

האוניברסיטה הפתוחה

20465

מעבדה בתכנות מערכות

חוברת הקורס – סתיו 2018א

כתבה: מיכל אבימור

אוקטובר 2017 – סמסטר סתיו – תשע"ח

פנימי – לא להפצה.

© כל הזכויות שמורות לאוניברסיטה הפתוחה.

תוכן העניינים

א	אל הסטודנט
ב	1. לוח זמנים ופעילויות
ד	2. תיאור המטלות
ו	3. התנאים לקבלת נקודות זכות
1	ממ"ן 11
5	ממ"ן 12
7	ממ"ן 22
13	ממ"ן 23
15	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

הספרייה 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, והמממנים 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 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 (2018/ 20465)

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
1	20.10.2017-17.10.2017	ספר C פרקים 1-2-3	מפגש ראשון	
2	27.10.2017-22.10.2017	ספר C פרקים 1-2-3		
3	3.11.2017-29.10.2017	ספר C פרק 4	מפגש שני	
4	10.11.2017-5.11.2017	ספר C פרק 4		
5	17.11.2017-12.11.2017	ספר C פרק 5	מפגש שלישי	ממ"ן 11 12.11.2017
6	24.11.2017-19.11.2017	ספר C פרק 5		
7	1.12.2017-26.11.2017	ספר C פרק 6	מפגש רביעי	
8	8.12.2017-3.12.2017	ספר C פרק 6		ממ"ן 12 3.12.2017

* 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	15.12.2017-10.12.2017 (ד-ו חנוכה)	ספר C פרק 6		
10	22.12.2017-17.12.2017 (א-ד חנוכה)	ספר C פרק 7	מפגש חמישי	
11	29.12.2017-24.12.2017	ספר C פרק 7		ממ"ן 22 24.12.2017
12	5.1.2018-31.12.2017	ספר C פרק 8 + פרויקט	מפגש שישי	
13	12.1.2018-7.1.2018	פרויקט וחזרה		
14	19.1.2018-14.1.2018	פרויקט וחזרה	מפגש שביעי	ממ"ן 23 14.1.2018
15	29.1.2018-21.1.2018	פרויקט וחזרה		ממ"ן 14** 18.3.2018

מועדי בחינות הגמר יפורסמו בנפרד

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

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

2. תיאור המטלות

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

רוב המטלות בקורס זה 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	31 (ממ"ן חובה)	פרויקט גמר

עליך להגיש במהלך הקורס את כל מטלות החובה. את התשובות לממ"נים יש להגיש באמצעות מערכת המטלות (במקרים מיוחדים ניתן להגיש את המטלות באמצעות הדואר או הגשה ישירה למנחה במפגשי ההנחיה. במקרה כזה יש לתאם את הדבר עם הבודק). יש להגיש את קבצי המקור (.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 יכול:

- 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 לב: חל איסור מוחלט של הכנה משותפת של מטלות או העתקת מטלות. תלמיד שייתפס באחד מאיסורים אלה ייענש בהתאם לנאמר בתקנון המשמעת נספח 1 בדיעון 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.
- ג. ציון סופי בקורס של 60 נקודות לפחות.

לתשומת לבכם!

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

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

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

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

מטלת מנחה (ממ"ן) 11

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 1,2,3

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

מועד אחרון להגשה: 12.11.2017

סמסטר: 2018א'

קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
 - שליחת מטלות באמצעות דואר אלקטרוני, באישור המנחה בלבד
- הסבר מפורט ב"נוהל הגשת מטלות מנחה"

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: **Wall -ansi -pedantic** - נדרש להגיש את קבצי המקור (.c, .h), קבצי ההרצה (את קבצי .o אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי MAKEFILE), קבצי קלט וקבצי פלט (לפי הנחייה במפגש / באתר). נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם קובץ התוכנית הראשית, ללא הסיימת .c. יש להגיש תכניות מלאות (בין השאר מכילות main), הניתנות להידור והרצה ובדיקת הנדרש, ללא תוספות קוד.

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

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

יש לקרוא בספר הלימוד (או ב-Google), את המוסבר לגבי הפונקציות הבאות מהספריה הסטנדרטית:

strcmp

strncmp

strchr

עליכם לכתוב מימוש משלכם לשלש הפונקציות לעיל, כאשר שמות הפונקציות, בהתאמה, יהיו:

my_strcmp

my_strncmp

my_strchr

בנוסף עליכם לכתוב תכנית ראשית (הפונקציה main) המאפשרת קליטת נתונים מהמשתמש דרך הקלט הסטנדרטי, הפעלת הפונקציות שכתבתם על נתוני הקלט, והדפסת התוצאות של הפונקציות.

כל הרצה של התכנית תפעיל אחת מהפונקציות שכתבתם. התכנית תבקש מהמשתמש לבחור את הפונקציה שתופעל, ולאחר מכן תבקש מהמשתמש לספק את הפרמטרים עבור הקריאה לפונקציה. בקשות הקלט יהיו באמצעות הודעות ידידותיות.

לאחר החזרה מהפונקציה, התכנית תדפיס פלט נאה ומפורט, הכולל את שם הפונקציה שהופעלה, ערכי הפרמטרים לפונקציה, וערך התוצאה.

הערות:

- א. במימוש הסטנדרטי לפונקציה `strchr`, הערך המוחזר מן הפונקציה הוא מטיפוס מצביע. במימוש שלכם (`my_strchr`), על הפונקציה להחזיר ערך מטיפוס `int`, שמציין את האינדקס במחרוזת (מערך התווים) שם נמצא התו שמחפשים. אם התו לא נמצא, יוחזר -1.
- ב. ניתן לקרוא את "מילת הקלט" הבאה (כאשר "מילת קלט" מוגדרת כרצף של תווים שתחום משני צדדיו בתווים "לבנים": רווח, `\t`, או `\n`) באמצעות הפונקציה הסטנדרטית `scanf` באופן הבא:
- ```
scanf("%s", charBuffer);
```
- הפרמטר `charBuffer` הוא חוצץ (מערך) של תווים אשר אליו ייכנס הקלט. אפשר להניח כי "מילת קלט" תהיה באורך לכל היותר 127 תווים. זכרו גם שכל מחרוזת בתכנית חייבת להסתיים בתו `\0`.
- כמו כן, ניתן לקרוא בבת אחת שורה שלמה מהקלט באמצעות הפונקציה הסטנדרטית `gets` באופן הבא:
- ```
gets(charBuffer);
```
- אפשר להניח ששורת קלט תהיה באורך לכל היותר 127 תווים (כולל התו `\n` הנקלט גם הוא כחלק מהשורה).
- ג. הניחו כי הקלט של הפרמטרים לפונקציה תמיד תקין. אין צורך לטפל בשגיאות בקלט.

חובה לצרף להגשה מספר הרצות בדיקה, המדגימות את מגוון האפשרויות השונות של תוצאות הפונקציות. יש להגיש תדפיסי מסך של כל ההרצות.

שאלה 2 (תכנית ראשית בקובץ `letters.c`) (50 נקודות)

עליכם לכתוב תכנית הקוראת מהקלט הסטנדרטי טקסט (רצף של תווים) עד EOF (סוף הקלט), ומדפיסה את הטקסט לפלט הסטנדרטי, בשינויים הבאים:

- יש להתחיל הדפסה של כל משפט בשורה חדשה.
- משפט הוא טקסט שמתחיל בתו שאינו לבן, ומסתיים בתו `'.'` (נקודה).
- יש להדפיס שורה ריקה בין כל שני משפטים.
- בתחילת כל משפט, אם התו הראשון הוא אות קטנה, יש להמירו לאות גדולה.
- בטקסט שנמצא בין מרכאות כפולות יש להמיר כל אות קטנה לאות גדולה.
- בכל מקום אחר בטקסט (שלא לפי סעיפים 3-4), יש להמיר כל אות גדולה לאות קטנה.
- אין להדפיס ספרות (התווים `'0'-'9'`) לפלט, אלא לדלג ולעבור לתו הבא.
- תווים לבנים בתוך משפט יודפסו לפלט ללא כל שינוי. תווים לבנים בין משפטים לא יודפסו.

