

# מטלת מנחה (ממ"ן) 14

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרויקט גמר

מספר השאלות: 1 משקל המטלה: 31 נקודות

סמסטר: 2011' מועד אחרון להגשה: 14.8.2011

## קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
- שליחת מטלות באמצעות הדואר או הגשה ישירה למנחה במפגשי ההנחיה

## 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.

עליך לכתוב תוכנת אסמבלר, לשפת אסמבלי שתוגדר בהמשך. הפרוייקט ייכתב בשפת C. עליך להגיש:

1. קבצי המקור של התוכנית שכתבת (קבצים בעלי סיומת c או h).
2. קבצי הרצה.
3. הגדרת סביבת העבודה (MAKEFILE).
4. דוגמא לקבצי קלט וקבצי הפלט, שנוצרו על ידי הפעלת התוכנית שלך על קבצי קלט אלה.

בגלל גודל הפרוייקט עליך 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 מבנה 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 הרלוונטים בנפרד תוך שימוש בפקודת `#define`, ולהימנע ממספרי קסם, שמשמעותם נהירה לך בלבד.
  3. תיעוד: יש להכניס בקבצי המקור תיעוד תמציתי וברור שיסביר תפקידה של כל פונקציה ופונקציה. כמו כן יש להסביר את תפקידם של משתנים חשובים.
- הערה: תוכנית "עובדת", דהיינו תוכנית שמבצעת את הדרוש ממנה אינה ערוכה לציון גבוה. כדי לקבל ציון גבוה על התכנית לעמוד בקריטריונים לעיל אשר משקלם המשותף מגיע לכ- 40% ממשקל הפרוייקט.

הפרוייקט כולל כתיבה של תוכנית אסמבלר עבור שפת אסמבלי, שהוגדרה במיוחד עבור פרוייקט זה. מותר לעבוד בזוגות. אין לעבוד בצוות גדול יותר משניים. פרוייקטים שיוגשו בשלישיות או יותר יקבלו ציון אפס. **חובה** ששני סטודנטים, הבוחרים להגיש יחד את הפרוייקט, יהיו שייכים לאותה קבוצה.

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

### רקע כללי

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 – 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, הנקראות בתים. כאשר נמצאת בזיכרון תוכנית משתמש, לא ניתן להבחין, בעין שאינה מיומנת, בהבדל פיסי כלשהו, בין אותו חלק בזיכרון, שבו נמצאת התוכנית, לבין שאר הזיכרון.

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מספר מסוים של הוראות פשוטות, ולשם כך היא משתמשת בכמה אוגרים (register) הקיימים בה. 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 ההוראות המרכיבות את תוכנית המשתמש כפי שהיא נמצאת בזיכרון. כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנת), תועבר בסופו של דבר באמצעות תוכנה מיוחדת לצורה סופית זו.

נסביר כיצד מתבצע קוד זה: כל הוראה בקוד יכולה להתייחס לנתון הנמצא בהוראה עצמה, לאוגר או למען בזיכרון. היע"מ מפרקת כל שורת קוד להוראה ולאופרנדים שלה, ומבצעת את ההוראה. אוגר מיוחד בתוך היע"מ מצביע תמיד על ההוראה הבאה לביצוע (program counter). כאשר מגיעה היע"מ להוראת עצירה, היא מחזירה את הפיקוד לתוכנית שהפעילה אותה או למערכת ההפעלה.

לכל שפת תכנות יש, כידוע, מהדר (compiler), או מפרש (interpreter), המתרגם תוכניות מקור לשפת מכונה. אם תוכנית מקור נכתבה בשפת אסמבלי, נקראת התוכנית המתרגמת בשם אסמבלר. פרוייקט זה עליך לכתוב אסמבלר. לשם כך נעקוב אחרי גלגולה של תוכנית שנכתבה בשפת אסמבלי, שנגדיר במיוחד עבור פרוייקט זה, עד לשליחתה לתוכנת הקישור והטעינה (linker/loader).

לתשומת לבך: בהסברים הכלליים על אופן עבודת תוכנת האסמבלר, יש התייחסות גם לעבודת תוכנת הקישור (linker) ותוכנת הטעינה (loader). התייחסויות אילו הובאו על מנת לאפשר לכם להבין את המשך תהליך העיבוד של הפלט של תוכנת האסמבלר. אל לך לטעות, עליך לכתוב את תוכנת האסמבלר בלבד, אין צורך לכתוב גם את תוכנת הקישור והטעינה!!!

תחילה נגדיר את שפת האסמבלי ואת המחשב הדמיוני שהגדרנו עבור פרוייקט זה.

## "חומרה":

המחשב מורכב מיע"מ (יחידת עיבוד מרכזית), אוגרים וזיכרון RAM, כאשר חלק מהזיכרון משמש גם כמחסנית (stack). גודלה של מלת זיכרון במחשב הוא 16 סיביות. האריתמטיקה נעשית בשיטת המשלים ל-2 (2's complement). מחשב זה מטפל רק במספרים שלמים חיוביים ושליילים, אין טיפול במספרים ממשיים.

## אוגרים:

למחשב 8 אוגרים כלליים (r0,r1,r2,r3,r4,r5,r6,r7), מונה תוכנית (PC – program counter), מצביע המחסנית של זמן ריצה (SP – stack pointer), ואוגר סטטוס (PSW – program status word) בעל שני דגלים: דגל נשא (Carry) ודגל אפס (Zero). גודלו של כל אוגר הוא 16 סיביות.

עבור ה-PSW הסיביות הראשונות הן C ו-Z, כלומר בתחביר של שפת C:

$$C = (PSW \& 01)$$

$$Z = (PSW \& 02)$$

גודל הזיכרון הוא 2000 מלים (גודל כל מלה 16 סיביות).

קידוד של תווים (characters) נעשה בקוד ascii.

## אפיון מבנה הוראת מכונה:

כל הוראת מכונה מורכבת ממילה אחת (בת 16 סיביות), שתי מלים (בנות 16 סיביות כל אחת), או שלוש מלים (בנות 16 סיביות כל אחת). בכל סוגי ההוראות המבנה של המילה הראשונה זהה. מבנה המילה הראשונה בהוראה הוא כדלהלן:

15	12	11	9	8	6	5	3	2	0
קוד ההוראה		שיטת מיעון אופרנד מקור		אוגר אופרנד מקור		שיטת מיעון אופרנד יעד		אוגר אופרנד יעד	

סיביות 12-15 מהוות את קוד ההוראה (opcode). בשפה שלנו יש 16 קודי הוראה והם:

הקוד בבסיס דצימלי (10)	הקוד בבסיס 12	פעולה
0	0	mov
1	1	cmp
2	2	add
3	3	sub
4	4	not
5	5	clr
6	6	lea
7	7	inc
8	8	Dec
9	9	Jmp
10	A	bne

11	B	Red
12	10	Prn
13	11	Jsr
14	12	Rts
15	13	Hlt

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

סיביות 9-11 מקודדות את שיטת המיעון של אופרנד המקור (source operand).

סיביות 3-5 מקודדות את שיטת המיעון של אופרנד היעד (destination operand).

בשפה שלנו קיימות חמש שיטות מיעון.

חלק משיטות המיעון דורשות מילות מידע נוספות. אם שיטת המיעון של רק אחד משני האופרנדים דורשת מילות נוספות, אזי המילות הנוספות מתייחסות לאופרנד זה. אך אם שיטות המיעון של שני האופרנדים דורשות מילות נוספות אזי המילות הנוספות הראשונות מתייחסות לאופרנד המקור (source operand) והמילות הנוספות האחרונות מתייחסות לאופרנד היעד (destination operand).

## סיביות 6-8

סיביות אלו מסמלות את מספרו של האוגר (0-7) המשתתף כאופרנד מקור (source) בפעולה. שים לב! לא כל שיטת מיעון דורשת זהות של אוגר. עבור אותן שיטות מיעון שלא דורשות אוגר, שדה זה לא מנוצל.

## סיביות 0-2

סיביות אלו מסמלות את מספרו של האוגר (0-7) המשתתף כאופרנד יעד (destination) בפעולה. שים לב! לא כל שיטת מיעון דורשת זהות של אוגר. עבור אותן שיטות מיעון שלא דורשות אוגר, שדה זה לא מנוצל.

