בס"ד

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

קובץ הכנה ניסוי מעבדה מס' 1 – חלק 1

Tutorial 1.1 – Assembly acquaintance, Part 1

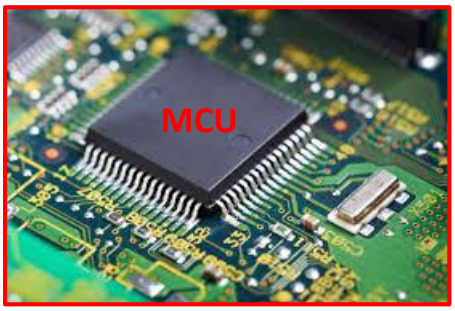
מעבדת מיקרומחשבים – המחלקה להנדסת חשמל ומחשבים

מס' קורס - 361.1.3353

כתיבה ועריכה: חנן ריבוא

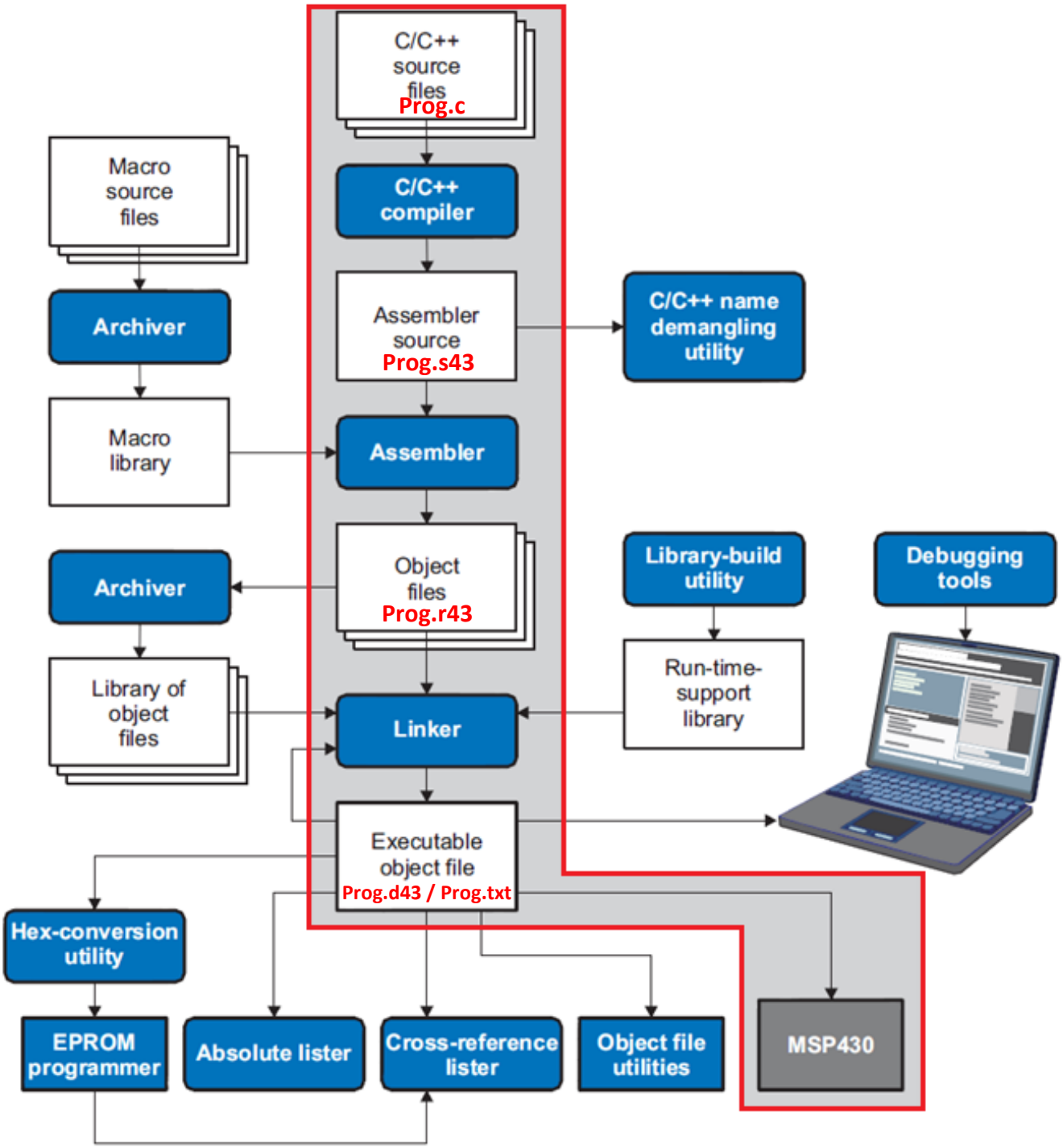
מהדורה 1 – שנה"ל תשע"ו

A. Embedded Systems:



Embedded Systems

B. Software Development:

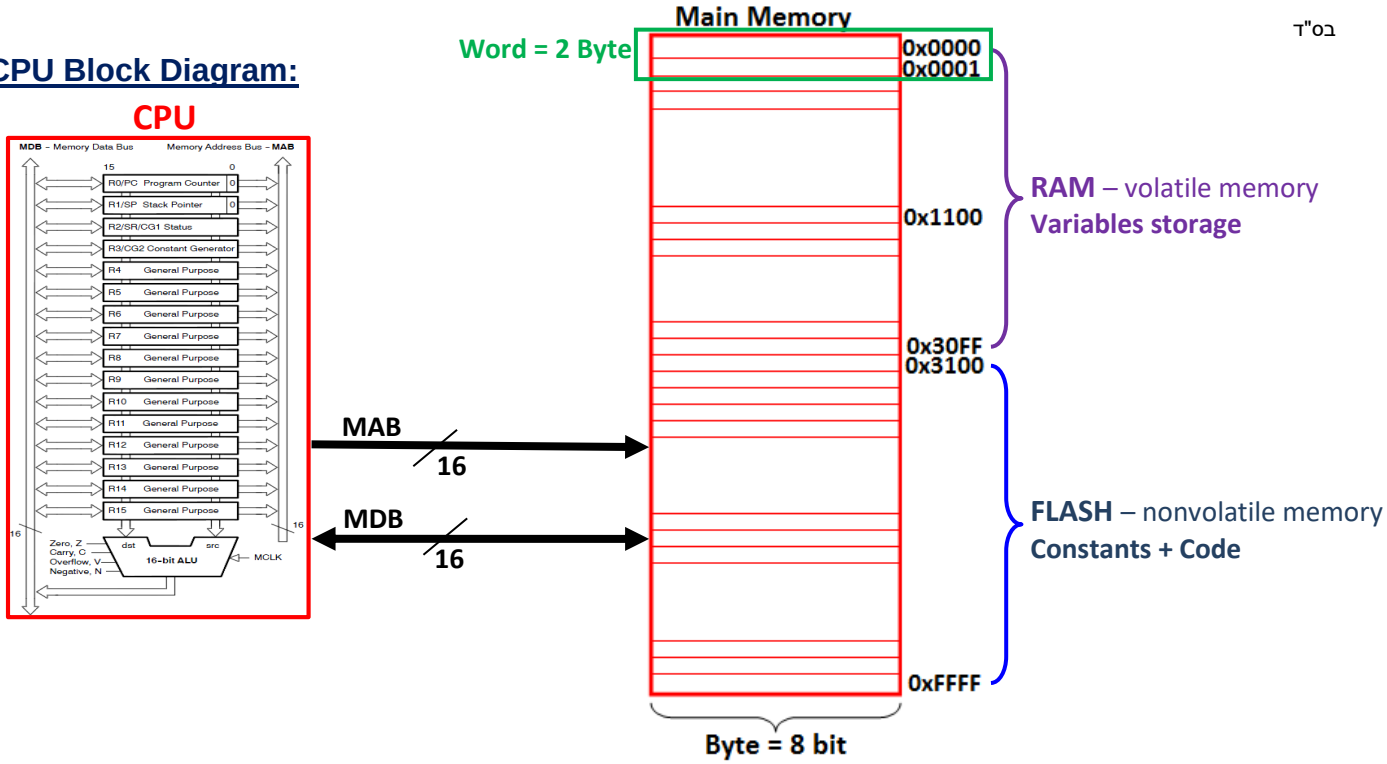


דגשים:

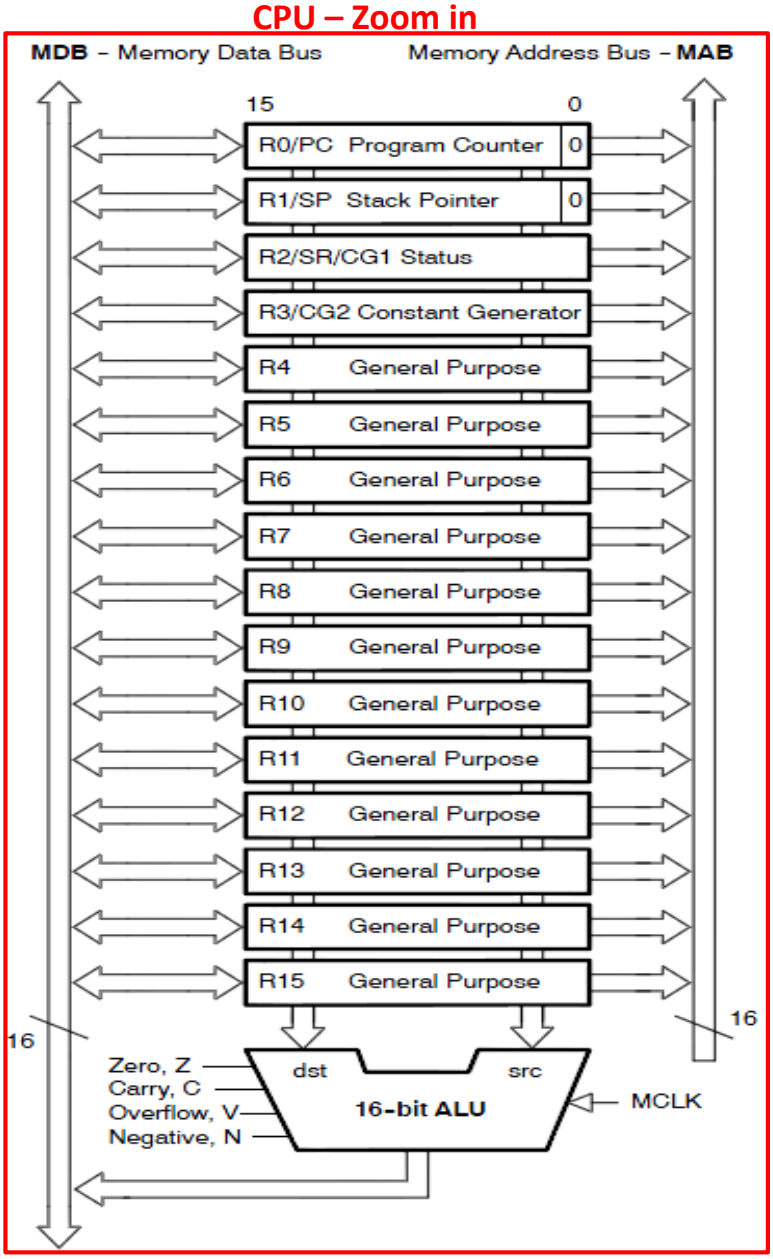
1. קוד בשפת אסמבלי – כתיבת קוד ברזולוציה הגבוהה ביותר, לתיאור פקודות מכונה (תיאור של אחדים ואפסים). בצורה זו נשיג ניצול מרבי של **מהירות** המעבד וחסכון **בזיכרון** המעבד.
2. לכל קובץ מקור מיוצר קובץ object ע"י ה- assembler. קובץ ה- object מכיל מידע על קובץ המקור אליו הוא שייך.
3. תפקיד ה- linker לאחד את כל קובצי ה- object ולייצר מהם קובץ הרצה המכיל את תוכן הזיכרון לפקודות ונתונים לפי כתובות מעשיות.
4. ישנם שני סוגי קובצי הרצה במוצא ה- linker. קובץ הרצה Prog.d43 המתאים לאופן **Debug mode** וקובץ הרצה Prog.txt המתאים לאופן **Release mode**.

בס"ד

C. CPU Block Diagram:



CPU – Zoom in



R0/PC register – contains the next instruction address

R1/SP register – You will learn at the LAB2

R2/SR/CG1 register – contains the Status bits, see Next page

R3/CG2 register – You will learn at the LAB2

R4 – R15 registers – General Purpose registers

SR	15 … 9	8	7						0
	Reserved	V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C

- **C** = **Carry bit.** This bit is **set** when the result of a (byte or word) operation produced a carry and **cleared** when no carry occurred.
- **Z** = **Zero bit.** This bit is **set** when the result of a (byte or word) operation is 0 and **cleared** when the result is not 0.
- **N** = **Negative bit.** This bit is **set** when the result of a (byte or word) operation is negative and **cleared** when the result is not negative.
- **GIE** = General interrupt enable.
- **CPUOFF** = CPU off
- **OSCOFF** = Oscillator Off.
- **SCG0** = System clock generator 0.
- **SCG1** = System clock generator 1.
- **V** = **Overflow bit.** This bit is **set** when the result of an arithmetic operation overflows the signed-variable range and **cleared** otherwise.

