

מטלת מנהה (ממ"ו) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטרת : פרויקט גמר

משקל המטרלה : 61 נקודות (חוובה)

מספר שאלות : 1

מועד אחרון להגשה : 02.04.2025

סמסטר : 2025AA'

קיימות אפשרות אחת להגשת המטרלה :
שליחה באמצעות מטלות המוקونة באתר הבית של הקורס

הסביר מפורט ב"גוזל הגשת מטלות מנהה"

אחד המטרות העיקריות של הקורס "20465 - מעבדה בתכנות מערכות" היא לאפשר לסטודנטים בקורס להתנסות בכתיבת פרויקט תוכנה גדול, אשר יחקה את פעולתה של אחת ממערכות המיצוג והשכיחות.

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

עליכם להגיש את הפריטים הבאים :

1. קבצי המקור של התוכנית שכתבתם (קבצים בעלי הסיומת .c או .h).
2. קובץ הרצה (מקומפל ומקשור) עבור מערכת אוביונטו.
3. קובץ makefile. הקימפול חייב להיות עם הקומפיילר gcc והדגלים : -Wall -ansi -pedantic יש לנפות את כל ההודעות שמודיאה הקומפיילר, כך שהתוכנית תתකפף ללא כל העוריות או אזהרות.
4. דוגמאות הרצה (קלט ופלט):
 - א. קובצי קלט בשפת אסמבלי, קובצי פלט שנוצרו מהפעלת האסמבילר על קבצי קלט אלה. יש להציגו במגוון הפעולות וטיפוסי הנתונים של שפת האסמבלי.
 - ב. קובצי קלט בשפת אסמבלי המציגים מגוון רחב של סוגי שגיאות אסמבלי ולבן נוצרים קבצי פלט, ותזדייסי המסקן המראים את הבעיות השגיאה שמודיאה האסמבילר.

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

נזכיר מספר היבטים חשובים של כתיבת קוד טוב :

1. הפשטה של מבני הנתונים : רצוי (כל האפשר) להפריד בין הגישה למבנה הנתונים לבין המיימוש של מבני הנתונים. כך, למשל, בעת כתיבת פונקציות לטיפול בטבלה, אין זה מעניינים של המשתמשים בפונקציות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקושרת.
2. קריאות הקוד : יש להשתמש בשמות משמעותיים למשתנים ופונקציות. יש לעורך את הקוד באופן מסודר: הוצאות עיקריות, שורות ריקות להפרדה בין קטעי קוד, ועוד'.
3. שימוש : יש להכנס בקבצי המקור תיעוד תמציתי וברור, שיסביר את תפקידה של כל פונקציה (באטען הערות כתורת לכל פונקציה). כמו כן יש להסביר את תפקידם של משתנים חשובים. כמו כן, יש להכנס הערות ברמת פירוט גבוהה בכל הקוד.

הערה: תוכניתת "עובדת", דהיינו תוכניתת שمبرכעת את כל הדריש ממנה, אינה לכשעצמה ערובה לציון גביה. כדי לקובל ציון גביה, על התוכניתת לעמוד בקריטריונים של כתיבה ותיעוד ברמה טוביה, כמוותואר לעיל, אשר משקלם המשותף מגיע עד לכ- 40% משקל הפרויקט.

על המטלה להיות **מקורית לחלווטין**: אין להיעזר בספריות חיצונית מלבד הספריות הסטנדרטיות, ומוגן לא בקוד ולא בחלקיק קוד הנמצאים בראשת, במקור חיצוני וכו'.

מומלץ לעבוד בזוגות. אין לעבוד בצוותים גדולים יותר. **פרויקט שיוגש על ידי שלשה או יותר, לא יבדק 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 עם ראשונה בראצ'ף, לקבלת תמונה כללית לגבי הנדרש, ורק לאחר מכן לקרוא שוב בצורה מעמיקה יותר.

פרק כללי ומטרת הפרויקט

כידוע, קיימות שפות תוכנות רבות, ומספר גדול של תוכניות, כתובות בשפות שונות, שעויות 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 מהוحسن ב>Show, ונראה כמו רצף של ספורות ביןarieות. יחידת העיבוד המרכזית - היע"מ (CPU) - יודעת לפרק את הריצף הזה לקטעים קטנים בעלי משמעות: הוראות, מענים ונתונים.

למעשה, זיכרונו המחשב יכול הוא אוסף של סיביות, שנוהגים לארווןן כמקובצות ליחידות בעלות אורך קבוע (בטים, מילימים). לא ניתן להבחין, בעין שאיתא מייננת, בהבדל פיסי כלשהו בין אותן חלק בזיכרון שבו נמצאת תוכנית לבין שאור הזיכרון.

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מגוון פעולות פשוטות, הנקראות **הוראות מכונה**, ולשם כך היא משתמש ברגיסטרים (registers) הקיימים בתוך היע"מ, ובזיכרון המחשב. **דוגמאות:** העברת מספר מתא בזיכרון לרגיסטר בע"מ או בזיכרון, הוספת 1 למספר הנמצא ברגיסטר, בדיקה האם מספר המואחסן ברגיסטר שווה לאפס, חיבור וחישור בין שני רגיסטרים, ועוד'.

הוראות המכונה ושילובים שליהן הן המרכיבות תוכנית כפי שהיא טעונה לזכרון בזמן ריצתה. כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנת), תתרגם בסופו של דבר באמצעות תוכנה מיוחדת לזרה סופית זו.

היע"מ יודע לבצע קוד שנמצא בפורטט של **שפה מכונה**. זהו רצף של ביטים, המהוויםקידוד בינהר של סדרת הוראות המכונה המרכיבות את התוכנית. קוד זה אינו קריא למשתמש, ולכן לא נכון לקובד (או לזרום) תוכניות ישירות בשפת מכונה. **שפה אסמבלי (assembly language)** היא שפת תוכנות מאפשרת ליציג את הוראות המכונה בזרה סימבולית קלה ונוחה יותר לשכנוע. כמובן שיש צורך לתרגם את הייצוג הסימבולי לקוד בשפת מכונה, כדי שהתוכנית תוכל לרוץ במחשב. תרגום זה נעשה באמצעות כל שנקרא **אסambilר (assembler)**.

כידוע, לכל שפת תוכנות עליית יש מהדר (compiler), או מפרש (interpreter), המתרגם תוכניתת מקור לשפת מכונה. האסambilר משמש בתפקיד דומה עבור שפת אסambilי.

כל מודל של יע"מ (כלומר לכל אירוגון של מחשב) יש שפת מכונה ייעודית משלו, ובהתאם גם שפת אסambilי ייעודית משלו. לפיכך, גם האסambilר (כל התרגומים) הוא ייעודי ושונה לכל יע"מ.

תפקידו של האסambilר הוא לבנות קובץ המכיל קוד מכונה, מקובץ נתון של תוכנית הכתובה בשפת אסambilי. זהו השלב הראשון במסלול אותו עברת התוכנית, עד לקבלת קוד המוכן לריצה על חומרת המחשב. השלבים הבאים הם קישור (linkage) וטעינה (loading), אך בהם לא נסוק במMING זה.

השימושה בפרויקט זה היא לכתוב אסambilר (כלומר תוכנית המתרגם לשפת מכונה), עבור שפת אסambilי שנגידר כאן במיוחד לצורך הפרויקט.

لتשומת לב: בהסברים הכלליים על אופן עבודה תוכנית האסambilר, תהיה מדי פעם התייחסות גם לעובדות שלבי הקישור והטעינה. התייחסויות אלה נועדו על מנת לאפשר לכם להבין את המשך

תהליך העיבוד של הפלט של תוכנת האסמלר. אין לטעות: עליהם ל כתוב את תוכנית האסמלר בלבד. אין ל כתוב את תוכניות הקישור והטיענה!!!

המחשב הדמיוני ושפת האסמלר

נגיד עתה את שפת האסמלר ואת מודל המחשב הדמיוני, עבור פרויקט זה.
הערה: תיאור מודל המחשב להלן הוא חלק בלבד, ככל שנחוץ לביצוע המשימות בפרויקט.

"חומרה":

המחשב בפרויקט מורכב ממעבד CPU (יע"מ - יחידת עיבוד מרכזית), רגיסטרים (אוגרים) ו זיכרון RAM. חלק מהזיכרון משמש גם כמחסנית (stack).

למעבד 8 רגיסטרים כלליים, בשמות: r₀, r₁, r₂, r₃, r₄, r₅, r₆, r₇. גודלו של כל רגיסטר הוא 24 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 23. שמות הרגיסטרים נכתבים תמיד עם אות 'z' קטנה.

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

גודל הזיכרון הוא 2²¹ תאים, בכתובות 1-2²¹-0, וכל תא הוא בגודל של 24 סיביות. לתא בזכרון נקרא גם בשם "מילה". הסיביות בכל מילה ממושפרות כמו ברגיסטר.

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

מבנה הוראת מכונה:

כל הוראת מכונה במודול 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 במספר מיליות זיכרון רצופות, החל מ밀יה אחת ועד למקסימום שלוש מיליטם, בהתאם לשיטת המיעון בה נתון כל אופרנד (ראו פרטים בהמשך).

בקובץ הפלט המכיל את קוד המכונה שבונה האסמלר, כל מילה תקודד בסיס הקסדצימלי (ראו פרטים לגבי קבצי פלט בהמשך).

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

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode				מייעון	מייעון	מקור	מקור	רегистר	רегистר	יעד	יעד	רегистר	רегистר	יעד									

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

להלן טבלת הפעולות:

קוד-הפעולה (בסיס עשרוני)	funct	שם הפעולה
0		mov
1		cmp
2	1	add
2	2	sub
4		lea
5	1	clr
5	2	not
5	3	inc
5	4	dec
9	1	jmp
9	2	bne
9	3	jsr
12		red
13		prn
14		rts
15		stop

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

להלן מפרט הפעולות בamilha הראשונה בקוד המכוна של כל הוראה.

סיביות 18-23: סיביות אלה מכילות את קוד-הפעולה (opcode). ישן מספר פעולות עם קוד פעולה זהה (ראו בטבלה לעיל, קוד-פעולה 2, 5 או 9), ומה שבדיל ביניהם הוא השדה funct.

