

מטלת מנחה (ממ"ן) 14

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרויקט גמר

מספר השאלות: 1 משקל המטלה: 31 נקודות (חובה)

סמסטר: 2020' מועד אחרון להגשה: 29.3.2020

קיימות שתי חלופות להגשת מטלות:

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

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:

1. קבצי המקור של התוכנית שכתבתם (קבצים בעלי סיומת c או h).
2. קובץ הרצה (מקומפל ומקושר) עבור מערכת אוֹבֹנוֹטו.
3. קובץ makefile. יש להשתמש בקומפיילר gcc עם הדגלים: -Wall -ansi -pedantic. יש לנפות את כל ההודעות שמוציא הקומפיילר, כך שהתכנית תתקמפל ללא כל הערות או אזהרות. דוגמאות הרצה (קלט ופלט):
4. **א.** קבצי קלט בשפת אסמבלי, וקבצי הפלט שנוצרו מהפעלת האסמבלר על קבצי קלט אלה. יש להדגים שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האסמבלי.
5. **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 התכנית יעמוד בקריטריונים של בהירות, קריאות וכתובה נאה ומובנית.

נזכיר מספר היבטים חשובים של כתיבת קוד טוב:

1. הפשטה של מבני הנתונים: רצוי (ככל האפשר) להפריד בין הגישה למבני הנתונים לבין המימוש של מבני הנתונים. כך, למשל, בעת כתיבת פונקציות לטיפול בטבלה, אין זה מעניינם של המשתמשים בפונקציות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקושרת.
2. קריאות הקוד: יש להשתמש בשמות משמעותיים למשתנים ופונקציות. כמו כן, רצוי להגדיר קבועים רלוונטיים תוך שימוש בהנחית #define, ולהימנע מ"מספרי קסם", שמשמעותם נהירה לכם בלבד. 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 הדרוש ממנה, אינה לכשעצמה ערובה לציון גבוה. כדי לקבל ציון גבוה, על התכנית לעמוד בקריטריונים של כתיבה ותייעוד ברמה טובה, כמתואר לעיל, אשר משקלם המשותף מגיע עד לכ- 40% ממשקל הפרויקט.

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

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

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

רקע כללי ומטרת הפרויקט

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

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

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

היע"מ יודע לבצע קוד שנמצא בפורמט של **שפת מכונה**. זהו רצף של ביטים, המהווים קידוד בינארי של סדרת הוראות המכונה המרכיבות את התוכנית. קוד כזה אינו קריא למשתמש, ולכן לא נוח לקודד (או לקרוא) תכניות 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 תכנית הכתובה בשפת אסמבלי. זהו השלב הראשון במסלול אותו עוברת התכנית, עד לקבלת קוד המוכן לריצה על חומרת המחשב. השלבים הבאים הם קישור (linkage) וטעינה (loading), אך בהם לא נעסוק בממ"ן זה.

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

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

המחשב הדמיוני ושפת האסמבלי

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

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

"חומרה":

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

למעבד 8 אוגרים כלליים, בשמות: r0, r1, r2, r3, r4, r5, r6, r7. גודלו של כל אוגר הוא 15 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 14. 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 (בבסיס עשרוני), וכל תא הוא בגודל של 15 סיביות. לתא בזיכרון נקרא גם בשם **"מילה"**. הסיביות בכל מילה ממוספרות כמו באוגר.

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

מבנה הוראת מכונה:

כל הוראת מכונה מקודדת למספר מילות זיכרון רצופות, **החל ממילה אחת ועד למקסימום שלוש מילים**, הכל בהתאם לשיטות המיעון בהן נעשה שימוש (ראו בהמשך).

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

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

14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				שיטת מיעון אופרנד מקור				שיטת מיעון אופרנד יעד				השדה A,R,E		
opcode				ש	ש	ש	ש	ש	ש	ש	ש	A	R	E
				י	י	י	י	י	י	י	י			
				ט	ט	ט	ט	ט	ט	ט	ט			
				ה	ה	ה	ה	ה	ה	ה	ה			
				3	2	1	0	3	2	1	0			

סיביות 11-14: במילה הראשונה של ההוראה סיביות אלה מהוות את **קוד-הפעולה** (opcode).

במודל המכונה שלנו יש 16 קודי פעולה. כל קוד-פעולה (opcode) מיוצג בשפת אסמבלי באופן סימבולי על ידי **שם-פעולה**.

להלן קודי-הפעולה :

שם הפעולה	קוד הפעולה (בבסיס עשרוני)
mov	0
cmp	1
add	2
sub	3
lea	4
clr	5
not	6
inc	7
dec	8
jmp	9
bne	10
red	11
prn	12
jsr	13
rts	14
stop	15

שם הפעולה נכתב תמיד באותיות קטנות בלבד. פרטים על הפעולות השונות יובאו בהמשך.

סיביות 3-6: מקודדות את שיטת המיעון של אופרנד היעד (destination operand). לכל שיטת מיעון יש סיבית נפרדת, שערכה 1 אם אופרנד היעד נתון בשיטה זו, ואחרת ערך הסיבית 0. אם אין בהוראה אופרנד יעד, כל ארבע הסיביות מאופסות.

סיביות 7-10: מקודדות את שיטת המיעון של אופרנד המקור (source operand). לכל שיטת מיעון יש סיבית נפרדת, שערכה 1 אם אופרנד המקור נתון בשיטה זו, ואחרת ערך הסיבית 0. אם אין בהוראה אופרנד מקור, כל ארבע הסיביות מאופסות.

סיביות 0-2 (השדה 'A,R,E'): אפיון של תפקיד השדה 'A,R,E' בקוד המכונה יובא בהמשך. במילה הראשונה של כל הוראה, ערך הסיבית A תמיד 1, ושתי הסיביות האחרות מאופסות.

לתשומת לב: השדה 'A,R,E' מתווסף לכל אחת מהמילים בקידוד ההוראה (ראו פירוט של שיטות המיעון בהמשך).

שיטות מיעון:

בשפת האסמבלי שלנו קיימות ארבע שיטות מיעון, המסומנות במספרים 0,1,2,3. השימוש בשיטות מיעון מצריך קידוד של מילות-מידע נוספות בקוד המכונה של כל הוראת מכונה. כל אופרנד של ההוראה דורש מילה אחת נוספת.