הערות:

- א. אין הגבלה על האורך של משפט.
- ב. בקלט, כל משפט יכול להשתרע על מספר שורות.

- המעבר לשורה חדשה (התו \n) באמצע המשפט יועתק לפלט ללא שינוי.
- ג. התו '.' (נקודה) המופיע בטקסט בין מרכאות כפולות אינו נחשב סוף משפט.
- ד. משפט יכול להיות ריק, כלומר מכיל רק נקודה, ללא טקסט נוסף (למשל, כאשר יש שתי נקודות ברצף). יש להדפיס גם משפט ריק.
- ה. עיינו בהסברים על הפונקציות הסטנדרטיות: isupper, islower, toupper, tolower. פונקציות אלו עשויות להיות שימושיות בתרגום הקלט לפלט.

לדוגמה, בהינתן הקלט הבא (5 שורות):

```
I am a dog. you are a cat. we all are animals.
"I think we need help. Yes we do." please.      NO, NO WE DO NOT NEED HELP.
hello,
This stone is 500000 years old.
This is a sequence: "1,2,3,4"... The end
```

התכנית תדפיס את הפלט הבא (20 שורות, כולל השורות הריקות בין המשפטים):

```
I am a dog.

You are a cat.

We all are animals.

"I THINK WE NEED HELP. YES WE DO." please.

No, no we do not need help.

Hello,
this stone is  years old.

This is a sequence: ",,,".

.

.

The end
```

הקלט לתכנית הוא מ-stdin, ויכול להגיע מהמקלדת או מקובץ (באמצעות redirection בעת הפעלת התכנית). לנוחיותכם, הכינו קובץ קלט נרחב, והשתמשו בו שוב ושוב כדיבוג התכנית.

חובה לצרף להגשה הרצת בדיקה, המדגימה את כל מגוון האפקטים של תרגום הקלט והדפסתו לפלט. יש להגיש קובץ קלט ותדפיס מסך (או קובץ פלט) של ההרצה.

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

בהצלחה!

מטלת מנחה (ממ"ן) 12

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 4,5 ובאופן חלקי 6

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

סמסטר: 2018' מועד אחרון להגשה: 3.12.2017

קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
 - שליחת מטלות באמצעות דואר אלקטרוני, באישור המנחה בלבד
- הסבר מפורט ב"נוהל הגשת מטלות מנחה"

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: **Wall -ansi -pedantic** - נדרש להגיש את קבצי המקור (.c, .h), קבצי ההרצה (את קבצי .o אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי MAKEFILE), קבצי קלט וקבצי פלט (לפי הנחייה במפגש / באתר). נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם קובץ התוכנית הראשית, ללא הסיימת .c. יש להגיש תכניות מלאות (בין השאר מכילות main), ניתנות להידור/הרצה/בדיקת הנדרש, ללא תוספות קוד.

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 magic.c)

ריבוע קסם הוא מטריצה בגודל $N \times N$ המכילה איברים מ-1 עד N^2 (כולל), כך שסכום האיברים זהה בכל שורה, בכל עמודה ובכל אלכסון. לדוגמה, להלן ריבוע קסם עבור $N = 5$, כאשר הסכום של כל שורה, עמודה ואלכסון 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, המקבלת מהקלט הסטנדרטי מספר שלם החיובי N , ואחריו N^2 ערכים שלמים (שהם אברי המטריצה לפי סדר השורות), ומדפיסה לפלט הסטנדרטי את המטריצה בתצוגה דו-מימדית נאה, וכן הודעה האם המטריצה היא ריבוע קסם או לא. יש להקפיד בהדפסה על עימוד נאה ומיושר של אברי המטריצה.

יש לקבוע בתכנית ערך מקסימלי אפשרי ל- N , באמצעות `#define` (לא פחות מ-7).

כל הערכים בקלט הם מטיפוס `int` ובבסיס עשרוני. יש להעביר את כל הנתונים בשורת קלט יחידה. איברי המטריצה יופיעו לפי סדר השורות במטריצה. הערכים בקלט יופרדו זה מזה ברווחים ו/או טאבים (אחד או יותר). יש לבקש את הקלט מהמשתמש באמצעות הודעה ידידותית.

להלן דוגמת קלט של ריבוע קסם בגודל 3×3 .

3 8 1 6 3 5 7 4 9 2

לנוחיותכם, אפשר לשמור את הקלט בקובץ, ולהשתמש בניתוב (redirection) של הקלט הסטנדרטי.

במקרה של שגיאה כלשהי בערכי הקלט, על התכנית להפסיק את עבודתה, ולהדפיס הודעת שגיאה מפורטת. להלן דוגמאות של שגיאות אפשריות:

- ערך לא מתאים עבור N (גדול מהמקסימום המוגדר בתכנית, או קטן מ-3).
- כמות לא נכונה של ערכים עבור המטריצה (יותר או פחות מ- N^2 ערכים).
- ערכי המטריצה אינם בתחום $1-N^2$, או לא כולם שונים זה מזה.
- ערך אינו מטיפוס `int` (לרבות הערך של N).

יש לחלק את התכנית לפונקציות באופן מובנה (מודולרי). כלומר, עליכם להגדיר פונקציות נפרדות לקלט, לבדיקות השונות, לפלט, וכד'. התכנית הראשית (הפונקציה `main`) תרכז את כל הפעילויות של התכנית.

חובה לצרף להגשה מספר הרצות בדיקה על גדלים שונים של N , לרבות הדגמה של הטיפול במגוון השגיאות בקלט. **יש להגיש תדפיסי מסך או קבצי קלט/פלט של הרצות הדוגמה.**

להכנת קלט לבדיקה, ניתן לחולל ריבועי קסם בגדלים שונים בקישור הבא:

<http://www.dcode.fr/magic-square>.

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

בהצלחה!

מטלת מנחה (ממ"ן) 22

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרקים 4,5,6

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

סמסטר: 2018א' מועד אחרון להגשה: 24.12.2017

קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
 - שליחת מטלות באמצעות דואר אלקטרוני, באישור המנחה בלבד
- הסבר מפורט ב"נוהל הגשת מטלות מנחה"

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: **Wall -ansi -pedantic** - נדרש להגיש את קבצי המקור (.c, .h), קבצי ההרצה (את קבצי .o אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי MAKEFILE), קבצי קלט וקבצי פלט (לפי הנחייה במפגש / באתר). נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם קובץ התוכנית הראשית, ללא הסיומת .c. יש להגיש תכניות מלאות (בין השאר מכילות main), ניתנות להידור/הרצה/בדיקת הנדרש, ללא תוספות קוד. 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 "מחשב כיס" אינטראקטיבי לביצוע פעולות חשבוניות על מטריצות.

תזכורת:

הפעולות החשבוניות הבסיסיות על מטריצות הן:

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 :

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

שחלוף (transposition) של מטריצה :

דוגמה :

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

כפל מטריצה בסקלר :

דוגמה :

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

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

עליכם להגדיר, תוך השימוש ב- `typedef`, את הטיפוס `mat` אשר מחזיק מטריצה בגודל 4 על 4. איברי המטריצה הם מספרים ממשיים. על מבנה הנתונים שבחרתם להיות יעיל מבחינת כמות זיכרון הנדרשת, ויעיל מבחינת הגישה אליו.

