

מטלת מנהה (ממ"ז) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלת : פרויקט גמר

משקל המטלת : 61 נקודות (חובה)

מספר השאלות : 1

מועד אחרון להגשה : 11.08.2023

סמסטר : 2023 ב

קיימות שתי חלופות להגשת מטלות:

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

הסבר מפורט ב"נוהל הגשת מטלות מנהה"

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

עליכם לכנות תוכנת אסמבילר, עבור שפת אסמבלי שתוגדר בהמשך. הפרויקט ייכתב בשפת C.

עליכם להגיש את הפריטים הבאים :

1. קבצי המקור של התוכנית שכתבתם (קבצים בעלי הסיומת .c או .h).
- 2.קובץ הרצה (מוקומפל ומקשור) עבור מערכת אובונטו.
- 3.קובץ makefile. הקימפול חייב להיות עם הקומפיילר gcc והדלים : -Wall -ansi -pedantic . יש לנפות את כל ההודעות שモוציא הקומפיילר, כך שהתוכנית תתקמפל ללא כל העוראות או זהירות.
4. דוגמאות הרצה (קלט ופלט) :
 - א. קובצי קלט בשפת אסמבלי, וקבצי הפלט שנוצרו מהפעלת האסמבילר על קבצי קלט אלה. יש להציג שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האסמבלי.
 - ב. קובצי קלט בשפת אסמבלי המציגים מגוון רחב של סוגי שגיאות אסמבלי (ולכן לא נוצרים קבצי פלט), ותדפיסי המסקן המראים את הودעות שהשגיאה שמוציא האסמבילר.

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

זכור מספר היבטים חשובים של כתיבת קוד טוב :

1. הפשטה של מבני הנתונים : רצוי (כל האפשר) להפריד בין הגישה למבנה הנתונים לבני המימוש של מבני הנתונים. כך, למשל, בעת כתיבת פונקציות לטיפול בטבלה, אין זה מעניינים של משתמשים בפונקציות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקוורת.
2. קריאות הקוד : יש להשתמש בשמות משמעותיים למשתנים ופונקציות. יש לעורוך את הקוד באופן מסודר : הזחות עקביות, שורות ריקות להפרדה בין קטעי קוד, וכו'.
3. תיעוד : יש להכניס בקבצי המקור תיעוד תמציתי וברור, שיסביר את תפקידיה של כל פונקציה (באמצעות הערות כותרת לכל פונקציה). כמו כן יש להסביר את תפקידים של משתנים חשובים. כמו כן, יש להכניס הערות ברמת פירוט גבוהה בכל הקוד.

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

מותר 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 יוכל ציון.** חובה שסטודנטים, הבוחרים יהוו יחד את הפרויקט, יהיו **שייכים** לאותה **קבוצת הנחיה.** הציון יהיה זהה לשני הסטודנטים.

מומלץ לקרוא את הגדרת הפרויקט עם ראשונה בראצ'ף, לקבלת תמונה כללית לגבי הנדרש, ורק לאחר מכן לקרוא שוב בצורה מעמיקה יותר.

רקע כללי ומטרת הפרויקט

כידוע, קיימות שפות תוכנות רבות, ומספר גדול של תוכניות, הכתובות בשפות שונות, שעשוות לרוץ באותו מחשב עצמו. כיצד "מכיר" המחשב כל כך הרבה שפה? התשובה פשוטה: המחשב מכיר למעשה שפה אחת בלבד: הוראות ונתוניים הכתובים בקוד ביינרי. קוד זה מואחסן בזיכרון, ונראה כמו רצף של טפרות ביןאיות. יחידת העיבוד המרכזי - היע"מ (CPU) - יודעת לפרק את הרץ הזה לקטעים קטעניים בעלי משמעות: הוראות, מענים ונתוניים.

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

יחידת העיבוד המרכזי (היע"מ) יכולה לבצע מגוון פעולות פשוטות, הנקראות **הוראות מכונה**, ולשם כך היא משתמש ברגיסטרים (registers - אוגרים) הקיימים בתוך היע"מ, ובזיכרון המחשב.

דוגמאות: העברת מספר מתא בזיכרון לרגיסטר בייע"מ או בחזרה, הוספה 1 למספר הנמצא ברגיסטר, בדיקה האם המספר המואחסן ברגיסטר שווה לאפס, חיבור וחישור בין שני רגיסטרים, וכו'.

ההוראות המכונה ושילוביהם שלهنן הן המרכיבות תוכנית כפי שהיא טעונה לזכרון בזמן ריצתה. כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנן), תתרגם בסופו של דבר באמצעות תוכנה מיוחדת לזרה סופית זו.

היע"מ יודע לבצע קוד שנמצא בפורמט של **שפת מכונה**. זהו רצף של ביטים, המהוויםקידוד בינארי של סדרת הוראות המכונה המרכיבות את התוכנית. קוד זה אינו קרייא למשתמש, ולכן לא נוח לקובד (או לקרוא) תוכניות ישירות בשפת מכונה. **שפת אסמבלי** (assembly language) היא שפת תוכנות מאפשרת לייצג את הוראות המכונה בזרה סימבולית קליה ונוחה יותר לשימוש. כМОון שיש צורך לתרגם את הייצוג הסימבולי לקוד בשפת מכונה, כדי שהתוכנית תוכל לרוץ במחשב. תרגום זה נעשה באמצעות כלישן **אסambilר** (assembler).

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

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

תפקידו של האסambilר הוא לבנות קובץ המכיל קוד מכונה, מקובץ נתון של תוכנית הכתובת בשפת אסambilר. זהו השלב הראשון במסלול אותו עברת התוכנית, עד לקבלת קוד המוכן לריצה על חומרת המחשב. השלבים הבאים הם קישור (linkage) וטעינה (loading), אך בהם לא נעסק במאין זה.

השימוש בפרויקט זה היא כתוב אסambilר (כלומר תוכנית המתרגם לשפת מכונה), עבר שפת אסambilרי שנדיר כאן במיוחד לצורך הפרויקט.

לתשומתך: בהסבירים הכלליים על אופן עבודת תוכנת האסטמבלר, תהיה מדי פעם התייחסות גם לעובדות שלבי הקישור והטעינה. התייחסויות אלה נועדו על מנת לאפשר לכם להבין את המשך תהליכי העיבוד של הפלט של תוכנת האסטמבלר. אין לטעות: עליכם לכתוב את תוכנית האסטמבלר בלבד. אין לכתוב את תוכניות הקישור והטעינה!!!

