מטלת מנחה (ממ"ן) 14

הקורס: 20465 - מעבדה בתכנות מערכות

חומר הלימוד למטלה: פרויקט גמר

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

סמסטר: 2021ב' | מועד אחרון להגשה: 15.08.2021

קיימות שתי חלופות להגשת מטלות:

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

הסבר מפורט ב"נוהל הגשת מטלות מנחה"

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

עליכם לכתוב תוכנת אסמבלר, עבור שפת אסמבלי שתוגדר בהמשך. הפרויקט ייכתב בשפת C.

עליכם להגיש את הפריטים הבאים:

1. קבצי המקור של התוכנית שכתבתם (קבצים בעלי סיומת c. או h.).
2. קובץ הרצה (מקומפל ומקושר) עבור מערכת אובונטו.
3. קובץ makefile. הקימפול חייב להיות עם הקומפיילר gcc והדגלים: Wall -ansi -pedantic-. יש לנפות את כל ההודעות שמוציא הקומפיילר, כך שהתוכנית תתקמפל ללא כל הערות או אזהרות.
4. דוגמאות הרצה (קלט ופלט):
   - **א.** קבצי קלט בשפת אסמבלי, וקבצי הפלט שנוצרו מהפעלת האסמבלר על קבצי קלט אלה. יש להדגים שימוש במגוון הפעולות וטיפוסי הנתונים של שפת האסמבלי.
   - **ב.** קבצי קלט בשפת אסמבלי המדגימים מגוון רחב של סוגי שגיאות אסמבלי (ולכן לא נוצרים קבצי פלט), ותדפיסי המסך המראים את הודעות השגיאה שמוציא האסמבלר.

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

נזכיר מספר היבטים חשובים של כתיבת קוד טוב:

1. הפשטה של מבני הנתונים: רצוי (ככל האפשר) להפריד בין הגישה למבני הנתונים לבין המימוש של מבני הנתונים. כך, למשל, בעת כתיבת פונקציות לטיפול בטבלה, אין זה מעניינם של המשתמשים בפונקציות אלה, האם הטבלה ממומשת באמצעות מערך או באמצעות רשימה מקושרת.

2. קריאות הקוד: יש להשתמש בשמות משמעותיים למשתנים ופונקציות. יש לערוך את הקוד באופן מסודר: הזחות עקביות, שורות ריקות להפרדה בין קטעי קוד, וכד'.

3. תיעוד: יש להכניס בקבצי המקור תיעוד תמציתי וברור, שיסביר את תפקידה של כל פונקציה (באמצעות הערות כותרת לכל פונקציה). כמו כן יש להסביר את תפקידם של משתנים חשובים. כמו כן, יש להכניס הערות ברמת פירוט טובה בכל הקוד.

הערה: תוכנית "עובדת", דהיינו תוכנית שמבצעת את כל הדרוש ממנה, אינה לכשעצמה ערובה לציון גבוה. כדי לקבל ציון גבוה, על התוכנית לעמוד בקריטריונים של כתיבה ותיעוד ברמה טובה, כמתואר לעיל, אשר משקלם המשותף מגיע עד לכ- 40% ממשקל הפרויקט.

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

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

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

רקע כללי ומטרת הפרויקט

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

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

יחידת העיבוד המרכזית (היע"מ) יכולה לבצע מגוון פעולות פשוטות, הנקראות **הוראות מכונה**, ולשם כך היא משתמשת באוגרים (registers) הקיימים בתוך היע"מ, ובזיכרון המחשב.

דוגמאות: העברת מספר מתא בזיכרון לאוגר ביע"מ או בחזרה, הוספת 1 למספר הנמצא באוגר, בדיקה האם מספר המאוחסן באוגר שווה לאפס, חיבור וחיסור בין שני אוגרים, וכד'.
הוראות המכונה ושילובים שלהן הן המרכיבות תוכנית כפי שהיא טעונה לזיכרון בזמן ריצתה.
כל תוכנית מקור (התוכנית כפי שנכתבה בידי המתכנת), תתורגם בסופו של דבר באמצעות תוכנה מיוחדת לצורה סופית זו.

היע"מ יודע לבצע קוד שנמצא בפורמט של **שפת מכונה.** זהו רצף של ביטים, המהווים קידוד בינארי של סדרת הוראות המכונה המרכיבות את התוכנית. קוד כזה אינו קריא למשתמש, ולכן לא נוח לקודד (או לקרוא) תוכניות ישירות בשפת מכונה. **שפת אסמבלי** (assembly language) היא שפת תכנות מאפשרת לייצג את הוראות המכונה בצורה סימבולית קלה ונוחה יותר לשימוש. כמובן שיש צורך לתרגם את הייצוג הסימבולי לקוד בשפת מכונה, כדי שהתוכנית תוכל לרוץ במחשב. תרגום זה נעשה באמצעות כלי שנקרא **אסמבלר** (assembler).

כידוע, לכל שפת תכנות עילית יש מהדר (compiler) , או מפרש (interpreter), המתרגם תוכניות מקור לשפת מכונה. האסמבלר משמש בתפקיד דומה עבור שפת אסמבלי.

לכל מודל של יע״מ (כלומר לכל אירגון של מחשב) יש שפת מכונה יעודית משלו, ובהתאם גם שפת אסמבלי יעודית משלו. לפיכך, גם האסמבלר (כלי התרגום) הוא יעודי ושונה לכל יע״מ.

תפקידו של האסמבלר הוא לבנות קובץ המכיל קוד מכונה, מקובץ נתון של תוכנית הכתובה בשפת אסמבלי. זהו השלב הראשון במסלול אותו עוברת התוכנית, עד לקבלת קוד המוכן לריצה על חומרת המחשב. השלבים הבאים הם קישור (linkage) וטעינה (loading), אך בהם לא נעסוק בממ״ן זה.

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

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

<u>המחשב הדמיוני ושפת האסמבלי</u>

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

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

<u>״חומרה״:</u>

המחשב בפרויקט מורכב **ממעבד** (יע״מ), **אוגרים** (רגיסטרים), **וזיכרון** RAM. חלק מהזיכרון משמש כמחסנית (stack).

למעבד 32 אוגרים כלליים, בשמות: $0,$1,$2….$31. גודלו של כל אוגר הוא 32 סיביות. הסיבית הכי פחות משמעותית תצוין כסיבית מס׳ 0, והסיבית המשמעותית ביותר כמס׳ 31.

גודל הזיכרון הוא 2^{25} תאים, בכתובות $0-(2^{25}-1)$, וכל תא הוא בגודל של 8 סיביות (Byte). כתובת בזיכרון ניתנת לייצוג ב-25 סיביות (כמספר ללא סימן).

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

<u>הוראות המכונה:</u>

למחשב זה מגוון הוראות, כל הוראה מורכבת מפעולה ואופרנדים. מספר האופרנדים תלוי בסוג ההוראה.

כל הוראה מקודדת בקוד המכונה ל-32 סיביות. הסיבית הפחות משמעותית של קידוד ההוראה תצוין כסיבית מס׳ 0, והסיבית המשמעותית ביותר כמס׳ 31.

הוראה תופסת בזיכרון המחשב ארבעה בתים רצופים, ומאוחסנת בשיטת little-endian. כלומר, סיביות 0-7 של ההוראה נמצאות בבית שכתובתו הנמוכה ביותר, סיביות 8-15 הן הבית הבא, לאחרי מכן סיביות 16-23, ואילו סיביות 24-31 נמצאות בכתובת הגבוהה ביותר. כשמתייחסים לכתובת של הוראה בזיכרון, זו תמיד כתובת הבית <u>הנמוך</u> ביותר.

לכל הוראה יש קוד-פעולה, הנקרא גם opcode, שמזהה את הפעולה שמבצעת ההוראה. לחלק מההוראות יש קוד זיהוי משני, שנקרא funct.

ההוראות נחלקות ל-3 סוגים: הוראות מסוג R, הוראות מסוג I, והוראות מסוג J.
להלן טבלה המפרטת את כל ההוראות, לפי סוגן, ועם הקודים המזהים שלהן:

opcode (בבסיס עשרוני)	**funct (בבסיס עשרוני)**	**סוג הפעולה**	**שם הפעולה**
0	1	R	add
0	2	R	sub
0	3	R	and
0	4	R	or
0	5	R	nor
1	1	R	move
1	2	R	mvhi
1	3	R	mvlo
10		I	addi
11		I	subi
12		I	andi
13		I	ori
14		I	nori
15		I	bne
16		I	beq
17		I	blt
18		I	bgt
19		I	lb
20		I	sb
21		I	lw
22		I	sw
23		I	lh
24		I	sh
30		J	jmp
31		J	la
32		J	call
63		J	stop

<u>הערה</u>: בתחביר ההוראה בשפת אסמבלי, שם-הפעולה נכתב תמיד באותיות <u>קטנות</u>.

<u>מבנה הוראות המכונה</u>:

נדון עתה ביתר פרוט במבנה התחבירי של כל הוראה בשפת האסמבלי, ובאופן הקידוד של ההוראה בקוד המכונה.

הוראות מסוג R:

הוראה מסוג R מקודדת באופן הבא:

opcode	rs	rt	rd	funct	לא בשימוש
31 26	25 21	20 16	15 11	10 6	5 0

בהוראות מסוג R כל האופרנדים הם אוגרים. לחלק מההוראות יש שני אופרנדים, ולחלקם שלשה אופרנדים.

