

ארגון ותכנות המחשב

תרגיל בית 3 (רטוב)

המתרגל האחראי על התרגיל: בועז מואב.

הנחיות:

- שאלות על התרגיל ב- Piazza בלבד.
- ההגשה בზוגות.
- על כל יום איחור או חלק ממנו, שאינם באישור מראש, יורדו 5 נקודות.
 - ניתן לאחר ב-3 ימים לכל היתר.
 - הגשות באיחור יבוצעו דרך אתר הקורס.
- את התרגיל יש להגיש באתר הקורס בקובץ zip.
- תיקונים לתרגיל, אם יהיו, יופיעו ממורכרים.

חלק א – שגרות, קונבנציות ומה שביניהם (75 נק')

מבוא

בחלק א' של תרגיל הבית נນם כפל מטריצות. נעשה זאת בשלושה שלבים, כאשר כל אחד מהשלבים יבדק בנפרד. התרגום, בכל שלביו, מתיחס למטריצות המכילות מספרים שלמים בלבד שאינם חורגים מגבולות הייצוג של 32 ביט.

פונקציות לIMPLEMENTATION בתרגום

בעת נספוק הסברים על שלוש הפונקציות שתממשו בתרגום זה **באסטטבלי**.

יותר ואך מומלץ להשתמש בפונקציות שכבר מישתם בתור פונקציות עזר לIMPLEMENTATION הפונקציות הבאות, אך עם זאת חשוב לשימוש לב שכל פונקציה תיבדק בפני עצמה.

get_element_from_matrix

חתימת הפונקציה הראשונה לIMPLEMENTATION:

```
int get_element_from_matrix(int* matrix[], unsigned int n, unsigned int row,
                            unsigned int col);
```

קלט: הפונקציה מקבלת מבצע בשם **matrix** (פרמטר ראשון) למערך דו-ממדי (מטריצה) בעל **n** (פרמטר שני) עמודות. כמו כן, מקבל הפונקציה שני אינדקסים - **row** (פרמטר שלישי) ו-**col** (פרמטר רביעי).

פלט: היא תחזיר את האיבר בשורה **row** ועמודה **col** במטריצה **matrix**.

שימוש: העמודה והשורה נתונים בצורה **zero based**!!!

הנחות: המיקום **[matrix][row][col]** קיים וחוקי במטריצה ובפרט **col > n** (לא צריך לבדוק).

inner_prod

חתימת הפונקציה השנייה לIMPLEMENTATION:

```
int inner_prod(int* mat_a[], int* mat_b[], unsigned int row_a, unsigned
                int col_b, unsigned int max_col_a, unsigned int max_col_b);
```

קלט: הפונקציה מקבלת שני מבצעים למערכות דו-ממדיים (**mat_a** (מטריצות) ו-**mat_b** (מטריצות ראשוני ו-**mat_b** (פרמטר שני)).

למטריצה **mat_a** יש **max_col_a** ולחטירה **mat_b** יש **max_col_b** (פרמטר שלישי) עמודות. בנוספ', היא מקבלת מספר שורה במטריצה **mat_a**, שנותנה בפרמטר **row_a** (פרמטר שלישי) ומספר עמודה

במטריצה **mat_b**, שנותנה בפרמטר **col_b** (פרמטר רביעי).

פלט: הפונקציה תחזיר את **המכפלה הפנימית** בין השורה **a_row** לבין העמודה **b_col**.

שימוש: גם כאן האינדקסים של העמודה והשורה נתונים בצורה **zero based**.

הנחות: גם כאן אפשר להניח שהשורה **a_row** והעמודה **b_col** אין חורגות מגבולות ממדים המטריצה. **ניתן גם להניח**

שהממדים מתאימים (המכפלה הפנימית חוקית).

matrix_multiplication

חתימת הפונקציה השלישית לIMPLEMENTATION:

```
int matrix_multiplication(int* res[], int* mat_a[], int* mat_b[], unsigned
                           int m, unsigned int n, unsigned int p, unsigned int q);
```

קלט: הפונקציה מקבלת שני מבצעים למערכות דו-ממדיים (**mat_a** (מטריצות ראשוני ו-**mat_b** (פרמטר שלישי)).

