

האוניברסיטה הפתוחה

20465

## **מעבדה בתכנות מערכות**

חוברת הקורס – סתיו 2021א

כתבה: מיכל אבימור

אוקטובר 2020 – סמסטר סתיו – תשפ"א

**פנימי – לא להפצה.**

© כל הזכויות שמורות לאוניברסיטה הפתוחה.

## תוכן העניינים

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



## 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](http://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](mailto: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 מאוד להשתמש באתר הקורס ובכל אמצעי העזר שבו וכמובן לפנות אלינו במידת הצורך.

אני מאחלת לך לימוד פורה ומהנה.

בברכה,

מיכל אבימור  
מרכזת ההוראה בקורס.



**1. לוח זמנים ופעילויות (20465 /א/2021)**

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
1	23.10.2020-18.10.2020	ספר C פרקים 1-2-3	מפגש ראשון	
2	30.10.2020-25.10.2020	ספר C פרקים 1-2-3		
3	06.11.2020-01.11.2020	ספר C פרק 4	מפגש שני	
4	13.11.2020-08.11.2020	ספר C פרק 4		
5	20.11.2020-15.11.2020	ספר C פרק 5	מפגש שלישי	ממ"ן 11 15.11.2020
6	27.11.2020-22.11.2020	ספר C פרק 5		
7	04.12.2020-29.11.2020	ספר C פרק 6	מפגש רביעי	
8	11.12.2020-06.12.2020 (ו חנוכה)	ספר C פרק 6		ממ"ן 12 06.12.2020

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

לוח זמנים ופעילויות - המשך

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
9	18.12.2020-13.12.2020 (א-ו חנוכה)	ספר C פרק 6		
10	25.12.2020-20.12.2020	ספר C פרק 7	מפגש חמישי	
11	01.01.2021-27.12.2020	ספר C פרק 7		ממ"ן 22 27.12.2020
12	08.01.2021-03.01.2021	ספר C פרק 8 + פרויקט	מפגש שישי	
13	15.01.2021-10.01.2021	פרויקט וחזרה		ממ"ן 23 10.01.2021
14	22.01.2021-17.01.2021	פרויקט וחזרה	מפגש שביעי	ממ"ן 14** 14.3.2021

מועדי בחינות הגמר יפורסמו בנפרד

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

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



## 2. תיאור המטלות

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

רוב המטלות בקורס זה הן **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 (.h, .c), 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 יעשה 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:

- הערה בראש תכנית שתכלול תיאור תמציתי של מטרות התכנית, כיצד מושגת מטרה זו, תיאור המודלים והאלגוריתם, קלט/פלט וכל הנחה שהנכם מניחים.
- 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.
- 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.

**לתשומת 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 בשלוש!), 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

משקל המטלה: 4 נקודות (חובה)

מספר השאלות: 2

מועד אחרון להגשה: 15.11.2020

סמסטר: 2021 א'

## 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 (`.c`, `.h`), קבצי ההרצה (את קבצי `.o` אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי `makefile`), וכן קבצי קלט ותדפיסי מסך או קבצי פלט (לפי הנחיות במטלה/במפגש/באתר). כל תוכנית תהיה בתיקה נפרדת. נדרש ששם התיקה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה `main`, ללא הסיומת `.c`. יש להגיש תכניות מלאות (בין השאר מכילות `main`), הניתנות להידור והרצה, ומאפשרות בדיקה של כל תוצאות הריצה המגוונות ללא צורך בשינויים כלשהם בקוד התוכנית. את המטלה יש להגיש בקובץ `zip`. לאחר ההגשה, יש להוריד את המטלה משרת האו"פ למחשב האישי, ולבדוק שהקבצים אכן הוגשו באופן תקין.

## שאלה 1 (תכנית ראשית בקובץ `count_bits.c`) (50 נקודות)

עליכם לכתוב פונקציה בשם `count_bits`, אשר סופרת כמה סיביות דלוקות (אלו שערכן 1) יש בייצוג הבינרי של מספר שלם מטיפוס `long int`, אותו היא מקבלת כפרמטר. הפונקציה מחזירה את כמות הסיביות הדלוקות.

על הפונקציה `count_bits` להיות ניידת (`portable`), כלומר עליה להיות בלתי תלויה ברוחב הייצוג של ערך מטיפוס `long int` בסביבת ריצה מסוימת (מעבד, או מערכת הפעלה או מהדר (`compiler`) מסוימים). רוחב הייצוג הוא כמות הבתים (`bytes`) המשמשים לייצוג הערך. למשל, במערכת אונוטו הוירטואלית של הקורס, רוחב הייצוג של `long int` הוא 4 בתים; במערכת חלונות על מעבד בטכנולוגית 64 סיביות, רוחב הייצוג הוא 8 בתים; ואילו במערכת חלונות על מעבד בטכנולוגית 32 סיביות, רוחב הייצוג הוא 4 בתים. כזכור, בכל בית יש תמיד 8 סיביות.

כמו כן, עליכם לכתוב תכנית ראשית (הפונקציה main), שתקלוט מהמשתמש מספר שלם בבסיס עשרוני, ותקרא לפונקציה count\_bits עם מספר זה כפרמטר. אחרי החזרה מהפונקציה, התכנית הראשית תדפיס באופן נאה את המספר שנקלט (בבסיס עשרוני) ואת כמות הסיביות הדלוקות שבו.

הקלט לתוכנית הוא מהמקלדת. לתשומת לב: כל הרצה של התכנית מטפלת במספר יחיד בקלט.

על התוכנית להדפיס הודעת בקשה ידידותית לקלט המפרטת מה על המשתמש להקליד. הניחו כי הקלט תקין, כלומר מספר שלם עשרוני שאינו חורג מהערך המקסימלי/מינימלי האפשרי בטיפוס long int בסביבת הריצה הנתונה. אין צורך לבדוק שגיאות בקלט.

**חובה לצרף להגשה מספר הרצות בדיקה**, המדגימות את פעולת התוכנית על מגוון נתוני קלט (חיוביים, שליליים, אפס). יש להגיש תדפיסי מסך (או קבצי פלט) של כל ההרצות.

## שאלה 2 (תכנית ראשית בקובץ count\_odd\_bits.c (50 נקודות)

עליכם לכתוב פונקציה בשם count\_odd\_bits, אשר סופרת ומחזירה את כמות הסיביות הדלוקות במקומות אי זוגיים במספר מטיפוס unsigned short int, המועבר כפרמטר לפונקציה. תזכורת: מנין הסיביות מתחיל ממקום 0 (הסיבית הכי פחות משמעותית), שהוא מקום זוגי.

### דוגמה 1:

הפרמטר הוא המספר העשרוני 149 (בבסיס בינרי ברוחב 2 בתים: 0000000010010101). רק סיבית 7 היא אי זוגית ודלוקה, לכן הפונקציה תחזיר 1.

### דוגמה 2:

הפרמטר הוא המספר העשרוני 32811 (בבסיס בינרי ברוחב 2 בתים: 1000000000101011). סיביות 1,3,5,15 הן אי זוגיות ודלוקות, לכן הפונקציה תחזיר 4.

על הפונקציה count\_odd\_bits להיות ניידת, בדומה לפונקציה בשאלה 1.