קבוצת ההוראות מסוג R כוללת את ההוראות הבאות:

- **הוראות אריתמטיות ולוגיות מסג R:** add, sub, and, or, nor
- **הוראות העתקה:** move, mvhi, mvlo

הוראות אריתמטיות ולוגיות מסוג R:

להוראות add, sub, and, or, nor יש שלשה אופרנדים, ושלשתם אוגרים. ההוראות מבצעות פעולה אריתמטית או לוגית בין שני האוגרים rs ו- rt, והתוצאה מאוחסנת באוגר השלישי rd.
לכל ההוראות האריתמטיות/לוגיות מסוג R יש קוד-פעולה זהה והוא 0 (כלומר שדה ה-opcode מכיל 0). על מנת להבחין בין ההוראות השונות, קיים שדה נוסף בשם funct שהוא ייחודי לכל הוראה (ראו טבלת ההוראות לעיל).

- ההוראה add מבצעת פעולת חיבור, כלומר rd=rs+rt
- ההוראה sub מבצעת פעולת חיסור, כלומר rd=rs-rt
- ההוראה and מבצעת פעולת and בין הסיביות של האופרנדים, כלומר rd= rs & rt
- ההוראה or מבצעת פעולת or בין הסיביות של האופרנדים, כלומר rd= rs | rt
- ההוראה nor מבצעת פעולת nor בין הסיביות של האופרנדים, כלומר rd= ~(rs | rt)

נדגים את השימוש בהוראות, ואת אופן קידודן:

```
add $3, $19, $8
```

הוראה זו מבצעת חיבור של תוכן האוגר $3 עם תוכן האוגר $19 ושומרת את התוצאה באוגר $8.

בקידוד הוראה זו, שדה ה- opcode יכיל את קוד הפעולה 0 (המשותף לכל ההוראות מסוג R) ובשדה ה- funct את הקוד הייחודי ל- add. שדה rs יכיל את מספרו של האוגר המשמש כמחובר הראשון (בדוגמה זו 3), שדה rt יכיל את מספרו של האוגר המשמש כמחובר השני (בדוגמה זו 19), ושדה rd יכיל את מספרו של אוגר התוצאה (בדוגמה זו 8). הסיביות 5-0 בקידוד ההוראה אינן בשימוש, ויכילו אפסים.

באופן דומה ל- add, משתמשים גם בהוראות: sub, and, or, nor.
נדגיש שוב שבהוראה sub, האוגר rs הוא המחוסר, והאוגר rt הוא המחסר.
הערה: אין מניעה שאותו אוגר ישמש ביותר מאופרנד אחד (למשל: or $1,$2,$1).

הוראות העתקה:

להוראות move, mvhi, mvlo יש שני אופרנדים, ושניהם אוגרים. ההוראות מעתיקות (משכפלות) תוכן של אוגר אחד (אוגר המקור) לאוגר שני (אוגר היעד).

לכל ההוראות ההעתקה מסוג R יש קוד-פעולה זהה והוא 1 (כלומר שדה ה-opcode מכיל 1). על מנת להבחין בין ההוראות השונות, קיים שדה funct ייחודי לכל אחת מהן (ראו טבלת ההוראות לעיל).

- ההוראה move מעתיקה את תוכנו של אוגר rs אל אוגר rd
- ההוראה mvhi מעתיקה את חציו הגבוה (סיביות 31-16) של אוגר rs אל חציו הנמוך של אוגר rd
- ההוראה mvlo מעתיקה את חציו הנמוך (סיביות 15-0) של אוגר rs אל חציו הגבוה של אוגר rd

נדגים את השימוש בהוראות, ואת אופן קידודן:

```
move $23, $2
```

הוראה זו מעתיקה אל האוגר $23 את תוכנו של אוגר $2.

בקידוד הוראה זו, שדה ה- opcode יכיל את קוד הפעולה 0 (המשותף לכל ההוראות מסג R), ושדה ה- funct את הקוד הייחודי ל- move. השדה rs יכיל את מספרו של אוגר המקור של ההעתקה (בדוגמה זו 2), והשדה rd יכיל את מספרו של אוגר היעד (בדוגמה זו 23). השדה rt, וכן הסיביות 0-5 אינם בשימוש ויכילו אפסים.

באופן דומה ל- move, משתמשים גם בהוראות mvhi, mvlo .

הוראות מסוג I:

הוראה מסוג I מקודדת באופן הבא:

opcode	rs	rt	immed
31 26	25 21	20 16	15 0

בהוראות מסוג I אחד האופרנדים הוא ערך קבוע, הנקרא גם ערך מיידי (immediate). הקבוע הוא מספר שלם, הניתן לייצוג ברוחב 16 סיביות בשיטת המשלים ל-2. סיבית 0 של קידוד ההוראה היא הסיבית הפחות משמעותית של הקבוע.

לכל אחת מההוראות מסוג I יש opcode ייחודי (ראו טבלת ההוראות לעיל).

קבוצת ההוראות מסוג I כוללת את ההוראות הבאות:

- **הוראות אריתמטיות ולוגיות:** addi, subi, andi, ori, nori
- **הוראות הסתעפות מותנית:** beq, bne, blt, bgt
- **הוראות טעינה ושמירה בזיכרון:** lb, sb, lw, sw, lh, sh

הוראות אריתמטיות ולוגיות מסוג I:

ההוראות addi, subi, andi, ori, nori מבצעות פעולה אריתמטית או לוגית בין האוגר rs לבין הערך המיידי immed והתוצאה מאוחסנת באוגר rt. בעת ביצוע ההוראה, המעבד מרחיב את הערך immed ל-32 סיביות, כך שהפעולה מתבצעת על אופרנדים ברוחב 32 סיביות.

- ההוראה addi מבצעת פעולת חיבור, כלומר rt=rs+immed
- ההוראה subi מבצעת פעולת חיסור, כלומר rt=rs-immed
- ההוראה andi מבצעת פעולת and בין הסיביות של האופרנדים, כלומר rt= rs & immed
- ההוראה ori מבצעת פעולת or בין הסיביות של האופרנדים, כלומר rt= rs | immed

- ההוראה nori מבצעת פעולת nor בין הסיביות של האופרנדים, כלומר rt= ~(rs | immed)

נדגים את השימוש בהוראות, ואת אופן קידודן:

```
addi $9, -45, $8
```

הוראה זו מבצעת חיבור של תוכן האוגר $9 עם הערך המיידי 45- ושומרת את התוצאה באוגר $8.
<u>הערה</u>: נשים לב שיש הבדל בסדר האופרנדים בין ההוראה בשפת אסמבלי לבין הקידוד הבינארי.

בקידוד הוראה זו, שדה ה- opcode יכיל את קוד הפעולה של addi, שדה rs יכיל את מספרו של האוגר שהוא המחובר הראשון (בדוגמה זו 9), השדה rt יכיל את מספרו של אוגר התוצאה (בדוגמה זו 8), והשדה immed יכיל את הקבוע שהוא המחובר השני (בדוגמה זו המספר 45-).

באופן דומה ל- addi, משתמשים גם בהוראות: subi, andi, ori, nori.
נגדיש שוב שבהוראה subi, המחוסר הוא האוגר rs, והמחסר הוא הערך המיידי.
אין מניעה שאותו אוגר ישמש בשני האופרנדים rs, rt.

<u>הוראות הסתעפות מותנית:</u>

ההוראות beq, bne, blt, bgt משוות בין תכני האוגרים rs ו-rt, ובודקות האם תוצאת ההשוואה מתאימה לתנאי הנתון (לפי קוד הפעולה), ואם כן, מתבצעת קפיצה להוראה בכתובת יעד שמצויינת ע"י התווית label.

יעד הקפיצה מקודד בשדה immed, באמצעות המרחק אל הכתובת label מהכתובת הנוכחית (כתובת הוראת ההסתעפות). המרחק יכול להיות חיובי או שלילי, אך אינו יכול לחרוג מגבולות של מספר ברוחב 16 סיביות בשיטת המשלים ל-2.

התווית שמציינת את יעד הקפיצה חייבת להיות מוגדרת בקובץ המקור <u>הנוכחי</u> (כלומר זו אינה תווית חיצונית).

- ההוראה beq בודקת האם תוכן האוגר rs שווה לתוכן האוגר rt, ואם כן אז מתבצעת הקפיצה
- ההוראה bne בודקת האם תוכן האוגר rs שונה מתוכן האוגר rt, ואם כן אז מתבצעת הקפיצה
- ההוראה blt בודקת האם תוכן האוגר rs קטן מתוכן האוגר rt, ואם כן אז מתבצעת הקפיצה
- ההוראה bgt בודקת האם תוכן האוגר rs גדול מתוכן האוגר rt, ואן כן אז מתבצעת הקפיצה

נדגים את השימוש בהוראות, ואת אופן קידודן:

```
blt $5, $24, loop
```

הוראה זו בודקת האם תוכנו של אוגר $5 קטן מתוכנו של אוגר $24, ואם כן, ההוראה הבאה שתתבצע תהיה בכתובת loop, ואחרת ההוראה הבאה תמשיך להיות ההוראה העוקבת.

בקידוד הוראה זו, שדה ה- opcode יכיל את קוד הפעולה של blt, השדה rs יכיל את מספרו של האוגר המשמש כאופרנד השמאלי של פעולת ההשוואה (בדוגמה זו 5), והשדה rt יכיל את מספרו של האוגר המשמש כאופרנד הימני (בדוגמה זו 24).

