

מטלת מנהה (ממ"ו) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטרלה : פרויקט גמר

משקל המטרלה : 61 נקודות (חוובה)

מספר השאלות : 1

מועד אחרון להגשה : 11.08.2023

סמסטר : ס' 2023

קיימות שתי חלופות להגשת מטלות:

- שליחת מטלות באמצעות מערכת המטלות המקוונת באתר הבית של הקורס
- שליחת מטלות באמצעות דואר אלקטרוני - **באישור המנהה בלבד**
הסביר מפורט ב"נהל הגשת מטלות מנהה"

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

עליכם לכתוב תוכנת אסמבילר, עברו 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 ומקשור) עברו מערכת אוביונו.
3. קובץ makefile. הקימפול חייב להיות עם הקומפיילר gcc והדגלים : -Wall -ansi יש לנפות את כל ההודעות שモוציאה הקומפיילר, כך שהתוכנית תתකפף ללא כל הערות או זהירות.
4. דוגמאות ריצה (קלט ופלט) :
 - א. קובצי קלט בשפת אסמבלי, קובצי פלט שנוצרו מהפעלת האסמבילר על קבצי קלט אלה. יש להציגים שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האסמבלי.
 - ב. קובצי קלט בשפת אסמבלי המציגים מגוון רחב של סוגים של מילים אסמבלי (ולכן לא נוצרים קבצי פלט), ותדפסי המשך המראים את ה Hodootot השגיאה שモוציאה האסמבילר.

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

זכור מספר היבטים חשובים של כתיבת קוד טוב :

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

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

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

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

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

רקע כללי ומטרת הפרויקט

כידוע, קיימות שפות תוכנות רבות, ומספר גדול של תוכניות, הכתובות בשפות שונות, עשויות לרוץ באופןו 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 ספירות ביןarieות. יחידת העיבוד המרכזית - היע"מ (CPU) - יודעת לפרק את הרץ הזה לקטעים קטעים בעלי משמעות: הוראות, מענים ונתונים.

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

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מגוון פעולות פשוטות, הנקראות **הוראות המכונה**, ולשם כך היא משתמשת ברגיסטרים (registers - אוגרים) הקיימים בתחום היע"מ, ובזיכרון המחשב.

דוגמאות: העברת מספר מתא בזיכרון לרегистר ביע"מ או בחזרה, הוספת 1 למספר הנמצא ברגיסטר, בדיקה האם מספר המאוחסן ברגיסטר שווה לאפס, חיבור וחיסור בין שני רגיסטרים, וכך'.

הוראות המכונה ושילובים שלתן הן המרכיבות תוכנית כפי שהיא טעונה לזכרון בזמן ריצתה. כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנן), תתרגם 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 בינהריו של סדרת הוראות המכונה המרכיבות את התוכנית. קוד זה אינו קריא למשתמש, ולכן לא נוח לקודד (או לקרוא) תוכניות ישירות בשפת מכונה. **שפת אסמבלי (assembly language)** היא שפת תוכנות מאפשרת לייצג את הוראות המכונה בצורה סימבולית קלה ונוחה יותר לשימוש. כМОון שיש צורך לתרגם את הייצוג הסימבולי לקוד בשפת מכונה, כדי שהתוכנית תוכל לרוץ במחשב. תרגום זה נעשה באמצעות כל שנקרא **אסambilר (assembler)**.

כידוע, לכל שפת תוכנות עילית יש מהדר (compiler), או מפרש (interpreter), המתרגמים תוכניות מקור לשפת מכונה. האסambilר משמש בתפקיד דומה עבור שפת אסambilי.

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

תפקידו של האסambilר הוא לבנות קובץ המכיל קוד מכונה, מקובץ נתון של תוכנית הכתובת בשפת אסambilי. 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 נעסק במאין זה.

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

لتשומת לב : בהסברים הכלליים על אופן עבודת תוכנת האסטブル, תהיה מדי פעם התייחסות גם לעובדות שלבי הקישור והטיעינה. התייחסויות אלה נועדו על מנת לאפשר לכם להבין את המשך תהליך העיבוד של הפלט של תוכנת האסטブル. אין לטעות : עליכם לכתוב את תוכנית האסטブル בלבד. אין לכתוב את תוכניות הקישור והטיעינה!!!

המחשב הדמיוני ושפת האסטブル

נדיר עתה את שפת האסטブル ואת מודל המחשב הדמיוני, עברו פרויקט זה.
הערה : תאור מודל המחשב להלן הוא חלק בלבד, ככל שנחוץ לביצוע המשימות בפרויקט.

"חומרה":

המחשב בפרויקט מורכב ממעבד CPU (יע"מ - יחידת עיבוד מרכזיית), אוגרים (רגיסטרים) וזכרון RAM. חלק מהזיכרון משמש גם כמחסנית (stack). גודלה של מילת זיכרון במחשב הוא 12 סיביות.

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

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

גודל הזיכרון הוא 1024 תאים, בכתובות 0-1023 (בסיס עשרוני), וכל תא הוא בגודל של 12 סיביות. לתא בזיכרון נקרא גם בשם "מילה". הסיביות בכל מילה ממושפרות כמו ברגיסטר. מחשב זה עובד רק עם מספרים שלמים חיוביים ושליליים, אין תמייה במספרים ממשיים. האריתמטיקה נעשית בשיטת המשלים ל-2 (2's complement).

כמו כן יש תמייה בתווים (characters), המוצגים בקוד ascii.

מבנה הוראת המכונה :

כל הוראת מכונה מוקודת **למספר מילوت זיכרון**, החל ממילה אחת ועד למקסימום שלוש מילים, בהתאם לשיטות המיעון בהן נעשה שימוש (ראו בהמשך). בכל סוג ההוראות, המבנה של המילה הראשונה זהה. מבנה המילה הראשונה בהוראה הוא כדלהלן :

11	9	8	5	4	2	1	0
מייעון אופרנד יעד	opcode	מייעון אופרנד יעד	A,R,E				

סיביות 8-5 במילה הראשונה של הוראה מהוות את קוד הפעולה (opcode). כל opcode מיוצג בשפת אסטブル 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
red	11
prn	12
jsr	13
rts	14
stop	15

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

סיביות 0-4 (A,R,E)

סיביות אלה מצינות את סוג הקידוד, האם הוא מוחלט (Absolute) , חיצוני (External) או מצריך מקום חדש (Relocatable)

ערך של 00 משמעו שהקידוד הוא מוחלט.

ערך של 01 משמעו שהקידוד הוא חיצוני.

ערך של 10 משמעו שהקידוד מצריך מקום חדש.

סיביות אלה מתווספות רק לקידודים של הוראות (לא של נתונים), והן מתווספות גם לכל המילימנוזיפוט שישי לקידודים אלה.

ראו הסבר נוסף על סיביות אלה בהמשך.

סיביות 4-2 מקודדות את מספרה של שיטת המיעון של אופרנד היעד (destination operand).

סיביות 11-9 מקודדות את מספרה של שיטת המיעון של אופרנד המקור (source operand).

בשפה שלנו קיימות שלוש שיטות מיון, שמסומנות במספרים 1,3,5.