כאשר בהוראה יש שני אופרנדים, קודם תופיע מילת-המידע הנוספת של האופרנד הראשון (אופרנד המקור), ולאחריה מילת-המידע הנוספת של האופרנד השני (אופרנד היעד). קיים גם מקרה מיוחד בו קידוד שני האופרנדים נעשה באמצעות מילת-מידע אחת משותפת לשני האופרנדים.

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

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

הסבר על התפקיד של השדה 'A,R,E' בקוד המכונה יבוא בהמשך.

ערך השדה 'A,R,E' הנדרש בכל שיטת מיעון מופיע בתיאור שיטות המיעון להלן.

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	אופן כתיבת האופרנד	דוגמה
0	מיעון מיידי	מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר שלם בשיטת המשלים ל-2, ברוחב של 12 סיביות, השוכן בסיביות 14-3 של המילה. הסיביות 2-0 של מילת המידע הן השדה A,R,E. במיעון מיידי, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני.	mov #-1,r2 בדוגמה זו האופרנד הראשון של הפקודה (אופרנד המקור) נתון בשיטת מיעון מיידי. ההוראה כותבת את הערך 1- אל אוגר r2
1	מיעון ישיר	מילת-מידע נוספת של ההוראה מכילה כתובת בזיכרון. המילה בכתובת זו בזיכרון היא האופרנד. הכתובת מיוצגת כמספר ללא סימן ברוחב של 12 סיביות, בסיביות 14-3 של מילת המידע. הסיביות 2-0 במילת המידע הן השדה A,R,E. במיעון ישיר, ערך הסיביות האלה תלוי בסוג הכתובת הרשומה בסיביות 14-3. אם זוהי כתובת שמייצגת שורה בקובץ המקור הנוכחי (כתובת פנימית), ערך הסיבית R הוא 1, ושתי הסיביות האחרות מאופסות. ואילו אם זוהי כתובת שמייצגת שורה בקובץ מקור אחר של התכנית (כתובת חיצונית), ערך הסיבית E הוא 1, ושתי הסיביות האחרות מאופסות.	האופרנד הוא תווית שכבר הוגדרה, או שתוגדר בהמשך הקובץ. ההגדרה נעשית על ידי כתיבת תווית בתחילת הנחית 'data' או 'string', או בתחילת הוראה של התוכנית, או באמצעות אופרנד של הנחית 'extern'. תווית היא ייצוג סימבולי של 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).

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	אופן כתיבת האופרנד	דוגמה
2	מיעון אוגר עקיף	<p>שיטת מיעון זו משמשת לגישה לזיכרון באמצעות מצביע שנמצא באוגר. תוכן האוגר הוא כתובת בזיכרון, והמילה בכתובת זו בזיכרון היא האופרנד. הכתובת מיוצגת באוגר כמספר ללא סימן ברוחב של 15 סיביות.</p> <p>אם האופרנד הוא אופרנד יעד, מילת-מידע נוספת של הפקודה תכיל בסיביות 3-5 את מספרו של האוגר שמשמש כמצביע. ואילו אם האוגר הוא אופרנד מקור, מספר האוגר יקודד בסיביות 6-8 של מילת-המידע הנוספת.</p> <p>הסיביות 0-2 של מילת המידע הן השדה A,R,E. במיעון אוגר עקיף, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.</p> <p>אם בפקודה יש שני אופרנדים, וכל אחד מהם בשיטת מיעון אוגר עקיף או בשיטת מיעון אוגר ישיר (ראו בהמשך), שני האוגרים יחלקו מילת-מידע אחת משותפת, כאשר סיביות 3-5 יכילו את מספר אוגר היעד, וסיביות 6-8 את מספר אוגר המקור. השדה A,R,E יהיה כמפורט לעיל.</p> <p>סיביות במילת-המידע שאינן בשימוש יכילו 0.</p>	<p>האופרנד מתחיל בתו * ולאחריו ובצמוד אליו מופיע שם של אוגר.</p>	<p>inc *r1</p> <p>בדוגמה זו, ההוראה inc מגדילה ב-1 את תוכן המילה בזיכרון עליה מצביע האוגר r1, כלומר המילה שכתובתה נמצאת באוגר r1.</p> <p>דוגמה נוספת:</p> <p>mov *r1,*r2</p> <p>בדוגמה זו, ההוראה mov מעתיקה את תוכן המילה בזיכרון עליה מצביע האוגר r1 אל המילה בזיכרון עליה מצביע האוגר r2.</p> <p>שני האופרנדים בדוגמה זו הם בשיטת מיעון אוגר עקיף, ולכן שני האוגרים יקודדו במילת-מידע נוספת אחת משותפת.</p>
3	מיעון אוגר ישיר	<p>האופרנד הוא אוגר. אם האוגר משמש כאופרנד יעד, מילת-מידע נוספת של הפקודה תכיל בסיביות 3-5 את מספרו של האוגר. ואילו אם האוגר משמש כאופרנד מקור, מספר האוגר יקודד בסיביות 6-8 של מילת-המידע.</p> <p>הסיביות 0-2 של מילת המידע הן השדה A,R,E. במיעון אוגר ישיר, ערך הסיבית A הוא 1, ושתי הסיביות האחרות מאופסות.</p> <p>אם בפקודה יש שני אופרנדים, וכל אחד מהם בשיטת מיעון אוגר ישיר או בשיטת מיעון אוגר עקיף (ראו לעיל), שני האוגרים יחלקו מילת-מידע אחת משותפת, כאשר סיביות 3-5 יכילו את מספר אוגר היעד, וסיביות 6-8 את מספר אוגר המקור. השדה A,R,E יהיה כמפורט לעיל.</p> <p>סיביות במילת-המידע שאינן בשימוש יכילו 0.</p>	האופרנד הוא שם של אוגר.	<p>clr r1</p> <p>בדוגמה זו, ההוראה clr מאפסת את תוכן האוגר r1.</p> <p>דוגמה נוספת:</p> <p>mov r1,*r2</p> <p>בדוגמה זו, ההוראה mov מעתיקה את תוכן האוגר r1 אל המילה בזיכרון אליה מצביע האוגר r2.</p> <p>אופרנד המקור r1 נתון בשיטת מיעון אוגר ישיר, ואופרנד היעד *r2 בשיטת מיעון אוגר עקיף, ולכן שני האוגרים יקודדו במילת-מידע נוספת אחת משותפת.</p>

מפרט הוראות המכונה :

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

קבוצת ההוראות הראשונה :

אלו הן הוראות הדורשות שני אופרנדים.

