של בהירות, קריאות וכתובה נכונה.

נזכיר מספר היבטים חשובים :

1. הפשטה של מבני הנתונים : רצוי (במידת האפשר) להפריד בין הגישה למבני הנתונים לבין המימוש של מבני הנתונים. כך, למשל, בעת כתיבת שגרות לטיפול במחסנית, אין זה מעניינם של המשתמשים בשגרות אלה, אם המחסנית ממומשת באמצעות מערך או באמצעות רשימה מקושרת.
2. קריאות הקוד : רצוי להצהיר על הקבועים הרלוונטיים בנפרד, תוך שימוש בפקודת `#define`, ולהימנע מ"מספרי קסם", שמשמעותם נהירה לכם בלבד.
3. תיעוד : יש להכניס בקבצי המקור תיעוד תמציתי וברור, שיסביר את תפקידה של כל פונקציה ופונקציה. 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 ערובה לציון גבוה. כדי לקבל ציון גבוה על התכנית לעמוד בקריטריונים לעיל, אשר משקלם המשותף מגיע עד לכ-40% ממשקל הפרויקט.

הפרויקט כולל כתיבה של תוכנית אסמבלר עבור שפת אסמבלי, שהוגדרה במיוחד עבור פרויקט זה. מותר לעבוד בזוגות. אין לעבוד בצוות גדול יותר משניים. **פרויקטים שיוגשו בשלישיות או יותר לא יבדקו.** חובה ששני סטודנטים, הבוחרים להגיש יחד את הפרויקט, יהיו **שייכים לאותה קבוצה.**

מומלץ לקרוא את הגדרת הפרויקט פעם ראשונה ברצף, לקבלת תמונה כללית לגבי הנדרש, ורק לאחר מכן לקרוא בשנית, 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. אופן הפירוק נקבע, באופן חד משמעי, על ידי המיקרו קוד של המעבד.

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 מספר מסוים של הוראות פשוטות, ולשם כך היא משתמשת בזיכרון המחשב ובאוגרים (registers) הקיימים בתוך היע"מ. דוגמאות: העברת מספר מתא בזיכרון לאוגר ביע"מ או בחזרה, הוספת 1 למספר הנמצא באוגר, בדיקה האם מספר המאוחסן באוגר שווה לאפס. הוראות פשוטות אלה ושילובים שלהן הן המרכיבות את תוכנית המשתמש כפי שהיא נמצאת בזיכרון. כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנת), נתורגם בסופו של דבר באמצעות תוכנה מיוחדת לצורה סופית זו.

קוד בשפת מכונה הוא רצף של ביטים המהווים קידוד של סדרת הוראות (תוכנית) שעל היע"מ לבצע. קוד מכונה אינו קריא למשתמש, ולכן לא נוח לקודד (או לקרוא) תכניות ישירות בשפת מכונה. שפת אסמבלי (assembly language) היא שפת תכנות מאפשרת לייצג את ההוראות של שפת המכונה בצורה סימבולית. כמובן שיש צורך לתרגם את הייצוג הסימבולי לקוד 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 (אירגון של מחשב) יש שפת מכונה יעודית משלו, ובהתאם גם שפת אסמבלי יעודית משלו. לפיכך, גם האסמבלר (כלי התירגום) הוא יעודי ושונה לכל יע"מ.

תפקידו של האסמבלר הוא לייצר קוד מכונה גולמי עבור קובץ של תכנית הכתובה בשפת אסמבלי. זהו השלב הראשון במסלול אותו עוברת התכנית, עד לקבלת קוד המוכן לריצה על גבי 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 לבך: בהסברים הכלליים על אופן עבודת תוכנת האסמבלר, תהיה מדי פעם התייחסות גם לעבודת שלבי הקישור והטעינה. התייחסויות אילו נועדו על מנת לאפשר לכם להבין את המשך תהליך העיבוד של הפלט של תוכנת האסמבלר. אין לטעות: עליכם לכתוב את תוכנית האסמבלר בלבד, **אין** צורך לכתוב גם 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 מיע"מ (יחידת עיבוד מרכזית), אוגרים וזיכרון RAM, כאשר חלק מהזיכרון משמש גם כמחסנית (stack). גודלה של מילת זיכרון במחשב הוא 10 סיביות. האריתמטיקה נעשית בשיטת המשלים ל-2 (2's complement). מחשב זה עובד רק עם מספרים שלמים חיוביים ושליילים, אין תמיכה במספרים ממשיים.

אוגרים:

למעבד 8 אוגרים כלליים (r0,r1,r2,r3,r4,r5,r6,r7). גודלו של כל אוגר הוא 10 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 9.

כמו כן יש במעבד אוגר בשם PSW (program status word), המכיל מספר דגלים המאפיינים את מצב הפעילות במעבד בכל רגע נתון. ראה בהמשך, בתאור הפקודות, לגבי השימוש בדגלים אלו.

גודל הזיכרון הוא 256 תאים, בכתובות 0-255 (בבסיס עשרוני), וכל תא הוא בגודל של 10 סיביות. לתא בזיכרון נקרא גם בשם "מילה". הסיביות בכל מילה ממוספרות בדומה לאוגר, כמפורט לעיל.

קידוד של תווים (characters) נעשה בקוד ascii.

מבנה הוראת מכונה:

כל הוראת מכונה מקודדת למספר מילות זיכרון, החל ממילה אחת ועד למקסימום חמש מילים, הכל בהתאם לשיטות המיעון בהן נעשה שימוש (ראה בהמשך). בכל סוגי ההוראות, המבנה של המילה הראשונה זהה. מבנה המילה הראשונה בהוראה הוא כדלהלן:

9 8 7 6	5 4	3 2	1 0
opcode	מיעון אופרנד מקור	מיעון אופרנד יעד	E,R,A

קידוד כל מילה בקוד המכונה ייעשה בבסיס 4 "מוזר" המוגדר כדלקמן: ארבע הספרות הן: a,b,c,d (כאשר a שקול ל-0, b ל-1, c ל-2, d ל-3). קידוד של מילה בגודל 10 סיביות בבסיס 4 מורכב מחמש ספרות (עם ספרות a מובילות לפי הצורך).

סיביות 6-9 במילה הראשונה של הפקודה מהוות את קוד ההוראה (opcode). בשפה שלנו יש 16 קודי הוראה והם:

הקוד בבסיס דצימלי (10)	פעולה
0	mov
1	cmp
2	add
3	sub
4	not
5	clr
6	lea

7	inc
8	dec
9	jmp
10	bne
11	red
12	prn
13	jsr
14	rts
15	stop

ההוראות נכתבות תמיד באותיות קטנות. פרוט משמעות ההוראות יבוא בהמשך.

סיביות 0-1 (A,R,E)

סיביות אלה מראות את סוג הקידוד, האם הוא מוחלט (Absolute), חיצוני (External) או מצריך מיקום מחדש (Relocatable)

ערך של 00 משמעו שהקידוד הוא מוחלט.
ערך של 01 משמעו שהקידוד הוא חיצוני.
ערך של 10 משמעו שהקידוד מצריך מיקום מחדש.

סיביות אלה מתווספות רק לקידודים של הוראות (לא של נתונים), והן מתווספות גם לכל המילים הנוספות שיש לקידודים אלה.

סיביות 2-3 מקודדות את מספרה של שיטת המיעון של אופרנד היעד (destination operand).

סיביות 4-5 מקודדות את מספרה של שיטת המיעון של אופרנד המקור (source operand).

