הטכניון – מכון טכנולוגי לישראל

ארגון ותכנות המחשב

תרגיל 1 - חלק רטוב

המתרגל האחראי על התרגיל: רון רובינשטין.

שאלותיכם במייל בעניינים מנהלתיים בלבד, יופנו רק אליו.

כתבו בתיבת subject: רטוב 1 אתמ.

שאלות בעל-פה ייענו על ידי כל מתרגל.

הוראות הגשה (לקרוא!!!):

- ההגשה בזוגות.
- שאלות הנוגעות לתרגיל יש לשאול דרך הפיאצה בלבד.
- על כל יום איחור או חלק ממנו, שאינו בתיאום עם המתרגל האחראי על התרגיל, יורדו 5 נקודות.
  - ניתן להגיש לכל היותר באיחור של 3 ימים (כאשר שישי ושבת נחשבים יחד כיום אחד בספירה).
  - הגשות באיחור יש לשלוח למייל של אחראי התרגיל בצירוף פרטים מלאים של המגישים (שם+ת.ז).
- הוראות הגשה נוספות מופיעות בסוף בתרגיל.

נושא התרגיל: תכנות אסמבלי, פקודות, שיטות מיעון, בקרת זרימה

בתרגיל זה תכתבו כמה תוכניות פשוטות באסמבלי.

הקדמה

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 (**ההגשה חייבת להכיל section text בלבד. בפרט *לא* להכיל את section data**).

לצורך פתרון התרגיל, אנא השתמשו ב-labels שתקבלו כקלט ופלט ובנוסף עשו שימוש ברגיסטרים בלבד. כלומר, אזור ה-data section הוא חלק מהטסטים (מוזמנים להציץ בקבצי הטסטים המצורפים).

אז איך בכל זאת תוכלו לבדוק את התרגיל שלכם? זה פשוט. לכל תרגיל מצורף טסט בודד בתיקייה tests. תוכלו לכתוב לעצמכם טסטים נוספים באותו הסגנון, כלומר חלק קוד נוסף ו-data section שיתווספו (בשרשור) לסוף הקובץ שלכם, ויבצעו את הבדיקות הנדרשות עם הקלטים הנתונים.

הבדיקה היא בעזרת הקובץ run_test.sh.

הריצו קובץ זה באופן הבא:

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

לדוגמה, עבור התרגיל הראשון והטסט שלו:

```
./run_test ex1.asm tests/test_1_1
```

הערה: ייתכן ותצטרכו להריץ את הפקודה:

```
chmod +x <your .sh file>
```

לפני הרצת קבצי sh על המכונה.

תרגיל 1

עליכם לממש את ex1 המוגדרת בקובץ ex1.asm.

בתרגיל זה תקבלו תווית num עם מספר בגודל מילה מרובעת (8 bytes). עליכם לשים בתווית countBits, שגודלה מילה כפולה (4 bytes), את מספר הביטים הדולקים במספר num.

למשל, במספר 0x202=0b001000000010 יש שני ביטים דולקים ולכן countBits יכיל את המספר 2.

עליכם לעשות זאת מבלי לפגוע בתוכן של num – כלומר, על num להכיל בסוף ריצת הקוד שלכם, את הערך שאיתו הוא התחיל את הריצה.
כלומר, עבור הקלט:

```
num: .quad 0x202
countBits: .zero 4
```

מצופה מהקוד שלכם לפעול כך שבסופו countBits יכיל את המספר 2 ו-num יכיל את המספר 0x202.

<u>הערה:</u> במהלך ריצת התוכנית מותר לשנות את הערך של num, כל עוד הוא יכיל בסופו של דבר את ערכו המקורי.

תרגיל 2

עליכם לממש את ex2 המוגדרת בקובץ ex2.asm.

קלט:

source,destination - שני מערכים

num - כמות בתים להעתיק

עליכם להעתיק num בתים המתחילים בכתובת source אל תוך destination.
(בדיוק כמו memmove)

num בגודל 4 בתים, ובמידה והוא שלילי (לפי 2s compliment), אין להעתיק אף איבר.

הניחו כי יש בכתובת destination מספיק מקום בשביל המידע

לשם פשטות ניתן להניח שהlabel של source יופיע לפני הlabel של destination ב data section בכל הטסטים.