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

הוראה	opcode	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
mov	0	מבצעת העתקה של האופרנד הראשון, אופרנד המקור (source) אל האופרנד השני, אופרנד היעד (destination) (בהתאם לשיטת המיעון).	mov A, r1	העתק את תוכן המשתנה A (המילה שבכתובת A בזיכרון) אל אוגר r1.
cmp	1	מבצעת "השוואה" בין שני האופרנדים שלה. אופן ההשוואה : תוכן אופרנד היעד (השני) מופחת מתוכן 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 דגל האפס, Z, באוגר הסטטוס (PSW) יודלק, אחרת הדגל יאופס.
add	2	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	3	אופרנד היעד (השני) מקבל את תוצאת החיסור של אופרנד המקור (הראשון) מאופרנד היעד (השני).	sub #3, r1	אוגר r1 מקבל את תוצאת החיסור של הערך 3 מתוכנו הנוכחי של האוגר r1.
lea	4	lea הוא קיצור (ראשי תיבות) של load effective address. פעולה זו מציבה את המען בזיכרון המיוצג על ידי התווית שבאופרנד הראשון (המקור), אל אופרנד היעד (האופרנד השני).	lea HELLO, r1	המען שמייצגת התווית HELLO מוצב לאוגר r1.

קבוצת ההוראות השנייה :

אלו הן הוראות הדורשות אופרנד אחד בלבד. אופן הקידוד של האופרנד הוא כמו של אופרנד היעד בפקודה עם שני אופרנדים. במקרה זה, השדה של אופרנד המקור (סיביות 7-10) במילה הראשונה בקידוד ההוראה הינו חסר משמעות, ולפיכך יכול 00.

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

הוראה	opcode	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
clr	5	איפוס תוכן האופרנד	clr r2	$r2 \leftarrow 0$
not	6	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך : 1 ל-0).	not r2	$r2 \leftarrow \text{not } r2$
inc	7	הגדלת תוכן האופרנד באחד.	inc r2	$r2 \leftarrow r2 + 1$
dec	8	הקטנת תוכן האופרנד באחד.	dec C	$C \leftarrow C - 1$

הוראה	opcode	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
jmp	9	קפיצה (הסתעפות) בלתי מותנית אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כלומר, כתוצאה מביצוע ההוראה, מצביע התוכנית (PC) יקבל את ערך אופרנד היעד.	jmp LINE	$PC \leftarrow LINE$ מצביע התוכנית מקבל את המען המיוצג על ידי התווית LINE, ולפיכך הפקודה הבאה שתבצע תהיה במען זה.
bne	10	bne הוא קיצור (ראשי תיבות) של: branch if not equal (to zero). זוהי הוראת הסתעפות מותנית. מצביע התוכנית (PC) יקבל את ערך אופרנד היעד אם ערכו של הדגל Z באוגר הסטטוס (PSW) הינו 0. כזכור, הדגל Z נקבע בפקודת cmp.	bne LINE	אם ערך הדגל Z באוגר הסטטוס (PSW) הינו 0, אזי: $PC \leftarrow LINE$
red	11	קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מהקלט ייכנס לאוגר r1.
prn	12	הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn r1	התו אשר קוד ה-ascii שלו נמצא באוגר r1 יודפס לפלט הסטנדרטי.
jsr	13	קריאה לשגרה (סברוטניה), מצביע התוכנית (PC) הנוכחי נדחף לתוך המחסנית שבזיכרון המחשב, והאופרנד מוכנס ל-PC.	jsr FUNC	push(PC) $PC \leftarrow FUNC$

קבוצת ההוראות השלישית:

אלו הן הוראות ללא אופרנדים. קידוד ההוראה מורכב ממילה אחת בלבד. במקרה זה, השדות של אופרנד המקור ואופרנד היעד אינם רלוונטים (כי אין אופרנדים), ויכילו 0.

ההוראות השייכות לקבוצה זו הן: rts, stop.

הוראה	opcode	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
rts	14	חזרה משיגרה. הערך שנמצא בראש המחסנית של המחשב מוצא מן המחסנית, ומוכנס אל מצביע התוכנית (PC).	rts	$PC \leftarrow pop()$
stop	15	עצירת ריצת התוכנית.	stop	התכנית עוצרת

מבנה שפת האסמבלי:

שפת האסמבלי בנויה ממשפטים (statements). קובץ בשפת אסמבלי מורכב משורות המכילות משפטים של השפה, כאשר כל משפט מופיע בשורה נפרדת. כלומר התו המפריד בין משפט למשפט בקובץ הינו התו 'מ' (שורה חדשה).

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

יש ארבעה סוגי משפטים (שורות) בשפת האסמבלי, 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 תכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצאת זיכרון ואתחול משתנים של התכנית, אך הוא אינו מייצר קידוד של הוראות מכונה המיועדות לביצוע בעת ריצת התכנית.
משפט הוראה	זהו משפט המייצר קידוד של הוראות מכונה לביצוע בעת ריצת התכנית. המשפט מורכב משם של הוראה שעל המעבד לבצע, ותיאור האופרנדים 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 לשים לב: למילים בקוד המכונה הנוצרות ממשפט הנחיה לא מצורף השדה A,R,E, והערך המוגדר על ידי ההנחיה ממלא את כל 15 הסיביות של המילה.

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 בתכנית את ההוראה :

```
mov XYZ, r1
```

אזי בזמן ריצת התכנית יוכנס לאוגר r1 הערך 7.

ואילו ההוראה :

```
lea XYZ, r1
```

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

2. 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 :

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

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

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

4. ההנחיה 'extern'.

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

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 יכולים להופיע אופרנדים (אחד או שניים), בהתאם לסוג הפעולה.

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזה בתו ' ', (פסיק). בדומה להנחיה 'data', **לא חייבת להיות הצמדה של האופרנדים לפסיק או לשם הפעולה באופן כלשהו**. כל כמות של רווחים ו/או טאבים בין האופרנדים לפסיק, או בין שם הפעולה לאופרנד הראשון, היא חוקית.

