בס"ד

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

קובץ הכנה ניסוי מעבדה מס' 2

Tutorial 2.1 – Assembly Part 2

Macro, Stack, Routines

מעבדת מיקרומחשבים – המחלקה להנדסת חשמל ומחשבים

מס' קורס - 361.1.3353

כתיבה ועריכה: חנן ריבוא

מהדורה 1 – שנה"ל תשע"ו

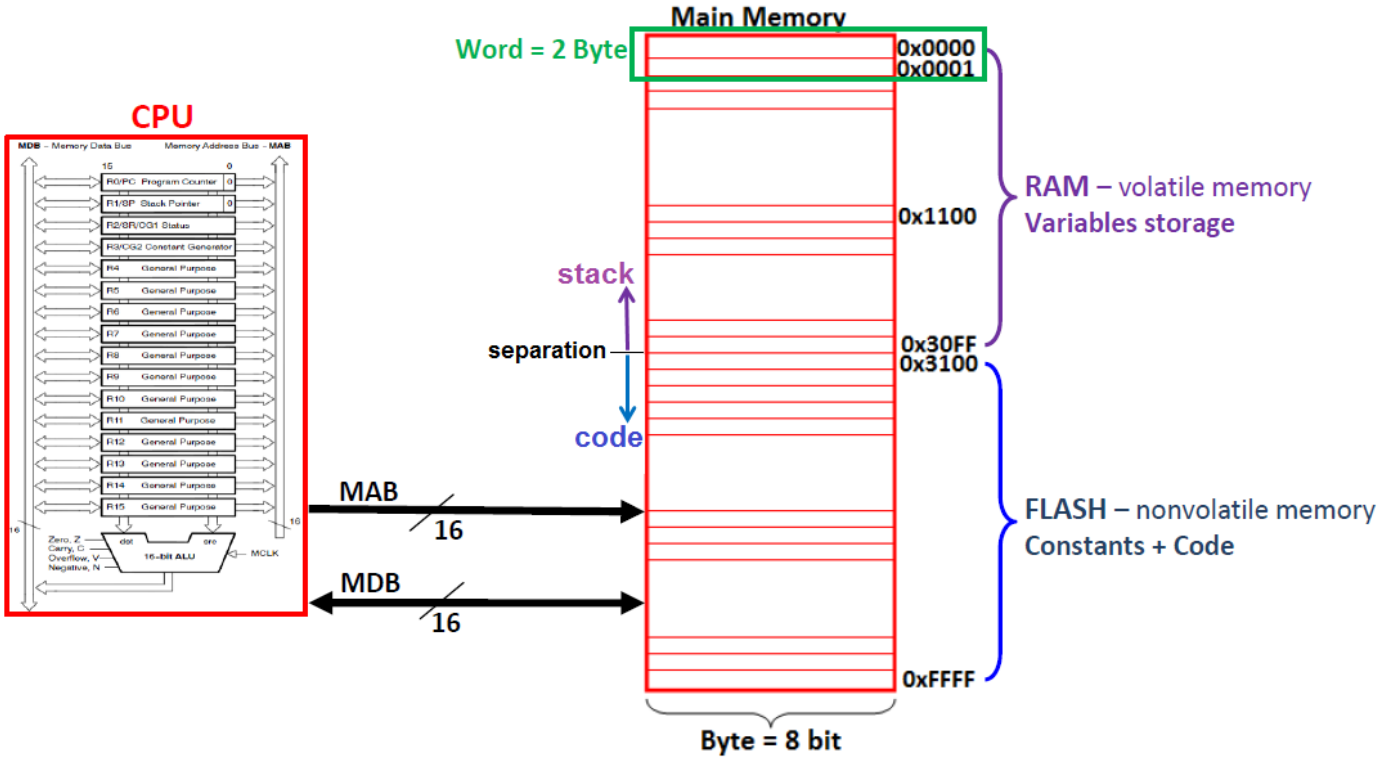
בתרגול הכנה זה, נלמד ונתרגל את הנושאים הבאים:

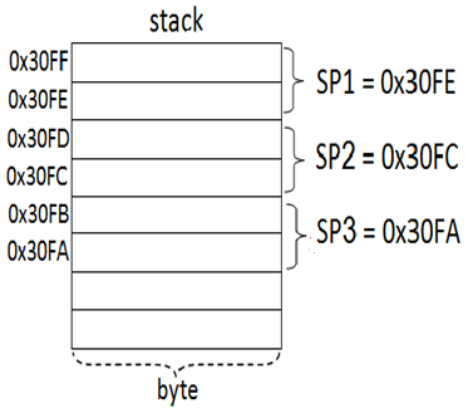
מחסנית (Stack), פונקציות MACRO, רוטינות (כולל העברת פרמטרים לרוטינה).

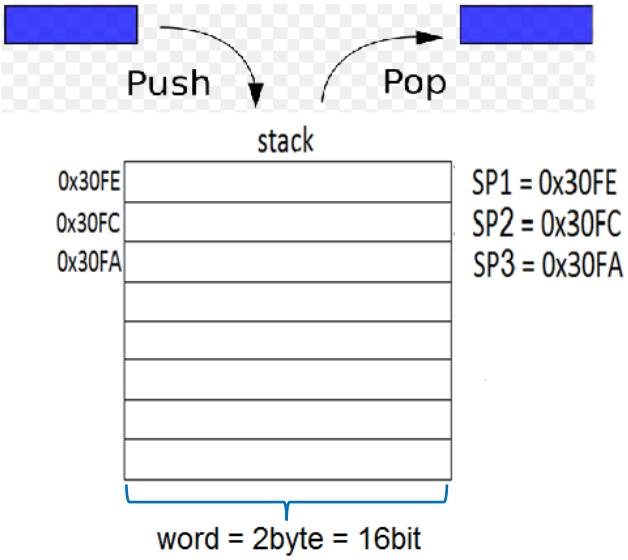
A. מחסנית (Stack) – ספר מעבדה MSP430x4xx user guide עמוד 45

קביעת התחלת אזור המחסנית במרחב זיכרון RAM. $MOV\ \#0x03100, SP$ $\quad SP\ register \equiv TOS$

קביעת התחלת אזור קטע CODE במרחב זיכרון FLASH. $ORG\ 0x03100$







LIFO policy = **L**ast **I**n **F**irst **O**ut

- המחסנית הינה אזור במרחב זיכרון ה-RAM (הגדרה כרצוננו) שבו התכנית משתמשת **לצרכים הבאים:**

1. **שמירת כתובת חזרה** מרוטינה ופסיקה (בנוסף לכתובת חזרה יש גם שמירה של ערך רגיסטר SR – **נושא זה תלמדו בהמשך הקורס**).
2. **לשימוש תכנותי כלשהוא** (אגירת מידע לצורך חישוב בנוסף לרגיסטרים של ה-CPU).

