מטלת מנחה (ממ"ן) 14

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרויקט גמר

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

סמסטר: 2021א' | מועד אחרון להגשה: 14.3.2021

קיימות שתי חלופות להגשת מטלות:

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

הסבר מפורט ב"נוהל הגשת מטלות מנחה"

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

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

עליכם להגיש את הפריטים הבאים:

1. קבצי המקור של התוכנית שכתבתם (קבצים בעלי סיומת c. או h.).
2. קובץ הרצה (מקומפל ומקושר) עבור מערכת אובונטו.
3. קובץ makefile. הקימפול חייב להיות עם הקומפיילר gcc והדגלים: Wall -ansi -pedantic-. יש לנפות את כל ההודעות שמוציא הקומפיילר, כך שהתוכנית תתקמפל ללא כל הערות או אזהרות.
4. דוגמאות הרצה (קלט ופלט):

א. קבצי קלט בשפת אסמבלי, וקבצי הפלט שנוצרו מהפעלת האסמבלר על קבצי קלט אלה. יש להדגים שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האסמבלי.

ב. קבצי קלט בשפת אסמבלי המדגימים מגוון רחב של סוגי שגיאות אסמבלי (ולכן לא נוצרים קבצי פלט), ותדפיסי המסך המראים את הודעות השגיאה שמוציא האסמבלר.

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

נזכיר מספר היבטים חשובים של כתיבת קוד טוב:

1. הפשטה של מבני הנתונים: רצוי (ככל האפשר) להפריד בין הגישה למבני הנתונים לבין המימוש של מבני הנתונים. כך, למשל, בעת כתיבת פונקציות לטיפול בטבלה, אין זה מעניינם של המשתמשים בפונקציות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקושרת.

2. קריאות הקוד: יש להשתמש בשמות משמעותיים למשתנים ופונקציות. יש לערוך את הקוד באופן מסודר: הזחות עקביות, שורות ריקות להפרדה בין קטעי קוד, וכד'.

3. תיעוד: יש להכניס בקבצי המקור תיעוד תמציתי וברור, שיסביר את תפקידה של כל פונקציה (באמצעות הערות כותרת לכל פונקציה). כמו כן יש להסביר את תפקידם של משתנים חשובים. כמו כן, יש להכניס הערות ברמת פירוט טובה בכל הקוד.

הערה: תוכנית "עובדת", דהיינו תוכנית שמבצעת את כל הדרוש ממנה, אינה לכשעצמה ערובה לציון גבוה. כדי לקבל ציון גבוה, על התוכנית לעמוד בקריטריונים של כתיבה ותיעוד ברמה טובה, כמתואר לעיל, אשר משקלם המשותף מגיע עד לכ- 40% ממשקל הפרויקט.

מותר להשתמש בפרויקט בכל מגוון הספריות הסטנדרטיות של שפת C, אבל אין להשתמש בספריות חיצוניות אחרות.

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

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

רקע כללי ומטרת הפרויקט

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

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

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מגוון פעולות פשוטות, הנקראות **הוראות מכונה**, ולשם כך היא משתמשת באוגרים (registers) הקיימים בתוך היע"מ, ובזיכרון המחשב.
דוגמאות: העברת מספר מתא בזיכרון לאוגר ביע"מ או בחזרה, הוספת 1 למספר הנמצא באוגר, בדיקה האם מספר המאוחסן באוגר שווה לאפס, חיבור וחיסור בין שני אוגרים, וכד'.
הוראות המכונה ושילובים שלהן הן המרכיבות תוכנית כפי שהיא טעונה לזיכרון בזמן ריצתה.
כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנת), תתורגם בסופו של דבר באמצעות תוכנה מיוחדת לצורה סופית זו.

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

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

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

תפקידו של האסמבלר הוא לבנות קובץ המכיל קוד מכונה, מקובץ נתון של תוכנית הכתובה בשפת אסמבלי. זהו השלב הראשון במסלול אותו עוברת התוכנית, עד לקבלת קוד המוכן לריצה על חומרת המחשב. השלבים הבאים הם קישור (linkage) וטעינה (loading), אך בהם לא נעסוק בממ"ן זה.