המחשב הדמיוני ושפת האסטמבלי

נגיד לך את שפת האסטמבלי ואת מודל המחשב הדמיוני, עבור 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 מרכזית), אוגרים (רגיסטרים) וזיכרון RAM. חלק מהזיכרון משמש גם כמחסנית (stack). גודלה של מילת זיכרון במחשב הוא 12 סיביות.

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

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

גודל הזיכרון הוא 1024 תאים, בכתביות 1023-0 (בסיס עשרוני), וכל תא הוא בגודל של 12 סיביות. לתא בזכרון נקרא גם בשם "מילה". הכתובת בכל מילה ממוספרות כמו ברגיסטר.

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

מבנה הוראות המכונה:

כל הוראה מכונה מקודדת למספר מילוט 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
מיעון אופrnd יעד	opcode	מיעון אופrnd יעד		2	4	5	8

סיביות 8-5 במילה הראשונה של הוראה מהוות את קוד הפעולה (opcode). כל opcode מיוצג בשפת אסטמבלי על ידי שם פעללה. 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 המשמעות של הפעולות יבוא בהמשך.

סיביות 1-0 (A,R,E)

סיביות אלה מציניות 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 נתוניים), והן מתווספות גם לכל המילימנוזיפות שיש לקידודים אלה.

ראו הסבר נוספת על סיביות אלה בהמשך.

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

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

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

השימוש בשיטות מייעון מצריך קידוד של מילוט-מידע נוספת בקוד המכונה של ההוראה.
אם בפקודה יש אופרנד אחד, תהיה מילת מידע אחת נוספת. אם בפקודה יש שני אופרנדים,
יתכנו שתי מילות-מידע נוספות, או מילת-מידע אחת המשותפת לשני האופרנדים, תלוי בשיטות
מייעון 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 1-0 הן השדה E,R,A.

להלן תיאור של שיטות המיעון במכונה שלנו:

קוד	שיטת מיון	תוכן המילים הנוספות	אופן הכתיבה	דוגמה
1	מיון מידי	מילת-מיען נוספת של ההוראה מכילה את האופrnd עצמו, שהוא מספר המיצג ב- 10 סיביות, אליו מתווספות זוג סיביות של השדה .A,R,E	האופrnd הוא מספר שלם בסיס עשרוני	mov -103,@r2 בדוגמה זו האופrnd הראשון של הפקודה נתון בשיטת מיון מידי. ההוראה כתובה את הערך -103 - אל אוגר r2
3	מיון ישיר	מילת-מיען נוספת של ההוראה מכילה כתובות של מילה בזיכרונו. מילה זו בזיכרונו היא האופrnd. המعن מיוצג ב- 10 סיביות אליו מתווספות זוג סיביות של השדה .A,R,E	האופrnd הוא <u>תוויות</u> שכבר הזכרה או שתווצרה בהמשך הקובץ. ההצעה נעשית על ידי כתיבת תוויות בתחלת הנקראת 'data'. או 'string', או בתחילת הוראה של התוכנית, או באמצעות אופrnd של הנקראת 'extern'.	נתונה למשל ההגדרה: x: .data 23 אפשר לכתוב הוראה: dec x בדוגמה זו, ההוראה מקטינה ב-1 את תוכן המילה שבכמתובת x בזיכרונו (ה"משתנה" x).
5	מיון אוגר ישיר	האופrnd הוא אוגר. אם האוגר משמש כאופrnd יעד, מילת-מיען נוספת של הפקודה תקודד בחמש הסיביות 2-6 את מספרו של האוגר. ואילו אם האוגר משמש כאופrnd מקור, מספר האוגר יקודד בחמש הסיביות 7-11 של מילת-הميدע. אם בפקודה יש שני אופrndים, 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 יכלו 0.	האופrnd הוא שם של אוגר. שם האוגר יכתב כאשר לפני ופניו התו @	mov @r1,@r2 בדוגמה זו, ההוראה מעתקה את תוכן אוגר 1 אל אוגר r2. בדוגמה זו שני האופrndים הם אוגרים, אשר יקודדו למילת-מידע נספחת את תוכן אחד-

הערה: מותר להתייחס לתוויות עוד לפני שימושם עלייה, בתנאי שהיא אכן מוחרת במקום כלשהו בקובץ.

מפורט הוראות המכונה:

בתיאור הוראות המכונה השתמש במונח PC (קיצור של "Program Counter".) זהו רגיסטר פנימי של המעבד (לא רגיסטר כללי), שמקיל בכל רגע נתון את כתובות הזיכרונו בה. נמצאת ההוראה **הנוכחית שמתבצעת** (הכוונה תמיד לכתובות המילה הראשונה של ההוראה).

הוראות המכונה מתחולקות לשושן קבוצות, לפי מספר האופrndים הנדרשים לפעולה.

קבוצת ההוראות הראשונה:
אלן ההוראות המקבלות שני אופrndים.

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

הסבר הדוגמה	דוגמה	הפעולה המתבצעת	הוראה
העתק את תוכן המשתנה A (המילה שככטובה בזיכרון) אל רגיסטר r1.	mov A,@r1	מבצעת העתקה של האופrnd הראשון, אופrnd המקור (source) אל האופrnd השני (destination) אופrnd היעד (destination). בהתאם לשיטת המיעון.)	mov
אם תוכן המשתנה A זהה לתוכנו של רגיסטר r1 אז דגל האפס, Z, ברגיסטר הסטטוס (PSW) יודלק, אחרת הדגל יאפס.	cmp A, @r1	מבצעת "השוואה" בין שני האופrndים שלה. אופן ההשוואה : תוכן אופrnd היעד (השני) מופחת מתוכן אופrnd המקור (הראשון), ללא שמירה תוצאה החישור. פעולה החישור מעדכנת דגל בשם Z ("דגל האפס") ברגיסטר (PSW).	cmp
רגיסטר r0 מקבל את תוצאה החיבור של תוכן המשתנה A ותוכנו הנוכחי של רגיסטר r0.	add A,@r0	אופrnd היעד (השני) מקבל את תוצאה החיבור של אופrnd המקור (הראשון) והיעד (השני).	add
רגיסטר r1 מקבל את תוצאה החישור של הערך 3 מתוכנו הנוכחי של הרגיסטר r1.	sub 3,@r1	אופrnd היעד (השני) מקבל את תוצאה החישור של אופrnd המקור (הראשון) מאופrnd היעד (השני)	sub
המען שימושה התוויות HELLO מוצב לרגיסטר r1	lea HELLO, @r1	lea הוא קיצור (ראשי תיבות) של : load effective address. פעולה זו מצייבה את המعن בזיכרון המ מיוצג על ידי התווית שבאופrnd הראשון (המקור), אל אופrnd היעד (האופrnd השני).	lea

