

מטלת מנהה (ממ"ז) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלת : פרויקט גמר

משקל המטלת : 61 נקודות (חוובה)

מספר השאלות : 1

מועד אחרון להגשה : 20.03.2022 סמסטר : א' 2022

קיימות שתי חלופות להגשת מטלות:

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

הסבר מפורט ב"נווה הגשת מטלות מנהה"

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

עליכם ל כתוב תוכנת אסמבלי, עבור שפת אסמבלי שתוגדר בהמשך. הפרויקט יכתב בשפת C.

עליכם להגיש את הפריטים הבאים :

1. קבצי המקור של התוכנית שכתבתם (קבצים בעלי הסיומת .c או .h).
2. קובץ הרצה (מוקומפל ומקוישר) עבור מערכת אוביונטו.
3. קובץ makefile. הקימפול חייב להיות עם הקומפיילר gcc והדגלים : -Wall -ansi -pedantic . יש לנפות את כל ההודעות שモוציאה הקומפיילר, כך שהתוכנית תתקמפל ללא כל העורות או אזהרות.
4. דוגמאות הרצה (קלט ופלט) :
 - א. קבצי קלט בשפת אסמבלי, ובceği הפלט שנוצרו מהפעלת האסמבלי על קבצי קלט אלה. יש להציג שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האסמבלי.
 - ב. קבצי קלט בשפת אסמבלי המציגים מגוון רחב של סוגי שגיאות אסמבלי (ולכן לא נוצרים קבצי פלט), ותדפסי המסקן המראים את ה הודעות השגיאה שモוציא האסמבלי.

בשל גודל הפרויקט, עליכם לחלק את התוכנית במספר קבצי מקור, לפי מישימות. יש להזכיר שקוד המקור של התוכנית יעמוד בקריטריונים של בהירות, קריאות וכתיבה נאה ומובנית.

זכור מספר היבטים חשובים של כתיבת קוד טוב :

1. הפיטה של מבני הנתונים : רצוי (כל האפשר) להפריד בין הגישה למבנה הנתונים לבין המימוש של מבני הנתונים. כך, למשל, בעת כינית פונקציות לטיפול בטבלה, אין זה מעניינים של המשתמשים בפונקציות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקוישת.
2. קריאות הקוד : יש להשתמש במסות משמעותיים למשתנים ופונקציות. יש לעורך את הקוד באופן מסודר : הזחות עקבות, שורות ריקות להפרדה בין קטעי קוד, וכו'.
3. תיעוד : יש להכניס בקבצי המקור תיעוד תמציתי וברור, שיסביר את תפקידה של כל פונקציה (באטען הערות כוורתת לכל פונקציה). כמו כן יש להסביר את תפקדים של משתנים חשובים. כמו כן, יש להכניס הערות ברמת פירוט גבוהה בכל הקוד.

הערה : תוכנית "עובדת", דהיינו תוכנית שמבצעת את כל הדריש ממנה, אינה לכשעצמה ערובה לצוין גביה. כדי לקבל ציון גבוה, על התוכנית לעמוד בקריטריונים של כתיבה ותיעוד ברמה טובה, כמפורט לעיל, אשר משקלם המשותף מגע עד לכ- 40% משקל הפרויקט.

ומותר להשתמש בפרויקט בכל מגוון הספריות הסטנדרטיות של שפת C, אבל אין להשתמש בספריות חיצונית אחרות.

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

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

רקע כללי ומטרת הפרויקט

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

למעשה, זיכרונו המחשב כולו הוא אוסף של סיביות, שנוהגים לראותן כמקובצות ליחידות בעלות אורך קבוע (בתים, מיללים). לא ניתן להבחין, בעין שאין מימנת, בהבדל פיסי כלשהו בין אותו חלק בזיכרונו שבו נמצאת תוכנית לבין שאר הזיכרונו.

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מגוון פעולות פשוטות, הנקראות **הוראות מכונה**, ולשם כך היא משתמשת באוגרים (registers) הקיימים בתוך היע"מ, ובזיכרונו המחשב. **דוגמאות**: העברת מספר מתא בזיכרונו לאוגר ביע"מ או בחזרה, הוספה 1 למספר הנמצא באוגר, בדיקה האם המספר באוגר שווה לאפס, חיבור וחיסור בין שני אוגרים, ועוד. הוראות המכונה ושילובים שלهنן הנקראות תוכנית כפיה שהיא טעונה לזכרון בזמן ריצתה. כל תוכנית מקור (התוכנית כפיה שנכתבה בידי המתכנן), תתרגם בסופו של דבר באמצעות תוכנה מיוחדת לצורה סופית זו.

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

כידוע, לכל שפת תוכנות עילית יש מהדר (compiler), או מפרש (interpreter), המתרגם תוכניות מקור לשפת מכונה. האסambilר משמש בתפקיך זוםה עבור שפת אסambilר.

כל מודל של יע"מ (כלומר לכל אירוגון של מחשב) יש שפת מכונה יעודית משלו, ובהתאם גם שפת אסambilר יעודית משלו. לפיכך, גם האסambilר (כלי התרגום) הוא יעודית ושונה לכל יע"מ.

תפקידו של האסambilר הוא לבנות קוד המכיל מילים מוכנה, מקובץ נתון של תוכנית הכתובת בשפת אסambilר. 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 נעסק במיל'ן זה.

המשימה בפרויקט זה היא לכתוב אסambilר (כלומר תוכנית המתרגם לשפת מכונה), עבר שפת אסambilרי שנגידר כאן במיוחד לצורך הפרויקט.

תשומת לב : בהסבירים הכלליים על אופן עבודה תוכנת האסambilר, תהיה מדי פעם התיחסות גם לעובדות שלבי הקישור והטעינה. התיחסויות אלה נועדו על מנת לאפשר לכם להבין את המשך תהליך העבודה של הפלט של תוכנת האסambilר. אין לטעות: **עליכם לכתוב את תוכנית האסambilר בלבד. אין לכתוב את תוכניות הקישור והטעינה!!!**

המחשב הדמיוני ושפת האסמבלי

נגיד לך את שפת האסמבלי ואת מודל המחשב הדמיוני, עברו פרויקט זה.

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

"חומרה":

המחשב בפרויקט מורכב ממעבד (יע"מ), אוגרים (רגיסטרים), זיכרון RAM. חלק מהזיכרון משמש כמחסנית (stack).

למעבד 16 אוגרים כלליים, בשמות : r0, r1, r2, r3, r4, r5, r6, r7, ..., r15. הסיבית ה-0 של כל אוגר היא סיביות. השם של אוגר ה-0 הוא מס' 0, והסיבית ה-15 של אוגר ה-0 היא סיבית מס' 15. שמות האוגרים כתובים תמיד עם אות 'r', קטנה.

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

גודל הזיכרון הוא 8192 תאים, בכתובות 0-8191, וכל תא הוא בגודל של 20 סיביות. לתא בזכרון נקרא גם בשם "מילה". הסיביות בכל מילה ממושפרות כמו באוגר.

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

מבנה הוראות המוכנה:

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

כל הוראה מוכנה מוקודת במספר מילوت זיכרון רצופות, החל ממילה אחת ועד למקסימום של מילים, בהתאם לסוג הפעולה (ראו פרטים בהמשך).

בקובץ הפלט המכיל את קוד המוכנה שבונה האסמלבלר, כל מילה תקודה "בסיס מיוחד" (ראו פרטים לגבי קבצי פלט בהמשך).

בהוראה מוכנה ללא אופרנדים, המבנה של המילה הראשונה (והיחידה) הוא :

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

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

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

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

להלן טבלת הפעולות:

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

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

להלן מפרט השדות בקידוד הראשונה בקוד המcona של כל הוראה.

שדה opcode: שדה זה מיוצג ב- 16 סיביות, והוא מכיל בית זולק בודד בהתאם לקוד הפעולה. למשל אם מדובר על קוד פעולה 9, אז סיבית 9 קיבל ערך של 1.