סיביות 3-7: שדה זה, הנקרא funct, מתפרק כאשר מדובר בפעולת שקוד-הפעולה (opcode) שלה משותף לכמה פעולות שונות (כאמור, קוד-פעולה 2, 5 או 9). השדה funct יכול ערך ייחודי לכל פעולה מקבצת הפעולות שיש להן אותו קוד-פעולה. אם קוד-הפעולה משמש ל פעולה אחת בלבד, הסיביות של השדה funct יהיו מאופסות.

סיביות 16-17: מכילות את מספרה של שיטת המיעון של אופרנד המקור. אם אין בהוראה אופרנד מקור, סיביות אלה יהיו מאופסות. מפרט של שיטות המיעון השונות יינטו בהמשך.

סיביות 13-15: מכילות את מספרו של רגיסטר המקור, במקרה שאופרנד המקור הוא רגיסטר אחריו, סיביות אלה יהיו מאופסות.

סיביות 11-12: מכילות את מספרה של שיטת המיעון של אופרנד היעד. אם אין בהוראה אופרנד יעד, סיביות אלה יהיו מאופסות.

סיביות 8-10: מכילות את מספרו של רגיסטר היעד, במקרה שאופרנד היעד הוא רגיסטר אחריו סיביות אלה יהיו מאופסות.

סיביות 0-2 (השדה 'A,R,E'): אפיון משמעותו של שדה זה בקוד המכוна יובא בהמשך. בamilha הראשונה של כל הוראה, ערך הסיבית A תמיד 1, ושתי הסיביות האחרות (R, E) מאופסות.

لتשומת לב: השדה 'A' מתייחס לכל אחת מהamilims בקידוד ההוראה (ראו המפרט לשיטות המיעון בהמשך).

שיטות מעון:

בשפת האסמבלי שלנו קיימות ארבע שיטות מעון, המסומנות במספרים 0,1,2,3. השימוש בחלק מסוימת המעון מצריך מילוט- מידע נוספת בקוד המכוна של הוראות המכונה, בנוסף למילה הראשונה.

כל אופרנד של הוראה נדרשת **לכל היותר מילת- מידע אחת נוספת**. כאשר בהוראה יש שני אופרנדים הדורשים מילת- מידע נוספת, קודם תופיע מילת- המידע של אופרנד המקור, ולאחריו מילת- המידע של אופרנד השני.

כל מילת- מידע נוספת של הוראה מקודדת באחד משלשה סוגים של קידוד. **סיביות 2-0** של כל מילת- מידע הן השדה 'A', המציין מהו סוג הקידוד של המילה. לכל סוג קידוד יש סיבית נפרדת, שערכה 1 אם מילת- המידע נתונה בסוג קידוד זה, ואחרת ערך הסיבית הוא 0.

- סיבית 2 (הסיבית A) מצינית שקידוד המילה הוא מוחלט (Absolute), ואינו מצריך שינוי בשלבי הקישור והטיענה.
- סיבית 1 (הסיבית R) מצינית שהקידוד הוא של כתובות פנימית הניתנת להזזה (Relocatable), ומצריך שינוי בשלבי הקישור והטיענה.
- סיבית 0 (הסיבית E) מצינית שהקידוד הוא של כתובות חיצונית (External), ומצריך שינוי בשלבי הקישור והטיענה.

הסביר על התפקיד של השדה 'A' בקוד המכונה יבוא בהמשך.
ערך השדה 'A,R,E' הנדרש בכל אחת משיטות המעון מופיע בתיאור שיטות המעון להלן.

מספר	שיטת המעון	תוכן מילת- המידע הנוספת	אופן כתיבת האופרנד	דוגמה
0	מעון מיידי	밀וט- מידע נוספת של הוראה מכילה את האופרנד עצמו, שהוא מספר שלם בשיטת המשלים ל-2, ברוחב של 21 סיביות, השוכן בסיביות 23-3 של המילה. הסיביות 0-2 של מילת המידע הן השדה A,R,E. במעון מיידי, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד מתחילה בתו # ולאחריו ובצמוד אליו מופיע מספרשלם בבסיס עשרוני.	mov #1, r2 בדוגמה זו האופרנד הראשון של הפקודה (אופרנד המקור) נתנו בשיטת מעון מיידי. ההוראה כתובת את הערך 1- אל רגיסטר 2.

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	אופן כתיבת האופרנד	דוגמה
1	מיון ישיר	<p>מיילט-מידע נוספת של הוראה מכילה כתובות בזיכרון. המילה בכתובות זו בזיכרון היא האופרנד.</p> <p>הכתובות מיוצגת כמספר <u>לא סימן</u> ברוחב של 21 סיביות, בסיביות 23-3 של מילת המידע.</p> <p>הסיביות 0-2 ב밀ת המידע הן השדה, A,R,E. במיון ישיר, ערך הסיביות האלה תלוי בסוג הכתובות הרשומה בסיביות 23-3. אם זהה כתובות שמייצגת שורה בקובץ המקור הנוכחי (כתובת פינימית), ערך הסיבית R הוא 1, ושתי הסיביות האחרות מאופסות. ואילו אם זהה כתובות שמייצגת שורה בקובץ מקור אחר של התוכנית (כתובת חיצונית), ערך הסיבית E הוא 1, ושתי הסיביות האחרות מאופסות.</p>	<p>האופרנד הוא תוויות שכבר הוגדרה, או שתוגדר בהמשך הקובץ. הגדירה נעשית על ידי כתיבת התווית בתחילת השורה של הנחית 'data'. או השורה של הנחית 'string', או בתחילת השורה אופרנד של הנחית 'extern'.</p> <p>התווית מייצגת באופן סימבולי כתובות בזיכרון.</p>	<p>x: .data 23 hhoraah: dec x mektina b-1 at tocn hamila shabctotb x zycron (h'mishtna) x. <u>zogmeh nospat:</u> hhoraah jmp next mbutzut kpfiza al shoraah ba mogdrat toviot next (klmr hhoraah baha sttbcu nmcata bctobt next). crtobt next tkod bsibiyot 23-3 sl mil mildu nospat.</p>
2	מיון יחסית	<p>שיטה זו רלוונטית אך ורק להוראות המבצעות קפיצה (הסתעפות) להוראה אחרת.</p> <p>מדובר בקוד-הפעולה הבאים בלבד: <u>jmp</u>, <u>bne</u>, <u>jsr</u>.</p> <p><u>la ntn</u> להשתמש בשיטה זו לא נtran להשתמש בשיטה זו בהוראות עם קוד-הפעולה אחרים.</p> <p>בשיטה זו, יש בקידוד הוראה מילת מידע נוספת המכילה את מרחוב הקפיצה, במילוי זיכרון, מכתובות הוראה הנוכחיות (פקודת הקפיצה) אל כתובות הוראה המבוקשת (ההוראה הבאה לביצוע).</p> <p>מרחוב הקפיצה מיוצג כמספר עם סימן בשיטת המשלים ל-2 ברוחב של 21 סיביות, השוכן בסיביות 23-3 של מילת המידע הנוספת.</p> <p>مרחוב זה יהיה שלילי במקרה שהקפיצה היא אל הוראה שכותבת יותר נמוכה, וחוביי במקרה שהקפיצה היא אל הוראה שכותבת יותר גבוהה.</p> <p>הסיביות 0-2 של מילת המידע הן השדה, A,R,E. במיון יחסית, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.</p>	<p>האופרנד מתחילה ב<u>&</u> ולאחריו ובצמוד אליו מופיע שם של תווית.</p> <p>התווית מייצגת באופן סימבולי כתובות של הוראה <u>בקובץ המקור הנוכחי של התוכנית</u>.</p> <p>יתכן שהתווית כבר הוגדרה, או שתוגדר בהמשך הקובץ. הגדירה נעשית על ידי כתיבת התווית בתחילת שורה.</p> <p>יודגש כי בשיטות מיון יחסית <u>לא ntn</u> להשתמש בטור (כתובות) שוגדרת בקובץ מדור אחר (כתובות חיצונית).</p>	<p>zogmeh zo, hhoraah mbutzut kpfiza al shoraah ba mogdrat tovot next (klmr hhoraah baha sttbcu nmcata bctobt next). nnich ci hhoraah jmp shbdgma nmzta bctobt 500 (usroni). cmo cn, nnich ci tovot next mogdrat bkvz mkor nvcchi bctobt 300 (usroni). mrchek kpfiza al hhoraah bctobt next ho 200, -200, ... tkod bsibiyot 23-3 sl mildu nospat.</p>

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	אופן כתיבת האופרנד	דוגמה
3	מייעון רגיסטר ישיר	הօפרנד הוא רגיסטר. לשיטת מייעון זו אין מילת מידע נוספת. מספרו של הרגיסטר מופיע במליה הראשונה של הוראה, בשדה המתאים: רגיסטר מקור/יעד.	הօפרנד הוא שם של רגיסטר.	clr r1 בדוגמה זו, ההוראה clr נאפסת את תוכן הרגיסטר r1.

מפורט הוראות המכונה:

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

הוראות המכונה מתחלקות לשלווש קבוצות, לפי מספר האופרנדים הדרושים לפעוללה.

קבוצת ההוראות הראשונות:
אלן הן הוראות הדורשות שני אופרנדים.

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

הוֹרָאָה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
mov	0		MOV אופרנד אחד (המקור) אל אופרנד אחר (היעד). המקור (האופרנד הראשון) אל רגיסטר r1. העתק את תוכן המשתנה A (המילה שבכתובת ז' זייכרונו) אל רגיסטר r1.	mov A, r1	
cmp	1		CMP מבצע השוואה בין שני אופרנדים. ערך אופרנד היעד (השני) מופחת מערך אופרנד המקור (הראשון), ללא שבירת תוצאת החישור. פעולה החישור מעדכנת דגל בשם Z ("דגל האפס") ברגיסטר הסטטוס (PSW).	cmp A, r1	אם תוכן המשתנה A זהה לתוכנו של רגיסטר r1 אז הדגל Z ("דגל האפס") ברגיסטר הסטטוס (PSW) יודלק, אחרת הדגל יאפס.
add	2	1	ADD אופרנד היעד (השני) מקבל את תוצאה החיבור של תוכן המשתנה A ותוכנו הנוכחי של r0.	add A, r0	רגיסטר r0 מקבל את תוצאה החיבור של תוכן המשתנה A ותוכנו הנוכחי של r0.
sub	2	2	SUB אופרנד היעד (השני) מקבל את תוצאה החיבור של אופרנד המקור (הראשון) מופרנד היעד (השני).	sub #3, r1	רגיסטר r1 מקבל את תוצאה החיבור של הקבוע 3 מתוכנו הנוכחי של הרגיסטר r1.
lea	4		LOAD EFFECTIVE ADDRESS LEA הוא קיצור (ראשי תיבות) של load effective address. פעולה זו מציבה את המعن זייכרונו. המיצג על ידי התווית שבאופרנד הראשון (המקור), אל אופרנד היעד (האופרנד השני). המعن שמייצגת התווית HELLO מוצב לרגיסטר r1.	lea HELLO, r1	המען שמייצגת התווית HELLO מוצב לרגיסטר r1.