חמש שיטות המיעון הקיימות במכונה שלנו הן :

ערך	שיטת מיעון	תוכן המילה נוספת	אוגר	אופן הכתיבה	דוגמא
0	מיעון מידי	המילה הנוספת מכילה את האופרנד עצמו.	סיביות זיהוי האוגר אינן בשימוש בשיטת מיעון זו.	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר חוקי.	mov #-1,r2
1	מיעון ישיר	המילה הנוספת מכילה מען בזיכרון. תוכן מען זה הינו האופרנד המבוקש.	סיביות זיהוי האוגר אינן בשימוש בשיטת מיעון זו.	האופרנד הינו תווית שהוצהרה או תוצהר בהמשך הקובץ. ההצהרה נעשית על ידי כתיבת תווית בקובץ (בפקודת 'data' או 'string'. או בהגדרת תווית ליד שורת קוד של התוכנית). או על ידי הנחת 'extern'. (אם התווית הוצהרה כ- external אזי תוכנית הקישור תדאג למתן הערך המתאים).	mov x, r2
2	מיעון אינדקס יחסי משתנה	למיעון זה 2 מילים נוספות. אחת היא הכתובת של הסמל עליו רוצים לבצע אינדקס, והשניה המרחק היחסי(שלילי או חיובי) של הפקודה הנוכחית אל הסמל הנוסף המשמש כאינדקס	אינן בשימוש	מיעון אינדקס משתנה הוא פניה לתווית בתוספת אינדקס (היסט של תאים בזיכרון). האינדקס ייכתב בסוגריים מרובעות ויחושב יחסית לפקודה הנוכחית. האינדקס הוא בעצמו תווית נוספת.	mov y[*j],r3
3	מיעון אינדקס דו מימדי	למיעון זה 2 מילים נוספות. האחת, הכתובת של הסמל עליו רוצים לבצע אינדקס כפול, והשניה, כתובת הסמל המוזכר באינדקס הראשון.	האוגר שמספרו מופיע כאינדקס	מיעון אינדקס דו מימדי הוא פניה לתווית בתוספת אינדקס דו מימדי (היסט של תאים בזיכרון בדומה לפניה למטריצה). האינדקס ייכתב בסוגריים מרובעות כפולות. האינדקס הראשון יינתן ע"י משתנה והאינדקס השני יינתן ע"י אוגר.	mov y[j][r5],r2
4	מיעון אוגר ישיר	אין מילה נוספת בשיטת מיעון זו.	האוגר שמספרו מופיע בשדה זה, מכיל את האופרנד המבוקש.	האופרנד הינו שם חוקי של אוגר.	mov r1,r2

הערה: מותר להתייחס לתווית עוד לפני שמצהירים עליה (באופן סתום או מפורש), בתנאי שהיא אכן מוצהרת במקום כלשהו בקובץ.

מילה שניה ושלישית :  
מופיעות באם נדרש בהתאם לשיטות המיעון המוגדרות במילה הראשונה של הפקודה.

אפיון הוראות המכונה :  
הוראות המכונה מתחלקות לשלוש קבוצות לפי מספר האופרנדים הדרוש להן.

**קבוצה ראשונה :**  
ההוראות הזקוקות לשני אופרנדים. הפקודות השייכות לקבוצה זו הן :  
mov, cmp, add, sub, not, clr, lea

קוד אוקטלי	פקודה	הסבר פעולה	דוגמא	הסבר דוגמא
0	mov	מבצעת העתקה של האופרנד הראשון, אופרנד המקור (source) אל האופרנד השני, אופרנד היעד (destination) (בהתאם לשיטת המיעון).	mov A, r1	העתק תוכן משתנה A לאוגר r1.
1	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 את דגל האפס, דגל Z, באוגר הסטטוס, PSW, יודלק, אחרת הוא יאופס.	cmp A, r1	אם תוכן הערך הנמצא במען A זהה לתוכנו של אוגר r1 אזי דגל האפס, Z, באוגר הסטטוס, PSW, יודלק, אחרת הוא יאופס.
2	add	אופרנד היעד (השני) מקבל את סכום אופרנד המקור (הראשון) והיעד (השני).	add A, r0	אוגר r0 מקבל את סכום תוכן משתנה A וערכו הנוכחי של אוגר r0.
3	sub	אופרנד היעד (השני) מקבל את ההפרש בין אופרנד היעד (השני) ואופרנד המקור (הראשון).	sub #3, r1	אוגר r1 מקבל את ההפרש בין תוכן האוגר, r1, והמספר 3.
6	lea	lea – 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 של תווית HELLO מוכנס לאוגר r1.

#### קבוצת הפקודות השניה :

הוראות הדורשות אופרנד אחד בלבד. במקרה זה ששת הסיביות (6-11) חסרות משמעות מכיוון שאין אופרנד מקור (אופרנד ראשון) אלא רק אופרנד יעד (שני). על קבוצה זו נמנות ההוראות הבאות :

inc, dec, jmp, bne, red, prn, jsr

קוד אוקטלי	פקודה	הסבר פעולה	דוגמא	הסבר דוגמא
4	not	הפיכת ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 וההיפך: 1 ל-0). האופרנד יכול להיות אוגר בלבד. אין השפעה על הדגלים.	not r2	r2 ← not r2

5	clr	אפס את תוכן האופרנד.	clr r2	$r2 \leftarrow 0$
7	inc	הגדלת תוכן האופרנד באחד.	inc r2	$r2 \leftarrow r2 + 1$
10	dec	הקטנת תוכן האופרנד באחד.	dec C	$C \leftarrow C - 1$
11	jmp	קפיצה בלתי מותנית אל המען המיוצג על ידי האופרנד.	jmp LINE	$PC \leftarrow LINE$
12	bne	bne הינו ראשי תיבות של : branch if not equal (to zero). זוהי פקודת הסתעפות מותנית. הערך במצביע התוכנית (PC) יקבל את ערך אופרנד היעד אם ערכו של דגל האפס (דגל Z) באוגר הסטטוס (PSW) הינו 0.	bne LINE	אם ערך דגל Z באוגר הסטטוס (PSW) הינו 0 אזי : $PC \leftarrow LINE$
13	red	קריאה של תו מתוך לוח המקשים אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מלוח המקשים יוכנס לאוגר r1.
14	prn	הדפסת התו שערך ה-ascii שלו נמצא באופרנד, אל קובץ הפלט הסטנדרטי (stdout).	prn r1	התו אשר קוד ה-ascii שלו נמצא באוגר r1 יודפס לקובץ הקלט הסטנדרטי.
15	jsr	קריאה לשגרה (סברוטניה). הכנסת מצביע התוכנית (PC) לתוך המחסנית של זמן ריצה והעברת ערך האופרנד למצביע התוכנית (PC).	jsr FUNC	$stack[SP] \leq PC$ $SP \leq SP - 1$ $PC \leq FUNC$



### קבוצת הפקודות השלישית:

מכילה את ההוראות ללא אופרנדים – כלומר ההוראות המורכבות ממלה אחת בלבד.

ההוראות השייכות לקבוצה זו הן: hlt, rts.

קוד אוקטלי	פקודה	הסבר פעולה	דוגמא	הסבר דוגמא
16	rts	הוראת חזרה משיגרה. ביצוע הוראת pop על המחסנית של זמן ריצה, והעברת הערך שהיה שם לאוגר התוכנית (PC). הוראה זו מורכבת ממלה אחת בלבד (בת 16 סיביות). במלה זו החלק המשמעותי הן הסיביות 15-12 המהוות את קוד ההוראה. לשאר הסיביות אין כל חשיבות.	rts	$SP \leq SP + 1$ $PC \leq stack[SP]$
17	hlt	הוראה לעצירת התוכנית. ההוראה מורכבת ממלה אחת בלבד (בת 16 סיביות). במלה זו החלק המשמעותי הן הסיביות 15-12 המהוות את קוד ההוראה. לשאר הסיביות אין כל חשיבות.	hlt	עצירת התוכנית.

### מספר נקודות נוספות לגבי תיאור שפת האסמבלי:

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

ישנם ארבעה סוגי משפטים בשפת האסמבלי, והם:

סוג המשפט	הסבר כללי
משפט ריק	זוהי שורה המכילה בתוכה אך ורק תווים לבנים (white spaces) כלומר מורכבת מצירוף של 't' ו-' ' (סימני tab ורווח).
משפט הערה	זהו משפט אשר התו בעמודה הראשונה בשורה בה הוא מופיע הינו תו ';' (נקודה פסיק). על האסמבלר להתעלם לחלוטין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמבלר בעת הביצוע. יש מספר סוגי משפטי הנחיה. משפט הנחיה אינו מייצר קוד.
משפט פעולה	זהו משפט המייצר קוד. הוא מכיל בתוכו פעולה שעל ה-CPU לבצע, ותיאור האופרנדים המשתתפים בפעולה.

כעת נפרט לגבי סוגי המשפטים השונים.

## משפט הנחיה :

משפט הנחיה הוא בעל המבנה הבא :

בתחילתו יכולה להופיע תווית (label) (התווית חייבת להיות בעלת תחביר חוקי. התחביר של תווית חוקית יתואר בהמשך).  
התווית היא אופציונלית. לאחר מכן מופיע התו ' ' (נקודה) ובצמוד אליה שם ההנחיה. לאחר שם ההנחיה יופיעו (באותה שורה) הפרמטרים שלו (מספר הפרמטרים נקבע בהתאם לסוג ההנחיה).