המטריצה הראשונה מממד **n × m** (פרמטרים רביעי ו חמישי) והמטריצה השנייה מממד **p × q** (פרמטרים שישי ושביעי). בנוספ', מקבלת הפונקציה מבצע לעמרכ דו-ממד של היעד, מטריצת היעד **res** (פרמטר ראשון).

פלט: אם המטריצות בממד תקין לצורך ביצוע כפל המטריצות **mat_a · mat_b** – המטריצה תחזיר במטריצת היעד **res** את תוצאת המכפלה וכערך חרזה תחזיר 1 (**true**). אחרת, לא תבצע דבר ותחזר 0 (**false**).

שימוש: כמו כן, ניתן ואך מומלץ להשתמש בפונקציה **set_element_in_matrix**, שנותנה לכם בקובץ

o_hw3_aux.h המצורף לתרגום. הסבר נוספת על הפונקציה יופיע מיד.

הנחה: המטריצה `res` מוגדרת באופן תקין, והוקצת לה מספיק זיכרון בהתאם לממדים הצפויים שלה.

פונקציית עזר נתונות

לתרגיל זה נתונה פונקציית העוז `set_element_in_matrix` שחתימתה היא:

```
void set_element_in_matrix(int* matrix[], unsigned int num_of_columns,
                           unsigned int row, unsigned int col, int value);
```

הfonקציה מקבלת מצביע בשם `matrix` (פרמטר ראשון) לערך דו-ממדי (מטריצה) בעל `num_of_columns` (פרמטר שני) עמודות. בנוסף, היא מקבלת שני אינדקסים `row`-`col`, שורה (פרמטר שלישי) ועמודה (פרמטר רביעי), וגם את `value`, ערך (פרמטר חמישי) שאותו היא تعدכן במטריצה במקום המבוקש.

כלומר, היא תבצע (רעיון): `matrix[row][col] = value`.

שימוש: גם כאן האינדקסים של העמודה והשורה נתונים בצורה `zero based`.

הערה: הfonקציה שומרת על קונבנציית הקיירות `System` לארכיטקטורת `i686-x86` כפי שנלמדה בקורס. מותר לכם להשתמש בfonקציה זו ומובהך שהיא תהיה קיימת גם בזמן הנטסים (בקובץ `aux_hw3.o`).

דרישות מימוש

את כל המימושים תשלימו בקובץ `S.students_code`. קובץ `S` הוא קובץ אסמבלי, המתאים ל-`gcc`. אין הבדל אמיתי בין לבין קובץ `asm`¹ (ויש שיטענו, בצדק, שדווקא קבצי `S` מתאימים יותר מ-`asm` לקורס).

הערות חשובות לגבי מימוש התרגיל:

1. שמרו על הקונבנציות!!! בתרגיל זה נכתב קוד שמשלב אסמבלי -C ולבן קוד שלא ישמר על קונבנציות, יכשל בטסמים.

2. אסור לכם להוסיף קבצים בלבד `S.students_code`. בפרט לא את `o.aux_hw3`.

3. אסור לכם להוסיף משתנים בלבד `data` (אותו קיבלתם נתון).

a. אם ברצונכם להשתמש במשתנים מקומיים, אתם מבונים יכולם (ואף מומלץ לעשות זאת) – אך תצטרכו לעשות זאת לפי הקונבנציות שנלמדו בקורס.

4. מומלץ להיעזר ב-GDB בעת דיבוג הקוד. מדריך לשימוש בדיבאגר GDB זמין באתר הקורס.

5. אנחנו ממליצים את הקבצים שלכם בצורה שגוררת עליהם את ההגבלה הבאה – אסור להשתמש בשיטת מיעון אבסולוטית, או בקבוע שהוא בתובות.

בפרט, בשאותם יוצרים לעצמכם טסמים אסור להוסיף את הדגלים fno-pie או no-pie לשורת הקימפוף (בהמשך הקורס נלמד על הדגלים האלה ובניון להוספת הדגלים קשורה לאיסור המדווח).

6. אני ודאו שהתוכנית שלכם יוצאה (מסתיימת) באופן תקין, דרך `main` של קובץ `main` שהבודקה שקוראת לפונקציה שלכם, ולא על ידי `exit` `syscall` שלכם, במקרה שבו השתמשתם באחד (וכמובן שגם לא בעקבות קירסת הקוד²). הערה זו נכתבה בdam ביטים (של קוד של סטודנטים מסמסטרים קודמים).