- ישנו רגיסטר ב-CPU הנקרא SP (Stack Pointer) המשמש מצביע (**מכיל כתובת**) לראש המחסנית ונקרא לעיתים TOS(**T**op **O**f **S**tack) . **בתחילה נאתחל את ערכו לכתובת תחילת המחסנית** ובכל פעולה נוספת הדורשת מחסנית ערכו משתנה בהתאם.
- קידום / חיסור, רגיסטר SP נעשה בצורה אוטומטית ע"י ה-CPU עבור פקודה המשתמשת במחסנית.
- למחסנית (פקודות – call,ret,reti,push,pop,עקב פסיקה).

★ **פקודת push** – כתיבת מילה **למחסנית**.

PUSH (.B) src → SP - 2 → SP
src → @SP

פירוט הפקודה בהרחבה עמוד 95

★ **פקודת pop** – קריאת מילה **מהמחסנית**.

POP (.B) dst → @SP -> temp
SP + 2 -> SP
temp -> dst

פירוט הפקודה בהרחבה עמוד 94

★ **פקודת call** - פקודה לביצוע רוטינה (כמעין פונקציה ללא העברת משתנים וערך החזרה – פירוט בסעיף 2).

CALL dst → SP - 2 → SP, PC+2 → @SP
dst → PC

פירוט הפקודה בהרחבה עמוד 69

★ **פקודת ret** – פקודת חזרה מרוטינה ללא ערך החזרה.

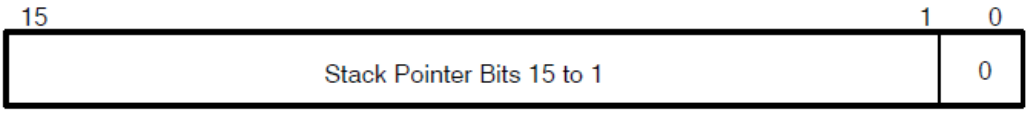
RET → @SP→ PC
SP + 2 → SP

פירוט הפקודה בהרחבה עמוד 96

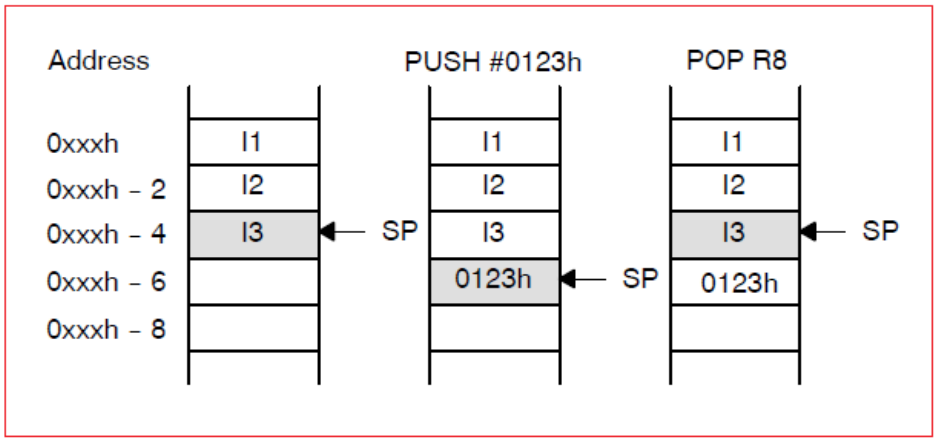
★ **פקודת reti** - פקודת חזרה מפסיקה (**נלמד במעבדה 4**).

RETI →
TOS → SR
SP + 2 → SP
TOS → PC
SP + 2 → SP

פירוט הפקודה בהרחבה עמוד 97



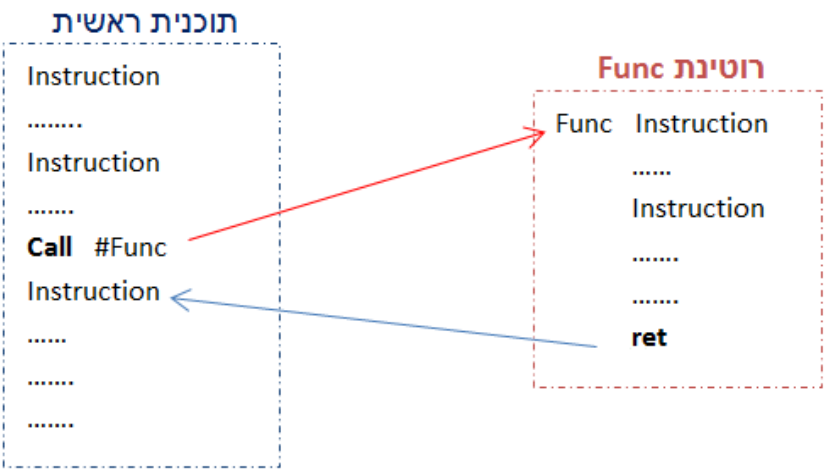

```
MOV    2(SP),R6  ; Item I2 -> R6
MOV    R7,0(SP)  ; Overwrite TOS with R7
PUSH   #0123h    ; Put 0123h onto TOS
POP    R8        ; R8 = 0123h
```

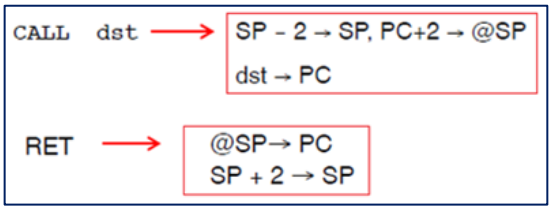


<u>הערה</u>: תוכן רגיסטר SP תמיד זוגי (ערך ביט LSB שווה תמיד 0) ולכן פקודת PUSH מחסרת את SP ב-2 ופקודת POP מקדמת את SP ב-2. כאשר עובדים בפקודת byte (PUSH.B , POP.B) רק ה- byte התחתון של המילה במחסנית מתעדכנת (ה- byte העליון לא משתנה ונשאר כמקודם).

