מיקרו בקר C8051F380 של SILICON LABS

כתב וערך : אריה פורת

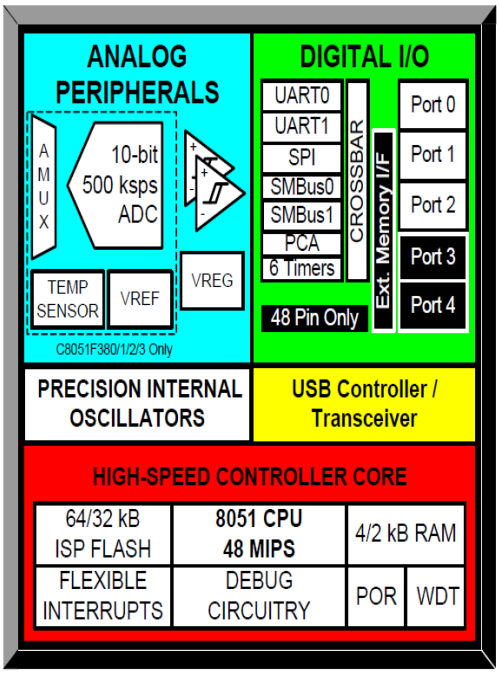
ספר לימוד לכתה יג : 4 פרקים ראשונים (על פי תוכנית הלימודים החדשה במיקרו בקרים החל מתשפ"א)

(חלק מפרק 4 - סעיפים 4.4 עד 4.6 - הוצאו מתוכנית הלימודים בתשפ"ב. הסעיפים 4.7 עד 4.13 הם חלק מפרק 5)

SILICON LABS

C8051F380/1/2/3/4/5/6/7/C

Full Speed USB Flash MCU Family



Analog Peripherals
- **10-Bit ADC (C8051F380/1/2/3/C only)**
  - Up to 500 ksps
  - Built-in analog multiplexer with single-ended and differential mode
  - VREF from external pin, internal reference, or V_{DD}
  - Built-in temperature sensor
  - External conversion start input option
- **Two comparators**
- **Internal voltage reference (C8051F380/1/2/3/C only)**
- **Brown-out detector and POR Circuitry**

USB Function Controller
- USB specification 2.0 compliant
- Full speed (12 Mbps) or low speed (1.5 Mbps) operation
- Integrated clock recovery; no external crystal required for full speed or low speed
- Supports eight flexible endpoints
- 1 kB USB buffer memory
- Integrated transceiver; no external resistors required

On-Chip Debug
- On-chip debug circuitry facilitates full speed, non-intrusive in-system debug (No emulator required)
- Provides breakpoints, single stepping, inspect/modify memory and registers
- Superior performance to emulation systems using ICE-chips, target pods, and sockets

Voltage Supply Input: 2.7 to 5.25 V
- Voltages from 2.7 to 5.25 V supported using On-Chip Voltage Regulators

High Speed 8051 µC Core
- Pipelined instruction architecture; executes 70% of instructions in 1 or 2 system clocks
- Up to 48 MIPS operation
- Expanded interrupt handler

Memory
- 4352 or 2304 Bytes RAM
- 64, 32, or 16 kB Flash; In-system programmable in 512-byte sectors

Digital Peripherals
- 40/25 Port I/O; All 5 V tolerant with high sink current
- Hardware enhanced SPI™, two I^2C/SMBus™, and two enhanced UART serial ports
- Six general purpose 16-bit counter/timers
- 16-bit programmable counter array (PCA) with five capture/compare modules
- External Memory Interface (EMIF)

Clock Sources
- Internal Oscillator: ±0.25% accuracy with clock recovery enabled. Supports all USB and UART modes
- External Oscillator: Crystal, RC, C, or clock (1 or 2 Pin modes)
- Low Frequency (80 kHz) Internal Oscillator
- Can switch between clock sources on-the-fly

Packages
- 48-pin TQFP (C8051F380/2/4/6)
- 32-pin LQFP (C8051F381/3/5/7/C)
- 5x5 mm 32-pin QFN (C8051F381/3/5/7/C)

Temperature Range: –40 to +85 °C

תוכן העניינים	עמוד

נוסחאון משרד החינוך מהבחינה במיקרו בקרים תשפ"ד (עד עמוד 41)

מדינת ישראל
משרד החינוך

סוג הבחינה: בגרות לבתי־ספר על־יסודיים
מועד הבחינה: אביב תשפ"ד, 2024
סמל השאלון: 711911

אין להעביר את הנוסחאון לנבחן אחר

נוסחאון במיקרו־בקר C8051F380 לכיתה י"ג (37 עמודים)

במטרה לשמור אחידות במרכיבים הפנימיים של הבקר, מיפוי הרגליים, האוגרים הפנימיים, שמות האוגרים והמשתנים, שאלות המבחן והנתונים במסמך זה מותאמים למיקרו־בקר C8051F380 ממשפחת Silicon Labs המבוסס על הבקר 8051 .

דף הנוסחאות מותאם למהדר Keil uVision . חלקים מדף נוסחאות זה מתאימים גם למהדרים אחרים.

שמירה על צורה אחידה למתן שמות אוגרים וטיפוסים של משתנים, יש להצהיר על ספריות הבאות:

```
#include "compiler_defs.h"
#include "C8051F380_defs.h"
```

פונקצייה לאתחול המעבד:

```
void Init_Device(void);
```

אין צורך לכתוב את מילות האתחול לאוגרים לביטול ה־watchdog וקביעת תדר הבקר, אלא להניח שהן קיימות כחלק ממימוש הקוד בפונקצייה Init_Device .

יש להניח שפונקציות השהייה במילי־שניות ומיקרו־שניות קיימות ואין צורך להצהיר עליהן או למימש אותן.

```
void delay_ms(unsigned int ms);
void delay_us(unsigned int us);
```

נוסחאון במיקרו־בקר 8051
נספח לשאלון 711911, אביב תשפ"ד

Data Types (טיפוסי נתונים)

Name	Description	תיאור	Size	Range
bit	One Bit	ביט בודד	1 bit	0 to 1
char or S8	Character or small integer.	תו בודד או בייט	1 byte	–128 to 127
unsigned char or U8	Unsigned small integer	בייט אחד ללא סימן	1 byte	0 to 255
int or S16	Integer	מספר שלם	2 bytes	–32768 to 32767
unsigned int or U16	Unsigned integer	מספר שלם ללא סימן	2 bytes	0 to 65535
long or S32	32-bit integer	מספר שלם ארוך	4 bytes	–2147483648 to 2147483647
unsigned long or U32	32-bit Unsigned integer	מספר שלם ארוך ללא סימן	4 bytes	0 to 4294967295
float	Floating point number	מספר ממשי	4 bytes	+/– 1.175494E–38 to +/– 3.402823E+38
sbit	Special Bit	ביט מיוחד	1 bit	0 to 1
sfr	Special Function Registers	בית מיוחד	1 byte	0 to 255
sfr16	16 bits special Function Registers	בית כפול מיוחד	2 bytes	0 to 65535

דוגמאות:

```
unsigned char a;
int b, c;
sfr P1=0x90;
sbit P1-7 = P1^7;
```

המשך בעמוד 3

Memory Areas (גישה לזיכרון)

Name	Description	Example
data	places the variable in directly addressable RAM in the micro core	unsigned int data dnum;
xdata	places the variable in external RAM	unsigned char xdata xnum_at_0x8000; *
idata	places the variable in indirectly addressable memory within the micro core	int idata inum;
code	places the variable in program memory	unsigned char code cnum=0xAA;

* _at_ – places the variable in absolute address.

Preprocessor-directives (הנחיות לקדם־מהדר)

Description	Syntax	Example
macro definitions	#define identifier replacement	#define LED P 1_7

identifier - מזהה ; replacement - תחליף

Operators (אופרטורים)

Description	תיאור	Operator
Assignment	השמה	=

Initalization of variables (אתחול משתנים)

```
unsigned char  d=0;
d=75;        // decimal  number
d=0x4B;      // hexadecimal  number
```

Arithmetic Operators (אופרטורים חשבוניים)

Description	תיאור	Operator
addition	חיבור	+
subtraction	חיסור	−
multiplication	כפל	*
division	חילוק	/
modulo	שארית	%

המשך בעמוד 4

Relational and equality operators (אופרטורים להשוואה ויחסים)

Description	תיאור	Operator
Equal to	שווה	= =
Not equal to	שונה	! =
Greater than	גדול מ־	>
Less than	קטן מ־	<
Greater than or equal to	גדול מ־/ שווה	> =
Less than or equal to	קטן מ־/ שווה	< =

Logical operators (אופרטורים לוגיים בין ביטויים)

Description	תיאור	Operator
NOT	היפוך	!
AND	וגם	&&
OR	או	| |

Bitwise Operators (אופרטורים על סיביות)

Description	תיאור	Operator
AND	וגם	&
Inclusive OR	או כולל	|
Exclusive OR (XOR)	או מוציא	^
Byte inversion	היפוך בית	~
Bit inversion	היפוך סיבית	!
Shift Left	הזזה שמאלה	<<
Shift Right	הזזה ימינה	>>

המשך בעמוד 5

Conditional Structures (מבני בקרה – משפטי תנאי)

Description	Syntax	Example

if	if (condition)
	statements
		if (d == 100) {
		P1 = 0xFF;
		}

if else	if (condition)
	statement1
	else
	statement2
		if (d == 100)
		P1 = 0xFF;
		else
		P1=0;

if else if else	if (condition)
	statement1
	else if (condition)
	statement2
	else
	statement3
		if (d > 0)
		P1=4;
		else if (d < 0)
		P1=2;
		else
		P1=1;

* condition (תנאי) * statement (הצהרה)

Iteration Structures (מבני בקרה - לולאות)

Description	Syntax	Example

while loop	while (expression) statements	while (n>0) {
		P1=n;
		n– –;
		}

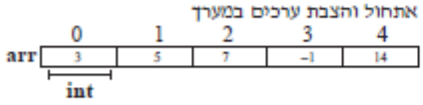
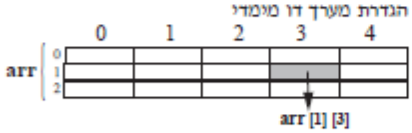
do-while loop	do statements while (condition);	do {
		n=P1;
		} while (n != 0);

for loop	for (initialization; condition;
	increase) statements ;
		for (i=0; i<10; i++) {
		P1=i;
		}

* condition (תנאי) * statement (הצהרה)

המשך בעמוד 6

Arrays (מערכים)

Description	Syntax	Example
הגדרת מערך חד מימדי 	type name [elements];	char arr[5];
אתחול והצבת ערכים במערך 	type name [elements] = {value1,..valueu};	int arr[5] = {3,5,7,–1, 14};
הגדרת מערך דו מימדי 	type name [elements, elements];	int arr[3][5];

* elements (פרטים) * value (ערך)

Structure of a program (מבנה כללי של תוכנית)

```
#include "compiler_defs.h"

#include "C8051F380_defs.h"


void Init_Device(void);

void delay_ms(unsigned int ms);

void delay_us(unsigned int us);
void main()


{
        Init_Device();

        while(1)

        {

                //Your code.

        }
```

המשך בעמוד 7

Functions (פונקציות)

Description	Syntax	Example

Functions with no type and no argument	void name (void)
	{
	statements;
	
	}
		void OutData(void)
		{
		P1=0xAB;
		}
		void main()
		{
		OutData();
		}

Functions with no type	void name (parameter1, parameter2, ...)
	{
	statements;
	
	}
		void OutData(unsigned char a, unsigned char b)
		{
		P0=a;
		P1=b;
		}
		void main()
		{
		OutData(0xF,0xF0);
		}

Functions with type and argument	type name (parameter1, parameter2, ...)
	{
	statements;
	
	}
		unsigned char InOutData(unsigned char a)
		{
		P1=a;
		return P0;
		}
		void main()
		{
		unsigned char r;
		r = InOutData(63);
		P2=r;
		}

parameter – ערך המועבר לפונקציה ; statement – הצהרה ; type – טיפוס ; argument – טיעון

המשך בעמוד 8

Interrput Service Routines (שגרות לטיפול בפסיקות)

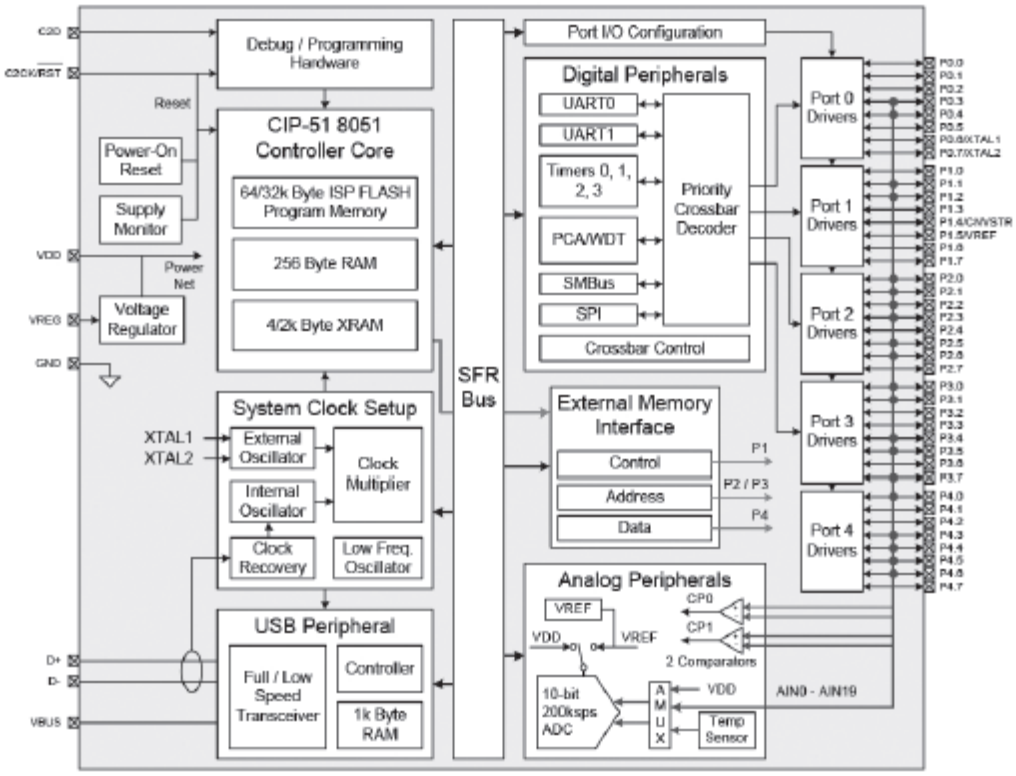
Interrupt Source	Interrupt Vector	Priority Order	Pending Flag	Bit addressable?	Cleared by HW?	Enable Flag	Priority Control
Reset	0x0000	Top	None	N/A	N/A	Always Enabled	Always Highest
External Interrupt 0 ($\overline{INT0}$)	0x0003	0	IE0 (TCON.1)	Y	Y	EX0 (IE.0)	PX0 (IP.0)
Timer 0 Overflow	0x000B	1	TF0 (TCON.5)	Y	Y	ET0 (IE.1)	PT0 (IP.1)
External Interrupt 1 ($\overline{INT1}$)	0x0013	2	IE1 (TCON.3)	Y	Y	EX1 (IE.2)	PX1 (IP.2)
Timer 1 Overflow	0x001B	3	TF1 (TCON.7)	Y	Y	ET1 (IE.3)	PT1 (IP.3)
UART0	0x0023	4	RI0 (SCON0.0) TI0 (SCON0.1)	Y	N	ES0 (IE.4)	PS0 (IP.4)
Timer 2 Overflow	0x002B	5	TF2H (TMR2CN.7) TF2L (TMR2CN.6)	Y	N	ET2 (IE.5)	PT2 (IP.5)
SPI0	0x0033	6	SPIF (SPI0CN.7) WCOL (SPI0CN.6) MODF (SPI0CN.5) RXOVRN (SPI0CN.4)	Y	N	ESPI0 (IE.6)	PSPI0 (IP.6)
SMB0	0x003B	7	SI (SMB0CN.0)	Y	N	ESMB0 (EIE1.0)	PSMB0 (EIP1.0)
USB0	0x0043	8	Special	N	N	EUSB0 (EIE1.1)	PUSB0 (EIP1.1)
ADC0 Window Compare	0x004B	9	AD0WINT (ADC0CN.3)	Y	N	EWADC0 (EIE1.2)	PWADC0 (EIP1.2)
ADC0 Conversion Complete	0x0053	10	AD0INT (ADC0CN.5)	Y	N	EADC0 (EIE1.3)	PADC0 (EIP1.3)

דוגמה:

```
void int2sub () interrupt 2 {
      // Interrupt code
}
```

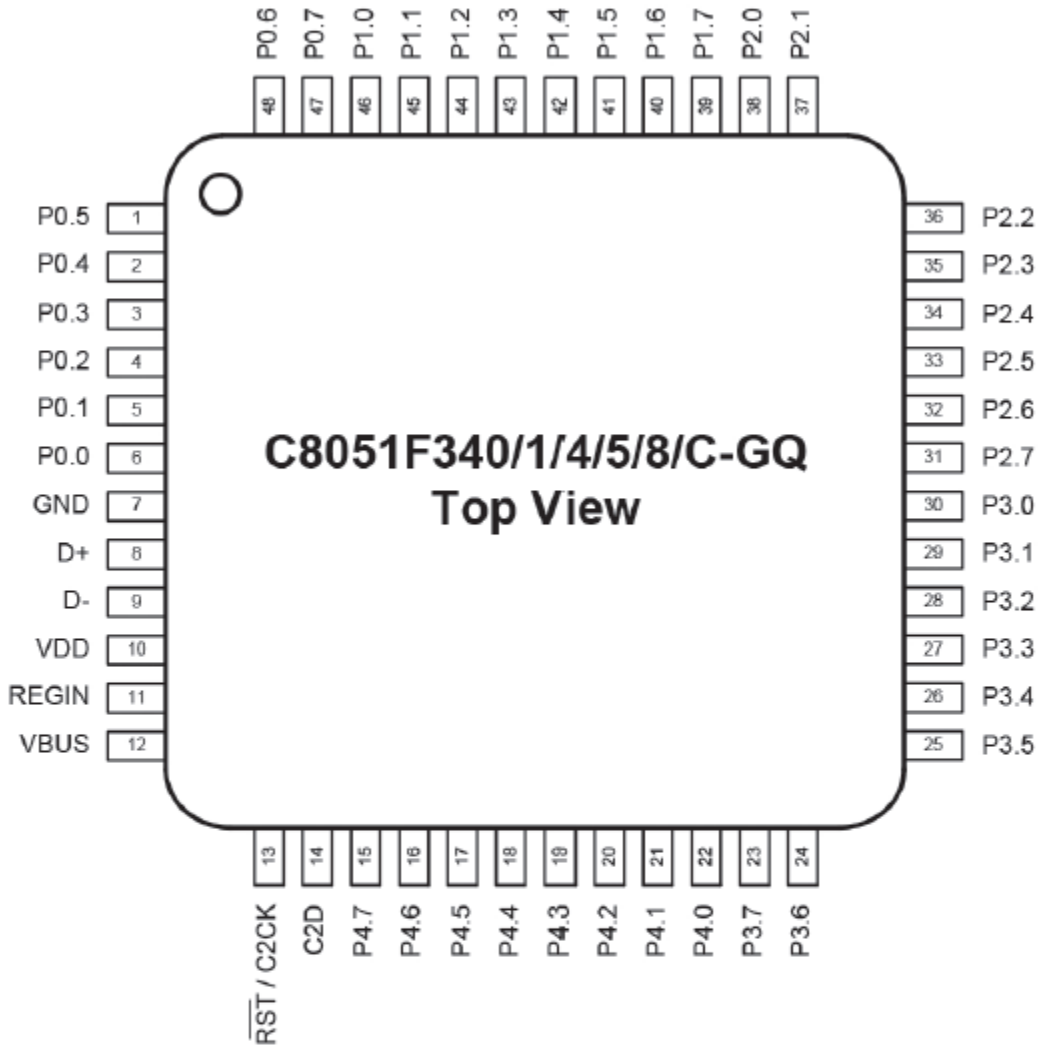
המשך בעמוד 9

C8051F380 Block Diagram



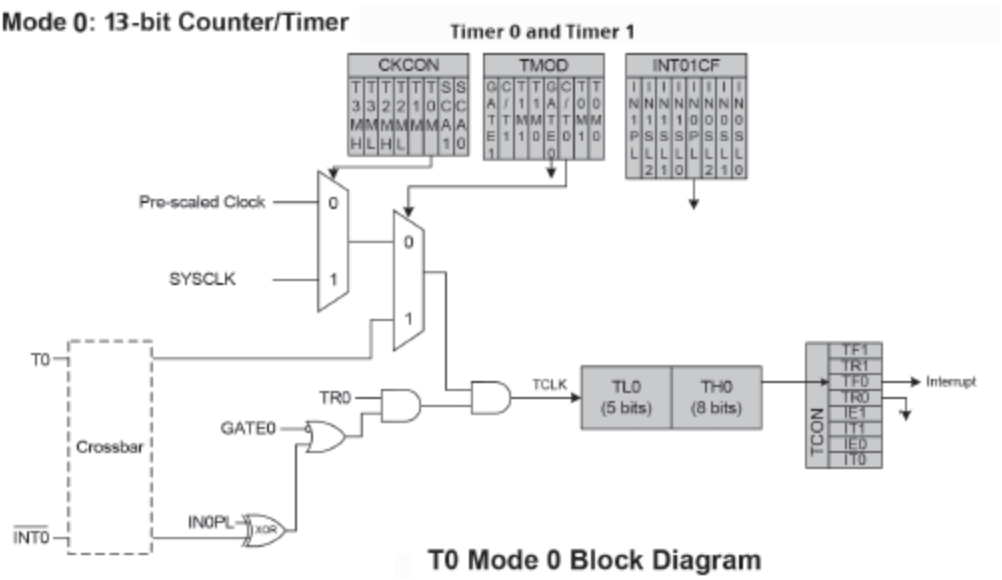
המשך בעמוד 10

TQFP – 48 Pinout Diagram (Top View)



המשך בעמוד 11

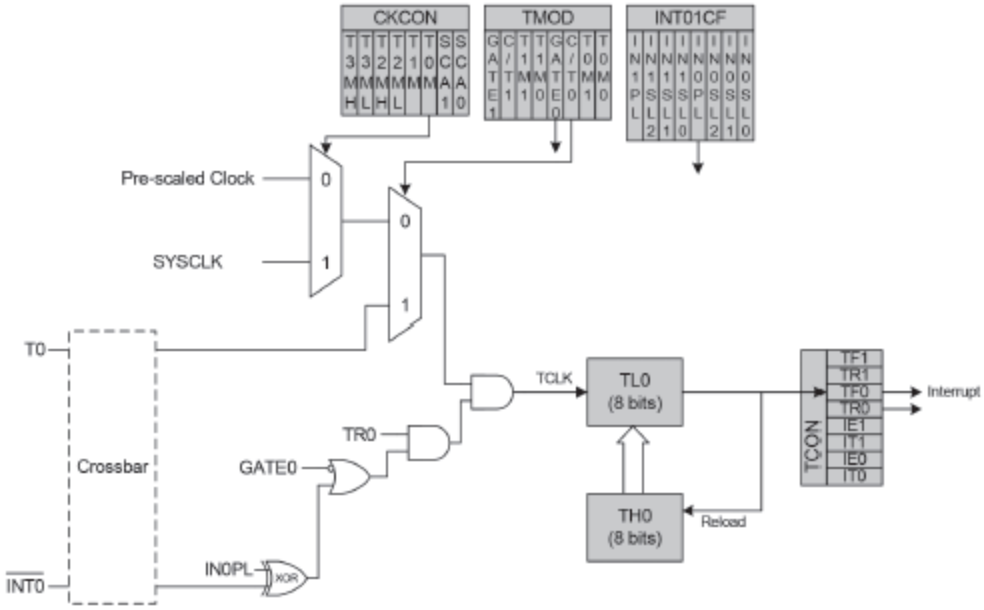
Mode 0: 13-bit Counter/Timer



T0 Mode 0 Block Diagram

Mode 1 operation is the same as Mode 0, except that the counter/timer registers use all 16 bits. The counter/timers are enabled and configured in Mode 1 in the same manner as for Mode 0.

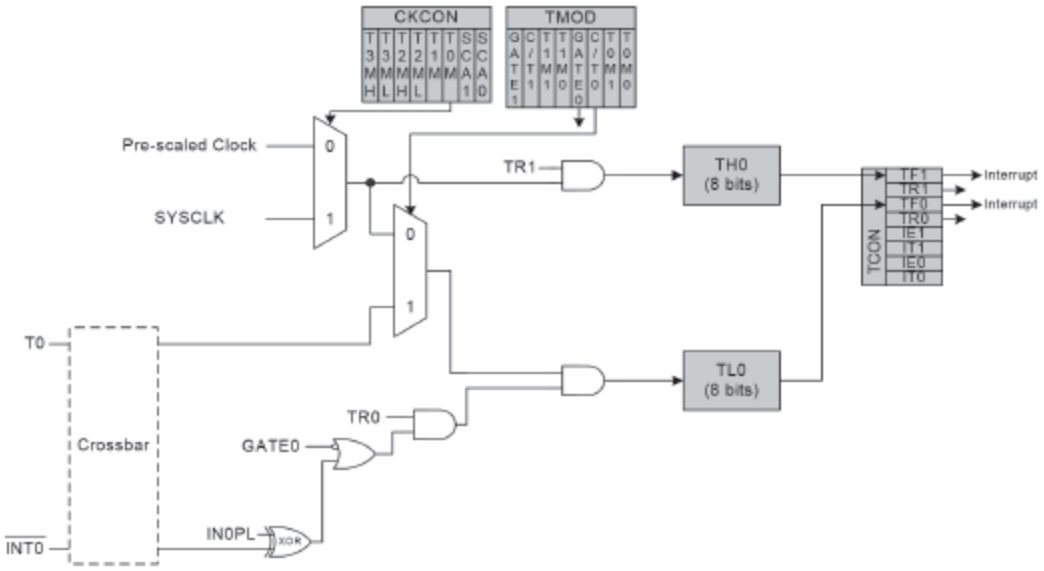
Mode 2: 8-bit Counter/Timer with Auto-Reload



T0 Mode 2 Block Diagram

המשך בעמוד 12

Mode 3: Two 8-bit Counter/Timers (Timer 0 Only)



T0 Mode 3 Block Diagram

המשך בעמוד 13

CKCON: Clock Control

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
T3MH	T3ML	T2MH	T2ML	T1M	T0M	SCA1	SCA0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0x8E

Bit7: T3MH: Timer 3 High Byte Clock Select.
This bit selects the clock supplied to the Timer 3 high byte if Timer 3 is configured in split 8-bit timer mode. T3MH is ignored if Timer 3 is in any other mode.
0: Timer 3 high byte uses the clock defined by the T3XCLK bit in TMR3CN.
1: Timer 3 high byte uses the system clock.