על מנת לוודא את ערך החזירה של התוכנית, תוכלם להשתמש בפקודת `bash`: `$? echo $? (תזכורת: ערך החזירה של התוכנית, במידה ויצאה בצורה תקינה, הוא הערך ש-main שמחזירה ב-return האחרון שלה).`

כתבו קוד סביר (מבחינת עילוות). קוד לא יעיל בצורה חריגה עלול להיכשל בטסמים.

8. אם הכל עובד בשורה, אתם יכולים לעבור לחלק ב' של תרגיל הבית, ולאחריו לחלק ג', שהוא בסך הכל הוראות הגשה לתרגיל כולם (שימוש לב שאותם מגישים את שני החלקים יחד!).

¹ <https://stackoverflow.com/questions/34098596/assembly-files-difference-between-a-s-asm>



חלק ב – פסיקות (25 נק')

מבוא

לפני שאתם מתחילה את חלק זה, אנאנו ממליצים לכם לחתור על תרגול וסדנה 6 ולראות שאתם יודעים לענות על השאלות הבאות – מה זה TID? מהן הפוקודות sidt ו-idt? מהן מכילות כניסה-OUT? מהי שגרת טיפול בפסיקה? באילו הרשותת היא רצתה? מהי חילוקת האחריות בין שגרת הטיפול לבין המעבד? באיזו מהנסנית משתמש? איך היא נראה בכל שלב? במה זה תלוי? בעת, קראו את כל השלבים בחלק זה לפניהם שתתחלו לעבוד על הקוד.

בתרגילים זה נרצה לכתבו שגרת טיפול בפסיקת המעבד המתבצעת כאשר מבצעים פקודה לא חוקית (כלומר, כאשר המעבד מקבל opcode שאינו מוגדר בו).

- כאשר המעבד מקבל קידוד פקודה שאינו חוקי, המעבד קורא לשגרת הטיפול בפסיקה ב-TID.
- שגרת הטיפול בLINPACK שולחת SIGSEGV לתובנית שביצעה את הפקודה הלא חוקית. למשל:

<https://github.com/torvalds/linux/blob/16f73eb02d7e1765ccab3d2018e0bd98eb93d973/arch/x86/kernel/traps.c#L321>

נרצה לשנות את קוד הkernel כך ששגרת הטיפול בפסיקה תשתנה (שאלה למחשבה – למה חייבים לשנות את קוד הkernel?) ונעשה זאת באמצעות .kernel module

מה תבצע שגרת הטיפול החדש?

שגרת הטיפול בפסיקה שלנו (שאותה אתם הולכים למשב באסמבי בעצמכם, בקובץ `asm_handler.asm`, תיקרא my_ili_handler) ותבצע את הדברים הבאים:

1. בדיקת הפקודה שהובילה לפסיקה זו. הנחות:

- הניחו כי הפקודה השגיה היא פקודה של אופקוד בלבד. ככלומר, לפני ואחריו opcode השגוי אין עוד ביטים (אין prefix legacy, אין REX).
- لكن, אוריך הפקודה השגיה הוא באורך 3 bytes. בתרגיל זה הניחו כי אוריך האופקוד השगוי הוא לכל היותר 2 ביטים.

2. קריאה לפונקציה `do_what_to` עם ה-`byte` האחרון של האופקוד הלא חוקי, כפרמטר.

- היזכרו בחומר של קידוד פוקודות:

- אם האופקוד אין מתחיל ב-0x0, הוא באורך byte אחד.
- אחרת (בן מתחיל ב-0x0), אם הוא אין מתחיל ב-A או 0xF38 או 0xF3, אז הוא באורך 2 ביטים. לבן, הניחו כי הביט השני באופקוד אינו 0x38 או 0x0 (אין צורך לבדוק זאת).

דוגמאות:

- עברו האופקוד 0x27, שהינה פקודה לא חוקית בארכיטקטורת 64-68x, נבצע קריאה ל-`what_to_do` עם `0x27`.
- עברו האופקוד 0x04, גם לא חוקית, נבצע קריאה ל-`what_to_do` עם הפרמטר 0x04.

