

# **בית הספר התיכון העירוני המקיף א' באר-שבע**



**מגמת הנדסת תוכנה**

## **פרויקט באסמבלי**

**על ידי: אנטולי זוגרביאן**

**ת.ז. 341197010**

**כתיבת פרויקט בוצעה בהנחייתו של מר צ'רבצקי אנטון**

**יוני 2025**

## תוכן עניינים

3	נושא העבודה
4	פריסת סיביות
5	משתנים והודעות שמורות
6+7	תוכנית המרכזית(main)
8-12	תת תכנית הראשונה
13-14	תת תכנית השנייה
15-19	תת תכנית השלישית
20-21	תת תכנית הרביעית
22	רפלקציה לפרויקט

## אנטולי זוגריאן – י'2

### פרויקט באסמבלי: ניתוח מכירות בחנות משחקים

הפרויקט שלי עוסק בניהול מכירות עבור חנות משחקים פיזית שנפתחה לאחרונה ומוכרת 8 משחקי מחשב. לבעל החנות היה חשוב לדעת מה הלקוחות הכי אוהבים כדי למקסם רווחים, ולכן הוא ביקש ממני לפתח עבורו מערכת שתאסוף נתונים על מכירות – וכך נוכל לנתח אילו משחקים הכי פופולריים.

נכון, כל המשחקים שהוא מוכר נחשבים לאהובים מאוד – אבל מה עדיף להציג ללקוחות קודם כדי להרוויח יותר? זו בדיוק השאלה שניסינו לפתור.

כחלק מהשדרוג לחוויית הקנייה, הצענו אפשרות לרכוש **DLC** – **תוכן נוסף למשחק**. הלקוח יכול לבחור להוסיף DLC לאותו משחק שקנה, ובמקרה כזה יקבל עליו הנחה. כל **DLC** עולה 100 ₪ ובגלל ההנחה זה יעלה לו 80 ₪.

שילבנו את כל הרעיונות האלה בתוכנה שלנו. המערכת מזהה אילו משחקים נמכרו הכי הרבה, מה כמות הנחה שהלקוח קיבל, מה הרוויח הסופי של החנות ואם הלקוח קנה באשראי אז בודק האם התשלום עבר.

בשלב הזה הבנתי שעם 8 סיביות בלבד אפשר לקודד את כל המידע על כל לקוח:

איזה משחק קנה, האם רכש DLC, אמצעי תשלום, אם התשלום עבר, אם הייתה הנחה לפי היסטוריית רכישות, ועוד.

התוצאה: מערכת חכמה, מדויקת וקלה לשימוש – שעוזרת לבעל העסק להבין מה נמכר, מה עובד, וכמה הוא באמת הרוויח מכל חודש.

## מערכת שמנתחת מכירות לפי 8 סיביות מידע לכל לקוח, ומחזירה:

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

### המשחק שהלקוח לקח:

- 000 – Elden Ring 250 ₪
- 001 – Red Dead Redemption 2 230 ₪
- 010 – Minecraft 80 ש"ח
- 100 – Grand Theft Auto V (GTA 5) 56 ₪
- 110 – Cyberpunk 2077 250 ₪
- 111 – EA SPORTS FC™ 25 255 ₪
- 011 – Far Cry 5 240 ₪
- 101 – Ghost of Tsushima 240 ₪

### סה"כ התוכנה הסופית:

1. מהו המשחק הכי נמכר
2. האם התשלום עבר
3. הסכום הסופי שהרוויח החנות
4. כמות ההנחות שניתנו

### האם האדם קנה כבר פה?

- 00-פעם אחד בעבר יקבל הנחה של 2%
- 01-בין 2 ל4 הנחה של 5%
- 10-בין 5 ל8 הנחה של 10%
- 11-מעל 8 הנחה של 20%

### האם שילם במזומן או אשראי?

- 0-אשראי
- 1-מזומן

### עבר התשלום או שלא?

- 0 – לא
- 1 – כן

### האם הלקוח רוצה לקנות תוספת למשחק או שלא (DLC):

- 0 – לא
- 1 – כן

## משתנים והודעות שמורות

```
; Game Store Sales Analysis System
; Author: Anatoly Zohrabyan
-8 ; bit encoding: [Game(3) | Purchase_History(2) |
Payment_Method(1) | Payment_Success(1) | DLC(1)]
```

```
.model small
.stack 100h
.data
; Memory variables for game counters
```

יצירת המשתנים אשר ישמרו בתוכם את  
כמות הפעמים בו הופיע כל משחק לפי  
המשתנה שלו.

```
elden_ring_count db 0
rdr2_count db 0
minecraft_count db 0
farcry_count db 0
gta5_count db 0
ghost_count db 0
cyberpunk_count db 0
fc25_count db 0
```

יצירת הודעות לתוצאה של התת תוכנה  
הראשונה. הפלט יופיע ב standard output.

```
; Mesagees to out put
best_seller_msg db 10, 13, "Best seller game    "$ :
elden_ring_msg db 10, 13, "Elden Ring    "$
rdr2_msg db 10, 13, "RDR2$    "
minecraft_msg db 10, 13, "Minecraft    "$
farcry_msg db 10, 13, "Far Cry 5$    "
gta5_msg db 10, 13, "GTA5$    "
ghost_msg db 10, 13, "Ghost of Tsushima    "$
cyberpunk_msg db 10, 13, "Cyberpunk 2077$    "
fc25_msg db 10, 13, "FC 25$    "
```

המשתנה אשר ישמור את הכמות הרווח  
הסופי של החנות.

```
; Variables for profit calculation
total_profit dw 0
```

```
.code
```

```
mov ax, @data
mov ds, ax
```

## תוכנית המרכזית (main)

קורא באופן רציף קלט משתמש מפורט 1  
משווה כל קלט לאפשרויות תפריט תקפות (1-5)  
קורא לתת-השגרה המתאימה בהתבסס על בחירת המשתמש  
חוזר לתפריט הראשי לאחר השלמת כל פעולה  
מטפל בקלט לא חוקי על ידי חזרה בלולאה כדי לבקש קלט שוב

כל הוראה מוסברת בפירוט, ומציגה בדיוק אילו אוגרים משמשים, אילו השוואות נעשות ולאן זרימת התוכנית עוברת בהתבסס על בחירת המשתמש.