בנוסף עליכם להגדיר בתכנית הראשית שישה משתנים מטיפוס `mat`, בשמות הבאים :

`MAT_A, MAT_B, MAT_C, MAT_D, MAT_E, MAT_F`

בפקודות שלהלן, כל פרמטר שהוא שם של מטריצה ילקח מתוך ששת המשתנים האלה. בתחילת התכנית, כל המטריצות מאותחלות לאפסים

להלן הפקודות המוגדרות כקלט לתכנית :

1. 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 בפורמט עשרוני.

לדוגמה, הפקודה : `read_mat MAT_A,5,6.25,-7,1,20`. תציב בתא (0,0) במטריצה `MAT_A` את הערך 5, בתא (0,1) את הערך 6.25, בתא (0,2) את הערך -7. בתא (0,3) את הערך 1, ובתא (1,0) את הערך 20. ביתר תאי מטריצה `MAT_A` יוצב הערך 0.

2. שם מטריצה `print_mat`

הפקודה מדפיסה את המטריצה ששמה ניתן בפקודה, בתצוגה דו-מימדית נאה .

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). במידה והחלק השלם של מספר דורש רוחב שדה גדול יותר, מותרת סטיה מהעימוד המיושר. מומלץ להשתמש ביכולות של הפונקציה printf לשלוט במבנה השדה המודפס.

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

5	6.25	-7	1
20	0	0	0
0	0	0	0
0	0	0	0

3. **שם-מטריצה-ג', שם-מטריצה-ב', שם-מטריצה-א' `add_mat`**

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

4. **שם-מטריצה-ג', שם-מטריצה-ב', שם-מטריצה-א' `sub_mat`**

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

5. **שם-מטריצה-ג', שם-מטריצה-ב', שם-מטריצה-א' `mul_mat`**

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

6. **שם-מטריצה ב', ערך-ממשי, שם-מטריצה-א' `mul_scalar`**

הפקודה מכפילה את מטריצה א' בערך ממשי (הפרמטר השני) ומאכסנת את התוצאה במטריצה ב'.

7. **שם מטריצה ב', שם-מטריצה-א' `trans_mat`**

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

8. **`stop`**

הפקודה גורמת לסיום התכנית.

לתשומת לב: אותו שם מטריצה יכול לשמש ביותר מפרמטר 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
```

אופן פעולת התכנית:

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

כל שורת קלט תכיל פקודה אחת מלאה, כולל כל הפרמטרים.

מותרות גם שורות ריקות (המכילות רק תווים לבנים).

בכל פקודה, שם הפקודה מופרד מהפרמטר הראשון באמצעות רווח או טאב, ובין כל שני אופרנדים סמוכים יש פסיק.

לתשומת לב: בשורת הקלט יכולים להיות רווחים וטאבים באופן חופשי ובכמות בלתי מוגבלת, מלבד באמצע מספרים, או באמצע שמות של פקודות או מטריצות. רווחים וטאבים יכולים להיות גם בתחילת השורה ובסופה. על התכנית להתעלם מכל הרווחים והטאבים העודפים.

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). עצירת התכנית שלא באמצעות פקודת stop אינה נחשבת תקינה (גם לא כאשר הקלט מגיע מקובץ), ויש להדפיס הודעת שגיאה על כך ואז לעצור.

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

לפקודה:

```
read_mat MAT_G, 3.2, 8
```

יש להגיב בהודעה כגון:

```
Undefined matrix name
```

לפקודה:

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

יש להגיב בהודעה כגון:

```
Unknown command name
```

לפקודה:

```
read_mat MAT_A, abc, 567
```

יש להגיב בהודעה כגון:

```
Parameter not a real number
```

לפקודה:

```
mul_mat MAT_B, MAT_C
```

יש להגיב בהודעה כגון:

```
Missing parameter
```

לפקודה:

```
trans_mat MAT_B, MAT_C, MAT_D
```

יש להגיב בהודעה כגון:

```
Extraneous text at end of command
```

לפקודה:

```
print_mat, MAT_A
```

יש להגיב בהודעה כגון:

```
Invalid comma after command name
```

לפקודה:

```
trans_mat MAT_A MAT_B
```

יש להגיב בהודעה כגון:

```
Missing comma
```

לפקודה:

```
add_mat MAT_A, , MAT_B, MAT_C
```

יש להגיב בהודעה כגון:

Multiple consecutive commas

לפקודה :

mul_scalar MAT_A, 4.25, MAT_A,

יש להגיב בהודעה כגון :

Extraneous text at end of command

להלן דוגמה של סדרת פקודות שכולן תקינות :