Overflow – גלישה

- בעיה זו נוצרת כשתוצאת החיבור לא יכולה להיות מיוצגת ע"י מספר הסיבות הנתון.
- דוגמה – חיבור 2 מספרים חיוביים וקבלת מספר שלילי:

$$\begin{array}{r} 0XXX \\ +\quad 0XXX \\ = \\ 1XXX \end{array}$$

- דוגמה - חיבור 2 מספרים שלילים וקבלת מספר חיובי:

$$\begin{array}{r} 1XXX \\ +\quad 1XXX \\ = \\ 10XXX \end{array}$$

- במקרה של Overflow אין פתרון לתיקון התוצאה, אלא התראה למשתמש שלא ניתן לבצע חיבור.

Carry – נשא

- בעיה זו נובעת מאופן ייצוג המספרים. במקרה זה תוצאת החיבור אינה חורגת מתחום המספרים, אך יש נשא (Carry).
- דוגמה – חיבור מספר חיובי עם שלילי

$$\begin{array}{r} 0XXX \\ +\quad 1XXX \\ = \\ 10XXX \end{array}$$

- דוגמה – חיבור מספר שלילי עם שלילי

$$\begin{array}{r} 1XXX \\ +\quad 1XXX \\ = \\ 11XXX \end{array}$$

- ניתן לתקן את התוצאה ע"י:
  1. בשיטת המשלים ל- 1, מוסיפים את ה- Carry.
  2. בשיטת המשלים ל- 2, מתעלמים מה- Carry.

דוגמה עבור m=4: 5+7

$$\begin{array}{r} 0101 \\ +\ 0111 \\ \hline 1100 \end{array}$$

ov

דוגמה עבור m=4:
-5-3 = -5+(-3)

$$\begin{array}{r} 1011 \\ +\ 1101 \\ \hline 11000 \end{array}$$

carry

דוגמה עבור m=4:
-5-5 = -5+(-5)

$$\begin{array}{r} 1011 \\ +\ 1011 \\ \hline 10110 \end{array}$$

carry ov

D. Big and Little Endian

ערך של כתובת במרחב הזיכרון הפיזי מחולק לפי בתים (היחידה הקטנה ביותר, גודלה **Byte=8 bit**) לכן כל כתובת משויכת ל-Byte ספציפי בזיכרון.

כאשר ברצוננו לאחסן מידע שגודלו Word (מעל Byte, תלוי בארכיטקטורה).

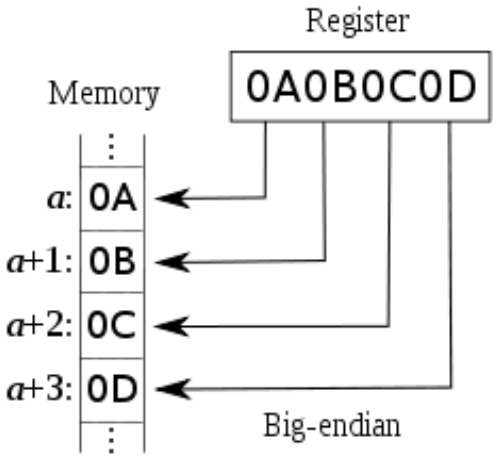
גדלים אופייניים ל-Word הם: **16** bit, **32** bit, **64** bit

שאלה: איך הוא תמוקם המילה בזיכרון בפיצול ליחידות של Byte ?

תשובה: ישנן 2 גישות (המובאות בהמשך) הנבדלות משיקולי חומרה ברמת הארכיטקטורה כתוצאה מהצורך לעסוק במידע בייצוגים שונים.

1. Big Endian:

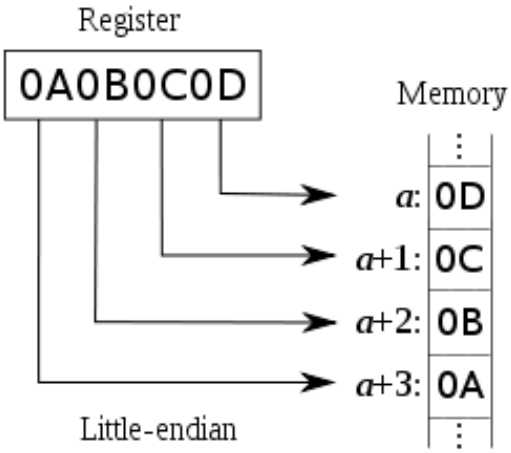
ה- Byte **במיקום MSB** של המילה (**Big**) ימוקם **בכתובת הנמוכה** בזיכרון (**Endian**).



בימינו השימוש נפוץ ברשתות מחשבים.

2. Little Endian:

ה- Byte **במיקום LSB** של המילה (**Little**) ימוקם **בכתובת הנמוכה** בזיכרון (**Endian**).



בימינו השימוש נפוץ ב-Microprocessors (כולל ב- **MSP430**).

E. Assembly Instruction – Field Division:

```
[Label]   MOV Operand_SRC, Operand_DST   ; Comment
```

F. Address Mode Descriptions:

את מיקום הנתונים ע"י אופרנד המקור (**Operand_SRC**) ניתן לציין בשבעה אופנים ואת מיקום הנתונים ע"י אופרנד היעד (**Operand_DST**) ניתן לציין בארבעה אופנים.
אופניי ייצוג אלו נקראים "שיטות מיעון".