קבוצת ההוראות השנייה:

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

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

הסבר דוגמה	דוגמה	הפעולה המתבצעת	הוראה
r2 \leftarrow not r2	not @r2	היפוך ערכי הסיביות באופrnd (כל סיבית שערךה 0 תהפוך ל-1 וליהיפך : 1 ל-0).	not
r2 \leftarrow 0	clr @r2	אייפוס תוכן האופrnd.	clr
r2 \leftarrow r2 + 1	inc @r2	הגדלת תוכן האופrnd באחד.	inc
C \leftarrow C - 1	dec C	הקטנת תוכן האופrnd באחד.	dec

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

קובוצת ההוראות השלישייה:

אלו הן ההוראות ללא אופרנדים. קידוד ההוראה מורכב ממיליה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד במליה הראושונה של ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השייכות לקבוצה זו הן : rts, stop

הסבר דוגמה	דוגמה	הפעולה המתבצעת	הוראה
PC \leftarrow pop()	rts	חרזה משיגרה. הערך בראש המחסנית של זמן-ריצча מוצא מן המחסנית ומוכנס אל מצביע הוכנית (PC).	rts
הוכנית עצרת	stop	עצירת ריצת הוכנית.	stop

מבנה תוכנית בשפת אסמבלי :

תוכנית בשפת אסמבלי בנויה **ממקראים ומשמעותים** (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 המקאו לאוטו.

הגדרת מקאו נעשית באופן הבא: (בדוגמה שם המקאו 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 ותו, התוכנית לאחר פרישת המקאו תיראה כך:

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

משפטים:

קובץ מקור במשפט אסמבייל מורכב משורות המכילות משפטיים של השפה, כאשר כל משפט מופיע בשורה נפרדת. לעומת זאת, ההפרדה בין משפטיים בקובץ המקורי אינה באמצעות התו 'ז' (שורה חדשה).

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

יש ארבעה סוגי משפטיים (שורות בקובץ המקורי) במשפט אסמבייל, והם :

סוג המשפט	הסבר כללי
משפט ריק	זו היא שורה המכילה אך ורק תווים לבנים (whitespace), כלומר מכילה רק את התווים 'ז' ו-' ' (טבבים ורווחים). ייתכן ובשורה אין אף תווים (למעט התו 'ז), כלומר השורה ריקה.
משפט הערתה	זו היא שורה בה התו הראשון הוא ';' (נקודה פסיק). על האסמבול להתעלם מהלוטין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמבול מה עליו לעשות כשהוא פועל על תכנית המקורי. יש מספר סוגים של משפטיי הנחיה. משפט הנחיה עשוי לגרום להקצת זיכרון ואתחול משתנים של התוכנית, אך הוא אינו מייצר קידוד של הוראות מכונה המיועדות לביצוע בעת ריצת התוכנית.
משפט הוראה	זהו משפט המייצר קידוד של הוראות מכונה לביצוע בעת ריצת התוכנית. המשפט מורכב ממשרתו הוראה (פעולה) שעל המעבד לבצע, והאופרנדים של הוראה.

cut נפרט לגבי סוגי המשפטיים השונים.

משפט הנחיה:

משפט הנחיה הוא בעל המבנה הבא :

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

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

יש לשים לב: ל밀ות הקוד הנוצרות ממשפט הנחיה לא מצורפות זוג סיביות E,R,A וקידוד מלא את כל 12 הסיביות של המילה.

יש ארבעה סוגי משפטיי הנחיה , והם :

1. ההנחיה 'data'.

פרמטרים 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 לבין מספר יכולים להופיע רווחים וטבבים בכלל כתמות (או בכלל לא), אולם הפסיק חייב להופיע בין

המספרים. כמו כן, אסור שיופיע יותר ממספר אחד בין שני מספרים, וגם לא פסיק אחרி המספר الآخرון או לפני המספר הראשון.

המשפט 'data', מנהה את האסמלר להקצות מקום בתמונה הנתונים (data image), אשר בו יוחסנו הערךם של הפרמטרים, וקדם לモונה הנתונים, בהתאם למספר הערךם. אם בלחיצת 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 דרכ לגדיר שם של משתנה).

כלומר אם נכתב :

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

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

אם נכתב בתכנית את ההוראה :

mov XYZ, @r1

אז בזמן ריצת התכנית יוכנס לרגיסטר 1ז הערך 7.

ויאלו ההוראה :

lea XYZ,@r1

תכנית לרגיסטר 1ז את ערך התווית XYZ (כלומר הכתובת בזיכרונו מהוחсан הערך 7).

.2. ההנחיה '.string'

להנחיה '.string' פרמטר אחד, שהוא מחרוזת חוקית. תווית המחרוזת מקודדים לפי ערכי-ascii המתאים, ומוכנסים אל תמונה הנתונים לפי סדרם, כל TWO במילה נפרדת. בסוף המחרוזת יתווסף התן '0' (הערך המספרי 0), המסמן את סוף המחרוזת. מונה הנתונים של האסמלר יקודם בהתאם לאורך המחרוזת (בतוספת מקום אחד עבור התו המסייע). 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 שבה מתחילה המחרוזת).

לדוגמה, ההנחיה :

STR: .string "abcdef"

מקצת בתמונה הנתונים רצף של 7 מילימ, ומאתחלת את המילימ לקובץ ascii של התווים לפי הסדר במחרוזת, ולאחריהם הערך 0 לסימון סוף מחרוזת. התווית STR מזוהה עם כתובות התחלת המחרוזת.

.3. ההנחיה '.entry'

להנחיה '.entry' פרמטר אחד והוא שם של תווית המוגדרת בקובץ המקור הנוכחי (כלומר תווית שמקבלת את ערכה בקובץ זה). מטרת ההנחיה entry. היא לאפיין את התווית הזה באופן שיאפשר לקוד אסמלרי הנמצא בקובץ מקור אחרים לשמש בה (כאופrnd של הוראה).

