תורת הקומפילציה

תרגיל בית 5 – יצור קוד ביניים

מתרגל אחראי: תומר ביתן – tomerbitan@campus.technion.ac.il

ההגשה בזוגות

עבור כל שאלה על התרגיל, יש לעין ראשית **בפיאצה** ובמידה שלא פורסמה אותה השאלה, ניתן להוסיף אותה ולקבל מענה, אין לשלוח מיילים בנושא התרגיל בית כדי שנוכל לענות על השאלות שלכם ביעילות.

תיקונים לתרגיל יסומנו בצהוב, חובתכן להתעדכן בהם באמצעות קובץ התרגיל.

התרגיל ייבדק בבדיקה אוטומטית. **הקפידו למלא אחר ההוראות במדויק**. הבדיקה תתבצע על שרת הקורס csComp.

הנחיות כלליות

בתרגיל זה תממשו תרגום לשפת ביניים LLVM IR, עבור השפה FanC מתרגילי הבית הקודמים. בין היתר תממשו השמה של משתנים מקומיים במחסנית, ומימוש מבני בקרה באמצעות Visitor.

LLVM IR

בתרגיל תשתמשו בשפת הביניים של llvm שראיתם בהרצאה ובתרגול. ניתן למצוא מפרט מלא של השפה כאן: https://llvm.org/docs/LangRef.html.

תוכלו לדבג את קוד הביניים שלכם ע״י שימוש בהדפסות.

פקודות אפשריות

בשפת llvm יש מספר גדול מאוד של פקודות. בתרגיל תרצו להשתמש בפקודות המדמות את שפת הרביעיות שנלמדה בתרגול. להלן הפקודות שתרצו להשתמש בהן:

1. טעינה לרגיסטר: `load`
2. שמירת תוכן רגיסטר: `store`
3. פעולות חשבוניות: `add`, `sub`, `mul`, `udiv`, `sdiv`
4. פקודת השוואה: `icmp`
5. קפיצות מותנות ולא מותנות: `br`
6. קריאה לפונקציה: `call`
7. חזרה מפונקציה: `ret`
8. הקצאת זיכרון: `alloca`
9. חישוב כתובת: `getelementptr`
10. צומת phi: `phi`

הפקודה phi מקבלת רשימת זוגות של ערכים ולייבלים ומשימה לרגיסטר את הערך המתאים לליבל של הבלוק שקדם לבלוק הנוכחי של צומת ה-phi בזמן ריצת התוכנית.
במידה ומשתמשים בפקודה phi היא חייבת להיות הפקודה הראשונה בבלוק הבסיסי.

תוכלו למצוא תיעוד של כולן במדריך llvm לעיל.

במידה ותרצו להשתמש בפקודות נוספות - מותר להשתמש בכל פקודת LLVM שניתן להריץ באמצעות `lli`.

רגיסטרים

ב-LLVM ישנו מספר אינסופי של רגיסטרים לשימושכם. השפה היא Single Static Assignmet (SSA) כך שניתן לבצע השמה יחידה לרגיסטר.

שימו לב כי ב-LLVM לא קיימת פקודה ייעודית לביצוע השמה של קבוע לתוך רגיסטר. עם זאת, במידת ורוצים ניתן לבצע זאת ע"י שימוש בפקודה add עם הערך המבוקש ואופרנד נוסף 0.

תוויות קפיצה

ב-llvm יעדים של קפיצות מיוצגים בתור תוויות (labels): מחרוזות אלפאנומריות (+ קו תחתון, נקודה ודולר) שאחריהן מופיעות נקודותיים, כך:

```
label_42:
%t6 = load i32, i32* %ptr
```

קפיצה אל `%label_42` תקפוץ אל הבלוק הבסיסי המתחיל בשורה שאחריה במקרה הזה, הפקודה load.
כל תווית מתחילה בלוק בסיסי חדש וכל בלוק בסיסי צריך להסתיים בפקודת br או ret הקובע את מבנה גרף הבקרה של התוכנית.

ניתן ליצור תוויות על ידי קריאה למטודה freshLabel של מחלקת CodeBuffer. אין חובה להשתמש בה, ניתן לנהל את הלייבלים שלכם בעצמכם.

