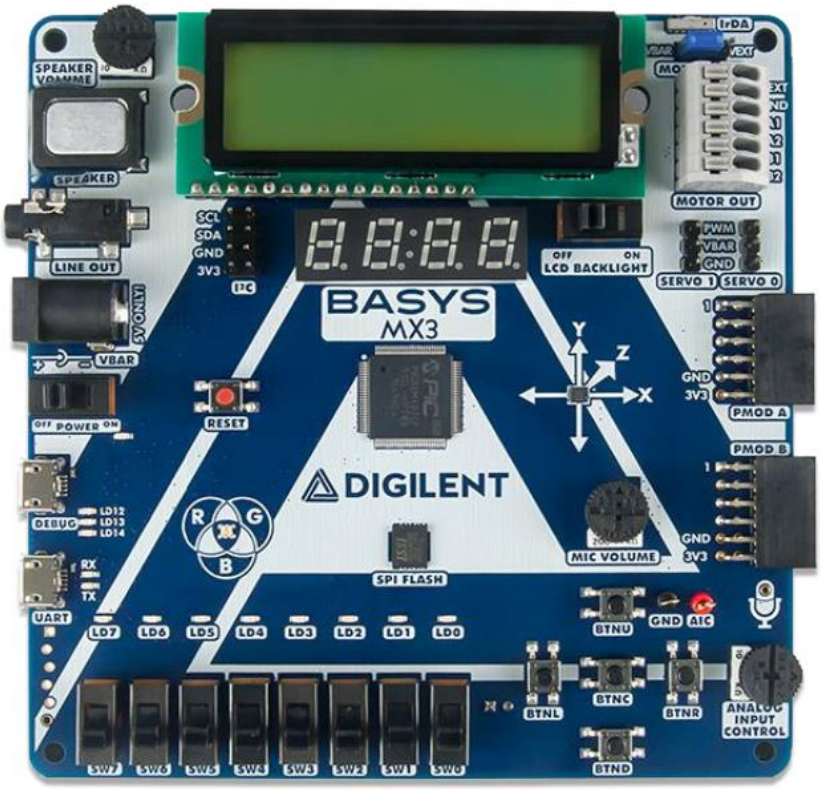
פרויקט 2 במבנה המחשב



מגישים:

תום שחר – 311494066

ניר קקון – 312203672

אדיר לייבטמן – 206828881

תיאור כללי

פרויקט זה הינו המשך ישיר של הפרויקט הראשון. בפרויקט זה מימשנו את רכיבי התוכנה אותם כתבנו בפרויקט הראשון לאחר שינויים והתאמות נדרשים על גבי רכיב חומרה - כרטיס מחשב מסוג BASYS MX3 של חברת DIGILENT.

השלבים והמשימות איתם התמודדנו במהלך הפרויקט:

- כתיבת תוכנית אסמבלי לצורך מימוש סטופר.
- עדכון האסמבלר מהפרויקט הראשון לצורך התאמת הפלט לרכיב החומרה ולפעולות חדשות.
- התאמת הסימולטור מהפרויקט הראשון לסביבת העבודה החדשה תוך התחשבות בפסיקות ובהפעלת רכיבי קלט פלט (מסכים, נורות לד, כפתורים וכו').

האסמבלר אותו ערכנו קורא את קובץ האסמבלי (שני קבצים סה"כ - סטופר ו-fib), מפענח את הפקודות לשפת מכונה (ספרות הקסאדצימליות) ופולט קובץ txt עם זיכרון התוכנית לפני תחילת הביצוע שלה. האופן בו האסמבלר פועל בפרויקט זה זהה כמור לזה שבפרויקט הראשון, אך כעת במקום ש-opcodes מס' 16-18 יהוו רזרבות הם יהוו פקודות כפי שיפורט בהמשך. את הזיכרון של כל תוכנית נזין בשני מערכים בסימולטור – אחד בפורמט char והשני int לטובת פעולת הסימולטור.

הסימולטור יסמלץ את מעבד ה - SIMP , כאשר כל הוראה במעבד מתבצעת במחזור שעון אחד. המעבד עובד בתדר של 1024 הרץ - מבצע 1024 הוראות אסמבלי בשנייה. כמו כן הסימולטור יודע להתמודד עם פסיקות אותן הוא מקבל מהמשתמש ע"י לחיצות כפתור או מהחומרה ע"י ספירת מחזורי השעון עד לערך מסוים שנקבע ע"י קוד האסמבלי, ולהציג פלטים בהתאם. פעולות אלה מתבצעות ע"י שימוש ברכיבי קלט פלט.

רגיסטרי חומרה

IORegister[14] הינו מערך של 14 "רגיסטרי חומרה", כפי שמתכנת האסמבלי חושב שהם, שמאותחלים לאפס ויסומלצו ע"י הסימולטור:

IORegister Number	Name	number bits	Meaning
0	irq0enable	1	IRQ 0 enabled if set to 1, otherwise disabled.
1	irq1enable	1	IRQ 1 enabled if set to 1, otherwise disabled.
2	irq2enable	1	IRQ 2 enabled if set to 1, otherwise disabled.
3	irq0status	1	IRQ 0 status. Set to 1 when irq 0 is triggered.
4	irq1status	1	IRQ 1 status. Set to 1 when irq 1 is triggered.
5	irq2status	1	IRQ 2 status. Set to 1 when irq 2 is triggered.
6	irqhandler	9	PC of interrupt handler
7	irqreturn	9	PC of interrupt return address
8	clks	32	cyclic clock counter. Starts from 0 and increments every clock. After reaching 0xffffffff, the counter rolls back to 0.
9	reserved	32	Reserved for future use.
10	display7seg	16	Connected to 7-segment display of 4 letters. Each 4 bits displays one digit from 0 – F, where bits 3:0 control the rightmost digit, and bits 15:12 the leftmost digit.
11	timerenable	1	1: timer enabled 0: timer disabled
12	timercurrent	32	current timer counter
13	timermax	32	max timer value

תמיכה בהתקני קלט / פלט

לצורך מימוש הסטופר באסמבלי, נוסיף למעבד תמיכה בקלט ופלט ע"י 3 הוראות חדשות, "רגיסטרי חומרה", ופסיקות. על מנת לגשת לרגיסטרי החומרה נשתמש באופקודים 17 ו- 18, שהיו reserved בפרויקט הראשון, תוך שימוש בקבוע, וכמו כן נוסיף הוראת חזרה מפסיקה באופקוד 16:

Opcode	Name	Meaning
16	reti	PC = IORegister[7]
17	in	R[rd] = IORegister[R[rs] + R[rt]]
18	out	IORegister [R[rs]+R[rt]] = R[rd]

הפקודות שמומשו בפרויקט הראשון, כולל הפסאודו-קוד word., נותרו ללא שינוי והסימולטור תומך גם בהן.