סיבית 19: לא בשימוש,Urca קבוע לאפס.

סיביות 18-16: עבור קידוד הוראות, סיביות אלה מכילות את סיווג הקידוד (A=Absolute, R=Relocatable, E=External) לכל מילת קידוד של הוראה יש סיווג, ובהתאם לסיווג הסיביות המתאימה בשדה ARE מקבל ערך 1.

סיביות 15-12: שדה זה, הנקרא **funct**, מתפרק כאשר מדובר בפעולת שקוד-הפעולה (opcode) שלא משותף לכמה פעולות (כאמור, קוד-פעולה 2, 5 או 9). השדה funct יכול לקבל ערך ייחודי לכל פעולה מקובצת הפעולות שיש להן אותו קוד-פעולה. אם קוד-הפעולה משמש ל פעולה אחת בלבד, הסיביות של השדה funct יהיו מאופסות.

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

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

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

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

שיטות מייען:

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

השימוש בשיטות המייען מצריך מילוט-מידע נוספת בקוד המכונה של הוראה, בנוסף למילויים הראשוניים. קידוד אופרנד של הוראה עשוי לייצר מילוט זיכרון נוספת בהתאם לסוג שיטת המייען. כאשר בהוראה יש שני אופרנדים, קודם יופיעו מילוט-המידע הנוספות של אופרנד המקור, ולאחריהם מילוט-המידע הנוספות של אופרנד השני.

להלן המפרט של שיטות המייען.

מספר	שיטה המייען	תוכן מילוט-המידע הנוספות	תחביר האופרנד באסמבלי	דוגמה
0	מייען מיידי (immediate)	מילוט-מידע נוספת של הhorאה מכילה את האופרנד עצמו, שהוא מספר שלם בסיסי המשלים ל-2, ברוחב של 16 סיביות מילה זו תסוג מסוג A	האופרנד מתחליל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בסיסי עשרוני.	mov #1, r2 בדוגמה זו האופרנד הראשון של הhorאה (אופרנד המקור) נתנו (אנו מתייחסים ל-2). הhorאה כתובת את הערך 1 אל אוגר r2.
1	מייען ישיר (direct)	2 מילוט-מידע נוספת של הhorאה מכילו את כתובת האופרנד במבנה של כתובת בסיס והיסט. כתובת בסיס = הכתובת הקרובה ביותר לכתובת האופרנד, הקטנה ממנו, ומתחלקת ב-16. למשל, אם כתובת הבסיס היא 36, אז כתובת הักษם היא 32, והוא המספר הקרוב ביותר ל-36 שקטן ממנו ומתחלך ב-16. היסט = המרחק מכתובת הבסיס לכתובת האופרנד. בדוגמה שניתנה בהסביר לכתובת בסיס, היסט יהיה 4. מאחר והמרחק (ההיסט) מ-32 ל-36 הוא 4. מילוט-המידע נוספת המכילה את כתובת הבסיס. ומילוט-המידע השנייה מכיל את הhisst. כתובת הבסיס והhisst מיוצגים כמספר <u>לא סימן</u> ברוחב של 16 סיביות. והם יסווגו מסוג R במידה והאופרנד הוא תווית חייצונית, כתובת הבסיס והhisst יכילו אפסים, וקידודים אלה יסווגו מסוג E במקרה כזה.	האופרנד הוא <u>תוויות</u> שכבר הוגדרה, או שתוגדר בהמשך הקובץ. הגדירה נעשית על ידי כתיבת התווית בתחילת השורה של הנחית 'data'. או '.string' של horאה, או באמצעות אופרנד של הנחית 'extern'. התווית מייצגת באופן סימבולי כתובת זיכרון.	השורה הבאה מגדרה את התווית x : x: .data 23 horאה : dec x מקטינה ב-1 את תוכן המילה שבכתובת x בזיכרון (ה"משתנה" x). הכתובת של x מקודדת במילוט-המידע הנוספות, במבנה של כתובת בסיס והיסט. <u>דוגמה נוספת :</u> horאה jmp next מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר horאה הבאה שתתבצע נמצאת בכתובת next). הכתובת next מקודדת במילוט-המידע הנוספות, במבנה של כתובת בסיס והיסט.

מספר	שיטת המיעון	תוכן מילוט-המידע הנוספות	תחביר האופרנד באסמבלי	דוגמה
2	מיעון אינדקס	בשיטה זו, יש בקידוד ההוראה 2 מילוט מידע נוסף, המכילות את כתובת התוויות בצורת כתובות בסיס והיסט כפי שתואר בשיטת מיעון מספר 1.	האופרנד מתחליל בשם של תווית ואחריה בסוגרים מרובעות, שם של אוגר שמספרו בין 10 ל- 15 בלבד.	השורה הבאה מגדרה את התוויות א : x: .data 23,12,34,50 הפקודה : mov x[r12], r4 בדוגמה זו האופרנד הראשון הוא גישה כתובת של X בזיכרו, והחל משם מתקדמים כמוות של מילוט זיכרון נוספות לפי הערך שהוא,r, בזמן ריצעה באוגר 12, ומה שיש באותה כתובות ישמש את המעבד כאופרנד המקורי. בדוגמה זו אופרנד זו יישמר באוגר 14
3	מיעון אוגר ישיר (register direct)	אין מילוט-מידע נוספת בגין האוגר. מספרו של האוגר יישמר בסיביות של אוגר המקורי/עד בהתאם לכך אם האופרנד הוא אופרנד המקורי/עד.	האופרנד הוא שם של אוגר.	clr r1 בדוגמה זו, ההוראה clr מספקת את האוגר.r1. <u>דוגמה נוספת :</u> mov #-1, r2 האופרנד השני של ההוראה (אופרנד היעד) נתנו בשיטת מיעון אוגר ישיר. ההוראה כתובת את הערך המיידי -1 אל אוגר r2.

מפורט הוראות המכונה:

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

הוראות המכונה מתחולקות לשלוש קבוצות, לפי מספר האופרנדים הנדרשים לפחות.

קבוצת ההוראות הראשונה:

אלו הן הוראות המקבלות שני אופרנדים.

ההוראות השיכות לקובוצה זו הן: mov, cmp, add, sub, lea

הסבר הדוגמה	דוגמאות	הפעולה המתבצעת	funct	opcode	הוראה
העתק את תוכן המשתנה A (המילה שבכתובת r1 בזיכרון) אל אוגר r2.	mov A, r1	מבצע העתקה של תוכן אופרנד המקור (האופרנד הראשון) אל אופרנד היעד (האופרנד השני).		0	mov
אם תוכן המשתנה A זהה לתוכנו של אוגר r1 אז הדגל Z ("דגל האפס") באוגר הסטטוס (PSW) יודלק, אחרת הדגל יאפס.	cmp A, r1	מבצע השוואة בין שני האופרנדים. ערך אופרנד היעד (השני) מופחת מערך אופרנד המקור (הראשון), ללא שבירת תוצאת החישור. פועלות החישור מעדכנת דגל בשם Z ("דגל האפס") באוגר הסטטוס (PSW).		1	cmp
אוגר r2 מקבל את תוצאה החיבור של תוכן המשתנה A ותוכנו הנוכחי של r0.	add A, r0	אופרנד היעד (השני) מקבל את תוצאה החיבור של אופרנד המקור (הראשון) והיעד (השני).	10	2	add
אוגר r1 מקבל את תוצאה החישור של הקבוע 3 מתוכנו הנוכחי של האוגר r2.	sub #3, r1	אופרנד היעד (השני) מקבל את תוצאה החישור של אופרנד המקור (הראשון) מאופרנד היעד (השני).	11	2	sub
המען המייצג התוויות HELLO מוצב לאוגר r1.	lea HELLO, r1	lea הוא קיצור (ראשי תיבות) של load effective address. מצייבה את מען הזיכרון המוצג על ידי התווית שבאופרנד הראשון (המקור), אל האופרנד השני (היעד).		4	lea

קבוצת ההוראות השנייה:

אלו הן הוראות המקבלות אופרנד אחד בלבד. אופן הקידוד של האופרנד הוא כמו של **אופרנד היעד** בהוראה עם שני אופרנדים. השדה של אופרנד המקור (סיביות 2-3) במילה הראשונה בקידוד ההוראה אינם בשימוש, ולפיכך יהיו מאפס.