Bit6: T3ML: Timer 3 Low Byte Clock Select.
This bit selects the clock supplied to Timer 3. If Timer 3 is configured in split 8-bit timer mode, this bit selects the clock supplied to the lower 8-bit timer.
0: Timer 3 low byte uses the clock defined by the T3XCLK bit in TMR3CN.
1: Timer 3 low byte uses the system clock.

Bit5: T2MH: Timer 2 High Byte Clock Select.
This bit selects the clock supplied to the Timer 2 high byte if Timer 2 is configured in split 8-bit timer mode. T2MH is ignored if Timer 2 is in any other mode.
0: Timer 2 high byte uses the clock defined by the T2XCLK bit in TMR2CN.
1: Timer 2 high byte uses the system clock.

Bit4: T2ML: Timer 2 Low Byte Clock Select.
This bit selects the clock supplied to Timer 2. If Timer 2 is configured in split 8-bit timer mode, this bit selects the clock supplied to the lower 8-bit timer.
0: Timer 2 low byte uses the clock defined by the T2XCLK bit in TMR2CN.
1: Timer 2 low byte uses the system clock.

Bit3: T1M: Timer 1 Clock Select.
This select the clock source supplied to Timer 1. T1M is ignored when C/T1 is set to logic 1.
0: Timer 1 uses the clock defined by the prescale bits, SCA1-SCA0.
1: Timer 1 uses the system clock.

Bit2: T0M: Timer 0 Clock Select.
This bit selects the clock source supplied to Timer 0. T0M is ignored when C/T0 is set to logic 1.
0: Counter/Timer 0 uses the clock defined by the prescale bits, SCA1-SCA0.
1: Counter/Timer 0 uses the system clock.

Bits1–0: SCA1-SCA0: Timer 0/1 Prescale Bits.
These bits control the division of the clock supplied to Timer 0 and/or Timer 1 if configured to use prescaled clock inputs.

* parameter (ערך המועבר לפונקציה) * statement (הצהרה)

המשך בעמוד 14

TMOD: Timer Mode

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
GATE1	C/T1	T1M1	T1M0	GATE0	C/T0	T0M1	T0M0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0x89

Bit7: GATE1: Timer 1 Gate Control.
0: Timer 1 enabled when TR1 = 1 irrespective of $\overline{INT1}$ logic level.
1: Timer 1 enabled only when TR1 = 1 AND $\overline{INT1}$ is active as defined by bit IN1PL in register INT01CF (see SFR Definition 9.13).

Bit6: C/T1: Counter/Timer 1 Select.
0: Timer Function: Timer 1 incremented by clock defined by T1M bit (CKCON.3).
1: Counter Function: Timer 1 incremented by high-to-low transitions on external input pin (T1).

Bits5–4: T1M1–T1M0: Timer 1 Mode Select.
These bits select the Timer 1 operation mode.

T1M1	T1M0	Mode
0	0	Mode 0: 13-bit counter/timer
0	1	Mode 1: 16-bit counter/timer
1	0	Mode 2: 8-bit counter/timer with auto-reload
1	1	Mode 3: Timer 1 inactive

Bit3: GATE0: Timer 0 Gate Control.
0: Timer 0 enabled when TR0 = 1 irrespective of $\overline{INT0}$ logic level.
1: Timer 0 enabled only when TR0 = 1 AND $\overline{INT0}$ is active as defined by bit IN0PL in register INT01CF (see SFR Definition 9.13).

Bit2: C/T0: Counter/Timer Select.
0: Timer Function: Timer 0 incremented by clock defined by T0M bit (CKCON.2).
1: Counter Function: Timer 0 incremented by high-to-low transitions on external input pin (T0).

Bits1–0: T0M1–T0M0: Timer 0 Mode Select.
These bits select the Timer 0 operation mode.

T0M1	T0M0	Mode
0	0	Mode 0: 13-bit counter/timer
0	1	Mode 1: 16-bit counter/timer
1	0	Mode 2: 8-bit counter/timer with auto-reload
1	1	Mode 3: Two 8-bit counter/timers

המשך בעמוד 15

TCON: Timer Control

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
							(bit addressable)	0x88

Bit7: TF1: Timer 1 Overflow Flag.
Set by hardware when Timer 1 overflows. This flag can be cleared by software but is automatically cleared when the CPU vectors to the Timer 1 interrupt service routine.
0: No Timer 1 overflow detected.
1: Timer 1 has overflowed.

Bit6: TR1: Timer 1 Run Control.
0: Timer 1 disabled.
1: Timer 1 enabled.

Bit5: TF0: Timer 0 Overflow Flag.
Set by hardware when Timer 0 overflows. This flag can be cleared by software but is automatically cleared when the CPU vectors to the Timer 0 interrupt service routine.
0: No Timer 0 overflow detected.
1: Timer 0 has overflowed.

Bit4: TR0: Timer 0 Run Control.
0: Timer 0 disabled.
1: Timer 0 enabled.

Bit3: IE1: External Interrupt 1.
This flag is set by hardware when an edge/level of type defined by IT1 is detected. It can be cleared by software but is automatically cleared when the CPU vectors to the External Interrupt 1 service routine if IT1 = 1. When IT1 = 0, this flag is set to '1' when $\overline{\text{INT1}}$ is active as defined by bit IN1PL in register INT01CF (see SFR Definition 9.13).

Bit2: IT1: Interrupt 1 Type Select.
This bit selects whether the configured $\overline{\text{INT1}}$ interrupt will be edge or level sensitive. $\overline{\text{INT1}}$ is configured active low or high by the IN1PL bit in the IT01CF register (see SFR Definition 9.13).
0: $\overline{\text{INT1}}$ is level triggered.
1: $\overline{\text{INT1}}$ is edge triggered.

Bit1: IE0: External Interrupt 0.
This flag is set by hardware when an edge/level of type defined by IT0 is detected. It can be cleared by software but is automatically cleared when the CPU vectors to the External Interrupt 0 service routine if IT0 = 1. When IT0 = 0, this flag is set to '1' when $\overline{\text{INT0}}$ is active as defined by bit IN0PL in register INT01CF (see SFR Definition 9.13).

Bit0: IT0: Interrupt 0 Type Select.
This bit selects whether the configured $\overline{\text{INT0}}$ interrupt will be edge or level sensitive. $\overline{\text{INT0}}$ is configured active low or high by the IN0PL bit in register IT01CF (see SFR Definition 9.13).
0: $\overline{\text{INT0}}$ is level triggered.
1: $\overline{\text{INT0}}$ is edge triggered.

המשך בעמוד 16

SFR Definition 21.4. TL0: Timer 0 Low Byte

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0x8A

Bits 7–0: TL0: Timer 0 Low Byte.
The TL0 register is the low byte of the 16-bit Timer 0.

SFR Definition 21.5. TL1: Timer 1 Low Byte

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0x8B

Bits 7–0: TL1: Timer 1 Low Byte.
The TL1 register is the low byte of the 16-bit Timer 1.

SFR Definition 21.6. TH0: Timer 0 High Byte

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0x8C

Bits 7–0: TH0: Timer 0 High Byte.
The TH0 register is the high byte of the 16-bit Timer 0.

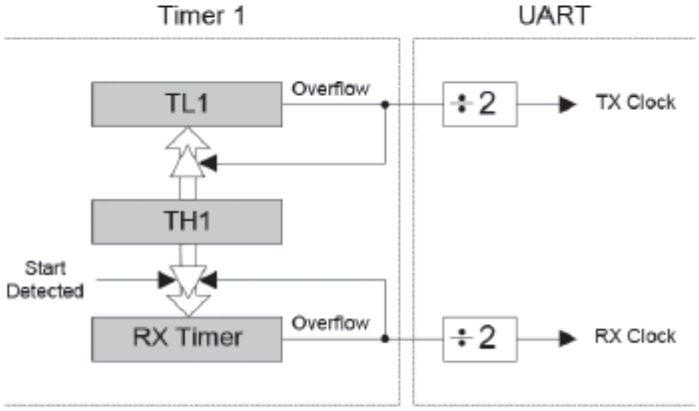
SFR Definition 21.7. TH1: Timer 1 High Byte

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0x8D

Bits 7–0: TH1: Timer 1 High Byte.
The TH1 register is the high byte of the 16-bit Timer 1.

המשך בעמוד 17

UART0



UARTO Baud Rate Logic

דוגמה:

A) $\text{UARTBaudRate} = \frac{1}{2} \times \text{T1\_Overflow\_Rate}$

B) $\text{T1\_Overflow\_Rate} = \frac{\text{T1}_{\text{CLK}}}{256 - \text{TH1}}$

9-Bit UART Timing Diagram

המשך בעמוד 18

SCON0: Serial Port 0 Control

R/W	R	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
S0MODE	-	MCE0	REN0	TB80	RB80	TI0	RI0	01000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit Addressable

SFR Address: 0x98

Bit7: S0MODE: Serial Port 0 Operation Mode.
This bit selects the UART0 Operation Mode.
0: 8-bit UART with Variable Baud Rate.
1: 9-bit UART with Variable Baud Rate.

Bit6: UNUSED. Read = 1b. Write = don't care.

Bit5: MCE0: Multiprocessor Communication Enable.
The function of this bit is dependent on the Serial Port 0 Operation Mode.
S0MODE = 0: Checks for valid stop bit.
0: Logic level of stop bit is ignored.
1: RI0 will only be activated if stop bit is logic level 1.
S0MODE = 1: Multiprocessor Communications Enable.
0: Logic level of ninth bit is ignored.
1: RI0 is set and an interrupt is generated only when the ninth bit is logic 1.

Bit4: REN0: Receive Enable.
This bit enables/disables the UART receiver.
0: UART0 reception disabled.
1: UART0 reception enabled.

Bit3: TB80: Ninth Transmission Bit.
The logic level of this bit will be assigned to the ninth transmission bit in 9-bit UART Mode. It is not used in 8-bit UART Mode. Set or cleared by software as required.

Bit2: RB80: Ninth Receive Bit.
RB80 is assigned the value of the STOP bit in Mode 0; it is assigned the value of the 9th data bit in Mode 1.

Bit1: TI0: Transmit Interrupt Flag.
Set by hardware when a byte of data has been transmitted by UART0 (after the 8th bit in 8-bit UART Mode, or at the beginning of the STOP bit in 9-bit UART Mode). When the UART0 interrupt is enabled, setting this bit causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software.

Bit0: RI0: Receive Interrupt Flag.
Set to '1' by hardware when a byte of data has been received by UART0 (set at the STOP bit sampling time). When the UART0 interrupt is enabled, setting this bit to '1' causes the CPU to vector to the UART0 interrupt service routine. This bit must be cleared manually by software.

המשך בעמוד 19

SBUF0: Serial (UART0) Port Data Buffer

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

SFR Address: 0x99

Bits7–0: SBUF0[7:0]: Serial Data Buffer Bits 7–0 (MSB-LSB)
This SFR accesses two registers; a transmit shift register and a receive latch register. When data is written to SBUF0, it goes to the transmit shift register and is held for serial transmission. Writing a byte to SBUF0 initiates the transmission. A read of SBUF0 returns the contents of the receive latch.

	Target Baud Rate (bps)	Actual Baud Rate (bps)	Baud Rate Error	Oscillator Divide Factor	Timer Clock Source	SCA1-SCA0 (pre-scale select*	T1M*	Timer 1 Reload Value (hex)
SYSCLK = 12 MHz	230400	230769	0.16%	52	SYSCLK	XX	1	0xE6
	115200	115385	0.16%	104	SYSCLK	XX	1	0xCC
	57600	57692	0.16%	208	SYSCLK	XX	1	0x98
	28800	28846	0.16%	416	SYSCLK	XX	1	0x30
	14400	14423	0.16%	832	SYSCLK / 4	01	0	0x98
	9600	9615	0.16%	1248	SYSCLK / 4	01	0	0x64
	2400	2404	0.16%	4992	SYSCLK / 12	00	0	0x30
	1200	1202	0.16%	9984	SYSCLK / 48	10	0	0x98
SYSCLK = 24 MHz	230400	230769	0.16%	104	SYSCLK	XX	1	0xCC
	115200	115385	0.16%	208	SYSCLK	XX	1	0x98
	57600	57692	0.16%	416	SYSCLK	XX	1	0x30
	28800	28846	0.16%	832	SYSCLK / 4	01	0	0x98
	14400	14423	0.16%	1664	SYSCLK / 4	01	0	0x30
	9600	9615	0.16%	2496	SYSCLK / 12	00	0	0x98
	2400	2404	0.16%	9984	SYSCLK / 48	10	0	0x98
	1200	1202	0.16%	19968	SYSCLK / 48	10	0	0x30
SYSCLK = 48 MHz	230400	230769	0.16%	208	SYSCLK	XX	1	0x98
	115200	115385	0.16%	416	SYSCLK	XX	1	0x30
	57600	57692	0.16%	832	SYSCLK / 4	01	0	0x98
	28800	28846	0.16%	1664	SYSCLK / 4	01	0	0x30
	14400	14388	0.08%	3336	SYSCLK / 12	00	0	0x75
	9600	9615	0.16%	4992	SYSCLK / 12	00	0	0x30
	2400	2404	0.16%	19968	SYSCLK / 48	10	0	0x30

המשך בעמוד 20

IE: Interrupt Enable

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
EA	ESPI0	ET2	ES0	ET1	EX1	ET0	EX0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
							(bit addressable)	0xA8

Bit7: EA: Enable All Interrupts.
This bit globally enables/disables all interrupts. It overrides the individual interrupt mask settings.
0: Disable all interrupt sources.
1: Enable each interrupt according to its individual mask setting.

Bit6: ESPI0: Enable Serial Peripheral Interface (SPI0) Interrupt.
This bit sets the masking of the SPI0 interrupts.
0: Disable all SPI0 interrupts.
1: Enable interrupt requests generated by SPI0.

Bit5: ET2: Enable Timer 2 Interrupt.
This bit sets the masking of the Timer 2 interrupt.
0: Disable Timer 2 interrupt.
1: Enable interrupt requests generated by the TF2L or TF2H flags.

Bit4: ES0: Enable UART0 Interrupt.
This bit sets the masking of the UART0 interrupt.
0: Disable UART0 interrupt.
1: Enable UART0 interrupt.

Bit3: ET1: Enable Timer 1 Interrupt.
This bit sets the masking of the Timer 1 interrupt.
0: Disable all Timer 1 interrupt.
1: Enable interrupt requests generated by the TF1 flag.

Bit2: EX1: Enable External Interrupt 1.
This bit sets the masking of External Interrupt 1.
0: Disable external interrupt 1.
1: Enable interrupt requests generated by the $\overline{\text{INT1}}$ input.

Bit1: ET0: Enable Timer 0 Interrupt.
This bit sets the masking of the Timer 0 interrupt.
0: Disable all Timer 0 interrupt.
1: Enable interrupt requests generated by the TF0 flag.

Bit0: EX0: Enable External Interrupt 0.
This bit sets the masking of External Interrupt 0.
0: Disable external interrupt 0.
1: Enable interrupt requests generated by the $\overline{\text{INT0}}$ input.

המשך בעמוד 21

TCON: Timer Control

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
							(bit addressable)	0x88

Bit7: TF1: Timer 1 Overflow Flag.
Set by hardware when Timer 1 overflows. This flag can be cleared by software but is automatically cleared when the CPU vectors to the Timer 1 interrupt service routine.
0: No Timer 1 overflow detected.
1: Timer 1 has overflowed.

Bit6: TR1: Timer 1 Run Control.
0: Timer 1 disabled.
1: Timer 1 enabled.

Bit5: TF0: Timer 0 Overflow Flag.
Set by hardware when Timer 0 overflows. This flag can be cleared by software but is automatically cleared when the CPU vectors to the Timer 0 interrupt service routine.
0: No Timer 0 overflow detected.
1: Timer 0 has overflowed.

Bit4: TR0: Timer 0 Run Control.
0: Timer 0 disabled.
1: Timer 0 enabled.

Bit3: IE1: External Interrupt 1.
This flag is set by hardware when an edge/level of type defined by IT1 is detected. It can be cleared by software but is automatically cleared when the CPU vectors to the External Interrupt 1 service routine if IT1 = 1. When IT1 = 0, this flag is set to '1' when $\overline{INT1}$ is active as defined by bit IN1PL in register INT01CF (see SFR Definition 9.13).

Bit2: IT1: Interrupt 1 Type Select.
This bit selects whether the configured $\overline{INT1}$ interrupt will be edge or level sensitive. $\overline{INT1}$ is configured active low or high by the IN1PL bit in the IT01CF register (see SFR Definition 9.13).
0: $\overline{INT1}$ is level triggered.
1: $\overline{INT1}$ is edge triggered.

Bit1: IE0: External Interrupt 0.
This flag is set by hardware when an edge/level of type defined by IT0 is detected. It can be cleared by software but is automatically cleared when the CPU vectors to the External Interrupt 0 service routine if IT0 = 1. When IT0 = 0, this flag is set to '1' when $\overline{INT0}$ is active as defined by bit IN0PL in register INT01CF (see SFR Definition 9.13).

Bit0: IT0: Interrupt 0 Type Select.
This bit selects whether the configured $\overline{INT0}$ interrupt will be edge or level sensitive. $\overline{INT0}$ is configured active low or high by the IN0PL bit in register IT01CF (see SFR Definition 9.13).
0: $\overline{INT0}$ is level triggered.
1: $\overline{INT0}$ is edge triggered.

המשך בעמוד 23

IT01CF: $\overline{\text{INT0}}$ / $\overline{\text{INT1}}$ Configuration

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
IN1PL	IN1SL2	IN1SL1	IN1SL0	IN0PL	IN0SL2	IN0SL1	IN0SL0	00000001
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xE4

Note: Refer to SFR Definition 21.1 for INT0/1 edge- or level-sensitive interrupt selection.

Bit7: IN1PL: $\overline{\text{INT1}}$ Polarity
0: $\overline{\text{INT1}}$ input is active low.
1: $\overline{\text{INT1}}$ input is active high.

Bits6–4: IN1SL2–0: $\overline{\text{INT1}}$ Port Pin Selection Bits
These bits select which Port pin is assigned to $\overline{\text{INT1}}$. Note that this pin assignment is independent of the Crossbar; $\overline{\text{INT1}}$ will monitor the assigned Port pin without disturbing the peripheral that has been assigned the Port pin via the Crossbar. The Crossbar will not assign the Port pin to a peripheral if it is configured to skip the selected pin (accomplished by setting to '1' the corresponding bit in register P0SKIP).

IN1SL2–0	$\overline{\text{INT1}}$ Port Pin
000	P0.0
001	P0.1
010	P0.2
011	P0.3
100	P0.4
101	P0.5
110	P0.6
111	P0.7

Bit3: IN0PL: $\overline{\text{INT0}}$ Polarity
0: $\overline{\text{INT0}}$ interrupt is active low.
1: $\overline{\text{INT0}}$ interrupt is active high.

Bits2–0: INT0SL2–0: $\overline{\text{INT0}}$ Port Pin Selection Bits
These bits select which Port pin is assigned to $\overline{\text{INT0}}$. Note that this pin assignment is independent of the Crossbar. $\overline{\text{INT0}}$ will monitor the assigned Port pin without disturbing the peripheral that has been assigned the Port pin via the Crossbar. The Crossbar will not assign the Port pin to a peripheral if it is configured to skip the selected pin (accomplished by setting to '1' the corresponding bit in register P0SKIP).

IN0SL2–0	$\overline{\text{INT0}}$ Port Pin
000	P0.0
001	P0.1
010	P0.2
011	P0.3
100	P0.4
101	P0.5
110	P0.6
111	P0.7

IP: Interrupt Priority

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
-	PSPI0	PT2	PS0	PT1	PX1	PT0	PX0	10000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
							(bit addressable)	0xB8

Bit7: UNUSED. Read = 1, Write = don't care.
Bit6: PSPI0: Serial Peripheral Interface (SPI0) Interrupt Priority Control.
This bit sets the priority of the SPI0 interrupt.
0: SPI0 interrupt set to low priority level.
1: SPI0 interrupt set to high priority level.
Bit5: PT2: Timer 2 Interrupt Priority Control.
This bit sets the priority of the Timer 2 interrupt.
0: Timer 2 interrupt set to low priority level.
1: Timer 2 interrupts set to high priority level.
Bit4: PS0: UART0 Interrupt Priority Control.
This bit sets the priority of the UART0 interrupt.
0: UART0 interrupt set to low priority level.
1: UART0 interrupts set to high priority level.
Bit3: PT1: Timer 1 Interrupt Priority Control.
This bit sets the priority of the Timer 1 interrupt.
0: Timer 1 interrupt set to low priority level.
1: Timer 1 interrupts set to high priority level.
Bit2: PX1: External Interrupt 1 Priority Control.
This bit sets the priority of the External Interrupt 1 interrupt.
0: External Interrupt 1 set to low priority level.
1: External Interrupt 1 set to high priority level.
Bit1: PT0: Timer 0 Interrupt Priority Control.
This bit sets the priority of the Timer 0 interrupt.
0: Timer 0 interrupt set to low priority level.
1: Timer 0 interrupt set to high priority level.
Bit0: PX0: External Interrupt 0 Priority Control.
This bit sets the priority of the External Interrupt 0 interrupt.
0: External Interrupt 0 set to low priority level.
1: External Interrupt 0 set to high priority level.

המשך בעמוד 24

Interrput Service Routines (שגרות לטיפול בפסיקות)

Interrupt Source	Interrupt Vector	Priority Order	Pending Flag	Bit Address?	Cleared by HW?	Enable Flag	Priority Control
Reset	0x0000	Top	None	N/A	N/A	Always Enabled	Always Highest
External Interrupt 0 ($\overline{INT0}$)	0x0003	0	IE0(TCON.1)	Y	Y	EX0(IE.0)	Px0(IP.0)
Timer 0 Overflow	0x000B	1	TF0(TCON.5)	Y	Y	ET0(IE.1)	PT0(IP.1)
External Interrupt 1 ($\overline{INT1}$)	0x0013	2	IE1(TCON.3)	Y	Y	EX1(IE.2)	PX1(IP.2)
Timer 1 Overflow	0x001B	3	TF1(TCON.7)	Y	Y	ET1(IE.3)	PT1(IP.3)
UART0	0x0023	4	RI0 (SCON.0) TI0 (SCON.1)	Y	N	ES0(IE.4)	PS0(IP.4)
Timer 2 Overflow	0x002B	5	TF2H(TMR2CN.7) TF2L(TMR2CN.6)	Y	N	ET2(IE.5)	PT2(IP.5)
SPI0	0x0033	6	SPIF (SPI0CN.7) WCOL (SPI0CN.6) MODF (SPI0CN.5) RXOVRN (SPI0CN.4)	Y	N	ESPI0(IE.6)	PSPI0(IP.6)
SMB0	0x003B	7	SI (SMB0CN.0)	Y	N	ESMB0 (EIE1.0)	PSMB0 (EIP1.0)
USB0	0x0043	8	Special	N	N	EUSB0 (EIE1.1)	PUSB0 (EIP1.1)
ADC0 Window Com-pare	0x004B	9	AD0WINT (ADC0CN.3)	Y	N	EWADC0 (EIE1.2)	PWADC0 (EIP1.2)
ADC0 Conversion Complete	0x0053	10	AD0INT (ADC0CN.5)	Y	N	EADC0 (EIE1.3)	PADC0 (EIP1.3)

דוגמה:

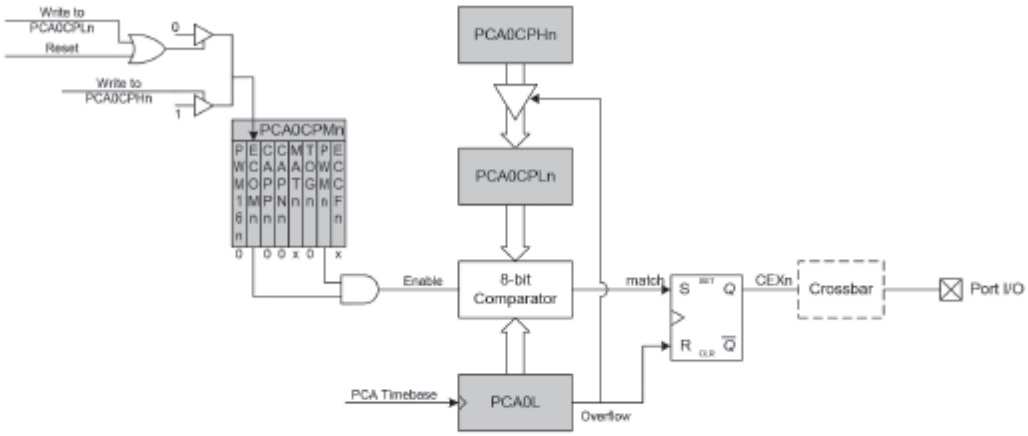
```
void int2sub () interrupt 2 {
      // Interrupt code
}
```

המשך בעמוד 25

Bit Pulse Width Modulator Mode

$$DutyCycle = \frac{(256 - PCA0CPHn)}{256}$$

Equation 8-Bit PWM Duty Cycle

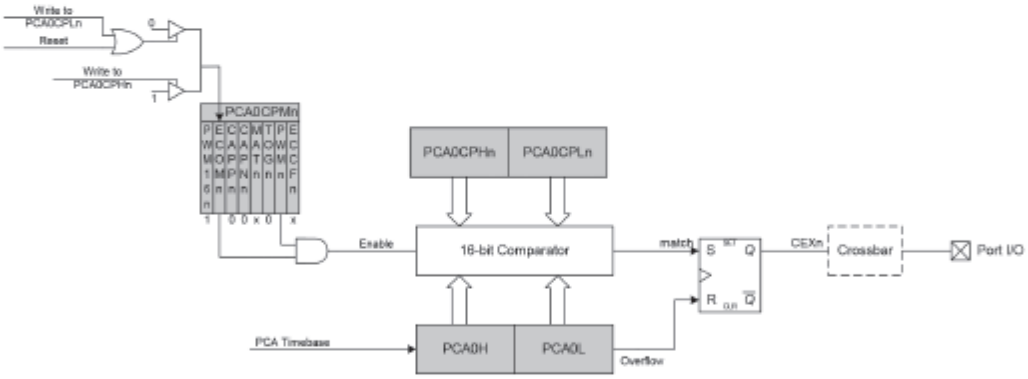


PCA 8-Bit PWM Mode Diagram

16-Bit Pulse Width Modulator Mode

$$DutyCycle = \frac{(65536 - PCA0CPn)}{65536}$$

Equation 16-Bit PWM Duty Cycle



PCA 16-Bit PWM Mode

PCA0CN: PCA Control

המשך בעמוד 26

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
CF	CR	-	CCF4	CCF3	CCF2	CCF1	CCF0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
							(bit addressable)	0xD8

Bit7: CF: PCA Counter/Timer Overflow Flag.
Set by hardware when the PCA Counter/Timer overflows from 0xFFFF to 0x0000. When the Counter/Timer Overflow (CF) interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

Bit6: CR: PCA Counter/Timer Run Control.
This bit enables/disables the PCA Counter/Timer.
0: PCA Counter/Timer disabled.
1: PCA Counter/Timer enabled.