ADDRESS MODE	S	D	SYNTAX	EXAMPLE	OPERATION
Register	●	●	MOV Rs,Rd	MOV R10,R11	R10 —> R11
Indexed	●	●	MOV X(Rn),Y(Rm)	MOV 2(R5),6(R6)	M(2+R5)—> M(6+R6)
Symbolic (PC relative)	●	●	MOV EDE,TONI		M(EDE) —> M(TONI)
Absolute	●	●	MOV & MEM, & TCDAT		M(MEM) —> M(TCDAT)
Indirect	●		MOV @Rn,Y(Rm)	MOV @R10,Tab(R6)	M(R10) —> M(Tab+R6)
Indirect autoincrement	●		MOV @Rn+,Rm	MOV @R10+,R11	M(R10) —> R11 R10 + 2—> R10
Immediate	●		MOV #X,TONI	MOV #45,TONI	#45 —> M(TONI)

NOTE: S = source D = destination

Note:

```
MOV   #LABEL,PC ; Branch to address LABEL
MOV   LABEL,PC  ; Branch to address contained in LABEL
MOV   @R14,PC   ; Branch indirect to address in R14
```

G. Word and Byte instructions:

Register-Byte		Byte-Register		Operation
High Byte Low Byte Unused → Byte Register → Memory		High Byte Low Byte Byte → 0h Memory → Register		Description
R5 = 0A28Fh R6 = 0203h Mem(0203h) = 012h	ADD.B R5,0(R6) 08Fh Low Byte + 012h Addressed byte 0A1h Mem (0203h) = 0A1h C = 0, Z = 0, N = 1	R5 = 01202h R6 = 0223h Mem(0223h) = 05Fh	ADD.B @R6,R5 05Fh Addressed byte + 002h Low Byte 00061h R5 = 00061h C = 0, Z = 0, N = 0	Example

בס"ד

H. MSP430 Instruction Set: *27 CORE Instructions + 24 EMULATED Instructions*

Mnemonic		Description		V	N	Z	C
ADC(.B)*	dst	Add C to destination (**ADDC #0,dst**)	dst + C → dst	*	*	*	*
ADD(.B)	src,dst	Add source to destination	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	Add source and C to destination	src + dst + C → dst	*	*	*	*
AND(.B)	src,dst	AND source and destination	src .and. dst → dst	0	*	*	*
BIC(.B)	src,dst	Clear bits in destination	.not.src .and. dst → dst	-	-	-	-
BIS(.B)	src,dst	Set bits in destination	src .or. dst → dst	-	-	-	-
BIT(.B)	src,dst	Test bits in destination	src .and. dst	0	*	*	*
BR†	dst	Branch to destination (**MOV dst, PC**)	dst → PC	-	-	-	-
CALL	dst	Call destination	PC+2 → stack, dst → PC	-	-	-	-
CLR(.B)*	dst	Clear destination **(MOV #0, dst)**	0 → dst	-	-	-	-
CLRC†		Clear C (**BIC #0x0001, SR**)	0 → C	-	-	-	0
CLRN†		Clear N (**BIC #0x0004, SR**)	0 → N	-	0	-	-
CLRZ†		Clear Z (**BIC #0x0002, SR**)	0 → Z	-	-	0	-
CMP(.B)	src,dst	Compare source and destination	dst - src	*	*	*	*
DADC(.B)*	dst	Add C decimally to destination (**DADD #0, dst**)	dst + C → dst (decimally)	*	*	*	*
DADD(.B)	src,dst	Add source and C decimally to dst.	src + dst + C → dst (decimally)	*	*	*	*
DEC(.B)*	dst	Decrement destination (**SUB #1, dst**)	dst - 1 → dst	*	*	*	*
DECD(.B)*	dst	Double-decrement destination (**SUB #2, dst**)	dst - 2 → dst	*	*	*	*
DINT*		Disable interrupts (**BIC #0x0008, SR**)	0 → GIE	-	-	-	-
EINT*		Enable interrupts (**BIS #0x0008, SR**)	1 → GIE	-	-	-	-
INC(.B)*	dst	Increment destination (**ADD #1, dst**)	dst +1 → dst	*	*	*	*
INCD(.B)*	dst	Double-increment destination (**ADD #2, dst**)	dst+2 → dst	*	*	*	*
INV(.B)*	dst	Invert destination (**XOR #FFFFh, dst**)	.not.dst → dst	*	*	*	*
JC/JHS	label	Jump if C set/Jump if higher or same		-	-	-	-
JEQ/JZ	label	Jump if equal/Jump if Z set		-	-	-	-
JGE	label	Jump if greater or equal		-	-	-	-
JL	label	Jump if less		-	-	-	-
JMP	label	Jump	PC + 2 x offset → PC	-	-	-	-
JN	label	Jump if N set		-	-	-	-
JNC/JLO	label	Jump if C not set/Jump if lower		-	-	-	-
JNE/JNZ	label	Jump if not equal/Jump if Z not set		-	-	-	-
MOV(.B)	src,dst	Move source to destination	src → dst	-	-	-	-
NOP*		No operation (**MOV #0, R3**)		-	-	-	-
POP(.B)*	dst	Pop item from stack to destination **(MOV @SP+, dst)**	@SP → dst, SP+2 → SP	-	-	-	-
PUSH(.B)	src	Push source onto stack	SP - 2 → SP, src → @SP	-	-	-	-
RET*		Return from subroutine (**MOV @SP+, PC**)	@SP → PC, SP + 2 → SP	-	-	-	-
RETI		Return from interrupt		*	*	*	*
RLA(.B)*	dst	Rotate left arithmetically (**ADD dst, dst**)		*	*	*	*
RLC(.B)*	dst	Rotate left through C **(ADDC dst,dst)**		*	*	*	*
RRA(.B)	dst	Rotate right arithmetically		0	*	*	*
RRC(.B)	dst	Rotate right through C		*	*	*	*
SBC(.B)*	dst	Subtract not(C) from destination **(SUBC #0,dst)**	dst + 0FFFFh + C → dst	*	*	*	*
SETC*		Set C **(BIS #1,SR)**	1 → C	-	-	-	1
SETN*		Set N **(BIS #4,SR)**	1 → N	-	1	-	-
SETZ*		Set Z **(BIS #2,SR)**	1 → Z	-	-	1	-
SUB(.B)	src,dst	Subtract source from destination (2's complement SUB)	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	Subtract source and not(C) from dst. (1's complement SUB)	dst + .not.src + C → dst	*	*	*	*
SWPB	dst	Swap bytes		-	-	-	-
SXT	dst	Extend sign		0	*	*	*
TST(.B)*	dst	Test destination **(CMP #0,dst)**	dst + 0FFFFh + 1	0	*	*	1
XOR(.B)	src,dst	Exclusive OR source and destination	src .xor. dst → dst	*	*	*	*