פעולת הסימולטור

כאמור, הסימולטור מבוסס על הסימולטור אותו מימשנו בפרויקט הראשון ומבצע בכל לולאה fetch-decode-execute. הסימולטור שמור בזיכרון ה-flash של הכרטיס ולכן לאחר טעינה ראשונית של הקוד הכרטיס מבצע את הסימולציה באופן עצמאי ללא התערבות נוספת, עוד הוא מחובר למקור מתח. להלן השינויים אותם ביצענו בסימולטור על מנת לתמוך בדרישות ביצועי התקני הקלט-פלט והממשק עם המעבד:

- התחשבות בהתקני קלט / פלט – קריאת ערכים מרכיבי הקלט (כפתורים, סוויצ'ים ושעון) ופליטת ערכים אל רכיבי הפלט ע"י שימוש באינדיקטורים ופונקציות ייעודיות.
- הגדרת רגיסטרי חומרה ותמיכה בהזנת ערכים אליהם וקריאת ערכים מהם.
- התחשבות בקצב פעולת שעון הכרטיס על מנת לקצוב שניות בשיטת polling, קצב עדכון השניות יאפשר לסימולטור לבצע בקרה על קצב ריצת פקודות האסמבלי.

בניגוד לפרויקט הראשון בו הסימולטור קיבל קובץ txt בקלט – בפרויקט זה הסימולטור נטען מראש בשתי תוכניות אסמבלי, סטופר ופיבונאצ'י (לכן כלל הפונ' אשר משמשות לקריאת וכתיבת קבצי טקסט נמחקו מהסימולטור של הכרטיס, אך נותרו כפי שהן בגרסת ה-PC). לאחר שהסימולטור יוצא ממצב RESET או מופעל לראשונה אחת מהתוכנית מופעלת כאשר הבחירה מתבצעת ע"י קריאת הערך מ-SW7, כאשר הסוויץ' כלפי ממטה (0) תטען תוכנית פיבונאצ'י, כאשר הוא כלפי מעלה תטען תוכנית סטופר. עם תחילת הפעולה רכיבי הקלט פלט, כולל השעון, מאותחלים (ע"י פונ' initiation) וערך ה-PC מאותחל לאפס, נבחרת תוכנית כאמור ע"י SW7 ונכנסים ללולאה המרכזית האינסופית אשר מבצעת את פעולות הסימולטור.

מיד לאחר מכן התכונית תתחיל להתבצע בקצב של 1024 פעולות בשנייה, דרישה זו הציבה בפנינו אתגר שאותו פתרנו ע"י שימוש בשתי התניות - אחת בסוף הלולאה מייצרת השהייה במידה ומימוש הוראת האסמבלי הנוכחית הייתה מהירה מדי ביחס לקצב הרצוי, והשנייה בתחילת הלולאה לפני ביצוע ההוראה בודקת האם בוצעו 1024 הוראות אסמבלי לפני שעברה שנייה – במידה וכן תשהה את המשך ביצוע ההוראות עד שתעבור שנייה כאשר החיווי לכך מתבסס על השעון של החומרה.

כמו כן, דאגנו לסדר את מיקום הפונקציות כך שהחומרה תוכל לקלות מידע מהמשתמש ע"י רכיבי הקלט פלט ולעדכן רכיבים אחרים בהתאם, גם כאשר ברגע נתון לא מתבצעות פעולות אסמבלי (סיום תוכנית / הפעלת PAUSE / מתבצעת השהייה לטובת התאמת קצב המעבד).

במהלך מימוש הסימולטור נעזרנו בפונ' מובנות של הכרטיס השמורות בספריות btn.h, lcd.h, ssd.h וכו'.

בנוסף יצרנו סימולטור אשר קורא וכותב קבצי txt בדומה לסימולטור מהפרויקט הראשון, אך לא מבצע את פעולות הקלט-פלט כפי שמבוצע בכרטיס אלא רק תומך בפעולות החדשות, מעדכן מערך של רגיסטרי חומרה ומעדכן אותם ע"פ הפקודות החדשות, תומך בטיימר ובפסיקה 0 כפי שיפורט בהמשך, תומך בפסיקה 1 ו-2 בהנחה שקוד האסמבלי ידרוש אותן ומוודא שלא תתבצע פסיקה בתוך פסיקה – אופן המימוש של פעולות אלו דומה לזה שבכרטיס ולכן לא נפרט על קובץ זה בנפרד.

