בס"ד

אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

קובץ הכנה ניסוי מעבדה מס' 3

Tutorial 3.1 – GPIO (General Purpose Input Output)

Led, Switch, Push-Button

מעבדת מיקרומחשבים – המחלקה להנדסת חשמל ומחשבים

מס' קורס - 361.1.3353

כתיבה ועריכה: חנן ריבוא

מהדורה 1 – שנה"ל תשע"ו

בקובץ זה נלמד לכתוב תוכנית לביצוע פעולות I/O בין הבקר לבין העולם החיצון (רכיבי I/O המחוברים לבקר – לדים, לחצנים וכדומה).

חומר קריאה מקדים: עמודים 407-414 (ללא עמודים 411,412) בספר המעבדה (MSP430x4xx user guide)

★ הקדמה:

נחדד מס' נקודות לפני שניגש להגדרת הבעיה שעלינו לפתור.

1. אסמבלר - תוכנה שנמצא בסביבת הפיתוח וממירה קוד בשפת אסמבלי לקוד בשפת מכונה (מותאם לשפת מכונה ספציפית – במקרה שלנו בקר MSP430).
2. כמו שלמדנו, בשפת אסמבלי (של בקר MSP430) יש 27 **פקודות ליבה** (Core Instruction) שרק הן מתורגמות לקוד מכונה ע"י האסמבלר ועוד 24 **פקודות חיקוי** (Emulated Instruction) הנועדות לנוחות המשתמש ומומרות ישירות לפקודות ליבה ע"י האסמבלר ורק אז מקודדות לקוד מכונה.
3. בשונה מפקודות (Instructions) ישנן הוראות (**Directives**) שאין להם תרגום לקוד מכונה (אינן רצות ב-CPU). הוראות אלו נועדו לאסמבלר לצורך ביצוע פעולות מקדימות, למשל: הקצאת מקום בזיכרון עבור משתנים/מחרוזות, תיחול ערכי משתנים, חלוקת הזיכרון למקטעים מסוימים וכו'.
4. ישנם 2 סוגי זיכרונות:

✓ **זיכרון RAM** - זיכרון זה נדיף (כאשר **מכבים** את המתח מהבקר תוכנו נמחק). כאשר מבצעים RESET תוכן זיכרון ה-RAM נשמר (ביצוע RESET משמעו טעינת ערך כתובת הפקודה הראשונה של התוכנית לרגיסטר PC **ולא איפוס מתח הבקר**).
זיכרון זה נועד לשימוש "שולחן עבודה- דף טיוטה" עבור התוכנית שלנו ולשימוש המחסנית (STACK).

✓ **זיכרון FLASH** - זיכרון זה אינו נדיף (כאשר מכבים את המתח מהבקר תוכנו נשמר).
סביבת הפיתוח טוענת לאזור זה בזיכרון את תוכנית הקוד שנכתוב ואת הקבועים אותם נרצה לשמור ללא תלות במצב המתח של הבקר.
כדי לבצע **כתיבה לזיכרון FLASH** מתוך תוכנית שנכתוב לא נוכל לכתוב ישירות לזיכרון זה.
יש צורך לעבוד עם מתווך הנקרא Flash Controller (נלמד זאת בקורס "מבנה מחשבים ספרתיים").
לעומת זאת קריאה מזיכרון זה נבצע כרגיל.

בתרגול זה, סביבת הפיתוח תעשה את הכתיבה ל-FLASH בשבילנו בשלב מקדים לפני טעינת הקוד.
ה-Directives המתאימים לצורך כך,

```
RSEG   CODE
; your code
END
```

