תורת הקומפילציה

תרגיל 5

תורת הקומפילציה
תרגיל 5

ההגשה בזוגות בלבד.

מתרגל אחראי על התרגיל: תומר כהן tomer.cohen@campus.technion.ac.il

שאלות בנוגע לתרגיל יש לשאול בפיאצה של הקורס. לפני ששואלים שאלה יש לבדוק שהיא לא הופיעה בעבר. שאלה שהופיעה בעבר לא תקבל תשובה. הערות והבהרות המופיעות בפיאצה מחייבות.

במידה ויהיו שינויים \ עדכונים בתרגיל הבית, הודעה תפורסם עם עדכון מסמך זה, והשינויים יסומנו בצהוב.

התרגיל יבדק בבדיקה אוטומטית. הקפידו למלא אחר ההוראות במדויק.

1. כללי

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.

שימו לב ! בקבצי bp מופיעה הפונקציה bpatch. סטודנטים הלוקחים את הקורס בסמסטר הנוכחי לא!!!! צריכים להשתמש בה. סטודנטים החוזרים על הקורס ומעוניינים להשתמש בפונקציה יכולים לעשות זאת.

2. 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.

שימו לב !!!! קוד ה llvm שלכם אמור להיות עטוף בפונקציית main ולכן יש לכתוב לבאפר את השורה

"define i32 @main(){"

לפני רצף הפקודות,

ולסיום את השורה

"ret i32 0"

"}"

א. 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.

ב. 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.

אין להשתמש ברגיסטרים לאחסון ערכי ביטויים בוליאניים בזמן שערוך הביטוי (דרישה זו תובהר בפרק הסמנטיקה).

ג. לייבלים (תוויות קפיצה)

ב-llvm יעדים של קפיצות מיוצגים בתור לייבלים: 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 לייבל מתחיל בלוק בסיסי חדש וכל בלוק בסיסי צריך להסתיים בפקודת br או ret הקובע את מבנה גרף הבקרה של התוכנית.

את הלייבלים בפקודות קפיצה של מבני בקרה יש לייצר ולהשלים כפי שנלמד בשיעור. נתונה לכן המחלקה CodeBuffer בקובץ bp.hpp עם מימוש של באפר.

יצירת לייבלים:

נמצאת בה הפונקציה freshLabel() היוצרת לייבל חדש בזהה למה שנלמד בתרגול 7.

לסטודנטים החוזרים על הקורס נמצאת בה הפונקציה genLabel() המקורית (על אף שנראית שונה היא מבצעת אותו דבר כמו הפונקציה מסמסטר קודם, אין צורך לשנות את הממשק בקוד שלכם) הכותבת לייבל חדש לבאפר ומחזירה אותו.

אין חובה להשתמש בהן, ניתן לנהל את הלייבלים שלכם בעצמכם.

עבודה עם המחלקה CodeBuffer

לצורך העבודה עם באפר הקוד נתונה לכם מחלקה CodeBuffer בקובץ bp.hpp. במחלקה תוכלו למצוא מתודות לעבודה עם באפר קוד ומתודה שמדפיסה את באפר הקוד ל-stdout.

המחלקה מממשת את הפונקציות emit בה ניתן להשתמש כפי שראיתם בתרגול 7 קראו את תיעוד הפונקציות במחלקה.

ד. משתנים גלובלים

ניתן לשמור ליטרל מחרוזת כמשתנה גלובלי.

את ליטרל המחרוזת יש להגדיר עם null בסופה (להוסיף לה את התו \00, כמו בדוגמאות שבהרצאה ובתרגול). ניתן להניח כי המחרוזות בתוכניות הבדיקה לא יכילו תוים מיוחדים כמו: \t, \r, \n.

המחלקה CodeBuffer מכילה מתודה להדפסת באפר הקוד, ומכילה בנוסף לכך גם שתי מתודות לטיפול במשתנים הגלובלים של התוכנית: המתודה emitGlobal כותבת שורות לבאפר נפרד, והמתודה printGlobalBuffer מדפיסה את תוכן הבאפר הנפרד.

3. מחסנית

בתרגיל אתם לא נדרשים לנהל את המחסנית מומלץ להקצות בתחילת ה main מקום לכל המשתנים הלוקלים על המחסנית באמצעות הפקודה alloca ובה נתייחס לכל משתנה ללא תלות בטיפוסו כ-i32.

ניתן להניח כי מספר המשתנים הלוקלים ב main קטן מ-50.

~~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.~~

4. סמנטיקה

יש לממש את ביצוע כל ה-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. ניתן להיעזר בדוגמאות מהתרגולים.

א. משתנים

I. 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.

II. 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.

ב. 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.

ו. 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).

ג. 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.