Bit5: UNUSED. Read = 0b, Write = don't care.

Bit4: CCF4: PCA Module 4 Capture/Compare Flag.
This bit is set by hardware when a match or capture occurs. When the CCF4 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

Bit3: CCF3: PCA Module 3 Capture/Compare Flag.
This bit is set by hardware when a match or capture occurs. When the CCF3 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

Bit2: CCF2: PCA Module 2 Capture/Compare Flag.
This bit is set by hardware when a match or capture occurs. When the CCF2 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

Bit1: CCF1: PCA Module 1 Capture/Compare Flag.
This bit is set by hardware when a match or capture occurs. When the CCF1 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

Bit0: CCF0: PCA Module 0 Capture/Compare Flag.
This bit is set by hardware when a match or capture occurs. When the CCF0 interrupt is enabled, setting this bit causes the CPU to vector to the PCA interrupt service routine. This bit is not automatically cleared by hardware and must be cleared by software.

המשך בעמוד 27

PCA0CPM Bit Setting for PCA Capture/Compare Modules

PWM16	ECOM	CAPP	CAPN	MAT	TOG	PWM	ECCF	Operation Mode
X	X	1	0	0	0	0	X	Capture triggered by positive edge on CEXn
X	X	0	1	0	0	0	X	Capture triggered by negative edge on CEXn
X	X	1	1	0	0	0	X	Capture triggered by transition on CEXn
X	1	0	0	1	0	0	X	Software Timer
X	1	0	0	1	1	0	X	High Speed Output
X	1	0	0	X	1	1	X	Frequency Output
0	1	0	0	X	0	1	X	8-Bit Pulse Width Modulator
1	1	0	0	X	0	1	X	16-Bit Pulse Width Modulator

SFR Definition 22.6. PCA0CPLn: PCA Capture Module Low Byte

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xFB, 0xE9, 0xEB, 0xED, 0xFD

PCA0CPLn Address: PCA0CPL0 = 0xFB (n = 0), PCA0CPL1 = 0xE9 (n = 1), PCA0CPL2 = 0xEB (n = 2), PCA0CPL3 = 0xED (n = 3), PCA0CPL4 = 0xFD (n = 4)

Bits7–0: PCA0CPLn: PCA Capture Module Low Byte.
The PCA0CPLn register holds the low byte (LSB) of the 16-bit capture module n.

SFR Definition 22.7. PCA0CPHn: PCA Capture Module High Byte

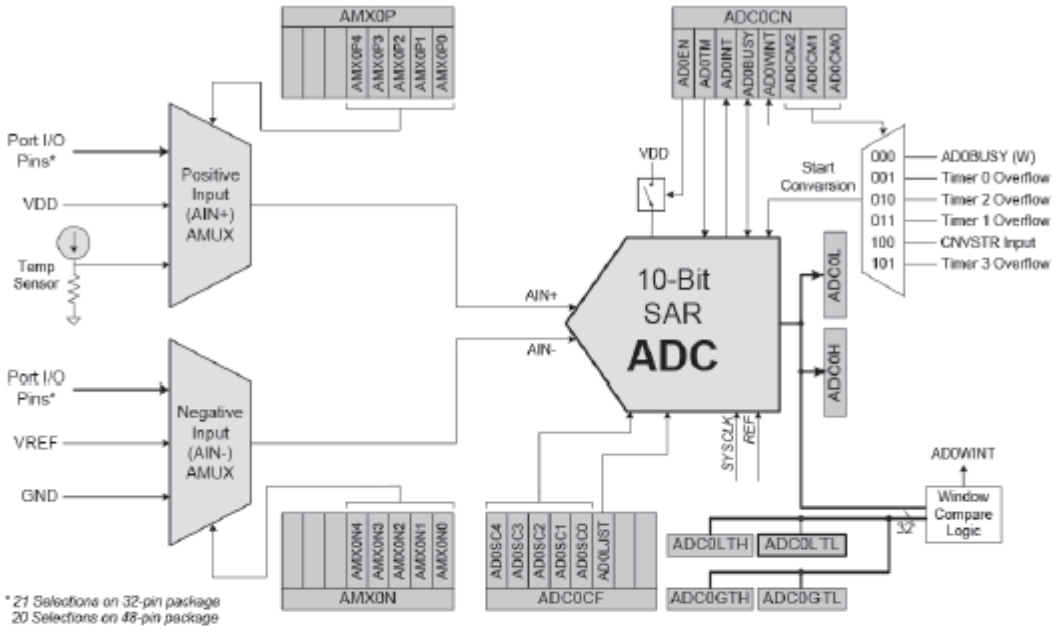
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xFC, 0xEA, 0xEC, 0xEE, 0xFE

PCA0CPHn Address: PCA0CPH0 = 0xFC (n = 0), PCA0CPH1 = 0xEA (n = 1), PCA0CPH2 = 0xEC (n = 2), PCA0CPH3 = 0xEE (n = 3), PCA0CPH4 = 0xFE (n = 4)

Bits7–0: PCA0CPHn: PCA Capture Module High Byte.
The PCA0CPHn register holds the high byte (MSB) of the 16-bit capture module n.

המשך בעמוד 28

10–Bit ADC



ADC0 Functional Block Diagram

המשך בעמוד 29

ADC0CF: ADC0 Configuration

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

SFR Address: 0x99

Bits7–0: SBUF0[7:0]: Serial Data Buffer Bits 7–0 (MSB-LSB)
This SFR accesses two registers; a transmit shift register and a receive latch register. When data is written to SBUF0, it goes to the transmit shift register and is held for serial transmission. Writing a byte to SBUF0 initiates the transmission. A read of SBUF0 returns the contents of the receive latch.

SFR Address = 0xBC; SFR Page = All Pages

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
AD0SC4	AD0SC3	AD0SC2	AD0SC1	AD0SC0	AD0LJST	-	-	11111000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xBC

Bits7–3: AD0SC4–0: ADC0 SAR Conversion Clock Period Bits.
SAR Conversion clock is derived from system clock by the following equation, where *AD0SC* refers to the 5-bit value held in bits AD0SC4-0. SAR Conversion clock requirements are given in Table 5.1.

$$AD0SC = \frac{SYSCLK}{CLK_{SAR}} - 1$$

Bit2: AD0LJST: ADC0 Left Justify Select.
0: Data in ADC0H:ADC0L registers are right-justified.
1: Data in ADC0H:ADC0L registers are left-justified.

Bits1–0: UNUSED. Read = 00b; Write = don't care.

המשך בעמוד 30

ADC0CN: ADC0 Control

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
AD0EN	AD0TM	AD0INT	AD0BUSY	AD0WINT	AD0CM2	AD0CM1	AD0CM0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address:
							(bit addressable)	0xE8

Bit7: AD0EN: ADC0 Enable Bit.
0: ADC0 Disabled. ADC0 is in low-power shutdown.
1: ADC0 Enabled. ADC0 is active and ready for data conversions.

Bit6: AD0TM: ADC0 Track Mode Bit.
0: Normal Track Mode: When ADC0 is enabled, tracking is continuous unless a conversion is in progress.
1: Low-power Track Mode: Tracking Defined by AD0CM2-0 bits (see below).

Bit5: AD0INT: ADC0 Conversion Complete Interrupt Flag.
0: ADC0 has not completed a data conversion since the last time AD0INT was cleared.
1: ADC0 has completed a data conversion.

Bit4: AD0BUSY: ADC0 Busy Bit.
Read:
0: ADC0 conversion is complete or a conversion is not currently in progress. AD0INT is set to logic 1 on the falling edge of AD0BUSY.
1: ADC0 conversion is in progress.
Write:
0: No Effect.
1: Initiates ADC0 Conversion if AD0CM2-0 = 000b

Bit3: AD0WINT: ADC0 Window Compare Interrupt Flag.
0: ADC0 Window Comparison Data match has not occurred since this flag was last cleared.
1: ADC0 Window Comparison Data match has occurred.

Bits2–0: AD0CM2–0: ADC0 Start of Conversion Mode Select.
When AD0TM = 0:
000: ADC0 conversion initiated on every write of '1' to AD0BUSY.
001: ADC0 conversion initiated on overflow of Timer 0.
010: ADC0 conversion initiated on overflow of Timer 2.
011: ADC0 conversion initiated on overflow of Timer 1.
100: ADC0 conversion initiated on rising edge of external CNVSTR.
101: ADC0 conversion initiated on overflow of Timer 3.
11x: Reserved.
When AD0TM = 1:
000: Tracking initiated on write of '1' to AD0BUSY and lasts 3 SAR clocks, followed by conversion.
001: Tracking initiated on overflow of Timer 0 and lasts 3 SAR clocks, followed by conversion.
010: Tracking initiated on overflow of Timer 2 and lasts 3 SAR clocks, followed by conversion.
011: Tracking initiated on overflow of Timer 1 and lasts 3 SAR clocks, followed by conversion.
100: ADC0 tracks only when CNVSTR input is logic low; conversion starts on rising CNVSTR edge.
101: Tracking initiated on overflow of Timer 3 and lasts 3 SAR clocks, followed by conversion.
11x: Reserved.

המשך בעמוד 31

SFR Definition 5.1. AMX0P: AMUX0 Positive Channel Select

R	R	R	R/W	R/W	R/W	R/W	R/W	Reset Value
-	-	-	AMX0P4	AMX0P3	AMX0P2	AMX0P1	AMX0P0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xBB

Bits7–5: UNUSED. Read = 000b; Write = don't care.
Bits4–0: AMX0P4–0: AMUX0 Positive Input Selection

AMX0P4-0	ADC0 Positive Input (32-pin Package)	ADC0 Positive Input (48-pin Package)
00000	P1.0	P2.0
00001	P1.1	P2.1
00010	P1.2	P2.2
00011	P1.3	P2.3
00100	P1.4	P2.5
00101	P1.5	P2.6
00110	P1.6	P3.0
00111	P1.7	P3.1
01000	P2.0	P3.4
01001	P2.1	P3.5
01010	P2.2	P3.7
01011	P2.3	P4.0
01100	P2.4	P4.3
01101	P2.5	P4.4
01110	P2.6	P4.5
01111	P2.7	P4.6
10000	P3.0	RESERVED
10001	P0.0	P0.3
10010	P0.1	P0.4
10011	P0.4	P1.1
10100	P0.5	P1.2
10101 - 11101	RESERVED	RESERVED
11110	Temp Sensor	Temp Sensor
11111	V_{DD}	V_{DD}

המשך בעמוד 32

SFR Definition 5.2. AMX0N: AMUX0 Negative Channel Select

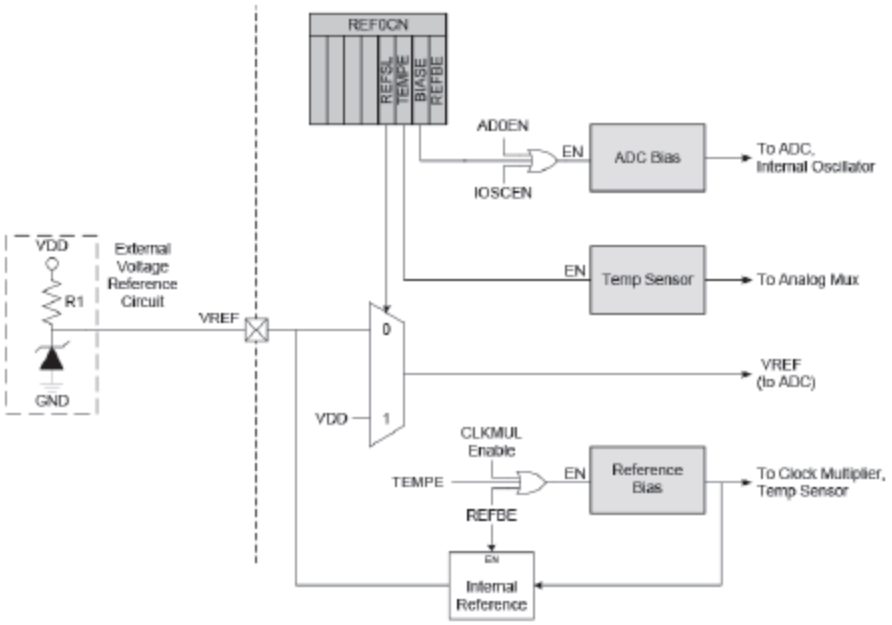
R	R	R	R/W	R/W	R/W	R/W	R/W	Reset Value
-	-	-	AMX0N4	AMX0N3	AMX0N2	AMX0N1	AMX0N0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xBA

Bits7–5: UNUSED. Read = 000b; Write = don't care.
Bits4–0: AMX0N4–0: AMUX0 Negative Input Selection.
Note that when GND is selected as the Negative Input, ADC0 operates in Single-ended mode. For all other Negative Input selections, ADC0 operates in Differential mode.

AMX0N4-0	ADC0 Negative Input (32-pin Package)	ADC0 Negative Input (48-pin Package)
00000	P1.0	P2.0
00001	P1.1	P2.1
00010	P1.2	P2.2
00011	P1.3	P2.3
00100	P1.4	P2.5
00101	P1.5	P2.6
00110	P1.6	P3.0
00111	P1.7	P3.1
01000	P2.0	P3.4
01001	P2.1	P3.5
01010	P2.2	P3.7
01011	P2.3	P4.0
01100	P2.4	P4.3
01101	P2.5	P4.4
01110	P2.6	P4.5
01111	P2.7	P4.6
10000	P3.0	RESERVED
10001	P0.0	P0.3
10010	P0.1	P0.4
10011	P0.4	P1.1
10100	P0.5	P1.2
10101 - 11101	RESERVED	RESERVED
11110	VREF	VREF
11111	GND (Single-Ended Mode)	GND (Single-Ended Mode)

Voltage Reference Option

המשך בעמוד 33



Voltage Reference Functional Block Diagram

דוגמאות למתחי ייחוס

- מתח ייחוס של VDD = 3.3V — REFOCN = 0x08;
- מתח ייחוס פנימי של 1.8V — REFOCN = 0x10;
- מתח VREF חיצוני דרך הדק P1.5 — REFOCN = 0x00;
- מתח פנימי של (1xGain) — REFOCN = 0x81;
  או 2.4v (2xGain) — REFOCN = 0x01;
  (הדק P1.5 צריך להיות פנוי)

המשך בעמוד 34

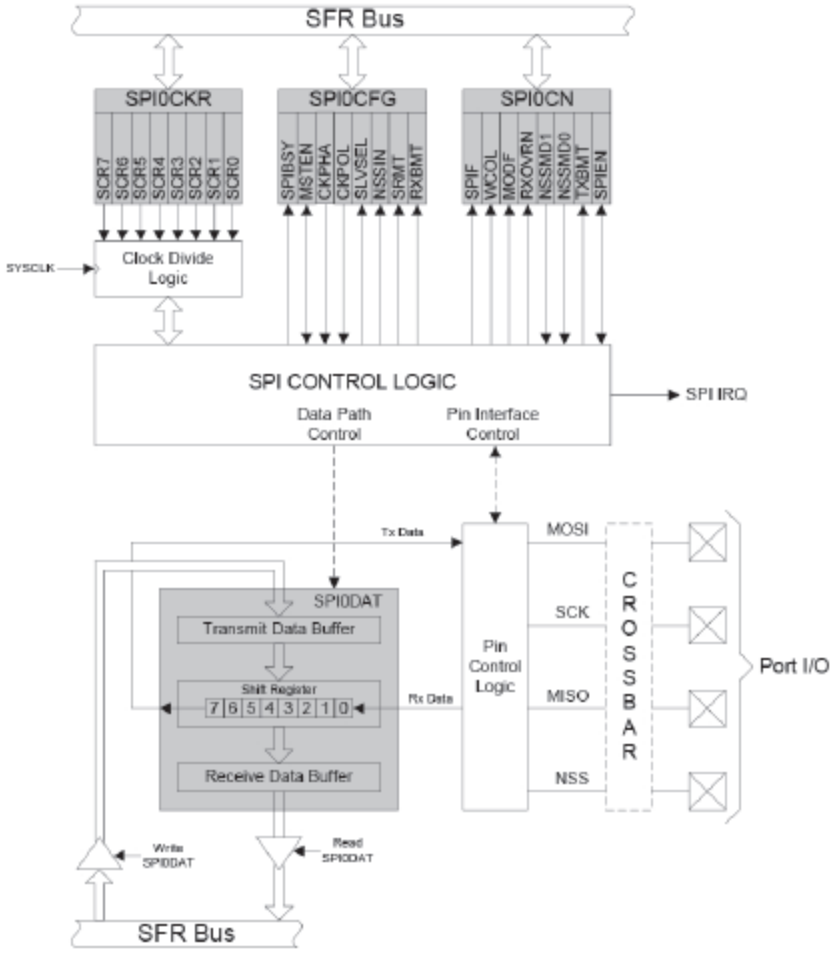
REF0CN: Reference Control

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
-	-	-	-	REFSL	TEMPE	BIASE	REFBE	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	SFR Address: 0xD1

Bits7–3: UNUSED. Read = 00000b; Write = don't care.
Bit3: REFSL: Voltage Reference Select.
This bit selects the source for the internal voltage reference.
0: VREF pin used as voltage reference.
1: V_{DD} used as voltage reference.
Bit2: TEMPE: Temperature Sensor Enable Bit.
0: Internal Temperature Sensor off.
1: Internal Temperature Sensor on.
Bit1: BIASE: Internal Analog Bias Generator Enable Bit.
0: Internal Bias Generator off.
1: Internal Bias Generator on.
Bit0: REFBE: Internal Reference Buffer Enable Bit.
0: Internal Reference Buffer disabled.
1: Internal Reference Buffer enabled. Internal voltage reference driven on the VREF pin.

המשך בעמוד 35

Enhanced Serial Peripheral Interface (SPI0)



SPI Block Diagram

המשך בעמוד 36

SPI0CFG: SPI0 Configuration

R	R/W	R/W	R/W	R	R	R	R	Reset Value
SPIBSY	MSTEN	CKPHA	CKPOL	SLVSEL	NSSIN	SRMT	RXBMT	00000111
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

SFR Address: 0xA1

Bit 7: SPIBSY: SPI Busy (read only).
This bit is set to logic 1 when a SPI transfer is in progress (Master or slave Mode).

Bit 6: MSTEN: Master Mode Enable.
0: Disable master mode. Operate in slave mode.
1: Enable master mode. Operate as a master.

Bit 5: CKPHA: SPI0 Clock Phase.
This bit controls the SPI0 clock phase.
0: Data centered on first edge of SCK period.*
1: Data centered on second edge of SCK period.*

Bit 4: CKPOL: SPI0 Clock Polarity.
This bit controls the SPI0 clock polarity.
0: SCK line low in idle state.
1: SCK line high in idle state.

Bit 3: SLVSEL: Slave Selected Flag (read only).
This bit is set to logic 1 whenever the NSS pin is low indicating SPI0 is the selected slave. It is cleared to logic 0 when NSS is high (slave not selected). This bit does not indicate the instantaneous value at the NSS pin, but rather a de-glitched version of the pin input.

Bit 2: NSSIN: NSS Instantaneous Pin Input (read only).
This bit mimics the instantaneous value that is present on the NSS port pin at the time that the register is read. This input is not de-glitched.

Bit 1: SRMT: Shift Register Empty (Valid in Slave Mode, read only).
This bit will be set to logic 1 when all data has been transferred in/out of the shift register, and there is no new information available to read from the transmit buffer or write to the receive buffer. It returns to logic 0 when a data byte is transferred to the shift register from the transmit buffer or by a transition on SCK.
NOTE: SRMT = 1 when in Master Mode.

Bit 0: RXBMT: Receive Buffer Empty (Valid in Slave Mode, read only).
This bit will be set to logic 1 when the receive buffer has been read and contains no new information. If there is new information available in the receive buffer that has not been read, this bit will return to logic 0.
NOTE: RXBMT = 1 when in Master Mode.

*Note: In slave mode, data on MOSI is sampled in the center of each data bit. In master mode, data on MISO is sampled one SYSCLK before the end of each data bit, to provide maximum settling time for the slave device. See Table 20.1 for timing parameters.

המשך בעמוד 37

SPI0N: SPI0 Control

R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	Reset Value
SPIF	WCOL	MODF	RXOVRN	NSSMD1	NSSMD0	TXBMT	SPIEN	00000110
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	Bit Addressable

SFR Address: 0xF8

Bit 7: SPIF: SPI0 Interrupt Flag.
This bit is set to logic 1 by hardware at the end of a data transfer. If interrupts are enabled, setting this bit causes the CPU to vector to the SPI0 interrupt service routine. This bit is not automatically cleared by hardware. It must be cleared by software.

Bit 6: WCOL: Write Collision Flag.
This bit is set to logic 1 if a write to SPI0DAT is attempted when the transmit buffer has not been emptied to the SPI shift register. When this occurs, the write to SPI0DAT will be ignored, and the transmit buffer will not be written. This flag can occur in all SPI0 modes. It must be cleared by software.

Bit 5: MODF: Mode Fault Flag.
This bit is set to logic 1 by hardware (and generates a SPI0 interrupt) when a master mode collision is detected (NSS is low, MSTEN = 1, and NSSMD[1:0] = 01). This bit is not automatically cleared by hardware. It must be cleared by software.

Bit 4: RXOVRN: Receive Overrun Flag (Slave Mode only).
This bit is set to logic 1 by hardware (and generates a SPI0 interrupt) when the receive buffer still holds unread data from a previous transfer and the last bit of the current transfer is shifted into the SPI0 shift register. This bit is not automatically cleared by hardware. It must be cleared by software.

Bits 3–2: NSSMD1–NSSMD0: Slave Select Mode.
Selects between the following NSS operation modes:
(See Section "20.2. SPI0 Master Mode Operation" on page 224 and Section "20.3. SPI0 Slave Mode Operation" on page 226).
00: 3-Wire Slave or 3-wire Master Mode. NSS signal is not routed to a port pin.
01: 4-Wire Slave or Multi-Master Mode (Default). NSS is always an input to the device.
1x: 4-Wire Single-Master Mode. NSS signal is mapped as an output from the device and will assume the value of NSSMD0.

Bit 1: TXBMT: Transmit Buffer Empty.
This bit will be set to logic 0 when new data has been written to the transmit buffer. When data in the transmit buffer is transferred to the SPI shift register, this bit will be set to logic 1, indicating that it is safe to write a new byte to the transmit buffer.

Bit 0: SPIEN: SPI0 Enable.
This bit enables/disables the SPI.
0: SPI disabled.
1: SPI enabled.

המשך בעמוד 38

SPI0CKR: SPI0 Clock Rate

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
SCR7	SCR6	SCR5	SCR4	SCR3	SCR2	SCR1	SCR0	00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

SFR Address: 0xA2

Bits 7–0: SCR7–SCR0: SPI0 Clock Rate.
These bits determine the frequency of the SCK output when the SPI0 module is configured for master mode operation. The SCK clock frequency is a divided version of the system clock, and is given in the following equation, where *SYSCLK* is the system clock frequency and *SPI0CKR* is the 8-bit value held in the SPI0CKR register.

$$f_{SCK} = \frac{SYSCLK}{2 \times (SPI0CKR + 1)}$$

for 0 <= SPI0CKR <= 255

Example: If SYSCLK = 2 MHz and SPI0CKR = 0x04,

$$f_{SCK} = \frac{2000000}{2 \times (4 + 1)}$$

$$f_{SCK} = 200kHz$$

SPI0DAT: SPI0 Data

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Reset Value
								00000000
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	

SFR Address: 0xA3

Bits 7–0: SPI0DAT: SPI0 Transmit and Receive Data.
The SPI0DAT register is used to transmit and receive SPI0 data. Writing data to SPI0DAT places the data into the transmit buffer and initiates a transfer when in Master Mode. A read of SPI0DAT returns the contents of the receive buffer.

בהצלחה!

פרק 1 : מבוא למיקרו בקרים

1.1 : מידע על החברה Silicon Laboratories המייצרת את הרכיב

המיקרו בקר C8051F380 מיוצר על ידי חברת **Silicon Labs** או בשמה המלא יותר **Silicon Laboratories, Inc.** . החברה נוסדה ב 1996 והפכה לחברה ציבורית בשנת 2000 . היא מייצרת מוליכים למחצה, התקני סיליקון אחרים ותוכנות. עיקר עיסוקה הוא בתשתיות של IOT - (Internet Of Things) , אוטומציה תעשייתית לצרכן הפרטי ולשוק הרכב ברחבי העולם. ההנהלה של החברה ממוקמת באוסטין , טקסס , ארצות הברית והחברה מתמקדת במיקרו בקרים , מערכות SoCs (System On Chips) רכיבי תזמון, רכיבי בידוד ספרתיים, חיישנים ורכיבי שידור . כמו כן היא מייצרת Software Stacks - מחסניות תוכנה - כולל ספריות ותוכנה מבוססת פרוטוקולים וגם פלטפורמה של פיתוח תוכנה חופשית הנקראת Simplicity Studio .

