ה א ו נ י ב ר ס י ט ה ה פ ת ו ח ה

20465
מעבדה בתכנות מערכות
חוברת הקורס – אביב 2018ב

כתבה: מיכל אבימור

מרץ 2018 – סמסטר אביב – תשע"ח

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.

לתשומת לב הסטודנטים הלומדים בחו"ל:

למרות הריחוק הפיסי הגדול, נשתדל לשמור אתכם על קשרים הדוקים ולעמוד לרשותכם ככל האפשר.
הפרטים החיוניים על הקורס נכללים בחוברת הקורס וכן באתר הקורס.
מומלץ מאד להשתמש באתר הקורס ובכל אמצעי העזר שבו וכמובן לפנות אלינו במידת הצורך.

אני מאחלת לך לימוד פורה ומהנה.

בברכה,

מיכל אבימור
מרכזת ההוראה בקורס.

1. לוח זמנים ופעילויות (20465 / ב2018)

שבוע לימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח ממ"ן (למנחה)
1	16.3.2018-6.3.2018	ספר C פרקים 1-2-3	מפגש ראשון	
2	23.3.2018-18.3.2018	ספר C פרקים 1-2-3		
3	30.3.2018-25.3.2018 (ו ערב פסח)	ספר C פרק 4	מפגש שני	
4	6.4.2018-1.4.2018 (א-ו פסח)	ספר C פרק 4		
5	13.4.2018-8.4.2018 (ה יום הזכרון לשואה)	ספר C פרק 5	מפגש שלישי	ממ"ן 11 8.4.2018
6	20.4.2018-15.4.2018 (ד יום הזיכרון) (ה יום העצמאות)	ספר C פרק 5		
7	27.4.2018-22.4.2018	ספר C פרק 6	מפגש רביעי	
8	4.5.2018-29.4.2018 (ה ל"ג בעומר)	ספר C פרק 6		ממ"ן 12 29.4.2018
9	11.5.2018-6.5.2018	ספר C פרק 6		

** 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)
10	18.5.2018-13.5.2018 (א יום ירושלים)	ספר C פרק 7	מפגש חמישי	
11	25.5.2018-20.5.2018 (א שבועות)	ספר C פרק 7		ממ"ן 22 20.5.2018
12	1.6.2018-27.5.2018	ספר C פרק 8 + פרויקט	מפגש שישי	
13	8.6.2018-3.6.2018	פרויקט וחזרה		
14	15.6.2018-10.6.2018	פרויקט וחזרה	מפגש שביעי	ממ"ן 23 10.6.2018
15	19.6.2018-17.6.2018	פרויקט וחזרה		ממ"ן 14** 19.8.2018

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
12	5 (ממ"ן חובה)	5,4
22	8 (ממ"ן רשות)	6,5,4
23	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, 12) 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: 2018ב' **מועד אחרון להגשה:** 8.4.2018

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 . יש להגיש את קבצי המקור (.h, .c), קבצי ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי makefile), וכן קבצי קלט ותדפיסי מסך או קבצי פלט (לפי הנחיות במטלה/במפגש/באתר). כל תכנית תהיה בתיקיה נפרדת. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת c.

יש להגיש תכניות מלאות (בין השאר מכילות main), הניתנות להידור והרצה, ומאפשרות בדיקה של כל תוצאות הריצה המגוונות, ללא צורך בשינויים כלשהם בקוד התכנית.

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

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

עליכם לכתוב פונקציה **int suffix(char str[], char ch)** , אשר מדפיסה את כל הסייפות (סיומות) של המחרוזת str המתחילות בתו ch. כל סייפא תודפס בשורה נפרדת, לפי סדר הסייפות במחרוזת משמאל לימין. כמו כן, על הפונקציה להחזיר את מספר הסייפות שמצאה.

<u>לדוגמה</u>: הקריאה suffix("abrahmrnsbre",'b')

תדפיס:

brahmrnsbre

bre

ותחזיר את הערך 2.

בנוסף, עליכם לכתוב תכנית ראשית (הפונקציה main), שתבצע קלט של תו בודד ולאחריו מחרוזת, תפעיל את הפונקציה suffix על נתוני הקלט, ותדפיס באופן נאה את הערך המוחזר מהפונקציה.

מותר להניח שהאורך המקסימלי של מחרוזת הקלט הוא 80 תווים. כמו כן, הניחו שהקלט תקין.

הקלט לתכנית הוא מ-stdin, ויכול להגיע **מהמקלדת או מקובץ** (באמצעות redirection בעת הפעלת התכנית). לנוחיותכם, הכינו מספר קבצי קלט והשתמשו בהם שוב ושוב לדיבוג התכנית.

על התכנית **להדפיס הודעת בקשה ידידותית לקלט**. כמו כן יש **להדפיס באופן נאה את הנתונים שנקלטו**, וזאת לפני הקריאה לפונקציה. באופן זה, הקלט יוצג גם כאשר הוא מגיע מקובץ.

חובה לצרף להגשה מספר הרצות בדיקה, המדגימות את פעולת התכנית. **יש להגיש תדפיסי מסך (או קבצי פלט) של כל ההרצות**. במידה ותשתמשו בקבצי קלט, **יש להגיש גם קבצים אלה.**

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

נגדיר **מחרוזת מקוצרת** באופן הבא:
בהינתן מחרוזת תווים, נחפש בה רצפים עולים של תווים מתוך אותיות האלף-בית האנגלי, שאורכם לפחות 3. לדוגמה, במחרוזת הבאה ישנם שלושה רצפים כאלה, המודגשים בקווים:

dabcefLMNOpQrstuv567zyx

כל רצף יקוצר לשלשה תווים: האות הראשונה ברצף, מקף, והאות האחרונה ברצף. עבור הדוגמה למעלה נקבל את המחרוזת המקוצרת:

da-cefL-OpQr-v567zyx

לתשומת לב: מדובר ברצפים של אותיות האלף-בית בלבד (ולא כל תווים אחרים). אותיות קטנות וגדולות נחשבות שונות זו מזו. הרצף נקבע לפי קודי האסקי המתאימים.

עליכם לכתוב פונקציה המקבלת כפרמטר מחרוזת, והופכת אותה למחרוזת מקוצרת.
לתשומת לב: הפונקציה אינה בונה מחרוזת חדשה, אלא משנה את המחרוזת המועברת אליה.

בנוסף, עליכם לכתוב תכנית ראשית (הפונקציה main), שתבצע קלט של מחרוזת מהמשתמש, תפעיל את הפונקציה על מחרוזת הקלט, ותדפיס באופן נאה את המחרוזת המקוצרת.

מותר להניח שהאורך המקסימלי של מחרוזת הקלט הוא 80 תווים.

הקלט לתכנית הוא מ-stdin, ויכול להגיע **מהמקלדת או מקובץ** (באמצעות redirection בעת הפעלת התכנית). לנוחיותכם, הכינו מספר קבצי קלט, והשתמשו בהם שוב ושוב, בכדי לדבג את התכנית.

על התכנית **להדפיס הודעת בקשה** ידידותית לקלט. כמו כן יש **להדפיס באופן נאה את המחרוזת שנקלטה**, וזאת לפני הקריאה לפונקציה. באופן זה, הקלט יוצג גם כאשר הוא מגיע מקובץ.

חובה לצרף להגשה מספר הרצות בדיקה, המדגימות את פעולת התכנית. **יש להגיש תדפיסי מסך (או קבצי פלט) של כל ההרצות**. במידה ותשתמשו בקבצי קלט, **יש להגיש גם קבצים אלה**.

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) 12

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 4,5 ובאופן חלקי 6

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

סמסטר: 2018ב' **מועד אחרון להגשה:** 29.4.2018

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 . יש להגיש את קבצי המקור (.h, .c), קבצי ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי makefile), וכן קבצי קלט ותדפיסי מסך או קבצי פלט (לפי הנחיות במטלה/במפגש/באתר). כל תכנית תהיה בתיקיה נפרדת. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת c.

יש להגיש תכניות מלאות (בין השאר מכילות main), הניתנות להידור והרצה, ומאפשרות בדיקה של כל תוצאות הריצה המגוונות ללא צורך בשינויים כלשהם בקוד התכנית.

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

שאלה 1 (תכנית ראשית בקובץ my_bcmp.c)

עליכם **לממש** את הפונקציה הבאה:

```
int my_bcmp (const void *b1, const void *b2, size_t len);
```

b1 – מצביע לקטע זיכרון ראשון.
b2 – מצביע לקטע זיכרון שני.
len – מספר הבתים (bytes) להשוואה.

<u>מטרת הפונקציה</u>: להשוות את len הבתים, החל מהמקום אליו מצביע b1 , אל len הבתים, החל מהמקום אליו מצביע b2.

<u>הערך המוחזר</u>: 0 אם נמצא ששני קטעי הזיכרון מכילים תוכן זהה, ואחרת ערך כלשהו שונה מ-0.

