

מטלת מנחה (ממ"ן) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה : פרויקט גמר

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

סמסטר : 2022 מועד אחרון להגשה : 14.08.2022

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.

מוותר 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 ברגיסטרים (registers - אוגרים) הקיימים בתוך היע"מ, ובזיכרון המחשב.

דוגמאות: העברת מספר מתא בזיכרון לרגיסטר ביע"מ או בחזרה, הוספת 1 למספר הנמצא ברגיסטר, בדיקה האם מספר המאוחסן ברגיסטר שווה לאפס, חיבור וחסור בין שני רגיסטרים, 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 CPU (יע"מ - יחידת עיבוד מרכזית), אוגרים (רגיסטרים) וזיכרון RAM. חלק מהזיכרון משמש גם כמחסנית (stack).

למעבד 8 רגיסטרים כלליים, בשמות: r0, r1, r2, r3, r4, r5, r6, r7. גודלו של כל רגיסטר הוא 10 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס' 0, והסיבית המשמעותית ביותר כמס' 9. שמות הרגיסטרים נכתבים תמיד עם אות 'r' קטנה.

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

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

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

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

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

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

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

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

סיביות 0-1 (A,R,E)

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

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

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

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

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

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

השימוש בשיטות מיעון מצריך קידוד של מילות-מידע נוספות בקוד המכונה של ההוראה. אם בפקודה יש אופרנד אחד, תהיה מילת מידע אחת נוספת. אם בפקודה יש שני אופרנדים, ייתכנו שתי מילות-מידע נוספות, או מילת-מידע אחת המשותפת לשני האופרנדים, תלוי בשיטות המיעון בהן נעשה שימוש (ראו מפרט בהמשך). כאשר בקידוד הפקודה יש שתי מילות-מידע נוספות, אזי מילת-המידע הראשונה מתייחסת לאופרנד המקור, והשנייה מתייחסת לאופרנד היעד.

בכל מילת-מידע נוספת, סיביות 0-1 הן השדה A,R,E.

להלן תיאור של שיטות המיעון במכונה שלנו:

| מספר | שיטת המיעון | תוכן מילות המידע הנוספות | תחביר האופרנד באסמבלי | דוגמה |
|------|-------------------|--|---|--|
| 0 | מיעון מידי | מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר המיוצג ב- 8 סיביות, אליהן מתווספות זוג סיביות של השדה A,R,E | האופרנד מתחיל בתו # ולאחריו ובצמוד אליו מופיע מספר שלם בבסיס עשרוני | <p>mov #-1,r2</p> <p>בדוגמה זו האופרנד הראשון של הפקודה נתון בשיטת מיעון מידי. ההוראה כותבת את הערך 1- אל רגיסטר r2</p> |
| 1 | מיעון ישיר | מילת-מידע נוספת של ההוראה מכילה כתובת של מילה בזיכרון. מילה זו בזיכרון היא האופרנד. המען מיוצג ב- 8 סיביות אליהן מתווספות זוג סיביות של השדה A,R,E | האופרנד הוא תווית שכבר הוצהרה או תוצהר בהמשך הקובץ. ההצהרה נעשית על ידי כתיבת תווית בקובץ המקור (בתחילת הנחית 'data' או 'string' או 'struct', או בתחילת הוראה של התוכנית), או על ידי אופרנד של הנחית 'extern' | <p>נתונה למשל ההגדרה : x: .data 23</p> <p>אפשר לכתוב הוראה : dec x</p> <p>בדוגמה זו, ההוראה מקטינה ב-1 את תוכן המילה שבכתובת x בזיכרון ("משנתנה" x).</p> |
| 2 | מיעון גישה לרשומה | בשיטת מיעון זו יש 2 מילות מידע נוספות בקוד ההוראה. המילה הראשונה הנוספת היא כתובת התחלת הרשומה אליה פונים. המילה הנוספת השניה היא מספר השדה המבוקש. לכל אחת משתי המילים הנוספות של שיטת המיעון מתווספות זוג סיביות של שדה A,R,E. | האופרנד מורכב משם של תוית המציינת רשומה (struct), ולאחריה התו נקודה (.). ולאחריו 1 אם פונים לשדה הראשון או 2 אם פונים לשדה השני. ראו בהמשך הסבר על הגדרת רשומה. | <p>נתונה למשל ההגדרה : s: .struct 9,"abcd"</p> <p>אפשר לכתוב הוראה : add #4, s.1</p> <p>בדוגמה זו, ההוראה מוסיפה את הערך 4 לשדה הראשון של הרשומה s (המכיל את המספר 9).</p> |
| 3 | מיעון רגיסטר ישיר | האופרנד הוא רגיסטר. אם הרגיסטר משמש כאופרנד יעד, מילה נוספת של הפקודה תכיל בארבע הסיביות 2-5 את מספרו של הרגיסטר. אם הרגיסטר משמש כאופרנד מקור, הוא יקודד במילה נוספת שתכיל בששת הסיביות 6-9 את מספרו של הרגיסטר. אם בפקודה יש שני אופרנדים ושניהם רגיסטרים, הם יחלקו מילה נוספת אחת משותפת. כאשר הסיביות 2-5 הן עבור רגיסטר היעד, והסיביות 6-9 הן עבור רגיסטר המקור. לייצוג זה מתווספות זוג סיביות של שדה A,R,E. סיביות שאינן בשימוש יכילו 0. | האופרנד הוא שם של רגיסטר. | <p>mov r1,r2</p> <p>בדוגמה זו, ההוראה מעתיקה את תוכן רגיסטר r1 אל רגיסטר r2.</p> <p>בדוגמה זו שני האופרנדים הם אוגרים, אשר יקודדו למילת-מידע נוספת אחת משותפת.</p> |