כמו כן, עליכם לכתוב תכנית ראשית (הפונקציה main), שתקלוט מהמשתמש סדרה של מספרים מטיפוס unsigned short int בבסיס עשרוני, ותקרא לפונקציה count\_odd\_bits עם כל אחד מהמספרים שהועברו בקלט. אחרי החזרה מהפונקציה, התכנית הראשית תדפיס באופן נאה את המספר (בבסיס עשרוני) ואת כמות הסיביות האי-זוגיות הדלוקות שבו.

לתשומת לב: כמות המספרים בקלט **אינה חסומה, ואינה ידועה לתכנית**. יש לקלוט ולטפל בכל מספר בנפרד (כולל פלט), ורק לאחר מכן לקלוט ולטפל במספר הבא. ממשיכים כך עד שהתכנית מזהה שנגמר הקלט (מצב EOF בקלט), ואז התכנית מסתיימת. בקלט מהמקלדת, אפשר לדמות מצב EOF על ידי הקלדה של צירוף מקשים: ctrl+d באובונטו, או ctrl+z בחלונות.

על התוכנית להדפיס הודעת בקשה ידידותית לקלט, המפרטת מה על המשתמש להעביר בקלט. הניחו כי הקלט תקין, כלומר כל מספר בקלט הוא שלם עשרוני שאינו חורג מהערך המקסימלי האפשרי בטיפוס unsigned short int בסביבת הריצה הנתונה. אין צורך לבדוק שגיאות בקלט.

הקלט לתוכנית הוא מ-stdin, ויכול להגיע מהמקלדת או מקובץ (באמצעות redirection בעת הפעלת התוכנית). המספרים בקלט יופרדו זה מזה באמצעות תווים לבנים (אחד או יותר). תזכורת: תו לבן הוא רווח, טאב, או שורה חדשה. לנוחיותכם, הכינו מספר קבצי קלט והשתמשו בהם שוב ושוב לדיבוג התוכנית.

**חובה לצרף להגשה הרצות דוגמה** (אחת או יותר), המדגימות את פעולת התוכנית על מגוון נתוני קלט. יש להגיש תדפיסי מסך (או קבצי פלט) של כל ההרצות. במידה ותשתמשו בקבצי קלט, יש להגיש גם קבצים אלה.

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

**בהצלחה!**





# מטלת מנחה (ממ"ן) 12

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 4,5 ובאופן חלקי 6

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

סמסטר: 2021' מועד אחרון להגשה: 06.12.2020

## 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 (.c, .h), קבצי ההרצה (את קבצי .o אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי `makefile`), וכן קבצי קלט ותדפיסי מסך או קבצי פלט (לפי הנחיות במטלה/במפגש/באתר).

כל תוכנית תהיה בתיקה נפרדת. נדרש ששם התיקה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה `main`, ללא הסיומת .c.

יש להגיש תכניות מלאות (בין השאר מכילות `main`), הניתנות להידור והרצה, ומאפשרות בדיקה של כל תוצאות הריצה המגוונות ללא צורך בשינויים כלשהם בקוד התוכנית. את המטלה יש להגיש בקובץ `zip`. לאחר ההגשה, יש להוריד את המטלה משרת האו"פ למחשב האישי, ולבדוק שהקבצים אכן הוגשו באופן תקין.

## שאלה 1 (תכנית ראשית בקובץ `partialSums.c`)

עליכם לכתוב פונקציה בשם `partialSums`, אשר מקבלת כפרמטרים מערך של מספרים שלמים מטיפוס `int`, ואת גודל המערך. הפונקציה מחשבת ומחזירה מערך של "סכומים חלקיים", שבו כל איבר מהמערך המקורי מוחלף בסכום האיברים עד אליו (כולל) החל מאינדקס 0. **אסור** לפונקציה לגרום לשינויים במערך המקורי (רמז: חישבו כיצד להחזיר את מערך הסכומים).

## לדוגמה:

בהינתן המערך הבא בגודל 8 (משמאל לימין):

14 10 -1 10 -15 1 5 3

הפונקציה תחזיר את המערך הבא:

27 13 3 4 -6 9 8 3

כמו כן עליכם לכתוב תכנית ראשית (הפונקציה main) הקולטת מהמשתמש גודל של מערך, ואז קולטת מהמשתמש את איברי המערך (מספרים שלמים). התכנית הראשית קוראת לפונקציה partialSums עם מערך הקלט, ואחרי החזרה מהפונקציה מדפיסה באופן נאה את מערך הסכומים החלקיים.

בקלט ובפלט יש להשתמש בייצוג עשרוני. הניחו כי הקלט תקין. אין צורך לטפל בשגיאות 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.

על התוכנית להדפיס הודעת בקשה ידידותית בכל פעם כשנדרש קלט.

כמו כן יש להדפיס באופן יזום מתוך התוכנית את הנתונים כפי שנקלטו, וזאת לפני הקריאה לפונקציה. באופן זה, הקלט יוצג במסך גם כאשר הוא מגיע מקובץ (כידוע, נתונים הנקראים מקובץ אינם מוצגים במסך בזמן הקלט).

חובה לצרף להגשה מספר הרצות דוגמה, המדגימות את פעולת התוכנית על מערכים בגדלים שונים ועל מגוון ערכים בכל מערך. יש להגיש תדפיסי מסך (או קבצי פלט) של כל ההרצות. במידה ותשתמשו בקבצי קלט, יש להגיש גם קבצים אלה.

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

**בהצלחה!**

# מטלת מנחה (ממ"ן) 22

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 4,5,6

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

סמסטר: 2021' מועד אחרון להגשה: 27.12.2020

## 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 (.c, .h), קבצי ההרצה (את קבצי 0. אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי makefile), וכן קבצי קלט ותדפיסי מסך או קבצי פלט (לפי הנחיות במטלה/במפגש/באתר). כל תוכנית תהיה בתיקיה נפרדת. נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם הקובץ המכיל את הפונקציה main, ללא הסיומת .c.

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

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

## שאלה 1 (בקבצים mat.c, mat.h, ותכנית ראשית בקובץ mymat.c)

עליכם לכתוב תכנית שפועלת כ"מחשב כיס" אינטראקטיבי לביצוע פעולות חשבוניות על מטריצות.

תזכורת:

להלן כמה פעולות חשבוניות בסיסיות על מטריצות.

חיבור מטריצות.

דוגמה:

$$\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{array} + \begin{array}{cccc} 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \end{array} = \begin{array}{cccc} 3 & 4 & 5 & 4 \\ 3 & 4 & 5 & 4 \\ 3 & 4 & 5 & 4 \\ 3 & 4 & 5 & 4 \end{array}$$

חיסור מטריצות.