השדה immed יכיל את המרחק מכתובת ההוראה הנוכחית (blt) אל ההוראה בכתובת loop. את המרחק הזה על האסמבלר לחשב בעצמו. למשל, נניח שההוראה blt הנוכחית נמצאת בכתובת 300, והכתובת loop היא 200. השדה immed יקודד לערך 100-.

באופן דומה ל- blt, משתמשים גם בהוראות bne, beq, bgt

<u>הוראות טעינה ושמירה בזיכרון:</u>

ההוראות lb, sb, lw, sw, lh, sh טוענות לאוגר ערך (שלם) שנמצא בזיכרון, או שומרות בזיכרון ערך שנמצא באוגר (תלוי בהוראה). גודלו של הערך שייטען/יישמר יכול להיות: byte יחיד, או word (מילה = 4 bytes), או half-word (חצי מילה = 2 bytes).

כל ההוראות האלה ניגשות לכתובת בזיכרון שהיא הסכום של תוכן האוגר rs והערך immed. הקבוע immed משמש כאן בתפקיד של offset (היסט) מכתובת בסיס שהיא האוגר rs. ההיסט רשום בתכנית בשפת אסמבלי כקבוע, ויכול להיות חיובי או שלילי (במגבלות תחום הערכים ברוחב 16 סיביות בשיטת המשלים ל-2).

- ההוראה lb טוענת מהזיכרון ערך בגודל byte, אל 8 הסיביות הנמוכות (סיביות 0-7) של אוגר rt. rt=Mem[rs+immed]
- ההוראה lw טוענת מהזיכרון ערך בגודל word, אל אוגר rt. rt=Mem[rs+immed]
- ההוראה lh טוענת מהזיכרון בגודל half-word, אל 16 הסיביות הנמוכות (סיביות 0-15) של אוגר rt. rt=Mem[rs+immed]
- ההוראה sb שומרת בזיכרון את 8 הסיביות הנמוכות של האוגר rt. Mem[rs+immed]=rt
- ההוראה sw שומרת בזיכרון את תוכן האוגר rt. Mem[rs+immed]=rt
- ההוראה sh שומרת בזיכרון את 16 הסיביות הנמוכות של האוגר rt. Mem[rs+immed]=rt

הזיכרון במחשב הדמיוני שלנו מאורגן בשיטת little-endian. ערך (שלם) שגודלו יותר מבית (תא זיכרון) אחד, מאוחסן בזיכרון כך שהבית הפחות משמעותי של הערך מאוחסן בכתובת הנמוכה ביותר ברצף הבתים שתופס הערך, ואחריו שאר הבתים בסדר עולה של משמעות. כשמתייחסים לכתובת של ערך בזיכרון, זו תמיד כתובת הבית <u>הנמוך</u> ביותר.

נדגים את השימוש בהוראות טעינה ושמירה, ואת אופן קידודן:

```
lh  $9, 34, $2
```

הוראה זו טוענת אל האוגר $2 שני בתים (חצי-מילה) מהזיכרון, החל מכתובת שמתקבלת מחיבור האוגר $9 והקבוע 34.

בקידוד הוראה זו, שדה ה-opcode יכיל את קוד הפעולה של lh, השדה rs יכיל את מספר האוגר המשמש לחישוב הכתובת (בדוגמה זו 9), האוגר rt יכיל את מספר האוגר אליו ייטען הערך (בדוגמה זו 2), והשדה immed יכיל את ההיסט (בדוגמה זו 34).
<u>הערה</u>: נשים לב שיש הבדל בסדר האופרנדים, בין תחביר ההוראה בשפת אסמבלי לבין הקידוד הבינארי.

באופן דומה ל- lh, משתמשים גם בהוראות lb, lw

דוגמה נוספת:

```
sw $7, -28, $18
```

הוראה זו שומרת את תוכן האוגר $18 בזיכרון, החל בכתובת שמתקבלת מחיבור של האוגר $18 והקבוע 28-.

בקידוד הוראה זו, שדה ה- opcode יכיל את קוד ההוראה של sw, השדה rs יכיל את מספר האוגר המשמש לחישוב הכתובת (בדוגמה זו $7), השדה rt יכיל את מספר האוגר שיישמר בזיכרון (בדוגמה זו $18), והשדה immed יכיל את ההיסט (בדוגמה זו 28-).
<u>הערה</u>: נשים לב שיש הבדל בסדר האופרנדים, בין תחביר ההוראה בשפת אסמבלי לבין הקידוד הבינארי.

באופן דומה ל- sw, משתמשים גם בהוראות sb, sh

הוראות מסוג J:

הוראה מסוג J מקודדת באופן הבא:

opcode	reg	address
31 26	25	24 0

לכל אחת מההוראות מסוג J יש opcode ייחודי (ראו טבלת ההוראות לעיל).

קיימות ארבע הוראות מסוג זה, והן: jmp, la, call, stop

הוראת jmp:
הוראה שגורמת לקפיצה למקום אחר בתוכנית לשם המשך הריצה. תחביר ההוראה jmp בשפת האסמבלי יכול להיכתב באחת משתי הצורות הבאות:

```
jmp  label
jmp  $register
```

בצורה הראשונה, מתבצעת קפיצה להוראה הנמצאת בכתובת שמצויינת ע"י התווית label. נדגיש שהתווית יכולה להיות מוגדרת בקובץ המקור הנוכחי, או בקובץ מקור אחר (תווית חיצונית).

נדגים את השימוש בצורת תחביר זו. נניח שקיימת בתכנית הוראה עם תווית main וברצוננו לקפוץ אליה, אזי נכתוב:

```
jmp main
```

בקידוד הוראה זו לשפת מכונה, נציב בשדה ה- opcode את קוד הפעולה של jmp (ראו טבלת הפעולות לעיל). בשדה reg נציב 0, כדי לציין שיעד הקפיצה נתון באמצעות תווית. אם התווית main מוגדרת בקובץ המקור הנוכחי (אינה חיצונית) אזי נציב בשדה ה-address את כתובתה (סיבית 0 בקידוד היא הסיבית הפחות משמעותית של הכתובת). אם התווית היא חיצונית, אזי נציב בשדה ה- address אפסים, כי הכתובת האמיתית לא ידועה לאסמבלר (קידוד השדה יושלם בשלב הקישור של תהליך בניית התכנית, שאותו **אינכם נדרשים לממש).**

בצורת התחביר השניה של הוראת jmp מתבצעת קפיצה לכתובת הנמצאת באוגר, שמספרו מקודד בשדה address (כלומר address יכיל מספר בין 0-31 ברוחב 25 סיביות).
הערה: מרחב הכתובות במחשב הדמיוני הוא התחום $[0..2^{25}-1]$, והאוגר אמור להכיל ערך שאינו חורג מתחום זה.

נדגים את השימוש בצורת תחביר זו. נניח שאוגר $7 מכיל כתובת שאליה נרצה לקפוץ, אזי נכתוב:

```
jmp $7
```

בקידוד הוראה זו לשפת מכונה, נציב בשדה ה- opcode את קוד הפעולה של jmp. בשדה reg נציב 1, כדי לציין שיעד הקפיצה נתון באמצעות אוגר. בשדה address נציב את מספר האוגר (בדוגמה זו 7). למעשה, סיביות 0-4 יכילו את מספר האוגר, וסיביות 5-24 יכילו אפסים.

הוראת la:
הוראת la (load address) טוענת לאוגר $0 כתובת של תווית מסוימת. נדגיש שהתווית יכולה להיות מוגדרת בקובץ המקור הנוכחי, או בקובץ מקור אחר (תווית חיצונית).

תחביר ההוראה la בשפת האסמבלי הוא:

```
la  label
```

label מציין את התווית שאת כתובתה מעוניינים לטעון לאוגר $0.
<u>הערה</u>: לא ניתן בהוראה la להשתמש באוגר אחר מאשר $0.

נדגים את השימוש בהוראה זו. נניח שבזיכרון מוגדר משתנה בשם num1 (משתנה שנמצא בכתובת שהתווית שלה num1), ואנו מעוניינים לטעון את כתובת המשתנה לאוגר $0, אזי נכתוב:

```
la num1
```

בקידוד הוראה זו לשפת מכונה, נציב בשדה ה- opcode את קוד הפעולה של la. השדה reg אינו בשימוש בהוראה זו, ולכן יוצב בו תמיד 0. אם התווית num1 מוגדרת בקבץ המקור (אינה חיצונית) אזי נציב בשדה ה-address את כתובתה. אם התווית היא חיצונית, אזי נציב בשדה ה-address אפסים (עם אותה משמעות כמו בהוראת jmp).

<u>הוראת call:</u>
הוראה שגורמת לקפיצה למקום אחר בתכנית לשם המשך הריצה, ושומרת את כתובת ההוראה הבאה (ההוראה העוקבת אחרי call). ההוראה משמשת בעיקר לקריאה לשגרה. תחביר הוראה זו בשפת האסמבלי הוא:

```
call label
```

מתבצעת קפיצה להוראה הנמצאת בכתובת שמצויינת ע"י התווית label, ואילו כתובת ההוראה העוקבת ל- call נשמרת באוגר $0. נדגיש שהתווית יכולה להיות מוגדרת בקובץ המקור הנוכחי, או בקובץ מקור אחר (תווית חיצונית).

