ה א ו נ י ב ר ס י ט ה ה פ ת ו ח ה

20465
מעבדה בתכנות מערכות
חוברת הקורס – סתיו 2022א

כתבה: מיכל אבימור

אוקטובר 2021 – סמסטר סתיו – תשפ"ב

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 (מס' קורס 20465 /א2022)

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
1	22.10.2021-17.10.2021	ספר C פרקים 3-2-1	מפגש ראשון	
2	29.10.2021-24.10.2021	ספר C פרקים 3-2-1		
3	05.11.2021-31.10.2021	ספר C פרק 4	מפגש שני	
4	12.11.2021-07.11.2021	ספר C פרק 4		
5	19.11.2021-14.11.2021	ספר C פרק 5	מפגש שלישי	ממ"ן 11 14.11.2021
6	26.11.2021-21.11.2021	ספר C פרק 5		
7	03.12.2021-28.11.2021 (ב-ו חנוכה)	ספר C פרק 6	מפגש רביעי	
8	10.12.2021-05.12.2021 (א-ב חנוכה)	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)
9	17.12.2021-12.12.2021	ספר C פרק 6,7	מפגש חמישי	ממ"ן 12 12.12.2021
10	24.12.2021-19.12.2021	ספר C פרק 7		
11	31.12.2021-26.12.2021	ספר C פרק 7	מפגש שישי	
12	07.01.2022-02.01.2022	ספר C פרק 8 + פרויקט		ממ"ן 22 02.01.2022
13	14.01.2022-09.01.2022	פרויקט וחזרה	מפגש שביעי	
14	21.01.2022-16.01.2022	פרויקט וחזרה	מפגש שמיני	ממ"ן 23 16.01.2022 ממ"ן 14** 20.03.2022

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 מחלה ממושכת, במקרים אלו יש לתאם את מועד ההגשה 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: 2022א' **מועד אחרון להגשה:** 14.11.2021

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 ההנחיות במטלה/במפגש/באתר). 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 con.c) (50 נקודות)

עליכם לכתוב תכנית, המכילה את הפונקציה **void contract (char s1[], char s2[])**.
הפונקציה מקבלת במחרוזת s1 סדרת תווים (קודי ascii), ומחזירה במחרוזת s2 את אותה הסדרה בפורמט מקוצר. כל רצף מקסימלי של תווים אלפא-נומריים שקודי ה-ascii שלהם עוקבים בסדר עולה יקוצר ל-3 תווים כדלקמן: התו הראשון ברצף, אחריו מקף, ואחריו התו האחרון ברצף. להלן דוגמאות.

- הרצף cdefgh יתורגם ל: c-h
- הרצף 5678 יתורגם ל: 5-8
- הרצף XYZ יתורגם ל: X-Z

לתשומת לב: אין "לקצר" רצף מקסימלי באורך 2 או 1.

- בהנתן המחרוזת s1: "abcdef#LLMN 67890123#HIJKMNOpqrstu(?@AB,1124-8)"
  המחרוזת המקוצרת s2 היא: "a-f#LL-N 6-90-3#H-KM-Op-u(?@AB,1124-8)"

הערה: ניתן להניח שמערך התווים s2 המועבר כפרמטר הוא באורך גדול או שווה לאורך המערך s1.

כמו כן, עליכם לכתוב תכנית ראשית (הפונקציה main), אשר קולטת מהמשתמש שורת קלט אחת שלמה (כולל הרווחים והטאבים, ככל שקיימים כאלה בשורת הקלט), וקוראת לפונקציה contract כאשר שורת הקלט מועברת כמחרוזת בפרמטר s1.
אחרי החזרה מהפונקציה, התכנית הראשית מדפיסה באופן נאה את שתי המחרוזות: מחרוזת הקלט, והמחרוזות המקוצרת. אין לבצע פלט בפונקציה contract.

הנחיות והערות נוספות:

- הניחו כי שורת הקלט באורך מקסימלי של 80 תווים.
- רמז: ניתן לבצע את הקלט באמצעות קריאה יחידה לפונקצית הספריה הסטנדרטית **fgets**.
  לדוגמה: **fgets(str, 81, stdin)**, כאשר **str** הוא שם של מערך תווים המוגדר בתכנית הראשית.
  נשים לב כי המערך **str** צריך להיות בגודל לפחות 81 (מדוע?).
- על התוכנית **להדפיס הודעת בקשה ידידותית לקלט** המפרטת מה על המשתמש להקליד.
- הניחו כי הקלט תקין, כלומר אין צורך לבדוק שגיאות בקלט.
- 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 להדגים את כל אפשרויות הטיפול של הפונקציה contract.
יש להגיש תדפיסי מסך של כל ההרצות. אם תשתמשו בקבצי קלט, יש להגיש גם אותם.

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

עליכם לכתוב תכנית, המקבלת כקלט טקסט דמוי תכנית בשפת C, ובודקת האם בטקסט זה יש בעיה באיזון (קינון נכון) של סוגריים. עליכם לבדוק את האיזון עבור שלשת סוגי הסוגריים: ( ), [ ], { }.
הסוגריים מהסוגים השונים יכולים להופיע במשולב. לדוגמה: הרצף [(]) אינו מאוזן, אך [()] וגם ([]) מאוזנים. הערה: בין הסוגריים יכול להופיע כל טקסט.

על התכנית לבדוק את איזון הסוגריים של כל שורה בטקסט בנפרד. יש להדפיס את הקלט שורה אחרי שורה, ואחרי כל שורה להדפיס הודעה נאה ובולטת המציינת האם בשורה זו בפני עצמה הסוגרים מאוזנים או לא. אין צורך לפרט מהי שגיאת האיזון הספציפית שנתגלתה.
בגמר הטיפול בקלט (אחרי כל השורות), יש להדפיס הודעת סיכום המציינת האם בטקסט כולו קיים איזון מלא של כל הסוגרים, או לא.

יש להתחשב בכך שסוגר שנמצא בין זוג גרשיים (כלומר בתוך מחרוזת לפי תחביר שפת C) לא נכלל בבדיקת האיזון, לדוגמה: "ab[x}". הניחו שכל מחרוזת נמצאת בשלמותה בשורה אחת של הטקסט, וכי אין מחרוזת שלא מסתיימת עם גרשיים לפני סוף השורה.

בנוסף, גם סוגר שנמצא בתוך הערה לא נכלל בבדיקת האיזון, לדוגמה: /* abc[xy}*/. יש לזהות רק הערות בסגנון הקיים בגרסת ANSI-C. הערה יכולה להתפרס על יותר משורה אחת, ולהתחיל ו/או להסתיים באמצע שורה. הניחו כי אין קינון של הערה בתוך הערה, ואין הערה שלא מסתיימת.

כמו כן יש לטפל <u>במקרה מיוחד</u>: שורת קלט המכילה תו יחיד } או { (בנוסף ייתכנו בשורה זו גם תווים לבנים). שורה מיוחדת כזו תסומן כבלתי מאוזנת לכשעצמה, אבל היא לא תקלקל את האיזון של הטקסט כולו, כל עוד הסוגרים המסולסלים בכל השורות המיוחדות מאוזנים זה עם זה באופן גלובלי. לדוגמה, הטקסט שלהלן ((חמש שורות) ייחשב כמאוזן באופן מלא, למרות שהשורה השנייה והשורה האחרונה לכשעצמן מסומנות כבלתי מאוזנות.

```
if (x > y)
{
    z[(1)] =  0;
    printf("x>y\n");
}
```

יש לבנות את התכנית באופן מודולרי, תוך שימוש בפונקציות למשימות שונות (למשל, פונקציה לבדיקת האיזון בשורת קלט נתונה, פונקציה לזיהוי שורה מיוחדת, פונקציה להדפסת הודעה, וכד').

<u>הנחיות והערות נוספות</u>:

- אין חשיבות לנכונות השימוש בסוגריים בקלט מבחינת התחביר של שפת C, אך יש משמעות לקינון תקין של הסוגריים. כמו כן, אין חשיבות לתקינות תחבירית של שאר חלקי הטקסט לפי שפת C, למעט המבנה של מחרוזות והערות כמפורט לעיל.
- בתחילת הריצה, על התוכנית **להדפיס הודעת בקשה ידידותית לקלט,** המפרטת מה יש 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.
- הקלט מסתיים כשהתכנית מזהה בזרם הקלט מצב EOF (סוף הקלט). ראו במפרטים של פונקציות הקלט המגוונות בספריה הסטנדרטית כיצד ניתן לזהות מצב זה. אפשר לדמות מצב EOF במקלדת באמצעות הקלדה של צרוף המקשים ctrl+d באובונטו, או ctrl+z בחלונות.
- ניתן להניח כי אורך מקסימלי של שורה בקלט הוא 100 תווים, כולל התו \n (מעבר לשורה חדשה). <u>אין חסם</u> על מספר השורות בקלט. <u>לתשומת לב</u>: השורה האחרונה בקלט אינה חייבת להסתיים בתו \n. מצב זה רלוונטי בעיקר כאשר הקלט מגיע מקובץ.