קבוצת ההוראות השניה:

אלו הן ההוראות הדורשות אופרנד אחד בלבד. אופן הקידוד של האופרנד הוא כמו של אופרנד היעד בפקודה עם שני אופרנדים. השודות של אופרנד המקור (סיביות 13-17) בambilת הראשונה בקידוד ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

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

הסבר הדוגמה	דוגמה	הפעולה המתבצעת	funct	opcode	הוראה
הרגיסטר 2 _r מקבל את הערך .0.	clr r2	איפוס תוכן האופרנד	1	5	clr
כל בית ברגיסטר 2 _r מתחפה.	not r2	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולחיפוך 1-0).	2	5	not
תוכן הרגיסטר 2 _r מוגדל ב-1.	inc r2	הגדלת תוכן האופרנד באחד.	3	5	inc
תוכן המשנה Count מוקטן ב-1.	dec Count	הקטנת תוכן האופרנד באחד.	4	5	dec
PC \leftarrow PC+distanceTo(Line) מצביע התוכנית מקבל את המען שמחושב על ידי חיבור המרחק לתוויות Line עם מען ההוראה הנוכחית, ולפיכך ההוראה הבאה שתבוצע תהיה בمعنى Line.	jmp &Line	קפיצה (הסתעפות) בלתי מותנית אל ההוראה שנמצאת בمعنى המיזוג על ידי האופרנד. ככלומר, כתוצאה מביצוע ההוראה, מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה.	1	9	jmp
אם ערך הדגל Z ברגיסטר הסטטוס (PSW) הוא 0, אז PC \leftarrow address(Line) מצביע התוכנית מקבל את כתובת התוויות Line, ולפיכך כתובת התוויות Line נקבע באמצעות הוראת cmp. Line תהיה בمعنى Line.	bne Line	bne הוא קיצור (ראשי תיבות) של branch if not equal (to zero). זוהי הוראות הסטעפות מותנית. אם ערכו של הדגל Z ברגיסטר הסטטוס (PSW) הינו 0, אז מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה. כזכור, הדגל Z נקבע באמצעות הוראת cmp.	2	9	bne
push(PC+2) PC \leftarrow address(SUBR) מצביע התוכנית מקבל את כתובת התוויות SUBR, SUBR, ההוראה הבאה ולפיכך, ההוראה הבאה SUBR שתבוצע תהיה בمعنى SUBR. כתובת החזרה מהשורה נשמרת במחסנית.	jsr SUBR	קריאה לשגרה (סברותית). כתובת ההוראה שאחריה הוראות jsr הנוכחית (PC+2) נדחפת לתוך המחסנית שבויירון (PC) המחשב, ומצביע התוכנית (PC) מקבל את כתובת השגרה. הערה: חוזרת מהשגרה מתבצעת באמצעות הוראת ts, תוך שימוש בכתובת שבמחסנית.	3	9	jsr
קוד ascii של התוenkrai. ממקלט ייכנס לרגיסטר r1.	red r1	קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.		12	red
יודפס לפט התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout) (ascii)r1 הנמצא ברגיסטר r1	prn r1	הדפסת התו הנמצא באופרנד, אל .(stdout)		13	prn

קבוצת הוראות השלישייה:

אלו הן הוראות ללא אופרנדים. קידוד ההוראה מורכב ממילה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד (סיביות 8-17) בambil הראונה של קידוד ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השויות לקבוצה זו הן : rts, stop

הסבר הדוגמה	דוגמה	הפעולה המתבצעת	opcode	הוראה
PC \leftarrow pop()	rts	מתבצעת חזרה משיגרה. הערך שבראש המחסנית של המחשב מצוי מן המחסנית, (PC) ומוכנס למצוין התוכנית (jsr).	14	rts
ההוראה הבאה שתבוצע תהיה זו לאחר הוראת jsr שקרה לשגרה.		<u>הערה :</u> ערך זה נכנס למחסנית בקריאה לשגרה ע"י הוראת jsr		
התוכנית עצרת מיידית.	stop	עצירת ריצת התוכנית.	15	stop

מבנה שפת האסמבלי :

תכנית בשפת אסמבלי בנויה **ממקראים ומשפטים** (statements).

מקראים :

מקראים הם קטעי קוד הכלולים בתוכם משפטיים. בתוכנית ניתן להגדיר מקאו ולהשתמש בו במקומות שונים בתוכנית. השימוש במקאו ממקום מסוים בתוכנית יגרום לפרישת המקאו לאותו מקום.

הגדרת מקאו נעשית באופן הבא : (בדוגמה שם המקאו הוא a_mc)

```
macro a_mc
    inc r2
    mov A,r1
macroend
```

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

```
a_mc
```

```
a_mc
```

בדוגמה זו, השתמשנו פעמיים במקאו cm_a, התוכנית לאחר פרישת המקאו תיראה כך :

```
inc r2
mov A,r1

inc r2
mov A,r1
```

התוכנית לאחר פרישת המאקרו היא התוכנית שהאסטמבלר אמור לתרגם.

הנחות והניחות לגבי מאקרו:

- אין במערכת הגדירות מאקרו מוקנות (אין צורך לבדוק זאת).
- שם של הוראה או הנחה לא יכול להיות שם של מאקרו (יש לבדוק זאת).
- ניתן להניח שלכל שורת מאקרו בקוד המקור קיימת סגירה עם שורת microend (אין צורך לבדוק זאת).
- הגדרת מאקרו תהיה תמיד לפני הקראת למאקרו (אין צורך לבדוק זאת).
- נדרש שהקדם-אסטמבלר ייצור קובץ עם הקוד המורחב הכלול פרישה של המאקרו (הרחבת של קובץ המקור המתואר בהמשך). "קובץ המקור המורחב" הוא "קובץ מקור" לאחר פרישת המאקרו, לעומת "קובץ מקור ראשוני" שהוא קובץ הקלט למערכת, כולל הגדרות המאקרים.

לסיכום, במאקרו יש לבדוק:

- (1) שם המאקרו תקין (איןו שם הוראה וכדומה)
 - (2) בשורת ההגדרה ובשורת הסיום אין תוים נוספים
אם נמצאה שגיאה בשלב פרישת המאקרו - אי אפשר לעبور לשלבים הבאים:
יש לעצור להודיע על השגיאות ולחזור לקובץ המקור הבא (אם קיים).
- הערה: **שגיאות בגוף המאקרו** (אם יש) מגלים בשלבים הבאים.

משפטים:

קובץ מקור בשפט אסטטלי מורכב משורות המכילות משפטיים של השפה, כאשר כל משפט מופיע **בשורה נפרדת**. כלומר, הפרדה בין משפט בקובץ המקור הינה באמצעות התו 'ת' (שורה חדשה).

אורכה של שורה בקובץ המקור הוא 80 תוים לכל היתר (לא כולל התו וו).

יש ארבעה סוגי משפטיים (שורות בקובץ המקור) בשפט אסטטלי, והם :

סוג המשפט	הסביר כלל
משפט ריק	זהי שורה המכילה אך ורק תוים לבנים (whitespace), ככלומר רק את התווים ' ' ו- '\n' (רווחים וטבבים). יתכן ובשורה אין אף תו (למעט התו וו), ככלומר השורה ריקה.
משפט העלה	זהי שורה בה התו הריאן ';' (נקודה פסיק). על האסטטטלי להתעלם לחלווטין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסטטטלי מה עליו לעשות כשהוא פועל על תוכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצתת זיכרון ואותחול משתנים של התכנית, אך הוא אינו מייצר קידוד של הוראות מכוננה המיועדות לביצוע בעת ריצת התכנית.
משפט הוראה	זהו משפט המ指引 קידוד של הוראות מכוננה לביצוע בעת ריצת התכנית. המשפט מורכב ממש של הוראה שעל המעבד לבצע, ותיאור האופרנדים של ההוראה.

cut נפרט יותר לגבי סוגי המשפטים השונים.

משפט הנחיה:

משפט הנחיה הוא בעל המבנה הבא :

בתחלת המשפט יכולה להופיע הגדרה של תווית (label). לתווית יש לחבר חוקי, שיתואר בהמשך. התווית היא אופציינלית.

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

יש לשים לב: לmailto: בקוד המבונה הנוצרות משפט הначיה לא מצורף השדה E,A,R, והקידוד ממלא את כל הסיבות של המילה.

יש ארבעה סוגים של משפטי הначיה, והם:

1. הначיה '.data'

הפרמטרים של הначיה '.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 מקום (או בכלל לא), אולם הפסיק חייב להופיע בין המספרים. כמו כן, אסור שיופיע יותר מפסיק אחד בין שני מספרים, וגם לא פסיק אחרி המספר האחרון או לפני המספר הראשון.

המשפט '.data' מנהה את האSEMBLER להקצאות מקומות בתמונות הנתונים (data image), אשר בו יאותנו הערכים של הפרמטרים, ולאחר מכן מונה הנתונים, בהתאם למספר הערכים. אם בהנחיה .data מוגדרת תווית, אז תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום), ומוכנסת אל טבלת הסמלים. דבר זה מאפשר להתייחס אל מקום מסוים בתמונות הנתונים דרך שם התווית (למעשה, זהה דרך להגדיר שם של משתנה).

כלומר אם נכתב:

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

אז יוקצו בתמונות הנתונים ארבע מילימ' רצופות שיכילו את המספרים שמופיעים בהנחיה. התווית XYZ מזוהה עם כתובות המילה הראשונה.

אם נכתב בתכנית את ההוראה:

mov XYZ, r1

אז בזמן ריצת התכנית יוכנס לרегистר 1ז הערך 7.

ואילו ההוראה:

lea XYZ, r1

תכנס לרегистר 1ז את ערך התווית XYZ (כלומר הכתובת בזיכרונו בה מאוחשן הערך 7).