<u>הערות</u>:

- קטעי הזכרון יכולים להיות חופפים זה לזה באופן חלקי או מלא
- קטעי זיכרון באורך 0 הם תמיד זהים זה לזה
- <u>לא ניתן</u> להניח שקטעי הזיכרון מסתיימת בתו '\0', ואסור לפונקציה לשנות את הקטעים.

לצורך בדיקת הפונקציה, נשתמש בקטעי זיכרון המוזנים לתכנית על ידי המשתמש. התכנית תקבל כקלט מחרוזת, שני אינדקסים לתוך המחרוזת, ואת הערך len. האינדקסים מייצגים מצביעים לקטעי הזיכרון (כמובן שיש לבנות מהאינדקסים מצביעים לצורך הקריאה לפונקציה).

<u>סדר הנתונים בקלט</u> הוא: len, אחריו שני האינדקסים, ולבסוף המחרוזת.
מותר להניח שהאורך המקסימלי של מחרוזת הקלט הוא 512.
<u>לתשומת לב</u>: len <u>אינו</u> אורך מחרוזת הקלט, אלא מספר הבתים להשוואה.

יש לכתוב תכנית ראשית (הפונקציה main) שתבצע קלט מהמשתמש, תפעיל את הפונקציה my_bcmp, ותדפיס באופן נאה את התוצאה המוחזרת מן הפונקציה.

<u>חובה</u> לבצע בדיקות תקינות של הקלט <u>טרם</u> הקריאה לפונקציה. במקרה של שגיאה כלשהי בנתוני הקלט, על התכנית להפסיק את עבודתה, ולהדפיס הודעת שגיאה מפורטת, לרבות מידע על הנתון השגוי ככל הניתן. להלן דוגמאות של שגיאות אפשריות (אפשר להוסיף עוד שגיאות):

- חסרים נתונים בקלט (כמוגדר לעיל, נדרשים בדיוק 4 נתונים)
- אחד האינדקסים ו/או הערך len אינו מספר עשרוני שלם אי-שלילי
- אחד מקטעי הזיכרון גולש מגבולות המחרוזת (לפי ערכי האינדקס ו- len)

הקלט לתכנית הוא מ-stdin, ויכול להגיע **מהמקלדת או מקובץ** (באמצעות redirection בעת הפעלת התכנית). לנוחיותכם, הכינו מספר קבצי קלט והשתמשו בהם שוב ושוב לדיבוג התכנית.

על התכנית **להדפיס הודעת בקשה ידידותית לקלט**. כמו כן יש **להדפיס באופן נאה את הנתונים שנקלטו**, וזאת <u>לפני</u> הקריאה לפונקציה. באופן זה, הקלט יוצג גם כאשר הוא מגיע מקובץ.

חובה לצרף להגשה מספר <u>הרצות בדיקה</u> המדגימות את פעולת התכנית על גדלים שונים של נתוני הקלט, לרבות הדגמה של הטיפול במגוון השגיאות בקלט. **יש להגיש תדפיסי מסך (או קבצי פלט) של כל הרצות הדוגמה**. במידה ותשתמשו בקבצי קלט, **יש להגיש גם קבצים אלה**.

<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) 22

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 4,5,6

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

סמסטר: 2018ב' **מועד אחרון להגשה:** 20.5.2018

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 . יש להגיש את קבצי המקור (.h, .c), קבצי ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי makefile), וכן קבצי קלט ותדפיסי מסך או קבצי פלט (לפי הנחיות במטלה/במפגש/באתר). כל תכנית תהיה בתיקיה נפרדת. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת c.

יש להגיש תכניות מלאות (בין השאר מכילות main), הניתנות להידור והרצה, ומאפשרות בדיקה של כל תוצאות הריצה המגוונות, ללא צורך בשינויים כלשהם בקוד התכנית.

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

שאלה 1 (תכנית ראשית בקובץ myset.c, ובנוסף הקבצים set.c, set.h)

עליכם לכתוב תכנית מחשב אינטראקטיבית הקוראת פקודות מהקלט הסטנדרטי, מפענחת ומבצעת אותן. הפקודות עוסקות בפעולות על קבוצות.

<u>תזכורת</u>: קבוצה (set) היא אוסף של איברים, ברי מניה, וללא חזרות (לכל איבר בקבוצה ערך שונה מכל האיברים האחרים).

עליכם להגדיר, תוך שימוש ב- typedef, את הטיפוס set, אשר מחזיק קבוצה של מספרים שלמים מהתחום הסגור [0...127]. על הטיפוס (מבנה הנתונים) set להיות <u>חסכוני מבחינת כמות הזיכרון</u> הנדרשת. כך למשל, מערך של 128 בתים <u>אינו</u> חסכוני.
<u>רמז</u>: אפשר להסתפק בסיבית (bit) אחת לכל איבר בקבוצה.

בנוסף עליכם להגדיר, בתכנית הראשית, שישה משתנים מטיפוס set, בשמות הבאים:
SETA, SETB, SETC, SETD, SETE, SETF.

בפקודות שיפורטו להלן, כל פרמטר שהוא שם של קבוצה, יהיה אחד מששת המשתנים האלה.
בתחילת הריצה של התכנית, יש לאתחל את כל המשתנים לקבוצה הריקה.

להלן מפרט הפקודות המשמשות כקלט לתכנית:

לתשומת לב: סדר קריאת השדות בפקודה הוא משמאל לימין.

1. read_set שם-קבוצה, רשימת-ערכים-מופרדים-זה-מזה-בפסיקים

הפקודה מכניסה את הערכים שברשימה לתוך הקבוצה ששמה ניתן בפקודה. רשימת הערכים אינה סדורה, ומותר לערך להופיע בה יותר מפעם אחת (מופעים חוזרים לא יילקחו בחשבון). סוף רשימת הערכים מסומן על ידי המספר השלילי 1-.

לדוגמה, הפקודה הבאה:

read_set SETA, 5, 6, 5, 76, 44, 23, 23, 98, 23, -1

תאחסן במשתנה SETA את הקבוצה הבאה:

5, 6, 23, 44, 76, 98

אם הרשימה אינה מכילה אף ערך (מלבד 1- המסיים), תיווצר קבוצה ריקה.

2. print_set שם-קבוצה

זוהי פקודת הדפסה. הקבוצה ששמה ניתן תודפס בצורה נאה ובסדר עולה של הערכים, לכל היותר 16 ערכים בכל שורת פלט. אם הקבוצה ריקה, יש להדפיס "The set is empty".

3. union_set שם-קבוצה-א', שם-קבוצה-ב', שם-קבוצה-ג'

הפקודה מבצעת איחוד של קבוצה א' וקבוצה ב', ואת התוצאה מאחסנת בקבוצה ג'.

4. intersect_set שם-קבוצה-א', שם-קבוצה-ב', שם-קבוצה-ג'

הפקודה מבצעת חיתוך של קבוצה א' וקבוצה ב' ואת התוצאה מאחסנת בקבוצה ג'.

5. sub_set שם-קבוצה-א', שם-קבוצה-ב', שם-קבוצה-ג'

הפקודה מבצעת חיסור של קבוצה ב' מקבוצה א', ואת התוצאה מאחסנת בקבוצה ג'.
תזכורת: תוצאת החיסור של קבוצה ב' מקבוצה א' הם כל האיברים הנמצאים בקבוצה א' ולא נמצאים בקבוצה ב'.

6. stop

זוהי פקודה ללא פרמטרים, הגורמת לסיום התכנית.

לתשומת לב: אותו שם קבוצה יכול לשמש ביותר מפרמטר אחד באותה הפקודה. מימוש הפעולות על קבוצות צריך להתחשב באפשרות זו (לא לדרוס נתונים תוך כדי חישוב).
לדוגמה, הפעולות שלהלן תקינות ומוגדרות היטב:

union_set SETC, SETD, SETD
intersect_set SETA, SETF, SETA
sub_set SETC, SETC, SETC
union_set SETA,SETA,SETF

המבנה התחבירי של הקלט:

- כל פקודה תופיע בשלמותה בשורת קלט יחידה, כולל כל הפרמטרים. מותרות גם שורות ריקות (שורות המכילות רק תווים לבנים).
- שם הפקודה מופרד מהפרמטר הראשון באמצעות רווחים ו/או טאבים (אחד או יותר).
- בין כל שני אופרנדים יש פסיק אחד. לפני ואחרי הפסיק יכולים להיות רווחים ו/או טאבים בכמות בלתי מוגבלת. אסור שיהיה פסיק אחרי הפרמטר האחרון.

- יכולים להיות רווחים ו/או טאבים בכמות בלתי מוגבלת לפני שם הפקודה, וגם בסוף השורה (אחרי הפרמטר האחרון).
- אסור שיהיו תווי זבל בסוף השורה (למעט תווים לבנים).
- שמות הפקודות יופיעו באותיות קטנות בלבד, ושמות הקבוצות באותיות גדולות בלבד.