CodeBuffer

לצורך העבודה עם באפר הקוד נתונה לכם מחלקה CodeBuffer בקובץ **output.hpp**. במחלקו זו תוכלו למצוא מטודות הבאות:

- freshLabel – מחזירה תווית חדשה. המטודה רק מחזירה את התווית ללא הוספתה לבאפר. ניתן להשתמש בתווית כחלק מפקודות אחרות ולהוסיף אותה לבאפר בעזרת emitLabel.
- freshVar – מחזירה רגיסטר חדש שעוד לא נעשה בו שימוש. המטודה עובדת בהנחה שכל הרגיסטרים בבפאר הוצרו על ידי המטודה.
- emitLabel – מקבלת תווית ומוסיפה אותה לבפאר כפקודה. למשל, עבור תווית `%label_42` תוסף לבאפר שורת קוד: `label_42:`
- emitString – מקבלת מחרוזת ומוסיפה אותה לאזור ההגדרות המקדימות את הבאפר. ניתן להשתמש במטודה זו על מנת להוסיף ליטרלים של מחרוזות בקוד. ראו דוגמת שימוש בתיעוד.
  ניתן להניח כי המחרוזות בתוכניות הבדיקה לא יכילו בריחה.
- emit – פולטת שורת קוד לבאפר. שימו לב כי הפקודה פולטת שורת קוד ביניים מלאה. ניתן להשתמש באופרטור >> במקום emit על מנת להעביר לבאפר שורות קוד לא שלמות ואופן עבודה איתו דומה ל-std::cout. למשל, שתי השורות שקולות מבחינת קוד באפר:

```
buffer.emit(buffer.freshVar() + " = icmp eq i32 0, 0");

buffer << buffer.freshVar() << " = icmp eq i32 0, 0" << endl;
```

בסוף יצירת קוד הביניים, ניתן להדפיס את הבאפר על ידי עברתו לערוץ הקלט הסטנדרטי:

```
std::cout << buffer;
```

מחסנית

בתרגיל אתם לא נדרשים לנהל את המחסנית עם רשומות ההפעלה של הפונקציות הנקראות.

את המשתנים הלוקלים של הפונקציות יש לאחסן על מחסנית, לפי ה-offsets שחושבו בתרגיל 3.

מומלץ להקצות בתחילת הפונקציה מקום לכל משתנים הלוקלים על המחסנית באמצעות הפקודה alloca ובה נתייחס לכל משתנה ללא תלות בטיפוסו כ-i32.

בכדי לאחסן טיפוס בוליאני או byte כ-i32 ניתן להשתמש בפקודה zext המשלימה את הביטים העליונים עם אפסים.

ניתן להניח כי מספר המשתנים הלוקלים בכל פונקציה קטן מ-50.

סמנטיקה

יש לממש את ביצוע כל המשפטים (statements) בפונקציה ברצף בסדר בו הוגדרו. הסמנטיקה של ביטויים אריתמטיים ושל קריאות לפונקציות מוגדרת כמו הסמנטיקה שלהם בשפת C. ההרצה תתחיל בפונקציה main, ותסתיים כשהקריאה החיצונית ביותר לפונקציה main חוזרת.

משתנים

אתחול משתנים

יש לאתחל את כל המשתנים בתכנית כך שיכילו ערך ברירת מחדל במידה ולא הוצב לתוכם ערך. הטיפוסים המספריים יאותחלו ל-0. הטיפוס הבוליאני יאותחל ל-false.

גישה למשתנים

כאשר מתבצעת פניה בתוך ביטוי למשתנה מטיפוס פשוט, יש לייצר קוד הטוען מן המחסנית את הערך האחרון שנשמר עבור המשתנה. כאשר מתבצעת השמה לתוך משתנה, יש לייצר קוד הכותב למחסנית את ערך הביטוי במשפט ההשמה.

ביטויים חשבוניים

יש לממש פעולות חשבוניות לפי הסמנטיקה של שפת C.

הטיפוס המספרי int הינו signed, כלומר מחזיק מספרים חיוביים ושליליים.
הטיפוס המספרי byte הינו unsigned, כלומר מחזיק מספרים אי-שליליים בלבד.
חילוק יהיה חילוק שלמים.

השוואות רלציוניות בין שני טיפוסים מספריים שונים יתייחסו לערכים המספריים עצמם (כלומר, כאילו הערך הנמצא ב-byte מוחזק על ידי int). לכן, למשל, הביטוי

```
8b == 8
```

יחזיר אמת.

יש לממש שגיאת חלוקה באפס. במידה ועומדת להתבצע חלוקה באפס, תדפיס התכנית

```
"Error division by zero"
```

באמצעות הפונקציה print ותסיים את ריצתה.