2. הначיה '.string'

הначיה '.string' פרמטר אחד, שהוא מחזוזות חוקיות. תוווי המחרוזות מקודדים לפי ערכי ASCII המתאימים, ומוכנסים אל תמונות הנתונים לפי סדרם, כל تو במילה נפרד. בסוף המחרוזת יתווסף התו '\0' (הערך המספרי 0), המסמן את סוף המחרוזות. מונה הנתונים של האSEMBLER יקודם בהתאם לאורך המחרוזות (בתווסף מקום אחד עבור התו המסיים). אם בשורת ההנחיה מוגדרת תווית, אז תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום) ומוכנסת אל טבלת הסמלים, בדומה למה שנעשה עבור '.data'. (כלומר ערך התווית יהיה הכתובת בזיכרונו שבמהילה המחרוזת).

לדוגמה, ההנחיה:

STR: .string "abcdef"

מקצת בתמונה הנתונים רצף של 7 מילימט, ומאחרות את המילים לקוד ascii של התווים לפי הסדר במחרוזת, ולאחריהםערך 0 לסיומו סוף המחרוזת. התוויות STR מזוחה עם כתובות התחלה המחרוזת.

3. הנקיה 'entry'.

הנקיה 'entry', פרמטר והוא שם של תווית המוגדרת בקובץ המקור הנוכחי (כלומר תווית שמקבלת את ערכה בקובץ זה). מטרת הנקיה entry. היא לאפין את התוויות זהו באופן שיאפשר לקוד אסמליה הנמצא בקבצי מקור אחרים לשימוש בה (כօפרנד של הוראה).

לדוגמה, השורות:

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

מודיעות לאסמלר שאפשר להתייחס בקובץ אחר לתוויות HELLO המוגדרת בקובץ הנוכחי.

לשומות לב: תווית המוגדרת בתחילת שורת entry. הינה חסרת משמעות והאסמבלר **מתעלם** מתוויות זו (אפשר שהאסמבלר יוציא הודעה אזהרה).

4. הנקיה 'extern'.

הנקיה 'extern', פרמטר והוא שם של תווית שאינה מוגדרת בקובץ המקור הנוכחי. מטרת ההוראה היא להודיע לאסמלר כי התוויות מוגדרת בקובץ מקור אחר, וכי קוד האסמליה בקובץ הנוכחי עשויה בתוויות שימוש.

נשים לב כי הנקיה זו תואמת להנקיה 'entry', המופיעה בקובץ בו מוגדרת התווית. בשלב הקישור תבוצע התאמה בין ערך התווית, כפי שנקבע בקוד המכונה של הקובץ שהגדיר את התווית, לבין קידוד ההוראות המשמשות בתוויות בקבצים אחרים (שלב הקישור אינו רלוונטי למינ'ן זה).

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

```
.extern    HELLO
```

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

לשומות לב: תווית המוגדרת בתחילת שורת extern. הינה חסרת משמעות והאסמבלר **מתעלם** מתוויות זו (אפשר שהאסמבלר יוציא הודעה אזהרה).

משפט הוראה:

משפט הוראה מורכב ממחולקים הבאים:

1. **תוויות אופציונליות.**
2. **שם הפעולה.**
3. **אופרדים,** בהתאם לסוג הפעולה (בין 0 ל-2 אופרדים).

אם מוגדרת תווית בשורת ההוראה, אז היא תוכנס אל טבלת הסמלים. ערך התווית יהיה מען המילה הראשונה של ההוראה בתוך תMOVת 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 האופrnd הראשון באמצעות רווחים ו/או טאים (אחד או יותר).

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזה בטעו ' ' (פסיק). בדומה להנחיה 'data.'.
לא חיבת להיות הצמדה של האופרנדים לפסיק. כל כמות של רווחים ו/או טאים 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 אופrnd אחד המבנה הבא :

label: opcode target-operand

לדוגמה :

HELLO: bne &XYZ

למשפט הוראה ללא אופרנדים המבנה הבא :

label: opcode

לדוגמה :

END: stop

אפיון השדות במשפטים של שפת האסמבלי

תווית:
בתיאור שיטות המיעון לעלה הסברנו כי תווית היא ייצוג סימבולי של כתובות בזיכרון. נרჩיב כאן את ההסבר :
תווית היא למעשה סמל שמודדר בתחילת המשפט הוראה, או בתחילת הנחיה .data.string. או .string.
תווית חוקית מתחילה באות אלפביתית (גדולה או קטנה), ולאחריה סדרה של תוויות אלאותיות אלפביתיות (גדולות או קטנות) ו/או ספרות. האורך המקסימלי של תווית הוא 31 תווים.

הגדרה של תווית מסתויימת בתו ' : (נקודתיים). تو זה אינו מהווה חלק מהתווית, אלא רק סימן המציין את סוף ההגדרה. התו ' חייב להיות צמוד לתווית (לא רווחים).

אסור שאותה תווית תוגדר יותר מפעם אחת (כموון בשורות שונות). אותיות קטנות וגדולות נחשבות שונות זו מזו.

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hEllo:

x:

He78902:

لتשומת לב : מילים שמורות של שפת האסמבלי (כלומר שם של פעולה או הנחיה, או שם של רגיסטר) אינן יכולות לשמש גם כשם של תווית. כמו כן, אסור שאותו סמל ישמש הן כתווית והן כשם של מקרו (יש לבדוק זאת).

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

لتשומת לב: מותר במשפט הוראה להשתמש באופrnd שהוא סמל שאינו מוגדר כתווית בקובץ הנקחי, כל עוד הסמל מאופיין כחיצוני (באמצעות הנחיה `extern`. `extern` בקובץ הנקחי).

מספר:

מספר חוקי מתחילה בסימן אופציונלי: ‘-’, או ‘+’, ולאחריו סדרה כלשהי של ספרות בסיס עשרוני. דוגמה: `-123`, `+5`, `76` הם מספרים חוקיים. אין תמיכה בשפת האסמבלי שלוו ביצוג בסיס אחר מאשר עשרוני, ואין תמיכה במספרים שאינם שלמים.

מחuzeות:

מחuzeות חוקית היא סדרת תווי ascii נראים (שניתנים להדפסה), המוקפים במרכאות כפولات (המרכאות אין נחותות חלק מהמחuzeות). דוגמה למחuzeות חוקית: “hello world”.

סימנו המילים בקוד המכונה באמצעות המאפיין “A,R,E”

בכל מילה בקוד המכונה של הוראה (לא של נתוני), האסמבלי מכניס מידע עבור תהליך הקישור והטעינה. זה השדה A,R,E. המידע ישמש לתיקונים בקוד בכל פעם שיתטען לזכור לצורך הרצה. האסמבלי בונה מלכתחילה קוד שמיועד לטעינה החל מכתובת ההתחלת. התיקונים יאפשרו לטען את הקוד בכל פעם למקום אחר, בלי צורך לחזור על תהליך האסמבלי.

שלוש הסיבות בשדה E,A,R, יכלו ערכים ביינריים כפי שהוסבר בתיאור שיטות המיעון. המשמעות של כל ערך מפורטת להלן.

האות ‘A’ (קייזר של absolute) באה לציין שתוכן המילה אינו תלוי במקומות בזיכרון בו ייטען בפועל קוד המכונה של התכנית בעת ביצועה (למשל מילה המכילה אופrnd מיידי).

האות ‘R’ (קייזר של relocatable) באה לציין שתוכן המילה תלוי במקומות בזיכרון בו ייטען בפועל קוד המכונה של התכנית בעת ביצעה (למשל מילה המכילה כתובת של תווית המוגדרת בקובץ המקור).

האות ‘E’ (קייזר של external) באה לציין שתוכן המילה תלוי בערכו של סמל חיצוני (external) (למשל מילה המכילה כתובת של תווית חיצונית, ככלmr תווית שאינה מוגדרת בקובץ המקור).

כאשר האסמבלי מקבל קולט תוכנית בשפת אסמבלי, עליו לטפל תחילת הפרישת המאקרוואים, ורק לאחר מכן לעבור על התוכנית אליה נפרשوا המאקרוואים. כלומר, פרישת המאקרוואים תעשה בשלב “קדם אסמבלי”, לפני שלב האסמבלי (המתוואר בהמשך).

אם התכנית אינה מכילה מאקרו, תוכנית הפרישה תהיה לתוכנית המקור.

דוגמה לשלב קדם אסמבלי. האסמבלי מקבל את התוכנית הבאה בשפת אסמבלי :

```

MAIN:    add    r3, LIST
LOOP:    prn    #48
          mcrro a_mc
          cmp    K, #-6
          bne    &END
          mcrroend
          lea    STR, r6
          inc    r6
          mov    r3, K
          sub    r1, r4
          bne    END
          a_mc
          dec    K
          jmp    &LOOP
END:     stop
STR:     .string "abcd"
LIST:    .data   6, -9
          .data   -100
K:       .data   31

```

תחילה האסמבולר עבר על התוכנית ופורש את כל המאקרואים הקיימים בה. רק אם תהליך זה מסתיים בהצלחה, ניתן לעבورو לשלב הבא. אחרת, יש להציג את השגיאות ולא ליצר קבצים. בדוגמה זו, התוכנית לאחר פרישת המאקרוו תיראה כך:

```

MAIN:    add    r3, LIST
LOOP:    prn    #48
          lea    STR, r6
          inc    r6
          mov    r3, K
          sub    r1, r4
          bne    END
          cmp    K, #-6
          bne    &END
          dec    K
          jmp    &LOOP
END:     stop
STR:     .string "abcd"
LIST:    .data   6, -9
          .data   -100
K:       .data   31

```

קוד התוכנית, לאחר הפרישה, ישמור בקובץ חדש, כפי שיווסף בהמשך.