לדוגמה, השורות :

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

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

لتשומת לב:תוויות המוגדרת בתחילת שורת `entry`. הינה חסרת משמעות והאסמלר **מתעלם** מתוויות זו (אפשר שהאסמלר יוציא הודעה אחרת).

4. הנקיה 'extern'.

הנקיה 'extern' פרמטר אחד, והוא שם של תוויות שאינה מוגדרת בקובץ המקור הנוכחי. מטרת ההוראה היא להודיע לאסמלר כי התוויות מוגדרת בקובץ מקור אחר, וכי קוד האסמלר בקובץ הנוכחי עושה בתוויות שימוש.

נשים לב כי הנקיה זו תואמת להנקיה 'entry'. המופיעה בקובץ בו מוגדרת התוויות. בשלב הקישור תבוצע התאמה בין ערך התוויות, כפי שנקבע בקוד המכונה של הקובץ שהגדיר את התוויות, לבין קידוד ההוראות המשתמשות בתוויות בקבצים אחרים (שלב הקישור אינו רלוונטי למים זה).

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

```
.extern HELLO
```

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

لتשומת לב:תוויות המוגדרת בתחילת שורת `extern`. הינה חסרת משמעות והאסמלר **מתעלם** מתוויות זו (אפשר שהאסמלר יוציא הודעה אחרת).

משפט הוראה :

משפט הוראה מורכב מחלקים הבאים :

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

אם מוגדרת תוויות בשורת ההוראה, אז היא תוכנס אל טבלת הסמלים. ערך התוויות יהיה מען המילה הראשונה של ההוראה בתוך תMONTH הקוד שבודנה האסמלר.

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 ט-abs (אחד או יותר).

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזה בתו ',' (פסיק). בדומה להנקיה 'data', **לא חייבת להיות הצמדה של האופרנדים לפסיק**. כל כמה 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

תווית :

תווית היא סמל שמודגר בתחילת המשפט הוראה, או בתחילת הначיה .string. או 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 מסתויימת בתו ': (נקודותים).תו זה אינו מהו חילק מהתווית, 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 החלטה, או שם של רегистר) אין יכולות לשמש גם שם של תווית.

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

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

מספר :

מספר חוקי מתחילה בסימן אופציוני: '-' או '+' ולאחריו סדרה של ספרות בסיס עשרוני. לדוגמה : +123, -5, 76 ועוד מספרים חוקיים. אין תמיכה בשפת האסמבלי שלו ביצוג בסיס אחר מאשר עשרוני, ואין תמיכה במספרים שאינם שלמים.

מחuzeות:

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

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

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

שתי הסיבות בשדה E יכilio את אחד הערכים הבינאיים: 00, 10, או 01. המשמעות של כל ערך מפורטת להלן.

האות 'A' (קייזר של absolute) בא להציג שתוכן המילה אינו תלוי במקום בו זיכרו בו יייען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל מילה המכילה אופrnd מיידי).
במקרה זה שתי הסיבות הימניות יכilio את הערך 00.

האות 'R' (קייזר של relocatable) בא להציג שתוכן המילה תלוי במקומות בו זיכרו בו יייען בפועל קוד המכונה של התוכנית בעת ביצועה (למשל מילה המכילה כתובת של תוכית המוגדרת בקובץ המקור). במקרה זה שתי הסיבות הימניות יכilio את הערך 10.

האות 'E' (קייזר של external) בא להציג שתוכן המילה תלוי בערכו של סמל חיצוני (external) (למשל מילה המכילה כתובת של תוכית חיצונית, הכולמת תוכית שאינה מוגדרת בקובץ המקור).
במקרה זה שתי הסיבות הימניות יכilio את הערך 01.

כאשר האסטמבלר מקבל כקלט תוכנית בשפת אסטמבלר, עליו לטפל תחילה בפרישת המאקרוואים, ורק לאחר מכן לעבור על התוכנית אליה נפרשו המקרוואים. כמובן, פרישת המקרוואים תעשה בשלב "קדם אסטמבלר", לפני שלב האסטמבלר (המתוар בהמשך).
אם התוכנית אינה מכילה מקרו, תוכנית הפרישה תהיה זהה לתוכנית המקור.

דוגמה לשלב קדם אסטמבלר. האסטמבלר מקבל את התוכנית הבאה בשפת אסטמבלר :

```

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

```

תחילה האסמבלי עבר על התוכנית ופורש את כל המאקרוואים הקיימים בה. רק אם תהליך זה מסתיים בהצלחה, ניתן לעبور 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 קוד האסמבלי

נציג להלן אלגוריתם שלדי לenthalיך קוד האסמבלי. لتשומת לב: אין חובה להשתמש דווקא באלגוריתם זה:

1. קרא את השורה הבאה מקובץ המקורי. 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, חוזר ל- 1. אחרת, המשך.
3. האם השדה הראשון הוא **"mero"** (התחלת הגדרת מאקרו)? אם לא, עבור ל- 6.
4. הדלק דגל **"יש mero"**.
5. (קיימת הגדרת מאקרו) הכנס לטבלת שורות מאקרו את שם המאקרו (לדוגמא m1).
6. קרא את השורה הבאה מקובץ המקורי. אם נגמר קובץ המקורי, עבור ל- 9 (סיום).
7. אם דגל **"יש mero"** דולק ולא זההתו תווית **endmero** הכנס את השורה לטבלת המאקרו ומחק את השורה הניל מהקובץ. אחרת (לא מאקרו) חוזר ל- 1.
8. האם זההתו תווית **endmero**? אם כן, מחק את התוויות מהקובץ והמשך. אם לא, חוזר ל- 6.
9. סיום : שמירת קובץ מקטו פוגש.

אסמבלי עם שני מעברים

במעבר הראשון של האסמבלי, יש להזות את הסמלים (תוויות) המופיעים בתוכנית, ולתת לכל סמל ערך מסווני שהוא בזיכרונו שהסמל מייצג. במעבר השני, באמצעות ערכי הסמלים, וכן קודי-הפעולה ומספריו האוגרים, בונים את קוד המוכנה.

