תורת הקומפילציה

תרגיל 5

ההגשה ביחידים וזוגות בלבד.

שאלות בנוגע לתרגיל ניתן לשלוח לאיימן במייל אל aiman.ay.28@gmai.com עם "236360_" בתחילת כותרת ההודעה.

לתרגיל יפתח דף FAQ באתר הקורס. יש להתעדכן בו. הנחיות והבהרות שיופיעו בדף ה-FAQ עד יומיים לפני הגשת התרגיל מחייבות. שאלות המופיעות בדף ה-FAQ לא יענו.

התרגיל ייבדק בבדיקה אוטומטית. **הקפידו למלא אחר ההוראות במדויק.**

1. כללי

בתרגיל זה עליכם לממש תרגום ל-LLVM IR עבור שפת FanC שהכרתם בתרגיל בית 3. תחביר השפה הוא כפי שהכרתם ומימשתם בתרגיל בית 3. בתרגיל תממשו בין השאר את השימוש במחסנית למשתנים לוקלים ואת מבני הבקרה (אותם תממשו בשיטת **backpatching**).

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.

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 ללייבל 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 בשיטת backpatching כפי שנלמד בשיעור. נתונה לכן המחלקה CodeBuffer בקובץ bp.hpp עם מימוש של באפר ופונקציה המבצעת backpatching. נמצאת בה הפונקציה genLabel() הכותבת לייבל חדש לבאפר ומחזירה אותו. אין חובה להשתמש בה, ניתן לנהל את הלייבלים שלכם בעצמכם. אפשר להשתמש בפונקציית bp המבצעת backpatching עם כל לייבל שתבחרו (שימו לב לתיעוד בקובץ ה-hpp איך לעבוד עימה). כמו כן, ניתן לכתוב לבאפר גם לייבלים אחרים.

<u>עבודה עם המחלקה CodeBuffer</u>

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

המחלקה מממשת את הפונקציות emit, makelist ו-merge בהן ניתן להשתמש כפי שראיתם בתרגולים 7 ו-8 עם שינויים קלים – קראו את תיעוד הפונקציות במחלקה.

הפונקציות מטפלות ברשימת כתובות בבאפר הקוד. ניתן להשתמש בכתובות אלה לצורך דיבוג הבאפר. שימו לב, ערך החזרה של הפונקציה emit הוא הכתובת אליה כתבתם. זו הכתובת בה עליכם להשתמש לצורך backpatching.

שימו לב כי **בשונה משפת הביניים התיאורטית שלמדתם בשיעור**, ישנה הפרדה בין מיקום בבאפר בו יבוצע backpatching (הערך המוחזר על ידי emit) וכתובת אליה תתבצע קפיצה (לייבל). השפה התיאורטית מניחה כי פקודות נכתבות לבאפר הקוד לאותו המקום בו ייכתבו בזיכרון, ולכן אותה הצבעה לפקודה יכולה לשמש גם כמצביע לפקודה בבאפר שיש לבצע עליה backpatching וגם כיעד קפיצה. בתכנית llvm, לעומת זאת, הכתובות בזיכרון של פקודות – כלומר, הכתובת אליה יש לקפוץ כדי להגיע לפקודה מסויימת – ייקבעו רק כאשר התוכנית תקומפל לשפת מכונה. לכן יעדי קפיצה מסומנים על ידי לייבלים טקסטואליים, אשר יהפכו לכתובת בזיכרון בזמן התרגום לשפת מכונה, בעוד שהמיקום של פקודות לצורך ביצוע backpatching מסומן על ידי מיקומים בתוך ה-CodeBuffer ומוחזר על ידי emit.

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 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. עבור מבני הבקרה יש להשתמש ב-backpatching. 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 שה-0-0x7fffffff חיוביים ו-0x80000000-0xffffffff 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.

ד. 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.

ה. 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.

ו. משפט while

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

1) אם מדובר במשפט while-else יבוצע ה-Statement שנמצא בענף ה-else
2) יבוצע ה-statement אחרי ה-while/while-else

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

ז. משפט break

ביצוע משפט break בגוף לולאה יגרום לכך שהמשפט הבא שיתבצע הוא המשפט הבא אחרי הלולאה הפנימית ביותר בתוכה ה-break מופיע. (לא מבצעים את ה-statement של ה-else אם קיים)

ח. משפט continue

ביצוע משפט continue בגוף לולאה יגרום לקפיצה לתנאי הלולאה הפנימית ביותר בה ה-continue. תנאי הלולאה ייבדק ויבוצעו הפעולות לפי ההוראות בחלק ו' (משפט while) לעיל

ט. משפט return

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

5. פונקציות פלט

קיימות שתי פונקציות פלט בשפת FanC. הראשונה מקבלת פרמטרים מספריים, והשנייה פרמטר מחרוזת. עליכן לכלול את המימוש שלהן בקוד ה-llvm שתייצרו. להלן מימושים מומלצים לשתי הפונקציות, העושות שימוש בפונקציה printf שניתן להשתמש בה אחרי שמצהירים עליה בתוכנית אותה יצרתם. ניתן לשנות אותן **כל עוד האפקט זהה**. שימו לב שבסוף כל הדפסה צריכה להיות ירידת שורה: בדוגמה הבאה זה בא לידי ביטוי בתו ה-'\0A' במחרוזות של ה-specifiers.

לדוגמה, ניתן להצהיר על הפונקציות הנופוצות בשפת C, printf ו-exit באמצעות הפקודות:

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

להגדיר את המשתנים הגלובלים:הפונקציה printi תוגדר כך:

```
define void @printi(i32) {
   call i32 (i8*, ...) @printf(i8* getelementptr ([4 x i8], [4
x i8]* @.int_specifier, i32 0, i32 0), i32 %0)
   ret void
}
```

הפונקציה print:

```
define void @print(i8*) {
   call i32 (i8*, ...) @printf(i8* getelementptr ([4 x i8], [4
x i8]* @.str_specifier, i32 0, i32 0), i8* %0)
   ret void
}
```

6. 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.

7. 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.

8. 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 singleton (design pattern).

9. 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!