בשפה שלנו קיימות ארבע שיטות מיעון, שמספרן הוא בין 0 ל-3.

השימוש בשיטות מיעון מצריך קידוד של מילות-מידע נוספות בהוראה, לכל היותר שתי מילים נוספות לכל אופרנד. אם שיטת המיעון של רק אחד משני האופרנדים דורשת מילות מידע נוספות, אזי מילות המידע הנוספות מתייחסות לאופרנד זה. אם שיטות המיעון של שני האופרנדים דורשות מילות-מידע נוספות, אזי מילות-המידע הנוספות הראשונות מתייחסות לאופרנד המקור ומילות-המידע הנוספות האחרונות מתייחסות לאופרנד היעד.

ארבע שיטות המיעון הקיימות במכונה שלנו הן:

ערך	שיטת מיעון	תוכן המילה נוספת	אופן הכתיבה	דוגמא
0	מיעון מידי	המילה הנוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר המיוצג ב- 8 סיביות, אליהם מתווספות זוג סיביות של שדה A,R,E	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני	mov #-1,r2 בדוגמה זו האופרנד הראשון של הפקודה נתון בשיטת מיעון מיידי. הפקודה כותבת את הערך 1- לתוך אוגר r2
1	מיעון ישיר	המילה הנוספת של ההוראה מכילה מען של מילה בזיכרון. מילה זו בזיכרון הינה האופרנד. המען מיוצג ב- 8 סיביות אליהן מתווספות זוג סיביות של שדה A,R,E	האופרנד הינו <u>תווית</u> שהוצהרה או תוצהר בהמשך הקובץ. ההצהרה נעשית על ידי כתיבת תווית בקובץ המקור	dec x בדוגמה זו, תוכן המילה שבכתובת x בזיכרון (ה"משתנה" x) מוקטן

ב-1.	(בהנחיית 'data' או 'string' או 'mat', או בתחילת הוראה של התוכנית), או על ידי אופרנד של הנחית 'extern'.			
add #4, a[r2][r5] בדוגמה זו, הפקודה מוסיפה את הערך 4 לתא במטריצה שנמצאת בתווית a. התא הוא בשורה המצוינת ע"י תוכן האוגר r2 ובעמודה המצוינת ע"י תוכן האוגר r5	האופרנד מורכב משם של תווית המציינת מטריצה, ולאחריה שורה ועמודה במטריצה המצוינים ע"י אוגרים בלבד, ורשומים כל אחד בסוגריים מרובעות.	בשיטת מיעון זו משתתפים שני אוגרים ותווית. בזמן ביצוע ההוראה, המעבד יפנה למטריצה (המצוינת ע"י התווית), לתא (בגודל מילה) הנמצא באינדקסים (שורה ועמודה) המצוינים ע"י האוגרים. בשיטת מיעון זו יש 2 מילות מידע נוספות בקוד ההוראה. המילה הנוספת הראשונה היא כתובת התחלת המטריצה (מצוין ע"י התווית). והמילה הנוספת השנייה תכיל את האינדקסים באופן הבא: ארבע הסיביות 6-9 מקודדים את מספר האוגר המכיל את אינדקס השורה, וארבע הסיביות 2-5 מקודדים את מספר האוגר המכיל את אינדקס העמודה. האינדקסים מתחילים ב-0.	מיעון גישה למטריצה	2
mov r1,r2 בדוגמה זו, אופרנד המקור הוא האוגר r1, ואופרנד היעד הוא האוגר r2. הפקודה מעתיקה את תוכן אוגר r1 לתוך אוגר r2. בדוגמה זו שני האופרנדים יקודדו למילה משותפת.	האופרנד הינו שם של אוגר.	האופרנד הוא אוגר. אם האוגר משמש כאופרנד יעד, מילה נוספת של הפקודה תכיל בארבע הסיביות 2-5 את מספרו של האוגר. אם האוגר משמש כאופרנד מקור, הוא יקודד במילה נוספת שתכיל בששת הסיביות 6-9 את מספרו של האוגר. אם בפקודה יש שני אופרנדים ושניהם אוגרים, הם יחלקו מילה נוספת אחת משותפת. כאשר הסיביות 2-5 הן עבור אוגר היעד, והסיביות 6-9 הן עבור אוגר המקור. לייצוג זה מתווספות זוג סיביות של שדה A,R,E. סיביות שאינן בשימוש יכילו 0.	מיעון אוגר ישיר	3

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

אפיון הוראות המכונה:

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

קבוצת ההוראות הראשונה:

הוראות הדורשות שני אופרנדים. ההוראות השייכות לקבוצה זו הן:

mov, cmp, add, sub, lea

הוראה	הסבר פעולה	דוגמא	הסבר דוגמא
Mov	מבצעת העתקה של האופרנד הראשון, אופרנד המקור (source) אל האופרנד השני, אופרנד היעד (destination) (בהתאם לשיטת המיעוט).	mov A, r1	העתק תוכן המשתנה A (המילה שבכתובת A בזכרון) לאוגר 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 סכום אופרנד המקור (הראשון) והיעד (השני).	add A, r0	אוגר r0 מקבל את סכום תוכן משתנה A ותוכנו הנוכחי של אוגר r0.
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 הינו ראשי תיבות של load effective address. פעולה זו מבצעת טעינה של המען בזיכרון המצוין על ידי התווית שבאופרנד הראשון (המקור), אל אופרנד היעד (האופרנד השני).	lea HELLO, r1	המען שמציינת התווית HELLO מוכנס לאוגר r1.

קבוצת ההוראות השניה:

הוראות הדורשות אופרנד אחד בלבד. במקרה זה זוג הסיביות 4-5 במילה הראשונה של קידוד ההוראה הן חסרות משמעות, מכיוון שאין אופרנד מקור (אופרנד ראשון) אלא רק אופרנד יעד (שני). לפיכך הסיביות 4-5 יכולו תמיד 0. על קבוצה זו נמנות ההוראות הבאות:

not, clr, inc, dec, jmp, bne, red, prn, jsr

פקודה	הסבר פעולה	דוגמא	הסבר דוגמא
not	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0).	not r2	$r2 \leftarrow \text{not } r2$
clr	איפוס תוכן האופרנד.	clr r2	$r2 \leftarrow 0$
inc	הגדלת תוכן האופרנד באחד.	inc r2	$r2 \leftarrow r2 + 1$
dec	הקטנת תוכן האופרנד באחד.	dec C	$C \leftarrow C - 1$
jmp	קפיצה בלתי מותנית אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כלומר, בעת ביצוע ההוראה, מצביע התוכנית (PC) יקבל את ערך אופרנד היעד.	jmp LINE	$PC \leftarrow \text{LINE}$
bne	bne הינו ראשי תיבות של: branch if not equal (to zero). זוהי הוראת הסתעפות מותנית. מצביע התוכנית (PC) יקבל את ערך אופרנד היעד אם ערכו של	bne LINE	אם ערך הדגל Z באוגר הסטטוס (PSW) הינו 0 אזי: $PC \leftarrow \text{LINE}$

		הדגל Z באוגר הסטטוס (PSW) הינו 0. כזכור, הדגל Z נקבע בפקודת cmp.	
red	red r1	קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	קוד ה-ascii של התו הנקרא מהקלט יוכנס לאוגר r1.
prn	prn r1	הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	התו אשר קוד ה-ascii שלו נמצא באוגר r1 יודפס לקלט הסטנדרטי.
jsr	jsr FUNC	קריאה לשגרה (סברוטניה). מצביע התוכנית (PC) הנוכחי נדחף לתוך המחסנית שבזכרון המחשב, ותוכן האופרנד מוכנס ל-PC.	$SP \leftarrow SP - 1$ $stack[SP] \leftarrow PC$ $PC \leftarrow FUNC$

קבוצת ההוראות השלישית:

הוראות ללא אופרנדים – כלומר ההוראות המורכבות ממילה אחת בלבד. הסיביות 2-5 במילה זו אינן רלוונטיות (כי אין אופרנדים) ויכילו 0.