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

מפרט הוראות המכונה :

בתיאור 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 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 | הפעולה המתבצעת | דוגמה | הסבר הדוגמה |
|-------|---|---------------|--|
| mov | מבצעת העתקה של תוכן 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 בזכרון) אל רגיסטר r1. |
| cmp | מבצעת "השוואה" בין שני האופרנדים. אופן ההשוואה: תוכן אופרנד היעד (השני) מופחת מתוכן אופרנד המקור (הראשון), ללא שמירת תוצאת החיסור. פעולת החיסור מעדכנת דגל בשם Z ("דגל האפס") ברגיסטר הסטטוס (PSW). | cmp A, r1 | אם תוכן המשתנה A זהה לתוכנו של רגיסטר r1 אזי דגל האפס, Z, ברגיסטר הסטטוס (PSW) יודלק, אחרת הדגל יאופס. |
| add | אופרנד היעד (השני) מקבל את תוצאת החיבור של אופרנד המקור (הראשון) והיעד (השני). | add A, r0 | רגיסטר r0 מקבל את תוצאת החיבור של תוכן המשתנה A ותוכנו הנוכחי של רגיסטר r0. |
| sub | אופרנד היעד (השני) מקבל את תוצאת החיסור של אופרנד המקור (הראשון) מאופרנד היעד (השני). | sub #3, r1 | רגיסטר r1 מקבל את תוצאת החיסור של הערך 3 מתוכנו הנוכחי של הרגיסטר r1. |
| lea | lea הוא קיצור (ראשי תיבות) של load effective address. פעולה זו מציבה את מען הזיכרון המיוצג על ידי התווית שבאופרנד הראשון (המקור), אל אופרנד היעד (האופרנד השני). | lea HELLO, r1 | המען שמייצגת התווית HELLO מוצב לרגיסטר r1. |

קבוצת ההוראות השנייה :

אלו הן הוראות המקבלות אופרנד אחד בלבד. אופן הקידוד של האופרנד הוא כמו של אופרנד היעד בהוראה עם שני אופרנדים. במקרה זה, השדה של אופרנד המקור (סיביות 4-5) במילה הראשונה בקידוד ההוראה אינו בשימוש, ולפיכך יכול 0.

ההוראות השייכות לקבוצה זו הן :

not, clr, inc, dec, jmp, bne, get, prn, jsr

| הוראה | הפעולה המתבצעת | דוגמה | הסבר דוגמה |
|-------|--|-----------|--|
| not | היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך: 1 ל-0). | not r2 | כל ביט ברגיסטר r2 מתהפך |
| clr | איפוס תוכן האופרנד. | clr r2 | הרגיסטר r2 מקבל את הערך 0 |
| inc | הגדלת תוכן האופרנד באחד. | inc r2 | תוכן הרגיסטר r2 מוגדל ב-1 |
| dec | הקטנת תוכן האופרנד באחד. | dec Count | תוכן משתנה Count מוקטן ב-1 |
| jmp | קפיצה (הסתעפות) בלתי מותנית אל ההוראה שנמצאת במען המיוצג על ידי האופרנד. כתוצאה מביצוע ההוראה, מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה. | jmp Line | הכתובת של תווית Line נשמרת לתוך מצביע התכנית, לכן ההוראה הבאה שתבצע תהיה במען Line. |
| bne | bne הוא קיצור (ראשי תיבות) של branch if not equal (to zero). זוהי הוראת הסתעפות מותנית. מצביע התוכנית (PC) מקבל את ערך אופרנד היעד אם ערכו של הדגל Z ברגיסטר הסטטוס (PSW) הינו 0. כזכור, ערך הדגל Z נקבע בפקודת cmp. | bne Line | אם ערך הדגל Z ברגיסטר הסטטוס (PSW) הוא 0 אז: מצביע התכנית יקבל את כתובת התווית Line ולכן ההוראה הבאה שתבצע תהיה במען Line. |
| get | קריאה של תו מהקלט הסטנדרטי (stdin) אל האופרנד. | get r1 | קוד ה-ascii של התו הנקרא מהקלט ייכנס לרגיסטר r1. |
| prn | הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout). | prn r1 | התו אשר קוד ה-ascii שלו נמצא ברגיסטר r1 יודפס לקלט הסטנדרטי. |
| jsr | קריאה לשגרה (סברוטניה). כתובת ההוראה שאחרי הוראת jsr הנוכחית (PC+2) נדחפת לתוך המחסנית שבזיכרון המחשב, ומצביע התוכנית (PC) מקבל את כתובת השגרה. <u>הערה</u> : חזרה מהשגרה מתבצעת באמצעות הוראת rts, תוך שימוש בכתובת שבמחסנית. | jsr SUBR | מצביע התכנית יקבל את כתובת התווית SUBR, ולכן ההוראה הבאה שתבצע תהיה במען SUBR. כתובת החזרה מהשגרה נשמרת במחסנית. |

