הטכניון – מכון טכנולוגי לישראל

ארגון ותכנות המחשב

תרגיל 1 - חלק רטוב

המתרגל האחראי על התרגיל: עידו סופר

שאלות על התרגיל – ב- Piazza בלבד.

הוראות הגשה:

- ההגשה בזוגות.
- על כל יום איחור או חלק ממנו, שאינו באישור מראש, יורדו נקודות.
  - ניתן לאחר ב-3 ימים לכל היותר. הקנס יהיה 8\15\20 בהתאם לכמות ימי האיחור.
  - לבקשות מיוחדות להגשה באיחור יש לשלוח מייל למתרגל האחראי על התרגיל עידו סופר.
- הגשות באיחור רגיל (עם קנס) יתבצעו דרך אתר הקורס.
- יש להגיש את התרגיל בקובץ zip לפי ההוראות בעמוד הבא. **אי עמידה בהוראות אלו תעלה לכם בנקודות יקרות**.

הוראות והנחות לפתרון והגשת התרגיל

הוראות כלליות

- התרגיל וכל ההוראות כתובים בלשון זכר/נקבה באופן אקראי אך פונים לגברים ונשים באופן שווה.
- ישנם 5 סעיפים, הפתרון של כל אחד צריך להופיע בקובץ נפרד ex1.asm, ex2.asm וכו' המצורפים לכם להשלמה. ההגשה שלכם צריכה להיות קובץ zip אחד שמכיל את חמשת הקבצים שקיבלתם, בלי שהורדתם מהם כלום וכאשר הוספתם להם רק פקודות אסמבלי ולא שום דבר אחר.

טסטים

- מסופק לכם קובץ בדיקה לתרגיל 4 והוראות כיצד להשתמש בו. הטסט מכסה מקרה בסיסי ומסופק לכם כדי שתדעו כיצד אנו מצפים לקבל את הפתרון (בכל אחד מחמשת התרגילים) וכיצד אנו מתכוונים לבדוק אותם.
- מומלץ ביותר לבצע טסטים נוספים בעצמכם לכל חמשת התרגילים, על בסיס מבנה הטסט הזה.

חוקים לכתיבת הקוד

- בקוד שאתם מגישים אין לשנות שום ערך בזיכרון מלבד אלו שנאמר לכם מפורשות.
- עליכם לבצע כל חישוב אפשרי ברגיסטרים. ביצוע חישובים על ערכים בזכרון הוא לא יעיל.
- בכל סעיף הניחו כי בתוויות הפלט הוקצה מספיק מקום.
- על כל תווית שתשמשו בה לצורך מימוש להסתיים בארבעת התוים _HW1. מה שמופיע לפני מומלץ כמובן שיהיה קשור למטרת התווית לצורך נוחות שלכם ולצורך קריאות של הקוד. לדוגמא –

```
loop: #this is a bad label name
loop_HW1: #this is a good label name
```

זכרו: אי עמידה בכל אחת מההוראות הנ"ל עלול לגרור הורדת ניקוד לתרגיל ולהשפיע על זכותכם לערעור במקרים מסוימים.

חוקים לשאלות בפיאצה (או – למה לא עונים לי ?)

- נא לקרוא את כל המסמך לפני שבכלל נכנסים לפיאצה, בטח ובטח לפני ששואלים משהו.
- נא לקרוא את השאלות ששאלו לפניכם. אתם יכולים רק להרוויח מכך וסביר שזה גם יקרה.
- כל מקרה קצה שלא מופיע מפורשות בהוראות הסעיף אפשרי. התרגיל נכתב בקפידה ותחת ווידוא שלכל מקרה קצה כנ"ל יש תוצאה מצופה שניתנת להבנה מהוראות התרגיל ומקרי קצה הם כמובן, חלק מהקורס וחלק מהחיים כמתכנת. כמובן שהסגל אנושי ויכול לפספס משהו וכל שאלה בפיאצה תקרא ובמידה ועומדת בכל הסעיפים בחלק זה של ההוראות, גם תענה.
- אנו לא מדבגים לכם. זה חלק מהקורס כמו בכל הקורסים בפקולטה.
- שאלות פרטניות ובפרט כאלה הכוללות תמונות קוד, נא לכתוב בשאלה פרטית שלא חשופה לשאר הכיתה.