B. פונקציות MACRO ורוטינות (Routine):

רוטינה (Routine):





כאשר בביצוע התוכנית יש צורך **בקטעי קוד החוזרים על עצמם**, נוכל להגדירם כרוטינה.

בכל שפה עילית **שם הפונקציה משמש מצביע לקטע במרחב הזיכרון בו היא מאוחסנת**. קטע זה מתחיל בכתובת שהתווית שלו (label – משמש כתובת) היא שם הפונקציה והפקודה האחרונה היא פקודת return.

הגדרת רוטינה נעשית ע"י כתיבת קטע קוד המתחיל בתווית (כדאי שתהיה לה שם עם משמעות) המשמשת dst לפקודת call (קריאה לרוטינה) ומסתיימת בפקודת ret (יציאה מרוטינה).

הרווח במקרה של שימוש ברוטינות, אנו לא "מנפחים" את זיכרון הבקר שלא לצורך והתכנית יותר מובנת ונוחה לתכנות ול- debug. **קטע קוד שחוזר על עצמו אין צורך שייכתב בזיכרון בכל פעם שנצטרך אותו**, אלא נקרא לו כל פעם מחדש. התשלום על רווח זה הוא הוספת מחזורי שעון לזמן ריצת התוכנית (כתוצאה מיציאה לרוטינה וחזרה ממנה).

❖ <u>שלבים בהפעלת רוטינה וחזרה ממנה</u>:

- כאשר ה- CPU מבצע פקודת call רגיסטר SP ערכו קטן ב-2 (אתחול כתובת באורך מילה) לצורך **אחסון כתובת לחזרה** (כתובת הפקודה הבאה אחרי פקודת call).
- **רגיסטר PC** (רגיסטר המכיל את כתובת הפקודה הבאה שעל ה- CPU לבצע) **נטען בכתובת התחלתית של הרוטינה** לצורך ביצועה.
- **קטע קוד הרוטינה מתבצע עד ההגעה לפקודת ret**. כשה-CPU מגיע לפקודת ret, **רגיסטר PC נטען בערך כתובת חזרה** הנמצאת במחסנית בכתובת שערכה הוא ערך רגיסטר SP , כך שלמעשה רגיסטר PC נטען בכתובת הפקודה הבאה אחרי פקודת call.

- **<u>דוגמה 1:</u>** הרוטינה הבאה, מחשבת את **סכום אברי מערך ומחלקת אותו ב-2**.
  את התוצאה נכתוב למשתנה Avg (למרות שזה לא חישוב ממוצע).

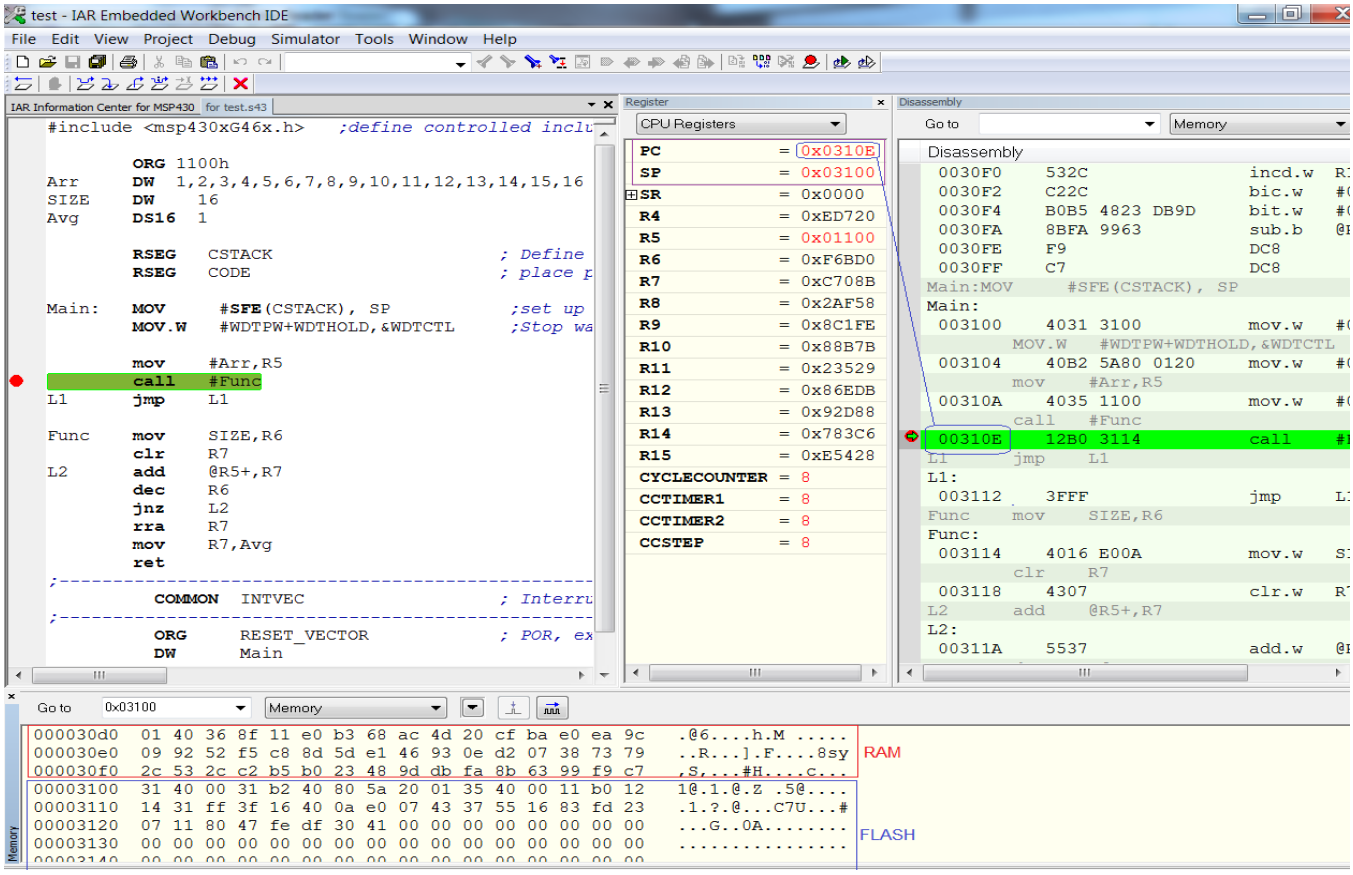
```
#include <msp430xG46x.h>    ;define controlled include file

        ORG     1100h
Arr     DW      1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
SIZE    DW      16
Avg     DS16    1

        RSEG  CSTACK    ; defines stack segment of 80 words (default)
        RSEG  CODE      ; ORG 0x3100 - place program in 'CODE' segment in to Flash memory

Main    MOV   #0x3100,SP     ; set the TOS to address 0x3100
        MOV   #Arr,R5
        MOV   SIZE,R6
        CALL  #Func
L1      JMP   L1

Func    CLR   R7
L2      ADD   @R5+,R7
        DEC   R6
        JNZ   L2
        RRA   R7
        MOV   R7,Avg
        RET
;-------------------------------------------------------------------------------
     COMMON  INTVEC          ; Interrupt Vectors
;-------------------------------------------------------------------------------
     ORG   RESET_VECTOR      ; POR, ext. Reset
     DW    Main
     END
```