קבוצת ההוראות השלישית:

אלו הן הוראות ללא אופרנדים. קידוד ההוראה מורכב ממילה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד (סיביות 2-5) במילה הראשונה של ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השייכות לקבוצה זו הן: hlt, rts.

| הוראה | הפעולה המתבצעת | דוגמה | הסבר דוגמה |
|-------|--|-------|--|
| rts | מתבצעת חזרה משיגרה. הערך שבראש המחסנית של המחשב מוצא מן המחסנית, ומוכנס למצביע התוכנית (PC). | rts | ההוראה הבאה שתבצע תהיה זו שאחרי הוראת jsr שקראה לשגרה. |
| hlt | עצירת ריצת התוכנית. | hlt | התכנית עוצרת מיידית |

מבנה תכנית בשפת אסמבלי :

תכנית בשפת אסמבלי בנויה ממקראים וממשפטים (statements).

מקראים :

מקראים הם קטעי קוד הכוללים בתוכם משפטים. בתוכנית ניתן להגדיר מקרו ולהשתמש בו במקומות שונים בתוכנית. השימוש במקרו ממקום מסוים בתוכנית יגרום לפרישת המקרו לאותו מקום.

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

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

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

```
.
.
.
m1
.
.
m1
.
.
.
```

בדוגמה זו, השתמשנו פעמיים במקרו m1, התוכנית לאחר פרישת המקרו תיראה כך :

```
.
.
.
inc r2
mov A,r1
.
.
inc r2
mov A,r1
.
.
.
```

התוכנית לאחר פרישת המקרו היא התוכנית שהאסמבלר אמור לתרגם.

הנחיות לגבי מאקרו:

- אין במערכת הגדרות מאקרו מקוננות.
- שם של הוראה או הנחיה לא יכול להיות שם של מאקרו.
- ניתן להניח שלכל שורת מאקרו בקוד המקור קיימת סגירה עם שורת endmacro (אין צורך לבדוק זאת).
- הגדרת מאקרו תהיה תמיד לפני הקריאה למאקרו
- נדרש שהקדם-אסמבלר ייצור קובץ עם הקוד המורחב הכולל פרישה של המאקרו (הרחבה של קובץ המקור המתואר בהמשך). "קובץ המקור המורחב" הוא "קובץ מקור" לאחר פרישת המאקרו, לעומת "קובץ מקור ראשוני" שהוא קובץ הקלט למערכת, כולל הגדרת המאקרואים.

משפטים:

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 מכילה רק את התווים '\t' ו-' ' (טאבים ורווחים). ייתכן ובשורה אין אף תו (למעט התו \n, כלומר השורה ריקה). |
| משפט הערה | זוהי שורה בה התו הראשון הוא ';' (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 זכרון ואתחול משתנים של התכנית, 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 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 אופציונלית.

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

יש ארבעה סוגים 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 לרגיסטר r1 הערך 7.

ואילו ההוראה:

lea XYZ, r1

תכניס לרגיסטר 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 'struct'

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

```
struct8: struct 8, "xyz"
```

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

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

4. 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.

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

5. 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: hlt

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 30 תווים.

הגדרה של תווית 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 **שונות זו מזו**.

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hello:

x:

He78902:

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

התוויות מקבלות את ערכה בהתאם להקשר בו היא מוגדרת. תוויות בהנחיות `string`, `data`, `struct`, תקבל את ערך מונה הנתונים (`data counter`) הנוכחי, בעוד שתוויות המוגדרות 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).

מספר:

מספר חוקי מתחיל בסימן אופציונלי: `+` או `-` ולאחריו סדרה של ספרות בבסיס עשרוני. לדוגמה: `123`, `-5`, `76`. הם מספרים חוקיים. אין תמיכה בשפת האסמבלר שלנו 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"`.

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

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

שתי הסיביות בשדה `A,R,E` יכולו את אחד הערכים הבינאריים: `00`, `10`, או `01`. המשמעות של כל ערך מפורטת להלן.

האות '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 מיידי). במקרה זה שתי הסיביות הימניות יכולו את הערך `00`.

האות '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). במקרה זה שתי הסיביות הימניות יכולו את הערך `10`.

האות 'E' (קיצור של `external`) באה לציין שתוכן המילה תלוי בערכו של סמל חיצוני (`external`) (למשל מילה המכילה כתובת של תווית חיצונית, כלומר תווית שאינה מוגדרת בקובץ המקור). במקרה זה שתי הסיביות הימניות יכולו את הערך `01`.

כאשר האסמבלר מקבל כקלט תוכנית בשפת אסמבלר, עליו לטפל תחילה בפרישת המקרואים, ורק לאחר מכן לעבור על התוכנית אליה נפרשו המקרואים. כלומר, פרישת המקרואים תעשה בשלב "קדם אסמבלר", לפני שלב האסמבלר (המתואר בהמשך). אם התכנית אינה מכילה מקרו, תוכנית הפרישה תהיה זהה לתכנית המקור.

