

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



קובץ הכנה ניסוי מעבדה מס' 1 – חלק 2

**Tutorial 1.2 – IAR IDE, Code Running, Debug**

מעבדת מיקרומחשבים – המחלקה להנדסת חשמל ומחשבים

מס' קורס - 361.1.3353

כתיבה ועריכה: חנן ריבוא

מהדורה 1 – שנה"ל תשע"ו

## A. הקדמה:

בתרגול זה נתמקד בהרצת קוד אסמבלי בסביבת הפיתוח (IDE= Integrated Development Environment) הנקראת IAR. באופן כללי סביבת הפיתוח משמשת לפיתוח קוד מכונה (בינארי) מתוך טקסט (קוד) הנרשם ב- Editor של סביבת הפיתוח שאותו יש לצרוב לתוך זיכרון הבקר לצורך שליטה עליו. נבצע בהמשך סימולציית הרצת קוד שכתבנו בפתרון תרגול כיתה 1.

בפיתוח קוד בסביבת הפיתוח ישנם 3 שלבים:

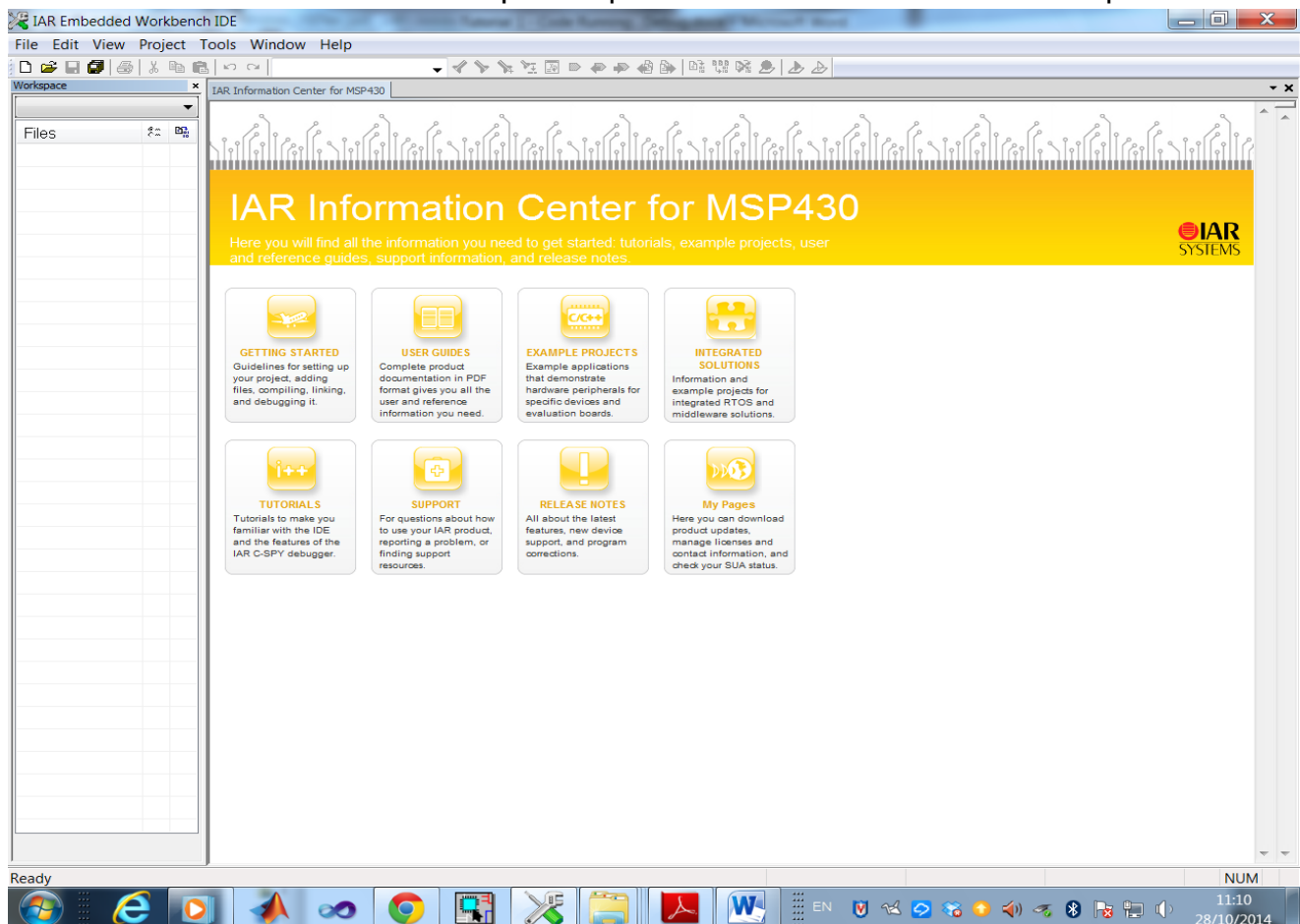
1. **סימולציה** – סביבת הפיתוח משמשת סימולטור לבקר שלנו. את הקוד שכתבנו נפעיל ב**מצב סימולטור** והוא ירוץ בפועל ב-PC (מחשב אישי) בלבד, לצורך דימוי הבקר.
2. **Debug** – הקוד שכתבנו ייצרב לבקר (מה-PC) ובהפעלתו הוא ירוץ בבקר ולא ב-PC, אולם ישנה תקשורת בין הבקר ל-PC לצורך תמיכה ב-DEBUG (נקודות עצירה, ריצה בצעדים, בדיקת ערכי רגיסטרים, ערכים בזיכרון וכו').
3. **Active Application** – הקוד שכתבנו ייצרב לבקר (מה-PC) ובהפעלתו הוא ירוץ בבקר בלבד ללא קשר עם ה-PC (בשונה ממצב DEBU). מצב זה מונה גם Stand Alone, מאחר ובמצב זה הבקר בפני עצמו ללא קשר ל-PC.

**B.** תחילת עבודה בסביבת הפיתוח דורשת **פתיחת פרויקט**. להלן שלבי פתיחת פרויקט של סביבת IAR IDE.

1. עבודה בסביבת הפיתוח במצב סימולטור אפשרית גם בביתכם. אפשר להוריד את קובץ ההפעלה המצורף.