השימוש בשיטות מיון מצריך קידוד של מילות-מידע נוספת ונספנות בקוד המכונה של ההוראה.
אם בפקודה יש אופרנד אחד, תהיה מילת מידע אחת נוספת. אם בפקודה יש שני אופרנדים,
יתכנו שתי מילות-מידע נוספת ונספנות, או מילת-מידע אחת המשותפת לשני האופרנדים, תלוי בשיטות
המיון בהן נעשה שימוש (ראו מפרט בהמשך). כאשר בקידוד הפקודה יש שתי מילות-מידע
ונספנות,ਐי 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 המיעו במכונה שלנו :

קוד	שיטת מייעו	תוכן המילאים הנוספות	אופן הכתיבה	דוגמה
1	מייעו מיידי	מילת-מידע נוספת של ההוראה מכילה את האופרנד עצמו, שהוא מספר המיצג ב- 10 סיביות, אליו מתווסף זוג סיביות של השדה .A,R,E	האופרנד הוא מספר שלם בסיס עשרוני	mov -103,@r2 בדוגמה זו האופרנד הראשון של הפקודה נתון בשיטת מעון מיידי. ההוראה כתובת את הערך -103 אל אוגר r2
3	מייעו ישיר	מילת-מידע נוספת של ההוראה מכילה כתובת של מילה בזיכרו. מילה זו בזיכרו היא האופרנד. המعن מייצג ב- 10 סיביות אליו מתווסף זוג סיביות של השדה .A,R,E	האופרנד הוא <u>תווית</u> שכבר הוצאה או שתוצחה בהמשך הקובץ. החראה נועשית על ידי כתיבת תווית בתחילת הנקהית 'data.' או 'string.', או בתחילת הוראה של התוכנית, או באמצעות אופרנד של הנקהית 'extern.'	נתונה למשל ההגדרה : x:.data 23 אפשר לכתוב הוראה : dec x בדוגמה זו, ההוראה מקטינה ב-1 את תוכן המילה שבכנתובת x בזיכרון (ה"משתנה" x).
5	מייעו אוגר ישיר	האופרנד הוא אוגר. אם האוגר משמש כאופרנד יעד, מילת-מידע נוספת של הפקודה וקובד ביחס להסיביות 2-6 את מספרו של האוגר. ואם אם האוגר משמש כאופרנד מקור, מספר האוגר יקודד ביחס להסיביות 7-11 של מילת-המידע. אם בפקודה יש שני אופרנדים, ושניהם אוגרים, הם יחלקו מילת-מידע אחת משותפת, כאשר הסיביות 2-6 הן עברו אוגר היעד, והסיביות 7-11 עברו אוגר המקור. בכל מילה נוספת מתווספות זוג סיביות של השדה .A,R,E. סיביות שאינן בשימוש יכולו 0.	האופרנד הוא שם של אוגר. שם האוגר ייכתב כאשר לפני יופיע התו @	mov @r1,@r2 בדוגמה זו, ההוראה מעתקה את תוכן אוגר 1 אל אוגר 2. בדוגמה זו שני האופרנדים הם אוגרים, אשר יקודדו למילת-מידע נוספת לשותפת אחת משותפת.

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

פרט הוראות המכונה :

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

הוראות המכונה מתחולקות לשלווש קבוצות, לפי מספר האופרנדים הנדרשים לפועלה.

קבוצת ההוראות הראשונה :
אלו הן הוראות המקבלות שני אופרנדים.

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

הורה	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
mov	מבצעת העתקה של האופרנד הראשון, אופרנד המקור (source) אל האופרנד השני, אופרנד היעד (destination). (בהתאם לשיטת המיעון).	mov A,@r1	העתק את תוכן המשטנה A (המילה שבכתובת r.1) בזיכרון אל רגיסטר r.
cmp	מבצעת "השוואה" בין שני האופרנדים שלה. אופן ההשוואה : תוכן אופרנד היעד (השני) מופחת מתוכן אופרנד המקור (הראשון), ללא שמרות תוצאת החישור. פועלות היחסור מעודכנת דגל בשם Z ("דגל האפס") ברגיסטר הסטטוס(PSW).	cmp A, @r1	אם תוכן המשטנה A זהה לתוכנו של רגיסטר r אז דגל האפס, Z, ברגיסטר הסטטוס (PSW) יודלק, אחרת הדגל יאפס.
add	אופרנד היעד (השני) מקבל את תוצאת החיבור של אופרנד המקור (הראשון) והיעד (השני).	add A,@r0	רגיסטר 0 מקבל את תוצאת החיבור של תוכן המשטנה A ותוכנו הנוכחי של רגיסטר 0.
sub	OPERAND היעד (השני) מקבל את תוצאת החישור של אופרנד המקור (הראשון) מאופרנד היעד (השני).	sub 3,@r1	רגיסטר r מקבל את תוצאת החישור של הערך 3 מתוכנו הנוכחי של הרגיסטר r.
lea	lea הוא קיצור (ראשי תיבות) של : load effective address. פועלה זו מציבה את המعن בזיכרון 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 :

ההוראות הדורשות אופרנד אחד בלבד . במקרה זה זוג הסיביות 11-9 בambilת הראשונה של קידוד ההוראה הן חסרות משמעות, מכיוון שאין אופרנד מקור (אופרנד ראשון) אלא רק אופרנד יעד (שני) . לפיכך הסיביות 11-9 יכולו תמיד 0.

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

הורה	הפעולה המתבצעת	דוגמה	הסבר דוגמה
not	היפוך ערכי הסיביות באופרנד (כל סיבית שערכה 0 תהיפוך ל-1 ולהיפוך : 1 ל-0).	not @r2	r2 ← not r2
clr	אייפוס תוכן האופרנד.	clr @r2	r2 ← 0
inc	הגדלת תוכן האופרנד באחד.	inc @r2	r2 ← r2 + 1
dec	הקטנת תוכן האופרנד באחד.	dec C	C ← C - 1

הסבר דוגמה	דוגמה	הפעולה המתבצעת	הוראה
PC \leftarrow LINE מצביע התוכנית מקבל את המعنן המוצג על ידי התווית LINE, ולפיכך הפקודה הבאה שתתבצע תהיה במען זה.	jmp LINE	קפיצה (הסתעפות) בלתי מותנית אל הוראה שנמצאת במען המוצג על ידי האופרנד. כלומר, בעת ביצוע ההוראה, מצביע התוכנית (PC) קיבל את ערך אופרנד היעד.	jmp
Z אס ערך הדגל Z ברגיסטר הסטטוס :0 (PSW) PC \leftarrow LINE	bne LINE	bne הוא קיצור (ראשי תיבות) של branch if not equal (to zero). זוהי הוראת הסטעפות מותנית. מצביע התוכנית (PC) יקבל את ערך אופרנד היעד אם ערכו של הדגל Z באוגר הסטטוס (PSW) הינו 0. כזכור, ערך הדגל Z נקבע בהוראת cmp.	bne
קוד ascii של התו הנקרא מהקלט הסטנדרטי ייכנס לאוגר r1.	red @r1	קריאה שלתו מהקלט הסטנדרטי (stdin) אל האופרנד.	red
התו אשר קוד ascii שלו נמצא באוגר r1 יזופט לפלט הסטנדרטי.	prn @r1	הדפסת התו הנמצא באופרנד, אל הפלט הסטנדרטי (stdout).	prn
push(PC) PC \leftarrow FUNC	jsr FUNC	קריאה לשגרה (סברוטינה), מצביע התוכנית (PC) הנוכחי נדחף לתוך המחסנית שబזכרון המחשב, והאופרנד מוכנס ל-PC.	jsr

קבוצת ההוראות השלישייה:

אלו הן ההוראות ללא אופרנדים. קידוד ההוראה מורכב ממילה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד במילה הראשונה של ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השויות לקבוצה זו הן : rts, stop .

הסבר דוגמה	דוגמה	הפעולה המתבצעת	הוראה
PC \leftarrow pop()	rts	חרזה משירה. הערך בראש המחסנית של זמן-ריצה מושצא מן המחסנית ומוכנס אל מצביע התוכנית (PC).	rts
התוכנית עצרת	stop	עצירת ריצת התוכנית.	stop

מבנה תוכנית בשפת אסמבלי :

תוכנית בשפת אסמבלי בנויה **ממקראים וממשפטים** (statements).

מקראים :

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

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

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

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

```
m1
```

```
m1
```

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

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

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

התוכנית לאחר פרישת המקרו היא התוכנית שהאSEMBLER אמר לתרגם.