פונ' התקני קלט / פלט אשר נוספו לסימולטור

- **אתחול החומרה בתחילת הריצה initiation** – בכל לחיצה על RESET או הדלקה ראשונית של הכרטיס מתבצע אתחול של רכיבי הקלט פלט ע"י פונ' מובנות של הכרטיס השמורות בספריות:

```
// This function initiates the microchip's IO modules
void initiation()
{
    // libraries initialization for the IO modules
    LCD_Init();
    LED_Init();
    SWT_Init();
    BTN_Init();
    SSD_Init();
}
```

- **בחירת תוכנית אסמבלי ע"י SW7** - עדכון המערכים הגלובליים ע"י העתקת המערכים של התוכנית שנבחרה כל פי מפסק SW7, ההעתקה מתבצעת באמצעות פונ' **memory_cpy** ו- **memin_cpy** אשר לא נפרט עליהן כאן (פונ' טכניות שמבצעות העתקת מערכים ע"י מעבר בכל רכיבי המערך):

```
if (SWT_GetValue(7) == 0) // if SW7 is off -> load "fib" memin
{
    int memory_fib[MAX_MEMO_LINES] = { 0x00D01001, 0x04DD1007, 0x0E301020, 0x0D100006, 0x0F201021, 0x13000000, 0x
    char memin_arr_fib[MAX_MEMO_LINES][MAX_LINE] = { "00D01001", "04DD1007", "0E301020", "0D100006", "0F201021",
    memory_cpy(memory_fib, memory, MAX_MEMO_LINES); // copy fib memory to the global array
    memin_cpy(memin_arr_fib, memin_arr, MAX_MEMO_LINES, MAX_LINE); // copy fib memin to the global array
}

else // if SW7 is on -> load "stopper" memin
{
    int memory_stopper[MAX_MEMO_LINES] = {0x007013FF, 0x1270100D, 0x00701001, 0x1270100B, 0x00901000, 0x00A00000,
    char memin_arr_stopper[MAX_MEMO_LINES][MAX_LINE] = {"007013FF", "1270100D", "00701001", "1270100B", "00901000
    memory_cpy(memory_stopper, memory, MAX_MEMO_LINES); // copy stopper memory to the global array
    memin_cpy(memin_arr_stopper, memin_arr, MAX_MEMO_LINES, MAX_LINE); // copy stopper memin to the global array
}
```

- **קציבת שניות בשיטת polling** – מתבצעת קריאה של מס' מחזורי השעון של הכרטיס שעברו מרגע הפעלתו/לחיצה על RESET, פונ' **clk_poll()** מבצעת את הקריאה ומעבירה את הנתון למשתנה עם 64 ביטים היות והקוצב של השעון מכיל 32 ביטים בלבד, כך שבכל פעם שמזוהה overflow בקוצב השעון הפונ' מקדמת את 32 הביטים המשמעותיים (msb) באחד. היות והגדרנו את קצב השעון להיות [MHz] 40 לאחר כל פעם שנקרא את הערך הנוכחי נבצע בדיקה האם עברו 40 מיליון מחזורי שעון מהפעם האחרונה שעברה שעבר מספר זהה של מחזורים, במידה וכן – נסיק שעברה שנייה ונבצע פעולות שברצוננו לבצע רק בחלוף שנייה ולא בכל איטרציה של הלולאה המרכזית (האינסופית) – עדכון המסך (גורם לדיליי משמעותי) ועדכון אינדיקטורים עבור פונ' אחרות. בין ניקוי המסך לעדכון המסך ישנה השהייה לטובת פעולה תקינה של המסך:

```
clk_poll(); // update clk64 according to the microchip's clock
seconds = clk64 / CLKS_IN_SECOND;
int beginning_time = clk64; // used later to determine how long each assembly command is being processed

if (seconds != last_seconds) // if a second has passed
{
    second_ind = 1;
    execute_allow = 1;
    last_seconds = seconds;
    LCD_DisplayClear(); // clear the lcd before updating it
    DelayAprox10Us(1000); // delay required between lcd clear to lcd wright
    lcd_update(BTNL, seconds, memory);
}
// This function updates clk62 to the current number of clock cycles of the
// microchip that has been passed since last "RESET"
static void clk_poll(void)
{
    clk32 = _CP0_GET_COUNT();

    if (clk32 < clk32_prev) {
        clk64_msb++;
    }
    clk32_prev = clk32;
    clk64 = (((unsigned long long) clk64_msb) << 32) | clk32;
}
```

- **קליטת לחיצות על לחצני btn** – בכל איטרציה של הלולאה המרכזית מתבצעת בדיקה האם אחד מהכפתורים BTNR/BTNL/BTNC/BTNR/BTND נלחצו. במידה וכן, מעודכן אינדיקטור גלובלי מערך 0 לערך 1 על מנת לסמן שהכפתור נלחץ. על מנת לסנן רעשים ולוודא שאכן בוצעה לחיצה, אחת שנקלטה לחיצה ע"י הפונ' **btn_check()** מתבצעת השהיה של כ- [ms] 70, במידה ולאחר השהיה זו הכפתור עדיין לחוץ הפונ' מסיקה שהכפתור אכן נלחץ ולא מדובר ברעש – ורק אז מעדכנת את האינדיקטור:

```
void btn_check()
{
    if (BTN_GetValue('L') == 1) // check if BTNL was pressed
    {
        DelayAprox10Us(7000);
        if (BTN_GetValue('L') == 1) // check again that BTNL is pressed after a delay to make sure it's not noise
            BTNL = 1;
    }
    if (BTN_GetValue('R') == 1) // check if BTNR was pressed
    {
        DelayAprox10Us(7000);
        if (BTN_GetValue('R') == 1) // check again that BTNR is pressed after a delay to make sure it's not noise
            BTNR = 1;
    }
    if (BTN_GetValue('U') == 1) // check if BTNU was pressed
    {
        DelayAprox10Us(7000);
        if (BTN_GetValue('U') == 1) // check again that BTNU is pressed after a delay to make sure it's not noise
            BTNU = 1;
    }
    if (BTN_GetValue('D') == 1) // check if BTND was pressed
    {
        DelayAprox10Us(7000);
        if (BTN_GetValue('D') == 1) // check again that BTND is pressed after a delay to make sure it's not noise
            BTND = 1;
    }
    if (BTN_GetValue('C') == 1) // check if BTNC was pressed
    {
        DelayAprox10Us(7000);
        if (BTN_GetValue('C') == 1) // check again that BTNC is pressed after a delay to make sure it's not noise
            BTNC = 1;
    }
```

- **מצבי single step/pause** – לאחר בדיקת כפתורי btn מתבצעת פונ' **pause_check()** אשר בודקת אם הופעל pause (לחיצה על BTNL), במידה וכן יעודכן אינדיקטור pause_ind שלא יאפשר ביצוע הוראת אסמבלי כל עוד הוא דולק וכמו כן יאופס אינדיקטור הכפתור על מנת לזהות לחיצה נוספת.