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

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

המחשב הדמיוני ושפת האסמבלי

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

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

"חומרה":

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

למעבד 8 אוגרים כלליים, בשמות: r0, r1, r2, r3, r4, r5, r6, r7.
גודלו של כל אוגר הוא 12 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 11. שמות האוגרים נכתבים תמיד עם אות 'r' קטנה.

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

גודל הזיכרון הוא 4096 תאים, בכתובות 0-4095, וכל תא הוא בגודל של 12 סיביות . לתא בזיכרון נקרא גם בשם "**מילה**". הסיביות בכל מילה ממוספרות כמו באוגר.

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

מבנה הוראת המכונה:

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

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

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

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

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

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

להלן טבלת הפעולות:

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

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

להלן מפרט השדות במילה הראשונה בקוד המכונה של כל הוראה.

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

סיביות 4-7: שדה זה, הנקרא **funct**, מתפקד כאשר מדובר בפעולה שקוד-הפעולה (opcode) שלה משותף לכמה פעולות שונות (כאמור, קודי-פעולה 2, 5 או 9). השדה funct יכיל ערך ייחודי לכל פעולה מקבוצת הפעולות שיש להן אותו קוד-פעולה. אם קוד-הפעולה משמש לפעולה אחת בלבד, הסיביות של השדה funct יהיו מאופסות.

סיביות 2-3: מכילות את מספרה של שיטת המיעון של אופרנד המקור.
אם אין בהוראה אופרנד מקור, סיביות אלה יהיו מאופסות.
מפרט של שיטות המיעון השונות יינתן בהמשך.

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

<u>שיטות מיעון</u>:

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

השימוש בשיטות המיעון מצריך מילות-מידע נוספות בקוד המכונה של הוראה, בנוסף למילה הראשונה. לכל אופרנד של הוראה נדרשת **מילת-מידע אחת נוספת**. כאשר בהוראה יש שני אופרנדים, קודם תופיע מילת-המידע של אופרנד המקור, ולאחריה מילת-המידע של אופרנד היעד.

להלן המפרט של שיטות המיעון.

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	תחביר האופרנד באסמבלי	דוגמה
0	מיעון מיידי (immediate)	מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר שלם בשיטת המשלים ל-2, ברוחב של 12 סיביות	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני.	mov #-1, r2 בדוגמה זו האופרנד הראשון של ההוראה (אופרנד המקור) נתון בשיטת מיעון מיידי. ההוראה כותבת את הערך 1- אל אוגר r2.
1	מיעון ישיר (direct)	מילת-מידע נוספת של ההוראה מכילה כתובת בזיכרון. המילה בכתובת זו בזיכרון היא האופרנד. הכתובת מיוצגת כמספר <u>ללא סימן</u> ברוחב של 12 סיביות.	האופרנד הוא <u>תווית</u> שכבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת התווית בתחילת השורה של הנחית 'data.' או 'string.', או בתחילת השורה של הוראה, או באמצעות אופרנד של הנחית 'extern.'. התווית מייצגת באופן סימבולי כתובת בזיכרון.	השורה הבאה מגדירה את התווית x: x: .data 23 ההוראה: dec x מקטינה ב-1 את תוכן המילה שבכתובת x בזיכרון (ה"משתנה" x). הכתובת x מקודדת במילת-המידע הנוספת. <u>דוגמה נוספת</u>: ההוראה jmp next מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתתבצע נמצאת בכתובת next). הכתובת next מקודדת במילת-המידע הנוספת.
2	מיעון יחסי (relative)	שיטה זו רלוונטית <u>אך ורק</u> להוראות המבצעות קפיצה (הסתעפות) להוראה אחרת. מדובר בהוראות עם קוד-פעולה 9 בלבד: jmp, bne, jsr. <u>לא ניתן</u> להשתמש בשיטה זו בהוראות עם קודי-פעולה אחרים. בשיטה זו, יש בקידוד ההוראה מילת מידע נוספת, המכילה את מרחק הקפיצה, במילות זיכרון, ממילת-המידע הנוכחית אל המילה הראשונה של ההוראה המבוקשת (ההוראה הבאה לביצוע). מרחק הקפיצה מיוצג כמספר עם סימן בשיטת המשלים ל-2 ברוחב של 12 סיביות. מרחק זה יהיה שלילי אם הקפיצה היא אל הוראה בכתובת יותר נמוכה, וחיובי אם הקפיצה היא אל הוראה בכתובת יותר גבוהה.	האופרנד מתחיל בתו % ולאחריו ובצמוד אליו מופיע שם של תווית. התווית מייצגת באופן סימבולי כתובת של <u>הוראה בקובץ המקור הנוכחי של התוכנית</u>. ייתכן שהתווית כבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת התווית בתחילת שורת הוראה. יודגש כי בשיטת מיעון יחסי <u>לא ניתן</u> להשתמש בתווית (כתובת) שמוגדרת בקובץ מקור אחר (כתובת חיצונית).	jmp %next בדוגמה זו, ההוראה jmp מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתתבצע נמצאת בכתובת next). נניח, לדוגמה, כי ההוראה jmp שבדוגמה נמצאת בכתובת 500 (עשרוני). כמו כן, נניח כי התווית next מוגדרת בקובץ המקור הנוכחי בכתובת 300 (עשרוני). מרחק הקפיצה ממילת-המידע של ההוראה jmp אל הכתובת next הוא 300-(500+1)=-201, ולכן המילה הנוספת תכיל את הערך 201-.