ישנם ארבעה סוגי משפטי הנחיה והם :

### 1. 'data'

הפרמטר(ים) של data. הינו רשימת מספרים חוקיים המופרדים על ידי התו ' ' (פסיק). למשל :

9, 17, -57, +7 data.

שים לב שהפסיקים אינם חייבים להיות צמודים למספרים. בין מספר לפסיק ובין פסיק למספר יכול להופיע מספר כלשהו של רווחים לבנים, או ללא רווחים לבנים כלל. אולם, הפסיק חייב להופיע בין הערכים.

משפט ההנחיה : 'data'. מדריכה את האסמבלר להקצות מקום בהמשך חלק תמונת הנתונים (data image) שלו אשר בו יאוחסנו הערכים המתאימים ולקדם את מונה הנתונים בהתאם למספר הערכים ברשימה. אם להוראת data. הייתה 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.

כלומר אם נכתוב :

XYZ:	data.	+7, -57, 17, 9
	mov	XYZ, r1

אזי יוכנס לאוגר r1 הערך +7.

לעומת זאת :

lea	XYZ, r1
-----	---------

יכניס לאוגר r1 את המען בזיכרון (בחלק הנתונים) אשר בו אוחסן הערך +7.

### 2. 'string'

הפרמטר של הוראת 'string'. הינו מחרוזת חוקית אחת. משמעותה דומה להוראת 'data'. תווי ascii המרכיבים את המחרוזת מקודדים לפי ערכי ה-ascii המתאימים ומוכנסים אל תמונת הנתונים לפי סדרם. בסוף יוכנס ערך אפס, לסמן סיום מחרוזת. ערך מונה הנתונים יוגדל בהתאם לאורך המחרוזת + 1. אם יש תווית באותה שורה אזי ערכה יצביע אל המקום בזיכרון שבו מאוחסן קוד ה-ascii של התו הראשון במחרוזת. באותה צורה כפי שנעשה הדבר עבור 'data'.

כלומר משפט ההנחיה :

“abcdef” .string ABC:

מקצה “מערך תווי” באורך של 7 מילים החל מהמען המזוהה עם התויות ABC, ומאתחלת “מערך” זה לערכי ה-ascii של התוים a,b,c,d,e,f בהתאמה ולאחריהם ערך 0 לסימון סוף מחרוזת תויות.

3. ‘.entry’

להוראת ‘.entry’ פרמטר אחד והוא שם של תויות המוגדרת בקובץ (מקבלת את ערכה שם). מטרת entry היא להצהיר על התויות הזו כעל תויות אשר קטעי אסמבלי הנמצאים בקבצים אחרים מתייחסים אליה.

לדוגמא :

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

מודיע שקטע (או קטעי) אסמבלי אחר הנמצא בקובץ אחר יתייחס לתויות HELLO.

הערה : תויות בתחילת שורת entry חסרת משמעות.

4. ‘.extern’

להוראת ‘.extern’ פרמטר אחד בלבד והוא שם של תויות. מטרת ההוראה היא להצהיר כי התויות מוגדרת בקובץ אחר וכי קטע האסמבלי בקובץ זה עושה בו שימוש. בזמן הקישור (link) תתבצע ההתאמה, בין הערך כפי שהוא מופיע בקובץ שהגדיר את התויות, לבין ההוראות המשתמשות בו בקבצים אחרים. גם בהוראה זו תויות המופיעה בתחילת השורה הינה חסרת ממשמעות.

לדוגמא, משפט הנחית ה-‘.extern’ המתאים למשפט הנחית ה-‘.entry’ בדוגמא הקודמת תהיה :

HELLO .extern

### שורת הוראה :

שורת הוראה מורכבת מ :

1. תויות אופציונליות.
2. שם ההוראה עצמה.
3. 0, 1 או 2 אופרנדים בהתאם להוראה.

אורכה 80 תוים לכל היותר.

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

לאחר שם ההוראה יכול להופיע האופרנד או האופרנדים.

במקרה של שני אופרנדים, שני האופרנדים מופרדים בתו ‘,’ (פסיק). כמקודם, לא חייבת להיות שום הצמדה של האופרנדים לתו המפריד או להוראה באופן כלשהו. כל עוד מופיעים רווחים או tabs בין האופרנדים לפסיק ובין האופרנדים להוראה, הדבר חוקי.

להוראה בעלת שני אופרנדים המבנה של :

אופרנד יעד, אופרנד מקור      הוראה      תווית אופציונלית  
לדוגמא :

HELLO:      add      r7, B

לפקודה בעלת אופרנד אחד המבנה הבא :

אופרנד      הוראה      תווית אופציונלית  
לדוגמא :

HELLO:      bne      XYZ

להוראה ללא אופרנדים המבנה הבא :

תווית אופציונלית      הוראה  
לדוגמא :

END:      hlt

אם מופיעה תווית בשורת ההוראה אזי היא תוכנס אל טבלת הסמלים. ערך התווית יצביע על מקום ההוראה בתוך תמונת הקוד שבונה האסמבלר.

### תווית :

תווית חוקית מתחילה באות (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 משם התווית. זהו רק סימן המייצג את סופה. כמו כן התווית חייבת להתחיל בעמודה הראשונה בשורה. אסור שיופיעו שתי הגדרות שונות לאותה התווית. התווית שלהלן הן תוויות חוקיות.

hEllo:

x:

He78902:

שם של הוראה או שם חוקי של רגיסטר אינם יכולים לשמש כשם של תווית.

התווית מקבלת את ערכה בהתאם להקשר בו היא מופיעה. תווית בהוראות 'data', 'string'. תקבל את ערך מונה הנתונים (data counter) המתאים בעוד שתווית המופיעה בשורת הוראה תקבל את ערך מונה ההוראות (instruction counter) המתאים.

### 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 76, -5, +123 הינם מספרים חוקיים. (אין טיפול במספרים ממשיים).

### מחרוזת :

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

## אסמבלר שני מעברים

כאשר מקבל האסמבלר קוד לתרגום, עליו לבצע שתי משימות עיקריות: הראשונה היא לזהות ולתרגם את קוד ההוראה, והשנייה היא לקבוע מענים לכל המשתנים והנתונים המופיעים בתוכנית. לדוגמא: אם האסמבלר קורא את קטע הקוד הבא:

```

MAIN:      mov     LENGTH, r1
           lea     STR[*LENGTH], r4
LOOP:      jmp     END
           prn     STR[K][r3]
           sub     #1, r1
           inc     r0
           mov     r3,STR[*K]
           bne     LOOP
END:        hlt
STR:        .string "abcdef"
LENGTH:     .data   6
K:          .data   2

```

עליו להחליף את שמות הפעולה `mov`, `lea`, `jmp`, `prn`, `sub`, `inc`, `bne`, `hlt` בקוד הבינארי השקול להם במחשב שהגדרנו.

כמו כן, על האסמבלר להחליף את הסמלים `STR`, `LENGTH`, `MAIN`, `LOOP`, `END` במענים של שני האתרים שהוקצו לשני הנתונים, ובמענים של ההוראות המתאימות.

נניח לרגע שקטע הקוד למעלה יאוחסן (הוראות ונתונים) בקטע זיכרון החל ממען 0100 בזיכרון. הערה: במקרה זה נקבל את ה"תרגום" הבא:

<i>Label</i>	<b>Decimal Address</b>	<b>Base 12 Address</b>	<b>Command</b>	<b>Operands</b>	<b>Binary machine code</b>	<b>Base 12 machine code</b>
<i>MAIN:</i>	<b>0100</b>	0084	<b>mov</b>	LENGTH, r1	0000 001 000 100 001	00000395
	<b>0101</b>	0085			0000 000 001 111 011	000000A6
	<b>0102</b>	0086	<b>lea</b>	STR[*LENGTH], r4	0110 010 000 100 100	0012A04
	<b>0103</b>	0087			0000 000 000 111 100	000009B
	<b>0104</b>	0088			0000 000 000 011 000	0000020
<i>LOOP:</i>	<b>0105</b>	0089	<b>jmp</b>	END	1001 000 000 001 000	0019408
	<b>0106</b>	008A			0000 000 000 111 011	000009A
	<b>0107</b>	008B 0090 0091	<b>prn</b>	STR[K][r3]	1100 000 000 011 011	0024563
					0000 000 000 111 100	000009B
					0000 000 001 111 100	00000A7
	<b>0108</b>	0092	<b>sub</b>	#1, r1	0011 000 000 100 001	00007169
	<b>0109</b>	0093			0000 000 000 000 001	00000001
	<b>0110</b>	0094	<b>inc</b>	r0	0111 000 000 100 000	0014740
	<b>0111</b>	0095	<b>mov</b>	r3,STR[*k]	0000 100 011 010 000	0001380
	<b>0112</b>	0096 0097			0000 000 000 111 100	000009B
					0000 000 000 001 110	0000012
	<b>0113</b>	0098	<b>bne</b>	LOOP	1010 000 000 001 000	001B860
	<b>0114</b>	0099			0000 000 000 111 001	0000089
<i>END:</i>	<b>0115</b>	009A	<b>hlt</b>		1111 000 000 000 000	002B680

STR:	0116	009B	.string	"abcdef"	0000 000 001 100 001	0000081
	0117	00A0			0000 000 001 100 010	0000082
	0118	00A1			0000 000 001 100 011	0000083
	0119	00A2			0000 000 001 100 100	0000084
	0120	00A3			0000 000 001 100 101	0000085
	0121	00A4			0000 000 001 100 110	0000086
	0122	00A5			0000 000 000 000 000	0000000
LENGTH:	0123	00A6	.data	6	0000 000 000 000 110	0000006
K:	0124	00A7	.data	2	0000 000 000 000 010	0000002

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

כדי לעשות את אותה פעולה לגבי מענים סמליים, יש צורך לבנות טבלה דומה. אולם הקודים של הפעולות ידועים מראש, ואילו היחס בין הסמלים שבשימוש המתכנת לבין מעני התווית שלהם בזיכרון אינו ידוע, עד אשר התוכנית מקודדת ונקראת על יד המחשב.