<u>הערה</u>: לא ניתן בהוראה call להשתמש באוגר אחר מאשר $0.

נדגים את השימוש בהוראה זו. נניח שקיימת בתכנית הוראה עם תווית myfunc, שהיא ההוראה הראשונה של שיגרה, וברצוננו לקרוא לשגרה. אזי נכתוב:

```
call myfunc
```

בקידוד הוראה זו לשפת מכונה, נציב בשדה ה- opcode את קוד הפעולה של call. השדה reg אינו בשימוש בהוראה זו, ולכן יוצב בו תמיד 0. אם התווית myfunc מוגדרת בקבץ המקור (אינה חיצונית) אזי נציב בשדה ה-address את כתובתה. אם התווית היא חיצונית, אזי נציב בשדה ה-address אפסים (עם אותה משמעות כמו בהוראת jmp).

כאמור, כתובת ההוראה שאחרי call (כתובת החזרה מהשגרה) נשמרת באוגר $0. כדי לחזור מהשגרה, יש להשתמש בהוראה jmp בצורתה השניה (ראה לעיל)

<u>הוראת stop:</u>
להוראה זו אין אופרנדים והיא נועדה לעצור את התוכנית. בקידוד הוראה זו לשפת מכונה, נציב בשדה ה- opcode את קוד הפעולה של stop, ובכל יתר השדות יוצבו אפסים.

<u>מבנה תכנית בשפת אסמבלי:</u>

תכנית בשפת אסמבלי בנויה ממשפטים (statements). קובץ מקור בשפת אסמבלי מורכב משורות המכילות משפטים של השפה, כאשר כל משפט מופיע <u>בשורה נפרדת</u>. כלומר, ההפרדה בין משפט למשפט בקובץ המקור הינה באמצעות התו '\n' (שורה חדשה).

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

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

סוג המשפט	הסבר כללי
משפט ריק	זוהי שורה המכילה אך ורק תווים לבנים (whitespace), כלומר רק רווחים וטאבים. ייתכן ובשורה אין אף תו (למעט התו '\n'), כלומר השורה ריקה.
משפט הערה	זוהי שורה בה התו הראשון שאינו תו לבן הינו ';' (נקודה-פסיק). על האסמבלר להתעלם לחלוטין משורה זו.
משפט הנחיה	זהו משפט המנחה את האסמבלר מה עליו לעשות כשהוא פועל על תוכנית המקור. יש מספר סוגים של משפטי הנחיה. משפט הנחיה עשוי לגרום להקצאת 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 משם ההוראה (פעולה) שעל המעבד לבצע, והאופרנדים של ההוראה.

כעת נפרט יותר לגבי סוגי המשפטים השונים.

<u>משפט הנחיה:</u>

משפט הנחיה הוא בעל המבנה הבא:

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

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

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

יש כמה סוגים (שמות) של משפטי הנחיה, והם:

1. משפטי ההנחיה '.db', '.dw', '.dd'

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

```
.db 7, −57, 17, +9
.dw 120056
.dh 0, −60431, 1700, 3, −1
```

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

משפטי הנחיה אלה מנחים את האסמבלר להקצות מקום בתמונת הנתונים (data image), שבו יאוחסנו הערכים של הפרמטרים, ולקדם את מונה הנתונים, בהתאם למספר הערכים ובהתאם לגודלם. כל נתון המוגדר ע"י משפט הנחיה '.db' תופס בית (byte) בודד. נתון המוגדר ע"י משפט הנחיה '.dw' תופס ארבעה בתים (מילה). נתון המוגדר ע"י '.dh' תופס שני בתים (חצי-מילה).

<u>הערה</u>: נשים לב שכל מספר חייב להיות בגודל מתאים, שלא יחרוג מגבולות הייצוג של סוג הנתון המוגדר.

אם בהנחית מוגדרת תווית, אזי תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום), ומוכנסת אל טבלת הסמלים. דבר זה מאפשר להתייחס אל מקום מסוים בתמונת הנתונים דרך שם התווית (למעשה, זוהי דרך להגדיר שם של משתנה).

לדוגמה, אם נכתוב:

```
XYZ:   .dh 0, −60431, 1700, 3, −1
```

אזי יוקצו בתמונת הנתונים 10 בתים (bytes) רצופים, וכל שני בתים (חצי-מילה) מאותחלים עם אחד המספרים שמופיעים בהנחיה (לפי הסדר). התווית XYZ מזוהה עם כתובת הבית הראשון.
אפשר להתיחס ל-XYZ כמשתנה שהוא מערך של 5 איברים בגודל חצי-מילה.

2. ההנחיה '.asciz'

להנחיה '.asciz' פרמטר אחד, שהוא מחרוזת חוקית. תווי המחרוזת מקודדים לפי ערכי ה-ascii המתאימים, ומוכנסים אל תמונת הנתונים לפי סדרם, כל תו ב-byte נפרד. בסוף המחרוזת יתווסף התן '\0' (הערך המספרי 0), המסמן את סוף המחרוזת. מונה הנתונים של האסמבלר יקודם בהתאם לאורך המחרוזת (בתוספת מקום אחד עבור התו המסיים). אם בשורת ההנחיה מוגדרת תווית, אזי תווית זו מקבלת את ערך מונה הנתונים (לפני הקידום) ומוכנסת אל טבלת הסמלים.

לדוגמה, אם נכתוב:

```
STR:   .asciz "hello world"
```

אזי יוקצו בתמונת הנתונים 12 בתים רצופים, שיאותחלו לתווי המחרוזת לעיל (כולל האפס המסיים). התווית STR מזוהה עם כתובת התו הראשון.
אפשר להתיחס ל- STR כמשתנה שהוא מחרוזת תווים באורך (מקסימלי) 11.

3. ההנחיה '.entry'

להנחיה '.entry' פרמטר אחד, והוא שם של תווית המוגדרת בקובץ המקור הנוכחי (כלומר תווית שמקבלת את ערכה בקובץ זה). מטרת ההנחיה entry. היא לאפיין את התווית הזו באופן שיאפשר לקוד אסמבלי הנמצא בקבצי מקור אחרים להשתמש בה (כאופרנד של הוראה).

לדוגמה, השורות:

```
        .entry  HELLO
HELLO:  add $1,$2,$5
```

מודיעות לאסמבלר שייתכן ותהיה התיחסות בקובץ מקור אחר לתווית HELLO המוגדרת בקובץ הנוכחי.

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

4. ההנחיה '.extern'

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

נשים לב כי הנחיה זו תואמת להנחיית 'entry.' המופיעה בקובץ בו מוגדרת התווית. בשלב הקישור תתבצע התאמה בין ערך התווית, כפי שנקבע בקוד המכונה של הקובץ שהגדיר את התווית, לבין קידוד ההוראות המשתמשות בתווית בקבצים אחרים (שלב הקישור אינו רלוונטי לממ"ן זה).

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

.extern HELLO

<u>לתשומת לב</u>: <u>לא ניתן</u> להגדיר באותו הקובץ את אותה התווית גם כ-entry וגם כ-extern (בדוגמאות לעיל, התווית HELLO). כל הרעיון הוא לאפשר להוראה בקובץ אחד להשתמש בנתונים מקובץ אחר, או לקפוץ להוראה בקובץ אחר.

<u>לתשומת לב</u>: תווית המוגדרת בתחילת שורת extern. הינה חסרת משמעות והאסמבלר **מתעלם** מתווית זו (אפשר שהאסמבלר יוציא הודעת אזהרה).

<u>משפט הוראה</u>:

משפט הוראה מורכב מהחלקים הבאים:

1. תווית <u>אופציונלית</u>.
2. שם הפעולה.
3. אופרנדים, בהתאם לסוג הפעולה (כפי שהוסבר לכל פעולה לעיל)

אם מוגדרת תווית בשורת ההוראה, אזי היא תוכנס אל טבלת הסמלים. ערך התווית יהיה מען ה-byte הראשון של ההוראה בתוך תמונת הקוד שבונה האסמבלר.

שם הפעולה ייכתב תמיד באותיות קטנות (lower case). לאחר שם הפעולה יופיעו האופרנדים, בהתאם לסוג הפעולה. יש להפריד בין שם-הפעולה לבין האופרנד הראשון באמצעות רווחים ו/או טאבים (אחד או יותר).

כאשר יש מספר אופרנדים, האופרנדים מופרדים זה מזה בתו ',' (פסיק). **לא חייבת להיות הצמדה של האופרנדים לפסיק**. כל כמות של רווחים ו/או טאבים משני צידי הפסיק היא חוקית.

<u>אפיון השדות במשפטים של שפת האסמבלי</u>

<u>תווית</u>:

תווית היא סמל שמוגדר בתחילת משפט הוראה או בתחילת משפט הנחיה המגדיר משתנים. 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.

<u>הגדרה של תווית</u> מסתיימת בתו ':' (נקודתיים). תו זה אינו מהווה חלק מהתווית, אלא רק סימן המציין את סוף ההגדרה. התו ':' חייב להיות צמוד לתווית (ללא רווחים).

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

לדוגמה, התוויות המוגדרות להלן הן תוויות חוקיות.

hEllo:
x:
He78902:

לתשומת לב: מילים שמורות של שפת האסמבלי (כלומר שם של פעולה או הנחיה) אינן יכולות לשמש גם כשם של תווית. לדוגמה: הסמלים add, asciz לא יכולים לשמש כתוויות, אבל הסמלים Add, ASCiz הן תוויות חוקיות.