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

<u>מפרט הוראות המכונה:</u>

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

הוראות המכונה מתחלקות לשלוש קבוצות, לפי מספר האופרנדים הנדרשים לפעולה.

קבוצת ההוראות הראשונה:
אלו הן הוראות המקבלות שני אופרנדים.
ההוראות השייכות לקבוצה זו הן: mov, cmp, add, sub, lea

הוראה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
mov	0		מבצעת העתקה של תוכן אופרנד המקור (האופרנד הראשון) אל אופרנד היעד (האופרנד השני).	mov A, r1	העתק את תוכן המשתנה A (המילה שבכתובת A בזיכרון) אל אוגר r1.
cmp	1		מבצעת השוואה בין שני האופרנדים. ערך אופרנד היעד (השני) מופחת מערך אופרנד המקור (הראשון), ללא שמירת תוצאת החיסור. פעולת החיסור מעדכנת דגל בשם Z ("דגל האפס") באוגר הסטטוס (PSW).	cmp A, r1	אם תוכן המשתנה A זהה לתוכנו של אוגר r1 אזי הדגל Z ("דגל האפס") באוגר הסטטוס (PSW) יודלק, אחרת הדגל יאופס.
add	2	10	אופרנד היעד (השני) מקבל את תוצאת החיבור של אופרנד המקור (הראשון) והיעד (השני).	add A, r0	אוגר r0 מקבל את תוצאת החיבור של תוכן המשתנה A ותוכנו הנוכחי של r0.
sub	2	11	אופרנד היעד (השני) מקבל את תוצאת החיסור של אופרנד המקור (הראשון) מאופרנד היעד (השני).	sub #3, r1	אוגר r1 מקבל את תוצאת החיסור של הקבוע 3 מתוכנו הנוכחי של האוגר r1.
lea	4		lea הוא קיצור (ראשי תיבות) של load effective address. פעולה זו מציבה את מען הזיכרון המיוצג על ידי התווית שבאופרנד הראשון (המקור), אל האופרנד השני (היעד).	lea HELLO, r1	המען שמייצגת התווית HELLO מוצב לאוגר r1.