קוד שקול - שפת C

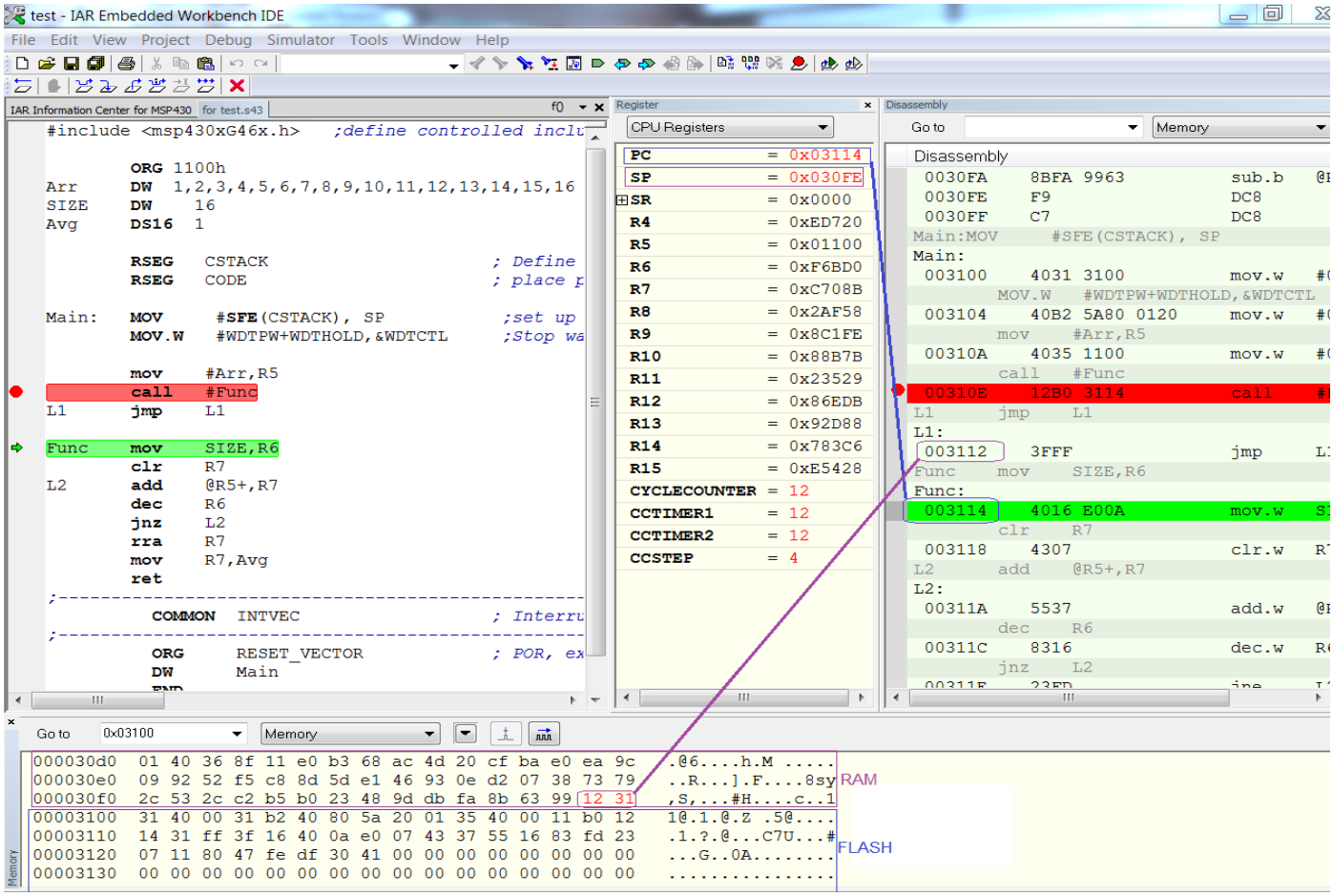
```c
int  Func(int Arr[],int SIZE){
   int temp = 0;
   for (int i=0 ; i<SIZE ; i++) temp += Arr[i];
   return temp;
}

void main(){
    int Arr[16]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16} , Avg;
    Avg = Func(Arr,16);
}
```

★ בתצלום הבא מופיעים ערכי התוכנית לפני ביצוע פקודת call.

