ה א ו נ י ב ר ס י ט ה ה פ ת ו ח ה

20465
מעבדה בתכנות מערכות
חוברת הקורס – סתיו 2016א

כתבה: מיכל אבימור

אוקטובר 2015 – סמסטר סתיו – תשע"ו

פנימי – לא להפצה.

תוכן העניינים

אל הסטודנט,

אני מקדמת את פניך בברכה, עם הצטרפותך אל הלומדים בקורס "מעבדה בתכנות מערכות". בחוברת זו תמצא את הדרישות לקבלת נקודות זכות בקורס, לוח הזמנים ומטלות הקורס.

לקורס קיים אתר באינטרנט בו תמצאו חומרי למידה נוספים, אותם מפרסם/מת מרכז/ת ההוראה. בנוסף, האתר מהווה עבורכם ערוץ תקשורת עם צוות ההוראה ועם סטודנטים אחרים בקורס.
פרטים על למידה מתוקשבת ואתר הקורס, תמצאו באתר שה"ם בכתובת:
http://telem.openu.ac.il
מידע על שירותי ספרייה ומקורות מידע שהאוניברסיטה מעמידה לרשותכם, תמצאו באתר הספריה באינטרנט www.openu.ac.il/Library.

קורס זה הינו קורס מתוקשב. מידע על אופן ההשתתפות בתקשוב ישלח לכל סטודנט באופן אישי. ניתן להפנות שאלות בנושאי חומר הלימוד, והממ"נים לקבוצת הדיון של הקורס. בנוסף יופיעו שם הודעות ועדכונים מצוות הקורס. כניסה תכופה לאתר הקורס ולקבוצת הדיון שלה, מאפשרת לך להתעדכן בכל המידע, ההבהרות וכו' במסגרת הקורס.

ניתן לפנות אלי בשעות הייעוץ שלי (יפורסמו בהמשך באתר) או מחוץ לשעות הקבלה, באמצעות email, לכתובת: michav@openu.ac.il, ואשתדל לענות בהקדם.

אני מאחלת לך לימוד פורה ומהנה.

בברכה,

מיכל אבימור
מרכזת ההוראה בקורס.

1. לוח זמנים ופעילויות (20465 /א2016)

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
1	23.10.2015-18.10.2015	ספר C פרקים 3-2-1	מפגש ראשון	
2	30.10.2015-25.10.2015	ספר C פרקים 3-2-1		
3	6.11.2015-1.11.2015	ספר C פרק 4	מפגש שני	
4	13.11.2015-8.11.2015	ספר C פרק 4		
5	20.11.2015-15.11.2015	ספר C פרק 5	מפגש שלישי	ממ"ן 11 15.11.2015
6	27.11.2015-22.11.2015	ספר C פרק 5		
7	4.12.2015-29.11.2015	ספר C פרק 6	מפגש רביעי	
8	11.12.2015-6.12.2015 (ב-ו חנוכה)	ספר C פרק 6		ממ"ן 21 6.12.2015

** התאריכים המדויקים של המפגשים הקבוצתיים מופיעים ב"לוח מפגשים ומנחים".*

לוח זמנים ופעילויות - המשך

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
9	18.12.2015-13.12.2015 (א-ב חנוכה)	ספר C פרק 6		
10	25.12.2015-20.12.2015	ספר C פרק 7	מפגש חמישי	
11	1.1.2016-27.12.2015	ספר C פרק 7		ממ"ן 22 27.12.2015
12	8.1.2016-3.1.2016	ספר C פרק 8 + פרויקט	מפגש שישי	
13	15.1.2016-10.1.2016	פרויקט וחזרה		
14	22.1.2016-17.1.2016	פרויקט וחזרה	מפגש שביעי	ממ"ן 13 17.1.2016
15	29.1.2016-24.1.2016	פרויקט וחזרה		ממ"ן 14** 20.3.2016

מועדי בחינות הגמר יפורסמו בנפרד

**** התאריכים המדויקים של המפגשים הקבוצתיים מופיעים ב"לוח מפגשים ומנחים".***

***** לא תינתן דחייה בהגשת הפרויקט, פרט למקרים של מילואים או מחלה, במקרים אלו יש לתאם את מועד ההגשה עם צוות הקורס.***

2. תיאור המטלות

על מנת לתרגל את החומר הנלמד ולבדוק את ידיעותיך, עליך לפתור את המטלות המצויות בחוברת המטלות.

רוב המטלות בקורס זה הנן **מטלות חובה,** והן בעיקרן תוכניות מחשב. שתי מטלות הן רשות.
להלן מספרי המטלות ומשקליהן:

ממ"ן	משקל	פרקים
11	4 (ממ"ן חובה)	3,2,1
21	5 (ממ"ן חובה)	5,4
22	8 (ממ"ן רשות)	6,5,4
13	12 (ממ"ן רשות)	8,7,6
14	31 (ממ"ן חובה)	פרויקט גמר

עליך להגיש במהלך הקורס את כל מטלות החובה.
את התשובות לממ"נים יש להגיש באמצעות מערכת המטלות (במקרים מיוחדים ניתן להגיש את המטלות באמצעות הדואר או הגשה ישירה למנחה במפגשי ההנחיה. במקרה כזה יש לתאם את הדבר עם הבודק).
יש להגיש את קבצי המקור (.c, .h), קבצי ההרצה, קבצי הסביבה המתאימים (כולל קבצי MAKEFILE), קבצי קלט וקבצי פלט (או צילומי מסך, אם לא נדרשו הקבצים הנ"ל).

הנחיות לכתיבת מטלות וניקודן

ניקוד המטלות ייעשה לפי המשקלים הבאים:

א. ריצה - 20%
התכנית עובדת על פי הדרישות בתרגיל, תוך השגת **כל** המטרות שהוגדרו. התכנית עוברת קומפילציה ללא הערות.

ב. תיעוד - 20%
התיעוד ייכתב בתוך הקוד. אין להוסיף הערות בקבצים נפרדים.
התיעוד יכלול:

1) הערה בראש תכנית שתכלול תיאור תמציתי של מטרות התכנית, כיצד מושגת מטרה זו, תיאור המודלים והאלגוריתם, קלט/פלט **וכל הנחה** שהנכם מניחים.
2) לכל מציג (אב-טיפוס) prototype של פונקציה (בקובץ ה-header הצמוד לקוד), יוצמד תיאור של קלט/פלט, ופעולת הפונקציה. **מטרה:** זהו קובץ היצוא ועל כן עליו להסביר למי שאין לו גישה לקוד איך עליו להשתמש בפונקציה.
3) לפני הכותרת (header) של כל פונקציה יבוא תיאור של פעולתה, הנחות ואלגוריתם.

מטרה: התיעוד לפני כל פונקציה נועד לתת היכרות ראשונית, לפעולת הפונקציה, תוך פירוט כיצד הפונקציה עושה זאת. תיעוד זה אמור לאפשר לקורא את הקוד (שלא כתב את הקוד), להבין את הקוד.