ההוראות השיכות לקובוצה זו הן: clr, not, inc, dec, jmp, bne, jsr, red, prn

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

הסבר הדוגמה	דוגמה	הפעולה המתבצעת	funct	opcode	הורה
אם ערך הדגל Z באוגר הסטטוס (PSW) הוא 0, אז PC \leftarrow address(Line) מצבי התוכנית קיבל את כתובות התוויות Line, ולפיכך ההוראה הבאה שתתבצע תהיה במען Line.	bne Line	bne הוא קיצור (ראשי תיבות) של : branch if not equal (to zero). זוהי הוראות הסטטופות מותניות. אם ערכו של הדגל Z באוגר הסטטוס (PSW) הינו 0, אז מצביע התוכנית (PC) מקבל את כתובות יעד הקפיצה. כזכור, הדגל Z נקבע באמצעות הוראת .cmp.	11	9	bne
push(PC+2) PC \leftarrow address(SUBR) מצבי התוכנית קיבל את כתובות התוויות SUBR, ולפיכך, ההוראה הבאה שתש揆 צהה במען SUBR. כתובות החזרה מהשגרה נשמרת במחסנית.	jsr SUBR	קריאה לשגרה (סברוטינה). כתובות ההוראה שאחרי הוראות js הוכחת (PC+2) נדחפת לתוכן המחסנית שבזיכרונו המחשב, ומצביע התוכנית (PC) מקבל את כתובות השגרה. הערת : זורה מהשגרה מתבצעת באמצעות הוראות rts, תוך שימוש בכתובות שבמחסנית.	12	9	jsr
קוד ascii של התו הנקרא מהקלט הסטנדרטי .r1 ייכנס לאוגר r1.	red r1	קריאה שלתו מהקלט הסטנדרטי (stdin) אל האופרנד.		12	red
יודפס לפלט התו הנמצא באופרנד, אל r1 הנמצא באוגר r1	prn r1	הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).		13	prn

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

ההוראות השייכות לקבוצה זו הן : rts, stop

הסבר הדוגמה	דוגמה	הפעולה המתבצעת	opcode	הורה
PC \leftarrow pop() ההוראה הבאה שתתבצע תהיה זו שאחרי הוראת js רשות לשגרה. התוכנית עוצרת מיידית.	rts	מתבצעת זורה משיגרה. הערך שבראש המחסנית של המחשב מוצא מן המחסנית, ומוכנס למצביע התוכנית (PC). הערת : ערך זה נכנס למחסנית בקריאה לשגרה ע"י הוראת js.	14	rts

מבנה תוכנית בשפת אסמבלי :

תוכנית בשפת אסמבלי בנויה **ממרקוריום ומשפטים** (statements).

מרקוריום :
מרקוריום הם קטעי קוד הכלולים בתוכם משפטיים. בתוכנית ניתן להגדיר מקרו ולהשתמש בו במקומות שונים בתוכנית. השימוש במרקרו מסיים במסוים בתוכנית יגרום לפרישת המקרו לאותו מקום.

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

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

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

m1

m1

בדוגמה זו, השתמשנו פעמיים במקרו **1ות**, התוכנית לאחר פרישת המקרו תיראה כך:

inc r2
mov A,r1

inc r2
mov A,r1

התוכנית לאחר פרישת המקרו היא התוכנית שהאSEMBLER אמר לתרגם.

הנחיות לגבי מקרו:

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

משפטים:

קובץ מקור בשפת אSEMBLER מורכב משורות המכילות משפטיים של השפה, כאשר כל משפט מופיע בשורה נפרדת. כלומר, הפרדה בין משפט בקובץ המקור הינה באמצעות התוו 'ז' (שורה חדשה).

אורכה של שורה בקובץ המקור הוא 80 תווים לכל היותר (לא כולל התוו 'ז).

יש ארבעה סוגי משפטיים (שורות בקובץ המקור) בשפת אSEMBLER, וهم:

סוג המשפט	הסבר כללי
משפט ריק	זהי שורה המכילה אך ורק תווים לבנים (whitespace), ככלומר רק את התווים ' ' ו- 't' (רווחים וטאבים). יתכן ובשורה אין אף תו (למעט התו t\), ככלומר השורה ריקה.
משפט הערת	זהי שורה בה התו הראשון הינו ',' (נקודה פסיק). על האסמלר להתעלם לחולטיין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמלר מה עליו לעשות כשהוא פועל על תוכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצת זיכרונו ואתחול משתנים של התוכנית, אך הוא אינו מייצר קידוד של הוראות מכונה המיועדות לביצוע בעת ריצת התוכנית.
משפט הוראה	זהו משפט המיציר קידוד של הוראות מכונה לביצוע בעת ריצת התוכנית. המשפט מורכב ממשם ההוראה (פעולה) שעל המעבד לבצע, והופרנדים של ההוראה.

cut נפרט יותר לגבי סוגי המשפטים השונים.

משפט הנחיה:

משפט הנחיה הוא בעל המבנה הבא :

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

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

יש ארבעה סוגים (שמות) של משפטי הנחיה, וهم :

1. הנחיה 'data'.

פרמטרים של הנחיה '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 לפסק ובין פסיק למספר יכולים להופיע רווחים וטאבים בכלל כמוות (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 האסמלר להקצות מקומות בתמונה הנתונים (data image), אשר בו יוחסנו הערכים של הפרמטרים, ולקדם את מונה הנתונים, בהתאם למספר הערכים. אם בначית data. מוגדרת תווית, אז תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום), ומוכנסת אל טבלת הסמלים. דבר זה מאפשר להתייחס אל מקום מסוים בתמונה הנתונים דרך שם התווית (למעשה, זהה דרך להגדיר שם של משתנה).

כלומר אם נכתב :

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

אז יוקצו בתמונה הנתונים ארבע מילימ רצופות שכילו את המספרים שמופיעים בהנחיה. התווית XYZ מזוהה עם כתובות המילה הראשונה.

אם נכתב בתוכנית את ההוראה :

mov XYZ, r1

אזי בזמן ריצת התוכנית יוכנס לאוגר 1ז הערך 7.

ואילו הוראה :

```
lea XYZ, r1
```

תוכניס לאוגר 1ז את ערך התווית XYZ (כלומר הכתובת בזיכרון בה מוחSEN הערך 7).

2. הנקיה '.string'

לנקיה '.string' פרמטר אחד, שהוא מחוזת חוקית. תווית המחווזת מקודדים לפי ערכי ה-ascii המתאים, ומוכנסים אל תומנת הנתונים לפי סדרם, כל TWO ב밀יה נפרדת. בסוף המחווזת יתוסף התו '0' (הערך המספרי 0), המסמן את סוף המחווזת. מונה הנתונים של האסמלר יקודם בהתאם לערך המחווזת (בהתאמה מקום אחד עבר התו המסייע). אם בשורת הנקיה מוגדרת תווית, אזי תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום) ומוכנסת אל טבלת הסמלים, בדומה כפי שנעשה עבור '.data'. (כלומר ערך התווית יהיה הכתובת בזיכרון שבה מתחילה המחווזת).

לדוגמה, הנקיה :

```
STR: .string "abcdef"
```

מקצת בתומנת הנתונים רצף של 7 מיללים, ומאחרת את המיללים לקודי ה-ascii של התווים לפי הסדר במחווזת, ולאחריהם הערך 0 לסימון סוף מחווזת. התווית STR מזוהה עם כתובת התחלת המחווזת.