גלישה נומרית

יש לדאוג שתתבצע גלישה מסודרת של ערכים נומריים במידה ופעולה חשבונית חורגת (מלמעלה או מלמטה) מהערכים המותרים לטיפוס.

טווח הערכים המותר ל-int הוא 0-0xffffffff (כך שf0x7fffffff-0 חיוביים ו0x80000000-0xffffffff שליליים). גלישה נומרית עבור int אמורה לעבוד באופן אוטומטי במידה ומימשתן את התרגיל לפי ההנחיות (כלומר, תתקבל תמיד תוצאה בטווח הערכים המותר, ללא שגיאה).

טווח הערכים המותר ל-byte הוא 0-255. יש לוודא כי גם תוצאת פעולה חשבונית מסוג byte תניב תמיד ערך בטווח הערכים המותר, על ידי truncation של התוצאה (איפוס הביטים הגבוהים בתוצאה).

ביטויים בוליאניים

יש לממש עבור ביטויים בוליאניים short-circuit evaluation, באופן הזהה לשפת C: במידה וניתן לקבוע בשלב מסוים בביטוי בוליאני את תוצאתו, אין להמשיך לחשב חלקים נוספים שלו. כך למשל בהינתן הפונקציה printfoo:

```
bool printfoo() {
      printi(1);
      return true;
}
```

והביטוי הבוליאני:

```
true or printfoo()
```

לא יודפס דבר בעת שערוך הביטוי.

קריאה לפונקציה

בעת קריאה ל-Call, ישוערכו קודם כל הארגומנטים של הפונקציה לפי הסדר (משמאל לימין) ויועברו לפונקציה הנקראת. קוד הפונקציה יקרא באמצעות הפקודה call. בסוף ביצוע הפונקציה תקרא הפקודה ret. סוף הפונקציה הוא סוף רצף הפקודות בבלוק של הפונקציה, גם אם אינו כולל אף פקודת return.

- ניתן להניח שבגוף הפונקציה **לא** תתבצע השמה לתוך פרמטר של פונקציה. דוגמה להשמה של פרמטר של פונקציה:

```
foo(int x) {

      x = 1;

}
```

במידה והפונקציה מחזירה ערך מטיפוס כלשהו (int, byte, bool) והפקודה האחרונה שמתבצעת בה אינה פקודת return, הערך שיוחזר יהיה ערך ברירת המחדל עבור אתחול משתנים מטיפוס זה.

משפט if

בראשית ביצוע משפט if משוערך התנאי הבוליאני Exp. במידה וערכו true, יבוצע המשפט בענף הראשון, ואחריו המשפט שנמצא בקוד אחרי ה-if. במידה וערכו false (ומדובר במשפט if-else) יבוצע המשפט בענף השני, ואחריו המשפט שנמצא בקוד אחרי ה-if.

התנאי הבוליאני של המשפט עשוי לכלול ביטויים מורכבים, לפי המוגדר בתרגיל 3.

משפט while

בראשית ביצוע משפט while משוערך התנאי הבוליאני Exp. במידה וערכו true, יבוצע המשפט, והריצה תחזור לשערוך של Exp. במידה וערכו false, יבוצע המשפט שנמצא בקוד אחרי ה-while.

התנאי הבוליאני עשוי לכלול ביטויים מורכבים, לפי המוגדר בתרגיל 3.

משפט break

ביצוע משפט break בגוף לולאה יגרום לכך שהמשפט הבא שיתבצע הוא המשפט הבא אחרי הלולאה הפנימית ביותר בתוכה ה-break מופיע.

משפט continue

ביצוע משפט continue בגוף לולאה יגרום לקפיצה לתנאי הלולאה הפנימית ביותר בה ה-continue. תנאי הלולאה ייבדק ובמידה והתנאי מתקיים, המשפט הבא שיתבצע הוא המשפט הראשון בתוך אותה לולאה. אחרת הוא המשפט הבא אחרי לולאה זו.

משפט return

במידה וזהו משפט return Exp, יש לקרוא ל-ret כך שיחזיר את Exp.

שימוש בפונקציות ספרייה

ניתן להשתמש בפונקציות printf ו-exit מהספרייה סטנדרטית, ע"י הכרזה שלהם:

```
declare i32 @printf(i8*, ...)
declare void @exit(i32)
```

יש להוסיף הכרזות אלו לקוד המיוצר על מנת שיעבוד כראוי.