חובה לצרף להגשה מספר הרצות בדיקה (לפחות שתי הרצות), המדגימות את פעולת התכנית על מגוון קלטים. לפחות אחת ההרצות תהיה על קלט מאוזן במלואו, ואחת על קלט שאינו מאוזן. כמו כן, אחת ההרצות חייבת להיות עם קלט המכיל לפחות 10 שורות.
יש להגיש תדפיסי מסך של כל ההרצות. אם תשתמשו בקבצי קלט, יש להגיש גם אותם.

<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) 12

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 4,5 ובאופן חלקי 6

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

סמסטר: 2022א' **מועד אחרון להגשה:** 12.12.2021

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 או קבצי פלט (לפי ההנחיות במטלה/במפגש/באתר). 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 (למעט שינוי הקבוע N – ראו פרטים בהמשך).

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 magic.c)

ריבוע קסם בסיסי הוא מטריצה בגודל N x N המכילה ערכים שלמים מ- 1 עד N^2 (כולל), כך שסכום N האיברים בכל שורה, בכל עמודה, ובכל אלכסון הוא זהה (משותף).

<u>דוגמה</u>: להלן ריבוע קסם בסיסי עבור N = 5, כאשר הסכום המשותף הוא 65.

17	**24**	**1**	**8**	**15**
23	**5**	**7**	**14**	**16**
4	**6**	**13**	**20**	**22**
10	**12**	**19**	**21**	**3**
11	**18**	**25**	**2**	**9**

עליכם לכתוב תכנית, המטפלת במטריצה בגודל N x N עם איברים מטיפוס int. המימד N מוגדר בתכנית באמצעות הנחיית #define. כמובן, ניתן לשנות את N מדי פעם ולקמפל מחדש.

התכנית קוראת מהקלט הסטנדרטי N^2 ערכים מטיפוס int בזה אחר זה. אלו הם איברי המטריצה, המועברים לפי סדר השורות, ומשמאל לימין בכל שורה. הערכים מופרדים זה מזה בקלט באמצעות תווים לבנים בלבד (אחד או יותר).

התכנית מדפיסה לפלט הסטנדרטי את המטריצה בתצוגה דו-מימדית נאה (ראו הדוגמה לעיל), וכן הודעה האם המטריצה מהווה ריבוע קסם בסיסי, או לא.

<u>אין להניח</u> שהקלט תקין. על התוכנית להפסיק את עבודתה, ולהדפיס הודעת שגיאה במקרים הבאים: אין מספיק ערכים בקלט; יש יותר מדי ערכים בקלט; יש בקלט ערך שאינו מטיפוס int. לתשומת לב: ערך מטיפוס int שחורג מהתחום $1–N^2$, או יותר ממופע אחד של אותו ערך, <u>אינם נחשבים כשגיאת קלט</u> (אם כי כמובן המטריצה לא תהיה ריבוע קסם בסיסי).

נדרש לבנות את התכנית באופן מודולרי ולחלק את העבודה לפונקציות בצורה מושכלת, כלומר להשתמש בפונקציות נפרדות למשימות שונות, כגון קלט, בדיקות, פלט, וכו'.

<u>הנחיות והערות נוספות:</u>

- בתחילת הריצה, על התוכנית **להדפיס הודעת בקשה ידידותית לקלט,** המפרטת מה על המשתמש להקליד, לרבות מהו מספר הערכים הנדרש (לפי המימד N הקבוע בקוד).
- <u>אין להגביל</u> את ארגון הקלט באופן כלשהו. כלומר, המשתמש יכול להעביר כרצונו כל כמות ערכים בכל שורת קלט. למשל, כל שורה של המטריצה בשורת קלט נפרדת, או כל המטריצה בשורת קלט אחת, או כל ערך בשורה נפרדת, 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.
- רמז: ניתן לקבוע שאין מספיק ערכים בקלט אם מזהים בזרם הקלט מצב EOF (סוף הקלט) לפני שנקלטו N^2 ערכים. אפשר לדמות מצב EOF במקלדת באמצעות הקלדה של צרוף המקשים ctrl+d באובונטו, או ctrl+z בחלונות.
- לבניית ריבוע קסם בכל גודל אפשר להיעזר באתר: www.dcode.fr/magic-square

חובה לצרף להגשה מספר הרצות דוגמה. יש להציג מטריצות בכמה גדלים שונים, כאשר חלקן מהוות ריבוע קסם בסיסי וחלקן לא. כמו כן, יש להדגים את הטיפול בשגיאות מגוונות בקלט. **יש להגיש תדפיסי מסך (או קבצי פלט) של כל ההרצות. אם תשתמשו בקבצי קלט, יש להגיש גם אותם.**

<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: 2022א' **מועד אחרון להגשה:** 02.01.2022

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 או קבצי פלט (לפי ההנחיות במטלה/במפגש/באתר). 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 (או קובץ 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: 2022א' **מועד אחרון להגשה:** 16.01.2022

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 או קבצי פלט (לפי ההנחיות במטלה/במפגש/באתר). 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)

בכל סעיף עליכם לכתוב האם הטענה נכונה, לא נכונה, לפעמים נכונה. עליכם לנמק את תשובתכם. תשובה לא מנומקת, גם אם היא נכונה, לא תזכה בנקודות.

(5 נק') א. בשפת **ANSI-C** אין לנו שליטה על אופן הקצאת המערך בזיכרון, ולא נוכל להסתמך על כך שהמערך יופיע תמיד כגוש זיכרון רציף.

(5 נק') ב. מכיוון שהפרמטר **argv** המועבר לפונקציה **main** הוא למעשה מצביע למערך, ניתן לבצע על הפרמטר פעולות אריתמטיות של מצביעים.

את הפתרון לשאלה זו יש להגיש במסמך (קובץ) מוקלד, בכל פורמט.

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

עליכם לכתוב תכנית המקבלת כארגומנטים בשורת הפקודה:

- מספר שלם חיובי **n** (גדול ממש מ- 0).
- רשימה של שמות של קבצי קלט

עבור כל קובץ ברשימת הקבצים שהועברה, התכנית מדפיסה לפלט הסטנדרטי הודעה נאה, המכילה את קוד האסקי של התו במקום ה- **n** בקובץ (עבור התו הראשון בקובץ **n**=1).
קוד האסקי יודפס בבסיס עשרוני.

לדוגמה, נניח שהתכנית נקראת **seek**. הפעלת התכנית משורת הפקודה יכולה להיות למשל:

> ./seek 760 file1.in file2.in file3.in

בדוגמה זו, התכנית תדפיס הודעה עם קוד האסקי של התו ה- 760 בקובץ, עבור כל אחד משלושת הקבצים:

file1.in
file2.in
file3.in

<u>לתשומת לב</u>: כל סוגי התווים, לרבות תווים לבנים (רווח, טאב, שורה חדשה, וכד') נחשבים לצורך מניית התווים בקובץ.

על מנת להגיע לתו ה- **n** בקובץ, <u>אין לבצע</u> קריאה של **n** תווים החל מתחילת הקובץ (למשל בלולאה), אלא יש להשתמש בפונקצית הספריה הסטנדרטית **fseek**.
בעבודה עם **fseek**, התו ה- **n** בקובץ נמצא בהיסט (offset) של **n-1** מתחילת הקובץ.

על התכנית לטפל במצבי שגיאה כדלקמן.

1. להדפיס הודעה מתאימה ולהפסיק את הריצה, במקרים הבאים:
   - הועברו פחות משני ארגומנטים לתכנית בשורת הפקודה.
   - הארגומנט הראשון אינו מספר שלם חיובי (גדול ממש מ- 0).

2. להדפיס הודעה מפורטת, ואז להמשיך את הריצה, במקרים הבאים:
   - אי אפשר לפתוח את קובץ הקלט הנוכחי.
   - מספר התווים בקובץ הנוכחי קטן מ- **n**.

נדרש לבנות את התכנית באופן מודולרי ולחלק את העבודה לפונקציות בצורה מושכלת, כלומר להשתמש בפונקציות נפרדות למשימות שונות.

הנחיות והערות נוספות:

- מספר קבצי הקלט (הארגומנטים) שמועברים לתכנית אינו מוגבל.
- קובץ קלט יכול להימצא בכל תיקיה, כלומר ארגומנט שהוא קובץ הוא למעשה מסלול במערכת הקבצים. למשל, **file1.in** (ראו דוגמת ההפעלה לעיל) הוא קובץ בתיקיה הנוכחית, ואילו **test/input.txt** הוא קובץ בתיקיה **test** הנמצאת בתוך התיקיה הנוכחית, וכד'.
- לתשומת לב: בקובץ טקסט שנערך במערכת Windows, סוף שורה עשוי לתפוס שני תווים רצופים (שני קודי אסקי: 13+10). כדי למנוע בלבול, מומלץ להקליד ולערוך את קבצי הקלט ב- Linux, ואז סוף שורה בקובץ תופס תו יחיד (קוד אסקי 10).