3. בדיקת ערך החזירה של `do_what_to` אם הוא 0 – חרזה מהפסיקה, בן שהתוכנית תוביל להמשך לחוץ (תציג לפקודה הבאה לביצוע מיד לאחר הפקודה הסוררת) וערכו של גיסטר rd % הינה פסיקה מסוג `fault`. חשבו מה זה אומר על

ערך של גיסטר rd? בעית החזרה משגרת הטיפול ושינו אותו בהתאם.

- שים לב #1: שימו לב ש-`opcode`-`volume` הינה פסיקה מסוג `fault`. המדריך על הפסיקה שלנו, ב כדי

לוזיא את תשובהיכם ל"שים לב #1" וגם כדי להחליט האם יש או לא.

- שים לב #3: do_what_to הינה שגרה שתיתנת על דען בזמן הבדיקה. אין להניח לביבה דבר, מלבד חתימתה (כלומר – שם השגרה, טיפוס פרמטר הקלט וטיפוס ערך החזרה).

- אחרת (הוא 0) – העברת השליטה לשגרת הטיפול המקורי.

³ "עולם האמיתי" אסור לשנות ערכיהם של גיסטורים וצריך להחזיר את מצב התוכנית כפי שקיבלתם אותם. בגין אתם נדרשים כן

לשנות ערך של גיסטר, בן שמצב התוכנית לא יהיה כפי שהוא בשחרור הפסיקה. זה בסדר, זה לצורך התרגיל ☺

<https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325384-sdm-vol-3abcd.pdf>⁴

לפני תחילת העבודה – מה קיבלתם?

בתרגיל זה תעבדו על מכונה וירטואלית דרך `Qemu` (בתוך המכונה הווירטואלית - VirtualMachine). על המכונה זו, אנחנו נריץ `kernel module` שיבצע את החלתת שגרת הטיפול לו שמיימותם בעצמכם. היה וקוד רץ ב-`kernel mode` (0), במקרה של תקלת מערכת ההפעלה תקרו. אך זה לא נראה! עלייכם פשוט להפעיל את `Qemu` מחדש.

לרשומכם נמצאים הקבצים הבאים בתיקייה 2 part:

- `initial_setup.sh` – הריצו סקורייפט זה לפני כל דבר אחר. סקורייפט זה מכין את המכונה הווירטואלית לריצת `Qemu`. עליכם להריץ אותו פעם אחת בלבד (לא יקרה כלום אם תריצו יותר, אך זה לא נחוץ).
- יכול להיות שתצטרכו להריץ את הפקודה הבאה, לפני ההרצה (בגלל הרשאות):
- `chmod +x initial_setup.sh`
- `compile.sh` – הריצו סקורייפט זה בכל פעם שתרצו לkompile את הקוד ולתען אותו (עם המודול המקומפל) למכונה הווירטואלית של `Qemu` (ישמו לב: עליכם לצאת מ-`Qemu` קודם).
 - גם כאן ייתכן ותזדקקו להרצה של `chmod` באותו אופן כמו בסעיף הקודם.
- `start.sh` – הריצו סקורייפט זה כדי להפעיל את המכונה הווירטואלית של `Qemu`, לאחר שkimפלו את תיקיית `code` ויעננטם אותה אל המכונה הווירטואלית של `Qemu`.
 - גם כאן ייתכן ותזדקקו להרצה של `chmod` באותו אופן כמו בסעיף הקודם.
- `filesystem.img` – המכונה הווירטואלית אותה תריצו ב-`Qemu`.
- `makefile` – קבצי הקוד שנכתבו, חלק מהמודול (והיא זו שתkomפל ותורץ לבסוף ב-`Qemu`) וה-`ili_handler.asm`, `ili_main.c`, `ili_utils.c`, `inst_test.c`, `Makefile`

איך הבל מתחבר – כתיבת המודול

בתיקיה `code` סיפקנו לבן מספר קבצים:

- `inst_test.c` – simple code example that executes invalid opcode. Use it for basic testing.
- `ili_main.c` – initialize the kernel module – provided to you for testing.
- `ili_utils.c` – implementation of `ili_main`'s functionality – **YOUR JOB TO FILL**
- `ili_handler.asm` – exception handling in assembly – **YOUR JOB TO FILL**
- `Makefile` – commands to build the kernel module and `inst_test`.