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

<u>לתשומת לב</u>: מותר במשפט הוראה להשתמש באופרנד שהוא תווית המוגדרת בשורה יותר מאוחרת בקובץ המקור. כמו כן, מותר במשפט הוראה להשתמש באופרנד שהוא סמל שאינו מוגדר כתווית בקובץ הנוכחי, כל עוד הסמל מאופיין כחיצוני (באמצעות הנחיית extern. בקובץ הנוכחי). נושא זה יובהר בהמשך, בדיון על מימוש האסמבלר.

<u>מספר</u>:

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

<u>מחרוזת</u>:

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

אסמבלר עם שני מעברים

האסמבלר מקבל כקלט קובץ המכיל תוכנית בשפת אסמבלי, ותפקידו להמיר את קוד המקור לקוד מכונה, ולבנות קובץ פלט המכיל את קוד המכונה.

<u>הערה</u>: האסמבלר לא מבצע את התכנית, אלא רק מתרגם אותה לבינארי. לפיכך, על קוד המקור להיות נכון מבחינה תחבירית, אבל זה לא משנה לאסמבלר (ולפרויקט שלנו) מה התכנית אמורה לעשות והאם היא בכלל תעבוד נכון.
לדוגמה: האסמבלר מקבל את התוכנית הבאה בשפת אסמבלי:

```
MAIN:  add     $3,$5,$9
LOOP:  ori     $9,-5,$2
       la      val1
       jmp     Next
Next:  move    $20,$4
       bgt     $4,$2,END
       la      K
       sw      $0,4.$10
       bne     $31,$9,LOOP
       call    val1
       jmp     $4
END:   stop
STR:   .asciz  "aBcd"
LIST:  .db     6,-9
       .dh     27056
.entry K
K:     .dw     31,-12
.extern val1
```

התרגום של תוכנית המקור שבדוגמה לקוד מכונה מוצג להלן. לכל שורה בקוד המקור, מוצג הקידוד הבינארי של ההוראה או של הנתונים, בצירוף הכתובת בזיכרון לשם ייטען הקוד לצורך הרצה.

האסמבלר בונה את קוד המכונה של התוכנית כך שיתאים לטעינה בזיכרון **החל מכתובת 100 (עשרוני)**.

שימו לב לקפיצות ב-4 בעמודת הכתובת בכל פעם שמקודדים הוראה, מאחר והקידוד מורכב מ- 32 סיביות, שתופסים 4 תאי זיכרון רצופים.

Address (decimal)	**Source Code**			**Machine Code (binary)**
0100	MAIN:	add	$3,$5,$9	000000 00011 00101 01001 00001 000000
0104	LOOP:	ori	$9,-5,$2	001101 01001 00010 1111111111111011
0108		la	val1	011111 0 00000000000000000000000000
0112		jmp	Next	011110 0 00000000000000000001110100
0116	Next:	move	$20,$4	000001 10100 00000 00100 00001 000000
0120		bgt	$4,$2,END	010010 00100 00010 0000000000011000
0124		la	K	011111 0 00000000000000000010011101
0128		sw	$0,4,$10	010110 00000 01010 0000000000000100
0132		bne	$31,$9,LOOP	001111 11111 01001 1111111111100100
0136		call	val1	100000 0 00000000000000000000000000
0140		jmp	$4	011110 1 00000000000000000000000100
0144	END:	stop		111111 0 00000000000000000000000000
0148	STR:	.asciz	"aBcd"	01100001
0149				01000010
0150				01100011
0151				01100100
0152				00000000
0153	LIST:	.db	6,-9	00000110
0154				11110111
0155		.dh	27056	0110100110110000
0157	K:	.dw	31,-12	00000000000000000000000000011111
0161				11111111111111111111111111110100

נתאר עתה כיצד האסמבלר יבצע את התרגום לקוד מכונה באופן שיטתי.

האסמבלר מחזיק טבלה שבה רשומים כל שמות הפעולה של ההוראות והקודים הבינאריים (opcode, funct) המתאימים להם, בדומה לטבלה שהצגנו קודם. לכן קל לתרגם לבינארי את שם הפעולה: כשנקראת שורת הוראה מקובץ המקור, מחפשים בטבלה את הקוד הבינארי המתאים.

גם אופרנדים שהם אוגרים או קבועים מיידיים, אפשר לקודד ישירות לבינארי. כך גם לגבי נתונים שמוגדרים ומאותחלים באמצעות שורות הנחיה.

הקושי העיקרי הוא לקודד לבינארי אופרנדים המשתמשים בסמלים (תוויות). על האסמבלר לדעת מה הערך הנומרי (הכתובת) שמייצג כל סמל. אולם בהבדל מהקודים של שמות הפעולה, שהם ערכים קבועים וידועים, הרי הכתובות בזיכרון של הסמלים שבשימוש התוכנית לא ידועות, עד אשר תוכנית המקור נסרקה כולה ונתגלו כל הגדרות הסמלים.

למשל, בקוד המקור לעיל, האסמבלר אינו יכול לדעת שהסמל END אמור להיות משויך לכתובת 144 (עשרוני), ושהסמל K אמור להיות משויך לכתובת 157, אלא רק לאחר שנקראו כל שורות התוכנית.

לכן מפרידים את הטיפול של האסמבלר לשני שלבים. בשלב הראשון מזהים את כל הסמלים שבתכנית המקור וקובעים מהם הערכים המספריים המשויכים להם. בשלב השני מחליפים את הסמלים שבאופרנדים של הוראות התוכנית בערכיהם המספריים, וכך ניתן לקודד את האופרנדים לבינארי.

הביצוע של שני שלבים אלה כרוך בשתי סריקות (הנקראות **"מעברים"**) של השורות בתכנית המקור.
במעבר הראשון, האסמבלר בונה טבלת סמלים (symbol table), ומכניס לתוכה כל סמל שמוגדר בתוכנית המקור, יחד עם הערך המספרי (המען בזיכרון) המשויך לו.
במעבר השני, האסמבלר משתמש בטבלת הסמלים כדי להמיר את השורות בקוד המקור לקוד מכונה. נשים לב שבתחילת המעבר השני צריכים הערכים של כל הסמלים להיות כבר ידועים. אם יש סמל שנעשה בו שימוש באופרנד, אבל הסמל לא נמצא בטבלת הסמלים, לא ניתן להשלים את הקידוד.

עבור הדוגמה שהצגנו, 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 בתכנית המקור).

Symbol	**Value (decimal)**	**Attributes**
MAIN	100	code
LOOP	104	code
Next	116	code
END	144	code
STR	148	data
LIST	153	data
K	157	data, entry
val1	0	external

לתשומת לב: כאמור, תפקיד האסמבלר, על שני המעברים שלו, לתרגם קוד מקור לקוד בשפת מכונה. בגמר פעולת האסמבלר, התוכנית טרם מוכנה לטעינה לזיכרון לצורך ביצוע. קוד המכונה חייב לעבור לשלבי הקישור/טעינה, ורק לאחר מכן לשלב הביצוע (שלבים אלה אינם חלק מהממ"ן).

המעבר הראשון

במעבר הראשון נדרשים כללים כדי לקבוע איזה מען ישויך לכל סמל. העיקרון הבסיסי הוא לספור את המקומות בזיכרון, אותם תופסות ההוראות. אם כל הוראה תיטען בזיכרון למקום העוקב להוראה הקודמת, תציין ספירה כזאת את מען ההוראה הבאה. הספירה נעשית על ידי האסמבלר ומוחזקת במונה ההוראות (IC) . ערכו ההתחלתי של IC הוא 100 (עשרוני), ולכן קוד המכונה של ההוראה הראשונה נבנה כך שייטען לזיכרון החל ממען 100. ה-IC מוגדל ב-4 אחרי כל שורת הוראה (כמספר התאים שתופס קידודה ההוראה). כך ה-IC מצביע תמיד על המקום הפנוי הבא בזיכרון.

כמו כן, באופן דומה, האסמבלר סופר את המקומות בזיכרון שתופסים הנתונים (המוגדרים באמצעות הנחיות). הספירה מוחזקת במונה הנתונים (DC) . (ראו פרטים נוספים בהמשך, בדיון על תהליך העבודה של האסמבלר).

כאמור, כדי לקודד את ההוראות בשפת מכונה, מחזיק האסמבלר טבלה, שיש בה לכל פעולה קידוד מתאים (opcode, ובחלק מהפעולות גם funct). בזמן התרגום לקוד מכונה, מחליף האסמבלר כל שם פעולה בקידוד שלה.

כאשר נתקל האסמבלר בתווית המופיעה בתחילת השורה, הוא יודע שלפניו הגדרה של תווית, ואז הוא משייך לה מען – ערכו הנוכחי של ה-IC או ה-DC. כך מקבלת כל תווית את מענה בשורה בה היא מוגדרת. התווית מוכנסת לטבלת הסמלים, המכילה בנוסף לשם התווית גם את המען

ומאפיינים נוספים. כאשר יש התייחסות לתווית באופרנד של הוראה כלשהי, יוכל האסמבלר לשלוף את המען המתאים מטבלת הסמלים.