קרא בייט מיציאת קלט 1 לתוך אוגר ה-  
AL. בייט זה מייצג את בחירת התפריט של  
המשתמש כאשר:

1 = חיפוש המשחק הנמכר ביותר, 2 =  
בדיקת הצלחה של התשלום, 3 = חישוב  
הרווח הכולל, 4 = ספירת הנחה, 5 = יציאה  
מהתוכנית.

השווה את הערך ב-AL עם 1. אם הם שווים  
(דגל אפס מוגדר), עבור לתווית  
call\_bestseller כדי לבצע את ניתוח  
המשחק הנמכר ביותר.

השווה את הערך ב-AL עם 2. אם הם שווים  
(דגל אפס מוגדר), עבור לתווית  
call\_payment\_check כדי לבצע את שגרת  
אימות התשלום.

השווה את הערך ב-AL עם 3. אם הם שווים  
(דגל אפס מוגדר), עבור לתווית  
call\_profit\_calc כדי לבצע את שגרת  
חישוב הרווח.

השווה את הערך ב-AL עם 4. אם הם שווים  
(דגל אפס מוגדר), דלג לתווית  
call\_discount\_count כדי לבצע את שגרת  
ספירת ההנחות.

השווה את הערך ב-AL עם 5. אם הם שווים  
(דגל אפס מוגדר), דלג לתווית  
end\_program כדי לסיים את התוכנית.

אם אף אחת מההשוואות לעיל לא תואמת  
(המשתמש הזין בחירה לא חוקית), חזור ל-  
main\_loop כדי לבקש קלט שוב. פעולה זו  
יוצרת לולאה אינסופית עד שתבצע בחירה  
תקפה.

סעיף זה קורא לתת-תוכנית bestseller כדי  
למצוא ולהציג את המשחק הנמכר ביותר,  
לאחר מכן קופץ חזרה ל-main\_loop כדי  
להציג שוב את התפריט ולהמתין לבחירת  
המשתמש הבאה.

סעיף זה קורא לתת-תוכנית  
payment\_check כדי לאמת את הצלחת  
התשלום עבור לקוחות, לאחר מכן קופץ  
חזרה ל-main\_loop כדי להציג שוב את  
התפריט ולהמתין לבחירת המשתמש הבאה.

main\_loop:

in al, 1 ; Get user choice(1 - best-selling game, 2 -

Check payment success, 3 - Calculate total profit, 4 - Count

discount, 5 - Exit)

cmp al, 1

jz call\_bestseller

cmp al, 2

jz call\_payment\_check

cmp al, 3

jz call\_profit\_calc

cmp al, 4

jz call\_discount\_count

cmp al, 5

jz end\_program

jmp main\_loop

call\_bestseller:

call bestseller

jmp main\_loop

call\_payment\_check:

סעיף זה קורא לתת-שגרה  
PROFIT\_CALC כדי לחשב את הרווח  
הכולל מכל המכירות, לאחר מכן קופץ  
חזרה ל-MAIN\_LOOP כדי להציג שוב את  
התפריט ולהמתין לבחירת המשתמש הבאה.

מקטע זה קורא לתת-שגרת ה-  
discount\_count כדי לספור כמה לקוחות  
קיבלו הנחות, לאחר מכן קופץ חזרה ל-  
main\_loop כדי להציג שוב את התפריט  
ולהמתין לבחירת המשתמש הבאה.

jmp main\_loop

```
call payment_check  
jmp main_loop
```

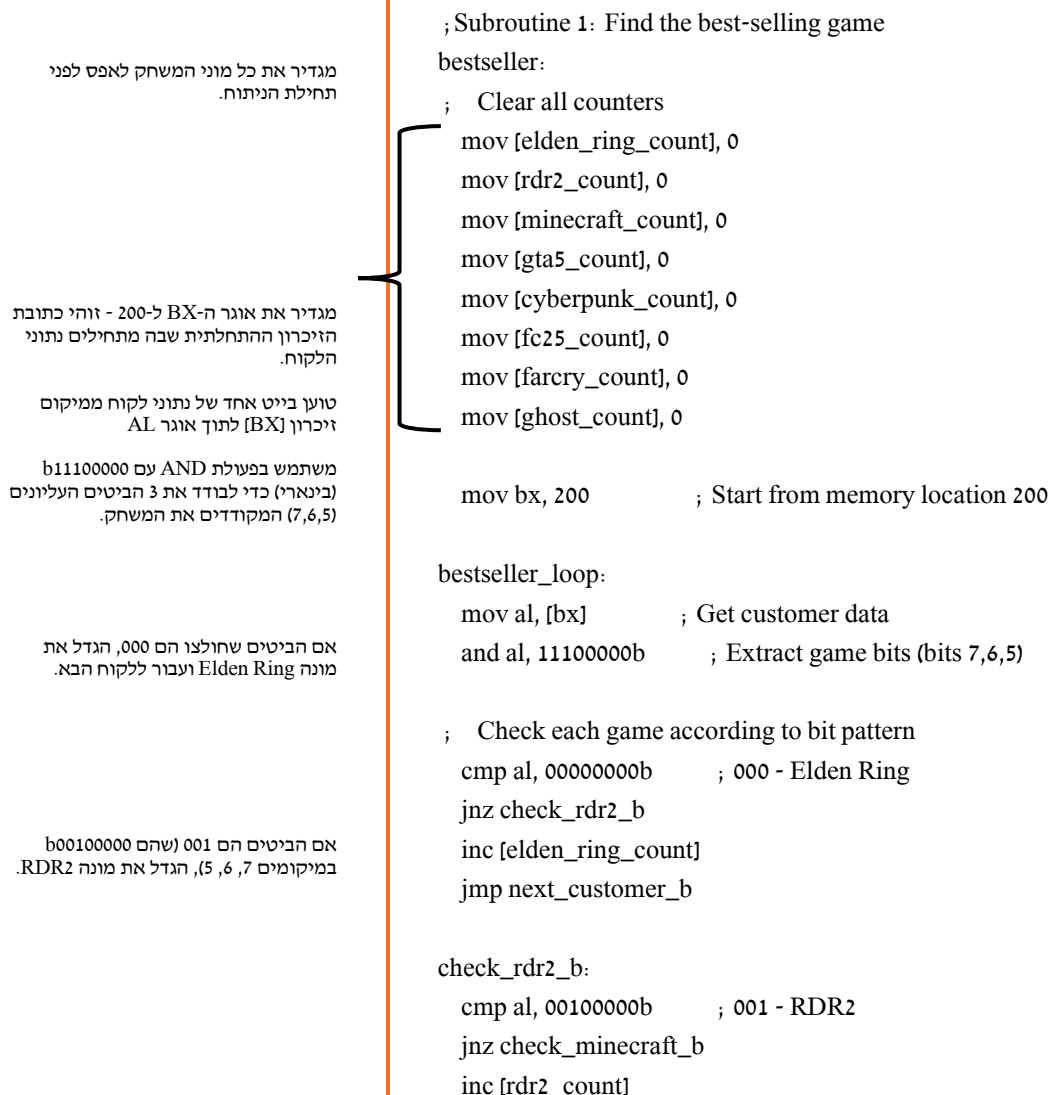
```
call_profit_calc:  
call profit_calc  
jmp main_loop
```

```
call_discount_count:  
call discount_count
```