חובה לצרף להגשה מספר הרצות בדיקה, המדגימות את פעולת התכנית.
יש להדגים הרצה עם יותר מקובץ קלט אחד, וכן את הטיפול במצבי השגיאה שפורטו לעיל.
לכל הרצה יש להגיש תדפיס מסך מלא, וכן את כל קבצי הקלט.

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: 2022א | מועד אחרון להגשה: 20.03.2022

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 באוגרים (registers) הקיימים בתוך היע"מ, ובזיכרון המחשב.
דוגמאות: העברת מספר מתא בזיכרון לאוגר ביע"מ או בחזרה, הוספת 1 למספר הנמצא באוגר, בדיקה האם מספר המאוחסן באוגר שווה לאפס, חיבור וחיסור בין שני אוגרים, 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** (יע"מ), **אוגרים** (רגיסטרים), **וזיכרון** RAM. חלק מהזיכרון משמש כמחסנית (stack).

למעבד 16 אוגרים כלליים, בשמות: r0, r1, r2, r3, r4, r5, r6, r7….,r15.
גודלו של כל אוגר הוא 20 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 19. שמות האוגרים נכתבים תמיד עם אות 'r' קטנה.

כמו כן יש במעבד אוגר 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 8192 תאים, בכתובות 0-8191, וכל תא הוא בגודל של 20 סיביות. לתא בזיכרון נקרא גם בשם "**מילה**". הסיביות בכל מילה ממוספרות כמו באוגר.

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 במודל שלנו מורכבת מפעולה ואופרנדים. מספר האופרנדים הוא בין 0 ל-2, בהתאם לסוג הפעולה. מבחינת התפקיד של כל אופרנד, נבחין בין אופרנד מקור (source) ואופרנד יעד (destination).

כל הוראת מכונה מקודדת למספר מילות זיכרון רצופות, **החל ממילה אחת ועד למקסימום שש מילים,** בהתאם לסוג הפעולה (ראו פרטים בהמשך).

בקובץ הפלט המכיל את קוד המכונה שבונה האסמבלר, כל מילה תקודד "בבסיס מיוחד" (ראו פרטים לגבי קבצי פלט בהמשך).

בהוראת מכונה ללא אופרנדים, המבנה של המילה הראשונה (והיחידה) הוא:

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	A	R	E	opcode															

ואילו בהוראות עם אופרנדים המבנה של הקידוד יכיל לפחות 2 מילות זיכרון במבנה הבא:

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	A	R	E	opcode															
0	A	R	E	funct				אוגר מקור				מיעון מקור		אוגר יעד				מיעון יעד	
מילים נוספות בהתאם לשיטות המיעון																			

במודל המכונה שלנו יש 16 פעולות, בפועל, למרות שניתן לקודד יותר פעולות. כל פעולה מיוצגת בשפת אסמבלי באופן סימבולי על ידי **שם-פעולה**, ובקוד המכונה על ידי קומבינציה ייחודית של ערכי שני שדות במילה הראשונה של ההוראה: **קוד-הפעולה (opcode)**, **ופונקציה (funct)**.

להלן טבלת הפעולות:

שם הפעולה	funct (בבסיס עשרוני)	opcode (בבסיס עשרוני)
mov		0
cmp		1
add	10	2
sub	11	2
lea		4
clr	10	5
not	11	5
inc	12	5
dec	13	5
jmp	10	9
bne	11	9
jsr	12	9
red		12
prn		13
rts		14
stop		15

<u>הערה</u>: שם-הפעולה נכתב תמיד באותיות קטנות. פרטים על מהות הפעולות השונות יובאו בהמשך.

להלן מפרט השדות בקידוד הראשונה בקוד המכונה של כל הוראה.

שדה opcode: שדה זה מיוצג ב- 16 סיביות, והוא מכיל ביט דולק בודד בהתאם לקוד הפעולה. למשל אם מדובר על קוד פעולה 9, אזי סיבית 9 תקבל ערך של 1.

סיבית 19: לא בשימוש, ערכה קבוע לאפס.

סיביות 16-18: עבור קידוד הוראות, סיביות אלה מכילות את סיווג הקידוד (A=Absolute, R=Relocatable, E=External) לכל מילת קידוד של הוראה יש סיווג, ובהתאם לסיווג הסיבית המתאימה בשדה ARE תקבל ערך 1.
סיביות 12-15: שדה זה, הנקרא **funct**, מתפקד כאשר מדובר בפעולה שקוד-הפעולה (opcode) שלה משותף לכמה פעולות שונות (כאמור, קודי-פעולה 2, 5 או 9). השדה funct יכיל ערך ייחודי לכל פעולה מקבוצת הפעולות שיש להן אותו קוד-פעולה. אם קוד-הפעולה משמש לפעולה אחת בלבד, הסיביות של השדה funct יהיו מאופסות.

סיביות 8-11: מכילות את מספרו של אוגר המקור במידה ואופרנד המקור הוא אוגר.
אם אין בהוראה אופרנד מקור שהוא אוגר, סיביות אלה יהיו מאופסות.

סיביות 6-7: מכילות את מספרה של שיטת המיעון של אופרנד המקור.
אם אין בהוראה אופרנד מקור, סיביות אלה יהיו מאופסות.
מפרט של שיטות המיעון השונות יינתן בהמשך.

סיביות 2-5: מכילות את מספרו של אוגר היעד במידה ואופרנד היעד הוא אוגר.
אם אין בהוראה אופרנד יעד שהוא אוגר, סיביות אלה יהיו מאופסות.

סיביות 0-1: מכילות את מספרה של שיטת המיעון של אופרנד היעד.
אם אין בהוראה אופרנד יעד, סיביות אלה יהיו מאופסות.

שיטות מיעון:

שיטות מיעון (addressing modes) הן האופנים השונים בהם ניתן להעביר אופרנדים של הוראת מכונה. בשפת האסמבלי שלנו קיימות ארבע שיטות מיעון, המסומנות במספרים 0,1,2,3.

השימוש בשיטות המיעון מצריך מילות-מידע נוספות בקוד המכונה של הוראה, בנוסף למילים הראשונות. קידוד אופרנד של הוראה עשוי לייצר מילות זיכרון נוספות בהתאם לסוג שיטת המיעון. כאשר בהוראה יש שני אופרנדים, קודם יופיעו מילות-המידע הנוספות של אופרנד המקור, ולאחריהם מילות-המידע הנוספות של אופרנד היעד.

להלן המפרט של שיטות המיעון.

מספר	שיטת המיעון	תוכן מילות-המידע הנוספות	תחביר האופרנד באסמבלי	דוגמה
0	מיעון מיידי (immediate)	מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר שלם בשיטת המשלים ל-2, ברוחב של 16 סיביות מילה זו תסווג מסוג A	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני.	mov #-1, r2 בדוגמה זו האופרנד הראשון של ההוראה (אופרנד המקור) נתון בשיטת מיעון מיידי. ההוראה כותבת את הערך 1- אל אוגר r2.
1	מיעון ישיר (direct)	2 מילות-מידע נוספות של ההוראה יכילו את כתובת האופרנד במבנה של כתובת בסיס והיסט. **<u>כתובת בסיס</u>** = הכתובת הקרובה ביותר לכתובת האופרנד, הקטנה ממנו, ומתחלקת ב-16. למשל, אם כתובת האופרנד היא 36, אזי כתובת הבסיס היא 32, מאחר ו-32 הוא המספר הקרוב ביותר ל-36 שקטן ממנו ומתחלק ב-16. **<u>היסט</u>** = המרחק מכתובת הבסיס לכתובת האופרנד. בדוגמה שניתנה בהסבר לכתובת בסיס, ההיסט יהיה 4. מאחר והמרחק (ההיסט) מ- 32 ל- 36 הוא 4. מילת-המידע הנוספת הראשונה תכיל את כתובת הבסיס. ומילת המידע השנייה תכיל את ההיסט. כתובת הבסיס וההיסט מיוצגים כמספר <u>ללא סימן</u> ברוחב של 16 סיביות. והם יסווגו מסוג R במידה והאופרנד הוא תווית חיצונית, כתובת הבסיס וההיסט יכילו אפסים, וקידודים אלה יסווגו מסוג E במקרה כזה.	האופרנד הוא <u>תווית</u> שכבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת התווית בתחילת השורה של הנחית 'data.' או 'string.', או בתחילת השורה של הוראה, או באמצעות אופרנד של הנחית 'extern.'. התווית מייצגת באופן סימבולי כתובת בזיכרון.	השורה הבאה מגדירה את התווית x: x: .data 23 ההוראה: dec x מקטינה ב-1 את תוכן המילה שבכתובת x בזיכרון (ה"משתנה" x). הכתובת של x מקודדת במילות-המידע הנוספות, במבנה של כתובת בסיס והיסט. <u>דוגמה נוספת</u>: ההוראה jmp next מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתתבצע נמצאת בכתובת next). הכתובת next מקודדת במילות-המידע הנוספות במבנה של כתובת בסיס והיסט. .