<u>אופן פעולת התכנית:</u>

יש לממש ממשק משתמש ידידותי, כך שהמשתמש יוכל להבין בכל שלב של התכנית מה עליו לעשות. בפרט, על התכנית להודיע באמצעות סימן (prompt) בכל פעם שהיא מוכנה לקלוט את הפקודה הבאה. התכנית תמשיך לקלוט ולבצע פקודה אחרי פקודה, עד שתתקבל הפקודה stop.

התכנית <u>אינה</u> מניחה שהקלט תקין. על התכנית לנתח כל פקודה ולוודא שאין בה שגיאות (ראו דוגמאות בהמשך). במידה ונתגלתה שגיאה, התכנית תדפיס הודעת שגיאה פרטנית, ותעבור לפקודה הבאה, בלי לבצע את הפקודה השגויה. <u>אין לעצור</u> את התכנית עם גילוי השגיאה הראשונה. אין צורך לדווח על יותר משגיאה אחת בכל שורת קלט.

יש לטפל גם במצב של EOF (גמר הקלט). עצירת התכנית שלא באמצעות פקודת stop אינה נחשבת תקינה (גם לא כאשר הקלט מגיע מקובץ באמצעות redirection), ויש להדפיס הודעת שגיאה על כך ואז לעצור. <u>שימו לב</u>: השורה האחרונה בקובץ קלט אינה חייבת להסתיים בתו '\n'.

<u>להלן דוגמאות של קלט שגוי:</u>

שימו לב: ייתכנו סוגים נוספים של שגיאות בקלט.
עליכם לחשוב על <u>כל מגוון השגיאות</u> האפשריות, ולטפל בכולן.

1. לפקודה:

read_set SETG, 3, 6, 5, 4, 4, -1

יש להגיב בהודעה כגון:

Undefined set name

2. לפקודה:

read_set setA, 3, 6, 5, 4, 4, -1

יש להגיב בהודעה כגון:

Undefined set name

3. לפקודה:

do_it SETA, SETB, SETC

יש להגיב בהודעה כגון:

Undefined command name

4. לפקודה:

UNION_set SETA, SETB, SETC

יש להגיב בהודעה כגון:

Undefined command name

5. לפקודה:

read_set SETB, 45, 567, 34, -1

יש להגיב בהודעה כגון:

Invalid set member – value out of range

6. לפקודה:

read_set SETA, 45, 56, 45, 34

יש להגיב בהודעה כגון:

List of set members is not terminated correctly

7. לפקודה:

read_set SETA, 45 ,-3, 2, 45, 34, -1

יש להגיב בהודעה כגון:

Invalid set member – value out of range

8. לפקודה:

read_set SETA, 45, 2, xyz, 34, -1

יש להגיב בהודעה כגון:

Invalid set member – not an integer

9. לפקודה:

read_set SETA, 45, 2, 24.0, 34, -1

יש להגיב בהודעה כגון:

Invalid set member – not an integer

10. לפקודה:

union_set SETC, SETA

יש להגיב בהודעה כגון:

Missing parameter

11. לפקודה:

print_set SETC, SETD

יש להגיב בהודעה כגון:

Extraneous text after end of command

12. לפקודה:

sub_set SETF, , SETD, SETA

יש להגיב בהודעה כגון:

Multiple consecutive commas

13. לפקודה:

intersect_set SETF SETD SETA

יש להגיב בהודעה כגון:

Missing comma

14. לפקודה:

print_set, SETF

יש להגיב בהודעה כגון:

Illegal comma

<u>להלן דוגמה של סדרת פקודות שכולן תקינות</u>:

הערה: סידרה כגון זו יכולה לשמש כקלט לבדיקת נכונות הביצוע של הפקודות.

```
read_set SETA, 45, 23, 6, 7, 4, 3, 75 ,45, 34, -1
read_set SETB, 5, 4, 3, 2, 78, 45, 43, -1
read_set SETC,100,105,101,103,104,-1
read_set SETD,  -1
print_set SETA
print_set SETB
print_set SETC
print_set SETD
print_set SETE
read_set      SETC ,     110   ,   111,   112   ,   -1
print_set SETC
```

```
union_set SETA, SETB, SETD
print_set SETD
intersect_set SETA, SETB, SETE
print_set SETE
sub_set SETA, SETB, SETF
print_set SETF
intersect_set SETA, SETC, SETD
print_set SETD
union_set SETB, SETB, SETE
print_set SETE
intersect_set SETB, SETA, SETB
print_set SETB
union_set SETA, SETC, SETC
print_set SETC
sub_set SETC,SETC,SETC
print_set SETC
union_set   SETF  ,  SETC  ,   SETF
print_set SETF
stop
```

<u>דרישות נוספות</u>:

- יש לחלק את התוכנית למספר קבצי מקור: myset.c, set.c, ו- set.h.
  - בקובץ set.c יש לרכז את הפעולות על קבוצות. לכל פעולה יש לממש פונקציה נפרדת, עם פרמטרים לפי מפרט הפעולה כמוגדר לעיל.
  - בקובץ myset.c תהיה הפונקציה main, וכן כל פעילויות האינטראקציה עם המשתמש וניתוח הקלט (לרבות הדפסת הודעות השגיאה). כמו כן, יוגדרו בקובץ זה ששת המשתנים מטיפוס set.
  - בקובץ set.h תהיה הגדרת טיפוס הנתונים set, וכן ההצהרות (אב טיפוס) של הפונקציות הממומשות בקובץ set.c. יש להכליל (#include) את הקובץ set.h בקבצי המקור האחרים.
  - באפשרותכם לבנות קבצי מקור נוספים (למשל: קובץ המכיל פונקציות עזר לניתוח הקלט, וכד').
- הקלט לתכנית הוא מ-stdin, ויכול להגיע **מהמקלדת או מקובץ** (באמצעות redirection בעת הפעלת התכנית). לנוחיותכם, הכינו מספר קבצי קלט והשתמשו בהם שוב ושוב לדיבוג התכנית. בכל קובץ קלט תהיה סדרה של פקודות על קבוצות.
- לפני הניתוח של כל שורת קלט, **על התכנית להדפיס את השורה לפלט בדיוק כפי שנקראה**. זאת כדי שניתן יהיה לראות בפלט את הפקודות המקוריות, גם כאשר הקלט מגיע מקובץ.
- **חובה לצרף להגשה הרצות בדיקה (אחת או יותר)**, המדגימות את השימוש בכל סוגי הפקודות על קבוצות, וכן את הטיפול בכל מגוון השגיאות בקלט. מומלץ להכניס פקודת הדפסה אחרי כל פקודה, כדי להראות שהתוצאה אכן נכונה (ראו לעיל הדוגמה של סדרת פקודות תקינות). **יש להגיש קובץ קלט** + **תדפיס מסך (או קובץ פלט) של כל הרצה**.

<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) 23

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 6,7,8

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

סמסטר: 2018ב' **מועד אחרון להגשה:** 10.6.2018

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- . יש להגיש את קבצי המקור (.h, .c), קבצי ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי makefile), וכן קבצי קלט ותדפיסי מסך או קבצי פלט (לפי הנחיות במטלה/במפגש/באתר). כל תכנית תהיה בתיקיה נפרדת. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת c.

יש להגיש תכניות מלאות (בין השאר מכילות main**), ניתנות להידור והרצה, ומאפשרות בדיקה של כל התוצאות המגוונות של הריצה ללא צורך בשינויים כלשהם בקוד התכנית.**

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

שאלה 1 (10 נקודות)

בכל סעיף, עליכם לכתוב האם "**תמיד נכון**" בשפת ANSI-C, "**לפעמים נכון ולפעמים אינו נכון**", או "**תמיד אינו נכון**". עליכם לנמק את תשובתכם. תשובה לא מנומקת, גם אם היא נכונה, לא תזכה בנקודות (כל סעיף 5 נקודות).

א. קובץ הקלט הסטנדרטי stdin הוא **תמיד** המקלדת, קובץ הפלט הסטנדרטי stdout הוא **תמיד** המסך, וקובץ השגיאות הסטנדרטי stderr הוא **תמיד** המסך.

ב. ניתן לשנות את תחום האינדקסים של מערך. ברירת המחדל היא אינדקסים החל מ- 0, אך ניתן לשנות זאת לאינדקסים החל מ- 1.

<u>הערה</u>: את התשובות לשאלה זו יש להקליד לקובץ, ולצרף את הקובץ להגשה.

שאלה 2 (90 נקודות) (תכנית ראשית בקובץ permut.c)

הגדרה: **תמורה (permutation) של מחרוזת תווים** היא מחרוזת המתקבלת על ידי ערבוב כלשהו של סדר התווים במחרוזת המקורית.

עליכם לכתוב תכנית המקבלת שני ארגומנטים בשורת הפקודה: שם של קובץ קלט, ומחרוזת תווים. על התוכנית לסרוק את תוכן הקובץ, ולהדפיס לפלט הסטנדרטי את כל התמורות של המחרוזת אשר תימצאנה בקובץ.

