

מטלת מנהה (ממ"ו) 14

הקורס : 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטרת : פרויקט גמר

משקל המטלה : 61 נקודות (חובה)

מספר השאלות : 1

מועד אחרון להגשה : 14.3.2021

סמסטר : 2021/א'

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"

אחת המטרות העיקריות של הקורס "20465 - מעבדה בתכנות מערכות" היא לאפשר לסטודנטים בקורס להתנסות בכתיבת פרויקט תוכנה גדול, אשר יחבר את פעולתה של אחת ממערכות המערכות השכירות.

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

עליכם להגיש את הפריטים הבאים :

1. קבצי המקור של התוכנית שכתבתם (קבצים בעלי סימט 2 או 4).
2. קובץ הרצה (מקומפל ומקושר) עבור מערכת אוביונטו.
3. קובץ makefile. הקימפול חייב להיות עם הקומפיילר gcc והדגלים : -Wall -ansi. יש לנפות את כל ההודעות שモציא הקומפיילר, כך שהתוכנית תתקמפל ללא כל הערות או אזהרות.
4. דוגמאות הרצה (קלט ופלט):
 - א. קבצי קלט בשפת אסמבלי, ובקבצי הפלט שנוצרו מהפעלת האסמבילר על קבצי קלט אלה. יש להציג שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האסמבלי.
 - ב. קבצי קלט בשפת אסמבלי המציגים מגוון רחב של סוגים של אסמבלי (ולכן נוצרים קבצי פלט), ותדפסי המסן המראים את הודעות השגיאה שモציא האסמבילר.

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

נזכיר מספר היבטים חשובים של כתיבת קוד טוב :

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

הערה : תוכנית "עובדת", דהיינו תוכנית שمبرכעת את כל הדורש ממנה, אינה לכשעצמה ערובה לצוין גובה. כדי לקבל צוין גובה, על התוכנית לעמוד בקריטריונים של כתיבה ותיעוד ברמה טובה, כמוואר לעיל, אשר משקלם המשותף מגע עד לכ- 40% משקל הפרויקט.

יותר 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.** חובה שסטודנטים, הבוחרים להגיש יחד את הפרויקט, יהיו **שייכים** לאותה קבוצת הנחיה. הצוין יהיה זהה לשני הסטודנטים.

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

רקע כללי ומטרת הפרויקט

כידוע, קיימות שפות תוכנות, ומספר גדול של תוכניות, הכתובות בשפות שונות, עשויות לזרז באותו מחשב עצמו. כיצד "מכיר" המחשב כל אחד הרבה שפות? התשובה: המחשב מכיר למעשה שפה אחת בלבד: הוראות ונתונים הכתובים בקוד בינארי. קוד זה מאוחסן בOSH זיכרון, ונראה כמו רצף של ספרות ביןאייות. יחידת העיבוד המרכזית - היע"מ (CPU) - יודעת לפרק את הרץ הזה לקטעים קטנים בעלי משמעות: הוראות, מענים ונתונים.

למעשה, זיכרונו המחשב יכול הוא אוסף של סיביות, שנוהגים לראותן כמקובצות לייחדות בעלות אורך קבוע (בתים, מילימ). לא ניתן להבחין, בין שайינה מיומנת, בהבדל פיסי כלשהו בין אותו חלק בזיכרון שבו נמצאת תוכנית לבין שאר הזיכרונו.

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מגוון פעולות פשוטות, הנקראות **הוראות מכונה**, ולשם כך היא משתמש באוגרים (registers) הקיימים בתוך היע"מ, ובזיכרון המחשב. **דוגמאות**: העברת מספר מתא בזיכרון לאוגר ביע"מ או בחרזה, הוספה 1 למספר הנמצא באוגר, בדיקה האם מספר המאוחסן בicode שווה לאפס, חיבור וחישור בין שני אוגרים, וכו' . הוראות המכונה ושילובים שלהן הן המרכיבות תוכנית כפי שהיא טעונה לזכרונו בזמן ריצתה. כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנן), מתורגם בסופו של דבר באמצעות תוכנה מיוחדת לזרעה סופית זו.

היע"מ יודע לבצע קוד שנמצא בפורמט של **שפת מכונה**. זהו רצף של ביטים, המהווים קידוד ביניاري של סדרת הוראות המכונה המרכיבות את התוכנית. קוד זה אינו קרייא למשתמש, ולכן לא נוח לקוד (או לקרוא) תוכניות ישירות בשפת מכונה. **שפת אסמבלי** (assembly language) היא שפת תוכנות מאפשרת לייצג את הוראות המכונה בזרחה סימבולית קלה ונוחה יותר לשימוש. כמובן שיש צורך לתרגם את הייצוג הסימבולי לקוד בשפת מכונה, כדי שהתוכנית תוכל לזרע 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 אסמבלי.

כל מודל של יע"מ (כלומר לכל אירוגון של מחשב) יש שפת מכונה ייחודית משלו, ובהתאם גם שפת אסambilר ייחודית משלו. לפיכך, גם האסambilר (כל התרגומים) הוא ייחודי ו שונה לכל יע"ם.

תפקידו של האסambilר הוא לבנות קוד המכיל קוד מכונה, מקובץ נתון של תוכנית הכתובת בשפת אסמבלי. 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.

המשימה בפרויקט זה היא ל כתוב אסambilר (כלומר תוכנית המתרגם לשפת מכונה), עבור שפת אסambilר שנדריך כאן במיוחד לצורך הפרויקט.

لتשומת לב : בהסבירים הכלליים על אופן העבודה תוכנת האסambilר, תהיה מדי פעם התייחסות גם לעבודת שלבי הקישור והטעינה. התייחסויות אלה נועדו על מנת לאפשר לכם להבין את המשך תהליך העיבוד של הפלט של תוכנת האסambilר. אין לטעות: עלייכם ל כתוב את תוכנית האסambilר בלבד. אין ל כתוב את תוכניות הקישור והטעינה!!!

המחשב הדמיוני ושפת האסטבלי

נגידר עתה את שפת האסטבלי ואת מודל המחשב הדמיוני, עבור פרויקט זה.

הערה : תואר מודל המחשב להלן הוא חלקו בלבד, ככל שנחוץ לביצוע המשימות בפרויקט.

"חומרה":

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

למעבד 8 אוגרים כלליים, בשמות : r0, r1, r2, r3, r4, r5, r6, r7. גודלו של כל אוגר הוא 12 סיביות. הסיבית ה-1 הינה פחוסה מ-0, והסיבית המשמעותית ביותר במס' 11. שמות האוגרים נכתבים תמיד עם אות 'z' קטנה.

כמו כן יש במעבד אוגר 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 4096 תאים, בכתבות 4-095, וכל תא הואה בגודל של 12 סיביות. לתא בזכרון נקרא גם בשם "מילה". הסיביות בכל מילה ממושפרות כמו באוגר.

מחשב זה עובד ורק 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 הוראה מכונה במודול שלנו מורכבת מפעולה ואופרנדים. מספר האופרנדים הוא בין 0 ל-2, בהתאם לסוג הפעולה. מבחינת התפקיד של כל אופרנד, נבחין בין אופרנד מקור (source) ואופרנד יעד (destination).

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

בקובץ הפלט המכיל את קוד המכונה שבונה האסטבלי, כל מילה תקודד בסיסי הקסאדיימלי (או פרטים לגבי קבצי פלט בהמשך).

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

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

במודל המכונה שלנו יש 16 פעולות, בפועל, למגוון שימושים לקודד יותר פעולות. כל פעולה מיוצגת בשפת אסטבלי באמצעות סימבולי על ידי **שם-פעולה**, ובקוד המכונה על ידי קומבינציה ייחודית של ערבי שני שדות במילה הראשונה של ההוראה : **קוד-הפעולה (opcode)**, **ופונקציה (funct)**.

להלן טבלת הפעולות :

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

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

להלן מפרט הפעולות בamilha הראשונה בקוד המכוונה של כל הוראה.