## תת תכנית הראשונה

קורא נתוני לקוחות ממיקומי זיכרון 200-400 (200 לקוחות בסך הכל)  
מחלץ מידע על המשחק משלושת הביטים העליונים של כל בייט נתונים של לקוח  
סופר מכירות עבור כל אחד מ-8 משחקים שונים  
מוצא את רבי המכר על ידי השוואת כל הספירות  
מציג את התוצאה הן כהודעה והן כקוד מספרי

תת-תכנית הרכבה זו מנתחת נתוני לקוחות כדי למצוא איזה משחק מכר הכי הרבה עותקים. נתוני כל לקוח מאוחסנים כבייט שבו ביטים 7, 6 ו-5 מקודדים איזה משחק הוא קנה.





אם הביטים הם 010 (שזה b01000000 במיקומים 5, 6, 7), הגדל את מונה ה-Minecraft ועבור ללקוח הבא.

jmp next\_customer\_b

אם הביטים הם 011 (שזה b01100000 במיקומים 5, 6, 7), הגדל את מונה Far Cry 5 ועבור ללקוח הבא.

check\_minecraft\_b:

```
cmp al, 01000000b ; 010 - Minecraft
jnz check_farcry_b
inc [minecraft_count]
jmp next_customer_b
```

אם הביטים הם 100 (שזה b10000000 במיקומים 5, 6, 7), הגדל את מונה GTA V ועבור ללקוח הבא.

check\_farcry\_b:

```
cmp al, 01100000b ; 011 - Far Cry 5
jnz check_gta_b
inc [farcry_count]
jmp next_customer_b
```

אם הביטים הם 101 (שזה b10100000 כאשר הם במיקומים 5, 6, 7), הגדל את מונה Ghost of Tsushima ועבור ללקוח הבא.

check\_gta\_b:

```
cmp al, 10000000b ; 100 - GTA V
jnz check_ghost_b
inc [gta5_count]
jmp next_customer_b
```

אם הביטים הם 110 (שזה b11000000 במיקומים 5, 6, 7), הגדל את מונה הסייברפאנק ועבור ללקוח הבא.

check\_ghost\_b:

```
cmp al, 10100000b ; 101 - Ghost of Tsushima
jnz check_cyberpunk_b
inc [ghost_count]
jmp next_customer_b
```

אם הביטים הם 111 (שזה b11100000 במיקומים 5, 6, 7), הגדל את מונה ה-FC 25. זוהי הבדיקה האחרונה, כך שהיא עוברת ישירות לעיבוד הלקוח הבא.

check\_cyberpunk\_b:

```
cmp al, 11000000b ; 110 - Cyberpunk
jnz check_fc25_b
inc [cyberpunk_count]
jmp next_customer_b
```

הגדלת BX כדי לעבור לנתוני הלקוח הבא  
בדוק אם עיבדנו את כל הלקוחות (מיקומי זיכרון 200-400 = 200 לקוחות)

check\_fc25\_b:

```
cmp al, 11100000b ; 111 - FC 25
jnz next_customer_b
inc [fc25_count]
```

אם  $BX < 400$ , המשך לולאה

next\_customer\_b:

```
inc bx
cmp bx, 400
jl bestseller_loop
```

אתחול עם ELDEN RING (אינדקס 0)  
כרב המכר הנוכחי וטען את הספירה שלו  
לתוך BL.

השווה את ספירת RDR2 למקסימום  
הנוכחי. אם ספירת RDR2 גדולה  
מהמקסימום הנוכחי, הגדר את AL ל-1  
(אינדקס RDR2) ועדכן את BL עם ספירת  
RDR2.

השווה את ספירת Minecraft למקסימום  
הנוכחי. אם ספירת Minecraft גדולה יותר,  
הגדר את AL ל-2 (אינדקס Minecraft)  
ועדכן את BL עם ספירת Minecraft.

השווה את ספירת Far Cry 5 למקסימום  
הנוכחי. אם ספירת Far Cry 5 גדולה יותר,  
הגדר את AL ל-3 (אינדקס Far Cry 5)  
ועדכן את BL עם ספירת Far Cry 5.

השווה את ספירת GTA V למקסימום  
הנוכחי. אם ספירת GTA V גדולה יותר,  
הגדר את AL ל-4 (אינדקס GTA V) ועדכן  
את BL עם ספירת GTA V.

השווה את ספירת רוח הרפאים של צושימה  
למקסימום הנוכחי. אם ספירת רוח  
הרפאים של צושימה גדולה יותר, הגדר את  
AL ל-5 (אינדקס רוח הרפאים של צושימה)  
ועדכן את BL עם ספירת רוח הרפאים של  
צושימה.

השווה את ספירת הסייברפאנק למקסימום  
הנוכחי. אם ספירת הסייברפאנק גדולה  
יותר, הגדר את AL ל-6 (אינדקס  
הסייברפאנק) ועדכן את BL עם ספירת  
הסייברפאנק.

השווה את ספירת FC 25 למקסימום  
הנוכחי. אם ספירת FC 25 גדולה יותר,  
הגדר את AL ל-7 (אינדקס FC 25). זוהי  
ההשוואה הסופית, כך שהיא ממשיכה  
ישירות לפלט.

```
; Find the game with maximum sales
mov al, 0 ; Game index (Elden Ring)
mov bl, [elden_ring_count] ; Max count so far
```

```
cmp [rdr2_count], bl
jle check_minecraft_max
mov al, 1
mov bl, [rdr2_count]
```

```
check_minecraft_max:
cmp [minecraft_count], bl
jle check_farcry_max
mov al, 2
mov bl, [minecraft_count]
```

```
check_farcry_max:
cmp [farcry_count], bl
jle check_gta_max
mov al, 3
mov bl, [farcry_count]
```

```
check_gta_max:
cmp [gta5_count], bl
jle check_ghost_max
mov al, 4
mov bl, [gta5_count]
```

```
check_ghost_max:
cmp [ghost_count], bl
jle check_cyberpunk_max
mov al, 5
mov bl, [ghost_count]
```

```
check_cyberpunk_max:
cmp [cyberpunk_count], bl
jle check_fc25_max
mov al, 6
mov bl, [cyberpunk_count]
```

```
check_fc25_max:
cmp [fc25_count], bl
jle output_bestseller
mov al, 7
```

טען את הפונקציה 9 (מחרוזת תצוגה) לתוך  
אוגר AH, טען את כתובת הודעת המוכר  
הטוב ביותר לתוך אוגר DX, לאחר מכן  
קרא לפסיקת DOS 21h כדי להציג את  
ההודעה.