אתר החברה נמצא בקישור : https://www.silabs.com . דפי נתונים על המיקרו בקר 8051F38x נמצאים בקישור :
https://www.silabs.com/documents/public/data-sheets/C8051F38x.pdf

1.2 : מבוא למחשבים ספרתיים , מיקרו מעבדים ומיקרו בקרים

1.2.1 היסטוריה

אדם נולד עם 10 אצבעות בידיים ולכן עד היום משתמשים במושג "דיגיטאלי" שבא מהמילה digitus שהיא אצבע בשפה הלטינית. בגלל 10 האצבעות אנחנו עובדים בחיי היום יום בחשבון עשרוני שבו 10 ספרות בסיסיות מ 0 ועד 9 .

כאשר האצבעות לא הספיקו לספירה, השתמשו אבותינו הקדמונים בחלוקי נחל שאפשרו להם גם פעולות חיבור חיסור כפל וחילוק. חלוקי נחל נקראים בלטינית CALCULUS ומכאן שם של מחשבון הוא CALCULATOR .

בבבל העתיקה (מגדל בבל...זוכרים ?) הייתה שיטת ספירה בת 60 ספרות ומכאן קיבלנו את השעה בת 60 הדקות וממנה את 60 השניות בדקה ולכן במעגל יש גם 360 מעלות.

המחשב הספרתי בן ימינו עובד עם חשבון בינארי (2 ספרות : 0 ו 1) ועם כללים של האלגברה הבוליאנית.

במאה ה 19 היו שני מדענים חשובים לנושא המחשבים. שם האחד היה צ'רלס באבאדג' – **Charles BABBADGE** - שניסה לבנות מחשב מכני עם עקרון דומה למחשבים של היום ושם השני ג'ורג' בול **GEORGE BOOLE** - שפיתח את האלגברה הבוליאנית. שניהם לא זכו לראות תוצאות נושאות פרי בחייהם. המכונה/מחשב של באבאדג' הייתה גדולה ומורכבת והאלגברה הבוליאנית התחילה להתפתח רק במאה ה 20 עם התפתחות המחשבים.

בשנות ה 40 של המאה הקודמת, במלחמת העולם השנייה, פיתחו מכונות לפיצוח הקודים/צפנים של הצבא הגרמני עם העקרונות של באבאדג' ובול. אלו היו המחשבים הראשונים. הם היו בנויים משפופרות ריק , היו גדולים מאוד והיו ממוקדים למשימה אחת של פיצוח צופן.

שני מדענים נוספים שצריך להזכיר והם נחשבים אבות המחשב המודרני הם אלאן טיורינג – ALAN YURING ו JHON VON NEWMANN . הראשון קבע שהמחשב יהיה מכונה אוניברסלית שיבצע עם אותה חומרה משימות שונות עם תוכנות שונות. השני קבע את העיקרון שיהיה קיים זיכרון אחד שבו יהיו גם התוכנית וגם הנתונים (העיקרון נקרא : "המחשב עם התוכנית המאוחסנת") . העיקרון הזה הגדיל את מהירות ביצוע התוכניות בצורה משמעותית.

דורות המחשב מבחינת האלקטרוניקה מחולקים בצורה הבאה:

- דור ראשון - דור השפופרות האלקטרוניות (שפופרות ריק) : שנים 1946 עד 1953 .

- דור שני – דור הטרנזיסטורים : שנים 1953 עד 1975)
- דור שלישי – דור שבבי הסיליקון (הג'וקים) : מ 1975 עד היום.
- הדור הרביעי – דור מחשבים מולקולריים , מחשבים ביולוגיים (מימים אלו עד....)

המחשב הראשון יצא בשנת 1946 ונקרא **ENIAC** . היו בו למעלה מ 17000 שפופרות ו 5 מיליון חיבורים. משקלו היה 30 טון והוא היה מבצע 5000 פעולות בשנייה.

המחשב הראשון של דור הטרנזיסטורים הופעל ב 1957 ונקרא **PHILCO S 2000** . הזיכרון היה בגודל 32Kbytes (כן , זו לא טעות , קילו בתים...) והוא ביצע 60000 פעולות בשנייה.

בדור שבבי הסיליקון , הרכיבים והחיבורים ביניהם נמצאים בשכבות סיליקון זעירות. הרכיבים והחיבורים מוטבעים בשכבות הסיליקון ובכל שכבה של סנטימטרים בודדים יכולים להיות מיליוני רכיבים והחיבורים ביניהם. המעבד הראשון בדור השבבים יוצר ב 1971 - INTEL 4004 - ונקרא מיקרו מעבד – מיקרו פרוסור.

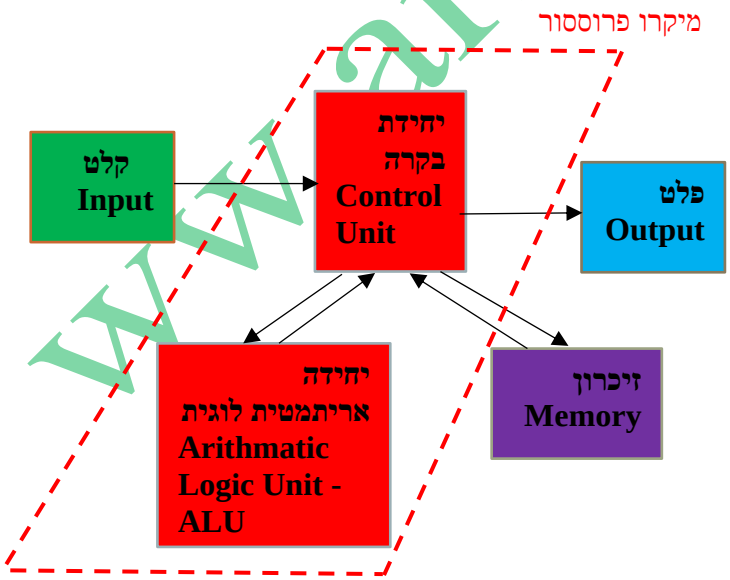
בתחילה שימשו המיקרו מעבדים במכונות חישוב או מחשבים למטרות מסוימות. עם השנים הם התפתחו גם לשימוש ביתי ונקראו מחשבים אישיים – PC- Personal Computer . המעבד במחשבים האישיים לקראת שנות 2020 הוא INTEL CORE I9 ועד שנקרא את הפרק הזה יגיע מעבד מתקדם יותר .

מאז המחשבים האישיים הראשונים חלו שיפורים עצומים במהירות הביצוע, בגודל הזיכרון הפנימי , בקיבולת הדיסקים (טרות של זיכרון) ועם כל זאת מחירם יורד. במחשב ביתי צנוע יש היום כוח חישוב גדול יותר מזה שהיה במחשבים ובלוויינים של סוכנות החלל האמריקאי - NASA .

היום (שנת 2020) יש **מחשבי על** הבנויים מאלפי שבבים דומים לאלו שבמחשב האישי ומשולבים יחד לעבודה מקבילית. במחשב שנקרא CRAY XK6 וניתן להגיע ל $50 * 10^{15}$ פעולות בשנייה אחת.

1.2.2 : מבנה המחשב

בכל מחשב יש את 5 המלבנים שבאיור 1:



איור 1 : מבנה מחשב ספרתי

יחידת הקלט – יחידה המכניסה נתונים מהעולם החיצון למחשב. בשפה מקצועית יותר : יחידה המתרגמת את שפת העולם שמחוץ למחשב לשפת המחשב. אנחנו נמצאים מחוץ למחשב. בתוך המחשה יש רק אפסים ואחדים. דוגמאות להתקני קלט הם מפסק, עכבר, לוח מקשים וכו'. אנחנו יודעים ללחוץ על מקש, להזיז עכבר, להקיש על המקלדת. תפקיד יחידת הקלט להמיר את לחיצת המפסק או הזזת העכבר או הקשה במקלדת לשפה של אפסים ואחדים שהיא השפה שאותה מדבר המחשב

יחידת הפלט – יחידה המוציאה נתונים מהמחשב אל העולם מחוץ למחשב. בשפה מקצועית : יחידה המתרגמת את שפת המחשב לשפה שמחוץ למחשב. יחידות פלט הן מסך, מנורה, מדפסת וכו'. תפקיד יחידת הפלט להוציא נתון מהמחשב ולהציג אותו בעולם שמחוץ למחשב כמו במסך או במדפסת או להדליק/לכבות מנורה.

יחידה אריתמטית לוגית – ALU - זוהי היחידה שבה מתבצעות כל הפעולות החשבוניות והלוגיות. ביחידה הזו יש מעגלי אלקטרוניקה שיודעים לבצע פעולות חשבוניות כמו חיבור , חיסור , כפל וחילוק. כמו כן יש בה מעגלים לוגיים שיודעים לבצע AND , OR , NOT ו XOR .

יחידת הבקרה – זוהי היחידה שמנהלת את כל מה שקורה המחשב. תפקידה לבצע את הפקודות של התוכנית אחת אחרי השנייה. היא מבצעת את התוכנית בצורה הבאה :

א. מביאה את הפקודה – הוראה - מהזיכרון -FETCH ושומרת את הפקודה ברגיסטר הנקרא רגיסטר ההוראה – IR- Instruction Register. בסיום שלב הבאת הפקודה היא מקדמת מונה הנקרא רגיסטר שנקרא מונה התוכנית – PC – Program Counter – ב 1 כדי שיצביע על הכתובת הבאה בזיכרון ממנה נביא את הפקודה.

ב. מפענחת מהו אוסף האפסים והאחדים שנמצאים בפקודה שב IR .

ג. מבצעת את הפקודה על ידי שרשרת אותות ליחידות המתאימות לפקודה .

בסיום שלב ג' חוזרים לשלב א' ומביאים את הפקודה הבאה מהכתובת שמראה מונה הכתובות.

יחידת הזיכרון - זוהי יחידת האחסון – שמירה – אגירה של המחשב. ביחידה הזו נמצאת התוכנית שהמחשב מריץ, נתונים שהוכנסו לפני שהתוכנית התחילה לרוץ, נתונים ותוצאות שעובדו במהלך ריצת התוכנית.

חברת אינטל הכניסה את 2 המלבנים שבתוך הקווים המקווקוים באדום לתוך ג'וק אחד שנקרא מיקרו פרוססור ונתנה לו שם INTEL 4001 . ומרגע זה השתנה משמעותית עולם המחשבים. ממחשב בגודל פיזי גדול הם הפכו להיות קטנים משמעותית. כל שיש לעשות הוא לקחת מיקרו פרוססור, להוסיף לו זיכרון בנפח מתאים והתקני קלט ופלט (התלויים באפליקציה שאנחנו מפתחים) ויש לנו מחשב.

בשנות ה 50 וה 60 של המאה הקודמת המחשבים היו מאוד גדולים , יקרים ונדירים. כאשר התחילו לעבוד עם מעגלים מוכללים – שבבים - ג'וקים – הם אפשרו למזער את גודל המחשב והוזילו בהדרגה את מחירי המחשב. כך נוצר בסוף שנות ה־60 סוג חדש של מחשב המיני מחשב שהורכב יותר ויותר ממעגלים מוכללים. המרכיב הראשי של מחשב הוא יחידת העיבוד המרכזית - CPU - Central Processing Unit - הכוללת מספר של מעגלים שונים.

פריצה משמעותית בעולם המחשבים הייתה דחיסה של יחידת העיבוד המרכזית בתוך מעגל מוכלל, שבב , ג'וק יחיד. בהתחלה הג'וק נקרא מיקרו פרוססור או מיקרו מעבד והג'וק הראשון היה ה 4004 שפותח בשנת 1971 על ידי חברת אינטל. עם השנים הוא התפתח ונהיה פרוססור – מעבד. המעבדים הוכנסו במחשבים אישיים, מחשבי מיני, מחשבים גדולים ומחשבי על.

בנוסף למיקרו מעבד או המעבד קיים גם מיקרו-בקר. המיקרו בקר הוא מערכת מחשב שלמה, הנמצאת בג'וק אחד. בתוך הרכיב נמצא יחידת עיבוד, זיכרון, פורטים לקלט פלט ועוד רכיבים פריפריאליים (היקפיים) כמו טיימרים , מערכת פסיקות, אפשרות תקשורת עם רכיבים אחרים ועוד . ההבדל בין מחשב רגיל למיקרו-בקר הוא בכך שלמיקרו-בקר יש לרוב ביצועים חלשים יותר. מטרתו של המיקרו-בקר היא לבקר על תהליכים . במקום שלא צריך מעבד עם "המון כוח" תופס את מקומו המיקרו בקר. הוא נמצא בכל המערכות הסובבות אותנו כמו טלוויזיה, רדיו, נגני CD , מערכות אזעקה, מערכות גישה וכניסה למשרדים, צעצועים, תנור המיקרו, מכונת הכביסה, מדיח הכלים ועוד.

המעבדים עובדים עם ארכיטקטורה של **ון ניומן** שבה זיכרון התוכנית וזיכרון הנתונים נמצאים באותו מרחב זיכרון ונמצאים בכתובות שונות, לעומת מיקרו בקרים העובדים בארכיטקטורה הנקראת **הארוארד** (HARVARD) שבה זיכרון התוכנית וזיכרון הנתונים הם במרחבי זיכרון שונים - יכולה להיות כתובת 100 בזיכרון התוכנית וכתובת 100 בזיכרון הנתונים.

אחד המיקרו בקרים הנפוצים והפופולאריים הוא ה **8051** של חברת אינטל (צאצא של מיקרו בקר קודם שנקרא 8048) . הוא מכיל את יחידת ה CPU , זיכרון מסוג ROM ו RAM ובנוסף הוא מכיל את היחידות I/O PORTS, TIMERS, תקשורת טורית ובקרת פסיקות.

1.3 : השוואה בין מיקרו מעבד ומיקרו בקר

המונחים מיקרו מעבד ומיקרו בקר גורמים בדרך כלל לבלבול בין האחד לשני. שניהם מתוכננים ליישומי זמן אמת ויש להם תכונות משותפות רבות אבל גם הבדלים משמעותיים לא מעטים. נתאר בעזרת טבלה את ההבדלים העיקריים ביניהם.

	מיקרו מעבד - מיקרופרוססור	**מיקרו בקר – מיקרוקונטרולר**
תפקיד	עיבוד נתונים	בקרה על יישומים בית/תעשייה
דוגמאות	Pentium 1,2,3,4, core 2 duo, i3, i5, i7	Intel 8051,AT89S52, Arduino uno, Motorola 68HCxx, PIC,TI Philips, Freescale ,Silicon Labs .
מה יש ברכיב	במיקרו פרוססורים הראשונים היה CPU בלבד (Central Processing Unit – יחידת עיבוד מרכזית) . לא היו זיכרונות RAM או ROM או רכיבים היקפיים בתוך הרכיב. מתכנן המערכת היה צריך להוסיף זיכרונות ורכיבים היקפיים כדי לבצע את התפקיד הרצוי. עם השנים המיקרו פרוססורים הפכו להיות גדולים יותר עם זיכרונות בתוך הרכיב	בנוסף ל CPU יש גם זיכרונות RAM , ROM , פורטים לקלט/פלט ורכיבים היקפיים (פסיקות, טיימרים , תקשורת) בתוך הרכיב

	ועם רכיבים פנימיים נוספים והם נקראים פרוסורים - מעבדים.	
שימוש	Desktop PC's, Laptops, notepads מערכות עם כושר עיבוד מתמטי/נתונים גבוה, משחקי מחשב עם גרפיקה , מעבדי תמלילים, בסיסי נתונים ועוד	יישום ספציפי שבו קלט מחיישנים או מפסקים או/ו פלט למנורות, מנועים, לדים וכו' כדוגמת מערכות אזעקה, טלפון, מכוניות, מכונת כביסה, ייבוש , לוח מודעות, שעונים נגני MP3 וכו' .
זיכרון	הזיכרון החיצוני שיחובר יהיה בגודל עשרות גיגה בתים ויותר. במעבדים החדשים יש זיכרונות בגודל עצום.	היות והמשימה מאד ספציפית מספיק זיכרון תכנית קטן (עד עשרות קילו בתים) וזיכרון נתונים קטן (מאות בתים עד קילו בתים).
כמות הדקים	יכולה להגיע למאות	קטנה יחסית. החל מ 6 הדקים עד כמה עשרות .
מהירות עבודה	גבוהה – גיגה הרץ.	עשרות מגה הרץ.
תצרוכת הספק	גדולה ולכן במעבדים המתקדמים נדרשת מערכת קירור.	נמוכה יחסית (מילי וואט) ויש מצבי חיסכון של מיקרו וואט.
ארכיטקטורה	וואן ניומן – Von Neumann - התוכנית והנתונים נמצאים באותו הזיכרון	הרווארד - Harvard – זיכרון התוכנית והנתונים נפרדים.
מחיר	יקר – עשרות דולר עד אלפי דולר	זול (החל בדולרים בודדים עד עשרות דולר)

טבלה 1 : השוואה בין מיקרו מעבד למיקרו בקר

1.4 : המיקרו בקר C8051F380

חברת Silicon Labs מייצרת בין שאר המוצרים שהיא מפתחת , גם מיקרו בקרים שהליבה (core) היא 8051 וסביב הליבה הם הוסיפו מעגלי אלקטרוניקה נוספים. יש מספר גרסאות שונות הנקראות C8051Fxxx , כאשר כל x בשם הוא סדרה אחרת. לדוגמה יש C8051F010 ו C8051F020 וגם C8051F340 ועוד. אנחנו נתמקד ב C8051F380 . גם בסדרה C8051F38x יש גרסאות C8051F380 , C8051F381 , C8051F382 וכך הלאה עד הספרה 7 ויש ביניהם הבדלים קטנים . בכל הגרסאות הם שמרו תאימות - compatibility – של הליבה ל 8051 המקורי של אינטל גם מבחינת החומרה וגם מבחינת התוכנה. כמובן שמהירות העבודה גבוהה בהרבה מזו של ה 8051 המקורי וגם נפח זיכרון התוכנית (מסוג FLASH) וה RAM גדולים בהרבה. כמו כן ישנן תוספות משודרגות של מעגלים כמו ADC , DAC , סוגי תקשורת שונים וכו'. חברות נוספות מייצרות מיקרו בקרים עם ליבה דומה ל 51. בין החברות נזכיר את ATMEL , PHILLIPS ,CYGNAL ,TI ועוד.

לחברות אלקטרוניקה נוספות יש מיקרו בקרים שאינם מבוססים על ה 8051 המקורי של אינטל. ה PIC של חברת MICRO CHIP הוא דוגמא למיקרו בקר שלו ליבה אחרת מזו של ה 51 .

כל המיקרו בקרים במשפחת ה C8051Fxxx הם מיקרו בקרים של 8 ביט שהגרעין (CORE) הוא כמו של 8051 של אינטל. הם שומרים תאימות קוד עם 8051 של אינטל אבל הביצועים שלהם מהירים יותר , הזיכרון שלהם גדול יותר ויש בהם מעגלי אלקטרוניקה נוספים שלא היו ב 8051 המקורי של אינטל. כולם מיקרו בקרים של 8 ביט שאומר שפס הנתונים שלהם הוא של 8 הדקים. קיימים רכיבי מיקרו בקר

אחרים של החברה עם פס נתונים של 16 ביט. ההבדלים בין הרכיבים הם בנפח הזיכרון בתוך הרכיב, גודל זיכרון הנתונים, כמות הטיימרים, אילו סוגי רכיבים פריפריאליים יש ברכיב ועוד.

החברה מוכרת גם לוחות פיתוח – Development Board - עם המיקרו בקר הרצוי עם אפשרות להתחבר להדקי המיקרו. כמו כן היא מספקת תוכנת IDE - Integrated Development Environment – סביבת פיתוח משולבת - הנקראת **Simplicity Studio 5** (בשנת 2020 הגרסה של התוכנה היא 5). הגרסה היא בחינם.

החברה מספקת גם תוכנה נוספת העוזרת למשתמש לאתחל את המיקרו בקר , את הרגיסטרים שלו ואת הרכיבים הפריפריאליים והיא נקראת **Configuration Wizard** .

1.5 : הרכיבים במשפחת ה C8051F380

טבלה מספר 2 מתארת מה הרכיבים הנמצאים בכל אחד מהמיקרו בקרים במשפחה C8051F38x. השורות של הטבלה מתארות את מספר המיקרו בקר. העמודה הראשונה שנקרא לה עמודה 0 מראה מהו שם המיקרו בקר.

העמודה הראשונה היא מספר ה MIPS - Mega Instructions Per Second – מיליון הוראות בשנייה. בכל הרכיבים מדובר ב 48 מיליון הוראות בשנייה אחת (יש לזכור שמהברים גביש בתדר 48MHz). העמודה השנייה מראה מה גודל זיכרון התוכנית שהוא ROM מסוג הבזק (FLASH). ברכיב C8051F380 הזיכרון בגודל 64 קילו בתים.

Ordering Part Number	MIPS (Peak)	Flash Memory (Bytes)	RAM	Calibrated Internal Oscillator	Low Frequency Oscillator	USB with 1k Endpoint RAM	Supply Voltage Regulator	SMBus/I2C	Enhanced SPI	UARTs	Timers (16-bit)	Programmable Counter Array	Digital Port I/O	External Memory Interface (EMIF)	10-bit 500ksps ADC	Temperature Sensor	Voltage Reference	Analog Comparators	Package
C8051F380-GQ	48	64k	4352	✓	✓	✓	✓	2	✓	2	6	✓	40	✓	✓	✓	✓	2	TQFP48
C8051F381-GQ	48	64k	4352	✓	✓	✓	✓	2	✓	2	6	✓	25	—	✓	✓	✓	2	LQFP32
C8051F381-GM	48	64k	4352	✓	✓	✓	✓	2	✓	2	6	✓	25	—	✓	✓	✓	2	QFN32
C8051F382-GQ	48	32k	2304	✓	✓	✓	✓	2	✓	2	6	✓	40	✓	✓	✓	✓	2	TQFP48
C8051F383-GQ	48	32k	2304	✓	✓	✓	✓	2	✓	2	6	✓	25	—	✓	✓	✓	2	LQFP32
C8051F383-GM	48	32k	2304	✓	✓	✓	✓	2	✓	2	6	✓	25	—	✓	✓	✓	2	QFN32
C8051F384-GQ	48	64k	4352	✓	✓	✓	✓	2	✓	2	6	✓	40	✓	—	—	—	2	TQFP48
C8051F385-GQ	48	64k	4352	✓	✓	✓	✓	2	✓	2	6	✓	25	—	—	—	—	2	LQFP32
C8051F385-GM	48	64k	4352	✓	✓	✓	✓	2	✓	2	6	✓	25	—	—	—	—	2	QFN32
C8051F386-GQ	48	32k	2304	✓	✓	✓	✓	2	✓	2	6	✓	40	✓	—	—	—	2	TQFP48
C8051F387-GQ	48	32k	2304	✓	✓	✓	✓	2	✓	2	6	✓	25	—	—	—	—	2	LQFP32
C8051F387-GM	48	32k	2304	✓	✓	✓	✓	2	✓	2	6	✓	25	—	—	—	—	2	QFN32
C8051F38C-GQ	48	16k	2304	✓	✓	✓	✓	2	✓	2	6	✓	25	—	✓	✓	✓	2	LQFP32
C8051F38C-GM	48	16k	2304	✓	✓	✓	✓	2	✓	2	6	✓	25	—	✓	✓	✓	2	QFN32

טבלה 2 : הרכיבים במיקרו בקרים השונים במשפחה C8051F38x .

העמודה השלישית מתארת את נפח זיכרון ה RAM הפנימי. עבור ה C8051F380 יש 4352 בתים שהם 4.25 קילו בתים (1 קילו בתים שווה 1024 בתים). הזיכרון מורכב מ 2 זיכרונות RAM . **א.** 256 בתים (1/4 קילו) של RAM . **ב.** עוד 4 קילו בתים של RAM שנקרא XRAM .

העמודה הרביעית מתארת האם קיים מתנד פנימי מכויל ברכיב. רואים שבכולם קיים מתנד פנימי מכויל.

העמודה החמישית מראה שבכולם יש מתנד בתדר נמוך.

העמודה השישית שקיים בכולם חוצץ (RAM בגודל 1K)המאפשר תקשורת עם USB עם מיקרו בקר.

העמודה השביעית מראה שבכל הרכיבים קיים מייצב מתח ל 3.3 וולט.

העמודה השמינית מראה שבכולם יש 2 ערוצי SMBus שנקרא גם תקשורת IIC או I2C .

העמודה התשיעית מראה שלכולם יש תקשורת SPI מועשרת.

העמודה העשירית מראה שלכולם יש 2 ערוצי תקשורת טורית UART .

העמודה ה 11 מראה שלכולם יש 6 טיימרים , כל טיימר של 16 ביט .

העמודה ה 12 מראה שלכולם יש מערך מונים מתוכנת.

העמודה ה 13 מראה כמה הדקי קלט פלט יש . ברכיב C8051F380 יש 40 הדקי קלט/פלט. ב C8051F381 רק 25 הדקים.

העמודה ה 14 מראה לאילו מיקרו בקרים יש ממשק לחיבור זיכרון חיצוני (הרחבת זיכרון) . ב C8051F380 יש ממשק כזה.

העמודה ה 15 מראה למי יש ADC של 10 ביט עם 500 – 500K sps אלף דגימות בשנייה.