הנחיות לנבי מאקרו :

- אין במערכת הגדרות מאקרו מוקנות.
- 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 לפניה הקראיה למאקרו
- נדרש שהקדם-אSEMBLER ייצור קובץ עם הקוד המורחב הכלול פרישה של המacro והרחבה של קובץ המקור המתואר בהמשך). "קובץ המקור המורחב" הוא "קובץ מקו"
- לאחר פרישת המacro, לעומת "קובץ מקו ראשוני" שהוא קובץ הקלט למערכת, כולל הגדרת המאקרוים.

משפטים:

קובץ מקור בשפט אסמבייל מרכיב משורות המכילות משפטיים של השפה, כאשר כל משפט מופיע בשורה נפרדת. כלומר, ההפרדה בין משפט למשפט בקובץ המקור הינה באמצעות התו ' ’ (שורה חדשה).

אורכה של שורה בקובץ המקור הוא 80 תוויים לכל היתר (לא כולל התו ').

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

סוג המשפט	הסבר כללי
משפט ריק	זהי שורה המכילה אך ורק תוויים לבנים (whitespace), כלומר מכילה רק את התווים ' ’ ו- ' ’ (טבבים ורווחים). יתרון ושורה אין אף תו (למעט התו ')’, כלומר השורה ריקה.
משפט הערת	זהי שורה בה התו הראשון הוא ' ’; (נקודה פסיק). על האסמבלר להתעלם מהלוטן משה זו.
משפט הначיה	זהו משפט המנחה את האסמבלר מה עליו לעשות כשהוא פועל על תכנית המקור. יש מספר סוגים של משפטי הначיה. משפט הначיה עשוי לגזור להקצת זיכרון ואתחל 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 ממשם ההוראה (פעלה) שעל המעבד לבצע, והאופרנדים של ההוראה.

cut נפרט לגבי סוגי המשפטיים השונים.

משפט הначיה:

משפט הначיה הוא בעל המבנה הבא :

בתחילת המשפט יכולה להופיע הגדרה של תווית (label). לתווית יש תחביר חוקי, שיתואר בהמשך. התווית היא אופצионаלית.

לאחר מכן מופיע שם הначיה. לאחר שם הначיה יופיעו פרמטרים (מספר הפרמטרים בהתאם להנchina). שם של הначיה מתחילה בתו ' ’ (נקודה) ולאחריו תוויים באותיות קטנות (lower case) בלבד.

יש לשים לב: למילوت הקוד הנוצרות ממשפט הначיה לא מצורפות זוג סיביות E,R,A וקייזר ממלא את כל 12 הסיביות של המילה.

יש ארבעה סוגי משפטי הначיה , והם :

1. ההנchina ' .data ’.

פרמטרים של ההנchina ' .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', מנהה את האסטבלר להקצות מקום בתמונה הנתונים (data image), אשר בו יאוחסנו הערכים של הפרמטרים, ולקדם את מונה הנתונים, בהתאם למספר הערכים. אם בהנחתה data. מוגדרת תווית, אז תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום), ומוכנסת אל טבלת הסמלים. דבר זה מאפשר להתייחס אל מקום מסוים בתמונה הנתונים דרך שם התווית (למעשה, זהה דרך להגדיר שם של משתנה).

כלומר אם נכתב :

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

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

אם נכתב בתכנית את ההוראה :

mov XYZ, @r1

אז בזמן ריצת התכנית יוכנס לרגיסטר 1ז ערך 7.

ואילו ההוראה :

lea XYZ,@r1

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

2. ההנחיה '.string'

להנחיה '.string' פרמטר אחד, שהוא מחוזצת חוקית. תווי המחרוזת מקודדים לפי ערכי-h-ascii המתאים, ומוכנסים אל תמונה הנתונים לפי סדרם, כל תוו בamilיה נפרדת. בסוף המחרוזת יתווסף התו '0' (הערך המספרי 0), המסמך את סוף המחרוזת. מונה הנתונים של האסטבלר יקודם בהתאם לאורך המחרוזת (בתוספת מקום אחד עבור התו המסיים). אם בשורת ההנחיה מוגדרת תווית, אז תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום) ומוכנסת אל טבלת הסמלים, בדומה כפי שנעשה עבור 'data'. (כלומר ערך התווית יהיה הכתובת בזיכרונו שבמהילה המחרוזת).

לדוגמה, ההנחיה :

STR: .string "abcdef"

מקצת בתמונה הנתונים רצף של 7 מילימ', ומתחילה את המילים לקודי-h-ascii של התווים לפי הסדר במחרוזת, ולאחריהם הערך 0 לסימון סוף מחרוזת. התווית STR מזוהה עם כתובות התחילה המחרוזת.

3. ההנחיה '.entry'

להנחיה '.entry' פרמטר אחד והוא שם של תווית המוגדרת בקובץ המקור הנוכחי (כלומר תווית שמקבלת את ערכיה בקובץ זה). מטרת ההנחיה entry. היא לאפין את התווית זו באופן שיאפשר לקוד אסטבלי הנמצא בקובץ מקור אחרים להשתמש בה (כօפרנד של הוראה).

לדוגמה, השורות :

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

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

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

4. ההנחיה 'extern'.

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

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

לדוגמה, משפט ההנחיה 'extern'. התואם לשפט ההנחיה 'entry' מהדוגמא הקודמת יהיה :

```
.extern    HELLO
```

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

لتשומת לב : תווית המוגדרת בתחילת שורת extern. הינה חסרת משמעות והאסמלר מתעלם מהתווית זו (אפשר שהאסמלר יוציא הודעה אזהרה).

משפט הוראה :

משפט הוראה מורכב מחלקים הבאים :

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

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

שם הפעולה תמיד בתוויות 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 ט-abs (אחד או יותר).

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

למשפט הוראה עם שני אופרנדים המבנה הבא :

label: opcode source-operand, target-operand

לדוגמה :

HELLO: add @r7, B

למשפט הוראה עם אופרנד אחד המבנה הבא :

label: opcode target-operand

לדוגמה :

HELLO: bne XYZ

למשפט הוראה ללא אופרנדים המבנה הבא :

label: opcode

לדוגמה :

END: stop

אפיון השדות במשפטים של שפת האסמבלי

תוויות :

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

הגדרה של תווית מסוימת בתו : ('נקודותים).תו זה אינו מהויה חלק מהתוית, אלא רק סימן המציין את סוף ההגדרה. התו ',' חייב להיות צמוד לתווית (לא רווחים).

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

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hEllo:

x:

He78902:

لتשומת לב : מילים שמורות של שפת האסמבלי (כלומר שם של פעולה או הנטיה, או שם של רגייסטר) אין יכולות לשמש גם כשם של תווית.

התווית מקבלת את ערכה בהתאם להקשר בו היא מוגדרת. תווית בהנחיות .string, מקבלת ערך מונה הנטויים (data counter) (data counter) (instruction counter) (instruction counter) (הנטוי). ערך מונה ההוראות (instruction counter) (הנטוי).

لتשומת לב : מותר המשפט הוראה להשתמש באופרנד שהוא סמל שאינו מוגדר כתווית בקובץ הנטוי, כל עוד הסמל מאופיין כחיצוני (באמצעות הנקיטת extern. כלשהו בקובץ הנטוי).

מספר :

מספר חוקי מתחילה בסימן אופציוני : '-' או '+' ולאחריו סדרה של ספרות בסיס عشرוני.

לדוגמה : -5, +76, +123 הם מספרים חוקיים. אין תמייה בשפת האסמבלי שלנוobi ביצוג בסיס אחר מאשר עשרוני, ואין תמייה במספרים שאינם שלמים.

מחuzeות:

מחuzeות חוקית היא סדרת תווים ascii נראים (שניטנים להדפסה), המוקפים במרקאות כפולות (המרקאות אינן נחשבות חלק מהמחuzeות). דוגמה למחuzeות חוקית: "hello world".