מספר	שיטת המיעון	תוכן מילות-המידע הנוספות	תחביר האופרנד באסמבלי	דוגמה
2	מיעון אינדקס	בשיטה זו, יש בקידוד ההוראה 2 מילות מידע נוספות, המכילות את כתובת התווית בצורת כתובת בסיס והיסט כפי שתואר בשיטת מיעון מספר 1. מספרו של האוגר שצוין בסוגריים המרובעות, יישמר כפי שמוסבר בשיטת מיעון מספר 3, בסיביות אוגר המקור/יעד בהתאם לכך אם האופרנד הוא אופרנד מקור/יעד.	האופרנד מתחיל בשם של תווית ולאחריה בסוגריים מרובעות, שם של אוגר שמספרו בין 10 ל- 15 בלבד.	השורה הבאה מגדירה את התווית x: x: .data 23,12,34,50 הפקודה: mov x[r12], r4 בדוגמה זו האופרד הראשון הוא גישה לכתובת של X בזיכרון, והחל משם מתקדמים כמות של מילות זיכרון נוספות לפי הערך שיהיה בזמן ריצה באוגר r12, ומה שיש באותה כתובת ישמש את המעבד כאופרנד המקור. בדוגמה זו אופרנד זו יישמר באוגר r4
3	מיעון אוגר ישיר (register direct)	אין מילות-מידע נוספות בגין האוגר. מספרו של האוגר יישמר בסיביות של אוגר המקור/יעד בהתאם לכך אם האופרנד הוא אופרנד מקור/יעד.	האופרנד הוא שם של אוגר.	clr r1 בדוגמה זו, ההוראה clr מאפסת את האוגר r1. <u>דוגמה נוספת:</u> mov #-1, r2 האופרנד השני של ההוראה (אופרנד היעד) נתון בשיטת מיעון אוגר ישיר. ההוראה כותבת את הערך המיידי 1- אל אוגר r2.

מפרט הוראות המכונה:

בתיאור הוראות המכונה נשתמש במונח **PC** (קיצור של "**Program Counter**").
זהו אוגר פנימי של המעבד (לא אוגר 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	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
mov	0		מבצעת העתקה של תוכן אופרנד המקור (האופרנד הראשון) אל אופרנד היעד (האופרנד השני).	mov A, r1	העתק את תוכן המשתנה A (המילה שבכתובת A בזיכרון) אל אוגר r1.
cmp	1		מבצעת השוואה בין שני האופרנדים. ערך אופרנד היעד (השני) מופחת מערך 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 הדגל Z ("דגל האפס") באוגר הסטטוס (PSW) יודלק, אחרת הדגל יאופס.
add	2	10	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 r0.
sub	2	11	אופרנד היעד (השני) מקבל את תוצאת החיסור של אופרנד המקור (הראשון) מאופרנד היעד (השני).	sub #3, r1	אוגר r1 מקבל את תוצאת החיסור של הקבוע 3 מתוכנו הנוכחי של האוגר r1.
lea	4		lea הוא קיצור (ראשי תיבות) של load effective address. פעולה זו מציבה את מען הזיכרון המיוצג על ידי התווית שבאופרנד הראשון (המקור), אל האופרנד השני (היעד).	lea HELLO, r1	המען שמייצגת התווית HELLO מוצב לאוגר r1.

קבוצת ההוראות השניה:
אלו הן הוראות המקבלות אופרנד אחד בלבד. אופן הקידוד של האופרנד הוא כמו של אופרנד היעד בהוראה עם שני אופרנדים. השדה של אופרנד המקור (סיביות 2-3) במילה הראשונה בקידוד ההוראה אינו בשימוש, ולפיכך יהיו מאופס.

ההוראות השייכות לקבוצה זו הן: clr, not, inc, dec, jmp, bne, jsr, red, prn

הוראה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
clr	5	10	איפוס תוכן האופרנד.	clr r2	האוגר r2 מקבל את הערך 0.
not	5	11	היפוך הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0).	not r2	כל ביט באוגר r2 מתהפך.
inc	5	12	הגדלת תוכן האופרנד באחד.	inc r2	תוכן האוגר r2 מוגדל ב- 1.
dec	5	13	הקטנת תוכן האופרנד באחד.	dec Count	תוכן המשתנה Count מוקטן ב- 1.
jmp	9	10	קפיצה (הסתעפות) בלתי מותנית אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כלומר, כתוצאה מביצוע ההוראה, מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה.	jmp Line	PC←PC+addressOf(Line) הכתובת של תווית Line נשמרת לתוך מצביע התכנית ולפיכך ההוראה הבאה שתתבצע תהיה במען Line.

הוראה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
bne	9	11	bne הוא קיצור (ראשי תיבות) של: branch if not equal (to zero). זוהי הוראת הסתעפות מותנית. אם ערכו של הדגל Z באוגר הסטטוס (PSW) הינו 0, אזי מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה. כזכור, הדגל Z נקבע באמצעות הוראת cmp.	bne Line	אם ערך הדגל Z באוגר הסטטוס (PSW) הוא 0, אזי PC ← address(Line) מצביע התכנית יקבל את כתובת התווית Line, ולפיכך ההוראה הבאה שתתבצע תהיה במען Line.
jsr	9	12	קריאה לשגרה (סברוטינה). כתובת ההוראה שאחרי הוראת jsr הנוכחית (PC+2) נדחפת לתוך המחסנית שבזיכרון המחשב, ומצביע התוכנית (PC) מקבל את כתובת השגרה. <u>הערה</u>: חזרה מהשגרה מתבצעת באמצעות הוראת rts, תוך שימוש בכתובת שבמחסנית.	jsr SUBR	push(PC+2) PC ← address(SUBR) מצביע התכנית יקבל את כתובת התווית SUBR, ולפיכך, ההוראה הבאה שתתבצע תהיה במען SUBR. כתובת החזרה מהשגרה נשמרת במחסנית.
red	12		קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מהקלט ייכנס לאוגר r1.
prn	13		הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn r1	יודפס לפלט התו (קוד ascii) הנמצא באוגר r1

קבוצת ההוראות השלישית:
אלו הן הוראות ללא אופרנדים. קידוד ההוראה מורכב ממילה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד (סיביות 0-3) במילה הראשונה של ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השייכות לקבוצה זו הן: rts, stop.

הוראה	opcode	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
rts	14	מתבצעת חזרה משיגרה. הערך שבראש המחסנית של המחשב מוצא מן המחסנית, ומוכנס למצביע התוכנית (PC). <u>הערה</u>: ערך זה נכנס למחסנית בקריאה לשגרה ע"י הוראת jsr.	rts	PC ← pop() ההוראה הבאה שתתבצע תהיה זו שאחרי הוראת jsr שקראה לשגרה.
stop	15	עצירת ריצת התוכנית.	stop	התוכנית עוצרת מיידית.

<u>מבנה תכנית בשפת אסמבלי:</u>

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

<u>מקרואים</u>:
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)

```
macro m1
  inc r2
  mov  A,r1
endm
```

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
.
.
.
```

<u>התוכנית לאחר פרישת המקרו היא התוכנית שהאסמבלר אמור לתרגם.</u>

הנחיות לגבי מקרו:

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

<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 רק את התווים ' ' ו- '\t' (רווחים וטאבים). ייתכן ובשורה אין אף תו (למעט התו \n), כלומר השורה ריקה.
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 תוכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצאת זיכרון ואתחול משתנים של התוכנית, 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 יותר 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 אופציונאלית.

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 ארבעה סוגים (שמות) 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 בתוכנית את ההוראה:

mov XYZ, r1

אזי בזמן ריצת התוכנית 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. 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).

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

משפט הוראה:

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
```

אפיון השדות במשפטים של שפת האסמבלי

תווית:

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.

הגדרה של תווית 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 שונות זו מזו.

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hEllo:
x:
He78902:

לתשומת לב: מילים שמורות של שפת האסמבלי (כלומר שם של פעולה או הנחיה, או שם של אוגר) אינן יכולות לשמש גם כשם של תווית. לדוגמה: הסמלים add, r3 לא יכולים לשמש כתוויות, אבל הסמלים Add, r19, R3 הם תוויות חוקיות.

התווית מקבלת את ערכה בהתאם להקשר בו היא מוגדרת. תווית המוגדרת בהנחיות data. או 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).

מספר:

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"

האסמבלר בונה מלכתחילה קוד מכונה שמיועד לטעינה החל מכתובת 100. אולם, לא בכל פעם שהקוד ייטען לזיכרון לצורך הרצה, מובטח שאפשר יהיה לטעון אותו החל מכתובת 100. במקרה כזה, קוד המכונה הנתון אינו מתאים ויש צורך לתקן אותו. לדוגמה, מילת-המידע של אופרנד בשיטת מיעון ישיר לא תהיה נכונה, כי הכתובת השתנתה.