מרחב הסתעפות של
PC+511words
PC-512words

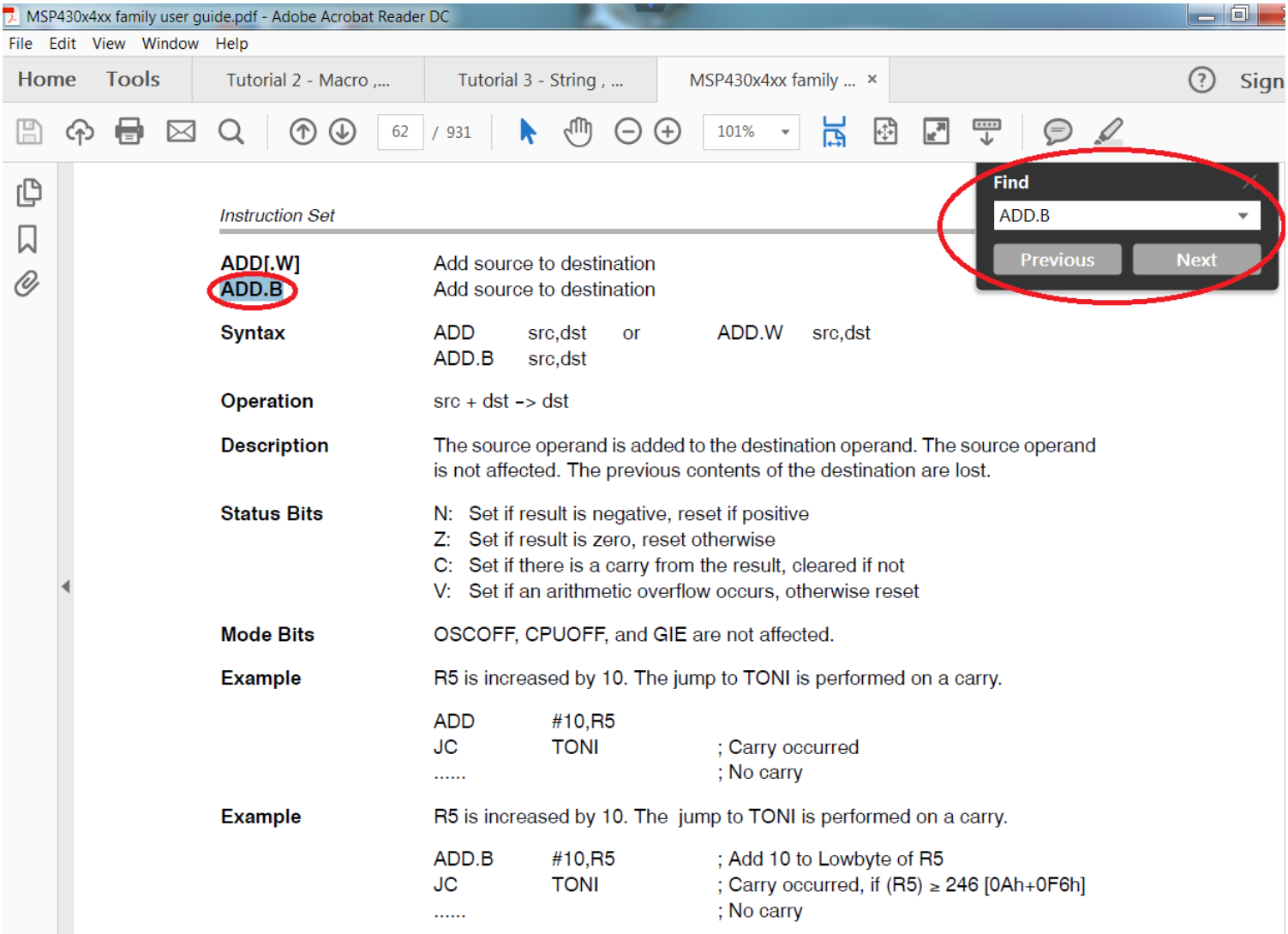
* Emulated Instruction

מקרא: סימון * אומר שביט המצב (N,Z,C,V) מושפע מן הפקודה.
סימון - אומר שביט המצב (N,Z,C,V) אינו מושפע מן הפקודה ונשאר כמקודם.

I. Command details:

Question - Do you want to look for details of command **ADD.B**

Answer – Open "User guide file ..", Click ctrl+f and search for string **ADD.B**



MSP430x4xx family user guide.pdf - Adobe Acrobat Reader DC

File Edit View Window Help

Home Tools | Tutorial 2 - Macro ,... | Tutorial 3 - String , ... | MSP430x4xx family ... | Sign

62 / 931 101%

Find: ADD.B — Previous Next

Instruction Set

ADD[.W]	Add source to destination
ADD.B	Add source to destination
Syntax	ADD src,dst or ADD.W src,dst ADD.B src,dst
Operation	src + dst -> dst
Description	The source operand is added to the destination operand. The source operand is not affected. The previous contents of the destination are lost.
Status Bits	N: Set if result is negative, reset if positive Z: Set if result is zero, reset otherwise C: Set if there is a carry from the result, cleared if not V: Set if an arithmetic overflow occurs, otherwise reset
Mode Bits	OSCOFF, CPUOFF, and GIE are not affected.
Example	R5 is increased by 10. The jump to TONI is performed on a carry.

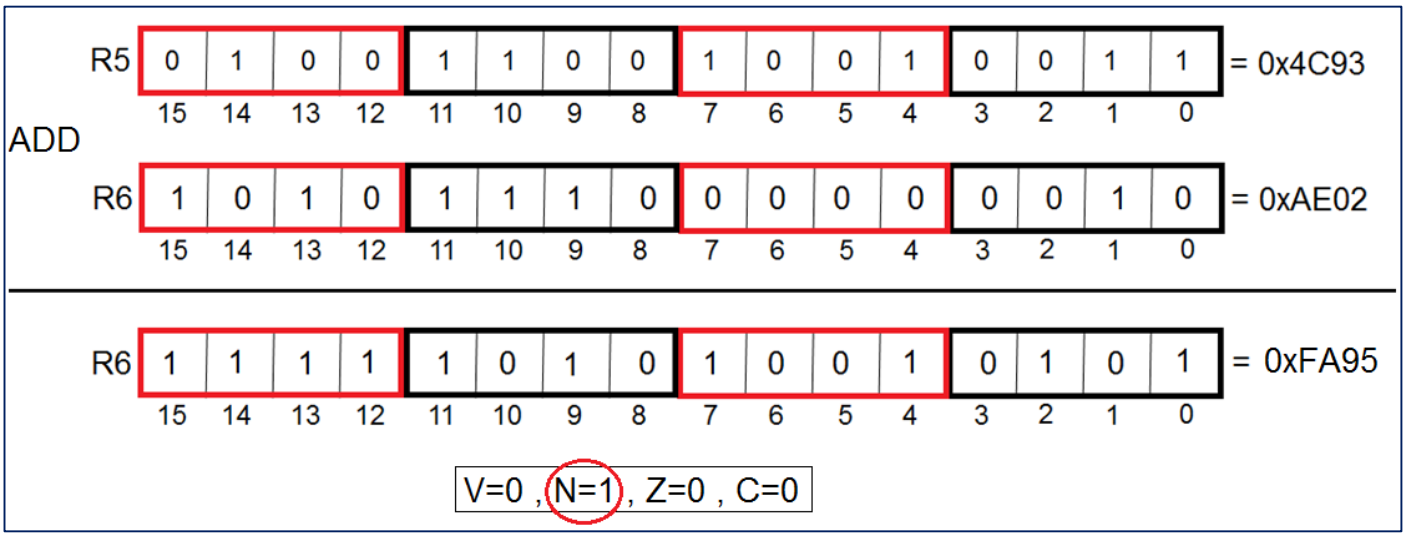
```
ADD     #10,R5
JC      TONI        ; Carry occurred
......              ; No carry
```