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

בכל מילה בקוד המכונה של **הוֹרָאָה** (לא של נתונים), האסמביל מכנים מידע עבור תהליך הקישור והטיענה. זהו השדה A,R,E . הميدע ישמש לתיקונים בקוד בכל פעם שייתן לזרכו לצורך הרצתה. האסמביל בונה מלכתחילה קוד שמיועד לטיענה החל מכתובות התחלה. התיקונים יאפשרו לטען את הקוד בכל פעם למקום אחר, בלי צורך לחזור על תהליך האסמביל.

שתי הסיבות בשדה A,R,E יכולו את אחד הערכיהם הבינאיים : 00, 10, או 01. המשמעות של כל ערך מפורטת להלן.

האות 'A' (קיוצר של absolute) בא להציג שתוכן המילה אינו תלוי במקום בו ייתן בפועל בפועל קוד המכונה של התכנית בעת ביצועה (למשל מילה המכילה אופrnd מידי).
במקרה זה שתי הסיבות הימניות יכולו את הערך 00.

האות 'R' (קיוצר של relocatable) בא להציג שתוכן המילה תלוי במקום בו ייתן בפועל קוד המכונה של התכנית בעת ביצועה (למשל מילה המכילה כתובות של תוויות המוגדרת בקובץ המוקור). במקרה זה שתי הסיבות הימניות יכולו את הערך 10.

האות 'E' (קיוצר של external) בא להציג שתוכן המילה תלוי בערכו של סמל חיצוני (external) (למשל מילה המכילה כתובות של תוויות חיצונית, כולל תווית שאינה מוגדרת בקובץ המוקור).
במקרה זה שתי הסיבות הימניות יכולו את הערך 01.

כאשר האסמביל מקבל תוכנית בשפת אסמביל, עליו לטפל תחילת בפרישת המאקרואים, ורק לאחר מכן לעבור על התוכנית אליה נפרשו המקרואים. כמובן, פרישת המקרואים תעשה בשלב "קדם אסמביל", לפני שלב האסמביל (המtoaר בהמשך).
אם התכנית אינה מכילה מקרו, תוכנית הפרישה תהיה זהה לתכנית המקורי.

דוגמה לשלב קדם אסמביל. האסמביל מקבל את התוכנית הבאה בשפת אסמביל :

```

MAIN:    mov   @r3 ,LENGTH
LOOP:    jmp   L1
         mcre m1
         sub   @r1, @r4
         bne   END
         endmcre
         prn   -5
         bne   LOOP
         m1
L1:      inc   K
         bne   LOOP
END:     stop
STR:     .string "abcdef"
LENGTH:  .data  6,-9,15
K:       .data  22

```

תחילת האסמבילר עובר על התוכנית ופורש את כל המאקרוואים המקוריים בה. רק אם תהליך זה מסתיים בהצלחה, ניתן לעבור לשלב הבא. בדוגמה זו, התוכנית לאחר פרישת המאקרו תיראה כך:

```

MAIN:      mov    @r3 ,LENGTH
LOOP:      jmp    L1
           prn   -5
           bne   LOOP
           sub   @r1, @r4
           bne   END
L1:        inc    K
           bne   LOOP
END:       stop
STR:       .string "abcdef"
LENGTH:    .data  6,-9,15
K:         .data  22

```

קוד התוכנית, לאחר הפרישה, ישמר בקובץ חדש, כפי שIOSCAR משתמש:

אלגוריתם שלדי של קדם האסמבילר

נציג להלן אלגוריתם שלדי לתהיליך קדם האסמבילר. لتשומת לב: אין חובה להשתמש דווקא באלגוריתם זה:

1. קרא את השורה הבאה מקובץ המקור. אם נגמר הקובץ, עبور ל- 9 (סיום).
2. האם השדה הראשון הוא שם מאקרו המופיע בטבלת המאקרו (כגון t1m)? אם כן, החלף את שם המאקרו והעתק במקומו את כל השורות המתאימות מהטבלה לקובץ, חזור ל- 1. אחרת המשך.
3. האם השדה הראשון הוא **"mero"** (התחלת הגדרת מאקרו)? אם לא, עبور ל- 6.
4. הדלק דגל **"יש mero"**.
5. (קיימת הגדרת מאקרו) הכנס לטבלת שורות מאקרו את שם המאקרו (לדוגמא t1m).
6. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עبور ל- 9 (סיום). אם דגל **"יש mero"** דולק ולא זההה תווית **endmero** הכנס את השורה לטבלת המאקרו ומחק את השורה הניל מקובץ. אחרת (לא מאקרו) חזור ל- 1.
7. האם זההה תווית **endmero**? אם כן, מחק את התווית מהקובץ והמשך. אם לא, חזור ל- 6.
8. כבה דגל **"יש mero"**. חזור ל- 1. (סיום שמיירת הגדרת מאקרו).
9. סיום: שמיירת קובץ מקרו פרוש.

אסמבילר עם שני מעברים

במעבר הראשון של האסמבילר, יש להזיהות את הסמלים (תוויות) המופיעים בתוכנית, ולתת לכל סמל ערך מסווני שהוא המعنין בו זיכרנו שהסמל מייצג. במעבר השני, באמצעות ערכי הסמלים, וכן קודי-הפעולה ומספריו האוגרים, בונים את קוד המcona.

עליו להחליף את שמות הפעולות (mov, jmp, prn, sub, inc, bne, stop) בקוד הבינארי השקoil להם במודל המחשב שהגדכנו.

כמו כן, על האסמבילר להחליף את הסמלים K, STR, LENGTH, MAIN, LOOP, END במשמעותם של המיקומות בזיכרון שם נמצאים כל נתון או הוראה בהתאם.

נניח שקטע הקוד לעיל (הוראות ונתונים) ייטע בזכרון החל מען 100 (בסיס 10). במקרה זה נקבל את ה"תרגום" הבא:

Decimal Address	Source Code	Explanation	Binary Machine Code
0100	MAIN: mov @r3 ,LENGTH	first word of instruction source register 3 address of label LENGTH	101000001100 000110000000 00011110110
0101			
0102			
0103	LOOP: jmp L1		000100101100
0104		address of label L1	000111000110
0105	prn -5		000110000100
0106		immediate value -5	111111101100
0107	bne LOOP		000101001100
0108		address of label LOOP	000110011110
0109	sub @r1, @r4		101001110100
0110		registers r1 and r4	000010010000
0111	bne END		000101001100
0112		address of label END	000111010110
0113	L1: inc K		000011101100
0114		address of label K	001000000010
0115	bne LOOP		000101001100
0116		address of label LOOP	000110011110
0117	END: stop		000111100000
0118	STR: .string "abcdef"	ascii code 'a' ascii code 'b' ascii code 'c' ascii code 'd' ascii code 'e' ascii code 'f' ascii code '\0' (end of string)	000001100001 000001100010 000001100011 000001100100 000001100101 000001100110 000000000000
0119			
0120			
0121			
0122			
0123			
0124			
0125	LENGTH: .data 6,-9,15	integer 6 integer -9 integer 15	000000000110 11111110111 0000000001111
0126			
0127			
0128	K: .data 22	integer 22	000000010110

האסמבילר מחזיק טבלה שבה רשומים כל שמות הפעולות של ההוראות והקודים הבינאריים המתאים להם, וכן שמות הפעולות ניתנים להמרה לבינארי בקלות. כאשר נקרא שם פעולה, אפשר פשוט לעיין בטבלה ולמצוא את הקידוד הבינארי.

כדי לבצע המרה לבינארי של אופרנדים שכחובים בשיטות מייען המשמשות בסמלים (תוויות), יש צורך לבנות טבלה המכילה את ערכי כל הסמלים. אולם בהבדל ממקודים של הפעולות, הידעו מראש, הרי המענים בזיכרונו עבור הסמלים בשימוש התוכנית אינם ידועים, עד אשר תוכנית המקור נסקרה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד לעיל, האסמבילר אינו יכול לדעת שהסמל END משוויך למען 117 (עשרוני), אלא רק לאחר שנקראו כל שורות התוכנית.