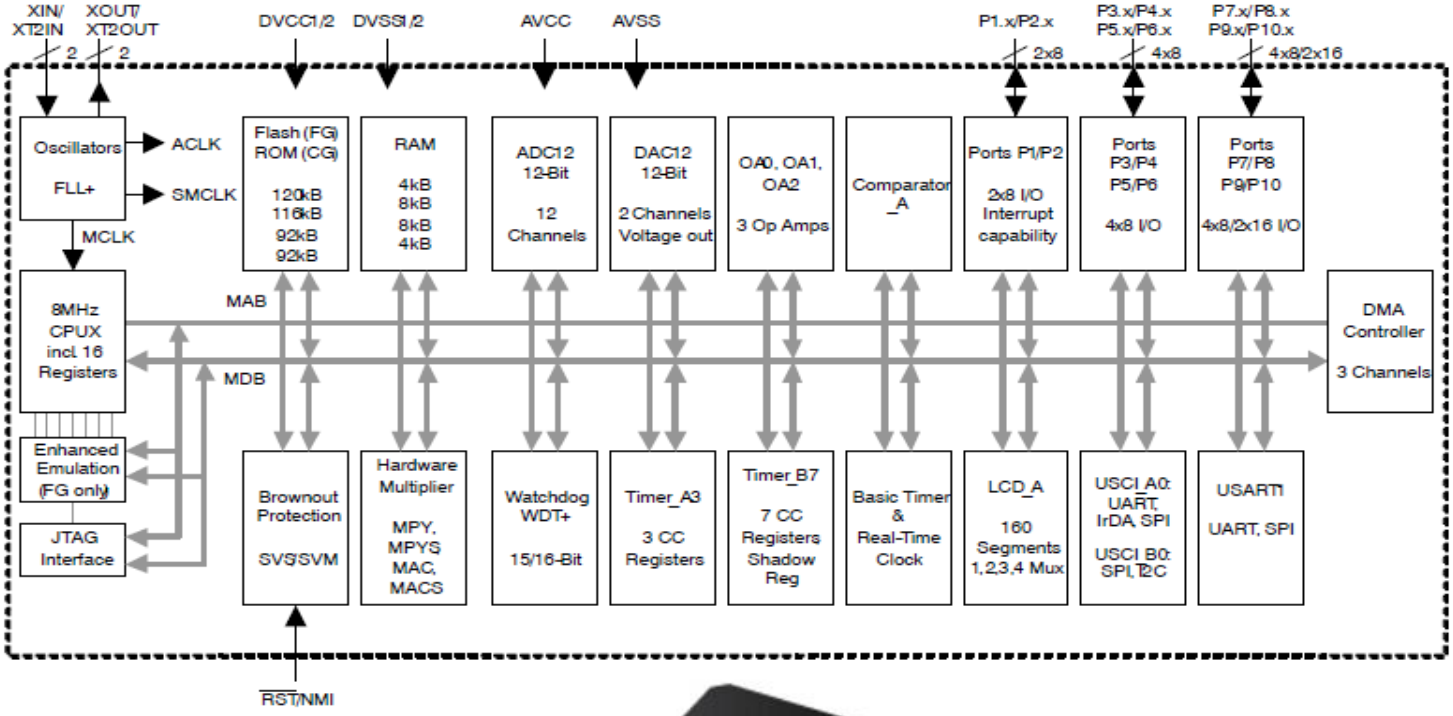
```
RSEG   DATA
; your constant
END
```

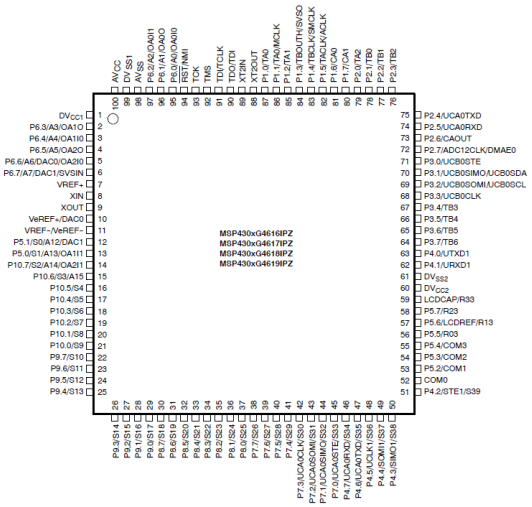
שימוש ב- DATA -**RSEG** מצריך שינוי קטן בקובץ ה- Linker של ברירת המחדל.
כדי להימנע מזה, אנו נעדכן משתנים קבועים **בזיכרון ה-FLASH** בצורה הבאה:

בתחילת זיכרון ה-FLASH ימוקמו **הקבועים**
ואחריהם תמוקם **התוכנית**.

```
RSEG   CODE
; your constant
; your code
END
```

★ MSP430FG461x Digital I/O:





1) PORT:

- פורט (PORT) הינה מילה שקולה ל- 8 רגליים של הבקר המשמשים ל- I/O.
- לרכיב הבקר הנ"ל ישנם 10 פורטים של I/O , הנקראים P1 עד P10.
- ניתן **לתכנת** כל פורט בצורה פרטנית וברמת הרגל הבודדת, לשמש כ- INPUT או OUTPUT.
- כדי לתכנת PORT (בגודל Byte, רצף של 8 רגליים בבקר) ברמת הרגל הבודדת, לכל PORT מוקצים שלושה רגיסטרים ייעודיים **PxIN**, **PxOUT**, **PxDIR** ולכל רגל בקר ישנו ביט מקביל מתאים ברגיסטרים אלו.

Register	Designation	Functionality
PxDIR	**קביעת כיווניות** רגל הבקר (INPUT or OUTPUT)	$OUTPUT \rightarrow Bit = 1$, $INPUT \rightarrow Bit = 0$
PxOUT	**קביעת** ערך לוגי **במוצא** ברגל הבקר (3.3v or 0v)	$high = '1' \rightarrow Bit = 1$, $low = '0' \rightarrow Bit = 0$
PxIN	קריאת ערך מתח לוגי ברגל הבקר	$high = '1' \rightarrow Bit = 1$, $low = '0' \rightarrow Bit = 0$

- באופן כללי רגלי הבקר משמשים למודולי חומרה נוספים, לצורך כך ישנו רגיסטר נוסף **PxSEL** לצורך ברירה בין מודולי החומרה המשמשים באותה רגל בקר. לבחירת רגל במצב I/O ערך הביטים של **PxSEL** שווה 0, זהו ערך ברירת המחדל של רגיסטר זה.
- הטבלה הבאה מתארת את ערך הרגיסטרים לצורך קינפוג PORT1 במצב GPIO .

Port P1 (P1.0 to P1.5) pin functions

PIN NAME (P1.X)	X	FUNCTION	CONTROL BITS / SIGNALS	
			P1DIR.x	P1SEL.x
P1.0/TA0	0	P1.0 (I/O)	I: 0; O: 1	0
		Timer_A3.CCI0A	0	1
		Timer_A3.TA0	1	1
P1.1/TA0/MCLK	1	P1.1 (I/O)	I: 0; O: 1	0
		Timer_A3.CCI0B	0	1
		MCLK	1	1
P1.2/TA1	2	P1.2 (I/O)	I: 0; O: 1	0
		Timer_A3.CCI1A	0	1
		Timer_A3.TA1	1	1
P1.3/TBOUTH/SVSOUT	3	P1.3 (I/O)	I: 0; O: 1	0
		Timer_B7.TBOUTH	0	1
		SVSOUT	1	1
P1.4/TBCLK/SMCLK	4	P1.4 (I/O)	I: 0; O: 1	0
		Timer_B7.TBCLK	0	1
		SMCLK	1	1
P1.5/TACLK/ACLK	5	P1.5 (I/O)	I: 0; O: 1	0
		Timer_A3.TACLK	0	1
		ACLK	1	1

Port P1 (P1.6 and P1.7) pin functions

PIN NAME (P1.X)	X	FUNCTION	CONTROL BITS / SIGNALS		
			CAPD.x	P1DIR.x	P1SEL.x
P1.6/CA0	6	P1.6 (I/O)	0	I: 0; O: 1	0
		CA0	1	X	X
P1.7/CA1	7	P1.7 (I/O)	0	I: 0; O: 1	0
		CA1	1	X	X

NOTE 1: X: Don't care.

2) LED:

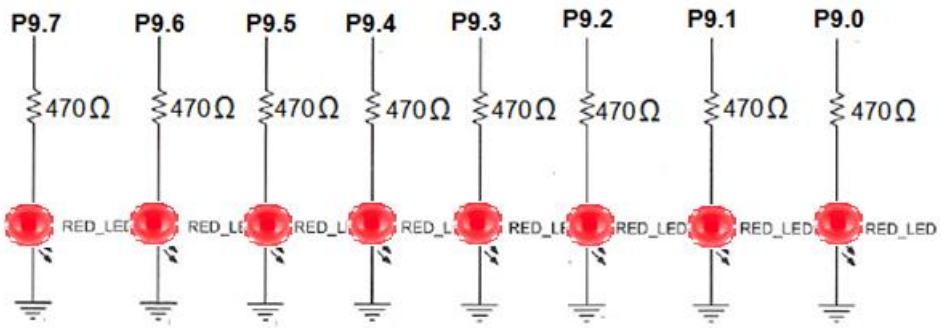
חיבור חומרתי של הלדים לרגלי הבקר מתואר באיור הבא.

כאשר ערך הביט המתאים ברגיסטר P9OUT שווה '1' לוגי הלד המתאים דולק, עבור '0' לוגי הלד כבוי.