  עם זאת, במידה ובנוסף לאינדיקטור הדולק תזוהה לחיצה על BTNR – יעודכן אינדיקטור step_ind ותבוצע הוראת אסמבלי בודדת, כלומר single step, במהלכה האינדיקטור יאופס. כמו כן אינדיקטור הכפתור יאופס בפונ' על מנת לזהות לחיצה נוספת.

  כאשר תזוהה לחיצה על BTNL כאשר כבר נבחר מצב pause, הפונ' תבטל את ה-pause ע"י איפוס האינדיקטור:

```
// This function executes "PAUSE" - return 1 if PAUSE is activated, return 2 if SINGLE STEP is activated, return 0 if PAUSE is deactivated
void pause_check()
{
    if (((BTNL == 1) && (pause_ind == 0)) || (((BTNL == 0) && (pause_ind == 1)) && (BTNR == 0))) // user clicked BTNL to activate PAUSE
    {
        pause_ind = 1;
        BTNL = 0;
    }
    if ((BTNL == 0) && (pause_ind == 1) && (BTNR == 1)) // user clicked BTNR while PAUSE to activate SINGLE STEP
    {
        BTNR = 0;
        step_ind = 1;
    }
    if (((BTNL == 1) && (pause_ind == 1)) || ((BTNL == 0) && (pause_ind == 0))) // user clicked BTNL to deactivate PAUSE or didn't click BTNL at all
    {
        BTNL = 0;
        pause_ind = 0;
    }
}
```

- זיהוי וטיפול בפסיקות – פסיקה למעשה היא הדרך של המשתמש לתקשר באמצעות רכיבי החומרה עם התכנה. בפרויקט שלנו היו 3 פסיקות:
פסיקה 0 – פסיקה המעדכנת על כל מעבר של שנייה.
פסיקה 1 – זיהוי לחיצה כל כפתור BTNC. פסיקה זו משמשת לצורך השהייה בתוכנית הסטופר.
פסיקה 2 – זיהוי לחיצה על כפתור BTND. תודה פסיקה זו מאפסת את הסטופר.
הפונ' שמזהה ומטפלת בפסיקות היא **irq_check()**, הדרישה לביצוע פסיקה מתקבלת כאשר החומרה מזהה שאחד מהתנאים לעיל מתקיים ובהתאם מדליקה את רגיסטר ה-status ע"י שינוי ערכו ל-1. קוד האסמבלי, אשר בודק האם ה-irqstatus של אחת הפסיקות הופעל ע"י החומרה, יזהה שהתקבל irqstatus ויבצע את הפעולות הבאות:
  - עדכון רגיסטר irqhandler ל-PC בו תתבצע הפסיקה.
  - הפעלת רגיסטר enable המתאים אשר יסמן לחומרה שניתן לבצע את הפסיקה.

לאחר מכן החומרה תזהה שישנו גם irqenable וגם irqstatus פעילים, ובמידה ולא מתבצעת פסיקה ברגע זה (חיווי על כך מתקבל ע"י אינדיקטור irq_ongoing_ind) החומרה תשמור את ה-PC הנוכחי לחזרה ברגיסטר irqreturn, תעדכן את הPC לערך שנשמר ע"י קוד האסמבלי ב-irqhandler ותפעיל את האינדיקטור שמסמן על טיפול בפסיקה. במידה והתקבלו כמה פסיקות בו זמנית תתבצע פסיקה אחת בכל זמן נתון, הראשונה כתלות בPC שבו הופעל הstatus והשאר ע"פ הסדר שבו קוד האסמבלי בודק את הסטאטוסים – קודם פסיקה 0, לאחר מכן 1 ולבסוף 2.
בסיום כל פסיקה בקוד האסמבלי ישנה פקודת reti, בזיהוי פקודה זו במהלך פונ' **execute()** הסימולטור יכבה את אינדיקטור irq_ongoing_ind על מנת לסמן על סיום טיפול בפסיקה וכמו כן ה-PC יעודכן לערך השמור ברגיסטר irqreturn.

```
// This function updates irq registers according to irqstatus and irqenable
void irq_update()
{
    if (BTNC == 1)
    {
        IO_register_arr[4] = 1; // irq1status = 1
        BTNC = 0; // reset indicator to detect next activation
    }

    if (BTND == 1)
    {
        IO_register_arr[5] = 1; // irq2status = 1
        BTND = 0; // reset indicator to detect next activation
    }

    // check if one of the irq's enable and status are "1" at the same time, if so and there is no ongoing irq - jump to irq
    int irq_ind = ((IO_register_arr[0] && IO_register_arr[3]) || \
    (IO_register_arr[1] && IO_register_arr[4]) || (IO_register_arr[2] && IO_register_arr[5]));
    if ((irq_ind == 1) && (irq_ongoing_ind == 0))
    {
        IO_register_arr[7] = PC; // load PC to irqreturn register
        PC = IO_register_arr[6]; // load irqhandler register's value to PC
        irq_ongoing_ind = 1; // indicates that there is a ongoing irq
    }
    else
        return;
}
```

- **עדכון תצוגת LCD** – הכרטיס שלנו מכיל מסך LCD בעל שתי שורות הצגה, הפונ' שמעדכנת את התצוגה בהתאם למצב הסוויצ'ים SW0 ו-SW1 היא **lcd_update()**, פונ' זו מתבצעת אחת לשנייה היות והיא גורמת לדיליי משמעותי ביחס לשאר הפונ' בסימולטור. שורת ההצגה הראשונה מציגה ערכים שונים לפי בחירת המשתמש כאשר הבחירה מתבצעת ע"י הפעלת הסוויצ'ים הרלוונטיים:
  - כאשר SW0 = off, SW1 = off ההוראה INST שאותה מריצים כעת תוצג בשורה הראשונה באותו פורמט כמו ה-TRACE בפרויקט הראשון.
  - כאשר SW0 = on, SW1 = off תוכן הרגיסטרים יוצג בשורה הראשונה בפורמט
  - RXX=YYYYYYYY כאשר XX מייצג את מספר הרגיסטר שמוצג ברגע בספרות דצימליות, החל מרגיסטר 0 ו-YYYYYYYY הינו תוכנו ב-8 ספרות הקסדצימליות. באמצעות לחיצה על BTNU

ניתן לקדם את הרגיסטר המוצג ב-1 כאשר הקידום הינו ציקלי וחוזר להתחלה לאחר הרגיסטר האחרון.