אופן בדיקת התרגיל וכתיבת טסטים בעצמכם

כאמור, כל חמשת התרגילים יבדקו בצורה דומה. לכן, עקבו אחר ההוראות שיתוארו להלן, כאשר תרצו לבדוק את הקוד שלכם לפני ההגשה. חבל שההגשה שלכם לא תהיה לפי הפורמט ותצטרכו להתעסק עם ערעורים ולאבד נקודות סתם.

בכל תרגיל, תקבלו קובץ asm המכיל .text. section בלבד. עליכם להשלים את הקוד שם, אך לא להוסיף sections נוספים לקובץ בעת ההגשה (<u>ההגשה חייבת להכיל **section text** בלבד. בפרט *לא* להכיל את **section data**</u>).

אז איך בכל זאת תוכלו לבדוק את התרגיל שלכם? זה פשוט. לתרגיל 4 מצורף טסט בודד בתיקייה tests. הבדיקה היא בעזרת הקובץ run_test.sh.

שימו לב שכל הטסטים על קבצי הקוד שלכם ירוצו עם timeout (כפי שגם מופיע ב-run_test.sh) ולכן כתבו אותם ביעילות. קוד שלא ייכתב ביעילות ולא יסיים את ריצתו על טסט מסוים עד ה-timeout, ייחשב כקוד שלא עמד בדרישות הטסט. בפרט הקוד ייבדק באותו מגבלת זמן שמופיעה בטסטים, אתם תראו שאין באמת צורך לאלגוריתמים פורצי דרך כדי לעמוד בהם.

הריצו את הקובץ run_test.sh באופן הבא:

./run_test.sh <path to asm file> <path to test file>

לדוגמה, עבור התרגיל הראשון והטסט שלו ומתוך התיקייה שמכילה את קבצי הקוד של הסעיפים ואת תיקיית הטסטים:

./run_test.sh ex4sample_test ex1.asm

הערה:

ייתכן ולפני הרצת קבצי sh על המכונה, תצטרכו להריץ את הפקודה –

chmod +x <your .sh file>

כתיבת טסטים בעצמכם

לכל תרגיל תוכלו לכתוב טסט, שהמבנה שלו דומה למבנה של ex4sample_test , עם שינוי תוויות ובדיקות בהתאם.

תרגיל 1 - Memory Access (12 נק')

עליכם לממש את ex1 המוגדרת בקובץ ex1.asm.

בתרגיל זה תקבלו שלושה תויות מקור ולפניכם שתי תוויות יעיד:

Adress – כתובת של האיבר הראשון במערך של integers.
Index – אינדקס במערך של איבר שמעניין אותנו. (unsigned int)
length – כמות האיברים במערך. (unsigned int)
Legal – האם הכל תקין (עליכם לשים 1 או 0)
num – עליכם לשים פה את האיבר שבאינדקס הנתון במערך, אם ורק אם שמתם בLegal 1.

<u>שימו לב: אין לשאול בפיאצה את השאלה "מה זה אומר שהכל תקין", זה בעצם כל התרגיל. תחשבו מה הבעיות האפשריות בהתאם למה שנלמד בתרגולים 1-3 ולמה שאתם יודעים והגיוני במדעי המחשב. בכללי אין כנראה שום שאלה על סעיף זה שתיענה בפיאצה, אז אל תהיו מופתעים. תקראו ותחשבו.</u>

תרגיל 2 – סיווג מידע (16 נק')

עליכם לממש את ex2 המוגדרת בקובץ ex2.asm.

