ה א ו נ י ב ר ס י ט ה ה פ ת ו ח ה

20465
מעבדה בתכנות מערכות
חוברת הקורס – אביב 2023ב

כתבה: מיכל אבימור

מרץ 2023 – סמסטר אביב – תשפ"ג

פנימי – לא להפצה.

תוכן העניינים

אל הסטודנט	א
1. לוח זמנים ופעילויות	ג
2. תיאור המטלות	ה
3. התנאים לקבלת נקודות זכות	ז
ממ״ן 11	1
ממ״ן 12	5
ממ״ן 22	9
ממ״ן 23	17
ממ״ן 14	19

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.

- **שאילתא** - לפניות בנושאים אקדמיים שונים כגון מועדי בחינה מעבר לטווח זכאות ועוד, אנא עשו שימוש מסודר במערכת הפניות דרך שאילתא. לחצו על הכפתור פניה חדשה ואחר כך לימודים אקדמיים > משימות אקדמיות, ובשדה פניות סטודנטים: השלמת בחינות בקורס. המערכת תומכת גם בבקשות מנהלה שונות ומגוונות.

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 (מס' קורס 20465 / ב2023)

שבוע לימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח ממ"ן (למנחה)
1	10.03.2023-5.03.2023	ספר C פרקים 3-2-1	מפגש ראשון	
2	17.03.2023-12.03.2023	ספר C פרקים 3-2-1		
3	24.03.2023-19.03.2023	ספר C פרק 4	מפגש שני	
4	31.03.2023-26.03.2023	ספר C פרק 4		
5	07.04.2023-02.04.2023 (ד-ו פסח)	ספר C פרק 5	מפגש שלישי	ממ"ן 11 02.04.2023
6	14.04.2023-09.04.2023 (א-ד פסח)	ספר C פרק 5		
7	21.04.2023-16.04.2023 (ג יום הזכרון לשואה)	ספר C פרק 6	מפגש רביעי	
8	28.04.2023-23.04.2023 (ג יום הזיכרון, ד יום העצמאות	ספר C פרק 6		
9	05.05.2023-30.04.2023	ספר C פרק 6,7	מפגש חמישי	ממ"ן 12 30.04.2023

** 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	12.05.2023-07.05.2023 (ג ל"ג בעומר)	ספר C פרק 7		
11	19.05.2023-14.05.2023	ספר C פרק 7	מפגש שישי	
12	26.05.2023-21.05.2023 (ו שבועות)	ספר C פרק 8 + פרויקט		ממ"ן 22 21.05.2023
13	02.06.2023-28.05.2023	פרויקט וחזרה	מפגש שביעי	
14	09.06.2023-04.06.2023	פרויקט וחזרה		ממ"ן 23 04.06.2023
15	16.06.2023-11.06.2023	פרויקט וחזרה	מפגש שמיני	ממ"ן 14** 11.08.2023

מועדי בחינות הגמר יפורסמו בנפרד

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

.

***** לא תינתן דחייה בהגשת הפרויקט (ממ"ן 14), פרט למקרים חריגים של מילואים או אשפוז, במקרים אלו יש לתאם את מועד ההגשה מראש עם מנחה הקבוצה.***

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	61 (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 ובפרויקט 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 והפרויקט 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: 2023ב' **מועד אחרון להגשה:** 02.04.2023

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 וקבצי פלט .

הקבצים של כל תוכנית יהיו 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 התוצאות המגוונות של הריצה ללא צורך בשינויים כלשהם בקוד המקור של התוכנית.

את המטלה יש להגיש בקובץ zip.

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

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

יש לקרוא בספר הלימוד, את המוסבר לגבי הפונקציות הבאות:

strcmp

strncmp

strchr

עליכם לכתוב מימוש **משלכם** לפונקציות לעיל, כאשר שמות הפונקציות יהיו:

my_strcmp

my_strncmp

my_strchr

בהתאמה. בנוסף **עליכם לכתוב** תוכנית (פונקציה בשם main) המאפשרת קליטת נתונים, הרצת הפונקציות שכתבתם והדפסת תוצאותיהן.

<u>הערה 1:</u> במימוש של הספרייה הסטנדרטית לפונקציה strchr, הערך המוחזר של הפונקציה הוא מטיפוס מצביע. במימוש שעליכם לכתוב (my_strchr), על הפונקציה להחזיר ערך int, שמציין את האינדקס של התו מתחילת המערך. אם התו לא נמצא יש להחזיר מספר שלילי (מינוס אחד).

<u>הערה 2:</u> ניתן לקלוט את "מילת הקלט" הבאה (כאשר "מילת קלט" מוגדרת כאוסף של תווים בין שני תווים "לבנים" - רווח, \t, או \n) על ידי שימוש בפקודת scanf באופן הבא:

scanf("%s", *character_vector_name*);

אשר יגרום לקריאת "מילת הקלט" הבאה, לתוך המערך ששמו מופיע כפרמטר השני של הפונקציה. על המערך להיות מספיק גדול בכדי לקלוט את "מילת הקלט". בשאלה זו ניתן להניח ערך מקסימום 80.

<u>הנחיות והערות נוספות</u>:

- על התוכנית **להדפיס הודעת בקשה ידידותית לקלט** המפרטת מה על המשתמש להקליד.
- הקלט לתוכנית הוא מ-stdin, ויכול להגיע **מהמקלדת או מקובץ** (באמצעות redirection בעת הפעלת התוכנית). לנוחיותכם, הכינו מספר קבצי קלט והשתמשו בהם שוב ושוב לדיבוג התוכנית.

חובה לצרף להגשה הרצות בדיקה (אחת או יותר), המדגימות את פעולת התכנית על מגוון קלטים. **יש להגיש תדפיסי מסך של כל ההרצות. אם תשתמשו בקבצי קלט, יש להגיש גם אותם**.

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

עליכם לכתוב פונקציה בשם count_bits אשר מסכמת ומחזירה את מספר הביטים, הדלוקים במקומות זוגיים, במשתנה מסוג unsigned long.

תזכורת: מנין הביטים מתחיל מהמספר 0.

<u>דוגמה 1</u>: המשתנה מכיל 10010100

ביט 2 הוא זוגי ודלוק לכן המספר שיוחזר הוא 1.

<u>דוגמה 2</u>: המשתנה מכיל 01010110

ביטים 6,4,2 הם זוגיים ודלוקים, לכן המספר שיוחזר הוא 3.

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 כפי שנקלטו**, וזאת <u>לפני</u> הקריאה לפונקציה. באופן זה, הקלט יוצג גם כאשר הוא מגיע מקובץ.

חובה לצרף להגשה מספר הרצות בדיקה, המדגימות את פעולת התכנית על מגוון נתוני קלט. **יש להגיש תדפיסי מסך (וקבצי פלט) של כל ההרצות**. כמו כן יש להגיש קבצי קלט.

לתשומת לבכם: לא תינתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או אשפוז. במקרים אלו יש לבקש ולקבל אישור **מראש** ממנחה הקבוצה.

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: 2023ב' **מועד אחרון להגשה:** 30.04.2023

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), קובץ ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קובץ makefile), וכן קבצי קלט ותדפיסי מסך ו קבצי פלט.
קבצי התוכנית יהיו בתיקיה. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת c.
יש להגיש תכנית מלאה (בין השאר מכילה main), הניתנת להידור והרצה, ומאפשרת בדיקה של כל התוצאות המגוונות של הריצה ללא צורך בשינויים כלשהם בקוד המקור של התוכנית.
את המטלה יש להגיש בקובץ zip.

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

שאלה 1 (תכנית ראשית בקובץ my_set. c)

עליכם לכתוב תוכנית המקבלת בקלט רשימת ערכים שלמים (מטיפוס int). מרשימת ערכים אלו עליכם לבנות קבוצה.

<u>תזכורת</u>: על פי הגדרה, קבוצה (set) היא אוסף איברים ברי מניה, שכולם <u>שונים</u> זה מזה.

על התוכנית לקלוט מהמשתמש את רשימת הערכים, לבנות מהם את הקבוצה, ולבסוף להדפיס את איברי הקבוצה לפי סדר הגעתם.

ניתן להניח תקינות של הקלט, אך ייתכנו ערכים שחוזרים יותר מפעם אחת ברשימת הקלט. עבור ערך כזה (שמופיע יותר מפעם אחת בקלט), הופעתו הראשונה קובעת את מיקומו בהדפסת הפלט.