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

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

הוראה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
clr	5	10	איפוס תוכן האופרנד.	clr r2	האוגר r2 מקבל את הערך 0.
not	5	11	היפוך הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0).	not r2	כל ביט באוגר r2 מתהפך.
inc	5	12	הגדלת תוכן האופרנד באחד.	inc r2	תוכן האוגר r2 מוגדל ב- 1.
dec	5	13	הקטנת תוכן האופרנד באחד.	dec Count	תוכן המשתנה Count מוקטן ב- 1.
jmp	9	10	קפיצה (הסתעפות) בלתי מותנית אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כלומר, כתוצאה מביצוע ההוראה, מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה.	jmp %Line	PC←PC+distanceTo(Line) בשיטת מיעון יחסי, המרחק לתווית Line מתווסף למצביע התכנית ולפיכך ההוראה הבאה שתתבצע תהיה במען Line.
bne	9	11	bne הוא קיצור (ראשי תיבות) של: branch if not equal (to zero). זוהי הוראת הסתעפות מותנית. אם ערכו של הדגל Z באוגר הסטטוס (PSW) הינו 0, אזי מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה. כזכור, הדגל Z נקבע באמצעות הוראת cmp.	bne Line	אם ערך הדגל Z באוגר הסטטוס (PSW) הוא 0, אזי PC ← address(Line) מצביע התכנית יקבל את כתובת התווית Line, ולפיכך ההוראה הבאה שתתבצע תהיה במען Line.
jsr	9	12	קריאה לשגרה (סברוטינה). כתובת ההוראה שאחרי הוראת jsr הנוכחית (PC+2) נדחפת לתוך המחסנית שבזיכרון המחשב, ומצביע התוכנית (PC) מקבל את כתובת השגרה. הערה: חזרה מהשגרה מתבצעת באמצעות הוראת rts, תוך שימוש בכתובת שבמחסנית.	jsr SUBR	push(PC+2) PC ← address(SUBR) מצביע התכנית יקבל את כתובת התווית SUBR, ולפיכך, ההוראה הבאה שתתבצע תהיה במען SUBR. כתובת החזרה מהשגרה נשמרת במחסנית.
red	12		קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מהקלט ייכנס לאוגר r1.
prn	13		הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn r1	יודפס לפלט התו (קוד ascii) הנמצא באוגר r1

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

ההוראות השייכות לקבוצה זו הן: rts, stop.

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

מבנה תכנית בשפת אסמבלי:

תכנית בשפת אסמבלי בנויה ממשפטים (statements). קובץ מקור בשפת אסמבלי מורכב משורות המכילות משפטים של השפה, כאשר כל משפט מופיע בשורה נפרדת. כלומר, ההפרדה בין משפט למשפט בקובץ המקור הינה באמצעות התו '\n' (שורה חדשה).

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

יש ארבעה סוגי משפטים (שורות בקובץ המקור) בשפת אסמבלי, והם:

סוג המשפט	הסבר כללי
משפט ריק	זוהי שורה המכילה אך ורק תווים לבנים (whitespace), כלומר רק את התווים ' ' ו- '\t' (רווחים וטאבים). ייתכן ובשורה אין אף תו (למעט התו \n), כלומר השורה ריקה.
משפט הערה	זוהי שורה בה התו הראשון הינו ';' (נקודה פסיק). על האסמבלר להתעלם לחלוטין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמבלר מה עליו לעשות כשהוא פועל על תוכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצאת זיכרון ואתחול משתנים של התוכנית, אך הוא אינו מייצר קידוד של הוראות מכונה המיועדות לביצוע בעת ריצת התוכנית.
משפט הוראה	זהו משפט המייצר קידוד של הוראות מכונה לביצוע בעת ריצת התוכנית. המשפט מורכב משם ההוראה (פעולה) שעל המעבד לבצע, והאופרנדים של ההוראה.

כעת נפרט יותר לגבי סוגי המשפטים השונים.

משפט הנחיה:

משפט הנחיה הוא בעל המבנה הבא:
בתחילת המשפט יכולה להופיע הגדרה של תווית (label). לתווית יש תחביר חוקי שיתואר בהמשך. התווית היא אופציונאלית.

לאחר מכן מופיע שם ההנחיה. לאחר שם ההנחיה יופיעו פרמטרים (מספר הפרמטרים בהתאם להנחיה).

שם של הנחיה מתחיל בתו '.' (נקודה) ולאחריו תווים באותיות קטנות (lower case) בלבד.