3. הנקיה '.entry'

לנקיה '.entry' פרמטר אחד, והוא שם של תווית המוגדרת בקובץ המקור הנוכחי (כלומר תווית שמקבלת את ערכה בקובץ זה). מטרת הנקיה entry היא לאפיין את התווית זו באופן שיאפשר לקוד אסמלרי הנמצא בקובץ מקור אחרים להשתמש בה (כօפרנד של הוראה).

לדוגמה, השורות :

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

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

لتשומת לב: תווית המוגדרת בתחלת שורת entry. הינה חסרת משמעות והאסמלר **מתעלם** מהתווית זו (אפשר שהאסמלר יוציא הודעה אחרת).

4. הנקיה '.extern'

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

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

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

```
.extern HELLO
```

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

لتשומת לב : תווית המוגדרת בתחילת שורת `extern`. הינה חסרת משמעות והאסמבלר **מטעלים מתווית זו** (אפשר שהאסמבלר יוציא הודעה זהה).

משפט הוראה :

משפט הוראה מורכב מחלקים הבאים :

1. תווית אופציונלית.
2. שם הפעולה.
3. אופרנדים, בהתאם לסוג הפעולה (בין 0 ל-2 אופרנדים).

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

שם הפעולה תמיד באותיות קטנות (lower case), והוא אחת מ- 16 הפעולות שפורטו לעיל.
לאחר שם הפעולה יופיעו האופרנדים, בהתאם לסוג הפעולה. יש להפריד בין שם-הפעולה לבין האופרנד הראשון באמצעות רווחים ו/או טאים (אחד או יותר).

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזה בטו ' , (פסיק). בדומה להנחיה 'data' , **לא חייבת 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 אופרנד אחד המבנה הבא :

label: opcode target-operand

לדוגמה :

HELLO: bne XYZ

למשפט הוראה ללא אופרנדים המבנה הבא :

label: opcode

לדוגמה :

END: stop

אפיון השדות במשפטים של שפת האסמבלי

תווית :

תווית היא סמל שМОגדר בתחילת משפט הוראה' או בתחילת הנחיה `data`. או `string`.
תווית חוקית מתחילה באות אלביניטי (גדולה או קטנה), ולאחריה סדרה כלשהי של אותיות אלביניטיות (גדולות או קטנות) ו/או ספרות. האורך המקסימלי של תווית הוא 31 תווים.

הגדרה של תווית מסוימת בטו ' : (נקודותים). זו זה אינו מהו חלק מהנתווית, אלא רק סימן המציין את סוף ההגדרה. הטו ' : ' חייב להיות צמוד לתווית (ללא רווחים).

אסור שאויה תווית תוגדר יותר מפעם אחת (כਮון בשורות שונות). נתווית קטנות וגדולות נכחות 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 הינה, או שם של אוגר) אינן יכולות לשמש גם כשם של תווית. לדוגמה : הסמלים add, z לא יכולים לשמש כתוויות, אבל הסמלים Add, R, r הם תוויות חוקיות.

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

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

מספר :

מספר חוקי מתחילה בסימן אופציוני : ‘—’ או ‘+’, ולאחריו סדרה של ספרות בסיס עשרוני. דוגמה : 5, 76, +123 הם מספרים חוקיים. אין תמיכה בשפת האסמבלי שלנו בייצוג בסיס אחר מאשר עשרוני, ואין תמיכה במספרים שאינם שלמים.

מחרוות :

מחרוות חוקית היא סדרת תווי ascii נראים (שניינги לדפסה), המוקפים במרכאות כפולות (המרכאות אינן נחשבות חלק מהמחרוות). דוגמה למחרוות חוקית : “hello world”.

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

האסמבלי בונה מלכתחילה קוד מכונה שמיועד לטיענה החל מכתובת 100. אולם, לא בכל פעם שהקוד יטען לזרקן הרצה, מובטח שאפשר יהיה לטען אותו החל מכתובת 100. במקרה כזה, קוד המכונה הנוכחי אינו מתאים ויש צורך לתקן אותו. לדוגמה, מילת-המידע של אופrnd בשיטת מעון ישיר לא תהיה נכונה, כי הכתובת השנתנה.

הרעיון הוא להכניס תיקונים נקודתיים בקוד המכונה בכל פעם שייטען לזרקן הרצה. כך אפשר יהה לטען את הקוד בכל פעם למקום אחר, בלי צורך לחזור על תהליך האסמבלי. תיקונים כאלה נעשים בשלב הקישור והטיענה של הקוד (אנו לא מטפלים בכך במילן זה), אולם על האסמבלי להוסיף מידע בקוד המכונה שיאפשר לזהות את הנקודות בקוד בהן נדרש תיקון.

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

- האות A (קייזר של Absolute) באח לציין שתוכן המילה אינו תלוי במקום בו זיכרונו בו ייטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל, 2 המילים המכילים את קוד ההוראה ואת שיטות המידע, או מילת-מידע המכילה אופrnd מיידי).
- האות R (קייזר של Relocatable) באח לציין שתוכן המילה תלוי במקום בו זיכרונו בו ייטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל, מילוט-מידע המכילות כתובות של תווית בצורת כתובות בסיס והיסט).
- האות E (קייזר של External) באח לציין שתוכן המילה תלוי בערכו של סמל שאינו מוגדר בקובץ המקור הנוכחי (למשל, מילת-מידע המכילה ערך של סמל המופיע בהנחיהtern.).

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

כאשר האסמבילר מקבל כקלט תוכנית בשפת אסמבלי, עליו לטפל תחילה בפרישת המקרואים ורק לאחר מכן לעבור על התוכנית אליה נפרשו המקרואים. לעומת זאת, פרישת המקרואים תעשה בשלב "קדם אסמבילר", לפני שלב האסמבילר (המtoaר בהמשך).
אם התוכנית אינה מכילה מקרו, תוכנית הפרישה תהיה לתוכנית המקור.

דוגמה לשלב קדם אסמבילר. האסמבילר מקבל את התוכנית הבאה בשפת אסמבלי :

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

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

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

```

; file ps.am
.entry LIST
.extern W

MAIN:    add    r3, LIST
LOOP:    prn   #48
          lea    STR, r6
          inc    r6
          mov    r3, W
          sub    r1, r4
          bne   END
          cmp    val1, #-6
          bne   END[r15]
          dec    K

.entry MAIN
          sub    LOOP[r10], r14
END:      stop
STR:      .string "abcd"
LIST:     .data  6, -9
          .data  -100
.entry K
K:        .data  31
.extern val1

```

קוד התכנית, לאחר הפרישה, ישמור בקובץ חדש, כפי שIOSVER בהמשך.

אלגוריתם שלדי של קוד האסמלר

נזכיר להלן אלגוריתם שלדי לתהיליך קוד האסמלר. 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 המקור (כגון m1)? אם כן, החלף את שם המקור והעתק במקומו את כל השורות המתאימות מטבלה לקובץ, חוזר ל.1. אחרת, המשך.
- .3. האם השדה הראשון הוא "macro" (התחלת הגדרת מקרו)? אם לא, עبور ל- 6.
- .4. הדלק דגל "יש macro".
- .5. (קיימת הגדרת מקרו) הכנס לטבלת שורות מקרו את שם המקרו (לדוגמא b1.m).
- .6. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 9 (סיום).
- .7. האם "יש macro" דולק ולא זזהה תווית endm הכנס את השורה לטבלת המקור ומחק את השורה הנ"ל מהקובץ. אחרת (לא מקרו) חוזר ל.1.
- .8. כבה דגל "יש macro". חוזר ל- 1. (סיום שמירת הגדרת מקרו)
- .9. סיום : שמירת קובץ מקרו פרוש.

cut לאחר פרישת כל המקוראים ניתן לעבור לשלב התרגום לקוד מכונה, שלב האסמלר.

אסמלר עם שני מעברים

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

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

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

האסמבלר מחזיק טבלה שבה רשומים כל שמות הפעולה של הhorאות והקודים הבינאריים (opcode, funct) המתאימים להם, וכן שמות הפעולות ניתנים להמרה לבינארי בקלות. כאשר נקרא שם פעולה, אפשר פשוט לעיין בטבלה ולמצוא את הקידוד הבינארי.