תרגיל קצר: ברצוננו להדליק את הלדים במיקום הזוגי ולכבות את הלדים במיקום האי זוגי.

```
bis.b   #0xff,&P9DIR
bis.b   #0xAA,&P9OUT
bic.b   #0x55,&P9OUT
```

```
bis.b   #0xff,&P9DIR
mov.b   #0xAA,&P9OUT
```



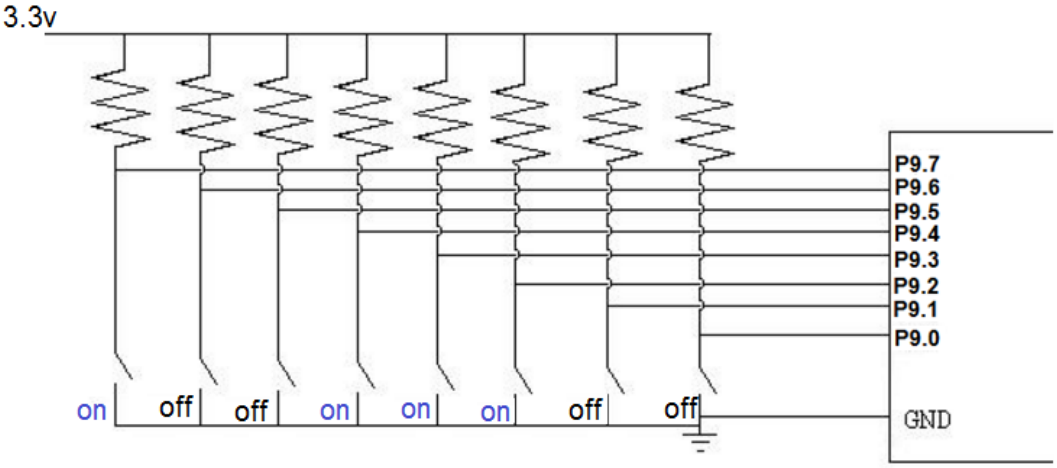
P9OUT	1	0	1	0	1	0	1	0	= 0xAA
	7	6	5	4	3	2	1	0	

P9OUT	0	1	0	1	0	1	0	1	= 0x55
	7	6	5	4	3	2	1	0	

3) Switch: ברצוננו לקרוא את המתח במערך של שמונת המתגים, לרגיסטר R5.

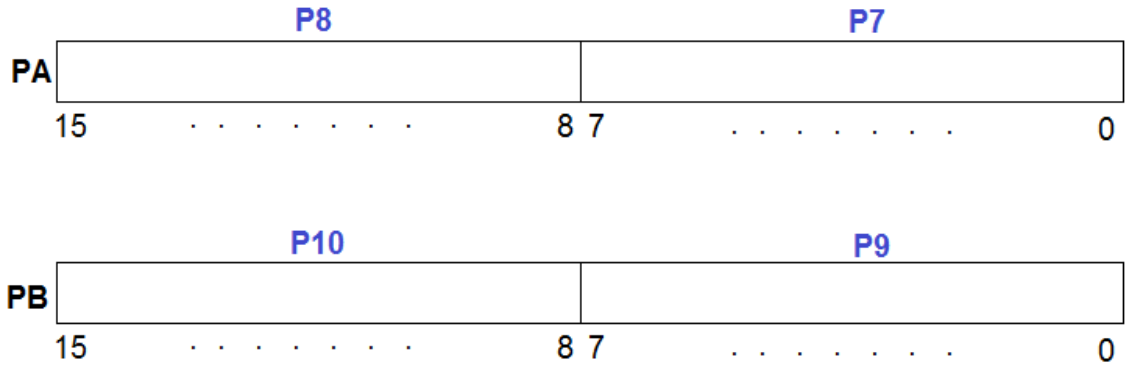

```
bic.b   #0xff,&P9DIR
mov.b   &P9IN, R5

R5 = 0x63
```



4) ניתן לגשת לפורטים P7,P8 (כ"א באורך 8 רגליים) כזוג הנקרא PA (באורך מילה, 16 רגליים), כ"כ ניתן לגשת לפורטים P9,P10 (כ"א באורך 8 רגליים) כזוג הנקרא PB (באורך מילה, 16 רגליים).

Port PA 16 bit **P7+P8**	Port PA selection	PASEL	03Eh
	Port PA direction	PADIR	03Ch
	Port PA output	PAOUT	03Ah
	Port PA input	PAIN	038h
Port PB 16 bit **P9+P10**	Port PB selection	PBSEL	00Eh
	Port PB direction	PBDIR	00Ch
	Port PB output	PBOUT	00Ah
	Port PB input	PBIN	008h



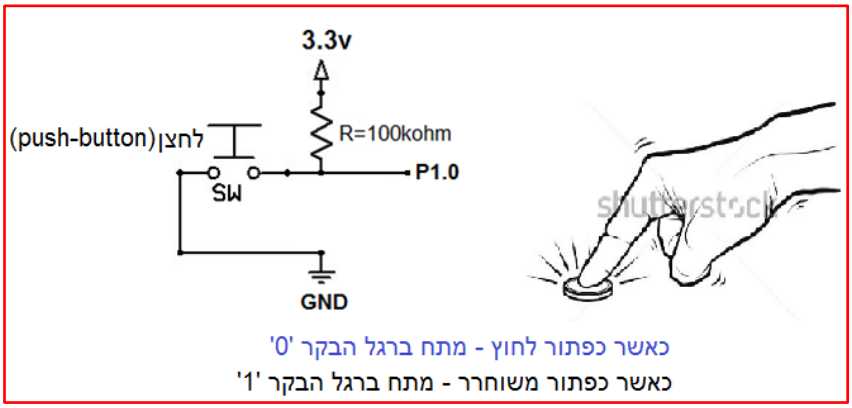
Write a Word to Port_A

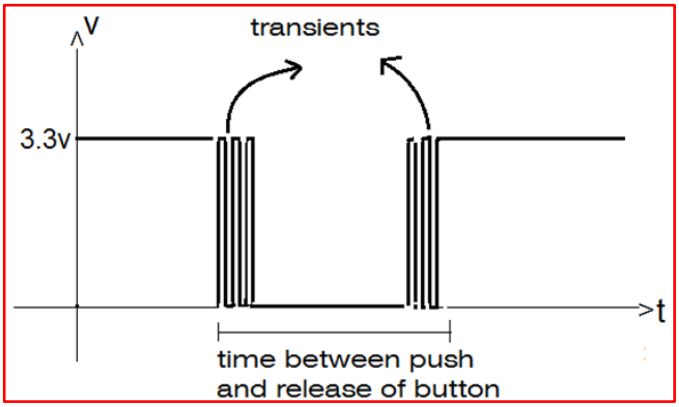
```
bis.w   #0FFFFh,&PADIR      ; all PA.x output
bis.w   #0FFFFh,&PAOUT      ; Set all PA pins to '1'
```

Read a Word from Port_A

```
bic.w   #0FFFFh,&PADIR      ; all PA.x input
mov.w   &PAIN,R5            ; Read a Word from PA to R5
```

5) Push-Button:





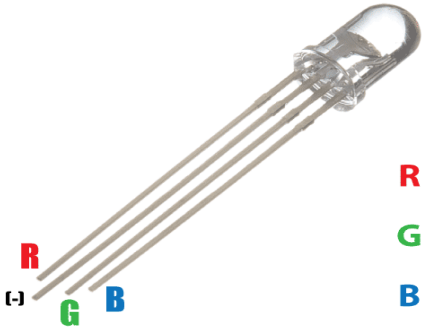
הסבר:

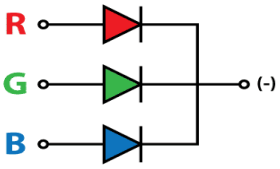
חיבור חומרתי של הלחצן לרגל הבקר מתואר באיור משמאל. כאשר לחצן לחוץ ערך המתח ברגל P1.0 הוא 0v (0 לוגי), אחרת המתח ברגל P1.0 הוא 3.3v (1 לוגי). כפי מתואר באיור מימין ישנה תופעת מעבר בין לחיצת הלחצן ושחרורו. נדרש להתעלם מתופעת מעבר זו, אחרת הבקר יכול לזהות אותן כלחיצות שלא לצורך.
במקרה שלנו נפתור זאת בתוכנה (ישנם גם פתרונות חומרה שונים). אנו נתעלם מתופעות מעבר אלו ע"י השהייה.

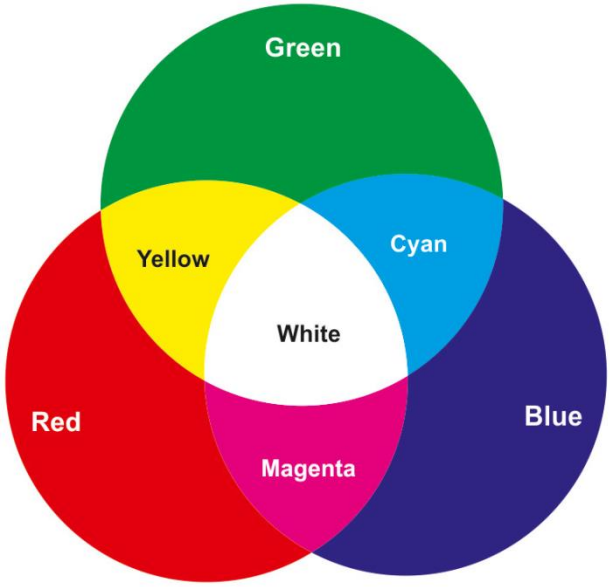
6) RGB Led:

RGB Led, הינו לד בעל 3 רגליים ולכל רגל שליטה על הארה בצבע בסיס שונה (Red, Green, Blue).

בצורה זו נוכל הלד RGB יכולה להאיר ב- 8 צבעים שונים (מצב שחור, כאשר הלד אינו מאיר כלל).







R	G	B	colo
0	0	0	blacl
0	0	1	blue
0	1	0	gree
0	1	1	cyar
1	0	0	red
1	0	1	magen
1	1	0	yellow
1	1	1	white

7) תרגיל 1:

מטרתנו לכתוב קוד כך שבלחיצה על לחצן P1.0 הבקר ידליק את 8 הלדים המחוברים ל- PORT9 **לד אחר לד** בצורה טורית סיבובית (מימין לשמאל) ללא הפסקה. בלחיצה על לחצן P1.1 הדלקת הלדים תיפסק.

```
#include <msp430xG46x.h>
;-------------------------------------------------------------------------------
        RSEG    CSTACK              ; Begins a relocatable segment name of CSTACK - Define stack segment
;-------------------------------------------------------------------------------
        RSEG    CODE                ; Assemble to Flash memory -like void main in C
;-------------------------------------------------------------------------------
RESET     mov.w   #0x03100,SP        ; Initialize stackpointer

StopWDT    mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT = #WDTPW+WDTHOLD -->M(&WDTCTL)  when
#WDTPW+WDTHOLD it is constant number as #0x49+0x57
                                ; The names WDTPW,WDTHOLD are defined in the attached header file.

SetupP9    bis.b   #0xff,&P9DIR         ; P9.0-P9.7 output = #0xff -->M(P9DIR)
SetupP1    bic.b   #0x03,&P1DIR         ; P1.0,P1.1 input

Check       bit.b   #0x01,&P1IN
            jnz     Check

Mainloop   mov.b   #0x01,R14
Subloop    mov.b   R14,&P9OUT
```

3 שורות אלו מיועדות ליצירת השהייה בין הדלקות הלדים.
השהייה זו נחוצה כדי שנוכל להבחין בעין אנושית בהבהוב הלדים.

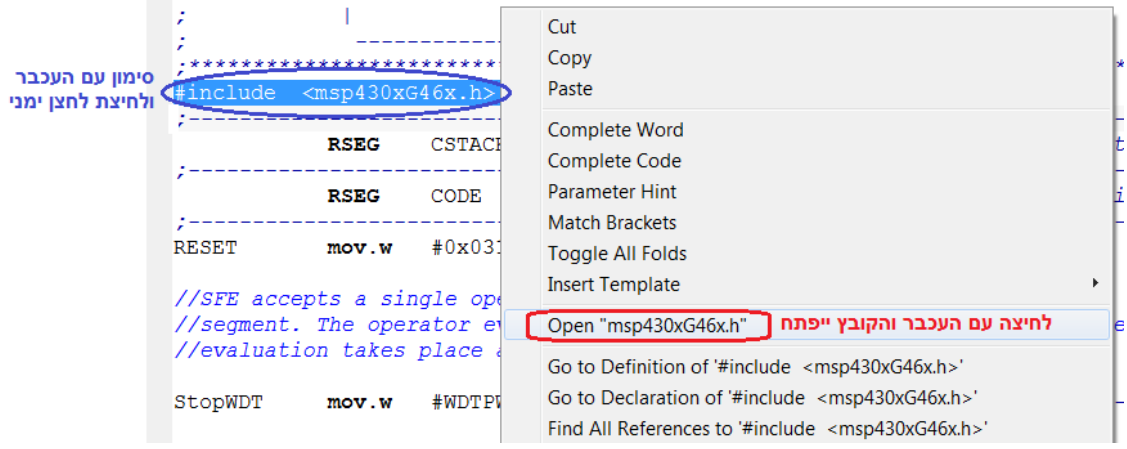
```
Wait      mov.w   #065000,R15          ; Delay to R15, #050000 --> R15
L         dec.w   R15                  ; Decrement R15
          jnz     L                    ; Delay over?, jump if not zero(if bit Z in SR isn't zero from the last command)

          bit.b   #0x02,&P1IN
          jz      Check
          rla.b   R14
          tst.b   R14
          jz      Mainloop
          jmp     Subloop
          nop                          ; NO MEANING, JUST TO AVOID FROM WARNING
;-------------------------------------------------------------------------------
        COMMON  INTVEC              ; Interrupt Vectors-Begins a common segment with name of INTVEC
;-------------------------------------------------------------------------------
        ORG     RESET_VECTOR        ; MSP430 RESET Vector
        DW      RESET
        END
```

הערות:

- הפקודות הבאות שקולות. $jmp\ \$ \stackrel{\text{def}}{=} L\ \ jmp\ \ L$
- השורה הראשונה בקוד - $\#include\ \ <msp430xG46x.h>$ מהווה הפנייה לקובץ כותר (header file) .

