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 ועל כל יום איחור יורדו 5 נקודות.
  - 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 חמישה 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, מבלי שהורדתם מהם כלום וכאשר הוספתם להם אך ורק פקודות אסמבלי ולא שום דבר אחר.

טסטים

- מסופק לכם קובץ בדיקה לתרגיל 4 והוראות כיצד להשתמש בו. הטסט מכסה מקרה בסיסי ומסופק לכם על מנת שתדעו כיצד אנו מצפים לקבל את הפתרונות (בכל אחד מחמשת התרגילים) וכיצד הם יבדקו.
- מומלץ לבצע טסטים נוספים בעצמכם לכל חמשת התרגילים, על בסיס מבנה טסט זה.

חוקים לכתיבת הקוד

- בקוד שאתם מגישים אין לשנות שום ערכים בזיכרון מלבד אלו שנאמר לכם מפורשות.
- עליכם לבצע כל חישוב אפשרי ברגיסטרים. ביצוע חישובים על ערכים בזיכרון אינו יעיל.
- בכל סעיף הניחו כי בתוויות הפלט מוקצה מספיק מקום ואל תניחו כלום על תוויות המקור שלא כתוב במפורש.
- על כל תווית שתשמשו בה לצורך המימוש להסתיים בארבעת התווים _HW1. כמובן מומלץ כי מה שמופיע לפני כן יהיה קשור למטרת התווית לצורך נוחותכם 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 וכמובן להשפיע על זכותכם לערעור במקרים מסוימים.

חוקים לשאלות

- נא לקרוא את כל המסמך יש בו תשובות לרוב השאלות שתחשבו ולא תחשבו עליהן.
- מפעם לפעם, על אחריותכם לקרוא שאלות ששאלו בפיאצה, סביר להניח ששאלו שאלות שיכולות לעזור לכם.
- מקרי קצה, כמו בחיים האמיתיים של מתכנתים, הם מקרים אפשריים, ולכן כל מקרה קצה, גם כזה שלא מופיע מפורשות בהוראות הסעיף, הוא מקרה אפשרי. התרגיל נכתב בקפידה ותחת ווידוא שלכל מקרה קצה כנ"ל יש תוצאה מצופה שניתנת להבנה מהוראות התרגיל.
- עבור מקרי קצה שלא מצוינים במפורש קיימת ברירה מחדל הגיונית. למשל, אם אפשר להתייחס לקלט 0 באותה מידה שאפשר להתייחס לקלט חיובי, תתייחסו אליו כך.
- צוות הקורס לא יעזור בדיבוג הקוד, לא בפיאצה ולא בשעות קבלה. זה הוא חלק מהקורס.
- שאלות פרטניות ובפרט כאלה הכוללות תמונות קוד, נא לכתוב בשאלה פרטית שלא חשופה לשאר הכיתה. שאלות שיכולות להיות רלוונטיות לעוד אנשים, בבקשה לשאול בפיאצה מול כולם. קיימת אופציה לשאול אנונימי.

אופן בדיקת התרגיל וכתיבת טסטים בעצמכם

כל חמשת התרגילים יבדקו בצורה דומה. לכן, עקבו אחר ההוראות שיתוארו להלן, על מנת לבדוק את הקוד שלכם לפני ההגשה. הפורמט המתואר מחייב ואי עמידה בו יוביל להורדת נקודות (שימו לב - לא בהכרח נוכל להתחשב בכך בערעור).

בכל תרגיל, תקבלו קובץ asm המכיל .text. section בלבד. עליכם להשלים את הקוד שם. אין 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. כאמור, לנוחותכם, צירפנו לתרגיל 4 טסט בודד בזיפ של קבצי הסטודנטים, היעזרו בו. הבדיקה היא בעזרת הקובץ run_test.sh שגם קיבלתם.

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 באותה מגבלת זמן שמופיעה בrun_test.sh, 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, הריצו מתוך התיקייה שמכילה את קבצי הקוד של הסעיפים ואת תיקיית הטסטים את שורת הפקודה הבאה:

```
./run_test.sh ex4_test ex4.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 - Bit Check (8 נק')

עליכם לממש את ex1 המוגדרת בקובץ ex1.asm.

בתרגיל זה תקבלו שתי תוויות מקור ותווית יעד אחת:

Num1 – מספר. (Int) (4 בתים)
Num2 – מספר. (Int)
Bit Check – Bool . (בית אחד) כאן תשימו 1 אם בשני המספרים שקיבלתם דולקים אותה כמות ביטים, ו 0 אחרת.

תרגיל 2 - Little Endian (18 נק')

עליכם לממש את ex2 המוגדרת בקובץ ex2.asm.

בתרגיל זה תקבלו שלוש תוויות מקור ותווית יעד אחת:

Adrress – כתובתו של האיבר הראשון במערך (כתובת) (8 בתים)
Type – גודלו של כל איבר במערך, בבתים, unsigned. (בית 1)
Length – כמות האיברים במערך. (unsigned int) (4 בתים)
LittleEndianResult – לכאן עליכם להעתיק את המערך בצורה הבאה – עבור כל איבר במערך, לפי גודל האיבר בבתים שנתון לכם, עליכם להפוך את סדר הבתים. לפניכם דוגמא עבור Type=Length=3 וכתובת התחלתית 100:

Before	Address	100	101	102	103	104	105	106	107	108
	Value	0x10	0x20	0x30	0x40	0x50	0x60	0x70	0x80	0x90

After	Address	100	101	102	103	104	105	106	107	108
	Value	0x30	0x20	0x10	0x60	0x50	0x40	0x90	0x80	0x70

תרגיל 3 – Binary Tree Search (20 נק')

עליכם לממש את ex3 המוגדרת בקובץ ex3.asm.

לפניכם ההגדרה של עץ בינארי ממוין:

In computer science, a **binary search tree** (**BST**), also called an **ordered** or **sorted binary tree**, is a rooted binary tree data structure with the following invariant: Data of each node within the tree is greater than all the Data values in the respective node's left subtree and less than the ones in its right subtree

בתרגיל זה תקבלו עץ בתווית root (כלומר תווית זו מכילה מצביע) שמצביעה לצומת שהיא שורש העץ. כל צומת בעץ היא Struct Node המוגדר האופן הבא:

```
struct Node {
    struct Node* LeftSon;
    int Data;
    struct Node* RightSon;
}
```

כאשר כל אחד מבין המצביעים בתוך הstruct Node לעיל, מקיים אחד מבין התנאים הבאים:

1. מצביע לstruct Node נוסף.
2. מכיל 0 המסמל כי אין לצומת הנוכחי בן מתאים.

הגישה לשדות הstruct היא לפי הסדר בו מתואר הstruct לעיל, כלומר כפי שראיתם בתרגולים.

בתווית node מסוג struct node אתם תקבלו צומת, עם ערכי זבל התחלתיים במצביעים של הבנים שלה, ו data רלוונטי. עליכם להוסיף אותו לעץ ולסדר את המצביעים בהתאם או להשאיר את העץ ללא שינוי אם המספר קיים בעץ. שימו לב שאין צורך באיזון או בגלגולים (למי מכם שלא מכיר את המילים האלו, מעולה, אין בהן צורך). עבדו עם ההגדרה שרשומה לעיל. זכרו - לא חייב להיות בדיוק אפס או שני בנים לכל צומת, אפשר גם אחד.

תרגיל 4 – סדרה ברשימה (24 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 זו יורכב משלושה חלקים, כמתואר ב-struct הבא:

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

רמז 1: struct node הוא מצביע.(8בתים)
רמז 2: unsigned int הוא מספר ללא סימן ובתרגיל זה ערכו תמיד גדול ממש מ0. (4בתים)
רמז 3: 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:

- node – מצביע לאיבר ברשימה מסוג ה struct שנתון לעיל
- Result – תווית יעד (1 byte)

עליכם לבדוק אם שינוי של הdata של הnode שקיבלתם לכל מספר שהוא, יכול להפוך את הסדרה כולה לחשבונית או הנדסית. עליכם להחזיר בresult מספר לפי המפתח הבא:

1. אם אפשרי להפוך אותה לסדרה חשבונית ולא להנדסית החזירו את הערך 1.
2. אם אפשרי להפוך אותה לסדרה הנדסית ולא לחשבונית החזירו את הערך 2.
3. אם שני המקרים לעיל אפשריים החזירו את הערך 3.
4. אם אף אחד מהמקרים לעיל אפשרי החזירו את הערך 0.

שימו לב – כל האיברים בסדרה יהיו כפולה שלמה של המספרים לפניהם ולכן <u>אין צורך</u> לבדוק שאריות.

שימו לב 2 – עבור תרגיל זה סיפקנו טסט לדוגמא.

תרגיל 5 – בדיקת מחרוזות (30 נק')

עליכם לממש את ex5 המוגדרת בקובץ ex5.asm.

בתרגיל זה תקבלו תווית command שהיא כתובת של מחרוזת null-terminated. עליכם לעדכן את תווית היעד, result, שגודלה בית 1, באופן הבא: אם המחרוזת הנתונה מציינת פקודה חוקית בשפת C, הציבו בתווית היעד את הערך 1. אחרת הציבו בה את הערך 0.

פקודה חוקית בשפת C היא אחת המקיימת את אחד המקרים הבאים (כאשר מצוין "רווחים" , הכוונה לרווחים או tabים):

1. מורכבת מארבעה חלקים – מחרוזת ללא רווחים, אחריה מחרוזת נוספת ללא רווחים, לאחר מכן התו "=" ולאחר מכן מחרוזת נוספת ללא רווחים, כאשר בין כל אחד מהחלקים הללו יכולה להופיעה כל כמות שהיא של רווחים. לדוגמא int x = 17
2. כמו 1 בלי המחרוזת הראשונה. לדוגמא x=37
3. מורכבת משני חלקים – מחרוזת ללא רווחים ואחריה מחרוזת כלשהי (שבה יכולים להופיע רווחים) שהתווים הראשון והאחרון שלה הם סוגר שמאלי וימני בהתאמה (ובין הסוגריים הללו יכולים להיות עוד סוגריים). בין שני חלקים אלו יכולה להופיעה כל כמות שהיא של רווחים. לדוגמא func (param)
4. מחרוזת אחת ללא רווחים. לדוגמא break

אין צורך לוודא שיש בסוף המחרוזת אחד מהתווים הבאים – } ;

כלומר המחרוזות שתקבלו לא באמת יהיו בסוף פקודות חוקיות בשפת סי, על מנת להקל את הבדיקות שלכם.

הערות אחרונות

איך בונים ומריצים לבד?

כאמור, נתון לכם קובץ טסט לתרגיל 4 וקובץ run_test.sh. אתם יכולים לשנות את הקובץ הטסט של תרגיל 4 ולהשתמש במבנה שלו כדי לבדוק תרגילים אחרים באותה הצורה.

כדי להריץ, או לנפות שגיאות:

```
as ex4.asm ex4.sample_test -o my_test.o        #run assembler (merge the 2 files into one asm before)

ld q4.o my_test.o -o ex4                       #run linker

./ex4                                          #run the code

gdb ex4                                        #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. בהתחלה זה קשה, אבל זה יהיה שווה את זה! (ואם הסתבכתם – אנחנו זמינים בפיאצה).

שימו לב – למכונה הווירטואלית 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).