מספר האיברים בקבוצה אינו מוגבל ויש להשתמש בפונקציה realloc.
גם מספר הערכים בקלט אינו מוגבל. הקלט יכול להיות במספר רב של שורות, ובכל שורה יכולים להופיע מספר בלתי מוגבל של ערכים. סוף הקלט יזוהה לפי הערך EOF שמוחזר מפונקציית הספרייה הסטנדרטית, באמצעותה מבוצע הקלט (בקלט מהמקלדת, ניתן להכניס EOF על ידי הקלדת ctrl-d בתחילת שורה חדשה).

קלט לדוגמה:

13 13 13 45 -9 -9 -9 18

18 18 3 4 55 45 66 13 66

עבור קלט זה, הפלט יהיה:

13 45 -9 18 3 4 55 66

עליכם לכתוב שתי **פונקציות**:

get_set מבצעת את הקלט מהמשתמש ובונה את הקבוצה.

print_set מדפיסה את איברי הקבוצה בסדר הנדרש ובאופן נאה.

עליכם לקבוע מהם הפרמטרים והערך המוחזר של כל פונקציה, ולתעד זאת בהערת כותרת של הפונקציה. אין להשתמש במשתנים גלובליים.

עליכם לכתוב **מקרו** בשם **ENLARGE_SIZE** המוסיף מספר קבוע למשתנה, וישמש עבור הפונקציה realloc לשם הגדלת גודל המבנה המכיל את הקבוצה.

כמו כן, יש לכתוב תכנית ראשית (הפונקציה main) שתקרא ל-get_set ולאחר מכן ל-print_set.

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 כל הנתונים כפי שנקלטו.** באופן זה, הקלט יוצג גם כאשר הוא מגיע מקובץ.

חובה לצרף להגשה מספר <u>הרצות בדיקה</u> המדגימות את פעולת התכנית על קלטים בגדלים ובערכים מגוונים. **יש להגיש תדפיסי מסך וקבצי פלט) של כל הרצות הדוגמה**. כמו כן, יש להגיש קבצי קלט,

<u>לתשומת לבכם:</u> לא תינתן דחיה בהגשת הממ״ן, פרט למקרים מיוחדים כגון מילואים או אשפוז. במקרים אלו יש לבקש ולקבל אישור **מראש** ממנחה הקבוצה.

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: 2023ב' **מועד אחרון להגשה:** 21.05.2023

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), קובץ ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קובץ makefile), וכן קבצי קלט ותדפיסי מסך וקבצי פלט.

קבצי התוכנית יהיו בתיקיה. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת c.

יש להגיש תכנית מלאה (בין השאר מכילה main), הניתנת להידור והרצה, ומאפשרת בדיקה של כל התוצאות המגוונות של הריצה ללא צורך בשינויים כלשהם בקוד המקור של התוכנית.

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

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

עליכם לכתוב תוכנית מחשב אינטראקטיבית הקוראת פקודות מהקלט הסטנדרטי, מפענחת ומבצעת אותן. הפקודות עוסקות בפעולות אריתמטיות על מספרים מרוכבים.

<u>תזכורת מספרים מרוכבים:</u>

מספר מרוכב (complex) הוא מספר בן שני חלקים: חלק ממשי וחלק מדומה, כאשר ביניהם רשום הסימן "+" או הסימן "–".

מבנה המספר הוא: a + bi כאשר a החלק הממשי ו- bi החלק המדומה. החלק המדומה הוא מכפלה של שני גורמים: b הוא מספר ממשי, ואילו i מציין את השורש הריבועי של המספר 1-, כלומר $i = \sqrt{-1}$.

דוגמאות של מספרים מרוכבים:

$24.65 + (15.376)i$ $\quad$ $87.5 - (14.3)i$ $\quad$ $-153+24i$

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

חיבור של שני מספרים מרוכבים:

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

חיסור של שני מספרים מרוכבים:

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

כפל של מספר מרוכב במספר ממשי:

$m * (a + bi) = ma + (mb)i$

כפל של מספר מרוכב במספר מדומה:

$mi * (a + bi) = mi * a + mi * bi = -mb + (ma)i$

כפל של מספר מרוכב במספר מרוכב:

$(a + bi) * (c + di) = a * c + a * di + bi * c + bi * di = (ac - bd) + (ad + bc)i$

חישוב הערך המוחלט של מספר מרוכב (התוצאה היא מספר ממשי אי-שלילי):

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

<u>משימות התוכנית:</u>

עליכם להגדיר, באמצעות שימוש ב- typedef את הטיפוס complex אשר מחזיק מספר מרוכב.
על מבנה הנתונים שבחרתם להיות יעיל מבחינת כמות זיכרון הנדרשת, ויעיל מבחינת הגישה אליו.

בנוסף, עליכם להגדיר בתוכנית הראשית 6 משתנים: A,B,C,D,E,F מטיפוס complex.
בתחילת הריצה, יש לאתחל את כל ששת המשתנים לאפס (הערך המרוכב 0+0i).

כעת, עליכם לבצע פעולות חשבוניות עם מספרים מרוכבים. כל פעולה תופעל באמצעות פקודה שמועברת בקלט לתוכנית, כמפורט להלן. בפקודות אלה, אופרנד שהוא משתנה מרוכב יהיה אחד מששת המשתנים שהוגדרו לעיל.

מפרט הפקודות המשמשות כקלטים לתוכנית:

1. הצבת מספר מרוכב במשתנה:

מספר-ממשי-שני ,מספר-ממשי-ראשון ,שם-משתנה-מרוכב read_comp

הפקודה תגרום להצבת ערך מרוכב במשתנה המרוכב ששמו מופיע בפקודה. המספר הממשי הראשון הוא החלק הממשי של המספר המרוכב, והמספר הממשי השני הוא החלק המדומה של המספר המרוכב (החלק המדומה נתון בפקודה ללא הגורם i)

לדוגמה, הפקודה הבאה:

read_comp A, 5.1, 6.23

תבצע את ההצבה:

A = 5.1 + (6.23)i

הערה: זוהי הפקודה היחידה שמשנה את ערכו של משתנה מרוכב בתוכנית.

2. הדפסת ערך של משתנה מרוכב:

שם-משתנה-מרוכב print_comp

ערכו של המשתנה המרוכב ששמו ניתן בפקודה יודפס בצורה נאה בפלט.

לדוגמה, הפקודה הבאה:

print_comp A

תגרום להדפסת ערך המשתנה A. בהנחה שהפקודה היא בהמשך לדוגמה בסעיף 1, יודפס:

5.10 + (6.23)i

הערה: יש להדפיס כל מספר עם דיוק של לפחות שתי ספרות מימין לנקודה.

3. חיבור מספרים מרוכבים:

שם-משתנה-מרוכב-ב' ,שם-משתנה-מרוכב-א' add_comp

יתבצע חיבור של שני המספרים המרוכבים אשר במשתנים המופיעים בפקודה:

מספר-מרוכב-ב' + מספר-מרוכב-א'

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

4. חיסור מספרים מרוכבים:

שם-משתנה-מרוכב-ב' ,שם-משתנה-מרוכב-א' sub_comp

יתבצע חיסור של המספר המרוכב במשתנה ב' מן המספר המרוכב במשתנה א':

מספר-מרוכב-ב' – מספר-מרוכב-א'

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

5. כפל מספר מרוכב עם מספר ממשי:

מספר-ממשי ,שם-משתנה-מרוכב mult_comp_real

יתבצע כפל של המשתנה המרוכב והמספר הממשי הנתונים בפקודה.

מספר-ממשי * מספר-מרוכב

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

6. כפל מספר מרוכב עם מספר מדומה:

מספר-מדומה ,שם-משתנה-מרוכב mult_comp_img

יתבצע כפל של המשתנה המרוכב והמספר המדומה הנתונים בפקודה.

המספר המדומה נתון בפקודה ללא הגורם i.

(i * מספר-מדומה) * מספר-מרוכב

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

7. כפל שני מספרים מרוכבים:

שם-משתנה-מרוכב-ב' ,שם-משתנה-מרוכב-א' mult_comp_comp

יתבצע כפל של שני המשתנים המרוכבים המופיעים בפקודה:

מספר-מרוכב-ב' * מספר-מרוכב-א'

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

8. ערך מוחלט של מספר מרוכב:

שם-משתנה-מרוכב abs_comp

יחושב ערכו המוחלט של המשתנה המרוכב שמופיע בפקודה:

| מספר-מרוכב |

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

9. סיום התוכנית:

stop

פקודה זו היא ללא פרמטרים, ומטרתה לסיים את התוכנית.