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

סיביות 4-7 : שדה זה, הנקרא funct, מתפרק כאשר מדובר בפעולת Skd-hpoula (opcode) שליה משותף לכמה פעולות שונות (כאמור, קוד-פעולה 2, 5 או 9). השדה funct יכול ערך ייחודי לכל פעולה מקובצת הפעולות שיש להן אותו קוד-פעולה. אם קוד-הפעולה משמש ל פעולה אחת בלבד, הסיביות של השדה funct יהיו מאופסות.

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

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

שיטות מיעון :

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

השימוש בשיטות המיעון מצריך מידע-מידע נוספת מילת-מידע אחת נספtha. כאשר בהוראה יש שני אופרנדים, קודם תופיע מילת-המידע של אופרנד המקור, ולאחריה מילת-המידע של אופרנד היעד.

להלן המפרט של שיטות המיעון.

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	תחביר האופרנד באסמלבי	דוגמה
0	מיעון מיידי (immediate)	AMILIT-MIDU NOSFAT SHL HORAHA MCILAH AT AOPRND UZMOM, SHOAH MASFER SELM BSISIIM L-2, BRUCHB SEL 12 SIVIOT	האופרנד מתחילה בטע # ולאחריו ובצמוד אליו מופיע מספר שלם בסיסי עשרוני. ברוחב של 12 סיביות	mov #1, r2 בדוגמה זו האופרנד הראשון של הוראה (אופרנד המקור) נתנו בשיטת מיעון מיידי. ההוראה כתובת את הערך 1- אל אוגר 2.
1	מיעון ישיר (direct)	AMILIT-MIDU NOSFAT SHL HORAHA MCILAH CTOBOT BZIKRON. MILA BCTBOT ZO BZIKRON HIA AOPRND. HCTOBOT MIOTZGAT CMSFER LA-SIMEN BRUCHB SEL 12 SIVIOT.	האופרנד הוא <u>תווית</u> שכבר הגדרה, או שתוגדר בהמשך הקובץ. הגדרה נעשית על ידי כתיבת התווית בתחילת השורה של הנקית 'data'. או .string', או בתחלת השורה של הוראה, או באמצעות אופרנד של הנקית 'extern'. התווית מייצגת באופן סימבולי כתובות בזיכרון.	השורה הבאה מוגדרת את התווית א' : x: .data 23 ההוראה : dec x מקטינה ב-1 את תוכן המילה שכותבת א' בזיכרון ("משתנה" א'). הכתובת א' מקודדת בAMILIT-HIDU NOSFAT. <u>דוגמה נוספת :</u> ההוראה jmp next מבצעת קפיצה אל השורה בה מוגדרת התווית next (כלומר ההוראה הבאה שתתבצע נמצאת כתובות next). הכתובת next מקודדת בAMILIT-HIDU NOSFAT.
2	מיעון יחסית (relative)	SHIPA ZO RLOONIYAT AK DRUK LHORAOOT HMDTZOT KPIZA (HSUTUPOT) LHORAOA ACHRAH. MDOVR BHORAOOT UM KOD-FUALEH 9 BLBD : bne, jsr, jmp, ... LA-NITUN LASHTEMISH BSHIPTA ZO BHORAOOT UM KODI-FUALEH ACHRAM. BSHIPA ZO, YSH BKIDUD HORAHA AMILIT-HIDU NOSFAT, MCILAH AT MRACH KPIZA, BMILOT ZIKRON, MMILIT-HIDU NOCHCHIET AL MILIL HORAHA SHL HORAHA HMVKASH (HORAHA BAHA LBIVZO).	האופרנד מתחילה בטע % ולאחריו ובצמוד אליו מופיע שם של תווית. התווית מייצגת באופן סימבולי כתובות של <u>הוראה</u> <u>בקובץ המקור הנוכחי של</u> <u>התוכנית</u> .	jmp %next בדוגמה זו, ההוראה jmp מבצעת קפיצה אל השורה במה מוגדרת התווית next (כלומר ההוראה הבאה שתתבצע נמצאת כתובות next). נניח, לדוגמה, כי ההוראה jmp שבדוגמה נמצאת בכתובת 500 (עשרוני). כמו כן, נניח כי התווית next מוגדרת בקובץ המקור הנוכחי כתובות 300 (עשרוני). mrach kpiza mmilat-hidu jmp next (ctovot chizognit).

מספר	שיטת המיעון	תוכן מילת-המידע הנוספת	תחביר האופרנד באסמלבי	דוגמה
3	מייעון אוגר ישיר (register direct)	האופרנד הוא אוגר. AMILT-MIDU NOSFAT SHL HAHORAH MCILAH BSIYOT 7-0 BIT DOLK YHID HAMYIZG AT HAOGER HMTAIM. SIYOT 0 TZDLOK AM MDOBR BAOGER 0Z, SIYOT 1 TZDLOK AM MDOBR BAOGER 1Z VKO. SIYOT 8-11 YHIO TMID MAOPSFOT	clr r1 בדוגמה זו, ההוראה clr נפסת את האוגר 1.r. המילה הנוספת של ההוראה תכיל (בבינארי) 000000000010 <u>דוגמה נוספת:</u> mov #1, r2 האופרנד השני של ההוראה (אופרנד היעד) נתון בשיטת מייעון אוגר ישיר. ההוראה כתובת את הערך המיידי 1- אל אוגר 2.r. המילה הנוספת השנייה של ההוראה תכיל (בבינארי) 0000000000100	clr r1 בדוגמה זו, ההוראה clr נפסת את האוגר 1.r. המילה הנוספת של ההוראה תכיל (בבינארי) 000000000010 <u>דוגמה נוספת:</u> mov #1, r2 האופרנד השני של ההוראה (אופרנד היעד) נתון בשיטת מייעון אוגר ישיר. ההוראה כתובת את הערך המיידי 1- אל אוגר 2.r. המילה הנוספת השנייה של ההוראה תכיל (בבינארי) 0000000000100

פרט הוראות המכונה:

בתיאור הוראות המכונה השתמש במונח **PC** (קיצור של "Program Counter".) זהו אוגר פנימי של המעבד (לא אוגר כלל), שמכיל בכל רגע נתון את כתובת הזיכרון בה נמצא ההוראה **הקיימת שמתבצעת** (הכוונה תמיד לכתובת המילה הראשונה של ההוראה). הוראות המכונה מתחולקות לשולש קבוצות, לפי מספר האופרנדים הנדרשים לפעולה.

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

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

קבוצת הוראות השניה:

אלו הן הוראות המקבילות אופרנד אחד בלבד. אופן הקידוד של האופרנד הוא כמו של אופרנד היעד בהוראה עם שני אופרנדים. השדה של אופרנד המקור (סיביות 2-3) ב밀יה הראשונה בקידוד ההוראה אינו בשימוש, ולפיכך יהיו מאופסים.

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

הורה	opcode	funct	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
clr	5	10	אייפוס תוכן האופרנד.	clr r2	האגור 2 ז' מקבל את הערך 0.
not	5	11	היפוך הסיביות באופרנד (כל סיבית שערכה 0 תהפוך ל-1 ולהיפך : 1 ל-0).	not r2	כל בית באגור 2 ז' מתחפה.
inc	5	12	הגדלת תוכן האגור 2 ז' מוגדל ב-1.	inc r2	תוכן האגור 2 ז' מוגדל ב-1.
dec	5	13	הקטנת תוכן האופרנד באחד.	dec Count	תוכן המשתנה Count מוקטן ב-1.
jmp	9	10	קפיצה (הסתעפות) בלתי מותנית אל הוראה שנמצאת בمعنى המיוצג על ידי האופרנד. ככלומר, כתוצאה מביצוע ההוראה, מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה.	jmp %Line	בשיטות מייען יחסית, המרחק לתוויות Line מוטוסף למצביע התוכנית ולפיכך ההוראה הבאה שתתבצע תהיה בمعنى Line.
bne	9	11	bne הוא קיצור (ראשי תיבות) של : branch if not equal (to zero) הוראות הסתעפות מותניות. אם ערכו של הדגל Z באגור הסטטוס (PSW) (PC) הינו 0, אז מצביע התוכנית (PC) מקבל את כתובת יעד הקפיצה. כזכור, הדגל Z נקבע באמצעות הוראת .cmp	bne Line	אם ערך הדגל Z באגור הסטטוס (PSW) (הו 0, אז PC \leftarrow address(Line) מצביע התוכנית קיבל את כתובת התוויות Line, ולפיכך ההוראה הבאה שתתבצע תהיה בمعنى Line.)
jsr	9	12	קריאה לשגרה (סברוטינה). כתובת ההוראה שאחרי הוראת jsr הנוכחית (PC+2) נדחפת לתוך המחסנית שביברנו המחשב, ומצביע התוכנית (PC) מקבל את כתובת השגרה. הערת : חוזרת מהשגרה מתבצעת באמצעות הוראת .tz, תוך שימוש בכתובת שבמחסנית.	jsr SUBR	push(PC+2) PC \leftarrow address(SUBR) מצביע התוכנית קיבל את כתובת התוויות SUBR, ולבסוף, ההוראה הבאה שתתבצע תהיה בمعنى SUBR. SUBR. כתובת החזרה מהשגרה נשמרת במחסנית.
red	9	12	קריאה שלתו מהקלט הסטנדרטי (stdin) אל האופרנד.	red r1	קוד ה-ascii של התו הנקרא מהקלט ייכנס לאגור r1.
prn	13		הדפסת התו הנמצא (קוד ascii) לאגור r1 אל הפלט התו הנמצא (stdout).	prn r1	יודפס לפט התו הנמצא באגור r1.