דוגמה לשלב קדם אסמבלר. האסמבלר מקבל את התוכנית הבאה בשפת אסמבלר:

```

MAIN:      mov     S1.1,LENGTH
           add     r2,STR
LOOP:      jmp     END
           macro m1
               inc     K
               mov     S1.2,r3
           endmacro
           prn     #-5
           sub     r1, r4
           m1
           bne     LOOP
END:        hlt
STR:        .string "abcdef"
LENGTH:     .data  6,-9,15
K:          .data  22
S1:         .struct 8, "ab"

```

תחילה האסמבלר עובר על התוכנית ופורש את כל המאקרואים הקיימים בה. רק אם תהליך זה מסתיים בהצלחה, ניתן לעבור לשלב הבא. בדוגמה זו, התוכנית לאחר פרישת המאקרו תיראה כך:

```

MAIN:      mov     S1.1,LENGTH
           add     r2,STR
LOOP:      jmp     END
           prn     #-5
           sub     r1, r4
           inc     K
           mov     S1.2,r3
           bne     LOOP
END:        hlt
STR:        .string "abcdef"
LENGTH:     .data  6,-9,15
K:          .data  22
S1:         .struct 8, "ab"

```

קוד התכנית, לאחר הפרישה, ישמר בקובץ חדש, כפי שיוסבר בהמשך.

אלגוריתם שלדי של קדם האסמבלר

נציג להלן אלגוריתם שלדי לתהליך קדם האסמבלר. לתשומת 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 מאקרו המופיע בטבלת המאקרו (כגון m1)? אם כן, החלף את שם המאקרו והעתק במקומו את כל השורות המתאימות מהטבלה לקובץ, חזור ל- 1. אחרת, המשך.
3. האם השדה הראשון הוא " **macro** " (התחלת הגדרת מאקרו)? אם לא, עבור ל- 6.
4. הדלק דגל "יש **macro**".

5. (קיימת הגדרת מאקרו) הכנס לטבלת שורות מאקרו את שם המאקרו (לדוגמה m1).
6. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 9 (סיום).
אם דגל "יש macro" דולק ולא זוהתה תווית endmacro הכנס את השורה לטבלת המאקרו ומחק את השורה הנ"ל מהקובץ. אחרת (לא מאקרו) חזור ל 1.
7. האם זוהתה תווית endmacro? אם כן, מחק את התווית מהקובץ והמשך. אם לא, חזור ל- 6.
8. כבה דגל "יש macro". חזור ל- 1. (סיום שמירת הגדרת מאקרו).
9. סיום : שמירת קובץ מקרו פרוש.

כעת, לאחר פרישת כל המאקרואים ניתן לעבור לשלב התרגום לקוד מכונה, שלב האסמבלר.

אסמבלר עם שני מעברים

במעבר הראשון של האסמבלר, 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.

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

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

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

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

| Label | Decimal Address | Base 32 Address | Operation | Operands | Binary machine code |
|-------|-----------------|-----------------|-----------|----------------------------|---------------------|
| MAIN: | 0100 | \$% | mov | S1,LENGTH | 0000-10-01-00 |
| | 0101 | \$^ | | כתובת של הרשומה S1 | 10000101-10 |
| | 0102 | \$& | | מספר השדה הראשון ברשומה | 00000001-00 |
| | 0103 | \$* | | כתובת של LENGTH | 10000001-10 |
| | 0104 | \$< | add | r2, STR | 0010-11-01-00 |
| | 0105 | \$> | | קידוד מספר הרגיסטר | 0010-0000-00 |
| | 0106 | \$a | | כתובת של STR | 01111010-10 |
| LOOP: | 0107 | \$b | jmp | END | 1001-00-01-00 |
| | 0108 | \$c | | כתובת של END | 01111001-10 |
| | 0109 | \$d | prn | #-5 | 1100-00-00-00 |
| | 0110 | \$e | | המספר -5 | 11111011-00 |
| | 0111 | \$f | sub | r1,r4 | 0011-11-11-00 |
| | 0112 | \$g | | קידודי מספרי הרגיסטרים | 0001-0100-00 |
| | 0113 | \$h | inc | K | 0111-00-01-00 |
| | 0114 | \$i | | כתובת של K | 10000100-10 |
| | 0115 | \$j | mov | S1.2,r3 | 0000-10-11-00 |
| | 0116 | \$k | | כתובת של S1 | 10000101-10 |
| | 0117 | \$l | | מספר השדה השני ברשומה | 00000010-00 |
| | 0118 | \$m | | קידוד מספר הרגיסטר של היעד | 0000-0011-00 |
| | 0119 | \$n | bne | LOOP | 1010-00-01-00 |
| | 0120 | \$o | | כתובת של LOOP | 01101011-10 |
| END: | 0121 | \$p | hlt | | 1111-00-00-00 |
| STR: | 0122 | \$q | .string | "abcdef" | 0001100001 |
| | 0123 | \$r | | | 0001100010 |
| | 0124 | \$s | | | 0001100011 |

| | | | | | |
|---------|------|-----|---------|---------|------------|
| | 0125 | \$t | | | 0001100100 |
| | 0126 | \$u | | | 0001100101 |
| | 0127 | \$v | | | 0001100110 |
| | 0128 | %! | | | 0000000000 |
| LENGTH: | 0129 | %@ | .data | 6,-9,15 | 0000000110 |
| | 0130 | %# | | | 1111110111 |
| | 0131 | %% | | | 0000001111 |
| K: | 0132 | %% | .data | 22 | 0000010110 |
| SI | 0133 | %^ | .struct | 8, "ab" | 0000001000 |
| | 0134 | %& | | | 0001100001 |
| | 0135 | %* | | | 0001100010 |
| | 0136 | %< | | | 0000000000 |