ممשו את הפונקציות ב-`c.ili_utils`, כך שהשגרה ב-`c.ili_handler` תיראה כאשר מנוטים לבצע פקודה לא חוקית. איך? Well, זהו לב התרגיל, אז נסו להזכיר בחומר הקורס. כיצד נקבעת השגרה שנתקראת בעת פסיקה? פועל בהתאם. שימו לב כי ב-`c.ili_utils` אנו רוצים לגשת לחומרה. מהי הדרך לשנות קוד כאשר אנו רוצים לבצע פקודות ב-level low? לאחר מכן, ממשו את הפונקציה `my_ili_handler` ב-`asm.ili_handler` שתשבע את מה שהוגדר בשלב II.

⁵ למי שלא מכיר את המונח `kernel module`, ביל' פאניקה (Pi panic) זה רע, אבל זה עוד יותר רע בקורס. פאניקה! בדיסקו זה דוחוק בסדר) – מדובר בדרך להוסיף ל kernell kod בזמן ריצה (ניתן להוסיף ל kernell kod ולקומפל לאחר מכן את כל kernell החדש, אך כאן לא הזמן ולא המקום לה. למעשה, נכתב קוד שירוץ ב-`kernel mode` ולבן יהיה בעל הרשות מלאות. אנו נדרש לה – הרי אם רוצים לשנות את קוד kernell.

זמן בדיקות - הרצה המודול

לאחר שסיימתם לבתוב את המודול, בצעו את השלבים הבאים:

1. הריצו את `compile.sh`. כדי לкомפלט קוד הkernel ולהכיןו למוכנת ה-QEMU.
2. הריצו את `start.sh`. כדי לפתח מוכנה פנימית באמצעות QEMU.
 - a. התעלמו מכל המל שיפוי, כולל הערות בצבעים אדומים וירוקים. זה תקין.
 - b. לאחר העילה – משתמש: root, סיסמה: root. בעת אתם בתוך ה-QEMU וכל השלבים הבאים מתיחסים לריצת QEMU.
3. באתם בתוך ה-QEMU. כדי להריץ את הקוד `inst_test.asm`, עם הפקודה הלא חוקית (ולקבל הודעה שגיאה בהתאם). ניתן גם להריץ את `bad_inst_2.asm` כדי להריץ את הקוד ב-`asm`.
4. נאנו גם לטעון את המודול שלכם (ודאו שהוא נטען ע"י הריצת `dmesg`).
5. הריצו `insmod ili.ko` כדי לקבל התנהגות שונה, מכיוון שהפעם ה-`handler` שלכם נקבע.

שיםו לב שלאחר ביצוע שלב 1 קיבלם את האזהרה הבאה:

```
root@ubuntu18:~# ./modules
WARNING: could not find /home/student/Documents/atam2/part2/.ili_handler.o.cmd
or /home/student/Documents/atam2/part2/ili_handler.o
CC       /home/student/Documents/atam2/part2/ili_mod.o
```

יש להתעלם מזהירה זו. שגיאות אחרות עלולות להציגן על בעיה מהותית ואף לא יצירת הקבצים הרלוונטיים (בעיות בנייה, בעיות קישור ועוד), لكن שימו לב מהן הערות שמצוינות בעת הרצה `compile`. גם בעית עליית ה-QEMU יוחז הרבה טקסט במסך, בצבעים משתנים של אדום, ירוק וצהוב. לא להיבהל, זה תקין.

דוגמת הרצה תקינה ב-QEMU (עם הטיטים `what_to_do_inst_test_2` ו- `inst_test` שסופק לכם כדוגמה):

```
root@ubuntu18:~# ./bad_inst
start
Illegal instruction
root@ubuntu18:~# insmod ili.ko
root@ubuntu18:~# ./bad_inst
start
root@ubuntu18:~# echo $?
35
root@ubuntu18:~# rmmod ili.ko
rmmod: ERROR: ../libkmod/libkmod.c:514 lookup_builtin_file() could not open built-in file '/lib/modules/4.15.0-60-generic/modules.builtin.bin'
root@ubuntu18:~# ./bad_inst_2
start
Illegal instruction
root@ubuntu18:~# insmod ili.ko
root@ubuntu18:~# ./bad_inst_2
start
Illegal instruction
```

(`what_to_do` ממחירה את הקלט שלו פחותות 4. בטעט הראשון הפקודה הלא חוקית היא `0x27`, אך ערך החזרה הוא `0x23`, שזה גם ערך הייצאה של התוכנית, כי כך נכתב הטיט⁶, אך \$? echo 35. בטיט השני, הפקודה הלא חוקית היא `0xF04`, אך ערך החזרה של `so_to_do what_to_do` הוא 0 והתוכנית חוזרת לשגרה המקורית לטיפול, ששולחת את הסיגנל `Illegal Instruction`).