התמורות תודפסנה לפי סדר הופעתן בקובץ. כל תמורה תודפס בשורה חדשה. ייתכנו מופעים מרובים של כל תמורה, ויש להדפיס את כולם. הניחו שאין בקובץ תמורות שחופפות זו לזו.

לדוגמה: אם שורת הפקודה נראית כך:

> prog data.in chair

ותוכן הקובץ **data.in** הוא:

I am a chair
You are an arich and not a CHAIR
We are haircs and not gariches
Goodbye

הפלט של התוכנית יהיה:

chair
arich
hairc
arich

על התוכנית להדפיס הודעת שגיאה מתאימה לקובץ **השגיאות** הסטנדרטי, ולהפסיק את עבודתה, במקרים הבאים: מספר הארגומנטים בשורת הפקודה אינו כנדרש; בעיה בפתיחת הקובץ; הקובץ ריק.

במקרה והתוכנית סרקה את כל תוכן הקובץ אך לא מצאה אף תמורה, עליה להדפיס הודעה מתאימה לקובץ **הפלט** הסטנדרטי, ולסיים את ריצתה.

חובה לצרף להגשה מספר הרצות בדיקה המדגימות את פעולת התכנית, לרבות הטיפול בשגיאות. יש להגיש **קובץ קלט** + **תדפיס מסך** של כל הרצת דוגמה.

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) 14

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרויקט גמר

מספר השאלות: 1 | משקל המטלה: 31 נקודות

סמסטר: 2018ב' | מועד אחרון להגשה: 19.8.2018

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.

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

עליכם להגיש:

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

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

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

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

נזכיר מספר היבטים חשובים:

1. הפשטה של מבני הנתונים: רצוי (במידת האפשר) להפריד בין הגישה למבני הנתונים לבין המימוש של מבני הנתונים. כך, למשל, בכתיבת שגרות לטיפול בטבלה, אין זה מעניינם של המשתמשים בשגרות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקושרת.

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

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

<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.

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

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

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

<u>רקע כללי ומטרת הפרויקט</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 (בתים, מילים). לא ניתן להבחין, בעין שאינה מיומנת, בהבדל פיסי כלשהו בין אותו חלק בזיכרון שבו נמצאת תוכנית לבין שאר הזיכרון.

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מספר מסוים של הוראות פשוטות, ולשם כך היא משתמשת בזיכרון המחשב ובאוגרים (registers) הקיימים בתוך היע"מ. <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 המרכיבות את תוכנית המשתמש כפי שהיא נמצאת בזיכרון. כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנת), תתורגם בסופו של דבר באמצעות תוכנה מיוחדת לצורה סופית זו.

קוד בשפת מכונה הוא רצף של ביטים המהווים קידוד של סדרת הוראות (תוכנית) שעל היע"מ לבצע. קוד מכונה אינו קריא למשתמש, ולכן לא נוח לקודד(או לקרוא) תכניות ישירות בשפת מכונה. שפת אסמבלי (assembly language) היא שפת תכנות מאפשרת לייצג את ההוראות של שפת המכונה בצורה סימבולית. כמובן שיש צורך לתרגם את הייצוג הסימבולי לקוד מכונה כדי שהתוכנית תוכל לרוץ במחשב. תרגום זה נעשה באמצעות כלי שנקרא אסמבלר (assembler).

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

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

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

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

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

המחשב הדמיוני ושפת האסמבלי

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

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

"חומרה":

המחשב בפרויקט מורכב מיע"מ (יחידת עיבוד מרכזית), אוגרים וזיכרון RAM, כאשר חלק מהזיכרון משמש גם כמחסנית (stack). גודלה של מילת זיכרון במחשב הוא 14 סיביות.
האריתמטיקה נעשית בשיטת המשלים ל-2 (2's complement). מחשב זה עובד רק עם מספרים שלמים חיוביים ושליליים, אין תמיכה במספרים ממשיים.

אוגרים:

למעבד 8 אוגרים כלליים (r0,r1,r2,r3,r4,r5,r6,r7).
גודלו של כל אוגר הוא 14 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 13.

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

גודל הזיכרון הוא 256 תאים, בכתובות 0-255 (בבסיס עשרוני), וכל תא הוא בגודל של 14 סיביות.
לתא בזיכרון נקרא גם בשם "מילה". הסיביות בכל מילה ממוספרות בדומה לאוגר, כמפורט לעיל.

קידוד של תווים (characters) נעשה בקוד ascii.

מבנה הוראת מכונה:

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

13 12	11 10	9 8 7 6	5 4	3 2	1 0
פרמטר1	פרמטר2	opcode	מיעון אופרנד מקור	מיעון אופרנד יעד	E,R,A

קידוד כל מילה בקוד המכונה ייעשה בבסיס 2 "מוזר" המוגדר כדלקמן:
0 שקול לתו נקודה (0=.) , ו- 1 שקול לתו / (1=/). ראו דוגמאות בסוף הגדרת הפרויקט.

סיביות 6-9 במילה הראשונה של ההוראה מהוות את קוד הפעולה (opcode). כל opcode מיוצג בשפת אסמבלי על ידי "שם פעולה". בשפה שלנו יש 16 קודי פעולה והם:

שם הפעולה	קוד הפעולה (בבסיס 10)
mov	0
cmp	1
add	2
sub	3

4	not
5	clr
6	lea
7	inc
8	dec
9	jmp
10	bne
11	red
12	prn
13	jsr
14	rts
15	stop

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

סיביות 0-1 (A,R,E)

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

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

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

ראו הסבר נוסף על סיביות אלה בהמשך, בתיאור קובץ הפלט המכיל את קוד המכונה.

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) .

סיביות 10-13 רלוונטיות רק אם מדובר בשיטת מיעון מספר 2 (ראו הסבר בהמשך), ואחרת ערכן הוא 0. הסיביות מאפיינות שני פרמטרים. סיביות 10-11 אחראיות על פרמטר2, ואילו סיביות 12-13 אחראיות על פרמטר1. הסבר מפורט מופיע בהמשך, בדיון על שיטות מיעון.
אם הפרמטר הוא מספר מיידי, הסיביות יקבלו 00.
אם הפרמטר הוא אוגר, הסיביות יקבלו 11.
אם הפרמטר הוא שם של תוית, הסיביות יקבלו 01

<u>שיטות מיעון:</u>

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

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

להלן תיאור של ארבע שיטות המיעון:

ערך	שיטת מיעון	תוכן המילים הנוספות	אופן הכתיבה	דוגמא
0	מיעון מידי	המילה הנוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר המיוצג ב- 12 סיביות, אליהם מתווספות זוג סיביות של שדה A,R,E.	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני	mov #-1,r2 בדוגמה זו האופרנד הראשון של הפקודה נתון בשיטת מיעון מיידי. ההוראה כותבת את הערך 1- לתוך אוגר r2
1	מיעון ישיר	המילה הנוספת של ההוראה מכילה מען של מילה בזיכרון. מילה זו בזיכרון הינה האופרנד. המען מיוצג ב- 12 סיביות אליהן מתווספות זוג סיביות של שדה A,R,E.	האופרנד הינו <u>תווית</u> שכבר הוצהרה או שתוצהר בהמשך הקובץ. ההצהרה נעשית על ידי כתיבת תווית בתחילת הנחיית 'data.' או 'string.', או בתחילת הוראה של התוכנית, או באמצעות אופרנד של הנחיית 'extern.'	נתונה למשל ההגדרה: x: .data 23 אפשר לכתוב הוראה: dec x בדוגמה זו, ההוראה מקטינה ב-1 את תוכן המילה שבכתובת x בזיכרון (ה"משתנה" x).
2	מיעון קפיצה עם פרמטרים	שיטת מיעון זו רלוונטית וניתנת לשימוש רק בפעולות קפיצה (jmp,bne,jsr). השיטה מיועדת להעברה נוחה של שני פרמטרים לצורך שימוש ביעד הקפיצה. הפרמטר הראשון יועבר באוגר r6, והשני באוגר r7 (התוכן הקודם של אוגרים אלה נדרס). בשיטת מיעון זו יש לכל היותר 3 מילות מידע נוספות בקוד ההוראה. המילה הראשונה הנוספת היא כתובת התוית אליה קופצים. המילה הנוספת השניה היא קידוד הפרמטר הראשון. המילה הנוספת השלישית היא קידוד הפרמטר השני. ייתכן ושני הפרמטרים יקודדו למילה משותפת, ואז יהיו רק 2 מילות מידע נוספות. לכל אחת מהמילים הנוספות של שיטת המיעון מתווספות זוג סיביות של שדה A,R,E.	האופרנד מורכב משם של תוית אליה רוצים לקפוץ, ואחריה בתוך סוגריים שני הפרמטרים, מופרדים זה מזה בפסיק. אין רווחים בין מרכיבי האופרנד. לא ניתן לציין יותר או פחות משני פרמטרים. כל פרמטר יכול להיות מספר מיידי (בדומה לשיטת מיעון 0), או תוית (בדומה לשיטת מיעון 1), או אוגר (בדומה לשיטת מיעון 3 – ראו הגדרה בהמשך הטבלה). הסיביות של כל פרמטר במילת הקוד הראשונה ייקבעו בהתאם לסוג הפרמטר. המילה הנוספת תקודד בדומה לשיטת המיעון המקבילה. אם הפרמטר הראשון הוא אוגר, הוא יקודד במילה נוספת, כמו אוגר מקור בשיטת מיעון 3. אם הפרמטר השני אוגר, הוא יקודד כמו אוגר יעד. אם שני הפרמטרים הם אוגרים, הם יחלקו מילה משותפת.	jmp L1(#5,N) בדוגמה זו, הקבוע 5 מקודד במילה השלישית של ההוראה, והכתובת המיוצגת על ידי התווית N מקודדת במילה הרביעית. ההוראה מבצעת קפיצה לתוית L1, ומועברים שני פרמטרים: הקבוע 5 (באוגר r6), והמילה שבכתובת N (באוגר r7). או למשל, jsr L1(r3,r5) בדוגמה זו, שני הפרמטרים הם אוגרים, והם חולקים את המילה השלישית של קוד ההוראה.