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 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 משויך למען 0121 (עשרוני), 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. בדוגמה דלעיל, טבלת הסמלים לאחר מעבר ראשון היא:

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

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

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 0, ולכן נטענת ההוראה הראשונה במען 0. ה-IC מתעדכן בכל שורת הוראה המקצה מקום בזיכרון. לאחר שהאסמבלר

קובע מהו אורך ההוראה, ה-IC מוגדל במספר התאים (מילים) הנתפסים על ידי ההוראה, וכך הוא מצביע על התא הפנוי הבא.

כאמור, כדי לקודד את ההוראות בשפת מכונה, מחזיק האסמבלר טבלה, שיש בה 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 לרגיסטר, או להעתקת תוכן רגיסטר לרגיסטר 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 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 יודע את המען המשוך 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 מידי, או רגיסטר, וכן את הקידוד הבינארי של כל הנתונים (המתקבלים מההנחיות .data, .string, .struct).

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 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.

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 היא הסתעפות לא נכונה. התכנית כמובן לא תעבוד נכון, אך לרוב הנזק הוא יותר חמור, כי נוצרת חריגת חומרה ברגע שהמעבד מבצע פעולה שאינה חוקית.

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

גילוי שגיאות בתכנית המקור

כפי שהוסבר למעלה, הנחת המטלה היא שאין שגיאות בהגדרות המקור, ולכן שלב קדם האסמבלר אינו מכיל שלב גילוי שגיאות, לעומת זאת האסמבלר אמור לגלות ולדווח על שגיאות בתחביר של תוכנית 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".

הערה: אם יש שגיאה בקוד האסמבלי בגוף מקרו, הרי שגיאה זו יכולה להופיע ולהתגלות שוב ושוב, בכל מקום בו נפרש המקארו. נשים לב שכאשר האסמבלר בודק שגיאות, כבר לא ניתן לזהות שזה קוד שנפרש ממאקרו, כך שלא ניתן לחסוך גילויי שגיאה כפולים.

האסמבלר ידפיס את הודעות השגיאה אל הפלט הסטנדרטי stdout. בכל הודעת שגיאה יש לציין גם את מספר השורה בקובץ המקור בה זוהתה השגיאה (מניין השורות בקובץ מתחיל ב-1).

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

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

| שם ההוראה | שיטות מיעון חוקיות עבור אופרנד המקור | שיטות מיעון חוקיות עבור אופרנד היעד |
|-----------|--------------------------------------|-------------------------------------|
| mov | 0,1,2,3 | 1,2,3 |
| cmp | 0,1,2,3 | 0,1,2,3 |
| add | 0,1,2,3 | 1,2,3 |
| sub | 0,1,2,3 | 1,2,3 |
| not | אין אופרנד מקור | 1,2,3 |
| clr | אין אופרנד מקור | 1,2,3 |
| lea | 1,2 | 1,2,3 |
| inc | אין אופרנד מקור | 1,2,3 |
| dec | אין אופרנד מקור | 1,2,3 |
| jmp | אין אופרנד מקור | 1,2,3 |
| bne | אין אופרנד מקור | 1,2,3 |
| get | אין אופרנד מקור | 1,2,3 |
| prn | אין אופרנד מקור | 0,1,2,3 |
| jsr | אין אופרנד מקור | 1,2,3 |
| rts | אין אופרנד מקור | אין אופרנד יעד |

| | | |
|-----|-----------------|----------------|
| hlt | אין אופרנד מקור | אין אופרנד יעד |
|-----|-----------------|----------------|

אלגוריתם שלדי של האסמבלר

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

אנו מחלקים את קוד המכונה לשני חלקים: תמונת ההוראות (code) ותמונת הנתונים (data). לכל חלק יש מונה משלו, ונסמנם IC (מונה ההוראות - Instruction-Counter) ו-DC (מונה הנתונים - Data-Counter).

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

בכל מעבר מתחילים לקרוא את קובץ המקור מהתחלה.

מעבר ראשון

1. אתחל $DC \leftarrow 0$, $IC \leftarrow 0$.
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-16.
3. האם השדה הראשון הוא סמל? אם לא, עבור ל-5.
4. הדלק דגל "יש הגדרת סמל".
5. האם זוהי הנחיה לאחסון נתונים, כלומר, האם הנחית data, או string או struct? אם לא, עבור ל-8.
6. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם סימון (סמל מסוג data) - ערכו יהיה DC. (אם הסמל כבר נמצא בטבלה, יש להודיע על שגיאה).
7. זהה את סוג הנתונים, קודד אותם בזיכרון, עדכן את מונה הנתונים DC בהתאם לאורכם, חזור ל-2.
8. האם זו הנחיית extern או הנחיית entry? אם לא, עבור ל-11.
9. האם זוהי הנחיית extern? אם כן, הכנס כל סמל (אחד או יותר) המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים ללא ערך, עם סימון (סמל מסוג external).
10. חזור ל-2.
11. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם סימון (סמל מסוג code). ערכו יהיה IC (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
12. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא – הודע על שגיאה בשם ההוראה.
13. נתח את מבנה האופרנדים של ההוראה וחשב את L. בנה כעת את הקוד הבינארי של המילה הראשונה של הפקודה.
14. עדכן $IC \leftarrow L + IC$.
15. חזור ל-2.
16. אם נמצאו שגיאות בקובץ המקור, עצור.
17. עדכן בטבלת הסמלים את ערכם של הסמלים מסוג data, ע"י הוספת הערך הסופי של IC (ראה הסבר בהמשך).
18. התחל מעבר שני.