- כאשר SW0 = off, SW1 = on תוכן הזכרון יוצג בשורה הראשונה בפורמט MAAA = DDDDDDDD כאשר AAA הינו הכתובת בזכרון ו-DDDDDDDD מייצג את הדאטה. כאשר BTNU מאפשר קידום של הכתובת.
- כאשר SW0 = on, SW1 = on יוצגו מספר ההוראות שבוצעו עד עכשיו בספירה הקסדצימלית.

השורה השניה במסך תציג לנו את הזמן שעבר מאז היציאה מ-RESET, את ה-PC וה-RSP.
ההדפסה נקבע לפי ערכי הסוויצ'ים וההדפסה מתבצעת ע"י שימוש מובנת של ספריות החומרה LCD_WriteStringAtPos המקבלת את תוכן ההדפסה, שורת ההדפסה (מבין ה-2) ומיקום התו.
כמו כן, כאשר מציגים רגיסטרים/זיכרון, ישנה בדיקה בפונ' אשר מקדמת את מס' הרגיסטר/כתובת הזיכרון ומאפסת את אינדיקטור הכפתור BTNR על מנת לזהות לחיצה נוספת ע"י המשתמש. אופן מימוש הפונ' לעדכון השורה השנייה ושתי דוגמאות להדפסת ועדכון השורה הראשונה:

```
// This function updates the data that the LCD displays according to SW0 and SW1
void lcd_update(int BTNL, int seconds, int memory[MAX_MEMO_LINES])
{
    // update second line:

    char PC_buffer[3];
    char RSP_buffer[3];
    char TIME_buffer[8];

    sprintf(PC_buffer, "%03x", PC); // convert pc to hexa and store it in a buffer
    LCD_WriteStringAtPos(PC_buffer, 1, 0);

    sprintf(RSP_buffer, "%03x", register_arr[13]); // convert $sp content to hexa and store it in a buffer
    LCD_WriteStringAtPos(RSP_buffer, 1, 4);

    sprintf(TIME_buffer, "%08d", seconds); // convert seconds content to hexa and store it in a buffer
    LCD_WriteStringAtPos(TIME_buffer, 1, 8);

    // update first line:

    // SW0=OFF and SW1=OFF:
    if (SWT_GetValue(0) == 0 && SWT_GetValue (1) == 0)
    {
        char MEMORY_buffer[8];
        sprintf(MEMORY_buffer, "%08x", memory[PC]); // convert memory[PC] to hexa and store it in a buffer
        LCD_WriteStringAtPos(MEMORY_buffer, 0, 0); // display next instruction to be executed in the lcd
    }

    // SW0=OFF and SW1=OFF
    if (SWT_GetValue(0) == 1 && SWT_GetValue (1) == 0)
    {
        char hex_i[2];
        char hex_reg[8];
        sprintf(hex_i, "%02x", LCD_reg_index); // convert register number to hexa
        sprintf(hex_reg, "%08x", register_arr[LCD_reg_index]); // convert register value to hexa

        // display the register and its value
        LCD_WriteStringAtPos("R", 0, 0);
        LCD_WriteStringAtPos(hex_i, 0, 1);
        LCD_WriteStringAtPos("=", 0, 4);
        LCD_WriteStringAtPos(hex_reg, 0, 6);
        if (BTNU == 1) // if BTNU was pressed show next register
        {
            LCD_reg_index++;
            LCD_reg_index = LCD_reg_index%16; // if currently watching R15 and clicked "BTNU" the next register shown will be R0
            BTNU = 0; // restart indicator to detect next click on BTNU
        }
    }
```

- **מסך SSD** – תצוגה נוספת המכילה ספרות הניתנות להצגה ע"י 7 סיגמנטים לכל ספרה ונקודה עשרונית (ממומש ע"י דיודות). מסך זה מתעדכן ע"י פונ' **ssd_update()** המתבצעת אחת לכל כמה מחזורי שעות של מעבד ה-SIMP וממומשת ע"י פונ' מובנית:

```
// This function updates the SSD (second display) according to display7seg register
void ssd_update()
{
    SSD_WriteDigitsGrouped(IO_register_arr[10], 0);
}
```

ניהול נורות ה-LED – כחלק מהממשק עם המשתמש הכרטיס שלנו מדליק ומכבה נורות בחיווי לביצוע פעולת reti אשר באמור מתבצעת כאשר מסתיימת פסיקה, בפעם הראשונה שתתבצע הוראת reti תדלק נורה מס' 0, בפעם הבא נורה מס' 0 תכבה ותדלק מס' 1, עד שלבסוף תדלק נורה מס' 7 ולאחר מכן היא תכבה ותדלק שוב נורה מס' 0 – וחוזר חלילה. פונ' **led_update()** מממשת את הפונקציונליות הנ"ל בעזרת פונ' מובנית LED_SetValue() המקבלת את מספר הלד הרצוי ואת הערך 0/1 המייצג הדלקה / כיבוי, וכמו כן אינדיקטור:

```
// This function makes sure that the right led is on when a "reti" command is encountered
void led_update()
{
        LED_SetValue(current_LED, 0); // Turn off old LD
        current_LED++;
        current_LED = (current_LED) % 8;
        LED_SetValue(current_LED, 1); // Turn on current LD
}
```

