ארגון ותכנות המחשב

תרגיל 4 - חלק יבש

המתרגל האחראי על התרגיל: תומר כץ.

שאלות על התרגיל – ב- **Piazza** בלבד.

הוראות הגשה:

- ההגשה בזוגות.
- על כל יום איחור או חלק ממנו, שאינו באישור מראש, יורדו 5 נקודות.
  - ניתן לאחר ב-3 ימים לכל היותר.
- הגשות באיחור יתבצעו דרך אתר הקורס.
- לכל שאלה יש לרשום את התשובה במקום המיועד לכך.
- יש לענות על גבי טופס התרגיל ולהגיש אותו באתר הקורס כקובץ **PDF**.
  - ניתן להקליד את התשובות במסמך ה-WORD, או לכתוב אותן על גבי גרסת ה-PDF בעזרת הטאבלט החביב עליכן. העיקר להגיש בסופו של דבר קובץ PDF לבדיקה, בכתב ברור וקריא.
- תיקונים בקובץ ממורקרים.

שאלה 1 – קידוד פקודות:

גרמניה חוותה הפסד כואב מנבחרת יפן בשלב הבתים במונדיאל. בגלל שהכבוד העצמי שלהם נפגע הם פנו למומחה המחשבים הכי טוב בגרמניה שיעזור להם להרוס את כל המחשבים ביפן.

1. בגלל אותה מתקפה, כל האסמבלרים ביפן הפסיקו לתרגם פקודות לשפה מכונה. עזרו ליפנים לתרגם את הפקודות הבאות בצורה תקינה מאסמלי (AT&T syntax) לשפת מכונה.
הערה: יש למלא את הערכים בhexadecimal.

```
<start>:
400000:  48 31 C9          xor %rcx, %rcx

400003: 66 B9 7B 00        mov  $123 , %cx

400007: 67 83 E9 05        sub $5, %ecx

40000A: 4A 8D 05 0C 00 00 00   lea 12(%rip) , %r8

400011:  FF 25 34 12 00 00     jmp *0x1234(%rip)
```

2. מה יהיה ערכו של רגיסטר r8 בעת ההגעת הקוד לכתובת 0x400011 ? **0x40001D**
3. היפנים שחשבו שהפצצות ב45 הם הדבר הכי נורא שקרה להם, אבל אף אחד לא הכין אותם לכך שהמעבדים שלהם יפסיקו לעבוד. עזרו ליפנים לתרגם את הרצף הבינארי הבא מפקודות מכונה לפקודות אסמבלי.

```
8d 05 02 00 00 00 c1 eb 05
```

הרצף הנ"ל נתון בהקסא, משמאל לימין (הבית הראשון ברצף הוא 0x8d). את רצף הפקודות שמקודד עליכם לכתוב בשורות הבאות:

lea 2(%rip), %eax

shr $5, %ebx

הערות: כל פקודה חייבת להופיע בשורה נפרדת. ניתן להשאיר שורות ריקות.

שאלה 2 – קבצי ELF וקישור סטטי:

לרגל תקופת המונדיאל חברכם גיא החילט לכתוב תוכנית באסמבלי המתפרשת על שני קבצים.
להלן תוכן הקבצים:

worldCup1.asm

```
.global _start
.extern worldCup, overtime

.section .text
_start:
    movq $1, %rax
    movq $1, %rdi
    movq $worldCup, %rsi
    movq $21 , %rdx
    syscall
    movl $end, overtime(%rip)
    jmpq *overtime
    movq $60, %rax
    syscall
end:
    imulq %rax, %rdx
    movq %rdx , %rdi
    movq $60, %rax
    syscall
```

worldCup2.asm

```
.global worldCup, overtime

.extern _start

.section .data
worldCup: .ascii "How is going to win?\n"
overtime: .quad _start
```

גיא התלהב מהקוד שכתב והריץ בטרמינל את הפקודות הבאות:

```
as worldCup1.asm -o worldCup1.o
as worldCup2.asm -o worldCup2.o
ld worldCup1.o worldCup2.o -o worldCup.out
./worldCup.out
```

גיא טס לצפות במונדיאל בקטאר ושם הוא דיבר עם אוהדים מכל העולם. התברר לגיא שאף אחד מהם לא יודע איך טבלאות הסמלים של שני הקבצים יראו.