בדוגמא שלפנינו אין האסמבלר יכול לדעת שהסמל LOOP משויך למען 0105 (עשרוני או 0089 בבסיס 12), אלא רק לאחר שהתוכנית נקראה כולה.

אי לכך יש שתי פעולות נפרדות שצריך לבצע לגבי כל הסמלים שהוגדרו ביד המתכנת. הראשונה היא לבנות טבלה של כל הסמלים והערכים המספריים המשוויכים להם, והשנייה היא להחליף את כל הסמלים, המופיעים בתוכנית בשדה המען, בערכיהם המספריים. שלבים אלה קשורים בשתי סריקות, מעברים, של קוד המקור. במעבר הראשון נבנית טבלת סמלים בזיכרון, שמותאמים בה מענים לכל הסמלים. בדוגמא דלעיל, טבלת הסמלים לאחר מעבר ראשון היא:

סמל	ערך דצימלי	ערך בבסיס 12
MAIN	0100	0084
LOOP	0105	0089
END	0115	009A
STR	0116	009B
LENGTH	0123	00A6
K	0124	00A7

במעבר השני נעשית ההחלפה כדי לתרגם את הקוד לבינארי. עד אותו זמן צריכים הערכים של כל הסמלים להיות כבר ידועים.

שים לב, שני המעברים של האסמבלר נעשים עוד לפני שתוכנית המשתמש נטענת לזיכרון לצורך הביצוע: כלומר, התרגום נעשה בזמן אסמבלי. שהוא הזמן שבו נמצאת הבקרה בידי האסמבלר.

לאחר השלמת תהליך התרגום יכולה תוכנית המשתמש לעבור לשלב הקישור/טעינה ולאחר מכן לשלב הביצוע. הביצוע נעשה בזמן ריצה.

## המעבר הראשון

במעבר הראשון נדרשים כללים כדי לקבוע איזה ערך מען ישויד לכל סמל. העיקרון הבסיסי הוא לספור את המקומות בזיכרון שאותם צורכת כל הוראה כאשר היא נקראת. אם כל הוראה תיטען לאחר האסמבלי לאתר העוקב של אתר ההוראה הקודמת, תציין ספירה כזאת את מען ההוראה. הספירה נעשית על ידי האסמבלר ומוחזקת באוגר הנקרא מונה אתרים. ערכו ההתחלתי הוא 100, ולכן נטען משפט ההוראה הראשון במען 100. הוא משתנה על ידי כל הוראה, או הוראה מדומה, המקצה מקום. לאחר שהאסמבלר קובע מהו אורך ההוראה, תוכנו של מונה האתרים עולה במספר הבתים הנתפסים על ידי ההוראה, וכך הוא מצביע על התא הריק הבא.

כאמור לעיל, כדי לקודד את ההוראות בשפת מכונה, מחזיק האסמבלר טבלה שיש בה קוד מתאים לכל הוראה. בזמן התרגום מחליף האסמבלר כל הוראה בקוד שלה. אך פעולת ההחלפה

איננה כה פשוטה. יש הרבה הוראות המשתמשות בצורות מיעון שונות. אותה הוראה יכולה לקבל משמעויות שונות בכל אחת מצורות המיעון, ולכן יתאימה לה קודים שונים. לדוגמא, הוראת `mov` יכולה להתייחס להעברת תוכן תא זיכרון לאוגר, או להעברת תוכן אוגר לאוגר, וכן הלאה. לכל צורה כזאת של `mov` מתאים קוד שונה.

על האסמבלר לסרוק את שורת ההוראה בשלמותה, ולהחליט לגבי קוד ההוראה לפי האופרנדים שלה. בדרך כלל מתחלק קוד ההוראה המתורגם לשדה קוד ההוראה ושדות נוספים המכילים מידע לגבי שיטות המיעון.

כך גם במקרה שלנו.

סיביות 12-15 במילה הראשונה של ההוראה מייצגות את קוד ההוראה, וסיביות 3-5 וסיביות 9-11 מייצגות שיטות מיעון.

במחשב שלנו קיימת גמישות לגבי שיטת המיעון של שני האופרנדים. הערה: דבר זה לא מחייב לגבי כל מחשב. ישנם מחשבים שכל הפקודות הן בעלות אופרנד יחיד (והפעולות מתבצעות על אופרנד זה ואוגר קבוע) או מחשבים המאפשרים מבחר של שיטות מיעון עבור אופרנד אחד והאופרנד השני חייב להיות אוגר כלשהו, או מחשבים בעלי 3 אופרנדים, כאשר האופרנד השלישי משמש לאחסון תוצאת הפעולה.

כאשר נתקל האסמבלר בתווית המופיעה בתחילת השורה, הוא יודע שלפניו הגדרה של תווית, ואז ניתן לה מען – תוכנו הנוכחי של מונה האתרים. כך מקבלות כל התוויות את מעניהן בעת ההגדרה. תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנוסף לשם התווית גם את מענה ומאפיינים נוספים שלה, כגון סוג התווית. כאשר תהיה התייחסות לתווית כזאת בשדה המען של ההוראה, יוכל האסמבלר לשלוף את המען המתאים מהטבלה.

כידוע, מתכנת יכול להתייחס גם לסמל שלא הוגדר עד כה בתכנית אלא רק לאחר מכן. לדוגמא: פקודת הסתעפות למען, המופיע בהמשך הקוד:

```
bne    A
```

```
·
·
·
```

A: .....

כאשר מגיע האסמבלר לשורה זו (bne A), הוא עדיין לא הקצה מען לתווית A ולכן אינו יכול להחליף את הסמל A במענו בזיכרון. נראה עתה כיצד נפתרת בעיה זו.

### מעבר שני

בעת המעבר הראשון אין האסמבלר יכול להשלים בטבלה את מעני הסמלים שלא הוגדרו עדיין, והוא מציין אותם באפסים. רק לאחר שהאסמבלר עבר על כל התכנית, כך שכל התוויות הוכנסו כבר לטבלת הסמלים, יכול האסמבלר להחליף את התוויות, המופיעות בשדה המען של ההוראה, במעניהן המתאימים. לשם כך עובר האסמבלר שנית על כל התוכנית, ומחליף את התוויות המופיעות בשדה המען במעניהן המתאימים מתוך הטבלה. זהו המעבר השני, ובסופו תהיה התוכנית מתורגמת בשלמותה.

### אסמבלר של מעבר אחד

יש תוכניות אסמבלר שאינן מבצעות את המעבר השני, והחלפת המענים נעשית בהם בדרך הבאה: בזמן המעבר הראשון שומר האסמבלר טבלה שבה נשמר עבור כל תווית, מען ההוראה שיש בה התייחסות אל התווית בחלק המען.

דוגמא:

נניח שבמען 400 בתוכנית מוגדרת התווית TAB.

נניח גם שבמען 300 מופיע add TAB, r1.

ובמען 500 מופיע jmp TAB.

```
300:      add    TAB, r1
```

```
.....
400:  TAB:  .....
```

```
.....
500:      jmp    TAB
```

האסמבלר יקצה כניסה בטבלה לתווית TAB, ויניח בה את המענים 301 ו-501. (המענים הנשמרים הם 301 ו-501 ולא 300 ו-500 מכיוון ששורת ההוראה מופיעה בכתובות אלה, והמילה



הנוספת מופיעה בכתובת שבאה מיד לאחר מכך). (המענים יכולים להישמר גם בכניסות נפרדות, הדבר תלוי בצורת היישום). בסוף המעבר הראשון ימלא האסמבלר את המענים החסרים בתרגום הקוד מתוך הטבלה. היתרון בשיטה זו הוא כמובן, שהאסמבלר חוסך את המעבר השני על כל התוכנית.