למשפט הוראה עם שני אופרנדים המבנה הבא:

label(optional): opcode source-operand, target-operand

לדוגמה:

HELLO: add r7, B

למשפט הוראה עם אופרנד אחד המבנה הבא:

label(optional): opcode target-operand

לדוגמה:

HELLO: bne XYZ

למשפט הוראה ללא אופרנדים המבנה הבא:

label(optional): opcode

לדוגמה:

END: stop

אפיון השדות במשפטים של שפת האסמבלי

תווית:

תווית חוקית מתחילה באות אלפביתית (גדולה או קטנה), ולאחריה סדרה כלשהי של אותיות אלפביתיות (גדולות או קטנות) ו/או ספרות. האורך המקסימלי של תווית הוא 31 תווים.

הגדרה של תווית מסתיימת בתו ' ': (נקודתיים). תו זה אינו מהווה חלק מהתווית, אלא רק סימן המציין את סוף ההגדרה.

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

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hEllo:

x:

He78902:

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

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.

מספר:

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

מחרוזת:

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

תפקיד השדה A,R,E בקוד המכונה

בכל מילה בקוד המכונה של הוראה (לא של נתונים), האסמבלר מכניס מידע עבור תהליך הקישור והטעינה. זהו השדה A,R,E (שלוש הסיביות הימניות 2,1,0 בהתאמה). המידע ישמש לתיקונים בקוד בכל פעם שייטען לזיכרון לצורך הרצה. האסמבלר בונה מלכתחילה קוד שמיועד לטעינה החל מכתובת 100. התיקונים יאפשרו לטעון את הקוד בכל פעם למקום אחר, בלי צורך לחזור על תהליך האסמבלי.

בכל מילה של ההוראה, בדיוק אחת משלוש הסיביות של השדה A,R,E מכילה 1, ושתי הסיביות האחרות מאופסות. מפרט שיטות המיעון שהוצג קודם מציין איזו סיבית תכיל 1 בכל שיטת מעון.

סיבית 'A' (קיצור של Absolute) באה לציין שתוכן המילה אינו תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התכנית בעת ביצועה (למשל מילה המכילה אופרנד מיד).

סיבית 'R' (קיצור של Relocatable) באה לציין שתוכן המילה תלוי במקום בזיכרון בו ייטען בפועל קוד המכונה של התכנית בעת ביצועה (למשל מילה המכילה כתובת של תווית המוגדרת בקובץ המקור הנוכחי).

סיבית 'E' (קיצור של External) באה לציין שתוכן המילה תלוי בערכו של סמל חיצוני (External) (למשל מילה המכילה כתובת של תווית שמוגדרת בקובץ מקור אחר).

אסמבלר עם שני מעברים

כאשר מקבל האסמבלר תכנית בשפת אסמבלי, עליו לעבור על התכנית 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 התכנית הבאה בשפת אסמבלי :

```

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

```

קוד המכונה של התכנית (הוראות ונתונים) נבנה כך שיתאים לטעינה בזיכרון החל ממען 100 (עשרוני). התרגום של תכנית הדוגמה לקוד בינארי מוצג להלן :

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102	MAIN: add r3, LIST	First word of instruction Source register 3 Address of label LIST	001010000010100 000000011000100 000010000010010
0103 0104	LOOP: prn #48	Immediate value 48	110000000000100 000000110000100
0105 0106 0107	lea STR, r6	Address of label STR Target register 6	010000101000100 000001111101010 000000000110100
0108 0109	inc r6	Target register 6	011100001000100 000000000110100
0110 0111 0112	mov *r6,K	Source register 6 Address of label K	000001000010100 000000110000100 000010000101010
0113 0114	sub r1, r4	Source register 1 and target register 4	001110001000100 000000001100100
0115 0116 0117	cmp r3, #-6	Source register 3 Immediate value -6	000110000001100 000000011000100 11111111010100
0118 0119	bne END	Address of label END	101000000010100 00000111001010
0120 0121	dec K	Address of label K	011100000010100 000010000001010
0122 0123	jmp LOOP	Address of label LOOP	100100000010100 000001100111010
0124	END: stop		111100000000100
0125	STR: .string "abcd"	Ascii code 'a'	000000001100001
0126		Ascii code 'b'	000000001100010
0127		Ascii code 'c'	000000001100011
0128		Ascii code 'd'	000000001100100
0129		Ascii code '\0' (end of string)	000000000000000
0130 0131	LIST: .data 6, -9	Integer 6 Integer -9	000000000000110 11111111110111
0132	.data -100	Integer -100	111111110011100
0133	K: .data 31	Integer 31	000000000011111

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

כדי לעשות המרה לבינארי של אופרנדים שהם מענים סמליים (תוויות), יש צורך לבנות טבלה דומה. אולם בהבדל מהקודים של הפעולות, הידועים מראש, הרי המענים בזיכרון עבור הסמלים שבשימוש התכנית 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 124 (עשרוני), ושהסמל K אמור להיות משויך למען 133, אלא רק לאחר שנקראו כל שורות התכנית.

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

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

סמל	ערך (בבסיס עשרוני)
MAIN	100
LOOP	103
END	124
STR	125
LIST	130
K	133

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

לתשומת לב: תפקיד האסמבלר, על שני המעברים שלו, לתרגם קובץ מקור לקוד בשפת מכונה. בגמר פעולת האסמבלר, התכנית 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 קוד מתאים לכל שם פעולה. בזמן התרגום מחליף האסמבלר כל שם פעולה בקוד שלה, וכל אופרנד בקידוד מתאים. אך פעולת ההחלפה אינה כה פשוטה. ההוראות משתמשות בשיטות מיעון מגוונות לאופרנדים. אותה פעולה יכולה לקבל משמעויות שונות, בכל אחת משיטות המיעון, ולכן יתאימו לה קידודים שונים לפי שיטות המיעון. 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.

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

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

הוראה יכולה להתייחס גם לסמל שטרם הוגדר עד כה בתכנית, אלא יוגדר רק בהמשך התכנית. להלן 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 נתן לה מען, ולכן אינו יכול להחליף את הסמל A (האופרנד של ההוראה bne) במענו בזיכרון. נראה בהמשך כיצד נפתרת בעיה זו.

בכל מקרה, תמיד אפשר לבנות במעבר הראשון את הקוד הבינארי המלא של המילה הראשונה של כל הוראה, וכן את הקוד 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 התכנית, כך שכל הסמלים נכנסו כבר לטבלת הסמלים, יכול האסמבלר להשלים את קוד המכונה של כל האופרנדים.

לשם כך עובר האסמבלר שנית על כל קובץ המקור, ומעדכן את קוד המכונה של האופרנדים המשתמשים בסמלים, באמצעות ערכי הסמלים מטבלת הסמלים. זהו המעבר השני, ובסופו תהיה התוכנית מתורגמת בשלמותה לקוד מכונה.