ההוראות השייכות לקבוצה זו הן: stop, rts.

פקודה	הסבר פעולה	דוגמא	הסבר דוגמא
Rts	חזרה משיגרה. הערך בראש המחסנית של זמן ריצה מוצא מן המחסנית ומועבר לאוגר התוכנית (PC).	rts	$PC \leftarrow stack[SP]$ $SP \leftarrow SP + 1$
Stop	עצירת ריצת התוכנית.	stop	עצירת התוכנית.

מספר נקודות נוספות לגבי תיאור שפת האסמבלי:

שפת האסמבלי מורכבת ממשפטים (statements) כאשר התו המפריד בין משפט למשפט הינו תו 'n' (שורה חדשה). כלומר, כאשר מסתכלים על קובץ המקור, רואים אותו כמורכב משורות של משפטים, כאשר כל משפט מופיע בשורה נפרדת.

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

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

סוג המשפט	הסבר כללי
משפט ריק	זוהי שורה המכילה אך ורק תווים לבנים (whitespace), כלומר מכילה רק תווים 't' ו-' ' (סימני tab ורווח). ייתכן ובשורה אין אף תו (למעט התו n)
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 להקצאה ואתחול משתנים של התכנית, אך הוא אינו מייצר קוד המיועד לביצוע בעת ריצת התכנית.
משפט הוראה	זהו משפט המייצר קוד לביצוע בעת ריצת התכנית. המשפט מכיל שם של הוראה שעל ה-CPU לבצע, ותיאור האופרנדים המשתתפים בביצוע.

כעת נפרט לגבי סוגי המשפטים השונים.

משפט הנחיה:

משפט הנחיה הוא בעל המבנה הבא :

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

לאחר מכן מופיע התו '!' (נקודה), ובצמוד לנקודה (ללא רווח) שם ההנחיה. לאחר שם ההנחיה יופיעו (פרמטרים) (מספר הפרמטרים נקבע בהתאם לסוג ההנחיה).

יש לשים לב: למילות הקוד הנוצרות ממשפט הנחיה לא מצורפות זוג סיביות A,R,E והקידוד ממלא את כל 10 הסיביות של המילה.

ישנם ארבעה סוגים של משפטי הנחיה, והם :

1. 'data'.

הפרמטר(ים) של data הם רשימת מספרים שלמים חוקיים (אחד או יותר) המופרדים על ידי התו ',', (פסיק). למשל :

9, 17, -57, +7 data.

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, ולא פסיק אחרי המספר האחרון או לפני המספר הראשון.

משפט ההנחיה: 'data'. מנחה את האסמבלר להקצות מקום בתמונת הנתונים (data image), אשר בו יאוחסנו הערכים המתאימים, ולקדם את מונה הנתונים, בהתאם למספר הערכים ברשימה. אם להוראת data יש גם 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 הגדרת שם של משתנה).

כלומר אם נכתוב :

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

אזי יוקצו בתמונת הנתונים ארבע מילים שיכילו את המספרים שמופיעים בהנחייה.

אם נכתוב בתכנית הוראה לביצוע :

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. 'string'.

להנחייה '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 יוכנס הערך אפס, לסמן סיום מחרוזת. מונה הנתונים של האסמבלר יוגדל בהתאם לאורך המחרוזת + 1. אם בשורת ההנחיה מגדרת גם תווית, אזי ערכה יהיה מען המקום בזיכרון, שבו מאוחסן התו הראשון במחרוזת, באופן דומה כפי שנעשה עבור 'data'.

לדוגמא, משפט ההנחיה :

STR: .string "abcdef"

מקצה "מערך תווים" באורך של 7 מקומות החל מהמען המזוהה עם התווית ABC, ומאתחל "מערך" זה לערכי ה-ascii של התווים : a, b, c, d, e, f בהתאמה, ולאחריהם הערך 0 לסימון סוף מחרוזת.

3. 'mat'.

משפט הנחיה זה מקצה מטריצה. למשפט הנחיה 'mat' המבנה הבא :

MAT8: .mat [2][3]

MAT5: .mat [2][2] 4,-5,7,9

בדוגמא הראשונה מקצים מטריצה בשם MAT8 שגודלה 2 שורות ו-3 עמודות והיא אינה מאותחלת בערכים (אבל כן צורכת מקום בתמונת הנתונים).

בדוגמא השניה מקצים מטריצה בשם MAT5 בגודל 2 שורות ו-2 עמודות, המאותחלת לערכים המפורטים. הערכים לאתחול רשומים משמאל לימין לפי סדר השורות. מטריצה תכיל רק מספרים שלמים.

4. 'entry'.

להנחייה 'entry' פרמטר אחד והוא שם של תווית המוגדרת בקובץ המקור הנוכחי (כלומר מקבלת את ערכה בקובץ זה). מטרת entry. היא להצהיר על התווית הזו באופן שיאפשר לקטעי אסמבלי הנמצאים בקבצי מקור אחרים, להשתמש בה.

לדוגמא, השורות :

	entry	HELLO
HELLO:	add	#1, r1
	

מודיעות שקטעי אסמבלי הנמצאים בקובץ אחר יכולים להתייחס לתווית HELLO המוגדרת בקובץ הנוכחי.

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

5. 'extern'.

להנחייה 'extern' פרמטר אחד והוא שם של תווית. מטרת ההוראה היא להצהיר כי התווית מוגדרת בקובץ אחר וכי קטע האסמבלי, בקובץ זה, עושה בו שימוש. זוהי הנחיה תואמת להנחיית entry המופיעה בקובץ בו מוגדרת התווית. בזמן הקישור (link) תתבצע ההתאמה, בין ערך התווית, כפי שנקבע בקוד המכונה של הקובץ שהגדיר את התווית, לבין קוד ההוראות המשתמשות בו בקבצים אחרים.

לדוגמא, משפט ההנחיה 'extern'. התואם למשפט ההנחיה 'entry' בדוגמא הקודמת תהיה:

HELLO extern.

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

משפט הוראה:

משפט הוראה מורכב מ:

1. תווית אופציונלית.
2. שם ההוראה.
3. 0, 1 או 2 אופרנדים בהתאם לשם ההוראה.

שם ההוראה נכתב באותיות קטנות (lower case), וההוראה היא אחת מבין 16 ההוראות שהוזכרו לעיל.

לאחר שם ההוראה, יכולים להופיע אופרנדים (אחד או שניים).

במקרה של שני אופרנדים, שני האופרנדים מופרדים בתו ' ', (פסיק). כמו בהנחייה data, **לא חייבת להיות שום הצמדה של האופרנדים לפסיק או להוראה באופן כלשהו. כל כמות רווחים או tabs בין האופרנדים לפסיק ובין ההוראה לאופרנד הראשון היא חוקית.**

להוראה בעלת שני אופרנדים המבנה הבא:

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

לדוגמא:

HELLO: add r7, B

לפקודה בעלת אופרנד אחד המבנה הבא:

אופרנד שם-ההוראה תווית-אופציונלית:

לדוגמא:

HELLO: bne XYZ

להוראה ללא אופרנדים המבנה הבא:

שם-ההוראה

לדוגמא:

END:

stop

אם מופיעה תווית בשורת ההוראה 'אזי היא תוכנס אל טבלת הסמלים. ערך התווית יהיה כתובת המילה הראשונה של ההוראה בתוך תמונת הקוד שבונה האסמבלר.