3	מיעון אוגר ישיר	האופרנד הוא אוגר. אם האוגר משמש כאופרנד יעד, מילה נוספת של הפקודה תכיל בששת הסיביות 2-7 את מספרו של האוגר. אם האוגר משמש כאופרנד מקור, הוא יקודד במילה נוספת שתכיל בששת הסיביות 8-13 את מספרו של האוגר. אם בפקודה יש שני אופרנדים, ושניהם אוגרים, הם יחלקו מילה נוספת אחת משותפת, כאשר הסיביות 2-7 הן עבור אוגר היעד, והסיביות 8-13 עבור אוגר המקור. לייצוג זה מתווספות זוג סיביות של שדה A,R,E. סיביות שאינן בשימוש יכילו 0.	האופרנד הינו שם של אוגר.	mov r1,r2 בדוגמה זו, ההוראה מעתיקה את תוכן אוגר r1 לתוך אוגר r2. בדוגמה זו שני האופרנדים הם אוגרים, אשר יקודדו למילה נוספת משותפת.

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

<u>אפיון הוראות המכונה</u>:
הוראות המכונה מתחלקות לשלוש קבוצות, לפי מספר האופרנדים הדרוש להן.

קבוצת ההוראות הראשונה:
הוראות הדורשות שני אופרנדים. ההוראות השייכות לקבוצה זו הן:
mov, cmp, add, sub, lea

הפעולה	הסבר הפעולה	דוגמא	הסבר הדוגמא
mov	מבצעת העתקה של האופרנד הראשון, אופרנד המקור (source) אל האופרנד השני, אופרנד היעד (destination) (בהתאם לשיטת המיעון).	mov A, r1	העתק תוכן המשתנה A (המילה שבכתובת A בזכרון) לאוגר r1.
cmp	מבצעת "השוואה" בין שני האופרנדים שלה. אופן ההשוואה: תוכן אופרנד היעד (השני) מופחת מתוכן אופרנד המקור (הראשון), ללא שמירת תוצאת החיסור. פעולת החיסור מעדכנת דגל בשם Z ("דגל האפס") באוגר הסטטוס(PSW).	cmp A, r1	אם תוכן המשתנה A זהה לתוכנו של אוגר r1 אזי דגל האפס, Z, באוגר הסטטוס (PSW) יודלק, אחרת הדגל יאופס.
add	אופרנד היעד (השני) מקבל את תוצאת החיבור של אופרנד המקור (הראשון) והיעד (השני).	add A, r0	אוגר r0 מקבל את תוצאת החיבור של תוכן המשתנה A ותוכנו הנוכחי של אוגר r0.
sub	אופרנד היעד (השני) מקבל את תוצאת החיסור של אופרנד המקור (הראשון) מאופרנד היעד (השני)	sub #3, r1	אוגר r1 מקבל את תוצאת החיסור של הערך 3 מתוכנו הנוכחי של האוגר r1.
lea	lea הינו ראשי תיבות של load effective address. פעולה זו מבצעת טעינה של המען בזיכרון המייוצג על ידי התווית שבאופרנד הראשון (המקור), אל אופרנד היעד (האופרנד השני).	lea HELLO, r1	המען שמייצגת התווית HELLO מוכנס לאוגר r1.

קבוצת ההוראות השניה:

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

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

פקודה	הסבר פעולה	דוגמא	הסבר דוגמא
not	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0).	not r2	r2 ← not r2
clr	איפוס תוכן האופרנד.	clr r2	r2 ← 0
inc	הגדלת תוכן האופרנד באחד.	inc r2	r2 ← r2 + 1
dec	הקטנת תוכן האופרנד באחד.	dec C	C ← C – 1
jmp	קפיצה בלתי מותנית (עם או בלי פרמטרים) אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כלומר, בעת ביצוע ההוראה, מצביע התוכנית (PC) יקבל את ערך אופרנד היעד. אם ההוראה כוללת פרמטרים, ערכי הפרמטרים מועתקים לאוגרים r6, r7.	jmp LINE או jmp LINE(#–6,r4)	PC ← LINE PC ← LINE r6 ← –6 r7 ← r4
bne	bne הינו ראשי תיבות של: branch if not equal (to zero). זוהי הוראת הסתעפות מותנית (עם או בלי פרמטרים). מצביע התוכנית (PC) יקבל את ערך אופרנד היעד אם ערכו של הדגל Z באוגר הסטטוס (PSW) הינו 0. כזכור, הדגל Z נקבע בהוראת cmp. אם ההסתעפות מתבצעת, וההוראה כוללת פרמטרים, ערכי הפרמטרים יועתקו לאוגרים r6, r7. אחרת, האוגרים נשארים ללא שינוי.	bne LINE או bne LINE(X,r4)	אם ערך הדגל Z באוגר הסטטוס (PSW) הינו 0: PC ← LINE אם ערך הדגל Z באוגר הסטטוס (PSW) הינו 0: PC ← LINE r6 ← word at X r7 ← r4
red	קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מהקלט הסטנדרטי יוכנס לאוגר r1.
prn	הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn r1	התו אשר קוד ה-ascii שלו נמצא באוגר r1 יודפס לפלט הסטנדרטי.
jsr	קריאה לשגרה (סברוטינה), עם או בלי פרמטרים. מצביע התוכנית (PC) הנוכחי נדחף לתוך המחסנית שבזכרון המחשב, והאופרנד מוכנס ל-PC. אם ההוראה כוללת פרמטרים, ערכי הפרמטרים מועתקים	jsr FUNC או jsr FUNC(#75,X)	push(PC) PC ← FUNC push(PC) PC ← FUNC

	לאוגרים r6, r7.		r6 ← 75 r7 ← word at X

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

ההוראות השייכות לקבוצה זו הן: rts, stop.

פקודה	הסבר פעולה	דוגמא	הסבר דוגמא
rts	חזרה משיגרה. הערך בראש המחסנית של זמן ריצה מוצא מן המחסנית ומועבר לאוגר התוכנית (PC).	rts	PC ← pop()
stop	עצירת ריצת התוכנית.	stop	התכנית עוצרת

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

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

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

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

סוג המשפט	הסבר כללי
משפט ריק	זוהי שורה המכילה אך ורק תווים לבנים (whitespace), כלומר מכילה רק תווים '\t' ו-' ' (סימני tab ורווח). ייתכן ובשורה אין אף תו (למעט התו \n).
משפט הערה	זוהי שורה בה התו הראשון הינו ';' (נקודה פסיק). על האסמבלר להתעלם לחלוטין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמבלר מה עליו לעשות כשהוא פועל על תכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצאה ואתחול משתנים של התכנית, אך הוא אינו מייצר קוד המיועד לביצוע בעת ריצת התכנית.
משפט הוראה	זהו משפט המייצר קוד לביצוע בעת ריצת התכנית. המשפט מכיל שם של 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 בתחביר חוקי. התחביר של תווית חוקית יתואר בהמשך). התווית היא אופציונלית.

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

יש לשים לב: למילות הקוד הנוצרות ממשפט הנחיה לא מצורפות זוג סיביות A,R,E והקידוד ממלא את כל 14 הסיביות של המילה.

יש ארבעה סוגים של משפטי הנחיה , והם:

1. ‘.data’

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

.data 7, −57, +17, 9

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

משפט ההנחיה: ‘.data’מנחה את האסמבלר להקצות מקום בתמונת הנתונים (data image), אשר בו יאוחסנו הערכים המתאימים, ולקדם את מונה הנתונים, בהתאם למספר הערכים ברשימה. אם להוראת data. יש גם תווית, אזי תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום), ומוכנסת אל טבלת הסמלים. דבר זה מאפשר להתייחס אל מקום מסוים בתמונת הנתונים דרך שם התווית (למעשה, זוהי הגדרת שם של משתנה).