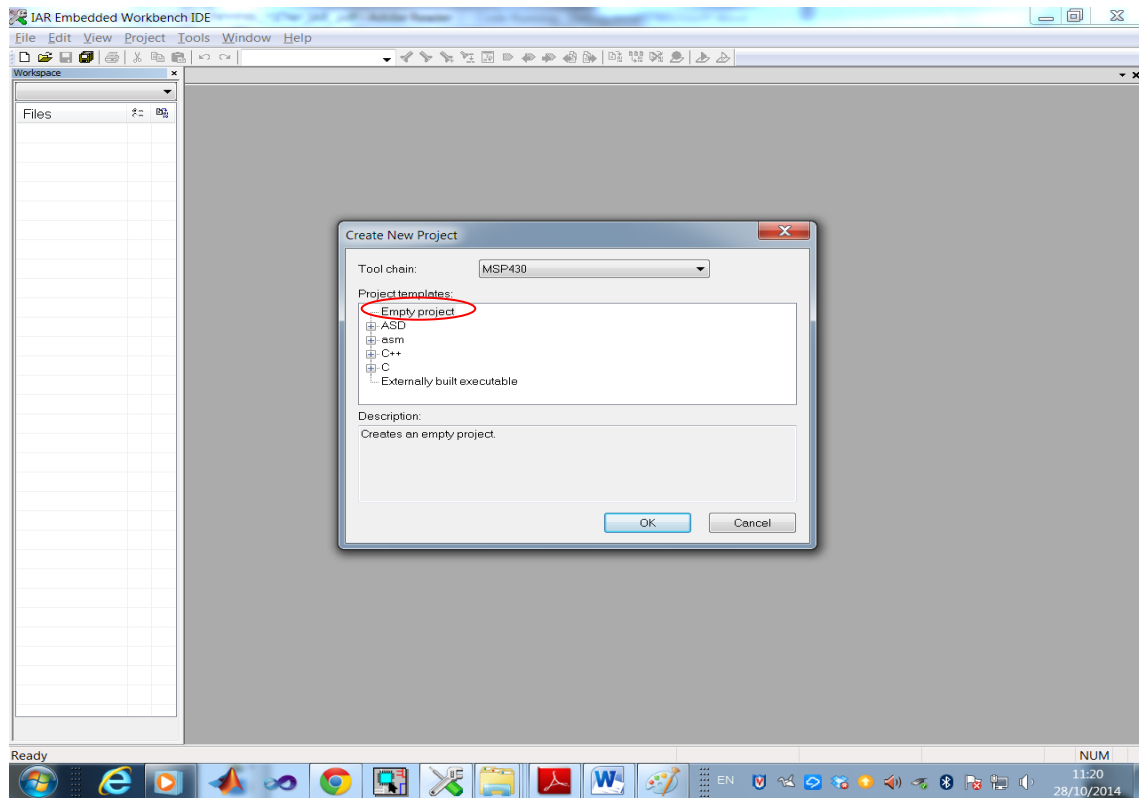
IAR - IDE setup.zip

2. לאחר ההתקנה נפתח את תוכנת סביבת הפיתוח ומתקבל החלון הבא:

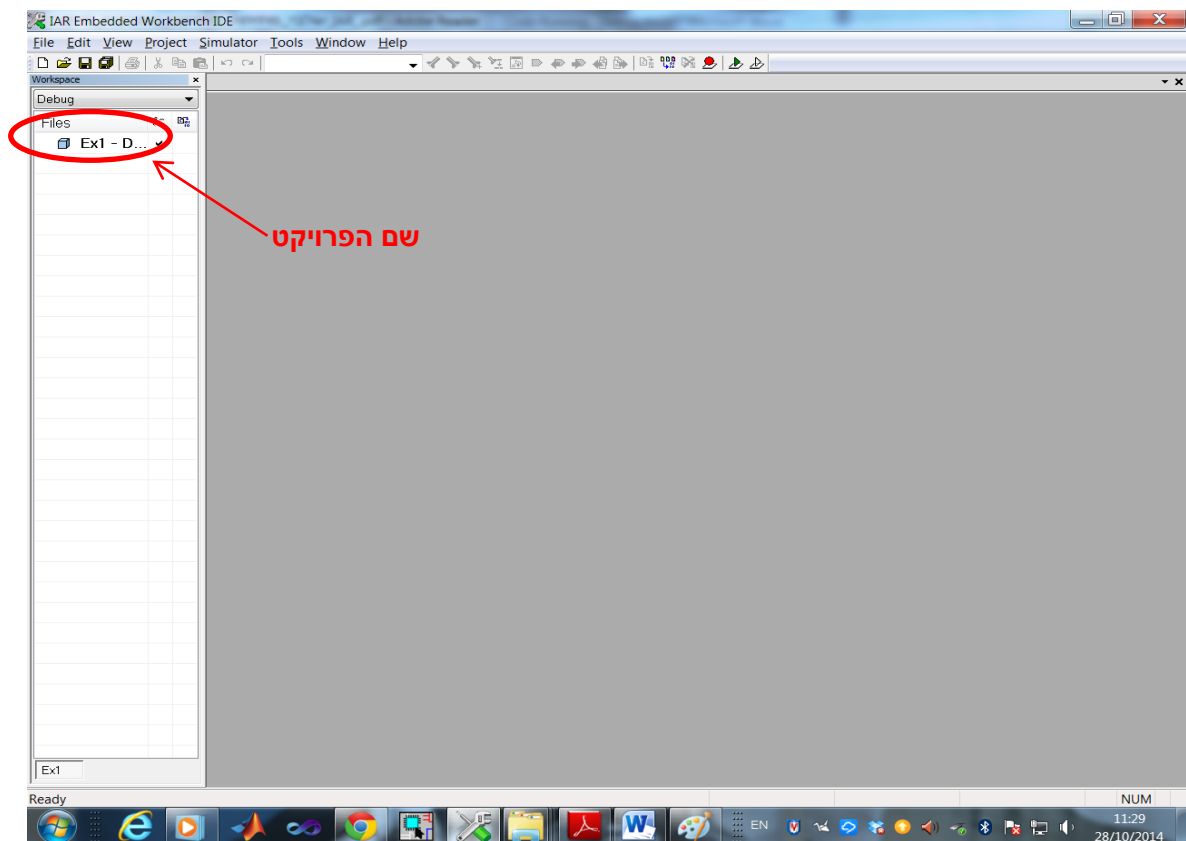


3. לאחר מכן, **רקע החלון נעשה אפור** → File → New → Workspace

4. לאחר מכן, ייפתח החלון הבא → Project → Create New Project



לחצו OK ושמו את מיקום הפרויקט , במקום בו תבחרו במחשב. לאחר המירה ייפתח החלון הבא (שם הפרויקט שבוחרתם יופיע צד שמאל למעלה).

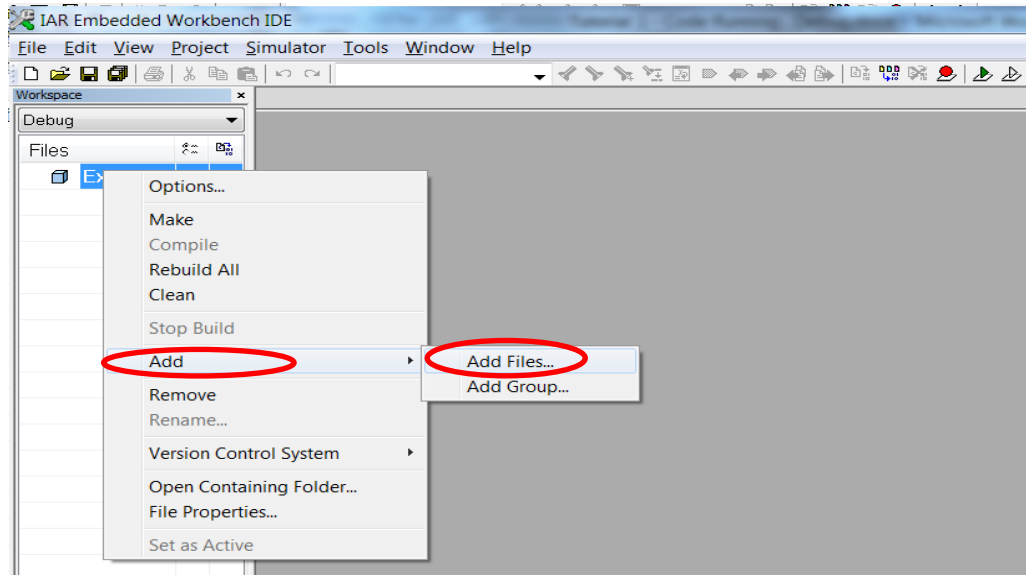


5. כדי לצרף קובץ מקור (קובץ קוד) ישנן 2 אפשרויות:

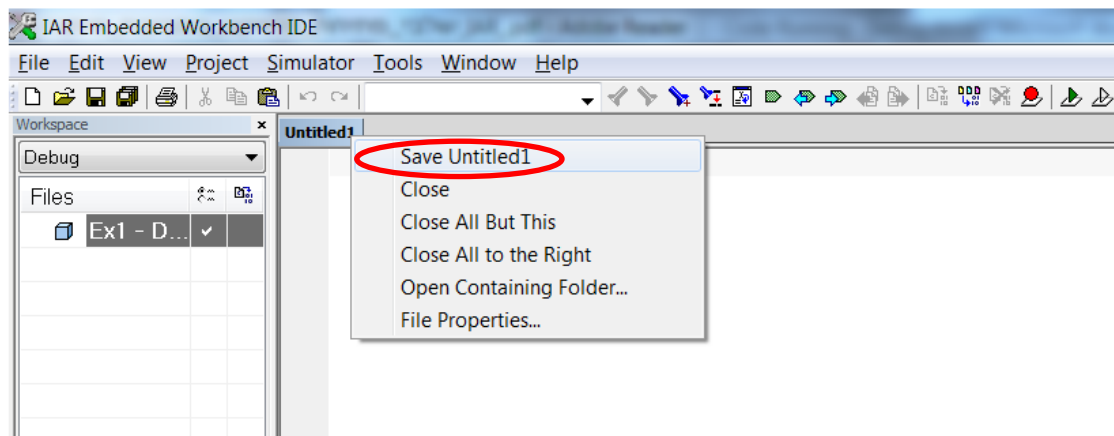
★ **אפשרות 1:** נכין קובץ txt ולתת סיומת s43. (שם הקובץ לפי בחירתנו) לצרף באופן עצמאי לתיקיית הפרויקט ששמרנו בשלב קודם. קובץ עם סיומת s43. מהווה קובץ קוד בשפת אסמבלי, קובץ עם סיומת c. מהווה קובץ קוד בשפת C. בקורס זה נכתוב תוכניות בשפת אסמבלי בלבד.

צירוף קובץ קוד לפרויקט - סדר פעולות:

שמירת קובץ קוד בתיקיית הפרויקט לא מצרפת אותו באופן אוטומטי לפרויקט, לכן נבצע את הפעולה הבאה. לחיצת ימנית בעכבר על שם הפרויקט - Add -> Add Files -> נפתח חלון ובו יש לבחור את קבצי הקוד s43. אותם נרצה לצרף לפרויקט, בסיום שמות קבצים אלו יופיעו תחת שם הפרויקט.



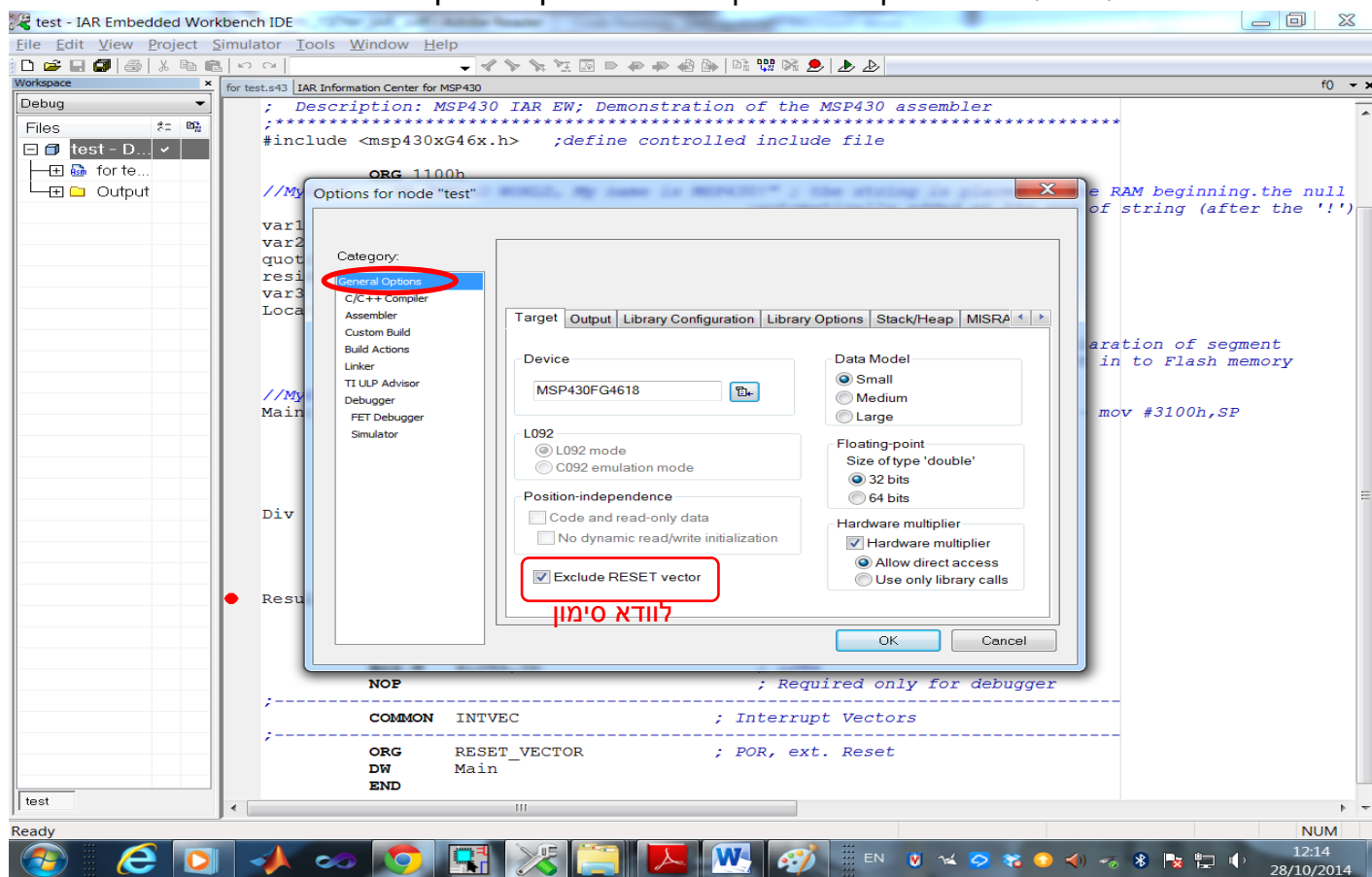
★ **אפשרות 2:** בלחיצה על הלשונית בצורת דף חלק לבן (צד שמאל למעלה), ייפתח דף חלק לבן ובו נכתוב את קוד התכנית שלנו. לחיצת ימנית בעכבר על שם הקובץ, יש אופציית שמירה.