קבוצת הוראות השלישייה:

אלו הן הוראות ללא אופרנדים. קידוד ההוראה מורכב מ밀יה אחת בלבד. השדות של אופרנד המקור ושל אופרנד היעד (סיביות 0-3) ב밀יה הראשונה של ההוראה אינם בשימוש, ולפיכך יהיו מאופסים.

ההוראות השויות לקבוצה זו הן : rts, stop

הורה	opcode	הפעולה המתבצעת	דוגמה	הסבר הדוגמה
rts	14	מתבצעת חזרה משגרה. הערך שבראש המחסנית של המחשב מוצא מן המחסנית, ומוכנס למצביע התוכנית (PC).	rts	PC \leftarrow pop() ההוראה הבאה שתתבצע תהיה זו שאחרי הוראת jsr שקרה לשגרה.
stop	15	עצירת ריצת התוכנית.	stop	התוכנית עצרת מיידית.

מבנה תכנית בשפת אסמבלי :

תכנית בשפת אסמבלי בנויה ממשפטים (statements). קובץ מקור בשפת אסמבלי מורכב משורת המכילות משפטיים של השפה, כאשר כל משפט מופיע בשורה נפרדת. לעומת זאת, הפרדה בין משפט למשפט בקובץ המקור הינה באמצעות התו 'ז' (שורה חדשה).

אורכה של שורה בקובץ המקור הוא 80 תווים לכל היתר (לא כולל התו טה).

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

סוג המשפט	הסבר כללי
משפט ריק	זהי שורה המכילה אך ורק תווים לבנים (whitespace), ככלומר רק את התווים ' ' ו- 't' (רווחים וטאבים). יתכן ו בשורה אין אף תו (למעט התו טה), ככלומר השורה ריקה.
משפט הערתה	זהי שורה בה התו הראשון הינו ';' (נקודה פסיק). על האסמלר להעתלם לחלווטין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמלר מה עליו לעשות כשהוא פועל על תוכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצת ויכרונו ואתחול משתנים של התוכנית, אך הוא אינו מייצר קידוד של הוראות מכונה המיועדות לביצוע בעת ריצת התוכנית.
משפט הוראה	זהו משפט המציין קידוד של הוראות מכונה לביצוע בעת ריצת התוכנית. המשפט מורכב ממשם ההוראה (פעולה) שעל המעבד לבצע, והאפרנדים של ההוראה.

cutet נפרט יותר לגבי סוגי המשפטיים השונים.

משפט הנחיה :

משפט הנחיה הוא בעל המבנה הבא :

בתחילת המשפט יכולה להופיע הגדרה של תווית (label). לתווית יש תחביר חוקי שיתוואר בהמשך. התווית היא אופציונלית.

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

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

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

1. ההנחיה 'data'.

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

.data 7, -57, +17, 9

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

המשפט 'data' מנהה את האסמלר להקצת מקום בתמונה הנתונים (data image), אשר בו יאוחסנו הערכים של הפרמטרים, ולקדם את מונה הנתונים, בהתאם למספר הערכים. אם בהנחת '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 להציג שם של משתנה).

כלומר אם נכתב :

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

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

אם נכתב בתוכנית את הוראה :

mov XYZ, r1

אז בזמן ריצת התוכנית יוכנס לאוגר r1 הערך 7.

ואילו הוראה :

lea XYZ, r1

תכנס לאוגר r1 את ערך התווית XYZ (כלומר הכתובת בזיכרון בה מאוחSEN הערך 7).

2. ההנחה 'string'

להנחה 'string', פרמטר אחד, שהוא מחרוזת חוקית. תווים המחרוזת מקודדים לפי ערכי ASCII המטאימים, ומוכנסים אל תמונה הנתונים לפי סדרם, כל TWO ב밀יה נפרדת. בסוף המחרוזת יתווסף הthin '0' (הערך המספרי 0), המסמן את סוף המחרוזת. מונה הנתונים של האסמבילר יקודם בהתאם לאורך המחרוזת (בתוספת מקום אחד עבור התו המסיים). אם בשורת ההנחה מוגדרת תווית, אז תווית זו מקבלת את ערך מונה הנתונים (לפניהם) ומוכנסת אל טבלת הסמלים, בדומה כפי שנעשה עבור 'data'. (כלומר ערך התווית יהיה הכתובת בזיכרון שבו מתחלפת המחרוזת).

לדוגמה, ההנחה :

STR: .string "abcdef"

מקצת בתמונה הנתונים רצף של 7 מילימ', ומאחלה את המילימ' לקוד ASCII של התווים לפי הסדר במחרוזת, ולאחריהם הערך 0 לסימון סוף מחרוזת. התווית STR מזוהה עם כתובות התחלה המחרוזת.

3. ההנחה 'entry'

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

לדוגמה, השורות :

HELLO: .entry HELLO
 add #1, r1

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

לשומות לב : תווית המוגדרת בתחילת שורת entry. הינה חסרת משמעות והאסמבילר מועלם מהתווית זו (אפשר שהאסמבילר יוציא הודעה אזהרה).

4. ההנחה 'extern'

להנחה 'extern', פרמטר אחד, והוא שם של תווית שאינה מוגדרת בקובץ המקור הנוכחי. מטרת ההוראה היא להודיע לאסמבילר כי התווית מוגדרת בקובץ מקור אחר, וכי קוד האסמבילר בקובץ הנוכחי עשויה בתווית שימוש.

נשים לב כי הינה זו תואמת להנחיה 'entry.' המופיעה בקובץ בו מוגדרת התווiot. בשלב הקישור תתבצע התאמה בין ערך התווiot, כפי שנקבע בקובץ המוכנה של הקובץ שהגדיר את התווiot, לבין קידוד החוראות המשמשות בתווiot בקבצים אחרים (שלב הקישור אינו רלוונטי למניין זה).

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

.extern HELLO

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

لتשומת לב : תווiot המוגדרת בתחילת שורת extern. הינה חסרת משמעות והאSEMBLER מ**תעלם** מתווiot זו (אפשר שהוא אSEMBLER יוציא הודעה אזהרה).

משפט הוראה :

משפט הוראה מורכב מחלקים הבאים:

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

אם מוגדרת תווiot בשורת הוראה, אז היא תוכנס אל טבלת הסמלים. ערך התווiot יהיה מען המילה הראשונה של ההוראה בתוך תמונה הקוד שבונה האSEMBLER.

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 האOPERAND הראשון באמצעות רווחים ו/או טאים (אחד או יותר).

כאשר יש שני אופרנדים, האופרנדים מופרדים זה מזו ב' , (פסיק). בדומה להנחיה 'data.', **לא חייבת להיות הצמדה של האופרנדים לפסיק.** כל כמה של רווחים ו/או טאים משנה צידי הפסיק היא חוקית.