כלומר אם נכתוב:

XYZ: .data 7, –57, +17, 9

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

אם נכתוב בתכנית הוראה לביצוע:

mov XYZ, r1

אזי בזמן ריצת התכנית יוכנס לאוגר r1 הערך 7.

ואילו ההוראה:

lea XYZ,r1

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

2. ‘.string’

להנחיה ‘.string’ פרמטר אחד שהוא מחרוזת חוקית. תווי המחרוזת מקודדים לפי ערכי ה-ascii המתאימים ומוכנסים אל תמונת הנתונים, לפי סדרם, כל תו במילה נפרדת. בסוף המחרוזת יוכנס הערך אפס, לסמן סיום מחרוזת. מונה הנתונים של האסמבלר יוגדל בהתאם לאורך המחרוזת+ 1. אם בשורת ההנחיה מגדרת גם תווית, אזי ערכה יהיה מען המקום בזיכרון, שבו מאוחסן התו הראשון במחרוזת, באופן דומה כפי שנעשה עבור ‘.data’.

לדוגמא, משפט ההנחיה:

STR: .string “abcdef”

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

3. '.entry'

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

לדוגמא, השורות:

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

מודיעות שאפשר להתייחס מקובץ אחר לתווית HELLO המוגדרת בקובץ הנוכחי.

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

4. '.extern'

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

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

```
.extern   HELLO
```

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

<u>משפט הוראה</u>:

משפט הוראה מורכב מ:
1. תווית אופציונלית.
2. שם פעולה.
3. 0, 1 או 2 אופרנדים בהתאם לסוג הפעולה.

שם הפעולה נכתב באותיות קטנות (lower case), והפעולה היא אחת מבין 16 הפעולות שהוזכרו לעיל.

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

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזה בתו ',' (פסיק). כמו בהנחייה data., **<u>לא חייבת להיות שום הצמדה של האופרנדים לפסיק או לשם הפעולה באופן כלשהו. כל כמות רווחים או tabs בין האופרנדים לפסיק ובין שם הפעולה לאופרנד הראשון היא חוקית.</u>**

להוראה בעלת שני אופרנדים המבנה הבא:

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

לדוגמא:

HELLO: add r7, B

לפקודה בעלת אופרנד אחד המבנה הבא:

אופרנד שם-הפעולה :תווית-אופציונלית

לדוגמא:

HELLO: bne XYZ

להוראה ללא אופרנדים המבנה הבא:

שם-הפעולה :תווית-אופציונלית

לדוגמא:

END: stop

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

<u>תווית:</u>

תווית חוקית מתחילה באות אלפבית (גדולה או קטנה) ולאחריה סדרה כלשהי של אותיות וספרות. האורך המקסימלי של תווית הוא 31 תווים. הגדרת התווית מסתיימת על ידי התו ':' (נקודתיים). תו זה אינו מהווה חלק מהתווית. זהו רק סימן המציין את סופה. הגדרת התווית חייבת להתחיל בעמודה הראשונה בשורה. אסור שאותה תווית תוגדר יותר מפעם אחת (בשורות שונות).

לדוגמה, התוויות שלהלן הן תוויות חוקיות.

hEllo:
x:
He78902:

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

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

<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 תמיכה בשלמים בייצוג אחר מאשר עשרוני, ואין תמיכה במספרים ממשיים.

<u>מחרוזת:</u>

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

אסמבלר עם שני מעברים

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

לדוגמא: האסמבלר קורא את קטע הקוד הבא:

```
MAIN:     mov     r3 ,LENGTH
LOOP:     jmp     L1(#-1,r6)
          prn     #-5
          bne     LOOP(r4,r3)
          sub     r1, r4
          bne     END
L1:       inc     K
          bne     LOOP(K,STR)
END:      stop
STR:      .string "abcdef"
LENGTH:   .data   6,-9,15
K:        .data   22
```

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

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

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

Decimal Address	Source Code	Explanation	Binary Machine Code
0100	MAIN: mov r3 ,LENGTH	first word of instruction	**00000000110100**
0101		source register r3	**00001100000000**
0102		address of label LENGTH	**00001000001010**
0103	LOOP: jmp L1(#-1,r6)		**00111001001000**
0104		address of label L1	**00000111010010**
0105		immediate #-1 (2's complement)	**11111111111100**
0106		immediate r6	**00000000011000**
0107	prn #-5		**00001100000000**
0108		immediate #-5	**11111111101100**
0109	bne LOOP(r4,r5)		**11111010001000**
0110		address of LOOP	**00000110011110**
0111		registers r4,r5	**00010000010100**
0112	sub r1, r4		**00000011111100**
0113		registers r1,r4	**00000100010000**
0114	bne END		**00001010000100**
0115		address of label END	**00000111101010**
0116	L1: inc K		**00000111000100**
0117		address of label K	**00001000010110**
0118	bne LOOP(K,STR)		**01011010001000**
0119		address of label LOOP	**00000110011110**
0120		address of label K	**00001000010110**
0121		address of label STR	**00000111101110**
0122	END: stop		**00001111000000**

0123	STR: .string "abcdef"	ascii code 'a'	**00000001100001**
0124		ascii code 'b'	**00000001100010**
0125		ascii code 'c'	**00000001100011**
0126		ascii code 'd'	**00000001100100**
0127		ascii code 'e'	**00000001100101**
0128		ascii code 'f'	**00000001100110**
0129		ascii code '\0' (end of string)	**00000000000000**
0130 **0131** **0132**	LENGTH: .data 6,-9,15	integer 6 integer -9 (2's complement) integer 15	**00000000000110** **11111111110111** **00000000001111**
0133	K: .data 22	interger 22	**00000000010110**

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

כדי לעשות המרה לבינארי של אופרנדים שהם מענים סמליים (תוויות), יש צורך לבנות טבלה דומה. אולם בהבדל מהקודים של הפעולות, הידועים מראש, הרי המענים בזיכרון עבור הסמלים שבשימוש התכנית אינו ידוע, עד אשר התוכנית כולה נקראת ועוברת טיפול על ידי האסמבלר.

למשל, בדוגמא לעיל, האסמבלר אינו יכול לדעת שהסמל END משויך למען 122 (עשרוני), אלא רק לאחר שהתוכנית נקראה כולה.

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

במעבר הראשון נבנית טבלת סמלים בזיכרון, ובה לכל סמל שבתכנית המקור משוייך ערך שהוא מען בזיכרון. בדוגמא דלעיל, טבלת הסמלים לאחר מעבר ראשון היא:

סמל	**ערך (בבסיס דצימלי)**
MAIN	100
LOOP	103
L1	116
END	122
STR	123
LENGTH	130
K	133

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

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

המעבר הראשון

במעבר הראשון נדרשים כללים כדי לקבוע איזה מען ישויך לכל סמל. העיקרון הבסיסי הוא לספור את המקומות בזיכרון, אותם תופסות ההוראות. אם כל הוראה תיטען בזיכרון למקום העוקב להוראה הקודמת, תציין ספירה כזאת את מען ההוראה הבאה. הספירה נעשית על ידי האסמבלר ומוחזקת במונה ההוראות (IC) . ערכו ההתחלתי של IC הוא 0, ולכן נטענת ההוראה הראשונה במען 0. ה-IC מתעדכן בכל שורת הוראה המקצה מקום בזיכרון. לאחר שהאסמבלר קובע מהו אורך ההוראה, ה-IC מוגדל במספר התאים (מילים) הנתפסים על ידי ההוראה, וכך הוא מצביע על התא הפנוי הבא.

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

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

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

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

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

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

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

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

המעבר השני

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

הפרדת הוראות ונתונים

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

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

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

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

גילוי שגיאות בתכנית המקור

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

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

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

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

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

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

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

אלגוריתם שלדי של האסמבלר

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

אנו מחלקים את קוד המכונה לשני אזורים, אזור ההוראות (code) ואזור הנתונים (data) . לכל אזור יש מונה משלו, ונסמנם IC (מונה ההוראות - Instruction-Counter) ו-DC (מונה הנתונים - Data-Counter).

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

בכל מעבר מתחילים לקרוא את קובץ המקור מהתחלה.