4) לכל משתנה יהיה שם משמעותי ויוצמד אליו תיעוד לגבי תפקודו בתכנית. i,j,k - משמשים בד"כ כשמות אינדקסים ואין צורך לתעד אותם.

5) לא יופיעו "מספרי קסם" בגוף התכנית למעט 0,1 לאיתחול משתני לולאות. יש להשתמש בקבועים בעלי שמות משמעותיים שיכתבו באותיות גדולות, ויתועדו בשלב ההגדרה. כל טיפוס מורכב יוגדר כ- typedef ויתועד. נהוג לקרוא לטיפוסים מורכבים בשמות משמעותיים ולהשתמש באותיות גדולות.

6) יש להשתמש בשמות משמעותיים ל: פונקציות, מקרואים, משתנים, קבועים, הגדרת טיפוסים וקבצים.

7) יש להקפיד על קריאות ובהירות תוך שימוש באינדנטציה (היסח) מסודרת ואחידה.

ג. תכנות - 40%

יש להקפיד על כתיבה מסודרת ומודולרית של קוד:

- חלוקה לקבצים - כשלכל קובץ מוצמד קובץ header אם צריך (כאשר נדרש בתרגיל).
- חלוקה לפונקציות.
- שימוש במקרואים.
- שימוש נכון ב-MAKEFILE, (במיוחד כאשר אתם נדרשים לחלק את התוכנית למספר קבצים, במסגרת הממ"ן).
- הסתרת אינפורמציה - ושימוש בהפשטת מידע.
- הימנעות ככל שניתן משימוש במשתנים גלובליים.
- שימוש מירבי ונכון במלוא הכלים שמעמידה השפה לרשותכם.
- קוד אלגנטי ולא מסורבל.

ד. יעילות התכנית והתרשמות כללית - 20%

המשקלים הנ"ל מהווים קו מנחה לחלוקת הנקודות. מובן שתהיה התייחסות לכל תכנית לגופה, בהתאם למידת המורכבות של התרגיל.

ינתנו קנסות במיקרים הבאים:

- אי הגשת קבצי סביבה – MAKEFILE – 20 נקודות.
- עבור אותם ממ"נים בהם מוגדר שם קובץ, פונקציה, או פרמטר, שימוש בשם שונה מזה המוגדר בממ"ן – 10 נקודות.

להשומת לבך: חל איסור מוחלט של הכנה משותפת של מטלות או העתקת מטלות. תלמיד שייתפס באחד מאיסורים אלה ייענש בהתאם לנאמר בתקנון המשמעת נספח 1 בידיעון של האו"פ. רק את ממ"ן 14 מותר להגשה בזוגות (**לא** ניתן להגיש בשלשות!), כאשר שני הסטודנטים המגישים שיכים לאותה קבוצת לימוד.

3. התנאים לקבלת נקודות זכות בקורס

א. להגיש את מטלות החובה בקורס (11, 21) וכן את פרויקט הגמר (14).

ב. ציון של לפחות 60 נקודות בבחינת הגמר.

ג. ציון סופי בקורס של 60 נקודות לפחות.

לתשומת לבכם!

כדי לעודדכם להגיש לבדיקה מספר רב של מטלות הנהגנו את ההקלה שלהלן:

אם הגשתם מטלות מעל למשקל המינימלי הנדרש בקורס, **המטלות** בציון הנמוך ביותר, שציוניהן נמוכים מציון הבחינה (**עד שתי מטלות**), לא יילקחו בחשבון בעת שקלול הציון הסופי.

זאת בתנאי שמטלות אלה **אינן חלק מדרישות החובה בקורס** ושהמשקל הצבור של המטלות האחרות שהוגשו, מגיע למינימום הנדרש.

זכרו! ציון סופי מחושב רק לסטודנטים שעברו את בחינת הגמר בציון 60 ומעלה והגישו מטלות כנדרש באותו קורס.

מטלת מנחה (ממ"ן) 11

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 1,2,3

מספר השאלות: 2 **משקל המטלה:** 4 נקודות (**חובה**)

סמסטר: 2016א' **מועד אחרון להגשה:** 15.11.2015

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"

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: -Wall -ansi -pedantic נדרש להגיש את כל קבצי המקור שכתבתם (.h, .c), קבצי ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי MAKEFILE), קבצי קלט וקבצי פלט (לפי הנחייה במפגש / באתר).

נדרש ששם הספריה ושם הקובץ לריצה יהיו כשם קובץ התוכנית הראשית, ללא הסיומת c.

את המטלה יש להגיש בקובץ **zip**. כל שאלה בספריה נפרדת, אך הדחיסה בקובץ **zip** אחד.

לאחר ההגשה, יש **להוריד** את המטלה משרת האו"פ למחשב האישי, ולבדוק שהקבצים אכן הוגשו באופן **תקין**.

שאלה 1 (תכנית ראשית בקובץ bitcount.c) (50 נקודות)

עליכם לכתוב פונקציה בשם bit_count , אשר סופרת את מספר הסיביות הדלוקות (אלו שערכן 1) במספר שלם, integer, אותו היא מקבלת כפרמטר. על הפונקציה להיות ניידת - portable, כלומר עליה להיות בלתי תלויה, באופן הייצוג של מספר שלם במחשב או במהדר (compiler) מסוים. אופן ייצוג הנו מספר הבתים – bytes, המשמשים לייצוג של מספר שלם - integer.

שאלה 2 (תכנית ראשית בקובץ shiftbit.c) (50 נקודות)

תוך שימוש בפונקציה משאלה 1, עליכם לכתוב פונקציה המקבלת כפרמטר מספר שלם, integer, ומחזירה מספר בעל אותו מספר סיביות דלוקות, אך כולן מיושרות לשמאל.

<u>לדוגמא</u>:

נניח שבמחשב שלנו מיוצג מספר שלם בשני בתים. נניח שהפרמטר שלנו הוא המספר 164, שהייצוג הבינארי שלו הוא: 0000000010100100. על הפונקציה להחזיר 1110000000000000. אין להניח כל הנחה לגבי אופן הייצוג של מספר שלם במחשב או במהדר - compiler, שעליו תרוץ התוכנית.

<u>להזכירכם</u>: לא תנתן דחייה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או מחלה, במקרים אלו יש לקבל אישור הגשה מצוות הקורס.

בהצלחה!

מטלת מנחה (ממ"ן) 21

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 4,5

מספר השאלות: 1 **משקל המטלה:** 5 נקודות (**חובה**)

סמסטר: 2016א' **מועד אחרון להגשה:** 6.12.2015

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"

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: -Wall -ansi -pedantic נדרש להגיש את קבצי המקור (.c, .h), קבצי ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי MAKEFILE), קבצי קלט וקבצי פלט (לפי הנחייה במפגש / באתר).

נדרש ששם הספריה ושם הקובץ לריצה יהיו כשם קובץ התוכנית הראשית, ללא הסיומת c.

את המטלה יש להגיש בקובץ **zip**. כל שאלה בספריה נפרדת, אך הדחיסה בקובץ **zip** אחד.

לאחר ההגשה יש **להוריד** את המטלה משרת האו"פ למחשב האישי, ולבדוק שהקבצים אכן הוגשו באופן **תקין**.