## הפרדת הוראות ונתונים

לכמה תוכניות אסמבלר יש מוני אתרים אחדים. אחד השימושים לכך הוא הפרדת הקוד והנתונים לקטעים שונים בזיכרון, שיטה שהיא עדיפה על פני הצמדה של הגדרות הנתונים להוראות המשתמשות בהן.

אחת הסכנות הטמונות באי הפרדת הקוד מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה קלה, לנסות לבצע את הנתונים. שגיאה שיכולה לגרום זאת היא, למשל, השמטת הוראת עצירה או הסתעפות לא נכונה. אם הנתון שאותו מנסה המעבד לבצע אינו מהווה קוד של הוראה חוקית, תתקבל מיד הודעת שגיאה. אך אילו הנתון נראה כהוראה חוקית, הייתה הבעיה חמורה יותר, משום שהשגיאה לא הייתה מתגלית מיד.

בתוכנית האסמבלר, שעליך לממש, יש להפריד בין קטע הנתונים לקטע ההוראות.

## גילוי שגיאות אסמבלר

האסמבלר יכול לגלות שגיאות בתחביר של השפה, כגון הוראה שאינה קיימת, מספר אופרנדים שגוי, אופרנד שאינו מתאים להוראה ועוד. כן מוודא האסמבלר שכל הסמלים מוגדרים פעם אחת בדיק. מכאן שאת השגיאות המתגלות בידי האסמבלר אפשר לשייך בדרך כלל לשורת קלט מסוימת. אם, לדוגמא, ניתנו שני מענים בהוראה שאמור להיות בה רק מען יחיד, האסמבלר עשוי לתת הודעת שגיאה בנוסח "יותר מדי מענים". בדרך כלל מודפסת הודעה כזאת בתדפיס הפלט באותה שורה או בשורה הבאה, אם כי יש תוכניות אסמבלר המשתמשות בסימון מקוצר כלשהו, ומפרטות את השגיאות בסוף התוכנית.

הטבלה הבאה מכילה מידע על שיטות מיעון חוקיות עבור אופרנד המקור, ואופרנד היעד, עבור הפקודות השונות הקיימות בשפה הנתונה:

פעולה	הקוד בבסיס 12	שיטות מיעון חוקיות עבור אופרנד מקור	שיטות מיעון חוקיות עבור אופרנד יעד
mov	0	0,1,2,3,4	1,2,3,4
cmp	1	0,1,2,3,4	0,1,2,3,4
add	2	0,1,2,3,4	1,2,3,4
sub	3	0,1,2,3,4	1,2,3,4
not	4	אין אופרנד מקור	1,2,3,4
clr	5	אין אופרנד מקור	1,2, 3,4
lea	6	1,2,3	1,2,3,4
inc	7	אין אופרנד מקור	1,2,3,4
dec	8	אין אופרנד מקור	1,2,3,4
jmp	9	אין אופרנד מקור	1,2,3,4
bne	A	אין אופרנד מקור	1,2,3,4
red	B	אין אופרנד מקור	1,2,3,4
prn	10	אין אופרנד מקור	0,1,2,3,4
jsr	11	אין אופרנד מקור	1
rts	12	אין אופרנד מקור	אין אופרנד יעד
hlt	13	אין אופרנד מקור	אין אופרנד יעד

שגיאות נוספות אפשריות הן פקודה לא חוקית, שם רגיסטר לא חוקי, תווית לא חוקית וכו'.

## אלגוריתם כללי

להלן נציג אלגוריתם כללי למעבר הראשון ולמעבר השני: אנו נניח כי הקוד מחולק לשני אזורים, אזור ההוראות (code) ואזור הנתונים (data). נניח כי לכל אזור יש מונה משלו, ונסמנם באותיות IC (מונה ההוראות - Instruction counter) ו-DC (מונה הנתונים - Data counter). האות L תסמן את מספר המילים שתופסת ההוראה.

### מעבר ראשון

1.  $DC \leq 0, IC \leq 0$ .
2. קרא שורה.
3. האם השדה הראשון הוא סמל? אם לא, עבור ל-5.
4. הצב דגל "יש סמל".
5. האם זוהי הוראה מדומה (הנחיה לאחסון נתונים, כלומר, האם הנחית data או string)? אם לא, עבור ל-8.
6. אם יש סמל, הכנס אותו לטבלת הסמלים עם סימון (סמל data). ערכו יהיה DC.
7. זהה את סוג הנתונים, אחסן אותם בזיכרון, עדכן את מונה הנתונים בהתאם לאורכם, חזור ל-2.
8. האם זו הנחית extern או הנחית entry? אם לא, עבור ל-11.
9. האם זוהי הצהרת extern? אם כן, הכנס את הסמלים לטבלת הסמלים החיצוניים, ללא מען.
10. חזור ל-2.
11. אם יש סמל, הכנס אותו לטבלת הסמלים עם סימון (סמל code). ערכו יהיה IC.
12. חפש בטבלת ההוראות, אם לא מצאת – הודע על שגיאה בקוד ההוראה.
13.  $IC \leq L + IC$ .
14. חזור ל-2.

### מעבר שני

1.  $IC \leq 0$ .
2. קרא שורה. אם סיימת, עבור ל-11.
3. אם השדה הראשון הוא סמל, דלג עליו.
4. האם זוהי הוראה מדומה (string, data)? אם כן, חזור ל-2.
5. האם זוהי הנחיה extern, entry? אם לא, עבור ל-7.
6. זהה את ההנחיה. השלם את הפעולה המתאימה לה. אם זאת הנחיית entry. סמן את הסמלים המתאימים כ-entry. חזור ל-2.
7. הערך את האופרנדים, חפש בטבלת ההוראות, החלף את ההוראה בקוד המתאים.
8. אחסן את האופרנדים החל מהבית הבא. אם זהו סמל, מצא את המען בטבלת הסמלים, חשב מענים, קודד שיטת מיעון.
9.  $IC \leq IC + L$ .
10. חזור ל-2.
11. שמור על קובץ נפרד את אורך התוכנית, אורך הנתונים, טבלת סמלים חיצוניים, טבלת סמלים עם סימוני נקודות כניסה.

נפעיל אלגוריתם זה על תוכנית הדוגמא שראינו קודם :

```
MAIN:      mov    LENGTH, r1
           lea     STR[*LENGTH], r4
LOOP:      jmp     END
           prn     STR[K][r3]
           sub     #1, r1
           inc     r0
           mov     r3,STR[*K]
           bne     LOOP
END:       hlt
STR:       .string "abcdef"
LENGTH:    .data   6
K:         .data   2
```

נבצע עתה מעבר ראשון על הקוד הנתון. נבצע במעבר זה גם את החלפת ההוראה בקוד שלה. כמו כן נבנה את טבלת הסמלים. את החלקים שעדיין לא מתורגמים בשלב זה נשאיר כמות שהם. נניח שהקוד ייטען החל מהמען 100 (בבסיס 10).

<i>Label</i>	<b>Decimal Address</b>	<b>Base 12 Address</b>	<b>Command</b>	<b>Operands</b>	<b>Binary machine code</b>	<b>Base 12 machine code</b>
<i>MAIN:</i>	<b>0100</b>	0084	<b>mov</b>	LENGTH, r1	<b>0000 001 000 100 001</b>	00000395
	<b>0101</b>	0085			<b>LENGTH</b>	????????
	<b>0102</b>	0086	<b>lea</b>	STR[*LENGTH], r4	<b>0110 010 000 100 100</b>	0012A04
	<b>0103</b>	0087			<b>STR</b>	????????
	<b>0104</b>	0088			<b>LENGTH</b> מרחק ל-	????????
<i>LOOP:</i>	<b>0105</b>	0089	<b>jmp</b>	END	<b>1001 000 000 001 000</b>	0019408
	<b>0106</b>	008A			<b>END</b>	????????
	<b>0107</b>	008B	<b>prn</b>	STR[K][r3]	<b>1100 000 000 011 011</b>	0024563
	<b>0108</b>	0090			<b>STR</b>	????????
	<b>0109</b>	0091			<b>K</b>	????????
	<b>0110</b>	0092	<b>sub</b>	#1, r1	<b>0011 000 000 100 001</b>	00007169
	<b>0111</b>	0093			<b>0000 000 000 000 001</b>	00000001
	<b>0112</b>	0094	<b>inc</b>	r0	<b>0111 000 000 100 000</b>	0014740
	<b>0113</b>	0095	<b>mov</b>	r3,STR[*k]	<b>0000 100 011 010 000</b>	0001380
	<b>0114</b>	0096			<b>STR</b>	????????
	<b>0115</b>	0097			<b>K</b> מרחק ל-	????????
	<b>0116</b>	0098	<b>bne</b>	LOOP	<b>1010 000 000 001 000</b>	001B860
	<b>0117</b>	0099			<b>0000 000 000 111 001</b>	0000089
<i>END:</i>	<b>0118</b>	009A	<b>hlt</b>		<b>1111 000 000 000 000</b>	002B680
<i>STR:</i>	<b>0119</b>	009B	<b>.string</b>	“abcdef”	<b>0000 000 001 100 001</b>	0000081
	<b>0120</b>	00A0			<b>0000 000 001 100 010</b>	0000082
	<b>0121</b>	00A1			<b>0000 000 001 100 011</b>	0000083
	<b>0122</b>	00A2			<b>0000 000 001 100 100</b>	0000084
	<b>0123</b>	00A3			<b>0000 000 001 100 101</b>	0000085
	<b>0124</b>	00A4			<b>0000 000 001 100 110</b>	0000086
	<b>0125</b>	00A5			<b>0000 000 000 000 000</b>	0000000
<i>LENGTH:</i>	<b>0126</b>	00A6	<b>.data</b>	6	<b>0000 000 000 000 110</b>	0000006
<i>K:</i>	<b>0127</b>	00A7	<b>.data</b>	2	<b>0000 000 000 000 010</b>	0000002