מעבר ראשון

1. אתחל $DC \leftarrow 0, IC \leftarrow 0$.
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-16.
3. האם השדה הראשון הוא סמל? אם לא, עבור ל-5.
4. הדלק דגל "יש הגדרת סמל".
5. האם זוהי הנחיה לאחסון נתונים, כלומר, האם הנחית data. או string.? אם לא, עבור ל-8.
6. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם סימון (סמל מסוג data). ערכו יהיה DC. (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
7. זהה את סוג הנתונים, קודד אותם בזיכרון, עדכן את מונה הנתונים DC בהתאם לאורכם, חזור ל-2.
8. האם זו הנחית extern. או הנחית entry. ? אם לא, עבור ל-11.
9. האם זוהי הנחית extern. ? אם כן, הכנס כל סמל (אחד או יותר) המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים ללא ערך, עם סימון (סמל מסוג external).
10. חזור ל-2.
11. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם סימון (סמל מסוג code). ערכו יהיה IC (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
12. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא – הודע על שגיאה בשם ההוראה.
13. נתח את מבנה האופרנדים של ההוראה וחשב את L. בנה כעת את הקוד הבינארי של המילה הראשונה של הפקודה.
14. עדכן $IC \leftarrow L + IC$
15. חזור ל-2.
16. אם נמצאו שגיאות בקובץ המקור, עצור.
17. עדכן בטבלת הסמלים את ערכם של הסמלים מסוג data , ע"י הוספת הערך הסופי של IC (ראו הסבר בהמשך).
18. התחל מעבר שני.

מעבר שני

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

נפעיל אלגוריתם זה על תוכנית הדוגמא שהצגנו קודם:

```
MAIN:     mov    r3 ,LENGTH
LOOP:     jmp    L1(#-1,r6)
          prn    #-5
          bne    LOOP(r4,r7)
          sub    r1, r4
          bne    END
L1:       inc    K
          bne    LOOP(K,STR)
END:      stop
STR:      .string "abcdef"
LENGTH:   .data  6,-9,15
K:        .data  22
```

נבצע עתה מעבר ראשון על הקוד הנתון. נבנה את טבלת הסמלים. כמו כן, נבצע במעבר זה גם את קידוד כל הנתונים, וקידוד המילה הראשונה של כל הוראה. את החלקים שעדיין לא מתורגמים במעבר זה, נשאיר כמות שהם.

אנו מניחים שהקוד ייטען לזיכרון החל מהמען 100 (בבסיס דצימאלי).

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 **0101** **0102**	MAIN: mov r3 ,LENGTH	first word of instruction source register r3 address of label LENGTH	**00000000110100** **00001100000000** **?**
0103 **0104** **0105** **0106**	LOOP: jmp L1(#-1,r6)	 address of label L1 immediate #-1 (2's complement) immediate r6	**00111001001000** **?** **11111111111100** **00000000011000**
0107 **0108**	prn #-5	 immediate #-5	**00001100000000** **11111111101100**
0109 **0110** **0111**	bne LOOP(r4,r5)	 address of LOOP registers r4,r7	**11111010001000** **?** **00010000010100**
0112 **0113**	sub r1, r4	 registers r1,r4	**00000011111100** **00000100010000**
0114 **0115**	bne END	 address of label END	**00001010000100** **?**
0116 **0117**	L1: inc K	 address of label K	**00000111000100** **?**
0118 **0119** **0120** **0121**	bne LOOP(K,STR)	 address of label LOOP address of label K address of label STR	**01011010001000** **?** **?** **?**
0122	END: stop		**00001111000000**
0123	STR: .string "abcdef"	ascii code 'a'	**00000001100001**
0124		ascii code 'b'	**00000001100010**
0125		ascii code 'c'	**00000001100011**
0126		ascii code 'd'	**00000001100100**
0127		ascii code 'e'	**00000001100101**
0128		ascii code 'f'	**00000001100110**
0129		ascii code '\0' (end of string)	**00000000000000**
0130 **0131** **0132**	LENGTH: .data 6,-9,15	integer 6 integer -9 (2's complement) integer 15	**00000000000110** **11111111110111** **00000000001111**
0133	K: .data 22	interger 22	**00000000010110**

טבלת הסמלים:

סמל	ערך (בבסיס דצימלי)
MAIN	100
LOOP	103
L1	116
END	122
STR	123
LENGTH	130
K	133

נבצע עתה את המעבר השני ונרשום את הקוד בצורתו הסופית:

Decimal Address	Source Code	Binary Machine Code
0100 **0101** **0102**	MAIN: mov r3 ,LENGTH	**00000000110100** **00001100000000** **00001000001010**
0103 **0104** **0105** **0106**	LOOP: jmp L1(#-1,r6)	**00111001001000** **00000111010010** **11111111111100** **00000000011000**
0107 **0108**	prn #-5	**00001100000000** **11111111101100**
0109 **0110** **0111**	bne LOOP(r4,r5)	**11111010001000** **00000110011110** **00010000010100**
0112 **0113**	sub r1, r4	**00000011111100** **00000100010000**
0114 **0115**	bne END	**00001010000100** **00000111101010**
0116 **0117**	L1: inc K	**00000111000100** **00001000010110**
0118 **0119** **0120** **0121**	bne LOOP(K,STR)	**01011010001000** **00000110011110** **00001000010110** **00000111101110**
0122	END: stop	**00001111000000**
0123	STR: .string “abcdef”	**00000001100001**
0124		**00000001100010**
0125		**00000001100011**
0126		**00000001100100**
0127		**00000001100101**
0128		**00000001100110**
0129		**00000000000000**
0130 **0131** **0132**	LENGTH: .data 6,-9,15	**00000000000110** **11111111110111** **00000000001111**
0133	K: .data 22	**00000000010110**

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

לאחר סיום שלבים אלה, התוכנית תהיה טעונה בזיכרון ומוכנה לריצה.

קבצי קלט ופלט של האסמבלר

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

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

assembler x y hello

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

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

<u>מבנה כל קובץ פלט יתואר בהמשך</u>.

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

אופן פעולת האסמבלר

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

האסמבלר מחזיק שני מערכים, שייקראו להלן מערך ההוראות ומערך הנתונים. מערכים אלו נותנים למעשה תמונה של זיכרון המכונה (גודל כל כניסה במערך זהה לגודלה של מילת מכונה: 10סיביות). במערך ההוראות מכניס האסמבלר את הקידוד של הוראות המכונה שנקראו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמבלר את קידוד הנתונים שנקראו מקובץ המקור (שורות מסוג data. string.).

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

בנוסף יש לאסמבלר טבלה, אשר בה נאספות כל התוויות בהן נתקל האסמבלר במהלך המעבר על הקובץ. לטבלה זו קוראים טבלת סמלים (symbol-table). לכל סמל (תווית) נשמרים שמו, ערכו וטיפוסו (external או relocatable).

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

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

2. שורת הוראה:

האסמבלר מוצא מהי הפעולה, ומהן שיטות המיעון של האופרנדים. מספר האופרנדים אותם הוא מחפש נקבע בהתאם להוראה שנמצאה. האסמבלר קובע לכל אופרנד את ערכו באופן הבא:

- אם זה אוגר – האופרנד הוא מספר האוגר.
- אם זו תווית (מיעון ישיר) – האופרנד הוא ערך התווית כפי שמופיע בטבלת הסמלים (ייתכן והסמל טרם נמצא בטבלת הסמלים).
- אם זה מספר (מיעון מיידי) – האופרנד הוא המספר עצמו.
- אם זו שיטת מיעון אחרת – ערכו של האופרנד נקבע לפי המפרט של שיטת המיעון (ראו תאור שיטות המיעון לעיל)

קביעת שיטת המיעון נעשית בהתאם לתחביר של האופרנד, כפי שהוסבר לעיל בהגדרת שיטות המיעון. למשל, התו # מציין מיעון מיידי, תווית מציינת מיעון ישיר, שם של אוגר מציין מיעון אוגר, וכד'.

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

אם זוהי פעולה בעלת שני אופרנדים, אזי האסמבלר מכניס למערך ההוראות, במקום עליו מצביע מונה ההוראות IC, את קוד המילה הראשונה של ההוראה (בשיטת הייצוג של הוראות המכונה כפי שתואר קודם לכן). מילה זו מכילה את קוד הפעולה, ואת שיטות המיעון. הסיביות של הפרמטרים (סיביות 13-10) יכילו 0. בנוסף "משריין" האסמבלר מקום במערך עבור המילים הנוספות הנדרשות עבור הוראה זו, ומגדיל את מונה ההוראות בהתאם. אם אחד או שני האופרנדים הם בשיטת מיעון אוגר או מיידי, האסמבלר יקודד כעת גם את המילים הנוספות הרלוונטיות במערך ההוראות.

אם זוהי פעולה בעלת אופרנד אחד בלבד, כלומר אין אופרנד מקור, אזי הקידוד הינו דומה לעיל, פרט לסיביות של שיטת המיעון של אופרנד המקור במילה הראשונה, אשר יכילו תמיד 0, מכיוון שאינן רלוונטיות לפעולה. כמו כן, אם זו פעולת קפיצה, והאופרנד נתון בשיטת מיעון "קפיצה עם פרמטרים", יקודדו הסיביות של הפרמטרים במילה הראשונה. אם אחד או שני הפרמטרים הם אוגר או מיידי, האסמבלר יקודד כעת גם את המילים הנוספות הרלוונטיות במערך ההוראות.