למשפט הוראה עם שני אופרנדים המבנה הבא:

label: opcode source-operand, target-operand

לדוגמא:

HELLO: add r7, B

למשפט הוראה עם אופרנד אחד המבנה הבא:

label: opcode target-operand

לדוגמא:

HELLO: bne %XYZ

למשפט הוראה ללא אופרנדים המבנה הבא:

label: opcode

לדוגמא:

END: stop

אפיון השדות במשפטים של שפת האSEMBLER

תוiot:

תוiot היא סמל שמוגדר בתחילת משפט הוראה' או בתחילת הנחיה data. או string. אוalfabetית מתחילה באות alfabetית (גדולה או קטנה), ולאחריה סדרה של תווiot של אותיות alfabetיות (גדולות או קטנות) ו/או ספרות. האורך המקסימלי של תווiot הוא 31 תווים.

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

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

לדוגמא, התוויות המוגדרות להלן הן תוויות חוקיות.

hHello:

x:

He78902:

لتשומת לב: מילים שמורות של שפת האסטמבי (כלומר שם של פעולה או הינה, או שם של פעולה) אינן יכולות לשמש גם שם של תווית. לדוגמה: הסמליים add, z לא יכולים לשמש כתווית, אבל הסמליים Add, R, z, R3 הם תוויות חוקיות.

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

لتשומת לב: מותר במשפט הוראה להשתמש באופרנד שהוא סמל שאינו מוגדר כתווית בקובץ הנוכחי, כל עוד הסמיל מאופיין כחיצוני (באמצעות הנחיהtern). כלשהו בקובץ הנוכחי.

מספר:

מספר חוקי מתחילה בסימן אופציוני: '-' או '+' ולאחריו סדרה של ספרות בסיס עשרוני. לדוגמה: -5, +76, +123 הם מספרים חוקיים. אין תמייחת שפה האסטמבי שלנו ביצוג בסיס אחר מאשר עשרוני, ואין תמייחת במספרים שאינם שלמים.

מחרוזות:

מחרוזות חוקית היא סדרת תוו ascii נראים (שניינטס להדפסה), המוקפים במרקאות כפולות (המרקאות אינן נחשבות חלק מהמחרוזות). דוגמה למחרוזות חוקית: "hello world".

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

האסטמבלר בונה מלכתחילה קוד מכונה שמיועד לטעינה החל מכתובת 100. אולם, לא בכל פעע שהקוד ייטען לזכור הרצה, מובטח שאפשר יהיה לטען אותו החל מכתובת 100. במקרה כזה, קוד המכונה הנוכחי אינו מתאים ויש צורך לתקן אותו. לדוגמה, מילת-המידע של אופרנד בשיטת מעון ישיר לא תהייה נכונה, כי הכתובת השתנתה.

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

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

- האות A (קייזור של Absolute) באח לצין שתוקן המילה אינו תלוי במקום בו זיכרונו בו יייטה-בפועל קוד המכונה של התוכנית בעת ביצועה (למשל, המילה הראשונה בכל הוראה, או מילת-מידע המכילה אופרנד מיידי).

- האות R (קייזור של Relocatable) באח לצין שתוקן המילה תלוי במקום בו זיכרונו בו יייטה-בפועל קוד המכונה של התוכנית בעת ביצועה (למשל, מילת-מידע המכילה כתובת של תווית המוגדרת בקובץ המקור הנוכחי).

- האות E (קייזור של External) באח לצין שתוקן המילה תלוי בערכו של סמל שאינו מוגדר בקובץ המקור הנוכחי (למשל, מילת-מידע המכילה ערך של סמל המופיע בהנחיית extern).

נשים לב כי רוב המילים בקוד המכונה מאופיינות על ידי האות A. למעשה, רק מילת-המידע הנוספת של שיטת מיון ישר תופיע על ידי האות R או E (תלו依 אם האופרנד בקוד האסמבלי הוא תווית מקומית או סמל חיצוני).

אסמבילר עם שני מעברים

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

לדוגמא: האסמבילר מקבל את התוכנית הבאה בשפת אסמבלי:

```

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

```

קוד המכונה של התוכנית (הוראות ונתונים) נבנה כך שיתאים לטיעינה בזיכרון החל מען 100 (עדרוני).

התרגום של תוכנית המכור שבדוגמה لكוד בינארי מוצג להלן:

Address (decimal)	Source Code	Explanation	Machine Code (binary)	"A,R,E"
0100	MAIN: add r3, LIST	First word of instruction	001010101101	A
0101		Register r3	000000001000	A
0102		Address of label LIST	000010000101	R
0103	LOOP: prn #48		110100000000	A
0104		Immediate value 48	000000110000	A
0105	lea STR, r6		010000000111	A
0106		Address of label STR	000010000000	R
0107		Register r6	000001000000	A
0108	inc r6		010111000011	A
0109		Register r6	000001000000	A
0110	mov r3, K		000000001101	A
0111		Register r3	000000001000	A
0112		Address of label K	000010001000	R
0113	sub r1, r4		001010111111	A
0114		Register r1	000000000010	A
0115		Register r4	000000010000	A
0116	bne END		100110110001	A
0117		Address of label END	000001111111	R
0118	cmp val1, #-6		000100000100	A
0119		Address of extern label val1	000000000000	E
0120		Immediate value -6	11111111010	A

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

האסמבלר מחזיק טבלה שבה רשומים כל שמות הפעולה של הhorאות והקודים הבינאריים (opcode, funct) המתאימים להם, וכן שמות הפעולות ניתנים להמרה לביניاري בקלות. כאשר נקרא שם פעולה, אפשר פשוט לעיין בטבלה ולמצאו את הקידוד הביניاري.

כדי לבצע המרה לביניاري של אופרנדים שכותבים בשיטות מייען המשמשות בסמלים (תוויות), יש צורך לבנות טבלה המכילה את ערכיו כל הסמלים. אולם בהבדל ממקודים של הפעולות, הידיעות מראש, הרוי המענינים בזיכרונו עבור הסמלים בשימוש התוכנית אינם ידועים, עד אשר תוכנת המקור נסרקה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד לעיל, האסמבלר אינו יכול לדעת שהסמל END אמור להיות משוויך למן 127 (עשרוני), והסמל K אמור להיות משוויך למן 136, אלא רק לאחר שנקרוו 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 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 מספרי, שהוא מן בזיכרון.

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

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

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

لتשומת לב: תפקיד האסמבלר, על שני המעברים שלו, לתרגם קובץ מקור לקוד בשפת מכונה. בגמר פועלות האסמבלר, התוכנית טרם מוכנה לטעינה לזכרון לצורך ביצוע. קוד המכונה חייב בעבר לשלי הקיישור/טיעינה, ורק לאחר מכן לשלב הביצוע (שלבים אלה אינם חלק מההממי'ן).

המעבר הראשון

במעבר הראשון נדרשים כללים כדי לקבוע איזה מעו ישוק לכל סמל. העיקרונו הבסיסי הוא לספור את המיקומות בזיכרון, ואולם תופעות ההוראות. אם כל הוראה תיטען בזיכרון למקום העוקב להוראה הקודמת, תציג ספירה כזו את מען ההוראה הבאה. הספירה נעשית על ידי האסטברל ומוחזקת במונה ההוראות (IC). ערכו ההתחלתי של IC הוא 100 (עשרה), ולכן המכמה של ההוראה הראשונה נבנה כך שיטען מהן 100. IC מעדכן בכל שורת הוראה המקצת מקום בזיכרון. לאחר שהאסטברל קובע מהו אורך ההוראה, ה-IC מוגדל במספר התאים (מילימטרים) הנתפסים על ידי ההוראה, וכך הוא מציביע על התא הפניו הבא.

כאמור, כדי לקודד את ההוראות בשפט מכונה, מחייב האסטברל טבלה, שיש בה קידוד מתאים לכל שם פעולה. בזמן התרגום מחליף האסטברל כל שם פעולה בקידוד שלו. כמו כן, כל אופרנד מוחלף בקידוד מותאים, אך פועלות החלפה זו אינה כה פשוטה. ההוראות משתמשות בשיטות מייען מגוונות לאופרנדים. אחתן פעולה יכולה לקבל שימושויות שונות, בכל אחת משתמשות המייען, וכן יתאיםו לה קידודים שונים לפי שיטות המייען. לדוגמה, פעולה ההזזה *Setm* יכולה להתיחס להעתקת תוכן תא זיכרון לאוגר, או להעתקת תוכן אוגר אחר, וכן הלאה. ככל אפשרות זאת של *Setm* עשוי להתאים קידוד שונה.