Example R5 is increased by 10. The jump to TONI is performed on a carry.

```
ADD.B   #10,R5      ; Add 10 to Lowbyte of R5
JC      TONI        ; Carry occurred, if (R5) ≥ 246 [0Ah+0F6h]
......              ; No carry
```

J. Examples:

1) ADD R5,R6 when R5=0x4C93 , R6=0xAE02 , R5+R6->R6



בס"ד

2) AND R5,R6 when R5=0x4C93 , R6=0xAE02 , R5 and R6 -> R6

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R5	0	1	0	0	1	1	0	0	1	0	0	1	0	0	1	1	= 0x4C93
AND																	
R6	1	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	= 0xAE02
R6	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	= 0x0C02

3) BIC #0x4242,R6 when R6=0xFEFE , not(0x4242) and R6 -> R6

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	0	= 0x4242
BIC																	
R6	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	= 0xFEFE
R6	1	0	1	1	1	1	0	0	1	0	1	1	1	1	0	0	= 0xBCBC

4) BIS #0x0101,R6 when R6=0xFEFE , 0x4242 or R6 -> R6

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	= 0x0101
BIS																	
R6	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	= 0xFEFE
R6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	= 0xFFFF

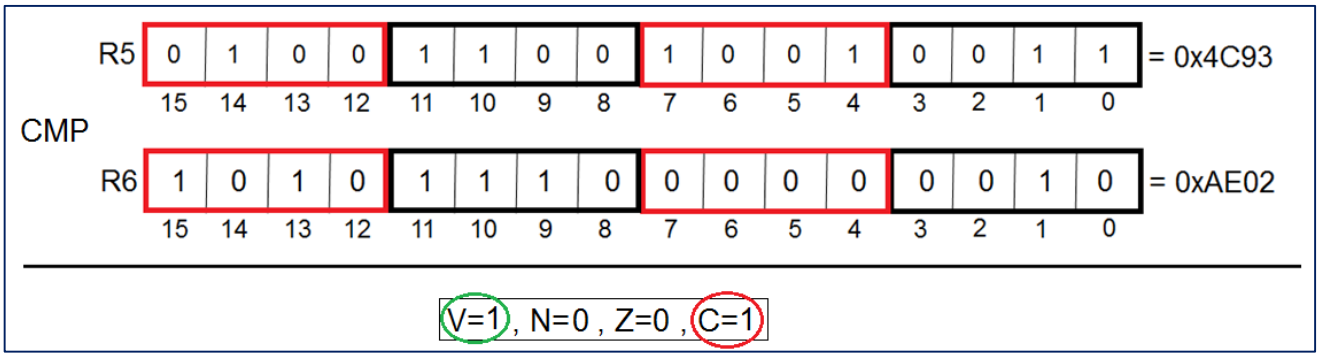
5)

```
CMP   R5,R6   ; R6 - R5
JHS   label   ; JC  label
```

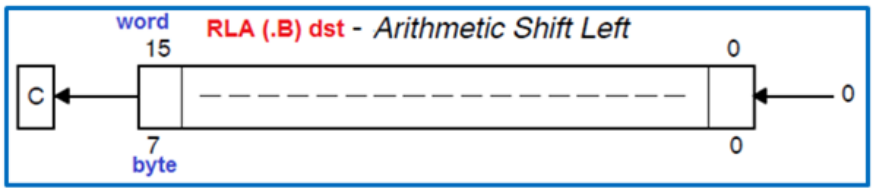
when R5=0x4C93 , R6=0xAE02

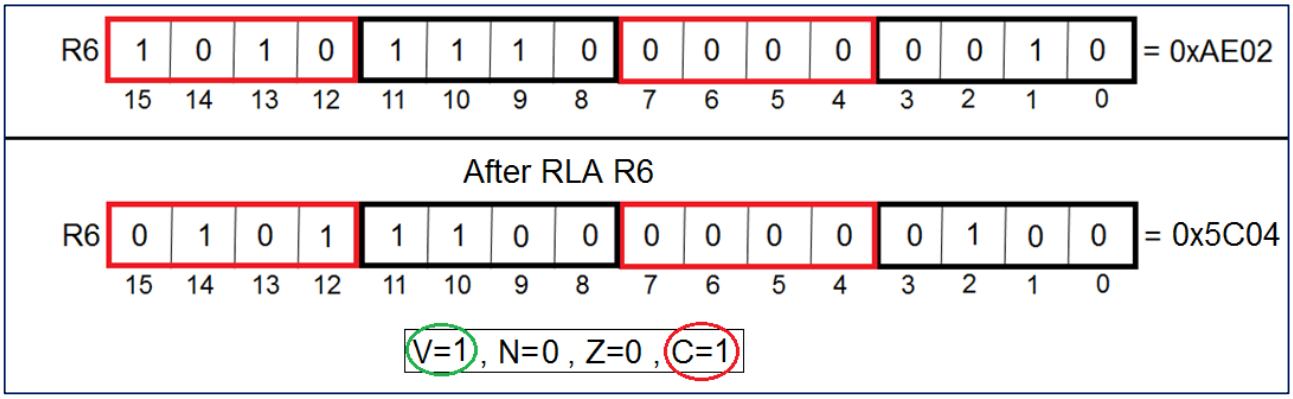
Only flags are influenced by this command.