שאלה 1 (תכנית ראשית בקובץ mem_cmp.c)

עליכם **לממש** את הפונקציה הבאה:

```
int memcmp(const void *p1, const void *p2, int count);
```

p1 -מצביע לקטע זיכרון ראשון.

p2 -מצביע לקטע זיכרון שני.

count -מספר הבתים (bytes) שיש להשוות.

<u>מטרת הפונקציה</u>: על הפונקציה memcmp להשוות את count הבתים מהמקום ש-p1 מצביע אליו אל count הבתים מהמקום ש-p2 מצביע אליו.

<u>הערך המוחזר</u>: ההשוואה המבוצעת היא השוואה לקסיקוגרפית, כלומר הערך המוחזר מציין את הסדר בו שתי קבוצות תווים אלה מופיעות במילון (אין להניח שקבוצת התווים מסתיימת בתו '\0').

פרוש הערך המוחזר הוא:

קטן מאפס - קבוצת התווים הראשונה (p1 מצביע אליו) "קטן" (יופיע קודם במילון) מזו בקבוצה השניה (p2).

שווה לאפס - שתי הקבוצות זהות ב-count התווים הראשונים שלהן.

גדול מאפס - p1 "גדול" מ-p2.

להזכירכם: לא תנתן דחייה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או מחלה, במקרים אלו יש לקבל אישור הגשה מצוות הקורס.

בהצלחה!

מטלת מנחה (ממ"ן) 22

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 4,5,6

מספר השאלות: 1 **משקל המטלה:** 8 נקודות (רשות)

סמסטר: 2016א' **מועד אחרון להגשה:** 27.12.2015

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"

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: Wall -ansi -pedantic- נדרש להגיש את קבצי המקור (.h, .c), קבצי ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי MAKEFILE), קבצי קלט וקבצי פלט (לפי הנחייה במפגש / באתר). נדרש ששם הספריה ושם הקובץ לריצה יהיו כשם קובץ התוכנית הראשית, ללא הסיומת c.

את המטלה יש להגיש בקובץ **zip**. כל שאלה בספריה נפרדת, אך הדחיסה בקובץ **zip** אחד.

לאחר ההגשה יש **להוריד** את המטלה משרת האו"פ למחשב האישי, ולבדוק שהקבצים אכן הוגשו באופן **תקין**.

שאלה 1 (בקבצים עיקריים main.c, complex.c, complex.h)

<u>תזכורת מספרים מרוכבים:</u>

מספר מרוכב הוא מספר המורכב ממספר ממשי ומספר מדומה, כאשר ביניהם רשום הסימן "+" או הסימן "-".

לדוגמה: a + bi, כאשר a המספר הממשי ו-bi המספר המדומה.

מספר מדומה מורכב ממכפלת שני איברים: b מספר ממשי כלשהו, i שורש ריבועי של המספר 1-.

$i = \sqrt{-1}$

<u>להלן הפעולות החשבוניות הבסיסיות על מספרים מרוכבים:</u>

חיבור בין שני מספרים מרוכבים:

(a + bi) + (c + di) = (a + c) + (b + d) i

חיסור בין שני מספרים מרוכבים:

(a + bi) - (c + di) = (a - c) + (b - d) i

כפל של מספר מרוכב עם מספר ממשי:

m∗ (a + bi) =ma + mbi

כפל של מספר מרוכב ומספר מדומה:

mi * (a + bi) = mia + mibi = ami + bmii = -bm + ami

כפל של מספר מרוכב במספר מרוכב:

(a + bi) * (c + di) = ac + adi + bic + bidi = (ac - bd) + (ad + bc) i

הערך המוחלט של מספר מרוכב $a + bi$ הוא המספר הממשי החיובי:

$$|a+bi| = \sqrt{a^2+b^2}$$

עליכם לכתוב תכנית מחשב אינטראקטיבית הקוראת פקודות, מפענחת ומבצעת אותן.
הפקודות עוסקות בפעולות על מספרים מרוכבים (תזכורת למעלה). עליכם להגדיר, תוך השימוש בפקודת typedef את הטיפוס complex אשר מסוגל להחזיק מספר מרוכב. על מבנה הנתונים שבחרתם להיות יעיל מבחינת כמות זיכרון הנדרשת לשמירתו ויעיל מבחינת הגישה אליו.

בנוסף עליכם להגדיר 6 משתנים חיצוניים A,B,C,D,E,F מטיפוס זה.

כל שם של מספר מרוכב בפקודות שלהלן יילקח מתוך השישה הנ"ל.

הפקודות המותרות כקלט לתכנית:

1. **מספר ממשי, מספר ממשי, שם-מספר-מרוכב read_comp**

הפקודה תגרום לקריאת הערכים של המספר המרוכב, לתוך המספר המרוכב ששמו ניתן בפקודה כפרמטר ראשון. מותר לכם להניח שהמספר הממשי הראשון הוא החלק הממשי של המספר המרוכב והמספר הממשי השני הוא המקדם של החלק המדומה של המספרהמרוכב.

לדוגמה, הפקודה הבאה:

read_comp A, 5.1, 6.2

מייצגת את המספר המרוכב:

A = 5.1 + (6.2)i

2. **שם- מספר- מרוכב print_comp**

המספר המרוכב ששמו ניתן יודפס בצורה נאה בפלט.

3. חיבור מספרים מרוכבים:

שם-מספר-מרוכב-ב',שם-מספר-מרוכב-א' **add_comp**

תתבצע הפעולה הבאה:

מספר-מרוכב-ב' + מספר-מרוכב-א'

תוצאת הפעולה תודפס לפלט בפורמט זהה לפורמט ההדפסה של סעיף 2.

4. חיסור מספרים מרוכבים:

שם-מספר-מרוכב-ב',שם-מספר-מרוכב-א' **sub_comp**

תתבצע הפעולה הבאה:

מספר-מרוכב-ב' - מספר-מרוכב-א'

תוצאת הפעולה תודפס לפלט בפורמט זהה לפורמט ההדפסה של סעיף 2.

5. כפל מספר מרוכב עם מספר ממשי:

מספר ממשי,שם-מספר-מרוכב-א' **mult_comp_real**

תתבצע הפעולה הבאה:

מספר-ממשי * מספר-מרוכב-א'

תוצאת הפעולה תודפס לפלט בפורמט זהה לפורמט ההדפסה של סעיף 2.

6. כפל מספר מרוכב עם מספר מדומה:

מספר ממשי,שם-מספר-מרוכב-א' **mult_comp_img**

תתבצע הפעולה הבאה:

i *מספר-ממשי * מספר-מרוכב-א'

תוצאת הפעולה תודפס לפלט בפורמט זהה לפורמט ההדפסה של סעיף 2.

7. כפל מספר מרוכב אחד עם מספר מרוכב שני:

מספר-מרוכב-ב' ,שם-מספר-מרוכב-א' **mult_comp_comp**

תתבצע הפעולה הבאה:

מספר-מרוכב-ב' * מספר-מרוכב-א'

תוצאת הפעולה תודפס לפלט בפורמט זהה לפורמט ההדפסה של סעיף 2.