```
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
stop
```

דרישות נוספות :

יש לפצל את התכנית למספר קבצים : mat.c, mymat.c, ו-mat.h.
בקובץ mat.c יש לרכז את הפונקציות החישוביות על מטריות. בקובץ mymat.c תהיה הפונקציה main, וכן כל פעילויות האינטראקציה עם המשתמש, ניתוח הפקודות, והדפסת התוצאות. ההצהרות (אב טיפוס) של הפונקציות מהקובץ mat.c מיוצאות באמצעות הקובץ mat.h, לשימוש הקובץ mymat.c.

באפשרותכם לבנות קבצי מקור נוספים (למשל : קובץ המכיל פונקציות עזר לניתוח הקלט, וכד').

הקלט לתכנית הוא מ-stdin, ויכול להגיע מהמקלדת או מקובץ (באמצעות redirection בעת הפעלת התכנית).

לפני הניתוח של כל שורת קלט, על התכנית להדפיס את השורה לפלט בדיוק כפי שנקראה. זאת כדי שניתן יהיה לראות בפלט את הפקודות המקוריות, גם כאשר הקלט מגיע מקובץ.

חובה לצרף להגשה הרצת בדיקה, המשתמשת בכל סוגי הפקודות, וכן מדגימה את הטיפול במגוון השגיאות השונות. מומלץ להכניס פקודת הדפסה של מטריצת התוצאה אחרי כל פקודה, כדי להראות שהתוצאה אכן נכונה (ראה לעיל הדוגמה של סדרת פקודות תקינות).
יש להגיש קובץ קלט + תדפיס מסך (או קובץ פלט) של הרצת הבדיקה.

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: 2018'א מועד אחרון להגשה: 14.1.2018

קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
 - שליחת מטלות באמצעות דואר אלקטרוני - באישור המנחה בלבד
- הסבר מפורט ב"נוהל הגשת מטלות מנחה"

יש לקמפל עם דגלים מקסימליים, לקבלת כל האזהרות: Wall-ansi-pedantic - נדרש להגיש את קבצי המקור (.c, .h), קבצי ההרצה (את קבצי .o אין צורך לצרף), קבצי הסביבה המתאימים (כולל קבצי MAKEFILE), קבצי קלט וקבצי פלט (לפי הנחייה במפגש / באתר).
נדרש ששם התיקיה ושם הקובץ לריצה יהיו כשם קובץ התוכנית הראשית, ללא הסיומת .c.
יש להגיש תכניות מלאות (בין השאר מכילות main), ניתנות להידור/הרצה/בדיקת הנדרש, ללא תוספות קוד.
את המטלה יש להגיש בקובץ zip. לאחר ההגשה יש להוריד את המטלה משרת האו"פ למחשב האישי, ולבדוק שהקבצים אכן הוגשו באופן תקין.

שאלה 1 (10 נקודות)

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

א. פקודת `exit(1)` ופקודת `return(1)` הן למעשה פקודות שקולות.

ב. איגוד (`union`) ידרוש תמיד, כמות זכרון הזהה לגודל של השדה (`member`) הגדול ביותר שלו.

א. עליכם לכתוב פונקציה בשם var_and המקבלת **מספר בלתי-קבוע** של פרמטרים מטיפוס unsigned int, ומחזירה ערך מטיפוס unsigned int שהוא התוצאה של פעולת and בין הביטים (bitwise and) של כל הפרמטרים האלה.

ב. כתבו תכנית ראשית הקוראת לפונקציה var_and ארבע פעמים, בכל קריאה מספר שונה של פרמטרים, ובערכים מגוונים. התכנית תדפיס בצורה נאה את ערכי הפרמטרים שהועברו לפונקציה, ואת הערך המוחזר מהפונקציה. יש להדפיס את המידע של כל קריאה לפונקציה בשורה נפרדת.

חובה לצרף להגשה תדפיס מסך (או קובץ פלט) של הרצת התכנית.

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

בהצלחה!

מטלת מנחה (ממ"ן) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה : פרויקט גמר

מספר השאלות : 1 משקל המטלה : 31 נקודות

סמסטר : 2018'א מועד אחרון להגשה : 18.3.2018

קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
 - שליחת מטלות באמצעות דואר אלקטרוני - באישור המנחה בלבד
- הסבר מפורט ב"נוהל הגשת מטלות מנחה"**

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

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

עליכם להגיש :

1. קבצי המקור של התוכנית שכתבת (קבצים בעלי סיומת c או h).
2. קבצי הרצה.
3. הגדרת סביבת העבודה (makefile). יש לקמפל עם הדגלים : Wall –ansi –pedantic - ולנפות את כל ההערות שמוציא הקומפיילר, כך שהתכנית תתקמפל ללא הערות.
4. דוגמאות של קבצי קלט, וקבצי הפלט שנוצרו על ידי הפעלת האסמבלר על קבצי קלט אלה.
5. דוגמאות של הפעלת האסמבלר על קבצי קלט המכילים מגוון של שגיאות אסמבלי (ולכן לא נוצרים קבצי פלט). יש לצרף את תדפיסי המסך המראים את הודעות השגיאה שמוציא האסמבלר.

בשל גודל הפרויקט, עליכם לחלק את התוכנית למספר קבצי מקור. יש להקפיד שהקוד הנמצא בתוכניות המקור יעמוד בקריטריונים של בהירות, קריאות וכתובה נכונה.

נזכיר מספר היבטים חשובים :

1. הפשטה של מבני הנתונים : רצוי (במידת האפשר) להפריד בין הגישה למבני הנתונים לבין המימוש של מבני הנתונים. כך, למשל, בעת כתיבת שגרות לטיפול במחסנית, אין זה מעניינם של המשתמשים בשגרות אלה, אם המחסנית ממומשת באמצעות מערך או באמצעות רשימה מקושרת.
2. קריאות הקוד : רצוי להצהיר על הקבועים הרלוונטיים בנפרד, תוך שימוש בפקודת #define, ולהימנע מ"מספרי קסם", שמשמעותם נהירה לכם בלבד.
3. תיעוד : יש להכניס בקבצי המקור תיעוד תמציתי וברור, שיסביר את תפקידה של כל פונקציה ופונקציה. כמו כן יש להסביר את תפקידם של משתנים חשובים. כמו כן, יש להכניס הערות ברמת פירוט טובה בכל הקוד.

הערה: תוכנית "עובדת", דהיינו תוכנית שמבצעת את הדרוש ממנה, אינה ערובה לציון גבוה. כדי לקבל ציון גבוה על התכנית לעמוד בקריטריונים לעיל, אשר משקלם המשותף מגיע עד לכ-40% ממשקל הפרויקט.

הפרויקט כולל כתיבה של תוכנית אסמבלר עבור שפת אסמבלי, שהוגדרה במיוחד עבור פרויקט זה. מותר לעבוד בזוגות. אין לעבוד בצוות גדול יותר משניים. **פרויקטים שיוגשו בשלישיות או יותר לא יבדקו.** חובה ששני סטודנטים, הבוחרים להגיש יחד את הפרויקט, יהיו **שייכים לאותה קבוצה.**

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

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. אופן הפירוק נקבע, באופן חד משמעי, על ידי המיקרו קוד של המעבד.

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

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מספר מסוים של הוראות פשוטות, ולשם כך היא משתמשת בזיכרון המחשב ובאוגרים (registers) הקיימים בתוך היע"מ. דוגמאות: העברת מספר מתא בזיכרון לאוגר ביע"מ או בחזרה, הוספת 1 למספר הנמצא באוגר, בדיקה האם מספר המאוחסן באוגר שווה לאפס. הוראות פשוטות אלה ושילובים שלהן הן המרכיבות את תוכנית המשתמש כפי שהיא נמצאת בזיכרון. כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנת), תתורגם בסופו של דבר באמצעות תוכנה מיוחדת לצורה סופית זו.

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

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

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

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

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

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

המחשב הדמיוני ושפת האסמבלי

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

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

"חומרה":

המחשב בפרויקט מורכב מיע"מ (יחידת עיבוד מרכזית), אוגרים וזיכרון RAM, כאשר חלק מהזיכרון משמש גם כמחסנית (stack). גודלה של מילת זיכרון במחשב הוא 10 סיביות. האריתמטיקה נעשית בשיטת המשלים ל-2 (2's complement). מחשב זה עובד רק עם מספרים שלמים חיוביים ושליילים, אין תמיכה במספרים ממשיים.

אוגרים:

למעבד 8 אוגרים כלליים (r0,r1,r2,r3,r4,r5,r6,r7). גודלו של כל אוגר הוא 10 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 9.

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

גודל הזיכרון הוא 256 תאים, בכתובות 0-255 (בבסיס עשרוני), וכל תא הוא בגודל של 10 סיביות. לתא בזיכרון נקרא גם בשם "מילה". הסיביות בכל מילה ממוספרות בדומה לאוגר, כמפורט לעיל.

קידוד של תווים (characters) נעשה בקוד ascii.

מבנה הוראת מכונה:

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

9 8 7 6	5 4	3 2	1 0
opcode	מיעון אופרנד מקור	מיעון אופרנד יעד	E,R,A

קידוד כל מילה בקוד המכונה ייעשה בבסיס 32 "מוזר" המוגדר כדלקמן:
32 ספרות הבסיס הן: !,@,#,\$,%,^,&,*,<,>,a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v (כאשר ! שקול ל-0, @ ל-1, # ל-2, וכך עד v השקול ל-31). קידוד של מילה ברוחב 10 סיביות בבסיס 32 זה מורכב משתי ספרות (עם ספרות ! מובילות לפי הצורך).

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

שם הפעולה	קוד הפעולה בבסיס דצימלי (10)
mov	0
cmp	1
add	2
sub	3
not	4
clr	5

6	lea
7	inc
8	dec
9	jmp
10	bne
11	red
12	prn
13	jsr
14	rts
15	stop

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

סיביות 0-1 (A,R,E)

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

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

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

סיביות 2-3 מקודדות את מספרה של שיטת המיעון של אופרנד היעד (destination operand).

סיביות 4-5 מקודדות את מספרה של שיטת המיעון של אופרנד המקור (source operand).

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

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

ארבע שיטות המיעון הקיימות במכונה שלנו הן:

ערך	שיטת מיעון	תוכן המילה נוספת	אופן הכתיבה	דוגמא
0	מיעון מידי	המילה הנוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר המיוצג ב-8 סיביות, אליהם מתווספות זוג סיביות של שדה A,R,E	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני	mov #-1,r2 בדוגמה זו האופרנד הראשון של הפקודה נתון בשיטת מיעון מידי. ההוראה כותבת את הערך 1- לתוך אוגר r2
1	מיעון ישיר	המילה הנוספת של ההוראה מכילה מען של מילה בזיכרון. מילה זו בזיכרון הינה האופרנד. המען מיוצג ב-8 סיביות אליהן מתווספות זוג	האופרנד הינו <u>תווית</u> שהוצהרה או תוצאה בהמשך הקובץ. ההצהרה נעשית על ידי כתיבת	נתונה למשל ההגדרה : x: .data 23 אפשר לכתוב הוראה :