בקובץ זה מוגדרות כל התוויות ושמות הרגיסטרים הקשורים לחומרה. לפי התצלום הבא נוכל לראות את תוכן הקובץ.

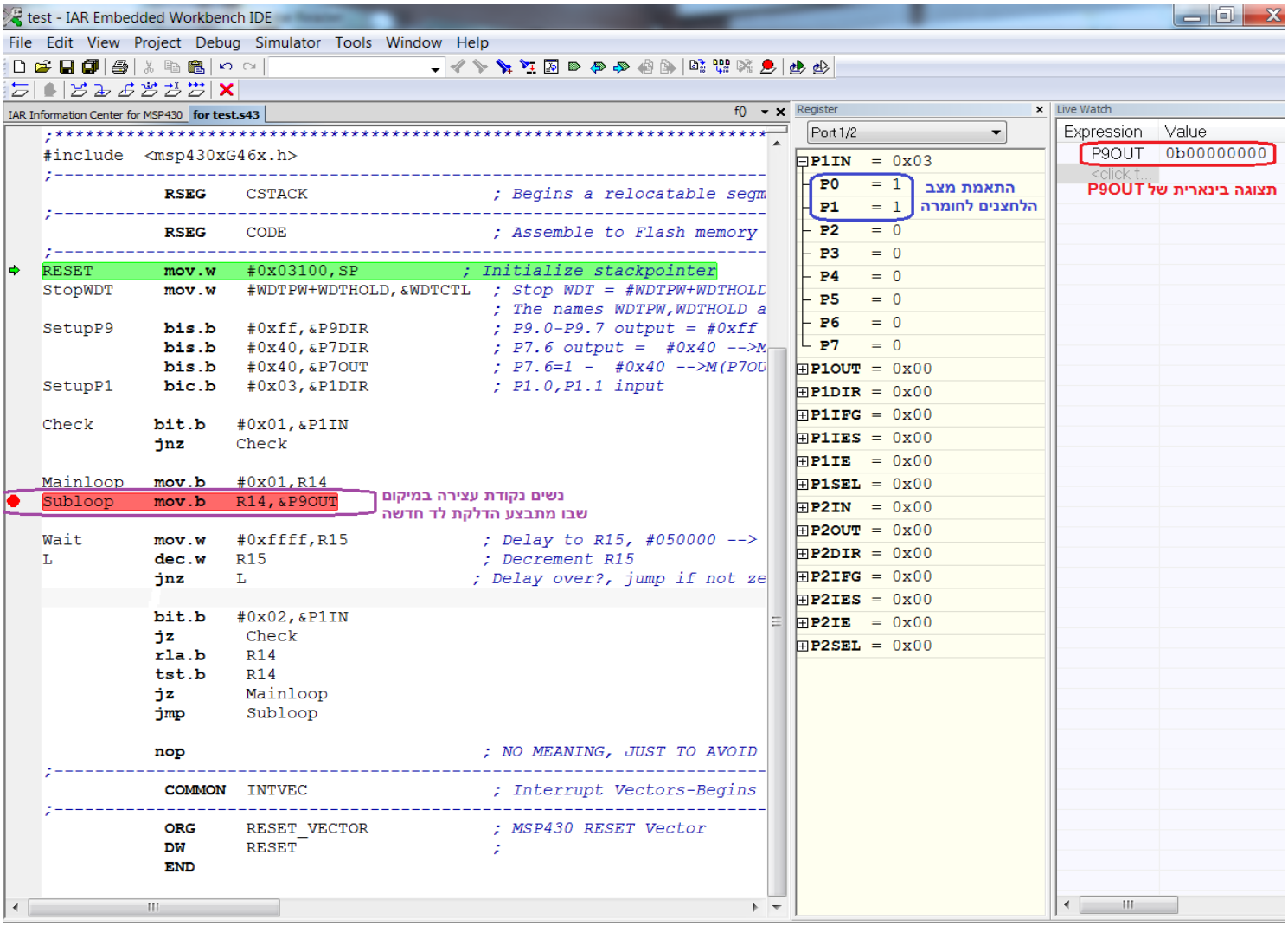


★ שימוש במיעון אבסולוטי (Absolute addressing mode) מיועד לפקודות הקשורות לחומרה שערך התווית שלה קבוע (אבסולוטי) לכל הבקרים של ה- MSP430 כך שאם נעביר קוד זה לבקר אחר עדיין תוויות אלו יתאימו. לעומת זאת שימוש במיעון סימבולי (Symbolic addressing mode) התוויות שאנו כותבים ערכם מתאים עבור סוג הבקר אליו אנו כותבים עכשיו.

★ בקובץ המפרט של הבקר שנמצא במעבדה - MSP430xG461x Data Sheet , אפשר לראות את ערכי הרגיסטרים לצורך קינפוגם (Register Configuration) עבור input ו- output.
עמוד 93 עבור קינפוג PORT9 (ניתן למצוא ע"י חיפוש למשל P9.1 בעזרת ctrl+F) ולדלג למיקום בו אתם מעוניינים. כך תעשו עבור שאר הפורטים.

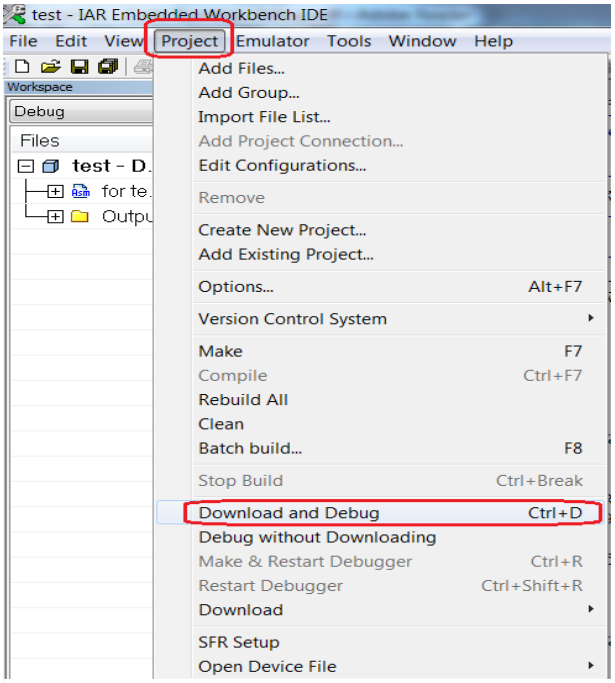
ביצוע סימולציה (מצב simulator):

1. במצב זה נוכל לפתוח את חלון Live Watch כדי לראות את שינוי ערך רגיסטר P9OUT במהלך ריצת התוכנית (הדלקת לד אחר לד) **View → Live Watch** . בנוסף נפתח את חלון הרגיסטרים במצב של PORT1 כדי לבצע הדמיית לחצנים (נאתחל ברגיסטר P1IN את הביטים P1.1 , P1.0 ל-'1' לוגי).
2. נשים נקודת עצירה במיקום שבו מתבצע הדלקת לד חדשה ובכל לחיצה על F5 נוכל לראות את ערך P9OUT נדלק ביט אחר ביט (יש לבחור תצוגה בינארית עבור P9OUT).