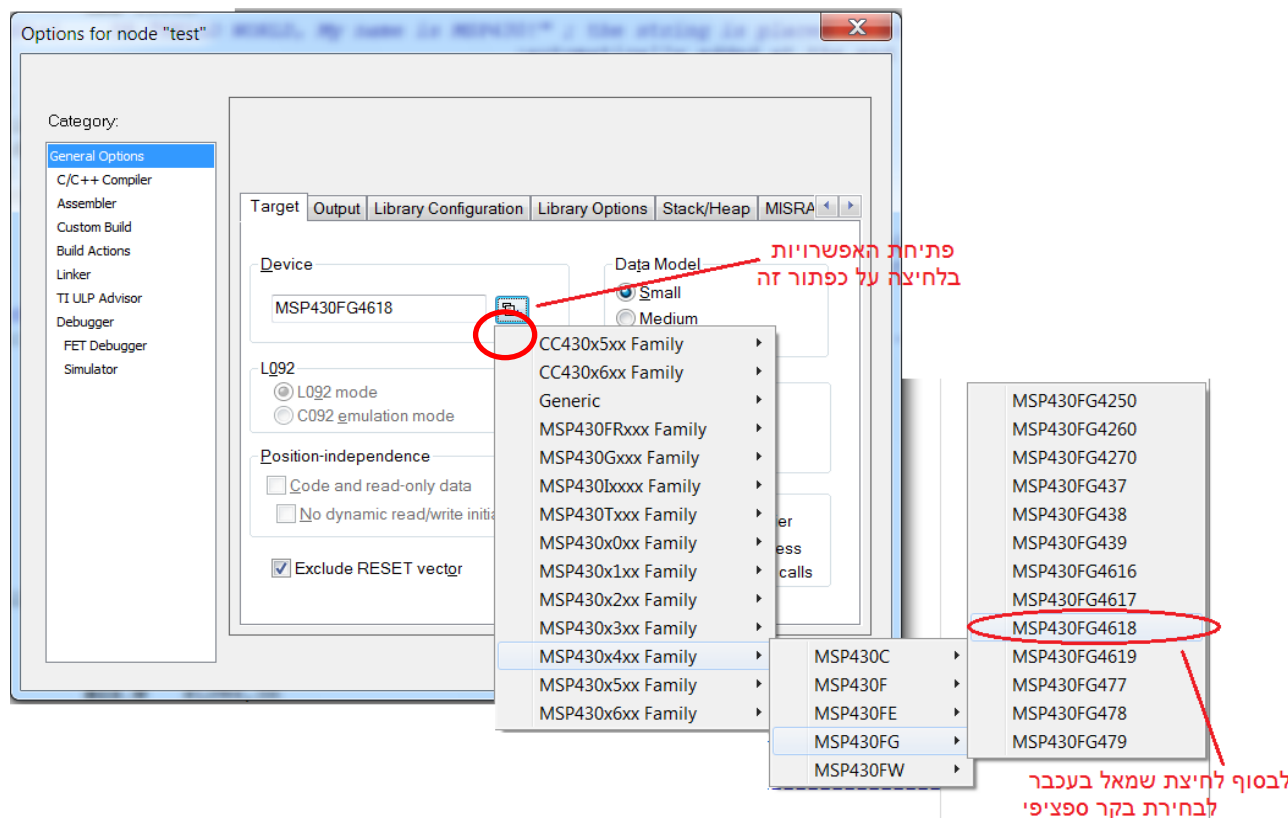
נשמור את הקובץ בתיקיית הפרויקט וניתן לו שם עם סיומת s43. את ההמשך נבצע כמתואר באפשרות 1.

6. בחירת פלטפורמה חומרתית של הבקר בסביבת הפיתוח:

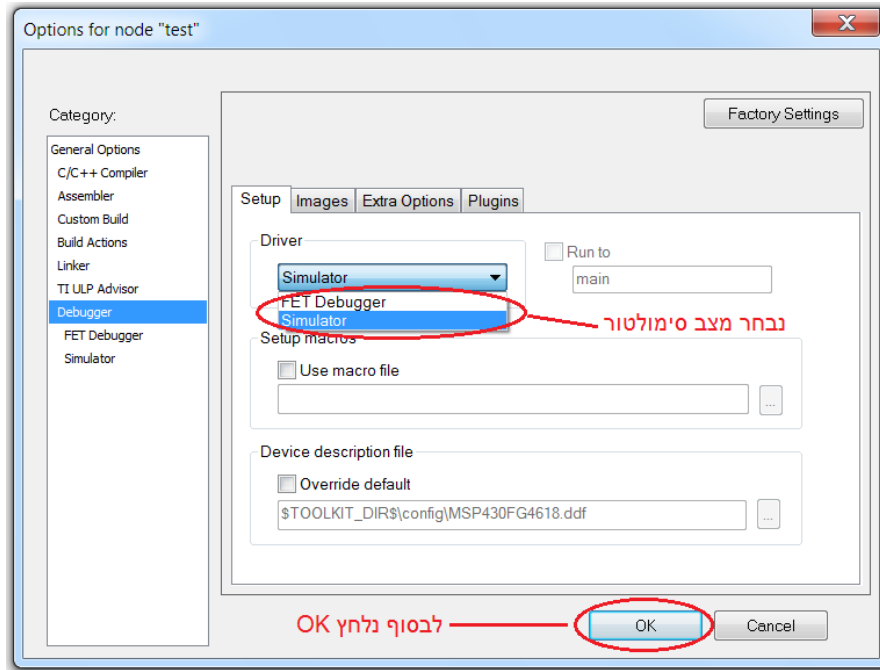
לחיצת ימנית בעכבר על שם הפרויקט - Options - נפתח חלון ובו נסמן את הנדרש.



נעדכן בחלון הנ"ל את שדה Device המתאים לבקר שלנו, בלחיצה על הכפתור מימין לשדה זה.



7. מצב סימולטור – בחירת סביבת הפיתוח לעבודה במצב סימולטור יש לפעול כך:



בשלב זה שאר ההגדרות יישארו לפי הגדרות ברירת המחדל של היצרן.

8. שמירת ה- Workspace :

נפתח חלון לשמירה, יש לשמור בתיקיית הפרויקט → **File → Save Workspace**

9. הידור הפרויקט:

**Project → Rebuild All**

(אם שכחתם לבצע שלב קודם, תתבקשו תחילה לשמור את ה- Workspace ורק אח"כ הוא יבצע את בניית הפרויקט).

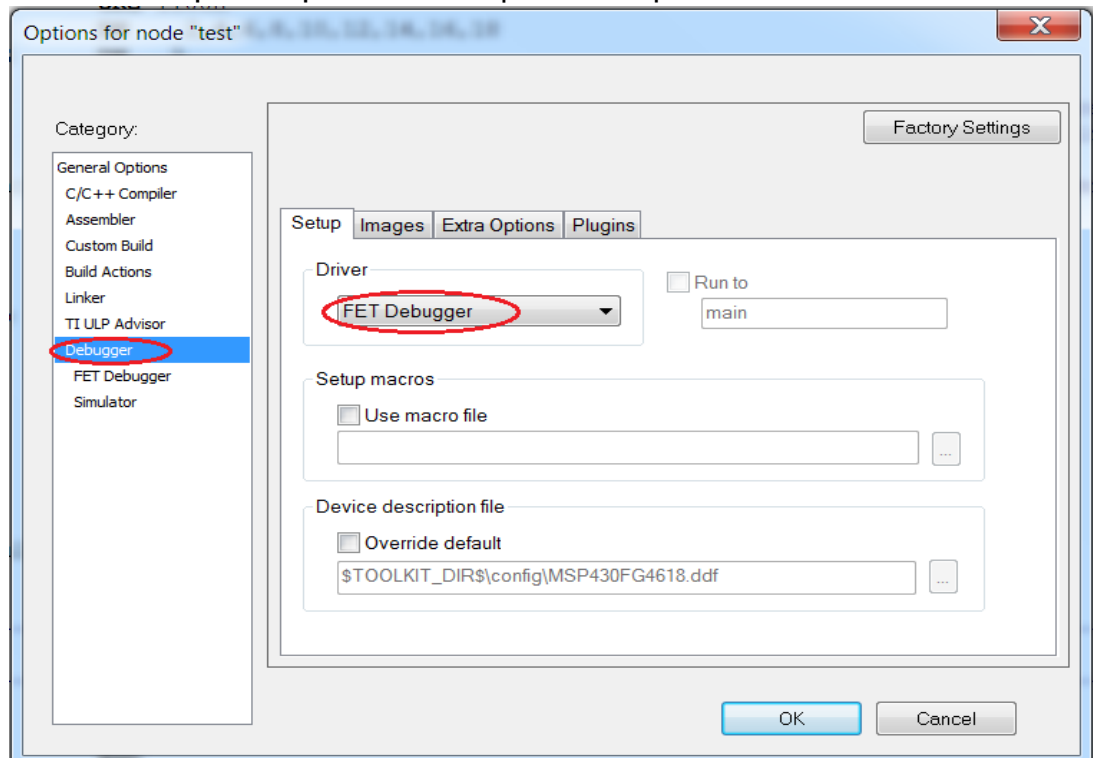
10. לתחילת ביצוע הסימולציה.