טבלת הסמלים :

סמל	ערך דצימלי	ערך בבסיס 12
MAIN	0100	0084
LOOP	0105	0089
END	0118	009A
STR	0119	009B
LENGTH	0126	00A6
K	0127	00A7

נבצע עתה את המעבר השני ונרשום את הקוד בצורתו הסופית :

<i>Label</i>	<b>Decimal Address</b>	<b>Base 12 Address</b>	<b>Command</b>	<b>Operands</b>	<b>Binary machine code</b>	<b>Base 12 machine code</b>
<i>MAIN:</i>	<b>0100</b>	0084	<b>mov</b>	LENGTH, r1	0000 001 000 100 001	00000395
	<b>0101</b>	0085			0000 000 001 111 011	000000A6
	<b>0102</b>	0086	<b>lea</b>	STR[*LENGTH], r4	0110 010 000 100 100	0012A04
	<b>0103</b>	0087			0000 000 000 111 100	000009B
	<b>0104</b>	0088			0000 000 000 011 000	0000020
<i>LOOP:</i>	<b>0105</b>	0089	<b>jmp</b>	END	1001 000 000 001 000	0019408
	<b>0106</b>	008A			0000 000 000 111 011	000009A
	<b>0107</b>	008B	<b>prn</b>	STR[K][r3]	1100 000 000 011 011	0024563
	<b>0108</b>	0090			0000 000 000 111 100	000009B
	<b>0109</b>	0091			0000 000 001 111 100	00000A7
	<b>0110</b>	0092	<b>sub</b>	#1, r1	0011 000 000 100 001	00007169
	<b>0111</b>	0093			0000 000 000 000 001	00000001
	<b>0112</b>	0094	<b>inc</b>	r0	0111 000 000 100 000	0014740
	<b>0113</b>	0095	<b>mov</b>	r3,STR[*k]	0000 100 011 010 000	0001380
	<b>0114</b>	0096			0000 000 000 111 100	000009B
	<b>0115</b>	0097			0000 000 000 001 110	0000012
	<b>0116</b>	0098	<b>bne</b>	LOOP	1010 000 000 001 000	001B860
	<b>0117</b>	0099			0000 000 000 111 001	0000089
<i>END:</i>	<b>0118</b>	009A	<b>hlt</b>		1111 000 000 000 000	002B680
<i>STR:</i>	<b>0119</b>	009B	<b>.string</b>	"abcdef"	0000 000 001 100 001	0000081
	<b>0120</b>	00A0			0000 000 001 100 010	0000082
	<b>0121</b>	00A1			0000 000 001 100 011	0000083
	<b>0122</b>	00A2			0000 000 001 100 100	0000084
	<b>0123</b>	00A3			0000 000 001 100 101	0000085
	<b>0124</b>	00A4			0000 000 001 100 110	0000086
	<b>0125</b>	00A5			0000 000 000 000 000	0000000
	<b>0126</b>	00A6	<b>.data</b>	6	0000 000 000 000 110	0000006
<i>LENGTH:</i> <i>K:</i>	<b>0127</b>	00A7	<b>.data</b>	2	0000 000 000 000 010	0000002

לאחר סיום עבודת תוכנית האסמבלר, התוכנית נשלחת אל תוכנית הקישור/טעינה.

תפקידיה של תוכנית זו הן :

1. להקצות מקום בזיכרון עבור התוכנית (allocation).
2. לגרום לקישור נכון בין הקבצים השונים של התוכנית (linking).
3. לשנות את כל המענים בהתאם למקום הטעינה (relocation).
4. להטעין את הקוד פיסית לזיכרון (loading).

לא נדון כאן בהרחבה באופן עבודת תוכנית הקישור/טעינה.

לאחר עבודת תוכנית זו, התוכנית טעונה בזיכרון ומוכנה לריצה.

כעת נעיר מספר הערות ספציפיות לגבי המימוש שלכם :

על תוכנית האסמבלר שלכם לקבל כארגומנטים של שורת פקודה (command line arguments) רשימה של קבצי טקסט בהם רשומות הוראות בתחביר של שפת האסמבלר שהוגדרה למעלה. עבור כל קובץ יוצר האסמבלר קובץ מטרה (object). כמו כן ייווצר (עבור אותו קובץ) קובץ

externals באם המקור (source) הצהיר על משתנים חיצוניים, וקובץ entries באם המקור (source) הצהיר על משתנים מסיימים כעל נקודות כניסה.

קבצי המקור של האסמבלר חייבים להיות בעלי הסיומת ".as". השמות x.as, y.as, ו-hello.as הם שמות חוקיים. הפעלת האסמבלר על הקבצים הללו נעשית ללא ציון הסיומת. לדוגמא: אם תוכנית האסמבלר שלנו נקראת assembler, אזי שורת הפקודה הבאה:

```
assembler x y hello
```

תגרום לכך שתוכנית האסמבלר שלנו תקרא את הקבצים: x.as, y.as, hello.as.

האסמבלר יוצר את קבצי ה-object, קבצי ה-entries וקבצי ה-externals על ידי לקיחת שם הקובץ כפי שהופיע בשורת ההוראה והוספת הסיומת "ob" עבור קובץ ה-object, סיומת "ent" עבור קובץ ה-entries, וסיומת "ext" עבור קובץ ה-externals.

מבנה כל קובץ יתואר בהמשך.

לדוגמא: הפקודה: `assembler x` תיצור את הקובץ `x.ob` ואת הקבצים `x.ext` ו-`x.ent` אם קיימים `entries/externals` בקובץ. העבודה על קובץ מסוים נעשית בצורה הבאה:

האסמבלר מחזיק שני מערכים שייקראו להלן מערך הקוד ומערך הנתונים. מערכים אלו נותנים למעשה תמונה של זיכרון המכונה (גודל כל כניסה במערך זהה לגודלה של מילת מכונה – 16 סיביות). במערך הקוד מכניס האסמבלר את הקידוד של הוראות המכונה בהן הוא נתקל במהלך האסמבלי. במערך הנתונים מכניס האסמבלר נתונים המתקבלים תוך כדי האסמבלי (על ידי `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 את פעולתו על קובץ מסוים שני מונים אלו מאופסים. בנוסף יש לאסמבלר טבלת סמלים אשר בה נשמרים המשתנים בהם נתקל האסמבלר במהלך ריצתו על הקובץ. לכל משתנה נשמרים שמו, ערכו וטיפוסו (`external` או `relocatable`).

## אופן פעולת האסמבלר

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

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

כאשר האסמבלר נתקל בשורת פעולה הוא מחליט מהי הפעולה, מהי שיטת המיעון ומי הם האופרנדים. (מספר האופרנדים אותם הוא מחפש נקבע בהתאם לפעולה אותה הוא מצא). האסמבלר קובע לכל אופרנד את ערכו בצורה הבאה:

- אם זה אוגר – ערכו הוא מספר האוגר.
- אם זו תווית – ערכו הוא הערך שלה כפי שהוא מופיע בטבלת הסמלים.
- אם זה מספר (מיעון ישיר) – ערכו הוא הערך של המספר.
- אם זה תווית יחסית (\*) – ערכו הוא המרחק בין מען הפקודה הנוכחית לתווית הרצויה (המרחק יכול להיות שלילי או חיובי בהתאם למיקומה של התווית ביחס לפקודה הנוכחית).

קביעת שיטת המיעון נעשית בהתאם לתחביר של האופרנד כפי שהוא מתואר בחלק העוסק בשיטות המיעון. ככלל התו # מציין מיעון מידי, תווית מציינת מיעון ישיר, שם של אוגר מציין מיעון אוגר, \* לפני תווית מציין מיעון יחסי.

שימו לב: ערך שדה האופרנד הינו ערך תווית המשתנה כפי שהוא מופיע בטבלת הסמלים.