השווה את אינדקס המשחק ב-AH עם כל  
ערך אפשרי (0-7) ודלג לשגרת תצוגת  
ההודעות המתאימה. כל השוואה בודקת  
אם AH שווה לאינדקס משחק ספציפי  
וקופצת להצגת ההודעה של אותו משחק  
אם היא נכונה.

טען את פונקציה 9 לתוך AH, טען את  
כתובת הודעת Elden Ring לתוך DX, קרא  
לפסיקת DOS כדי להציג את ההודעה, ואז  
קפץ לסוף.

טען את פונקציה 9 לתוך AH, טען את  
כתובת הודעת RDR2 לתוך DX, קרא  
לפסיקת DOS כדי להציג את ההודעה, ואז  
קפץ לסוף.

טען את פונקציה 9 לתוך AH, טען את  
כתובת הודעת Minecraft לתוך DX, קרא  
לפסיקת DOS כדי להציג את ההודעה, ואז  
קפץ לסוף.

טען את פונקציה 9 לתוך AH, טען את  
כתובת הודעת Far Cry 5 לתוך DX, קרא  
לפסיקת DOS כדי להציג את ההודעה, ואז  
קפץ לסוף.

output\_bestseller:

```
mov ah, 9
lea dx, best_seller_msg
int 21h
```

```
cmp al, 0
jz out_elden_ring_msg
cmp al, 1
jz out_rdr2_msg
cmp al, 2
jz out_minecraft_msg
cmp al, 3
jz out_farcry_msg
cmp al, 4
jz out_gta5_msg
cmp al, 5
jz out_ghost_msg
cmp al, 6
jz out_cyberpunk_msg
cmp al, 7
jz out_fc25_msg
```

out\_elden\_ring\_msg:

```
mov ah, 9
lea dx, elden_ring_msg
int 21h
```

jmp best\_seller\_end

out\_rdr2\_msg:

```
mov ah, 9
lea dx, rdr2_msg
int 21h
jmp best_seller_end
```

out\_minecraft\_msg:

```
mov ah, 9
lea dx, minecraft_msg
int 21h
jmp best_seller_end
```

out\_farcry\_msg:

```
mov ah, 9
```

טען את פונקציה 9 לתוך AH, טען את  
כתובת הודעת GTA V לתוך DX, קרא  
לפסיקת DOS כדי להציג את ההודעה, ואז  
קפוץ לסוף.

טען את פונקציה 9 לתוך AH, טען את  
כתובת הודעת Ghost of Tsushima לתוך  
DX, קרא לפסיקת DOS כדי להציג את  
ההודעה, ואז קפוץ לסוף.

טען את פונקציה 9 לתוך AH, טען את  
כתובת הודעת הסייברפאנק לתוך DX, קרא  
לפסיקת DOS כדי להציג את ההודעה, ואז  
קפוץ לסוף.

טען את פונקציה 9 לתוך AH, טען את  
כתובת הודעת FC 25 לתוך DX, קרא  
לפסיקת DOS כדי להציג את ההודעה. זוהי  
ההודעה האחרונה, ולכן היא נופלת עד  
הסוף.

פלט את אינדקס המשחק המנצח לפורט 2  
חזרה מתת-התכנית

out 2, al ; Output bestseller game index  
ret

```
lea dx, farcry_msg  
int 21h  
jmp best_seller_end
```

```
out_gta5_msg:  
mov ah, 9  
lea dx, gta5_msg  
int 21h
```

```
jmp best_seller_end
```

```
out_ghost_msg:  
mov ah, 9  
lea dx, ghost_msg  
int 21h
```

```
jmp best_seller_end
```

```
out_cyberpunk_msg:  
mov ah, 9  
lea dx, cyberpunk_msg  
int 21h
```

```
jmp best_seller_end
```

```
out_fc25_msg:  
mov ah, 9  
lea dx, fc25_msg  
int 21h
```

```
best_seller_end:
```

## תת תכנית השנייה

מעבד את כל 200 הלקוחות ממיקומי זיכרון 200-400  
מבחין בין סוגי תשלום באמצעות ביט 1 (תשלום עבר או שלא)  
מאתחל מונים - מגדיר שני מונים לתשלומים מוצלחים וכושלים  
מנתח את סטטוס התשלום - בודק את סיבית 1 כדי לקבוע אם התשלום הצליח או נכשל  
מטפל בתשלומים כושלים - מפיך נתונים מקוריים של כשל ומשנה אותם למצב מוצלח  
כל הוראה מוסברת בפירוט, ומציגה את לוגיקת מניפולציית הביט, את השימוש בקופה ובקרת זרימת התוכנית.

מגדיר את אוגר ה-BX ל-200, וקובע את  
כתובת הזיכרון ההתחלתית שבה מתחילים  
נתוני התשלום של הלקוח. ה-BX ישמש  
כמצביע למעבר על רשומות הלקוח.

מאתחל שני אוגרי מונה לאפס: CH =  
מונה עבור לקוחות עם תשלומים  
מוצלחים, CL = מונה עבור לקוחות עם  
תשלומים שנכשלו.

טען בייט אחד של נתוני לקוח ממיקום  
הזיכרון שאליו מצביע BX לתוך אוגר ה-  
AL. בייט זה מכיל את כל המידע המקודד  
של הלקוח.

בודק את סיבית 1 של נתוני הלקוח  
באמצעות פעולת AND בצורת סיבית עם  
מסכה בינארית b00000010 פעולה זו  
מבודדת את סיבית 1 המציינת את סטטוס  
התשלום: אם סיבית 1 = 1 אז התשלום  
הצליח, אם סיבית 1 = 0 אז התשלום נכשל.  
הוראת הבדיקה מגדירה דגלי מעבד מבלי  
לשנות את אוגר ה-AL.