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

1. ההנחיה '.data'

הפרמטרים של ההנחיה '.data' הם מספרים שלמים חוקיים (אחד או יותר) המופרדים על ידי התו ',' (פסיק). לדוגמה:

.data 7, −57, +17, 9

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

המשפט '.data' מנחה את האסמבלר להקצות מקום בתמונת הנתונים (data image), אשר בו יאוחסנו הערכים של הפרמטרים, ולקדם את מונה הנתונים, בהתאם למספר הערכים. אם בהנחית data. מוגדרת תווית, אזי תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום),

ומוכנסת אל טבלת הסמלים. דבר זה מאפשר להתייחס אל מקום מסוים בתמונת הנתונים דרך שם התווית (למעשה, זוהי דרך להגדיר שם של משתנה).

כלומר אם נכתוב:

```
XYZ:   .data 7, –57, +17, 9
```

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

אם נכתוב בתוכנית את ההוראה:

```
mov  XYZ, r1
```

אזי בזמן ריצת התוכנית יוכנס לאוגר r1 הערך 7.

ואילו ההוראה:

```
lea XYZ, r1
```

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

2. ההנחיה '.string'

להנחיה '.string' פרמטר אחד, שהוא מחרוזת חוקית. תווי המחרוזת מקודדים לפי ערכי ה-ascii המתאימים, ומוכנסים אל תמונת הנתונים לפי סדרם, כל תו במילה נפרדת. בסוף המחרוזת יתווסף התו '\0' (הערך המספרי 0), המסמן את סוף המחרוזת. מונה הנתונים של האסמבלר יקודם בהתאם לאורך המחרוזת (בתוספת מקום אחד עבור התו המסיים). אם בשורת ההנחיה מוגדרת תווית, אזי תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום) ומוכנסת אל טבלת הסמלים, בדומה כפי שנעשה עבור '.data' (כלומר ערך התווית יהיה הכתובת בזיכרון שבה מתחילה המחרוזת).

לדוגמה, ההנחיה:

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

מקצה בתמונת הנתונים רצף של 7 מילים, ומאתחלת את המילים לקודי ה-ascii של התווים לפי הסדר במחרוזת, ולאחריהם הערך 0 לסימון סוף מחרוזת. התווית STR מזוהה עם כתובת התחלת המחרוזת.

3. ההנחיה '.entry'

להנחיה '.entry' פרמטר אחד, והוא שם של תווית המוגדרת בקובץ המקור הנוכחי (כלומר תווית שמקבלת את ערכה בקובץ זה). מטרת ההנחיה entry. היא לאפיין את התווית הזו באופן שיאפשר לקוד אסמבלי הנמצא בקבצי מקור אחרים להשתמש בה (כאופרנד של הוראה).

לדוגמה, השורות:

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

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

<u>לתשומת לב</u>: תווית המוגדרת בתחילת שורת entry. הינה חסרת משמעות והאסמבלר **מתעלם** מתווית זו (אפשר שהאסמבלר יוציא הודעת אזהרה).

4. ההנחיה '.extern'

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

נשים לב כי הנחיה זו תואמת להנחית 'entry.' המופיעה בקובץ בו מוגדרת התווית. בשלב הקישור תתבצע התאמה בין ערך התווית, כפי שנקבע בקוד המכונה של הקובץ שהגדיר את התווית, לבין קידוד ההוראות המשתמשות בתווית בקבצים אחרים (שלב הקישור אינו רלוונטי לממ"ן זה).

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

.extern HELLO

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

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

משפט הוראה:

משפט הוראה מורכב מהחלקים הבאים:

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

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

שם הפעולה תמיד באותיות קטנות (lower case), והוא אחת מ- 16 הפעולות שפורטו לעיל.

לאחר שם הפעולה יופיעו האופרנדים, בהתאם לסוג הפעולה. יש להפריד בין שם-הפעולה לבין האופרנד הראשון באמצעות רווחים ו/או טאבים (אחד או יותר).

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזה בתו ',' (פסיק). בדומה להנחיה 'data.', **לא חייבת להיות הצמדה של האופרנדים לפסיק**. כל כמות של רווחים ו/או טאבים משני צידי הפסיק היא חוקית.