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

למשל, בקוד לעיל, האסמבלר אינו יכול לדעת שהסמל END אמרו להיות משוייך למן 140 (עשרוני), והסמל K אמרו להיות משוייך למן 149, אלא רק לאחר שנקראו כל שורות התוכנית.

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

במעבר הראשון נבנית טבלת סמלים בזיכרון, ובה לכל סמל שבתוכנית המקור משוייך ערך מסווג, שהוא ממן בזיכרון.

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

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

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

لتשומת לב: תפקיד האסמבלר, על שני המעברים שלו, לתרגם קובץ מקור לקוד בשפת מכונה. בגמר פעולת האסמבלר, התוכנית טרם מוכנה לטיענה לזיכרון לצורך ביצוע. קוד המכונה חייב לעبور לשלב הקישור/טיענה, ורק לאחר מכן לשלב הביצוע (שלבים אלה אינם חלק מהממע"ז).

המעבר הראשון

במעבר הראשון נדרשים כלליים כדי לקבוע איזה ממן ישוייך לכל סמל. העיקרונו הבסיסי הוא לספור את המיקומות בזיכרון, אותן תופסות הhorאות. אם כל הוראהティען בזיכרון למקום העוקב להוראה הקודמת, תציין ספרירה כזואת את ממן ההוראה הבאה. הספרירה נעשית על ידי האסמבלר ומוחזקת במבנה הhorאות (IC). ערכו ההתחלתי של IC הוא 100 (עשרוני), ולכן קוד המכונה של ההוראה הראשונה נבנה כך שייטען לזיכרון החל ממן 100. ה-IC מתעדכן בכל שורה הוראה המקצתה מקום בזיכרון. לאחר שהאסמבלר קובע מהו אורך ההוראה, ה-IC מוגדל במספר התאים (מיילים) הנתפסים על ידי ההוראה, וכך הוא מצביע על התא הפוי הבא.

כאמור, כדי לקודד את הhorאות בשפת מכונה, מחזיק האסמבלר טבלה, שיש בה קידוד מתאים לכל שם פעולה. בזמן התרגומים מחליף האסמבלר כל שם פעולה בקידוד שלה. כמו כן, כל אופרנד מוחלף בקידוד מתאים, אך פועלות החלפה זו אינה כה פשוטה. הhorאות משתמשות בשיטות מייען מגוונות לאופרנדים. אותה פעולה יכולה לקבל משמעותות שונות, בכל אחת ממשיטות המייען, וכן יתאיםו לה קידודים שונים לפי שיטות המייען. לדוגמה, פעולה ההזזה sow יכולה להתיחס

להעתיקת תוכן תא זיכרונו לאוגר, או להעתיקת תוכן אוגר לאוגר אחר, וכן הלאה. לכל אפשרות כזו את של **7 סוטו** עשוי להתאים קידוד שונה.

על האסטמבלר לסרוק את שורת ההוראה בשמורה, ולהחליט לגבי הקידוד לפי האופרנדים. בדרך כלל מתחלק הקידוד לשדה של שם הפעלה, ושדות נוספים המכילים מידע לגבי שיטות המיעון. כל השדות ביחיד דורשים מילה אחת או יותר בקוד המכונה.

כאשר נתקל האסטמבלר בתווית המופיעעה בתחילת השורה, הוא יודע שלפנוי הגדרה של תווית, ואז הוא משיק לה מעו – תוכנו הנוכחי של IC. כך מקבלות כל התוויות את מעניין בעת ההגדרה. תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנוסף לשם התווית גם את המعن ומאפיינים נוספים. כאשר תהייה התייחסות לתווית באופרנד של הוראה כלשהי, יוכל האסטמבלר לשולף את המعن המתאים מטבלת הסמלים.

הוראה יכולה להתיחס גם לסמל שטרם הוגדר עד כה בתוכנית, אלא יוגדר רק בהמשך התוכנית. להלן, לדוגמה, הוראות הסתעפות מען שמוגדר על ידי התווית A שופיעעה רק בהמשך הקוד :

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

כאשר מגיע האסטמבלר לשורת ההסתעפות (bne A), הוא טרם נתקל בהגדרת התווית A וכמוון לא יודע את המعن המשוייך לתווית. לכן האסטמבלר לא יוכל לבנות את הקידוד הבינארי של האופרנד A. נראה בהמשך כיצד נפתרת בעיה זו.

בכל הוראה, תמיד אפשר לבנות בעבר הראשון את הקידוד הבינארי המלא של המילה הראשונה של כל הוראה, את הקידוד הבינארי של מילת-המידע הנוספת של אופרנד מיידי, או אוגר, וכן את הקידוד הבינארי של כל הנתונים (המתפלבים מהנהניות .string, .data, .).

המעבר השני

ראינו שבמעבר הראשון, האסטמבלר אינו יכול לבנות את קוד המכמה של אופרנדים המשמשים בסמלים שעדיין לא הוגדרו. רק לאחר שהאסטמבלר עבר על כל התוכנית, כך שכל הסמלים נכנסו כבר לטבלת הסמלים, יוכל האסטמבלר להשלים את קוד המכמה של כל האופרנדים.

לשס כך מבצע האסטמבלר מעבר נוסף (מעבר שני) על כל קובץ המקור, ומעדכן את קוד המכמה של האופרנדים המשמשים בסמלים, באמצעות ערכי הסמלים מטבלת הסמלים. בסוף המעבר השני, תהיה התוכנית מתורגמת בשמורה לקוד מכונה.

הפרדת הוראות ונתונים

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

אחת הסכנות הטמונה באירוע הפרדת ההוראות מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית בתוכנית, לנסות "לבצע" את הנתונים כאלו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום לתופעה כזו היא הסתעפות לא נcona. התוכנית כموון לא תעבד נכון, אך לרוב הנזק הוא יותר חמוץ, כי נוצרת חריגת חומרה ברגע שהמעבד מבצע פעולה שאינה חוקית.

האסטמבלר שלנו חייב להפריד, בקוד המכמה שהוא מיצר, בין קטע הנתונים לקטע ההוראות. כולם בקובץ הפלט (בקוד המcono) תהיה הפרדה של הוראות ונתונים לשני קטעים נפרדים, אם כי בקובץ הקלט אין חובה שתהייה הפרדה כזו. בהמשך מתואר אלגוריתם של האסטמבלר, ובו פרטים כיצד מבצע את הפרדה.

גilio שגיאות בתוכנית המקור

כפי שהוסבר לעיל, הנחת המטלה היא שאין שגיאות בהגדרות המקור, ולכן שלב קדם לאסמבלר אינו מכיל שלב גilio שגיאות, לעומת זאת האסמבller אמר לגולות ולדוח על שגיאות בתחריב של תוכנית המקור, כגון פעולה שאינה קיימת, מספר אופרנדים שגוי, סוג אופרנד שאינו מותאים לפעולה, שם אוגר לא קיים, ועוד שגיאות אחרות. כמו כן מודא האסמבller שככל סמל מוגדר פעם אחת בדוק.

מכאן, שככל שגיאה המתגלת על ידי האסמבller נגרמת (בדרכ כלל) על ידי שורת קלט מסויימת.

לדוגמה, אם מופיעים שני אופרנדים בהוראה שאמור להיות בה רק אופרנד אחד, האסמבller ייתן הودעת שגיאה בנוסח "יותר מדי אופרנדים".

הערה: אם יש שגיאה בקוד האסמבלי בגוף מקשו, הרי שגיאה זו יכולה להופיע ולהתגלות שוב ושוב, בכל מקום בו נפרש המקור. נשים לב לכך האסמבller בודק שגיאות, כבר לא ניתן להזוז שזה קוד שנפרש ממקרו, כך שלא ניתן לחסוך גilio שגיאה כפולים.