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

כמו כן, על האסמבלי 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) ייטען בזיכרון החל ממען 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 000111110110
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 111111110111 000000000111
0126			
0127			
0128	K: .data 22	integer 22	000000010110

האסמבלר מוחזק טבלה שבה רשומים כל שמות הפעולה של החראות 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 המירה לבינארי של אופרנדים שכחובים בשיטות מייען המשמשות בסמלים (תוויות), יש ליצור לבנות טבלה המכילה את ערכי כל הסמלים. אולם בהבדל מהקודים של הפעולות, הידעעים מראש, הרי המענינים בזיכרונו עברו הסמלים שבשימוש התוכנית אינם ידועים, עד אשר תוכניתה המקור נסקרה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד לעיל, האסמבלר איינו יכול לדעת שהסמל END משוויך לערך 117 (עשרוני), אלא רק לאחר שנקרואו כל שורות התוכנית.

לכן מפרידים את הטיפול של האסמבלר בסמלים לשני שלבים. בשלב הראשון בונים טבלה של כל הסמלים, עם הערכים המספריים המשווים להם, ובשלב השני מחליפים את כל הסמלים, המופיעים באופרנדים של החראות 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 משוויך ערך מסווני 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 (בבסיס 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.

לתשומת לב: תפקיד האSEMBLER, על שני המעברים שלו, לתרגם קובץ מקור לקוד בשפת מכונה. בגין פועלות האSEMBLER, התוכנית טרם מוכנה לטעינה ל זיכרון לצורך ביצוע. קוד המכונה חייב לעبور לשלב הקיישור/טעינה, ורק לאחר מכן לשלב הביצוע (שלבים אלה אינם חלק מהממשק).

המעבר הראשון

במעבר הראשון נדרשים כללים כדי לקבוע איזה מען ישוק לכל סמל. העיקרונו הבסיסי הוא לספור את המיקומות בזיכרון, אותן תופסות הHoraeot. אם כל הוראה תיתען בזיכרון במקום העוקב להוראה הקודמת, תציג ספירה כזאת את מען ההוראה הבאה. הספירה נועשית על ידי האSEMBLER ומוחזקת במוניה הHoraeot (IC). ערכו ההתחלתי של IC הוא 0, ולכן נוענת ההוראה הראשונה במען 0. IC מתחדש בכל שורת הוראה המקצת מקום בזיכרון. לאחר שהאSEMBLER קובע מהו אורך ההוראה, IC מוגדל במספר התאים (밀ימטרים) הנתפסים על ידי ההוראה, וכך הוא מצביע על התא הפנוי הבא.

כאמור, כדי לקודד את הHoraeot בשפת מכונה, מחייב האSEMBLER טבלה, שיש בה קידוד מתאים לכל שם פעולה. בזמן התרגום מחליף האSEMBLER כל שם פעולה בקידודו שלה. כמו כן, כל אופרנד מוחלף בקידוד מתאים, אך פעולות החלפה זו אינה כה פשוטה. הHoraeot משתמשות בשיטות מען מגוונות לאופרנדים. אחת פעולה יכולה לקבל שימושיות שונות, בכל אחת משיטות המיעון, וכן יכולות לה קידודים שונים לפי שיטות המיעון. לדוגמה, פעולה ההזזה湫ו יכולה להתיחס להעתקת תוכן תא זיכרון לרגיסטר, או להעתקת תוכן רגיסטר לרגיסטר אחר, וכן הלאה. ככל אפשרות כזו של湫ו עשוי להתאים קידוד שונה.

על האSEMBLER 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 מילוה אחת או יותר בקוד המכונה.

כאשר נתקל האSEMBLER בתווית המופיעעה בתחילת השורה, הוא יודע שלפנוי הגדרה של תווית, ואז הוא מחייב לה מען – תוכנו הנוכחי של IC. כך מקבלות כל התוויות את מענייןן בעת ההגדירה. תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנוסף לשם התווית גם את המعن ומאפיינים נוספים. כאשר תהיה התיחסות לתווית באופרנד של הוראה כלשהי, יוכל האSEMBLER לשולף את המعن המתאים מטבלת הסמלים.

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

bne A

.

.

.

A:

כאשר מגיע האSEMBLER לשורת ההסתעפות (A bne), הוא טרם נתקל בהגדרת התווית A וכמוון לא יודע את המعن המשויך לתווית. לכן האSEMBLER לא יכול לבנות את הקידוד הבינארי 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-המיעד נוספת מיידי, או רגיסטר, וכן את הקידוד הבינארי של כל הנתונים (המתקבלים מההנחיות `.data`, `.string`).

המעבר השני

ראינו שבמעבר הראשון, האסטבלר אינו יכול לבנות את קוד המכונה של אופרנדים המשמשים בסמלים שעדין לא הוגדרו. רק לאחר שהאסטבלר עבר על כל התוכנית, כך שכל הסמלים נכנסו כבר לטבלת הסמלים, יוכל 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 המשמשים בסמלים, באמצעות ערכי הסמלים מטבלת הסמלים. בסוף המעבר השני, תהיה התוכנית מתרגמת בשמותה לקוד מכונה.

הפרדת הוראות ונתונים

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

אחד הסכנות הטමונות באירוע ההוראות מהנתונים היא, לעיתים קרובות, בעקבות שגיאה לוגית בתוכנית, לנסתו "לבצע" את הנתונים כאילו הם הוראות חוקיות. למשל, שגיאה שיכולה לגרום תופעה כזו היא הסתעפות לא נכונה. התוכנית כפונה לא תעבור נכוון, אך לרוב הנזק הוא יותר חמור, כי נוצרת חריגת חומרה ברגע שהמעבד מבצע פולוה שאינה חוקית.

האסטבלר שלו חייב להפריד, בקוד המכונה שהוא מייצר, בין קטע הנתונים לקטע ההוראות. כלומר בקובץ הפלט (בקוד המכונה) תהיה הפרדה של הוראות ונתונים לשני קטעים נפרדים, ואם בקובץ הקלט אין חובה שתהייה הפרדה כזו. בהמשך מואר אלגוריתם של האסטבלר, ובו פרטים כיצד לבצע את הפרדה.

גילוי שגיאות בתכנית המקור

כפי שהוסבר לעיל, הנחת המטלה היא **שאין שגיאות בהגדירות המקור**, ולכן בשלב קדם האסטבלר אינו מכיל שלב גילוי שגיאות, לעומת זאת האסטבלר אמר לו לגנות ולדוח על **שגיאות בתחריר של תוכנית המקור**, כגון פעולה שאינה קיימת, מספר אופרנדים שגוי, סוג אופרנד שאינו מתאים לפעולה, שם אונגר לא קיים, ועוד שגיאות אחרות. כמו כן מודיא האסטבלר שככל סמל מוגדר פעמיים אחד בבדיקה.