<u>המבנה התחבירי של הקלט</u>:

- 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 המשתנים באותיות גדולות בלבד.

אופן פעולת התוכנית:

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

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

יש לטפל גם במצב של EOF (גמר הקלט). עצירת התוכנית שלא באמצעות פקודת stop אינה נחשבת תקינה (גם לא כאשר הקלט מגיע מקובץ באמצעות redirection), ויש להדפיס הודעת שגיאה על כך ורק אז לעצור. שימו לב: השורה האחרונה בקובץ קלט אינה חייבת להסתיים בתו '\n'. במקרה כזה, אם יש בשורה האחרונה פקודה, יש לטפל בה כרגיל (סוף הקובץ מסמן את סוף הפקודה).

להלן דוגמאות של קלט שגוי:

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

1. לפקודה:

read_comp G, 3.1, 6.5

יש להגיב בהודעה כגון:

Undefined complex variable

2. לפקודה:

read_comp a, 3.6, 5.1

יש להגיב בהודעה כגון:

Undefined complex variable

3. לפקודה:

do_it A, B

יש להגיב בהודעה כגון:

Undefined command name

4. לפקודה:

Add_Comp A, C

יש להגיב בהודעה כגון:

Undefined command name

5. לפקודה:

read_comp A, 3.5, xyz

יש להגיב בהודעה כגון:

Invalid parameter – not a number

6. לפקודה:

read_comp A, 3.5

יש להגיב בהודעה כגון:

Missing parameter

7. לפקודה:

read_comp A, 3.5, -3,

יש להגיב בהודעה כגון:

Extraneous text after end of command

8. לפקודה:

add_comp B

יש להגיב בהודעה כגון:

Missing parameter

9. לפקודה:

print_comp C, D

יש להגיב בהודעה כגון:

Extraneous text after end of command

10. לפקודה:

sub_comp F, , D

יש להגיב בהודעה כגון:

Multiple consecutive commas

11. לפקודה:

mult_comp_comp F D

יש להגיב בהודעה כגון:

Missing comma

12. לפקודה:

mult_comp_real, A, 2.5

יש להגיב בהודעה כגון:

Illegal comma

13. לפקודה:

mult_comp_img A, B

יש להגיב בהודעה כגון:

Invalid parameter – not a number

14. לפקודה:

abs_comp

יש להגיב בהודעה כגון:

Missing parameter

15. לפקודה:

abs_comp 2.5

יש להגיב בהודעה כגון:

Undefined complex variable

16. לפקודה:

stop A

יש להגיב בהודעה כגון:

Extraneous text after end of command

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

הערה: סדרה כגון זו יכולה לשמש כקלט בהרצת בדיקה של התוכנית על קלט תקין.

```
print_comp A
print_comp B
print_comp C
read_comp A, 45.1, -23.75
print_comp A
read_comp B, 54.2, 3.56
print_comp B
read_comp C, 0, -1
print_comp C
add_comp A, B
sub_comp C, A
sub_comp B, B
sub_comp D, A
```

```
mult_comp_real A, 2.51
mult_comp_img A, -2.564
mult_comp_comp A, B
mult_comp_comp    E   ,    C
abs_comp A
abs_comp B
abs_comp C
abs_comp F
stop
```

<u>דרישות והנחיות נוספות</u>:

- יש לחלק את התוכנית למספר קבצי מקור: mycomp.c, complex.c, ו-complex.h.
  - בקובץ mycomp.c תהיה התוכנית הראשית main, וכן כל פעילויות האינטראקציה עם המשתמש וניתוח הקלט (לרבות הדפסת הודעות השגיאה). כמו כן, יוגדרו בקובץ זה ששת המשתנים מטיפוס complex.
  - בקובץ complex.c יש לרכז את הפעולות החשבוניות על מספרים מרוכבים. לכל פעולה יש לממש פונקציה נפרדת, עם פרמטרים לפי מפרט הפעולה המוגדר לעיל. אין לבצע ניתוח של הקלט או הדפסות מתוך קובץ זה, למעט הדפסת המספר המרוכב כנדרש בפעולה print_comp.
  - בקובץ complex.h תהיה הגדרת טיפוס הנתונים complex, וכן ההצהרות (אב-טיפוס) של הפונקציות הממומשות בקובץ complex.c. יש לכלול (#include) את הקובץ complex.h בקבצי המקור האחרים.
  - באפשרותכם לבנות קבצי מקור נוספים (למשל: קובץ המכיל פונקציות עזר לניתוח הקלט, וכד').
- הקלט לתוכנית הוא מ-stdin, ויכול להגיע **מהמקלדת או מקובץ** (באמצעות redirection בעת הפעלת התוכנית). לנוחיותכם, הכינו מספר קבצי קלט והשתמשו בהם שוב ושוב לדיבוג התוכנית. בכל קובץ קלט תהיה סדרה של פקודות על מספרים מרוכבים.
- לפני הניתוח של כל שורת קלט, על התוכנית **להדפיס באופן יזום 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 ובכל ששת המשתנים המרוכבים, וכן את הטיפול בכל מגוון השגיאות בקלט.
יש להגיש קובץ קלט + תדפיס מסך (וקובץ פלט) של כל הרצה.

<u>לתשומת לבכם:</u> לא תינתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או אשפוז. במקרים אלו יש לבקש ולקבל אישור מראש ממנחה הקבוצה.

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: 2023ב' **מועד אחרון להגשה:** 04.06.2023

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), קובץ ההרצה (את קבצי o. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קובץ makefile), וכן קבצי קלט ותדפיסי מסך וקבצי פלט. קבצי התכנית יהיו בתיקיה. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת c.

יש להגיש תכנית מלאה (בין השאר מכילה main), הניתנת להידור והרצה, ומאפשרת בדיקה של כל התוצאות המגוונות של הריצה ללא צורך בשינויים כלשהם בקוד המקור של התוכנית.

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

שאלה 1 (10 נקודות)

עליכם להסביר מה מבצע קטע הקוד הבא, כאשר A ו- B הם משתנים מסוג int:

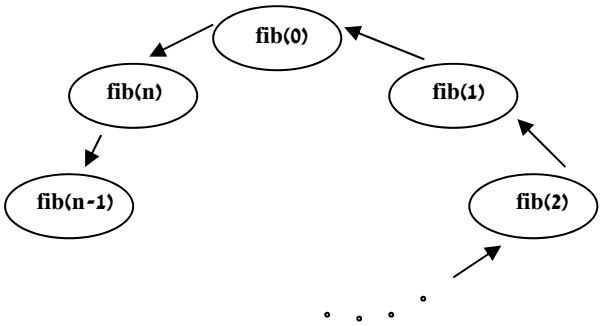
```
A ^= B;
B ^= A;
A ^= B;
```

יש לנמק בדייקנות. השתמשו גם בדוגמה מספרית, לצורך הבהרה.

את הפתרון לשאלה זו יש להגיש במסמך (קובץ) מוקלד, בפורמט word או pdf.

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

עליכם כתוב תכנית המקבלת בשורת הפקודה שם קובץ. כמו כן מקבלת מהמשתמש, באופן אינטראקטיבי (ולא משורת הפקודה) מספר שלם **n**, ויוצרת סדרת פיבונאצ'י ברשימה המעגלית הבאה:



לאחר יצירת הרשימה המעגלית, על התוכנית להדפיס את איברי הסדרה בסדר יורד, וכן לשמור את האיברים בקובץ ששמו התקבל בקריאה לתכנית. בנוסף לאיברים, יש להוסיף לקובץ הנ"ל כותרת קצרה וברורה, המתארת במילים את מטרת התכנית ואת המספר **n**.

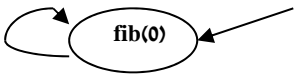
לתזכורת: סדרת פיבונאצ'י מוגדרת כך:

$$fib(0) = fib(1) = 1$$

$$n>1 \text{ ל- } fib(n) = fib(n-1) + fib(n-2)$$

הבהרות:

א. עבור n=0 יתקבל :



ב. עבור קלט של מספר שלילי תצא הודעה מתאימה ולא יתבצע דבר.