ביצוע טעינת קוד לבקר במצב DEBUG:

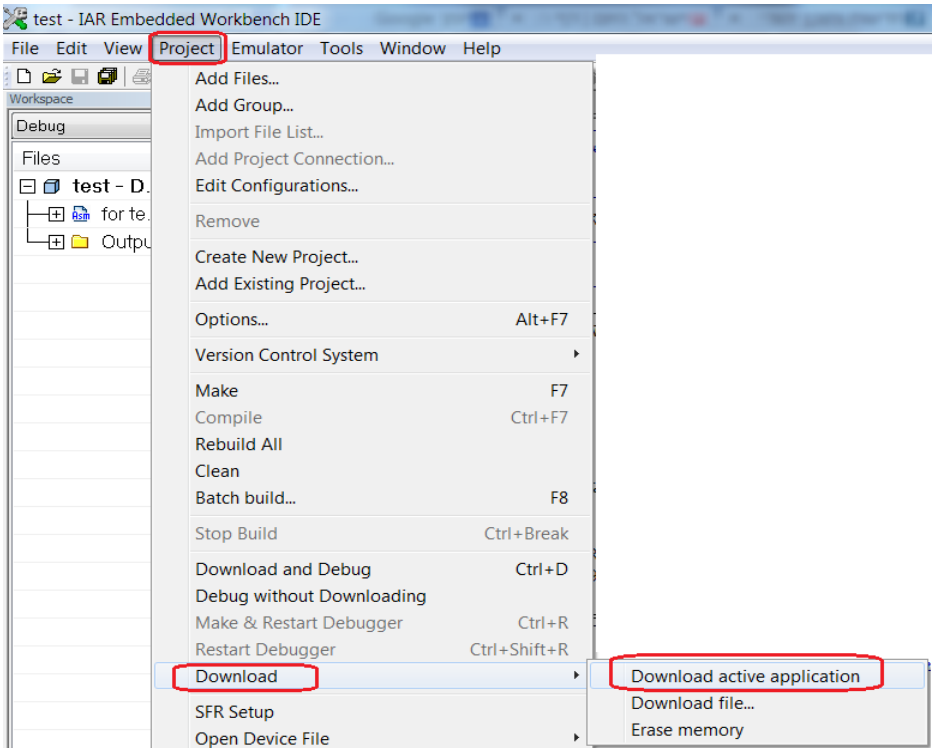
להזכירכם, במצב זה הקוד נטען לבקר אבל נשלט ע"י ה- PC (אין לגעת, בחיבורי החומרה בין הבקר ל-PC , הם כבר נעשו). לאחר שינוי ההגדרות (כמתואר ב- Tutorial 1) יש לבצע טעינת הקוד לבקר כמתואר בתצלום הבא.



לאחר סיום טעינת הקוד הבינארי לבקר. אפשר לבצע את כל אשר עשיתם בסימולציה (אולם הפעם הקוד רץ בבקר ולא במחשב) ולראות את רכיבי החומרה שהקוד שכתבתם מפעיל (במקרה זה, ללחוץ על לחצן P1.0 ולראות לדים דולקים- לד אחר לד ובלחיצה על לחצן P1.1 המצב מופסק).

ביצוע טעינת קוד לבקר במצב Active Application:

להזכירכם, במצב זה הקוד נטען לבקר **ולא נשלט** ע"י ה- PC (כמתואר ב- Tutorial 1). יש לבצע טעינת הקוד לבקר כמתואר בתצלום הבא.



לאחר סיום טעינת הקוד הבינארי לבקר , נדרש ללחוץ על לחצן RESET כדי שהבקר יתחיל לעבוד.

8) תרגיל 2:

מטרתנו לכתוב תוכנית בשפת אסמבלי שתסרוק **מחרוזת נתונה** ותמצא עבורה את **מספר ההופעות** של האות 'L' במחרוזת ותשמור את **הכתובת של כל הופעה** בזיכרון.

את מספר ההופעות נציג בהצגה בינארית על הלדים (PORT9).

המחרוזת שנרצה לבדוק - "HELLO WORLD, My name is MSP430!"

★ לפתרון הבעיה עלינו לחשוב על דרך לאחסן את המחרוזת, לדעת את גודלה ועל אלגוריתם למציאת מס' ההופעות ומיקומם עבור אות במחרוזת (במקרה הזה עבור האות 'L').

גודל המחרוזת 31 תווים והם מקודדים ע"י טבלת ASCII .

★ כדי להקצות מקום בזיכרון למחרוזת ולאתחל אותה בערכים אנו יכולים להשתמש בהוראה (Directive) DB.

הוראת DB (Define Byte) נועדה להקצות מקום של Byte עבור קבוע/קבועים.

אנו יכולים להוסיף תווית (Label) לציון מיקום תחילת המחרוזת בזיכרון.

כאשר האסמבלר רואה את הוראת DB ומיד אחריה סימון של מחרוזת הוא מקצה לכל תו תא בגודל Byte ושם בו ערך מתאים מטבלת ה-ASCII בסוף המחרוזת הוא תמיד מוסיף תו נוסף שערכו 0x00 לצורך סימון סוף מחרוזת. **לכן המחרוזת הנ"ל תופסת 32 בתים בזיכרון.**

★ לאחר שמיקמנו את המחרוזת בזיכרון עלינו לסרוק אותה תו אחר תו ע"י לולאה חיצונית ובתוך לולאה זו עלינו להשוות כל תו ל-'L'. **במידה והתו הנבדק שווה ל-'L'** נקדם רגיסטר המונה את מס' ההופעות של האות 'L' ונשמור בזיכרון את כתובת כל הופעה. **אחרת**, נמשיך בלולאה החיצונית.

כשאנו מגיעים לתו שערכו 0x00 זה סימן שהגענו לסוף המחרוזת, נוציא את ערכו של רגיסטר המונה ללדים.

★ **<u>הבדל בין מערך למחרוזת</u>**:

מערך ומחרוזת שניהם מבנה נתונים של איברים הסמוכים פיזית אחד לשני.