תרגיל 3

עליכם לממש את ex3 המוגדרת בקובץ ex3.asm.

בתרגיל זה תקבלו מערך של מילים כפולות (bit 32).

עליכם לחשב את הממוצע של כלל האיברים במערך, מהאיבר הראשון עד לאיבר הראשון שערכו 0.
את הממוצע יש לכתוב לתווית avg, שגם היא מילה כפולה.

(ניסוח ברור יותר: עליכם לחשב את הממוצע של האיברים במערך, עד לאיבר הראשון שערכו אפס.
הכוונה ב"כלל האיברים במערך", אינו כל האיברים במערך, אלא כל האיברים במערך הנמצאים
מהאיבר הראשון ועד האפס הראשון)

שימו לב, במידה והממוצע אינו שלם, עליכם להחזיר ערך שלם תחתון שלו.

<u>הערה:</u> המספרים במערך הם חסרי סימן (unsigned). ניתן להניח כי קיים איבר 0 במערך.

<u>שימו לב:</u> ממוצע של 0 איברים הוא 0.

לדוגמה:
עבור המערך

```
arr: .int 6,12,9,8,0
```

נקבל

```
avg = 8
```

תרגיל 4

עליכם לממש את ex4 המוגדרת בקובץ ex4.asm

קלט:

- arr - מערך של מספרים חסרי סימן בגודל מילה כפולה.
- n - גודל המערך (מספר חסר סימן בגודל מילה כפולה).

עליכם למצוא את תת הסדרה המונוטונית יורדת ממש הארוכה ביותר במערך arr.

החזירו את האינדקס של תחילת הסדרה בתווית begin ואת אורכה בתווית len.

דוגמא:

```
arr = 6,4,1,9,7,6,3
```

⇒ begin = 3, len = 4

<u>הערה:</u> אם יש כמה סדרות באותו אורך, החזירו את הראשונה מביניהם.

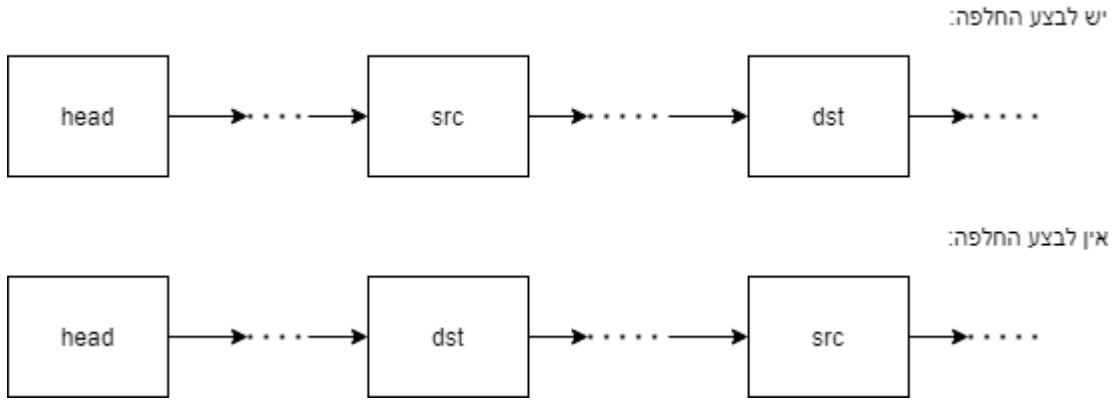
<u>תרגיל 5</u>

עליכם לממש את ex5 המוגדרת בקובץ ex5.asm
קלט:

- head - מצביע לתחילת רשימה מקושרת.
- src, dst - שני ערכים להחלפה (מילים מרובעות).

עליכם להחליף בין שני האיברים המכילים את הערכים שבתווית src ובתווית dst, אם ורק אם הצומת המכיל את src, נמצא לפני הצומת המתאים ל-dst ברשימה (src, dst לאו דווקא סמוכים).

לשם בהירות מצורף ציור להמחשה:



<u>הערה טובה:</u> ניתן להניח שsrc,dst מופיעים בדיוק פעם אחת ברשימה.