הקלט לתכנית הוא מ-**stdin**. על התכנית **להדפיס הודעת בקשה ידידותית לקלט n**.
יש להדפיס **הודעות שגיאה** : מספר הארגומנטים לא תקין, קובץ לא נפתח, הקצאת זיכרון נכשלה.

חובה לצרף להגשה מספר הרצות בדיקה המדגימות את פעולת התכנית על קלטים בגדלים ובערכים מגוונים. **יש להגיש קבצי קלט ותדפיסי מסך של כל הרצות הדוגמה ואת הקבצים שנוצרו בדוגמאות.**

לתשומת לבכם: לא תינתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או אשפוז. במקרים אלו יש לבקש ולקבל אישור מראש ממנחה הקבוצה.

בהצלחה!

מטלת מנחה (ממ"ן) 14

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרויקט גמר

מספר השאלות: 1 | משקל המטלה: 61 נקודות **(חובה)**

סמסטר: 2023ב | מועד אחרון להגשה: 11.08.2023

קיימות שתי חלופות להגשת מטלות:

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

הסבר מפורט ב"נוהל הגשת מטלות מנחה"

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

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

עליכם להגיש את הפריטים הבאים:

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

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

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

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

נזכיר מספר היבטים חשובים של כתיבת קוד טוב:

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

2. קריאות הקוד: יש להשתמש בשמות משמעותיים למשתנים ופונקציות. יש לערוך את הקוד באופן מסודר: הזחות עקביות, שורות ריקות להפרדה בין קטעי קוד, וכד'.

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

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

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

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

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

רקע כללי ומטרת הפרויקט

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

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

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

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

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

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

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

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

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

המחשב הדמיוני ושפת האסמבלי

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

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

"חומרה":

המחשב בפרויקט מורכב ממעבד CPU (יע"מ - יחידת עיבוד מרכזית), אוגרים (רגיסטרים) וזיכרון RAM. חלק מהזיכרון משמש גם כמחסנית (stack). גודלה של מילת זיכרון במחשב הוא 12 סיביות.

למעבד 8 רגיסטרים כלליים, בשמות: r0,r1,r2,r3,r4,r5,r6,r7 .
גודלו של כל רגיסטר הוא 12 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 11. שמות הרגיסטרים נכתבים תמיד עם אות 'r' קטנה.

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

גודל הזיכרון הוא 1024 תאים, בכתובות 0-1023 (בבסיס עשרוני), וכל תא הוא בגודל של 12 סיביות. לתא בזיכרון נקרא גם בשם "מילה". הסיביות בכל מילה ממוספרות כמו ברגיסטר.

מחשב זה עובד רק עם מספרים שלמים חיוביים ושליליים, אין תמיכה במספרים ממשיים.
האריתמטיקה נעשית בשיטת המשלים ל-2 (2's complement).
כמו כן יש תמיכה בתווים (characters), המיוצגים בקוד ascii.

מבנה הוראת המכונה:

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

11 9	8 5	4 2	1 0
מיעון אופרנד מקור	opcode	מיעון אופרנד יעד	A,R,E

סיביות 5-8 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:

קוד הפעולה בבסיס דצימלי (10)	שם הפעולה
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

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

סיביות 0-1 (A,R,E)

סיביות אלה מציינות 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.

סיביות 2-4 מקודדות את מספרה של שיטת המיעון של אופרנד היעד (destination operand).

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

בשפה שלנו קיימות שלוש שיטות מיעון, שמסומנות במספרים 1, 3, 5.

השימוש בשיטות מיעון מצריך קידוד של מילות-מידע נוספות בקוד המכונה של ההוראה.
אם בפקודה יש אופרנד אחד, תהיה מילת מידע אחת נוספת. אם בפקודה יש שני אופרנדים, ייתכנו שתי מילות-מידע נוספות, או מילת-מידע אחת המשותפת לשני האופרנדים, תלוי בשיטות המיעון בהן נעשה שימוש (ראו מפרט בהמשך). כאשר בקידוד הפקודה יש שתי מילות-מידע נוספות, אזי מילת-המידע הראשונה מתייחסת לאופרנד המקור, והשניה מתייחסת לאופרנד היעד.

בכל מילת-מידע נוספת, סיביות 0-1 הן השדה A,R,E.

להלן תיאור של שיטות המיעון במכונה שלנו:

קוד	שיטת מיעון	תוכן המילים הנוספות	אופן הכתיבה	דוגמה
1	מיעון מידי	מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר המיוצג ב- 10 סיביות, אליהן מתווספות זוג סיביות של השדה A,R,E.	האופרנד הוא מספר שלם בבסיס עשרוני	mov -103,@r2 בדוגמה זו האופרנד הראשון של הפקודה נתון בשיטת מיעון מיידי. ההוראה כותבת את הערך 103- אל אוגר r2
3	מיעון ישיר	מילת-מידע נוספת של ההוראה מכילה כתובת של מילה בזיכרון. מילה זו בזיכרון היא האופרנד. המען מיוצג ב- 10 סיביות אליהן מתווספות זוג סיביות של השדה A,R,E.	האופרנד הוא <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).
5	מיעון אוגר ישיר	האופרנד הוא אוגר. אם האוגר משמש כאופרנד יעד, מילת-מידע נוספת של הפקודה תקודד בחמש הסיביות 2-6 את מספרו של האוגר. ואילו אם האוגר משמש כאופרנד מקור, מספר האוגר יקודד בחמש הסיביות 7-11 של מילת-המידע. אם בפקודה יש שני אופרנדים, ושניהם אוגרים, הם יחלקו מילת-מידע אחת משותפת, כאשר הסיביות 2-6 הן עבור אוגר היעד, והסיביות 7-11 עבור אוגר המקור. בכל מילה נוספת מתווספות זוג סיביות של השדה A,R,E. סיביות שאינן בשימוש יכילו 0.	האופרנד הוא שם של אוגר. שם האוגר ייכתב כאשר לפניו יופיע התו @	mov @r1,@r2 בדוגמה זו, ההוראה מעתיקה את תוכן אוגר r1 אל אוגר r2. בדוגמה זו שני האופרנדים הם אוגרים, אשר יקודדו למילת-מידע נוספת אחת משותפת.

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

<u>מפרט הוראות המכונה:</u>

בתיאור הוראות המכונה נשתמש במונח **PC** (קיצור של "**Program Counter**").
זהו רגיסטר פנימי של המעבד (<u>לא</u> רגיסטר כללי), שמכיל בכל רגע נתון את כתובת הזיכרון בה נמצאת **ההוראה הנוכחית שמתבצעת** (הכוונה תמיד לכתובת המילה הראשונה של ההוראה).

הוראות המכונה מתחלקות לשלוש קבוצות, לפי מספר האופרנדים הנדרשים לפעולה.

קבוצת ההוראות הראשונה:
אלו הן הוראות המקבלות שני אופרנדים.

ההוראות השייכות לקבוצה זו הן: mov, cmp, add, sub, lea

הוראה	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
mov	מבצעת העתקה של האופרנד הראשון, אופרנד המקור (source) אל האופרנד השני, אופרנד היעד (destination) (בהתאם לשיטת המיעון).	mov A,@r1	העתק את תוכן המשתנה A (המילה שבכתובת A בזכרון) אל רגיסטר 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") ברגיסטר הסטטוס(PSW).	cmp A, @r1	אם תוכן המשתנה A זהה לתוכנו של רגיסטר r1 אזי דגל האפס, Z, ברגיסטר הסטטוס (PSW) יודלק, אחרת הדגל יאופס.
add	אופרנד היעד (השני) מקבל את תוצאת החיבור של אופרנד המקור (הראשון) והיעד (השני).	add A,@r0	רגיסטר r0 מקבל את תוצאת החיבור של תוכן המשתנה A ותוכנו הנוכחי של רגיסטר r0.
sub	אופרנד היעד (השני) מקבל את תוצאת החיסור של אופרנד המקור (הראשון) מאופרנד היעד (השני)	sub 3,@r1	רגיסטר r1 מקבל את תוצאת החיסור של הערך 3 מתוכנו הנוכחי של הרגיסטר r1.
lea	lea הוא קיצור (ראשי תיבות) של: load effective address. פעולה זו מציבה את המען בזיכרון המיוצג על ידי התווית שבאופרנד הראשון (המקור), אל אופרנד היעד (האופרנד השני).	lea HELLO, @r1	המען שמייצגת התווית HELLO מוצב לרגיסטר r1.