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

במעבר הראשון נבנית טבלת סמלים בזיכרון, ובה לכל סמל שבתוכנית המקור משוויךערך מסופרי שהוא מען בזיכרון. בדוגמה דלעיל, טבלת הסמלים לאחר מעבר ראשון היא:

סמל	ערך (בסיס דצימלי)
MAIN	100
LOOP	103
L1	113
END	117

סמל	ערך (בסיס דצימלי)
STR	118
LENGTH	125
K	128

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

لتשומת לב: תפקיד האסטבלר, על שני המעברים שלו, לתרגם קובץ מקור לקוד בשפת מכונה. בגמר פעולה האסטבלר, התוכנית טרם מוכנה לטעינה לזכרון לצורך ביצוע. קוד המכונה חייב לעبور לשבי ה קישור/טעינה, ורק לאחר מכן לשלב הביצוע (שלבים אלה אינם חלק מהממן').

המעבר הראשון

במעבר הראשון נדרשים כללים כדי לקבוע איזה מען ישוק לכל סמל. העיקרונות הבסיסיים הוא לספור את המקומות בזיכרון, אותן תופסות הhorאות. אם כל הוראה תיטען בזיכרון למקום העוקב להוראה הקודמת, תציג ספירה כזאת את מען הוראה הבא. הספירה נשעת על ידי האסטבלר ומוחזקת במונה ההוראות (IC). ערך ההתחלתי של IC הוא 0, ולכן גענות הוראה הראשונה במען 0. IC-0 מתחדש בכל שורת הוראה מקצת מקום בזיכרון. לאחר שהאסטבלר קובע מהו אורך ההוראה, IC-0 מוגדל במספר התאים (מיילים) הנתפסים על ידי ההוראה, וכך הוא מציבע על התא הפניו הבא.

כאמור, כדי לקודד את הhorאות במשפט מכונה, מוחזק האסטבלר טבלה, שיש בה קידוד מתאים לכל שם פעולה. בזמן התרגומים מחליף האסטבלר כל שם פעולה בקידוד שללה. כמו כן, כל אופרנד מוחלף בקידוד מותאים, אך פעולה החלפה זו אינה כה פשוטה. הhorאות משתמשות בשיטות מייען מגוונות לאופרנדים. אותה פעולה יכולה לקבל משמעותות שונות, בכל אחת משיטות המייען, ולכן ניתן לה קידודים שונים לפי שיטות המייען. לדוגמה, פעולה ההזזה sow יכולה להתיחס להעתקת תוכן תא זיכרון לרגיסטר, או להעתקת תוכן רגיסטר אחר, וכן הלאה. ככל אפשרויות כזו של sow עשוי להתאים קידוד שונה.

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

כאשר נתקל האסטבלר בתווית המופיעת בתחילת ההוראה, הוא יודע שלפניו הגדרה של תווית, ואז הוא משייך לה מען – תוכנו הנוכחי של IC. כך מקבלות כל התוויות את מעניין בעט ההגדירה. תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנוסף לשם התווית גם את המعن ומאפיינים נוספים. כאשר תהיה התיחסות לתווית באופרנד של הוראה כלשהי, יוכל האסטבלר לשולף את המعن המתאים מטבלת הסמלים.

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

bne A
.
.
.
.
A:

כאשר מגיע האסטבלר לשורת הסתעפות (A bne), הוא טרם נתקל בהגדרת התווית A וכמוון לא יודע את המعن המשויך לתווית. לכן האסטבלר לא יכול לבנות את הקידוד הבינאי של האופרנד A. נראה בהמשך כיצד נפתרת בעיה זו.

בכל מקרה, תמיד אפשר לבנות מעבר הראשון את הקידוד הבינארי המלא של המילה הראשונה של כל הוראה, את הקידוד הבינארי של מילת-המידע נוספת של אופרנד מיידי, או רגיסטר, וכן את הקידוד הבינארי של כל הנתונים (המתקבלים מהחניות `.string`, `.data`).

המעבר השני

ראינו שבמעבר הראשון, האסטבלר אינו יכול לבנות את קוד המכונה של אופרנדים המשתמשים בסמלים שעדין לא הוגדרו. רק לאחר שהאסטבלר עבר על כל התכנית, כך שכל הסמלים נכנסו כבר לטבלת הסמלים, יכול האסטבלר להשלים את קוד המכונה של כל האופרנדים.

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

הפרזה הוראות ונתונים

בתכנית מבחןים שני סוגים של תוכן: הוראות ונתונים. יש לארגן את קוד המכונה כך שתיהיה הפרזה בין הנתונים וההוראות. הפרזה הוראות והנתונים לקטועים שונים בזיכרון היא שיטה עדיפה על פני הצמדה של הגדרות הנתונים להוראות המשתמשות בהן.

אחת הסכנות הטമונות באירוע ההוראות מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית בתכנית, לנסתו "לבצע" את הנתונים כאילו היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום לתופעה זו היא הסתעפות לא נcona. התכנית כמובן לא תעבור נconi, אך לרוב הנזק הוא יותר חמור, כי נוצרת חריגת חומרה ברגע שהמעבד מבצע פעולה שאינה חוקית.

האסטבלר שלו חייב להפריד, בקוד המכונה שהוא מייצר, בין קטוע הנתונים לקטוע ההוראות. **בລומר בקובץ הפלט (בקוד המקורי) תהייה הפרזה של הוראות ונתונים לשני קטיעים נפרדים, אם כי בקובץ הקלט אין חובה שתהייה הפרזה כזו.** בהמשך מתואר אלגוריתם של האסטבלר, ובו פרטיהם כיצד ביצע את ההפרזה.

גילוי שגיאות בתכנית המקור

כפי שהוסבר לעיל, הנחתת המטלה היא **שאין שגיאות בהגדרות המקור**, ולכן שלב קדם לאסטבלר אינו מכיל שלב גילוי שגיאות, לעומת זאת האסטבלר אמר לגלות ולדוח על **שגיאות בתחריב של תוכנית המקור**, כגון פעולה שאינה קיימת, מספר אופרנדים שגוי, סוג אופרנד שאינו מותאים לפעולה, שם אונגר לא קיים, ועוד שגיאות אחרות. כמו כן מודוא האסטבלר שככל סמל מוגדר פעם אחת בדיק.

מכאן, ככל שגיאת המתגללה על ידי האסטבלר נגרמת (בדרכו כלל) על ידי שורת קלט מסוימת.

לדוגמה, אם מופיעים שני אופרנדים בהוראה שאמור להיות בה רק אופרנד אחד, האסטבלר ייתן הודעת שגיאיה בנוסח "יותר מדי אופרנדים".

הערה: אם יש שגיאת בקוד האסטבלרי בגוף מקשו, הרי שגיאיה זו יכולה להופיע ולהתגלות שוב ושוב, בכל מקום בו נפרש המקאו. נשים לב שכאשר האסטבלר בודק שגיאות, כבר לא ניתן לזהות שזה קוד שנפרש ממשך, כך שלא ניתן לחסוך גילויי שגיאיה כפולים.

האסטבלר ידפיס את הודעת השגיאיה אל הפלט הסטנדרטי `cout`. בכל הודעת שגיאיה יש לציין גם את מספר השורה בקובץ המקורי בה זוהתה השגיאיה (מנין השורה בקובץ מתחילה ב-1).

لتשומת לב: האסטבלר אינו עוצר את פעולתו אחרי שנמצאה השגיאיה הראשונה, אלא ממשיך לעبور על הקלט כדי לגלו שגיאות נוספות, ככל שישן. כਮובן שאין כל טעם לייצר את קבצי הפלט אם נתגלו שגיאות (ممילא אי אפשר להשלים את קוד המכונה).