א) עזרו לאוהדי העולם ומלאו את טבלאות הסמלים של worldCup1.o ושל worldCup2.o.
הערות:
1. ניתן להשאיר שורות ריקות
2. בעמודה Nxt עליכם לכתוב את שם ה section או UND (ולא מספר).

worldCup1.o symbol table:

(section) Nxt	Bind(נראות)	name
.text	global	_start
.text	local	end
UND	global	worldCup
UND	global	overtime

המשך השאלה בעמוד הבא

worldCup2.o symbol table:

(section) Nxt	Bind(נראות)	name
UND	global	_start
.data	global	worldCup
.data	global	overtime

גיא החליט להתחפש כדי שאף אחד לא יזהה אותו ולכן גם חבריו של גיא לא מזהים אותו. בשביל לדעת באמת מי זה גיא אותם חברים הראו לו את טבלת הsection header של הקובץ worldCup1.o שנוצרה ע"י הרצת הפקודה : readelf -S worldCup1.0. ואת התוכן של הקובץ worldCup1.o ע"י הפקודה hexdump. להלן התוצאות:

readelf -S worldCup1.o:

```
There are 8 section headers, starting at offset 0x200:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000000000000  00000040
       0000000000000048  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA             0000000000000000  00000168
       0000000000000060  0000000000000018   I       5     1     8
  [ 3] .data             PROGBITS         0000000000000000  00000088
       0000000000000000  0000000000000000  WA       0     0     1
  [ 4] .bss              NOBITS           0000000000000000  00000088
       0000000000000000  0000000000000000  WA       0     0     1
  [ 5] .symtab           SYMTAB           0000000000000000  00000088
       00000000000000c0  0000000000000018           6     5     8
  [ 6] .strtab           STRTAB           0000000000000000  00000148
       000000000000001e  0000000000000000           0     0     1
  [ 7] .shstrtab         STRTAB           0000000000000000  000001c8
       0000000000000031  0000000000000000           0     0     1
```

ב) אותם חברים רצו שגיא יסמן ב Hexdump את מקטע הtext בשביל להוכיח שהוא הגיא האמיתי. עזרו לגיא וסמנו את מקטע הtext בhexdump הבא:

hexdump worldCup1.o:

```
0000000 457f 464c 0102 0001 0000 0000 0000 0000
0000010 0001 003e 0001 0000 0000 0000 0000 0000
0000020 0000 0000 0000 0000 0200 0000 0000 0000
0000030 0000 0000 0040 0000 0000 0040 0008 0007
0000040 c748 01c0 0000 4800 c7c7 0001 0000 c748
0000050 00c6 0000 4800 c2c7 0015 0000 050f 05c7
0000060 0000 0000 0000 0000 24ff 0025 0000 4800
0000070 c0c7 003c 0000 050f 0f48 d0af 8948 48d7
0000080 c0c7 003c 0000 050f 0000 0000 0000 0000
0000090 0000 0000 0000 0000 0000 0000 0000 0000
00000a0 0000 0000 0003 0001 0000 0000 0000 0000
00000b0 0000 0000 0000 0000 0000 0000 0003 0003
00000c0 0000 0000 0000 0000 0000 0000 0000 0000
00000d0 0000 0000 0003 0004 0000 0000 0000 0000
00000e0 0000 0000 0000 0000 0001 0000 0000 0001
```

offset: 0x40

size: 0x48

end: 0x40 + 0x48 = 0x88

המשך השאלה בעמוד הבא

לצורך הסעיף הבא נתון פלט ה objdump של worldCup1.o:

```
0000000000000000 <_start>:
   0:   48 c7 c0 01 00 00 00    mov    $0x1,%rax
   7:   48 c7 c7 01 00 00 00    mov    $0x1,%rdi
   e:   48 c7 c6 00 00 00 00    mov    $0x0,%rsi
  15:   48 c7 c2 15 00 00 00    mov    $0x15,%rdx
  1c:   0f 05                   syscall
  1e:   c7 05 00 00 00 00 00    movl   $0x0,0x0(%rip)        # 28 <_start+0x28>
  25:   00 00 00
  28:   ff 24 25 00 00 00 00    jmpq   *0x0
  2f:   48 c7 c0 3c 00 00 00    mov    $0x3c,%rax
  36:   0f 05                   syscall

0000000000000038 <end>:
  38:   48 0f af d0             imul   %rax,%rdx
  3c:   48 89 d7                mov    %rdx,%rdi
  3f:   48 c7 c0 3c 00 00 00    mov    $0x3c,%rax
  46:   0f 05                   syscall
```

ג) מלאו את הטבלה הבאה של relocation של ה text section:

offset	type	Symbol name	adden
0x11	קבוע	worldCup	0
0x20	יחסי	overtime	-8
0x24	קבוע	.text	38
0x2b	קבוע	overtime	0

<u>הערה</u>: ב" Type"ניתן להשלים רק "יחסי" או "קבוע" ואין צורך להשתמש בשמות המלאים.