קבוצת ההוראות השניה:

הוראות הדורשות אופרנד אחד בלבד. במקרה זה זוג הסיביות 9-11 במילה הראשונה של קידוד ההוראה הן חסרות משמעות, מכיוון שאין אופרנד מקור (אופרנד ראשון) אלא רק אופרנד יעד (שני). לפיכך הסיביות 9-11 יכילו תמיד 0.
ההוראות השייכות לקבוצה זו הן:
not, clr, inc, dec, jmp, bne, red, prn, jsr

הוראה	הפעולה המתבצעת	דוגמה	הסבר דוגמה
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

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

קבוצת ההוראות השלישית:

אלו הן הוראות ללא אופרנדים. קידוד ההוראה מורכב ממילה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד במילה הראשונה של ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השייכות לקבוצה זו הן : rts, stop.

הוראה	הפעולה המתבצעת	דוגמה	הסבר דוגמה
rts	חזרה משיגרה. הערך בראש המחסנית של זמן-ריצה מוצא מן המחסנית ומוכנס אל מצביע התוכנית (PC).	rts	PC ← pop()
stop	עצירת ריצת התוכנית.	stop	התכנית עוצרת

<u>מבנה תכנית בשפת אסמבלי :</u>

תכנית בשפת אסמבלי בנויה **<u>ממקרואים</u>** **<u>וממשפטים</u>** (statements).

מקרואים:

מקרואים הם קטעי קוד הכוללים בתוכם משפטים. בתוכנית ניתן להגדיר מקרו ולהשתמש בו במקומות שונים בתוכנית. השימוש במקרו ממקום מסוים בתוכנית יגרום לפרישת המקרו לאותו מקום.

הגדרת מקרו נעשית באופן הבא: (בדוגמה שם המקרו הוא m1)

```
mcro  m1
  inc r2
  mov  A,r1
endmcro
```

שימוש במקרו הוא פשוט אזכור שמו.
למשל, אם בתוכנית במקום כלשהו כתוב:

```
.
.
.
m1
.
.
m1
.
.
.
```

בדוגמה זו, השתמשנו פעמיים במקרו m1, התוכנית לאחר פרישת המקרו תיראה כך:

```
.
.
.
  inc r2
  mov  A,r1
.
.
  inc r2
  mov  A,r1
.
.
.
```

התוכנית לאחר פרישת המקרו היא התוכנית שהאסמבלר אמור לתרגם.

הנחיות לגבי מאקרו:

- אין במערכת הגדרות מאקרו מקוננות.
- שם של הוראה או הנחיה לא יכול להיות שם של מאקרו.
- ניתן להניח שלכל שורת מאקרו בקוד המקור קיימת סגירה עם שורת endmcro (אין צורך לבדוק זאת).
- הגדרת מאקרו תהיה תמיד לפני הקריאה למאקרו
- נדרש שהקדם-אסמבלר ייצור קובץ עם הקוד המורחב הכולל פרישה של המאקרו (הרחבה של קובץ המקור המתואר בהמשך). "קובץ המקור המורחב" הוא "קובץ מקור" לאחר פרישת המאקרו, לעומת "קובץ מקור ראשוני" שהוא קובץ הקלט למערכת, כולל הגדרת המאקרואים.

<u>משפטים</u>:

קובץ מקור בשפת אסמבלי מורכב משורות המכילות משפטים של השפה, כאשר כל משפט מופיע <u>בשורה נפרדת</u>. כלומר, ההפרדה בין משפט למשפט בקובץ המקור הינה באמצעות התו '\n' (שורה חדשה).

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

יש ארבעה סוגי משפטים (שורות בקובץ המקור) בשפת אסמבלי, והם:

סוג המשפט	הסבר כללי
משפט ריק	זוהי שורה המכילה אך ורק תווים לבנים (whitespace), כלומר מכילה רק את התווים '\t' ו-' ' (טאבים ורווחים). ייתכן ובשורה אין אף תו (למעט התו \n), כלומר השורה ריקה.
משפט הערה	זוהי שורה בה התו הראשון הוא ';' (נקודה פסיק). על האסמבלר להתעלם לחלוטין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמבלר מה עליו לעשות כשהוא פועל על תכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצאת זיכרון ואתחול משתנים של התכנית, אך הוא אינו מייצר קידוד של הוראות מכונה המיועדות לביצוע בעת ריצת התוכנית.
משפט הוראה	זהו משפט המייצר קידוד של הוראות מכונה לביצוע בעת ריצת התוכנית. המשפט מורכב משם ההוראה (פעולה) שעל המעבד לבצע, והאופרנדים של ההוראה.

כעת נפרט לגבי סוגי המשפטים השונים.

<u>משפט הנחיה:</u>

משפט הנחיה הוא בעל המבנה הבא:

בתחילת המשפט יכולה להופיע הגדרה של תווית (label). לתווית יש תחביר חוקי, שיתואר בהמשך. התווית היא אופציונלית.

לאחר מכן מופיע שם ההנחיה. לאחר שם ההנחיה יופיעו פרמטרים (מספר הפרמטרים בהתאם להנחיה). שם של הנחיה מתחיל בתו '.' (נקודה) ולאחריו תווים באותיות קטנות (lower case) בלבד.

יש לשים לב: למילות הקוד הנוצרות ממשפט הנחיה לא מצורפות זוג סיביות A,R,E והקידוד ממלא את כל 12 הסיביות של המילה.

יש ארבעה סוגים של משפטי הנחיה , והם:

1. ההנחיה '.data'

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

.data 7, −57, +17, 9

יש לשים לב שהפסיקים אינם חייבים להיות צמודים למספרים. בין מספר לפסיק ובין פסיק למספר יכולים להופיע רווחים וטאבים בכל כמות (או בכלל לא), אולם הפסיק חייב להופיע בין

המספרים. כמו כן, אסור שיופיע יותר מפסיק אחד בין שני מספרים, וגם לא פסיק אחרי המספר האחרון או לפני המספר הראשון.

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

כלומר אם נכתוב:

XYZ: .data 7, −57, +17, 9

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

אם נכתוב בתכנית את ההוראה:

mov XYZ, @r1

אזי בזמן ריצת התכנית יוכנס לרגיסטר r1 הערך 7.

ואילו ההוראה:

lea XYZ,@r1

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

2. ההנחיה 'string.'

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

לדוגמה, ההנחיה:

STR: .string "abcdef"

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

3. ההנחיה 'entry.'

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

לדוגמה, השורות:

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

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

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

4. ההנחיה 'extern.'

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

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

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

```
.extern   HELLO
```

<u>הערה</u>: לא ניתן להגדיר באותו הקובץ את אותה התווית גם כ-entry וגם כ-extern (בדוגמאות לעיל, התווית HELLO).

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

<u>משפט הוראה</u>:

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

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

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

לאחר שם הפעולה יופיעו האופרנדים, בהתאם לסוג הפעולה. יש להפריד בין שם-הפעולה לבין האופרנד הראשון באמצעות רווחים ו/או טאבים (אחד או יותר).

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזה בתו ',' (פסיק). בדומה להנחיה 'data.', **לא חייבת להיות הצמדה של האופרנדים לפסיק**. כל כמות של רווחים ו/או טאבים משני צידי הפסיק היא חוקית.

למשפט הוראה עם שני אופרנדים המבנה הבא:

```
label:  opcode  source-operand, target-operand
```

לדוגמה:

HELLO: add @r7, B

למשפט הוראה עם אופרנד אחד המבנה הבא:

label: opcode target-operand

לדוגמה:

HELLO: bne XYZ

למשפט הוראה ללא אופרנדים המבנה הבא:

label: opcode

לדוגמה:

END: stop

<u>אפיון השדות במשפטים של שפת האסמבלי</u>

<u>תווית:</u>

תווית היא סמל שמוגדר בתחילת משפט הוראה , או בתחילת הנחיית data. או string.
תווית חוקית מתחילה באות אלפביתית (גדולה או קטנה), ולאחריה סדרה כלשהי של אותיות אלפביתיות (גדולות או קטנות) ו/או ספרות. האורך המקסימלי של תווית הוא 31 תווים.