התבלה הבאה מפרטת מהן שיטות המיון 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	1,3,5	3,5
cmp	1,3,5	1,3,5
add	1,3,5	3,5
sub	1,3,5	3,5
not	אין אופרנד מקור	אין אופרנד יעד
clr	אין אופרנד מקור	3,5
lea	אין אופרנד מקור	3
inc	אין אופרנד מקור	3,5
dec	אין אופרנד מקור	3,5
jmp	אין אופרנד מקור	3,5
bne	אין אופרנד מקור	3,5
red	אין אופרנד מקור	3,5
prn	אין אופרנד מקור	1,3,5
jsr	אין אופרנד מקור	3,5
rts	אין אופרנד מקור	אין אופרנד יעד
stop	אין אופרנד מקור	אין אופרנד יעד

אלגוריתם שלדי של האסטמבלר

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

. אנו מחלקים את קוד המכונה לשני חלקים : **תמונה ההוראות (code)** ו**תמונה הנתונים (data)** לכל חלק יש מונה משלהו, ונסמן IC (מונה ההוראות - Monia of Instructions) ו- DC (מונה הנתונים - Data-Counter).

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

בכל מעבר מתחילה לקרוא את קובץ המקור מהתחלתה.

מעבר ראשון

- .1. **אתחל DC $\leftarrow 0$, IC $\leftarrow 0$.**
- .2. קרא את השורה הבאה מקובץ המקור. אם גמר קובץ המקור, עברו ל-16.
- .3. האם השדה הראשון הוא סמל? אם לא, עברו ל-5.
- .4. הדלק דגל "יש הגדרת סמל".
- .5. האם זהה הנחיה לאחסון נתונים, כלומר, אם הנתונית data.string או ?.string? אם לא, עברו ל-8.
- .6. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם סימונו (סמל מסוג data-).
ערכו יהיה DC. (אם הסמל כבר נמצא בטבלה, יש להזדיע על שגיאיה).
- .7. זהה את סוג הנתונים, קודו אותם בזיכרון, עדכן את מונה הנתונים DC בהתאם לאותם, חזרו ל-2.
- .8. האם זו הנחיה extern. או הנחיה entry.? אם לא, עברו ל-11.
- .9. האם זהה הנחיה extern.? אם כן, הכנס כל סמל (אחד או יותר) המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים ללא ערך, עם סימונו (סמל מסוג external).
- .10. חזרו ל-2.

11. אם יש הגדרת סמל, הכנס אותו לטבלת הסמלים עם סימון (סמל מסווג code). ערכו יהיה IC (אם הסמל כבר נמצא בטבלה יש להודיע על שגיאה).
12. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא – הודיע על שגיאה בשם הוראה.
13. נתח את מבנה האופרנדים של ההוראה וחשב את L. בנה כתה את הקוד הבינארי של המילה הראשונה של הפוקודה.
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 מסווג data, ע"י הוספה הערך הסופי של IC (ראו הסבר בהמשך).
18. התחל מעבר שני.

מעבר שני

1. אתחל 0 IC \leftarrow 0
2. קרא את השורה הבאה מקובץ המקור. אם גמר קובץ המקור, עברו ל-10.
3. אם השדה הראשון הוא סמל, דרג עליו.
4. האם זהה הנחיה ?.data, .string, .extern ?: אם כן, חזור ל-2.
5. האם זהה הנחיה ?.entry?: אם לא, עברו ל-7.
6. סמן בטבלת הסמלים את הסמלים המתאימים כ-.entry. חזור ל-2.
7. השלים את קידוד האופרנדים החל מהמילה השניה בקוד הבינארי של ההוראה, בהתאם לשיטת המיעון. אם אופרנד הוא סמל, מצא את המعن בטבלת הסמלים.
8. עדכן L \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 חיוניים, וקובץ סמלים של נקודות כניסה (ראו פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו לעיל **(לאחר שלב פרישת המאקרוואים)**, ונציג את הקוד הבינארי שモתקבל במעבר ראשון ובמעבר שני. להלן שוב תוכנית הדוגמה.

```

MAIN:    mov   @r3 ,LENGTH
LOOP:    jmp   L1
          prn   -5
          bne   LOOP
          sub   @r1, @r4
          bne   END
L1:      inc   K
          bne   LOOP
END:     stop
STR:     .string "abcdef"
LENGTH:  .data  6,-9,15
K:       .data  22

```

נבצע עתה מעבר ראשון על הקוד הנוכחי. נבנה את טבלת הסמלים. כמו כן, נבצע במעבר זה גם את קידוד כל הנתונים, וקידוד המילה הראשונה של כל הוראה. את החלקים שעדיין לא מותרגמים במעבר זה, נשאיר כ모ות שם.

אנו מניחים שהקוד ייטען לזכרון החל מהמען 100 (בבסיס דצימלי).

Decimal Address	Source Code	Explanation	Binary Machine Code
0100	MAIN: mov @r3 ,LENGTH	first word of instruction source register 3 address of label LENGTH	101000001100 000110000000 ?
0101			
0102			
0103	LOOP: jmp L1		000100101100 ?
0104		address of label L1	
0105	prn -5		000110000100 111111101100
0106		immediate value -5	
0107	bne LOOP		000101001100 ?
0108		address of label LOOP	
0109	sub @r1, @r4		101001110100 000010010000
0110		registers r1 and r4	
0111	bne END		000101001100 ?
0112		address of label END	
0113	L1: inc K		000011101100 ?
0114		address of label K	
0115	bne LOOP		000101001100 ?
0116		address of label LOOP	
0117	END: stop		000111100000
0118	STR: .string "abcdef"	ascii code 'a' ascii code 'b' ascii code 'c' ascii code 'd' ascii code 'e' ascii code 'f' ascii code '\0' (end of string)	000001100001 000001100010 000001100011 000001100100 000001100101 000001100110 000000000000
0119			
0120			
0121			
0122			
0123			
0124			
0125	LENGTH: .data 6,-9,15	integer 6 integer -9 integer 15	000000000110 111111101111 000000001111
0126			
0127			
0128	K: .data 22	integer 22	000000010110

טבלת הסמלים :

סמל	ערך (בבסיס דצימלי)
MAIN	100
LOOP	103
L1	113
END	117
STR	118
LENGTH	125
K	128

נכזע עתה את המעבר השני ונרשום את הקוד בצורתו הסופית:

Decimal Address	Source Code	Binary Machine Code
0100	MAIN: mov @r3 ,LENGTH	101000001100 000110000000
0101		

0102		000111110110
0103	LOOP: jmp L1	000100101100
0104		000111000110
0105	prn -5	000110000100
0106		111111101100
0107	bne LOOP	000101001100
0108		000110011110
0109	sub @r1, @r4	101001110100
0110		000010010000
0111	bne END	000101001100
0112		000111010110
0113	L1: inc K	000011101100
0114		001000000010
0115	bne LOOP	000101001100
0116		000110011110
0117	END: stop	000111100000
0118	STR: .string "abcdef"	000001100001
0119		000001100010
0120		000001100011
0121		000001100100
0122		000001100101
0123		000001100110
0124		000000000000
0125	LENGTH: .data 6,-9,15	000000000110
0126		111111110111
0127		000000001111
0128	K: .data 22	000000010110

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

קבצי קלט ופלט של האסמבלייר

בהתפעלה של האסמבלייר, יש להעביר אליו באמצעות ארגומנטים של שורת הפקודה (command line arguments) רשימה של שמות קבצי מקור (אחד או יותר). אלו הם קבצי טקסט, וביהם תוכניות בתחביר של שפת האסמבלי שהוגדרה במיל'ין זה.

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

- קובץ `main`, המכיל את קוד המקור לאחר שלב קדם האסמבלייר (לאחר פרישת המאקרוואים).
- קובץ `object`, המכיל את קוד המוכנה.
- קובץ `externals`, ובו פרטים על כל המיקומות (הכתובות) בקוד המוכנה בהם יש מילת-מידע שמקודדת ערך של סמל שהוצע בחיצוני (סמל שהופיע כאופרנד של הנחיה `extern`, ומופיע בטבלת הסמלים `c-`). `extern`.
- קובץ `entries`, ובו פרטים על כל סמל שמצוחר כנקודות כניסה (סמל שהופיע כאופרנד של הנחיה `entry`, ומופיע בטבלת הסמלים `c-`). `entry`.