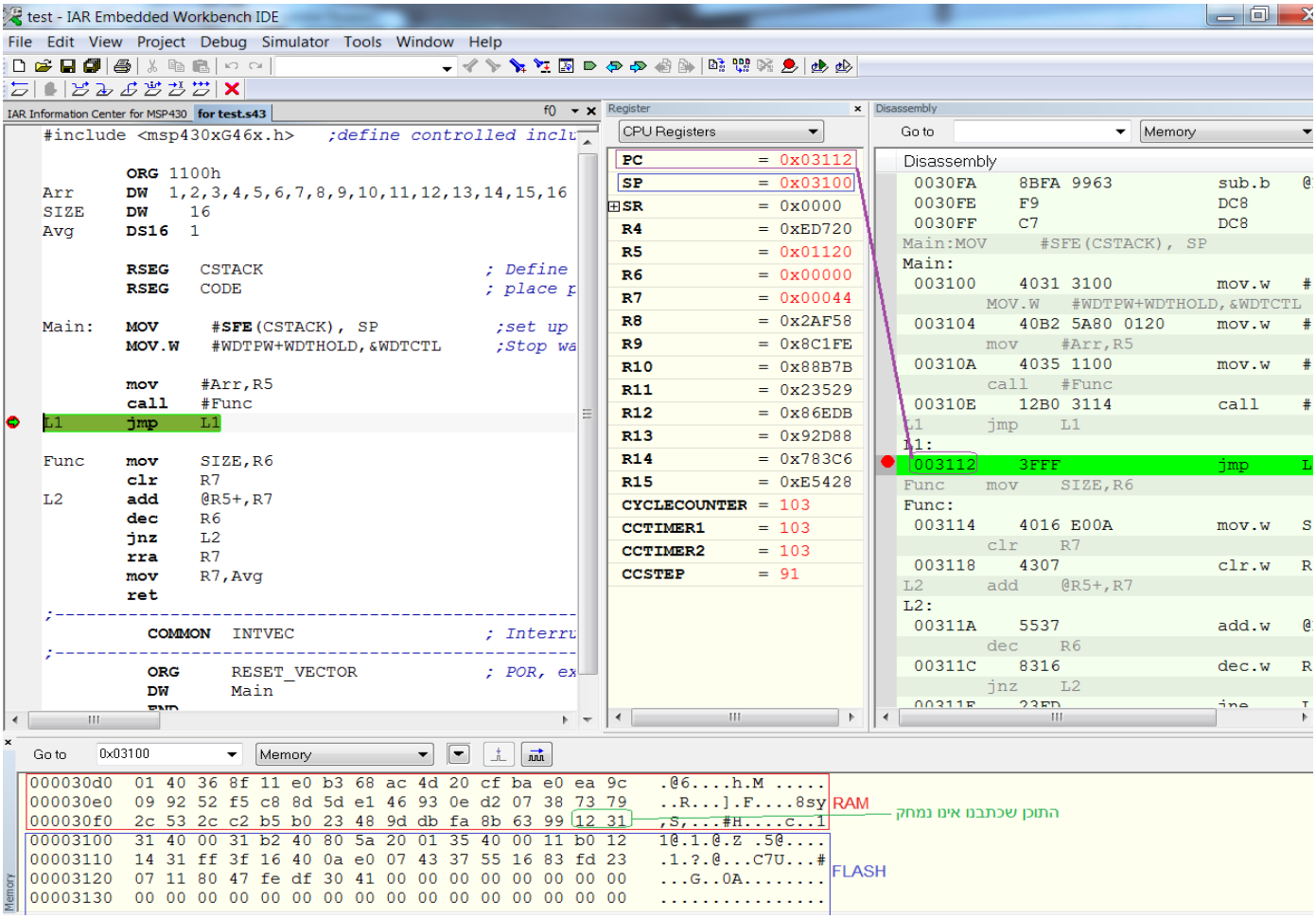
רגיסטר PC מכיל את כתובת פקודת call (ראו חלון Disassembly). רגיסטר SP מכיל את כתובת תחילת המחסנית **{**במקרה זה 0x03100 עקב הפקודה השמורה $Main:\quad MOV\quad \#SFE(CSTACK),SP$ השקולה לפקודה $MOV\ \#0x03100,SP$. כתובת זו היא סוף מרחב ה-RAM. נהוג לבצע זאת מאחר וטעינת המחסנית מתבצעת בחיסור רגיסטר SP, בשונה מרגיסטר PC שהולך ומתקדם מכתובת 0x03100 שבו מתחיל מרחב זיכרון ה-FLASH. כך נוכל "לנפח" את המחסנית בצורה הטובה ביותר**}** באופן כללי נוכל להגדירה כרצוננו **במרחב זיכרון ה-RAM בלבד**.



בתצלום הבא מופיעים ערכי התוכנית לאחר ביצוע פקודת call , לרגיסטר PC נטענת כתובת מיקום הרוטינה (תווית Func) בזיכרון והמחסנית נטענת בכתובת הפקודה שלאחר פקודת call (נקרא return address).



בתצלום הבא מופיעים ערכי התוכנית לאחר ביצוע פקודת ret ולפני ביצוע הפקודה שאנו אמורים לחזור אליה בחזרה מהרוטינה שאת כתובתה שמרנו במחסנית. לאחר חזרה מהרוטינה תוכן רגיסטר SP משתנה (חוזר להתחלה) אולם התכנים שנטענו למחסנית אינם נמחקים.



<u>הערה</u>: את נתוני כתובות מרחב זיכרון ה- RAM ומרחב זיכרון ה-FLASH לקחתי מקובץ נתוני מפרט החומרה (MSP430xG461x Datasheet – עמ' 16) של הבקר אתו נעבוד MSP430FG4618 (נמצא ב- MOODLE).

memory organization

		MSP430FG4616	MSP430FG4617	MSP430FG4618	MSP430FG4619
Memory Main: interrupt vector Main: code memory	Size Flash Flash	92KB 0FFFFh - 0FFC0h 018FFFh - 002100h	92KB 0FFFFh - 0FFC0h 019FFFh - 003100h	116KB 0FFFFh - 0FFC0h 01FFFFh - 003100h	120KB 0FFFFh - 0FFC0h 01FFFFh - 002100h
RAM (Total)	Size	4KB 020FFh - 01100h	8KB 030FFh - 01100h	8KB 030FFh - 01100h	4KB 020FFh - 01100h
Extended	Size	2KB 020FFh - 01900h	6KB 030FFh - 01900h	6KB 030FFh - 01900h	2KB 020FFh - 01900h
Mirrored	Size	2KB 018FFh - 01100h	2KB 018FFh - 01100h	2KB 018FFh - 01100h	2KB 018FFh - 01100h
Information memory	Size Flash	256 Byte 010FFh - 01000h	256 Byte 010FFh - 01000h	256 Byte 010FFh - 01000h	256 Byte 010FFh - 01000h
Boot memory	Size ROM	1KB 0FFFh - 0C00h	1KB 0FFFh - 0C00h	1KB 0FFFh - 0C00h	1KB 0FFFh - 0C00h
RAM (mirrored at 018FFh - 01100h)	Size	2KB 09FFh - 0200h	2KB 09FFh - 0200h	2KB 09FFh - 0200h	2KB 09FFh - 0200h
Peripherals	16 bit 8 bit 8-bit SFR	01FFh - 0100h 0FFh - 010h 0Fh - 00h	01FFh - 0100h 0FFh - 010h 0Fh - 00h	01FFh - 0100h 0FFh - 010h 0Fh - 00h	01FFh - 0100h 0FFh - 010h 0Fh - 00h