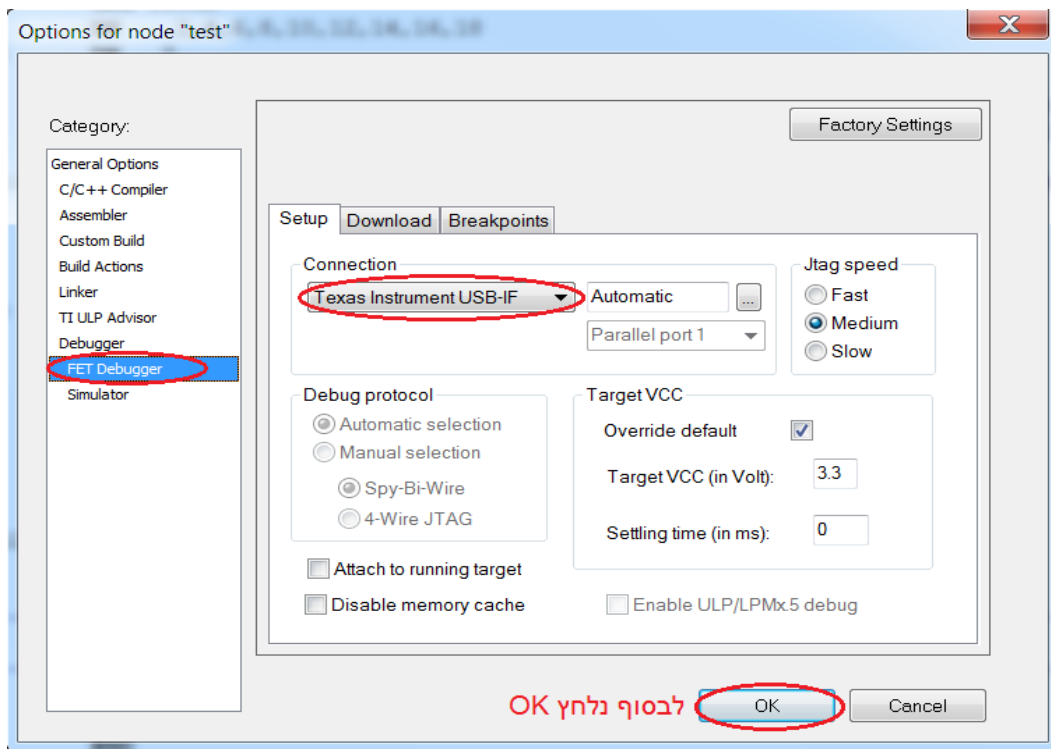
כנסתם למצב סימולציה (כמו שלמדתם בקורס JAVA) → **Project → Download and Debug**

**תוכלו לדלג בשלב זה על סעיפים 11,12 ולעבור לכותרת סימולציית תרגילים – תרגול כיתה מס' 1**

11. מצב DEBUG - בחירת סביבת הפיתוח לעבודה במצב DEBUG יש לפעול כך:

לחיצת ימנית בעכבר על שם הפרויקט -> Options -> נפתח חלון ובו נסמן את הנדרש.





נבנה את הפרויקט: Project → Rebuild All

נכנס למצב DEBUG (צריבת קוד לזיכרון הבקר והרצתו בבקר בלבד עם יכולות DEBUG דרך ה-PC):

**נכנסתם למצב DEBUG** (כמו שלמדתם בקורס JAVA) → **Project → Download and Debug**

אפשר להריץ את הקוד כמו במצב סימולציה כמפורט לעיל עם הבדל אחד, מדובר במצב אמת (קוד רץ בבקר ולא ב-PC).

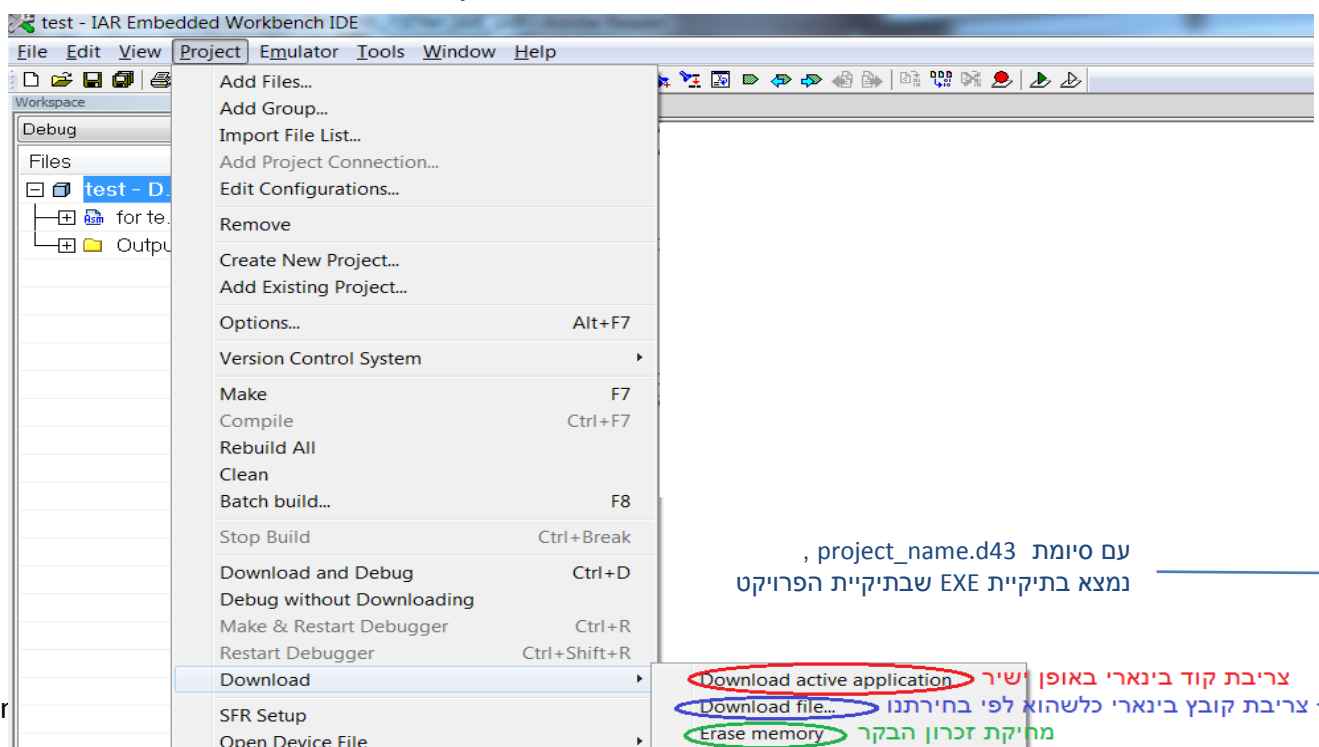
## 12. מצב Active Application –

במצב זה נרצה לצרוב את הקובץ הבינארי של התכנית לזיכרון הבקר ומאז הבקר עובד במנותק מה-PC. מצב זה הוא התוצר הסופי של האפליקציה אותה בצענו. כאשר נרצה לבצע פעולת RESET לבקר, נלחץ על לחצן RESET הנמצא על כרטיס ערכת הפיתוח.