8. חישוב ערך מוחלט של מספר מרוכב:

שם-מספר-מרוכב-א' **abs_comp**

תתבצע הפעולה הבאה:

| מספר-מרוכב א' |

תוצאת הפעולה תודפס לפלט בפורמט זהה לפורמט ההדפסה של סעיף 2.

9. **halt**:

התכנית תפסיק לרוץ ותצא לרמת מערכת ההפעלה.
התכנית צריכה לתת סימן (prompt) על המסך שמודיע שהיא מוכנה לקבל קלט.
התכנית תמשיך לעבוד עד שתקבל את פקודת halt.

התכנית אינה מניחה נכונות הקלט ויש להודיע על שגיאות בקלט.

<u>הערה</u>: ניתן להיעזר בתוכנית בקובץ תוכניות לדוגמא. לתשומת לבכם, פתרון שיבנה בדומה לפתרון המוצע בקובץ תוכניות לדוגמא, יכלול נושאים המופיעים בפרקים שהם מחוץ לתחום של ממ"ן זה, לכן ניתן כמובן להגיש מימוש פשוט יותר.

<u>דוגמאות</u>:

לפקודה:

read_comp W, 3.2, 8

יש להגיב בהודעה:

"No such complex number"

לפקודה:

kkkk A,B

יש להגיב בהודעה:

"No such command"

לפקודה:

read_comp A, B,567

יש להגיב:

"Wrong parameters, second parameter must be a real number"

וכו'...

<u>דוגמאות לקלט תקין</u>:

```
read_comp A,45.1,23.7
read_comp B,54.2,3.56
print_comp A
add_comp A, B
sub_comp C, A
mul_comp_real A, 2.5
mult_comp_img A, 2.5
mult_comp_comp A, B
abs A
abs B
abs C
halt
```

יש לחלק את התוכנית למספר קבצים: main.c, complex.c, ו-complex.h.

בקובץ complex.c יש לרכז את הפונקציות החישוביות ובקובץ main.c את פעילויות האינטראקציה.

קובץ complex.c מייצא את אבות הטיפוס של הפונקציות הקיימות בו באמצעות complex.h.

יש לכתוב לתכנית ממשק סביר, כך שהמשתמש יוכל להבין מה עליו לעשות, על מנת להריץ את התוכנית.

להזכירכם: לא תנתן דחייה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או מחלה, במקרים אלו יש לקבל אישור הגשה מצוות הקורס.

בהצלחה!

מטלת מנחה (ממ"ן) 13

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 6,7,8

מספר השאלות: 2 **משקל המטלה**: 12 נקודות (רשות)

סמסטר: 2016א' **מועד אחרון להגשה:** 17.1.2016

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"

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: **-Wall -ansi -pedantic** נדרש להגיש את קבצי המקור (**.c**, **.h**), קבצי ההרצה (את קבצי **o**. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי **MAKEFILE**), קבצי קלט וקבצי פלט (לפי הנחייה במפגש / באתר).

נדרש ששם הספריה ושם הקובץ לריצה יהיו כשם קובץ התוכנית הראשית, ללא הסיומת c.

את המטלה יש להגיש בקובץ zip. כל שאלה בספריה נפרדת, אך הדחיסה בקובץ zip אחד.

לאחר ההגשה יש **להוריד** את המטלה משרת האו"פ למחשב האישי, ולבדוק שהקבצים אכן הוגשו באופן **תקין**.

שאלה 1 (12 נקודות)

בכל סעיף עליך לכתוב האם נכון, לא נכון, לפעמים נכון. עליך לנמק את תשובתך. תשובה לא מנומקת, גם אם היא נכונה, לא תזכה בנקודות. (כל סעיף 4 נקודות).

א. הפונקציה הנקראת יכולה לשנות את ערכי הפרמטרים, המועברים אליה, על ידי הפונקציה הקוראת.

ב. אם נגדיר משתנה כ-**static register**, נוכל לייעל את זמן השימוש במשתנה, וכן לשמור על ערכו מקריאה לקריאה.

ג. הביטוי:

#define STR1 'a'

שקול למעשה לביטוי:

#define STR1 "a"

שאלה 2 (88 נקודות) (תכנית ראשית בקובץ seek.c)

עליכם לכתוב תוכנית המדפיסה את התו ה-**n**-י מתחילת כל קובץ מתוך רשימת קבצים נתונה.

על התכנית לקבל כארגומנטים של שורת פקודה:

- מספר **n**
- רשימה של שמות קבצי קלט

לדוגמא, אם התכנית נקראת **n_char**, קריאה לתכנית מתוך מערכת ההפעלה, יכולה להיות:

> **n_char**760 **file1.in file2.in file3.in**

במקרה זה יש להדפיס את התו ה-760 מתחילת כל אחד משלושת הקבצים:

file1.in

file2.in

file3.in

<u>הערה</u>: גם תוים מיוחדים כגון **\n** **\t** וכדומה נחשבים תוים לצורך המניה. תווים אלו עשויים להיות שקולים לשני תווים רגילים (למשל **\n** גורר את 10+13).

על מנת להגיע לתו ה-**n**-י בקובץ נתון, אין להשתמש בלולאה הקוראת **n** תווים מהקובץ, אלא יש להשתמש בפונקציות **fseek**.

על התכנית לבצע:

1. להודיע הודעות שגיאה ולהפסיק ריצה במקרים הבאים:
   - ארגומנטים של שורת פקודה לא מתאימים
   - אי אפשר לפתוח את אחד מקבצי הקלט
2. להודיע הודעה מתאימה ולהמשיך את ריצת התכנית, במקרה ש-**n** גדול ממספר התווים הקיימים בקובץ נתון.

<u>להזכירכם</u>: לא תנתן דחייה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או מחלה, במקרים אלו יש לקבל אישור הגשה מצוות הקורס.

בהצלחה!

מטלת מנחה (ממ"ן) 14

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרויקט גמר

מספר השאלות: 1 | משקל המטלה: 31 נקודות

סמסטר: 2016א' | מועד אחרון להגשה: 20.3.2016

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.

עליך לכתוב תוכנת אסמבלר, לשפת אסמבלי שתוגדר בהמשך. הפרויקט ייכתב בשפת C. אין להוסיף ספריות חיצוניות: ניתן להשתמש רק בספריות, מתוך הספריה הסטנדרטית.
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.

בשל גודל הפרויקט, עליך לחלק את התוכנית למספר קבצי מקור. יש להקפיד שהקוד הנמצא בתוכניות המקור יעמוד בקריטריונים של בהירות, קריאות וכתיבה 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 מ"מספרי 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.

<u>הערה</u>: 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.

<u>רקע כללי</u>

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

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מספר מסוים של הוראות פשוטות, ולשם כך היא משתמשת בכמה אוגרים (register) הקיימים בה. <u>דוגמאות:</u> 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. בפרויקט זה עליך לכתוב אסמבלר. לשם כך נעקוב אחרי גלגולה של תוכנית שנכתבה בשפת אסמבלי, שנגדיר במיוחד עבור פרויקט זה, עד לשליחתה לתוכנת הקישור והטעינה (linker/loader).

