מטלה 2

בתרגיל זה נעשה מספר ניסויים שמטרתם להכיר את הקומפיילר החכם של C++ ואת הדברים שהוא עושה עבורנו מאחרי הקלעים.

שימו לב: התשובות לרוב השאלות לא נלמדו בהרצאה. נסו לענות עליהן כמיטב יכולתכם ע"י (א) הגיון, (ב) אינטואיציה, (ג) ניסוי וטעיה, (ד) קריאה באינטרנט למשל
כאן: https://www3.nd.edu/~dthain/courses/cse40243/fall2015/intel-intro.html
או כאן: https://youtu.be/bSkpMdDe4g4 (סרטון מעולה, מומלץ מאד).
(ה) אם אתם מסתבכים תבואו לשאול אותנו, התרגיל לא אמור לקחת מעל חצי יום!

הגשה בזוגות. ניתן להוריד את התרגיל כקובץ וורד ולענות בגוף התרגיל.

א. הכנה

ודאו שיש לכם גישה למערכת לינוקס. אפשר להשתמש במחשבים בבניין 9 קומה 3; להתקין על המחשב שלכם: https://itsfoss.com/install-ubuntu-dual-boot-mode-windows/, או להתקין אובונטו על דיסק-און-קי: http://unetbootin.github.io , או להשתמש באתר כגון c9.io.

ודאו שיש לכם קומפיילר טוב ל-C++. למשל, אפשר להתקין clang++-5.0 ע"י הרצת הפקודה הבאה במסוף של אובונטו:

```
sudo apt install clang++-5.0
```

ב. תוכנית מינימלית

כיתבו תוכנית C++ מינימלית:

```
int main () { return 1234; }
```

1. קמפלו והריצו את התוכנית. בידקו את הערך המוחזר ממנה למערכת ההפעלה ע"י:

```
echo $?
```

מהו הערך המוחזר ומדוע?

- מוחזר 210, אם משנים את הערך 1234 למספרים אחרים רואים שגם הערך המוחזר משתנה בהתאם. אחרי כמה ניסיונות מבינים את החוקיות - הערך 210 הוא השארית של 1234 בחלוקה ל-256. מכאן שהערך המוחזר חייב להיכנס לבייט אחד ואם מחזירים ערך גדול יותר הוא מקוצץ.

2. קמפלו את התוכנית לאסמבלר ע"י:

```
clang++-5.0  --assemble  -fno-asynchronous-unwind-tables
        -fno-exceptions -fno-rtti <filename>.cpp
```

(הדגלים המתחילים ב-f אומרים לקומפיילר ליצור קוד אסמבלי נקי יותר - בלי הערות מיותרות).

קיראו את קוד האסמבלי שהתקבל. מה לדעתכם משמעות השורות שמתחילות בנקודה (.)?

- אלה הערות עבור האסמבלר - מעין תיעוד פנימי, כגון שם קובץ-המקור וכד'.

3. איזה מהשורות הללו אפשר למחוק בלי לשנות את התוצאה?

(הערה: כדי לקמפל קובץ בשפת אסמבלי אפשר להשתמש ב-clang++ כרגיל בלי הדגל --assemble).

- ע"י ניסוי וטעיה מקבלים שכמעט את כל השורות אפשר למחוק - השורה היחידה החשובה היא . globl main , בלעדיה מקבלים שגיאה בשלב הקישור (link) על כך שהסמל main לא מוגדר.

4. איזה פעולה באסמבלי משמשת להחזרת מספר שלם כלשהו מתוך פונקציה?

- מחפשים בקוד את המספר 1234 ומוצאים את הפקודה:

```
movl <value>, %eax
```

כאשר value הוא הערך המוחזר. כנראה המוסכמה היא, כשכשפונקציה רוצה להחזיר מספר שלם, היא שמה אותו ברגיסטר שנקרא eax.

ג. משתנים מקומיים

שנו את הקוד בתוך main ל:

```
int i=2, j=3, k=i+j; return k;
```

5. קמפלו וקיראו את הקוד. איפה נמצאים המשתנים i, j, k?

