תיעוד התוכנית

הסבר כללי:

הקומפיילר מקבל קובץ cmm, ומקמפל אותו בצורה הבאה:

קובץ הbison (part2.y) מקבל מקובץ הpart2.lex אסימונים כאשר כל אסימון מכיל מבנה נתונים בשם Node (המוגדר בקובץ part3_helpers.hpp) שהשדות הרלוונטיים בו לאותו אסימון מעודכנים ע"י הlex. כל עוד הbison לא מזהה כלל גזירה הוא מכניס את האסימון למחסנית (shift) וכאשר הוא מזהה כלל גזירה (במידה ויש חוסר בהירות האם באמת הכוונה לכלל הזה הוא משתמש בכללי ההחלטה שנקבעו) הוא מוציא מהמחסנית את כל האסימונים של הכלל (reduce), מפענח אותו , בודק את הכללים הסמנטיים הנובעים מהאסימון, מדפיס קוד (emit) במידת הצורך, מעדכן את מבני הנתונים של הקומפיילר (במידת הצורך) ומעביר הלאה את הנתונים החשובים. לאחר סיום יצירת שפת המכונה, בעזרת מבני הנתונים מתווסף לקובץ המידע הרלוונטי ללינקר. במידה ויש שגיאה (מכל סוג) השגיאה תודפס למסך (סוג ופירוט) ולא ייווצר קוד מכונה (קובץ rsk.).

מבני נתונים:

- Node - מבנה הנתונים של כל אסימון, מבנה נתונים כללי שמאפשר שימוש לכל האסימונים ללא תלות בתוכנם. מכיל בתוכו:
  1. מחרוזת name – עבור אסימונים של id
  2. type – עבור משתנים וביטויים (int, float, pointer, void, volatile) ופונקציות (ערך החזרה).
  3. offset – מיקום ביחס לתחילת רשומת ההפעלה – עבור משתנים, יכול להיות שלילי
  4. offsetReg– הרגיסטר שמחזיק את ההיסט עבור de-reference למצביעים (LVAL).
  5. dereferencedPtr– משתנה בוליאני שמצביע על כך שהמשתנה הוא de-reference למצביע.
  6. quad– מספר שורה בקוד המכונה.
  7. place– מספר רגיסטר בתוכו מאוכסן הערך של האסימון (עבור ערך של ביטויים)
  8. כל סוגי הרשימות עבור backpatching: nextList, falseList, trueList
  9. argDefinitions– שמות משתנים שהוגדרו – עבור הצהרה על מספר משתנים או עבור ארגומנטים של פונקציה.
  10. argTypes– הטיפוסים של המשתנים מהסעיף הקודם.
  11. argRegs– מספרי הרגיסטרים של המשתנים המועברים לפונקציה.
- Symbol – מבנה הנתונים של משתנה. מכיל בתוכו:
  1. offset– היסט מתחילת רשומת ההפעלה.
  2. type – סוג המשתנה (int, float, pointer, volatile)
- SymbolTable– מבנה נתונים המחזיק בתוכו את כל המשתנים ה"חיים" בזמן מסוים, בנוי כווקטור של maps כאשר כל map מחזיק בתוכו את כל המשתנים של בלוק מסוים, ומיקום המap בוקטור מצביע על עומק הscope. במבנה נתונים זה ניתן לחפש משתנה, להכניס משתנה, לפתוח בלוק חדש, לסגור בלוק ולאפס את כולו, והוא מכיל בתוכו:
  1. symTable–וקטור של maps, כאשר בכל map הkey הוא שם המשתנה, והvalue הוא מבנה הנתונים של אותו המשתנה.

- `FunctionInformation`– מבנה נתונים שמחזיק בתוכו את כל המידע שצריך לדעת על פונקציה, והוא מכיל בתוכו:
  1. `argTypes`– סוגי המשתנים שהפונקציה מקבלת (לפי סדר ההצהרה על המשתנים)
  2. `argNames` – שמות המשתנים של הפונקציה (להשוואה בין הכרזה למימוש).
  3. `returnType`– סוג ערך החזרה (`int, float, void`)
  4. `locationInFile`– מיקום תחילת הפונקציה בקוד המכונה (מספר שורה)
  5. `callLocations`– רשימת כל המיקומים בקוד המכונה בהם הפונקציה נקראה (לטובת backpatch וlink)
- `CodeBuffer`– מבנה נתונים שמחזיק בתוכו את הקוד שנוצר בשלב הקומפילציה, מחזיק כל שורה בקוד בתא בתוך ווקטור (לפי סדר השורות), ומאפשר להוסיף שורה חדשה, לבצע backpatch לשורה קיימת, לקבל את מספר השורה הבאה ולקבל את הקוד כמחרוזת אחת ארוכה. מחזיק בתוכו:
  1. `code`– וקטור של הקוד, כמתואר למעלה.