הוראה יכולה להתייחס גם לסמל שטרם הוגדר עד כה בתוכנית, אלא יוגדר רק בהמשך התוכנית. לדוגמה, להלן הוראת קפיצה למען שמשוייך לתווית A, אך התווית מוגדרת רק בהמשך הקוד:

```
        jmp  A
        .
        .
        .
A:      ......
```

כאשר מגיע האסמבלר להוראת הקפיצה (jmp A), הוא טרם נתקל בהגדרת התווית A וכמובן לא יודע את המען המשויך לתווית. לכן האסמבלר לא יכול לבנות את הקידוד הבינארי של האופרנד A. ראו בהמשך כיצד נפתרת בעיה זו.

בכל מקרה, תמיד אפשר כבר במעבר הראשון לבנות באופן חלקי את קוד המכונה של ההוראה. למשל, אפשר לקודד את שדה ה-opcode, את שדה ה-funct (בהוראה בפורמט R), וכן את כל השדות המכילים מספרי אוגרים. בהוראות אריתמטיות/לוגיות בפורמט I אפשר לקודד במעבר ראשון גם את השדה immed.

המעבר השני

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

לשם כך מבצע האסמבלר מעבר נוסף (מעבר שני) על כל קובץ המקור, ובונה את קוד המכונה של כל אופרנד שהוא ערך של סמל, באמצעות ערכי הסמלים מטבלת הסמלים.

ספציפית, יש לעדכן את שדה ה-immed בהוראת הסתעפות מותנית מסוג I, ואת שדה הכתובת בהוראה מסוג J.

בסוף המעבר השני, תהיה התוכנית מתורגמת בשלמותה לקוד מכונה.

הפרדת הוראות ונתונים

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

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

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

גילוי שגיאות בתוכנית המקור

על האסמבלר לבצע ניתוח תחבירי (parsing) מדוקדק של תכנית המקור בשפת אסמבלי, ולהיות מסוגל לגלות ולדווח על מגוון שגיאות, כגון: שם-פעולה שאינו מוגדר, מספר אופרנדים שגוי, סוג אופרנד שאינו מתאים לפעולה, מספר אוגר שלא קיים, שימוש בסמל שאינו מוגדר, סמל שמוגדר יותר מפעם אחת, ערך מספרי בגודל חריג, ועוד שגיאות אחרות.

האסמבלר ידווח על השגיאות באמצעות הודעות אל הפלט הסטנדרטי stdout. כל שגיאה נגרמת (בדרך כלל) על ידי שורה מסוימת בקוד המקור. בכל הודעת שגיאה יש לציין גם את מספר השורה בה זוהתה השגיאה (מניין השורות בקובץ מתחיל ב-1).

דוגמה: אם בשורה מס' x בקובץ הקלט מופיעים שני אופרנדים בהוראה שאמור להיות בה רק אופרנד יחיד, האסמבלר ייתן הודעת שגיאה בנוסח "Line x: extraneous operand".

דוגמה: אם בשורה מס' y בקובץ הקלט מופיע שם לא מוכר של הנחיה, האסמבלר ייתן הודעת שגיאה בנוסח: "Line y: unrecognized directive <directive-name>"

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

תהליך העבודה של האסמבלר

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

האסמבלר מתחזק' שני מערכים, שייקראו להלן תמונת ההוראות (code) ותמונת הנתונים (data). מערכים אלו נותנים למעשה תמונה של זיכרון המכונה (כל איבר במערך הוא בגודל byte). במערך ההוראות בונה האסמבלר את הקידוד של הוראות המכונה שנקראו במהלך המעבר על קובץ המקור. במערך הנתונים מכניס האסמבלר את קידוד הנתונים שנקראו מקובץ המקור (כלומר הנתונים בשורות הנחיה מסוג .db, .dw, .dh).

האסמבלר משתמש בשני מונים, שנקראים IC (מונה ההוראות - Instruction-Counter), ו- DC (מונה הנתונים - Data-Counter). מונים אלו מצביעים על המקום הבא הפנוי במערך ההוראות ובמערך הנתונים, בהתאמה. בכל פעם כשמתחיל האסמבלר לעבור על קובץ מקור, המונה IC מקבל ערך התחלתי 100, והמונה DC מקבל ערך התחלתי 0. הערך ההתחלתי IC=100 נקבע כדי שקוד המכונה של התוכנית יתאים לטעינה לזיכרון (לצורך ריצה) החל מכתובת 100.

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

במעבר הראשון האסמבלר בונה את טבלת הסמלים, ואת השלד של תמונת הזיכרון (מערך ההוראות ומערך הנתונים לחוד).

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

1. שורה ריקה או שורת הערה:

האסמבלר מתעלם מהשורה ועובר לשורה הבאה.

2. שורת הוראה:

האסמבלר מנתח את השורה ומפענח מהי ההוראה.

אם האסמבלר מוצא בתחילת שורת ההוראה גם הגדרה של תווית, אזי התווית (הסמל) המוגדרת מוכנסת לטבלת הסמלים. ערך הסמל בטבלה הוא IC, והמאפיין הוא code

האסמבלר מכניס לתמונת הזיכרון שורה עבור ההוראה שנקראה, ובה הכתובת IC, והשדות של קוד המכונה בהתאם לסוג ההוראה (R, I, J). כל שדה שניתן לקודד כבר עתה, יקבל את הקידוד המתאים (ראו תאור המעבר הראשון לעיל).

לבסוף, האסמבלר מוסיף 4 לערכו של IC, מאחר וכל הוראה תקודד ל-4 תאי זיכרון (32 סיביות).

3. שורת הנחיה:

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

I. הנחיית .db, .dw, .dh
האסמבלר קורא את רשימת המספרים, מכניס כל מספר אל מערך הנתונים (בקידוד בינארי) ובכמות בתים בהתאם לסוג משפט ההנחיה . ומקדם את מצביע הנתונים DC ב-בכמות הבתים הכוללת שכל המספרים צרכו.

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

II. הנחיית .asciz
הטיפול ב-'.asciz' דומה ל- '.db' , אלא שקודי ה-ascii של התווים הם אלו המוכנסים אל מערך הנתונים (כל תו ב-byte נפרד). לבסוף מוכנס למערך הנתונים התו '\0' (המציין סוף מחרוזת). המונה DC מקודם באורך המחרוזת + 1 (גם התו המסיים את המחרוזת תופס מקום).

הטיפול בתווית המוגדרת בהנחיה '.asciz' זהה לטיפול הנעשה בהנחיות .dh .dw .db.

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

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

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

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

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

במעבר השני, האסמבלר משלים באמצעות טבלת הסמלים את קידוד כל ההוראות שטרם קודדו באופן מלא במעבר הראשון.

אלגוריתם שלדי של האסמבלר

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

לתשומת לב: אין חובה להשתמש דווקא באלגוריתם זה.

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

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

בכל מעבר מתחילים לקרוא את קוד המקור מההתחלה.

מעבר ראשון

1. אתחל DC ← 0, IC ← 100.
2. קרא את השורה הבאה מקובץ המקור. אם נגמר קובץ המקור, עבור ל- 17.
3. אם זוהי שורת הערה או שורה ריקה, חזור ל- 2
4. האם השדה הראשון בשורה הוא תווית? אם לא, עבור ל-6.
5. הדלק דגל "יש הגדרת סמל".
6. האם זוהי שורת הנחיה לאחסון נתונים, כלומר, האם הנחיית db, .dw, .dh. או asciz. ? אם לא, עבור ל- 9.
7. 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).
8. זהה את סוג הנתונים, וקודד אותם בתמונת הזיכרון של הנתונים. רשום בתמונת הנתונים גם את הערך הנוכחי של מונה הנתונים DC, ככתובת הזיכרון של (הבית הראשון של) הנתונים שהוגדרו בשורה הנוכחית. לאחר מכן, הגדל את DC על ידי הוספת האורך הכולל של הנתונים שהוגדרו. חזור ל-2.
9. האם זו הנחית extern. או הנחית entry. ? אם לא, עבור ל-12.
10. אם זוהי הנחית entry. חזור ל-2 (ההנחיה תטופל במעבר השני).

11. אם זו הנחית extern., הכנס את הסמל המופיע כאופרנד של ההנחיה לתוך טבלת הסמלים עם הערך 0, ועם המאפיין external. (אם הסמל אינו תווית חוקית, או שהסמל כבר נמצא בטבלה ללא המאפיין external, יש להודיע על שגיאה). חזור ל-2.
12. זוהי 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).
13. חפש את שם הפעולה בטבלת שמות הפעולות, ואם לא נמצא, אז הודע על שגיאה.
14. נתח את מבנה האופרנדים של ההוראה לפי סוג ההוראה. אם נתגלתה שגיאה, יש להודיע עליה.
15. בנה את הקידוד הבינארי של ההוראה, באופן מלא או באופן חלקי ככל שהדבר אפשרי (את מה שלא ניתן לקודד כעת, יש להשלים במעבר השני). הוסף את הקידוד לתמונת הזיכרון של ההוראות. רשום בתמונת ההוראות גם את הערך הנוכחי של IC ככתובת הזיכרון של ההוראה שנוספה.
16. עדכן IC ← IC + 4, וחזור ל-2.
17. קובץ המקור נקרא בשלמותו. אם נמצאו שגיאות במעבר הראשון, עצור כאן (לא יהיה מעבר שני ולא ייבנו קבצי פלט).
18. שמור את הערכים הסופיים של IC ושל DC (נקרא להם ICF ו- DCF). נשתמש בהם לבניית קבצי הפלט, אחרי המעבר השני.
19. עדכן בטבלת הסמלים את ערכו של כל סמל המאופיין כ- data , ע"י הוספת הערך ICF (ראו הסבר לכך בהמשך).
20. עדכן בתמונת הזיכרון של הנתונים את הכתובות של כל הנתונים (כפי שנרשמו בצעד 8), ע"י הוספת הערך ICF לכל כתובת.
21. התחל מעבר שני.