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

שמות קבצי המקור חייבים להיות עם הסיומת `".as"`. למשל, השמות `x.as`, `y.as`, ו-`hello.as` הן שמות חוקיים. העברת שמות הקבצים הללו כארוגומנטים לאסמבלייר נעשית לא ציון הסיומת.

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

assembler x y hello

תրץ את האסמבילר על הקבצים : .x.as, y.as, hello.as

שמות קבצי הפלט מבוססים על שם קובץ הקלט, כפי שהופיע בשורת הפקודה, בתוספת סימנת מתאימה : הסימנת "object.am" עבר קובץ לאחר פרישת מאקרו, הסימנת "ob" עבר קובץ ה-.externals, והסימנת "ext.entries" עבר קובץ ה-.ent.

לדוגמה, בהפעלת האסמבילר באמצעות שורת הפקודה : assembler x.y, וכן קבץ פלט .ext.x.entry או .extern.as. בקובץ המקור. אם אין מאקרו בקובץ המקור, אז קובץ "am" יהיה זהה לקובץ "as".

אופן פועלות האסמבילר

נרחיב כאן על אופן פועלות האסמבילר, בנוסף לאלגוריתם השודי שניתן לעיל.

האסמבילר מחזק שני מערכים, שייקראו להן מערך ההוראות ומערך הנתונים. מערכים אלו נתונים למעשה תמונה של זיכרון המcona (גודל כל כניסה במערך זהה לגודלה של מילת המכונה : 12 סיביות). במערך ההוראות מכניס האסמבילר את הקידוד של הוראות המכונה שנקראו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמבילר את קידוד הנתונים שנקראו מקובץ המקור (שורות מסווג .data.string).

לאסמבילר יש שני מונחים : מונה ההוראות (IC) ומונה הנתונים (DC). מונחים אלו מצבעים על המיקום הבא הפנוי במערכות לעיל, בהתאם. כמשמעותו של האסמבילר לעבר על קובץ מקור, שני מונחים אלו מואפסים.

בנוסף יש לאסמבילר טבלה, אשר בה נאספota כל התוויות בהן נתקל האסמבילר במהלך המעבר על הקובץ. לטבלה זו קוראים טבלת סמלים (symbol-table). לכל סמל (תווית) נשמרם اسمו, ערכו וטיפוסו (external או relocatable).

האסמבילר קורא את קובץ המקור שורה אחר שורה, מחייב מהסוג שורה (הערה, הוראה, הנחיה או שורה ריקה) ופועל בהתאם.

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

2. שורת הוראה :

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

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

קביעת שיטת המייעון נעשית בהתאם לתחביר של האופרנד, כפי שהוסבר לעיל בהגדרת שיטות המייעון. למשל, מספר מציין מיידי, תווית מציין מייעון ישיר, שם של רגיסטר עם @ לפני מציין מייעון רגיסטר, ועוד'.

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

אם זהה פעולה בעלת שני אופרנדים, אזי האסמבלר מכניס לערך ההוראות, במקומות עליו מציביע 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 שיטתה המייען. בנוסף "משרויין" האסמבלר מักם בערך עבור המילים הנוספות הנדרשות עבור הוראה זו, ומגדיל את מונה ההוראות בהתאם. אם אחד או שני האופרנדים הם בשיטת מייען גיסטר או מיידי, האסמבלר מקודד כעת את המילים הנוספות הרלוונטיות בערך ההוראות.

אם זהה פעולה בעלת אופרנד אחד בלבד, כלומר אין אופרנד מקור, אזי הקידוד הינו זהה לעיל, פרט לשיביות של שיטת המייען של אופרנד המקור במילה הראשונה, אשר ייכלו תמיד 0, מכיוון שאין רלוונטיות לפעולה.

אם זהה פעולה ללא אופרנדים (rts, stop) אזי תקודד רק המילה הראשונה (והיחידה). הסיבות של שיטות המייען של שני האופרנדים ייכלו 0.

אם בשורת ההוראה קיימת תווית, אזי התווית מוכנסת אל בטבת 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 רשימות המספרים, המופיעעה לאחר '.data', מכניס כל מספר אל מערך הנתונים, ומגדם 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, וכך גם מסומן שההגדלה ניתנה בחלק הנתונים.

בסוף המעבר הראשון, ערך התווית יעדכו בטבלת הסמלים על ידי הוספה ה-IC (כלומר הוספה האורך הכלול של קידוד כל ההוראות). הסיבה לכך היא שבתמונה קוד המcona, הנתונים מופרדים מהוראות, וכל הנתונים יופיעו אחרי כל ההוראות (או תאור קבצי הפלט בהמשך).

II. '.string'
הטיפול ב-'string' דומה ל-'data', אלא שקובדי ascii של התווים הם אלו המוכנסים אל מערך הנתונים (כלתו בכניסה נפרדת). לאחר מכן מוכנס הערך 0 (המצียน סוף מחוזת) אל מערך הנתונים. מונה הנתונים מוקודם באורך המחרוזת + 1 (כי גם האפס בסוף המחרוזת תופס מקום).

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

III. '.entry'
זהה בקשה לאסמבלר להכניס את התווית המופיעעה כאופרנד של '.entry'. אל קובץ ה-entries. האסמבלר רושם את הבקשה ובסיום העבודה, התווית הניל תירשם בקובץ ה-entries.

IV. '.extern'
זהה הצהרה על סמל (תווית) המוגדר בקובץ אחר, ואשר קטע האסמבלר בקובץ הנוכחי עושה בו שימוש. האסמבלר מכניס את הסמל אל בטבת הסמלים. ערכו הוא 0 (הערך האמתי לא ידוע, ויקבע 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 הצהרה עליו ניתנת בהמשך הקובץ (אם באופן ישיר על ידי תווית ואם באופן עקיף על ידי הנחיתות extern).

פורמט קובץ ה-object

האSEMBLER בונה את תМОונת זיכרונם המכוֹנָה כז שקידוד ההוראה הראשונה מקובץ האSEMBLER יכנס למען 100 (בבסיס 10) בזיכרונו, קידוד ההוראה השנייה יכנס למען העוקב אחרי ההוראה הראשונה (תלויה במספר המיללים של ההוראה הראשונה), וכך הלאה עד להוראה אחרת.

מיד לאחר קידוד ההוראה אחרת, מכניסים לתМОונת הזיכרונו את קידוד הנתונים שנבנו על ידי הנקודות 'string'. הנתונים יוכנסו בסדר בו הם מופיעים בקובץ המקור. אופרנד של הוראה שמתיחס לSAMPLE שהוגדר באותו קובץ, יקודד כך שיצבע על המיקום המתאים בתМОונת הזיכרונו שבונה האSEMBLER.

נשים לב שהמשתנים מופיעים בתМОונת הזיכרונו אתריי ההוראות. זהה הסיבה בגללה יש לעדכן בטבלת הסמלים, בסוף המעבר הראשוני, את ערכי הסמלים המגדירים נתונים (סמלים מסווג).(.)data

עקרונית, קובץ object מכיל את תМОונת הזיכרונו שתוארה כאן. קובץ object מורכב משורות של טקסט כדלקמן:

השורה הראשונה היא כותרת המכילה שני מספרים בסיס 10: האורך הכלול של קטע ההוראות (במילים זיכרונו) ואחריו האורך הכלול של SAMPLE הנתונים (במילים זיכרונו). בין שני המספרים יש רווח אחד.

השורות הבאות מכילות את תוכן הזיכרונו. בכל שורה מופיע תוכן של מילה אחת, לפי הסדר החל מהמילה בכתבoted 100. תוכן המילה מקודד בשיטת Base64.

