ארגון ותכנות המחשב

תרגיל 1 - חלק יבש

המתרגל האחראי על התרגיל: תום הרמן.

שאלותיכם במייל בעניינים מנהלתיים בלבד, יופנו רק אליו.

כתבו בתיבת **subject**: יבש 1 את"ם.

שאלות בעל-פה ייענו על ידי כל מתרגל.

תאריך הגשה: 02.05.21

הוראות הגשה:

- לכל שאלה יש לרשום את התשובה במקום המיועד לכך.
- יש לענות על גבי טופס התרגיל ולהגיש אותו באתר הקורס כקובץ PDF.
- על כל יום איחור או חלק ממנו, שאינו בתיאום עם המתרגל האחראי על התרגיל, יורדו 5 נקודות.
- הגשות באיחור יש לשלוח למייל של אחראי התרגיל בצירוף פרטים מלאים של המגישים (שם+ת.ז).
- שאלות הנוגעות לתרגיל יש לשאול דרך הפיאצה בלבד.
- ההגשה בזוגות.

שאלה 1 – מעקב אחר פקודות:

לפניכם קטע קוד. נתון כי הכתובת של תחילת מקטע הנתונים היא 0x492054. עליכם לעקוב אחר הפקודות ולרשום תוכן של נתון מבוקש במקומות שמבקשים מכם (בערכי הקסדצימלי).
במידה ומתבצעת פקודה לא חוקית בשלב מסוים, יש לרשום X במקום שצריך להשלים, ולהתייחס כאילו הפקודה מעולם לא נרשמה:

```
.global _start

.section .data
arr: .short 0, 1, 0x42, 0x67, 0x1234
b: .int 0x19283746
c: .quad 0x0404202102052021

.section .bss
.lcomm garbage, 8
.lcomm g_byte, 1

.section .text
_start:
xor %rcx, %rcx
movq $0x1234, %rax
movb $0, %al
```

ערך rax: ___0x1200_____.

```
xor %rax, %rax
xor %rsi, %rsi
lea b+1, %rbx
```

ערך rbx: _0x49205F____.

```
movb 3(%rbx), %al
```

ערך al: _____0x21_____.

```
mov %bh, %al
xor %al, %sil
shr $3 %rsi
movw -11(%rbx, %rsi, 2), %dx
```

ערך dx: __0x1234___.

```
shr $2, %rsi
movb $0x66, g_byte
addb (%rbx, %rsi, 4), g_byte
```

ערך הבית שב- g_byte: _____X_____.

```
xor %rax, %rax
dec %ax
```

ערך rax: _____0xFFFF_____.

```
mov $garbage, %rcx
lea c, %rbx
movw arr+4, %ax
ror $4, %ax
```

ערך rax: _____0x2004_____.

```
xor %ax, %ax
incb %ax
```

ערך rax: _____X_____.

```
movq (%rbx), %rbx
mov $0x40, %si
dec %rcx
movl %ebx, 2(%rcx)
```

התוכן שבבית בכתובת 3+garbage _____0x05_____.

```
mov $78, b
```

ערך הבית b (הבית שb מהווה פניה אליו): _____X_____.

```
movq $arr, b
```

ערך הבית c (הבית שc מהווה פניה אליו): _____0x0____.

```
movswq (b), %rdx
```

ערך rdx: ____0x2054__.

```
mov $0x9937, %ax
cwd
```

ערך rdx: ____0xFFFF_____.

```
movw $-0x33, garbage
idivw garbage
```

ערך eax: ____0x203____. ערך edx: _____0xFFD0_____.

```
movq $0x11, (b)
imul $0x8, b, %rdx
```

ערך rdx: _____0x88__.

```
xor %rax, %rax
mov $0xff, %ax
mov $3, %bl
mov $33, %rdx
imulb %bl
```

ערך ax: ___0xFFFD_____. ערך dx: ____0x21_____.

```
mov $380, %ax
mov $760, %bx
mul %bx
```

ערך ax: ____0x6820_____. ערך dx: ____0x4_____.

```
shl $19, %edx
lea 24(%edx, %eax, 8), %eax
```

ערך ax: _____0x4118_____. ערך dx: _____0____.

שאלה 2 – תרגום מC לאסמבלי:

לפניכם קטעי קוד בשפת c עליכם לתרגם כל קטע בשפת c לאסמבלי על ידי השלמת המקומות שמסומנים בקו. במידה וכל השורה מסומנת בקו עליכם להשלים את השורה איך שאתם רוצים אך עליכם להשתמש בפוקדה אחת בלבד! נתון ש-a ו-b הוגדרו כ int.
מומלץ לעבור על "אופטימיזציה אריתמטית" מתרגול 2, ולראות דוגמאות לפני המעבר על השאלה.
<u>הערה 1:</u> בשורה הרביעית הרווח אחרי lea(אינו טעות. אין להשלים שם ערך. זהו רמז (וחלק מהסינטקס).
<u>הערה 2:</u> נזכיר כי '|' בשפת C היא הפעולה or.