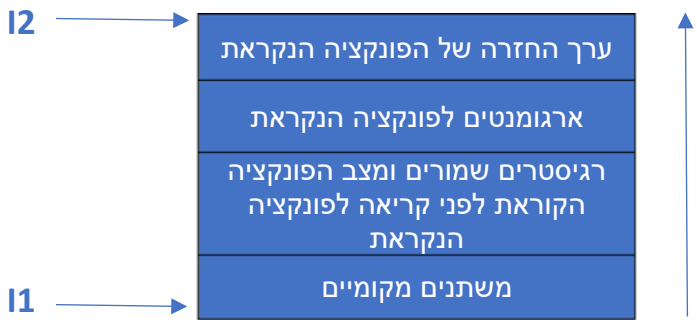
Backpatch:

המקרים בהם נבצע backpatch:

- בעת מימוש פונקציה: כאשר נזהה שאנחנו נמצאים בכלל גזירה שמבצע reduce למימוש פונקציה, נבדוק האם הפונקציה כבר הוכרזה, ואם כן, נבדוק האם היא כבר נקראה בקוד, במידה וכן, נלך לכל השורות הרלוונטיות (שמור בתוך המידע של כל פונקציה) ונבצע backpatch לשורה בה הפונקציה ממומשת.
- בעת סיום פונקציה (סגירת בלוק חיצוני של פונקציה): נבצע backpatch של הnextList של הבלוק לשורה הבאה בקוד. בnextList של הבלוק ישבו כל הnextLists של STMT שלא בוצע להם backpatch עד לשלב זה.
- עבור STMT_LIST: נבצע backpatch של הnextList לquad של תחילת הSTMT הבא. (כאשר הSTMT האחרון יטופל ע"י הreduce לבלוק)
- עבור ASSN_C: דומה לכלל if-then-else, יש צורך לבצע backpatch ל trueList, falseList, nextList ע"מ לקבל את הערך הנכון. פירוט בתרגיל היבש.
- עבור CNTRL מהסוגים השונים(if-then, if-then-else, while-do): יבוצע backpatch בדומה לנעשה בהרצאה ובתרגול.
- עבור תנאי or/and בBEXP: יבוצע backpatch בדומה לנעשה בהרצאה ובתרגול.

מבנה רשומת הפעלה:

מבנה רשומת ההפעלה הוא:



כאשר I1 מצביע לתחילת רשומת ההפעלה, וI2 לסופה.

קריאה לפונקציה:

תהליך הקריאה לפונקציה מתרחש בצורה הבאה:

1. הפונקציה **הקוראת** שומרת את הרגיסטרים שלה (כולל I0,I1) ברשומת הקריאות ומקדמת את I2 לפי כמות הרגיסטרים שצריך לשמור (כולל הרגיסטרים השמורים).
2. הפונקציה **הקוראת** מכניסה להמשך רשימת ההפעלה את הארגומנטים לפונקציה הנקראת מהסוף להתחלה (הארגומנט האחרון של הפונקציה נקראת מוכנס ראשון, ומעליו הארגומנט הקודם וכך הלאה) ומקדמת את I2 בהתאם.
3. הפונקציה **הקוראת** מקצה מקום לערך ההחזרה של הפונקציה הנקראת מעל המקום המוקצה לארגומנט הראשון של הפונקציה הנקראת, ומקדמת את I2 בהתאם.
4. הפונקציה **הקוראת** מקדמת את I1 לI2.
5. הפונקציה **הקוראת** מבצעת קפיצה ושומרת ברגיסטר I0 את כתובת החזרה.
6. הפונקציה **הנקראת** מתחילה בפעילותה.

חזרה מפונקציה:

תהליך החזרה מפונקציה מתרחש בצורה הבאה:

1. הפונקציה **הנקראת** מעדכנת את ערך ההחזרה במקום המיועד לכך (I1 -4)
2. הפונקציה **הנקראת** מעתיקה את ערכו של I1 אל I2.
3. הפונקציה **הנקראת** מבצעת קפיצה לכתובת ברגיסטר I0.
4. הפונקציה **הקוראת** מחסרת מI2 את המקום שהקצתה לשמירת רגיסטרים ולערך החזרה.
5. הפונקציה **הקוראת** משחזרת את ערכי הרגיסטרים שלה (כולל השמורים) מרשומת ההפעלה.
6. הפונקציה **הקוראת** מכניסה לרגיסטר את ערך החזרה של הפונקציה הנקראת.
7. הפונקציה **הקוראת** ממשיכה בפעילותה.