- ערך כל תא במחרוזת מכיל ערך ASCII של תו כלשהו (כך שגודל כל איבר הוא Byte) בעוד שתא במערך יכול להיות גדול מ-Byte (בכפולות של Byte).
- מחרוזת מכילה תו סיום, ערכו 0x00 ונמצא בסוף המחרוזת. כדי לעבוד עם מחרוזת יש לדעת את **<u>כתובת האיבר הראשון שלו בלבד</u>**. עבור מערך אין תו סיום וכדי לעבוד אתו צריכים לדעת את **<u>כתובת האיבר הראשון</u>** ואת **<u>גודל המערך</u>**.

קוד אסמבלי (תיאור מפורט בסעיף לאחר מכן)

מס' שורת קוד

```
1   ;*******************************************************************************
2   ;  Program : Counts the number of characters L in a string
3   ;  Input : The input string is the MyStr1/ MyStr2
4   ;  Output : The port one displays the number of E's in the string
5   ;  Written by : Hanan Ribo
6   ;  Date : 12/2013
7   ;  Description: MSP430 IAR EW; Demonstration of the MSP430 assembler
8   ;*******************************************************************************
9   #include "msp430xG46x.h"                          ; define controlled include file

10          ORG    1100h
11  MyStr1  DB   "HELLO WORLD, My name is MSP430!"  ; the string is placed onto RAM .the null character is
                                                     ;automatically added at the end of string (after the '!')
12  STRsize DC16   STRsize-MyStr1          ; size of string
13  Location EQU   01130h                  ; Results array storage

14          RSEG   CSTACK                  ; Define stack segment - pre-declaration of segment
15          RSEG   CODE                    ; place program in 'CODE' segment in to flash memory

16  MyStr2  DB     "HELLO WORLD, My name is MSP430!"

17  Main:   MOV    #SFE(CSTACK), SP        ;set up stack, Synonym Command - mov #3100h,SP
18          MOV    #WDTPW+WDTHOLD,&WDTCTL  ;Stop watchdog timer

19  SetP9   BIS.B  #0xff,&P9DIR            ; P9.0-P9.7 output = #0xff -->M(P9DIR)
20          BIS.B  #0x40,&P7DIR            ; P7.6 output =  #0x40 -->M(P7DIR)
21          BIS.B  #0x40,&P7OUT            ; P7.6=1 -  #0x40 -->M(P7OUT)

22          MOV    #MyStr2, R4             ;load the starting address of the string into the register R4
23          CLR.B  R5                      ;register R5 will serve as a counter
24  gnext:  MOV.B  @R4+, R6                ;get a new character
25          CMP    #0,R6                   ;Is this the string's final
26          JEQ    Lend                    ;go to the end
27          CMP.B  #'L',R6
28          JNE    gnext
29          MOV    R4,Location(R5)         ;addresses of 'L' appearances
30          INCD   R5                      ;increment counter
31          JMP    gnext
32  Lend:   RRA    R5                      ; Total 'L' appearances
33          MOV.B  R5,&P9OUT
34          BIS.W  #LPM4,SR                ; LPM4
35          NOP                            ; Required only for debugger
36  ;-------------------------------------------------------------------------------
37          COMMON INTVEC                  ; Interrupt Vectors
38  ;-------------------------------------------------------------------------------
39          ORG    RESET_VECTOR            ; POR, ext. Reset
40          DW     Main
41          END
```

תיאור מפורט - קוד אסמבלי:

a. 8 שורות ראשונות מכילות הערות. סימון הערה נעשה ע"י (**;**) וכל מה שנכתב מימין ל- **;** ועד סוף אותה שורה נצבע בכחול ונחשב הערה.

b. שורה 9, ההוראה **#include <msp430xG46x.h>** אומרת לאסמבלר מהיכן לקחת את כל ההגדרות של כתובות הרגיסטרים והביטים של רגיסטרי-הבקרה.

c. שורה 10, ההוראה **ORG 1100h** אומרת לאסמבלר למקם את הקצאות/השמות זיכרון הבאות (הרשומות עד שורה 14) החל מכתובת 1100h (שזה למעשה מיקום תחילת ה-RAM בבקר שלנו).

d. שורה 11, אתחול מחרוזת בזיכרון ה-RAM החל מכתובת 1100h עד כתובת 111Fh (32 בתים כולל תו סיום). כל איבר במחרוזת תופס בית אחד (לכן ההוראה **DB**).
התווית (label) MyStr1 היא למעשה כתובת האיבר הראשון של המחרוזת (1100h).

e. שורה 12, הצהרה על משתנה בגודל של 16 bit בכתובת STRsize. תוכן המשתנה שווה לגודל המחרוזת (מס' בתים) משום שיש חיסור בין כתובת סיום לכתובת התחלה של המחרוזת.

f. שורה 13, אתחול label בערך מספרי בעזרת הוראת **EQU**. במקרה שלנו תווית Location שווה לערך 1130h.

g. שורה 14, אנו מצהירים על הקצאת מקום בזיכרון עבור ה-stack (מחסנית). מיקום זה מפורט בקובץ ה-Linker ואנו מצהירים על צורך שיוקצה לתוכנית שלנו.

h. שורה 15, אנו מצהירים על צורך בהקצאת מקום בזיכרון ה- FLASH לצורך CODE של התוכנית (אפשר להקצות מקום בזיכרון עבור משתנים **<u>קבועים</u>** באזור זה- כמפורט לעיל).

i. שורה 16, אתחול מחרוזת קבועה (כתובת איבר הראשון ממוקם בכתובת MyStr2).

j. שורה 17, תחילת התוכנית הראשית, מתחילה בכתובת **Main:** והפקודה הראשונה היא לאתחל את ערך SP (כתובת תחילת מיקום המחסנית). במקרה זה יכולנו לרשום את הפקודה השקולה **MOV #3100h,SP** בתוכנית שלנו אין שימוש במחסנית, כך שיכולנו להשמיט שורה זו.

k. שורה 18, הפקודה **MOV** #WDTPW+WDTHOLD,&WDTCTL עושה השמה לביטי בקרה ספציפיים עבור רגיסטר בקרה WDTCTL השייך למודול watchdog timer כדי לא-לאפשר אותו.
(תפקידו של ה-watchdog timer הוא לאפשר בקשת פסיקה באופן מחזורי כדי לאפשר למשתמש בדיקה האם אין בעיות בהרצת הקוד ואם כן לבצע RESET באופן יזום).
במצב של RESET ה-watchdog timer מאופשר באופן אוטומטי ולכן עלינו לבטלו (מאחר ואין לנו צורך בו בתוכנית זו).

l. שורות 19-21, יש קינפוג של פורט 9 לצורך הדלקת הלדים.