הפרדת הוראות ונתונים

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

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

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

גילוי שגיאות בתכנית המקור

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

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

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

האסמבלר ידפיס את הודעות השגיאה אל הפלט הסטנדרטי stdout. בכל הודעת שגיאה יש לציין גם את מספר השורה בקובץ המקור בה זוהתה השגיאה.

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

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

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

תהליך העבודה של האסמבלר

נתאר כעת את אופן העבודה של האסמבלר. בהמשך, יוצג אלגוריתם שלדי למעבר ראשון ושני.

האסמבלר מתחזק שני מערכים, שייקראו להלן תמונת ההוראות (code) ותמונת הנתונים (data). מערכים אלו נותנים למעשה תמונה של זיכרון המכונה (כל איבר במערך הוא בגודל מילה של המכונה, כלומר 15 סיביות). במערך ההוראות בונה האסמבלר את הקידוד של הוראות המכונה שנקראו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמבלר את קידוד הנתונים שנקראו מקובץ המקור (שורות הנחיה מסוג 'data' ו-'string').

האסמבלר משתמש בשני מונים, שנקראים IC (מונה ההוראות - Instruction-Counter) ו-DC (מונה הנתונים - Data-Counter). מונים אלו מצביעים על המקום הבא הפנוי במערך ההוראות

ובמערך הנתונים, בהתאמה. בכל פעם כשמתחיל האסמבלר לעבור על קובץ מקור, שני המונים מאופסים.

בנוסף, מתחזק האסמבלר טבלה, אשר בה נאספות כל התוויות בהן נתקל האסמבלר במהלך המעבר על קובץ המקור. לטבלה זו קוראים טבלת-הסמלים (symbol-table). לכל סמל נשמרים בטבלה שם הסמל, ערכו המספרי, ומאפיינים שונים, כגון המיקום (code או data), וסוג הסמל (entry או external).

במעבר הראשון האסמבלר בונה את טבלת הסמלים ואת השלד של תמונת הזיכרון (הוראות ונתונים)

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

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

2. שורת הוראה:

האסמבלר מנתח את השורה ומפענח מהי ההוראה, ומתן שיטות המיעון של האופרנדים. מספר האופרנדים נקבע בהתאם להוראה שנמצאה. שיטות המיעון נקבעות בהתאם לתחביר של כל אופרנד, כפי שהוסבר לעיל במפרט שיטות המיעון. למשל, התו # מציין מיעון מידי, תווית מציינת מיעון ישיר, שם של אוגר מציין מיעון אוגר ישיר, וכד'.

אם האסמבלר מוצא בשורת ההוראה גם הגדרה של תווית, אזי התווית מוכנסת אל טבלת הסמלים. ערך התווית הוא IC+100, והמאפיין הוא code.

הערה: הערך IC+100 נקבע כדי שקוד המכונה של התכנית יתאים לטעינה לזיכרון (לצורך ריצה) החל מכתובת 100.

כעת האסמבלר קובע לכל אופרנד את ערכו באופן הבא:

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

האסמבלר מכניס למערך ההוראות, בכניסה עליה מצביע מונה ההוראות IC, את קוד המילה הראשונה של ההוראה (בפורמט קידוד כפי שתואר קודם). מילה זו מכילה את קוד הפעולה, ואת מספרי שיטות המיעון. ה- IC מקודם ב-1.

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

אם זוהי הוראה בעלת אופרנדים (אחד או שניים), האסמבלר "משריין" מקום במערך ההוראות עבור מילות-המידע הנוספות הנדרשות בהוראה זו, ומקדם את IC בהתאם. כאשר אחד או שני האופרנדים הם בשיטת מיעון אוגר-ישיר, אוגר-עקיף, או מידי, האסמבלר מקודד גם את המילים הנוספות הרלוונטיות במערך ההוראות. מילות-המידע של שיטת מיעון ישיר תישאר ללא קידוד.

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, ומקדם את מצביע הנתונים DC ב-1 עבור כל מספר שהוכנס.

אם בשורה 'data' מוגדרת גם תווית, אזי התווית מוכנסת לטבלת הסמלים. ערך התווית הוא ערך מונה הנתונים DC שלפני הכנסת המספרים למערך. המאפיין של התווית הוא data.

II. 'string'.

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

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

III. 'entry'.

זוהי הנחיה לאסמבלר לאפיין את התווית הנתונה כאופרנד כ- entry בטבלת הסמלים. בעת הפקת קבצי הפלט (ראו בהמשך), התווית תירשם בקובץ ה-entries. לתשומת לב: זה לא נחשב כשגיאה אם בקובץ המקור מופיעה יותר מהנחיית entry. אחת עם אותה תווית כאופרנד. המופעים הנוספים אינם מוסיפים דבר, אך גם אינם מפריעים.

IV. 'extern'.

זוהי הצהרה על סמל (תווית) המוגדר בקובץ מקור אחר, והקובץ הנוכחי עושה בו שימוש. האסמבלר מכניס את הסמל המופיע כאופרנד לטבלת הסמלים, עם הערך 0 (הערך האמיתי לא ידוע, וייקבע רק בשלב הקישור), ועם המאפיין external. לא ידוע באיזה קובץ נמצאת הגדרת הסמל, ואין זה רלוונטי עבור האסמבלר. לתשומת לב: זה לא נחשב כשגיאה אם בקובץ המקור מופיעה יותר מהנחיית extern. אחת עם אותה תווית כאופרנד. המופעים הנוספים אינם מוסיפים דבר, אך גם אינם מפריעים.

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

בסוף המעבר הראשון, האסמבלר מעדכן בטבלת הסמלים כל סמל המאופיין כ- data, על ידי הוספת $IC + (100)$ (עשרוני) לערכו של הסמל. הסיבה לכך היא שבתמונה הכוללת של קוד המכונה, תמונת הנתונים מופרדת מתמונת ההוראות, וכל הנתונים נדרשים להופיע בקוד המכונה אחר כל ההוראות. סמל מסוג data הוא תווית בתמונת הנתונים, והעדכון מוסיף לערך הסמל (כלומר לכתובתו בזיכרון) את האורך הכולל של תמונת ההוראות, בתוספת כתובת התחלת הטעינה של הקוד, שהיא 100.