מכאן, שככל שגיאה המתגלגה על ידי האסטבלר נגרמת (בדרך כלל) על ידי שורת קלט מסויימת. לדוגמה, אם מופיעים שני אופרנדים בהוראה שאמור להיות בה רק אופרנד אחד, האסטבלר ייתן הودעת שגיאה בנוסח **"יוטר מדי אופרנדים"**.

הערה: אם יש שגיאה בקוד האסטבלרי בגוף מקשו, הרוי שגיאה זו יכולה להופיע ולהתגלות שוב ושוב, בכל מקום בו נפרש המקארו. נשים לב שכאר האסטבלר בודק שגיאות, כבר לא ניתן להזחות שזה קוד שנפרש ממאקו, כך שלא ניתן לחסוך גילוי שגיאה כפולים.

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

لتשומת לב: האסטבלר אינו עוצר את פעולתו אחרי שנמצאה השגיאה הראשונה, אלא ממשיך לעبور על הקלט כדי לגנות שגיאות נוספות, ככל שישן. כמובן שאין כל טעם לייצר את קבצי הפלט אם נתגלו שגיאות (משמעותה היא לא ניתן 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 מייען חוקיות עבור אופרנד המקור	שיטות מייען 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	3,5
clr	אין אופרנד מקור	3,5
lea	3	3,5
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 תהליכי העבודה של האסטמבלר, נציג להלן אלגוריתם שלדי למעבר הראשון ולמעבר השני. לתשומתلبך: אין חובה להשתמש דווקא באלגוריתם זה.

. אנו מחלקים את קוד המכונה לשני חלקים: **תMOVINT ההוראות (code) ותMOVINT הנתונים (data)** (Instruction-Counter - IC (MOVINT ההוראות - DC (MOVINT הנתונים - DC-Counter - DC).

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. **ATCHL 0, IC 0.**
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עברו ל-16.
3. האם השדה הראשון הוא סמל? אם לא, עברו ל-5.
4. הדלק דגל "יש גדרת סמל".
5. האם זהה הנחיה לאחסון נתונים, כלומר, האם הנחיה `data`. או `string`? אם לא, עברו ל-8.
6. אם יש גדרת סמל (תוויות), הכנס אותו לטבלת הסמלים עם סימונו (סמל מסוג `data`-).
7. ערכו ייה DC. (אם הסמל כבר נמצא בטבלה, יש להודיע על שגיאה).
8. זהה את סוג הנתונים, קודו אותם בזיכרון, עדכן את מוניה הנתונים DC בהתאם לאורכם, חוזר ל-2.
9. האם זו הנחיה `extern`. או הנחיה `entry`? אם לא, עברו ל-11.
10. האם זו הנחיה `extern`? אם כן, הכנס כל סמל (אחד או יותר) המופיע כאופרנד של `extern`.
11. החניה לתוך טבלת הסמלים ללא ערך, עם סימונו (סמל מסוג `external`).
12. חוזר 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 – הודיע על שגיאת שם הוראה.
13. נתח את מבנה האופרנדים של ההוראה וחשב את L. בנה CUT את הקוד הבינארי של המילה הראשונה של הפוקודה.
14. עדכן $IC \leftarrow L + IC$
15. חזר 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 (ראו הסבר בהמשך).
18. התחל מעבר שני.

מעבר שני

1. 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 הначיה ?.data,.string,.extern?. אם כן, חזר ל-2.
5. האם זהה הначיה ?.entry?. אם לא, עברו ל-7.
6. סמן בטבלת הסמלים את הסמלים החל מהמילה השנייה בקוד הבינארי של ההוראה, בהתאם להלן את קידוד האופרנדים הצל מהמילה השנייה בקוד הבינארי של ההוראה, בהתאם לשיטת המיעון. אם אופרנד הוא סמל, מצא את המعن בטבלת הסמלים.
7. עדכן $IC \leftarrow IC + L$
8. חזר ל-2.
9. אם נמצאו שגיאות במעבר שני, עצור.
10. צור ושמור את קבצי הפלט: קובץ קוד המוכנה קובץ סמלים חיוניים, וקובץ סמלים של נקודות כניסה (ראו פרטיהם נוטפים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו לעיל **(לאחר שלב פרישת המאקרוואים)**, ונציג את הקוד הבינארי שמתקיים 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. בניית טבלת הסמלים. כמו כן, נבצע במעבר זה גם את קידוד כל הנתונים, וקידוד המילה הראשונה של כל הוראה. את החלקים שעדיין לא מתרגמים במעבר זה, נשאיר כמוות שהם.

אנו מניחים שהקוד ייתען לזכרון החל מהמען 100 (בסיס דצימלי).

Decimal Address	Source Code	Explanation	Binary Machine Code
0100 0101 0102	MAIN: mov @r3 ,LENGTH	first word of instruction source register 3 address of label LENGTH	101000001100 000110000000 ?
0103 0104	LOOP: jmp L1	address of label L1	000100101100 ?
0105 0106	prn -5	immediate value -5	000110000100 111111101100
0107 0108	bne LOOP	address of label LOOP	000101001100 ?
0109 0110	sub @r1, @r4	registers r1 and r4	101001110100 000010010000
0111 0112	bne END	address of label END	000101001100 ?
0113 0114	L1: inc K	address of label K	000011101100 ?
0115 0116	bne LOOP	address of label LOOP	000101001100 ?
0117	END: stop		000111100000
0118 0119 0120 0121 0122 0123 0124	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
0125 0126 0127	LENGTH: .data 6,-9,15	integer 6 integer -9 integer 15	0000000000110 11111110111 000000001111
0128	K: .data 22	integer 22	000000010110

טבלת הסמלים:

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

בצע עתה את המעבר השני ורשום את הקוד בצורתו הסופית:

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

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		0000000001111
0128	K: .data 22	000000010110

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

קבצי קלט ופלט של האסמבלי

בפעולת של האסמבלי, יש להעיר אליו באמצעות ארגומנטים של שורת הפקודה (command line arguments) רשימה של שמות קבצי מקור (אחד או יותר). אלו הם קבצי טקסט, ובهم תוכניות בתחביר של שפת האסמבלי שהוגדרה בມמ"ז זה.