בנוסף, **אין להשתמש ברגיסטרים לתוצאות או תוצאות ביניים של ביטויים בוליאניים**. יש לתרגם אותם לסדרת קפיצות כפי שנלמד בתרגול. לדוגמה, במידה והביטוי הבוליאני הוא ה-Exp במשפט השמה למשתנה, יש להשתמש רק ברגיסטר אחד לתוצאה הסופית כמשתנה ביניים לצורך ביצוע store (שמירה לזיכרון).

רמז – שימוש בפקודה phi יוכל להקל על המימוש במקרים מסוימים, אך איננו מחייב.

- ניתן לשמור ערך של ביטוי בוליאני ברגיסטר במקרים הבאים:
  - כתיבה וקריאה למשתנים
  - חישוב ערך של relop על ידי הפקודה icmp
  - העברה של ערך בוליאני לפונקציה
  - החזרה של ערך בוליאני מפונקציה
- עם זאת, אין לבצע חישובים בוליאנים כדי לחשב תוצאה של פעולות לוגיות - not, and, or, רק עבור התוצאה הסופית.

ד. משפט if

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

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

ה. משפט while

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

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

ו. משפט break

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

ז. 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.

ח. משפט return

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

עבור משפט return יש להדפיס ret i32 0

5. שימוש בפונקציות ספרייה

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

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

```
declare i32 @scanf(i8*, ...)
```

יש להוסיף הכרזות אלו לקוד המיוצר על מנת שיעבוד כראוי.

6. פונקציות פלט

קיימות 3 פונקציות בשפת Fanc. הראשונה printi, המקבלת מספר, והשנייה print, המקבלת מחרוזת והשלישית readi המקבלת מספר. עליכם לכלול את המימוש שלהן בקוד שתייצרו. שימו לב שיש לכלול את ההגדרות של @.str specifier @.int_specifier ו- @.int_specifier_scan.

מימוש מומלץ לפונקציות הללו ניתן למצוא בקובץ print_functions.llvm.

ניתן להניח שלא יופיעו escape sequence במחרוזות המודפסות כדוגמת \n,\r,\t.

7. 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.

8. קלט ופלט המנתח

קובץ ההרצה של המנתח יקבל את הקלט מ-stdin.

את תכנית ה-llvm השלמה יש להדפיס ל-stdout (באמצעות הפונקציות המתאימות במחלקה CodeBuffer). הפלט ייבדק על ידי הפניה לקובץ של stdout ו-stderr והרצה על ידי התוכנית lli.

9. הדרכה

כדאי לממש את התרגיל בסדר הבא:

1. קוד להקצאת רגיסטרים (בדומה ל-freshVar מההרצאה).
2. חישובים לביטויים אריתמטיים. התחילו מחישובים פשוטים והתקדמו לחישובים מורכבים יותר. בדקו אותם בעזרת הדפסות.
3. חישובים לביטויים בוליאניים מורכבים. בדקו אותם בעזרת הדפסות.
4. שמירת וקריאת משתנים במחסנית.
5. רצף של statements.
6. מבני בקרה.
7. קריאה לפונקציות הפלט.
8. ~~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.

מומלץ להיעזר במבני הנתונים של stl. מומלץ לכתוב מחלקות למימוש פונקציונליות נחוצה. כדאי מאוד להיעזר בתבנית העיצוב (design pattern) singleton.

10. הוראות הגשה

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 המגישים. על הקובץ להכיל:

- קובץ flex בשם scanner.lex המכיל את כללי הניתוח הלקסיקלי
- קובץ בשם parser.ypp המכיל את המנתח
- את כל הקבצים הנדרשים לבניית המנתח, כולל *.output שסופקו כחלק מתרגיל 3 וקבצי *.bp שסופקו כחלק מתרגיל זה, אם השתמשתם בהם.

בנוסף יש להקפיד שהקובץ לא יכיל:

- את קובץ ההרצה
- קבצי הפלט של flex ו-bison
- את 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.** הפקודות הבאות יגרמו ליצירת קובץ ההרצה hw5:

```
unzip id1-id2.zip
cp path-to/makefile .
make
```

פלט המנתח צריך להיות ניתן להרצה על ידי הסימולטור. כך למשל, יש לוודא כי תכניות הדוגמה באתר מייצרות פלט זהה לפלט הנדרש. ניתן לבדוק את עצמכם כך:

```
./hw5 < path-to/t1.in >& t1.ll
lli path-to/t1.ll > t1.res
diff path-to/t1.res path-to/t1.out
```

יריץ את המנתח, ייצר קובץ llvm, יריץ את lli עליו ללא שגיאות, ו-diff יחזיר 0.

בדקו היטב שההגשה שלכם עומדת בדרישות הבסיסיות הללו לפני ההגשה עצמה. מומלץ לכתוב גם טסטים נוספים שיבדקו את נכונות המימוש עבור מבני הבקרה השונים.

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

הגשות שלא יעמדו בדרישות לעיל (ובפרט שלא עוברות את ה-selfcheck) יקבלו ציון 0 ללא אפשרות לבדיקה חוזרת.

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

בהצלחה!