הטכניון – מכון טכנולוגי לישראל

ארגון ותכנות המחשב

תרגיל 1 - חלק רטוב

המתרגל האחראי על התרגיל: עידו סופר

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.

תרגולים רלוונטיים: עד תרגול 3 (כולל). תרגולי הכיתה ההפוכה אינם הכרחיים לתרגיל הרטוב והתרגולים המכינים (2 ו-3) מספיקים.

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.

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.

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:

```
./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.

***תנסו תמיד לבצע חישובים על רגיסטרים ועם מינימום גישות לזכרון. זה תכנות נכון ויעיל.

תרגיל 1 (15 נק')

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.

דוגמא:

עבור הקלט:

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

מצופה מהקוד שלכם לפעול כך שבסופו countBits יכיל את המספר 2 ו-num יכיל את המספר 202x0.

תרגיל 2 (24 נק')

עליכם לממש את ex2 המוגדרת בקובץ ex2.asm.

קלט:

source, destination – שתי כתובות זיכרון (מה גודל התוויות האלו?)

num - כמות בייטים להעתיק (num בגודל 4 בייטים)

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

במקרה ש num שלילי (לפי שיטת המשלים ל-2), יש להעתיק את תוכנו (באורך 4 בייטים) לdestination (**ולא** שום חלק מsource).

הניחו כי יש בכתובת destination מספיק מקום בשביל המידע.

תרגיל 3 (20 נק')

עליכם לממש את ex3 המוגדרת בקובץ ex3.asm.

עליכם לחשב את הכפולה המשותפת הקטנה ביותר (LCM) של המילים המרובעות a ו-b ולרשום את תוצאת החישוב במילה המרובעת c. ניתן להניח שגודל הLCM הוא לכל היותר 64 ביט.

כלומר, עבור הקלט:

```
a: .quad 0x6
b: .quad 0x4
c: .zero 8
```

מצופה מהקוד שלכם לפעול כך שבסופו c יכיל את המספר 12.

בתרגיל זה המספרים הם חיוביים ממש ($a, b > 0$) ולכן גם חסרי סימן (unsigned).

תרגיל 4 (20 נק')

עליכם לממש את ex4 המוגדרת בקובץ ex4.asm.

בתרגיל זה עליכם לממש הוספה של איבר לעץ חיפוש בינארי.

בעץ חיפוש בינארי, לכל איבר בעץ מתקיימת התכונה הבאה – כל האיברים בתת העץ השמאלי שלו קטנים ממנו, וכל האיברים בתת העץ הימני שלו גדולים ממנו. שימו לב שתת העץ יכול להיות ריק.

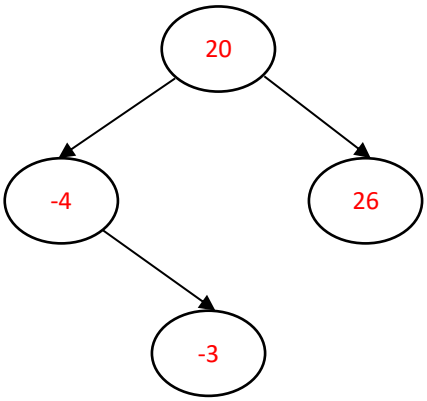
כל איבר בעץ בתרגיל זה יורכב מ3 חלקים, כמתואר ב-struct הבא:

```
struct Node {
    long data;
    struct Node *lson;
    struct Node *rson;
}
```

למשל, נתונה לכן הדוגמה של עץ חיפוש בינארי תקין, כאשר root מצביע על האיבר הראשון בעץ (השורש).

```
root:
  .quad   A
A:
  .quad   20
  .quad   B
  .quad   C
B:
  .quad   -4
  .quad   0
  .quad   D
C:
  .quad   26
  .quad   0
  .quad   0
D:
  .quad   -3
  .quad   0
  .quad   0
```

שמתארת את העץ הבא:



אז איך תתבצע הוספת איבר? אתם תקבלו label נוסף, ששמו new באופן הבא:

```
new_node:   .quad ????, 0, 0
```

ועליכם להוסיף אותו לעץ.

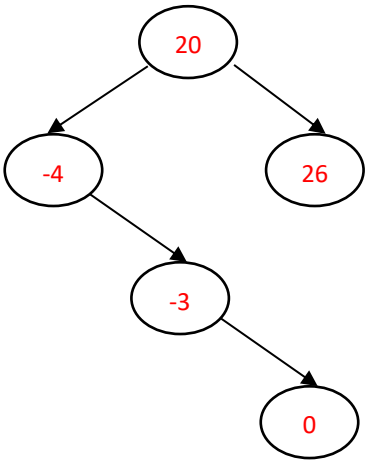
במידה והערך של new (שמסומן בדוגמה הנ"ל כ-????) כבר קיים בעץ, העץ לא ישתנה.

למשל, עבור העץ הנתון מעלה נדגים שתי הוספות.

<u>הוספה מהצורה:</u>

```
new_node:   .quad 0, 0, 0
```

יגרור שהעץ החדש ייראה כך:



לעומת זאת, <u>הוספה מהצורה הבאה:</u>

```
new_node:   .quad 20, 0, 0
```

יגרור שהעץ החדש ייראה בדיוק כמו העץ המקורי, כי 20 כבר קיים בעץ.

הנחות והערות:

- תזכורת: עץ הוא גרף קשיר ללא מעגלים.
- לכל צומת בעץ יש לכל היותר 2 בנים.
- המספרים (תוכן הצומת בעץ) הם בעלי סימן ובגודל 8 bytes.
- התווית שמצביעה על הצומת הראשי של העץ תקרא root, אך אין התחייבות על השמות של שאר התוויות בעץ הנתון.
- אם root הוא NULL (מצביע לכתובת 0), הצומת new_node צריך להיות שורש העץ.
- אין לשנות את מבנה העץ, מלבד הוספת האיבר החדש (בפרט, אין לבצע איזונים למיניהם).

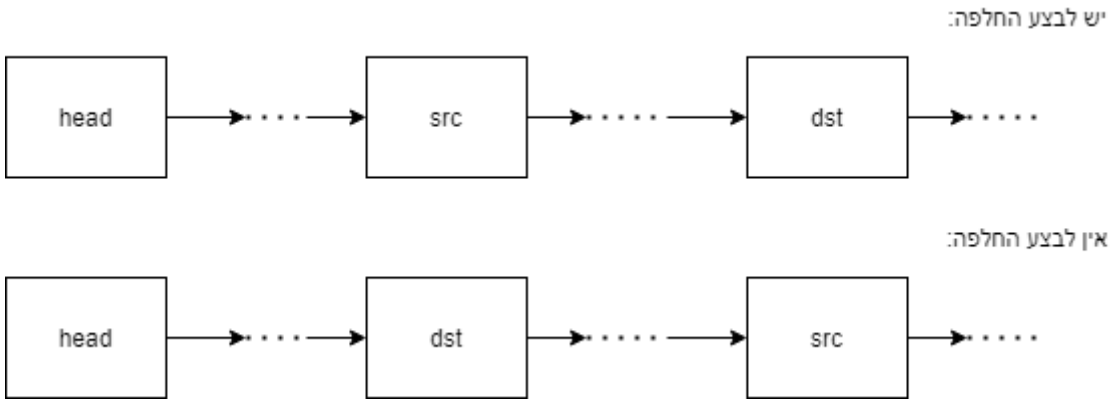
תרגיל 5 (21 נק')

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 המכיל את dst ברשימה (src, dst לאו דווקא סמוכים).

לשם בהירות מצורף ציור להמחשה:



הערות:

1. ייתכן src==dst, ואז אין צריך לבצע דבר.
2. אם src או dst לא קיימים ברשימה, אז אין צורך לבצע דבר.

בשני המקרים הנ"ל התוכנית צריכה להסתיים בהצלחה (ללא קריסה) והרשימות ללא שינוי.

בדומה לדוגמה מתרגול 3 (כיתה הפוכה), נדגיש איך רשימה מקושרת מיוצגת בזיכרון בתרגיל זה:
head הינה מילה מרובעת, שערכה היא הכתובת של ראש הרשימה.
כל איבר ברשימה הוא צמד של שתי מילים מרובעות, המילה הראשונה היא הערך (value), המילה השניה היא מצביע לאיבר הבא (next).
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

שימו לב 1: אסור לכם להשתמש בתוויות A, B (לא מובטח שהן יופיעו בטסטים כלל). מותר להשתמש רק ב-head!
שימו לב 2: במקרה בו 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!)

שימו לב: 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!