<u>לתשומת לבך</u>: 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. אין 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 פרויקט זה.

"חומרה":

המחשב בפרויקט מורכב מיע"מ (יחידת עיבוד מרכזית), אוגרים וזיכרון RAM, כאשר חלק מהזיכרון משמש גם כמחסנית (stack). גודלה של מלת זיכרון במחשב הוא 12 סיביות.
האריתמטיקה נעשית בשיטת המשלים ל -2 (2's complement). מחשב זה מטפל רק במספרים שלמים חיוביים ושליליים, 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 15 סיביות.

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

C = (PSW & 01)
Z = (PSW & 02)

גודל הזיכרון הוא 1000 תאים, וכל תא הוא בגודל של 15 סיביות.

קידוד של תווים (characters) נעשה בקוד ascii.

אפיון מבנה הוראת מכונה:

כל הוראת מכונה מקודדת למספר מילות זיכרון, החל מ- תא אחד ועד למקסימום של 3 תאי זיכרון, הכל בהתאם לשיטות המיעון בהן נעשה שימוש. בכל סוגי ההוראות, המבנה של המילה הראשונה זהה. מבנה המילה הראשונה בהוראה הוא כדלהלן:

14	13 12	11 10	9 8 7 6	5 4	3 2	1 0
לא בשימוש	rnd	group	opcode	מיעון אופרנד מקור	מיעון אופרנד יעד	E,R,A

קידוד ההוראות ייעשה לבסיס 32. שהוא בסיס המורכב מ 0-9A-V (כלומר מהספרות 0 עד 9 ולאחריהן האותיות A עד V)
סיביות 6-9 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, והן מתווספות גם למילים 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. אם שיטת המיעון של רק אחד משני האופרנדים, דורשת מילות מידע נוספות, אזי מילות המידע הנוספות מתייחסות לאופרנד זה. אך אם שיטות המיעון של שני האופרנדים דורשות מילות-מידע נוספות, אזי מילות-המידע הנוספות הראשונות מתייחסות לאופרנד המקור (source operand) ומילות-המידע הנוספות האחרונות מתייחסות לאופרנד היעד (destination operand).
גם למילות-המידע הנוספות יהיו זוג סיביות בצד ימין, המציינות את השדה A,R,E

סיביות 6-9 מהוות את קוד ההוראה כפי שהוסבר קודם.

סיביות 10-11 (group)

סיביות אלו מסמלות את סוג ההוראה המקודדת. בשפה קיימות הוראות ללא אופרנדים, הוראות עם אופרנד יחיד, והוראות עם 2 אופרנדים.
ערכי סיביות אלה יהיו 00 עבור קידוד הוראה ללא אופרנדים. 01 עבור קידוד הוראה עם אופרנד יחיד, ו- 10 עבור קידוד הוראה עם 2 אופרנדים.

סיביות 12-13 (rnd)

סיביות אלה בשימוש רק אם מופעלת שיטת מיעון מספר 2. אחרת ערכם 00.
סיביות אלה מכילות את כמות הכוכביות שנרשמו במסגרת שיטת המיעון מספר 2.
בשיטת מיעון זו ניתן לרשום 1,2 או 3 כוכביות, על כן, סיביות אלה יכילו את הערכים 01,10,11.
הסבר על שיטת מעיון זו ניתן למצוא בעמוד ההסבר על שיטות מיעון (עמוד הבא)

סיבית 14 – לא בשימוש

ארבע שיטות המיעון הקיימות במכונה שלנו הן :

ערך	שיטת מיעון	**תוכן המילה נוספת**	אופן הכתיבה	דוגמא
0	מיעון מידי	המילה הנוספת מכילה את האופרנד עצמו, כאשר הוא מיוצג ב- 13 סיביות אליהם מתווספות זוג סיביות של שדה A,R,E	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, מיוצג ב- 13 סיביות אליהן מתווספות זוג סיביות של שדה A,R,E	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 הנחית 'extern.'	mov x, r2

2	מיעון אקראי	1) אם נרשמה כוכבית אחת, סיביות rnd יכילו 01. האופרנד יהיה אחד מהאוגרים שייבחר אקראית ע"י האסמבלר. מספרו של האוגר הנבחר יקודד כמילת זיכרון נוספת בדומה לקידוד האוגרים בשיטת מיעון מספר 3.
		2) אם נרשמו 2 כוכביות, סיביות rnd יכלו 10. האופרנד יתפקד כמספר מידי שייבחר אקראית ע"י האסמבלר, המספר האקראי יקודד במילת זיכרון נוספת.
		3) אם נרשמו 3 כוכביות, סיביות rnd יכילו 11. האופרנד יתפקד בתור אחת התוויות הלא חיצוניות שיש בתוכנית. התוית תיבחר אקראית ע"י האסמבלר
			האופרנד מורכב מכוכבית אחת או 2 כוכביות או 3 כוכביות	mov *,r2
				או
				mov **,r2
				או
				mov ***,r2

3	מיעון אוגר ישיר	אם האוגר משמש כאופרנד יעד, הוא יקודד במילה נוספת שתכיל ב-6 הסיביות 7-2 את מספרו של האוגר. אם האוגר הוא אופרנד מקור, הוא יקודד במילה נוספת שתכיל ב-6 הסיביות 13-8 את	האופרנד הינו שם חוקי של אוגר.	mov r1,r2

		מספרו של האוגר. אם 2 האופרנדים הם אוגרים הם יחלקו מילה נוספת משותפת. כאשר 6 הביטים 2-7 הם עבור אוגר היעד, ו-6 הביטים 8-13 הם עבור אוגר המקור. לייצוג זה מתווספות זוג סיביות של שדה A,R,E. סיבית 14 תכיל 0.		

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

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, lea

פקודה	הסבר פעולה	דוגמא	**הסבר דוגמא**
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.
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.	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.
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.
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.
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 זוג הסיביות (4-5) חסרות משמעות מכיוון שאין אופרנד מקור (אופרנד ראשון) אלא רק אופרנד יעד (שני). על קבוצה זו נמנות ההוראות הבאות:
not, clr, inc, dec, jmp, bne, red, prn, jsr

פקודה	הסבר פעולה	דוגמא	**הסבר דוגמא**
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
clr	אפס את תוכן האופרנד.	clr r2	r2 ← 0
inc	הגדלת תוכן האופרנד באחד.	inc r2	r2 ← r2 + 1
dec	הקטנת תוכן האופרנד באחד.	dec C	C ← C – 1

jmp	קפיצה בלתי מותנית אל המען המיוצג על ידי האופרנד.	jmp LINE	PC ← LINE
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 ← LINE
red	קריאה של תו מתוך לוח המקשים אל האופרנד.	red r1	קוד ה-ascii של התו, הנקרא מלוח המקשים, יוכנס לאוגר r1.
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.
jsr	קריאה לשגרה (סברוטינה). הכנסת מצביע התוכנית (PC) לתוך המחסנית של זמן ריצה והעברת ערך האופרנד למצביע התוכנית (PC).	jsr FUNC	stack[SP] ← PC SP ← SP –1 PC ← FUNC

קבוצת הפקודות השלישית:
מכילה את ההוראות ללא אופרנדים – כלומר ההוראות המורכבות ממילה אחת בלבד.