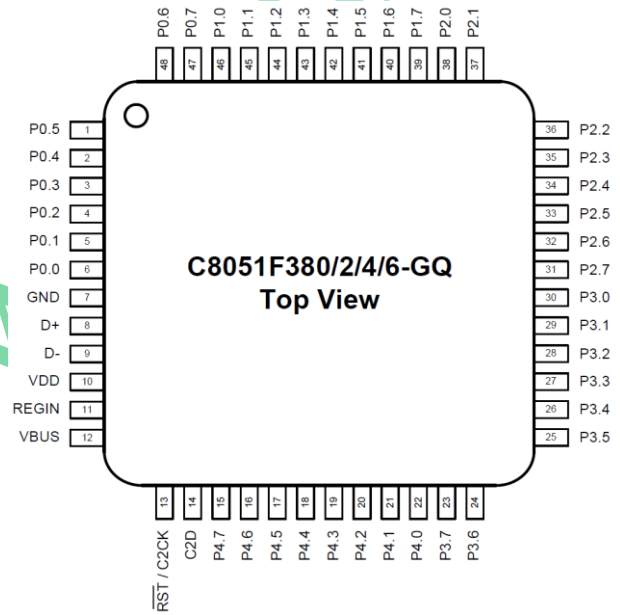
העמודה ה 16 מראה למי יש חיישן טמפרטורה פנימי.

העמודה ה 17 מראה למי יש מתח ייחוס עבור ה ADC .

העמודה ה 18 מראה שלכולם יש 2 משווים אנאלוגיים.

העמודה האחרונה מציינת את סוג הזיווד (האריזה) של המיקרו בקר.

1.6 : הדקי הרכיב במבנה TQFP - 48



איור מספר 2 – הדקי הרכיב במבנה TQFP – 48

הרכיב באיור הוא בזיווד TQFP - Thin Quad Flat Pack - חבילה שטוחה דקה מרובעת שבה 48 הדקים. העיגול בצד שמאל למעלה מראה את הדק מספר 1 ממנו מתחילים לספור את מספרי ההדקים נגד כיוון השעון. בכל צד יש 12 הדקים. הזיווד הזה נכון לרכיבים המסתיימים ב 0 , 2 , 4 ו 6 . הרכיבים המסתיימים ב 1 , 3 , 5 , 7 הם בזיווד LQFP – 32 - Low Profile Quad Flat Package - חבילה שטוחה מרובעת בעלת פרופיל נמוך . באריזה זו יש 32 הדקים. קיימים גם זיוודים נוספים.

באיור רואים שבנוסף ל 4 הפורטים מ P0 ועד P3 שהיה ב 51 "הישן" נוסף כאן פורט חמישי הנקרא P4 שגם לו 8 הדקים.

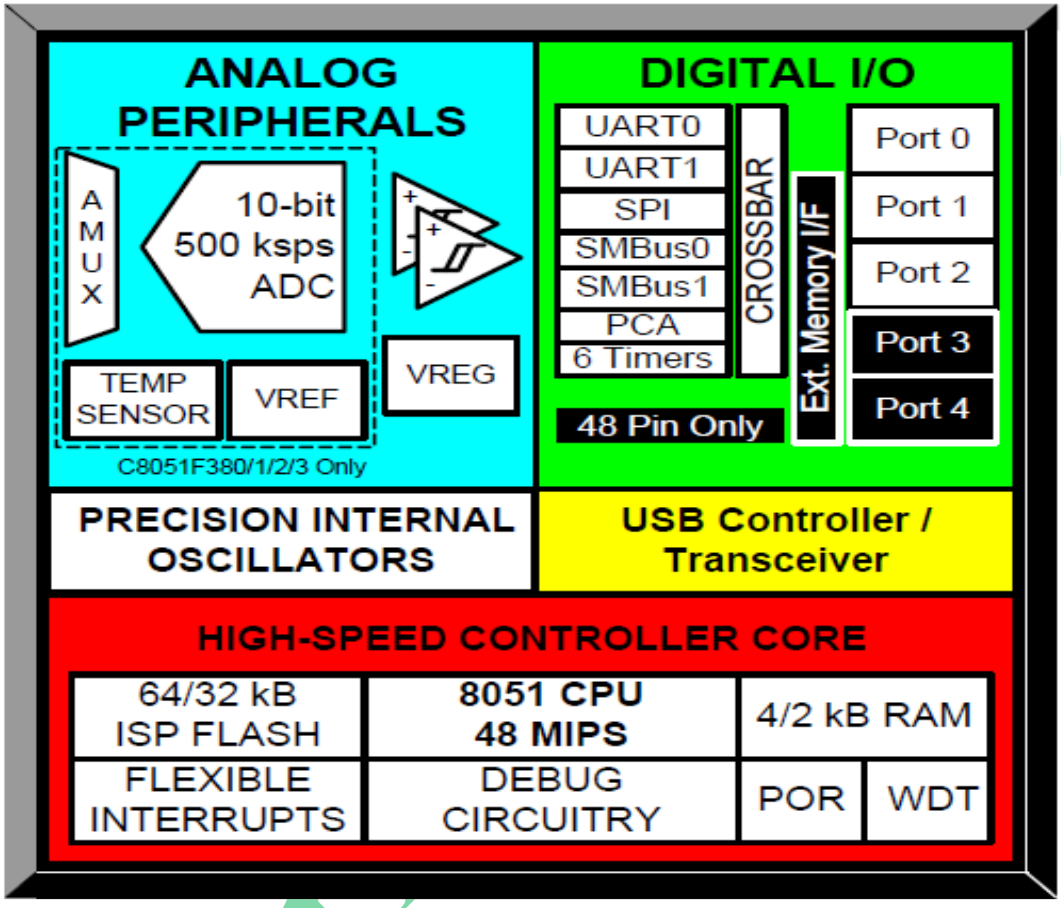
1.7 : משפחות של מיקרו בקרים נוספים

לחברות נוספות יש מיקרו בקרים שהליבה מכילה 51 בסיסי והיום יש יצרנים שלהם מיקרו בקרים שלא מתבססים על ה 51 המקורי אלא מיקרו בקרים של החברה עצמה. לחלק מהחברות ישנן סדרות שמתבססות על ה 51 וסדרות שלא מבססות על ה 51 המקורי. לרובן יש מיקרו בקרים של 8 ביטים, 16 ביטים ו 32 ביטים. בין החברות שמייצרות מיקרו בקרים נזכיר את : Altera , AMD , Analog devices, Arduino , Atmel , Cygnal , Cypress , Dallas , Elan , Epson , Espressif , Freescale, Infineon , Microchip , Holtek , Nec, National Phillips, Siemens, Signetics ועוד.

פרק 2 : הארכיטקטורה של המיקרו בקר C8051F380

2.1 : סכמה מלבנית של הרכיב C8051F380

באיור 3 מתוארת הסכמה המלבנית של המיקרו בקר C8051F380



איור 3 : המלבנים המרכיבים את המיקרו בקר C8051F380

באיור רואים 5 אזורים שונים (לפי הצבעים השונים) והם:

א. **HIGH-SPEED CONTROLLER CORE** - הליבה של הבקר העובד במהירות גבוהה (צבע אדום)

ב. **PRECISION INTERNAL OSCILLATOR** - מתנד פנימי מדויק. (בצבע לבן).

ג. **USB CONTROLLER TRANSCEIVER** – בקר משדר מקלט USB . (בצבע צהוב)

ד. **ANALOG PERIPHERALS** – רכיבים היקפיים אנאלוגיים.

ה. **DIGITAL I/O** - קלט/פלט דיגיטאליים.

2.1.1 : הליבה – CORE – של הבקר

החלק בצבע האדום הוא הליבה - core - של ה 51 . הוא נקרא HIGH-SPEED CONTROLLER CORE – ליבה של בקר מהירות גבוהה. בחלק הזה של המיקרו בקר יש את המלבנים הבאים :

2.1.2 : 64/32 kB ISP FLASH - זיכרון תכנית בגודל 32 או 64 קילו בתים (תלוי במיקרו בקר לפי טבלה 2) מסוג FLASH שניתן לתכנת/לצרוב אותו ISP – In System Programming - שאומר תכנות בתוך המערכת כלומר ללא הוצאת המיקרו בקר מהמעגל שהוא נמצא (בעבר היו מיקרו בקרים שכדי לתכנת אותם היה צריך להוציא אותם מהמעגל שבו הם נמצאים ולתכנת אותם בעזרת מכונה מיוחדת שנקראת Programmer).

2.1.3 : 8051 CPU 48 MIPS – יחידת עיבוד מרכזית מסוג 8051 עם מהירות עבודה של 48 מיליון הוראות בשנייה . מדובר במיקרו בקר שחיברנו אליו גביש בתדר 48MHz (הגביש הכי מהיר שהמיקרו יכול לעבוד אתו) ונותנים לו פקודות המבוצעות במחזור שעון אחד. (יש פקודות שנמשכות מספר מחזורי שעון).

2.1.4 : 4/2 kB RAM - זיכרון RAM בגודל 4 קילו (או 2 קילו בתלות במספר המיקרו).

2.1.5 : FLEXIBLE INTERRUPTS - מערכת פסיקות גמישה.

2.1.6 : DEBUG CIRCUITRY – מעגל ניפוי שגיאות. עוזר למשתמש לפעולות DEBUG .

2.1.7 : Power On Reset – POR - איפוס בהפעלת חשמל - מעגל שדואג בהפעלת חשמל לתת reset לזמן קצר למיקרו. בזמן ה reset התוכנית איננה מתחילה לרוץ אבל מבוצעות פעולות אתחול בתוך המיקרו בקר כדי להביא אותו למצב אתחול ראשוני (לדוגמה : מונה התוכנית מאופס, הפסיקות לא מאופשרות, מונים לא סופרים ועוד).

2.1.8 : WDT - Watch Dog Timer – טיימר כלב שמירה – הוא טיימר המשמש כדי לזהות ולהתאושש כאשר קורית תקלה . במהלך ריצת התוכנית הטיימר מתאפס באופן קבוע כל פרק זמן מסוים . כאשר נוצרת בעיה , כמו תוכנה "תקועה" , ולא הגענו לאיפוס הטיימר מתקבלת פסיקת WDT - "כלב שמירה" ומתבצעת פעולת אתחול התוכנית מחדש.

2.1.9 : PRECISION INTERNAL OSCILLATOR

במיקרו בקר יש מתנד פנימי מכויל לתדר של 48 מגה הרץ.

2.1.10 : USB CONTROLLER TRANSCEIVER – בקר משדר מקלט USB .

קיים בקר USB המאפשר התחברות לרכיבים תואמי USB 2.0 . ניתן לעבוד עם קצב גבוה של 12Mbps או קצב נמוך של 1.5Mbps .

2.1.11 : ANALOG PERIPHERALS – רכיבים היקפיים אנאלוגיים.

בחלק האנלוגי יש את המלבנים הבאים :

- ממיר מאנאלוגי לדיגיטאלי - ADC - של 10 ביט עם עד 500ksps (500 אלף אלף דגימות בשנייה).
- מרבב אנאלוגי - AMUX – עם אפשרות לבחור האם הכניסה ל ADC תהיה דיפרנציאלית (הפרש מתח בין 2 הדקים) או בין הדק לאדמה .
- חיישן טמפרטורה .
- מתחי ייחוס .
- 2 משווים.
- מייצב מתח VREG .

2.1.12 : DIGITAL I/O - קלט/פלט דיגיטאליים.

כאן נמצא את הדברים הבאים:

- 5 פורטים לקלט/פלט מפורט 0 ועד פורט 4 .
- EMIF - External Memory Interface- יחידה לממשק עם זיכרונות חיצוניים.
- 2 יחידות לתקשורת I2C (נקראות גם SMBUS)
- 2 יחידות תקשורת UART .
- תקשורת SPI מועשרת
- 6 טיימרים של 16 ביטים.
- מערך טיימרים/מונים בר תכנות.
- יחידת CROSSBAR שמקשרת בין המעגלים בתוך המיקרו בקר להדקים החיצוניים של הרכיב לפי רצון המשתמש.

2.2 : ארגון הזיכרון

ארגון הזיכרון דומה לזה של רכיב 8051 סטנדרטי. קיימים 2 מרחבי זיכרון נפרדים:

1. זיכרון תוכנית – Program Memory נקרא גם זיכרון קוד . **2. זיכרון נתונים – Data Memory.**

שניהם חולקים אותו מרחב כתובות אבל ניגשים אליהם באמצעות הוראות שונות. לדוגמה : לזיכרון התוכנית נפנה עם פקודת MOVC כאשר ה C בפקודה מבטא Code (קוד תוכנית) ואילו לזיכרון הנתונים נפנה עם פקודת MOV או MOVX.

ניתן לבצע פירוט נוסף של סוגי הזיכרון השונים שבתוך המיקרו וניתן גם הוסיף זיכרון חיצוני. החלוקה המפורטת היא :

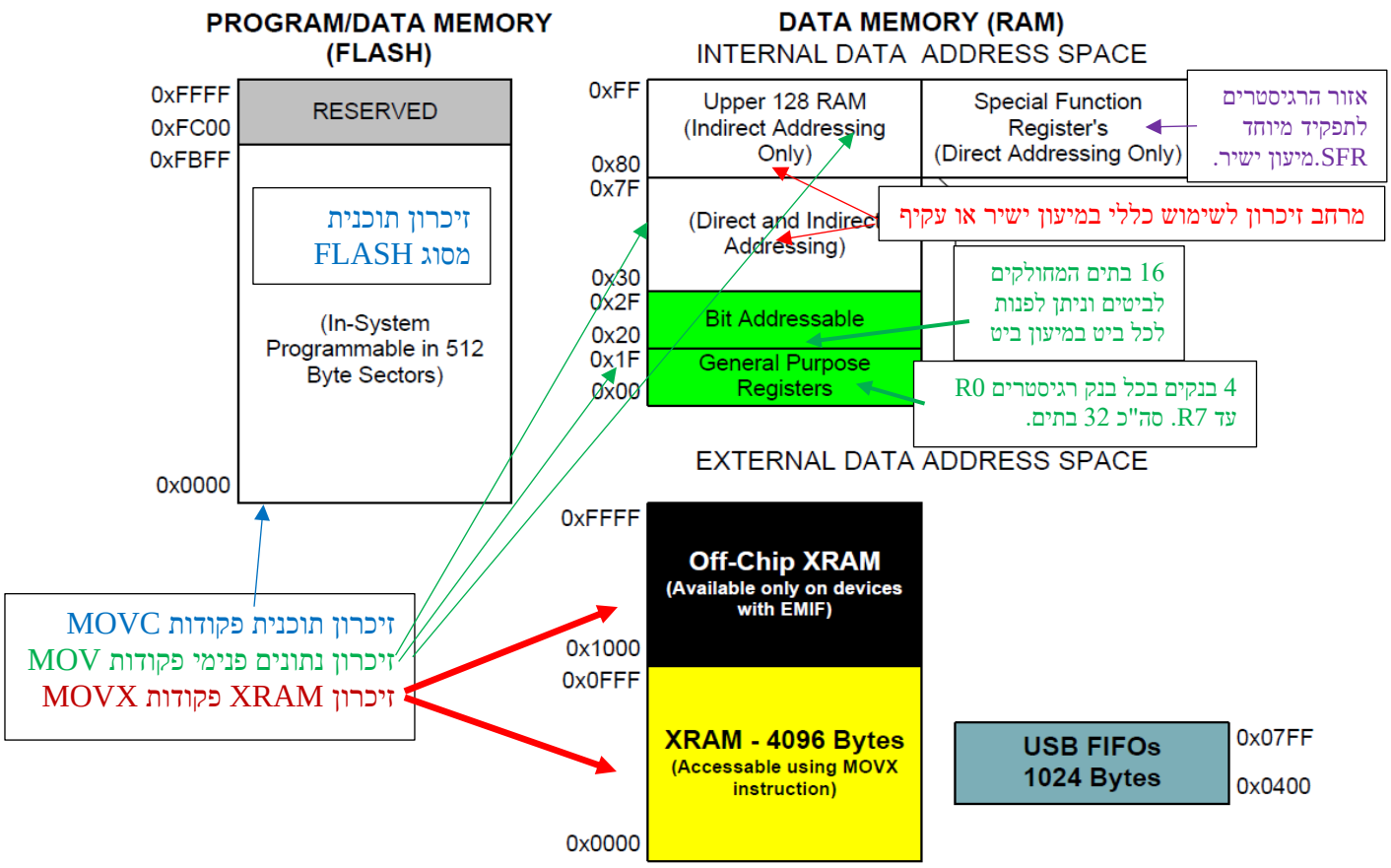
א. זיכרון תוכנית מסוג **FLASH** . בזיכרון זה נמצאת התוכנית ובנוסף הוא מכיל קבועים שישמשו במהלך ריצת התוכנית. זהו זיכרון בגודל 64 קילו בתים (ברכיבים המסתיימים בספרה זוגית 0/2/4/6) , הוא בלתי נדיף - None Volatile - והוא נקרא גם זיכרון קוד - Code Memory.

ב. זיכרון נתונים פנימי – מסוג RAM . בזיכרון זה נמצאים משתני התוכנית , המחסנית , נתונים/ארגומנטים המועברים אל פונקציות (נסביר בהמשך כשנדבר על פונקציות בשפת C) , 4 בנקים ועוד.

ג. אזור הרגיסטרים לתפקידים מיוחדים – SFR - Special Functions Registers . אלו 128 בתים שמכילים את הרגיסטרים של הפורטים ושאר רגיסטרים של העברת נתונים (כמו A ו B) ורגיסטרים של שליטה ובקרה על הרכיבים הפריפריאליים הפנימיים.

ד. זיכרון נתונים חיצוני הנקרא XRAM . ה X אומר eXternal – חיצוני אבל למעשה יש 4 קילו בייתים בתוך הרכיב וניתן להוסיף עוד 60 קילו בייתים מחוץ לרכיב.

האיור הבא – איור 4 - מתאר את ארגון הזיכרון ברכיב C8051F380 :



איור 4 : ארגון הזיכרון ב CIP-51 של ה C8051F380

2.2.1 : זיכרון התוכנית מסוג FLASH

נפח זיכרון התוכנית הוא 64Kbytes מסוג FLASH – הבזק . (64K הם 65536 בתים. ליתר דיוק יש 65512 בתים לשימוש המתכנת כי בחלק העליון יש 512 בתים שמורים לפעולות ניפוי). מתייחסים לזיכרון התוכנית כזיכרון ROM – Read Only Memory – זיכרון קריאה בלבד אבל ברכיב הזה ניתן גם לכתוב על ידי הגדרת ביט PSCTL.0 (Program Store Write Enable bit - ביט אפשור כתיבה בזיכרון אחסון – זיכרון התוכנית). משתמשים לכתיבה בפקודת MOVX. ברכיב 51 של אינטל היה זיכרון קטן בתוך הרכיב ואם היה צריך עוד זיכרון הוסיפו זיכרון תכנית חיצוני (עד 64 קילו בתים).

בפקודה MOVX (קיצור של MOV to eXternal) השתמשו ברכיב 51 המקורי כדי "לדבר" עם **זיכרון נתונים** חיצוני ולא עם זיכרון תוכנית שהיה מסוג ROM ולא ניתן לכתוב אליו. ב CIP-51 כל זיכרון התוכנית הוא פנימי – בתוך הרכיב – וניצלו את הפקודה

MOVX ואת הביט PSCTL.0 כדי לכתוב נתונים/תוצאות לזיכרון לא נדיף (None Volatile) כך שגם אם החשמל ייכבה הנתונים/תוצאות יישמר בזיכרון ולא ייעלמו.

2.2.2 : זיכרון הנתונים – Data Memory

הערה : בהמשך מופיע המושג מיעון (addressing) שעליו נסביר בפרק התוכנה. ל Byte נקרא גם רגיסטר (או אוגר). בזיכרון ה -CIP 51 יש 256 בתים של RAM פנימי מכתובות 0 עד 255 (0 עד FFH הסימול H - הקסה דצימאלי) . 128 הבתים הראשונים משמשים כרגיסטרים לשימוש כללי , קוראים להם אזור הנתונים הפנימי הנמוך וניתן לגשת אליהם במיעון ישיר או עקיף (נושא המיעון ידובר בהרחבה בהמשך בפרק 4 המתאר את האסמבלי של התוכנה) . הבתים מ 0 עד 31 (מ 0 עד 1FH) משמשים כ 4 בנקים לשימוש כללי. בכל בנק 8 רגיסטרים מ R0 ועד R7 .

16 הבתים מ 20H ועד 2FH (32 עד 47) הם בתים המחולקים לביטים וניתן לגשת אליהם במיעון ביט. שאר הבתים מ30H עד 7FH (48 עד 127) הם בתים לשימוש כללי.

ל 128 הבתים הגבוהים, שגם הם לשימוש כללי, קוראים אזור הנתונים הפנימי הגבוה וניתן לגשת אליהם רק בעזרת מיעון עקיף (בעזרת רגיסטר המראה מהי הכתובת אליה פונים).

בזיכרון הפנימי קיים אזור הנקרא – Special Functions Registers - SFR – רגיסטרים לתפקידים מיוחדים. בעזרת הרגיסטרים האלו מבקרים/שולטים על הרכיבים הפריפריאליים שבתוך הרכיב (טיימרים פסיקות וכו'). לאזור זה אותן כתובות כמו לאזור הנתונים הגבוה - מ 80H עד FFH (128 עד 255) . בהתאם לסוג הפקודה יודעים לאיזו כתובת באיזה אזור פונים. אם המיעון ישיר פונים לכתובת באזור ה SFR ואם המיעון עקיף פונים לכתובת באזור הנתונים הפנימי הגבוה.

באיור מספר 4 נפרט כיצד נראים 128 הבתים הראשונים (האזור הנמוך) של RAM הנתונים הפנימי. מהאיור רואים ש 32 בתים ראשונים – כתובות 0 עד 31 או 0 עד 1FH ב HEX – משמשים כ 4 בנקים. כל 8 בתים הם בנק. בכל בנק יש את רגיסטרים R0 עד R7 בהתאמה. לדוגמה : כתובת 0 היא R0 של בנק 0 וכתובות 8 , 16, 24 הן הכתובות של R0 בבנקים 1, 2 , 3 בהתאמה. R5 של בנק 0 נמצא בכתובת 5 ואילו R5 של בנק 3 נמצא בכתובת 29 וכך הלאה.

דוגמה לתפקיד הבנקים הוא לשמש כמאגר נתונים קטן כאשר עוברים בין פרוצדורות (באסמבלי) או פונקציות (בשפת C). לכל פרוצדורה יהיה בנק נתונים קטן משלה. המעבר מבנק לבנק נשלט על ידי המתכנת בעזרת 2 הביטים RS0 ו RS1 הנמצאים ברגיסטר PSW שעליו נדבר בהמשך. בשלב לימוד התוכנה נלמד כיצד לפנות אל הביטים האלה.

פנייה לבנקים בעזרת ביטים RS0 RS1 באוגר PSW

הנמצא באזור הרגיסטרים המיוחדים SFR.

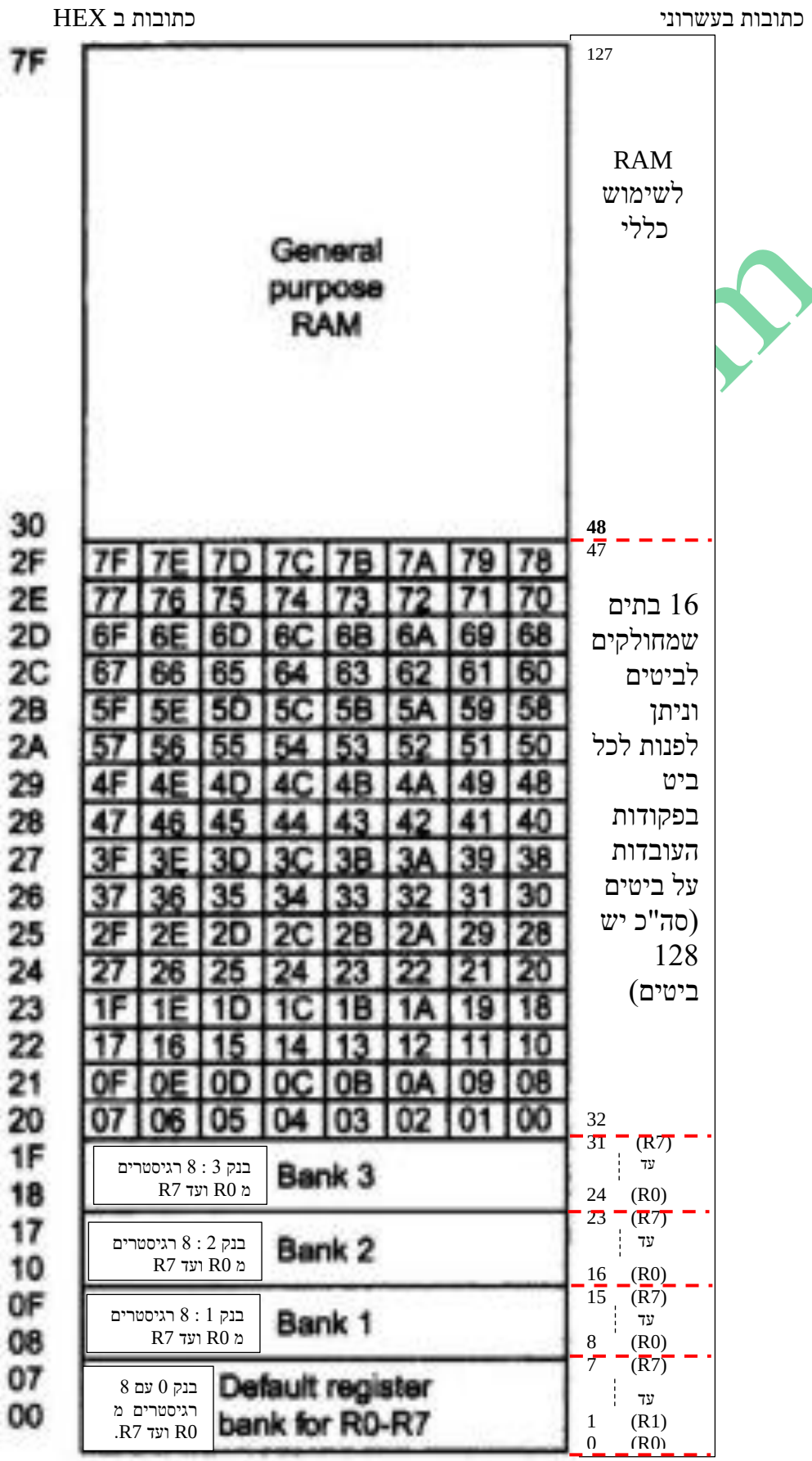
RS1	RS0	בנק	
0	0	0	אחרי RESET RS1=RS0=0
0	1	1	
1	0	2	
1	1	3	

טבלה 3 - הפנייה אל הבנקים בעזרת הביטים RS1 RS0 .