אלגוריתם שלי של קדם האסמבול

נציג להלן אלגוריתם שלי לתהיליך קדם האסמבול. لتשומת לב: אין חובה להשתמש דווקא באלגוריתם זה:

1. קרא את השורה הבאה מקובץ המקור. אם נגמר הקובץ, עבור ל- 9 (סיום).
2. האם השדה הראשון הוא שם מקורי המופיע בטבלת המאקרו (כגון a_mc)? אם כן, החלף את שם המאקרו והעתק במקומו את כל השורות המתאימות מהטבלה לקובץ, חזרו ל- 1. אחרת, המשך.
3. האם השדה הראשון הוא "macro" (התחלת הגדרת מאקרו)? אם לא, עבור ל- 6.
4. הדלק דגל "יש macro".
5. (קיימת הגדרת מאקרו) הכנס לטבלת שורות מקורי את שם המאקרו (לדוגמא a_mc).
6. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 9 (סיום).
7. אם דגל "יש macro" דולק ולא זזהה תווית **macroend** הכנס את השורה לטבלת המאקרו ומחק את השורה הניל' מקובץ. אחרת (לא מאקרו) חזרו ל- 1.
8. כבاه דגל "יש macro". חזרו ל- 1. (סיום שמירת הגדרת מאקרו).
9. סיום: שמירת קובץ מקורי פרוש.

אסמבול עם שני מעברים

במעבר הראשון של האסמבול, יש לזיהות את הסמלים (תוויות) המופיעים בתוכנית, ולתת לכל סמל ערך מסווני שהוא המعنוי בזיכרון שהSAMPLE מייצג. במעבר השני, באמצעות ערכי הסמלים, וכן קוד-הפעולה ומספריו הרגיסטריים, בונים את קוד המכונה.

עליו להחליף את שמות הפעולות בקוד הבינארי השקול להם במודל המחשב שהגדנו.

כמו כן, על האסמבול להחליף את כל הסמלים (למשל LIST, MAIN) במשמעותם של המקבילות בזיכרון שם נמצאים כל נתון או הוראה בהתאם.

נניח שקטע הקוד לעיל (הוראות ונתונים) ייטע בזיכרון החל ממען 100 (בסיס 10). במקרה זה נקבל את ה"תרגומים" הבא:

Address (decimal)	Source Code	Explanation	Binary Machine Code
0000100	MAIN: add r3, LIST	First word of instruction Address of label LIST	000010110110100000001100 00000000000000111111010
0000102	LOOP: prn #48		00110100000000000000100 00000000000000110000100
0000104	lea STR, r6		00010001000111000000100 000000000000001111010010
0000106	inc r6		000101000001110000011100
0000107	mov r3, K		00000011011010000000100 0000000000000010000010010
0000109	sub r1, r4		00001011001110000010100
0000110	bne END		001001000000100000010100 000000000000001111001010
0000112	cmp K, #-6		00000101000000000000100 0000000000000010000010010 11111111111111111111010100
0000115	bne &END		001001000000100000010100 0000000000000000110100
0000117	dec K		0001010000001000000100100 000000000000000010000010010
0000119	jmp &LOOP		001001000000100000001100 1111111111111111111101111100
0000121	END: stop		00111100000000000000000000000000100
0000122	STR: .string "abcd"	Ascii code 'a'	000000000000000000001100001
0000123		Ascii code 'b'	0000000000000000000011000010
0000124		Ascii code 'c'	0000000000000000000011000011
0000125		Ascii code 'd'	0000000000000000000011000100
0000126		Ascii code '\0'	00000000000000000000000000000000
0000127	LIST: .data 6, -9	Integer 6 Integer -9	00000000000000000000000000000000110 1111111111111111111111110111110111
0000128	.data -100	Integer -100	1111111111111111111111110011100
0000130	K: .data 31	Integer 31	0000000000000000000000000000000011111

האסמבלר מוחזק טבלה שבה רשומים כל שמות הפעולה של הוראות והקודים הבינאריים המותאים להם, וכן שמות הפעולות ניתנים להמרה לבינארי בקלות. כאשר נקרא שם פעולה, אפשר פשוט לעיין בטבלה ולמצוא את הקוד הבינארי.

כדי לבצע הממרה לבינארי של אופרנדים שכותבים בשיטות מייען המשמשות בסמלים (תוויות), יש צורך לבנות טבלה המכילה את ערכיו כל הסמלים. אולם בהבדל מהקודים של הפעולות, הידועים מראש, הרוי המעניינים בזיכרון עבור הסמלים בשימוש התוכנית אינם ידועים, עד אשר תוכנית המקור נסרקה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד לעיל, האסמבלר אינו יכול לדעת שהסמל END משוויך למן 121 (עשרהוני), ושהסמל K משוויך למן 130, אלא רק לאחר שנקרוו כל שורות התוכנית.

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

במעבר הראשון נבנית טבלת סמלים בזיכרון, ובها לכל סמל שבתוכנית המקור משוויך ערך מספרי, שהוא מען בזיכרון. בדוגמה לעיל, טבלת הסמלים לאחר מעבר ראשון היא:

סמל	ערך (בסיס עשרוני)
MAIN	100
LOOP	102
END	121
STR	122
LIST	127
K	130

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

لتשומת לב : תפקיד האסטブル, על שני המערבים שלו, לתרגם קובץ מקור לקוד בשפת מוכנה. בגמר פעולה האסטブル, התכנית טרם מוכנה לטעינה לזכרון לצורך ביצוע. קוד המוכנה חייב לעבור לשבי הקשר/טעינה, ורק לאחר מכן לשלב הביצוע (שלבים אלה **אינם** חלק מהממי"ן).

המעבר הראשון

במעבר הראשון נדרשים כללים כדי לקבוע איזה מען ישוק לכל סמל. העיקרונו הבסיסי הוא לספור את המיקומות בזיכרון, אותן תופסות ההוראות. אם כל הוראה תיתען בזיכרון למקום העוקב להוראה הקודמת, תציין ספירה כזאת את מען ההוראה הבאה. הספירה נמשכת על ידי האסטブル ומוחזקת במונה ההוראות (IC) . ערכו ההתחלתי של IC הוא 100 (עשרוני), ולכן קוד המוכנה של ההוראה הראשונה נבנה כך שייתען אליו חלול מען 100. ה-IC מתעדכן בכל שורת הוראה המקצתה מוקם בזיכרון. לאחר שהאסטブル קובע מהו אורך ההוראה, ה-IC מוגדל במספר התאים (מילימטרים) הנתפסים על ידי ההוראה, וכך הוא מביע על התא הפניו הבא.

כאמור, כדי לקודד את ההוראות בשפת מוכנה, מחזיק האסטブル טבלה, שיש בה קוד מותאים לכל שם פעולה. בזמן התרגום מחליף האסטブル כל שם פעולה בקוד שלו, וכן כל אופrnd מוחלף בקידוד מותאים, אך פעולה החלפה זו אינה כה פשוטה. ההוראות משתמשות בשיטות מעוון מגוונות לאופrndים. אותה פעולה יכולה לקבל שימושויות שונות, בכל אחת משיטות המיעון, וכן תאיימו לה קידודים שונים לפי שיטות המיעון. לדוגמה, פעולות ההזהה בסות' יכולה להתיחס להעתקת תוכן תא זיכרון לרגיסטר, או להעתקת תוכן רגיסטר לרגיסטר אחר, וכן הלאה. ככל אפשרות כזו של בסות' עשוי להתאים קידוד שונה.

על האסטブル לסרוק את שורת ההוראה בשלמותה, ולהחיליט לגבי הקידוד לפי האופrndים. בדרך כלל מתחלק הקידוד לשדה של שם הפעולה, וסדרות נוספים המכילים מידע לגבי שיטות המיעון. כל השדות ביחד דורשים מילה אחת או יותר בקוד המוכנה.

כאשר נתקל האסטブル בתווית המופיעעה בתחילת השורה, הוא יודע שלפנוי הגדרה של תווית, ואז הוא משייך לה מען – תוכנו הnockי של ה-IC. כך מקבלות כל התוויות את מענייה בעת ההגדירה. תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנותר לשם התווית גם את המعن ומאפיינים נוספים. כאשר תהיה התייחסות לתווית באופrnd של הוראה כלשהי, יוכל האסטブル לשולב את המعن המותאים מטבלת הסמלים.

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

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

כאשר מגיע האסטמבלר לשורה היחסתית (A bne), הוא טרם נתקל בהגדרת התווית A וכמוון לא יודע את המשמעות להוויה. לכן האסטמבלר לא יכול לבנות את הקידוד הבינאי של האופrnd A. נראה בהמשך כיצד נפתרת בעיה זו.

בכל מקרה, תמיד אפשר לבנות מעבר הראשון את הקידוד הבינאי המלא של המילה הראשונה של כל הוראה, את הקידוד הבינאי של מילת-המידע הנוספת של אופrnd מיידי, או רגיסטר, וכן את הקידוד הבינאי של כל הנתונים (המתקבלים מהנהניות .data,.string).

המעבר השני

ראינו שבמעבר הראשון, האסטמבלר אינו יכול לבנות את קוד המכמה של אופrndים המשתמשים בסמלים שעדין לא הוגדרו. רק לאחר שהאסטמבלר עבר על כל התכנית, כך שכל הסמלים נכנסו כבר לטבלת הסמלים, יוכל האסטמבלר להשלים את קוד המכמה של כל האופrndים.

לשם כך מבצע האסטמבלר מעבר נוסף (מעבר שני) על כל קובץ המקור, ומעדכו את קוד המכמה של האופrndים המשתמשים בסמלים, באמצעות ערכי הטבלת הסמלים. בסוף המעבר השני, תהיה התוכנית מתורגמת בשמלהוּתְהָ לקוד מכמה.

הפרצת הוראות ונתונים

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

אחר הסכנות הטמונה באירוע ההפרדה מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית בתכנית, לנסוט "לבצע" את הנתונים כאילו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום לתופעה זו הסתעפות לא נכונה. התכנית מבונן לא תעבד נכון, אך לרוב הנזק הוא יותר חמור, כי נוצרת חריגת חומרה ברגע שהמעבד מבצע פעולה שאינה חוקית.

האסטמבלר שלנו **חייב** להפריד, בקוד המכמה שהוא מיצר, בין קטע הנתונים לבין קטע ההוראות. **בלומר בקובץ הפלט (בקוד המכמה) תהיה הפרזה של הוראות ונתונים לשני קטיעים נפרדים, ואילו בקובץ הקלט אין חובה שתהייה הפרזה כזו.** בהמשך מתואר אלגוריתם של האסטמבלר, ובו פרטים כיצד לבצע את ההפרדה.