בתרגיל זה תקבלו בתווית size שהיא unsigned quad מספר חיובי ממש, שיציין כמה בתים של מידע יש בתווית data. בתווית יעד type עליכם לשים מספר לפי ההוראות הבאות:

1 אם data הוא מחרוזת פשוטה (הבתים שם מציינים תווי אסקי של אותיות, מספרים, פסיקים, נקודות, סימני שאלה וקריאה, רווחים, והתו האחרון הוא null terminator)

2 אם data הוא מחרוזת מדעית (כל תו אסקי שניתן לכתוב. בפרט, כל תווי האסקי שמספרם הדיצמלי הוא בן 32 ל126 כולל, חוץ מהתו האחרון שהוא null terminator)

3 אם מספר הבתים מתחלק ב8 ואף quad ברצף אינו 0.

4 אחרת.

תרגיל 3 – עץ עשיר בעלים (16 נק')

עליכם לממש את ex3 המוגדרת בקובץ ex3.asm.

בתרגיל זה תקבלו תווית אחת בשם root. ותמלאו תווית יעד rich ב1 אם root מייצג עץ עשיר בעלים, ואפס אחרת, לפי ההוראות הבאות:

תחילה נגדיר עץ למטרות התרגיל – גרף מכוון בו לכל הצמתים חוץ מהשורש יש בדיוק קשת אחת שנכנסת אליהם, ולשורש אפס. כמו כן נגדיר עלה – צומת שלא יוצאות ממנו קשתות.

הגרף בתרגיל שלנו ייוצג ע"י רשימת שכנים, בצורה הבאה - הdata section מורכב מlabel עבור כל צומת בגרף , ובכל לייבל כזו יש מערך של מצביעים לצמתים אחרות שיוצא קשת מהצומת הזו אליהם עם אפס שמסמן את סוף המערך, לדוגמה:

```
root: .quad a, b, 0
a: .quad c, 0
b: .quad d, 0
c: .quad 0
d: .quad 0
```

כאשר הלייבל היחיד שנתונה לכם היא root ואתם לא יכולים להניח שום דבר על ה data section.

אם התוצאה של החלוקה של כמות הצמתים בגרף בכמות העלים בגרף קטנה או שווה ל3 – מדובר בעץ עשיר בעלים.

תרגיל 4 – סדרה ברשימה (20 נק')

עליכם לממש את ex4 המוגדרת בקובץ ex4.asm.

בתרגיל זה תעבדו עם רשימה מקושרת דו כיוונית שהdata של האיברים בה מייצג סדרה מספרית.
כל איבר ברשימה בתרגיל זה יורכב מ3 חלקים, כמתואר ב-struct הבא:

```
struct Node {
    struct Node prev;
    int data;
    struct Node next;
}
```

רמז1: struct node זה מצביע.
רמז2: int זה מספר עם סימן.
רמז3: ניתן לראות דוגמה לרשימה בתרגול 3.
רמז 4: ניתן לדעת שאיבר כלשהו הוא הראשון ברשימה אם הprev שלו 0, ואחרון אם הnext 0.

בתרגיל זה תקבלו את שתי התוויות הבאות:

- nodes – רשימה באורך 3 עם מצביעים לאיברים כלשהם ברשימה. ניתן להניח שאכן יתקבל מצביע חוקי לאיבר ברשימה.
- result – תווית יעד (1 byte)

עבור כל איבר ברשימת nodes שקיבלתם, חלקו את הסדרה שלכם לשני חלקים – מהאיבר הראשון ועד האיבר שלפני האחד שקיבלתם, ומהאיבר אחרי האיבר שקיבלתם ועד האחרון. עבור כל אחד מהשלוש איברים בדקו, האם כל אחת מהתת סדרות היא ממוינת (עולה או יורדת, לא ממש – כלומר אפשר שני איברים סמוכים זהים)

בתווית היעד שימו מספר בין 0 ל3 – בכמה מהמקרים קיבלתם ששני התת סדרות כשלעצמן בנפרד, ממוינות.