אזור הכתובות 32 (20H) עד 47 (3FH) הוא אזור המחולק לביטים. ניתן לפנות לכל 8 הביטים בכל כתובת בפקודה אחת או לקבוע מצב כל אחד מהביטים בנפרד בעזרת מיעון ביט (נסביר בפרק התוכנה).

הכתובות מ 48 (30H) ועד 127 (7FH) הן RAM לשימוש כללי .

איור 5 מתאר את הנאמר.



איור 5 : מבנה 128 הבתים הנמוכים של זיכרון הנתונים הפנימי

2.2.3 : אזור הרגיסטרים לתפקידים מיוחדים – SFR (הכתובות מ 80H עד FFH).

אזור ה SFR - **Special Functions Registers** , הוא אזור הרגיסטרים לתפקידים המיוחדים שברכיב. הוא נמצא בתחום הכתובות מ 80H ועד FFH. הסיבה לכך היא שבמיקרו 51 המקורי הראשון היו 128 בתים של RAM (מ 0 עד 7FH) ומעליו היה ה SFR (מ 80H ועד FFH) שגם הוא RAM . כל רגיסטר הוא של 8 ביטים. הם נקראים "תפקידים מיוחדים" כי באזור זה נמצאים הרגיסטרים השולטים על הפסיקות, על סוגי התקשורת השונים (טורית, USB ,SPI ,SMBUS) , על הטיימרים , הפורטים עצמם וכו'. ב 51 המקורי אזור ה SFR לא היה רציף כי היו לו פחות מ 128 רגיסטרים. במיקרו C8051F380 יש כמות רגיסטרים גדולה מ 128 וכדי שתהיה תאימות עם ה 51 המקורי הוסיפו עוד 128 בתים של אזור SFR ונתנו אפשרות ל"דפדוף" – paging - בין שני האזורים. השאירו את כל הרגיסטרים "הישנים" באותן הכתובות והוסיפו חלק מהרגיסטרים החדשים בכתובות שלא השתמשו בהן ב 51 המקורי וכדי להגיע לשאר הרגיסטרים שנשארו נתנו אפשרות של – paging - דף . "עוברים דף" ובכתובות מסוימות נמצאים הרגיסטרים הנוספים.

למעשה, ב CIP 51 נותנים לנו אפשרות ל 256 דפים שונים (אולי אופציות לעתיד ??) כאשר בכל דף יש את הכתובות 80H עד FFH . ב C8051f380/1/2/3/4/5/6/7 משתמשים בשני דפים בלבד. בדף 0 ובדף 15 (F בהקסה דצימלי). חלק מהרגיסטרים קיים גם בדף 0 וגם בדף F (בהקסה) . הרגיסטר שקובע עם איזה דף עובדים הוא SFRPAGE והוא נראה באיור הבא :

Bit	7	6	5	4	3	2	1	0
Name	SFRPAGE[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xBF; SFR Page = All Pages

Bit	Name	Function
7:0	SFRPAGE[7:0]	**SFR Page Bits.** Represents the SFR Page the C8051 core uses when reading or modifying SFRs. Write: Sets the SFR Page. Read: Byte is the SFR page the C8051 core is using.

איור 6 : רגיסטר SFRPAGE הקובע עם איזה דף עובדים.

הרגיסטר נמצא בכתובת 0xBF בכל הדפים. אחרי RESET עובדים עם דף 0 . כדי לעבור דף במיקרו C8051F380 יש לעבור לדף F ונרשום : SFRPAGE=0x0FH; או באסמבלי : MOV SFRPAGE,#0FH .

בטבלה הבאה מתואר אזור ה SFR עם הדפדוף שלו:

Address	Page	0(8)	1(9)	2(A)	3(B)	4(C)	5(D)	6(E)	7(F)
F8		SPI0CN	PCA0L	PCA0H	PCA0CPL0	PCA0CPH0	PCA0CPL4	PCA0CPH4	VDM0CN
F0		B	P0MDIN	P1MDIN	P2MDIN	P3MDIN	P4MDIN	EIP1	EIP2
E8		ADC0CN	PCA0CPL1	PCA0CPH1	PCA0CPL2	PCA0CPH2	PCA0CPL3	PCA0CPH3	RSTSRC
E0	0	ACC	XBR0	XBR1	XBR2	IT01CF	SMOD1	EIE1	EIE2
	F					CKCON1			
D8		PCA0CN	PCA0MD	PCA0CPM0	PCA0CPM1	PCA0CPM2	PCA0CPM3	PCA0CPM4	P3SKIP
D0		PSW	REF0CN	SCON1	SBUF1	P0SKIP	P1SKIP	P2SKIP	USB0XCN
C8	0	TMR2CN	REG01CN	TMR2RLL	TMR2RLH	TMR2L	TMR2H	SMB0ADM	SMB0ADR
	F	TMR5CN		TMR5RLL	TMR5RLH	TMR5L	TMR5H	SMB1ADM	SMB1ADR
C0	0	SMB0CN	SMB0CF	SMB0DAT	ADC0GTL	ADC0GTH	ADC0LTL	ADC0LTH	P4
	F	SMB1CN	SMB1CF	SMB1DAT					
B8	0	IP	CLKMUL	AMX0N	AMX0P	ADC0CF	ADC0L	ADC0H	SFRPAGE
	F		SMBTC						
B0		P3	OSCXCN	OSCICN	OSCICL	SBRLL1	SBRLH1	FLSCL	FLKEY
A8		IE	CLKSEL	EMI0CN		SBCON1		P4MDOUT	PFE0CN
A0		P2	SPI0CFG	SPI0CKR	SPI0DAT	P0MDOUT	P1MDOUT	P2MDOUT	P3MDOUT
98		SCON0	SBUF0	CPT1CN	CPT0CN	CPT1MD	CPT0MD	CPT1MX	CPT0MX
90	0	P1	TMR3CN	TMR3RLL	TMR3RLH	TMR3L	TMR3H	USB0ADR	USB0DAT
	F		TMR4CN	TMR4RLL	TMR4RLH	TMR4L	TMR4H		
88		TCON	TMOD	TL0	TL1	TH0	TH1	CKCON	PSCTL
80		P0	SP	DPL	DPH	EMI0TC	EMI0CF	OSCLCN	PCON
		0(8)	1(9)	2(A)	3(B)	4(C)	5(D)	6(E)	7(F)

Notes:
1. SFR Addresses ending in 0x0 or 0x8 are bit-addressable locations and can be used with bitwise instructions.
2. Unless indicated otherwise, SFRs are available on both page 0 and page F.

טבלה 4 : אזור הרגיסטרים לתפקידים מיוחדים – SFR

בעמודה השמאלית של הטבלה נמצאות כתובות הרגיסטרים ב HEX .

העמודה הבאה מראה שלחלק מהרגיסטרים יש דף . כל שורה מייצגת 8 רגיסטרים. כל מלבן הוא אחד מהרגיסטרים ב SFR ויש לו 8 ביטים. כל שורה מתארת 8 רגיסטרים . לדוגמה בשורה לפני אחרונה בטבלה , בכתובת 80H נמצא את פורט 0 (P0), בכתובת 81H נמצא מצביע המחסנית (Stack Pointer – SP), בכתובת 82H נמצא DPL וכך הלאה עד שבכתובת 87H נמצא את רגיסטר PCON.

בשורה מעליה בכתובת 88H נמצא רגיסטר TCON , בכתובת 89H נמצא ה TMOD בכתובת 8AH נמצא TL0 וכך הלאה עד שבכתובת 8FH נמצא את רגיסטר PSCTL.

בחלק מהכתובות ניתן לראות שיש דפדוף. לדוגמה : בכתובת 90H נמצא רגיסטר P1 (פורט 1). הוא נמצא גם בדף 0 וגם בדף FH . בכתובת 91H נמצא בדף 0 את TMR3CN שהוא רגיסטר הבקרה של טיימר 3 ובאותה כתובת 91H בדף FH נמצא רגיסטר TMR4CN שהוא רגיסטר הבקרה של טיימר 4 .

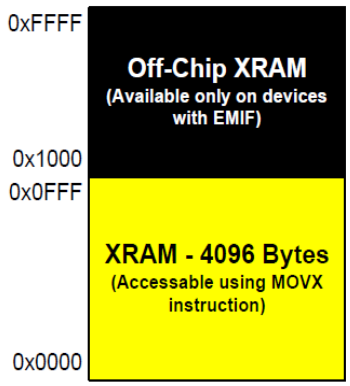
לחלק מהרגיסטרים ניתן לפנות בפקודות העובדות עם ביטים. רגיסטרים אלו נמצאים בכתובות המתחלקות ב 8 ללא שארית .

לדוגמא : הרגיסטר בכתובת 80H הוא ACC ונקרא אקומולאטור – ACCUMULATOR . הוא נמצא בכתובת E0H . הוא כמובן רגיסטר של 8 ביט כאשר ביט ה LSB שלו נמצא בכתובת 80H, הביט הבא שלו נמצא בכתובת 81H וכך הלאה עד ביט ה MSB שמספרו 87H. בפרק התוכנה נראה שאפשר לרשום פקודה clr P0.0 או פקודה זהה - clr 80H כדי לשים 0 בביט הנמוך שלו . כדי לשים 1 בביט נרשום SETB 80H או SETB P0.0 .

2.2.4 : ממשק עם זיכרון נתונים חיצוני ו XRAM בתוך הרכיב.

בנוסף ל RAM הנתונים הפנימי של 256 כתובות הקיים בתוך הרכיב , ניתן לחבר עוד 64 קילו בתים של זיכרון RAM חיצוני. ברכיב ה 51 המקורי של אינטל כל זיכרון הנתונים החיצוני היה מחוץ לרכיב. הפנייה לזיכרון ה RAM החיצוני היה עם פקודת **MOVX** כאשר ה X אומר שהזיכרון חיצוני – e**X**ternal . ב CIP-51 קיים זיכרון הנקרא XRAM "חיצוני" בגודל 2 קילו בתים (2048) **בתוך** הרכיב וניתן להתחבר לעוד 62 קילו בתים של RAM חיצוני. הייתרון הוא שבחלק מהתוכניות 256 הבתים של ה RAM הפנימי לא מספיקים ואז צריך לחבר עוד RAM חיצוני (דבר שמייקר את העלות וגם מאט את המהירות). ב CIP-51 יש 2 קילו בתים של XRAM פנימי וזה חוסך את הצורך להוסיף RAM חיצוני. גם ל RAM הפנימי וגם ל RAM החיצוני פונים בעזרת פקודות MOVX . האלקטרוניקה שבתוך הרכיב שנקראת **EMIF** - **E**xternal **M**emory **I**nter**F**ace יודעת שבפקודת MOVX מכתובת 0 ועד כתובת 7FFH (2047 עשרוני) פונים לזיכרון ה RAM הפנימי ומכתובת 800H ועד FFFFH לזיכרון הנתונים ב RAM החיצוני. האלקטרוניקה הזו קיימת ברכיבים C8051F380/2/4/6 . ניתן למפות את החלק הגבוה ב XRAM הפנימי (מכתובת 400H עד 07FFH – סה"כ 1024 בתים) לעבודה עם USB כך שמעשית הוא מקטין את נפח הזיכרון עבור המשתמש.

איור 6 שהוא חלק מאיור 3 מתאר את זיכרון הנתונים החיצוני.



איור 6 : זיכרון הנתונים XRAM .

החלק הצהוב באיור הוא של זיכרון ה RAM שבתוך הרכיב והחלק השחור הוא מחוץ לרכיב.

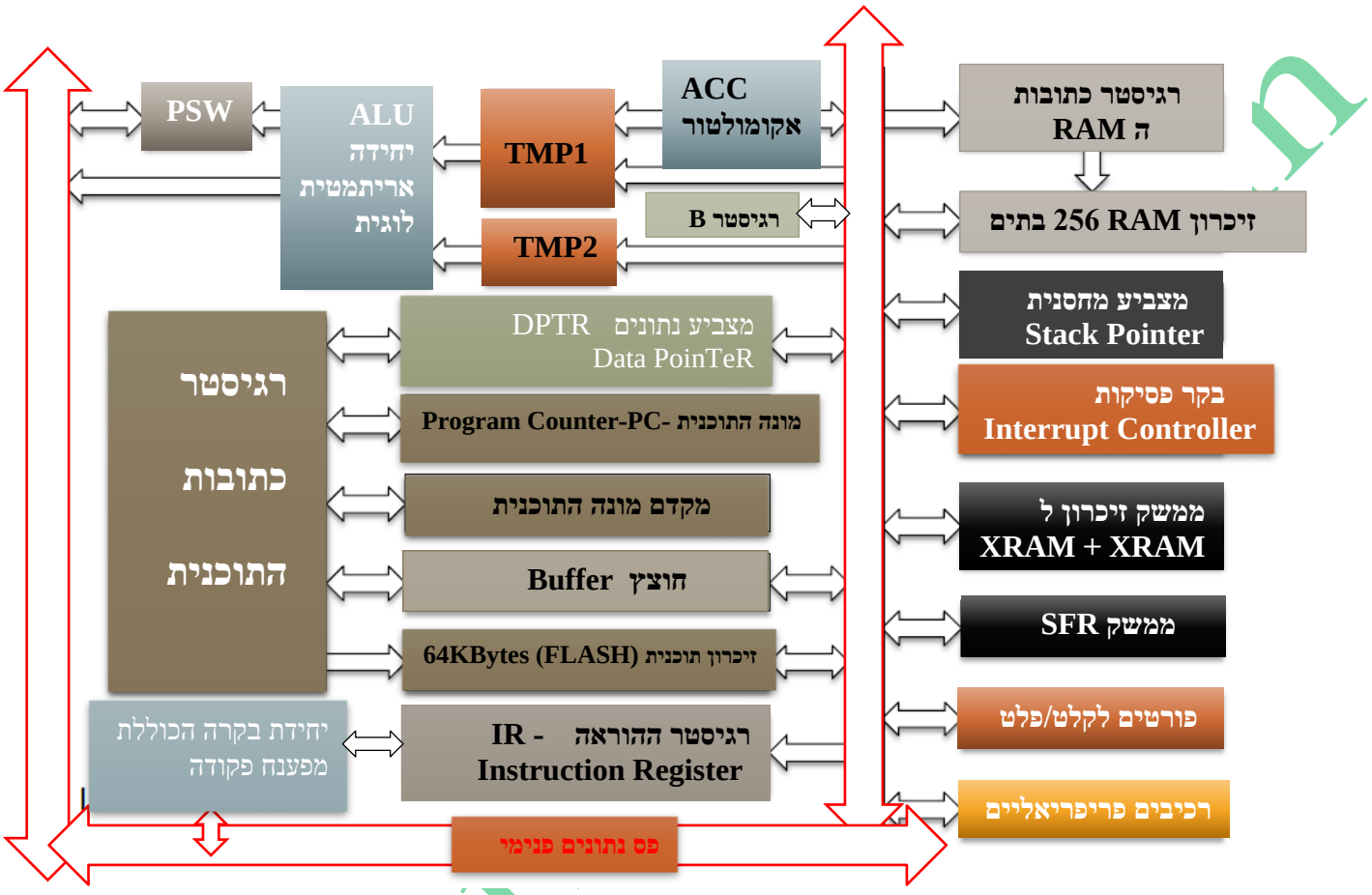
הזכרנו שהפקודה לעבודה עם ה XRAM היא MOVX וקיים רגיסטר הנקרא DPTR (Data pointer – מצביע נתונים) שמצביע על הכתובת אליה פונים ב XRAM . ישנן פקודות MOVX שיודעות להשתמש במיעון עקיף עם R0 או R1 כמצביעים לכתובת ב XRAM . במקרה כזה ניתן לחשוב שאפשר לגשת רק ל 256 הכתובות הנמוכות כי R0 או R1 הם רגיסטרים של 8 ביט אבל ניתן להוסיף את הביית הגבוה של הכתובת בעזרת רגיסטר EMI0CN (נסביר בהמשך).

הערה : ראינו בסעיפים קודמים שניתן לכתוב גם לזיכרון תוכנית עם פקודת MOVX במקרים מסוימים. ברירת המחדל של הפקודה היא ה XRAM . המלבן **USB FIFOs 1024 Bytes** משמש כאשר עובדים עם תקשורת USB . גודלו 1 קילו בתים והוא תופס את החלק הגבוה של ה XRAM שבתוך הרכיב.

2.2.5 : ארכיטקטורה של הליבה CIP-51

כאמור, בכל מיקרו בקר ממשפחת 8051Fxxx של חברת Silicon Labs יש ליבה של 51 הנקראת CIP-51 .

באיור מספר 7 מתואר המבנה הפנימי של הליבה .



איור 7 : ארכיטקטורה של ה CIP-51

האיור מראה שהארכיטקטורה כמעט זהה ל 8051 המקורי של אינטל, ובכך קיימת שמירה על תאימות מבחינת ביצוע הפקודות.

נתאר את המלבנים המרכיבים את הליבה.

DATA BUS - פס נתונים פנימי

החיצים בצבע אדום מתארים את פס הנתונים הנמשך ומתחבר אל כל היחידות באיור. זהו פס נתונים פנימי של 8 ביט . נתון מועבר בין היחידות השונות בפס הזה.

נמשיך ונתאר את המלבנים בחלק העליון מהמרכז שמאלה באיור.

ACCUMULATOR - ACC – צובר

נמצא במרכז למעלה של האיור מתחת לרגיסטר B. זהו רגיסטר שנמצא באזור ה SFR . זהו הרגיסטר "שעובד הכי קשה" . כמעט כל פעולות העברת הנתונים עם הזיכרונות השונים , הפעולות האריתמטיות והלוגיות והעברת הנתונים עם הרכיבים הפריפריאליים מתבצעות בעזרת הרגיסטר הזה.

TEMP1 , TEMP2 , ALU

TEMP הוא קיצור של TEMPorary – זמני. ה ALU היא היחידה Arithmatic Logic Unit – יחידה אריתמטית (חשבונית) לוגית. זוהי היחידה שבה מתבצעות הפעולות החשבוניות (חיבור , חיסור , כפל וחילוק) והפעולות הלוגיות (NOT , OR , AND ,XOR). הפעולות מתבצעות על שני אופרנדים (אופרנד - משתנה או קבוע) . אופרנד אחד מועבר אל TEMP1 והשני אל TEMP2 . הפעולה החשבונית או הלוגית מתבצעת ביחידת ה ALU והתוצאה מועברת בדרך כלל אל האקומולטור.

רגיסטר B :

רגיסטר הנמצא גם הוא ב SFR . רגיסטר העוזר בפעולות כפל וחילוק. בפקודת כפל MUL AB מתבצעת מכפלה בין הנתון שברגיסטר A – שהוא האקומולטור לנתון שברגיסטר B . החלק הנמוך (8 הביטים הנמוכים) של התוצאה נכנס ל A ו 8 הביטים הגבוהים ל B (ואם התוצאה גדולה מ 255 דגל OV מקבל 1). בפקודה DIV AB מתבצעת פעולת החילוק A לחלק ב B . התוצאה נכנסת ל A והשארית ל B . אם ב B יש 0 נקבל בדגל הגלישה 1 .

רגיסטר B יכול לשמש גם כרגיסטר לשימוש כללי .

Program Status Word – PSW – מילת מצב תכנית

זהו רגיסטר של 8 ביט. במעבדים אחרים הוא נקרא גם רגיסטר הדגלים. גם רגיסטר זה נמצא ב SFR . ברגיסטר יש 4 ביטים המשמשים כדגל ותפקידם לתת חיווי על התוצאה האריתמטית האחרונה שבוצעה ב ALU. 2 ביטים נוספים הם RS1 RS0 שקובעים עם איזה בנק עובדים. ל 2 ביטים נוספים אין תפקיד מיוחד.

נמשיך ונתאר את המלבנים בחלק הימני של האיור החל מהמרכז ומטה :

זיכרון RAM ומלבן רגיסטר כתובת ה RAM - RAM MEMORY and RAM ADDRESS REGISTER

ה RAM הוא זיכרון RAM הנתונים הפנימי . זהו RAM בגודל 256 בתים שהזכרנו בסעיפים קודמים. ה **RAM ADDRESS REGISTER** – רגיסטר הכתובת של ה SRAM מכיל את הכתובת אליה פונים ב RAM .

STACK POINTER – מצביע המחסנית

זהו רגיסטר הנמצא ב SFR והוא מראה באיזו כתובת נמצא הנתון האחרון שהוכנס למחסנית. המחסנית היא אזור בזיכרון ה RAM הפנימי שמשמש לשמירה של נתונים זמניים, שמירה של כתובות חזרה והעברת נתונים בעבודה עם פרוצדורות (באסמבלי) או פונקציות (בשפה עילית) . על המחסנית נדבר בהרחבה כשנעסוק בפקודות אסמבלי.

בקר פסיקות - INTERRUPT CONTROLLER

מערכת המטפלת בפסיקות, אפשור ועדיפות (על פסיקות נדבר בהמשך).

ממשק זיכרון ל XRAM - MEMORY INTERFACE + זיכרון XRAM

בעזרת ממשק זה פונים לזיכרון ה XRAM . כאן יש גם את הממשק וגם את הזיכרון עצמו בגודל 4Kbytes .

ממשק SFR - SRF BUS INTERFACE

ממשק פס של ה SRF (הרגיסטרים לתפקיד מיוחד). בעזרת ממשק זה מפעילים את אזור הרגיסטרים לתפקידים המיוחדים .

פורטים לקלט/פלט

פורט הוא מעגל אלקטרוני דרכו מעבירים נתונים מהמיקרו אל המעגלים האלקטרוניים מחוץ למיקרו. הפורט יכול להיות **פורט קלט** – מכניס נתון ממעגל חיצוני (לדוגמה : מפסק, לוח מקשים) למיקרו. לעומת פורט **פלט** (לדוגמה : מנורה, מנוע) שבעזרתו נוציא נתון מהמיקרו אל מעגל חיצוני. ל 8051F380 יש 5 פורטים , מפורט 0 ועד פורט 4 . כל אחד של 8 ביט.

רכיבים פריפריאליים

במיקרו בקרים ממשפחת C8051Fxxx יש בנוסף לליבה CIP-51 רכיבים פריפריאליים הנמצאים מסביב לליבה CIP-51 (להבדיל מרכיבים פריפריאליים הנמצאים מחוץ לרכיב המיקרו בקר) . רכיבים אלו יכולים להיות ADC , USB וממשק לרכיבי תקשורת טורית כמו SPI או/ו I2C .

בצד שמאל באיור, החל מהמרכז ומטה נמצא את המלבנים הבאים:

DATA POINTER מצביע נתון

רגיסטר בעזרתו פונים אל ה XRAM . הוא מראה לאיזו כתובת פונים ב XRAM . הרגיסטר נקרא גם בקיצור DPTR . יש לו 16 ביט והוא מורכב מ 2 רגיסטרים של 8 ביט. האחד נקרא DPH והשני DPL . ה H מציין את החלק הגבוה – High - (ביטים 8 עד 15) וה L מציין את החלק הנמוך – Low - (ביטים 0 עד 7) . אפשר להיעזר בו גם ליישומים אחרים . לדוגמה : מונה של 16 ביט בלולאות while או for .

מונה התוכנית ומקדם מונה התוכנית - PROGRAM COUNTER (PC) ו PC INCREMENTER

מונה התוכנית הוא רגיסטר של 16 ביט ומראה מאיזו כתובת בזיכרון התוכנית מביאים את הפקודה הבאה. בסיום הבאת הפקודה מונה התוכנית מקודם ב 1 על ידי ה **PC INCREMENTER** כדי להראות את כתובת הפקודה הבאה. את הכתובת שיש בו הוא מעביר לרגיסטר מתחתיו שנקרא PRGM. ADDRESS REG. .

BUFFER – חוצץ

מפריד בין פס הנתונים הפנימי ופס נתונים משני (מצד שמאל שלו) המתחבר אל היחידות שמתחתיו.

רגיסטר כתובת התוכנית - PROGRAM ADDRESS REGISTER

זהו רגיסטר שמראה לאיזו כתובת פונים בזיכרון התוכנית.