m. שורה 22, העברת כתובת האיבר הראשון של מחרוזת MyStr2 לתוך רגיסטר R4.

n. שורה 23, איפוס רגיסטר R5 הנועד להיות מונה של מספר ההופעות של 'L' המחרוזת.

o. שורה 24, מיועדת למעבר על המחרוזת כאשר בביצוע הפקודה יש קריאת תו ולאחר מכן קידום רגיסטר R4 ב-1 (למיקום התו הבא במחרוזת), זו תחילת הלולאה הראשית.

p. שורה 25, בדיקה האם הגענו לסוף המחרוזת (תו סיום שערכו 0x00).

q. שורה 26, בדיקת תנאי. אם הגענו לסוף המחרוזת? יש לקפוץ לתווית **Lend** (לסיום). אחרת יש לגשת לבדיקת התו הבא במחרוזת (האם הוא שווה ל-'L'?).

r. שורה 27, האם התו הנוכחי (המוצבע ע"י R4 – תוכן R4 מכיל את כתבתו) שווה לתו 'L' ?

s. שורה 28, כאשר תנאי קודם לא מתקיים יש להמשיך לתו הבא (ע"י קפיצה לתווית **gnext**) וכאשר מתקיים התנאי הקודם יש להמשיך ל-2 הפקודות הבאות (שורות 29,30) לצורך שמירת כתובת של המופע של 'L' במחרוזת ולקידום המונה R5 (קידום ב-2 עבור כל מופע של 'L' ולבסוף בשורה 32 נעשה חלוקה ב-2).

t. שורה 31, המשך לולאה ראשית לצורך בדיקת התו הבא במחרוזת.

u. שורות 32-33, כשמסיימים את סריקת המחרוזת מגיעים לשורה 32 לצורך חלוקה ב-2 לרגיסטר R5 לצורך קבלת מס' הופעות של 'L' במחרוזת. בשורה 33 נוציא ערך זה ללדים בפורט 9.

v. שורה 34, יש פקודה הגורמת לבקר להיכנס למצב מנוחה של LPM4. בשורה זו הגענו לסוף התוכנית.

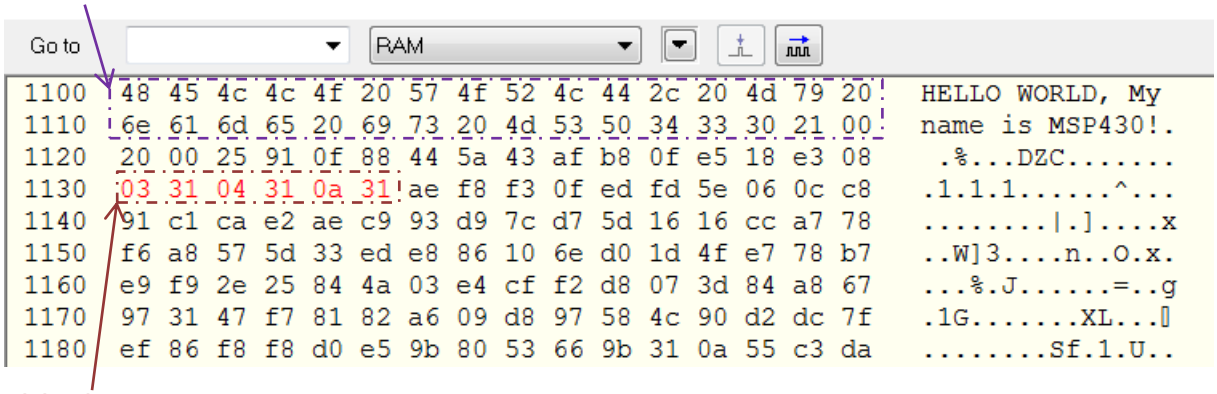
w. שורה 37, הצהרה על שימוש בווקטורי-הפסיקות.

x. שורה 39-40, הצהרה על שימוש בווקטור פסיקה RESET ואתחולו בכתובת של תחילת התוכנית שלנו בכתובת **Main**. בכל פעולת RESET הערך שייטען לתוך רגיסטר PC תהיה הכתובת **Main**.

★ הרצת התוכנית בסימולטור:

a) נסתכל על מצב זיכרון ה-RAM בסוף ריצת התוכנית.

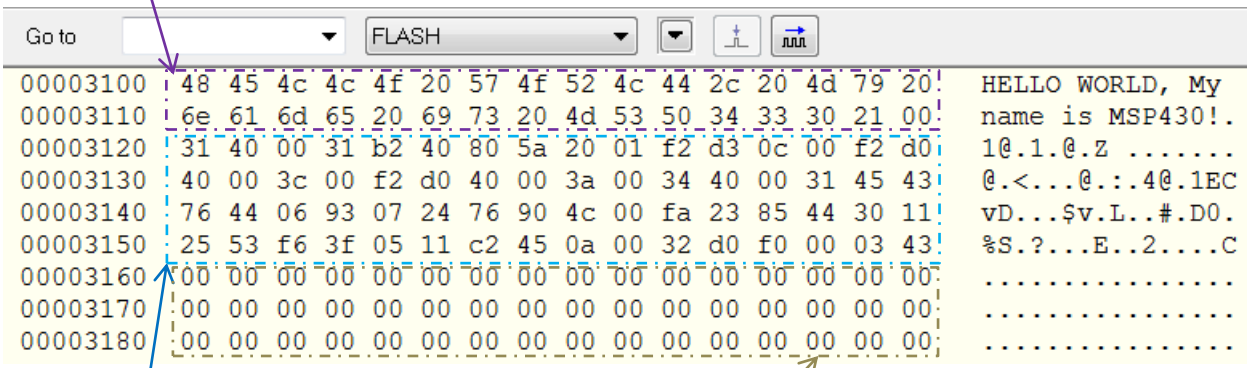
מיקום מחרוזת MyStr1 (כולל תו סיום), גודלה 32 בתים בין כתובות 1100h-111Fh



מיקום כתובות המופעים של תו 'L' במחרוזת. סה"כ ישנם 3 מופעים.

b) נסתכל על זיכרון ה-FLASH בסוף ריצת התוכנית.

מיקום מחרוזת MyStr2 (כולל תו סיום) ב- FLASH, גודלה 32 בתים בין כתובות 3100h-311Fh



מיקום התוכנית הראשית ב- FLASH, גודלה 63 בתים בין כתובות 3120h-315Fh.
רק פקודות אסמבלי (ולא הוראות-Directives) מומרות לקוד מכונה וממוקמות בזיכרון זה.

שאר זיכרון ה- FLASH, ערך כל Byte **במציאות** שווה 0xFF (בשונה ממה שמראה הסימולטור). לאזור זה לא נוכל לכתוב בתוך התוכנית באופן רגיל אלא ע"י שימוש ב-"מתווך" הנקרא FLASH controller.