מעבר שני

1. אתחל $IC \leftarrow 0$.
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל-10.
3. אם השדה הראשון הוא סמל, דלג עליו.
4. האם זוהי הנחיית extern, struct, string, data? אם כן, חזור ל-2.
5. האם זוהי הנחיה entry? אם לא, עבור ל-7.
6. סמן בטבלת הסמלים את הסמלים המתאימים כ-entry. חזור ל-2.

7. השלם את קידוד האופרנדים החל מהמילה השניה בקוד הבינארי של ההוראה, בהתאם לשיטת המיעון. אם אופרנד הוא סמל, מצא את המען בטבלת הסמלים.
8. עדכן $IC \leftarrow IC + L$
9. חזור ל-2.
10. אם נמצאו שגיאות במעבר שני, עצור.
11. צור ושמור את קבצי הפלט: קובץ קוד המכונה קובץ סמלים חיצוניים, וקובץ סמלים של נקודות כניסה (ראה פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו למעלה **(לאחר שלב פרישת המאקרואים)**, ונציג את הקוד הבינארי שמתקבל במעבר ראשון ובמעבר שני. להלן שוב תכנית הדוגמה.

```

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

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

```

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

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

| Label | Decimal Address | Base 32 Address | Operation | Operands | Binary machine code |
|-------|-----------------|-----------------|-----------|----------------------------|---------------------|
| MAIN: | 0100 | \$% | mov | S1.1,LENGTH | 0000-10-01-00 |
| | 0101 | \$^ | | כתובת של רשומה S1 | ? |
| | 0102 | \$& | | מספר השדה הראשון ברשומה | 00000001-00 |
| | 0103 | \$* | | כתובת של LENGTH | ? |
| | 0104 | \$< | add | r2, STR | 0010-11-01-00 |
| | 0105 | \$> | | קידוד מספר הרגיסטר | 0010-0000-00 |
| | 0106 | \$a | | כתובת של STR | ? |
| LOOP: | 0107 | \$b | jmp | END | 1001-00-01-00 |
| | 0108 | \$c | | כתובת של END | ? |
| | 0109 | \$d | prn | #-5 | 1100-00-00-00 |
| | 0110 | \$e | | המספר -5 | 11111011-00 |
| | 0111 | \$f | sub | r1,r4 | 0011-11-11-00 |
| | 0112 | \$g | | קידודי מספרי הרגיסטרים | 0001-0100-00 |
| | 0113 | \$h | inc | K | 0111-00-01-00 |
| | 0114 | \$i | | כתובת של K | ? |
| | 0115 | \$j | mov | S1.2,r3 | 0000-10-11-00 |
| | 0116 | \$k | | כתובת של S1 | ? |
| | 0117 | \$l | | מספר השדה השני ברשומה | 00000010-00 |
| | 0118 | \$m | | קידוד מספר הרגיסטר של היעד | 0000-0011-00 |
| | 0119 | \$n | bne | LOOP | 1010-00-01-00 |
| | 0120 | \$o | | כתובת של LOOP | ? |

| | | | | | |
|----------------|------|------|---------|----------|---------------|
| <i>END:</i> | 0121 | \$p | hlt | | 1111-00-00-00 |
| <i>STR:</i> | 0122 | \$q | .string | "abcdef" | 0001100001 |
| | 0123 | \$r | | | 0001100010 |
| | 0124 | \$s | | | 0001100011 |
| | 0125 | \$t | | | 0001100100 |
| | 0126 | \$u | | | 0001100101 |
| | 0127 | \$v | | | 0001100110 |
| | 0128 | %! | | | 0000000000 |
| <i>LENGTH:</i> | 0129 | %@ | .data | 6,-9,15 | 0000000110 |
| | 0130 | %# | | | 1111110111 |
| | 0131 | %%\$ | | | 0000001111 |
| <i>K:</i> | 0132 | %%% | .data | 22 | 0000010110 |
| <i>S1</i> | 0133 | %^ | .struct | 8, "ab" | 0000001000 |
| | 0134 | %& | | | 0001100001 |
| | 0135 | %* | | | 0001100010 |
| | 0136 | %< | | | 0000000000 |

טבלת הסמלים :