גילוי שגיאות בתכנית המקור

הנחת המטלה היא **אין שגיאות בהגדרות המאקרו**, ולכן קדם האסטמבלר אינו מכיל שלב גילוי שגיאות. אין גם צורך לבדוק שגיאות בפתיחת / סגירת המאקרו (למשל אם המאקרו לא מסתיים – ניתן להניח שהניל תקין). לעומת זאת, האסטמבלר אמר לגלות ולדווח על **שגיאות בתחריר של תוכנית המקור**, כגון פעולה שאינה קיימת, מספר אופrndים שגוי, סוג אופrnd שאינו מתאים לפעולה, שם רגיסטר לא קיים, ועוד שגיאות אחרות. כמו כן מודיא האסטמבלר שככל סמל מוגדר פעם אחת בדיק.

מכאן, שככל שגיאה המתגלגה על ידי האסטמבלר נגרמת (בדרך כלל) על ידי שורה קלט מסוימת. לדוגמה, אם מופיעים שני אופrndים בהוראה שאמור להיות בה רק אופrnd יחיד, האסטמבלר ייתן הודעה שגיאה בנוסח "יוטר מדי אופrndים".

הערה: אם יש שגיאה בקוד האסטמבלי בוגר ממאקרו, הרי שגיאה זו יכולה להופיע ולהתגלות שוב ושוב, בכל מקום בו נפרש המאקרו. נשים לב שכאשר האסטמבלר בוקח שגיאות, כבר לא ניתן לזרות שזה קוד שנפרש ממאקרו, כך שלא ניתן לחסוך גילויי שגיאה כפויים.

האסטמבלר ידפיס את הודעות השגיאה אל הפלט הסטנדרטי `stdout`. בכל הודעת שגיאה יש לציין גם את מספר השורה בקובץ המקור בה זוחתה השגיאה (מנין השורות בקובץ מתחילה ב-1).

لتשומת לב: האסמבילר איןנו עוצר את פועלתו אחרי שנמצאה השגיאה הראשונה, אלא ממשיך לעבר על הקלט כדי לגלות שגיאות נוספות, ככל שישן. כמוון שאין כל טעם לייצר את קבצי הפלט אם נתגלו שגיאות (ممילא אי אפשר להשלים את קוד המכונה).

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

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

אלגוריתם שלדי של האסמבילר

לחיזוד ההבנה של תהליך העבודה של האסמבילר, נציג להלן אלגוריתם שלדי למעבר הראשון ולמעבר השני. لتשומת לב: אין חובה להשתמש דווקא באלגוריתם זה.

אנו מחלקים את קוד המכונה לשני אזורים, אזור ההוראות (code) ואזור הנתונים (data). לכל אזור יש מונה משלה, ונסמן IC (МОונה ההוראות - Instruction-Counter) ו-DC (МОונה הנתונים - Data-Counter). נבנה את קוד המכונה כך שייתאים לטעינה לזיכרון **חול מכתובת 100**.

בכל מעבר מתחילה לקרוא את קובץ המקור מהתחלתה.

מעבר ראשון

- .1. אתחול 100 \leftarrow DC \leftarrow 0, IC \leftarrow 100.
- .2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עברו ל-17.
- .3. האם השדה הראשון בשורה הוא סמל? אם לא, עברו ל-5.
- .4. הדלק דגל "יש הגדרת סמל".
- .5. האם זהה הначיה לאחסון נתונים, כלומר, האם הначיה data.string או ?string. אם לא, עברו ל-8.
- .6. אם יש הגדרת סמל (תוויות), הכנס אותו לטבלת הסמלים עם המאפיין data.ערך יהיה DC.אם הסמל כבר נמצא בטבלת יש להודיע על שגיאיה.
- .7. זהה את סוג הנתונים, קודד אותם בזיכרון, ועדכן את מונה הנתונים DC בהתאם לאותם. חזרו ל-2.
- .8. האם זו הначיהtern.או הначיה entry.? אם לא, עברו ל-11.

9. אם זהה הначית `entry`. חזר ל-2 (ההנחיה תטופל במעבר השני).
10. אם זו הначית `extern`, הכנס את הסמל המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים עם הערך 0, ועם המאפיין `external`. חזר ל-2.
11. זהה שורת הוראה. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם המאפיין `code`. ערכו של הסמל יהיה IC (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאת).
12. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא, אז הודיע על שגיאת בשם ההוראה.
13. נתח את מבנה האופרנדים של ההוראה, וחשב מהו מספר המילים הכלול בתויפסת ההוראה בקוד המכונה (נקרא למספר זה L).
14. בנה כעת את הקוד הבינארי של המילה הראשונה של ההוראה, ושל כל מילת- מידע נוספת המקבצת אופרנד במייעון מיידי.
15. שומר את הערכים IC ו- L יחד עם נתוני קוד המכונה של ההוראה.
16. עדכן $IC \leftarrow IC + L$. וחזoor ל-2.
17. קובץ המקור נקרא בשלהמו. אם נמצא שגיאות במעבר הראשון, עצור כאן.
18. שומר את הערכים הסופיים של IC ושל DC (נקרא להם ICF ו- DCF). השתמש בהם לבניית קבצי הפלט, אחרי המעבר השני.
19. עדכן בטבלת הסמלים את ערכו של כל סמל המופיע כ- `data`, ע"י הוספת הערך ICF (ואיה הסבר לכך בהמשך).
20. התחל מעבר שני.