האסמבלי פועל על כל קובץ מקור בנפרד, ויוצר עבورو קבצי פלט כדלקמן :

- קובץ `.am`, המכיל את קובץ המקור לאחר שלב קדם האסמבלי (לאחר פרישת המאקרויאים).
- קובץ `.object`, המכיל את קוד המוכנה.
- קובץ `.externals`, ובו פרטיהם על כל המKENOTOT (הכטובות) בקוד המוכנה בהם יש מילת-מידע Shmekoddot Urz של סמל שהוצהר כחיצוני (סמל שהופיע כאופרנד של הנחיתת `..extern`, ומואפי בטבלת הסמלים `-external`).
- קובץ `.entries`, ובו פרטיהם על כל סמל שימושה ננקודת כניסה (סמל שהופיע כאופרנד של הנחיתת `..entry`, ומואפי בטבלת הסמלים `-entry`).

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

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

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

assembler x y hello

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

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

לדוגמא, בהפעלת האסמלר באמצעות שורת הפקודה : x יוצר קובץ פلت .ob.a, וכן קבצי פلت .ext.x ו-.ent.x כל שיש הנחיות entry. או .extern. בקובץ המקור. אם אין מאקרו בקובץ המקור, אז קובץ ".am" יהיה זהה לקובץ ".as".

אופן פעולות האסמלר

נרחיב כאן על אופן פעולות האסמלר, בנוסף לאלגוריתם השידי שניתן לעיל.

האסמלר מחזיק שני מערכיים, שיקראו להן מערך ההוראות ומערך הנתונים. מערכיים אלו נתונים למשה תמונה של זיכרון המכונה (וגדל כל כניסה בזיכרון זהה לגודלה של מילת מכונה : 12 סיביות). במערך ההוראות מכניס האסמלר את הקידוד של הוראות המכונה שנקרוו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמלר את קידוד הנתונים שנקרוו מקובץ המקור (שורות מסוג .string.data).

לאסמלר יש שני מונחים : מונה ההוראות (IC) ומונה הנתונים (DC). מונחים אלו מצביעים על המקום הבא הפני במערכות לעיל, בהתאם. כמשמעותו של האסמלר עבור על קובץ מקור, שני מונחים אלו מואפסים.

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

האסמלר 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 : האסמלר מתעלם מהשורה וועבר לשורה הבאה.
2. שורת הוראה :

האסמלר מוצא מהי הפעולה, ומהן שיטות המיעון של האופrndים. (מספר האופrndים אותם הוא מחפש נקבע בהתאם להוראה אותה הוא מצא). האסמלר קובע לכל אופrnd את ערכו באופן הבא :

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

קביעת שיטת המייעון נעשית בהתאם לתchapir של האופrnd, כפי שהסביר לעיל בהגדרת שיטות המייעון. למשל, מספר מצין מיידי, תווית מצינית מייעון ישיר, שם של רגיסטר עם (@) לפניו מצין מייעון רגיסטר, וכו'').

לאחר שהאסטמבלר ניתח את השורה והחליט לגבי הפעולה, שיטת מיון אופrnd המקור (אם יש), ושיטת מיון אופrnd היעד (אם יש), הוא פועל באופן הבא :

אם זהה פעולה בעלת שני אופrndים, אז האסטמבלר מכניס לערך ההוראות, במקומות עליו מציבע מונה ההוראות IC, את קוד המילה הראשונה של הוראה (בשיטת הייצוג של הוראות המכונה כפי שתואר קודם לכן). מילה זו מכילה את קוד הפעולה, ואת שיטות המיון. בנוסף "משרין" האסטמבלר מקום בערך עבור המילים הנוספות הנדרשות עבור הוראה זו, ומגדיל את מונה ההוראות בהתאם. אם אחד או שני האופrndים הם בשיטת מיון רגיסטר או מיידי, האסטמבלר מקובד כתת המילים הנוספות הרלוונטיות בערך ההוראות.

אם זהה פעולה בעלת אופrnd אחד בלבד, כלומר אין אופrnd מוקור, אז הקידוד הינו זהה לעיל, פרט לנסיבות של שיטת המיון של אופrnd המקור במילה הראשונה, אשר יכילה תמיד 0, מכיוון שאין רלוונטיות לפעולה.

אם זהה פעולה ללא אופrndים (its, stop) אז תקודד רק המילה הראשונה (והיחידה). הסיבות של שיטות המיון של שני האופrndים יכלו 0.

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

3. שורת הנחיה :

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

I. '.data'.
האסטמבלר קורא את רשות המספרים, המופיעה לאחר '.data.', מכניס כל מספר אלערך הנתונים, ומגדם את מצביע הנתונים DC באחד עבור כל מספר שהוכנס.

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

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

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

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

III. '.entry'.
זהו בקשה לאסטמבלר להכניס את התווית המופיעה כאופrnd של '.entry.' אל קובץ ה-entries. האסטמבלר רושם את הבקשה ובסיום העבודה, התווית הניל תירשם בקובץ ה-entries.

IV. '.extern'.
זהי הציהרה על סמל (תווית) המוגדר בקובץ אחר, ואשר קטע האסטמבל בקובץ הנוכחי עושה בו שימוש. האסטמבלר מכניס את הסמל אל טבלת 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 האסטמבלר).

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

פורמט קובץ ה-object

האSEMBLER בונה את תMOVות זיכרונו המוגנה כך שKIJDOD הHorאה הראשונה מקובץ האSEMBLER יכנס למשך 100 (בבסיס 10) בזיכרונו, קידוד הHorאה השניה יכנס מען העקב אחרי הHorאה הראשונה (תלויה במספר המילים של הHorאה הראשונה), וכן הלאה עד להHorאה الأخيرة.

מיד לאחר קידוד הHorאה الأخيرة, מכניםים לתMOVות הזיכרונו את קידוד הנתונים שנבנו על ידי ההנחיות 'data'.string. הנתונים יוכנסו בסדר בו הם מופיעים בקובץ המקור. אופrnd של Horאה שמתיחס לSAMPLE שהוגדר באותו קובץ, יקודך כך שיציביע על המיקום המתאים בתMOVות הזיכרונו שבונה האSEMBLER.

נשים לב שהMOVות מופיעים בתMOVות הזיכרונו אחרי הHorאות. זהה הסיבה בגללה יש לעדכן בטבלת SAMPLEים, בסוף המעבר הראשוני, את ערכי SAMPLEים המגדירים נתונים (SAMPLEים מסווג).(data).