דוגמה:

$$\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \end{array} - \begin{array}{cccc} 0.5 & 0 & 1 & 0 \\ 2 & 2 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 2 & 3 \end{array} = \begin{array}{cccc} 0.5 & 1 & 0 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 1 & 0 \\ 1 & 2 & 0 & -1 \end{array}$$

כפל מטריצות.  
דוגמה:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 1 & 0 \\ 2 & 2 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 5 & 3 & 3 & 4 \\ 10 & 6 & 6 & 8 \\ 5 & 3 & 3 & 4 \\ 10 & 6 & 6 & 8 \end{pmatrix}$$

כפל מטריצה בסקלר.  
דוגמה:

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 2 & 2 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 2 & 3 \end{pmatrix} * 0.5 = \begin{pmatrix} 0.5 & 0 & 0.5 & 0 \\ 1 & 1 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0.5 \\ 0.5 & 0 & 1 & 1.5 \end{pmatrix}$$

שחלוף (transposition) של מטריצה.  
דוגמה:

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ -1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 2 & -1 & 2 \\ 1 & 2 & 1 & 2 \\ 1 & 2 & 1 & 2 \\ 1 & 2 & 1 & 2 \end{pmatrix}$$

### משימות התכנית:

עליכם לכתוב תכנית הקוראת פקודות מהקלט הסטנדרטי, מפענחת ומבצעת אותן. הפקודות עוסקות בפעולות חשבוניות על מטריצות (על פי התזכורת לעיל).

עליכם להגדיר, תוך שימוש ב- `typedef`, את הטיפוס `mat` אשר מחזיק מטריצה בגודל 4 על 4. איברי המטריצה הם מספרים ממשיים. מבנה הנתונים שהגדרתם צריך להיות חסכוני מבחינת ניצול הזיכרון, ויעיל מבחינת הגישה אליו.

בנוסף, עליכם להגדיר בתכנית הראשית (בפונקציה `main`) שישה משתנים מטיפוס `mat`, בשמות: `MAT_A`, `MAT_B`, `MAT_C`, `MAT_D`, `MAT_E`, `MAT_F`

בתחילת ריצת התכנית, יש לאתחל את כל המטריצות עם אפסים בכל האיברים.

כעת, עליכם לבצע פעולות חשבוניות על מטריצות. כל פעולה תופעל באמצעות פקודה שמועברת מהמשתמש בקלט לתכנית. בפקודות שמפורטות להלן, כל אופרנד שהוא שם של מטריצה יהיה אחד מששת המשתנים שהוגדרו לעיל. כיוון הקריאה של נוסח הפקודה הוא משמאל לימין.

### מבנה הפקודות המשמשות כקלט לתכנית:

1. הצבת ערכים במטריצה:

#### **רשימת ערכים ממשיים מופרדים בפסיקים, שם-מטריצה `read_mat`**

הפקודה מציבה את הערכים שברשימה לתוך המטריצה ששמה ניתן בפקודה, לפי סדר השורות. אם ברשימה יש פחות מ-16 ערכים, האיברים שלא נתקבל עבורם ערך יכילו אפסים. אם יש יותר מ-16 ערכים, התכנית תתעלם מהערכים העודפים. חובה שיהיה ברשימה לפחות ערך אחד.

הערכים בקלט הם מספרים ממשיים בבסיס עשרוני.

לדוגמה, הפקודה: `read_mat MAT_A, 5, 6.253, -7, -200.5, 23`  
תציב בתא `[0,0]` במטריצה `MAT_A` את הערך 5, בתא `[0,1]` את הערך 6.253, בתא `[0,2]` את הערך -7, בתא `[0,3]` את הערך -200.5, ובתא `[1,0]` את הערך 23. ביתר תאי המטריצה `MAT_A` יוצב הערך 0.

2. הדפסת מטריצה :

### שם מטריצה `print_mat`

הפקודה מדפיסה את תוכן המטריצה ששמה ניתן בפקודה, בתצוגה דו-מימדית נאה. הערכים יודפסו בבסיס עשרוני.

יש להקפיד בהדפסה על עימוד נאה ומיושר של אברי המריצה. זכרו שמדובר במספרים ממשיים. מותר להסתפק בהדפסה עם דיוק של 2 ספרות מימין לנקודה, וכן ברוחב שדה של 7 תווים לכל המספר (כולל נקודה-עשרונית וסימן מינוס לפי הצורך). במידה והחלק השלם של מספר דורש רוחב שדה גדול יותר, מותרת סטיה מהעימוד המיושר בשורה זו. מומלץ להשתמש ביכולות של הפונקציה `printf` לשלוט בפורמט של השדה המודפס.

לדוגמה: הפקודה `print_mat MAT_A` (לאחר ביצוע דוגמת הפקודה `read_mat` מהסעיף הקודם) תבצע הדפסה בסגנון הבא (או דומה לו):

5.00	6.25	-7.00	-200.50
23.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00

3. חיבור מטריצות :

### שם-מטריצה-ג', שם-מטריצה-ב', שם-מטריצה-א' `add_mat`

הפקודה מחברת את מטריצה א' ומטריצה ב' ומאכסנת את התוצאה במטריצה ג'.

4. חיסור מטריצות :

### שם-מטריצה-ג', שם-מטריצה-ב', שם-מטריצה-א' `sub_mat`

הפקודה מחסרת את מטריצה ב' ממטריצה א' ומאכסנת את התוצאה במטריצה ג'.

5. כפל ממטריצות :

### שם-מטריצה-ג', שם-מטריצה-ב', שם-מטריצה-א' `mul_mat`

הפקודה מכפילה את מטריצה א' במטריצה ב' ומאכסנת את התוצאה במטריצה ג'.

6. כפל מטריצה בסקלר :

### שם-מטריצה ב', ערך-ממשי, שם-מטריצה-א' `mul_scalar`

הפקודה מכפילה את מטריצה א' בערך ממשי (הפרמטר השני) ומאכסנת את התוצאה במטריצה ב'. הערך הממשי נתון בבסיס עשרוני.

7. שחלוף מטריצה :

### שם מטריצה ב', שם-מטריצה-א' `trans_mat`

הפקודה מבצעת שחלוף (transpose) של מטריצה א' ומאכסנת את התוצאה במטריצה ב'.

8. סיום התכנית :

**stop**

הפקודה גורמת לסיום התכנית.

**לתשומת לב:** אותו שם מטריצה יכול לשמש ביותר מארגומנט אחד באותה הפקודה. מימוש הפעולות החשבוניות על מטריצות צריך להתחשב בכך (לא לדרוס נתונים תוך כדי החישוב). לדוגמה, הפקודות שלהלן תקינות ומוגדרות היטב:

```
mul_mat MAT_A, MAT_B, MAT_A
trans_mat MAT_C, MAT_C
```

#### המבנה התחבירי של הקלט:

- כל פקודה תופיע בשלמותה בשורת קלט יחידה, כולל כל הארגומנטים. מותרות גם שורות ריקות (שורות המכילות רק תווים לבנים).
- שם הפקודה מופרד מהארגומנט הראשון באמצעות רווחים ו/או טאבים (אחד או יותר).
- בין כל שני ארגומנטים 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 בתחילת השורה לפני שם הפקודה, וגם בסוף השורה (אחרי הארגומנט האחרון).
- אסור שיהיו תווים מיותרים (תווי זבל) בסוף השורה (למעט תווים לבנים).
- שמות הפקודות יופיעו באותיות קטנות בלבד, ושמות המשתנים באותיות גדולות בלבד.

#### אופן פעולת התכנית:

יש לממש ממשק משתמש ידידותי, כך שהמשתמש יוכל להבין בכל שלב של התכנית מה קורה. בפרט, על התכנית להדפיס הודעה או סימן (prompt) בכל פעם שהיא מוכנה לקלוט את הפקודה הבאה. התכנית תמשיך לקלוט ולבצע פקודה אחרי פקודה, עד שתקבל הפקודה stop.