- **עדכון מחזורי השעון ומימוש טיימר** – הכרטיס שלנו תומך בטיימר בעל 32 ביטים אשר מעדכן את פסיקת האפס. הטיימר מאופשר ע"י רגיסטר enable וגם נעצר על ידו בהתאם. באמצעות שימוש ברגיסטרי החומרה הרלוונטיים וקצב ריצת שעון הכרטיס - החומרה תעדכן את התוכנה כאשר תעבור שנייה ע"י הפעלת דרישה לפסיקה 0 באמצעות רגיסטר irq0status. השתמשנו ברגיסטר timermax על מנת להגדיר את כמות הפעולות שמבוצעות במשך זמן של שנייה ובאמצעות שימוש ברגיסטר timercurrent ביצענו השוואה האם בוצעו 1024 הוראות אסמבלי מתחילת השנייה הנוכחית – כלומר עברה שנייה.
לאחר כל ביצוע הוראת אסמבלי מתבצעת קריאה לפונ' **cycles()** אשר מקדמת את רגיסטר clks ובמידה והופעל רגיסטר timerenable תקדם גם את רגיסטר timercurrent. במידה ו-timercurrent יכיל ערך הזהה לערך הסף שהוגדר ע"י קוד האסמבלי ברגיסטר timermax הפונ' תאפס את רגיסטר timercurrent ותפעיל דרישה לפסיקה 0 ע"י הדלקת רגיסטר irq0status:

```
// This function updates IORegisters: "clks" and "timercurrent" according to "timermax"
void cycles()
{
    IO_register_arr[8]++; // increment clock counter
    if (IO_register_arr[11] == 1)
    {
        IO_register_arr[12]++; // increment timercurrent
        if (IO_register_arr[12] == IO_register_arr[13]) // if timercurrent reached timermax then restart the timer
        {
            IO_register_arr[12] = 0; // reset timercurrent
            IO_register_arr[3] = 1; // request irq0 has been called to - a second has passed
        }
    }
}
```

- **קצב ריצת המעבד** – כאמור, נדרשנו לסמלץ קצב ריצת מעבד של 1024 פעולות אסמבלי בשנייה, פתרנו ע"י שימוש בשתי התניות:
  - אחת בסוף הלולאה מייצרת השהייה במידה ומימוש הוראת האסמבלי הנוכחית הייתה מהירה מדי ביחס לקצב הרצוי:

```
// control minimum time of assembly command execution
do
{
    clk_poll();
    current_time = clk64;
}
while((current_time - beginning_time) < COMMAND_REQUIRED_TIME);
```

  - השנייה בתחילת הלולאה לפני ביצוע ההוראה בודקת האם בוצעו 1024 הוראות אסמבלי לפני שעברה שנייה – במידה וכן תשהה את המשך ביצוע ההוראות עד שתעבור שנייה באמצעות אינדיקטור execute_allow כאשר החיווי למעבר שנייה second_ind מתבסס על קריאת השעון של החומרה שכאמור עובד בקצב [MHz] 40:

```
if (((IO_register_arr[8] % 1024) == 0) && (IO_register_arr[8] != 0)) // check if 1024 commands has been executed
{
    if (second_ind != 1) // if they were executed in less than 1 second - don't proceed to next command
    {
        execute_allow = 0;
    }
    else // else - proceed to next command and check each iteration if a second has passed, by using "second_ind"
    {
        second_ind = 0;
    }
}
```

ההתניה שמאפשרת את השהיה זו וכן את הפעלת pause וביצוע הוראת halt (סוף התוכנית) הינה:

```
// if PAUSE is activated or HALT was executed or it's not the time to execute a command (execute_allow) -> then don't execute)
if (((pause_ind == 1) && (step_ind == 0)) || (halt_ind == 1) || (execute_allow == 0))
    continue;
```

תוכנית הסטופר באסמבלי

תוכנית הסטופר באסמבלי מממשת מונה לזמן מרגע הפעלת הכרטיס/לחיצה על RESET של הכרטיס כאשר הזמן מוצג על גבי מסך ה-SSD. הזמן מוצג בצורה הבאה: MM:SS ופועל באופן מעגלי – לאחר הזמן 59:59 השעון מתחיל את הספירה מחדש מהזמן 00:00. בנוסף, הסטופר אותו בנינו יודע לממש pause של הסטופר בלבד (לא pause של החומרה – כלומר לא עצירה של ביצוע הוראות אסמבלי) באמצעות פסיקה מס' 1 באמצעות לחיצה על BTNC והפסקת הpause בלחיצה נוספת, וגם איפוס של ספירת הזמן בסטופר ע"י פסיקה 2 באמצעות לחיצה על BTND.

אופן פעולת הסטופר:

- **איפוס הטיימר** - תחילה מתבצע initialization של רגיסטרי החומרה הרלוונטיים (timermax, timerenable) ואיפוס של הרגיסטרים אשר באמצעותם נספור ונציג את הזמן:
  ספרה ראשונה מימין (שניות בודדות) - $s0.
  ספרה שנייה מימין (עשרות שניות) - $s1.
  ספרה שנייה משמאל (דקות בודדות) - $s2.
  ספרה ראשונה משמאל (עשרות דקות) - $a0.
  רגיסטר $a1 משמש לשמירת ערך הזמן כספרות הקסדצימאליות, לדוגמה בזמן 16:23 הוא יכיל את הערך 0x1623.

```
        # initialize IOregisters:

        add $t2, $zero, $imm, 1023          # t2 = 1023, the max timer value // TODO 1023
        out $t2, $zero, $imm, 13            # export 1023 to timermax IOregister
        add $t2, $zero, $imm, 1             # $t2 = 1
        out $t2, $zero, $imm, 11            # export the value 1 to the timerenable IOregister

        # initialize stopper digits:

        add  $s0, $zero, $imm, 0            # $s0 contains the current clock 1st digit (starts from 1 since the first IRQ0 is after 1 second)
        add  $s1, $zero, $zero, 0           # $s1 contains the current clock 2nd digit
        add  $s2, $zero, $zero, 0           # $s2 contains the current clock 3rd digit
        add  $a0, $zero, $zero, 0           # $a0 contains the current clock 4th digit
        add  $a1, $zero, $zero, 0           # $a1 contains the current clock hexa value, including all the 4 digits
```