בתרגיל זה ניתן טסט לדוגמא.

תרגיל 5 – סדרה מדרגה שנייה (36 נק')

עליכם לממש את ex5 המוגדרת בקובץ ex5.asm.

תזכורת: סדרה חשבונית היא סדרת מספרים בה ההפרש בין כל שני מספרים עוקבים הוא קבוע, וסדרה הנדסית היא סדרת מספרים בה המנה בין כל שני מספרים עוקבים היא קבועה.

בתרגיל זה תקבלו סדרה של integers (מספרים בעלי סימן) במערך, בתווית series ואת גודל הסדרה בתווית size מסוג unsigned int . עליכם לשים בתווית seconddegree 1 אם הסדרה שהתקבלה היא סדרה מדרגה שנייה ו0 אחרת, לפי ההגדרה הבאה:

סדרת מספים היא סדרה מדרגה שנייה, אם סדרת ההפרשים בין האיברים העוקבים שלה **<u>או</u>** סדרת המנות בין האיברים העוקבים שלה הן כשלעצמן סדרה חשבונית **<u>או</u>** סדרה הנדסית. כלומר יש פה 4 מקרים אפשריים. במקרה שאחד מהמקרים מתקיים, תשימו 1 בתווית היעד, ו 0 אחרת.

על כל המקרי הקצה להיות מטופלים לפי הברירה המתמטית ההגיונית. אתם לא בסמסטר ראשון.

הערות אחרונות

איך בונים ומריצים לבד?

כאמור בתחילת התרגיל, נתון לכם קובץ טסט לתרגיל 1 וקובץ run_test.sh. אתם יכולים לשנות את הקובץ של תרגיל 1 ולהשתמש במבנה שלו כדי לבדוק תרגילים אחרים באותה הצורה, כפי שהוסבר קודם לכן בהקדמה לתרגיל.
כדי להריץ, או לנפות שגיאות:

as ex1.asm ex1sample_test -o my_test.o	#run assembler (merge the 2 files into one asm before)
ld q1.o my_test.o -o ex1	#run linker
./ex1	#run the code
gdb ex1	#enter debugger with the code

את הקוד מומלץ לדבג באמצעות gdb. לא בטוחים עדיין איך? על השימוש ב-gdb תוכלו לקרוא עוד במדריך באתר הקורס

קלי קלות! (ואם לא – אנחנו זמינים בפיאצה)

שימו לב: למכונה הוירטואלית של הקורס מצורפת תוכנת sasm, אשר תומכת בכתיבה ודיבוג של קוד אסמבלי וכן יכולה להוות כלי בדיקה בנוסף לgdb. (פגשתם אותה בתרגיל בית 0).
כתבו בcmd:

```
sasm <path_to_file>
```

כדי להשתמש ב-SASM לבנייה והרצת קבצי ה-asm, עליכם להחליף את שם התווית _start בשם main (זאת מכיוון ש-sasm מזהה את תחילת הריצה על-ידי התווית main. **אל תשכחו להחזיר את _start לפני ההגשה!).**

בדיקות תקינות

בטסט אתם תפגשו את השורות הבאות

```
movq $60, %rax
movq $X, %rdi      # X is 0 or 1 in the real code
syscall
```

שורות אלו יבצעו exit(X) כאשר X הוא קוד החזרה מהתוכנית – 0 תקין ו-1 מצביע על שגיאה.

בקוד שאתם מגישים, אסור לפקודה syscall להופיע. קוד שיכיל פקודה זו, יקבל 0.

ניתן גם לדבג באמצעות מנגנון הדיבוג של SASM במקום עם gdb, אך השימוש בו על אחריותכם (**כלומר לא נתמך על ידנו ולא נתחשב בבעיות בעקבותו בערעורים**. שימו לב לשוני בין אופן ההרצה ב-SASM לאופן ההרצה שאנו משתמשים בו בבדיקה שלנו).