פונקציות פלט

קיימות 2 פונקציות בשפת Fanc. הראשונה printi, המקבלת מספר, והשנייה print, המקבלת מחרוזת. עליכם לכלול את המימוש שלהן בקוד שתייצרו. שימו לב שיש לכלול את ההגדרות של @.str specifier ו-@.int_specifier.

מימוש מומלץ לפונקציות הללו ניתן למצוא בקובץ print_functions.llvm המסופק לתרגיל.

ניתן להניח שלא יופיעו רצפי בריחה (escape sequence) במחרוזות מודפסות.

טיפול בשגיאות

תרגיל זה מתמקד בייצור קוד אסמבלי ולא מוסיף שגיאות קומפילציה מעבר לאלה שהופיעו בתרגיל 3. יש לדאוג שהקוד המיוצר ייטפל בשגיאת חלוקה באפס שהוזכרה בפרק הסמנטיקה.

קלט ופלט המנתח

קובץ ההרצה של המנתח יקבל את הקלט מ-stdin.

את תכנית ה-llvm השלמה יש להדפיס ל-stdout. הפלט ייבדק על ידי הפניה לקובץ של stdout ו-stderr והרצה על ידי התוכנית lli.

הדרכה

כדאי לממש את התרגיל בסדר הבא:

1. חישובים לביטויים אריתמטיים. התחילו מחישובים פשוטים והתקדמו לחישובים מורכבים יותר. בדקו אותם בעזרת הדפסות.
2. חישובים לביטויים בוליאניים מורכבים. בדקו אותם בעזרת הדפסות.
3. שמירת וקריאת משתנים במחסנית.
4. רצף של statements.
5. מבני בקרה.
6. קריאה לפונקציות הפלט.
7. קריאה לפונקציות.

מומלץ ליצור llvm program template אליו תוכלו להעתיק קטעי קוד אסמבלי קצרים שיצרתן בשלבי עבודה מוקדמים. כך תוכלו להריץ ולבדוק את הקוד שאתם מייצרים בטרם יצרתם תכנית מלאה.

הוראות הגשה

מסופק לכם קובץ Makefile שאיתו תקומפל ההגשה שלכם. שימו לב כי קובץ ה-Makefile מאפשר שימוש ב-STL. אין לשנות את ה-Makefile.

יש להגיש קובץ אחד בשם ID1-ID2.zip, עם מספרי ת"ז של שתי המגישות. על הקובץ להכיל:

- קובץ flex בשם scanner.lex המכיל את כללי הניתוח הלקסיקלי.
- קובץ בשם parser.y המכיל את כללי הניתוח התחבירי.
- את כל הקבצים הנדרשים לבניית המנתח, כולל קבצים שסופקו כחלק מהתרגיל אם בחרתם להשתמש בהם.

בנוסף, יש להקפיד שהקובץ **לא יכיל** את:

- קובץ ההרצה.
- קבצי הפלט של flex ו-bison.
- קובץ Makefile שסופק כחלק מהתרגיל.

יש לוודא כי בביצוע unzip לא נוצרת תיקיה נפרדת. על המנתח להיבנות על השרת csComp ללא שגיאות באמצעות קובץ Makefile שסופק עם התרגיל. באתר הקורס מופיע קובץ zip המכיל קבצי בדיקה לדוגמה. יש לוודא כי פורמט הפלט זהה לפורמט הפלט של הדוגמאות הנתונות. כלומר, ביצוע הפקודות הבאות:

```
unzip id1-id2.zip
cp path-to/Makefile .
cp path-to/hw5-tests.zip  .
unzip hw5-tests.zip
make
./hw5 < t1.in 2>&1 > t1.ll
lli t1.ll > t1.res
diff t1.res path-to/t1.out
```

ייצור את קובץ ההרצה בתיקיה הנוכחית ללא שגיאות קומפילציה, יריץ אותו, ו-diff יחזיר 0.

הגשות שלא יעמדו בדרישות לעיל יקבלו ציון 0 ללא אפשרות לבדיקה חוזרת.

בדקו היטב שההגשה שלכן עומדת בדרישות הבסיסיות הללו לפני ההגשה עצמה.

שימו לב כי באתר מופיע script לבדיקה עצמית לפני ההגשה בשם selfcheck. תוכלו להשתמש בו על מנת לוודא כי ההגשה שלכם תקינה.

בתרגיל זה (כמו בתרגילים אחרים בקורס) **ייבדקו העתקות**. אנא כתבו את הקוד שלכם בעצמכם.

בהצלחה! ☺