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

במעבר השני, האסמבלר משלים באמצעות טבלת הסמלים את כל קידוד המילים במערך ההוראות שטרם קודדו במעבר הראשון. במודל המכונה שלנו אלו הן מילות-מידע נוספות של פקודות, אשר מקודדות אופרנד בשיטת מיעון ישיר. האופרנד הוא סמל שמוגדר כפנימי או כחיצוני, ולכן בשדה ה-A,R,E הסיבית R או הסיבית E, בהתאמה, תהיה 1 (ראו גם מפרט שיטות המיעון לעיל).

אלגוריתם שלדי של האסמבלר

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

לתשומת לב: אין חובה להשתמש דווקא באלגוריתם זה.

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

כמו כן, נסמן ב- L את המספר הכולל של מילים שתופס קוד המכונה של הוראה נתונה.

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

בכל מעבר מתחילים לקרוא את קובץ המקור מהתחלה.

מעבר ראשון

1. אתחל $DC \leftarrow 0, IC \leftarrow 0$.
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-16.
3. האם השדה הראשון בשורה הוא סמל? אם לא, עבור ל-5.
4. הדלק דגל "יש הגדרת סמל".
5. האם זוהי הנחיה לאחסון נתונים, כלומר, האם הנחית `data` או `string`? אם לא, עבור ל-8.
6. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם המאפיין `data`. ערך הסמל יהיה `DC` (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
7. זהה את סוג הנתונים, קודד אותם בתמונת הנתונים, ועדכן את מונה הנתונים `DC` בהתאם לאורך הנתונים. חזור ל-2.
8. האם זו הנחית `extern` או הנחית `entry`? אם לא, עבור ל-11.
9. אם זוהי הנחית `entry`. חזור ל-2 (ההנחיה תטופל במעבר השני).
10. אם זו הנחית `extern`, הכנס כל סמל (אחד או יותר) המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים, ללא ערך, עם המאפיין `external`. חזור ל-2.
11. זוהי שורת הוראה. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם המאפיין `code`. ערכו של הסמל יהיה $IC+100$ (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
12. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא, אז הודע על שגיאה בשם ההוראה.
13. נתח את מבנה האופרנדים של ההוראה וחשב את L (מספר המילים שתופסת ההוראה בקוד הבינארי).
14. בנה כעת את הקוד הבינארי של המילה הראשונה של ההוראה, ושל כל מילת-מידע נוספת המקודדת אופרנד שאינו מכיל סמל (מיעון מיידי, מיעון אוגר ישיר, מיעון אוגר עקיף).
15. עדכן $IC \leftarrow IC + L$, וחזור ל-2.
16. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר הראשון, עצור כאן.
17. עדכן בטבלת הסמלים את ערכו של כל סמל המאופיין ב- `data`, ע"י הוספת הערך $IC+100$ (ראה הסבר בהמשך).
18. התחל מעבר שני.

מעבר שני

1. אתחל $IC \leftarrow 0$.
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-9.
3. אם השדה הראשון בשורה הוא סמל (תווית), דלג עליו.
4. האם זוהי הנחית `data` או `string`. אם כן, חזור ל-2.
5. האם זוהי הנחית `entry`? אם לא, עבור ל-7.
6. הוסף בטבלת הסמלים את המאפיין `entry` לכל סמל (אחד או יותר) המופיע כאופרנד של ההנחיה (אם הסמל לא נמצא בטבלת הסמלים, יש להודיע על שגיאה). חזור ל-2.
7. השלם את הקידוד הבינארי של מילות-המידע של האופרנדים, בהתאם לשיטות המיעון שבשימוש. אם אופרנד בקוד המקור מכיל סמל, מצא את ערכו בטבלת הסמלים (אם הסמל לא נמצא בטבלה, יש להודיע על שגיאה).
8. עדכן $IC \leftarrow IC + L$, וחזור ל-2.
9. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר השני, עצור כאן.
10. בנה את קבצי הפלט (פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו קודם, ונציג את הקוד הבינארי שמתקבל במעבר ראשון ובמעבר שני.

```

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

```

נבצע מעבר ראשון על הקוד לעיל, ונבנה את טבלת הסמלים. כמו כן, נשלים במעבר זה את הקידוד של כל תמונת הנתונים, ושל המילה הראשונה של כל הוראה. כמו כן, נקודד מילות-מידע נוספות של כל הוראה, ככל שקידוד זה אינו תלוי בערך של סמל. את מילות-המידע שעדיין לא ניתן לקודד במעבר הראשון נסמן ב "???" בדוגמה להלן.

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102	MAIN: add r3, LIST	First word of instruction Source register 3 Address of label LIST	001010000010100 000000011000100 ???
0103 0104	LOOP: prn #48	Immediate value 48	110000000000100 000000110000100
0105 0106 0107	lea STR, r6	Address of label STR Target register 6	010000101000100 ?? 000000000110100
0108 0109	inc r6	Target register 6	011100001000100 000000000110100
0110 0111 0112	mov *r6, K	Source register 6 Address of label K	000001000010100 000000110000100 ??
0113 0114	sub r1, r4	Source register 1 and target register 4	001110001000100 000000001100100
0115 0116 0117	cmp r3, #-6	Source register 3 Immediate value -6	000110000001100 000000011000100 11111111010100
0118 0119	bne END	Address of label END	101000000010100 ??
0120 0121	dec K	Address of label K	011100000010100 ??
0122 0123	jmp LOOP	Address of label LOOP	100100000010100 ??
0124	END: stop		111100000000100
0125	STR: .string "abcd"	Ascii code 'a'	000000001100001
0126		Ascii code 'b'	000000001100010
0127		Ascii code 'c'	000000001100011
0128		Ascii code 'd'	000000001100100
0129		Ascii code '\0' (end of string)	000000000000000
0130 0131	LIST: .data 6, -9	Integer 6 Integer -9	000000000000110 111111111101111
0132	.data -100	Integer -100	111111110011100
0133	K: .data 31	Integer 31	000000000011111

טבלת הסמלים היא :

סמל	ערך (בבסיס עשרוני)
MAIN	100
LOOP	103
END	124
STR	125
LIST	130
K	133

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

הערה : כאמור, האסמבלר בונה קוד מכונה כך שיתאים לטעינה לזיכרון החל מכתובת 100 (עשרוני). אם הטעינה בפועל (לצורך הרצת התכנית) תהיה לכתובת אחרת, יידרשו תיקונים בקוד הבינארי בשלב הטעינה, שיוכנסו בעזרת מידע נוסף שהאסמבלר מכין בקבצי הפלט (ראו בהמשך).