אם תוצאת הבדיקה הייתה 0 (ביט 1 היה  
נקי), קופץ לתונית payment\_failed.  
משמעות הדבר היא שהתשלום לא צלח  
ודורש טיפול מיוחד.

אם הביצוע מגיע לנקודה זו, סיבית 1  
הוגדרה (תשלום מוצלח), לכן הגדל את אוגר  
ה-CH כדי לספור תשלום מוצלח זה.

קופץ ללא תנאי אל payment\_next כדי  
לדלג על קוד עיבוד התשלום שנכשל  
ולהמשיך עם הלקוח הבא.

; Subroutine 2: Check payment success rate

payment\_check:

mov bx, 200 ; Start from memory location 200

mov ch, 0 ; Successful payments counter

mov cl, 0 ; failed payments counter

payment\_loop:

mov al, [bx] ; Get customer data

test al, 00000010b ; Test payment success bit (bit 1)

jz payment\_failed

inc ch ; Count successful payment

jmp payment\_next

מגדיל את רישום ה-CL כדי לספור תשלום כושל זה בסטטיסטיקה שלנו.

מוציא את נתוני הלקוח המקוריים שנכשלו ליציאת קלט/פלט 2.

מוסיף את הערך הבינארי b000000010 ישירות למיקום הזיכרון שאליו מצביע BX. פעולה זו מגדירה את סיבית 1 בנתוני הלקוח המאוחסנים, ומשנה למעשה את סטטוס התשלום בזיכרון מנכשל למוצלח.

טוען מחדש את נתוני הלקוח שהשתנו מהזיכרון לתוך אוגר AL. מוציא את הנתונים המעודכנים לפורט 2, ומציג את רשומת הלקוח לאחר שינוי סטטוס התשלום. זה מספק השוואה לפני/אחרי של שינוי הנתונים.

מגדיל את אוגר ה-BX כדי להצביע למיקום הזיכרון הבא (רשומת הנתונים של הלקוח הבאה).

משווה את BX עם 400 (גבול הקצה). אם BX קטן מ-400, קופץ חזרה ל-payment\_loop כדי לעבד את הלקוח הבא.

מעביר את ספירת התשלומים המוצלחים מ-CH לאוגסטר AL ומוציא אותו לפורט 2.

מעביר את ספירת התשלומים שנכשלו מאוגר CL לאוגסטר AL ומוציא אותו לפורט 2.

חזרה מתת-התכנית

payment\_failed:

inc cl ; Count the failed payments  
out 2, al ; The failed customer data

add [bx], 00000010b ; Change to success payment

mov al, [bx]  
out 2, al ; After changing to successful payment

payment\_next:

inc bx

cmp bx, 400  
jl payment\_loop

mov al, ch  
out 2, al ; Successful payments  
mov al, cl  
out 2, al ; failed payments

ret

## תת תכנית השלישית

מאמת את הצלחת התשלום - סופר רק הכנסות מתשלומים מוצלחים (מזומן תמיד הצליח, הכרטיס תלוי בביט 1) מזהה משחקים - משתמש בביטים 7, 6, 5 כדי לקבוע איזה מבין 8 משחקים נרכש וקובע מחיר בסיס מתאים מיישם הנחות - משתמש בביטים 4, 3 כדי לקבוע את היסטוריית הרכישות של הלקוח ומחיל הנחה מתאימה (2%, 5%, 10% או 20%) מטפל במכירות DLC - בודק את ביט 0 עבור רכישות DLC ומוסיף 80 אם קיים צובר סך הכל - מוסיף את כל מחירי המשחק המוזלים והכנסות ה-DLC לסך הכל המצטבר מפיק תוצאות - שולח את סך הכל של 16 סיביות כשני ערכים של 8 סיביות (בייט נמוך, ואז בייט גבוה) כל פעולה מתמטית לחישוב הנחה מוסברת בפירוט, ומראה כיצד מבוצעים הכפל והחילוק כדי להחיל את הנחות האחוזים.

הגדר את אוגר BX ל-200, שהיא כתובת הזיכרון ההתחלתית שבה מתחילים נתוני הלקוח. כתובת זו תשמש כמצביע כדי לעבור על רשומות הלקוח.

אתחל את מיקום הזיכרון total\_profit (בגודל מילה, 16 סיביות) ל-0. משתנה זה יצבור את סך ההכנסות מכל הרכישות המוצלחות.

טען בייט אחד של נתוני לקוח ממיקום הזיכרון שאליו מצביע BX לתוך אוגר ה-AL. בייט זה מכיל את כל המידע המקודד של הלקוח.

השתמש בפעולת AND עם b00000100 כדי לבדוק את ביט 2, המציין את שיטת התשלום (0=מזומן, 1=כרטיס). פעולה זו קובעת כיצד לבדוק את הצלחת התשלום.

השווה את תוצאת ה-AND עם b00000100. אם שווה (תשלום בכרטיס), עבור אל profit\_pay\_with\_card כדי לאמת את הצלחת התשלום בכרטיס. אחרת, המשך עם התשלום במזומן (בהנחה שהצליח).

עבור אל payment\_pass עבור תשלומי מזומן, אשר תמיד נחשבים מוצלחים וצריכים לתרום לרווח.

טען מחדש את נתוני הלקוח המקוריים לתוך אוגר AL מאחר שפעולת ה-AND הקודמת שינתה אותם.

השתמש בפקודת TEST כדי לבדוק את ביט 1 מבלי לשנות את AL. עבור תשלומי כרטיס, ביט 1 מציין הצלחה (1=מוצלח, 0=נכשל).

אם ביט 1 הוא 0 (תשלום בכרטיס אשראי כושל), דלג ל- profit\_next כדי לדלג על לקוח זה ולעבור ללקוח הבא. לא ייספר רווח עבור תשלומים שנכשלו.

; Subroutine 3: Calculate total profit with discounts applied  
profit\_calc:

mov bx, 200 ; Start from memory location 200  
mov word ptr [total\_profit], 0 ; Clear total profit

profit\_loop:

mov al, [bx] ; Get customer data

; Check if payment was successful

and al, 00000100b ; Only card payment

cmp al, 00000100b

jz profit\_pay\_with\_card

jmp payment\_pass

profit\_pay\_with\_card:

mov al, [bx]

test al, 00000010b ; Test payment success bit (bit 1)

jz profit\_next

אתחל את אוגר ה-CX ל-0. אוגר זה יאחסן את מחיר הבסיס של המשחק שנרכש על ידי הלקוח.