- -4(%rbp), -8(%rbp), -12(%rbp)

6. לפי התשובה לסעיף א, מה לדעתכם המשמעות של %rbp?

- אנחנו יודעים שבשפת ++c משתנים מקומיים נמצאים על המחסנית. מכאן ש-rbp הוא כנראה מצביע לתחילת המחסנית (אולי ראשי-תיבות של register base pointer). הכתובות של המשתנים המקומיים מחושבות ביחס אליו.

7. לפי התשובה לסעיף ב והקוד, מה לדעתכם המשמעות של %rsp?

- רואים שבתחילת הפונקציה main, הערך של rsp מועבר לתוך rbp. מכאן ש-rsp מציין גם-כן מצביע למחסנית (register stack pointer). כנראה, rbp אמור להיות קבוע למשך כל הפונקציה (כי הוא מצביע למשתנים המקומיים שלה), בעוד ש-rsp יכול להשתנות כשדוחפים דברים למחסנית.

8. איזו פקודת אסמבלי משמשת לחיבור מספרים שלמים?

- אפשר לזהות בקוד את הפקודה addl.

9. איזו פקודת אסמבלי משמשת לכפל מספרים שלמים?

- משנים בקוד-המקור את + ל-* ומקבלים באסמבלי את הפקודה imull.

10. איזו פקודת אסמבלי משמשת לחילוק מספרים שלמים?

- משנים את הפעולה ל-/ ומקבלים idivl.

11. איזו פקודת אסמבלי משמשת למציאת שארית בין מספרים שלמים? איך זה מסתדר עם סעיף ו?

- משנים את הפעולה ל-% ומקבלים שוב idivl! איך זה מסתדר? כי חילוק ושארית זה אותה פעולה: כשעושים חילוק ארוך, מקבלים בו-זמנית גם את המנה וגם את השארית. המעבד בנוי כך שהוא שם את המנה ברגיסטר אחד ואת השארית ברגיסטר אחר.

ד. משתנים גלובליים

הוציאו את השורה "int i=2, j=3, k=i+j;" מתוך main והדביקו מעל main:

```
int i=2, j=3, k=i+j;
int main() { return k; }
```

12. קמפלו וקיראו את הקוד. איפה נמצאים המשתנים i, j, k עכשיו?

- עכשיו לכל אחד מהם יש מיוחד וקבוע בזיכרון, עם תוית מיוחדת - הוא כבר לא קשור למחסנית.

13. כמה ואיזה פונקציות נוספו לקוד עכשיו, מלבד הפונקציה main? מדוע הן נוספו?

- אצלי נוספו שתי פונקציות, __cxx_global_var_init ו_GLOBAL__sub_I_c.cpp, נראה שמטרתן היא לאתחל את המשתנים הגלובליים.

ה. פונקציות

כיתבו את התוכנית הבאה:

```
int triple(int x) {   return 3*x;  }
int main() {  return triple(1111); }
```

14. קמפלו וקיראו את הקוד. איזו פקודת אסמבלי משמשת לקריאה לפונקציה?

- callq <function-name>

15. מדוע הפונקציה נקראת triplei ולא רק triple?

- כפי שלמדנו, בשפת C++ ניתן לבצע העמסה של פונקציות - פונקציות שונות עם אותו שם ופרמטרים שונים. כדי שהאסמבלר יוכל להבחין ביניהן, צריך לתת לכל פונקציה שם הכולל את הפרמטרים שלה. במקרה זה יש פרמטר אחד int ולכן הקומפיילר מוסיף i לשם הפונקציה.

16. מה צריך לעשות כדי ששם הפונקציה באסמבלי ישתנה ל-tripled (בלי לשנות את פעולתה)? triplec? triplel?

- ע"י ניסוי וטעיה מגלים ששם הפונקציה משתנה בהתאם כאשר משנים את סוג הפרמטר של triple ל-double, char, long.

17. שנו את סיומת הקובץ מ-cpp ל-c, וקמפלו אותו שוב בתוספת הדגל **-ObjC** (בנוסף לכל הדגלים שהעברתם קודם). הדגל הזה אומר לקומפיילר להתייחס לקובץ כאל תוכנית בשפת Objective C. לצורך התוכנית שלנו, השפה הזאת שקולה לשפת סי שאתם מכירים. מדוע הפונקציה נקראת עכשיו רק triple בלי תוספת?