$\boldsymbol{if}\ \ R6 \geq R5\ \ \boldsymbol{then}\ \ flag\ C = 1\,, \boldsymbol{else}\ flag\ C = 0.$

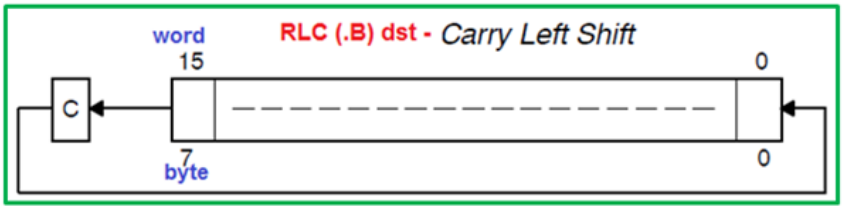


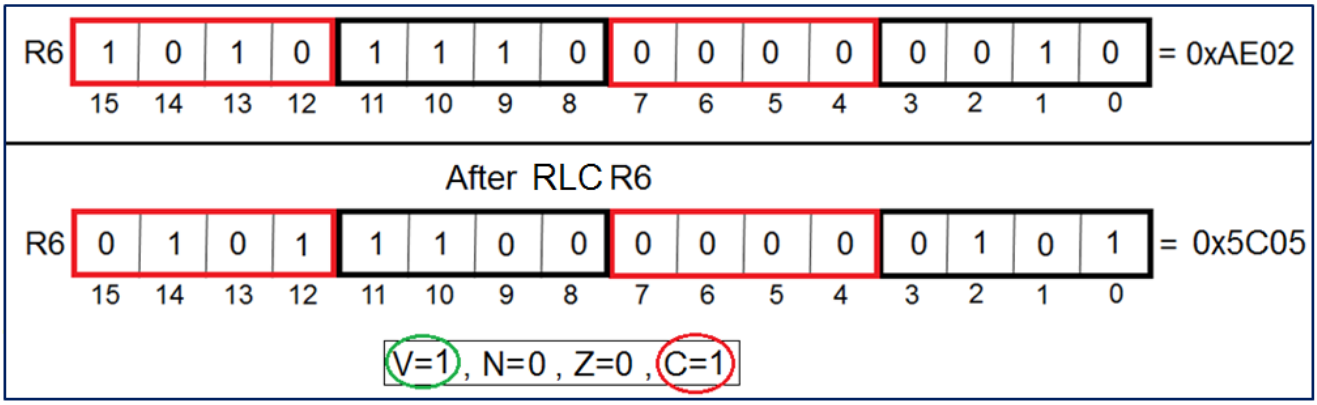
6) RLA R6 when R6=0xAE02



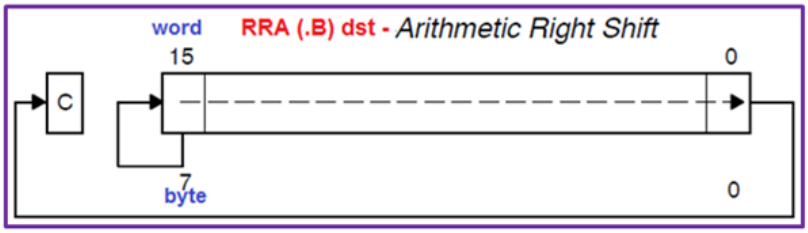


7) RLC R6 when R6=0xAE02 , flag C=1





8) RRA R6 when R6=0xAE02



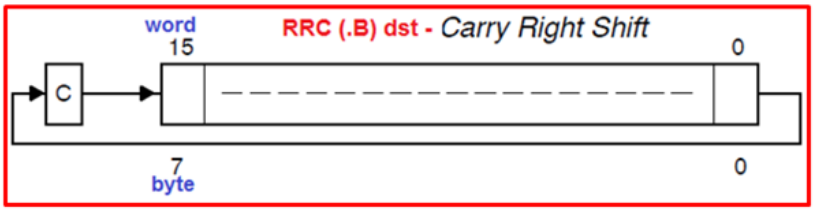
R6	1	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	= 0xAE02
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

After RRA R6

R6	1	1	0	1	0	1	1	1	0	0	0	0	0	0	0	1	= 0xD701
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

V=0 , N=1 , Z=0 , C=0

9) RRC R6 when R6=0xAE02 , flag C=0



R6	1	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	= 0xAE02
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

After RRA R6

R6	0	1	0	1	0	1	1	1	0	0	0	0	0	0	0	1	= 0x5701
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

V=0 , N=0 , Z=0 , C=0

10) SWPB R6 when R6=0xAE02

R6	1	0	1	0	1	1	1	0	0	0	0	0	0	0	1	0	= 0xAE02
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

After SWPB R6

R6	0	0	0	0	0	0	1	0	1	0	1	0	1	1	1	0	= 0x02AE
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

11) SXT R6 when R6=0x0084

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R6	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	= 0x0084

After SXT R6

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R6	1	1	1	1	1	1	1	1	1	0	0	0	0	1	0	0	= 0xFF84

K. Appendix – Program Skeleton

```
;*******************************************************************************
;  Program : Program Skeleton
;  comments
;*******************************************************************************
#include "msp430xG46x.h"                        ; define controlled include file

        ORG   1100h                     ; The beginng of RAM segment = ORG   1100h
Arr     DW 2,4,6,8,10,12,14,16,18       ; Array definition
Avg     DW 0
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 (at this string after the '!')
STRsize DC16   STRsize-MyStr1           ; size of string
Location EQU   01130h                   ; Results array storage

        RSEG   CODE                     ; place in 'CODE' segment in to flash memory = ORG  3100h

MyStr2  DB     "HELLO WORLD, My name is MSP430!"
SIZE    DW     8

Main:
        ; YOUR PROGRAM ..................

;-------------------------------------------------------------------------------
        COMMON  INTVEC                  ; Interrupt Vectors
;-------------------------------------------------------------------------------
        ORG     RESET_VECTOR            ; POR, ext. Reset
        DW      Main
        END
```