טען נתוני לקוח לתוך AL וצור עותק גיבוי ב-DL. הגיבוי נחוץ מכיוון ש-AL ישתנה במהלך זיהוי המשחק, אך נצטרך את הנתונים המקוריים מאוחר יותר לצורך חישובי הנחות ו-DLC.

השתמשו בפונקציה AND בשילוב עם b11100000 כדי לבדוד את הביטים 7, 6 ו-5, אשר מקודדים את בחירת המשחק (8 משחקים אפשריים).

בדוק אם קטעי המשחק הם Elden ( Elden Ring). אם כן, קבע את מחיר הבסיס ל-250 ש"ח ובעבור לחישוב ההנחה. אחרת, בדוק את המשחק הבא.

בדוק אם קטעי המשחק הם RDR2 (RDR2). אם כן, קבע את מחיר הבסיס ל-230 ש"ח ובעבור לחישוב ההנחה. אחרת, בדוק את המשחק הבא.

בדוק אם קטעי המשחק הם Minecraft (Minecraft). אם כן, קבע את מחיר הבסיס ל-80 ש"ח ובעבור לחישוב ההנחה. אחרת, בדוק את המשחק הבא.

בדוק אם קטעי המשחק הם Far Cry (Far Cry 5). אם כן, קבע את מחיר הבסיס ל-240 ש"ח ובעבור לחישוב ההנחה. אחרת, בדוק את המשחק הבא.

בדוק אם מספר הביטים של המשחק הוא 100 (GTA V). אם כן, קבע את מחיר הבסיס ל-56 ש"ח ובעבור לחישוב ההנחה. אחרת, בדוק את המשחק הבא.

בדוק אם קטעי המשחק הם 101 (רוח צושימה). אם כן, קבע את מחיר הבסיס ל-240 ש"ח ובעבור לחישוב ההנחה. אחרת, בדוק את המשחק הבא.

payment\_pass:

; Get base price based on game

mov cx, 0 ; Base price

mov al, [bx]

mov dl, al ; Keep original data

and al, 11100000b ; Extract game bits

cmp al, 00000000b ; Elden Ring - 250 ש"ח

jnz check\_rdr2\_price

mov cx, 250

jmp apply\_discount\_calc

check\_rdr2\_price:

cmp al, 00100000b ; RDR2 - 230 ש"ח

jnz check\_minecraft\_price

mov cx, 230

jmp apply\_discount\_calc

check\_minecraft\_price:

cmp al, 01000000b ; Minecraft - 80 ש"ח

jnz check\_farcry\_price

mov cx, 80

jmp apply\_discount\_calc

check\_farcry\_price:

cmp al, 01100000b ; Far Cry 5 - 240 ש"ח

jnz check\_gta\_price

mov cx, 240

jmp apply\_discount\_calc

check\_gta\_price:

cmp al, 10000000b ; GTA V - 56 ש"ח

jnz check\_ghost\_price

mov cx, 56

jmp apply\_discount\_calc

check\_ghost\_price:

cmp al, 10100000b ; Ghost of Tsushima - 240 ש"ח

jnz check\_cyberpunk\_price

mov cx, 240

jmp apply\_discount\_calc



בדקו אם ביט המשחק הם 110 (סייברפאנק). אם כן, הגדירו את מחיר הבסיס ל-250 CX ועברו לחישוב ההנחה. אחרת, בדקו את המשחק הסופי.

בדקו אם סיביות המשחק הן 111 (FC 25). אם כן, קבע את מחיר הבסיס ל-255 CX. זוהי אפשרות המשחק האחרונה, כך שהמשחק ממשיך ישירות לחישוב ההנחה.

שחזר את נתוני הלקוח המקוריים מ-DL בחזרה ל-AL, מכיוון ש-AL שונה במהלך זיהוי המשחק.

השתמשו בפונקציה AND לפי סיביות עם הפונקציה b00011000 כדי לבודד את סיביות 4 ו-3, המקודדות את היסטוריית הרכישות של הלקוח (4 רמות: רכישה ראשונה, 2-4 רכישות, 5-8 רכישות, +8 רכישות).

בדקו אם סיביות היסטוריית הרכישות הן 00 (לקוח ראשון). אם כן, יש להחיל הנחה של 2% על ידי הכפלת המחיר ב-98 וחלוקה ב-100. העבר את מחיר הבסיס ל-AX, הכפל ב-98, חלק ב-100 ואחסן את התוצאה בחזרה ב-CX.

בדקו אם סיביות היסטוריית הרכישות הן 01 (2-4 רכישות קודמות). אם כן, החל הנחה של 5% על ידי הכפלת המחיר ב-95 וחלוקת המחיר ב-100. אחסן את המחיר המזול בחזרה ב-CX.

```
check_cyberpunk_price:
    cmp al, 11000000b    ; Cyberpunk - 250
    jnz check_fc25_price
    mov cx, 250
    jmp apply_discount_calc
```

```
check_fc25_price:
    cmp al, 11100000b    ; FC 25 - 255
    jnz apply_discount_calc
    mov cx, 255
```

```
apply_discount_calc:
    ; Apply discount based on purchase history (bits 4,3)
    mov al, dl            ; Restore original data
    and al, 00011000b    ; Extract purchase history bits
```

```
    cmp al, 00000000b    ; 00 - First time (2% discount)
    jnz check_5_discount_calc
    ; Apply 2% discount: price = price * 98 / 100
    mov ax, cx
    mov dx, 98
    mul dx
    mov dx, 100
    div dx
    mov cx, ax
    jmp add_to_total
```

```
check_5_discount_calc:
    cmp al, 00001000b    ; 01 - 2-4 purchases (5% discount)
    jnz check_10_discount_calc
    ; Apply 5% discount: price = price * 95 / 100
    mov ax, cx
    mov dx, 95
    mul dx
    mov dx, 100
    div dx
    mov cx, ax
    jmp add_to_total
```

בדקו אם מספר הביטים של היסטוריית הרכישות הוא 10 (5-8 רכישות קודמות). אם כן, יש להחיל הנחה של 10% על ידי הכפלת המחיר ב-90 וחלוקתו ב-100. אחסן את המחיר המוזל בחזרה ב-CX.

check\_10\_discount\_calc:

```
cmp al, 00010000b ; 10 - 5-8 purchases (10% discount)
jnz check_20_discount_calc
; Apply 10% discount: price = price * 90 / 100
mov ax, cx
mov dx, 90
mul dx
mov dx, 100
div dx
mov cx, ax
jmp add_to_total
```