על האסטברל 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 האסטברל בתווית המופיעה בתחילת השורה, הוא יודע שלפני הגדרה של תווית, והוא משייך לה מען – תוכנו הנוכחי של IC. כך מקבלות כל התוויות את מענהן בעת ההגדירה. תוויות אלה מוכנסות לטבלת הסמלים, המכילה בנוסף לשם התווית גם את המعن ומאפיינים נוספים. כאשר תהיה התייחסות לתווית באמצעות הוראה כלשהי, יוכל האסטברל לשולף את המعن המתאים מטבלת הסמלים.

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

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

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

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

המעבר השני

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

לשם כך מבצע האסטברל 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 מתורגמת בשלמותה לקוד מכונה.

הפרזה הוראות ונתונים

בתוכנית מבחןים בשני סוגים של תוכן: הוראות ונתונים. יש לארון את קוד המוכנה כך שתתיה הפרדה בין הנתונים וההוראות. הפרזה ההוראות והנתונים לקטעים שונים בזיכרון היא שיטה עדיפה על פני הצמדה של הגדרות הנתונים להוראות המשמשות בהן.

אחת הסכנות הטמונה באין הפרזה ההוראות מהנתונים היא, שלפעמים עלול המעבד, בעקבות שגיאה לוגית בתוכנית, לנוטות "לבצע" את הנתונים כailo היו הוראות חוקיות. למשל, שגיאה שיכולה לגרום לתופעה כזו היא הסתעפות לא נכונה. התוכנית כמובן לא תעבור נכון, אך לרוב הנזק הוא יותר חמוץ, כי נוצרת חריגת חומרה ברגע שהמעבד מבצע פעולה שאינה חוקית.

האסטמבלר שלו חייב להפריד, בקוד המוכנה שהוא מיציר, בין קטע הנתונים לקטע ההוראות.
כלומר **בקובץ הפלט (בקוד המוכנה) תהיה הפרזה של הוראות ונתונים לשני קטיעים נפרדים**,
אם כי בקובץ הקלט אין חובה שתהייה הפרזה כזו. בהמשך מתואר אלגוריתם של האסטמבלר,
ובו פרטיהם כיצד לבצע את הפרזה.

גילוי שגיאות בתוכנית המקור

האסטמבלר אמר לגלות ולדוח על שגיאות בתחריב של תוכנית המקור, כגון פעולה שאינה קיימת, מספר אופרנדים שוגוי, סוג אופרנד שאינו מותאים לפעולה, שם אוגר לא קיים, ועוד שגיאות אחרות. כמו כן מודיא האסטמבלר שככל סמל מוגדר פעם אחת בדוק.

מכאן, שככל שגיאה המתגלה על ידי האסטמבלר נגרמת (בדרכן כלל) על ידי שורת קלט מסוימת.
לדוגמה, אם מופיעים שני אופרנדים בהוראה שאמור להיות בה רק אופרנד אחד, האסטמבלר ייתן הודעת שגיאה בנוסח "יותר מדי אופרנדים".

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

لتשומת לב: האסטמבלר אין עוצר את פעולתו אחרי שנמצאה השגיאה הראשונה, אלא ממשיך לעبور על הקלט כדי לגלוות שגיאות נוספות, ככל שישן. כמובן שאון כל טעם לייצר את קבצי הפלט אם נתגלו שגיאות (ממילא אי אפשר להשלים את קוד המוכנה).

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

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

תהליך העבודה של האסטמבלר

נותאר כעת את אופן העבודה של האסטמבלר. בהמשך, יוצג אלגוריתם שלדי למעבר ראשון ושני.

האסטמבלר מתחזק שני מערכים, שיקראו להן תМОונת הHorאות (code) ותМОונת הנתומנים (data). מערכים אלו נותנים למשהה תМОונה של זיכרון המכונה (כל איבר במערך הוא בגודל מילה של המכונה, כלומר 24 סיביות). במערך הHorאות בונה האסטמבלר את הקידוד של הHorאות המכונה שנקרוו במהלך המעבר על קובץ המקור. במערך הנתומנים מכניס האסטמבלר את קידוד הנתומנים שנקרוו מקובץ המקור (שורות הנחיה מסוג 'data.' ו-'string').

האסטמבלר משתמש בשני מונחים, שנקרואים IC (МОונה הHorאות - Instruction-Counter) ו- DC (Data-Counter) (МОונה הנתומנים - Data-Counter). מוניט אללו מצביעים על המיקום הבא הפוני במערך הHorאות ובמערך הנתומנים, בהתאם. בכל פעע כשותחיל האסטמבלר לעבר על קובץ מקור, המונה IC מקבל ערך התחלתי 0, והМОונה DC מקבל ערך התחלתי 0. הערך התחלתי IC=100 נקבע כדי שקווד המכונה של התוכנית יתאים לטעינה לזכרון (לצורך ריצה) החל מכתובת 100.

בנוסף, מתחזק האסטמבלר טבלה, אשר בה נאשפות כל התוויות בהן נתקל האסטמבלר במהלך המעבר על קובץ המקור. לטבלה זו קוראים טבלת-הסמלים (symbol-table). לכל סמל נשמרם בטבלה שם הסמל, ערכו המספרי, ומאפיינים נוספים (אחד או יותר), כגון המיקום בתמונה הזיכרון (code או data), וסוג הנראות של הסמל (external או entry).

במעבר הראשון האסטמבלר בונה את טבלת הסמלים ואת השילד של תМОונת הזיכרון (Horאות ונתומנים).

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

1. שורה ריקה או שורת הערה : האסטמבלר מתעלם מהשורה וועבר לשורה הבאה.
2. שורת הוראה :

האסטמבלר מנתח את השורה ופענה מהי ההוראה, ומבחן שיטות המייען של האופרנדים. מספר האופרנדים נקבע בהתאם להוראה שנמצאה. שיטות המייען נקבעות בהתאם לתחביר של כל אופרנד, כפי שהסביר לעיל במפרט שיטות המייען. למשל, התו '#' מייען מידיה, תווית מצינית מייען ישיר, שם של אוגר מייען אוגר ישיר, וכו'.

אם האסטמבלר מוצא בשורת ההוראה גם הגדרה של תווית, אזו התווית (הסמל) המוגדרת מוכנסת לטבלת הסמלים. ערך הסמל בטבלה הוא IC, ומהפיין הוא code.

כעת האסטמבלר קובע לכל אופרנד את ערכו באופן הבא :

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

האסטמבלר מכניס למערך הHorאות, בKİנסת עלייה מצביע מונה הHorאות IC, את הקוד הבינארי המלא של המילה הראשונה של ההוראה (בפורמט קידוד כפי שתואר קודם). מילה זו מכילה את קוד הפעולה וה-*opnd*, ואת מספרי שיטות המייען של אופרנד המקור והיעד. IC מקודם ב-1.

זכור שכאשר יש רק אופרנד אחד (כלומר אין אופרנד מקור), הסיבות של שיטות המייען של אופרנד המקור יכילה 0. בדומה, אם זהה ההוראה ללא אופרנדים (ts, stop וכו'), אזו הסיבות של שיטות המייען של שני האופרנדים יכילה 0.

אם זהה הוראה עם אופרנדים (אחד או שניים), האסטמבלר "משריאין" מקומם במערך ההוראות עברו מילות-המידע הנוספות הנדרשות בהוראה זו, ככל שנדרשות, ומקדם את IC בהתאם. כאשר אופרנד הוא בשיטת מיון מיידי או אוגר ישיר, האסטמבלר מקודד גם את המילה הנוספת המתאימה במערך ההוראות. ואילו בשיטת מיון ישיר או יחס, מילת המידע הנוספת במערך ההוראות נשארת ללא קידוד בשלב זה.