עקרונית, קובץ object מכיל את תMOVות הזיכרונו שתוארה כאן. קובץ object מורכב משורות של טקסט כדלקמן:
השורה הראשונה היא כוורת המכילה שני מספרים בסיס 10: האורך הכולל של קטע Horאות (SAMPLE זיכרונו) ואחריו האורך הכולל של SAMPLE הנתונים (SAMPLE זיכרונו). בין שני המספרים יש רווח אחד.

השורות הבאות מכילות את תוכן הזיכרונו. בכל שורה מופיע תוכן של מילה אחת, לפי הסדר החל מהמילה בכתובת 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 יומרו לטו מותאים לפי הטלחה המגדירה את Base64. לפיכך בכל שורה בקובץ object (מלבד השורה הראשונה) יהיו בדיקות שני תוויים בקוד Base64.

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

פורמט קובץ ה-entries

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

פורמט קובץ ה-externals

קובץ externals בנייתו אף הוא משורות טקסט. כל שורה מכילה שם של SAMPLE שהוגדר external, וכותבות בקוד המכונה בה יש קידוד של אופrnd המתיחס לSAMPLE זה. כМОון שיתכן ויש מספר כתובות בקוד המכונה בהם מתיחסים לאותו SAMPLE בלבד חיצוני. לכל התייחסות כזו תהיה שורה נפרדת בקובץ ה-externals. הכתובות מיוצגות בסיס 10.

לתשומתך: ניתן שיש מספר כתובות בקוד המכונה בהן SAMPLE-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 שהודם קודם.

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 שמתקיים מקובץ המקור, ולאחריה הפורמטים קבצי הפלט השונים.

טבלת הקידוד הבינארי:

Decimal Address	Source Code	Binary Machine Code
0100	MAIN: mov @r3 ,LENGTH	101000001100 000110000000 000111110110
0101		
0102		
0103	LOOP: jmp L1	000100101100 000111000110
0104		
0105		
0106	prn -5	000110000100 111111101100
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		000111100000
0118	STR: .string "abcdef"	000001100001 000001100010 000001100011 000001100100 000001100101 000001100110 0000000000000
0119		
0120		
0121		
0122		
0123		
0124		
0125	LENGTH: .data 6,-9,15	000000000110 111111110111 000000001111
0126		
0127		
0128	K: .data 22	000000010110

הקובץ ob.ps:

נבחן 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 הכוורת):

הערה: שורת הכוורת להלן אינה חלק מהקובץ, ונועדה להבהרה 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

לתשומת לב: אם בקובץ המקור אין הצחרת `extern`. אז לא ייווצר עבورو קובץ `.ext`. בדומה, אם אין בקובץ המקור הצחרת `entry`, לא ייווצר קובץ `.ent`. אין לייצור קובץ ext או ent שנשאר ריק.

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

סיכום והנחיות כלליות

- אורך התוכנית, הניתנת כקלט לאסטבלר אינו ידוע מראש, ולכן אורך התוכנית המתרגמת אינו אמור להיות צפוי מראש. אולם כדי להקל בימיוש האסטבלר, ניתן להניח גודל מסוימלי. לפיכך יש אפשרות לשימוש במערכים לאחסון תമונות קוד המכונה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים וטבלת המקרו), יש למשה באופן ייעיל וחסוני (למשל באמצעות רשימה מקושרת והקצתה זיכרון דינמי).
 - השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הקלט, למעט הסיומות. למשל, אם קובץ הקלט הוא as אז קבצי הפלט שייצרו הם : prog.ob, prog.ext, prog.ent
 - מתכונת הפעלת האסטבלר צריכה להיות כפי הנדרש בממ"ן, ללא שינוים כלשהם. ככלומר, ממשך המשתמש יהיה אך ורק באמצעות שורת הפוקודה. בפרט, שמורות קבצי המקור יועברו לתוכנית האסטבלר כארוגומנטים (אחד או יותר) בשורת הפוקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גרפיים למיניהם, ועוד'.
 - יש להקפיד לחלק את מימוש האסטבלר למספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסוימים שונים במודול יחיד. מומלץ לחלק למודולים כגון : מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחבירי של שורה), טבלת הסמלים, מפתח הזיכרון, טבלאות קבועות (קוד הפעולה, שיטות המיעון החוקיות לכל פעולה, ועוד').
 - יש להקפיד ולתעד את המימוש באופן מלא וברור, באמצעות הערות מפורטות בקוד.
 - יש לאפשר תווים לבנים עודפים בקובץ הקלט בשפת אסטבלר. למשל, אם בשורת הוראה יש שני אופרנדים המופרדים בפסיק, אזי לפני ואחרי הפסיק מותר شيء רוחחים וטאבים בכל כמות. בדומה, גם לפני ואחרי שם הפעולה. מותרות גם שורות ויקות. האסטבלר יתעלם מתיוים לבנים מיוחדים (כלומר יידלג עליהם).
 - הקלט (קוד האסטבלר) עלול להכיל שגיאות תחביריות. על האסטבלר לגלות ולדוח על כל השורות השגויות בקלט. אין לעצור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להדפיס לפחות הודעות מפורטות לכל הניגון, כדי שאפשר יהיה להבין מהו והיכן כל שגיאה. כמובן שאם קובץ קלט מכיל שגיאות, אין טעם להפיק עבورو את קבצי הפלט (ob, ext, ent).
- תס ונסלם פרק ההסבירים והגדרת הפרויקט.
- בשאלות ניתן לפנות לקבוצת הדיוון באתר הקורס, ועל כל אחד מהמנחים בשעות הקבלה שלהם.**
- לחזיכיכם, באפשרותו של כל סטודנט לפנות לכל מנהה, לאו דווקא למנהל הקבוצה שלו, לקבלת עזרה. שוב מומלץ לכל אלה שטרם בדקו את התכנים באתר הקורס לעשות זאת. נשאלות באתר זה הרבה שאלות בנושא חומר הלימוד והממ"נים, והתשובות יכולות להופיע בכלם.
- לתשומתיכם : לא ניתן דחיה בהגשת הממ"ן, פרט למקרים מיוחדים כגון מילואים או אשפוז. במקרים אלו יש לבקש ולקבל אישור **מראש** ממנהל הקבוצה.

ב הצלחה !