מעבר שני

1. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-7.
2. אם השדדה הראשונית בשורה הוא סמל (תוויות), דרג עליו.
3. האם זהה הначית `data`. או `string`. או ? אם כן, חזר ל-1.
4. האם זהה הначית `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 שגיאת). חזר ל-1.
6. השלם את הקידוד הבינארי של מילוט-המידע של האופרנדים, בהתאם לשיטות המייעון בשימוש. לכל אופרנד בקוד המקור המכיל סמל, מצא את ערכו של הסמל בטבלת הסמלים (אם הסמל לא נמצא בטבלה, יש להודיע על שגיאת). אם הסמל מאופיין `external`, הוסף את כתובת מילוט-המידע הרלוונטייה לרשימת מילוט-המידע שמתיחסות לSAMPLE חיצוני. לפי ה正确的, לחישוב הקידוד וה כתובות, אפשר להיעזר בערכים IC ו- L של ההוראה, כפי שנשמרו במעבר הראשוני. חזר ל-1.
7. קובץ המקור נקרא בשלהמו. אם נמצא שגיאות במעבר השני, עצור כאן.
8. בנה את קבצי הפלט (פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו לעיל (**לאחר שלב פרישת המאקרוואים**), ונציג את הקוד הבינארי שモתקבל 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   K, #-6
          bne   &END
          dec   K
          jmp   &LOOP
END:     stop
STR:    .string "abcd"
LIST:   .data  6, -9
          .data -100
K:      .data  31

```

בוצע עתה מעבר ראשון על הקוד הנוכחי. בניית טבלת הסמלים. כמו כן,בוצע מעבר זה גם את קידוד כל הנטונים, וקידוד המילה הראשונה של כל הוראה. כמו כן, נקבע מידע מילוט-מידע ומספרות של כל הוראה, ככל שקידוד זה אינו תלוי בערך של סמל. את החלקים שעדיין לא מתורגמים במעבר זה, נשאיר כמויות שהם (משמעותם ב- ? בדוגמה להלן).

Address (decimal)	Source Code	Explanation	Binary Machine Code
0000100	MAIN: add r3, LIST	First word of instruction	000010110110100000001100
0000101		Address of label LIST	?
0000102	LOOP: prn #48		001101000000000000001000
0000103		Immediate value 48	0000000000000000110000100
0000104	lea STR, r6		00010001000111000000100
0000105		Address of label STR	?
0000106	inc r6		00010100000111000011100
0000107	mov r3, K		0000001101101000000000100
0000108		Address of label K	?
0000109	sub r1, r4		00001011001110000010100
0000110	bne END		0010010000001000000010100
0000111		Address of label END	?
0000112	cmp K, #-6		000001010000000000000000100
0000113		Address of label K	?
0000114		Immediate value -6	1111111111111111010100
0000115	bne &END		0010010000010000000010100
0000116		Distance to label END	?
0000117	dec K		00010100000100000100100
0000118		Address of label K	?
0000119	jmp &LOOP		001001000001000000001100
0000120		Distance to label LOOP	?
0000121	END: stop		00111100000000000000000000100
0000122	STR: .string "abcd"	Ascii code 'a'	000000000000000000001100001
0000123		Ascii code 'b'	000000000000000000001100010
0000124		Ascii code 'c'	000000000000000000001100011
0000125		Ascii code 'd'	000000000000000000001100100
0000126		Ascii code '\0'	000000000000000000000000000000000
0000127	LIST: .data 6, -9	Integer 6	00000000000000000000000000000000110
0000128		Integer -9	1111111111111111111111110111
0000129	.data -100	Integer -100	1111111111111111111111110011100
0000130	K: .data 31	Integer 31	0000000000000000000000000000000011111

טבלת הסמלים אחרי המעבר הראשון היא :

סמל	ערך (בבסיס עשרוני)	איפיון הסמל
MAIN	100	code
LOOP	102	code
END	121	code
STR	122	data
LIST	127	data
K	130	data

נבע עתה את המעבר השני. נשלים באמצעות טבלת הסמלים את הקידוד החסר במיללים המסומנות ???" .
הקוד הבינארי בצוותו הסופית כאן זהה לקוד שהוצג בתחילת הנושא "אסמבלר עם שני מעברים".

הערה : כאמור, האסמבלר בונה קוד מכונה כך שייתאים לטיענה לזכרו החל מכתובת 100 (עשרוני).
אם הטיענה בפועל (לצורך הרצת התוכנית) תהיה לכתובת אחרת, ידרשו תיקונים בקוד הבינארי
בשלב הטיענה, שיוכנסו בעורת מידע נוסף שהאסמבלר מכין בקבצי הפלט (ראו בהמשך).

Address (decimal)	Source Code	Explanation	Binary Machine Code
0000100	MAIN: add r3, LIST	First word of instruction Address of label LIST	0000101101101000000001100 0000000000000001111111010
0000102	LOOP: prn #48	Immediate value 48	0011010000000000000000000100 0000000000000000110000100
0000104	lea STR, r6	Address of label STR	00010001000111000000100 000000000000000111010010
0000106	inc r6		000101000001111000001100
0000107	mov r3, K	Address of label K	000000110110100000000100 00000000000000010000010010
0000109	sub r1, r4		00001011001110000010100
0000110	bne END	Address of label END	0010010000001000000010100 0000000000000001111001010
0000112	cmp K, #-6	Address of label K Immediate value -6	000001010000000000000100 00000000000000010000010010 11111111111111111111010100
0000115	bne &END	Distance to label END	0010010000010000000010100 000000000000000000000110100
0000117	dec K	Address of label K	00010100000010000000100100 000000000000000000000100010
0000119	jmp &LOOP	Distance to label LOOP	0010010000010000000001100 1111111111111111111101111100
0000121	END: stop		0011110000000000000000000000100
0000122	STR: .string "abcd"	Ascii code 'a'	00000000000000000000001100001
0000123		Ascii code 'b'	00000000000000000000001100010
0000124		Ascii code 'c'	00000000000000000000001100011
0000125		Ascii code 'd'	00000000000000000000001100100
0000126		Ascii code '\0'	00000000000000000000000000000000
0000127	LIST: .data 6, -9	Integer 6 Integer -9	00000000000000000000000000000000110 11111111111111111111111101111
0000128	.data -100	Integer -100	1111111111111111111111110011100
0000129	K: .data 31	Integer 31	0000000000000000000000000000000011111
0000130			

בסוף המעבר השני, אם לא נתגלו שגיאות, האסמבלר בונה את קבצי הפלט (ראו בהמשך), שמכילים את הקוד הבינארי ומידע נוסף עבור שלבי הקישור והטיענה.

כאמור, שלבי הקישור והטיענה אינם למימוש בפרויקט זה, ולא נדוע בהם כאן.

קבצי קלט ופלט של האסמבלר

בפעולת 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, ובهم תוכניות בתחריר של שפת האסמביל שהוגדרה במיל'ין זה. האסמביל פועל על כל קובץ מקור בנפרד, ויוצר עבורו קבצי פלט כדלקמן:

- קובץ `.am`, המכיל את קובץ המקור לאחר שלב קדם האסמביל (לאחר פרישת המאקרוואים).
- קובץ `.object`, המכיל את קוד המוכנה.
- קובץ `.externals`, ובו פרטים על כל המKENOMOT (הכנתבות) בקוד המוכנה בהם יש מילת-מידע Shmekodzot Urk של סמל שהזכר כחיצוני (סמל שהופיע כאופרנד של הנחיה `extern` .., ומופיע בטבלת הסמלים `-external`).
- קובץ `.entries`, ובו פרטים על כל סמל שימושר ננקודת כניסה (סמל שהופיע כאופרנד של הנחיה .., ומופיע בטבלת הסמלים `-entry`).

אם אין בקובץ המקור אף הנחיה `extern` .., האסמביל לא יוצר את קובץ הפלט מסווג `.externals`. אם אין בקובץ המקור אף הנחיה `entry` .., האסמביל לא יוצר את קובץ הפלט מסווג `.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 כארוגמנטים לאסמביל נעשית לא ציון הסיומת.

לדוגמה: נניח שתוכנית האסמביל שלנו נקראת `assembler`, אז שורת הפקודה הבאה:

`assembler x y hello`

תրץ את האסמביל על הקבצים: `x.as`, `y.as`, `hello.as`

שמות קבצי הפלט מובוסים על שם קובץ הקלט, כפי שהופיע בשורת הפקודה, בתוספת סיימת מתאימה: הסיומת `".am"`. עברו קובץ לאחר פרישת מאקרו, הסיומת `".ob"`. עברו קובץ ה-`object`, הסיומת `".ext"`. עברו קובץ ה-`extern`, והסיומת `".ent"`. עברו קובץ ה-`entries`.

לדוגמה, בפעולת האסמביל באמצעות שורת הפקודה: `x assembler y ob`. וכך קבצי פלט `x.ext` ו- `x.ent` וככל שיש הנחיות `entry`. או `extern`. בקובץ המקור. אם אין מאקרו בקובץ המקור, אז קובץ `"am"`. יהיה זהה לקובץ `".as"`.

אופן פעולות האסמביל

נרחיב כאן על אופן פעולות האסמביל, בנוסף לאלגוריתם החלדי שניתן לעיל.

האסמביל מחזיק שני מערכיים, שייקראו להן מערך ההוראות ומערך הנתונים. מערכיים אלו נתונים למשזה תמורה של זיכרון המוכנה (גודל כל כניסה גודלה של מילת מוכנה (בטייבות). במערך ההוראות מכניס האסמביל את הקידוד של הוראות המוכנה שנקרו או במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמביל את קידוד הנתונים שנקרו או מקובץ המקור (שורות מסווג `.data` .., `.string`).

לאסמביל יש שני מונחים: מונה ההוראות (IC) ומונה הנתונים (DC). מונחים אלו מצביעים על המקום הבא הפנו במערכות לעיל, בהתאם. כשהמתחיל האסמביל לעבר על קובץ מקור, שני מונחים אלו מקבלים ערך התחלתי.

בנוסף יש לאסמבלר טבלה, אשר בה נאפסות כל התוויות בהן נתקל האסמבller במהלך המעבר על הקובץ. טבלה זו קוראים טבלת סמלים (symbol-table). לכל סמל (תוויות) נשמרים שמו, ערכו, ומאפיינים שונים שצינו קודם, כגון המיקום (data) או אופן העדכון (למשל external).

האסמבller 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 : האסמבller מתעלם מהשורה ועובד לשורה הבאה.

.2. שורת הוראה :

האסמבller מוצא מהי הפעולה, ומהן שיטות המיעון של האופרנדים. (מספר האופרנדים אותם הוא מփש נקבע בהתאם להוראה אותה הוא מצא).

אם האסמבller מוצא בשורת ההוראה גם הגדרה של תווית, אז התווית מוכנסת אל טבלת הסמלים. ערך התווית הוא IC, והמאפיין הוא code.

האסמבller קובע לכל אופרנד את ערכו באופן הבא :

- אם זה רגיסטר – האופרנד הוא מספר הרגיסטר.
- אם זו תווית (מיוען ישיר) – האופרנד הוא ערך התווית כפי שמופיע בטבלת הסמלים (ייתכן והסמל טרם נמצא בטבלת הסמלים).
- אם זה התו # ואחריו מספר – האופרנד הוא המספר עצמו.
- אם זו שיטת מיוען אחרת – ערכו של האופרנד נקבע לפי המפרט של שיטת המיעון (ראו תיאור שיטות המיעון לעיל)

קביעת שיטת המיעון נעשית בהתאם לחבר של האופרנד, כפי שהוסבר לעיל בהגדרת שיטות המיעון. למשל, מספר מצין מיידי, תווית מצינת מיוען ישיר ועוד>.

לאחר שהאסמבller ניתח את השורה והחליט לגבי הפעולה, שיטת מיוען אופרנד המקור (אם יש), ושיטת מיוען אופרנד החדש (אם יש), הוא פועל באופן הבא :

אם זה הפעולה בעלת שני אופרנדים, אזי האסמבller מכניס למערך ההוראות, במקומות עליו מצביע מוניה ההוראות IC, את קוד המילה הראשונה של ההוראה (בשיטה הייצוג של הוראות המכונה כפי שתואר קודם לכן). מילה זו מכילה את קוד הפעולה, ואת שיטות המיעון. בנוסך "מספרין" האסמבller מוקם במערך עבור המילים הנוספות הנדרשות עבור הוראה זו, ומגדיל את מוניה ההוראות בהתאם. אם אחד או שני האופרנדים הם בשיטת מיוען רגיסטר או מיידי, האסמבller מקודד כעת את המילים הנוספות הרלוונטיות למערך ההוראות.

אם זה הפעולה בעלת אופרנד אחד בלבד, כלומר אין אופרנד מקור, אזי הקידוד הינו זהה לעיל, פרט לנסיבות של שיטת המיעון של אופרנד המקור במילה הראשונה, אשר יכולו תמיד 0, מכיוון שאין רלוונטיות לפעולה.

אם זה הפעולה ללא אופרנדים אזי תקודד רק המילה הראשונה (והיחידה). הנסיבות של שיטות המיעון של האופרנדים יכילה 0.

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

.3. שורת הנחיה :

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

I. '.data'

האSEMBLER 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 שלפניהם הכנסת המספרים למערך הנתונים. כן מסומן שהגדרה ניתנה בחלק הנתונים.

II. '.string'

הטיפול ב-.string דומה ל-.data, אלא שקובדי ascii של התווים הם אלו המוכנסים אל מערך הנתונים (כל تو בכינסה נפרדת). לאחר מכן מוכנס הערך 0 (המצין סוף מחוזות) אל מערך הנתונים. מונה הנתונים מוקדם באורך המחרוזת + 1 (גם האפס בסוף המחרוזת תופס מקום).

הטיפול בתווית המוגדרת בשורה זו זהה לטיפול הנעשה בהנחיה '.data'.

III. '.entry'

זהי בקשה לאSEMBLER להכניס את התווית המופיעה כאופרנד של '.entry' אל קובץ ה-.entries. האSEMBLER רושם את הבקשה ובסיום העבודה, התווית הניל תירשם בקובץ ה-.entries.

IV. '.extern'

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

יש לשים לב: בהוראה או בהנחיה אפשר להשתמש בשם של סמל אשר הקריאה עליו ניתנת בהמשך הקובץ (אם באופן ישיר על ידי הגדרת תווית, ואם באופן עקיף על ידי הנחיתת extern).

בסוף המעבר הראשון, האSEMBLER מעדכו בטבלת הסמלים כל סמל המאופיין כ-.data, על ידי הוספת IC+100 (עשורי) לערכו של הסמל. הסיבה לכך היא שבתמונה הכלולות של קוד המוכנה, הנתונים מופרדים מההוראות, וכל הנתונים נדרשים להופיע אחרי כל ההוראות. סמל מסווג data הוא למעשה תווית באזור הנתונים, והעדכו מוסיף לערך הסמל (כלומר לכתובו בזיכרונו) את האורך הכלול של קידוד כל ההוראות, בתוספת כתובות התחלה הטעונה של הקוד, שהיא 100.

טבלת הסמלים מכילה כעת את כל הערכים הנחוצים להשלמת הקידוד (למעט ערכים של סמלים חיצוניים).

במעבר השני, האSEMBLER מקודד באמצעות טבלת הסמלים את כל המילים במערך ההוראות שטרם קודדו במעבר הראשון. אלו הן מילים שצרכיות להכיל כתובות של תוויות.

פורמט קובץ ה-object

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

כזכור, האסטובלר מקודד את ההוראות כך שתМОונת ההוראות תתאים לטעינה החל מכתובה 100 (עשרהוֹן) בזיכרונוֹ. נשים לב שרק בסוף המעבר הראשון יודיעם מהו הגודל הכולל של תМОונת ההוראות. מכיוון שתМОונת הנתונים נמצאת אחרי תМОונת ההוראות, גודל תМОונת ההוראות משפיע על הכתובות בתМОונת הנתונים. זו הסיבה שבגללה היה צריך לעדכן בטבלת הסמלים, בסוף המעבר הראשון, את ערכי הסמלים המקוריים כ-data (כזכור, בצד 19 הוסףנו לכל סמל כזה את הערך ICF). במעבר השני, בהשלמת הקידוד של מילוט-המידע, משתמשים בערכים המעודכנים של הסמלים, המתאימים לבניה המלא והסופי של תМОונת הזיכרון.

עת האסטובלר יכול לכתוב את תМОונת הזיכרון בשלמותה לתוך קובץ פלט (קובץ ה-object).

השורה הראשונה בקובץ ה-object היא "cotratt", המכילה שני מספרים (בבסיס עשרהּן): הראשון הוא האורך הכולל של תМОונת ההוראות (במילוט זיכרונוֹ), והשני הוא האורך הכולל של תМОונת הנתונים (במילוט זיכרונוֹ). בין שני המספרים מפריד רווח אחד. כזכור, במעבר הראשון, בצד 18, נשמרו הערכים ICF ו-IDF. האורך הכולל של תМОונת ההוראות הוא 100-ICF, והאורך הכולל של תМОונת הנתונים הוא IDF.

השורות הבאות בקובץ מכילות את תМОונת הזיכרון. בכל שורה שני שדות: כתובות של מילה בזיכרונוֹ, ותוכן המילה. הכתובות תירשם בסיס עשרהּן בשבע ספירות (כולל אפסים מוביילים). תוכן המילה יירשם בסיס הקסאדיימל ב-6 ספרות (כולל אפסים מוביילים). בין שני השדות בשורה יש רווח אחד.

פורמט קובץ ה-entries

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

פורמט קובץ ה-externals

קובץ ה-externals מבני אף הוא משורות טקסט. כל שורה מכילה שם של סמל שהוגדר external, וכתובות בקוד המוכונה בה יש קידוד של אופrndת המתיחיש לSymbol זה. כMOVN שיתכן ויש מספר כתובות בקוד המוכונה בהם מתיחישים לאותו Symbol חיצוני. לכל התיאחשות כזו תהיה שורה נפרדת בקובץ ה-externals. הכתובות מיוצגות בסיס עשרהּן.

לתשומת לב: ניתן ויש מספר כתובות בקוד המוכונה בהן מילוט-המידע מתיאחשת לאותו Symbol חיצוני. לכל כתובות כזו תהיה שורה נפרדת בקובץ ה-externals.

נדגים את קבצי הפלט שמייצר האסטובלר עבור קובץ מקור בשם as.dsk.
התוכנית לאחר שלב פרישת המאקרו נראה如此:

```
; 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 טבלת הקידוד הבינארי המלא שמתאפשר מקובץ המקור, כפי שנבנה בעבר הראשון והשני.

Address (decimal)	Source Code	Explanation	Binary Machine Code
0000100	MAIN: add r3, LIST	First word of instruction	000010110110100000001100
0000101		Address of label LIST	000000000000010000010010
0000102	LOOP: prn #48		0011010000000000000000000000100
0000103		Immediate value 48	0000000000000000000000000000100
0000104	lea W, r6		00010001000111000000100
0000105		Address of extern label W	00000000000000000000000000000001
0000106	inc r6		000101000001110000011100
0000107	mov r3, K		0000001101101000000000100
0000108		Address of label K	00000000000000001000010100
0000109	sub r1, r4		00001011001110000010100
0000110	bne END		00100100000100000010100
0000111		Address of label END	00000000000000001111000010
0000112	cmp K, #-6		000001010000000000000000100
0000113		Address of label K	00000000000000001000010100
0000114		Immediate value -6	11111111111111111111010100
0000115	bne &END		001001000001000000010100
0000116		Distance to label END	0000000000000000000000001001100
0000117	dec W		0001010000001000000100100
0000118		Address of extern label W	00000000000000000000000000000001
0000119	jmp &LOOP		00100100000100000001100
0000120		Distance to label LOOP	111111111111111101111100
0000121	add L3, L3		000010010000100000001100
0000122		Address of extern label L3	00000000000000000000000000000001
0000123		Address of extern label L3	00000000000000000000000000000001
0000124	END: stop		00111100000000000000000000000000100
0000125	STR: .string "abcd"	Ascii code 'a'	00000000000000000000000011000001
0000126		Ascii code 'b'	00000000000000000000000011000010
0000127		Ascii code 'c'	00000000000000000000000011000011
0000128		Ascii code 'd'	0000000000000000000000001100100
0000129		Ascii code '\0'	00000000000000000000000000000000
0000130	LIST: .data 6, -9	Integer 6	000000000000000000000000000000110
0000131		Integer -9	1111111111111111111111110111
0000132	.data -100	Integer -100	11111111111111111111110011100
0000133	K: .data 31	Integer 31	0000000000000000000000000000000011111

טבלת הסמלים הסופית בגמר המעבר השני היא :

סמל	ערך (בבסיס עשרוני)	אייפיון הסמל
W	0	external
MAIN	100	code, entry
LOOP	102	code
END	124	code
STR	125	data
LIST	130	data, entry
K	133	data
L3	0	external

להלן תוכן קבצי הפלט של הדוגמה.

הקובץ ps.ob

```
25 9
0000100 0b680c
0000101 000412
0000102 340004
0000103 000184
0000104 111e04
0000105 000001
0000106 141e1c
0000107 036804
0000108 00042a
0000109 0b3c14
0000110 240814
0000111 0003e2
0000112 050004
0000113 00042a
0000114 fffffd4
0000115 241014
0000116 00004c
0000117 140824
0000118 000001
0000119 24100c
0000120 fffff7c
0000121 09080c
0000122 000001
0000123 000001
0000124 3C0004
0000125 000061
0000126 000062
0000127 000063
0000128 000064
0000129 000000
0000130 000006
0000131 ffffff7
0000132 fffff9c
0000133 00001f
```

הקובץ ps.ent

```
MAIN 0000100
LIST 0000130
```

הקובץ ps.ext

```
W 0000105
W 0000118
L3 0000122
L3 0000123
```

לשומת לב: אם בקובץ המקור אין הנחיות `extern`. אז לא ייווצר קובץ `ext`. בדומה, אם אין בקובץ המקור הנחיות `entry`, לא ייווצר קובץ `ent`. **אין לייצור קובץ ext או ent שנשאר ריק**.

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

סיכום והנחיות כלליות

- אורך התוכנית, הניתנת כקלט לאסטבלר אינו ידוע מראש, ולכן התוכנית המתורגם אינו אמור להיות צפוי מראש. אולם כדי להקל בימוש האסטבלר, ניתן להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במעיריים לאחסן תמונה קוד המוגנה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים וטבלת המאקרו), יש למש באופן ייעיל וחסכוני (למשל באמצעות רשימה מקוישת והקצת זיכרון דינמי).
 - השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הקלט, למעט הסיומות. למשל, אם קובץ הקלט הוא `prog.as` אז קבצי הפלט שייצרו הם : `prog.ob`, `prog.ext`, `prog.ent`.
 - מתכוונת הפעלת האסטבלר צריכה להיות כפי הנדרש בממ'ין, ללא שינוים כלשהם. ככלומר, משק המשמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המוקור יועברו לתכנית האסטבלר באמצעות שורת הפקודה. אין להוציא תפריטי קלט איןקראקטיבים, חלונות גרפיים למיניהם, וכו').
 - יש להקפיד לחלק את מימוש האסטבלר למספר מודולים (קבצים בשפת C) לפי מישימות. אין לרוץ מישימות מסוימים שונים במודול יחיד. מומלץ לחלק למודולים כגון: מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרומות לבסיס, ניתוח תחבירי של שורה), טבלת הסמלים, מפת הזיכרון, טבלאות קבועות (קודם הפעלה, שיטות המיעון החוקיות לכל פעולה, וכו').
 - יש להקפיד ולתעד את המימוש באופן מלא וברור, באמצעות הערות מפורטות בקוד.
 - יש לאפשר תווים לבנים עודפים בקובץ הקלט בשפת אסטבלר. למשל, אם בשורת הוראה יש שני אופרנדים המופרדים בפסיק, איזי לפני ואחרי הפסיק מותר שיהיו רווחים וטאבים בכל מקומות. בדומה, גם לפני ואחרי שם הפעולה. מותירות גם שורות ריקות. האסטבלר יתעלם מתווים לבנים מיוחדים (כלומר ידלג עליהם).
 - הקלט (קוד האסטבלר) עלול להכיל שגיאות תחביריות. על האסטבלר לגלות ולדוח על כל השורות השגויות בקלט. אין לעזור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להדפיס למסך הוצאות מפורטות מכל הניתן, כדי שאפשר יהיה להבין מה והיכן כל השגאה. כמובן שגם קובץ קלט מכיל שגיאות, אין טעם להפיק עבورو את קבצי הפלט (`ent`, `ob`, `ext`).
 - תשוםת פרק ההסבירים והגדרת הפרויקט.**
 - בשאלות ניתן לפנות לקבוצת הדיוון באתר הקורס, ואל המנהים בשעות הקבלה. להזכירם, מומלץ לכל אלה שטרם בדקו את התכנים באתר הקורס לעשותות זאת. נשאלות באתר הרבה שאלות בנושא חומר הלימוד והממיינים, והתשובות יכולות להועיל לכם.
- לשומת לבכם :
- על המטלה להיות מקורית לגמרי: **אין להיעזר בספריות חיצונית מלבד הספריות הסטנדרטיות**, וכמובן לא בקוד שמצאתם בראש או קיבלתם בכל דרך. **אין לשתח בראשת קוד ללא סיסמה. אלו הן עבירות ממשעה.**
 - **לא תינטן דחיה בהגשת הממיין, פרט לMKRIM חריגים במיפוי.** במקרים אלו יש לבקש ולקבל אישור מראש מנהה הקבוצה.

ב ה צ ל ח ה !