ההוראות השייכות לקבוצה זו הן : rts, stop.

פקודה	הסבר פעולה	דוגמא	**הסבר דוגמא**
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. במילה זו החלק המשמעותי הן הסיביות 9-6 המהוות את קוד ההוראה. לשאר הסיביות אין כל חשיבות.	rts	SP ← SP +1 PC ← stack[SP]
stop	הוראה לעצירת התוכנית. ההוראה מורכבת ממלה אחת בלבד. במילה זו החלק המשמעותי הן הסיביות 9-6 המהוות את קוד ההוראה. לשאר הסיביות אין כל חשיבות.	stop	עצירת התוכנית.

<u>מספר נקודות נוספות לגבי תיאור שפת האסמבלי :</u>

שפת האסמבלי מורכבת ממשפטים (statements) כאשר התו, המפריד בין משפט למשפט, הינו תו '\n' (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.

<u>משפט הנחיה:</u>

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:

.data +7 , -57 , 17 , 9

יש לשים 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.
יש לשים לב: למשפט הנחיה לא מצורפות זוג סיביות A,R,E והקידוד ממלא את כל 14 הסיביות שיש בתא זיכרון.

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:

ABC:	.string	"abcdef"

מקצה "מערך תווי" באורך של 7 מילים החל מהמען המזוהה עם התווית ABC, ומאתחלת "מערך" זה לערכי ה- ascii של התווים a,b , c, d, e, f בהתאמה ולאחריהם ערך 0 לסימון סוף מחרוזת תווית.

3. '.entry'

להוראת '.entry' פרמטר אחד והוא שם של תווית המוגדרת בקובץ (מקבלת את ערכה שם). מטרת entry. היא להצהיר על התווית הזו כעל תווית אשר קטעי אסמבלי, הנמצאים בקבצים אחרים, מתייחסים אליה.

לדוגמא:

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

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

הערה: תווית בתחילת שורת entry. חסרת משמעות.

4. '.extern'

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

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

.extern	HELLO

<u>שורת הוראה:</u>

שורת הוראה מורכבת מ:
1. תווית אופציונלית.
2. שם ההוראה עצמה.
3. 0, 1 או 2 אופרנדים בהתאם להוראה.

אורכה 80 תווים 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). כקודם, **<u>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.</u>**

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:

הוראה :תווית אופציונלית

לדוגמא:

END: stop

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

<u>תווית:</u>

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.