3. שורת הנחיה :

כאשר האסטמבלר קורא בקובץ המקור שורת הנחיה, הוא פועל בהתאם לסוג הנחיה, באופן הבא :

I. '.data' .
האסטמבלר קורא את רשימת המספרים, המופיעיה לאחר '.data.', מכניס כל מספר אל מערך הנתונים (בקידוד בינהר), ומקדם את מצביע הנתונים DC ב-1 עברו כל מספר שהוכנס.

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

II. '.string' .
הטיפול ב-'string' דומה ל-'data.', אלא שקובדי ה-ascii של התווים הם אלו המוכנסים אל מערך הנתונים (כל تو במילה נפרצת). לבסוף מוכנס למערך הנתונים הערך 0 (המצין סוף מחוזות). המונה DC מקודם באורך המחרוזות + 1 (גם התו המסיים את המחרוזת תופס מקום).

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

III. '.entry' .
זהה הנחיה לאסטמבלר לאפיון את התווית הנתונה כאופרנד entry בטבלת הסמלים. בעת הפיקת קבצי הפלט (ראו בהמשך), התווית תירשם בקובץ ה-entry entries.
لتשומת לב : זה לא נחשב כשגיאה אם בקובץ המקור מופיעיה יותר מהנחיה entry. אחת עם אותה תווית כאופרנד. המופעים הנוספים אינם מוסיפים דבר, אך גם אינם מפריעים.

IV. '.extern' .
זהה הציהרה על סמל (תווית) המוגדר בקובץ מקור אחר, והקובץ הינו עושה בו שימוש. האסטמבלר מכניס את הסמל המופיע כאופרנד לטבלת הסמלים, עם הערך 0 (הערך האמייתי לא ידוע, ויקבע רק בשלב הקישור), ועם המאפיין external. לא ידוע באיזה קובץ נמצאת הגדרת הסמל, ואין זה רלוונטי עבור האסטמבלר.
لتשומת לב : זה לא נחשב כשגיאה אם בקובץ המקור מופיעיה יותר מהנחיה extern. אחת עם אותה תווית כאופרנד. המופעים הנוספים אינם מוסיפים דבר, אך גם אינם מפריעים.

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

בסוף המעבר הראשון, האסטמבלר מעדכן בטבלת הסמלים כל סמל המופיע כ-'data', על ידי הוספה (100+) IC (עשרות) לערכו של הסמל. הסיבה לכך היא שבתמונה הכלולת של קוד המcona, תמונה הנתונים מופרdataת מתמונה הוראות, וכל הנתונים נדרש להופיע בקוד המcona אחרי כל ההוראות. סמל מסווג data הוא תווית בתמונה הנתונים, והעדכו מוסיף לערך הסמל (כלומר לכנתובתו בזיכרון) את האורך הכולל של תמונה ההוראות, בתוספת כתובות התחלה הטעינה של הקוד, שהיא 100.

טבלת הסמלים מכילה בעת את ערכי כל הסמלים הנחוצים להשלמת תמונה הזיכרון (למעט ערכים של סמלים חיצוניים).

במעבר השני, האסטמבלר משלים באמצעות טבלת הסמלים את קידוד כל המילים במערך ההוראות שטרם קודדו במעבר הראשון. במודל המcona שלנו אלו הן מילות-מידע נוספת נוספה של ההוראות, אשר מקודדות אופרנד בשיטת מיון ישיר או יחס.

אלגוריתם שלדי של האסמבלי

ל釐וד ההבנה של תהליך העבודה של האסמבלי, נציג להלן אלגוריתם שלדי למעבר הראשון ולמעבר השני.

لتשומת לב: אין חובה להשתמש דווקא באלגוריתם זה.

כאמור, אנו מחלקים את תמונות קוד המכונה לשני חלקים: תמונות ההוראות (code), ותמונה הנטוינים (data). לכל חלק נתזקק מונה נפרד: IC (מונה ההוראות) ו-DC (מונה הנטוינים).

בנייה את קוד המכונה כך שיתאים לטעינה לזכרון החל מכתובת 100.

בכל מעבר מתחילה לקרוא את קובץ המקור מהתחלת.

מעבר ראשון

1. אתחל 100 $\leftarrow 0$, IC \leftarrow 0, DC \leftarrow 0.
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עברו ל-17.
3. האם השדה הראשון בשורה הוא תווית? אם לא, עברו ל-5.
4. הדליק דגל "יש הגדרת סמל".
5. האם זהה הנחיה לאחסון נתונים, ככלומר, האם הנחיה string. או data. ? אם לא, עברו ל-8.
6. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם המאפיין data. ערך הסמל יהיה DC. (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה, יש להודיע על שגיאה).
7. זהה את סוג הנטוינים, קודד אותו בתמונה הנטוינים, והגדיל את מונה הנטוינים DC על ידי הוספת האורך הכלול בתמונה הנטוינים שהוגדרו בשורה הנוכחית. חוזר ל-2.
8. האם זו הנחיה extern. או הנחיה entry. ? אם לא, עברו ל-11.
9. אם זהה הנחית entry. חזרו ל-2 (הנחיה תטופל במעבר השני).
10. אם זו הנחית external, הכנס את הסמל המופיע כאופrnd של ההנחיה לתוך טבלת הסמלים עם הערך 0, ועם המאפיין external, יש להודיע על שגיאה. (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה ללא המאפיין external, יש להודיע על שגיאה). חוזר ל-2.
11. זהה שורת הוראה. אם יש הגדרת סמל (תווית), הכנס אותו לטבלת הסמלים עם המאפיין code. ערכו של הסמל יהיה IC (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה, יש להודיע על שגיאה).
12. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא, אז הodus על שגיאה בשם ההוראה.
13. נתח את מבנה האופrndים של ההוראה, וחשב מהו מספר המילים הכוללת שתופסת ההוראה בקוד המכונה (נקרא למספר זה L).
14. בנה כעת את הקוד הבינארי של המילה הראשונה של ההוראה, ושל כל מילת-מידע נוספת המកודדת אופrnd במשמעותו מיידי.
15. שמר את הערכים IC ו- L יחד עם נתונים קוד המכונה של ההוראה.
16. עדכן L $\leftarrow IC + L$, וכךര ל-2.
17. קובץ המקור נקרא בשלהמו. אם נמצא שגיאות במעבר הראשון, עצור כאן.
18. שמר את הערכים הסופיים של IC ושל DC (נקרא להם ICF ו- DCF). השתמש בהם לבניית קבצי הפלט, אחרי המעבר השני.
19. עדכן בטבלת הסמלים את ערכו של כל סמל המופיע כ- data , ע"י הוספת הערך ICF (ראה הסבר לכך בהמשך).
20. התחל מעבר שני.

מעבר שני

1. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עברו ל-7.
2. אם השדה הראשון בשורה הוא סמל (תווית), דlg עליו.
3. האם זהה הנחיה data. או extern. string. ? אם כן, חוזר ל-1.
4. האם זהה הנחית entry. ? אם לא, עברו ל-6.
5. הוסף בטבלת הסמלים את המאפיין entry למאפייני הסמל המופיע כאופrnd של ההנחיה (אם הסמל לא נמצא בטבלת הסמלים, יש להודיע על שגיאה). חוזר ל-1.

6. השלם את הקידוד הבינארי של מילוט-המידע של האופרנדים, בהתאם לשיטות המיעון שבשימוש. לכל אופרנד בקוד המקור המכיל סמל, מצא את ערכו של הסמל בטבלת הסמלים (אם הסמל לא נמצא בטבלה, יש להודיע על שגיאה). אם הסמל מאופיין external, הוסף את כטובה מילוט-המידע הרלוונטי לרשימה מילוט-המידע שמתיחסות לסמל חיצוני. לפי הorzד, לחישוב הקידוד והכトבות, אפשר להיעזר בערכיהם IC ו-L של הוראה, כפי שנשמרו בעבר הראשון. חוזר ל-1.

7. קובץ המקור נקרא בשלמותו. אם נמצא שגיאות בעבר השני, עצור כאן.

8. בנה את קבצי הפלט (פרטים נוספים בהמשך).