העברת פרמטרים וערכי החזרה דרך רוטינה:

ישנן 2 דרכים להעברת פרמטרים וערכי החזרה דרך רוטינה (ניתן לשלב בין שתי השיטות).

1. שימוש ברגיסטרים:

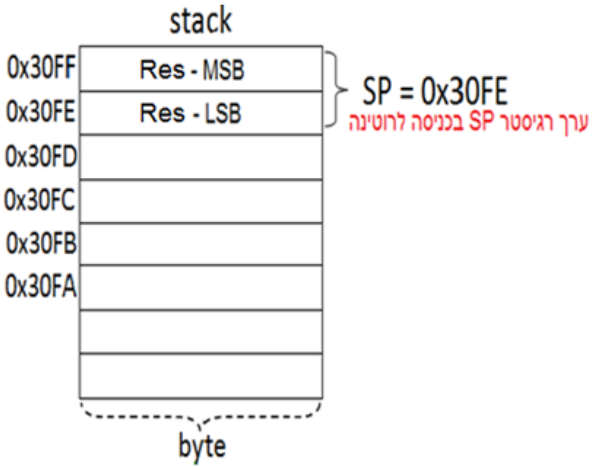
נגדיר מראש רגיסטרים שלתוכם נעביר את הפרמטרים, במעבר לרוטינה נשתמש עם ערכי רגיסטרים אלו בתוך פרמטרים של הרוטינה. באותה השיטה נשתמש עבור ערכי החזרה של הרוטינה.

- **דוגמה 2**:

שימוש ברוטינת adder לצורך חיבור var1+var2 והתוצאה ל- var3, הרגיסטרים R5,R4 מאחסנים את הפרמטרים var2,var1 בהתאמה. לפני יציאה לרוטינה (ברוטינה נעשה שימוש ב-2 רגיסטרים אלו שהוגדרו מראש) נטען את הפרמטרים ובחזרה מהרוטינה נטען את ערך ההחזרה (רגיסטר R5 למשתנה var3).

var3 = adder(var1 , var2)

```
;-----------------------------------------------
;                    Routine
;-----------------------------------------------
adder   add  R4,R5
        ret                ; return from routine
;-----------------------------------------------
;                    Main
;-----------------------------------------------
main    mov  var1,R4       ;content of var1 in R4
        mov  var2,R5       ;content of var2 in R5
        call #adder        ;call routine adder
Res     mov  R5,var3       ;result R5 in var3
```



2. שימוש במחסנית:

לפני יציאה לרוטינה נטען למחסנית את הפרמטרים (לפי סדר מוגדר מראש) כך שברוטינה נקרא את הפרמטרים מהמחסנית. באותה השיטה נשתמש בטעינת ערכי החזרה מתוך הרוטינה ונקרא אותם ביציאה ממנה.

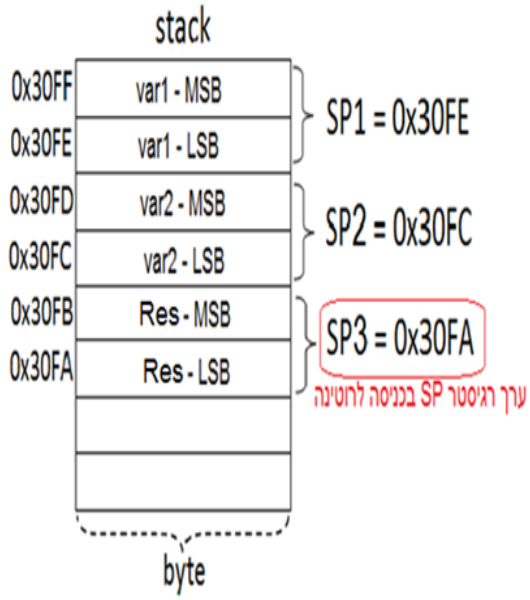
כל זאת נבצע בעזרת פקודות PUSH ו- POP.

- **דוגמה 3**:

שימוש ברוטינת adder לצורך חיבור var1+var2 והתוצאה ל- var3. נטען למחסנית את 2 הפרמטרים ונקרא לרוטינה. בתוך הרוטינה נטען את הפרמטרים לרגיסטרים R4,R5 (מאחסנים את הפרמטרים var2,var1 בהתאמה) ונבצע חיבור ביניהם. בסיום הרוטינה נדרוס במחסנית את תוכן var1 בערך ההחזרה (תוכן R5). ביציאה מהרוטינה נכתוב את התוצאה למשתנה var3.

var3 = adder(var1 , var2)

```
;-----------------------------------------------------
;                    Routine
;-----------------------------------------------------
adder   mov  2(SP),R4    ;get var2 from stack
        mov  4(SP),R5    ;get var1 from stack
        add  R4,R5
        mov  R5,4(SP)    ;place the result on stack, instead of var1
        ret              ;return from routine
;-----------------------------------------------------
;                    Main
;-----------------------------------------------------
main    push var1        ;place the var1 on stack
        push var2        ;place the var2 on stack
        call #adder      ;call routine adder
Res     add  #2,SP       ;point SP
        pop  var3        ;extract result to var3
```



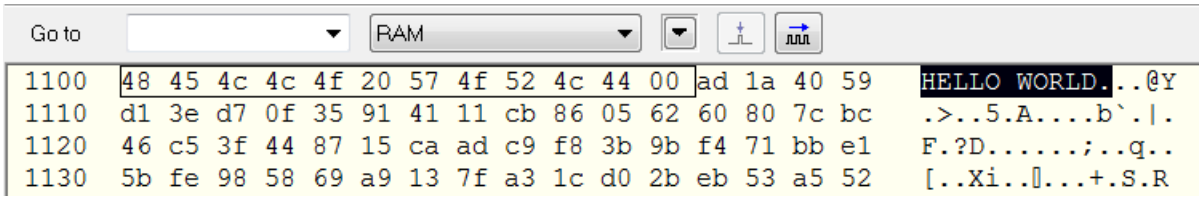
דוגמה 4: בקטע הקוד הבא, הרוטינה Func מקבלת כארגומנט מצביע למחרוזת ובעזרת המחסנית מבצעת היפוך למחרוזת (תמונת ראי), ללא שימוש במערכי עזר.