למשפט הוראה עם שני אופרנדים המבנה הבא:

label: opcode source-operand, target-operand

לדוגמה:

HELLO: add r7, B

למשפט הוראה עם אופרנד אחד המבנה הבא:

label: opcode target-operand

לדוגמה:

HELLO: bne %XYZ

למשפט הוראה ללא אופרנדים המבנה הבא:

label: opcode

לדוגמה:

END: stop

אפיון השדות במשפטים של שפת האסמבלי

תווית:

תווית היא סמל שמוגדר בתחילת משפט הוראה' או בתחילת הנחיית data. או string.
תווית חוקית מתחילה באות אלפביתית (גדולה או קטנה), ולאחריה סדרה כלשהי של אותיות אלפביתיות (גדולות או קטנות) ו/או ספרות. האורך המקסימלי של תווית הוא 31 תווים.

<u>הגדרה של תווית</u> מסתיימת בתו ':' (נקודתיים). תו זה אינו מהווה חלק מהתווית, אלא רק סימן המציין את סוף ההגדרה. התו ':' חייב להיות צמוד לתווית (ללא רווחים).

אסור שאותה תווית תוגדר יותר מפעם אחת (כמובן בשורות שונות). אותיות קטנות וגדולות נחשבות <u>שונות זו מזו</u>.

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hEllo:
x:
He78902:

לתשומת לב: מילים שמורות של שפת האסמבלי (כלומר שם של פעולה או הנחיה, או שם של אוגר) אינן יכולות לשמש גם כשם של תווית. לדוגמה: הסמלים add, r3 לא יכולים לשמש כתוויות, אבל הסמלים Add, r9, R3 הם תוויות חוקיות.

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

לתשומת לב: מותר במשפט הוראה להשתמש באופרנד שהוא סמל שאינו מוגדר כתווית בקובץ הנוכחי, כל עוד הסמל מאופיין כחיצוני (באמצעות הנחיית extern. כלשהי בקובץ הנוכחי).

<u>מספר:</u>

מספר חוקי מתחיל בסימן אופציונלי: '–' או '+' ולאחריו סדרה של ספרות בבסיס עשרוני.
לדוגמה: 76, 5–, 123+ הם מספרים חוקיים. אין תמיכה בשפת האסמבלי שלנו בייצוג בבסיס אחר מאשר עשרוני, ואין תמיכה במספרים שאינם שלמים.

<u>מחרוזת:</u>

מחרוזת חוקית היא סדרת תווי ascii נראים (שניתנים להדפסה), המוקפים במרכאות כפולות (המרכאות אינן נחשבות חלק מהמחרוזת). דוגמה למחרוזת חוקית: "hello world".

<u>סימון המילים בקוד המכונה באמצעות המאפיין "A,R,E"</u>

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

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

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

- האות A (קיצור של Absolute) באה לציין שתוכן המילה אינו תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל, המילה הראשונה בכל הוראה, או מילת-מידע המכילה אופרנד מיידי).
- האות R (קיצור של Relocatable) באה לציין שתוכן המילה תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל, מילת-מידע המכילה כתובת של תווית המוגדרת בקובץ המקור הנוכחי).
- האות E (קיצור של External) באה לציין שתוכן המילה תלוי בערכו של סמל שאינו מוגדר בקובץ המקור הנוכחי (למשל, מילת-מידע המכילה ערך של סמל המופיע בהנחיית extern.).

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

אסמבלר עם שני מעברים

כאשר מקבל האסמבלר כקלט תוכנית בשפת אסמבלי, עליו לעבור על התוכנית פעמיים. במעבר הראשון, יש לזהות את הסמלים (תוויות) המופיעים בתוכנית, ולתת לכל סמל ערך מספרי שהוא המען בזיכרון שהסמל מייצג. במעבר השני, באמצעות ערכי הסמלים, וכן קודי-הפעולה ומספרי האוגרים, בונים את קוד המכונה.