רגיסטרים שמורים:

I0	כתובת החזרה של הפונקציה
I1	תחילת רשומת ההפעלה
I2	סוף רשומת ההפעלה

מודולים:

המודולים בתוכנית הם:

- codeBuf– אובייקט מסוג `CodeBuffer` שמכיל בתוכו את קוד המכונה הנוצר.
- symTable– אובייקט מסוג `SymbolTable` שמכיל בתוכו את כל המשתנים ה"חיים" בזמן נתון לפי היררכיה של בלוקים.
- declaredFunctions– map של אובייקטים מסוג `FunctionInformation` כאשר הkey הוא שם הפונקציה, ומכיל בתוכו את כל הפונקציות שהוצהרו עד שלב זה בתוכנית, ארגומנטים, מיקום המימוש (במידה וקיים) ומספרי השורות בהן קוראים לפונקציה.
- implementedFunctions– `map` של אובייקטים מסוג `FunctionInformation` כאשר הkey הוא שם הפונקציה, ומכיל בתוכו את כל הפונקציות שמומשו עד שלב זה בתוכנית, ארגומנטים ומיקום המימוש.
- pendingFunc– אובייקט `pair<string, FunctionInformation>` שמחזיק בתוכו נתונים של פונקציה, בשלב בו עוד לא ידוע האם זהו מימוש או הכרזה. אובייקט זה רלוונטי לקריאה רקורסיבית לפונקציה שטרם הוכרזה. בעזרת אובייקט זה ניתן לדעת את פרטי הפונקציה לפני שהיא נכנסת למבנה הנתונים המתאים (פונקציות ממומשות או מוכרזות).

statics:

המשתנים הסטטיים בתוכנית הם:

- nextFreeRegI– הרגיסטר מסוג int הפנוי הבא.
- nextFreeRegF– הרגיסטר מסוג float הפנוי הבא.
- funcStartingLine– מספר השורה של תחילת הפונקציה הנוכחית (לטובת backpatch).
- currentStackOffset– המקום הפנוי הבא ברשומת ההפעלה (הoffset במחסנית מ-l1).
- functionReturnType– ערך ההחזרה של הפונקציה הנוכחית (לטובת בדיקת ערך ההחזרה לפני ביצוע reduce למימוש פונקציה).

קבצים:

Makefile– קובץ make לטובת קומפילציה של הקבצים מייצר קובץ הרצה בשם rx-cc

Part3_helpers.cpp- מכיל את הפונקציות הפועלות על מבנה הנתונים בתרגיל

Part3_helpers.hpp- מכיל את מבנה הנתונים, ההצהרות על הפונקציות של מבנה הנתונים והמשתנים הסטטיים של התוכנית.

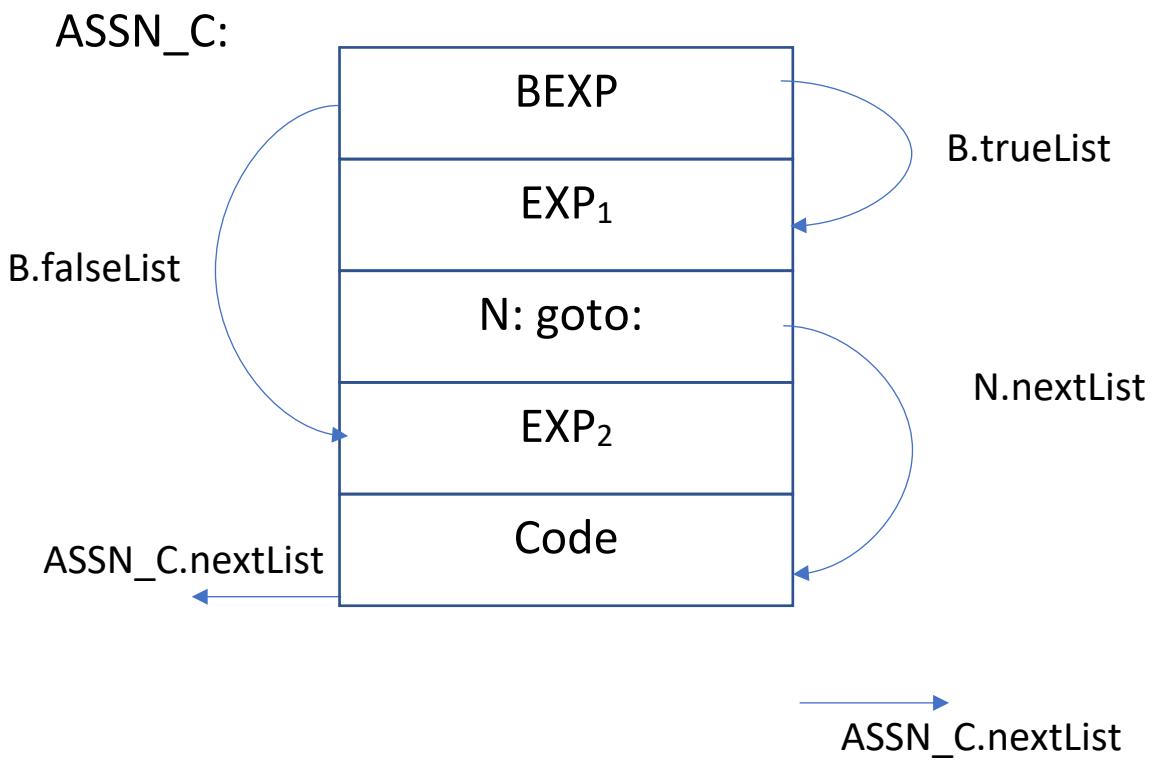
Part3.lex- מנתח לקסיקלי ומטפל בשגיאות לקסיקליות.