dec x	תווית בקובץ המקור (בתחילת הנחית 'data' או 'string'.או 'struct', או בתחילת הוראה של התוכנית), או על ידי אופרנד של הנחית 'extern'.	סיביות של שדה A,R,E		
נתונה למשל ההגדרה : s: .struct 9,"abcd" אפשר לכתוב הוראה : add #4, s.1 בדוגמה זו, ההוראה מוסיפה את הערך 4 לשדה הראשון של הרשומה s (המכיל את המספר 9).	האופרנד מורכב משם של תוית המציינת רשומה (struct), ולאחריה התו נקודה (.). ולאחריה 1 אם פונים לשדה הראשון או 2 אם פונים לשדה השני. ראה בהמשך הסבר על הגדרת רשומה.	בשיטת מיעון זו יש 2 מילות מידע נוספות בקוד ההוראה. המילה הראשונה הנוספת היא כתובת התחלת הרשומה אליה פונים. המילה הנוספת השניה היא מספר השדה המבוקש לכל אחת משתי המילים הנוספות של שיטת המיעון מתווספות זוג סיביות של שדה A,R,E.	מיעון גישה לרשומה	2
mov r1,r2 בדוגמה זו, ההוראה מעתיקה את תוכן אוגר r2 לתוך אוגר r1. בדוגמה זו שני האופרנדים יקודדו למילה נוספת משותפת.	האופרנד הינו שם של אוגר.	האופרנד הוא אוגר. אם האוגר משמש כאופרנד יעד, מילה נוספת של הפקודה תכיל בארבע הסיביות 2-5 את מספרו של האוגר. אם האוגר משמש כאופרנד מקור, הוא יקודד במילה נוספת שתכיל בששת הסיביות 6-9 את מספרו של האוגר. אם בפקודה יש שני אופרנדים ושניהם אוגרים, הם יחלקו מילה נוספת אחת משותפת. כאשר הסיביות 2-5 הן עבור אוגר היעד, והסיביות 6-9 הן עבור אוגר המקור. לייצוג זה מתווספות זוג סיביות של שדה A,R,E. סיביות שאינן בשימוש יכילו 0.	מיעון אוגר ישיר	3

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

אפיון הוראות המכונה:

הוראות המכונה מתחלקות לשלוש קבוצות, לפי מספר האופרנדים הדרוש להן.

קבוצת ההוראות הראשונה:

הוראות הדורשות שני אופרנדים. ההוראות השייכות לקבוצה זו הן:

mov, cmp, add, sub, lea

הפעולה	הסבר הפעולה	דוגמא	הסבר הדוגמא
mov	מבצעת העתקה של האופרנד הראשון, אופרנד המקור (source) אל האופרנד השני, אופרנד היעד (destination) (בהתאם לשיטת המיעון).	mov A, r1	העתק תוכן המשתנה A (המילה שבכתובת A בזכרון) לאוגר r1.
cmp	מבצעת "השוואה" בין שני האופרנדים שלה. אופן ההשוואה: תוכן אופרנד היעד (השני) מופחת מתוכן אופרנד	cmp A, r1	אם תוכן המשתנה A זהה לתוכנו של אוגר r1 אזי דגל האפס, Z, באוגר

המקור (הראשון), ללא שמירת תוצאת החיסור. פעולת החיסור מעדכנת דגל בשם Z ("דגל האפס") באוגר הסטטוס (PSW).		הסטטוס (PSW) יודלק, אחרת הדגל יאופס.	
add	add A, r0	אוגר r0 מקבל את תוצאת החיבור של אוגר A ותוכנו הנוכחי של אוגר r0.	אופרנד היעד (השני) מקבל את תוצאת החיבור של אוגר המקור (הראשון) והיעד (השני).
sub	sub #3, r1	אוגר r1 מקבל את תוצאת החיסור של הערך 3 מתוכנו הנוכחי של האוגר r1.	אופרנד היעד (השני) מקבל את תוצאת החיסור של אוגר המקור (הראשון) מאופרנד היעד (השני).
lea	lea HELLO, r1	המען שמייצגת התווית HELLO מוכנס לאוגר r1.	lea הינו ראשי תיבות של load effective address. פעולה זו מבצעת טעינה של המען בזיכרון המיוצג על ידי התווית שבאופרנד הראשון (המקור), אל אופרנד היעד (האופרנד השני).

קבוצת ההוראות השניה:

הוראות הדורשות אופרנד אחד בלבד. במקרה זה זוג הסיביות 4-5 במילה הראשונה של קידוד ההוראה הן חסרות משמעות, מכיוון שאין אופרנד מקור (אופרנד ראשון) אלא רק אופרנד יעד (שני). לפיכך הסיביות 4-5 יכילו תמיד 0. על קבוצה זו נמנות ההוראות הבאות: not, clr, inc, dec, jmp, bne, red, prn, jsr

פקודה	הסבר פעולה	דוגמא	הסבר דוגמא
not	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0).	not r2	$r2 \leftarrow \text{not } r2$
clr	איפוס תוכן האופרנד.	clr r2	$r2 \leftarrow 0$
inc	הגדלת תוכן האופרנד באחד.	inc r2	$r2 \leftarrow r2 + 1$
dec	הקטנת תוכן האופרנד באחד.	dec C	$C \leftarrow C - 1$
jmp	קפיצה בלתי מותנית אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כלומר, בעת ביצוע ההוראה, מצביע התוכנית (PC) יקבל את ערך אופרנד היעד.	jmp LINE	$PC \leftarrow \text{LINE}$
bne	bne הינו ראשי תיבות של: branch if not equal (to zero). זוהי הוראת הסתעפות מותנית. מצביע התוכנית (PC) יקבל את ערך אופרנד היעד אם ערכו של הדגל Z באוגר הסטטוס (PSW) הינו 0. כזכור, הדגל Z נקבע בפקודת cmp.	bne LINE	אם ערך הדגל Z באוגר הסטטוס (PSW) הינו 0 אזי: $PC \leftarrow \text{LINE}$
red	קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מהקלט יוכנס לאוגר r1.
prn	הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn r1	התו אשר קוד ה-ascii שלו נמצא באוגר r1

יודפס לקלט הסטנדרטי.			
$SP \leftarrow SP - 1$ $stack[SP] \leftarrow PC$ $PC \leftarrow FUNC$	jsr FUNC	קריאה לשגרה (סברוטניה). מצביע התוכנית (PC) הנוכחי נדחף לתוך המחסנית שבזכרון המחשב, והאופרנד מוכנס ל-PC.	jsr

קבוצת ההוראות השלישית:

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

ההוראות השייכות לקבוצה זו הן: stop, rts.

פקודה	הסבר פעולה	דוגמא	הסבר דוגמא
rts	חזרה משיגרה. הערך בראש המחסנית של זמן ריצה מוצא מן המחסנית ומועבר לאוגר התוכנית (PC).	rts	$PC \leftarrow stack[SP]$ $SP \leftarrow SP + 1$
stop	עצירת ריצת התוכנית.	stop	התכנית עוצרת

מספר נקודות נוספות לגבי תיאור שפת האסמבלי:

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

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

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

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

כעת נפרט לגבי סוגי המשפטים השונים.

משפט הנחיה:

משפט הנחיה הוא בעל המבנה הבא:

בתחילתו יכולה להופיע תווית (label) (התווית חייבת להיות בתחביר חוקי. התחביר של תווית חוקית יתואר בהמשך). התווית היא אופציונלית.

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

יש לשים לב: למילות הקוד הנוצרות ממשפט הנחיה לא מצורפות זוג סיביות A,R,E והקידוד ממלא את כל 10 הסיביות של המילה.

ישנם ארבעה סוגים של משפטי הנחיה, והם:

1. 'data'.

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

9, 17, -57, +7 data.

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

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

כלומר אם נכתוב:

XYZ: data 7. -57. +17. 9

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

אם נכתוב בתכנית הוראה לביצוע:

mov XYZ, r1

אזי בזמן ריצת התכנית יוכנס לאוגר r1 הערך 7.

ואילו ההוראה:

lea XYZ, r1

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

2. 'string'.

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

לדוגמא, משפט ההנחיה:

STR: string "abcdef"

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

3. 'struct'

משפט הנחיה זה מקצה רשומה (structure) המורכבת משדות. למשפט הנחיה 'struct'. המבנה הבא:

struct8: struct 8,"xyz"

ברשומה יהיו תמיד בדיוק שני שדות: הראשון מספר, והשני מחרוזת (השדות מקודדים באותו אופן כמו בהנחיות data ו-string בהתאמה).

התויות של ההנחיה struct. מזוהה עם כתובת המילה הראשונה ברשומה. נשים לב שאין תויות נפרדת לכל אחד מהשדות ברשומה.

4. 'entry'

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

לדוגמא, השורות:

	entry	HELLO
HELLO:	add	#1, r1
	

מודיעות שאפשר להתייחס מקובץ אחר לתויות HELLO המוגדרת בקובץ הנוכחי.

לתשומת לב: תויות המוגדרת בתחילת שורת entry. הינה חסרת משמעות והאסמבלר מתעלם מתויות זו (אפשר שהאסמבלר יוציא הודעת אזהרה).

5. 'extern'

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

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

.extern

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 מ :

1. תווית אופציונלית.
2. שם פעולה.
3. 0, 1 או 2 אופרנדים בהתאם לסוג הפעולה.

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

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

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

להוראה בעלת שני אופרנדים המבנה הבא :

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

HELLO: add r7, B

לפקודה בעלת אופרנד אחד המבנה הבא :

אופרנד שם-הפעולה תווית-אופציונלית
לדוגמא :

HELLO: bne XYZ

להוראה ללא אופרנדים המבנה הבא :

שם-הפעולה תווית-אופציונלית
לדוגמא :

END: stop

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

תווית :

תווית חוקית מתחילה באות אלפבית (גדולה או קטנה) ולאחריה סדרה כלשהי של אותיות וספרות, שאורכה קטן או שווה 30 תווים. התווית מסתיימת על ידי התו ' ' (נקודתיים). תו זה

אינו מהווה חלק מהתווית. זהו רק סימן המציין את סופה. התווית חייבת להתחיל בעמודה הראשונה בשורה. אסור שאותה תווית תוגדר יותר מפעם אחת (בשורות שונות).

לדוגמה, התוויות שלהלן הן תוויות חוקיות.

hEllo:

x:

He78902:

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

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

מספר:

מספר חוקי מתחיל בסימן אופציונלי '-' או '+' ולאחריו סדרה כלשהי של ספרות בבסיס עשר. הערך של המספר הוא הערך המיוצג על ידי מחרוזת הספרות והסימן. כך למשל, 76, -5, +123 הינם מספרים חוקיים. אין תמיכה בשלמים בייצוג אחר מאשר עשרוני, ואין תמיכה במספרים ממשיים.

מחרוזת:

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

אסמבלר עם שני מעברים

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

לדוגמא: האסמבלר קורא את קטע הקוד הבא:

```
MAIN:      mov     S1.1,LENGTH
           add     r2,STR
LOOP:      jmp     END
           prn     #-5
           sub     r1,r4
           inc     K

           mov     S1.2,r3
           bne     LOOP
END:       stop
STR:       .string "abcdef"
LENGTH:   .data   6,-9,15
K:         .data   22
S1:       .struct  8,"ab"
```

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

כמו כן, על האסמבלר להחליף את הסמלים S1,K,STR, LENGTH, MAIN, LOOP, END במענים של המקומות בזיכרון שם נמצאים כל נתון או הוראה בהתאמה.

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

לתשומת לב: המקפים המופיעים בקידוד הבינרי הם רק לצורך הדגשת ההפרדה בין השדות השונים בקידוד ונועדו לשם המחשה בלבד.

Label	Decimal Address	Base 32 Address	Operation	Operands	Binary machine code
MAIN:	0100	\$%	mov	S1.1,LENGTH	0000-10-01-00
	0101	\$^		כתובת של הרשומה S1	10000101-10
	0102	\$&		מספר השדה הראשון ברשומה	00000001-00
	0103	\$*		כתובת של LENGTH	10000001-10
	0104	\$<	add	r2, STR	0010-11-01-00
	0105	\$>		קידוד מספר האוגר	0010-0000-00
	0106	\$a		כתובת של STR	01111010-10
LOOP:	0107	\$b	jmp	END	1001-00-01-00
	0108	\$c		כתובת של END	01111001-10
	0109	\$d	prn	#-5	1100-00-00-00
	0110	\$e		המספר -5	11111011-00
	0111	\$f	sub	r1,r4	0011-11-11-00
	0112	\$g		קידודי מספרי האוגרים	0001-0100-00
	0113	\$h	inc	K	0111-00-01-00
	0114	\$i		כתובת של K	10000100-10
	0115	\$j	mov	S1.2,r3	0000-10-11-00
	0116	\$k		כתובת של S1	10000101-10
	0117	\$l		מספר השדה השני ברשומה	00000010-00
	0118	\$m		קידוד מספר האוגר של היעד	0000-0011-00
	0119	\$n	bne	LOOP	1010-00-01-00
	0120	\$o		כתובת של LOOP	01101011-10
END:	0121	\$p	stop		1111-00-00-00
STR:	0122	\$q	.string	"abcdef"	0001100001
	0123	\$r			0001100010
	0124	\$s			0001100011
	0125	\$t			0001100100
	0126	\$u			0001100101
	0127	\$v			0001100110
	0128	%!			0000000000
	0129	%@	.data	6,-9,15	0000000110
LENGTH:	0130	%#			1111110111
	0131	\$\$			0000001111
	0132	%%	.data	22	0000010110
K:	0133	%^	.struct	8, "ab"	0000001000
	0134	%&			0001100001
	0135	%*			0001100010
	0136	%<			0000000000

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

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

למשל, בדוגמא לעיל, האסמבלר אינו יכול לדעת שהסמל END משויך למען 0121 (עשרוני), אלא רק לאחר שהתוכנית נקראה כולה.

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

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

ערך (בבסיס דצימלי)	סמל
100	MAIN
107	LOOP
121	END
122	STR
129	LENGTH
132	K
133	S1

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

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

המעבר הראשון

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

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

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

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

כאשר נתקל האסמבלר בתווית המופיעה בתחילת השורה, הוא יודע שלפניו הגדרה של תווית, ואז הוא משייך לה מען – תוכנו הנוכחי של ה-IC. כך מקבלות כל התוויות את מעניהן בעת ההגדרה.

תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנוסף לשם התווית גם את המען ומאפיינים נוספים. כאשר תהיה התייחסות לתווית באופרנד של הוראה כלשהי, יוכל האסמבלר לשלוף את המען המתאים מטבלת הסמלים.

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

```

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

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

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

המעבר השני

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

הפרדת הוראות ונתונים

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

אחת הסכנות הטמונות באי הפרדת ההוראות מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית בתכנית, לנסות "לבצע" את הנתונים כאילו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום תופעה כזו הסתעפות לא נכונה. התכנית כמובן לא תעבוד נכון, אך לרוב הנזק הוא יותר חמור, כי נוצרת חריגת חומרה ברגע שהמעבד מבצע פעולה שאינה חוקית.