עבור היסטוריית רכישות 11 (+8) רכישות קודמות, יש להחיל 20% הנחה על ידי הכפלת המחיר ב-80 וחלוקתו ב-100. זוהי רמת ההנחה המקסימלית.

check\_20\_discount\_calc:

```
+8 - 11 ; purchases (20% discount)
; Apply 20% discount: price = price * 80 / 100
mov ax, cx
mov dx, 80
mul dx
mov dx, 100
div dx
mov cx, ax
```

הוסיף את מחיר המשחק המוזל (המאוחסן ב-CX) למיקום הזיכרון total\_profit. פעולה זו צוברת את ההכנסות מרכישת המשחק העיקרית.

add\_to\_total:

```
add [total_profit], cx ; Add discounted game price to total
```

שחזר את נתוני הלקוח המקוריים מגיבוי DL ל-AL לצורך בדיקת רכישת DLC.

```
; Check if customer bought DLC (bit 0)
```

השתמש בפקודת TEST כדי לבדוק את ביט 0, המציין רכישת DLC (1=נרכש, 0=לא נרכש). אם ביט 0 הוא 0, דלג ל-profit\_next כדי לדלג על הכנסות DLC.

```
mov al, dl ; Restore original customer data
test al, 00000001b ; Test DLC bit (bit 0)
jz profit_next ; Skip if no DLC
```

אם נרכש DLC, הגדירו את CX ל-80 (מחיר DLC) וקפצו כדי להוסיף אותו לסך הכל.

```
; Add DLC price(80ש"ח)
```

```
mov cx, 80
jmp add_dlc_to_total
```

הוסיפו את מחיר ה-DLC (80ש"ח) לרווח הכולל.

add\_dlc\_to\_total:

```
add [total_profit], cx ; Add discounted DLC price to total
```

הגדלת BX כדי להצביע על נתוני הלקוח  
הבא בזיכרון.

השווה BX עם 400. אם פחות מ-400, חזור  
ל- profit\_loop כדי לעבד את הלקוח הבא.  
אחרת, המשיך להפקת התוצאות.

טען את ערך הרווח הכולל של 16 סיביות  
מהזיכרון לתוך אוגר AX לצורך פלט.

פלט את הבייט הנמוך (AL) של הרווח  
הכולל לפורט 2. זה מייצג את 8 הביטים  
התחתונים מתוך סך 16 הביטים.

העבר את הבייט הגבוה (AH) ל-AL, ולאחר  
מכן שלח אותו לפורט 2. ההערה מציינת את  
הנוסחה לשחזור הסכום הכולל:  
$$\text{high\_byte} \times 256 + \text{low\_byte} = \text{total\_profit}$$

חזרה מתת-התכנית חזרה לתוכנית  
הקוראת.

profit\_next:

inc bx

cmp bx, 400

jl profit\_loop

; Output total revenue (as two bytes: low byte, then high byte)

mov ax, [total\_profit]

out 2, al ; Output low byte of total profit

mov al, ah ; Move high byte to al

out 2, al ; Output high byte of total profit => high \*

255 + low = total profit

ret

## תת תכנית הרביעית

מאתחל מונים - מגדיר ארבעה מונים נפרדים לכל רמת הנחה (2%, 5%, 10%, 20%)  
מעבד נתוני לקוחות - עובר בלולאה דרך מיקומי זיכרון 200-400 (200 לקוחות בסך הכל) כדי לבחון כל רשומת לקוח  
מסנן תשלומים מוצלחים - סופר רק לקוחות שקיבלו תשלומים מוצלחים (בדיקת ביט 1)  
מחלץ היסטוריית רכישות - משתמש בביטים 3, 4 כדי לקבוע את רמת נאמנות הלקוחות  
מסווג לקוחות - ממייין לקוחות לרמות הנחה מתאימות בהתבסס על היסטוריית רכישות  
מפיק תוצאות - שולח ספירת לקוחות בכל קטגוריית הנחה לפורט 2  
  
כל שורה מוסברת עם מטרתה הספציפית, פעולות מניפולציה של ביט מפורטות.

מגדיר את אוגר ה-BX ל-200, וקובע את כתובת הזיכרון ההתחלתית שבה מתחילים נתוני הלקוח.

מאתחל ארבעה אוגרי מונה לאפס:

CH = מונה ללקוחות חדשים (2% הנחה),  
CL = מונה ללקוחות עם 2-4 רכישות (5% הנחה),  
DH = מונה ללקוחות עם 5-8 רכישות (10% הנחה),  
DL = מונה ללקוחות עם 8+ רכישות (20% הנחה)

טוען בייט אחד של נתוני לקוח מכתובת הזיכרון המאוחסנת ב-BX לתוך אוגר AL.

בדיקת סיבית 1 (סיבית של הצלחת התשלום) של נתוני הלקוח:  
אם סיבית 1 = 0 (התשלום נכשל), קפיצה ל-discount\_next כדי לדלג על לקוח זה.

אם סיבית 1 = 1 (התשלום הצליח), המשיך העיבוד.

מחיל מסיכת סיביות כדי לבודד את סיביות 3 ו-4 (סיביות היסטוריית רכישות), תוך ניקוי כל שאר הסיביות: 00 = לקוח בפעם הראשונה, 01 = 2-4 רכישות קודמות, 10 = 5-8 רכישות קודמות, 11 = 8+ רכישות קודמות.

משווה את הביטים שחולצו עם 00000000b (ביטים 3, 4 שניהם אפס)

אם התאמה: מעלה את מונה ה-CH (לקוחות עם הנחה של 2%) וקפיצה ללקוח הבא.

אם אין התאמה: ממשיך לבדיקת ההנחה הבאה.

משווה עם 00001000b (ביט 3 מוגדר, ביט 4 נקה = תבנית "01")

אם התאמה: מגדיל את מונה ה-CL (לקוחות עם הנחה של 5%) וקפיצה ללקוח הבא.

אם אין התאמה: ממשיך לבדיקת ההנחה הבאה.