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

נבצע עתה את המעבר השני ונרשום את הקוד בצורתו הסופית :

| <i>Label</i> | Decimal Address | Base 32 Address | Operation | Operands | Binary machine code |
|--------------|-----------------|-----------------|-----------|----------------------------|---------------------|
| <i>MAIN:</i> | 0100 | \$% | mov | S1.1,LENGTH | 0000-10-01-00 |
| | 0101 | \$^ | | כתובת של רשומה S1 | 10000101-10 |
| | 0102 | \$& | | מספר השדה הראשון ברשומה | 00000001-00 |
| | 0103 | \$* | | כתובת של LENGTH | 10000001-10 |
| | 0104 | \$< | add | r2, STR | 0010-11-01-00 |
| | 0105 | \$> | | קידוד מספר הרגיסטר | 0010-0000-00 |
| | 0106 | \$a | | כתובת של STR | 01111010-10 |
| <i>LOOP:</i> | 0107 | \$b | jmp | END | 1001-00-01-00 |
| | 0108 | \$c | | כתובת של END | 01111001-10 |
| | 0109 | \$d | prn | #-5 | 1100-00-00-00 |
| | 0110 | \$e | | המספר -5 | 11111011-00 |
| | 0111 | \$f | sub | r1,r4 | 0011-11-11-00 |
| | 0112 | \$g | | קידודי מספרי הרגיסטרים | 0001-0100-00 |
| | 0113 | \$h | inc | K | 0111-00-01-00 |
| | 0114 | \$i | | כתובת של K | 10000100-10 |
| | 0115 | \$j | mov | S1.2,r3 | 0000-10-11-00 |
| | 0116 | \$k | | כתובת של S1 | 10000101-10 |
| | 0117 | \$l | | מספר השדה השני ברשומה | 00000010-00 |
| | 0118 | \$m | | קידוד מספר הרגיסטר של היעד | 0000-0011-00 |
| | 0119 | \$n | bne | LOOP | 1010-00-01-00 |
| | 0120 | \$o | | כתובת של LOOP | 01101011-10 |
| <i>END:</i> | 0121 | \$p | hlt | | 1111-00-00-00 |

| | | | | | |
|---------|------|-----|---------|----------|------------|
| STR: | 0122 | \$q | .string | "abcdef" | 0001100001 |
| | 0123 | \$r | | | 0001100010 |
| | 0124 | \$s | | | 0001100011 |
| | 0125 | \$t | | | 0001100100 |
| | 0126 | \$u | | | 0001100101 |
| | 0127 | \$v | | | 0001100110 |
| | 0128 | %! | | | 0000000000 |
| LENGTH: | 0129 | %@ | .data | 6,-9,15 | 0000000110 |
| | 0130 | %# | | | 1111110111 |
| | 0131 | %% | | | 0000001111 |
| K: | 0132 | %% | .data | 22 | 0000010110 |
| SI | 0133 | %^ | .struct | 8, "ab" | 0000001000 |
| | 0134 | %& | | | 0001100001 |
| | 0135 | %* | | | 0001100010 |
| | 0136 | %< | | | 0000000000 |

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 תוכניות בתחביר של שפת האסמבלי שהוגדרה בממ"ן זה.

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

- קובץ `am`, המכיל את קובץ המקור לאחר שלב קדם האסמבלר (לאחר פרישת המאקרואים)
- קובץ `object`, המכיל את קוד המכונה.
- קובץ `externals`, ובו פרטים על כל המקומות (הכתובות) בקוד המכונה בהם יש מילת-מידע שמקודדת ערך של סמל שהוצהר כחיצוני (סמל שהופיע כאופרנד של הנחיית `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 `".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 האסמבלר באמצעות שורת הפקודה: `assembler x`:
יוצר קובץ פלט `x.ob`, וכן קבצי פלט `x.ext` ו-`x.ent` ככל שיש הנחיות `entry` או `extern`. בקובץ המקור.

אם אין מאקרו בקובץ המקור, אזי קובץ "am" יהיה זהה לקובץ "as".

אופן פעולת האסמבלר

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

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

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

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

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

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

2. שורת הוראה:

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

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

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

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

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

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

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

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

3. שורת הנחיה:

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

I. '.data'

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

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

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

II. '.struct'

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

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

III. '.string'

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

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

IV. '.entry'

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

V. '.extern'

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

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

פורמט קובץ ה-object

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

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

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

עקרונית, קובץ object מכיל את תמונת הזיכרון שתוארה כאן. קובץ object מורכב משורות של טקסט כדלקמן:
השורה הראשונה היא כותרת המכילה שני מספרים: האורך הכולל של קטע ההוראות (במילות זיכרון) ואחריה האורך הכולל של קטע הנתונים (במילות זיכרון). בין שני המספרים יש רווח אחד.
השורות הבאות מכילות את תוכן הזיכרון. בכל שורה שני מספרים: כתובת של מילה בזיכרון, ותוכן המילה. כל המספרים בקובץ object הם בבסיס 32 **ייחודי** שהוגדר לעיל.
קובץ object לדוגמה, כפי שאמור להיבנות על ידי האסמבלר, נמצא בהמשך.

פורמט קובץ ה-entries

קובץ ה-entries בנוי משורות טקסט. כל שורה מכילה שם של סמל שהוגדר כ- entry ואת ערכו, כפי שנמצא בטבלת הסמלים. הערכים מיוצגים בבסיס 32 **הייחודי** (ראו קובץ דוגמה בהמשך). אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.