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

גילוי שגיאות בתכנית המקור

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

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

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

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

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

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

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

אלגוריתם שלדי של האסמבלר

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

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

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

בכל מעבר מתחילים לקרוא את קובץ המקור מהתחלה.

מעבר ראשון

1. אתחל $DC \leftarrow 0$, $IC \leftarrow 0$.
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-16.
3. האם השדה הראשון הוא סמל? אם לא, עבור ל-5.
4. הדלק דגל "יש הגדרת סמל".
5. האם זוהי הנחיה לאחסון נתונים, כלומר, האם הנחית data או string או struct? אם לא, עבור ל-8.
6. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם סימון (סמל מסוג data). ערכו יהיה DC. (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).

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

מעבר שני

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

נפעיל אלגוריתם זה על תוכנית הדוגמא שהראינו קודם:

```

MAIN:      mov     S1.1,LENGTH
           add     r2,STR
LOOP:      jmp     END
           prn     #-5
           sub     r1,r4
           inc     K

           mov     S1.2,r3
           bne     LOOP
END:       stop
STR:       .string "abcdef"
LENGTH:   .data    6,-9,15
K:         .data    22
S1:        .struct 8,"ab"

```


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

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

Label	Decimal Address	Base 32 Address	Operation	Operands	Binary machine code
MAIN:	0100	\$%	mov	S1,LENGTH	0000-10-01-00
	0101	\$^		כתובת של רשומה S1	?
	0102	\$&		מספר השדה הראשון ברשומה	00000001-00
	0103	\$*		כתובת של LENGTH	?
	0104	\$<	add	r2, STR	0010-11-01-00
	0105	\$>		קידוד מספר האוגר	0010-0000-00
	0106	\$a		כתובת של STR	?
LOOP:	0107	\$b	jmp	END	1001-00-01-00
	0108	\$c		כתובת של END	?
	0109	\$d	prn	#-5	1100-00-00-00
	0110	\$e		המספר -5	11111011-00
	0111	\$f	sub	r1,r4	0011-11-11-00
	0112	\$g		קידודי מספרי האוגרים	0001-0100-00
	0113	\$h	inc	K	0111-00-01-00
	0114	\$i		כתובת של K	?
	0115	\$j	mov	S1.2,r3	0000-10-11-00
	0116	\$k		כתובת של S1	?
	0117	\$l		מספר השדה השני ברשומה	00000010-00
	0118	\$m		קידוד מספר האוגר של היעד	0000-0011-00
	0119	\$n	bne	LOOP	1010-00-01-00
	0120	\$o		כתובת של LOOP	?
END:	0121	\$p	stop		1111-00-00-00
STR:	0122	\$q	.string	"abcdef"	0001100001
	0123	\$r			0001100010
	0124	\$s			0001100011
	0125	\$t			0001100100
	0126	\$u			0001100101
	0127	\$v			0001100110
	0128	\$!			0000000000
LENGTH:	0129	%@	.data	6,-9,15	0000000110
	0130	%			1111110111
	0131	%			0000001111
K:	0132	%	.data	22	0000010110
S1	0133	%	.struct	8, "ab"	0000001000
	0134	%			0001100001
	0135	%			0001100010
	0136	%			0000000000

טבלת הסמלים :

סמל	ערך (בבסיס דצימאלי)
MAIN	100
LOOP	107
END	121
STR	122
LENGTH	129
K	132
S1	133

נבצע עתה את המעבר השני ונרשום את הקוד בצורתו הסופית :

Label	Decimal Address	Base 32 Address	Operation	Operands	Binary machine code
MAIN:	0100	\$%	mov	S1.1,LENGTH	0000-10-01-00
	0101	\$^		כתובת של רשומה S1	10000101-10
	0102	\$&		מספר השדה הראשון ברשומה	00000001-00
	0103	\$*		כתובת של LENGTH	10000001-10
	0104	\$<	add	r2, STR	0010-11-01-00
	0105	\$>		קידוד מספר האוגר	0010-0000-00
	0106	\$a		כתובת של STR	01111010-10
LOOP:	0107	\$b	jmp	END	1001-00-01-00
	0108	\$c		כתובת של END	01111001-10
	0109	\$d	prn	#-5	1100-00-00-00
	0110	\$e		המספר -5	11111011-00
	0111	\$f	sub	r1,r4	0011-11-11-00
	0112	\$g		קידודי מספרי האוגרים	0001-0100-00
	0113	\$h	inc	K	0111-00-01-00
	0114	\$i		כתובת של K	10000100-10
	0115	\$j	mov	S1.2,r3	0000-10-11-00
	0116	\$k		כתובת של S1	10000101-10
	0117	\$l		מספר השדה השני ברשומה	00000010-00
	0118	\$m		קידוד מספר האוגר של היעד	0000-0011-00
	0119	\$n	bne	LOOP	1010-00-01-00
	0120	\$o		כתובת של LOOP	01101011-10
END:	0121	\$p	stop		1111-00-00-00
STR:	0122	\$q	.string	"abcdef"	0001100001
	0123	\$r			0001100010
	0124	\$s			0001100011
	0125	\$t			0001100100
	0126	\$u			0001100101
	0127	\$v			0001100110
	0128	\$!			0000000000
LENGTH:	0129	\$@	.data	6,-9,15	0000000110
	0130	\$#			1111110111
	0131	\$			0000001111
K:	0132	\$%	.data	22	0000010110
SI	0133	\$^	.struct	8, "ab"	0000001000
	0134	\$&			0001100001
	0135	\$*			0001100010
	0136	\$<			0000000000

לאחר סיום עבודת האסמבלר, קבצי הפלט מועברים לשלבי הקישור והטעינה.
לא נדון כאן באופן עבודת שלבי הקישור/טעינה (כאמור, אלה אינם למימוש בפרויקט זה).
לאחר סיום שלבים אלה, התוכנית תהיה טעונה בזיכרון ומוכנה לריצה.

קבצי קלט ופלט של האסמבלר

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

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

```
assembler x y hello
```

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

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

מבנה כל קובץ פלט יתואר בהמשך.

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

אופן פעולת האסמבלר

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

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

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

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

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

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

2. שורת הוראה:

האסמבלר מוצא מהי הפעולה, ומהן שיטות המיעון של האופרנדים. (מספר האופרנדים אותם הוא מחפש נקבע בהתאם להוראה אותה הוא מצא). האסמבלר קובע לכל אופרנד את ערכו בצורה הבאה:

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

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

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

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

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

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

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

3. שורת הנחיה:

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

I. 'data'.

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

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

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

II. 'struct'.

האסמבלר קורא את השדות של הרשומה, ומקודד כל שדה אל מערך הנתונים, ומקדם את מצביע הנתונים בהתאם.

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

III. 'string'.

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

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

IV. 'entry'.

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

V. 'extern'.

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

יש לשים לב! בהוראה או בהנחיה אפשר להשתמש בשם של סמל אשר ההצהרה עליו ניתנת בהמשך הקובץ (אם באופן ישיר על ידי תווית ואם באופן עקיף על ידי הנחיית extern).

פורמט קובץ ה-object

האסמבלר בונה את תמונת זיכרון המכונה כך שקידוד ההוראה הראשונה מקובץ האסמבלי יכנס למען 100 (בבסיס 10) בזיכרון, קידוד ההוראה השניה יכנס למען העוקב אחרי ההוראה הראשונה (תלוי במספר המילים של ההוראה הראשונה), וכך הלאה עד להוראה האחרונה.