רגיסטר ההוראה - IR-Instruction Register

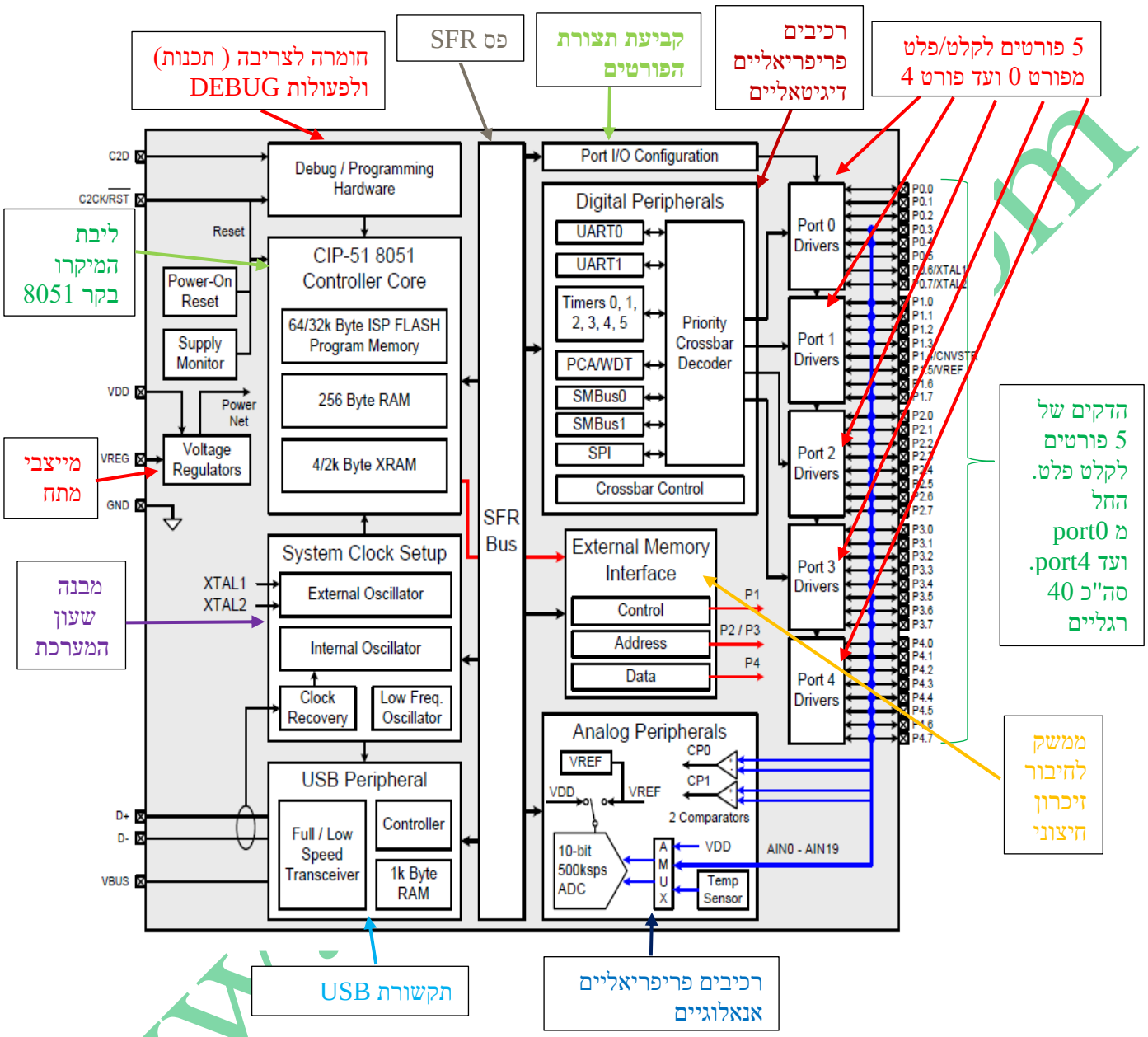
כאשר מביאים פקודה מזיכרון שומרים את הפקודה ברגיסטר ההוראה. שלב הבאת הפקודה נקרא FETCH .

יחידת הבקרה הכוללת מפענח פקודה - CONTROL UNIT AND INSRUCTION DECODER

זוהי היחידה השולטת על כל התהליכים במיקרו בקר. ביחידה זו יש את מפענח הפקודה המתרגם מהו אוסף האפסים והאחדים של הפקודה שברגיסטר ההוראה ובהתאמה היא יוצרת שרשרת פקודות ליחידות המתאימות ומבצעת את הפקודה.

2.3 : סכמה מלבנית של המיקרו בקר 8051F380/1/2/3/4/5/6/7

באיור הבא נתאר פירוט נוסף של היחידות המרכיבות את המיקרו בקר C8051F380 .



איור 8 – סכמה מלבנית של המיקרו בקרים C8051F380 , C8051F382, C8051F384 C8051F386

נסביר את המלבנים המרכיבים את הסכמה . נתחיל בצד שמאל למעלה ונלך עם כיוון השעון

2.3.1 : Debug Programming Hardware

חומרה לצריבה (תכנות) ולפעולות DEBUG - המלבן נמצא בצד שמאל למעלה . הוא משמש גם כהדק העוזר בתכנות (צריבה) של זיכרון הרכיב וגם לפעולות ניפוי Debug.

2.3.2 : SFR BUS – פס ה SFR

זהו פס המחבר בין הרגיסטרים באזור ה SFR ובין המערכות השונות במיקרו בקר. לדוגמה: ישנם מספר רגיסטרים השולטים על מערכת הפסיקות. כמו כן מספר רגיסטרים השולטים על הטיימרים וכו. הרגיסטרים נמצאים ב SFR ובעזרת הפס הרגיסטרים מגיעים למערכות השונות.

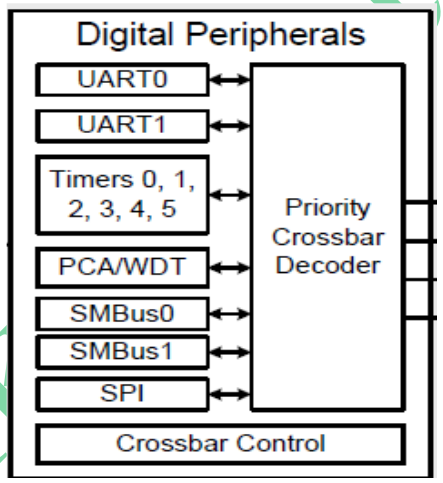
2.3.3 : PORT IO Configuration – קביעת תצורת הפורטים

בעזרת מלבן זה ניתן לקבוע את התצורה של כל הדק של פורט, כלומר האם הדק היצוני כלשהו של פורט יהיה קלט או פלט.

2.3.4 : Digital Peripherals – פריפריאליים דיגיטאליים.

קיימות מספר מערכות דיגיטאליות במיקרו בקר. נזכיר את המערכות:

איור 9 מראה לנו את הרכיבים הפריפריאליים הדיגיטאליים.

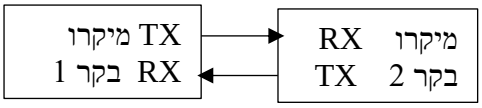


איור 9 : פריפריאליים דיגיטאליים

2.3.5 : UART0 ו UART1

2 מעגלי תקשורת טורית . UART – Universal Asynchronous Receiver Transmitter – משדר מקלט לא סינכרוני. לכל UART יש רגל קליטה RX ורגל שידור TX . ב 8051 המקורי היה רק UART אחד.

כאשר עובר נתון מצד אחד לשני הוא משודר ביט אחרי ביט עד לסיום כל הביטים של הנתון. באיור 9 נראה חיבור טורי.



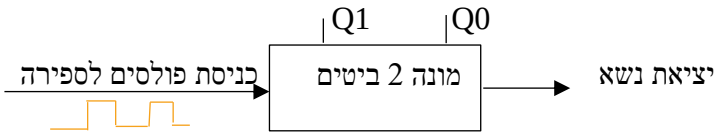
איור 10 : חיבור 2 מיקרו בקרים בתקשורת טורית

באיור רואים שהדק השידור - TX - של מיקרו בקר 1 מתחברת לרגל הקליטה RX של המיקרו בקר השני ולהפך, רגל הקליטה של מיקרו בקר 1 – RX - מתחברת לרגל השידור TX של מיקרו בקר 2 .

2.3.6 : TIMERS 0,1,2,3,4,5

טיימר הוא מעגל אלקטרוני המונה – סופר - את הפולסים בכניסה שלו. המונים במיקרו בקר שלנו סופרים כלפי מעלה. למיקרו בקר יש 6 טיימרים (ב 51 המקורי היו 2 טיימרים בלבד) מטיימר 0 ועד טיימר 5.

איור 11 מראה מונה עקרוני :



איור 11 : מונה ל 2 ביטים

באיור רואים מונה עקרוני ל 2 ביטים כאשר Q0 ו Q1 הן יציאות המונה. הספירה היא בינארית. נניח שבהתחלה מצב היציאות Q1Q0=00 . אחרי כניסת הפולס הראשון Q1Q0=01 אחרי הפולס השני Q1Q0=10 , אחרי הפולס השלישי Q1Q0=11 . בפולס הרביעי המונה מתאפס Q1Q0=00 וביציאת הנשא מקבלים 1 שאומר שהמונה עבר את התחום המקסימלי של הספירה שלו ומתחילה ספירה חדשה מ 00 . מונה כזה נקרא מונה ל 4 מצבים כי יש 2 יציאות והנוסחה היא : 2^n (n מבטא את כמות היציאות) . המונה הוא ל 4 מצבים והמספר המקסימאלי שהוא יכול להראות הוא כאשר כל הביטים שלו ב 1 כלומר 3.

במיקרו בקר C8051F380 יש 6 מונים כאשר כל מונה הוא של 16 ביטים. מספר המצבים שלו הוא 2^{16} שהם 65536 והמספר המקסימאלי שיכול להיות בו הוא 65535 . היציאה של הנשא במונה שבמיקרו היא יציאה המבקשת פסיקה (על פסיקות נסביר בהמשך בהרחבה). בצורה כזו המיקרו "יודע" שהמונה סיים ספירה .

המקור של הפולסים – המקום שממנו מגיעים הפולסים לספירה - יכול להיות מ 2 מקומות . או ממקור חיצוני הנותן פולסים ובמרה הזה אומרים שהמונה עובד כקאונטר- **COUNTER**. המקור השני פולסים המגיעים מתוך המיקרו בקר , ממעגל השעון שלו. במקרה הזה המרחק בין פולס לפולס הוא קבוע ולא משתנה. המרחק הוא למעשה הזמן בין פולס לפולס. במקרה כזה אומרים שהמונה עובד כ**טיימר** – זמנן –**TIMER** - ויודע למדוד זמן. לדוגמה : אם הפולסים בכניסה מגיעים ממקור של 1MHz , כלומר המרחק בין פולס לפולס (זמן המחזור) הוא : $1/f=1/(1*10^6)$ כלומר $1*10^{-6}$ שהם 1 מיקרו שנייה. אם המונה מתחיל מ 0 נקבל פסיקה (יציאת נשא) אחרי 65536 פולסים, כלומר הזמן שעבר הוא 65536 מיקרו שניות.

2.3.7 : Programmable Counter Array/ Watca Dog Timer – PCA/WDT - מערך מונה בר תכנות/ טיימר כלב שמירה

בנוסף ל 6 הטיימרים שבסעיף הקודם יש טיימר נוסף , גם הוא של 16 ביט . לטיימר זה יש 5 מודולים הנקראים capture/compare (לכידה/השוואה לכידה מלשון ללכוד, לתפוס). כל מודול כזה ניתן לחיבור עם אחד מהדקי היציאה . ניתן לתכנת את המודול כך שכאשר במונה יהיה את המספר הרצוי לנו הוא יודיע זאת בהדק המתאים.

מודול מספר 4 יכול לשמש כ WDT (טיימר כלב עקיבה). ה WDT משמש כדי ליצור איפוס RESET אם הזמן בין כתיבות לאוגר העדכונים של WDT חורג מהגבול שקבענו. נסביר : במידה ויש חשש שהמעבד "נתקע" בגלל סיבה מסוימת כמו לולאה אינסופית, מצב המתנה לקלט חיצוני שלא מגיע, גלישת זיכרון ועוד משתמשים בטיימר WDT שניתן לקבוע לו ערך גבולי רצוי. הטיימר מתחיל לרוץ לאחר אתחול הבקר. באם עבר משך זמן מסוים והטיימר לא אופס, הוא יגיע לערך העובר את הערך הגבולי שקבענו ואז יבצע RESET למערכת.

ניתן לאפשר עבודה עם WDT או למנוע עבודה איתו. אם מאפשרים את פעולתו יש לדאוג לאפס אותו בקוד בכל כמה זמן.

2.3.8 : SMBus0 and SMBus1

SMBUS – System Management Bus – פס ניהול מערכת הוא קבוצה בתוך תקשורת טורית הנקראת **I2C** או **IIC** או **I^2C** שהוא קיצור של **Inter-Integrated Circuit** - **מעגל בין-משולב** . זוהי תקשורת טורית סינכרונית למרחקים קטנים ויש לה 2 קווים בלבד : Serial Data – SDA (נתון טורי) ו Serial Clock – SCL – שעון טורי.

במיקרו בקר C051f380 יש 2 מערכות תקשורת כאלו הנקראות SMBUS0 ו SMBUS1 . גם על תקשורת זו נדבר בהרחבה בהמשך הספר.

2.3.9 : SPI - Serial Peripheral Interface – ממשק היקפי טורי

SPI הוא סוג נוסף של תקשורת טורית סינכרונית. גם על תקשורת זו נדבר בהרחבה בהמשך הספר.

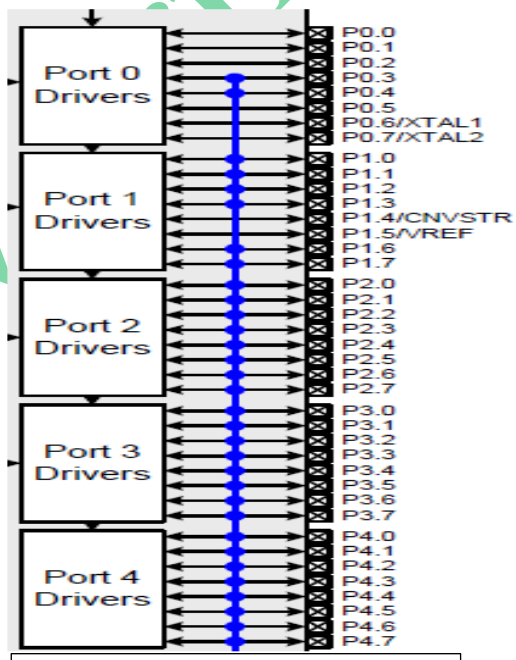
2.3.10 : Crossbar Control ו Priority Crossbar Decoder : בקרת קרוסבר ומפענח עדיפות קרוסבר.

Crossbar הוא קורה – מוט - המחבר בין שני גופים בדומה לקורה של שער בכדורגל. כאן הקרוסבר מחבר בין המעגלים האלקטרוניים שבתוך הרכיב ובין 40 הרגליים החיצוניות של המיקרו בקר. הוא עושה זאת בעזרת 3 רגיסטרים של קרוסבר הנקראים xbr0 , xbr1 , xbr2 ומפענח עדיפות קרוסבר. לדוגמה : המשתמש יכול לקבוע בתוכנה , האם ההדקים RX ו TX של התקשורת הטורית UART0 יתחברו לרגליים החיצוניות של port0 או לא , האם הדקים של כל רכיב פריפריאלי פנימי יתחבר אל הדק חיצוני של המיקרו בקר.

2.3.11 : 5 פורטים לקלט/פלט מפורט 0 ועד פורט 4

יש 5 פורטים (ב 51 המקורי היו רק 4 פורטים) מ PORT0 ועד PORT4 . לכל פורט יש 8 הדקים מ 0 עד 7 .

באיור 12 רואים את 5 הפורטים וההדקים שלהם.



איור 12 : 5 הפורטים וההדקים שלהם.

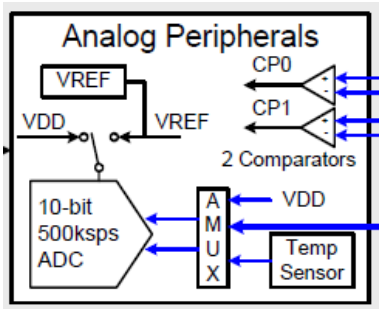
לדוגמה : PORT0.0 אומר שמדובר ברגל 0 של פורט 0 . PORT1.5 אומר שמדובר בפורט1 הדק 5 (ההדק השישי שלו). לכל הדק ניתן לקבוע תפקיד של קלט או פלט בעזרת מלבן PORT IO Configuration שהסברנו בסעיף 2.1.3 . ההדקים בצבע הכחול הם הדקים המתחברים אל הרכיבים הפריפריאליים האנלוגיים. ההדקים האנלוגיים לא עוברים דרך הפורטים שיכולים להעביר רק נתון דיגיטאלי (0 או 1).

2.3.13 : ממשק לחיבור זיכרון חיצוני

זהו הממשק עם זיכרון הנתונים הנקרא XRAM . במיקרו בקר ממשפחת ה 51 המקורי כל זיכרון הנתונים החיצוני XRAM היה מחוץ לרכיב. כאן יש 4 קילו בתוך הרכיב ועוד אפשרות לחבר עוד 60 קילו חיצוני. הפניה לזיכרון זה היא עם פקודת MOVX . בממשק שלושה מלבנים : DATA – נתון , ADDRESS כתובת ו CONTROL – בקרה (קריאה או כתיבה).

2.3.14 : רכיבים פריפריאליים אנאלוגיים

ברכיב יש מעגלים אנלוגיים הנראים באיור 13 .



איור 13 : הרכיבים הפריפריאליים האנאלוגיים

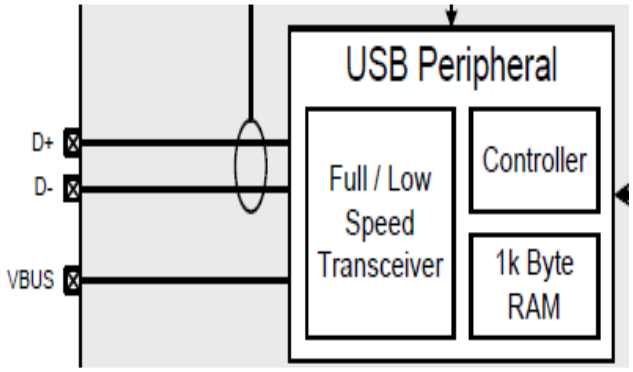
הרכיבים הם:

- **ADC** - ממיר אנאלוגי לדיגיטאלי של 10 ביט עם קצב דגימה של 500ksps - Kilo Samples Per Second – 500 אלף דגימות בשנייה. לממיר ניתן להכניס 20 כניסות אנלוגיות שונות דרך AMUX – מרבב – אנלוגי שקובע איזו כניסה מתוך ה 20 תיכנס להמרה. האם להכניס את המתח של חיישן הטמפרטורה להמרה
- **Temp Sensor** - חיישן טמפרטורה שגם את המתח שלו ניתן להכניס להמרה דרך ה AMUX .
- 2 Comparators - שני מעגלים משווים. המעגלים הם משווי מתח ואת הכניסות אליהם וגם את היציאות שלהם CP0 ו CP1 ניתן לחבר לרגליי היציאה של הרכיב. כך נוכל לדעת איזה מתח גדול יותר .

2.3.15 : תקשורת USB

USB הוא קיצור של **Universal Serial Bus** - **אפיק טורי אוניברסלי** שהוא תקן לחיבור בין מחשבים לבין התקני ציוד היקפי. זהו התקן הפופולרי ביותר לחיבור חומרה למחשבים אישיים . הממשק תומך בהעברת נתונים במהירות של עד 12 מגה ביטים לשנייה בגרסת USB 1.1 , עד 480 מגה ביטים לשנייה בגרסת USB 2.0 , ועד 4.8 ג'יגה ביטים לשנייה בגרסת USB 3.0 במאי 2020 יש גרסת USB 4 למהירות של עד 40 ג'יגה ביטים בשנייה. ה USB כאן הוא תואם USB 2.0 .

איור 14 מתאר את מערכת ה USB .



איור 14 : מעגל ה USB

הדק VBUS הוא הדק כניסת חישה. כאשר מקבלים בהדק זה 5 וולט יודעים שהתחברנו לרשת USB.

ההדקים D+ ו -D הם הדקי הנתונים המשמשים כקלט או פלט בזמן התקשורת.

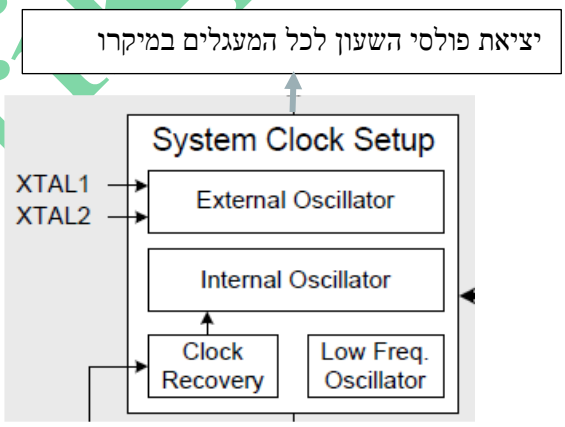
ה CONTROLLER הוא בקר השולט על התקשורת.

1k Byte RAM – הוא הזיכרון של ה USB .

FULL/LOW Speed Transceiver - משדר/מקלט למהירות נמוכה/גבוהה . מהירות גבוהה היא עד 12 מגה ביטים בשנייה ומהירות נמוכה היא עד 1.5 מגה ביטים בשנייה.

2.3.16 : מערכת קביעת השעון

לכל מערכת מיקרו יש מערכת שעון – Clock - היוצרת פולסים ומפעילה את כל האלקטרוניקה במיקרו. תדר השעון קובע את מהירות העבודה של המיקרו. איור 15 מראה את מבנה מערכת השעון.



איור 15 : מערכת השעון

המשתמש יכול לקבוע האם תדר השעון שיוצא לכל המעגלים יגיע ממתנד פנימי **Internal Oscillator** שהוא בתדר מדויק של 48 מגה הרץ או ממתנד חיצוני **External Oscillator** . אם החלטנו לעבוד עם מתנד חיצוני יש לחבר גביש חיצוני בהדקים XTAL1 ו XTAL2 . תדר הגביש שנחבר ייקבע את מהירות העבודה.

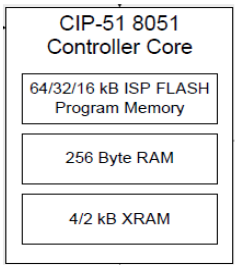
מדוע להשתמש בגביש חיצוני ? אם רוצים לעבוד עם מערכות קודמות שעבדו עם 51 מקורי ורק לשנות את המיקרו בקר ללא שינויי תוכנה.

2.3.17 : מייצבי מתח Voltage Regulators

המתח בהדק V_{DD} (מתח הספק יחסית להדק האדמה GND) המקסימאלי המותר הוא 4.2 וולט (יש לשים לב שלא 5 וולט !!). מייצבי המתח מייצבים מתח זה ל 3.3 וולט ו 1.8 וולט.

2.3.18 : ליבת המיקרו בקר

באיור 16 מראים את הזיכרונות ליבת המיקרו בקר CIP-51 .



איור 16 : זיכרונות בליבת המיקרו בקר

באיור רואים את סוגי הזיכרון השונים בליבה. את זיכרון התוכנית בגודל 64 קילו, את RAM הנתונים הפנימי בגודל 256 בתים ואת זיכרון ה XRAM של ה 4 קילו בתים. מעגלים נוספים שקיימים בליבה אפשר לראות באיור 6.

2.4 : הקרוסבר - CROSSBAR

לרכיבים C8051F380/2/4/6 יש 40 הדקים (פינים) ואילו לרכיבים 8051F381/3/5/7 יש 25 פינים. ההדקים יכולים להיות הדקי קלט או הדקי פלט I/O או הדקים אנאלוגיים לפי קביעה של המשתמש .

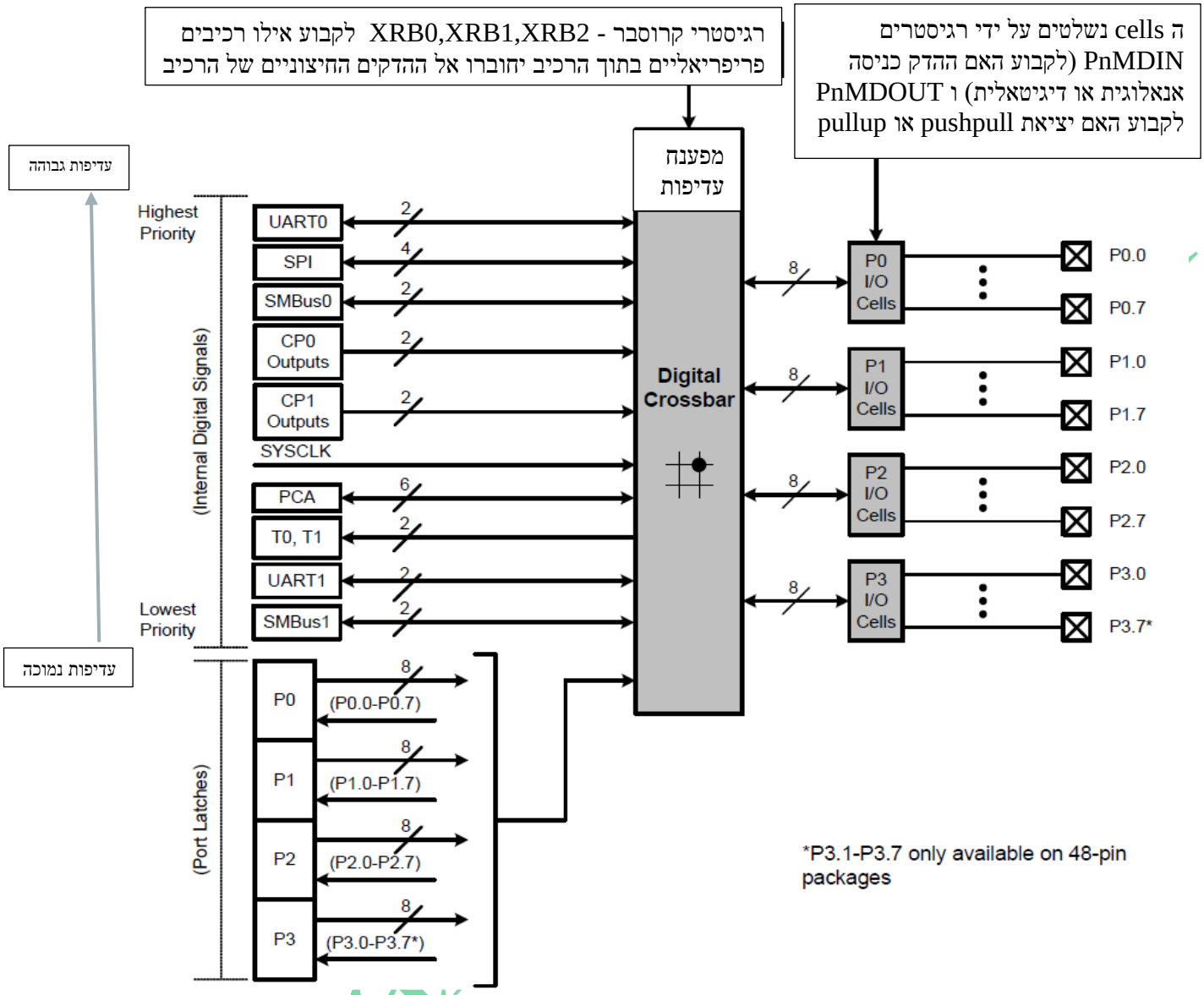
איור 17 מתאר חיבור הרכיבים הפריפריאליים בתוך הג'וק אל ההדקים החיצוניים שלו בעזרת הקרוסבר.