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

لتשומת לב: האסמבller אין עוצר את פעולתו אחרי שנמצאה השגיאה הראשונה, אלא ממשיך לעבור על הקלט כדי לגלות שגיאות נוספות, ככל שישן. כמו כן אין 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 הנתונה:

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

תהליך העבודה של האסמבller

נתאר כעט את אופן העבודה של האסמבller. בהמשך, יוצג אלגוריתם שלדי מעבר ראשון ושני.

האסמבller מתחזק' שני מערכיים, שייקראו להן **תמונה ההוראות (code)** ו**תמונה הנתונים (data)**. מערכיים אלו נתונים למעשה תמונה של זיכרון המכונה (כל איבר במערך הוא בגודל מילה של המכונה, כולל 24 סיביות). במערך ההוראות בונה האסמבller את הקידוד של הוראות המכונה שנקראו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמבller את קידוד הנתונים שנקראו מקובץ המקור (שורות הנוכחי מסוג 'data.' ו-'string').

האסמבלר משתמש בשני מונחים, שנקראים IC (מונה ההוראות - Instruction-Counter) ו- DC (Data-Counter). מונחים אלו מצביעים על המיקום הבא הפניו במערך ההוראות (מונה הنتائج - Data-Counter). מונחים אלו מצביעים על המיקום הבא הפניו במערך ההוראות ובמערך הنتائج, בהתאם. בכל פעם כשמתחליל האסמבלר לעבור על קובץ מקור, המונה IC מקבל ערך התחלתי 0, והמונה DC מקבל ערך התחלתי 0. הערך ההתחלתי IC=100 נקבע כדי שקוד המכונה של התוכנית יתאים לטעינה ל זיכרון (לצורך ריצחה) החל מכתובת 100.

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

במעבר הראשון האסמבלר בונה את טבלת הסמלים ואת השילד של תמונה הזיכרון (הוראות וنتائج).

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

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

2. שורת הוראה :

האסמבלר מנתח את השורה ומפענה מהי ההוראה, ומהן שיטות המיעון של האופרנדים. מספר האופרנדים נקבע בהתאם להוראה שנמצאה. שיטות המיעון נקבעות בהתאם לחברן של כל אופרנד, כפי שהוסבר לעיל במפרט שיטות המיעון. למשל, התו # מציין מיעון מיידי, תווית מצינית מיעון ישיר, שם של אוצר מציין מיעון אוצר ישיר, ועוד.

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

כעת האסמבלר קובע לכל אופרנד את ערכו באופן הבא :

- אם זה אוצר – האופרנד הוא מספר האוצר.
- אם זו תווית (מיון ישיר) – האופרנד הוא ערך התווית כפי שמופיע בטבלת הסמלים (ייתכן והסמל טרם נמצא בטבלת הסמלים, במידה והוא יוגדר רק בהמשך התוכנית).
- אם זה התו # ואחריו מספר (מיון מיידי) – האופרנד הוא המספר עצמו.
- אם זו שיטת מיעון אחרת – ערכו של האופרנד נקבע לפי המפרט של שיטת המיעון (ראו תאור שיטות המיעון לעיל)

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

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

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

3. שורת הנחיה :

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

I. '.data'

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

אם בשורה '.data' מוגדרת גם תווית, אז התיוית מוכנסת לטבלת הסמלים. ערך התיוית הוא ערך מונה הנתונים DC שלפניהם הכנסת המספרים למערך. המאפיין של התיוית הוא '.data'.

II. '.string'

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

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

III. '.entry'

זהי הנחה לאסמבller לאפיון את התיוית הנתונה כאופrndכ-entry בטבלת הסמלים. בעת הפיקט קבצי הפלט (ראו בהמשך), התיוית תירשם בקובץ ה-.entries. אחדותם לב: זה לא נחש כשגיאה אם בקובץ המקור מופיעה יותר מהנחהית entry. אחדותם עם אותה תווית כאופrnd. המופיעים הנוספים אינם מושיפים דבר, אך גם אינם מפריעים.

IV. '.extern'

זהי הצהרה על סמל (תיוית) המוגדר בקובץ מקור אחר, והקובץ הנוichi עושה בו שימוש. האסמבller מכניס את הסמל המופיע כאופrnd לטבלת הסמלים, עם הערך 0 (הערך האמיiy לא ידוע, וייקבע רק בשלב הקישור), ועם המאפיין external. לא ידוע באיזה קובץ נמצאת הגדרת הסמל, ואין זה רלוונטי עבור האסמבller.

لتשומת לב: זה לא נחש כשגיאה אם בקובץ המקור מופיעה יותר מהנחהית extern. אחדותם עם אותה תווית כאופrnd. המופיעים הנוספים אינם מושיפים דבר, אך גם אינם מפריעים.

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

בסוף המעבר הראשון, האסמבller מעדכן בטבלת הסמלים כל סמל המופיע כ-'data', על ידי החוספת (100 + IC) לערכו של הסמל. הסיבה לכך היא שבתמונה הכלולת של קוד המכונה, תמונה הנתונים מופרdatת מתמונה ההוראות, וכל הנתונים נדרשים להופיע בקוד המכונה אחרי כל ההוראות. סמל מסווג data הוא תווית בתמונה הנתונים, והעדכו מושיף לערך הסמל (כלומר לכתובתו בזיכרון) את האורך הכולל של תמונה ההוראות, בתוספת כתובת התחלה הטעינה של הקוד, שהיא 100.

טבלת הסמלים מכילה כתעת ערci כל הסמלים הנחוצים להשלמת תמונה הזיכרון (למעט ערciים של סמלים חיצוניים).

במעבר השני, האסמבller משלים באמצעות טבלת הסמלים את קידוד כל המילים במערך ההוראות שטרם קודזו במעבר הראשון. במודל המכונה שלנו אין מילות-מידע נוספת של ההוראות, אשר מקודדות אופrnd בשיטת מיעון ישיר או יחסי.

אלגוריתם שלדי של האסמבller

לחידוד ההבנה של תהליך העבודה של האסמבller, נציג להלן אלגוריתם שלדי למעבר הראשון ולמעבר השני.

لتשומת לב: אין חובה להשתמש דווקא באלגוריתם זה.

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

בנייה את קוד המכונה כך שיתאים לטעינה לזכרון החל מכתובת 100.

בכל מעבר מתחילה לקרוא את קובץ המקור מהתחלתו.

מעבר ראשיו

- .1. אתחל $DC \leftarrow 0, IC \leftarrow 100$.
- .2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עברו ל-17.
- .3. האם השדה הראשון בשורה הוא תווית? אם לא, עברו ל-5.
- .4. הדלק דגל "יש הגדרת סמל".
- .5. האם זהה הנקודה לאחסון נתונים, כלומר, האם הנקודה `data.string`? אם לא, עברו ל-8.
- .6. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם המאפיין `data`. ערך הסמל יהיה בסיס והיסטט. (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה, יש להודיע על שגיאה).
- .7. זהה את סוג הנתונים, קודו אותם בתמונת הנתונים, והגדל את מונה הנתונים `DC` על ידי הוספת האורך הכולל של הנתונים שהוגדרו בשורה הנוכחית. חזור ל-2.
- .8. האם זו הנקודה `extern`. או הנקודה `entry`? אם לא, עברו ל-11.
- .9. אם זהה הנקודה `entry`. חזור ל-2 (הණיה תטופל במעבר השני).
- .10. אם זו הנקודה `extern`, הכנס את הסמל המופיע כאופרנד של הנקודה לטז טבלת הסמלים עם פעמים הערך 0 (בסיס והיסטט), ועם המאפיין `external`. (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה ללא המאפיין `external`, יש להודיע על שגיאה). חזור ל-2.
- .11. זהה שורת הוראה. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם המאפיין `code`. ערכו של הסמל יהיה בסיס והיסטט (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה, יש להודיע על שגיאה).
- .12. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא, אז הודיע על שגיאה בשם הוראה.
- .13. נתח את מבנה האופרנדים של ההוראה, וחשב מהו מספר המילים הכוללתופסת ההוראה בקוד המכונה (נקרא למספר זה `L`).
- .14. בנה כעת את הקוד הבינארי של המילה הראשונה של ההוראה, ושל כל מילת- מידע נוספת המוקודדת אופרנד במייעון מיידי. אפשר לקוד גם את המילה השנייה בקוד ההוראה (אם קיימת).
- .15. שומר את הערכים `IC` ו- `L` יחד עם נתונים קוד המכונה של ההוראה.
- .16. עדכן $IC \leftarrow IC + L$, וחזור ל-2.
- .17. קובץ המקור נקרא בשלמותו. 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 המופיעים כ- `data`, ע"י שימוש בערך `ICF` באופן הבא: ראשית יש לחשב את ערכו המלא המעודכן של הסמל בהינתן `ICF+בסיס+היסטוריה הנוכחיים` (ככל שהוא מופיע כך בטבלה), ואז לחשב בסיס+היסטוריה חדשים.
- .20. התחל מעבר שני.