- **שגרת בדיקת פסיקות** – מתבצעת בדיקה אינסופית האם התקבלה בקשה לפסיקה מהחומרה, כלומר אם רגיסטר irqstatus של אחת הפסיקות הודלק (שווה בערכו ל-1). במידה וכן הקוד יעדכן את רגיסטר irqhandler ל-PC של ה-Label המתאים למס' הפסיקה ולאחר מכן יפעיל את רגיסטר irqenable המתאים ע"י עדכון ערכו ל-1.

```
IRQ CHECK 0:

        in $t0, $zero, $imm, 3              # check if irq0status IOregister is ON
        beq $imm, $zero, $t0, IRQ CHECK 1   # if $t0 == 0 then jump to IRQ CHECK 1, a second didn't pass yet
        add $t0, $zero, $imm, IRQ0          # $t0 = the PC of IRQ0
        out $t0, $zero, $imm, 6             # export IRQ0 PC to irqhandler IOregister
        out $imm, $zero, $zero, 1           # a second passed - change irq0enable IOregister to 1


IRQ CHECK 1:

        in $t0, $zero, $imm, 4              # check if irq1status IOregister is ON
        beq $imm, $zero, $t0, IRQ CHECK 2   # if $t0 == 0 then jump to IRQ CHECK 2, BTNC hasn't been pressed yet
        add $t0, $zero, $imm, IRQ1          # $t0 = the PC of IRQ1
        out $t0, $zero, $imm, 6             # export IRQ0 PC to irqhandler IOregister
        out $imm, $zero, $imm, 1            # change irq1enable IOregister to 1


IRQ CHECK 2:

        in $t0, $zero, $imm, 5              # check if irq2status IOregister is ON
        beq $imm, $zero, $t0, IRQ CHECK 0   # if $t0 == 0 then jump to IRQ CHECK 0, BTND hasn't been pressed yet
        add $t0, $zero, $imm, IRQ2          # $t0 = the PC of IRQ2
        out $t0, $zero, $imm, 6             # export IRQ0 PC to irqhandler IOregister
        add $t0, $zero, $imm, 1             # $t0 = 1
        out $t0, $zero, $imm, 2             # change irq2enable IOregister to 1
        beq $imm, $zero, $zero, IRQ CHECK 0 # return to IRQ check routine
```

- **ביצוע פסיקה 0** – תחילה, נבדוק האם הספרה ראשונה מימין (שניות בודדות) שווה 9 – כלומר צריכה להתעדכן כעת ל0 (כביכול "overflow"), במידה ולא – נקדם את הספרה ב-1 ונקפוץ ל-SEGMENTS UPDATE שבו נעדכן את רגיסטר $a1 עם ערך השניות המעודכן ע"פ הפורמט שהוסבר לעיל, לבסוף הקוד יודיע לחומרה על סיום הפסיקה ע"י הורדת irq0status ו-irq0enable ל-0 (כדי שהחומרה תוכל לקלוט את הפעם הבאה שתבוצע פסיקה) וביצוע פקודת reti על מנת לחזור לשגרת בדיקת הפסיקות. במידה והופעלו יותר מפסיקה אחת, הפסיקה הראשונה שתטופל תהיה רנדומלית כתלות במיקום ה-PC כאשר הופעלו הפסיקות, ולאחר סיום הטיפול בה תטופל הפסיקה הבאה ע"פ הסדר שהוגדר בשגרת הבדיקה – תחילה פסיקה 0, לאחר מכן פסיקה 1 ולבסוף פסיקה 2:

```
IRQ0:

        add $t2 , $zero , $imm, 9                       # $t2 = 9 -> used to check 1st digit overflow
        beq $imm , $t2 , $s0, INCREMENT SECOND DIGIT    # check if the 1st digit is about to overflow (bigger than 9), if so - jump to inner loop
        add $s0 , $s0 , $imm, 1                         # increment 1st digit: $s0 = $s0 + 1
        beq $imm, $zero, $zero, SEGMENTS UPDATE         # jump to SEGMENTS UPDATE after incrementing


SEGMENTS UPDATE:

        add $a1, $zero, $a0, 0                          # add the 4th digit to $a1
        sll $a1, $a1, $imm , 4                          # shift the digits one hexa-digit left
        add $a1, $a1, $s2, 0                            # add the 3rd digit to $a1
        sll $a1, $a1, $imm , 4                          # shift the digits one hexa-digit left
        add $a1, $a1, $s1, 0                            # add the 2nd digit to $a1
        sll $a1, $a1, $imm , 4                          # shift the digits one hexa-digit left
        add $a1, $a1, $s0, 0                            # add the 1st digit to $a1

        # now $a1 contains the current time in the following format: 0xMmSs (M = 10*minute, m = minutes, S = 10*second, s = seconds)

        out $a1, $zero, $imm, 10                        # display current clock value by exporting it to display7seg IOregister

        # return:

        out $zero, $zero, $imm, 3                       # change irq0status IOregister to OFF
        out $zero, $zero, $zero, 0                      # change irq0enable IOregister to OFF
        reti $zero, $zero, $zero, 0                     # no overflow, return to IRQ CHECK
```