תווית:

תווית חוקית מתחילה באות אלפביתית (גדולה או קטנה) ולאחריה סדרה כלשהי של אותיות וספרות, שאורכה קטן או שווה 30 תווים. התווית מסתיימת על ידי התו ':' (נקודתיים). תו זה אינו מהווה חלק מהתווית. זהו רק סימן המציינ את סופה. התווית חייבת להתחיל בעמודה הראשונה בשורה. אסור שאותה תווית תוגדר יותר מפעם אחת (בשורות שונות).

לדוגמה, התוויות שלהלן הן תוויות חוקיות.

hEllo:

X:

He78902:

לתשומת לב: מילים שמורות של שפת האסמבלי (כלומר שם של הוראה או הנחייה, או שם של רגיסטר) אינו יכולות לשמש כשם של תווית.

התוויות מקבלת את ערכה בהתאם להקשר בו היא מופיעה. תוויות בהנחיות 'data', 'string', 'mat'. תקבל את ערך מונה הנתונים (data counter) הנוכחי, בעוד שתוויות המוגדרת בשורת הוראה תקבל את ערך מונה ההוראות (instruction counter) הנוכחי.

מספר:

מספר חוקי מתחיל בסימן אופציונלי '–' או '+' ולאחריו סדרה כלשהי של ספרות בבסיס עשר. הערך של המספר הוא הערך המיוצג על ידי מחרוזת הספרות והסימן. כך למשל, 5, 76, +123 הינם מספרים חוקיים. אין תמיכה בשלמים בייצוג אחר מאשר עשרוני, ואין תמיכה במספרים ממשיים.

מחרוזת:

מחרוזת חוקית היא סדרת תווי `ascii` נראים, המוקפים במרכאות כפולות (המרכאות אינן נחשבות כחלק מהמחרוזת). דוגמא למחרוזת חוקית: `"hello world"`.

אסמבלר עם שני מעברים

כאשר מקבל האסמבלר קוד לתרגום, עליו לבצע שתי משימות עיקריות: הראשונה היא לזהות ולתרגם את קוד ההוראות, והשנייה היא לקבוע מענים לכל המשתנים והנתונים המופיעים בתוכנית.

לדוגמא: האסמבלר קורא את קטע הקוד הבא:

```

MAIN:      mov     M1[r2][r7],LENGTH
           add     r2,STR
LOOP:      jmp     END
           prn     #-5
           sub     r1,r4
           inc     K

```

```

mov    M1[r3][r3],r3
bne    LOOP
END:    stop
STR:    .string "abcdef"
LENGTH: .data 6,-9,15
K:      .data 22
M1:     .mat [2][2] 1,2,3,4

```

עליו להחליף את ההוראות mov, add, jmp, prn, sub, inc, bne, stop בקוד הבינארי השקול להם במודל המחשב שהגדרנו.

כמו כן, על האסמבלר להחליף את הסמלים M1, K, STR, LENGTH, MAIN, LOOP, END במענים של המקומות בזיכרון שם נמצאים כל נתון או הוראה בהתאמה.

נניח שקטע הקוד לעיל יאוחסן (הוראות ונתונים) בזיכרון החל ממען 0100 (בבסיס 10). במקרה זה נקבל את ה"תרגום" הבא:

לתשומת לב: המקפים המופיעים בקידוד הבינארי הם רק לצורך הדגשת ההפרדה בין השדות השונים בקידוד ונועדו לשם המחשה בלבד.

Label	Decimal Address	Base 4 Address	Instruction	Operands	Binary machine code
MAIN:	0100	1210	mov	M1[r2][r7],LENGTH	0000-10-01-00
	0101	1211		כתובת של M1	10000101-10
	0102	1212		קידוד אוגרי האינדקסים במטריצה	0010-0111-00
	0103	1213		כתובת של LENGTH	10000001-10
	0104	1220	add	r2, STR	0010-11-01-00
	0105	1221		קידוד מספר האוגר	0010-0000-00
	0106	1222		כתובת של STR	01111010-10
LOOP:	0107	1223	jmp	END	1001-00-01-00
	0108	1230		כתובת של END	01111001-10
	0109	1231	prn	#-5	1100-00-00-00
	0110	1232		המספר -5	11111011-00
	0111	1233	sub	r1,r4	0011-11-11-00
	0112	1300		קידודי מספרי האוגרים	0001-0100-00
	0113	1301	inc	K	0111-00-01-00
	0114	1302		כתובת של K	10000100-10
	0115	1303	mov	M1[r3][r3],r3	0000-10-11-00
	0116	1310		כתובת של M1	10000101-10
	0117	1311		קידוד אוגרי האינדקסים במטריצה	0011-0011-00
	0118	1312		קידוד מספר האוגר של היעד	0000-0011-00
	0119	1313	bne	LOOP	1010-00-01-00
	0120	1320		כתובת של LOOP	01101011-10
END:	0121	1321	stop		1111-00-00-00
STR:	0122	1322	.string	"abcdef"	0001100001
	0123	1323			0001100010
	0124	1330			0001100011
	0125	1331			0001100100
	0126	1332			0001100101
	0127	1333			0001100110
	0128	2000			0000000000
LENGTH:	0129	2001	.data	6,-9,15	0000000110
	0130	2002			1111110111
	0131	2003			0000001111
K:	0132	2010	.data	22	0000010110

<i>MI</i>	0133 0134 0135 0136	2011 2012 2013 2020	.mat	[2][2] 1,2,3,4	0000000001 0000000010 0000000011 0000000100
-----------	--	------------------------------	-------------	----------------	--

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 אינו ידוע, עד אשר התוכנית כולה נקראת ועוברת טיפול על ידי האסמבלר.

בדוגמא שלפנינו, אין האסמבלר יכול לדעת שהסמל END משויך למען 0121 (עשרוני), אלא רק לאחר שהתוכנית נקראה כולה.

לכן מפרידים את הטיפול של האסמבלר בסמלים לשני שלבים. בשלב הראשון בונים טבלה של כל הסמלים והערכים 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, שמותאמים בה מענים לכל הסמלים שבתכנית המקור. בדוגמא דלעיל, טבלת הסמלים לאחר מעבר ראשון היא:

סמל	ערך דצימלי
MAIN	100
LOOP	107
END	121
STR	122
LENGTH	129
K	132
M1	133

במעבר השני נעשית ההחלפה, כדי לתרגם את קוד המקור לקוד מכונה. בתחילת המעבר השני צריכים הערכים של הסמלים להיות כבר ידועים.

לתשומת לב: האסמבלר, על שני המעברים שלו, נועד כדי לתרגם קובץ מקור לקוד בשפת מכונה. בגמר פעולת האסמבלר, התכנית טרם מוכנה לטעינה לזיכרון לצורך ביצוע. לאחר השלמת תהליך התרגום, קוד המכונה יכול לעבור לשלבי הקישור/טעינה ולאחר מכן לשלב הביצוע.