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102	MAIN: add r3, LIST	First word of instruction Source register 3 Address of label LIST	001010000010100 000000011000100 000010000010010
0103 0104	LOOP: prn #48	Immediate value 48	110000000000100 000000110000100
0105 0106 0107	lea STR, r6	Address of label STR Target register 6	010000101000100 000001111101010 000000000110100
0108 0109	inc r6	Target register 6	011100001000100 000000000110100
0110 0111 0112	mov *r6,K	Source register 6 Address of label K	000001000010100 000000110000100 000010000101010
0113 0114	sub r1, r4	Source register 1 and target register 4	001110001000100 000000001100100
0115 0116 0117	cmp r3, #-6	Source register 3 Immediate value -6	000110000001100 000000011000100 111111111010100
0118 0119	bne END	Address of label END	101000000010100 000001111001010
0120 0121	dec K	Address of label K	011100000010100 000010000001010
0122 0123	jmp LOOP	Address of label LOOP	100100000010100 000001100111010
0124	END: stop		111100000000100
0125	STR: .string "abcd"	Ascii code 'a'	000000001100001
0126		Ascii code 'b'	000000001100010
0127		Ascii code 'c'	000000001100011
0128		Ascii code 'd'	000000001100100
0125		Ascii code '\0' (end of string)	000000000000000
0130 0131	LIST: .data 6, -9	Integer 6 Integer -9	000000000000110 111111111101111
0132	.data -100	Integer -100	111111110011100
0133	K: .data 31	Integer 31	000000000011111

בסוף המעבר השני, אם לא נתגלו שגיאות, האסמבלר בונה את קבצי הפלט (ראו בהמשך), שמכילים את הקוד הבינארי ומידע נוסף עבור שלבי הקישור והטעינה. כאמור, שלבי הקישור והטעינה אינם למימוש בפרויקט זה, ולא נדון בהם כאן.

קבצי קלט ופלט של האסמבלר

בהפעלה של האסמבלר, יש להעביר אליו באמצעות ארגומנטים של שורת הפקודה (command line arguments) רשימה של שמות קבצי מקור (אחד או יותר). אלו הם קבצי טקסט, ובהם תכניות בתחביר של שפת האסמבלי שהוגדרה בממ"ן זה.

האסמבלר פועל על כל קובץ מקור בנפרד, ויוצר עבורו קבצי פלט כדלקמן:

- קובץ object, המכיל את קוד המכונה.
- קובץ externals, ובו פרטים על כל המקומות (הכתובות) בקוד המכונה בהם מקודד ערך של סמל חיצוני (סמל שהוגדר באמצעות ההנחיה extern, ומאופיין בטבלת הסמלים כ- external).
- קובץ entries, ובו פרטים על כל סמל שמוצהר כנקודת כניסה (סמל שהופיע כאופרנד של הנחיית entry, ומאופיין בטבלת הסמלים כ- entry).

אם אין בקובץ המקור אף הנחיית extern, האסמבלר לא יוצר את קובץ הפלט מסוג externals. אם אין בקובץ המקור אף הנחיית entry, האסמבלר לא יוצר את קובץ הפלט מסוג entries.

שמות קבצי המקור חייבים להיות עם הסיומת ".as". למשל, השמות x.as, y.as, ו-hello.as הם שמות חוקיים. העברת שמות הקבצים הללו כארגומנטים לאסמבלר נעשית ללא ציון הסיומת.

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

```
assembler x y hello
```

תריץ את האסמבלר על הקבצים: x.as, y.as, hello.as.

שמות קבצי הפלט מבוססים על שם קובץ הקלט, כפי שהופיע בשורת הפקודה, בתוספת סיומת מתאימה: הסיומת ".ob" עבור קובץ ה-object, הסיומת ".ent" עבור קובץ ה-entries, והסיומת ".ext" עבור קובץ ה-externals.

לדוגמה, בהפעלת האסמבלר באמצעות שורת הפקודה: assembler x, ויוצר קובץ פלט x.ob, וכן קבצי פלט x.ent ו-x.ext. ככל שיש הנחיות entry או extern. בקובץ המקור.

נציג כעת את הפורמטים של קבצי הפלט. דוגמאות יובאו בהמשך.

פורמט קובץ ה-object

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

כזכור, האסמבלר מקודד את ההוראות כך שתמונת ההוראות תתאים לטעינה החל מכתובת 100 (עשרוני) בזיכרון. נשים לב שרק בסוף המעבר הראשון יודעים מהו הגודל הכולל של תמונת ההוראות. מכיוון שתמונת הנתונים נמצאת אחרי תמונת ההוראות, גודל תמונת ההוראות משפיע על הכתובות בתמונת הנתונים. זו הסיבה שבגללה היה צורך לעדכן בטבלת הסמלים, בסוף המעבר הראשון, את ערכי הסמלים המאופיינים כ-data (כזכור, הוספנו לכל סמל כזה את הערך IC+100). במעבר השני, השלמת הקידוד משתמשת בערכים המעודכנים של הסמלים, המותאמים למבנה המלא והסופי של תמונת הזיכרון.

כעת האסמבלר יכול לכתוב את תמונת הזיכרון בשלמותה לתוך קובץ פלט (קובץ ה-object).

השורה הראשונה בקובץ ה-object היא "כותרת", המכילה שני מספרים (בבסיס עשרוני): הראשון הוא האורך הכולל של תמונת ההוראות (במילות זיכרון), והשני הוא האורך הכולל של תמונת הנתונים (במילות זיכרון). בין שני המספרים יש רווח אחד.

השורות הבאות בקובץ מכילות את תמונת הזיכרון. בכל שורה שני שדות: כתובת של מילה בזיכרון, ותוכן המילה. הכתובת תירשם בבסיס עשרוני בארבע ספרות (כולל אפסים מובילים). תוכן המילה יירשם בבסיס אוקטאלי ב-5 ספרות. בין שני השדות יש רווח אחד.

פורמט קובץ ה-entries

קובץ ה-entries בנוי משורות טקסט, שורה אחת לכל סמל שמאופיין כ-entry. בשורה מופיע שם הסמל, ולאחריו ערכו כפי שנקבע בטבלת הסמלים (בבסיס עשרוני). אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.