במידה והספרה אכן 9 – נאפס את מונה הספרה ראשונה מימין (שניות בודדות) ונקפוץ ל-INCREMENT SECOND DIGIT אשר יבדוק אם הספרה השנייה מימין (עשרות שניות) שווה 5 - כלומר צריכה להתעדכן כעת ל-0, ובמידה ולא נקדם את הספרה ב-1 ונקפוץ ל-SEGMENTS UPDATE כמפורט מעלה. במידה והספרה אכן 5 – נעבור לספרה השלישית והרביעית באותו אופן (INCREMENT THIRD DIGIT, INCREMENT FOURTH DIGIT):

```
INCREMENT SECOND DIGIT:

        add $t2 , $zero , $imm, 5                       # $t2 = 5 -> used to check 2nd digit overflow
        beq $imm , $t2 , $s1, INCREMENT THIRD DIGIT     # check if the 2nd digit is about to overflow (bigger than 5), if so - jump to inner loop
        add $s1 , $s1 , $imm, 1                         # increment 2nd digit: $s1 = $s1 + 1
        add $s0 , $zero , $imm, 0                       # reset 1st digit: $s0 = 0
        beq $imm, $zero, $zero, SEGMENTS UPDATE         # jump to SEGMENTS UPDATE after incrementing



INCREMENT THIRD DIGIT:

        add $t2 , $zero , $imm, 9                       # $t2 = 9 -> used to check 3rd digit overflow
        beq $imm , $t2 , $s2, INCREMENT FOURTH DIGIT    # check if the 3th digit is about to overflow (bigger than 9), if so - jump to inner loop
        add $s2 , $s2 , $imm, 1                         # increment 3rd digit: $s2 = $s2 + 1
        add $s0 , $zero , $imm, 0                       # reset 1st digit: $s0 = 0
        add $s1 , $zero , $imm, 0                       # reset 2nd digit: $s1 = 0
        beq $imm, $zero, $zero, SEGMENTS UPDATE         # jump to SEGMENTS UPDATE after incrementing


INCREMENT FOURTH DIGIT:

        add $t2 , $zero , $imm, 5                       # $t2 = 5 -> used to check 4th digit overflow
        beq $imm , $t2 , $a0, RESET IRQ0                # check if the 4th digit is about to overflow (bigger than 5), if so - jump to RESET
        add $a0 , $a0 , $imm, 1                         # increment 4th digit: $a0 = $a0 + 1
        add $s0 , $zero , $imm, 0                       # reset 1st digit: $s0 = 0
        add $s1 , $zero , $imm, 0                       # reset 2nd digit: $s1 = 0
        add $s2 , $zero , $imm, 0                       # reset 3rd digit: $s2 = 0
        beq $imm, $zero, $zero, SEGMENTS UPDATE         # jump to SEGMENTS UPDATE after incrementing
```

בטיפול בספרה הרביעית, INCREMENT FOURTH DIGIT, במידה והספרה הראשונה משמאל (עשרות דקות) שווה 5 נסיק שהזמן האחרון היה 59:59 ובעת נדרש לאפס את המונה בחזרה ל- 00:00, לכן נקפוץ ל- RESET IRQ0 אשר בדומה לתחילת הקוד יאפס את הרגיסטרים שבאמצעותם נספר הזמן, ובנוסף יודיע לחומרה על סיום הפסיקה ע"י הורדת irq0status ו- irq0enable ל-0 וביצוע פקודת reti על מנת לחזור לשגרת בדיקת הפסיקות:

```
RESET IRQ0:

        out $zero, $zero, $imm, 10          # display 0 by changine display7seg IOregister to 0
        add  $s0, $zero, $imm, 0            # $s0 contains the current clock 1st digit (starts from 1 since the first IRQ0 is after 1 second)
        add  $s1, $zero, $zero, 0           # $s1 contains the current clock 2nd digit
        add  $s2, $zero, $zero, 0           # $s2 contains the current clock 3rd digit
        add  $a0, $zero, $zero, 0           # $a0 contains the current clock 4th digit
        add  $a1, $zero, $zero, 0           # $a1 contains the current clock hexa value, including all the 4 digits
        out $zero, $zero, $imm, 3           # change irq0status IOregister to OFF
        reti $zero, $zero, $zero, 0         # no overflow, return to IRQ CHECK
```

- **ביצוע פסיקה 1** – תחילה נאפס את רגיסטר irq1status על מנת שהחומרה תדע שנדרש לזהות שוב לחיצה על BTNC כדי לצאת ממצב ה-pause, עד אז תתבצע לולאה אינסופית אשר בודקת אם רגיסטר irq1status הופעל שוב (כלומר זוהתה לחיצה נוספת על BTNC) ובמידה וכן הקוד יוריד את רגיסטרים irq1status ו-irq1enable חזרה ל0 ותבוצע פקודת reti על מנת לחזור לשגרת בדיקת הפסיקות:

```
IRQ1:

        out $zero, $zero, $imm, 4           # change irq1status IOregister to 0

IRQ1_EXECUTE:

        in $t0, $zero, $imm, 4              # check if irq1status IOregister is 1, meaning BTNC has been clicked again
        beq $imm, $zero, $t0, IRQ1_EXECUTE  # if $t0 == 0 then BTNC hasn't been clicked yet, resart loop

        # return:

        out $zero, $zero, $imm, 4           # turn irq1status IOregister to OFF
        out $zero, $zero, $imm, 1           # turn irq1enable IOregister to OFF
        reti $zero, $zero, $zero, 0         # BTNC has been clicked, return to the PC that was saveed before handling IRQ1
```

- **ביצוע פסיקה 2** – בדומה לשאר החלקים שמבצעים RESET בקוד, נאפס את הרגיסטרים שבאמצעותם נספר הזמן, ובנוסף יודיע לחומרה על סיום הפסיקה ע"י הורדת status1irq ו- enable1irq ל-0 וביצוע פקודת reti על מנת לחזור לשגרת בדיקת הפסיקות:

```
IRQ2:

        # perform reset:

        out $zero, $zero, $imm, 5           # change irq2status IOregister to 0
        out $zero, $zero, $imm, 10          # display 0 by changine display7seg IOregister to 0
        add  $s0, $zero, $imm, 0            # $s0 contains the current clock 1st digit (starts from 1 since the first IRQ0 is after 1 second)
        add  $s1, $zero, $zero, 0           # $s1 contains the current clock 2nd digit
        add  $s2, $zero, $zero, 0           # $s2 contains the current clock 3rd digit
        add  $a0, $zero, $zero, 0           # $a0 contains the current clock 4th digit
        add  $a1, $zero, $zero, 0           # $a1 contains the current clock hexa value, including all the 4 digits

        # return:

        out $zero, $zero, $imm, 5           # turn irq2status IOregister to OFF
        out $zero, $zero, $imm, 2           # turn irq2enable IOregister to OFF
        reti $zero, $zero, $zero, 0         # the reset has been done, return to the PC that was saveed before handling IRQ2
```