Part3.ypp- מנתח תחבירי, מנתח שגיאות סמנטיות, מייצר הקוד ומטפל בשגיאות: תחביריות, סמנטיות ותפעוליות.

מגישים: ספיר מלכה 205794001, אלדד וינר 304901069

חלק יבש

1. פריסת קוד עבור ASSN_C:

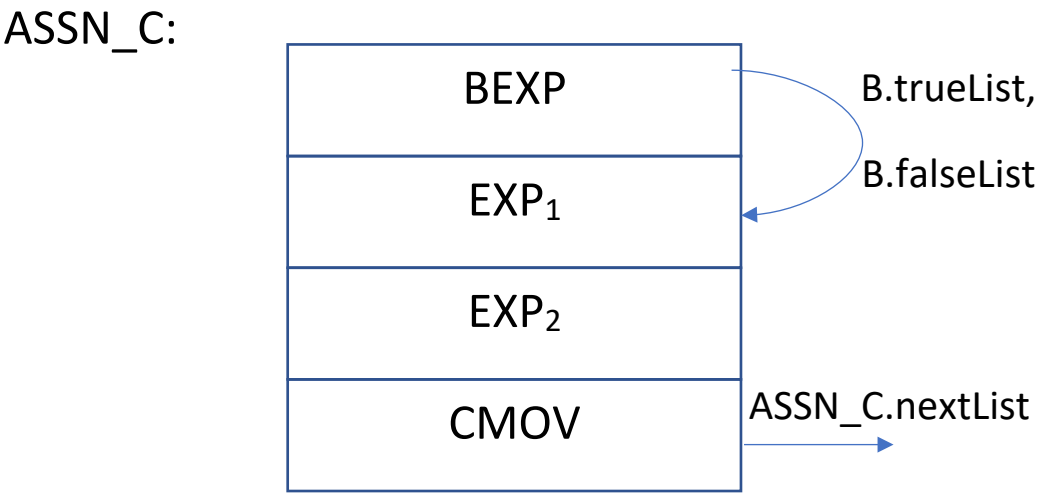


מכיוון שכתוצאה מBEXP נבצע רק פעולה אחת מ2 הEXP, כאשר נגיע לCode נדע בוודאות שברגיסטר נמצא הערך אותו נרצה לשים בLVAL.

2. קוד המכונה עבור: b = (3>4) ?1 : 2;

```
COPYI I3 3        # insert 3 to reg I3
COPYI I4 4        # insert 4 to reg I4
SGRTI I5 I3 I4    # if I3>I4 then I5 = 1 else I5 = 0;
BNEQZ 6           #if I5 = 1 goto B.truelist
UJUMP 8           #if I5 = 0 goto B.falselist
COPYI I6 1        # insert 1 to reg I6
UJUMP 9           # goto N.nextlist
COPYI I6 2        # insert 2 to reg I6
STORI I6 I0 0     # set b value to I6 value
```

.3



נכניס את ההשוואה לרגיסטר ולאחר מכן ללא תלות בתוצאת הBEXP נבצע את שני הEXP (ולא רק אחד כמו בסעיף 1) לאחר מכן בפקודת CMOV נבדוק את ערך רגיסטר ההשוואה, נציב את ערך הביטוי ונצא.

.4

```
COPYI I3 3        # insert 3 to reg I3
COPYI I4 4        # insert 4 to reg I4
SGRTI I5 I3 I4    # if I3>I4 then I5 = 1 else I5 = 0;
BNEQZ 6           #if I5 = 1 goto B.truelist
UJUMP 6           #if I5 = 0 goto B.falselist
COPYI I6 1        # insert 1 to reg I6
COPYI I7 2        # insert 2 to reg I7
ADD2I I8 I0 0     # insert b address to I8
CMOV I8 I5 I6 I7# MEM[I8] = (I5 = 1) ? I6 : I7
```