נבנה את הפרויקט: Project → Rebuild All

נבצע צריבת קובץ בינארי לזיכרון הבקר:

שלוש אפשרויות → Download → Project



## סימולציית תרגילים – תרגול כיתה מס' 1

### 1. חילוק בין מספרים שלמים:

שארית =  $residue$ , מנה =  $quotient$   $\rightarrow var1 \div var2$  (ללא שימוש במחלק חומרתי)

```

*****
#include <msp430xG46x.h> ;define controlled include file
*****

ORG 1100h
var1 DW 17
var2 DW 4
quotient DS16 1
residue DS16 1

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

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

      CLR R9 ;R9 = Quotient Result
      MOV var1,R4
      MOV var2,R5
Div    CMP R5,R4
      JLO Result
      INC R9
      SUB R5,R4
      JMP Div
Result MOV R9,quotient
      MOV R4,residue

      BIS.W #LPM4,SR ; LPM4
      NOP ; Required only for debugger

;-----
COMMON INTVEC ; Interrupt Vectors
;-----

ORG RESET_VECTOR ; POR, ext. Reset
DW Main
END
  
```

הגדרת משתנים ב-RAM

קטע קוד רלוונטי לתרגיל

כניסה למצב sleep

באופן כללי מבלי להיכנס לפרטים (תלמדו בהרצאה) הבקר מריץ פקודה אחר פקודה באופן טורי (כתובת אחר כתובת החל מנקודת התחלה מסוימת) מתוך הזיכרון. כאשר הגענו לסיום התכנית נרצה למנוע מהבקר להגיע לאזור בזיכרון שאינו מכיל פקודות ולהימנע מפעולות שאינן רצויות. יש 2 אפשרויות:

★ נדרש לוודא שהתוכנית כוללת לולאה אינסופית.

★ שימוש בפקודת LPM (תלמדו עליה בהמשך) אשר משמשת לרמות sleep שונות של הבקר.

נבצע הרצה לקוד ונסתכל על ערכי הרגיסטרים, משתנים והזיכרון. בכניסה למצב סימולציה המסך נראה כך:שורת הפקודה הצבועה בירוק היא נקודת ההתחלה אליה נגיע במצב של RESET.



```

*****
***** IAR Information Center for MSP430 *****
*****
#include <msp430xG46x.h> ;define controlled include file

var1      DW      17
var2      DW      4
quotient  DS16    1
residue   DS16    1

RSEG      CSTACK                ; Define stack segment - pre-declaration of segm
RSEG      CODE                  ; place program in 'CODE' segment in to Flash me

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

          CLR      R9              ;R9 = Quotient Result
          MOV      var1,R4
          MOV      var2,R5
          CMP      R5,R4
          JLO      Result
          INC      R9
          SUB      R5,R4
          JMP      Div
          MOV      R9,quotient
          MOV      R4,residue

          BIS.W    #LPM4,SR        ; LPM4
          NOP                    ; Required only for debugger

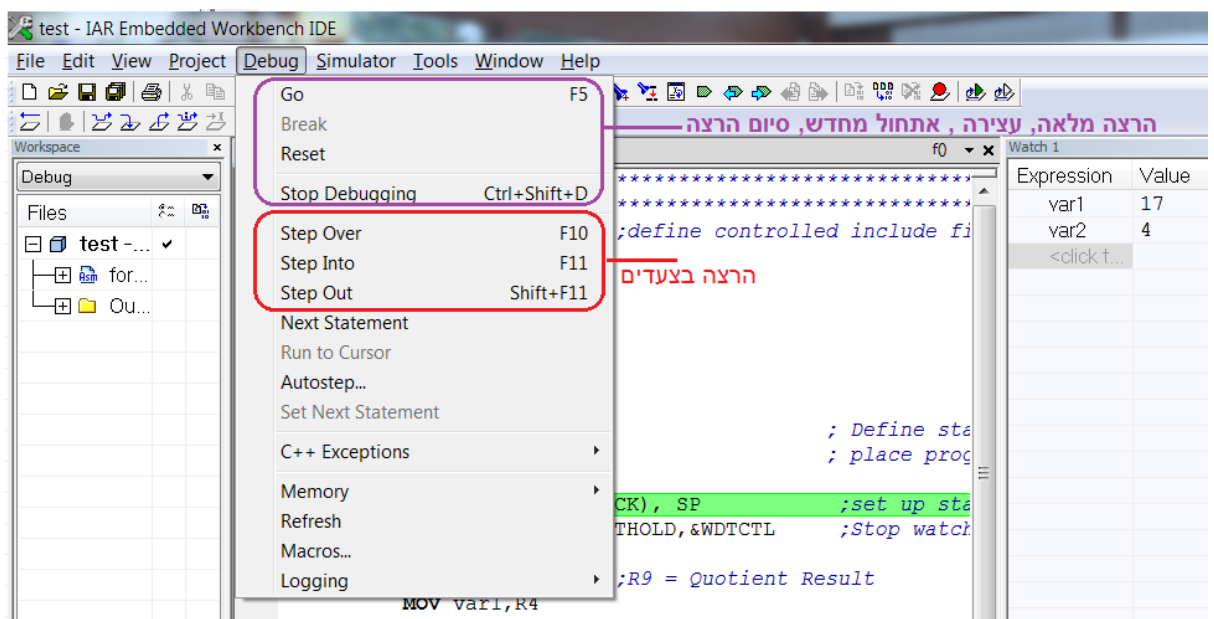
          -----
          COMMON   INTVEC          ; Interrupt Vectors
          -----

          ORG      RESET_VECTOR    ; POR, ext. Reset
          DW      Main
          END
  
```

נפתח את החלונות הבאים (רגיסטרים, משתנים וזיכרון) כדי לראות את ערכי התוכנית בכל שלב שנרצה.

- לפתיחת חלון Disassembly: בחלון זה נראה את מבנה הפקודות לפי כתובת כל פקודה בזיכרון, גודלה והקידוד שלה לקוד מכונה. View → Disassembly
- לפתיחת חלון רגיסטרים: View → Register
- לפתיחת חלון משתנים: View → Watch → Watch1 ÷ Watch4
- לפתיחת חלון הזיכרון: נבחר בלשונית לראות את זיכרון ה-RAM View → Memory → RAM

אנו מעוניינים כעת רק בחלונות רגיסטרים, משתנים וזיכרון. תוכלו בזמנכם החופשי לעבוד צעד אחר צעד.



## חלון Disassembly:

בחלון זה נוכל לראות עבור כל פקודה את כתובתה בזיכרון, גודלה ב-word וקידודה (קוד מכונה).

The screenshot displays the IAR Embedded Workbench IDE with the following components:

- Assembly Code (Left):**

```

#include <msp430xG46x.h> ;define controlled inc...

ORG 1100h
var1 DW 17
var2 DW 4
quotient DS16 1
residue DS16 1

RSEG CSTACK ; Des...
RSEG CODE ; pla...

Main: MOV #SFE(CSTACK), SP ;set...
      MOV.W #WDTPW+WDTHOLD,&WDCTL
      CLR R9 ;R9 = Quotient 1
      MOV var1,R4
      MOV var2,R5
Div:   CMP R5,R4
      JLO Result
      INC R9
      SUB R5,R4
      JMP Div
Result MOV R9,quotient
      MOV R4,residue

      BIS.W #LPM4,SR
      NOP

COMMON INTVEC ; Inte...
;-----
ORG RESET_VECTOR ; POR,
DW Main
END

```
- Disassembly (Right):**

Address	Hex	Assembly
0030F2	614E	addc.b SP,R14
0030F4	FE1D 0671	and.w 0x671(R...
0030F8	CEAD 4880	bic.w @R14,0x...
0030FC	F7C0 938D	and.b R7,0xC48B
Main:MOV #SFE(CSTACK), SP ;set ...		
003100	4031 3100	mov.w #0x3100,SP
MOV.W #WDTPW+WDTHOLD,&WDCTL ...		
003104	40B2 5A80 0120	mov.w #0x5A80...
00310A	4309	clr.w R9 ;R9 = Quotient Result
MOV var1,R4		
00310C	4014 DFF2	mov.w var1,R4
MOV var2,R5		
003110	4015 DFF0	mov.w var2,R5
Div: CMP R5,R4		
003114	9504	cmp.w R5,R4
JLO Result		
003116	2803	jnc Result
INC R9		
003118	5319	inc.w R9
SUB R5,R4		
00311A	8504	sub.w R5,R4
JMP Div		
00311C	3FFB	jmp Div
Result: MOV R9,quotient		
00311E	4980 DFE4	mov.w R9,quot...
MOV R4,residue		
003122	4480 DFE2	mov.w R4,residue
BIS.W #LPM4,SR ...		
003126	D032 00F0	bis.w #0xF0,SR ...
NOP		
00312A	4303	nop
00312C	0000	???