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

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

עקרונית, קובץ object מכיל את תמונת הזיכרון שתוארה כאן. קובץ object מורכב משורות של טקסט. השורה הראשונה מכילה שני מספרים: אורך כולל של קטע ההוראות (במילות זיכרון) ואורך כולל של קטע הנתונים (במילות זיכרון). השורות הבאות מתארות את תוכן הזיכרון. בכל שורה שני מספרים: כתובת של מילה בזיכרון, ותוכן המילה. כל המספרים בקובץ object הם בבסיס 32 "מוזר" שהוגדר לעיל.

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

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

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

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

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

פורמט קובץ ה-entries

קובץ ה-entries בנוי משורות טקסט. כל שורה מכילה שם של סמל שהוגדר כ- entry ואת ערכו, כפי שנמצא בטבלת הסמלים. הערכים מיוצגים בבסיס 32 המוזר (ראה קובץ דוגמא בהמשך).

פורמט קובץ ה-externals

קובץ ה-externals בנוי אף הוא משורות טקסט. כל שורה מכילה שם של סמל שהוגדר external, וכתובת בקוד המכונה בה יש קידוד של אופרנד המתייחס לסמל זה. כמוכן שייתכן ויש מספר כתובות בקוד המכונה בהם מתייחסים לאותו סמל חיצוני. לכל התייחסות כזו תהיה שורה נפרדת בקובץ ה-externals. הכתובות מיוצגות בבסיס 32 המוזר (ראה קובץ דוגמא בהמשך).

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

; file ps.as

```
.entry LOOP
.entry LENGTH
.extern L3
.extern W
MAIN:      mov    S1.1, W
           add    r2,STR
LOOP:      jmp    W
           prn    #-5
           sub    r1, r4
           inc    K

           mov    S1.2, r3
           bne    L3
END:       stop
STR:       .string "abcdef"
LENGTH:   .data  6,-9,15
K:         .data  22
S1:        .struct 8, "ab"
```

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

טבלת הקידוד הבינארי :

Label	Decimal Address	Base 32 Address	Operation	Operands	Binary machine code
MAIN:	0100	\$%	mov	S1.1,W	0000-10-01-00
	0101	\$^		כתובת של S1	10000101-10
	0102	\$&		מספר השדה הראשון ברשומה	00000001-00
	0103	\$*		כתובת של W (סמל חיצוני)	00000000-01
	0104	\$<	add	r2, STR	0010-11-01-00
	0105	\$>		קידוד מספר האוגר	0010-0000-00
	0106	\$a		כתובת של STR	01111010-10
LOOP:	0107	\$b	jmp	W	1001-00-01-00
	0108	\$c		כתובת של W (סמל חיצוני)	00000000-01
	0109	\$d	prn	#-5	1100-00-00-00
	0110	\$e		המספר -5	11111011-00

	0111 0112	\$f \$g	sub	r1,r4 קידודי מספרי האוגרים	0011-11-11-00 0001-0100-00
	0113 0114	\$h \$i	inc	K כתובת של K	0111-00-01-00 10000100-10
	0115 0116 0117 0118	\$j \$k \$l \$m	mov	S1.2,r3 כתובת של S1 מספר השדה השני ברשומה קידוד מספר האוגר של היעד	0000-10-11-00 10000101-10 00000010-00 0000-0011-00
	0119 0120	\$n \$o	bne	L3 כתובת של L3 (סמל חיצוני)	1010-00-01-00 00000000-10
END:	0121	\$p	stop		1111-00-00-00
STR:	0122	\$q	.string	"abcdef"	0001100001
	0123	\$r			0001100010
	0124	\$s			0001100011
	0125	\$t			0001100100
	0126	\$u			0001100101
	0127	\$v			0001100110
	0128	%!			0000000000
LENGTH:	0129 0130 0131	%@ %# %\$.data	6,-9,15	0000000110 1111110111 0000001111
K:	0132	%%	.data	22	0000010110
SI	0133 0134 0135 0136	%^ %& %* %<	.struct	8,"ab"	0000001000 0001100001 0001100010 0000000000

הקובץ ps.ob:

כל תוכן הקובץ מיוצג במספרים בבסיס 32 המוזר.

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

Base32 address Base32 code

```

m f
$%    @%
$^    gm
$&    !%
$*    !@
$<    ^k
$>    %!
$a    fa
$b    i%
$c    !@
$d    m!
$e    vc
$f    *s
$g    #g
$h    e%
$i    gi
$j    @c
$k    gm
$l    !<
$m    !c
$n    k%
$o    !#
$p    u!
$q    $@

```

\$r	\$#
\$s	\$\$
\$t	\$%
\$u	\$^
\$v	\$&
%!	!!
%@	!&
%#	vn
%%\$!f
%%%	!m
%^	!<
%&	\$@
%*	\$#
%<	!!

הקובץ ps.ent:

כל ערכי הסמלים מיוצגים בבסיס 32 המוזר.

LOOP	\$b
LENGTH	%@

הקובץ ps.ext:

כל הכתובות מיוצגות בבסיס 32 המוזר

W	\$*
W	\$c
L3	\$o

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

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

סיכום והנחיות כלליות

- אורך התוכנית, הניתנת כקלט לאסמבלר, אינו ידוע מראש (ואינו קשור לגודל 256 של הזיכרון במעבד הדמיוני). ולכן אורכה של התוכנית המתורגמת, אינו אמור להיות צפוי מראש. אולם בכדי להקל במימוש התוכנית, ניתן להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים לאחסון קוד המכונה. מבני נתונים אחרים בפרויקט (למשל טבלת הסמלים), יש לממשל על פי יעילות/תכונות נדרשות (למשל באמצעות רשימה מקושרת).
- קבצי הפלט של האסמבלר צריכים להיות בפורמטים המופיעים למעלה. שמם של קבצי הפלט צריך להיות תואם לשם קובץ הקלט, פרט לסיומות. למשל, אם קובץ הקלט הוא prog.as אזי קבצי הפלט שיווצרו הם : prog.ob, prog.ext, prog.ent.
- אופן הפעלת האסמבלר צריך להיות תואם לנדרש בממ"ן, ללא שינויים כלשהם. אין להוסיף תפריטים אינטראקטיביים למיניהם, וכד'. הפעלת התוכנית תיעשה רק עם ארגומנטים של שורת פקודה.
- יש להקפיד לחלק את מימוש האסמבלר למספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסוגים שונים במודול יחיד. מומלץ לחלק למודולים כגון : מבני נתונים, מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחבירי של שורה), טבלת סמלים, ועוד.
- יש להקפיד ולתעד את המימוש באופן מלא וברור, באמצעות הערות מפורטות בקוד.
- יש להקפיד להתעלם מרווחים וטאבים, ולהיות סלחנים כלפי תוכנית קלט העושה שימוש ביותר רווחים מהנדרש. למשל, אם בהוראה יש שני אופרנדים המופרדים בפסיק, אזי לפני או אחרי הפסיק יכולים להיות רווחים וטאבים בכל כמות. בדומה, גם לפני ואחרי שם ההוראה. בכל המקרים האלה השורה צריכה להיחשב חוקית (לפחות מבחינת הרווחים).
- במקרה שתוכנית הקלט בשפת אסמבלי מכילה שגיאות תחביריות, נדרש לגלות ולדווח, כמו באסמבלר אמיתי, על כל השגיאות הקיימות, ולא לעצור לאחר גילוי השגיאה הראשונה. כמובן שאם הקלט מכיל שגיאות, אין טעם להפיק קבצי פלט (ob, ext, ent).

תם ונשלם פרק ההסברים והגדרת הפרויקט.

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

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

לתשומת לבכם : לא יתקבלו ממ"נים באיחור ללא סיבה מוצדקת, וללא אישור צוות הקורס.

בהצלחה!!!!