<u>מספר:</u>

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 במרכאות כפולות (המרכאות 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
          add    r2,STR
LOOP:     jmp    END
          prn    #-5
          sub    r1, r4
          inc    K
          mov    **,r3
          bne    LOOP
END:      stop
STR:      .string "abcdef"
LENGTH:   .data  6,-9,15
K:        .data  2
```

עליו להחליף את שמות הפעולה mov, add, jmp, prn, sub, inc, bne, stop בקוד הבינארי השקול להם במחשב שהגדרנו.

כמו כן, על האסמבלר להחליף את הסמלים K,STR, LENGTH, MAIN, LOOP, END במענים של האתרים שהוקצו לנתונים, ובמענים של ההוראות המתאימות.

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

לתשומת לב: המקפים המופיעים בקידוד הבינרי הם רק לצורך הפרדה של השדות השונים בקידוד ונועדו לשם המחשה בלבד.

Label	Decimal Address	Base 32 Address	Command	Operands	Binary machine code
MAIN:	**0100**	034	**mov**	***, LENGTH	**0-11-10-0000-10-01-00**
	0101	035		כתובת של K(נבחר אקראית)	0000010000010-10
	0102	036		כתובת של LENGTH	**0000001111111-10**
	0103	037	**add**	r2, STR	**0-00-10-0010-11-01-00**
	0104	038		קידוד מספר האוגר	**0-000010-000000-00**
	0105	039		כתובת של STR	**0000001111000-10**
LOOP:	**0106**	03A	**jmp**	END	**0-00-01-1001-00-01-00**
	0107	03B		כתובת של END	**0000001110111-10**
	0108	03C	**prn**	#-5	**0-00-01-1100-00-00-00**
	0109	03D		המספר 5-	**1111111111011-00**
	0110	03E	**sub**	r1,r4	**0-00-10-0011-11-11-00**
	0111	03F		קידודי מספרי האוגרים	**0-000001-000100-00**
	0112	03G	**inc**	K	**0-00-01-0111-00-11-00**
	0113	03H		כתובת של K	**0000010000010-10**
	0114	04I	**mov**	**,r3	**0-10-10-0000-10-11-00**

	0115 **0116**	03J 03K		המספר האקראי 6 קידוד מספר האוגר	**0000000000110-00** **0-000000-000011-00**
	0117 **0118**	03L 03M	**bne**	LOOP כתובת של LOOP	**0-00-01-1010-00-01-00** **0000001101010-10**
END:	**0119**	03N	**stop**		**0-00-00-1111-00-00-00**
STR:	**0120**	03O	**.string**	"abcdef"	**000000001100001**
	0121	03P			**000000001100010**
	0122	03Q			**000000001100011**
	0123	03R			**000000001100100**
	0124	03S			**000000001100101**
	0125	03T			**000000001100110**
	0126	03U			**000000000000000**
LENGTH:	**0127** **0128** **0129**	03V 040 041	**.data**	6,-9,15	**000000000000110** **111111111110111** **000000000001111**
K:	**0130**	042	**.data**	2	**000000000000010**

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 0106 (עשרוני), אלא רק לאחר שהתוכנית נקראה כולה.

לכן יש שתי פעולות נפרדות, שצריך לבצע לגבי כל הסמלים, שהוגדרו ביד המתכנת. הראשונה היא לבנות טבלה של כל הסמלים והערכים המספריים המשויכים להם, והשניה 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 32**
MAIN	100	34
LOOP	106	3A
END	119	3N
STR	120	3O
LENGTH	127	3V
K	130	42

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

עליך לשים 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 ישויך 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 (IC) . ערכו ההתחלתי הוא 0, ולכן נטען משפט ההוראה הראשון במען 0. 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, 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 אינה כה פשוטה. יש הרבה הוראות המשתמשות בצורות מיעון שונות. אותה הוראה יכולה לקבל משמעויות שונות, בכל אחת מצורות המיעון, ולכן יתאימו לה קודים שונים. לדוגמא, הוראת ההזזה 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.

במחשב שלנו קיימת גמישות לגבי שיטת המיעון של שני האופרנדים. הערה: דבר זה לא מחייב לגבי כל מחשב. ישנם מחשבים בהם כל 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 מכן). הערה: 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.

<u>בתוכנית האסמבלר, שעליך לממש, יש להפריד בין קטע הנתונים לקטע ההוראות, כלומר בקבצי הפלט תהיה הפרדה של קוד ונתונים, ואילו בקובץ הקלט שניתן לתוכנית אין חובה שתהיה הפרדה.</u>

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 בדיוק. 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	שיטות מיעון חוקיות עבור אופרנד מקור	שיטות מיעון חוקיות עבור אופרנד יעד
mov	0,1,2,3,	1,2,3 (2 עם 3 *)
cmp	0,1,2,3,	0,1,2,3,
add	0,1,2,3,	1,2,3, , (2 עם 3 *)
sub	0,1,2,3,	1,2,3, , (2 עם 3 *)
not	אין אופרנד מקור	1,3, , (2 עם 3 *)
clr	אין אופרנד מקור	1, 3, (2 עם 3 *)
lea	1 או 2 עם 3 כוכביות	1,3, , (2 עם 3 *)
inc	אין אופרנד מקור	1,3, , (2 עם 3 *)
dec	אין אופרנד מקור	1,3, , (2 עם 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 עם 3 *)
rts	אין אופרנד מקור	אין אופרנד יעד
stop	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 <= 0, IC <= 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 <= L + IC
14. חזור ל-2.

מעבר שני

1. IC <= 0
2. קרא שורה. אם סיימת, עבור ל-11.
3. אם השדה הראשון הוא סמל, דלג עליו.
4. האם זוהי הוראה מדומה (data. , string.)? 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 <= 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
          add    r2,STR
LOOP:     jmp    END
          prn    #-5
          sub    r1, r4
          inc    K
          mov    **,r3
          bne    LOOP
END:      stop
STR:      .string "abcdef"
LENGTH:   .data  6,-9,15
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).

Label	Decimal Address	Base 32 Address	Command	Operands	Binary machine code
MAIN:	**0100** **0101** **0102**	034 035 036	**mov**	***, LENGTH כתובת של K(נבחר אקראית) כתובת של LENGTH	**0-11-10-0000-10-01-00** **?** **?**
	0103 **0104** **0105**	037 038 039	**add**	r2, STR קידוד מספר האוגר כתובת של STR	**0-00-10-0010-11-01-00** **0-000010-000000-00** **?**
LOOP:	**0106** **0107**	03A 03B	**jmp**	END כתובת של END	**0-00-01-1001-00-01-00** **?**
	0108 **0109**	03C 03D	**prn**	#-5 המספר 5-	**0-00-01-1100-00-00-00** **1111111111011-00**
	0110 **0111**	03E 03F	**sub**	r1,r4 קידודי מספרי האוגרים	**0-00-10-0011-11-11-00** **0-000001-000100-00**
	0112 **0113**	03G 03H	**inc**	K כתובת של K	**0-00-01-0111-00-11-00** **?**
	0114 **0115** **0116**	04I 03J 03K	**mov**	**,r3 המספר האקראי 6 קידוד מספר האוגר	**0-10-10-0000-10-11-00** **0000000000110-00** **0-000000-000011-00**
	0117 **0118**	03L 03M	**bne**	LOOP כתובת של LOOP	**0-00-01-1010-00-01-00** **?**
END:	**0119**	03N	**stop**		**0-00-00-1111-00-00-00**
STR:	**0120**	03O	**.string**	“abcdef”	**000000001100001**
	0121	03P			**000000001100010**
	0122	03Q			**000000001100011**
	0123	03R			**000000001100100**
	0124	03S			**000000001100101**
	0125	03T			**000000001100110**
	0126	03U			**000000000000000**
LENGTH:	**0127** **0128** **0129**	03V 040 041	**.data**	6,-9,15	**000000000000110** **111111111110111** **000000000001111**
K:	**0130**	042	**.data**	2	**000000000000010**

טבלת הסמלים :

סמל	ערך דצימלי	ערך בבסיס 32
MAIN	100	34
LOOP	106	3A
END	119	3N
STR	120	3O
LENGTH	127	3V
K	130	42

נבצע עתה את המעבר השני ונרשום את הקוד בצורתו הסופית:

Label	Decimal Address	Base 32 Address	Command	Operands	Binary machine code
MAIN:	**0100**	034	**mov**	***, LENGTH	**0-11-10-0000-10-01-00**
	0101	035		כתובת של K(נבחר אקראית)	0000010000010-10
	0102	036		כתובת של LENGTH	**0000001111111-10**
	0103	037	**add**	r2, STR	**0-00-10-0010-11-01-00**
	0104	038		קידוד מספר האוגר	**0-000010-000000-00**
	0105	039		כתובת של STR	**0000001111000-10**
LOOP:	**0106**	03A	**jmp**	END	**0-00-01-1001-00-01-00**
	0107	03B		כתובת של END	**0000001110111-10**
	0108	03C	**prn**	#-5	**0-00-01-1100-00-00-00**
	0109	03D		המספר 5-	**1111111111011-00**
	0110	03E	**sub**	r1,r4	**0-00-10-0011-11-11-00**
	0111	03F		קידודי מספרי האוגרים	**0-000001-000100-00**
	0112	03G	**inc**	K	**0-00-01-0111-00-11-00**
	0113	03H		כתובת של K	**0000010000010-10**
	0114	04I	**mov**	**,r3	**0-10-10-0000-10-11-00**
	0115	03J		המספר האקראי 6	**0000000000110-00**
	0116	03K		קידוד מספר האוגר	**0-000000-000011-00**
	0117	03L	**bne**	LOOP	**0-00-01-1010-00-01-00**
	0118	03M		כתובת של LOOP	**0000001101010-10**
END:	**0119**	03N	**stop**		**0-00-00-1111-00-00-00**
STR:	**0120**	03O	**.string**	"abcdef"	**000000001100001**
	0121	03P			**000000001100010**
	0122	03Q			**000000001100011**
	0123	03R			**000000001100100**
	0124	03S			**000000001100101**
	0125	03T			**000000001100110**
	0126	03U			**000000000000000**
LENGTH:	**0127**	03V	**.data**	6,-9,15	**000000000000110**
	0128	040			**111111111110111**
	0129	041			**000000000001111**
K:	**0130**	042	**.data**	2	**000000000000010**

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 (כאמור, אינה למימוש בפרויקט זה)

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 האסמבלי, 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.

<u>מבנה כל קובץ יתואר בהמשך</u>.

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 – 12 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.

קביעת שיטת המיעון נעשית בהתאם לתחביר של האופרנד, כפי שהוא מתואר בחלק העוסק בשיטות המיעון. למשל, התו # מציין מיעון מיידי, תווית מציינת מיעון ישיר, שם של אוגר מציין מיעון אוגר.

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

לאחר שהאסמבלר החליט לגבי הנ"ל (פעולה, שיטת מיעון אופרנד מקור, שיטת מיעון אופרנד יעד, אוגר אופרנד מקור, 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 הוראות המכונה כפי שתוארו קודם לכן) את קוד הפעולה, שיטות המיעון, ואת המידע על האוגרים. בנוסף הוא "משרייין" מקום עבור מספר המילים הנוספות, הנדרשות עבור פקודה זו ומגדיל את מונה הקוד בהתאם.

אם ההוראה היא בעלת אופרנד אחד בלבד, כלומר האופרנד הראשון (אופרנד המקור) אינו מופיע, אזי התרגום הינו זהה לחלוטין, למעט העובדה שסיביות מיעון המקור 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, stop) 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 לסוג שלה, באופן הבא:

I. '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 לפניה, 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, 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 הנ"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 האסמבלי בקובץ עכשווי עושה בו שימוש.
האסמבלר מכניס את המשתנה המבוקש אל טבלת הסמלים. ערכו הוא אפס (או כל ערך אחר), טיפוסו הוא external, היכן ניתנה 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**

<u>האסמבלר בונה את תמונת זיכרון המכונה כך שקידוד ההוראה הראשונה מקובץ האסמבלי יכנס למען 100(בבסיס 10)</u> 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 האסמבלי, וכך הלאה.
התייחסות בקובץ האסמבלי למשתנה, שהוגדר בקובץ, תקודד כך שתצביע על המקום המתאים, בתמונת הזיכרון שבונה האסמבלר. עקרונית פורמט של קובץ object הינו תמונת הזיכרון הנ"ל.

קובץ object מורכב משורות שורות של טקסט, השורה הראשונה מכילה,(בבסיס 32) את אורך הקוד (במילות זיכרון) ואת אורך הנתונים (במילות זיכרון). שני המספרים מופרדים ביניהם על יד רווח. השורות הבאות מתארות את תוכן הזיכרון (שוב, בבסיס 32)

בהמשך מופיע קובץ object לדוגמא ששמו : ps.obj המתאים ל-ps.as

בנוסף עבור כל תא זיכרון <u>המכיל הוראה</u> <u>(לא data)</u> מופיע מידע עבור תכנית הקישור. מידע זה ניתן ע"י 2 הסיביות הימניות של הקידוד (שדה ה- E,R,A)

האות 'A' מציינת את העובדה שתוכן התא הינו אבסלוטי (absolute) ואינו תלוי היכן באמת יטען הקובץ (האסמבלר יוצא מתוך הנחה שהקובץ נטען החל ממען אפס). במקרה כזה 2 הסיביות יכילו את הערך 00

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. במקרה כזה 2 הסיביות יכילו את הערך 10

האות 'E' מציינת שתוכן תא הזיכרון תלוי במשתנה חיצוני external וכי תכנית הקישור תדאג להכנסת הערך המתאים. במקרה כזה 2 הסיביות יכילו את הערך 01

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 (לדוגמא הקובץ ps.ext מתאים לקובץ האסמבלי ששמו ps.as).

להלן קובץ PS.AS לדוגמא :