מעבר שני

1. קרא את השורה הבאה בקוד המקור. אם נגמר קוד המקור, עבור ל- 9.
2. אם זוהי שורת הערה או שורה ריקה, חזור ל- 1.
3. אם השדה הראשון בשורה הוא סמל (תווית), דלג עליו.
4. האם זוהי שורת הנחיה שאינה הנחיית entry.? אם כן, חזור ל- 1.
5. האם זוהי הנחית entry. ? אם לא, עבור ל- 7.
6. הוסף בטבלת הסמלים את המאפיין entry למאפייני הסמל המופיע כאופרנד של ההנחיה (אם הסמל לא נמצא בטבלת הסמלים, יש להודיע על שגיאה). חזור ל- 1.
7. זוהי שורת הוראה. השלם בתמונת הזיכרון את הקידוד הבינארי החסר של ההוראה בעזרת טבלת הסמלים. בהוראה מסוג J (מלבד stop) יש לקודד את כתובת התווית של האופרנד. בהוראת הסתעפות מותנית יש לחשב ולקודד את המרחק מכתובת ההוראה הנוכחית אל כתובת היעד (כתובת התווית של אופרנד היעד). ולהכניס מרחק זה לשדה המתאים בקידוד ההוראה. אם נדרש סמל שאינו נמצא בטבלת הסמלים, או במקרה של הסתעפות מותנית אם נדרש סמל המאופיין כ- external, יש לדווח על שגיאה ולחזור ל-1.
8. אם בצעד 7 נעשה שימוש בסמל שמאופיין כ- external (בהוראה מסוג J), הוסף את כתובת ההוראה הנוכחית, יחד עם שם הסמל, לרשימת הוראות שמשתמשות בסמל חיצוני (רשימה זו תשמש לבניית קבצי הפלט – ראו בהמשך). חזור ל- 1.
9. קוד המקור נסרק בשלמותו. אם נמצאו שגיאות במעבר השני, עצור כאן (לא ייבנו קבצי פלט).
10. בנה את קבצי הפלט (פרטים נוספים בהמשך).

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

להלן שוב תכנית הדוגמה.

```
MAIN:  add     $3,$5,$9
LOOP:  ori     $9,-5,$2
       la      val1
       jmp     Next
Next:  move    $20,$4
       bgt     $4,$2,END
       la      K
       sw      $0,4.$10
       bne     $31,$9,LOOP
       call    val1
       jmp     $4
END:   stop
STR:   .asciz  "aBcd"
LIST:  .db     6,-9
       .dh     27056
.entry K
K:     .dw     31,-12
.extern val1
```

נבצע מעבר ראשון על הקוד לעיל, ונבנה את טבלת הסמלים. כמו כן, נקודד במעבר זה את כל תמונת הנתונים, וכן חלקים מתמונת ההוראות, ככל שהדבר אפשרי בשלב זה.

נסמן "?" בתמונת ההוראות בכל שדה שלא ניתן לקידוד במעבר הראשון.

Address (decimal)	**Source Code**			**Machine Code (binary)**
0100	MAIN:	add	$3,$5,$9	000000 00011 00101 01001 00001 000000
0104	LOOP:	ori	$9,-5,$2	001101 01001 00010 1111111111111011
0108		la	val1	011111 0 ?
0112		jmp	Next	011110 0 ?
0116	Next:	move	$20,$4	000001 10100 00000 00100 00001 000000
0120		bgt	$4,$2,END	010010 00100 00010 ?
0124		la	K	011111 0 ?
0128		sw	$0,4,$10	010110 00000 01010 0000000000000100
0132		bne	$31,$9,LOOP	001111 11111 01001 ?
0136		call	val1	100000 0 ?
0140		jmp	$4	011110 1 00000000000000000000000100
0144	END:	stop		111111 0 00000000000000000000000000
0148	STR:	.asciz	"aBcd"	01100001
0149				01000010
0150				01100011
0151				01100100
0152				00000000
0153	LIST:	.db	6,-9	00000110
0154				11110111
0155		.dh	27056	0110100110110000
0157	K:	.dw	31,-12	00000000000000000000000000011111
0161				11111111111111111111111111110100

טבלת הסמלים אחרי מעבר ראשון היא:

Symbol	Value (decimal)	Attributes
MAIN	100	code
LOOP	104	code
Next	116	code
END	144	code
STR	148	data
LIST	153	data
K	157	data
val1	0	external

נבצע עתה את המעבר השני. נשלים באמצעות טבלת הסמלים את הקידוד במקומות המסומנים ב- "?".
הקוד הבינארי בצורתו הסופית כאן זהה לקוד שהוצג בתחילת הנושא "אסמבלר עם שני מעברים".

Address (decimal)	Source Code			Machine Code (binary)
0100	MAIN:	add	$3,$5,$9	000000 00011 00101 01001 00001 000000
0104	LOOP:	ori	$9,-5,$2	001101 01001 00010 1111111111111011
0108		la	val1	011111 0 00000000000000000000000
0112		jmp	Next	011110 0 00000000000000000001110100
0116	Next:	move	$20,$4	000001 10100 00000 00100 00001 000000
0120		bgt	$4,$2,END	010010 00100 00010 0000000000011000
0124		la	K	011111 0 00000000000000000010011101
0128		sw	$0,4,$10	010110 00000 01010 0000000000000100
0132		bne	$31,$9,LOOP	001111 11111 01001 1111111111100100
0136		call	val1	100000 0 00000000000000000000000
0140		jmp	$4	011110 1 00000000000000000000000100
0144	END:	stop		111111 0 00000000000000000000000
0148	STR:	.asciz	"aBcd"	01100001
0149				01000010
0150				01100011
0151				01100100
0152				00000000
0153	LIST:	.db	6,-9	00000110
0154				11110111
0155		.dh	27056	0110100110110000
0157	K:	.dw	31,-12	00000000000000000000000000011111
0161				11111111111111111111111111110100

טבלת הסמלים אחרי מעבר שני היא:

Symbol	Value (decimal)	Attributes
MAIN	100	code
LOOP	104	code
Next	116	code
END	144	code
STR	148	data
LIST	153	data
K	157	data, entry
val1	0	external

בסוף המעבר השני, אם לא נתגלו שגיאות, האסמבלר בונה את קבצי הפלט (ראו בהמשך), שמכילים את הקוד הבינארי ומידע נוסף עבור שלב הקישור.

הערה: כאמור, האסמבלר בונה קוד מכונה כך שיתאים לטעינה לזיכרון החל מכתובת 100 (עשרוני). אם הטעינה בפועל (לצורך הרצת התוכנית) תהיה לכתובת אחרת, יידרשו תיקונים בקידוד של חלק מההוראות מסוג J בעת הטעינה.

כאמור, שלבי הקישור והטעינה אינם למימוש בפרויקט זה, ולא נדון בהם כאן.

קבצי קלט והפעלת האסמבלר

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

שם של קובץ המכיל תכנית בשפת אסמבלי חייב להיות עם הסיומת ".as". למשל, hello.as הוא שם תקין.

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

assembler test.as myprog.as hello.as

תריץ את האסמבלר על שלשת קבצי הקלט שהם הארגומטים בשורת הפקודה.

האסמבלר פועל בנפרד על כל קובץ קלט ברשימת הארגומנטים, אחד אחרי השני. אם קובץ קלט לא נפתח, יש להדפיס הודעת שגיאה ולעבור לקובץ הבא. אם קובץ הקלט עבר את תהליך האסמבלי ללא שגיאות, האסמבלר בונה עבורו קבצי פלט.

קבצי פלט של האסמבלר

עבור כל קובץ קלט שעבר אסמבלי ללא שגיאות, האסמבלר יוצר קבצי פלט כדלקמן.

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

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

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

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

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

פורמט קובץ ה- object

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

כזכור, האסמבלר מקודד את ההוראות כך שתמונת ההוראות תתאים לטעינה החל מכתובת 100 (עשרוני) בזיכרון. נשים לב שרק בסוף המעבר הראשון יודעים מהו הגודל הכולל של תמונת ההוראות. מכיוון שתמונת הנתונים נמצאת אחרי תמונת ההוראות, גודל תמונת ההוראות משפיע על הכתובות בתמונת הנתונים. זו הסיבה שבגללה היה צורך לעדכן בטבלת הסמלים, בסוף המעבר הראשון, את ערכי הסמלים המאופיינים כ-data (כזכור, באלגוריתם השלדי שהוצג לעיל, בצעד 19, הוספנו לכל סמל כזה את הערך ICF). במעבר השני, בהשלמת הקידוד, משתמשים בערכים המעודכנים של הסמלים, המותאמים למבנה המלא והסופי של תמונת הזיכרון.

כעת האסמבלר יכול לכתוב את תמונת הזיכרון בשלמותה לתוך קובץ פלט (קובץ ה- object). הקובץ בנוי משורות טקסט בפורמט שלהלן.