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 0, ולכן נטענת ההוראה הראשונה במען 0. ה-IC מתעדכן בכל שורת הוראה המקצה מקום בזיכרון. לאחר שהאסמבלר קובע מהו אורך ההוראה, ה-IC עולה 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 לשם של כל הוראה. בזמן התרגום מחליף האסמבלר כל הוראה בקוד שלה, וכל אופרנד בקידוד מתאים. אך פעולת ההחלפה אינה כה פשוטה. יש הוראות המשתמשות בשיטות מייעון שונות לאופרנדים. אותה הוראה יכולה לקבל משמעויות שונות, בכל אחת משיטות המייעון, ולכן יתאימו לה קידודים שונים. לדוגמא, הוראת ההזהר mov יכולה להתייחס להעברת תוכן תא זיכרון

לאוגר, או להעברת תוכן אוגר לאוגר, 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 ההוראה, ושדות נוספים המכילים מידע לגבי שיטות המיעון.

במחשב שלנו קיימת גמישות לגבי שיטת המיעון של כל אחד מהאופרנדים. הערה: דבר זה לא מחייב לגבי כל מחשב. ישנם מחשבים בהם, למשל, כל הפקודות הן בעלות אופרנד יחיד (והפעולות מתבצעות על אופרנד זה ואוגר קבוע), או 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 ניתן 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 המשך התכנית. לדוגמא, פקודת הסתעפות למען, שמופיע רק בהמשך הקוד:

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

כאשר מגיע האסמבלר לשורת ההסתעפות (bne A), הוא עדיין לא הקצה מען לתווית A ולכן אינו יכול להחליף את הסמל A (האופרנד של ההוראה bne) במענו בזיכרון. נראה בהמשך כיצד נפתר בעיה זו.

בכל מקרה, תמיד אפשר לייצר במעבר הראשון את הקוד המלא של המילה הראשונה של כל פקודה, וכן את הקוד של כל הנתונים (מההנחיות .data, .string, .mat).

המעבר השני

במעבר הראשון, אין האסמבלר יכול לבנות את קוד המכונה של אופרנדים המשתמשים בסמלים שלא הוגדרו עדיין. רק לאחר שהאסמבלר עבר על כל התכנית, כך שכל התוויות נכנסו כבר לטבלת הסמלים, יכול האסמבלר להשלים את קוד המכונה של כל האופרנדים. לשם כך עובר האסמבלר שנית על כל התוכנית, ומחליף את התוויות, המופיעות באופרנדים, במעניהן המתאימים מתוך הטבלה. זהו המעבר השני, ובסופו תהיה התוכנית מתורגמת בשלמותה.

הפרדת הוראות ונתונים

בתכנית בקוד המכונה מבחינים בשני סוגים של תוכן: הוראות ונתונים. יש לארגן את הקוד כך שתהיה אבחנה 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 הקוד מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית קלה בתכנית, לנסות "לבצע" את הנתונים כאילו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום זאת היא השמטת הוראת עצירה או הסתעפות לא נכונה. התכנית כמובן לא תעבוד נכון, אך לרוב הנזק הוא יותר חמור, כי נוצרת חריגת חומרה ברגע שמתבצעת הוראה שאינה חוקית.

האסמבלר שלנו חייב להפריד, בקוד המכונה שהוא מייצר, בין קטע הנתונים לקטע ההוראות. כלומר בקבצי הפלט תהיה הפרדה של הוראות ונתונים לשני קטעים נפרדים, ואילו בקובץ

הקלט אין חובה שתהיה הפרדה בזו. 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 הוראה שאינה קיימת, מספר אופרנדים שגוי, סוג אופרנד שאינו מתאים להוראה, שם רגיסטר לא קיים, ועוד שגיאות אחרות. כמו כן מוודא האסמבלר שכל סמל מוגדר פעם אחת בדיוק.

מכאן, שכל שגיאה המתגלית על ידי האסמבלר אפשר לשייך (בדרך כלל) לשורת קלט מסוימת.

לדוגמא, 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 שאי אפשר לייצר פלט (קבצי קוד מכונה) אם נתגלו שגיאות.

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

הוראה	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
dec	אין אופרנד מקור	1,2,3
jmp	אין אופרנד מקור	1,2,3
bne	אין אופרנד מקור	1,2,3
red	אין אופרנד מקור	1,2,3
prn	אין אופרנד מקור	0,1,2,3
jsr	אין אופרנד מקור	1,2,3
rts	אין אופרנד מקור	אין אופרנד יעד
stop	אין אופרנד מקור	אין אופרנד יעד

אלגוריתם של האסמבלר

להלן נציג אלגוריתם שלדי למעבר הראשון ולמעבר השני. אנו נניח כי קוד המכונה מחולק 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). כמו כן, L יסמן את מספר המילים שתופסת הוראה נתונה.

מעבר ראשון

1. $DC \leftarrow 0$, $IC \leftarrow 0$.
2. קרא שורה. אם נגמר קובץ המקור, עבור ל-16.

3. האם השדה הראשון הוא סמל? אם לא, עבור ל-5.
4. הדלק דגל "יש הגדרת סמל".
5. האם זוהי הנחיה לאחסון נתונים, כלומר, האם הנחית data או string או mat? אם לא, עבור ל-8.
6. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם סימון (סמל מסוג data). ערכו יהיה DC. (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
7. זהה את סוג הנתונים, קודד אותם בזיכרון, עדכן את מונה הנתונים DC בהתאם לאורכם, חזור ל-2.
8. האם זו הנחית extern או הנחית entry? אם לא, עבור ל-11.
9. האם זוהי הנחית extern? אם כן, הכנס כל סמל (אחד או יותר) המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים ללא ערך, עם סימון (סמל מסוג external).
10. חזור ל-2.
11. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם סימון (סמל מסוג code). ערכו יהיה IC (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
12. חפש את שם ההוראה בטבלת שמות ההוראות, אם לא נמצא – הודע על שגיאה בשם ההוראה.
13. נתח את מבנה האופרנדים של ההוראה וחשב את L. בנה כעת את קוד המילה הראשונה של הפקודה.
14. $IC \leftarrow L + IC$
15. חזור ל-2.
16. אם נמצאו שגיאות בקובץ המקור, עצור.
17. עדכן בטבלת הסמלים את ערכם של הסמלים מסוג data, ע"י הוספת הערך הסופי של IC (ראה הסבר בהמשך).
18. התחל מעבר שני.

מעבר שני

1. $IC \leftarrow 0$
2. קרא שורה. אם נגמר קובץ המקור, עבור ל-10.
3. אם השדה הראשון הוא סמל, דלג עליו.
4. האם זוהי הנחיה extern, mat, string, data? אם כן, חזור ל-2.
5. האם זוהי הנחיה entry? אם לא, עבור ל-7.
6. סמן בטבלת הסמלים את הסמלים המתאימים כ-entry. חזור ל-2.
7. השלם את קידוד האופרנדים החל מהמילה השניה של קוד ההוראה, בהתאם לשיטת המיעון. אם אופרנד הוא סמל, מצא את המען בטבלת הסמלים.
8. $IC \leftarrow IC + L$
9. חזור ל-2.
10. אם נמצאו שגיאות במעבר שני, עצור.
11. צור ושמור את קבצי הפלט: קובץ קוד המכונה קובץ סמלים חיצוניים, וקובץ סמלים של נקודות כניסה. (ראה פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמא שראינו קודם:

```

MAIN:      mov     M1[r2][r7],LENGTH
           add     r2,STR
LOOP:      jmp     END
           prn     #-5
           sub     r1, r4
           inc     K

           mov     M1[r3][r3],r3
           bne     LOOP
END:       stop

```


STR: .string "abcdef"
 LENGTH: .data 6,-9,15
 K: .data 22
 M1: .mat [2][2] 1,2,3,4

נבצע עתה מעבר ראשון על הקוד הנתון. נבנה את טבלת הסמלים. כמו כן, נבצע במעבר זה גם את קידוד כל הנתונים, וקידוד המילה הראשונה של כל הוראה. את החלקים שעדיין לא מתורגמים במעבר זה, נשאיר כמות שהם. נניח שהקוד ייטען החל מהמען 100 (בבסיס 10).