לאחר שהאסמבלר החליט לגבי הדברים הנ"ל (פעולה, שיטת מיעון אופרנד מקור, שיטת מיעון אופרנד יעד, אוגר אופרנד יעד, האם נדרשת מילה נוספת עבור אופרנד מקור באם יש, האם נדרשת מילה נוספת עבור אופרנד יעד באם יש) הוא פועל באופן הבא:

אם זוהי הוראה בעלת שני אופרנדים אזי האסמבלר מכניס למערך הקוד במקום עליו מצביע מונה ההוראות מספר אשר ייצג (בשיטת הייצוג של ההוראות המכונה כפי שתוארו קודם לכן) את קוד הפעולה, שיטות המיעון, ואת המידע על האוגרים. בנוסף הוא "משריין" מקום עבור מספר המילים הנוספות הנדרשות עבור פקודה זו: (הקידוד הארוך ביותר של פקודה הוא 5 מילות זיכרון) ומגדיל את מונה הקוד בהתאם.

דוגמא:

אם ההוראה הייתה  
cmp #-3, r1

אזי במלה הראשונה במערך הקוד יאוחסן הערך (בספרות בינאריות):

0001 000 000 100 001

או בבסיס 12:

00002481

במקום השני במערך הקוד יאוחסן הערך (ספרות בינאריות):

1111 111 111 111 101

או בבסיס 12:

00031B11

שהוא הערך 3- בשיטת המשלים ל-2 עבור מלה בגודל של 16 סיביות.

אם ההוראה היא בעלת אופרנד אחד בלבד, כלומר האופרנד הראשון (אופרנד המקור) אינו מופיע, אזי התרגום הינו זהה לחלוטין (ההוראה אף עשויה לתפוס שתי מילים בזיכרון) למעט העובדה שסיביות 6-11 במלה הראשונה המוכנסת לזיכרון (האמורות לייצג את המידע על אופרנד המקור) יכולות להיות בעלות כל ערך אפשרי מכיוון שערך זה אינו משמש כלל את ה-CPU.

אם ההוראה היא ללא אופרנדים (rts, hlt) אזי למקום במערך הקוד שאליו מצביע מונה ההוראות יוכנס מספר אשר מקודד אך ורק את קוד ההוראה של הפעולה. שיטות המיעון ומידע על האוגרים של אופרנדי המקור והיעד יכולים להיות בעלי ערך כלשהו ללא הגבלה.

אם לשורת הקוד קיימת תווית אזי התווית מוכנסת אל טבלת הסמלים תחת השם המתאים, ערכה הוא ערך מונה ההוראות לפני קידוד ההוראה. טיפוסה הוא relocatable.

3: שורת הנחיה:

כאשר האסמבלר נתקל בהנחיה הוא פועל בהתאם לסוגה באופן הבא:

1. 'data'.

האסמבלר קורא את רשימת המספרים המופיעה לאחר 'data'. הוא מכניס כל מספר שנקרא אל מערך הנתונים ומקדם את מצביע הנתונים באחד עבור כל מספר שהוכנס.



אם ל-`'data'` יש תווית לפנייה אזי תווית זו מוכנסת לטבלת הסמלים. היא מקבלת את הערך של מונה הנתונים לפני שהנתונים הוכנסו אל תוך הקוד + אורך הקוד הכללי. הטיפוס שלה הוא `relocatable`, והגדרתה ניתנה בחלק הנתונים.

## II. `'string'`

ההתנהגות לגבי `'string'` דומה לזו של `'data'`. אלא שקודי ה-`ascii` של התווים הנקראים הם אלו המוכנסים אל מערך הנתונים. לאחר מכן מוכנס הערך 0 (אפס, המציין סוף מחרוזת) אל מערך הנתונים. מונה הנתונים מקודם באורך המחרוזת + 1 (כי גם האפס תופס מקום). ההתנהגות לגבי תווית המופיעה בשורה הינה זהה להתנהגות במקרה של `'data'`.

## III. `'entry'`

זוהי בקשה מן האסמבלר להכניס את התווית המופיעה לאחר `'entry'` אל קובץ ה-`entries`. האסמבלר רושם את הבקשה ובסיום העבודה התווית הני"ל תירשם בקובץ ה-`entries`. `'entry'` באה להכריז על תווית שנעשה בה שימוש בקובץ אחר וכי על תוכנית הקישור להשתמש במידע המצוי בקובץ ה-`entries` ובקובץ ה-`externals` כדי להתאים בין ההתייחסויות ל-`externals`.

## IV. `'extern'`

זוהי בקשה הבאה להצהיר על משתנה המוגדר בקובץ אחר, אשר קטע האסמבלר בקובץ עכשווי עושה בו שימוש.

האסמבלר מכניס את המשתנה המבוקש אל טבלת הסמלים. ערכו הוא אפס (או כל ערך אחר), טיפוסו הוא `external`, היכן נתנה הגדרתו אין יודעים (ואין זה משנה עבור האסמבלר).

יש לשים לב! בפעולה או בהנחיה אפשר להשתמש בשם של משתנה אשר ההצהרה עליו ניתנת בהמשך הקובץ (אם באופן עקיף על ידי תווית ואם באופן מפורש על ידי `extern`).

## פורמט קובץ ה-`object`

האסמבלר בונה את תמונת זיכרון המכונה כך שקידוד ההוראה הראשונה מקובץ האסמבלר יכנס למען 100 (בבסיס 10) בזיכרון, קידוד ההוראה השניה למען שלאחר ההוראה הראשונה (מען 101 או 102 או 103, תלוי באורך ההוראה הראשונה) וכך הלאה עד לתרגום ההוראה האחרונה. מיד לאחר קידוד ההוראה האחרונה, מכילה תמונת הזיכרון את הנתונים שנבנו על ידי הוראות `'data'` ו-`'string'`. הנתונים שיהיו ראשונים הם הנתונים המופיעים ראשונים בקובץ האסמבלר, וכך הלאה.

התייחסות בקובץ האסמבלר למשתנה שהוגדר בקובץ תקודד כך שתצביע על המקום המתאים בתמונת הזיכרון שבונה האסמבלר. עקרונית פורמט של קובץ `object` הינו תמונת הזיכרון הני"ל.

קובץ `object` מורכב משורות שורות של טקסט, השורה הראשונה מכילה, (בבסיס 12) את אורך הקוד (במילות זיכרון) ואת אורך הנתונים (במילות זיכרון). שני המספרים מופרדים ביניהם על ידי רווח. השורות הבאות מתארות את תוכן הזיכרון (שוב, בבסיס 12) החל במען אפס וכלה במען 1) – גודל הנתונים + גודל הקוד).

בהמשך מופיע קובץ `object` לדוגמא ששמו: `ps.obj` המתאים ל-`ps.as` (המספרים המופיעים שם הם: 17, גודל הקוד, ו-9, גודל הנתונים). (המספרים הם מספרים בבסיס 12).

במקרה שבדוגמא בהמשך, הקוד ימצא בין המענים 84 ל-9A (בבסיס 12) והנתונים ימצאו החל במען 9B ועד למען  $A7 = 17 + 9 - 1$  (שוב, כל המספרים בבסיס 12).

בנוסף עבור כל תא זיכרון המכיל הוראה (לא `data`) מופיע מידע עבור תכנית הקישור. מידע זה הינו אחת משלושה התווים 'e' 'a' או 'r'.

האות 'a' מציינת את העובדה שתוכן התא הינו אבסולוטי (absolute) ואינו תלוי היכן באמת יטען הקובץ (האסמבלר יוצא מתוך הנחה שהקובץ נטען החל ממען אפס).

האות 'r' מציינת שתוכן תא הזיכרון הינו relocatable ויש להוסיף לתוכן התא את ההיסט (Offset) המתאים (בהתאם למקום בו יטען הקובץ באופן מעשי). ה-offset הינו מען הזיכרון שבו תטען למעשה ההוראה אשר האסמבלר אומר שעליה להיטען במען אפס.

האות 'e' מציינת שתוכן תא הזיכרון תלוי במשתנה חיצוני external וכי תכנית הקישור תדאג להכנסת הערך המתאים.

### קובץ ה-entries

קובץ ה-entries בנוי משורות טקסט. כל שורה מכילה את שם ה-entry וערכה, כפי שחושב עבור אותו קובץ (ראה לדוגמא את הקובץ ps.ent המתאים לקובץ האסמבלי ששמו ps.as).