הרעיון הוא להכניס תיקונים נקודתיים בקוד המכונה בכל פעם שייטען לזיכרון לצורך הרצה. כך אפשר יהיה לטעון את הקוד בכל פעם למקום אחר, בלי צורך לחזור על תהליך האסמבלי. תיקונים כאלה נעשים בשלב הקישור והטעינה של הקוד (אנו לא מטפלים בכך בממ"ן זה), אולם על האסמבלר להוסיף מידע בקוד המכונה שיאפשר לזהות את הנקודות בקוד בהן נדרש תיקון.

לצד כל מילה בקוד המכונה, האסמבלר מוסיף מאפיין שנקרא "A,R,E". לכל מילה בקוד, מוצמד שדה המכיל את אחת האותיות A או R או E.

- האות A (קיצור של Absolute) באה לציין שתוכן המילה אינו תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל, 2 המילים המכילים את קוד ההוראה ואת שיטות המיעון, או מילת-מידע המכילה אופרנד מיידי).
- האות R (קיצור של Relocatable) באה לציין שתוכן המילה תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל, מילות-מידע המכילות כתובת של תווית בצורת כתובת בסיס והיסט).
- האות E (קיצור של External) באה לציין שתוכן המילה תלוי בערכו של סמל שאינו מוגדר בקובץ המקור הנוכחי (למשל, מילת-מידע המכילה ערך של סמל המופיע בהנחיית extern.).

נשים לב כי רוב המילים בקוד המכונה מאופיינות על ידי האות A. למעשה, רק מילות-המידע הנוספות של שיטת מיעון ישיר ושל שיטת מיעון אינדקס מאופיינות על ידי האות R או E (תלוי אם האופרנד בקוד האסמבלי הוא תווית מקומית או סמל חיצוני).

כאשר האסמבלר מקבל כקלט תוכנית בשפת אסמבלי, עליו לטפל תחילה בפרישת המקרואים, 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 :

```
; file ps.as
.entry  LIST
.extern  W

MAIN:       add     r3, LIST
LOOP:       prn     #48
            macro m1
                inc  r6
                mov r3, W
            endm
            lea     STR, r6
            m1
            sub     r1, r4
            bne     END
            cmp     val1, #-6
            bne     END[r15]
            dec     K
.entry  MAIN
            sub     LOOP[r10] ,r14
END:        stop
STR:       .string  "abcd"
LIST:      .data    6, -9
           .data    -100
.entry  K
K:         .data    31
.extern  val1
```

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

```
; file ps.am
.entry  LIST
.extern  W

MAIN:       add     r3, LIST
LOOP:       prn     #48
            lea     STR, r6
            inc     r6
            mov     r3, W
            sub     r1, r4
            bne     END
            cmp     val1, #-6
            bne     END[r15]
            dec     K
.entry  MAIN
            sub     LOOP[r10] ,r14
END:        stop
STR:       .string  "abcd"
LIST:      .data    6, -9
           .data    -100
.entry  K
K:         .data    31
.extern  val1
```

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

כעת לאחר פרישת כל המקרואים ניתן לעבור לשלב התרגום לקוד מכונה, שלב האסמבלר.

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.

קוד המכונה של התוכנית (הוראות ונתונים) נבנה כך שיתאים לטעינה בזיכרון **החל ממען 100 (עשרוני)**.
התרגום של תוכנית המקור שבדוגמה לקוד בינארי מוצג להלן:

Address (decimal)	Source Code	Machine Code (binary)																			
		19	..	..	..	..	..	..	..	..	..	9	8	7	6	5	4	3	2	1	0
0100	**MAIN**: add r3, LIST	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0101		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
0102		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
0103		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
0104	**LOOP**: prn #48	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0105		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0106		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**
0107	lea STR, r6	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
0108		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**
0109		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0110		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**
0111	inc r6	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0112		**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**
0113	mov r3, W	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0114		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
0115		**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0116		**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0117	sub r1, r4	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0118		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**
0119	bne END	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0120		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0121		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0122		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**
0123	cmp val1, #-6	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
0124		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
0125		**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0126		**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0127		**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**
0128	bne END[r15]	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0129		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**
0130		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0131		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**
0132	dec K	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0133		**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0134		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
0135		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
0136	sub LOOP[r10],r14	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0137		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
0138		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**
0139		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
0140	**END**: stop	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0141	**STR**: .string "abcd"	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**
0142		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**
0143		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**
0144		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**
0145		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0146	**LIST**: .data 6, -9	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**
0147		**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**
0148	.data -100	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**
0149	**K**: .data 31	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**

האסמבלר מחזיק טבלה שבה רשומים כל שמות הפעולה של ההוראות והקודים הבינאריים (opcode, funct) 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 אמור להיות משויך למען 140 (עשרוני), ושהסמל K אמור להיות משויך למען 149, 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 שבתוכנית המקור משויך ערך מספרי, שהוא מען בזיכרון.

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

עבור הדוגמה, טבלת הסמלים נתונה להלן. לכל סמל יש בטבלה גם מאפיינים (attributes) שיוסברו בהמשך. אין חשיבות לסדר השורות בטבלה (כאן הטבלה לפי הסדר בו הוגדרו הסמלים בתכנית).

Symbol	**Value (decimal)**	**Base address**	**offset**	**Attributes**
W	0	0	0	external
MAIN	100	96	4	code, entry
LOOP	104	96	8	code
END	140	128	12	code
STR	141	128	13	data
LIST	146	144	2	data, entry
K	149	144	5	data, entry
val1	0	0	0	external

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 100 (עשרוני), ולכן קוד המכונה של ההוראה הראשונה נבנה כך שייטען לזיכרון החל ממען 100. 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 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. 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 בתוכנית, אלא יוגדר רק בהמשך התוכנית. 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 אוגר, 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 התוכנית, 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

בתוכנית 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 בתוכנית, לנסות "לבצע" את הנתונים כאילו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום תופעה כזו היא הסתעפות לא נכונה. התוכנית 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 מיצר, 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 בתוכנית 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 המקרו. נשים לב שכאשר האסמבלר בודק שגיאות, כבר לא ניתן לזהות שזה קוד שנפרש ממקרו, 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 של שיטות המיעון החוקיות, עבור אופרנד המקור ואופרנד היעד של ההוראות השונות הקיימות בשפה הנתונה:

שיטות מיעון חוקיות עבור אופרנד היעד	שיטות מיעון חוקיות עבור אופרנד המקור	שם ההוראה	funct	opcode
1,2,3	0,1,2,3	mov		0
0,1,2,3	0,1,2,3	cmp		1
1,2,3	0,1,2,3	add	10	2
1,2,3	0,1,2,3	sub	11	2
1,2,3	1,2	lea		4
1,2,3	אין אופרנד מקור	clr	10	5
1,2,3	אין אופרנד מקור	not	11	5
1,2,3	אין אופרנד מקור	inc	12	5
1,2,3	אין אופרנד מקור	dec	13	5
1,2	אין אופרנד מקור	jmp	10	9
1,2	אין אופרנד מקור	bne	11	9
1,2	אין אופרנד מקור	jsr	12	9
1,2,3	אין אופרנד מקור	red		12
0,1,2,3	אין אופרנד מקור	prn		13
אין אופרנד יעד	אין אופרנד מקור	rts		14
אין אופרנד יעד	אין אופרנד מקור	stop		15

תהליך העבודה של האסמבלר

נתאר כעת את אופן העבודה של האסמבלר. בהמשך, יוצג אלגוריתם שלדי למעבר ראשון ושני.

האסמבלר מתחזק' שני מערכים, שייקראו להלן תמונת ההוראות (code) ותמונת הנתונים (data). מערכים אלו נותנים למעשה תמונה של זיכרון המכונה (כל איבר במערך הוא בגודל מילה של המכונה, כלומר 24 סיביות). במערך ההוראות בונה האסמבלר את הקידוד של הוראות המכונה שנקראו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמבלר את קידוד הנתונים שנקראו מקובץ המקור (שורות הנחיה מסוג 'data.' ו- 'string.').

האסמבלר משתמש בשני מונים, שנקראים IC (מונה ההוראות - Instruction-Counter), ו- DC (מונה הנתונים - Data-Counter). מונים אלו מצביעים על המקום הבא הפנוי במערך ההוראות ובמערך הנתונים, בהתאמה. בכל פעם כשמתחיל האסמבלר לעבור על קובץ מקור, המונה IC מקבל ערך התחלתי 100, והמונה DC מקבל ערך התחלתי 0. הערך ההתחלתי IC=100 נקבע כדי שקוד המכונה של התוכנית יתאים לטעינה לזיכרון (לצורך ריצה) החל מכתובת 100.

בנוסף, מתחזק האסמבלר טבלה, אשר בה נאספות כל התוויות בהן נתקל האסמבלר במהלך המעבר על קובץ המקור. לטבלה זו קוראים טבלת-הסמלים (symbol-table). לכל סמל נשמרים בטבלה שם הסמל, ערכו המספרי, ומאפיינים נוספים (אחד או יותר), כגון המיקום בתמונת הזיכרון (data או code), וסוג הנראות של הסמל (external או entry).