Label	Decimal Address	Base 4 Address	instruction	Operands	Binary machine code
MAIN:	0100	1210	mov	M1[r2][r7],LENGTH	0000-10-01-00
	0101	1211		כתובת של M1	?
	0102	1212		קידוד אוגרי האינדקסים במטריצה	0010-0111-00
	0103	1213		כתובת של LENGTH	?
	0104	1220	add	r2, STR	0010-11-01-00
	0105	1221		קידוד מספר האוגר	0010-0000-00
	0106	1222		כתובת של STR	?
LOOP:	0107	1223	jmp	END	1001-00-01-00
	0108	1230		כתובת של END	?
	0109	1231	prn	#-5	1100-00-00-00
	0110	1232		המספר -5	11111011-00
	0111	1233	sub	r1,r4	0011-11-11-00
	0112	1300		קידודי מספרי האוגרים	0001-0100-00
	0113	1301	inc	K	0111-00-01-00
	0114	1302		כתובת של K	?
	0115	1303	mov	M1[r3][r3],r3	0000-10-11-00
	0116	1310		כתובת של M1	?
	0117	1311		קידוד אוגרי האינדקסים במטריצה	0011-0011-00
	0118	1312		קידוד מספר האוגר של היעד	0000-0011-00
	0119	1313	bne	LOOP	1010-00-01-00
	0120	1320		כתובת של LOOP	?
END:	0121	1321	stop		1111-00-00-00
STR:	0122	1322	.string	"abcdef"	0001100001
	0123	1323			0001100010
	0124	1330			0001100011
	0125	1331			0001100100
	0126	1332			0001100101
	0127	1333			0001100110
	0128	2000			0000000000
	0129	2001	.data	6,-9,15	0000000110
LENGTH:	0130	2002			1111110111
	0131	2003			0000001111
K:	0132	2010	.data	22	0000010110
M1	0133	2011	.mat	[2][2] 1,2,3,4	0000000001
	0134	2012			0000000010
	0135	2013			0000000011
	0136	2020			0000000100

טבלת הסמלים:

סמל	ערך דצימלי
MAIN	100
LOOP	107

END	121
STR	122
LENGTH	129
K	132
M1	133

נבצע עתה את המעבר השני ונרשום את הקוד בצורתו הסופית :

Label	Decimal Address	Base 4 Address	Command	Operands	Binary machine code
MAIN:	0100	1210	mov	M1[r2][r7],LENGTH	0000-10-01-00
	0101	1211		כתובת של M1	10000101-10
	0102	1212		קידוד אוגרי האינדקסים במטריצה	0010-0111-00
	0103	1213		כתובת של LENGTH	10000001-10
	0104	1220	add	r2, STR	0010-11-01-00
	0105	1221		קידוד מספר האוגר	0010-0000-00
	0106	1222		כתובת של STR	01111010-10
LOOP:	0107	1223	jmp	END	1001-00-01-00
	0108	1230		כתובת של END	01111001-10
	0109	1231	prn	#-5	1100-00-00-00
	0110	1232		המספר -5	11111011-00
	0111	1233	sub	r1,r4	0011-11-11-00
	0112	1300		קידודי מספרי האוגרים	0001-0100-00
	0113	1301	inc	K	0111-00-01-00
	0114	1302		כתובת של K	10000100-10
	0115	1303	mov	M1[r3][r3],r3	0000-10-11-00
	0116	1310		כתובת של M1	10000101-10
	0117	1311		קידוד אוגרי האינדקסים במטריצה	0011-0011-00
	0118	1312		קידוד מספר האוגר של היעד	0000-0011-00
	0119	1313	bne	LOOP	1010-00-01-00
	0120	1320		כתובת של LOOP	01101011-10
END:	0121	1321	stop		1111-00-00-00
STR:	0122	1322	.string	"abcdef"	0001100001
	0123	1323			0001100010
	0124	1330			0001100011
	0125	1331			0001100100
	0126	1332			0001100101
	0127	1333			0001100110
	0128	2000			0000000000
LENGTH:	0129	2001	.data	6,-9,15	0000000110
	0130	2002			1111110111
	0131	2003			0000001111
K:	0132	2010	.data	22	0000010110
M1	0133	2011	.mat	[2][2] 1,2,3,4	0000000001
	0134	2012			0000000010
	0135	2013			0000000011
	0136	2020			0000000100

לאחר סיום עבודת האסמבלר, קבצי הפלט מועברים לשלבי הקישור והטעינה.
לא נדון כאן באופן עבודת שלבי הקישור/טעינה (כאמור, אלה אינם למימוש בפרויקט זה).
לאחר סיום שלבים אלה, התוכנית תהיה טעונה בזיכרון ומוכנה לריצה.

קבצי קלט ופלט של האסמבלר

בהפעלה של האסמבלר, יש להעביר אליו באמצעות ארגומנטים של שורת הפקודה (command line arguments) רשימה של שמות קבצי מקור. אלו הם קבצי טקסט, ובהם תכניות בתחביר של שפת האסמבלי, שהוגדרה למעלה. האסמבלר פועל על כל קובץ מקור בנפרד, ויוצר עבורו קובץ מטרה (object) המכיל את קוד המכונה. כמו כן ייווצר (עבור כל קובץ מקור) קובץ externals, באם המקור הצהיר על סמלים כנקודות כניסה.

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

```
assembler x y hello
```

תגרום לכך שהאסמבלר יפעל על הקבצים: x.as, y.as, hello.as.

האסמבלר יוצר שמות לקבצי הפלט משם קובץ הקלט, כפי שהופיע בשורת הפקודה, בתוספת הסיומת ".ob". עבור קובץ ה-object, הסיומת ".ent". עבור קובץ ה-entries, והסיומת ".ext". עבור קובץ ה-externals.

מבנה כל קובץ פלט יתואר בהמשך.

לדוגמא: הפקודה: assembler x
תיצור את הקובץ x.ob וכן את הקבצים x.ext ו-x.ent אם קיימות הצהרות של entries/externals בקובץ המקור.

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 (גודל כל כניסה במערך זהה לגודלה של מילת מכונה: 10 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 מסוג .data, .string, ו-.mat).

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 (תווית) נשמרים שמו, ערכו וטיפוסו (external או relocatable).