Onto RAM

Onto FLASH

L. Assembly language directives: An MSP430 Example

```
     ORG 0xF000 ; ORG directive forces the assembler to move the
                ; location counter (location pointer) to the value
                ; specified by the ORG directive.
b1   DB 5  ; allocates a byte size(8bit)constant in memory and
           ; initialize it with 5. equivalent to DC8 5
b2   DB -122
b3   DB 10110111b   ; binary value of a constant
b4   DB 0xA0        ; hexadecimal value of a constant
b5   DB 123q        ; octal value of a constant
     EVEN           ; move a location pointer to the first even address
tf   EQU 25         ; assign value to a symbol (text replacement)
w1   DW 32330       ; allocates a word size(16bit)constant in memory
                    ; Equivalent to DC16 32330
w2   DW -32000
dw1  DL 100000      ; allocates a long word size(32bit)constant in memory
                    ; equivalent to DC32 100000

dw2  DL -10000
dw3  DL 0xFFFFFFFF
dw4  DL tf
s1   DB 'ABCD'      ; allocates 4 bytes in memory with string ABCD
s2   DB "ABCD"      ; allocates 5 bytes in memory with string ABCD
                    ; and '\0' character at the end
     ORG 0x0200
v1b  DS 1           ; allocates a byte in memory, equivalent to DS8
                    ; DS=Data Space
v2b  DS 1           ; allocates a byte in memory
v3w  DS 2           ; allocates a word of 2 bytes in memory, equivalent
                    ; to DS8 2 or DS16
v4b  DS32 4         ; allocates a buffer of 4 long words,4x4=16 bytes
                    ; in memory
```

M. Code Examples:

1. נדרש לחבר בין R5+R6 → R8 (ללא שינוי הרגיסטרים R5,R6)

```
mov R5,R8
add  R6,R8
```

2. כפל בין מספרים שלמים: $var1 \cdot var2 \rightarrow var3$ (ללא מכפל חומרתי).

המשתנים $var1, var2$ באורך 16 ביט כ"א. משתנה התוצאה $var3$ בגודל 32 ביט.

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

        ORG 1100h
var1    DW   17
var2    DW   4
var3    DS32  1

        RSEG   CODE              ; ORG   0x2100 - place program in 'CODE' segment in to Flash memory

Main    MOV    var1,R4
        MOV    var2,R5
        CLR    R7
Mul     DEC    R5
        JZ     Result
        ADD    var1,R4
        ADC    R7
        JMP    Mul
Result  MOV    R4,var3
        MOV    #2,R6
        MOV    R7,var3(R6)
        JMP    $        ; infinite loop

        NOP             ; No Operation  - Required for warning cancelation
;-------------------------------------------------------------------------------
        COMMON  INTVEC          ; Interrupt Vectors
;-------------------------------------------------------------------------------
        ORG     RESET_VECTOR        ; POR, ext. Reset
        DW      Main
        END
```

3. <u>חילוק בין מספרים שלמים</u>: שארית $= R10$, מנה $= R9 \leftarrow var1 \div var2$ (נבצע $\frac{var1}{var2}$ ללא מחלק חומרתי). המשתנים $var1, var2$ באורך 16 ביט כ"א.

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

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

        RSEG   CODE              ; ORG   0x2100 - place program in 'CODE' segment in to Flash memory

Main        CLR     R9           ;R9 = Quotient Result
            MOV     var1,R4
            MOV     var2,R5
Div         CMP     R5,R4        ; var1/var2 = R4 /R5
            JLO     Result
            INC     R9
            SUB     R5,R4
            JMP     Div
Result      MOV     R9,quotient
            MOV     R4,residue
            JMP     $        ; infinite loop
            NOP              ; No Operation  - Required for warning cancelation
;-------------------------------------------------------------------------------
        COMMON  INTVEC           ; Interrupt Vectors
;-------------------------------------------------------------------------------
        ORG     RESET_VECTOR     ; POR, ext. Reset
        DW      Main
        END
```

.4 חישוב ממוצע בין 2 אברים סמוכים במערך בכתובת **Arr** בגודל **SIZE** (**ולאפס איבר אחרון**)

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

        ORG 1100h
Arr     DW   2,4,6,8,10,12,14,16,18  ; positive numbers
SIZE    DW   9                       ; positive number

        RSEG   CODE                  ; ORG   0x2100 - place program in 'CODE' segment in to Flash memory

Main    MOV    SIZE,R4
        MOV    #Arr,R5
Loop    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)
        JMP    $      ; infinite loop
        NOP           ; No Operation  - Required for warning cancelation
;-------------------------------------------------------------------------------
        COMMON  INTVEC          ; Interrupt Vectors
;-------------------------------------------------------------------------------
        ORG    RESET_VECTOR     ; POR, ext. Reset
        DW     Main
        END
```

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

העקבה של מטריצה סימטרית בגודל nxn מוגדרת ע"י סכום איברי האלכסון של המטריצה, כדלקמן:

עבור מטריצה סימטרית A_{nxn} העקבה נתונה ע"י הנוסחה: $\mathbf{Trace(A)=a_{11}+a_{22}+\ldots\ldots+a_{nn}}$

גודל כל איבר במטריצה הוא 16 סיביות. הערך המתקבל עבור עקבת המטריצה המחושבת, יש לאחסן ברגיסטרים **R12 , R13** . כאשר, החלק הנמוך LSB של העקבה יאוחסן ברגיסטר R12 והחלק הגבוה MSB של העקבה יאוחסן ברגיסטר R13 . כאשר **Matrix** הינה מטריצה ותווית זו, היא כתובת האיבר הראשון של המטריצה. המשתנה **Lines** מייצג 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).

בס"ד

- Part of Diagonal index = MatrixPTR+2j+Lines*2i
- 0<= i <=Lines-1 and 0<= j <=Lines-1 for all Parts of Diagonal i=j So, Part of Diagonal index = MatrixPTR+2i(Lines+1)
- Trace=sum of M[MatrixPTR+2i(Lines+1)] when 0<= i <=Lines-1

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

        ORG 1100h
Matrix  DC16  1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
Lines   DW    4
Trace   DS32  1

        RSEG  CODE          ; ORG  0x2100 - place program in 'CODE' segment in to Flash memory

Main    mov   #Matrix,R5
        clr   R12           ; Trace_LSB
        clr   R13           ; Trace_MSB
        mov   Lines,R6      ; R6 = Lines
        mov   Lines,R7
        inc   R6            ; R6 = Lines+1
        rla   R6            ; R6 = 2(Lines+1)
L1      add   @R5,R12
        adc   R13
        add   R6,R5         ; calculate address to R5
        dec   R7
        jnz   L1
        mov   #Trace,R5
        mov   R12,0(R5)
        mov   R13,2(R5)
        jmp   $             ; infinite loop

        NOP                 ; No Operation - Required for warning cancelation
;-------------------------------------------------------------------------------
        COMMON  INTVEC      ; Interrupt Vectors
;-------------------------------------------------------------------------------
        ORG   RESET_VECTOR  ; POR, ext. Reset
        DW    Main
        END
```

$$Matrix_{4x4} \triangleq \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

$$a_{i,j} = Matrix[Cols \cdot \mathbf{2}i + \mathbf{2}j] \; \boldsymbol{when} \; 0 \le i,j \le Lines - 1$$