- בשפת סי אין אפשרות להעמיס פונקציות - יש רק פונקציה אחת עם כל שם. לכן אין צורך להוסיף את סוג הפרמטר לשם הפונקציה, ואפשר פשוט לקרוא לה בשמה.

ו. אינליין

18. היפכו את הפונקציה triple ל-inline כפי שנלמד בשיעור, וקמפלו שוב. מה ההבדלים בקוד האסמבלי שהתקבל?

- אין הבדל משמעותי (ייתכנו הבדלים לא משמעותיים כגון החלפת סדר בין פונקציות).

19. איך זה מסתדר עם מה שלמדנו בכיתה על inline?

- כפי שלמדנו, המילה inline היא רק רמז (hint) לקומפיילר - הוא לא חייב להתייחס לרמז הזה. ואכן בדרך-כלל הוא מתעלם מהרמז הזה.

ז. אופטימיזציות

בסעיף זה יש להוסיף לקימפול את הדגל **-O2** (בנוסף ל assemble--). הדגל אומר לקומפיילר לבצע אופטימיזציות שונות כדי להקטין את הקוד ולקצר את זמן הריצה שלו.

20. הורידו את ה-inline מהפונקנציה triple וקמפלו את התוכנית. איך השפיע הדגל O2 על הפונקציה main?

- הפונקציה התקצרה מאד - היא בכלל לא קוראת ל-triple. הקומפיילר למעשה קרא ל-triple בעצמו, חישב את הערך שהיא אמורה להחזיר, ושם אותו ישירות בתוך main, כדי לחסוך לנו את הקריאה לפונקציה.

21. איך השפיע הדגל O2 על הפונקציה triple? מדוע? כדי להבין טוב יותר, החליפו את המספר 3 במספרים אחרים - 6, 12, 24, 65537... מה עושה הקומפיילר ומדוע?

- הקוד שמכפיל בשלוש (imull) נעלם, ובמקומו הופיעה פקודה חדשה - leal. ע"י החלפת המספר 3 במספרים אחרים מבינים שהפעולה הזאת מבצעת כפל באופן עקיף - למשל במקום לבצע
- `y = x*3`
- היא מבצעת:
- `y = x+x*2`
- בנוסף, אם מחליפים למשל את ה-3 ל-6 הקומפיילר מוסיף פעולה נוספת:
- `addl %edi, %edi`
- כלומר, במקום להכפיל ב-6, הוא קודם-כל מוסיף את המספר לעצמו, ואז מכפיל ב-3.

מה ההסבר לכל השינויים האלה? כנראה שהפעולה imull היא פעולה איטית, והפעולות leal, addl הן פעולות מהירות. במצב אופטימיזציה, הקומפיילר מעדיף להחליף פעולה אחת איטית במספר פעולות מהירות.

22. הוסיפו בחזרה את inline לפונקציה triple וקמפלו שוב. איך השפיעה הוספת inline על הפונקציה main?

- לא השפיעה: היא עדיין כוללת רק את תוצאת הפונקציה ללא קריאה לפונקציה.

23. איך השפיעה הוספת inline על הפונקציה triple?

- הפונקציה נעלמה. כיוון שאמרנו לקומפיילר שאנחנו רוצים שהיא תהיה inline, הוא הבין שאנחנו לא צריכים את גוף הפונקציה בכלל, והחליט לחסוך מקום ולמחוק אותה.

24. נסו לשנות את הקוד של הפונקציה triple כך שיהיה מסובך יותר והקומפיילר לא יוכל "להעלים" אותו. מהו הקוד הכי פשוט שהצלחתם ליצור, שהוא מסובך מדי בשביל האופטימיזציה של הקומפיילר?

- זו שאלה קשה, הקומפיילר הוא חכם ומצליח לצמצם הרבה סוגים של פונקציות. סטודנטים הצליחו "לנצח" את הקומפיילר ע"י פונקציה רקורסיבית כגון עצרת, לולאה אינסופית, פונקציה בתוך פונקציה ועוד.