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

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

2. שורת הוראה:

האסמבלר מוצא מהי ההוראה, ומהן שיטות המיעון של האופרנדים. (מספר האופרנדים אותם הוא מחפש נקבע בהתאם להוראה אותה הוא מוצא). האסמבלר קובע לכל אופרנד את ערכו בצורה הבאה:

- 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 מספר (מיעון מיידי) – האופרנד הוא המספר עצמו.
 - אם זו שיטת מיעון אחרת – ערכו של האופרנד נקבע לפי המפרט של שיטת המיעון (ראה תאור שיטות המיעון לעיל)
- קביעת שיטת המיעון נעשית בהתאם לתחביר של האופרנד, כפי שוסבר לעיל בהגדרת שיטות המיעון. למשל, התו # מציין מיעון מיידי, תווית מציינת מיעון ישיר, שם של אוגר מציין מיעון אוגר, וכד'.
- לאחר שהאסמבלר ניתח את השורה והחליט לגבי ההוראה, שיטת מיעון אופרנד המקור (אם יש), ושיטת מיעון אופרנד היעד (אם יש), הוא פועל באופן הבא :
- אם זוהי הוראה בעלת שני אופרנדים, אזי האסמבלר מכניס למערך ההוראות, במקום עליו מצביע מונה ההוראות IC, את קוד המילה הראשונה של ההוראה (בשיטת הייצוג של הוראות המכונה כפי שתואר קודם לכן) מילה זו מכילה את קוד ההוראה, ואת שיטות המיעון. בנוסף "משריין" האסמבלר מקום במערך עבור המילים הנוספות הנדרשות עבור פקודה זו, ומגדיל את מונה ההוראות בהתאם. אם אחד או שני האופרנדים הם אוגר או מיידי, האסמבלר מקודד כעת את המילים הנוספות הרלוונטיות במערך ההוראות.
- אם ההוראה היא בעלת אופרנד אחד בלבד, כלומר אין אופרנד מקור, אזי התרגום הינו זהה, למעט שתי הסיביות של שיטת מיעון אופרנד המקור במילה הראשונה, שיכילו תמיד 0, מכיוון שאינן רלוונטיות להוראה.
- אם ההוראה היא ללא אופרנדים (rts, stop) אזי תקודד רק המילה הראשונה. הסיביות של שיטות המיעון של שני האופרנדים יכילו 0.
- אם בשורת ההוראה קיימת תווית, אזי התווית מוכנסת אל טבלת הסמלים תחת השם המתאים, ערך התווית הוא ערך מונה ההוראות לפני קידוד ההוראה, וסוג התווית הוא relocatable.
3. שורת הנחיה :
- כאשר האסמבלר נתקל בהנחיה, הוא פועל בהתאם לסוג שלה, 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.
- 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, וכמו כן מסומן שההגדרה ניתנה בחלק הנתונים.
- בסוף המעבר הראשון, ערך התווית יעודכן בטבלת הסמלים על ידי הוספת ה-IC (כלומר הוספת האורך הכולל של קידוד כל ההוראות). הסיבה לכך היא שבתמונת קוד המכונה, כל הנתונים יופיעו אחרי כל ההוראות (ראה תאור קבצי הפלט בהמשך).
- II. 'mat'.
- האסמבלר קורא את רשימת המספרים של איתחול המטריצה (אם ישנו), מכניס כל מספר שנקרא אל מערך הנתונים, ומקדם את מצביע הנתונים באחד עבור כל מספר שהוכנס. אם המטריצה אינה מאותחלת בערכים, יוקצו תאים במערך בהתאם לגודל המטריצה, והם יאותחלו ל-0.

הטיפול בתווית המופיעה בשורה, זהה לטיפול הנעשה בהנחיה 'data'.

III. 'string'.

הטיפול ב-'string' דומה לזה של '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 המופיעה בשורה, זהה לטיפול הנעשה בהנחיה 'data'.

IV. '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.

V. 'extern'.

זוהי הצהרה על סמל (תווית) המוגדר בקובץ אחר, ואשר קטע האסמבלי בקובץ הנוכחי עושה בו שימוש. האסמבלר מכניס את הסמל אל טבלת הסמלים. ערכו הוא אפס (הערך האמיתי לא ידוע, וייקבע רק בשלב הקישור), וטיפוסו הוא external. לא ידוע באיזה קובץ נמצאת הגדרת הסמל (ואין זה משנה עבור האסמבלר).

יש לשים לב! בהוראה או בהנחיה אפשר להשתמש בשם של סמל אשר ההצהרה עליו ניתנת בהמשך הקובץ (אם באופן ישיר על ידי תווית ואם באופן עקיף על ידי extern).

פורמט קובץ ה-object

האסמבלר בונה את תמונת הזיכרון המכונה כך שקידוד ההוראה הראשונה מקובץ האסמבלי יכנס למען 100 (בבסיס 10) בזיכרון, קידוד ההוראה השנייה יכנס למען העוקב אחרי ההוראה הראשונה (תלוי במספר המילים של ההוראה הראשונה), וכך הלאה עד להוראה האחרונה.

מיד לאחר קידוד ההוראה האחרונה, מכניסים לתמונת הזיכרון את קידוד הנתונים שנבנו על ידי ההנחיות 'data', 'string' ו-'mat'. הנתונים יוכנסו בסדר בו הם מופיעים בקובץ המקור. אופרנד של הוראה שמתייחס לסמל שהוגדר באותו קובץ, יקודד כך שיצביע על המקום המתאים בתמונת הזיכרון שבונה האסמבלר.

נשים לב שהמשתנים מופיעים בתמונת הזיכרון אחרי ההוראות. זוהי הסיבה בגללה יש לעדכן בטבלת הסמלים, בסוף המעבר הראשון, את ערכי הסמלים המגדירים נתונים (סמלים מסוג (.data).

עקרונית, קובץ object מכיל את תמונת הזיכרון שתוארה כאן. קובץ object מורכב משורות של טקסט. השורה הראשונה מכילה שני מספרים: אורך כולל של קטע ההוראות (במילות זיכרון) ואורך כולל של קטע הנתונים (במילות זיכרון). השורות הבאות מתארות את תוכן הזיכרון. בכל שורה שני מספרים: כתובת של מילה בזיכרון, ותוכן המילה. כל המספרים בקובץ object הם בבסיס 4 "מוזר" שהוגדר לעיל.

בהמשך מופיע קובץ object לדוגמא בשם ps.ob המתאים לקובץ המקור ps.as

עבור כל תא זיכרון המכיל הוראה (לא נתונים) מופיע בקובץ object מידע עבור תכנית הקישור. מידע זה ניתן ע"י 2 הסיביות הימניות של הקידוד (שדה ה- E,R,A)

האות 'A' (קיצור של absolute) מציינת שתוכן התא אינו תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התכנית בזמן ביצועה (האסמבלר יוצא מתוך הנחה שהתכנית נטענת החל ממען אפס). במקרה כזה 2 הסיביות הימניות יכילו את הערך 00

האות 'R' (קיצור של relocatable) מציינת שתוכן התא כן תלוי במקום בזיכרון שבו ייטען בפועל קוד המכונה של התכנית בזמן ביצועה. לכן יש לעדכן את תוכן התא, בשלב הטעינה, על ידי הוספת היסט (Offset) מתאים (היסט זה הינו המען בו תטען המילה הראשונה של התכנית). במקרה כזה 2 הסיביות הימניות יכילו את הערך 10

האות 'E' (קיצור של external) מציינת שתוכן התא תלוי בערכו של סמל חיצוני (external), וכי רק בזמן הקישור ניתן יהיה להכניס לתא את הערך המתאים. במקרה כזה 2 הסיביות הימניות יכילו את הערך 01.

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 כולם בבסיס 4 המוזר (ראה לדוגמא את הקובץ ps.ent המתאים לקובץ המקור ששמו ps.as).