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

        ORG     1100h
Str     DB      "HELLO WORLD"

        ORG     0x3100
Main    MOV     #0x2100, SP
        MOV     #Str,R5
        CLR     R6
        CALL    #Func
L1      JMP     L1
;-----------------------------------------------------------------------
;               Routine
;-----------------------------------------------------------------------
Func    TST.B   0(R5)
        JZ      Finish
        PUSH.B  @R5
        INC     R5
        CALL    #Func
        POP.B   Str(R6)
        INC     R6
Finish  RET
;-----------------------------------------------------------------------
        COMMON  INTVEC          ; Interrupt Vectors
;-----------------------------------------------------------------------
        ORG     RESET_VECTOR    ; POR, ext. Reset
        DW      Main
        END
```

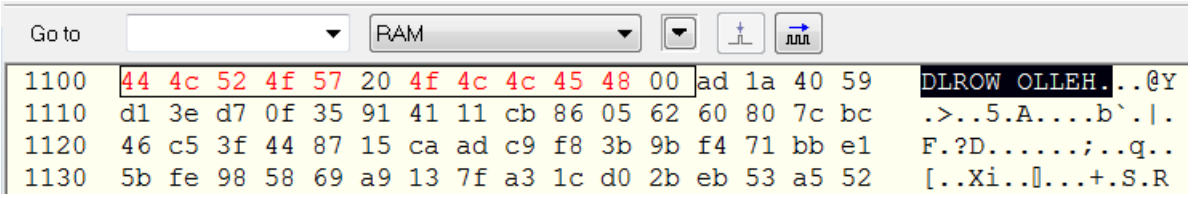
קוד שקול - שפת C

```c
int  reverse(Char *  str){
   char temp;
   if(str[0]==0) return;
   temp= str[0];
   reverse(str+1)
   Ptr ++ = temp;
}

void main(){
    Char *  Str = "HELLO WORLD", Ptr;
    Ptr = Str ;
    reverse (Str);
}
```

תוכן הזיכרון המכיל את המחרוזת – **בתחילת** ביצוע התוכנית



תוכן הזיכרון המכיל את המחרוזת – **בסיום** ביצוע התוכנית



דוגמה 5: ביצוע רקורסיבי של פעולת הסכום הבא $\sum_{i=n}^{k} i$

קוד שקול שפת C

```
int mySum (int n, int k) {
   if (n == k) return n;
  return n + mySum (n + 1, k);
}
```

```
#include <msp430xG46x.h>
                RSEG  CSTACK
                RSEG  CODE

Main            mov   #SFE(CSTACK),SP
                mov   #1,R4        ; n=1
                mov   #3,R5        ; k=3
                call  #mySum
Cont1           Jmp   $


mySum           cmp   R4,R5
                jne   recurse
                mov   R4,R6
                ret