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

יש לטפל גם במצב של EOF (גמר הקלט). סיום התכנית שלא באמצעות פקודת stop מפורשת בקלט אינה נחשבת תקינה (גם לא כאשר הקלט מגיע מקובץ באמצעות redirection), ויש להדפיס על כך הודעת שגיאה ורק אז לעצור. לתשומת לב: השורה האחרונה בקובץ קלט אינה חייבת להסתיים בתו "שורה חדשה".

#### 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_mat MAT_G, 3.2, 8
```

יש להגיב בהודעה כגון:

```
Undefined matrix name
```

2. לפקודה:

```
read_mat mat_a, 3.2, -5.3
```

יש להגיב בהודעה כגון:

```
Undefined matrix name
```

3. לפקודה:

```
do_it MAT_A, MAT_B, MAT_C
```

יש להגיב בהודעה כגון:

```
Undefined command name
```

Add_Mat MAT_A, MAT_B, MAT_C	4. לפקודה:
Undefined command name	יש להגיב בהודעה כגון:
read_mat MAT_A, abc, 567	5. לפקודה:
Argument is not a real number	יש להגיב בהודעה כגון:
read_mat MAT_A, 3, -4.2, 6,	6. לפקודה:
Extraneous text after end of command	יש להגיב בהודעה כגון:
read_mat MAT_A	7. לפקודה:
Missing argument	יש להגיב בהודעה כגון:
mul_mat MAT_B, MAT_C	8. לפקודה:
Missing argument	יש להגיב בהודעה כגון:
trans_mat MAT_B, MAT_C, MAT_D	9. לפקודה:
Extraneous text after end of command	יש להגיב בהודעה כגון:
print_mat, MAT_A	10. לפקודה:
Illegal comma	יש להגיב בהודעה כגון:
trans_mat MAT_A MAT_B	11. לפקודה:
Missing comma	יש להגיב בהודעה כגון:
sub_mat MAT_A, , MAT_B, MAT_C	12. לפקודה:
Multiple consecutive commas	יש להגיב בהודעה כגון:
mul_scalar MAT_A, MAT_B, MAT_C	13. לפקודה:
Argument is not a scalar	יש להגיב בהודעה כגון:
stop now	14. לפקודה:
Extraneous text after end of command	יש להגיב בהודעה כגון:

להלן דוגמה של סדרת פקודות שכולן תקינות:  
הערה: סדרה כגון זו יכולה לשמש כקלט בהרצת בדיקה ללא טיפול בשגיאות בקלט.

```
print_mat MAT_A
print_mat MAT_B
print_mat MAT_C
read_mat MAT_A, 1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 6, 6, 6, 6, 6
```