**Annotations:**

- Red:** Highlights the instruction `CLR R9` at address `00310A` in both views. A note in Hebrew states: "קידוד, הגודל, תיבה אחת כתובת" (Encoding, size, one box address).
- Purple:** Highlights the instruction `MOV var2,R5` at address `003110` in both views. A note in Hebrew states: "פקודות ליבה מתאימה במקרה של פקודת חישוב זו תהא פקודה שקולה" (Core instructions in the case of this calculation instruction will be equivalent instructions).

לבסוף, נשים נקודת עצירה בסוף התוכנית (אחרי הפקודה האחרונה כדי שגם היא תתבצע) ובבצע ריצה מלאה (F5) כדי לראות הערכים בסיומה.

The screenshot displays the IAR Embedded Workbench IDE interface during a debug session. The main window shows assembly code for a program that calculates a quotient and residue. The Watch window shows the values of variables `var1` (17) and `var2` (4). The Register window shows the values of CPU registers, with `R9` (0x00004) and `CYCLECOUNTER` (50) highlighted. The Memory window shows the memory dump starting at address 1100.

**Assembly Code:**

```
#include <msp430xG46x.h> ;define controlled include

ORG 1100h
var1 DW 17
var2 DW 4
quotient DS16 1
residue DS16 1

RSEG CSTACK ; Define
RSEG CODE ; place

Main: MOV #SFE(CSTACK), SP ;set up
      MOV.W #WDTPW+WDTHOLD,&WDTCTL ;Stop watch

      CLR R9 ;R9 = Quotient Result
      MOV var1,R4
      MOV var2,R5

Div    CMP R5,R4
      JLO Result
      INC R9
      SUB R5,R4
      JMP Div

Result MOV R9,quotient
      MOV R4,residue

      BIS.W #LPM4,SR ; LPM4
      NOP ; Request

COMMON INTVFC : Interrupt
```

**Watch Window:**

Expression	Value	Location
var1	17	Memory:
var2	4	Memory:

**Register Window:**

Register	Value
PC	= 0x03126
SP	= 0x03100
SR	= 0x0004
R4	= 0x00001
R5	= 0x00004
R6	= 0xF6B08
R7	= 0x9A083
R8	= 0xB3425
R9	= 0x00004
R10	= 0x0F240
R11	= 0x77EB9
R12	= 0x8E01A
R13	= 0x90D48
R14	= 0x360CC
R15	= 0x67C6C
CYCLECOUNTER	= 50
CCTIMER1	= 50
CCTIMER2	= 50
CCSTEP	= 50

**Memory Window:**

Address	Value
1100	11 00 04 00 01 00 be 1b 96 bf 42 8f fd 7a
1110	8c e3 90 a4 37 74 ec 88 b7 b8 fd 31 bf 59 c7 5a
1120	83 b8 de dc 0c d0 5a 14 08 09 67 10 55 6d 38 83
1130	b5 dd f5 c7 de b0 80 eb 60 cc 92 91 0d 45 a1 2c
1140	5f 71 03 d1 8d 0d b9 2e 3d 61 ed 5e 8e fa 7a 26
1150	73 f2 0b e3 80 c4 2c a1 39 4e e6 24 73 58 2d 6b
1160	e5 55 b5 76 f6 ae 17 3a 59 d5 b6 27 14 65 62 af
1170	7f 10 24 1d 56 2a 25 ad de 07 37 d0 5b 79 d8 6f

2. כפל בין מספרים שלמים:  $var1 \cdot var2 \rightarrow var3$  (ללא מכפל חומרתי)  
לאחר ביצוע השלבים המפורטים בתרגיל 1. לאחר הרצה מתקבל,

The screenshot shows the IAR Embedded Workbench IDE interface. The main window displays assembly code for an MSP430 microcontroller. The code includes variable declarations and a multiplication routine. The Watch window shows the values of variables: var1 = 17, var2 = 4, and var3 = 68. The CPU Registers window shows the state of the processor registers, with R4 and R5 highlighted. The memory window at the bottom shows the contents of RAM, with the value 44000000 highlighted.

**Assembly Code:**

```
#include <msp430xG46x.h> ;define controlled include

ORG 1100h
var1 DW 17
var2 DW 4
var3 DS32 1
Location EQU var3+2

RSEG CSTACK ; Define
RSEG CODE ; place in code

Main: MOV #SFE(CSTACK), SP ;set up
      MOV.W #WDTPW+WDTHOLD,&WDTCTL ;Stop watchdog timer

      CLR R5
      MOV var1,R4
      MOV var2,R7
      DEC R7
      JZ Result
      MUL Result
      ADD var1,R4
      ADC R5
      JMP Mul
      MOV R4,var3
      MOV R5,Location

      BIS.W #LPM4,SR ; LPM4

      COMMON INTVEC ; Interrupt vector
```

**Watch Window:**

Expression	Value	Location
var1	17	Memory:
var2	4	Memory:
var3	68	Memory:

**CPU Registers:**

Register	Value
PC	= 0x03128
SP	= 0x03100
SR	= 0x0003
R4	= 0x00044
R5	= 0x00000
R6	= 0x76703
R7	= 0x00000
R8	= 0x1849A
R9	= 0x4C01E
R10	= 0x11D77
R11	= 0x52BCB
R12	= 0xDD9D0
R13	= 0xAD928
R14	= 0x31B10
R15	= 0x9F4AA
CYCLECOUNTER	= 49
CCTIMER1	= 49
CCTIMER2	= 49
CCSTEP	= 49

**Memory Window:**

Address	Value
1100	11 00 04 00 44 00 00 00
1110	84 42 b3 be a4 be ad 03 c8 d5 bb 2b e7 08 82 1e
1120	08 bc 8d 39 5a 55 26 97 65 f1 15 9c 7f ca a9 35
1130	95 43 0f a2 d8 19 cb 4e 63 d7 7d 5c 72 eb 2a 21
1140	37 84 aa 16 f2 23 16 73 bc f2 b7 9e 48 e1 63 7d
1150	87 0e 42 e2 db eb 5e af 5c d1 17 53 56 cc 3d c2
1160	79 d5 f3 1e f8 da 30 14 74 b6 59 b7 04 80 79 dc
1170	ad 4c ee 33 2c 58 19 ab 49 a8 72 e8 24 1b 06 3a

משתנה התוצאה var3, גודלו 32bit מאחר ותוצאת כפל בין 2 מספרים בגודל 16bit יכולים לגלוש לתחום 32bit. רגיסטר R4 מהווה חלק תחתון של תוצאת המכפלה (Result LSB) ורגיסטר R5 מהווה חלק עליון של תוצאת המכפלה (Result MSB). הפלטפורמה שלנו הינה Little Endian ולכן תוכן R4 מועבר לזיכרון בכתובת var3 ורגיסטר R5 מועבר לזיכרון בכתובת var3+2.

3. חישוב ממוצע בין 2 אברים סמוכים במערך בכתובת **Arr** בגודל **SIZE** (לדרוס את מערך המקור ולאפס איבר אחרון).