השורה הראשונה בקובץ ה- object היא "כותרת", המכילה שני מספרים בבסיס עשרוני: הראשון הוא האורך הכולל של תמונת ההוראות (כלומר כמות תאי הזיכרון), והשני הוא האורך הכולל של תמונת הנתונים (כמות תאי הזיכרון). בין שני המספרים מפריד רווח אחד.
כזכור, במעבר הראשון, בצעד 18 באלגוריתם, נשמרו הערכים ICF ו- IDF. האורך הכולל של תמונת ההוראות הוא ICF−100, והאורך הכולל של תמונת הנתונים הוא IDF.

השורות הבאות בקובץ מכילות את תמונת הזיכרון, לפי סדר עולה של כתובות. בכל שורה 5 שדות: כתובת בזיכרון ולאחריה התכנים של 4 בתים שנמצאים החל מכתובת זו, בסדר עולה משמאל לימין בשורה. הכתובת תירשם בבסיס עשרוני בארבע ספרות (כולל אפסים מובילים). התוכן של כל בית יירשם בבסיס הקסאדצימלי בשתי ספרות (כולל אפס מוביל), כאשר הספרות A-F באותיות גדולות. בין שדות בשורה מפריד רווח אחד.
השורה האחרונה בקובץ יכולה להכיל פחות מ- 4 בתים, לפי הצורך.

הוראות מכונה מקודדות ל-4 בתים, ולפיכך כל הוראה תתפוס שורה שלמה בקובץ ה-object. כנאמר, הזיכרון מאורגן בשיטת little-endian, ולכן סיביות 0-7 של ההוראה יהיו הבית השמאלי בסדר הבתים בשורה, סיביות 8-15 הבית השני משמאל בשורה, וכן הלאה.

הנתונים, לעומת זאת, מקודדים באורכים שאינם בהכרח כפולה של 4 בתים. למרות זאת, יש להכניס את הנתונים לקובץ ה-object ברביעיות של בתים בכל שורה, ללא "חורים", ותוך הקפדה על הסדר הנכון (סדר עולה של כתובות בכל שורה משמאל לימין, וסדר עולה משורה לשורה). ראו דוגמה בהמשך.

פורמט קובץ ה-entries

קובץ ה-entries בנוי משורות טקסט, שורה אחת לכל סמל שמאופיין בטבלת הסמלים כ- entry. בשורה מופיע שם הסמל, ולאחריו ערכו כפי שנקבע בטבלת הסמלים (בבסיס עשרוני בארבע ספרות, כולל אפסים מובילים). בין שני השדות בשורה יש רווח אחד. אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.

פורמט קובץ ה- externals

קובץ ה-externals בנוי אף הוא משורות טקסט, שורה לכל הוראה מסוג J בקוד המכונה בה יש שימוש בסמל שמאופיין כ- external. כזכור, רשימה של הוראות אלה נבנתה במעבר השני (צעד 8 באלגוריתם השלדי).

כל שורה בקובץ ה-externals מכילה את שם הסמל החיצוני, ולאחריו כתובת ההוראה בה הוא נדרש (בבסיס עשרוני בארבע ספרות, כולל אפסים מובילים). בין שני השדות בשורה יש רווח אחד. אין חשיבות לסדר השורות, כי כל שורה עומדת בפני עצמה.

לתשומת לב: ייתכן ויש מספר כתובות בקוד המכונה בהן נדרשת כתובתו של הסמל החיצוני ,לכל כתובת כזו תהיה שורה נפרדת בקובץ ה-externals.

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

```
;file ps.as
;sample source code

.entry   Next
.extern  wNumber
STR:     .asciz   "aBcd"
MAIN:    add      $3,$5,$9
LOOP:    ori      $9,-5,$2
         la       val1
         jmp      Next
Next:    move     $20,$4
LIST:    .db      6,-9
         bgt      $4,$2,END
         la       K
         sw       $0,4,$10
         bne      $31,$9, LOOP
         call     val1
         jmp      $4
         la       wNumber
.extern  val1
         .dh      27056
K:       .dw      31,-12
END      stop
.entry   K
```

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

Address (decimal)	Source Code			Machine Code (binary)
0100	MAIN:	add	$3,$5,$9	000000 00011 00101 01001 00001 000000
0104	LOOP:	ori	$9,-5,$2	001101 01001 00010 1111111111111011
0108		la	val1	011111 0 00000000000000000000000000
0112		jmp	Next	011110 0 00000000000000000001110100
0116	Next:	move	$20,$4	000001 10100 00000 00100 00001 000000
0120		bgt	$4,$2,END	010010 00100 00010 0000000000011100
0124		la	K	011111 0 00000000000000000010100001
0128		sw	$0,4,$10	010110 00000 01010 0000000000000100
0132		bne	$31,$9,LOOP	001111 11111 01001 1111111111100100
0136		call	val1	100000 0 00000000000000000000000000
0140		jmp	$4	011110 1 00000000000000000000000100
0144		la	wNumber	011111 0 00000000000000000000000000
0148	END:	stop		111111 0 00000000000000000000000000
0152	STR:	.asciz	"aBcd"	01100001
0153				01000010
0154				01100011
0155				01100100
0156				00000000
0157	LIST:	.db	6,-9	00000110
0158				11110111
0159		.dh	27056	0110100110110000
0161	K:	.dw	31,-12	00000000000000000000000000011111
0165				11111111111111111111111111110100

טבלת הסמלים בגמר המעבר השני היא:

Symbol	Value (decimal)	Attributes
wNumber	0	external
STR	152	data
MAIN	100	code
LOOP	104	code
Next	116	code, entry
LIST	157	data
val1	0	external
K	161	data, entry
END	148	code

להלן תוכן קבצי הפלט של הדוגמה.

הקובץ **ps.ob**:

```
     52 17
0100 40 48 65 00
0104 FB FF 22 35
0108 00 00 00 7C
0112 74 00 00 78
0116 40 20 80 06
0120 1C 00 82 48
0124 A1 00 00 7C
0128 04 00 0A 58
0132 E4 FF E9 3F
0136 00 00 00 80
0140 04 00 00 7A
0144 00 00 00 7C
0148 00 00 00 FC
0152 61 42 63 64
0156 00 06 F7 B0
0160 69 1F 00 00
0164 00 F4 FF FF
0168 FF
```

הקובץ **ps.ext**:

```
val1 0108
val1 0136
wNumber 0144
```

הקובץ **ps.ent**:

```
Next 116
K 0161
```

סיכום והנחיות כלליות

- גודל תכנית המקור הניתנת כקלט לאסמבלר אינו ידוע מראש, ולכן גם גודלו של קוד המכונה אינו צפוי מראש. אולם בכדי להקל במימוש האסמבלר, מותר להניח גודל מקסימלי. לפיכך יש אפשרות להשתמש במערכים לאכסון תמונת קוד המכונה בלבד. כל מבנה נתונים אחר (למשל טבלת הסמלים), יש לממש באופן יעיל וחסכוני (למשל באמצעות רשימה מקושרת והקצאת זיכרון דינאמי).

- השמות של קבצי הפלט צריכים להיות תואמים לשם קובץ הקלט, למעט הסיומות. למשל, אם קובץ הקלט הוא prog.as אזי קבצי הפלט שיווצרו הם: prog.ob, prog.ext, prog.ent

- מתכונת הפעלת האסמבלר צריכה להיות כפי הנדרש בממ"ן, ללא שינויים כלשהם. כלומר, ממשק המשתמש יהיה אך ורק באמצעות שורת הפקודה. בפרט, שמות קבצי המקור יועברו לתוכנית האסמבלר כארגומנטים (אחד או יותר) בשורת הפקודה. אין להוסיף תפריטי קלט אינטראקטיביים, חלונות גרפיים למיניהם, וכד'.

- יש להקפיד לחלק את מימוש האסמבלר למספר מודולים (קבצים בשפת C) לפי משימות. אין לרכז משימות מסוגים שונים במודול יחיד. מומלץ לחלק למודולים כגון: מעבר ראשון, מעבר שני, פונקציות עזר (למשל, תרגום לבסיס, ניתוח תחבירי של שורה), טבלת הסמלים, מפת הזיכרון, טבלאות קבועות (כגון קודי הפעולה)

- יש להקפיד ולתעד את המימוש באופן מלא וברור, באמצעות הערות מפורטות בקוד.

- יש לאפשר תווים לבנים עודפים בקובץ הקלט בשפת אסמבלי. למשל, אם בשורת הוראה יש שני אופרנדים המופרדים בפסיק, אזי לפני ואחרי הפסיק מותר שיהיו רווחים וטאבים בכל כמות. בדומה, גם לפני ואחרי שם הפעולה. מותרות גם שורות ריקות. האסמבלר יתעלם מתווים לבנים מיותרים (כלומר ידלג עליהם).

- הקלט (קוד האסמבלי) עלול להכיל שגיאות תחביריות. על האסמבלר **לגלות ולדווח על כל השורות השגויות** בקלט. אין לעצור את הטיפול בקובץ קלט לאחר גילוי השגיאה הראשונה. יש להדפיס למסך הודעות מפורטות ככל הניתן, כדי שאפשר יהיה להבין מה והיכן כל שגיאה. כמובן שאם קובץ קלט מכיל שגיאות, אין טעם להפיק עבורו את קבצי הפלט (ob, ext, ent).

תם ונשלם פרק ההסברים והגדרת הפרויקט.

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

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

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

ב ה צ ל ח ה !