```

read_mat MAT_B, 1, 2.3456, -7.89
read_mat MAT_C, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
print_mat MAT_A
print_mat MAT_B
print_mat MAT_C
add_mat MAT_A, MAT_B, MAT_D
print_mat MAT_D
sub_mat MAT_B, MAT_A, MAT_E
print_mat MAT_E
mul_mat MAT_B, MAT_C, MAT_F
print_mat MAT_F
mul_scalar MAT_A, 12.5, MAT_A
print_mat MAT_A
trans_mat MAT_C, MAT_C
print_mat MAT_C
read_mat MAT_B, 0
print_mat MAT_B
mul_mat MAT_A, MAT_A, MAT_A
print_mat MAT_A
stop

```

#### ארגון קוד התכנית:

יש לחלק את התכנית למספר קבצי מקור: `mat.c`, `myamat.c` ו-`mat.h`.

- בקובץ `mat.c` יש לרכז את הפונקציות החשובות על מטריצות. לא יבוצע כל קלט/פלט בקובץ זה.
- בקובץ `myamat.c` תהיה הפונקציה `main`, וכן כל פעילויות האינטראקציה עם המשתמש, וניתוח הפקודות (לרבות הודעות שגיאה). כמו כן, בפונקציה `main` יוגדרו ששת המשתנים מטיפוס `mat`.
- בקובץ `mat.h` תהיה הגדרת טיפוס הנתונים `mat`, וכן ההצהרות (אב-טיפוס) של הפונקציות הממומשות בקובץ `mat.c`. יש לכלול (`#include`) את הקובץ `mat.h` בקבצי המקור האחרים.
- אפשר לבנות קבצי מקור נוספים (למשל: קובץ המכיל פונקציות עזר לניתוח הקלט, וכד').

**הקלט לתכנית הוא מ-`stdin`, ויכול להגיע מהמקלדת או מקובץ** (באמצעות `redirection` בעת הפעלת התכנית). לנוחיותכם, הכינו מספר קבצי קלט והשתמשו בהם שוב ושוב כדיבוג התכנית. בכל קובץ קלט תהיה סדרה של פקודות מגוונות על מטריצות.

על התכנית להדפיס הודעת בקשה ידידותית לקלט עבור כל שורת קלט (כל פקודה). כמו כן, יש להדפיס באופן יזום מתוך התכנית את השורה כפי שנקלטה, וזאת לפני הניתוח של הפקודה. באופן זה, שורת הקלט תוצג גם כאשר הקלט מגיע מקובץ (כידוע, נתונים הנקראים מקובץ אינם מוצגים במסך בזמן הקלט).

**חובה לצרף להגשה הרצות דוגמה (אחת או יותר)**, המדגימות את השימוש בכל סוגי הפעולות ובכל ששת המטריצות, וכן את הטיפול בכל מגוון השגיאות בקלט.  
**רמז:** מומלץ להכניס פקודת הדפסה של מטריצת התוצאה אחרי כל פעולה, כדי להראות שהתוצאה אכן נכונה (ראו לעיל הדוגמה של סדרת פקודות תקינות).  
**יש להגיש תדפיסי מסך (או קבצי פלט) של כל ההרצות.** במידה ותשתמשו בקבצי קלט, יש להגיש גם קבצים אלה.

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

**בהצלחה!**



# מטלת מנחה (ממ"ן) 23

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 6,7,8

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

סמסטר: 2021א' מועד אחרון להגשה: 10.01.2021

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

יש להגיש תכניות מלאות (בין השאר מכילות main), ניתנות להידור והרצה, ומאפשרות בדיקה של כל התוצאות המגוונות של הריצה ללא צורך בשינויים כלשהם בקוד התוכנית. את המטלה יש להגיש בקובץ zip. לאחר ההגשה יש להוריד את המטלה משרת האו"פ למחשב האישי, ולבדוק שהקבצים אכן הוגשו באופן תקין.

## שאלה 1 (10 נקודות)

בכל סעיף עליכם לכתוב האם הטענה נכונה, לא נכונה, לפעמים נכונה. עליכם לנמק את תשובתכם. תשובה לא מנומקת, גם אם היא נכונה, לא תזכה בנקודות. (כל סעיף 5 נקודות).

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

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

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

## שאלה 2 (90 נקודות) (תכנית ראשית בקובץ `preprocessor.c`)

בתחילת תהליך ההידור (קומפילציה) של קובץ בשפת C מופעל ה- `preprocessor` (קדם-מעבד). שתיים מהפעולות המבוצעות על ידי הקדם-מעבד הן:

(1) הורדת הערות.

(2) העתקת קבצי כותרת לתוך קובץ המקור (הנחיות `#include`).

עליכם לכתוב תכנית שמממשת גרסה מצומצמת ומוגבלת של הקדם-מעבד. התכנית מקבלת, כארגומנט בשורת הפקודה, שם של קובץ מקור בשפת C, ומבצעת על קובץ מקור זה את פעולות הקדם-מעבד שפורטו לעיל. התכנית תפעל בשני שלבים, כדלקמן:

שלב 1: הורדת הערות

קלט: קובץ מקור בשפת C ששמו הועבר כארגומנט בשורת הפקודה. לצורך הדיון כאן, נניח ששם הקובץ הוא `name.c`. בפועל, זה יכול להיות כל שם של קובץ עם הסיומת `“.c”`.  
פלט: קובץ בשם `name.c1` הזהה בתוכנו לקובץ `name.c`, פרט לכך שכל ההערות בקוד המקור נמחקו.

הניחו שההערות כתובות לפי גרסת ANSI, כלומר מתחילות ב- `/*` ומסתיימות ב- `*/`.  
כמו כן, הניחו שאין קינון של הערה בתוך הערה.  
לתשומת לב: הערות יכולות להתפרס על יותר משורה אחת

רמז: תכנית דוגמה להורדת הערות מופיעה במדריך הלמידה בעמ' 167.

שלב 2: טיפול בהנחיות `#include`

קלט: הקובץ `name.c1` (כלומר קובץ C ללא הערות שנוצר בשלב 1).  
פלט: קובץ `name.c2` הזהה בתוכנו לקובץ `name.c1` פרט לכך שכל ההנחיות `#include` כבר טופלו. כלומר, הקובץ `name.c2` מכיל גם את תוכן הקבצים ששמם מופיע בהנחיות `#include`, כאשר התוכן של כל קובץ כזה הועתק במקום השורה בה מופיעה ההנחיה, ואילו ההנחיה עצמה נמחקה.

לשם פשטות, הניחו שאין `include`-ים מקוננים. כלומר, קובץ שהוא ארגומנט של הנחית `#include` אינו מכיל בעצמו הנחיות `#include`. לפיכך, הקדם-מעבד צריך לזהות ולטפל בהנחיות `#include` רק בקובץ `name.c`.

כמו כן, למען פשטות, הניחו שהקבצים שיש לעשות להם `include` נמצאים תחת אותה התיקיה בה נמצא הקובץ `name.c`. אין הנחיות `#include` של קבצים מהספריה הסטנדרטית.

כלומר, התחביר של הנחית `#include` הוא מהצורה: `#include "someFileName.h"`

אפשר להניח שבקבצי ה-`include` אין הערות, כך שאין צורך להפעיל את שלב 1 על קבצים אלה.

### דרישות ודגשים נוספים :

1. על תכנית הקדם-מעבד לקבל ארגומנט יחיד בשורת הפקודה, והוא שם קובץ המקור עמו רוצים לעבוד. אין להשתמש בשם קובץ שמוגדר באופן קבוע בתכנית, או בשם שמוזן לתכנית באמצעות הקלט הסטנדרטי.
2. על התכנית לבדוק כי שם קובץ המקור שהועבר בשורת הפקודה הוא עם הסיומת "c". במקרה שהסיומת אינה נכונה, על התכנית להדפיס הודעת שגיאה מפורטת ולסיים את עבודתה.
3. שמות הקבצים שנוצרים בשלבים השונים של עבודת הקדם-מעבד, צריכים להיות זהים לשם קובץ הקלט, למעט הסיומות (c1, c2. כמוגדר לעיל).
4. ניתן להניח שקובץ הקלט וקבצי ה-include מכילים קוד בשפת C שעומד בדרישות ובהנחות שפורטו לעיל בתיאור שלבי הקדם-מעבד. אין צורך לטפל בשגיאות בתוכן הקבצים.
5. יש להתעלם מכל הנחיה אחרת של הקדם-מעבד בקבצי המקור (למשל: #ifdef, #define וכד').
6. ניתן להניח שבכל קובץ עמו עובדת התכנית, האורך המרבי של שורה הוא 500 תווים. מספר השורות בקובץ אינו מוגבל.
7. במקרה של בעיה בפתיחה או סגירה של קובץ תוך כדי ריצת התוכנית, על התכנית להדפיס הודעת שגיאה מפורטת ולסיים את עבודתה.
8. יש לממש את הקדם-מעבד באופן מודולרי, תוך חלוקה למספר קבצי מקור לפי משימות ולפי ראות עיניכם (למשל, קובץ לכל שלב, קובץ שרותי עזר לקלט/פלט, וכד').

**חובה לצרף להגשה מספר הרצות דוגמה**, לרבות הרצות שמסתיימות בהודעת שגיאה. יש להדגים את כל מגוון האפשרויות של הקדם-מעבד בכל אחד משלבים. **לכל הרצה, יש להגיש את קובץ הקלט, את קבצי ה-include, את הקבצים שנוצרו בשלבי הקדם-מעבד, וכן תדפיס מסך של ההרצה.**

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

**בהצלחה!**



# מטלת מנחה (ממ"ן) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה : פרויקט גמר

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

מועד אחרון להגשה : 14.3.2021 סמסטר : 2021א'