<u>הגדרה של תווית</u> מסתיימת בתו ':' (נקודתיים). תו זה אינו מהווה חלק מהתווית, אלא רק סימן המציין את סוף ההגדרה. התו ':' חייב להיות צמוד לתווית (ללא רווחים).

אסור שאותה תווית תוגדר יותר מפעם אחת (כמובן בשורות שונות). אותיות קטנות וגדולות נחשבות <u>שונות זו מזו</u>.

לדוגמה, התוויות המוגדרות להלן 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 גם כשם של תווית.

התווית מקבלת את ערכה בהתאם להקשר בו היא מוגדרת. 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.

לתשומת לב: מותר במשפט הוראה להשתמש באופרנד שהוא סמל שאינו מוגדר כתווית בקובץ הנוכחי, כל עוד הסמל מאופיין כחיצוני (באמצעות הנחיית extern. כלשהי בקובץ הנוכחי).

<u>מספר:</u>

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

מחרוזת:

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

סימון המילים בקוד המכונה באמצעות המאפיין "A,R,E"

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

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

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

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

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

כאשר האסמבלר מקבל כקלט תוכנית בשפת אסמבלי, עליו לטפל תחילה בפרישת המאקרואים, ורק לאחר מכן לעבור על התוכנית אליה נפרשו המקרואים. כלומר, פרישת המקרואים תעשה בשלב "קדם אסמבלר", לפני שלב האסמבלר (המתואר בהמשך) .
אם התכנית אינה מכילה מקרו, תוכנית הפרישה תהיה זהה לתכנית המקור.

דוגמה לשלב קדם אסמבלר. האסמבלר מקבל את התוכנית הבאה בשפת אסמבלי:

```
MAIN:     mov     @r3 ,LENGTH
LOOP:     jmp     L1
          mcro  m1
                sub     @r1, @r4
                bne     END
          endmcro
          prn     -5
          bne     LOOP
          m1
L1:       inc     K
          bne     LOOP
END:      stop
STR:      .string "abcdef"
LENGTH:   .data   6,-9,15
K:        .data   22
```

תחילה האסמבלר עובר על התוכנית ופורש את כל המאקרואים הקיימים בה. רק אם תהליך זה מסתיים בהצלחה, ניתן לעבור לשלב הבא. בדוגמה זו, התוכנית לאחר פרישת המאקרו תיראה כך:

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

קוד התכנית, לאחר הפרישה, ישמר בקובץ חדש, כפי שיוסבר בהמשך.

אלגוריתם שלדי של קדם האסמבלר

נציג להלן אלגוריתם שלדי לתהליך קדם האסמבלר. <u>לתשומת לב</u>: אין חובה להשתמש דווקא באלגוריתם זה:

1. קרא את השורה הבאה מקובץ המקור. אם נגמר הקובץ, עבור ל- 9 (סיום).
2. האם השדה הראשון הוא שם מאקרו המופיע בטבלת המאקרו (כגון m1)? אם כן, החלף את שם המאקרו והעתק במקומו את כל השורות המתאימות מהטבלה לקובץ, חזור ל 1. אחרת, המשך.
3. האם השדה הראשון הוא " **mcro** " (התחלת הגדרת מאקרו)? אם לא, עבור ל- 6.
4. הדלק דגל "יש **mcro** ".
5. (קיימת הגדרת מאקרו) הכנס לטבלת שורות מאקרו את שם המאקרו (לדוגמה m1).
6. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 9 (סיום).
   אם דגל "יש **mcro** " דולק ולא זוהתה תווית **endmcro** הכנס את השורה לטבלת המאקרו ומחק את השורה הנ"ל מהקובץ. אחרת (לא מאקרו) חזור ל 1.
7. האם זוהתה תווית **endmcro**? אם כן, מחק את התווית מהקובץ והמשך. אם לא, חזור ל- 6.
8. כבה דגל "יש **mcro** ". חזור ל- 1. (סיום שמירת הגדרת מאקרו).
9. סיום: שמירת קובץ מקרו פרוש.

אסמבלר עם שני מעברים

במעבר הראשון של האסמבלר, יש לזהות את הסמלים (תוויות) המופיעים בתוכנית, ולתת לכל סמל ערך מספרי שהוא המען בזיכרון שהסמל מייצג. במעבר השני, באמצעות ערכי הסמלים, וכן קודי-הפעולה ומספרי האוגרים, בונים את קוד המכונה.

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, MAIN, LOOP, END במענים של המקומות בזיכרון שם נמצאים כל נתון או הוראה בהתאמה.

נניח שקטע הקוד לעיל (הוראות ונתונים) ייטען בזיכרון החל ממען 100 (בבסיס 10) . במקרה זה נקבל את ה"תרגום" הבא:

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102	MAIN: mov @r3 ,LENGTH	first word of instruction source register 3 address of label LENGTH	**101000001100** **000110000000** **000111110110**
0103 0104	LOOP: jmp L1	 address of label L1	**000100101100** **000111000110**
0105 0106	prn -5	 immediate value -5	**000110000100** **111111101100**
0107 0108	bne LOOP	 address of label LOOP	**000101001100** **000110011110**
0109 0110	sub @r1, @r4	 registers r1 and r4	**101001110100** **000010010000**
0111 0112	bne END	 address of label END	**000101001100** **000111010110**
0113 0114	L1: inc K	 address of label K	**000011101100** **001000000010**
0115 0116	bne LOOP	 address of label LOOP	**000101001100** **000110011110**
0117	END: stop		**000111100000**
0118 0119 0120 0121 0122 0123 0124	STR: .string "abcdef"	ascii code 'a' ascii code 'b' ascii code 'c' ascii code 'd' ascii code 'e' ascii code 'f' ascii code '\0' (end of string)	**000001100001** **000001100010** **000001100011** **000001100100** **000001100101** **000001100110** **000000000000**
0125 0126 0127	LENGTH: .data 6,-9,15	integer 6 integer -9 integer 15	**000000000110** **111111110111** **000000001111**
0128	K: .data 22	integer 22	**000000010110**

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 הקידוד הבינארי.

כדי לבצע המרה לבינארי של אופרנדים שכתובים בשיטות מיעון המשתמשות בסמלים (תוויות), יש צורך לבנות טבלה המכילה את ערכי כל הסמלים. אולם בהבדל מהקודים של הפעולות, הידועים מראש, הרי המענים בזיכרון עבור הסמלים שבשימוש התוכנית אינם ידועים, עד אשר תוכנית המקור נסרקה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד לעיל, האסמבלר אינו יכול לדעת שהסמל END משויך למען 117 (עשרוני), אלא רק לאחר שנקראו כל שורות התוכנית.

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

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

סמל	ערך (בבסיס דצימלי)
MAIN	100
LOOP	103
L1	113
END	117

סמל	ערך (בבסיס דצימלי)
STR	118
LENGTH	125
K	128

במעבר השני נעשית ההמרה של קוד המקור לקוד מכונה. בתחילת המעבר השני צריכים הערכים של כל הסמלים להיות כבר ידועים.

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

המעבר הראשון

במעבר הראשון נדרשים כללים כדי לקבוע איזה מען ישויך לכל סמל. העיקרון הבסיסי הוא לספור את המקומות בזיכרון, אותם תופסות ההוראות. אם כל הוראה תיטען בזיכרון למקום העוקב להוראה הקודמת, תציין ספירה כזאת את מען ההוראה הבאה. הספירה נעשית על ידי האסמבלר ומוחזקת במונה ההוראות (IC) . ערכו ההתחלתי של IC הוא 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 קידוד מתאים לכל שם פעולה. בזמן התרגום מחליף האסמבלר כל שם פעולה בקידוד שלה. כמו כן, כל אופרנד מוחלף בקידוד מתאים, אך פעולת החלפה זו אינה כה פשוטה. ההוראות משתמשות בשיטות מיעון מגוונות לאופרנדים. אותה פעולה יכולה לקבל משמעויות שונות, בכל אחת משיטות המיעון, ולכן יתאימו לה קידודים שונים לפי שיטות המיעון. לדוגמה, פעולת ההזזה mov יכולה להתייחס להעתקת תוכן תא זיכרון לרגיסטר, או להעתקת תוכן רגיסטר לרגיסטר 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. כל השדות ביחד דורשים מילה אחת או יותר בקוד המכונה.