```
; file ps.as

.entry LOOP
.entry LENGTH
.extern L3
.extern W
MAIN:     mov     ***, W
          add     r2,STR
LOOP:     jmp     W
          prn     #-5
          sub     r1, r4
          inc     K
          mov     **,r3
          bne     L3
END:      stop
STR:      .string "abcdef"
LENGTH:   .data   6,-9,15
K:        .data   2
```

הקובץ שלהלן הוא קובץ object ששמו בעל סיומת '.ob'. זהו קובץ שהתקבל מהפעלת התוכנית assembler על קובץ האסמבלר דלעיל. להלן דוגמת הקידוד לביטים ולאחריה פורמט קובץ ה-OB . **כל תוכן הקובץ מיוצג על ידי מספרים בבסיס 32**.

קובץ ps.ob :

Label	Decimal Address	Base 32 Address	Command	Operands	Binary machine code
MAIN:	**0100**	034	**mov**	***, W	**0-11-10-0000-10-01-00**
	0101	035		כתובת של K(נבחר אקראית)	0000010000010-10
	0102	036		כתובת של W	**0000000000000-01**
	0103	037	**add**	r2, STR	**0-00-10-0010-11-01-00**
	0104	038		קידוד מספר האוגר	**0-000010-000000-00**
	0105	039		כתובת של STR	**0000001111000-10**
LOOP:	**0106**	03A	**jmp**	W	**0-00-01-1001-00-01-00**
	0107	03B		כתובת של W	**0000000000000-01**
	0108	03C	**prn**	#-5	**0-00-01-1100-00-00-00**
	0109	03D		המספר 5-	**1111111111011-00**
	0110	03E	**sub**	r1,r4	**0-00-10-0011-11-11-00**
	0111	03F		קידודי מספרי האוגרים	**0-000001-000100-00**
	0112	03G	**inc**	K	**0-00-01-0111-00-11-00**
	0113	03H		כתובת של K	**0000010000010-10**
	0114	04I	**mov**	**,r3	**0-10-10-0000-10-11-00**
	0115	03J		המספר האקראי 6	**0000000000110-00**
	0116	03K		קידוד מספר האוגר	**0-000000-000011-00**
	0117	03L	**bne**	L3	**0-00-01-1010-00-01-00**
	0118	03M		כתובת של L3	**0000000000000-01**
END:	**0119**	03N	**stop**		**0-00-00-1111-00-00-00**

STR:	**0120**	03O	**.string**	"abcdef"	**000000001100001**
	0121	03P			**000000001100010**
	0122	03Q			**000000001100011**
	0123	03R			**000000001100100**
	0124	03S			**000000001100101**
	0125	03T			**000000001100110**
	0126	03U			**000000000000000**
LENGTH:	**0127** **0128** **0129**	03V 040 041	**.data**	6,-9,15	**000000000000110** **111111111110111** **000000000001111**
K:	**0130**	042	**.data**	2	**000000000000010**

כלומר תוכן קובץ ps.ob הוא:

Base 32 Address	**Base 32 machine code**
	K B
034	**E14**
035	**0GA**
036	**001**
037	**25K**
038	**0G0**
039	**0F2**
03A	**1I4**
03B	**001**
03C	**1O0**
03D	**VVC**
03E	**27S**
03F	**08G**
03G	**1EC**
03H	**0GA**
03I	**A1C**
03J	**00O**
03K	**00C**
03L	**1K4**
03M	**001**
03N	**0U0**
03O	**031**
03P	**032**
03Q	**033**
03R	**034**
03S	**035**
03T	**036**
03U	**000**
03V	**006**
040	**VVN**
041	**00F**
042	**002**

ps.ent: קובץ

```
LOOP      3A
LENGTH    3V
```

ps.ext: קובץ

```
W     36
W     3B
L3    3M
```

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

סיכום והנחיות כלליות

- אורך התוכנית, הניתנת כקלט לאסמבלר, אינו ידוע מראש (ואינו קשור לגודל 1000 – 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, לשם מטרה זו. במבנים אחרים בפרויקט, יש להשתמש על פי יעילות/תכונות נדרשות.

- 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 תחבירית, אין להפיק את קבצי הפלט (ob, ext, ent) אלא רק לדווח על השגיאות שנמצאו .

תם ונשלם חלק ההסברים והגדרת הפרויקט.

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 והממ"נים, והתשובות יכולות לעזור לכולם.

לתשומת לבכם, לא יתקבלו ממ"נים באיחור ללא סיבה מוצדקת, באישור מרכזת הקורס.

בהצלחה!!!!