## 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 וכתובה 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 ותייעוד 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 וחסור 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 מ**מעבד** (יע"מ), **אוגרים** (רגיסטרים), **זיכרון** RAM. חלק מהזיכרון משמש כמחסנית (stack).

למעבד 8 אוגרים כלליים, בשמות:  $r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7$ . גודלו של כל אוגר הוא 12 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 11. 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 4096 תאים, בכתובות 0-4095, וכל תא הוא בגודל של 12 סיביות. לתא בזיכרון נקרא גם בשם **"מילה"**. הסיביות בכל מילה ממוספרות כמו באוגר.

מחשב זה עובד רק עם מספרים שלמים חיוביים ושלייליים. 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 שלוש מילים**, בהתאם לסוג הפעולה (ראו פרטים בהמשך).

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

בכל סוגי הוראות המכונה, **המבנה של המילה הראשונה תמיד זהה**. מבנה המילה הראשונה בהוראה הוא כדלהלן:

11	10	9	8	7	6	5	4	3	2	1	0
opcode				funct				מיעון מקור		מיעון יעד	

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), ו**פונקציה** (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

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

להלן מפרט השדות במילה הראשונה בקוד המכונה של כל הוראה.

**סיביות 8-11:** סיביות אלה מכילות את קוד-הפעולה (**opcode**). ישנן מספר פעולות עם קוד פעולה זהה (ראו בטבלה לעיל, קודי-פעולה 2, 5 או 9), ומה שמבדיל ביניהן הוא השדה funct.

**סיביות 4-7:** 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 יכול 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 2-3:** 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 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 למילה הראשונה. לכל אופרנד של הוראה נדרשת **מילת-מידע אחת נוספת**. כאשר בהוראה יש שני אופרנדים, קודם תופיע מילת-המידע של אופרנד המקור, ולאחריה מילת-המידע של אופרנד היעד.

להלן המפרט של שיטות המיעון.



מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	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 12 סיביות	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)	מילת-מידע נוספת של ההוראה מכילה כתובת בזיכרון. המילה בכתובת זו בזיכרון היא האופרנד. הכתובת מיוצגת כמספר <u>ללא סימן</u> ברוחב של 12 סיביות.	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 x מקודדת במילת-המידע הנוספת.  <u>דוגמה נוספת:</u> ההוראה jmp next  מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתבצע נמצאת בכתובת next). הכתובת next מקודדת במילת-המידע הנוספת.
2	מיעון יחסי (relative)	שיטה זו רלוונטית <u>אך ורק</u> להוראות המבצעות קפיצה (הסתעפות) להוראה אחרת. מדובר בהוראות עם קוד-פעולה 9 בלבד: jmp, bne, jsr. <u>לא ניתן</u> להשתמש בשיטה זו בהוראות עם קודי-פעולה אחרים.  בשיטה זו, יש בקידוד ההוראה מילת מידע נוספת, המכילה את מרחק הקפיצה, במילות זיכרון, ממילת-המידע הנוכחית אל המילה הראשונה של ההוראה המבוקשת (ההוראה הבאה לביצוע).  מרחק הקפיצה מיוצג כמספר עם סימן בשיטת המשלים ל-2 ברוחב של 12 סיביות. מרחק זה יהיה שלילי אם הקפיצה היא אל הוראה בכתובת יותר נמוכה, וחיובי אם הקפיצה היא אל הוראה בכתובת יותר גבוהה.	האופרנד מתחיל בתו % ולאחריו ובצמוד אליו מופיע שם של תווית.  התווית מייצגת באופן סימבולי כתובת של <u>הוראה</u> <u>בקובץ המקור הנוכחי של התוכנית</u> .  ייתכן שהתווית כבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת התווית בתחילת שורת הוראה.  יודגש כי בשיטת מיעון יחסי <u>לא ניתן</u> להשתמש בתווית (כתובת) שמוגדרת בקובץ מקור אחר (כתובת חיצונית).	jmp %next  בדוגמה זו, ההוראה jmp מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתבצע נמצאת בכתובת next).  נניח, לדוגמה, כי ההוראה jmp שבדוגמה נמצאת בכתובת 500 (עשרוני). כמו כן, נניח כי התווית next מוגדרת בקובץ המקור הנוכחי בכתובת 300 (עשרוני).  מרחק הקפיצה ממילת-המידע של ההוראה jmp אל הכתובת next הוא $-201 = (500 - 300)$ , ולכן המילה הנוספת תכיל את הערך -201.

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	תחביר האופרנד באסמבלי	דוגמה
3	מיעון אוגר ישיר (register direct)	האופרנד הוא אוגר. מילת-מידע נוספת של ההוראה מכילה בסיביות 0-7 ביט דולק יחיד המייצג את האוגר המתאים. סיבית 0 תדלוק אם מדובר באוגר r0, סיבית 1 תדלוק אם מדובר באוגר r1 וכי'. סיביות 8-11 יהיו תמיד מאופסות	האופרנד הוא שם של אוגר.	<p>clr r1</p> <p>בדוגמה זו, ההוראה clr מאפסת את האוגר r1. המילה הנוספת של ההוראה תכיל (בבינארי) 000000000010</p> <p><u>דוגמה נוספת:</u></p> <p>mov #-1, r2</p> <p>האופרנד השני של ההוראה (אופרנד היעד) נתון בשיטת מיעון אוגר ישיר. ההוראה כותבת את הערך המיידי -1 אל אוגר r2. המילה הנוספת השניה של ההוראה תכיל (בבינארי) 000000000100</p>

#### 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 מאופסים.

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	$PC \leftarrow PC + \text{distanceTo}(\text{Line})$ בשיטת מיעון יחסי, המרחק לתווית Line מתווסף למצביע התוכנית ולפיכך ההוראה הבאה שתתבצע תהיה במען Line.
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 $PC \leftarrow \text{address}(\text{Line})$ מצביע התוכנית יקבל את כתובת התווית Line, ולפיכך ההוראה הבאה שתתבצע תהיה במען Line.
jsr	9	12	קריאה לשגרה (סברוטניה). 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	$\text{push}(PC+2)$ $PC \leftarrow \text{address}(\text{SUBR})$ מצביע התוכנית 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	ידפס 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 משגרה. 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	$PC \leftarrow \text{pop}()$ ההוראה הבאה שתתבצע תהיה זו שאחרי הוראת jsr שקראה לשגרה.
stop	15	עצירת ריצת התוכנית.	stop	התוכנית עוצרת מיידית.

### מבנה תכנית בשפת אסמבלי:

תכנית בשפת אסמבלי בנויה ממשפטים (statements). קובץ מקור בשפת אסמבלי מורכב משורות המכילות משפטים של השפה, כאשר כל משפט מופיע בשורה נפרדת. 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.

### משפט הנחיה:

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.

לתשומת לב: 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 תווית 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

הערה: 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 האסמבלר

#### 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 r3 ,add לא יכולים לשמש כתוויות, אבל הסמלים R3 ,r9 ,Add 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 : 123 , -5 , 76 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, המילה הראשונה בכל הוראה, או מילת-מידע המכילה אופרנד מידי).
- האות R (קיצור של Relocatable) באה לציין שתוכן המילה תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל, מילת-מידע המכילה כתובת של תווית המוגדרת בקובץ המקור הנוכחי).
- 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 מילת-המידע הנוספת של שיטת מיעון ישיר תאופיין על ידי האות R או E (תלוי אם האופרנד בקוד האסמבלי הוא תווית מקומית או סמל חיצוני).

### אסמבלר עם שני מעברים

כאשר מקבל האסמבלר כקלט תוכנית בשפת אסמבלי, עליו לעבור על התוכנית פעמיים. במעבר הראשון, 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.

לדוגמה: האסמבלר מקבל את התוכנית הבאה בשפת אסמבלי:

```

MAIN:      add    r3, LIST
LOOP:      prn    #48
           lea    STR, r6
           inc    r6
           mov    r3, K
           sub    r1, r4
           bne    END
           cmp    val1, #-6
           bne    %END
           dec    K
           jmp    %LOOP
END:        stop
STR:        .string "abcd"
LIST:       .data  6, -9
           .data  -100

.entry K
K:          .data  31
.extern val1

```

קוד המכונה של התוכנית (הוראות ונתונים) נבנה כך שיתאים לטעינה בזיכרון החל ממען 100 (עשרוני).

התרגום של תוכנית תכנית המקור שבדוגמה לקוד בינארי מוצג להלן:

Address (decimal)	Source Code	Explanation	Machine Code (binary)	"A,R,E"
0100	MAIN: add r3, LIST	First word of instruction	001010101101	A
0101		Register r3	000000001000	A
0102		Address of label LIST	000010000101	R
0103	LOOP: prn #48		110100000000	A
0104		Immediate value 48	000000110000	A
0105	lea STR, r6		010000000111	A
0106		Address of label STR	000010000000	R
0107		Register r6	000001000000	A
0108	inc r6		010111000011	A
0109		Register r6	000001000000	A
0110	mov r3, K		000000001101	A
0111		Register r3	000000001000	A
0112		Address of label K	000010001000	R
0113	sub r1, r4		001010111111	A
0114		Register r1	000000000010	A
0115		Register r4	000000010000	A
0116	bne END		100110110001	A
0117		Address of label END	000001111111	R
0118	cmp val1, #-6		000100000100	A
0119		Address of extern label val1	000000000000	E
0120		Immediate value -6	111111111010	A



Address (decimal)	Source Code	Explanation	Machine Code (binary)	"A,R,E"
0121	bne %END		100110110010	A
0122		Distance to label END	000000000101	A
0123	dec K		010111010001	A
0124		Address of label K	000010001000	R
0125	jmp %LOOP		100110100010	A
0126		Distance to label LOOP	111111101001	A
0127	END: stop		111100000000	A
0128	STR: .string "abcd"	Ascii code 'a'	000001100001	A
0129		Ascii code 'b'	000001100010	A
0130		Ascii code 'c'	000001100011	A
0131		Ascii code 'd'	000001100100	A
0132		Ascii code '\0'	000000000000	A
0133	LIST: .data 6, -9	Integer 6	000000000110	A
0134		Integer -9	111111110111	A
0135	.data -100	Integer -100	111110011100	A
0136	K: .data 31	Integer 31	000000011111	A

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 127 (עשרוני), ושהסמל K אמור להיות משויך למען 136, 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 המשוויכים 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 בתוכנית).

Symbol	Value (decimal)	Attributes
MAIN	100	code
LOOP	103	code
END	127	code
STR	128	data
LIST	133	data
K	136	data, entry
val1	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 ישויד 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 משייד 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 המשויד 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 מידי, או אוגר, וכן את הקידוד הבינארי של כל הנתונים (המתקבלים מההנחות string, data).

## 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

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".

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:

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

## 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 ההוראות 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 (code או data), וסוג הנראות של הסמל (entry או external).

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, ומחן 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 יכולו 0. בדומה, אם זוהי הוראה ללא אופרנדים (rts, stop), אזי הסיביות של שיטות המיעון של שני האופרנדים יכולו 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  $IC + 100$  (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).

### בנה את קוד המכונה כך שיתאים לטעינה לזיכרון החל מכתובת 100.

בכל מעבר מתחילים לקרוא את קובץ המקור מהתחלה.

#### מעבר ראשון

1. אתחל  $DC \leftarrow 0, IC \leftarrow 100$ .
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 DC (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 הערך 0, 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 IC (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.
15. שמור את הערכים IC ו-L יחד עם נתוני קוד המכונה של ההוראה.
16. עדכן  $IC \leftarrow IC + L$ , 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 הוספת הערך ICF (ראה הסבר לכך בהמשך).
20. התחל מעבר שני.

#### מעבר שני

1. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-7.
2. האם השדה הראשון בשורה הוא סמל (תווית), דלג עליו.
3. האם זוהי הנחית data או string? 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 כתובת מילת-המידע הרלוונטית לרשימת מילות-מידע שמתייחסות לסמל חיצוני. לפי הצורך, לחישוב הקידוד והכתובות, אפשר להיעזר בערכים IC ו-L של ההוראה, כפי שנשמרו במעבר הראשון. חזור ל-1.
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 קודם, ונציג את הקוד הבינארי שמתקבל במעבר ראשון ובמעבר שני. להלן שוב תכנית הדוגמה.

```

MAIN:      add    r3, LIST
LOOP:      prn    #48
           lea    STR, r6
           inc    r6
           mov    r3, K
           sub    r1, r4
           bne    END
           cmp    val1, #-6
           bne    %END
           dec    K
           jmp    %LOOP
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. 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	Explanation	Machine Code (binary)	"A,R,E"
0100	MAIN: add r3, LIST	First word of instruction	001010101101	A
0101		Register r3	000000001000	A
0102		Address of label LIST	?	?
0103	LOOP: prn #48	Immediate value 48	110100000000	A
0104			000000110000	A
0105	lea STR, r6	Address of label STR	010000000111	A
0106			?	?
0107			000001000000	A
0108	inc r6	Register r6	010111000011	A
0109			000001000000	A
0110	mov r3, K	Register r3	000000001101	A
0111			000000001000	A
0112			?	?
0113	sub r1, r4	Register r1	001010111111	A
0114			000000000010	A
0115			000000010000	A
0116	bne END	Address of label END	100110110001	A
0117			?	?

Address (decimal)	Source Code	Explanation	Machine Code (binary)	"A,R,E"
0118	cmp val1, #-6		000100000100	A
0119		Address of label val1	?	?
0120		Immediate value -6	11111111010	A
0121	bne %END		100110110010	A
0122		Distance to label END	?	A
0123	dec K		010111010001	A
0124		Address of label K	?	?
0125	jmp %LOOP		100110100010	A
0126		Distance to label LOOP	?	A
0127	END: stop		111100000000	A
0128	STR: .string "abcd"	Ascii code 'a'	000001100001	A
0129		Ascii code 'b'	000001100010	A
0130		Ascii code 'c'	000001100011	A
0131		Ascii code 'd'	000001100100	A
0132		Ascii code '\0'	000000000000	A
0133	LIST: .data 6, -9	Integer value 6	000000000110	A
0134		Integer value -9	111111110111	A
0135	.data -100	Integer value -100	111110011100	A
0136	K: .data 31	Integer value 31	000000011111	A

טבלת הסמלים אחרי מעבר ראשון היא :

Symbol	Value (decimal)	Attributes
MAIN	100	code
LOOP	103	code
END	127	code
STR	128	data
LIST	133	data
K	136	data
val1	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	Explanation	Machine Code (binary)	"A,R,E"
0100	MAIN: add r3, LIST	First word of instruction	001010101101	A
0101		Register r3	000000001000	A
0102		Address of label LIST	000010000101	R
0103	LOOP: prn #48		110100000000	A
0104		Immediate value 48	000000110000	A
0105	lea STR, r6		010000000111	A
0106		Address of label STR	000010000000	R
0107		Register r6	000001000000	A
0108	inc r6		010111000011	A
0109		Register r6	000001000000	A
0110	mov r3, K		000000001101	A
0111		Register r3	000000001000	A
0112		Address of label K	000010001000	R



Address (decimal)	Source Code	Explanation	Machine Code (binary)	"A,R,E"
0113	sub r1, r4		001010111111	A
0114		Register r1	000000000010	A
0115		Register r4	000000010000	A
0116	bne END		100110110001	A
0117		Address of label END	000001111111	R
0118	cmp val1, #-6		000100000100	A
0119		Address of extern label val1	000000000000	E
0120		Immediate value -6	111111111010	A
0121	bne %END		100110110010	A
0122		Distance to label END	000000000101	A
0123	dec K		010111010001	A
0124		Address of label K	000010001000	R
0125	jmp %LOOP		100110100010	A
0126		Distance to label LOOP	111111101001	A
0127	END: stop		111100000000	A
0128	STR: .string "abcd"	Ascii code 'a'	000001100001	A
0129		Ascii code 'b'	000001100010	A
0130		Ascii code 'c'	000001100011	A
0131		Ascii code 'd'	000001100100	A
0132		Ascii code '\0'	000000000000	A
0133	LIST: .data 6, -9	Integer value 6	000000000110	A
0134		Integer value -9	111111110111	A
0135	.data -100	Integer value -100	111110011100	A
0136	K: .data 31	Integer value 31	000000011111	A

טבלת הסמלים אחרי מעבר שני היא :

Symbol	Value (decimal)	Attributes
MAIN	100	code
LOOP	103	code
END	127	code
STR	128	data
LIST	133	data
K	136	data, entry
val1	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 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 לא יוצר את קובץ הפלט מסוג externals.  
אם אין בקובץ המקור אף הנחיית entry, האסמבלר לא יוצר 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 ללא ציון הסיומת.