על מנת למנוע בלבול מסופקת לכם דוגמה בשורה הראשונה:

קוד בשפת c	קוד אסמבלי
`a += b;`	`movl b , %eax` `addl %eax , a`
`a = a / 16;`	`sarl $4 , a`

`a = a*3;`	`movl a, %eax`
	`lea (%eax, %eax , 2), %eax`
	`mov %eax, a`

`a = a*8;`	`movl a, %eax`
	`lea ( , %eax, 8), %eax`
	`mov %eax, a`

`a = 8b - 13 + a;`	`movl a, %eax`
	`movl b, %ebx`
	`lea -13(%eax, %ebx, 8), %eax`
	`mov %eax, a`

`a |= (1<<16);`	`Or 0x10000, a`
`a = 4*a;`	`imul 4, a, %eax` `mov %eax, a`

`a = a*a*a*a*a*a*a*a;`	`movl a, %eax`
	`imul %eax`
	`imul %eax`
	`imul %eax`
	`mov %eax, a`

`if (a >= 0)`
`  b = 0;`
`else`
`  b = -1;`
	`movl a, %eax`
	`cdq`
	`movl %edx, b`

שאלה 3 – לולאות ומספרים:

בשאלה זו נשתמש במספרים חסרי סימן (unsigned).
בנוסף, נניח כי הוגדר משתנה n שגודלו 4 בייטים ושכל ה-General Purpose Registers מכילים 0 בתחילת התוכנית (הכוונה היא לרגיסטרים שמשתמשים בהם לחישובים ולא לריגסטרים מיוחדים כמו rip או rflags)
רוס המרצה בקורס כתב קטע קוד. לפניכם הקוד שרוס כתב:

```
mov $1, %eax
mov %eax, %ebx
mov n, %ecx
loop:
    dec %ecx
    test %ecx, %ecx
    je end

    mov %eax, %edx
    mov %ebx, %eax
    add %edx, %ebx
    jmp loop

end:
    ret
```

1. נתון שבתחילת התוכנית n=10 (בעשרוני).
מה יהיה ערך רגיסטר eax בסיום קטע התוכנית? כתבו את התשובה גם בבסיס דצימלי וגם בהקסדצימלי (וכתבו את כל הבתים שלו ב-hexa)?
פתרון:
דצימלי – 55
הקסדצימלי – 0x37

2. הסבירו במשפט אחד מה עושה התוכנית.
פתרון:
מחשבת את האיבר ה-n בסדרת פיבונצ'י

3. מוניקה הסטודנטית החרוצה שמה לב שעבור n=50 מוחזרת תשובה לא נכונה. מה הסיבה לכך?
מהו המספר הגדול ביותר שניתן לשים ב-n בתחילת הריצה, ועדיין לקבל תשובה נכונה?
פתרון:
הסיבה לכך היא שעבור n=50 האיבר ה-n שמתקבל בסדרת פיבונצ'י גדול מדי להציגו ב32 ביט ואינו נכנס למשתנה בגודל 4 בתים (eax) ולכן במקרה הזה המספר שיתקבל שגוי.
המספר הגדול ביותר שניתן לשים ב-n ועדיין לקבל תשובה נכונה הוא 47.

4. מוניקה, שרוצה להרשים את רוס, שינתה את הקוד:

```
mov $1, %rax
mov %rax, %rbx
mov n, %ecx
loop:
    dec %rcx
    test %ecx, %ecx
    je end

    mov %eax, %edx
    mov %rbx, %rax
    add %rdx, %rbx
    jmp loop

end:
    ret
```

לצערה, הקוד עדיין לא תקין. מדוע?

פתרון:

בשורה: mov %eax, %edx, אנו מעבירים את הערך של רגיסטר eax ל-32 הביטים התחתונים של rdx, אך בהמשך התוכנית, אנו משתמשים בכל רגיסטר rdx ואיננו יכולים לדעת מה נמצא ב4 הבתים העליונים שלו – ייתכן שיש שם ערך זבל, מה שיפגע בחישוב.

5. לאחר שמוניקה תיקנה את התכנית שלה, רוס ראה שהיא עדיין לא מסופקת, והציע לה דרך לתמוך במספרים גדולים עוד יותר.
רוס מסביר למוניקה (שכבר יודעת!) על הפקודה adc. הסבירו למוניקה איך ניתן לחבר מספרים בגודל 16 בתים בעזרת פקודה זו.
בפרט, חברו את המספר rbx:rax (8 הבייטים התחתונים שמורים בrax, ו8 העליונים שמורים בrbx) אל המספר rdx:rcx.

פתרון:

נרצה להשתמש ב-adc. נערוך את הקוד (משום שנצטרך להשתמש ברגיסטר אחר עבור האינדקס) כך שהקוד החדש הינו:

```
# initialize registers
mov $1, %rax
xorq %rbx, %rbx
mov %rax, %rcx
xorq %rdx, %rdx
mov n, %r10 # new counter

# loop and calculate
loop:
	dec %r10
	je end
	mov %rax, %r8
	mov %rbx, %r9
	mov %rcx, %rax
	mov %rdx, %rbx
	add %r8, %rcx
	adc %r9, %rdx
	jmp loop
end:
	ret
```

6. מוניקה שמה לב שאחת מהפקודות בקוד של רוס, ואף בקוד שלה, מיותרת. על איזו פקודה מדובר, ולמה?

פתרון:

הפקודה המיותרת היא: test %ecx, %ecx.

פקודת ה-test הנ"ל מעדכנת את ZF להיות 1 אם 0 == %ecx, אבל ZF גם מתעדכן ל-1 אם התוצאה האחרונה היא 0. במקרה שלנו, התוצאה האחרונה תהיה 0 כאשר נכנס ללולאה בפעם ה-n ונעשה dec %rcx. כלומר רצינו לקפוץ ל-end ולשם כך בקוד השגוי זה נבדק פעמיים במקום פעם אחת.