מבוא למערכות מחשוב

**עבודה מספר 3.3 – Code generation**

**כללי** – בעבודה זאת תכתבו קומפיילר פשוט המתרגם משפטי השמה בפונקציה לקוד אסמבלי. משפטי ההשמה יוכלו לקבל ביטויים מורכבים, ולכן יש צורך בפישוט הביטויים באמצעות משתנים לוקליים. הקוד המתקבל בסיום הריצה יוכל לעבור תרגום לשפת מכונה על ידי האסמבלר, ואח"כ לרוץ על המכונה שבניתם.

המשימה הינה משימת רשות. עבור סטודנטים שיבחרו שלא לממש את המשימה, משקל העבודות יהיה 12% מהמשקל הסופי, ו-3% הנותרים יעברו למבחן.

הפעולות היחידות המותרות יהיו פעולות חיבור וחיסור. כל הפעולות יהיו בינאריות, כלומר בעלות שני אופרנדים בלבד. הביטויים לא יכללו קריאות לפונקציות (לדוגמה f(x)), או שימוש במערכים.

עליכם לממש את החלקים השונים ב-SimpleCompiler על מנת לקמפל משפטי השמה לקוד אסמבלי. כל משפטי ההשמה יהיו מהתצורה:

let <variable name> = <expression>;,

כאשר expression הוא ביטוי, ו-variable name הוא שם חוקי עבור משתנה.

**שלבי הביצוע:**

1. העתיקו את המחלקות החסרות מהמשימה הקודמת אל תיקיית הפרויקט. ודאו שהקוד מתקמפל. עליכם לפרק את הקוד של Tokenize כך שיטפל בשורה בודדת. כמו כן, המחלקה VarDeclaration עברה שינוי, ולכן עליכם להעתיק את השיטה Parse לתוך המחלקה החדשה.
2. ממשו את השיטה ComputeSymbolTable המקבלת רשימה של משפטי הכרזה ויוצרת טבלת סמלים. אין צורך לשמור טיפוס, מאחר ונשתמש רק ב-Int, ואין צורך לשמור סוג מאחר ונשתמש רק ב-local. על מנת להקל על הבדיקה, יש להכניס את המשתנים האמיתיים ראשונים לטבלה, על פי סדר הופעתם, ואח"כ את המשתנים המלאכותיים, על פי סדר הופעתם.
3. ממשו את הנדרש בשיטה GenerateCode, על מנת ליצור קוד אסמבלי עבור ביטויים פשוטים. השיטה מקבלת משפטי השמה פשוטים (כלומר, מכילים רק ביטוי פשוט יחיד) וממירה אותם לקוד אסמבלי ללא מקרו. השיטה ראשית מחשבת טבלת סמלים ע"פ רשימת משפטי הכרזה. שימו לב, זוהי אינה משימה פשוטה, שכן עלינו לתמוך בפעולות על משתנים הנמצאים בהיסט מסוים מ-LOCAL. נשתמש לכן בשלבים הבאים:
   1. כאשר מחשבים ביטוי המורכב מפעולה חישובית, נעביר ראשית את שני האופרנדים אל שני רגיסטרי אסמבלי שיקראו OPERAND1 ו-OPERAND2.
   2. חישוב הביטוי בהשמה לתוך RESULT (רגיסטר אסמבלי).
   3. העברת הערך מ-RESULT אל המשתנה הלוקלי עם שימוש ברגיסטר ADDRESS.
   4. רצוי לכתוב פונקציות גנריות כגון CopyToOperand1 במקום לשכפל את הקוד פעמים רבות. פונקציות אלו יחליפו את השימוש במקרו שנעשה בכיתה.
   5. יתכנו מימושים שונים של חלק זה. על מנת לבדוק את המימוש, הריצו את הקוד הנוצר על אמולטור ה-CPU, כפי שניתן לראות בדוגמאות, ובדקו שהתוצאה היא כרצוי.
   6. אם נתקלים במשתנה שלא הוגדר על ידי VarDeclaration יש לזרוק exception.
   7. ניתן לראות דוגמא למשימה עד כאן ב-Test1 ,Test2.
4. ממשו את השיטה SimplifyExpression המקבלת משפט השמה עם ביטוי (יתכן מורכב) ומפשטת אותו לסדרה של משפטי השמה פשוטים, כך שכל משפט השמה יכיל מספר, משתנה, או פעולה בינארית על משתנים או מספרים, ללא ביטויים מכוננים. לשם כך נגדיר משתנים מלאכותיים, שיקראו על פי המוסכמה “\_i” כאשר i הוא המספר הסידורי של המשתנה המלאכותי. גם משתנים אלו יש להכניס לתוך משפטי ההשמה. יתכנו פירוקים שונים חוקיים, ואין העדפה לפירוק כזה או אחר.

לדוגמא: המשפט "let x = ((53 - 12) + (467 - 3));" יכול ליצור את שרשרת המשפטים:

let \_1 = (53 - 12);

let \_2 = (467 - 3);

let x = (\_1 + \_2);

המחלקה CPUEmulator מכילה גם שיטה Compute המקבלת סדרת משפטי השמה ומבצעת אותם. ניתן להריץ את השיטה כדי לבדוק האם התוצאה לפני ואחרי הפירוק זהה. דוגמא לכך ניתן לראות ב-Test4.

להזכירכם, העבודה אינה בזוגות. כל העברה של קטעי קוד בין סטודנטים אסורה בהחלט. כל צורה של העתקה תטופל בחומרה המירבית – אל תעתיקו!

את העבודה יש להגיש עד לסוף הסימסטר. לעבודה זאת לא יהיו הארכות! תכננו את הזמן בהתאם.