במעבר הראשון האסמבלר בונה את טבלת הסמלים ואת השלד של תמונת הזיכרון (הוראות ונתונים).

האסמבלר קורא את קובץ המקור שורה אחר שורה, ופועל בהתאם לסוג השורה (הוראה, הנחיה, או שורה ריקה/הערה).

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

2. שורת הוראה:

האסמבלר מנתח את השורה ומפענח מהי ההוראה, ומהן שיטות המיעון של האופרנדים. מספר האופרנדים נקבע בהתאם להוראה שנמצאה. שיטות המיעון נקבעות בהתאם לתחביר של כל אופרנד, כפי שהוסבר לעיל במפרט שיטות המיעון. למשל, התו # מציין מיעון מידי, תווית מציינת מיעון ישיר, שם של אוגר מציין מיעון אוגר ישיר, וכד'.

אם האסמבלר מוצא בשורת ההוראה גם הגדרה של תווית, אזי התווית (הסמל) המוגדרת מוכנסת לטבלת הסמלים. ערך הסמל בטבלה הוא IC, והמאפיין הוא code.

כעת 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, במידה והוא יוגדר רק בהמשך התוכנית).
- אם זה התו # ואחריו מספר (מיעון מידי) – האופרנד הוא המספר עצמו.
- אם זו שיטת מיעון אחרת – ערכו של האופרנד נקבע לפי המפרט של שיטת המיעון (ראו תאור שיטות המיעון לעיל)

האסמבלר מכניס למערך ההוראות, בכניסה עליה מצביע מונה ההוראות IC, את הקוד הבינארי המלא של המילה הראשונה של ההוראה (בפורמט קידוד כפי שתואר קודם). מילה זו מכילה את קוד הפעולה וה-funct, ואת מספרי שיטות המיעון של אופרנד המקור והיעד.ה- IC מקודם ב-1.

נזכור שכאשר יש רק אופרנד אחד (כלומר אין אופרנד מקור), הסיביות של שיטת המיעון של אופרנד המקור יכילו 0. בדומה, אם זוהי הוראה ללא אופרנדים (rts, stop), אזי הסיביות של שיטות המיעון של שני האופרנדים יכילו 0.

אם זוהי הוראה עם אופרנדים (אחד או שניים), האסמבלר "משריין" מקום במערך ההוראות עבור מילות-המידע הנוספות הנדרשות בהוראה זו, ככל שנדרשות, ומקדם את IC בהתאם. כאשר אופרנד הוא בשיטת מיעון מיידי או אוגר ישיר, האסמבלר מקודד גם את המילה הנוספת המתאימה במערך ההוראות. ואילו בשיטת מיעון ישיר או יחסי, מילת המידע הנוספת במערך ההוראות נשארת ללא קידוד בשלב זה.

3. שורת הנחיה:

כאשר האסמבלר קורא בקובץ המקור שורת הנחיה, הוא פועל בהתאם לסוג ההנחיה, באופן הבא:

I. 'data'.

האסמבלר קורא את רשימת המספרים, המופיעה לאחר 'data.', מכניס כל מספר אל מערך הנתונים (בקידוד בינארי), ומקדם את מצביע הנתונים DC ב-1 עבור כל מספר שהוכנס.

אם בשורה 'data.' מוגדרת גם תווית, אזי התווית מוכנסת לטבלת הסמלים. ערך התווית הוא ערך מונה הנתונים DC שלפני הכנסת המספרים למערך. המאפיין של התווית הוא data.

II. 'string'.

הטיפול ב-'string.' דומה ל- 'data.', אלא שקודי ה-ascii של התווים הם אלו המוכנסים אל מערך הנתונים (כל תו במילה נפרדת). לבסוף מוכנס למערך הנתונים הערך 0 (המציין סוף מחרוזת). המונה DC מקודם באורך המחרוזת + 1 (גם התו המסיים את המחרוזת תופס מקום).

הטיפול בתווית המוגדרת בהנחיה 'string.' זהה לטיפול הנעשה בהנחיה 'data.'.

III. 'entry'.

זוהי הנחיה לאסמבלר לאפיין את התווית הנתונה כאופרנד כ-entry בטבלת הסמלים. בעת הפקת קבצי הפלט (ראו בהמשך), התווית תירשם בקובץ ה-entries.
לתשומת לב: זה לא נחשב כשגיאה אם בקובץ המקור מופיעה יותר מהנחיית entry. אחת עם אותה תווית כאופרנד. המופעים הנוספים אינם מוסיפים דבר, אך גם אינם מפריעים.

IV. 'extern'.

זוהי הצהרה על סמל (תווית) המוגדר בקובץ מקור אחר, והקובץ הנוכחי עושה בו שימוש. האסמבלר מכניס את הסמל המופיע כאופרנד לטבלת הסמלים, עם הערך 0 (הערך האמיתי לא ידוע, וייקבע רק בשלב הקישור), ועם המאפיין external. לא ידוע באיזה קובץ נמצאת הגדרת הסמל, ואין זה רלוונטי עבור האסמבלר.
לתשומת לב: זה לא נחשב כשגיאה אם בקובץ המקור מופיעה יותר מהנחיית extern. אחת עם אותה תווית כאופרנד. המופעים הנוספים אינם מוסיפים דבר, אך גם אינם מפריעים.

לתשומת לב: באופרנד של הוראה או של הנחית entry., מותר להשתמש בסמל אשר יוגדר בהמשך הקובץ (אם באופן ישיר על ידי הגדרת תווית, ואם באופן עקיף על ידי הנחית extern.).

בסוף המעבר הראשון, האסמבלר מעדכן בטבלת הסמלים כל סמל המאופיין כ- data, על ידי הוספת (100) + IC (עשרוני) לערכו של הסמל. הסיבה לכך היא שבתמונה הכוללת של קוד המכונה, תמונת הנתונים מופרדת מתמונת הוראות, וכל הנתונים נדרשים להופיע בקוד המכונה אחרי כל ההוראות. סמל מסוג data הוא תווית בתמונת הנתונים, והעדכון מוסיף לערך הסמל (כלומר לכתובתו בזיכרון) את האורך הכולל של תמונת ההוראות, בתוספת כתובת התחלת הטעינה של הקוד, שהיא 100.

טבלת הסמלים מכילה כעת את ערכי כל הסמלים הנחוצים להשלמת תמונת הזיכרון (למעט ערכים של סמלים חיצוניים).

במעבר השני, האסמבלר משלים באמצעות טבלת הסמלים את קידוד כל המילים במערך ההוראות שטרם קודדו במעבר הראשון. במודל המכונה שלנו אלו הן מילות-מידע נוספות של הוראות, אשר מקודדות אופרנד בשיטת מיעון ישיר או יחסי.

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.

כאמור, אנו מחלקים את תמונת קוד המכונה לשני חלקים: תמונת ההוראות (code), ותמונת הנתונים (data). לכל חלק נתחזק מונה נפרד: IC (מונה ההוראות) ו-DC (מונה הנתונים).

נבנה את קוד המכונה כך שיתאים לטעינה לזיכרון החל מכתובת 100.

בכל מעבר מתחילים לקרוא את קובץ המקור מההתחלה.

מעבר ראשון

1. אתחל DC ← 0, IC ← 100.
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 17.
3. האם השדה הראשון בשורה הוא תווית? 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 המאפיין data. ערך הסמל יהיה בסיס והיסט . (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה, יש להודיע על שגיאה).
7. זהה את סוג הנתונים, קודד אותם בתמונת הנתונים, והגדל את מונה הנתונים DC על ידי הוספת האורך הכולל של הנתונים שהוגדרו בשורה הנוכחית. חזור ל-2.
8. האם זו הנחית extern. או הנחית entry. ? אם לא, עבור ל-11.
9. אם זוהי הנחית entry. חזור ל-2 (ההנחיה תטופל במעבר השני).
10. אם זו הנחית extern., הכנס את הסמל המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים עם פעמיים הערך 0 (בסיס והיסט), ועם המאפיין external. (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה ללא המאפיין external, יש להודיע על שגיאה). חזור ל-2.
11. זוהי שורת הוראה. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם המאפיין code. ערכו של הסמל יהיה בסיס והיסט (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה, יש להודיע על שגיאה).
12. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא, אז הודע על שגיאה בשם ההוראה.
13. נתח את מבנה האופרנדים של ההוראה, וחשב מהו מספר המילים הכולל שתופסת ההוראה בקוד המכונה (נקרא למספר זה L).
14. בנה כעת את הקוד הבינארי של המילה הראשונה של ההוראה, ושל כל מילת-מידע נוספת המקודדת אופרנד במיעון מיידי. אפשר לקודד גם את המילה השנייה בקוד ההוראה (אם קיימת).
15. שמור את הערכים IC ו- L יחד עם נתוני קוד המכונה של ההוראה.
16. עדכן IC ← IC + L, וחזור ל-2.
17. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר הראשון, עצור כאן.
18. שמור את הערכים הסופיים של IC ושל DC (נקרא להם ICF ו- DCF). נשתמש בהם לבניית קבצי הפלט, אחרי המעבר השני.
19. עדכן בטבלת הסמלים את ערכו של כל סמל המאופיין כ- data , ע"י שימוש בערך ICF באופן הבא: ראשית יש לחשב את ערכו המלא המעודכן של הסמל בהינתן ICF-והבסיס+היסט הנוכחיים (ככל שזה מופיע כך בטבלה), ואז לחשב בסיס+היסט חדשים.
20. התחל מעבר שני.