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

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

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

כאשר מגיע האסמבלר לשורת ההסתעפות (bne A), הוא טרם נתקל בהגדרת התווית A וכמובן לא יודע את המען המשויך לתווית. לכן האסמבלר לא יכול לבנות את הקידוד הבינארי של האופרנד A. נראה בהמשך כיצד נפתרת בעיה זו.

בכל מקרה, תמיד אפשר לבנות במעבר הראשון את הקידוד הבינארי המלא של המילה הראשונה של כל הוראה, את הקידוד הבינארי של מילת-המידע הנוספת של אופרנד מיידי, או רגיסטר, וכן את הקידוד הבינארי של כל הנתונים (המתקבלים מההנחיות data., string.).

המעבר השני

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

לשם כך מבצע האסמבלר מעבר נוסף (מעבר שני) על כל קובץ המקור, ומעדכן את קוד המכונה של האופרנדים המשתמשים בסמלים, באמצעות ערכי הסמלים מטבלת הסמלים.
בסוף המעבר השני, תהיה התוכנית מתורגמת בשלמותה לקוד מכונה.

הפרדת הוראות ונתונים

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

אחת הסכנות הטמונות באי הפרדת ההוראות מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית בתכנית, לנסות "לבצע" את הנתונים כאילו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום תופעה כזו היא 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, אם כי בקובץ הקלט אין חובה שתהיה הפרדה כזו. בהמשך מתואר אלגוריתם של האסמבלר, ובו פרטים כיצד לבצע את ההפרדה.

גילוי שגיאות בתכנית המקור

כפי שהוסבר למעלה, הנחת המטלה היא **שאין שגיאות בהגדרות המקרו**, ולכן שלב קדם האסמבלר אינו מכיל שלב גילוי שגיאות, לעומת זאת האסמבלר אמור לגלות ולדווח על **שגיאות בתחביר של תוכנית המקור**, כגון פעולה שאינה קיימת, מספר אופרנדים שגוי, סוג אופרנד שאינו מתאים לפעולה, שם אוגר לא קיים, ועוד שגיאות אחרות. כמו כן מוודא האסמבלר שכל סמל מוגדר פעם אחת בדיוק.

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

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

הערה: אם יש שגיאה בקוד האסמבלי בגוף מקרו, הרי שגיאה זו יכולה להופיע ולהתגלות שוב ושוב, בכל מקום בו נפרש המקארו. נשים לב שכאשר האסמבלר בודק שגיאות, כבר לא ניתן לזהות שזה קוד שנפרש ממאקרו, כך שלא ניתן לחסוך גילויי שגיאה כפולים.

האסמבלר ידפיס את הודעות השגיאה אל הפלט הסטנדרטי stdout. בכל הודעת שגיאה יש לציין גם את מספר השורה בקובץ המקור בה זוהתה השגיאה (מניין השורות בקובץ מתחיל ב-1).

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

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

שם ההוראה	שיטות מיעון חוקיות עבור אופרנד המקור	שיטות מיעון חוקיות עבור אופרנד היעד
mov	1,3,5	3,5
cmp	1,3,5	1,3,5
add	1,3,5	3,5
sub	1,3,5	3,5
not	אין אופרנד מקור	3,5
clr	אין אופרנד מקור	3,5
lea	3	3,5
inc	אין אופרנד מקור	3,5
dec	אין אופרנד מקור	3,5
jmp	אין אופרנד מקור	3,5
bne	אין אופרנד מקור	3,5
red	אין אופרנד מקור	3,5
prn	אין אופרנד מקור	1,3,5
jsr	אין אופרנד מקור	3,5
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 חלקים: תמונת ההוראות (code) ותמונת הנתונים (data) . לכל חלק 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 ← 0, IC ← 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 הנחיית extern. או הנחיית entry. ? אם לא, עבור ל-11.
9. האם זוהי הנחיית 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 ← 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 ← 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, .string, .data? אם כן, חזור ל-2.
5. האם זוהי הנחיה entry.? אם לא, עבור ל-7.
6. סמן בטבלת הסמלים את הסמלים המתאימים כ-entry. חזור ל-2.
7. השלם את קידוד האופרנדים החל מהמילה השנייה בקוד הבינארי של ההוראה, בהתאם לשיטת המיעון. אם אופרנד הוא סמל, מצא את המען בטבלת הסמלים.
8. עדכן IC ← 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 הדוגמה שראינו למעלה (**לאחר שלב פרישת המאקרואים**), ונציג את הקוד הבינארי שמתקבל במעבר ראשון ובמעבר שני. להלן שוב תכנית הדוגמה.

```
MAIN:     mov     @r3 ,LENGTH
LOOP:     jmp     L1
          prn     -5
          bne     LOOP
          sub     @r1, @r4
          bne     END
L1:       inc     K
          bne     LOOP
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 3 address of label LENGTH	**101000001100** **000110000000** **?**
0103 0104	LOOP: jmp L1	 address of label L1	**000100101100** **?**
0105 0106	prn -5	 immediate value -5	**000110000100** **111111101100**
0107 0108	bne LOOP	 address of label LOOP	**000101001100** **?**
0109 0110	sub @r1, @r4	 registers r1 and r4	**101001110100** **000010010000**
0111 0112	bne END	 address of label END	**000101001100** **?**
0113 0114	L1: inc K	 address of label K	**000011101100** **?**
0115 0116	bne LOOP	 address of label LOOP	**000101001100** **?**
0117	END: stop		**000111100000**
0118 0119 0120 0121 0122 0123 0124	STR: .string "abcdef"	ascii code 'a' ascii code 'b' ascii code 'c' ascii code 'd' ascii code 'e' ascii code 'f' ascii code '\0' (end of string)	**000001100001** **000001100010** **000001100011** **000001100100** **000001100101** **000001100110** **000000000000**
0125 0126 0127	LENGTH: .data 6,-9,15	integer 6 integer -9 integer 15	**000000000110** **111111110111** **000000001111**
0128	K: .data 22	integer 22	**000000010110**

טבלת הסמלים:

סמל	ערך (בבסיס דצימלי)
MAIN	100
LOOP	103
L1	113
END	117
STR	118
LENGTH	125
K	128

נבצע עתה את המעבר השני ונרשום את הקוד בצורתו הסופית:

Decimal Address	Source Code	Binary Machine Code
0100 0101	MAIN: mov @r3 ,LENGTH	**101000001100** **000110000000**

0102		**000111110110**
0103 0104	LOOP: jmp L1	**000100101100** **000111000110**
0105 0106	prn -5	**000110000100** **111111101100**
0107 0108	bne LOOP	**000101001100** **000110011110**
0109 0110	sub @r1, @r4	**101001110100** **000010010000**
0111 0112	bne END	**000101001100** **000111010110**
0113 0114	L1: inc K	**000011101100** **001000000010**
0115 0116	bne LOOP	**000101001100** **000110011110**
0117	END: stop	**000111100000**
0118 0119 0120 0121 0122 0123 0124	STR: .string "abcdef"	**000001100001** **000001100010** **000001100011** **000001100100** **000001100101** **000001100110** **000000000000**
0125 0126 0127	LENGTH: .data 6,-9,15	**000000000110** **111111110111** **000000001111**
0128	K: .data 22	**000000010110**

בסוף המעבר השני, אם לא נתגלו שגיאות, האסמבלר בונה את קבצי הפלט (ראו בהמשך), שמכילים את הקוד הבינארי ומידע נוסף עבור שלבי הקישור והטעינה.
כאמור, שלבי הקישור והטעינה אינם למימוש בפרויקט זה, ולא נדון בהם כאן.

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 תוכניות בתחביר של שפת האסמבלי שהוגדרה בממ״ן זה.

האסמבלר פועל על כל קובץ מקור בנפרד, ויוצר עבורו קבצי פלט כדלקמן:

- קובץ am, המכיל את קובץ המקור לאחר שלב קדם האסמבלר (לאחר פרישת המאקרואים)
- קובץ object, המכיל את קוד המכונה.
- קובץ externals, ובו פרטים על כל המקומות (הכתובות) בקוד המכונה בהם יש מילת-מידע שמקודדת ערך של סמל שהוצהר כחיצוני (סמל שהופיע כאופרנד של הנחיית extern., ומאופיין בטבלת הסמלים כ- external).
- קובץ entries, ובו פרטים על כל סמל שמוצהר כנקודת כניסה (סמל שהופיע כאופרנד של הנחיית entry., ומאופיין בטבלת הסמלים כ- entry).

אם אין בקובץ המקור אף הנחיית extern., האסמבלר <u>לא יוצר</u> את קובץ הפלט מסוג externals.
אם אין בקובץ המקור אף הנחיית entry., האסמבלר <u>לא יוצר</u> את קובץ הפלט מסוג entries.

שמות קבצי המקור חייבים להיות עם הסיומת ".as". למשל, השמות x.as , y.as, ו-hello.as הם שמות חוקיים. העברת שמות הקבצים הללו כארגומנטים לאסמבלר נעשית <u>ללא ציון הסיומת</u>.

לדוגמה: נניח שתוכנית האסמבלר שלנו נקראת assembler, אזי שורת הפקודה הבאה:

assembler x y hello

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

שמות קבצי הפלט מבוססים על שם קובץ הקלט, כפי שהופיע בשורת הפקודה, בתוספת סיומת מתאימה: הסיומת "am." עבור קובץ לאחר פרישת מאקרו, הסיומת "ob." עבור קובץ ה-object, הסיומת "ent." עבור קובץ ה-entries, והסיומת "ext." עבור קובץ ה-externals.

לדוגמה, בהפעלת האסמבלר באמצעות שורת הפקודה: assembler x
יווצר קובץ פלט x.ob, וכן קבצי פלט x.ext ו- x.ent ככל שיש הנחיות entry. או extern. בקובץ המקור.
אם אין מאקרו בקובץ המקור, אזי קובץ "am." יהיה זהה לקובץ "as." .

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: 12 סיביות). במערך ההוראות מכניס האסמבלר את הקידוד של הוראות המכונה שנקראו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמבלר את קידוד הנתונים שנקראו מקובץ המקור (שורות מסוג data. string.).

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 אותה הוא מצא). האסמבלר קובע לכל אופרנד את ערכו באופן הבא:

- אם זה רגיסטר – האופרנד הוא מספר הרגיסטר.
- 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 תיאור שיטות המיעון לעיל)

קביעת שיטת המיעון נעשית בהתאם לתחביר של האופרנד, כפי שהוסבר לעיל בהגדרת שיטות המיעון. למשל, מספר מציין מיעון מיידי, תווית מציינת מיעון ישיר, שם של רגיסטר עם @ לפניו מציין מיעון רגיסטר, 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. בנוסף "משרייין" האסמבלר מקום במערך עבור המילים הנוספות הנדרשות עבור הוראה זו, ומגדיל את מונה ההוראות בהתאם. אם אחד או שני האופרנדים הם בשיטת מיעון רגיסטר או מיידי, האסמבלר מקודד כעת את המילים הנוספות הרלוונטיות במערך ההוראות.

אם זוהי פעולה בעלת אופרנד אחד בלבד, כלומר אין אופרנד מקור, אזי הקידוד הינו זהה לעיל, פרט לסיביות של שיטת המיעון של אופרנד המקור במילה הראשונה, אשר יכילו תמיד 0, מכיוון שאינן רלוונטיות לפעולה.

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

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

3. שורת הנחיה:

כאשר האסמבלר נתקל בהנחיה, הוא פועל בהתאם לסוג 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 (ראו 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 השנייה יכנס למען העוקב אחרי ההוראה הראשונה (תלוי במספר המילים של ההוראה הראשונה), וכך הלאה עד להוראה האחרונה.

מיד לאחר קידוד ההוראה האחרונה, מכניסים לתמונת הזיכרון את קידוד הנתונים שנבנו על ידי ההנחיות '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 כדלקמן:
השורה הראשונה היא כותרת המכילה שני מספרים בבסיס 10: האורך הכולל של קטע ההוראות (במילות זיכרון) ואחריו האורך הכולל של קטע הנתונים (במילות זיכרון). בין שני המספרים יש רווח אחד.

<u>השורות הבאות</u> מכילות את תוכן הזיכרון. בכל שורה מופע תוכן של מילה אחת, לפי הסדר החל מהמילה בכתובת 100. תוכן המילה מקודד בשיטת Base64.

ניתן לקרוא על שיטת Base64 בקישור הבא: https://en.wikipedia.org/wiki/Base64

מילה היא בגודל 12 ביטים. כל 6 ביטים יומרו לתו מתאים לפי הטבלה המגדירה את Base64.
לפיכך בכל שורה בקובץ object (מלבד השורה הראשונה) יהיו בדיוק שני תווים בקוד Base64.

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

פורמט קובץ ה-entries

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

פורמט קובץ ה- externals

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

<u>לתשומת לב</u>: ייתכן ויש מספר כתובות בקוד המכונה בהן מילות-המידע מתייחסות לאותו סמל חיצוני. לכל כתובת כזו תהיה שורה נפרדת בקובץ ה-externals.

<u>נדגים את הפלט שמייצר האסמבלר עבור קובץ מקור בשם **ps.as** שהודגם קודם לכן.</u>

התוכנית לאחר שלב פרישת המאקרו תיראה כך:

```
; file ps.as

.entry LENGTH
.extern W
MAIN:       mov     @r3 ,LENGTH
LOOP:       jmp     L1
            prn     -5
            bne     W
            sub     @r1, @r4
            bne     L3
L1:         inc     K
.entry LOOP
            jmp     W
END:        stop
STR:        .string “abcdef”
LENGTH:     .data   6,-9,15
K:          .data   22
.extern L3
```

להלן טבלת הקידוד המלא הבינארי שמתקבל מקובץ המקור, ולאחריה הפורמטים קבצי הפלט השונים.

טבלת הקידוד הבינארי :

Decimal Address	Source Code	Binary Machine Code
0100 0101 0102	MAIN: mov @r3 ,LENGTH	**101000001100** **000110000000** **000111110110**
0103 0104	LOOP: jmp L1	**000100101100** **000111000110**
0105 0106	prn -5	**000110000100** **111111101100**
0107 0108	bne W	**000101001100** **000000000001**
0109 0110	sub @r1, @r4	**101001110100** **000010010000**
0111 0112	bne L3	**000101001100** **000000000001**
0113 0114	L1: inc K	**000011101100** **001000000010**
0115 0116	jmp W	**000100101100** **000000000001**
0117	END: stop	**000111100000**
0118 0119 0120 0121 0122 0123 0124	STR: .string “abcdef”	**000001100001** **000001100010** **000001100011** **000001100100** **000001100101** **000001100110** **000000000000**
0125 0126 0127	LENGTH: .data 6,-9,15	**000000000110** **111111110111** **000000001111**
0128	K: .data 22	**000000010110**

הקובץ ps.ob:

נבחין כי בדוגמה לעיל, קטע ההוראות בגודל 18 מילים, ואילו קטע הנתונים בגודל 11 מילים.
להלן קובץ הפלט המתאים לדוגמה זו, מקודד בשיטת Base64 (סה״כ 30 שורות, כולל הכותרת) :

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

```
18 11
oM
GA
H2
Es
HG
GE
/s
FM
AB
p0
CQ
FM
AB
Ds
IC
Es
AB
Hg
Bh
Bi
Bj
Bk
Bl
Bm
AA
AG
/3
AP
AW
```

הקובץ ps.ent:

כל ערכי הסמלים מיוצגים בבסיס 10.

LOOP	103
LENGTH	125

הקובץ ps.ext:

כל הכתובות מיוצגות בבסיס 10.

W	108
L3	112
W	116

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, ולכן אורך התוכנית המתורגמת אינו אמור להיות צפוי מראש. אולם כדי להקל במימוש האסמבלר, ניתן להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים לאכסון תמונת קוד המכונה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים וטבלת המקרו), יש לממש באופן יעיל וחסכוני (למשל באמצעות רשימה מקושרת והקצאת זיכרון דינאמי).

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

- מתכונת הפעלת האסמבלר צריכה להיות כפי הנדרש בממ"ן, ללא שינויים כלשהם. כלומר, ממשק המשתמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו לתוכנית האסמבלר כארגומנטים (אחד או יותר) בשורת הפקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גרפיים למיניהם, וכד'.

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

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

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

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

תם ונשלם פרק ההסברים והגדרת הפרויקט.

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

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

לתשומת לבכם: לא תינתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או אשפוז. במקרים אלו יש לבקש ולקבל אישור **מראש** ממנחה הקבוצה.

ב ה צ ל ח ה !