⁶ הטיט נכתב כך שמיד לאחר הפקודה הלא חוקית יש ביצוע של קריית המערכת `exit`. אתם משים את `%rd` בשגרת הטיפול, אך ערך הייצאה של הטיט ישתנה בהתאם.

פקודות שימושיות

insmod ili.ko

(טען את המודול ili.ko ל kernell ו מפעיל את הפונקציה so init_ko שבמודול)

rmmod ili.ko

(מפעיל את הפונקציה so exit_ko שבמודול ili.ko מה kernell)

תקלות נפוצות (מתעדכן)

במקרה של תקלת "אין מקום בדיסק" שמתאפשרה בזמן הרצה ./compile – עליכם להוריד מחדש את הקובץ filesystem.img ולהחליף את העותק הישן באחד החדש ואז להריץ את ./compile. שוב. באופן כללי, במהלך העבודה יתכן שתצטרכו למחוק את img filesystem אצלכם ולהחליף אותו בעותק שבאתר, לאחר שניסיתם לבצע שינויים וממשהו השתבש בעותק של הקובץ שעליו אתם עובדים. מומלץ לשמר עותק ללא שינויים בכך, כדי לבצע את ההחלפה זו במהירות (ולא להוריד כל פעם מהאתר ☺).

הערות כלליות

נשים לב שבתרגיל זה שינוינו את הקוד של הגרעין! ובכיוון שהקו דורך לאו לנו דרך ללבג אותו ולהבין אם הוא אכן עובד כפי שציפינו שייעבוד. **דיבוג קוד kernell הוא קשה**. כאן לא תוכלו להיעזר ב-gdb. תצטרכו, ברוח הפעמים לנסות "לבדוק בצד" את מה שכabbתם, על דוגמאות עצוצע, ולראות שהקוד עשה מה שהוא אמר או לעשות. רוב המחשבה והדיבוג נעשים "בראש" וכן הקוד שאתם מבוטב הוא יחסית פשוט וקצר.

ובכל זאת, על מנת לעזור לכם להבין מה קורה בקורס – תוכלו להשתמש בפונקציה print() המוגדרת בקובץ c.main_ili, ולראות את הודעות הקרנל ע"י בתיבה של הפקודה dmesg בטרמינל של qemu. ומה print() ולא ?printf()

הפונקציה printf() היא פונקציה של הספרייה libc, ובאשר אנו כותבים קוד גרעין או עושים לו מודיפיקציה אין לנו את האפשרות לגשת למספריה זו (ואנחנו גם לא צריכים כי יש לנו רמת הרשות 0, ו- libc היא ספרייה שבור משתמשים), לכן אנו צריכים לגשת לפונקציות שנמצאות בגרעין – אך עדנו לכם וכתבנו עבורכם מעטפת נחמדה ☺

דבר נוסף, אני קראו את הערות בקבי הקוד שעלייכם למלא. זה יכול רק לעוזר.

תיעוד של qemu ניתן למצוא בא: <https://qemu.weilnetz.de/doc/2.11/qemu-doc.html>

חלק ג' - הוראות הגשה לתרגיל הבית

אם הגעתם לבאן, זו בהחלט סיבה לחגגה. אך בבקשתה, לא לנוכח על זרי הדפנה ולתת את הפוש האחרון אל עבר ההגשה – חבלי מואוד שתצטרכו להתעסך בעוד מספר שבועות מעכשי בערעוורים, רק על הגשת הקבצים לא כדי שנתבהקשותם. אז קראו בעיון ושימו לב שאתם מגישים את כל מה שצריך ורק את מה שצריך.

עליכם להגיש את הקבצים בתוך קובץ אחד (שם הקובץ לא משנה).

בתוך קובץ zip זה יהיו 2 תיקיות:

- part1
- part2

ובתוך כל תיקייה יהיו הקבצים הבאים (מחולק לפי תיקיות):

- part1:
 - students_code.S
- part2:
 - ili_handler.asm
 - ili_utils.c

בהצלחה!!!