מעבר שני

1. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 7.
2. אם השדה הראשון בשורה הוא סמל (תווית), דלג עליו.
3. האם זוהי הנחית data. או string. או extern. ? אם כן, חזור ל- 1.
4. האם זוהי הנחית entry. ? אם לא, עבור ל- 6.
5. הוסף בטבלת הסמלים את המאפיין entry למאפייני הסמל המופיע כאופרנד של ההנחיה (אם הסמל לא נמצא בטבלת הסמלים, יש להודיע על שגיאה). חזור ל- 1.
6. השלם את הקידוד הבינארי של מילות-המידע של האופרנדים, בהתאם לשיטות המיעון שבשימוש. לכל אופרנד בקוד המקור המכיל סמל, מצא את ערכו של הסמל בטבלת הסמלים (אם הסמל לא נמצא בטבלה, יש להודיע על שגיאה). אם הסמל מאופיין external, הוסף את <u>כתובת</u> מילת-המידע הרלוונטית <u>לרשימת מילות-מידע שמתייחסות לסמל חיצוני</u>. לפי הצורך, לחישוב הקידוד והכתובות, אפשר להיעזר בערכים IC ו-L של ההוראה, כפי שנשמרו במעבר

הראשון. חזור ל- 1. לתשומת לב: יש להשלים שתי מילות מידע לכל אופרנד. כמו כן, אם מדובר בסמל חיצוני, יש לרשום בקובץ ext את הכתובות של שתי מילות המידע. לפי הגדרת השפה, כתובת מילת ההיסט תהיה תמיד עוקבת לכתובת מילת הבסיס (דוגמת קובץ ext בהמשך) .

7. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר השני, עצור כאן.
8. בנה את קבצי הפלט (פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו למעלה (**לאחר שלב פרישת המקרואים**), ונציג את הקוד הבינארי שמתקבל במעבר ראשון ובמעבר שני. להלן שוב תכנית הדוגמה.

```
; file ps.as
.entry  LIST
.extern  W
MAIN:       add     r3, LIST
LOOP:       prn     #48
            lea     STR, r6
            inc     r6
            mov     r3, W
            sub     r1, r4
            bne     END
            cmp     val1, #-6
            bne     END[r15]
            dec     K
.entry  MAIN
            sub     LOOP[r10] ,r14
END:        stop
STR:       .string  "abcd"
LIST:      .data    6, -9
           .data    -100
.entry  K
K:         .data    31
.extern  val1
```

נבצע מעבר ראשון על הקוד לעיל, ונבנה את טבלת הסמלים. כמו כן, נשלים במעבר זה את הקידוד של כל תמונת הנתונים, ושל המילה הראשונה של כל הוראה (נשים לב שיש לקודד גם את המילה השנייה). כמו כן, נקודד מילות-מידע נוספות של כל הוראה, ככל שקידוד זה אינו תלוי בערך של סמל. את מילות-המידע שעדיין לא ניתן לקודד במעבר הראשון נסמן ב "?" בדוגמה להלן.

Address (decimal)	**Source Code**	**Machine Code (binary)**																			
		19	¨	¨	¨	¨	¨	¨	¨	¨	¨	9	8	7	6	5	4	3	2	1	0
0100	**MAIN**: add r3, LIST	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0101		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
0102		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0103		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0104	**LOOP**: prn #48	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0105		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0106		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**
0107	lea STR, r6	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
0108		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**
0109		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0110		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0111	inc r6	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0112		**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**
0113	mov r3, W	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0114		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
0115		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0116		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0117	sub r1, r4	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0118		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**
0119	bne END	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0120		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**

Address (decimal)	Source Code	Machine Code (binary)																			
		19	..	..	..	..	..	..	..	..	..	9	8	7	6	5	4	3	2	1	0
0121		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0122		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0123	cmp val1, #-6	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
0124		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
0125		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0126		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0127		**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**
0128	bne END[r15]	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0129		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**
0130		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0131		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0132	dec K	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0133		**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0134		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0135		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0136	sub LOOP[r10],r14	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0137		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
0138		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0139		**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**	**?**
0140	**END**: stop	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0141	**STR**: .string "abcd"	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**
0142		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**
0143		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**
0144		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**
0145		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0146	**LIST**: .data 6, -9	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**
0147		**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**
0148	.data -100	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**
0149	**K**: .data 31	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**

טבלת הסמלים אחרי מעבר ראשון היא:

Symbol	Value (decimal)	Base address	offset	Attributes
W	0	0	0	external
MAIN	100	96	4	code, entry
LOOP	104	96	8	code
END	140	128	12	code
STR	141	128	13	data
LIST	146	144	2	data, entry
K	149	144	5	data, entry
val1	0	0	0	external

נבצע עתה את המעבר השני. נשלים באמצעות טבלת הסמלים את הקידוד החסר במילים המסומנות "?".
הקוד הבינארי בצורתו הסופית כאן זהה לקוד שהוצג בתחילת הנושא "**אסמבלר עם שני מעברים**".

הערה: כאמור, האסמבלר בונה קוד מכונה כך שיתאים לטעינה לזיכרון החל מכתובת 100 (עשרוני).
אם הטעינה בפועל (לצורך הרצת התוכנית) תהיה לכתובת אחרת, יידרשו תיקונים בקוד הבינארי בשלב הטעינה, שיוכנסו בעזרת מידע נוסף שהאסמבלר מכין בקבצי הפלט (ראו בהמשך).

Address (decimal)	Source Code	Machine Code (binary)																			
		19	..	..	..	..	..	..	..	..	..	9	8	7	6	5	4	3	2	1	0
0100	**MAIN**: add r3, LIST	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0101		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
0102		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
0103		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
0104	**LOOP**: prn #48	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0105		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0106		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**
0107	lea STR, r6	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
0108		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**
0109		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0110		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**
0111	inc r6	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0112		**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**
0113	mov r3, W	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0114		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
0115		**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0116		**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0117	sub r1, r4	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0118		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**
0119	bne END	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0120		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0121		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0122		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**
0123	cmp val1, #-6	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
0124		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
0125		**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0126		**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0127		**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**
0128	bne END[r15]	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0129		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**
0130		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0131		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**
0132	dec K	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0133		**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0134		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
0135		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
0136	sub LOOP[r10],r14	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0137		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
0138		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**
0139		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
0140	**END**: stop	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0141	**STR**: .string "abcd"	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**
0142		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**
0143		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**
0144		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**
0145		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0146	**LIST**: .data 6, -9	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**
0147		**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**
0148	.data -100	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**
0149	**K**: .data 31	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**

טבלת הסמלים אחרי מעבר שני היא:

Symbol	Value (decimal)	Base address	offset	Attributes
W	0	0	0	external
MAIN	100	96	4	code, entry
LOOP	104	96	8	code
END	140	128	12	code
STR	141	128	13	data
LIST	146	144	2	data, entry
K	149	144	5	data, entry
val1	0	0	0	external

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 המקרואים)
- 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." .

נציג כעת את הפורמטים של קבצי הפלט. דוגמאות יובאו בהמשך.

פורמט קובץ ה- object

קובץ זה מכיל את תמונת הזיכרון של קוד המכונה, בשני חלקים: תמונת ההוראות ראשונה, ואחריה ובצמוד תמונת הנתונים.

כזכור, האסמבלר מקודד את ההוראות כך שתמונת ההוראות תתאים לטעינה החל מכתובת 100 (עשרוני) בזיכרון. נשים לב שרק בסוף המעבר הראשון יודעים מהו הגודל הכולל של תמונת ההוראות. מכיוון שתמונת הנתונים נמצאת אחרי תמונת ההוראות, גודל תמונת ההוראות משפיע על הכתובות בתמונת הנתונים. זו הסיבה שבגללה היה צורך לעדכן בטבלת הסמלים, בסוף המעבר הראשון, את ערכי הסמלים המאופיינים כ-data (כזכור, באלגוריתם השלדי שהוצג לעיל, בצעד 19, הוספנו לכל סמל כזה את הערך ICF). במעבר השני, בהשלמת הקידוד של מילות-המידע, משתמשים בערכים המעודכנים של הסמלים, המותאמים למבנה המלא והסופי של תמונת הזיכרון.

כעת האסמבלר יכול לכתוב את תמונת הזיכרון בשלמותה לתוך קובץ פלט (קובץ ה- object).