אם זוהי פעולה ללא אופרנדים (rts, stop) אזי תקודד רק המילה הראשונה (והיחידה). הסיביות של שיטות המיעון של שני האופרנדים יכילו 0, וכך גם הסיביות של הפרמטרים.

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

3. שורת הנחיה:

כאשר האסמבלר נתקל בהנחיה, הוא פועל בהתאם לסוג ההנחייה, באופן הבא:

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

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

בסוף המעבר הראשון, ערך התווית יעודכן בטבלת הסמלים על ידי הוספת ה-IC (כלומר הוספת האורך הכולל של קידוד כל ההוראות). הסיבה לכך היא שבתמונת קוד המכונה, הנתונים מופרדים מההוראות, וכל הנתונים יופיעו אחרי כל ההוראות (ראה תאור קבצי הפלט בהמשך).

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

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

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

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

יש לשים לב! בהוראה או בהנחיה אפשר להשתמש בשם של סמל אשר ההצהרה עליו ניתנת בהמשך הקובץ (אם באופן ישיר על ידי תווית ואם באופן עקיף על ידי הנחיית 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 יכנס למען העוקב אחרי ההוראה הראשונה (תלוי במספר המילים של ההוראה הראשונה), וכך הלאה עד להוראה האחרונה.

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

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

עקרונית, קובץ object מכיל את תמונת הזיכרון שתוארה כאן. קובץ object מורכב משורות של טקסט. <u>השורה הראשונה</u> מכילה שני מספרים: אורך כולל של קטע ההוראות (במילות זיכרון) ואורך כולל של קטע הנתונים (במילות זיכרון). <u>השורות הבאות</u> מתארות את תוכן הזיכרון. בכל שורה שני מספרים: כתובת של מילה בזיכרון, ותוכן המילה. הכתובת תירשם בבסיס 10, והקידוד יירשם בבסיס 2 "מוזר" שהוגדר בתחילת הממן.

בהמשך מופיע קובץ object לדוגמא, כפי שאמור להיבנות על ידי האסמבלר.

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

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

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

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

פורמט קובץ ה-entries

קובץ ה-entries בנוי משורות טקסט. כל שורה מכילה שם של סמל שהוגדר כ- entry ואת ערכו, כפי שנמצא בטבלת הסמלים. הערכים מיוצגים בבסיס 10.

פורמט קובץ externals

קובץ ה-externals בנוי אף הוא משורות טקסט. כל שורה מכילה שם של סמל שהוגדר external, וכתובת בקוד המכונה בה יש קידוד של אופרנד המתייחס לסמל זה. כמובן שייתכן ויש מספר כתובות בקוד המכונה בהם מתייחסים לאותו סמל חיצוני. לכל התייחסות כזו תהיה שורה נפרדת בקובץ ה-externals. הכתובות מיוצגות בבסיס 10

להלן קובץ מקור (קלט) לדוגמא, בשם ps.as

```
; file ps.as

.entry LENGTH
.extern W
MAIN:       mov     r3 ,LENGTH
LOOP:       jmp     L1(#-1,r6)
            prn     #-5
            bne     W(r4,r5)
            sub     r1, r4
            bne     L3
L1:         inc     K
.entry LOOP
            bne     LOOP(K,W)
END:        stop
STR:        .string "abcdef"
LENGTH:     .data   6,-9,15
K:          .data   22
.extern   L3
```

להלן טבלת הקידוד המלא הבינארי שמתקבל מהפעלת האסמבלר על קובץ המקור, ולאחריה הפורמטים של קבצי הפלט השונים הנבנים על ידי האסמבלר.

Decimal Address	Source Code	Binary Machine Code
0100 **0101** **0102**	MAIN: mov r3 ,LENGTH	**00000000110100** **00001100000000** **00001000001010**
0103 **0104** **0105** **0106**	LOOP: jmp L1(#-1,r6)	**00111001001000** **00000111010010** **11111111111100** **00000000011000**
0107 **0108**	prn #-5	**00001100000000** **11111111101100**
0109 **0110** **0111**	bne W(r4,r5)	**11111010001000** **00000000000001** **00010000010100**
0112 **0113**	sub r1, r4	**00000011111100** **00000100010000**
0114 **0115**	bne L3	**00001010000100** **00000000000001**
0116 **0117**	L1: inc K	**00000111000100** **00001000010110**

0118 **0119** **0120** **0121**	bne LOOP(K,W)	**01011010001000** **00000110011110** **00001000010110** **00000000000001**
0122	END: stop	**00001111000000**
0123	STR: .string “abcdef”	**00000001100001**
0124		**00000001100010**
0125		**00000001100011**
0126		**00000001100100**
0127		**00000001100101**
0128		**00000001100110**
0129		**00000000000000**
0130 **0131** **0132**	LENGTH: .data 6,-9,15	**00000000000110** **11111111110111** **00000000001111**
0133	K: .data 22	**00000000010110**

הקובץ ps.ob:

הערה: שורת הכותרת ”…Base 10 addess“ להלן אינה חלק מהקובץ, ונועדה להבהרה בלבד.

הערה: יש לקודד גם אפסים מובילים, כלומר 14 סיביות ״מוזרות״ בכל מילה בקוד המכונה, ו- 4 ספרות עשרוניות בכל כתובת.

```
Base 10 address  Base 2 code

            23  11
0100            ........//./..
0101            ....//........
0102            ..../....././.
0103            ..///../../...
0104            ...../////.././
0105            ////////////..
0106            ..........//...
0107            ....//........
0108            /////////.//..
0109            /////./.../...
0110            ............./
0111            .../....././..
0112            ...../////////..
0113            ...../.../....
0114            ...././..../..
0115            ............./
0116            ...../////.../..
0117            ..../...././/.
0118            ././/./.../...
0119            .....//..////.
0120            ..../...././/.
0121            ............./
0122            ..../////......
0123            ......./////..../
0124            ......./////.../.
0125            ......./////...//
0126            ......./////../..
0127            ......./////.././
0128            ......./////.//.
0129            .............
0130            ............//.
0131            ///////////.///
```

0132	**…….////**
0133	**…….././/.**

הקובץ ps.ent:

כל ערכי הסמלים מיוצגים בבסיס 10.

LOOP	103
LENGTH	130

הקובץ ps.ext:

כל הכתובות מיוצגות בבסיס 10.

W	110
L3	115
W	121

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

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

סיכום והנחיות כלליות

- אורך התוכנית, הניתנת כקלט לאסמבלר, אינו ידוע מראש (ואינו קשור לגודל 256 של הזיכרון במעבד הדמיוני). ולכן אורכה של התוכנית המתורגמת אינו אמור להיות צפוי מראש. אולם בכדי להקל במימוש התכנית, ניתן להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים לאכסון קוד המכונה בלבד. מבני נתונים אחרים בפרויקט (למשל טבלת הסמלים), יש לממשל על פי יעילות/תכונות נדרשות (למשל באמצעות הקצאת זיכרון דינמי ורשימה מקושרת).

- קבצי הפלט של האסמבלר צריכים להיות בפורמטים המופיעים למעלה. שמם של קבצי הפלט צריך להיות תואם לשם קובץ הקלט, פרט לסיומות. למשל, אם קובץ הקלט הוא prog.as אזי קבצי הפלט שייווצרו הם : prog.ob, prog.ext, prog.ent .

- אופן הפעלת האסמבלר צריך להיות תואם לנדרש בממ״ן, ללא שינויים כלשהם. אין להוסיף תפריטים אינטראקטיביים למיניהם, וכד׳. הפעלת התוכנית תיעשה רק עם ארגומנטים של שורת פקודה.

- יש להקפיד לחלק את מימוש האסמבלר למספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסוגים שונים במודול יחיד. מומלץ לחלק למודולים כגון : מבני נתונים, מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחבירי של שורה), טבלת סמלים, ועוד.

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

- יש להקפיד להתעלם מרווחים וטאבים, ולהיות סלחנים כלפי תוכנית קלט העושה שימוש ביותר רווחים מהנדרש. למשל, אם בהוראה יש שני אופרנדים המופרדים בפסיק, אזי לפני או אחרי הפסיק יכולים להיות רווחים וטאבים בכל כמות. בדומה, גם לפני ואחרי שם ההוראה. בכל המקרים האלה השורה צריכה להיחשב חוקית (לפחות מבחינת הרווחים).

- במקרה שתוכנית הקלט בשפת אסמבלי מכילה שגיאות תחביריות, נדרש לגלות ולדווח, כמו באסמבלר אמיתי, על כל השגיאות הקיימות, ולא לעצור לאחר גילוי השגיאה הראשונה. כמובן שאם הקלט מכיל שגיאות, אין טעם להפיק קבצי פלט (ob, ext, ent).

תם ונשלם פרק ההסברים והגדרת הפרויקט.

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

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

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

ב ה צ ל ח ה !