<u>הערה רעה:</u> לא ניתן להניח כי head שונה מ-dst ו-src. (כלומר לא ניתן להניח שהכתובת השמורה בhead אינה הכתובת של צומת המכילה את הערך של src או dst)

<u>איך רשימה מקושרת מיוצגת בזיכרון:</u>

head הינה מילה מרובעת, שערכה היא הכתובת של ראש הרשימה.
כל איבר ברשימה הוא צמד של שתי מילים מרובעות, המילה הראשונה היא הערך, המילה השניה היא מצביע לאיבר הבא.
בדוגמה המצורפת (נמצאת ב-test_5_1):

- head מכיל את הכתובת A
- A מכיל את הערך 5 ואת הכתובת B
- B מכיל את הערך 4 ואת הכתובת NULL (קרי – סוף הרשימה)

המצב הרצוי בהינתן src = 5, dst =4:

- Head מכיל את הכתובת B
- A מכיל את הערך 5 ואת הכתובת NULL
- B מכיל את הערך 4 ואת הכתובת A

<u>שימו לב 1:</u> אסור לכם להשתמש בתוויות A, B. מותר להשתמש רק ב-head!

<u>שימו לב 2:</u> במידה ו- src = 4, dst =5 בדוגמה מעלה, **לא** היינו מבצעים החלפה כי הצומת המכיל את הערך 4 מופיע לאחר הצומת עם הערך 5 ברשימה.

עוד מידע רלוונטי:

-איך בונים ומריצים?

שימו לב כי פורמט הגשה תקין אינו מכיל את המשתנים עליהם אנחנו עובדים. כפי שנאמר בתחילת התרגיל, ניתן לבצע בדיקות באמצעות הוספת טסטים והרצת run_test.sh.

run_test.sh למעשה משרשר לקוד שכתבתם קוד בדיקה עם המשתנים המתאימים ומריץ אותו.
הרצת run_test.sh שקולה להוספת המשתנים ובדיקות ישירות לקוד שלכם והרצת הפקודות הבאות לבניה ידנית (דוגמא עבור תרגיל 1):

```
as ex1_merged_with_test.asm -o q1.o
```

```
ld q1.o -o ex1
```

כדי להריץ, או לנפות שגיאות:

```
./ex1
```

```
gdb ex1
```

קלי קלות! (ואם לא – בואו לשעת קבלה!)

<u>שימו לב:</u> למכונה הוירטואילת 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!).

זאת כמובן בנוסף להוספת קטע .data. section מתאים לשאלה בדומה לנתון לכם בטסטים לדוגמה.

-בדיקות תקינות

בטסטים 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.

את הקוד מומלץ לדבג באמצעות gdb. לא בטוחים עדיין איך? תקראו את המדריך שמצורף לתרגיל בית 0 ואם צריך, בואו לשעות הקבלה. ניתן גם לדבג באמצעות מנגנון הדיבוג של SASM, אך השימוש בו על אחריותכם (שימו לב לשוני בין אופן ההרצה ב-SASM לאופן ההרצה שאנו משתמשים בו בבדיקה שלנו).

הוראות הגשה

אם הגעתם לכאן, זו בהחלט סיבה לחגיגה. אך בבקשה, לא לנוח על זרי הדפנה ולתת את הפוש האחרון אל עבר ההגשה – חבל מאוד שתצטרכו להתעסק בעוד כמה שבועות מעכשיו בערעורים, רק על הגשת הקבצים לא כפי שנתבקשתם. אז קראו בעיון ושימו לב שאתם מגישים את כל מה שצריך ורק את מה שצריך.

שימו לב שאתם מגישים את הקבצים שלכם עם text section (ואת שורת global) בלבד!

עליכם להגיש את הקבצים בתוך zip אחד:

hw1_wet.zip

בתוך קובץ zip זה יהיו חמישה קבצים:

- ex1.asm
- ex2.asm
- ex3.asm
- ex4.asm
- ex5.asm

בדיקת תקינות ההגשה:

נעשה זאת בעזרת הקובץ check_submission.sh. אופן ההרצה? ודאו כי בתיקייה איתו נמצאים גם הקובץ run_test.sh ותיקיית tests עם חמשת הטסטים הבסיסיים בתוכה. כעת הריצו את הפקודה:

./check_submission.sh <path to ‘hw1_wet.zip’ submission file>

אם הכל עבר כשורה (בדקו את feedback.txt שנוצר), אנא הגישו את הקובץ hw1_wet.zip ורק אותו.

זוג שלא יגיש קובץ הגשה תקין - על אחריותכם בלבד!