השורה הראשונה בקובץ ה- object היא "כותרת", המכילה שני מספרים (בבסיס עשרוני): הראשון הוא האורך הכולל של תמונת ההוראות (במילות זיכרון), והשני הוא האורך הכולל של תמונת הנתונים (במילות זיכרון). בין שני המספרים מפריד רווח אחד.
כזכור, במעבר הראשון, בצעד 19, נשמרו ערכי הסמלים תוך שימוש ב ICF.

השורות הבאות בקובץ מכילות את תמונת הזיכרון. בכל שורה זוג שדות: כתובת של מילה בזיכרון, ותוכן המילה,. הכתובת תירשם בבסיס עשרוני בארבע ספרות (כולל אפסים מובילים). תוכן המילה יירשם **בבסיס "מיוחד"** ב-3 ספרות (כולל אפסים מובילים). בין השדות בשורה יש רווח אחד.

"בסיס מיוחד"
כל שורה בתמונת הזיכרון היא באורך 20 סיביות, החל מסיבית 0 (מימין) ועד לסיבית 19 (משמאל).
נחלק את 20 הסיביות ל-5 קבוצות בנות 4 סיביות בכל קבוצה כך:
סיביות 16-19 יקראו קבוצה A
סיביות 12-15 יקראו קבוצה B
סיביות 8-11 יקראו קבוצה C
סיביות 4-7 יקראו קבוצה D
סיביות 0-3 יקראו קבוצה E

כל 4 סיביות של קבוצה מסוימת יומרו לספרה הקסאדצימלית וייכתבו לקובץ ה-OB באופן הבא: תחילה ייכתב שם הקבוצה באות גדולה, ולאחריו הייצוג ההקסהדצימלי של סיביות הקבוצה. כך ייכתבו כל הסיביות מכל הקבוצות עם הפרדה של תו מקף (-) בין קבוצה לקבוצה.
לדוגמה, נסתכל על 20 הסיביות הבאות:

0	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**

הם ייכתבו לקובץ ה-OB כך:

A4-B0-C3-Dc-E1

פורמט קובץ ה-entries

קובץ ה-entries בנוי משורות טקסט, שורה אחת לכל סמל שמאופיין בטבלת הסמלים כ- entry. בשורה מופיע שם הסמל, ולאחריו כתובת הבסיס שלו וההיסט, כפי שנקבע בטבלת הסמלים (בבסיס עשרוני בארבע ספרות, כולל אפסים מובילים). אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.
בין השדות בשורה יש פסיק אחד.

פורמט קובץ ה- externals

קובץ ה-externals בנוי אף הוא משורות טקסט, שורה לכל כתובת בקוד המכונה בה יש מילת מידע המתייחסת לסמל שמאופיין כ- external. כזכור, רשימה של מילות-מידע אלה נבנתה במעבר השני (צעד 6 באלגוריתם השלדי).

כל שורה בקובץ ה-externals מכילה את שם הסמל החיצוני, ולאחריו המילה BASE ולאחריה הכתובת של מילת-המידע בקוד המכונה בה נדרשת כתובת הבסיס (בבסיס עשרוני בארבע ספרות, כולל אפסים מובילים). ולאחר מכן, שורה נוספת המכילה את שם הסמל החיצוני, ולאחריו המילה OFFSET ולאחריה הכתובת של מילת-המידע בקוד המכונה בה נדרש ההיסט (OFFSET) (בבסיס עשרוני בארבע ספרות, כולל אפסים מובילים).
בין השדות בשורה יש רווח אחד. <u>אין חשיבות לסדר השורות</u>, כי כל שורה עומדת בפני עצמה.

<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 המקרו תיראה כך:

```
; file ps.am
.entry  LIST
.extern  W

MAIN:       add     r3, LIST
LOOP:       prn     #48
            lea     STR, r6
            inc     r6
            mov     r3, W
            sub     r1, r4
            bne     END
            cmp     val1, #-6
            bne     END[r15]
            dec     K
.entry  MAIN
            sub     LOOP[r10] ,r14
END:        stop
STR:       .string  “abcd”
LIST:      .data    6, -9
           .data    -100
.entry  K
K:         .data    31
.extern  val1
```

להלן הקידוד הבינארי המלא (תמונת הזיכרון) של קובץ המקור, בגמר המעבר השני.

Address (decimal)	Source Code	Machine Code (binary)																			
		19	..	..	..	..	..	..	..	..	..	9	8	7	6	5	4	3	2	1	0
0100	**MAIN**: add r3, LIST	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0101		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
0102		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
0103		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
0104	**LOOP**: prn #48	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0105		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0106		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**
0107	lea STR, r6	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
0108		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**
0109		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0110		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**
0111	inc r6	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0112		**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**
0113	mov r3, W	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0114		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
0115		**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0116		**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0117	sub r1, r4	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0118		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**
0119	bne END	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0120		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0121		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0122		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**
0123	cmp val1, #-6	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**
0124		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
0125		**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0126		**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0127		**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**
0128	bne END[r15]	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0129		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**
0130		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0131		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**
0132	dec K	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0133		**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
0134		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
0135		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
0136	sub LOOP[r10],r14	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**
0137		**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
0138		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**
0139		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
0140	**END**: stop	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0141	**STR**: .string “abcd”	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**
0142		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**
0143		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**
0144		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**
0145		**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0146	**LIST**: .data 6, -9	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**
0147		**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**
0148	.data -100	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**0**
0149	**K**: .data 31	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**

טבלת הסמלים בגמר המעבר השני היא:

Symbol	Value (decimal)	Base address	offset	Attributes
W	0	0	0	external
MAIN	100	96	4	code, entry
LOOP	104	96	8	code
END	140	128	12	code
STR	141	128	13	data
LIST	146	144	2	data, entry
K	149	144	5	data, entry
val1	0	0	0	external

<u>להלן תוכן קבצי הפלט של הדוגמה.</u>

<u>הקובץ **ps.ob**:</u>

```
          41    9
0100   A4-B0-C0-D0-E4
0101   A4-Ba-C3-Dc-E1
0102   A2-B0-C0-D9-E0
0103   A2-B0-C0-D0-E2
0104   A4-B2-C0-D0-E0
0105   A4-B0-C0-D0-E0
0106   A4-B0-C0-D3-E0
0107   A4-B0-C0-D1-E0
0108   A4-B0-C0-D5-Eb
0109   A2-B0-C0-D8-E0
0110   A2-B0-C0-D0-Ed
0111   A4-B0-C0-D2-E0
0112   A4-Bc-C0-D1-Eb
0113   A4-B0-C0-D0-E1
0114   A4-B0-C3-Dc-E1
0115   A1-B0-C0-D0-E0
0116   A1-B0-C0-D0-E0
0117   A4-B0-C0-D0-E4
0118   A4-Bb-C1-Dd-E3
0119   A4-B0-C2-D0-E0
0120   A4-Bb-C0-D0-E1
0121   A2-B0-C0-D8-E0
0122   A2-B0-C0-D0-Ec
0123   A4-B0-C0-D0-E2
0124   A4-B0-C0-D4-E0
0125   A1-B0-C0-D0-E0
0126   A1-B0-C0-D0-E0
0127   A4-Bf-Cf-Df-Ea
0128   A4-B0-C2-D0-E0
0129   A4-Bb-C0-D3-Ee
```

0130 A2-B0-C0-D8-E0
0131 A2-B0-C0-D0-Ec
0132 A4-B0-C0-D2-E0
0133 A4-Bd-C0-D0-E1
0134 A2-B0-C0-D9-E0
0135 A2-B0-C0-D0-E5
0136 A4-B0-C0-D0-E4
0137 A4-Bb-Ca-Db-Eb
0138 A2-B0-C0-D6-E0
0139 A2-B0-C0-D0-E8
0140 A4-B8-C0-D0-E0
0141 A4-B0-C0-D6-E1
0142 A4-B0-C0-D6-E2
0143 A4-B0-C0-D6-E3
0144 A4-B0-C0-D6-E4
0145 A4-B0-C0-D0-E0
0146 A4-B0-C0-D0-E6
0147 A4-Bf-Cf-Df-E7
0148 A4-Bf-Cf-D9-Ec
0149 A4-B0-C0-D1-Ef

הקובץ ps.ext:

```
W BASE 115
W OFFSET 116

val1 BASE 125
val1 OFFSET 126
```

הקובץ ps.ent:

```
MAIN,96,4
LIST,144,2
K,144,5
```

<u>סיכום והנחיות כלליות</u>

- גודל תוכנית המקור הניתנת כקלט לאסמבלר אינו ידוע מראש, ולכן גם גודלו של קוד המכונה אינו צפוי מראש. אולם בכדי להקל במימוש האסמבלר, מותר להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים לאכסון תמונת קוד המכונה <u>בלבד</u>. 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 **<u>לגלות ולדווח על כל השורות השגויות</u>** בקלט. <u>אין לעצור</u> 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 מחלה ממושכת. במקרים אלו יש לבקש ולקבל אישור מראש מצוות הקורס.

ב ה צ ל ח ה !