ניתן לקרוא על שיטת Base64 בקישור הבא: <https://en.wikipedia.org/wiki/Base64>

מילה היא בגודל 12 ביטים. כל 6 ביטים יומו לטו מתאימים לפי הטבלה המגדירה את Base64. לפיכך בכל שורה בקובץ object (מלבד השורה הראשונה) יהיו בדיזוק שני תווים בקוד Base64.

קובץ object לדוגמה, כפי שאמור להיבנות על ידי האSEMBLER, נמצא בהמשך.

פורמט קובץ ה-entries

קובץ ה-entries בניי משורות טקסט. כל שורה מכילה שם של SAMPLE שהוגדר כ-entry ואת ערכו, כפי שנמצא בטבלת הסמלים. הערכים מיוצגים בסיס 10.

פורמט קובץ ה-externals

קובץ ה-externals בניי אף הוא משורות טקסט. כל שורה מכילה שם של SAMPLE שהוגדר [external] וכותבת בקוד המכונה בה יש קידוד של אופרנד המתיחס לSAMPLE זה. כMOVן שייתכן ויש מספר כתובות בקוד המכונה בהם מתיחסים לאותו SAMPLE חיצוני. ככל התיאחות כזו תהיה שורה נפרדת בקובץ ה-externals. הכתובות מיוצגות בסיס 10.

لتשומתך: ניתן ויש מספר כתובות בקוד המכונה בהן מילים-המידע מתיחסות לאותו SAMPLE חיצוני. לכל כתובות כזו תהיה שורה נפרדת בקובץ ה-externals.

נדגים את הפלט שמייצר האSEMBLER עבור קובץ מקור בשם ps.as שהוזגמו קודם לכך.

התוכנית לאחר שלב פרישת המאקרו תיראה כך:

```

; file ps.as

.entry LENGTH
.extern W
MAIN:    mov   @r3 ,LENGTH
LOOP:     jmp   L1
          prn   -5
          bne   W
          sub   @r1, @r4
          bne   L3
L1:      inc   K
.entry LOOP
          jmp   W
END:      stop
STR:      .string "abcdef"
LENGTH:   .data  6,-9,15
K:        .data  22
.extern L3

```

להלן טבלת הקידוד המלא הבינארי שמתתקבל מקובץ המקור, ולאחריה הפורמטים קבצי הפלט השוניים.

טבלת הקידוד הבינארי:

Decimal Address	Source Code	Binary Machine Code
0100 0101 0102	MAIN: mov @r3 ,LENGTH	101000001100 000110000000 000111110110
0103 0104	LOOP: jmp L1	000100101100 000111000110
0105 0106	prn -5	000110000100 1111111101100
0107 0108	bne W	000101001100 0000000000001
0109 0110	sub @r1, @r4	101001110100 000010010000
0111 0112	bne L3	000101001100 0000000000001
0113 0114	L1: inc K	000011101100 0010000000010
0115 0116	jmp W	000100101100 0000000000001
0117	END: stop	000111100000
0118 0119 0120 0121 0122 0123 0124	STR: .string "abcdef"	000001100001 000001100010 000001100011 000001100100 000001100101 000001100110 0000000000000
0125 0126 0127	LENGTH: .data 6,-9,15	0000000000110 111111110111 000000001111
0128	K: .data 22	000000010110

:הקובץ ps.0b

נבחין כי בדוגמה לעיל, קטע ההוראות בגודל 18 מילימ', ואילו קטע הנתונים בגודל 11 מילימ'. להלן קובץ הפלט המתאים לדוגמה זו, מקודד בשיטת Base64 (סה"כ 30 שורות, כולל הוכתרת):

הערה: שורת הוכתרת להלן אינה חלק מהקובץ, ונועדה להבירה בלבד.

18 11
OM
GA
H2
ES
HG
GE
/s
FM
AB
p0
CQ
FM
AB
Ds
IC
Es
AB
Hg
Bh
Bi
Bj
Bk
Bl
Bm
AA
AG
/3
AP
Aw

הקובץ :ps.ent

כל ערכי הסמלים מיוצגים בבסיס 10.

LOOP	103
LENGTH	125

הקובץ :ps.ext

כל הכתובות מיוצגות בbasis 10.

W	108
L3	112
W	116

لتשומת לב : אם בקובץ המקור אין הערה `extern`.extern. אז לא ייווצר עבورو קובץ ext. בדומה, אם אין בקובץ המקור הערה entry., לא ייווצר קובץ ent.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 לאסטמבלר אינו ידוע מראש, ולכן אורך התוכנית המתורגמת אינו אמור להיות צפוי מראש. אולם כדי להקל בימיוש האסטמבלר, ניתן להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכיים לאחסן תमונות קוד המכונה **בלבד**. כל מבנה נתונים אחר (למשל טבלת הסמלים וטבלת המקור), יש למשם באופן ייעיל וחסכוני (למשל באמצעות רשימה מקוישת והקצתה זיכרון דינמי).
 - השמות של קבצי הפלט צרייכים להיות תואמים לשם קובץ הקלט, למעט הסיומות. למשל, אם קובץ הקלט הוא prog.as אז קבצי הפלט שיוציאו אותו הם : prog.ob, prog.ext, prog.ent.
 - מתכוון הפעלת האסטמבלר צריכה להיות כפי הנדרש בממ"ן, ללא שינוים כלשהם. ככלומר, משיק המשמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו לתוכנית האסטמבלר כארומנטים (אחד או יותר) בשורת הפקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גרפיים למיניהם, וכו'.
 - יש להזכיר לחלק את מימיוש האסטמבלר במספר מודולים (קבצים בשפת C) לפי MISMOOT. אין לרכז MISMOOT מסווגים שונים במודול יחיד. מומלץ לחלק למודולים כגון : מעבר ראשוני, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחרيري של שורה), טבלת הסמלים, מפת הזיכרון, טבלאות קבועות (קודם הפעלה, שיטות המיעון החוקיות לכל פעולה, וכו').
 - יש להזכיר ולתעד את המימוש באופן מלא וברור, באמצעות העורות מפורחות בקוד.
 - יש לאפשר תווים לבנים עודפים בקובץ הקלט בשפת אסטמבלר. למשל, אם בשורת הוראה יש שני אופרנדים המופרדים בפסיק, אז לפניו ואחריו הפסיק מותר להיות רווחים וטאבים בכל מקום. בדומה, גם לפניו ואחריו שם הפעולה. מותירות גם שורות ריקות. האסטמבלר יתעלם מהתווים לבנים מיוחדים (כלומר יידלג עליהם).
 - הקלט (קוד האסטמבלר) עלול להכיל שגיאות תחביריות. על האסטמבלר **לגלות ולזוזה על כל השורות השגויות** בקלט. אין לעצור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להציג לפחות הודעות מפורחות מכל הניתן, כדי שאפשר יהיה להבין מה והיכן כל שגיאה. כמובן שגם קובץ קלט מכיל שגיאות, אין טעם להפיק עבورو את קבצי הפלט (.ob, .ext, .ent).
- גם ונסלם פרק ההסבירים והגדרת הפרויקט.
- בשאלות ניתן לפנות ל专家组 הדיוון באתר הקורס, ואל כל אחד מהמנחים בשעות הקבלה שלהם.**
- lezericom, באפשרותו של כל סטודנט לפנות לכל מנהה הקבועה שלו, לקבלת עזרה. שוב מומלץ לכל אלה שטרם בדקו את התכנים באתר הקורס לעשות זאת. נשאלות באתר זה הרבה שאלות בנושא חומר הלימוד והממן'ים, והתשובות יכולות להועיל לכלום.
- لتשומתיכם : לא ניתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילאים או אשפוז. במקרים אלו יש לבקש ולקבל אישור **מראש מנהה הקבועה**.

ב ה צ ל ח ה !