לאחר ביצוע השלבים המפורטים בתרגיל 1 ולאחר מלאה הרצה מתקבל,

The screenshot shows the IAR Embedded Workbench IDE interface. The main window displays assembly code for an MSP430 microcontroller. The code includes a loop that calculates the average of two adjacent elements in an array. The array is defined with a size of 9. The loop starts at address 1100h and ends at 1170h. The code uses registers R4, R5, R6, R7, and R12. The final result is stored in R12. The registers window on the right shows the current values of the registers, with R12 containing 0x000000. The memory window at the bottom shows the memory dump for the program, with the array data starting at address 1100h.

**Assembly Code:**

```

;*****
#include <msp430xG46x.h> ;define controlled include fil
;*****

ORG 1100h

Arr DW 2,4,6,8,10,12,14,16,18
SIZE DW 9

RSEG CSTACK ; Define stack
RSEG CODE ; place program

Main: MOV #SFE(CSTACK), SP ;set up stack
      MOV.W #WDTPW+WDTHOLD,&WDTCTL ;Stop watchdog

      MOV SIZE,R4
      MOV #Arr,R5
      DEC R4
      JZ Result
      MOV @R5,R6
      MOV 2(R5),R7
      ADD R6,R7
      RRA R7
      MOV R7,0(R5)
      INCD R5
      JMP Loop
      Result MOV #0,0(R5)

      BIS.W #LPM4,SR ; LPM4

;*****
COMMON INTVEC ; Interrupt Vector
;*****

```

**Registers:**

Register	Value
PC	0x0312C
SP	0x03100
SR	0x0003
R4	0x00000
R5	0x01110
R6	0x00010
R7	0x00011
R8	0x38CC7
R9	0x5EDD9
R10	0x647DC
R11	0xDF6D0
R12	0xE8D33
R13	0x115D0
R14	0x82DEE
R15	0x4DE46
CYCLECOUNTER	145
CCTIMER1	145
CCTIMER2	145
CCSTEP	145

**Memory Dump:**

Address	Hex	ASCII
1100	03 00 05 00 07 00 09 00 0b 00 0d 00 0f 00 11 00	.....
1110	00 00 09 00 ac 68 bc 99 72 eb b4 b2 03 6b 3f 91	....h..r...k?.
1120	99 0f a6 d7 03 14 f9 b6 cf 94 d9 51 c7 6b c4 ac	.....Q.k..
1130	9f fa 6a 80 91 1a 61 ea e3 67 52 3f fa 4a e5 02	..j...a..gR?.J..
1140	02 69 19 4e 42 d0 b4 a8 ba 12 17 89 1a 29 88 9b	..i.NB.....)
1150	89 d4 c7 75 17 45 0e d2 8f 3f 2d 13 20 a6 90 fb	...u.E...?-. ...
1160	94 7b 41 89 79 4c 32 4e 15 a5 f8 22 5a f0 b7 ac	..{A.yL2N..."Z...
1170	61 6d d7 0c 8d 92 e1 12 4c 13 0e f5 b3 d6 d7 d5	am.....L.....

4. עליך לכתוב תוכנית לחישוב העקבה (TRACE) של מטריצה סימטרית.

העקבה של מטריצה סימטרית בגודל  $n \times n$  מוגדרת ע"י סכום איברי האלכסון של המטריצה, כדלקמן:

$$\text{Trace}(A) = a_{11} + a_{22} + \dots + a_{nn}$$

גודל כל איבר במטריצה הוא 32 סיביות הערך המתקבל עבור עקבת המטריצה המחושבת יש לאחסן ברגיסטרים R12 עד R15. כאשר החלק הנמוך LSB של העקבה יאוחסן ברגיסטר R12, וחלק הגבוה MSB של העקבה יאוחסן ברגיסטר R15. כאשר Matrix הינה מטריצה ותוויית זו היא כתובת האיבר הראשון של המטריצה ו- Lines מסמן את מספר השורות של המטריצה.

את תוצאת העקבה יש לאחסן במשתנה בשם Trace.

מטריצה מאוחסנת בזיכרון ע"י מערך של שורות (שורה אחר שורה החל משורה ראשונה).

- Part of Diagonal index = MatrixPTR+4\*i+Lines\*4j
- $0 \leq i \leq \text{Lines}-1$  and  $0 \leq j \leq \text{Lines}-1$  for all Parts of Diagonal  $i=j$  So, Part of Diagonal index = MatrixPTR+4i(Lines+1)
- Trace=sum of M[MatrixPTR+4i(Lines+1)] when  $0 \leq i \leq \text{Lines}-1$

#include <msp430xG46x.h> ;define controlled include file

ORG 1100h

```
Matrix DC32 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
Lines DW 4
Trace DS32 2
```

```
RSEG CSTACK ; Define stack segment - pre-declaration of segment
RSEG CODE ; place program in 'CODE' segment in to Flash memory
```

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

```
mov #Matrix,R5
clr R12
clr R13
clr R14
clr R15
mov Lines,R6
mov Lines,R7
inc R6
rla R6
rla R6
L1 add @R5,R12
adc R13
adc R14
adc R15
add R6,R5
dec R7
jnz L1
mov #Trace,R5
mov R12,0(R5)
mov R13,2(R5)
mov R14,4(R5)
mov R15,6(R5)
```

```
BIS.W #LPM4,SR ; LPM4
```

```
;-----
```

```
COMMON INTVEC ; Interrupt Vectors
```

```
;-----
```

```
ORG RESET_VECTOR ; POR, ext. Reset
DW Main
END
```

לאחר ביצוע השלבים המפורטים בתרגיל 1 ולאחר מלאה הרצה מתקבל,

The screenshot displays the IAR Embedded Workbench IDE interface. The main window shows assembly code for an MSP430 microcontroller. The code includes a header file, defines constants, and sets up a main function. The assembly code is as follows:

```
#include <msp430xG46x.h> ;define controlled include

ORG 1100h
Matrix DC32 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
Lines DW 4
Trace DS32 2

RSEG CSTACK ; Define stack
RSEG CODE ; place program code

Main: MOV #SFE(CSTACK), SP ;set up stack pointer
      MOV.W #WDTPW+WDTHOLD,&WDTCTL ;Stop watchdog timer

      mov #Matrix,R5
      clr R12
      clr R13
      clr R14
      clr R15
      mov Lines,R6
      mov Lines,R7
      inc R6
      rla R6
      rla R6
      L1: add @R5,R12
          adc R13
          adc R14
          adc R15
          add R6,R5
          dec R7
          jnz L1
```

The CPU Registers window on the right shows the current state of the registers. The PC register is 0x03146, SP is 0x03100, and SR is 0x0003. The R4 register is 0x0F726, R5 is 0x01142, R6 is 0x00014, R7 is 0x00000, R8 is 0x94B89, R9 is 0x0F426, R10 is 0x82D72, R11 is 0x5A60F, R12 is 0x00022, R13 is 0x00000, R14 is 0x00000, and R15 is 0x00000. The CYCLECOUNTER register is highlighted with a green box and shows a value of 71. The CCTIMER1, CCTIMER2, and CCSTEP registers also show a value of 71.

The Watch window on the right shows the values of the Matrix, Lines, and Trace variables. Matrix is 1, Lines is 4, and Trace is 34. The Location column shows the memory addresses for these variables.

The Memory window at the bottom shows the memory dump. The address 1100h is highlighted, and the data is shown in hexadecimal and ASCII. The data at 1100h is 01 00 00 00 02 00 00 00 03 00 00 00 04 00 00 00, which corresponds to the first row of the Matrix variable.

משתנה זה מונה את מספר מחזורי השעון של ה-CPU ומתעדכן במהלך ביצוע כל פקודה. מספר מחזורי שעון בסיום ההרצה הוא 71.

Trace