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

```
assembler x y hello
```

תריך 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 ".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, וכן קבצי פלט x.ob, x.ent ו-x.ext ככל שיש הנחיות entry או extern. בקובץ המקור. 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 מופיע רווח אחד. כזכור, במעבר הראשון, בצעד 18, נשמרו הערכים ICF ו-IDF. האורך הכולל של תמונת ההוראות הוא ICF-100, והאורך הכולל של תמונת הנתונים הוא IDF.

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

### פורמט קובץ ה-entries

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

### 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 הכתובת של מילת-המידע (בבסיס עשרוני בארבע ספרות, כולל אפסים מובילים). בין שני השדות בשורה יש רווח אחד. אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.

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

נדגים את הפלט שמייצר האסמבלר עבור קובץ מקור בשם ps.as הנתון להלן.

```
; file ps.as
```

```
.entry LIST
```

```
.extern W
```

```
MAIN:      add    r3, LIST
```

```
LOOP:      prn    #48
```

```
           lea    W, r6
```

```
           inc    r6
```

```
           mov    r3, K
```

```
           sub    r1, r4
```

```
           bne    END
```

```
           cmp    K, #-6
```

```
           bne    %END
```

```
           dec    W
```

```
.entry MAIN
```

```
           jmp    %LOOP
```

```
           add    L3, L3
```

```
END:       stop
```

```
STR:       .string "abcd"
```

```
LIST:      .data   6, -9
```

```
           .data   -100
```

```
K:         .data   31
```

```
.extern L3
```

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

Address (decimal)	Source Code	Explanation	Machine Code (binary)	"A,R,E"
0100	MAIN: add r3, LIST	First word of instruction	001010101101	A
0101		Register r3	000000001000	A
0102		Address of label LIST	000010001000	R
0103	LOOP: prn #48		110100000000	A
0104		Immediate value 48	000000110000	A
0105	lea W, r6		010000000111	A
0106		Address of extern label W	000000000000	E
0107		Register r6	000001000000	A
0108	inc r6		010111000011	A
0109		Register r6	000001000000	A
0110	mov r3, K		000000001101	A
0111		Register r3	000000001000	A
0112		Address of label K	000010001011	R
0113	sub r1, r4		001010111111	A
0114		Register r1	000000000010	A
0115		Register r4	000000010000	A
0116	bne END		100110110001	A
0117		Address of label END	000010000010	R
0118	cmp K, #-6		000100000100	A
0119		Address of label K	000010001011	R
0120		Immediate value -6	11111111010	A
0121	bne %END		100110110010	A
0122		Distance to label END	000000001000	A
0123	dec W		010111010001	A
0124		Address of extern label W	000000000000	E
0125	jmp %LOOP		100110100010	A
0126		Distance to label LOOP	111111101001	A
0127	add L3, L3		001010100101	A
0128		Address of extern label L3	000000000000	E
0129		Address of extern label L3	000000000000	E
0130	END: stop		111100000000	A
0131	STR: .string "abcd"	Ascii code 'a'	000001100001	A
0132		Ascii code 'b'	000001100010	A
0133		Ascii code 'c'	000001100011	A
0134		Ascii code 'd'	000001100100	A
0135		Ascii code '\0'	000000000000	A
0136	LIST: .data 6, -9	Integer value 6	000000000110	A
0137		Integer value -9	111111110111	A
0138	.data -100	Integer value -100	111110011100	A
0139	K: .data 31	Integer value 31	000000011111	A

טבלת הסמלים בגמר המעבר השני היא :

Symbol	Value (decimal)	Attributes
W	0	external
MAIN	100	code, entry
LOOP	103	code
END	130	code
STR	131	data
LIST	136	data, entry
K	139	data
L3	0	external

להלן תוכן קבצי הפלט של הדוגמה.

הקובץ ps.ob

```
31 9
0100 2AD A
0101 008 A
0102 088 R
0103 D00 A
0104 030 A
0105 407 A
0106 000 E
0107 040 A
0108 5C3 A
0109 040 A
0110 00D A
0111 008 A
0112 08B R
0113 2BF A
0114 002 A
0115 010 A
0116 9B1 A
0117 082 R
0118 104 A
0119 08B R
0120 FFA A
0121 9B2 A
0122 008 A
0123 5D1 A
0124 000 E
0125 9A2 A
0126 FE9 A
0127 2A5 A
0128 000 E
0129 000 E
0130 F00 A
0131 061 A
0132 062 A
0133 063 A
0134 064 A
0135 000 A
0136 006 A
0137 FF7 A
0138 F9C A
0139 01F A
```

:הקובץ ps.ent

```
MAIN 0100
LIST 0136
```

:הקובץ ps.ext

```
W 0106
W 0124
L3 0128
L3 0129
```

## סיכום והנחיות כלליות

- 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 לאחסון תמונת קוד המכונה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים), 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, ממשיק המשתמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו לתוכנית האסמבלר כארגומנטים (אחד או יותר) בשורת הפקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גרפיים למיניהם, וכד'.
- יש להקפיד לחלק את מימוש האסמבלר למספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסוגים שונים במודול יחיד. מומלץ לחלק למודולים כגון : מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחבירי של שורה), טבלת הסמלים, מפת הזיכרון, טבלאות קבועות (קודי הפעולה, שיטות המיעון החוקיות לכל פעולה, וכד').
- יש להקפיד ולתעד את המימוש באופן מלא וברור, באמצעות הערות מפורטות בקוד.
- יש לאפשר תווים לבנים עודפים בקובץ הקלט בשפת אסמבלי. למשל, אם בשורת הוראה יש שני אופרנדים המופרדים בפסיק, אזי לפני ואחרי הפסיק מותר שיהיו רווחים וטאבים בכל כמות. בדומה, גם לפני ואחרי שם הפעולה. מותרות גם שורות ריקות. האסמבלר יתעלם מתווים לבנים מיותרים (כלומר ידלג עליהם).
- הקלט (קוד האסמבלי) עלול להכיל שגיאות תחביריות. על האסמבלר לגלות ולדווח על כל השורות השגויות בקלט. אין לעצור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להדפיס למסך הודעות מפורטות ככל הניתן, כדי שאפשר יהיה להבין מה והיכן כל שגיאה. כמובן שאם קובץ קלט מכיל שגיאות, אין טעם להפיק עבורו את קבצי הפלט (ob, ext, ent).

תם ונשלם פרק ההסברים והגדרת הפרויקט.

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

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

לתשומת לבכם : לא תינתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או מחלה ממושכת. במקרים אלו יש לבקש ולקבל אישור מראש מצוות הקורס.

**ב ה צ ל ח ה !**