פורמט קובץ ה-externals

קובץ ה-externals בנוי אף הוא משורות טקסט. כל שורה מכילה שם של סמל שהוגדר external, וכתובת בקוד המכונה בה יש קידוד של אופרנד המתייחס לסמל זה. כמובן שייתכן ויש מספר כתובות בקוד המכונה בהם מתייחסים לאותו סמל חיצוני. לכל התייחסות כזו תהיה שורה נפרדת בקובץ ה-externals. הכתובות מיוצגות בבסיס 32 **הייחודי** (ראו קובץ דוגמה בהמשך). אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.

לתשומת לב: ייתכן ויש מספר כתובות בקוד המכונה בהן מילות-המידע מתייחסות לאותו סמל חיצוני. לכל כתובת כזו תהיה שורה נפרדת בקובץ ה-externals.

נדגים את הפלט שמייצר האסמבלר עבור קובץ מקור בשם ps.as שהודגם קודם לכן.

התוכנית לאחר שלב פרישת המאקרו תיראה כך:

```

; file ps.as

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

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

```

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

טבלת הקידוד הבינארי :

| Label | Decimal Address | Base 32 Address | Operation | Operands | Binary machine code |
|---------|-----------------|-----------------|-----------|----------------------------|---------------------|
| MAIN: | 0100 | \$% | mov | S1.1,W | 0000-10-01-00 |
| | 0101 | \$^ | | כתובת של S1 | 10000101-10 |
| | 0102 | \$& | | מספר השדה הראשון ברשומה | 00000001-00 |
| | 0103 | \$* | | כתובת של W (סמל חיצוני) | 00000000-01 |
| | 0104 | \$< | add | r2, STR | 0010-11-01-00 |
| | 0105 | \$> | | קידוד מספר הרגיסטר | 0010-0000-00 |
| | 0106 | \$a | | כתובת של STR | 01111010-10 |
| LOOP: | 0107 | \$b | jmp | W | 1001-00-01-00 |
| | 0108 | \$c | | כתובת של W (סמל חיצוני) | 00000000-01 |
| | 0109 | \$d | prn | #-5 | 1100-00-00-00 |
| | 0110 | \$e | | המספר -5 | 11111011-00 |
| | 0111 | \$f | sub | r1,r4 | 0011-11-11-00 |
| | 0112 | \$g | | קידודי מספרי הרגיסטרים | 0001-0100-00 |
| | 0113 | \$h | inc | K | 0111-00-01-00 |
| | 0114 | \$i | | כתובת של K | 10000100-10 |
| | 0115 | \$j | mov | S1.2,r3 | 0000-10-11-00 |
| | 0116 | \$k | | כתובת של S1 | 10000101-10 |
| | 0117 | \$l | | מספר השדה השני ברשומה | 00000010-00 |
| | 0118 | \$m | | קידוד מספר הרגיסטר של היעד | 0000-0011-00 |
| | 0119 | \$n | bne | L3 | 1010-00-01-00 |
| | 0120 | \$o | | כתובת של L3 (סמל חיצוני) | 00000000-01 |
| END: | 0121 | \$p | hlt | | 1111-00-00-00 |
| STR: | 0122 | \$q | .string | "abcdef" | 0001100001 |
| | 0123 | \$r | | | 0001100010 |
| | 0124 | \$s | | | 0001100011 |
| | 0125 | \$t | | | 0001100100 |
| | 0126 | \$u | | | 0001100101 |
| | 0127 | \$v | | | 0001100110 |
| | 0128 | %! | | | 0000000000 |
| LENGTH: | 0129 | %@ | .data | 6,-9,15 | 0000000110 |
| | 0130 | %# | | | 1111110111 |
| | 0131 | %% | | | 0000001111 |
| K: | 0132 | %%% | .data | 22 | 0000010110 |
| SI | 0133 | %^ | .struct | 8,"ab" | 0000001000 |
| | 0134 | %& | | | 0001100001 |
| | 0135 | %* | | | 0001100010 |
| | 0136 | %< | | | 0000000000 |

הקובץ ps.ob :

כל תוכן הקובץ מיוצג במספרים בבסיס 32 הייחודי.

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

Base32 address Base32 code

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

הקובץ ps.ent:

כל ערכי הסמלים מיוצגים בבסיס 32 הייחודי.

LOOP \$b
LENGTH %@

כל הכתובות מיוצגות בבסיס 32 הייחודי

| | |
|----|-----|
| W | \$* |
| W | \$c |
| L3 | \$o |

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

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

סיכום והנחיות כלליות

- אורך התוכנית, הניתנת כקלט לאסמבלר אינו ידוע מראש, ולכן אורך התוכנית המתורגמת אינו אמור להיות צפוי מראש. אולם כדי להקל במימוש האסמבלר, ניתן להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים לאחסון תמונת קוד המכונה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים וטבלת המקור), יש לממש באופן יעיל וחשכוני (למשל באמצעות רשימה מקושרת והקצאת זיכרון דינאמי).
- השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הקלט, למעט הסיומות. 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, ממשק המשתמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו לתוכנית האסמבלר כארגומנטים (אחד או יותר) 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 השוואות 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. במקרים אלו יש לבקש ולקבל אישור **מראש** ממנחה הקבוצה.

ב ה צ ל ח ה !