### קובץ externals

קובץ ה-externals בנוי אף הוא משורות טקסט. כל שורה מכילה את שם ה-external ואת המען בזיכרון שבו יש התייחסות למשתנה חיצוני זה. למשל בקובץ ps.as שבהמשך מופיעה ההוראה:

```
jsr    REVERSE
```

האופרנד של ההוראה (REVERSE) הינו חיצוני. ערכו יצטרך להיכנס למקום ה-124 (בבסיס 10) בתמונת הזיכרון אותה מכין האסמבלי עבור אותו הקובץ. לכן, בין היתר, מופיעה בקובץ ה-externals ששמו ps.ext השורה:

```
REVERSE    124
```

להלן קובץ PS.AS לדוגמא :

; file ps.as – includes main routine of reversing string “abcdef”

	.entry	STRADD
	.entry	MAIN
	.extern	REVERSE
	.extern	PRTSTR
	.extern	COUNT
STRADD:	.data	0
STR:	.string	“abcdef”
LASTCHAR:	.data	0
LEN:	.data	0
K:	.data	0

; count length of string, print the original string, reverse string, print reversed string.

```

MAIN:      lea          STR[*LEN],STRADD

           jsr          COUNT

           jsr          PRTSTR

           mov          STRADD, LASTCHAR[K][r3]

           mov          STR[*K],r7
           add          COUNT[*LEN],r3

           dec          LASTCHAR[*K]

           inc          K
           jsr          REVERSE

           jsr          PRTSTR

           hlt

```

הקובץ שלהלן הוא קובץ object ששמו בעל סיומת '.ob'. זהו קובץ שהתקבל מהפעלת התוכנית assembler על קובץ האסמבלר דלעיל. כל תוכן הקובץ מיוצג על ידי מספרים בבסיס 12. קובץ ps.ob :

Label	Decimal Address	Base 12 Address	Command	Operands	Binary machine code	Base 12 machine code	Absolute relocatable external
<i>24 B</i>							
(MAIN:)	0100	0084	lea	STR[*LEN],STRADD	0110 010 000 001 000	000129A0	a
	0101	0085			0000 000 010 000 001	000000A9	r
	0102	0086			0000 000 000 100 101	00000031	a
	0103	0087			0000 000 010 000 000	000000A8	r
	0104	0088	jsr	COUNT	1101 000 000 001 000	000269A0	a
	0105	0089			0000 000 000 000 000	00000000	e
	0106	008A	jsr	PRTSTR	1101 000 000 001 000	000269A0	a
	0107	008B			0000 000 000 000 000	00000000	e
	0108	0090	mov	STRADD, LASTCHAR[K][r3]	0000 001 000 011 011	0000038B	a
	0109	0091			0000 000 010 000 000	000000A8	r
	0110	0092			0000 000 010 001 000	000000B4	r
	0111	0093			0000 000 010 001 010	000000B6	r
	0112	0094	mov	STR[*K],r7	0000 010 000 100 111	00000747	a
	0113	0095			0000 000 010 000 001	000000A9	r
	0114	0096			0000 000 000 011 010	00000022	a
	0115	0097	add	COUNT[*LEN],r3	0010 010 000 100 011	0000542B	a
	0116	0098			0000 000 000 000 000	00000000	e
	0117	0099			0000 000 000 010 110	0000001A	a
	0118	009A	dec	LASTCHAR[*K]	1000 000 000 010 000	00016B80	a

	0119	009B			0000 000 010 001 000	000000B4	r
	0120	00A0			0000 000 000 010 100	00000018	a
	0121	00A1	inc	K	0111 000 000 001 000	00014720	a
	0122	00A2			0000 000 010 001 010	000000B6	r
	0123	00A3	jsr	REVERSE	1101 000 000 001 000	000269A0	a
	0124	00A4			0000 000 000 000 000	00000000	e
	0125	00A5	jsr	PRTSTR	1101 000 000 001 000	000269A0	a
	0126	00A6			0000 000 000 000 000	00000000	e
	0127	00A7	hlt		1111 000 000 000 000	0002B680	a
(STRADD:)	0128	00A8	.data	0	0000 000 000 000 000	00000000	
(STR:)	0129	00A9	.string	"abcdef"	0000 000 001 100 001	00000081	
	0130	00AA			0000 000 001 100 010	00000082	
	0131	00AB			0000 000 001 100 011	00000083	
	0132	00B0			0000 000 001 100 100	00000084	
	0133	00B1			0000 000 001 100 101	00000085	
	0134	00B2			0000 000 001 100 110	00000086	
	0135	00B3			0000 000 000 000 000	00000000	
(LASTCHAR:)	0136	00B4	.data	0	0000 000 000 000 000	00000000	
(LEN:)	0137	00B5	.data	0	0000 000 000 000 000	00000000	
(K:)	0138	00B6	.data	0	0000 000 000 000 000	00000000	

כלומר תוכן קובץ ps.ob הוא:

Base 12 Address	Base 12 machine code	Absolute, relocatable or external
24 B		
0084	000129A0	A
0085	000000A9	R
0086	00000031	A
0087	000000A8	R
0088	000269A0	A
0089	00000000	E
008A	000269A0	A
008B	00000000	E
0090	0000038B	A
0091	000000A8	R
0092	000000B4	R
0093	000000B6	R
0094	00000747	A
0095	000000A9	R
0096	00000022	A
0097	0000542B	A
0098	00000000	E
0099	0000001A	A
009A	00016B80	A
009B	000000B4	R
00A0	00000018	A
00A1	00014720	A
00A2	000000B6	R
00A3	000269A0	A
00A4	00000000	E
00A5	000269A0	A
00A6	00000000	E
00A7	0002B680	A
00A8	00000000	
00A9	00000081	
00AA	00000082	
00AB	00000083	
00B0	00000084	
00B1	00000085	
00B2	00000086	
00B3	00000000	
00B4	00000000	
00B5	00000000	
00B6	00000000	

MAIN 0084

STRADD 00A8

קובץ: ps.ent

COUNT 0089

PRTSTR 008B  
COUNT 0098

קובץ: ps.ext

REVERSE 00A4

PRTSTR 00A6

לתשומת לבך : אם בקובץ מסויים אין הצהרת *extern*, אזי לא ייוצר עבורו קובץ *ext*. המתאים.  
כנ"ל עבור קבצים שאינם מכילים הודעות *entry*, במקרה זה לא ייוצר קובץ *ent*. מתאים.

## סיכום והנחיות כלליות

- אורך התוכנית, הניתנת כקלט לאסמבלר, אינו ידוע מראש (ואינו קשור לגודל 2000 – של הזיכרון במעבד הדמיוני). ולכן אורכה של התוכנית המתורגמת, אינו אמור להיות צפוי מראש. אולם בכדי להקל במימוש התוכנית, ניתן להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים.
- קבצי הפלט של התוכנית, צריכים להיות בפורמט המופיע למעלה. שמם של קבצי הפלט צריך להיות תואם לשמה של תוכנית הקלט, פרט לסיומות. למשל, אם תוכנית הקלט היא *prog.as* אזי קבצי הפלט שייווצרו הם : *prog.ob*, *prog.ext*, *prog.ent*.
- אופן הרצת התוכנית צריך להיות תואם לנדרש בממ"ן, ללא שינויים כלשהם. אין להוסיף תפריטים למיניהם וכדומה. הפעלת התוכנית תיעשה רק ע"י ארגומנטים של שורת פקודה.
- יש להקפיד לחלק את התוכנית למודולים. אין לרכז מספר מטרות במודול יחיד. מומלץ לחלק למודולים כגון : מבני נתונים, מעבר ראשון, מעבר שני, טבלת סמלים וכדומה.
- יש להקפיד ולתעד את הקוד, בצורה מלאה וברורה.
- יש להקפיד על התעלמות מרווחים, ולהיות סלחנים כלפי תוכניות קלט, העושות שימוש ביותר רווחים מהנדרש. למשל, אם לפקודה ישנם שני אופרנדים המופרדים בפסיק, אזי לפני שם הפקודה או לאחריה או לאחר האופרנד הראשון או לאחר הפסיק, יכול להיות מספר רווחים כלשהו, ובכל המקרים זו צריכה להיחשב פקודה חוקית (לפחות מבחינת הרווחים).
- במקרה של תוכנית קלט, המכילה שגיאות תחביריות, נדרש להפיק, כמו באסמבלר אמיתי, את כל השגיאות הקיימות, ולא לעצור לאחר היתקלות בשגיאה הראשונה. כמובן שעבור קובץ שגוי תחבירי, אין להפיק את קבצי הפלט (*ob*, *ext*, *ent*) אלא רק לדווח על השגיאות שנמצאו.

תם ונשלם חלק ההסברים והגדרת הפרוייקט.

בשאלות ניתן לפנות אל :

**קבוצת הדיון באתר הקורס, לכל אחד מהמנחים בשעות הקבלה שלהם. להזכירכם, באפשרותו של כל סטודנט לפנות לכל מנחה, לאו דווקא למנחה הקבוצה שלו לקבלת עזרה. שוב מומלץ לכל אלה מכם שטרם בדקו את אתר הקורס, לעשות זאת. נשאלות באתר זה הרבה שאלות בנושא חומר הלימוד ופתרון הממ"נים, והתשובות יכולות לעזור לכולם.**  
לתשומת לבכם, לא יתקבלו ממ"נים באיחור ללא תיאום מראש עם מרכזת הקורס. ממ"נים שיוגשו באיחור ללא אישור, יקבלו ציון 0.

בהצלחה!!!!