פורמט קובץ 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. כמוכן שייתכן ויש מספר כתובות בקוד המכונה בהם מתייחסים לאותו סמל חיצוני. לכל התייחסות כזו תהיה שורה נפרדת בקובץ ה-externals. הערכים כולם בבסיס 4 המוזר (ראה לדוגמא את הקובץ ps.ext המתאים לקובץ המקור ששמו ps.as).

להלן קובץ מקור בשם ps.as לדוגמא:

; file ps.as

```
.entry LOOP
.entry LENGTH
.extern L3
.extern W
MAIN:      mov     M1[r2][r7],W
            add     r2,STR
LOOP:      jmp     W
            prn     #-5
            sub     r1, r4
            inc     K

            mov     M1[r3][r3],r3
            bne     L3
END:       stop
STR:       .string "abcdef"
LENGTH:   .data    6,-9,15
K:         .data    22
M1:        .mat [2][2] 1,2,3,4
```

הקובץ ששמו ps.ob הוא קובץ object שהתקבל מהפעלת האסמבלר על קובץ המקור דלעיל. להלן טבלת הקידוד המלא הבינארי, ולאחריה פורמט קובץ ה- obkect. **כל תוכן הקובץ מיוצג במספרים בבסיס 4 המוזר.**

טבלת הקידוד הבינארי:

Label	Decimal Address	Base 4 Address	Instruction	Operands	Binary machine code
MAIN:	0100	1210	mov	M1[r2][r7],W	0000-10-01-00
	0101	1211		כתובת של M1	10000101-10
	0102	1212		קידוד אוגרי האינדקסים במטריצה	0010-0111-00
	0103	1213		כתובת של W (סמל חיצוני)	00000000-01
	0104	1220	add	r2, STR	0010-11-01-00
	0105	1221		קידוד מספר האוגר	0010-0000-00
	0106	1222		כתובת של STR	01111010-10
LOOP:	0107	1223	jmp	W	1001-00-01-00
	0108	1230		כתובת של W	00000000-01
	0109	1231	prn	#-5	1100-00-00-00
	0110	1232		המספר -5	11111011-00
	0111	1233	sub	r1,r4	0011-11-11-00
	0112	1300		קידודי מספרי האוגרים	0001-0100-00
	0113	1301	inc	K	0111-00-01-00
	0114	1302		כתובת של K	10000100-10
	0115	1303	mov	M1[r3][r3],r3	0000-10-11-00
	0116	1310		כתובת של M1	10000101-10
	0117	1311		קידוד אוגרי האינדקסים במטריצה	0011-0011-00
	0118	1312		קידוד מספר האוגר של היעד	0000-0011-00
	0119	1313	bne	L3	1010-00-01-00
	0120	1320		כתובת של L3 (סמל חיצוני)	00000000-10
END:	0121	1321	stop		1111-00-00-00
STR:	0122	1322	.string	"abcdef"	0001100001
	0123	1323			0001100010
	0124	1330			0001100011
	0125	1331			0001100100
	0126	1332			0001100101
	0127	1333			0001100110
	0128	2000			0000000000
LENGTH:	0129	2001	.data	6,-9,15	0000000110
	0130	2002			1111110111
	0131	2003			0000001111
K:	0132	2010	.data	22	0000010110
M1	0133	2011	.mat	[2][2] 1,2,3,4	0000000001
	0134	2012			0000000010
	0135	2013			0000000011
	0136	2020			0000000100

הקובץ ps.ob:

הערה: שורת הכותרת אינה חלק מהקובץ, ונועדה להבהרה בלבד.

Base 4 address Base 4 code

```

bbc dd
bcba aacba
bcbb cabbc
bcbe acbda
becbd aaaab
beca acdba
becb acaaa
becc bdccc
becd cbaba
bcda aaaab
bcd bdaaaa
bcd dcda
bcd addda

```

bdaa	abbaa
bdab	bdaba
bdac	cabac
bdad	aacda
bdba	cabbc
bdbb	adada
bdbc	aaada
bdbd	ccaba
bdeca	bccdc
bdecb	ddaaa
bdec	abcab
bdec	abcac
bdda	abcad
bddb	abcba
bddc	abcbb
bddd	abcbc
caaa	aaaaa
caab	aaabc
caac	dddbd
caad	aaadd
caba	aabbc
cabb	aaaab
cabc	aaaac
cabd	aaaad
caca	aaaba

קובץ ps.ent:

LOOP	bccd
LENGTH	caab

קובץ ps.ext:

W	bcbd
W	bcda
L3	bdca

לתשומת לב: אם בקובץ המקור אין הצהרת extern. אזי לא ייווצר עבורו קובץ ext. בדומה, אם אין הצהרת entry, לא ייווצר קובץ ent.

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

סיכום והנחיות כלליות

- אורך התוכנית, הניתנת כקלט לאסמבלר, אינו ידוע מראש (ואינו קשור לגודל 256 של הזיכרון במעבד הדמיוני). ולכן אורכה של התוכנית המתורגמת, אינו אמור להיות צפוי מראש. אולם בכדי להקל במימוש התכנית, ניתן להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים לאחסון קוד המכונה. מבני נתונים אחרים בפרויקט (למשל טבלת הסמלים), יש לממשל על פי יעילות/תכונות נדרשות (למשל באמצעות רשימה מקושרת).
- קבצי הפלט של האסמבלר צריכים להיות בפורמטים המופיעים למעלה. שמם של קבצי הפלט צריך להיות תואם לשם קובץ הקלט, פרט לסיומות. למשל, אם קובץ הקלט הוא prog.as אזי קבצי הפלט שיווצרו הם : prog.ob, prog.ext, prog.ent .
- אופן הפעלת האסמבלר צריך להיות תואם לנדרש בממ"ן, ללא שינויים כלשהם. אין להוסיף תפריטים אינטראקטיביים למיניהם, וכד'. הפעלת התוכנית תיעשה רק עם ארגומנטים של שורת פקודה.
- יש להקפיד לחלק את המימוש למודולים לפי משימות. אין לרכז משימות מסוגים שונים במודול יחיד. מומלץ לחלק למודולים כגון : מבני נתונים, מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס 4 מוזר, ניתוח תחבירי של שורה), טבלת סמלים, ועוד.
- יש להקפיד ולתעד את המימוש באופן מלא וברור, באמצעות הערות מפורטות בקוד.
- יש להקפיד להתעלם מרווחים וטאבים, ולהיות סלחנים כלפי תוכנית קלט העושה שימוש ביותר רווחים מהנדרש. למשל, אם בהוראה יש שני אופרנדים המופרדים בפסיק, אזי לפני או אחרי הפסיק יכולים להיות רווחים וטאבים בכל כמות. בדומה, גם לפני ואחרי שם ההוראה. בכל המקרים האלה השורה צריכה להיחשב חוקית (לפחות מבחינת הרווחים).
- במקרה שתוכנית המקור (הקלט) מכילה שגיאות תחביריות, נדרש לגלות ולדווח, כמו באסמבלר אמיתי, על כל השגיאות הקיימות, ולא לעצור לאחר היתקלות בשגיאה הראשונה. כמובן שאם הקלט מכיל שגיאות, אין טעם להפיק קבצי פלט (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 אתר הקורס, לעשות זאת. שאלות באתר זה הרבה שאלות בנושא חומר הלימוד והממ"נים, והתשובות יכולות להועיל לכולם.

לתשומת לבכם : לא יתקבלו ממ"נים באיחור ללא סיבה מוצדקת, וללא אישור מצוות הקורס.

בהצלחה!!!!