ד) האם בניית התוכנית תצליח? (יווצר קובץ הרצה תקין?) **כן/לא**

ה) בהמשך לסעיף הקודם, אם עניתם לא הסבירו מדוע. אם כן רשמו מה יהיה פלט התוכנית ומה ערך היציאה שלה.

<u>Output:</u>
How is going to win?\n

<u>Exit Code</u>:
185

שאלה 3 – קישור דינמי:

1) לפניכם קוד של ספריה דינאמית שקומפלה:

```
extern int value;

void change_value(int a, int b){
        value++;
        value = a + 2*value * b;
        value = value -2;
}
```

כמה תיקונים יצטרך לעשות הקשר הדינאמי עבור הסמל value? הסבירו את איפה יתבצעו התיקונים.

נשים לב ש-value מופיע במקטע הקוד 3 פעמים, כלומר נצטרך בידיוק 3 תיקונים שיתבצעו במקטע ה-data. הספריה תדע לגשת לערך החיצוני הזה כפי שלמדנו בעזרת טבלאת ה-GOT.

2) נתון לכם PLT של תוכנה מסוימת.

```
Disassembly of section .plt:

0000000000001020 <.plt>:
    1020:       ff 35 e2 2f 00 00       pushq  0x2fe2(%rip)        # 4008 <_GLOBAL_OFFSET_TABLE_+0x8>
    1026:       ff 25 e4 2f 00 00       jmpq   *0x2fe4(%rip)        # 4010 <_GLOBAL_OFFSET_TABLE_+0x10>
    102c:       0f 1f 40 00             nopl   0x0(%rax)

0000000000001030 <printf@plt>:
    1030:       ff 25 e2 2f 00 00       jmpq   *0x2fe2(%rip)        # 4018 <printf@GLIBC_2.2.5>
    1036:       68 00 00 00 00          pushq  $0x0
    103b:       e9 e0 ff ff ff          jmpq   1020 <.plt>
```

נתמקד בפקודה בכתובת 0x1030.

i) מה סוג הקפיצה שבו משתמשים? **אבסולוטית.**

ii) מהו סוג האופרנד (אם מדובר בכתובת, ציינו שיטת מיעון)? **RIP=operand, קפיצה לכתובת שנמצאת בזכרון ועוד DISP.**

iii) האם ידוע לאיזה כתובת נקפוץ בעת ביצוע הפקודה? אם כן מהי הכתובת ואם לא מדוע לא ניתן לדעת ומה כן ניתן לדעת על אותה כתובת.

לא ניתן לדעת בוודאות לאיזה כתובת נקפוץ בעת ביצוע הפקודה, נפריד למקרים:

- **שימוש ב-Lazy Binding:**
  - **נעשתה קריאה ל-printf קודם לכן בתכנית, במקרה זה נדע בוודאות את הכתובת.**
  - **במקרה שלא נעשתה קריאה ל-printf למעשה לא נדע לאיזה כתובת בוודאות נקפוץ תחילה ל-GOT כדי להשיג את הכתובת ולבצע קישור דינמי.**
- **אחרת, נדע.**

3) הסבירו מה תכיל הכתובת 0x4018 בתחילת ריצת התוכנית. התייחסו למקרה שבו התוכנית קומפלה עם lazy binding ולמקרה שבו היא לא.

- **עבור Lazy Binding – היא תכיל את הכתובת של השורה הבאה אם זו הפעם הראשונה שניגשנו ל-printf ובעצם תבצע מה שהגדרנו כטרמפולינה אחרת את הכתובת של הזכרון ש-printf נטענה אליה.**
- **אחרת – את הכתובת שprintf נטענה אליה.**

4) הסבירו מתי נרצה לקמפל עם lazy binding ומתי לא נרצה.

- **קימפול עם Lazy Binding – כאשר מתבצעת טעינה של ספריות רבות המכילות מספר רב של פונקציות, עבור מקרה זה אילו לא היינו מקמפלים עם lazy binding עלולה להיווצר תקורה גבוהה ובפרט עיכוב תחילת ריצת התוכנית בעקבות טעינה מסוג זה, כמו כן ייתכן והתוכנית בכלל לא משתמש בכל הפונקציות הללו שנטענו מראש.**
- **קימפול ללא Lazy Binding – בידיוק המקרה ההפוך יש לנו מעט סיפריות או ספריות עם מעט פונקציות שאנחנו משתמשים בהם לעיתים תכופות יותר עבור מקרה זה אין צורך להשתמש ב-lazy binding.**