לדוגמה: האסמבלר מקבל את התוכנית הבאה בשפת אסמבלי:

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

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

התרגום של תוכנית תכנית המקור שבדוגמה לקוד בינארי מוצג להלן:

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

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

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

כדי לבצע המרה לבינארי של אופרנדים שכתובים בשיטות מיעון המשתמשות בסמלים (תוויות), יש צורך לבנות טבלה המכילה את ערכי כל הסמלים. אולם בהבדל מהקודים של הפעולות, הידועים מראש, הרי המענים בזיכרון עבור הסמלים שבשימוש התוכנית אינם ידועים, עד אשר תוכנית המקור נסרקה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד לעיל, האסמבלר אינו יכול לדעת שהסמל END אמור להיות משויך למען 127 (עשרוני), ושהסמל K אמור להיות משויך למען 136, אלא רק לאחר שנקראו כל שורות התוכנית.

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

במעבר הראשון נבנית טבלת סמלים בזיכרון, ובה לכל סמל שבתוכנית המקור משויך ערך מספרי, שהוא מען בזיכרון.

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

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

Symbol	Value (decimal)	Attributes
MAIN	100	code
LOOP	103	code
END	127	code
STR	128	data
LIST	133	data
K	136	data, entry
val1	0	external

לתשומת לב: תפקיד האסמבלר, על שני המעברים שלו, לתרגם קובץ מקור לקוד בשפת מכונה. בגמר פעולת האסמבלר, התוכנית טרם מוכנה לטעינה לזיכרון לצורך ביצוע. קוד המכונה חייב לעבור לשלבי הקישור/טעינה, ורק לאחר מכן לשלב הביצוע (שלבים אלה אינם חלק מהממ"ן).

המעבר הראשון

במעבר הראשון נדרשים כללים כדי לקבוע איזה מען ישויך לכל סמל. העיקרון הבסיסי הוא לספור את המקומות בזיכרון, אותם תופסות ההוראות. אם כל הוראה תיטען בזיכרון למקום העוקב להוראה הקודמת, תציין ספירה כזאת את מען ההוראה הבאה. הספירה נעשית על ידי האסמבלר ומוחזקת במונה ההוראות (IC) . ערכו ההתחלתי של IC הוא 100 (עשרוני), ולכן קוד המכונה של ההוראה הראשונה נבנה כך שייטען לזיכרון החל ממען 100. ה-IC מתעדכן בכל שורת הוראה המקצה מקום בזיכרון. לאחר שהאסמבלר קובע מהו אורך ההוראה, ה-IC מוגדל במספר התאים (מילים) הנתפסים על ידי ההוראה, וכך הוא מצביע על התא הפנוי הבא.

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

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

כאשר נתקל האסמבלר בתווית המופיעה בתחילת השורה, הוא יודע שלפניו הגדרה של תווית, ואז הוא משייך לה מען – תוכנו הנוכחי של ה-IC. כך מקבלות כל התוויות את מעניהן בעת ההגדרה. תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנוסף לשם התווית גם את המען ומאפיינים נוספים. כאשר תהיה התייחסות לתווית באופרנד של הוראה כלשהי, יוכל האסמבלר לשלוף את המען המתאים מטבלת הסמלים.

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

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

כאשר מגיע האסמבלר לשורת ההסתעפות (bne A), הוא טרם נתקל בהגדרת התווית A וכמובן לא יודע את המען המשויך לתווית. לכן האסמבלר לא יכול לבנות את הקידוד הבינארי של האופרנד A. נראה בהמשך כיצד נפתרת בעיה זו.

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

המעבר השני

ראינו שבמעבר הראשון, האסמבלר אינו יכול לבנות את קוד המכונה של אופרנדים המשתמשים בסמלים שעדיין לא הוגדרו. רק לאחר שהאסמבלר עבר על כל התוכנית, כך שכל הסמלים נכנסו כבר לטבלת הסמלים, יכול האסמבלר להשלים את קוד המכונה של כל האופרנדים.

לשם כך מבצע האסמבלר מעבר נוסף (מעבר שני) על כל קובץ המקור, ומעדכן את קוד המכונה של האופרנדים המשתמשים בסמלים, באמצעות ערכי הסמלים מטבלת הסמלים.
בסוף המעבר השני, תהיה התוכנית מתורגמת בשלמותה לקוד מכונה.