נפעיל אלגוריתם זה על תוכנית הדוגמה שראינו קודם, ונציג את הקוד הבינארי שמתקבל בעבר ראשון ובמעבר שני. להלן שוב תכנית הדוגמה.

```

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

```

נבצע מעבר ראשון על הקוד לעיל, ובנה את טבלת הסמלים. כמו כן, נשלים בעבר זה את הקידוד של כל תומנות הנתונים, ושל המילה הראשונה של הוראה. כמו כן, נקודד מילוט-המידע נוספת של הוראה, ככל שקיים זה איינו תלוי בערך של סמל. את מילוט-המידע שעדיין לא ניתן לקודד במעבר הראשון נסמן ב "?" בדוגמה להלן.

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

Address (decimal)	Source Code	Explanation	Machine Code (binary)	“A,R,E”
0118	cmp val1, #-6	Address of label val1	000100000100	A
0119		Immediate value -6	? 111111111010	? A
0120				
0121	bne %END	Distance to label END	100110110010	A
0122			? ?	A A
0123	dec K	Address of label K	010111010001	A
0124			? ?	? ?
0125	jmp %LOOP	Distance to label LOOP	100110100010	A
0126			? ?	A A
0127	END: stop		111100000000	A
0128	STR: .string “abcd”	Ascii code ‘a’	000001100001	A
0129		Ascii code ‘b’	000001100010	A
0130		Ascii code ‘c’	000001100011	A
0131		Ascii code ‘d’	000001100100	A
0132		Ascii code ‘\0’	000000000000	A
0133	LIST: .data 6, -9	Integer value 6	000000000010	A
0134		Integer value -9	111111110111	A
0135	.data -100	Integer value -100	111110011100	A
0136	K: .data 31	Integer value 31	000000011111	A

טבלת הסמלים אחרי מעבר ראשון היא :

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

נבע עתה את המעבר השני. נשלים באמצעות טבלת הסמלים את הקידוד החסר במילים המסומנות “?”.
הקוד הבינארי בקורסו הטופטי כאן זהה לקוד שהוגז בתחלת הנושא **“אסמבלר עם שני מעברים”**.

הערה : כאמור , האסמבלר בונה קוד מכונה כך שייתאים לטיעינה לזכרוון החל מכתובת 100 (עשרהוני).
אם הטיעינה בפועל (לצורך הרצת התוכנית) תהיה לכתובות אחרות, יידרשו תיקונים בקוד הבינארי
בשלב הטיעינה, שיוכנסו בעזרת מידע נוסף שהאסמבלר מכין בקבצי הפלט (ראו בהמשך).

Address (decimal)	Source Code	Explanation	Machine Code (binary)	“A,R,E”
0100	MAIN: add r3, LIST	First word of instruction	001010101101	A
0101		Register r3	000000001000	A
0102		Address of label LIST	000010000101	R
0103	LOOP: prn #48		110100000000	A
0104		Immediate value 48	000000110000	A
0105	lea STR, r6		010000000111	A
0106		Address of label STR	000010000000	R
0107		Register r6	000001000000	A
0108	inc r6		010110000011	A
0109		Register r6	000001000000	A
0110	mov r3, K		000000001101	A
0111		Register r3	000000001000	A
0112		Address of label K	000010001000	R

Address (decimal)	Source Code	Explanation	Machine Code (binary)	"A,R,E"
0113	sub r1, r4		001010111111	A
0114		Register r1	000000000010	A
0115		Register r4	000000010000	A
0116	bne END		100110110001	A
0117		Address of label END	000001111111	R
0118	cmp val1, #-6		000100000100	A
0119		Address of extern label val1	000000000000	E
0120		Immediate value -6	111111111010	A
0121	bne %END		100110110010	A
0122		Distance to label END	000000000101	A
0123	dec K		010111010001	A
0124		Address of label K	000010001000	R
0125	jmp %LOOP		100110100010	A
0126		Distance to label LOOP	111111101001	A
0127	END: stop		111100000000	A
0128	STR: .string "abcd"	Ascii code 'a'	000001100001	A
0129		Ascii code 'b'	000001100010	A
0130		Ascii code 'c'	000001100011	A
0131		Ascii code 'd'	000001100100	A
0132		Ascii code '\0'	000000000000	A
0133	LIST: .data 6, -9	Integer value 6	000000000110	A
0134		Integer value -9	111111101111	A
0135	.data -100	Integer value -100	111110011100	A
0136	K: .data 31	Integer value 31	000000011111	A

טבלת הסמלים אחרי מעבר שני היא :

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

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

קבצי קלט ופלט של האסמבלר

בהתפעלה של האסמבלר, יש להעיר 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 האסמבלר שהוגדרה במינ' זה.

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

- קובץ object, המכיל את קוד המכונה.
- קובץ externals, ובו פרטים על כל המיקומות (הכתובות) בקוד המכונה בהם יש מילת-מידע שמקודדת ערך של סמל שהציגו כחיצוני (סמל שהופיע כאופrnd של הנחיה `extern`, ומואפיין בטבלת הסמלים `-external`).
- קובץ entries, ובו פרטים על כל סמל שמוצחר כנקודת כניסה (סמל שהופיע כאופrnd של הנחיה `entry`, ומואפיין בטבלת הסמלים `-entry`).

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

שמות קבצי המקור חייבים להיות עם הסיוומת `".as"`. למשל, השמות `x.as`, `y.as`, ו-`as` הם
שמות חוקיים. העברת שמות הקבצים הללו כארגומנטים לאסמבלר נועשית לא ציון הסיוומת.

לדוגמא : נניח שתוכנית האסמבלר שלנו נקראת `assembler`, אז שורת הפקודה הבאה :

`assembler x y hello`

תרץ את האסמבלר על הקבצים : `.x.as, .y.as, hello.as` :

שמות קבצי הפלט מובוסים על שם קובץ הקלט, כפי שהופיע בשורת הפקודה, בתוספת סיוומת
מתאימה : הסיוומת `".ob"` עבור קובץ `object`, הסיוומת `".ent"` עבור קובץ `entries`, והסיוומת
`".ext"` עבור קובץ `extern`.

לדוגמא, בהפעלת האסמבלר באמצעות שורת הפקודה : `x` `assembler` `x` `-o ob.x`, וכן קבצי פלט `ext.x` ו- `ext.y` ככל שיש הנקיות `.entry` או `.extern`. בקובץ המקור.
נציג כתע את הפורמטים של קבצי הפלט. דוגמאות יובאו בהמשך.

פורמט קובץ ה- `object`

קובץ זה מכיל את תמונה הזיכרון של קוד המכוונה, שני חלקים : תמונה ההוראות ראשונה,
ואחריה ובצמוד תמונה הנתונים.

כזכור, האסמבלר מקודד את ההוראות כך שתמונה ההוראות תתאים לטיענה החלה מכתובות 100
(עשרוני) בזיכרון. נשים לב שركס בסוף המ עבר הראשון יודיעים מהו הגודל הכללי של תמונה
ההוראות. מכיוון שתמונות הנתונים נמצאת אחורי תמונה ההוראות, גודל תמונה ההוראות משפיע
על הכתובות בתמונה הנתונים. זו הסיבה שבגללה היה צריך לעדכן בטבלת הסמלים, בסוף המ עבר
הראשון, את ערכי הסמלים המאופיינים כ-`data` (כזכור, באлогוריתם השודי שהוצג לעיל, בצעד 19,
הושפנו לכל סמל כזה את הערך ICF). במעבר השני, בהשלמת הקידוז של מילוט-המידע,
משתמשים בערכיהם המעודכנים של הסמלים, המותאמים למבנה המלא וה壽פי של תמונה הזיכרון.

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

השורה הראשונה בקובץ ה- `object` היא "cotratt", המכילה שני מספרים (בסיס עשרוני) :
הראשון הוא האורך הכולל של תמונה ההוראות (ב밀ות זיכרון), והשני הוא האורך הכולל של
תמונה הנתונים (ב밀ות זיכרון). בין שני המספרים מפריד רווח אחד.
כזכור, במעבר הראשון, בצעד 18, נשמרו הערכים ICF ו-IDF. האורך הכולל של תמונה ההוראות
הוא 100-ICF, והאורך הכולל של תמונה הנתונים הוא IDF.