פורמט קובץ ה-externals

קובץ ה-externals בנוי אף הוא משורות טקסט, שורה לכל כתובת בקוד המכונה בה מקודדת מילת-מידע המתייחסת לסמל שמאופיין כ-external. בשורה מופיע שם הסמל, ולאחריו הכתובת של מילת-המידע (בבסיס עשרוני). אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.

כמובן שיתכן ויש מספר כתובות בקוד המכונה בהן הקידוד מתייחס לאותו סמל חיצוני. לכל כתובת כזו תהיה שורה נפרדת בקובץ ה-externals.

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

; file ps.as

.entry LIST

.extern fn1

```
MAIN:      add    r3, LIST
           jsr     fn1
LOOP:      prn     #48
           lea     STR, r6
           inc     r6
           mov     *r6, L3
           sub     r1, r4
           cmp     r3, #-6
           bne     END
           add     r7, *r6
           clr     K
           sub     L3, L3
```

.entry MAIN

```
           jmp     LOOP
END:        stop
STR:        .string "abcd"
LIST:       .data  6, -9
           .data  -100
K:          .data  31
.extern L3
```

להלן הקידוד הבינארי המלא (תמונת הזיכרון) של קובץ המקור, כפי שנבנה במעבר הראשון והשני.

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102	MAIN: add r3, LIST	First word of instruction Source register 3 Address of label LIST	001010000010100 000000011000100 000010001001010
0103 0104	jsr fn1	Address of label fn1 (external)	110100000010100 000000000000001
0105 0106	LOOP: prn #48	Immediate value 48	110000000001100 000000110000100
0107 0108 0109	lea STR, r6	Address of label STR Target register 6	010000101000100 000010000100010 000000000110100
0110 0111	inc r6	Target register 6	011100001000100 000000000110100
0112 0113 0114	mov *r6, L3	Source register 6 Address of label L3 (external)	000001000010100 000000110000100 000000000000001
0115 0116	sub r1, r4	Source register 1 and target register 4	001110001000100 000000001100100
0117 0118 0119	cmp r3, #-6	Source register 3 Immediate value -6	000110000001100 000000011000100 111111110101010
0120 0121	bne END	Address of label END	101000000010100 000010000011010
0122 0123	add r7, *r6	Source register r0 and target register 6	001010000100100 000000111110100
0124 0125	clr K	Address of label K	010100000010100 000010001100010
0126 0127 0128	sub L3, L3	Address of label L3 (external) Address of label L3 (external)	001100100010100 000000000000001 000000000000001
0129 0130	jmp LOOP	Address of label LOOP	100100000010100 000001101001010
0131	END: stop		111100000000100
0132	STR: .string "abcd"	Ascii code 'a'	000000001100001
0133		Ascii code 'b'	000000001100010
0134		Ascii code 'c'	000000001100011
0135		Ascii code 'd'	000000001100100
0136		Ascii code '\0' (end of string)	000000000000000
0137 0138	LIST: .data 6, -9	Integer 6 Integer -9	000000000000110 111111111101111
0139	.data -100	Integer -100	111111110011100
0140	K: .data 31	Integer 31	000000000011111

להלן תוכן קבצי הפלט של הדוגמה.

הקובץ ps.ob:

```
32 9
0100 12024
0101 00304
0102 02112
0103 64024
0104 00001
0105 60014
0106 00604
0107 20504
0108 02042
0109 00064
0110 34104
0111 00064
0112 01024
0113 00604
0114 00001
0115 16104
0116 00144
0117 06014
0118 00304
0119 77724
0120 50024
0121 02032
0122 12044
0123 00764
0124 24024
0125 02142
0126 14424
0127 00001
0128 00001
0129 44024
0130 01512
0131 74004
0132 00141
0133 00142
0134 00143
0135 00144
0136 00000
0137 00006
0138 77767
0139 77634
0140 00037
```

הקובץ ps.ent:

```
MAIN 100
LIST 137
```

הקובץ ps.ext:

```
fn1 0104
L3 0114
L3 0127
L3 0128
```

סיכום והנחיות כלליות

- גודל תוכנית המקור הניתנת כקלט לאסמבלר אינו ידוע מראש, ולכן גם גודלו של קוד המכונה אינו צפוי מראש. אולם בכדי להקל במימוש האסמבלר, מותר להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים לאחסון תמונת קוד המכונה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים), יש לממשל באופן יעיל וחסכוני (למשל באמצעות רשימה מקושרת והקצאת זיכרון דינאמי).
- השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הקלט, למעט הסיימות. למשל, אם קובץ הקלט הוא prog.as אזי קבצי הפלט שיווצרו הם : prog.ob, prog.ext, prog.ent
- מתכונת הפעלת האסמבלר צריכה להיות כפי הנדרש בממ"ן, ללא שינויים כלשהם. כלומר, ממשיך המשתמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו לתכנית האסמבלר כארגומנטים בשורת הפקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גרפיים למיניהם, וכד'.
- יש להקפיד לחלק את מימוש האסמבלר למספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסוגים שונים במודול יחיד. מומלץ לחלק למודולים כגון : מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחבירי של שורה), טבלת הסמלים, מפת הזיכרון, טבלאות קבועות (קודי הפעולה, שיטות המיעון החוקיות לכל פעולה, וכד').
- יש להקפיד ולתעד את המימוש באופן מלא וברור, באמצעות הערות מפורטות בקוד.
- יש לאפשר תווים לבנים עודפים בקובץ הקלט בשפת אסמבלי. למשל, אם בשורת הוראה יש שני אופרנדים המופרדים בפסיק, אזי לפני ואחרי הפסיק מותר שיהיו רווחים וטאבים בכל כמות. בדומה, גם לפני ואחרי שם הפעולה. מותרות גם שורות ריקות. האסמבלר יתעלם מתווים לבנים מיותרים (כלומר ידלג עליהם).
- הקלט (קוד האסמבלי) עלול להכיל שגיאות תחביריות. על האסמבלר לגלות ולדווח על כל השורות השגויות בקלט. אין לעצור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להדפיס למסך הודעות מפורטות ככל הניתן, כדי שאפשר יהיה להבין מה והיכן כל שגיאה. כמובן שאם קובץ קלט מכיל שגיאות, אין טעם להפיק עבורו את קבצי הפלט (ob, ext, ent).

תם ונשלם פרק ההסברים והגדרת הפרויקט.

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

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

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

ב ה צ ל ח ה !