; Subroutine 4: Count customers by their discount level

discount\_count:

mov bx, 200 ; Start from memory location 200  
mov ch, 0 ; 2% discount customers  
mov cl, 0 ; 5% discount customers  
mov dh, 0 ; 10% discount customers  
mov dl, 0 ; 20% discount customers

discount\_loop:

mov al, [bx] ; Get customer data

; Check if payment was successful (only count for successful payments)

test al, 00000010b  
jz discount\_next

; Extract purchase history bits

and al, 00011000b ; Extract bits 4,3

cmp al, 00000000b ; 00 - First time customers (2% discount)

jnz check\_5\_discount\_count

inc ch

jmp discount\_next

check\_5\_discount\_count:

cmp al, 00001000b ; 01 - 2-4 purchases (5% discount)

jnz check\_10\_discount\_count

inc cl

jmp discount\_next

משווה עם B00010000 (ביט 4 מוגדר, ביט 3 נקה = תבנית "10")

אם התאמה: מגדיל את מונה ה-DH (לקוחות הנחה של 10%) וקופץ ללקוח הבא.

אם אין התאמה: ממשיך לקטגוריית ההנחה הסופית.

אם אף אחת מהדפוסים הקודמים לא תואמת, ללקוח חייבת להיות דפוס "11" (+8 רכישות), כך שמונה ה-DL יעלה (לקוחות עם 20% הנחה).

מגדיל את BX כדי להצביע על מיקום הזיכרון הבא.

משווה את BX עם 400 (גבול קצה).

אם  $BX < 400$ , קופץ חזרה ל- discount\_loop כדי לעבד את הלקוח הבא.

טווח כולל: מעבד 30 לקוחות (מיקומי זיכרון 20-49).

מוציא את הספירות הסופיות ברצף לפורט 2:

פלט ראשון: מספר לקוחות שקיבלו 2% הנחה.

פלט שני: מספר לקוחות שקיבלו 5% הנחה.

פלט שלישי: מספר לקוחות שקיבלו 10% הנחה.

פלט רביעי: מספר לקוחות שקיבלו 20% הנחה.

מחזיר את השליטה לתוכנית הקוראת.

מסיים את התוכנה.

check\_10\_discount\_count:

cmp al, 00010000b ; 10 - 5-8 purchases (10% discount)

jnz check\_20\_discount\_count

inc dh

jmp discount\_next

check\_20\_discount\_count:

+8 - 11 ; purchases (20% discount)

inc dl

discount\_next:

inc bx

cmp bx, 400

jl discount\_loop

; Output customer counts by discount level

mov al, ch

out 2, al

; Customers

with 2% discount

mov al, cl

out 2, al

; Customers

with 5% discount

mov al, dh

out 2, al

; Customers

with 10% discount

mov al, dl

out 2, al

; Customers

with 20% discount

ret

end\_program:

mov ah, 4ch

int 21h

end

## רפלקציה לפרויקט

בפרויקטי פגשתי מלאה קטעים קשים ובו זמנית דברים קלים. הפרויקט מחיל פקודות אשר לא נלמדו בכיתה כגון, `test`, `lea` ועבודה עם משתנה בעלה 16 סיביות. הקטעים הקשים ביותר היו להבין איך זה יעבוד, כי הרעיון היה קיים אבל הפתרון היה רחוק מההשגה. מספר סרטונים הסבירו לי כיצד להדפיס קטע של הודעה למסך השחור הנקרא `standard output`. בסרטון היה מוסר שבעזרת פקודת `lea` ניתן להציג קטע של הודעה אשר שמורה כמשתנה בתוך `datan`. הפקודה של `test` נמצאה בטעות, ברגע שהבנתי שהתהליך של `and`, `cmp` לא תמיד נוה לעבודה. התחלתי לחפש פקודה נוספת אשר תעשה תהליך דומה ל`and` אך לא תשנה את ערכו. מצאתי את `test`, הפקודה הייתה קצת עם חשיבה הפוכה. ברגע שנרצה להבין האם המספר הוא קיים באותו המקום אז הוא לא ידליק לנו את דגל האפס אך ברגע שהם לא מתאימים הוא כן ידליק. והתהליך העבודה עם האוגר `ax` אשר מכיל 16 סיביות היה הכי בעייתי ובו זמנית דיי מובן. ההבנה של העובדה שהמספר בנוי מ2 אוגרים הייתה מהירה אך הפועל של הסדר שלהם הייתה הבעיה. בחלק זה עבדתי לא רק עם הגוגל אלה גם עם בינות מלאכותיות. הקושי שלי היה להבין איזה אוגר (`al`, `ah` המרכיבים את `ax`) מציין איזה מספר. יש מספרים עליונים, אלה שעברו את גבול של 255 וכבר מציין כמה פעמים היו עוד 255. ומכאן התהליך של החישוב ושל ההדפסה הלכה בצורה חלקה.

הקטעים הקלים היו לעבוד עם הספר ועם הפקודות הרגילות כמו, `and`, `cmp`, `add`, `inc`, `jz` וכו'. כל הפקודות אשר עבדנו איתם כבר בכיתה מאז או לפחות חשבנו בחשיבה של לעבוד איתם הם הלכו בצורה קלה. כל הפקודות אלה אפילו מוסברות לנו במחברות לכן העבודה איתם הייתה ברורה ביותר.

לגבי מערכת המחשב אין לי מה יותר מדי להרכיב ובכל זאת העובדה שלמדתי שהאוגרים הגדולים מורכבים ממספר אוגרים קטנים. וששוב העובדה שהמחרוזות שמור בזיכרון כרצף של תווים.

בתחילת העבודה היו לי מלאה שגיאות בקוד כגון אחד מהם שנקרה בגלל הרצון להדפיס את `ax` ישירות עם הפקודה `out`. ואחרי שהבנתי איך עובדת החישוב ב`ax` אז מהר מאוד התגברתי על בעיה זו. הבעיות האחרות בהם נתקעתי במהלך הכתיבה של הקוד היו בעיות רגילות כגון, בעיות חשיבה לא תקינה ושגיאות של כתיבה לא נכונה של הפקודה.

באמצע העבודה הבנתי שהיו קטעים מוגזים וכבר הייתי מוכן לוותר על הרבה מהם. התנת תכנית אשר מחשבת את הרוויח הסופי. בתת תוכנה זו היה הבעיה המרכזית שלי אשר אחרי שסידרתי את התהליך של `ax`, היה נשאר לי רק לחשב את גם את ההנחות ולכלול לריווח הסופי וכאן הייתי תקוע לזמנים רבים. התהליך של החישוב שזה ההנחה מינוס מאה כפול מספר ואז חילוק במאה מביאה את אותו הסכום עם ההנחה הייתה זריזה אך הסדר של תהליכים שאני אחשב את ההנחה לסכום נכון ולא למספרים אשר לא קשורים לאותו הלקוח היה הבעיה המרכזית.