מעבר שני

- .1. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עברו ל-7.
- .2. האם השדה הראשון בשורה הוא סמל (תווית), דגל עליו.
- .3. האם זהה הנקודה `data`. או `extern`. או `string`? אם כן, חזור ל-1.
- .4. האם זהה הנקודה `entry`? אם לא, עברו ל-6.
- .5. הוסף בטבלת הסמלים את המאפיין `entry` למאפייני הסמל המופיע כאופרנד של הנקודה (אם הסמל לא נמצא בטבלת הסמלים, יש להודיע על שגיאה). חזור ל-1.
- .6. השלם את הקידוד הבינארי של מילות-המידע של האופרנדים, בהתאם לשיטות המייעון בשימוש. לכל אופרנד בקוד המקור המכיל סמל, מצא את ערכו של הסמל בטבלת הסמלים (אם הסמל לא נמצא בטבלה, יש להודיע על שגיאה). אם הסמל מאופיין `external`, הוסף את כתובת מילת-המידע הרלוונטי לרשימת מילות- מידע שמתיחסות לסמל חיצוני. לפי הចורך, לחישוב הקידוד והכתובות, אפשר להיעזר בערכים `IC` ו- `L` של ההוראה, כפי ששמרו במעבר

- הראשון. חוזר ל-1. לתשומת לב: יש להשלים שתי מילות מידע לכל אופrnd. כמו כן, אם מדובר בסמל חיצוני, יש לרשום בקובץ ext את הכתובות של שתי מילות המידע. לפי הגדרת השפה, כתובות מילת היחס תהייה תמיד עוקבת לכתובות מילת הבסיס (דוגמת קובץ ext בהמשך).
7. קובץ המקור נקרא שלמותו. אם נמצא שגיאות במעבר השני, עזרו כאן.
8. בנה את קבצי הפלט (פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו לעיל **(לאחר שלב פרישת המקראים)**, ונציג את הקוד הבינארי שמתתקבל במעבר ראשון ובעבר שני. להלן שוב תוכנית הדוגמה.

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

בוצע מעבר ראשון על הקוד לעיל, ובנעה את טבלת הסמלים. כמו כן, נשלים במעבר זה את הקידוד של כל תMOVNT הנטוונים, ושל המילה הריאשונה של כל הוראה (ণשים לב שיש לקודד גם את המילה השניה). כמו כן, נקודד מילות-מידע מסוימות של כל הוראה, ככל שהקידוד זה אינו תלוי בערך של סמל. את מילות-המידע שעדיין לא ניתן לקודד במעבר הראשון נסמן ב'?' בדוגמה להלן.

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

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

טבלת הסמלים אחרי מעבר ראשון היא :

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

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

הערה : כאמור, האסmbלר בונה קוד מכונה כך שיתאים לטיעינה ל זיכרונו החל מכתובת 100 (עשורוני).
אם הטיעינה בפועל (לצורך ריצת התוכנית) תהיה לכתובות אחרות, יידרשו תיקונים בקוד הבינארי
בשלב הטיעינה, שיוכנסו בעורף מידע נוסף שהאסmbלר מכין בקבצי הפלט (ראו בהמשך).

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

טבלת הסמלים אחרי מעבר שני היא:

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

בסוף המעבר השני, אם לא נתגלו שגיאות, האסטבלר בונה את קבצי הפלט (ראו בהמשך), שמכילים את הקוד הבינארי ומידע נוסף עברו שלבי הקישור והטיענה. כאמור, שלבי הקישור והטיענה אינם למיושם בפרקיקט זה, ולא נדונם בהם כאן.

קבצי קלט ופלט של האסמבולר

בhfعلת hf של האסmbלר, יש להעיבר אליו באמצעות ARGUMENTS של שורת הפוקודה (command line arguments) רשימה של שמות קבצי מקור (אחד או יותר). אלו הם קבצי TEXT, וביהם תוכניות בתחריר של שפת האסmbלר שהוגדרה במא"ן זה.

האסטמבלר פועל על כל קוביץ מקור בנפרד, ויוצר עבورو קבצי פלט כדלקמן:

- קובץ `am`, המכיל את קבוע המקור לאחר שלב קדם האסמבולר (לאחר פרישת המקרואים)
 - קובץ `object`, המכיל את קוד המכוונה.
 - קובץ `externals`, ובו פרטים על כל המקומות (הכתובות) בקוד המכוונה בהם יש מילת- מידע שמקודדת ערך של סמל שהוצהר בחיצוני (סמל שהופיע כאופrnd של הנחיתת `extern`., ומופיעין בטבלת הסמלים כ- `external`.)
 - קובץ `entries`, ובו פרטים על כל סמל שמצוחר כנקודות כניסה (סמל שהופיע כאופrnd של הנחיתת `entry`., ומופיעין בטבלת הסמלים כ- `entry`.)

אם אין בקובץ המקור אף הנקיטת `extern`, האסמבולר לא יוצר את קובץ הפלט מסוג `externals`.
אם אין בקובץ המקור אף הנקיטת `entry`, האסמבולר לא יוצר את קובץ הפלט מסוג `entries`.

שמות קבועים המכוונים למקומות ספציפיים בזיכרון. למשל, השמות `x.as`, `y.as` ו-`hello.as` הם שמות קבועים. העברת שמות הקבצים הללו כארגומנטים לאסמבילר נעשית ללא ציון הסימוכין.

לדוגמא: נניח שתוכנית האסמבילר שלנו נקראת assembler, אזי שורת הפקודה הבאה:

```
assembler x y hello
```

תריצ' את האסמבולר על הקבצים : x.as, y.as, hello.as

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

לדוגמה, בהפעלת האסםבלר באמצעות שורת הפקודה : `x assembler` יוצר קובץ פלט `ob.x`, וכן קבצי פלט `ext.x` ו- `entry`. ככל שיש הנחיה `extern`. או בקובץ המקור. אם אין מאקרו בקובץ המקור, אז ייקובץ `am.as`.

פורמט קובץ ה-object

קובץ זה מכיל את תמונה הזיכרון של קוד המכונה, בשתי חלקים: תמונה ההוראות ראשונה, ואחריה ובצמוד לתמונה הנטוניות.

כזכור, האסמלר מקודד את ההוראות כך שתמונה ההוראות תתאים לטעינה החל מכתובת 100 (עשורוני) בזיכרון. נשים לב שرك המעבר הראשון יודיעים מהו הגולל הכללי של תמונה ההוראות. מכיוון שתמונה הנטוניות נמצאת אחרי תמונה ההוראות, גודל תמונה ההוראות משפיע על הכתובת בתמונה הנטוניות. זו הסיבה שבגללה היה צריך לעדכן בטבלת הסמלים, בסוף המעבר הראשון, את ערכי הסמלים המאופיינים כ-data (כזכור, באлогוריתם השודי שהוצע לעיל, בצעד 19, הוספנו לכל סמל כזה את הערך ICF). במעבר השני, בהשלמת הקיזוז של מילוט-המידע, משתמשים בערכים המעודכנים של הסמלים, המותאמים למבנה המלא וה壽פי של תמונה הזיכרון.