השורות הבאות בקובץ מכילות את תמונה הזיכרון. בכל שורה של שלשה שורות : כתובות של מילה
בזכרון, תוכן המילה, והמאפיין A,R,E". הכתובת תירשם בסיסי עשרוני בארכע ספרות (כולל
אפסים מוביילים). תוכן המילה יירשם בסיסי कठादციმლი - 3 ספרות (כולל אפסים מוביילים).
בין השורות בשורה יש רווח אחד.

פורמט קובץ ה- `entries`

קובץ `entries` בניית משורות טקסט, שורה אחת לכל סמל שמאופיין בטבלת הסמלים כ- `entry`.
בשורה מופיע שם הסמל, ולאחריו ערכו כפי שנקבע בטבלת הסמלים (בסיס עשרוני בארכע
ספרות, כולל אפסים מוביילים). בין שני השורות בשורה יש רווח אחד. אין חשיבות לדסדר השורות,
כי כל שורה עומדת בפני עצמה.

פורמט קובץ ה- `externals`

קובץ `externals` בניית משורות טקסט, שורה לכל כתובות בקוד המכוונה בה יש מילת
מידע המתיחסת לסמל שמאופיין כ- `external`. כזכור, רישמה של מילוט-מידע אלה נבנתה
במעבר השני (צעד 6 באлогוריתם השלים).

כל שורה בקובץ ה-`externals` מכילה את שם הסמל החיצוני, ולאחריו הכתובת של מילט-המידע (בבסיס עשרוני באربע ספרות, כולל אפסים מובילים). בין שני השדות בשורה יש רווח אחד. **אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.**

لتשומת לב: ניתן ויש מספר כתובות בקובץ המכוונה בהן מילט-המידע מתייחסות לאותו סמל חיצוני. לכל כתובה כזו תהיה שורה נפרדת בקובץ ה-`externals`.

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

; file ps.as

```
.entry LIST
.extern W
MAIN:    add   r3, LIST
LOOP:     prn   #48
          lea   W, r6
          inc   r6
          mov   r3, K
          sub   r1, r4
          bne   END
          cmp   K, #-6
          bne   %END
          dec   W
.entry MAIN
          jmp   %LOOP
          add   L3, L3
END:      stop
STR:      .string "abcd"
LIST:     .data   6, -9
          .data   -100
K:        .data   31
.extern L3
```

להלן הקידוד הבינארי המלא (תמונה הזיכרון) של קובץ המקור, בגמר המעבר השני.

Address (decimal)	Source Code	Explanation	Machine Code (binary)	"A,R,E"
0100	MAIN: add r3, LIST	First word of instruction	001010101101	A
0101		Register r3	000000001000	A
0102		Address of label LIST	000010001000	R
0103	LOOP: prn #48		110100000000	A
0104		Immediate value 48	000000110000	A
0105	lea W, r6		010000000111	A
0106		Address of extern label W	000000000000	E
0107		Register r6	000001000000	A
0108	inc r6		010111000011	A
0109		Register r6	000001000000	A
0110	mov r3, K		000000001101	A
0111		Register r3	000000001000	A
0112		Address of label K	000010001011	R
0113	sub r1, r4		001010111111	A
0114		Register r1	000000000010	A
0115		Register r4	000000010000	A
0116	bne END		100110110001	A
0117		Address of label END	000010000010	R
0118	cmp K, #-6		000100000100	A
0119		Address of label K	000010001011	R
0120		Immediate value -6	11111111010	A
0121	bne %END		100110110010	A
0122		Distance to label END	000000001000	A
0123	dec W		010111010001	A
0124		Address of extern label W	000000000000	E
0125	jmp %LOOP		100110100010	A
0126		Distance to label LOOP	111111101001	A
0127	add L3, L3		001010100101	A
0128		Address of extern label L3	000000000000	E
0129		Address of extern label L3	000000000000	E
0130	END: stop		111100000000	A
0131	STR: .string "abcd"	Ascii code 'a'	000001100001	A
0132		Ascii code 'b'	000001100010	A
0133		Ascii code 'c'	000001100011	A
0134		Ascii code 'd'	000001100100	A
0135		Ascii code '\0'	000000000000	A
0136	LIST: .data 6, -9	Integer value 6	000000000110	A
0137		Integer value -9	111111110111	A
0138	.data -100	Integer value -100	111110011100	A
0139	K: .data 31	Integer value 31	000000011111	A

טבלת הסמלים בגמר המעבר השני היא :

Symbol	Value (decimal)	Attributes
W	0	external
MAIN	100	code, entry
LOOP	103	code
END	130	code
STR	131	data
LIST	136	data, entry
K	139	data
L3	0	external

להלן תוכן קבצי הפלט של הדוגמה.

הקובץ ps.ob

```
    31 9
0100 2AD A
0101 008 A
0102 088 R
0103 D00 A
0104 030 A
0105 407 A
0106 000 E
0107 040 A
0108 5C3 A
0109 040 A
0110 00D A
0111 008 A
0112 08B R
0113 2BF A
0114 002 A
0115 010 A
0116 9B1 A
0117 082 R
0118 104 A
0119 08B R
0120 FFA A
0121 9B2 A
0122 008 A
0123 5D1 A
0124 000 E
0125 9A2 A
0126 FE9 A
0127 2A5 A
0128 000 E
0129 000 E
0130 F00 A
0131 061 A
0132 062 A
0133 063 A
0134 064 A
0135 000 A
0136 006 A
0137 FF7 A
0138 F9C A
0139 01F A
```

הקובץ ps.ent

MAIN 0100
LIST 0136

הקובץ ps.ext

W 0106
W 0124
L3 0128
L3 0129

סיכום והנחיות כלליות

- גודל תוכנית המקור הניתנת כקלט לאסטמבלר אינו ידוע מראש, ולכן גם גודלו של קוד המכמה אינו צפוי מראש. אולם בכדי להקל בימוש האסטמבלר, מותר להניח גודל מקסימלי. לפיכך יש אפשרות לשתמש במערכות לאחסן תומנות קוד המכמה בלבד. כל מבנה נתוני אחר (למשל טבלת הסמלים), יש למשם באופן ייעיל וחסכוני (למשל באמצעות רשיימה מקוורת והקצת זיכרון דינמי).
- השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הפלט, למעט הסיומות. למשל, אם קובץ הפלט הוא `prog.as` אז קבצי הפלט שיוציאו לו הם : `prog.ob`, `prog.ext`, `prog.ent`.
- מתוכנת הפעלת האסטמבלר צריכה להיות כפי הנדרש בממ"ן, ללא שינוים כלשהם. ככלומר, ממשך המשתמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו לתוכנית האסטמבלר כארגומנטים (אחד או יותר) בשורת הפקודה. אין להוסיף תפריטי פלט אינטראקטיביים, חלונות גרפיים למיניהם, וכו'').
- יש להזכיר לחלק את שימוש האסטמבלר במספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסוימים שונים במודול יחיד. מומלץ לחלק מודולים כגון : מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחריבי של שורה), טבלת הסמלים, מפת הזיכרון, טבלאות קבועות (קודם הפעלה, שיטות המיעון החוקיקות לכל פעולה, וכו').
- יש להזכיר ולתעד את השימוש באופן מלא וברור, באמצעות העורות מפורטות בקוד.
- יש לאפשר תווים לבנים עודפים בקובץ הפלט 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. מוגדרות גם שורות ריקות. האסטמבלר יתעלם מהתווים לבנים מיוחדים (כולם ידלג עליהם).
- הפלט (קוד האסטמבלר) עלול להכיל שגיאות תחריריות. על האסטמבלר לגלות ולזוזח על כל השורות השגויות בפלט. אין לעצור את הטיפול בקובץ פלט לאחר גילוי השגיאה הראשונה. יש להציג למשך הדעות מפורטות ככל הנין, כדי שאפשר יהיה להבין מה והיכן כל שגיאה. כמובן שגם קובץ פלט מכיל שגיאות, אין טעם להפיק עבورو את קבצי הפלט (`ob`, `ext`, `ent`).

גם ונשלם פרק החסברים והגדרת הפרויקט.

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

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

لتשומתיכם : לא תינטו דחיה בהגשת הממי"ן, פרט למקרים מיוחדים כגון מילואים או מחלוקת ממושכת. במקרים אלו יש לבקש ולקבל אישור מראש מצוות הקורס.

בזה צלחה !