recurse         push  R4
                inc   R4           ; n+1
                call  #mySum
Cont2           mov   @SP,R4
                add   R4,R6
                add   #2, SP
                ret
```

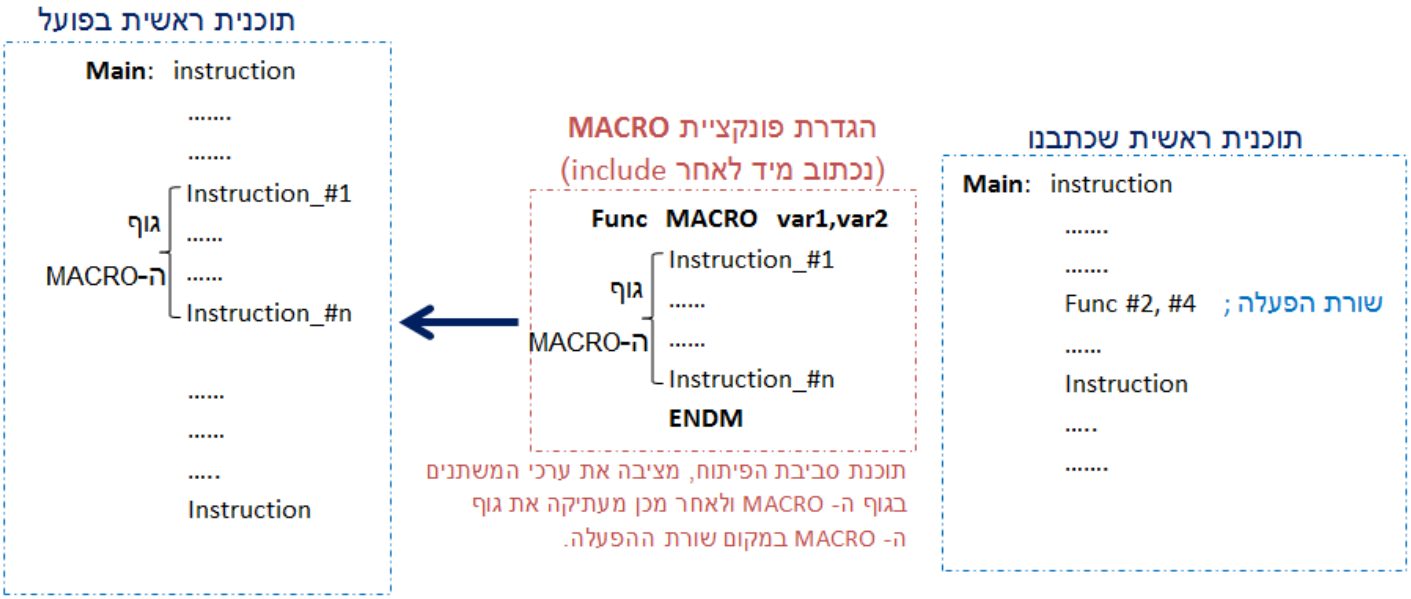
ביצוע הרצה עבור הקריאה ;mySum (1, 3)

SP	stack	R4 $\stackrel{def}{=}$ n	R5 $\stackrel{def}{=}$ k	R6
0x20FE	0x210E	1	3	3
0x20FC	0x0001	2		5
0x20FA	0x2120	3		6 Result
0x20F8	0x0002	2		
0x20F6	0x2120	1		
0x20F8	0x0002			
0x20FA	0x2120			
0x20FC	0x0001			
0x20FE	0x210E			

C. Macro Function:

כמו שראינו קודם, שימוש ברוטינות מאפשר כתיבת קוד בצורה תכנותית נוחה וקלה לתחזוקה. קוד הרוטינה נכתב פעם אחת במרחב הזיכרון ובכל קריאה לרוטינה ה-CPU פונה לקטע קוד זה מבצע אותו וממשיך מהמקום בו הפסיק לפני היציאה לרוטינה. בשונה משימוש ברוטינה ישנה אפשרות לשימוש בפונקציית MACRO העובדת על העיקרון שבכל שורת קוד המופיעה שימוש בפונקציית MACRO סביבת הפיתוח בתהליך בניית הפרויקט **תחליף שורה זאת בקטע קוד המהווה את גוף פונקציית ה-MACRO שהגדרנו מראש** (לאחר שורת ה- include). בשונה מקריאה מרוטינה וחזרה ממנה ע"י שימוש פונקציית MACRO אנו נמנעים מהוספת מחזורי שעון לזמן ריצת התוכנית, אולם אנו גורמים **"לניפוח הזיכרון"** לכל שורת קוד בה נשתמש בפונקציית MACRO.

להלן הגדרת פונקציית MACRO:



כאשר נרצה להשתמש בפונקציית MACRO המכילה תוויות (labels) לצורך מימוש לולאות וכדומה נשים לב שלא נוכל להשתמש בה יותר מפעם אחת. הסיבה לכך שפונקציית MACRO מחליפה את **שורת הקריאה לפונקציית ה- MACRO בקטע קוד המהווה את גוף פונקציית ה-MACRO שהגדרנו מראש** ומצד שני לכל תווית צריך להיות שם ייחודי. סתירה זו גורמת לשגיאת הידור התוכנית. בדוגמה הבאה ישנן 2 גישות לפתרון הבעיה.

דוגמה 6:

כתוב פונקציית MACRO המבצעת השהיית זמן ("שריפת" מחזורי שעון ללא ביצוע פעולות) של X מחזורי שעון.

1. פונקציית MACRO לשימוש חד פעמי , מאחר ומעבר להעתקה אחת **יהיו לנו תוויות (labels) זהות**, **דבר הגורם לשגיאה.**

```
;++++++++++++++++++++++++++++++++++++++++++++++++
;     MACRO kind 1 - the loop lables are One Use
;------------------------------------------------
Delay1      MACRO   time,multiple
            mov.w   multiple,R5
L1          mov.w   time,R4
L2          dec.w   R4
            jnz     L2
            dec.w   R5
            jnz     L1
            ENDM
```

2. פונקציית MACRO לשימוש **רב** פעמי בהגדרת התוויות כ- LOCAL.

```
;-----------------------------------------------------------
;     MACRO kind 2 - the loop lables are Multiple Occurrence
;-----------------------------------------------------------
Delay3      MACRO   time,multiple
            LOCAL   L1,L2
            mov.w   multiple,R5
L1          mov.w   time,R4
L2          dec.w   R4
            jnz     L2
            dec.w   R5
            jnz     L1
            ENDM
```

3. פונקציית MACRO לשימוש **רב** פעמי בשימוש פרמטר לחישוב כתובת יחסית (offset מהווה למעשה את כתובת שורת הקריאה לפונקציית ה- MACRO).

```
;-----------------------------------------------------
;     MACRO kind 3 - the loop lables are Multiple
;                    Occurrence using relative address
;-----------------------------------------------------
Delay2      MACRO   time,multiple,offset
            mov.w   multiple,R5
            mov.w   time,R4
            dec.w   R4
            jnz     offset+6
            dec.w   R5
            jnz     offset+2
            ENDM
```

<u>הערה:</u> כאשר נרצה לדבג את קוד פונקציית ה-MACRO לא נוכל לבצע זאת כאשר עשינו ברוטינה. מאחר והקוד מוחלף בתהליך בניית הפרויקט ובפועל אינו מופיע ב-Editor. על כן, נוכל לכתוב קוד בצורה רגילה ואח"כ "לחתוך" אותו ולהגדירו ע"י פונקציית MACRO.

דוגמה 7: בדוגמה זו, נחזור על המבוקש בדוגמה 1 בשימוש MACRO במקום שימוש ברוטינה.
תזכורת – התוכנית מחשבת את **סכום אברי מערך ומחלקת אותו ב-2**. את התוצאה נכתוב למשתנה Avg
(למרות שזה לא חישוב ממוצע).

```
#include <msp430xG46x.h>    ; define controlled include file
;-------------------------------------------------------------------------------
;     MACRO definition
;-------------------------------------------------------------------------------
Func    MACRO  arr,size
        LOCAL   L
        MOV     arr,R5
        MOV     size,R6
        CLR     R7
L       ADD     @R5+,R7
        DEC     R6
        JNZ     L
        RRA     R7
        ENDM
;-------------------------------------------------------------------------------
        ORG 1100h
Arr     DW   1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
SIZE    DW   16
Avg     DS16 1

        RSEG  CODE      ; ORG 0x3100 - place program in 'CODE' segment in to Flash memory

Main    MOV   #0x3100,SP        ; set the TOS to address 0x3100
        Func  #Arr,SIZE
        MOV   R7,Avg
L1      JMP   L1
        NOP
;-------------------------------------------------------------------------------
     COMMON  INTVEC             ; Interrupt Vectors
;-------------------------------------------------------------------------------
     ORG    RESET_VECTOR        ; POR, ext. Reset
     DW     Main
     END
```

תוכן הזיכרון המכיל את המחרוזת – **בסיום** ביצוע התוכנית