כעת האסמלר יוכל לכתוב את תמונה הזיכרון בשלמותה לתוך קובץ פلت (קובץ ה-object).

השורה הראשונה בקובץ ה-object היא "cotratt", המכילה שני מספרים (בסיס עשרוני): הראשון הוא האורך הכללי של תמונה ההוראות (במילוט זיכרון), והשני הוא האורך הכללי של תמונה הנטוניות (במילוט זיכרון). בין שני המספרים מפריד רווח אחד. כזכור, במעבר הראשון, בצעד 19, נשמרו ערכי הסמלים תוך שימוש ב-ICF.

השורות הבאות בקובץ מכילות את תמונה הזיכרון. בכל שורה זוג שדות: כתובות של מילה בזיכרון, ותוכן המילה, הכתובת תירשם בסיסי עשורוני באربע ספרות (כולל אפסים מוביילים). תוכן המילה יירשם בסיסי "מיוחד" ב-3 ספרות (כולל אפסים מוביילים). בין השדות בשורה יש רווח אחד.

"בסיס מיוחד"

כל שורה בתמונה הזיכרון היא באורך 20 סיביות, החל מסיבית 0 (מיימין) ועד לסיבית 19 (משמאלי). נחلك את 20 הסיביות ל-5 קובוצות בנות 4 סיביות בכל קבוצה כך:

- סיביות 19-16 יקראו קבוצה A
- סיביות 15-12 יקראו קבוצה B
- סיביות 11-8 יקראו קבוצה C
- סיביות 7-4 יקראו קבוצה D
- סיביות 3-0 יקראו קבוצה E

כל 4 סיביות של קבוצה מסוימת יומרו בספרה הקסדצימלית וייכתו לקובץ ה-OB באופן הבא:
תחיליה ייכתב שם הקבוצה באות גדולה, ולאחריו הייצוג הקסדצימלי של סיביות הקבוצה. כך ייכתו כל הסיביות מכל הקבוצות עם הפרדה של תו מקף (-) בין קבוצה לקבוצה.

לדוגמה, נסתכל על 20 הסיביות הבאות:

0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

הם ייכתו לקובץ ה-OB כך:

A4-B0-C3-Dc-E1

פורמט קובץ ה-entries

קובץ ה-entries בנוי משורות טקסט, שורה אחת לכל סמל שמאופיין בטבלת הסמלים כ-entry. בשורה מופיע שמו הסמלי, ולאחריו כתובות הבסיס שלו וההיסט, כפי שנקבע בטבלת הסמלים (בסיסי עשורוני באربע ספרות, כולל אפסים מוביילים). אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה. בין השdots בשורה יש פסיק אחד.

פורמט קובץ ה- **externals**

קובץ-h externals בנוי אף הוא משורות טקסט, שורה לכל כתובות בקובץ המכונה בה יש מילת מדע המתיחסת לסמל שמאופיין כ- **external**. כזכור, רשיימה של מילות- מידע אלה נבנתה במעבר השני (צעד 6 באלגוריתם השלדי).

כל שורה בקובץ-h externals מכילה את שם הסמל החיצוני, ולאחריו המילה BASE ולאחריה הכתובת של מילת-המידע בקובץ המכונה בה נדרש כתובות הבסיס (בסיס עשרוני באربע ספרות, כולל אפסים מוביילים). ולאחר מכן, שורה נוספת המכילה את שם הסמל החיצוני, ולאחריו המילה OFFSET ולאחריה הכתובת של מילת-המידע בקובץ המכונה בה נדרש החישט (OFFSET) (בסיס עשרוני באربע ספרות, כולל אפסים מוביילים).

בין השdots בשורה יש רווח אחד. אין חשיבות לסדר השdots, כי כל שורה עומדת בפני עצמה.

لتשומת לב : יתכן ויש מספר כתובות בקובץ המכונה בהן מילות- המידע מתיחסות לאותו סמל חיצוני. לכל כתובות כזו תהיה שורה נפרדת בקובץ-h externals.

נדגים את הפלט שמייצר האסמבולר עבור קובץ מקור בשם **ps.as** שהודגם קודם לנו.

התוכנית לאחר שלב פרישת המקרו תיראה כך :

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

MAIN:      add    r3, LIST
LOOP:       prn   #48
            lea    STR, r6
            inc    r6
            mov    r3, W
            sub   r1, r4
            bne   END
            cmp   val1, #-6
            bne   END[r15]
            dec    K

.entry MAIN
            sub   LOOP[r10],r14
END:        stop
STR:        .string "abcd"
LIST:       .data  6, -9
            .data  -100
.entry K
K:          .data  31
.extern val1
```

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

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

טבלת הסמלים בגמר המעבר השני היא :

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

להלן תוכן קבצי הפלט של הדוגמה.

:הקובץ ps.ob

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

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

: ps.ent הקובץ

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

: ps.ext הקובץ

W BASE 115
W OFFSET 116

vall BASE 125
vall OFFSET 126

סיכום והנחיות כלליות

- גודל תוכנית המקור הניתנת כקלט לאסטמבלר אינו ידוע מראש, ולכן גם גודלו של קוד המוכנה אינו צפוי מראש. אולם בכך להקל במשמעות האסטמבלר, מותר להניח גודל מסוימלי. לפיכך יש אפשרות להשתמש במערכות לאחסון תומנות קוד המוכנה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים וטבלת המקור), יש למשם באופן ייעיל וחסכוני (למשל באמצעות רישימה מקושתת והקצתה זיכרון דינמי).
- השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הקלט, למעט הסיומות. למשל, אם קובץ הקלט הוא prog.as אז קבצי הפלט שיוצרו הם : prog.ob, prog.ext, prog.ent.
- מתכונת הפעלת האסטמבלר צריכה להיות כפי הנדרש בממ"ן, ללא שינוי כלשהו. כמובן, ממשך המשמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו לתוכנית האסטמבלר כארגומנטים (אחד או יותר) בשורת הפקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גרפיים למיניהם, ועוד'.
- יש להקפיד לחלק את שימוש האסטמבלר למספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסוימים שונים במודול יחיד. מומלץ לחלק למודולים כגון : מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחבירי של שורה), טבלת הסמלים, מפת הזיכרון, טבלאות קבועות (קוד הפעולה, שיטות המיעון החוקיות לכל פעולה, ועוד').
- יש להקפיד ולתעד את השימוש באופן מלא וברור, באמצעות הערות מפורטות בקוד.
- יש לאפשר תווים לבנים עודפים בקובץ הקלט בשפת אסטמבלר. למשל, אם בשורת הוראה יש שני אופרנדים המופרדים בפסיק, אזי לפני ואחרי הפסיק מותר שייהיו רווחים וטאבים בכל מקום. בדומה, גם לפני ואחרי שם הפעולה. מותר גם שורות ריקות. האסטמבלר יתעלם מתחומים לבנים מיותרים (כלומר ידלג עליהם).
- הקלט (קוד האסטמבלר) עלול להכיל שגיאות תחביריות. על האסטמבלר לגלות ולדווח על כל השורות השגויות בקלט. אין לעצור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להדפיס למסך הודעות מפורטות ככל הניתן, כדי שאפשר יהיה להבין מה והיכן כל שגיאה. כמובן שגם קובץ קלט מכיל שגיאות, אין טעם להפיק עבورو את קבצי הפלט (ob, ext, ent).

גם ונשלם פרק ההסבירים והגדרת הפרויקט.

בשאלות ניתן לפנות לקובוצת הדיוון באתר הקורס, ועל כל אחד מהמנחים בשעות הקבלה שלהם.

לזהכירים, באפשרותו של כל סטודנט לפנות לכל מנהה, לאו דווקא למנהל 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 לכם.

لتשומתיכם : לא תינטן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או מחלת ממושכת. במקרים אלו יש לבקש ולקבל אישור מראש מצוות הקורס.

בזה לך !