כל הדקי הפורטים מפורט 0 ועד פורט 3 יכולים להתחבר אל מעגלי האלקטרוניקה השונים שברכיב לפי רצון המתכנת. גמישות כזו בהקצאת ההדקים מאופשרת בעזרת Priority Crossbar Decoder – מפענח עדיפות הקרוסבר (Crossbar הוא קורה – מוט - המחבר בין שני גופים בדומה לקורה של שער בכדורגל). אצלנו הקרוסבר מחבר בין המעגלים האלקטרוניים שבתוך הרכיב (המתוארים בצד שמאל של האיור הבא) ובין 40 הרגליים החיצוניות של המיקרו בקר (שמצד ימין של האיור) . ניתן לקרוא את מצב הדקי הפורטים ללא קשר האם ההדק קלט או פלט בעזרת Port Latches - נועלי הפורט שרואים בצד שמאל למטה באיור).

מהאיור רואים שהקרוסבר מקבל מצד שמאל שלו , בחלק העליון שלו, כניסות ממעגלי האלקטרוניקה שברכיב ומארבעת הפורטים .

העדיפות הגבוהה היא ל , UART0 אחריו ה SPI וכך הלאה עד לעדיפות הנמוכה ביותר שהיא SMBUS1.

הקרוסבר מקצה לכל מעגל דיגיטאלי או אנאלוגי בתוך הרכיב , הדק/רגל חיצונית של הג'וק שהיא I/O (קלט/פלט). הרגיסטרים XBR0, XBR1,XBR2 הנמצאים באזור ה SFR (הרגיסטרים לתפקידים מיוחדים) קובעים איזה הדק של מעגל פנימי יתחבר להדק/רגל חיצונית ברכיב. ברגיסטר XBR1 יש ביט הנקרא XBARE (ביט 6) הקובע האם הקרוסבר פועל ומחבר הדקים פנימיים לרגליים חיצוניות. מימין לקרוסבר, ממש לפני היציאה להדקים החיצוניים, יש את תאי 4 הפורטים – Cells - (מ P0 ועד P3). הם נשלטים על ידי רגיסטרים PnMDOUT ו PnMDIN שקובעים האם ההדק יהיה אנאלוגי או דיגיטאלי והאם ההדק נבחר כיציאה אז האם הוא יהיה push – pull או open drain . יש 4 תאים כאלה, אחד עבור כל פורט).



איור 17 : חיבור הרכיבים הפריפריאליים בתוך הג'וק אל ההדקים החיצוניים שלו בעזרת הקרוסבר

2.4.1 : מפענח עדיפות הקרוסבר

מפענח עדיפות הקרוסבר מאפשר למשתמש להקצות לכל רכיב פריפריאלי בתוך הג'וק רגליים חיצוניות בג'וק לפי היישום שהוא משתמש. שאר הפינים יהיו זמינים לפונקציות I/O למטרה כללית. הוא מתחיל ב UART0 . האיור הבא מראה את כל זמינות ההדקים של הרכיב לכל מעגל פריפריאלי/מעגל אלקטרוני ברכיב .

Port	P0								P1								P2								P3							
Pin Number	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
SF Signals (32-pin Package)		XTAL1	XTAL2				CNVSTR	VREF																		P3.1-P3.7 Unavailable on 32-pin packages						
SF Signals (48-pin Package)						XTAL1	XTAL2		ALE		CNVSTR	VREF	/RD	/WR																		
TX0					■																											
RX0						■																										
SCK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■		
MISO		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
MOSI			■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
NSS*				■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
SDA	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
SCL		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CP0	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CP0A	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CP1	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CP1A	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
SYSCLK	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CEX0	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CEX1		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CEX2			■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CEX3				■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
CEX4					■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
ECI	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
T0	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
T1	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
TX1	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
RX1		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
SDA1	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	
SCL1		■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■
Pin Skip Settings	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	P0SKIP								P1SKIP								P2SKIP								P3SKIP							

The crossbar peripherals are assigned in priority order from top to bottom, according to this diagram.

■ These boxes represent Port pins which can potentially be assigned to a peripheral.

□ Special Function Signals are not assigned by the crossbar. When these signals are enabled, the Crossbar should be manually configured to skip the corresponding port pins.

□ Pins can be "skipped" by setting the corresponding bit in PnSKIP to 1.

* NSS is only pinned out when the SPI is in 4-wire mode.

איור 18 : זמינות המעגלים הפריפריאליים ברכיב על הדקי פורט הקלט פלט

השורה העליונה בצבע צהוב מראה את 4 הפורטים ואת הדקיהם מ 0 עד 7 .

2 השורות בצבע הירוק מראות את ההדקים עבור זיווד של 32 פינים ברכיבים C8051F381/3/5/7 (שורה ירוקה עליונה) ומתחתיה עבור זיווד של 48 הדקים ברכיבים C8051F380/2/4/6 . עבור הרכיבים בשורה השנייה יש הדקים שאותם לא ניתן לשנות בעזרת הקרוסבר לדוגמה XTAL1 ו XTAL2 (עבור חיבור גביש חיצוני לקביעת תדר העבודה – בדרך כלל לא נשתמש במצב זה.) שאם אכן נחליט להשתמש בהדקים אלו כדי לחבר גביש חיצוני אז הדקים אלו יהיו תמיד P0.5 ו P0.7 בהתאמה. דבר דומה לגבי הדק ALE (הדק לנעילת כתובות בחיבור זיכרון חיצוני) שתמיד יהיה ב P1.2 וכך הלאה לגבי שאר ההדקים הרשומים בהמשך בשורה הירוקה השנייה.

המלבנים בצבע הסגול מראים איזה הדק של פורט ניתן לחבר להדק של רכיב פריפריאלי ברכיב. לדוגמה : הדק TXD של התקשורת הטורית ניתן לחבר רק להדק P0.4 . לעומת זאת הדק ה SCK (שורה לבנה שלישית) יכול להיות מוקצה לכל ההדקים חוץ מ 4 ההדקים הגבוהים של פורט 3 . הדק SCL1 (שורה לבנה אחרונה) יכול להיות מוקצה לכל ההדקים חוץ מהדק P0.0 .

השורה התחתונה בצבע התכלת מתארת את הביטים של רגיסטרי הדילוג PnSKIP . יש 4 רגיסטרים כאלו מ 0 עד 3 .

סדר העדיפות הוא מהשורה הלבנה הראשונה (עדיפות הכי גבוהה) ועד לשורה הלבנה האחרונה בעדיפות הכי נמוכה.

כאשר רכיב פריפריאלי/מעגל אלקטרוני פנימי נבחר אז הדק הפורט הכי פחות משמעותי שלא הייתה לו הקצאה, מוקצה למעגל הזה (לא כולל UART0 שהוא תמיד הדקים 4 ו 5). אם מוקצה הדק של פורט, הקרוסבר מדלג על הדק זה כאשר הוא מקצה את המעגל הבא. בנוסף, הקרוסבר ידלג על הדקי פורט שהביטים המתאימים שלהם ברגיסטר PnSKIP שמנו '1'. רגיסטרי ה PnSKIP מאפשרים לתוכנה לדלג על הדקי פורט שישמשו ככניסות להדקים אנלוגיים, לפונקציות ייעודיות או GPIO (General Purpose Input Output – קלט פלט לשימוש כללי).

אם הדק של פורט נדרש על ידי רכיב פריפריאלי ללא שימוש בקרוסבר יש לשים '1' בביט המתאים לו ב PnSKIP . זה מתייחס לאות VREF , להדקי המתנד החיצוני (XTAL1,XTAL2) , לאות "התחל המרה" CoNVert StaRT – CNVSTR אותות EMIF (חיבור זיכרונות חיצוני), וכל כניסת ADC או משווה – Comperator .

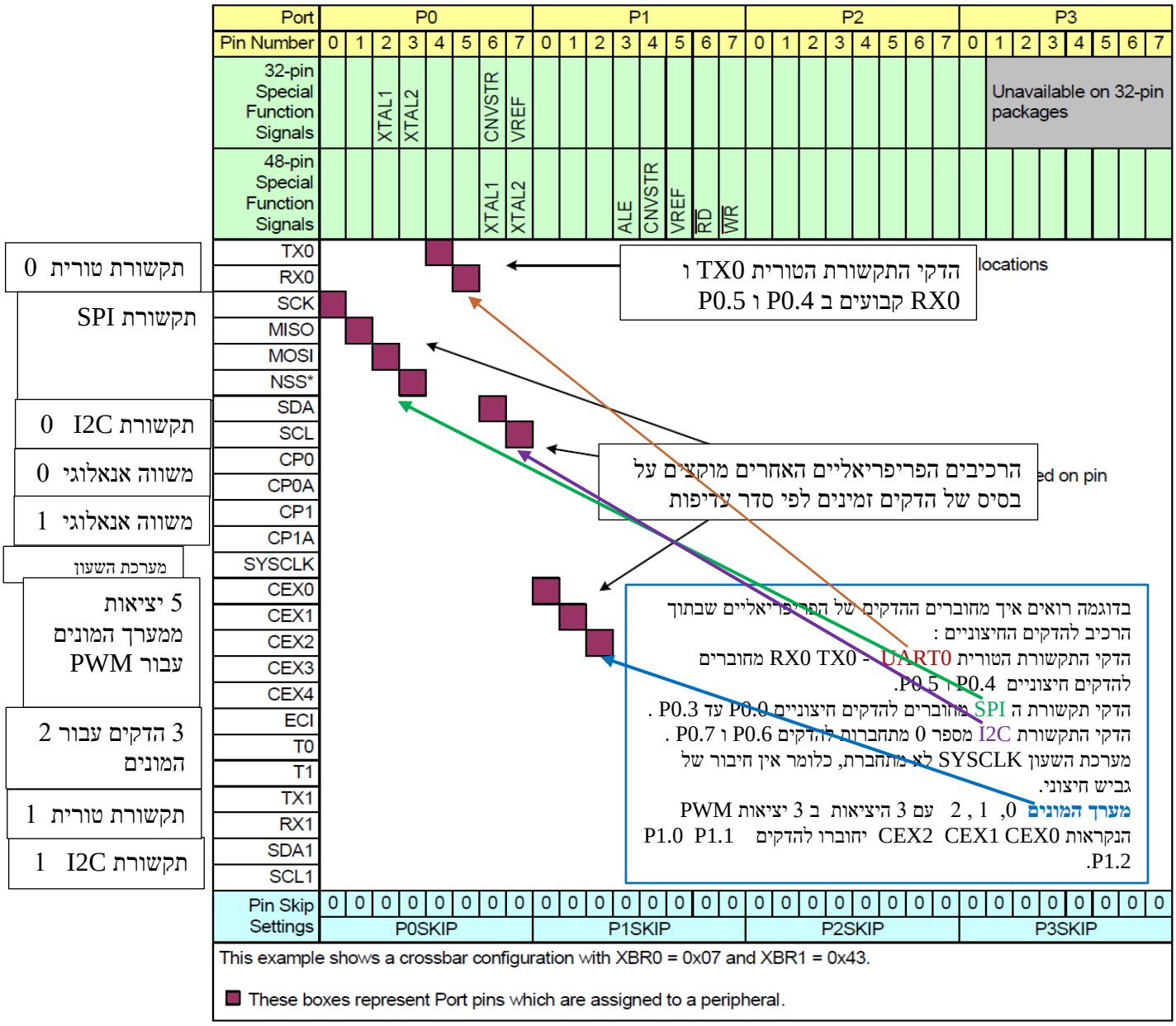
רגיסטרי PnSKIP יוכלו לשמש גם כדילוג על הדקי GPIO. הקרוסבר מדלג על הדקים נבחרים כאילו הם כבר הוקצו ועובר להדק ההקצאה הבא. האיור הבא מראה דוגמה לתצורה של קרוסבר ללא דילוג על הדקים.

באיור בעמוד הבא רואים את הריבועים בצבע סגול שאומרים אילו יחידות פריפריאליות פנימיות (מעגלים בתוך הרכיב) נרצה לחבר להדקים החיצוניים . החיבורים יתבצעו בעזרת רגיסטרי הקרוסבר XBR0 XRB1 XRB2 .

בעזרת הריבועים הסגולים באיור רואים שרוצים לחבר להדקי הג'וק את הרכיבים הפריפריאליים הבאים:

בעזרת XBR0

א. הדקי התקשורת הטורית מספר 0 (TX0 RX0 של UART0) והם יחוברו אל ההדקים החיצוניים P0.4 ו P0.5.
ב. ההדקים של מעגל תקשורת ה SPI שבתוך הרכיב והם יחוברו אל ההדקים החיצוניים מ P0.0 ועד P0.3 .
ג. הדקי תקשורת I2C (SMBUS) SDA SCL יחוברו אל ההדקים החיצוניים P0.6 ו P0.7 .
ד. מערך המונים 0, 1 , 2 3 עם יציאות PWM הנקראות CEX0 CEX1 CEX2 יחוברו אל ההדקים P1.2 P1.1 P1.0 בעזרת XBR1.



איור 19 תצורה של קרוסבר ללא דילוג על הדקי פורט

2.4.2 : מבנה XBR0 XBR1 XBR2

באיור הבא מתואר רגיסטר XBR0 . תפקידו לאפשר או לחסום חיבור בין הדקי הרכיבים הפריפריאליים הפנימיים ובין ההדקים החיצוניים של הג'וק. נסביר מדוע XBR0=7 ו XBR1=0x43 . הרגיסטר XBR2 לא אקטואלי בדוגמה זו.

השורה הראשונה היא שם הביט. השורה השנייה מראה האם הביט לקריאה או/ו כתיבה (במקרה כאן Read/Write - R/W - ניתן גם לכתוב לביט וגם לקרוא ממנו). השורה השלישית מראה את מצב הרגיסטר אחרי פעולת איפוס – RESET . רואים שכל הביטים שלו ב 0 .

הביטים הם ביטים של אפשור. כאשר נשים 0 בביט חוסמים חיבור וכאשר נשים 1 מאפשרים חיבור. **שאר הפורטים שאינם מוקצים לרכיבים פריפריאליים פנימיים משמשים כקלט פלט למטרה כללית GPIO – general Purpose Input Output.**

Bit	7	6	5	4	3	2	1	0
Name	CP1AE	CP1E	CP0AE	CP0E	SYSCKE	SMB0E	SPI0E	URT0E
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xE1; SFR Page = All Pages

Bit	Name	Function
7	CP1AE	**Comparator1 Asynchronous Output Enable.** 0: Asynchronous CP1A unavailable at Port pin. 1: Asynchronous CP1A routed to Port pin. אפשור משווה 0 להתחבר להדקים החיצוניים
6	CP1E	**Comparator1 Output Enable.** 0: CP1 unavailable at Port pin. 1: CP1 routed to Port pin. אפשור יציאת משווה 0 להתחבר להדקים החיצוניים
5	CP0AE	**Comparator0 Asynchronous Output Enable.** 0: Asynchronous CP0A unavailable at Port pin. 1: Asynchronous CP0A routed to Port pin. אפשור משווה 0 להתחבר להדקים החיצוניים
4	CP0E	**Comparator0 Output Enable.** 0: CP0 unavailable at Port pin. 1: CP0 routed to Port pin. אפשור יציאת משווה 0 להתחבר להדקים החיצוניים
3	SYSCKE	**$\overline{\text{SYSCLK}}$ Output Enable.** 0: $\overline{\text{SYSCLK}}$ unavailable at Port pin. 1: $\overline{\text{SYSCLK}}$ output routed to Port pin. אפשור SYSCLK להתחבר להדקים החיצוניים
2	SMB0E	**SMBus I/O Enable.** 0: SMBus I/O unavailable at Port pins. 1: SMBus I/O routed to Port pins. אפשור SMBUS (I2C) להתחבר להדקים החיצוניים
1	SPI0E	**SPI I/O Enable.** 0: SPI I/O unavailable at Port pins. 1: SPI I/O routed to Port pins. Note that the SPI can be assigned either 3 or 4 GPIO pins. אפשור SPI . אם מאפשרים אז ניתן עם 3 או 4 הדקי GPIO
0	URT0E	**UART I/O Output Enable.** 0: UART I/O unavailable at Port pin. 1: UART TX0, RX0 routed to Port pins P0.4 and P0.5. אפשור UART0 tx0 rx0 להתחבר להדקים החיצוניים אם כן החיבור יהיה ל P0.4 P0.5

איור 20 : מבנה רגיסטר XBR0

בעזרת איורים 19 ו 20 ובעזרת האיור הבא המתאר את XBR0 יש לשים ברגיסטר :

Bit	7	6	5	4	3	2	1	0
Name	CP1AE	CP1E	CP0AE	CP0E	SYSCKE	SMB0E	SPI0E	URT0E
	0	0	0	0	0	1	1	1

איור 21 : קביעת תצורה ב XRB0

כלומר XBR0=000001111B=7

נבדוק מהו הערך שנשים ב XRB1 . לשם כך נכיר אותו בעזרת איור 22 .

Bit	7	6	5	4	3	2	1	0
Name	WEAKPUD	XBARE	T1E	T0E	ECIE	PCA0ME[2:0]		
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xE2; SFR Page = All Pages

Bit	Name	Function
7	WEAKPUD	**Port I/O Weak Pullup Disable.** 0: Weak Pullups enabled (except for Ports whose I/O are configured for analog mode). 1: Weak Pullups disabled. אפשור נגדי weak pullups -משיכה למעלה חלשה (נגדים גדולים) חוץ מאלו שתצורתם נקבעה כאנלוגית
6	XBARE	**Crossbar Enable.** 0: Crossbar disabled. 1: Crossbar enabled. אפשור הקרוסבר
5	T1E	**T1 Enable.** 0: T1 unavailable at Port pin. 1: T1 routed to Port pin. אפשור כניסת ספירה חיצונית לטיימר 1
4	T0E	**T0 Enable.** 0: T0 unavailable at Port pin. 1: T0 routed to Port pin. אפשור כניסת ספירה חיצונית לטיימר 0
3	ECIE	**PCA0 External Counter Input Enable.** 0: ECI unavailable at Port pin. 1: ECI routed to Port pin. אפשור כניסת מונה חיצוני PCA0 0 – ECI לא מאופשר חיבור הדק לפורט 1 – הדק ECI מתחבר להדק פורט
2:0	PCA0ME[2:0]	**PCA Module I/O Enable Bits.** 000: All PCA I/O unavailable at Port pins. 001: CEX0 routed to Port pin. 010: CEX0, CEX1 routed to Port pins. 011: CEX0, CEX1, CEX2 routed to Port pins. 100: CEX0, CEX1, CEX2, CEX3 routed to Port pins. 101: CEX0, CEX1, CEX2, CEX3 ,CE4 routed to Port 11x: Reserved. אפשור כניסת ספירה חיצונית לטיימר 1

איור 22 מבנה רגיסטר XBR1

פעולה ראשונה שיש לבצע היא לאפשר את הקרוסבר. XBARE=1 . הפעולה הבאה היא לאפשר את היציאות של המונים שיוצרים PWM . בשורה האחרונה רשום שאם רוצים לאפשר את 3 ההדקים האלו : PCA0ME = 011 .

מכאן ש XBR1=01000011B = 0x43 .

XBR2 מחבר בין הפריפריאלים הפנימיים של התקשורת SMBUS1 (עם ההדקים הפנימיים SCL1 SDA1) ותקשורת UART1 (עם ההדקים TX1 RX1) ובין ההדקים החיצוניים של הרכיב.

הרגיסטר נראה באיור הבא :

Bit	7	6	5	4	3	2	1	0
Name							SMB1E	URT1E
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xE3; SFR Page = All Pages

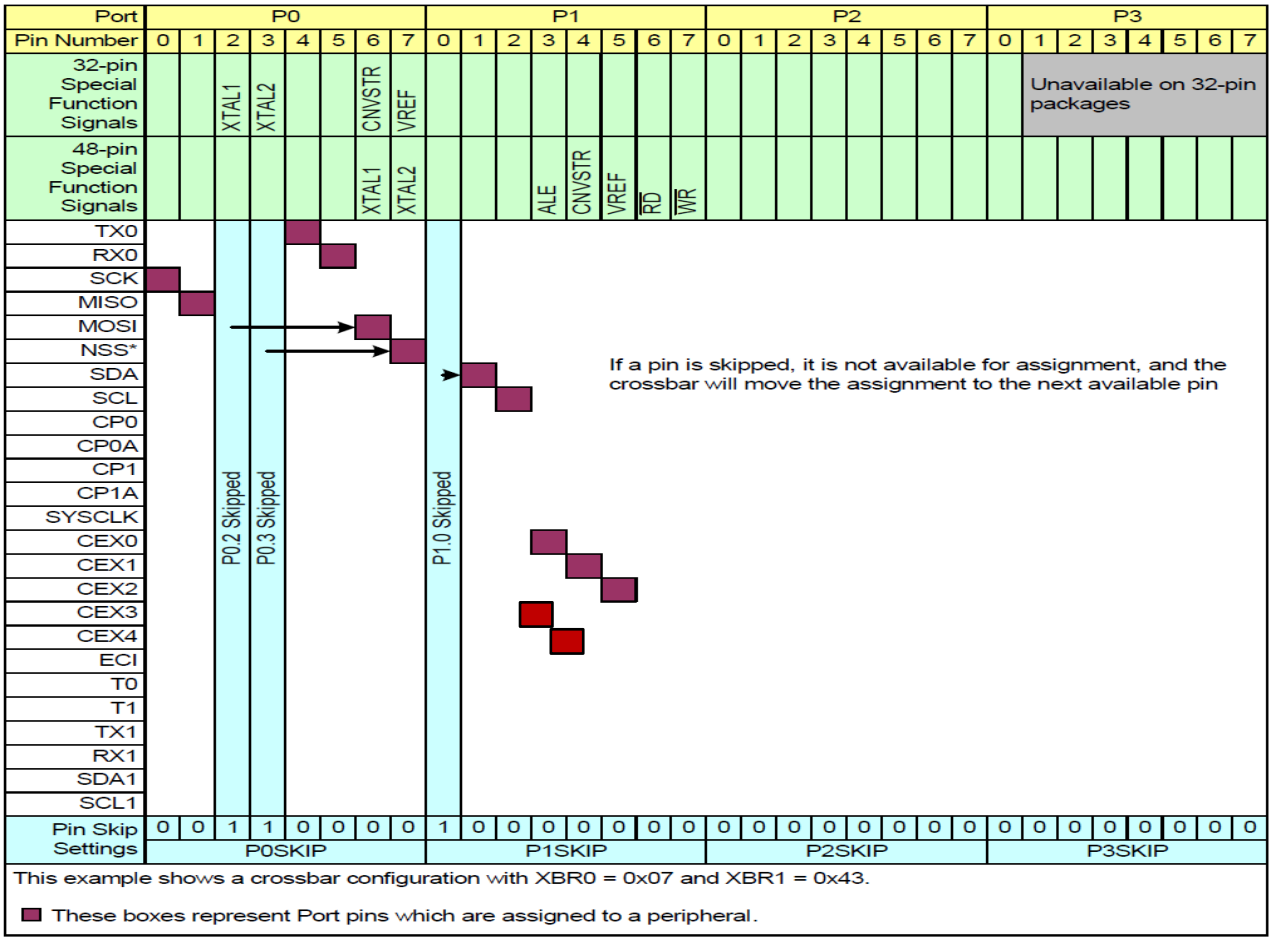
Bit	Name	Function
7:2	Reserved	Must write 000000b
1	SMB1E	**SMBus1 I/O Enable.** 0: SMBus1 I/O unavailable at Port pins. 1: SMBus1 I/O routed to Port pins. אפשור הדקים של SMBUS1
0	URT1E	**UART1 I/OEnable.** 0: UART1 I/O unavailable at Port pins. 1: UART1 TX1, RX1 routed to Port pins. אפשור הדקים של UART1

איור 23 : מבנה רגיסטר XBR2

ברגיסטר הזה יש את 2 הרכיבים הפריפריאליים האחרונים : SMBUS1 ו UART1 . שאר הביטים ללא תפקיד.

דוגמה נוספת

האיור הבא מתאר חיבור של 3 התקשורות הטוריות :UART0, SPI (SMBUS0) I2C וכל 5 ה PWM עם 3 דילוגים.



איור 24 : חיבור 3 תקשורות טוריות, ו PWM 5 .

מהאיור רואים שהתקשורת הטורית UART0 עם ההדקים Tx0 RX0 מתחברים אל ההדקים החיצוניים P0.4 ו P0.5. תקשורת ה SPI עם 4 ההדקים שלה SCLK ו MISO מתחילים ב P0.0 ו P0.1 אחר כך יש דילוג של שניים וההדקים MOSI ו NSS מתחברים ל P0.6 ו P0.7 .

בין כל שני הדקים של SPI מדלגים על P0.2 ו P0.3 . הדילוג מתבצע בעזרת רגיסטר P0SKIP שנראה ב 2 השורות התחתונות של האיור שבו שמים 1 בשני הביטים 2 ו 3 שלו .

תקשורת ה I2C עם ההדקים SDA SCL מתחברים אל ההדקים P1.1 ו P1.2 כאשר יש דילוג על P1.0 .

מה נשים ב XBR0 ?

Bit	7	6	5	4	3	2	1	0
Name	CP1AE	CP1E	CP0AE	CP0E	SYSCKE	SMB0E	SPI0E	URT0E
	0	0	0	0	0	1	1	1

XRBR0=7

מה נשים ב XBR1 ?

Bit	7	6	5	4	3	2	1	0
Name	WEAKPUD	XBARE	T1E	T0E	ECIE	PCA0ME[2:0]		
	0	1	0	0	0	1	0	1

XRBR1=0x45

XBR2 – ללא תפקיד.

התוכנית שנרשום היא :

P0SKIP=0x0c;	בסי	MOV P0SKIP,#0CH	באסמבלי
P1SKIP=0x1;	בסי	MOV P1SKIP,#1	באסמבלי
XBR0=0x7;	בסי	MOV XBR0,#7	באסמבלי
XBR1=0x45;	בסי	MOV XBR1,#45H	באסמבלי

הדקים של פורט שלא מוקצה להם הדק חיצוני על ידי הקרוסבר ולא מתחברים לרכיבים אנלוגיים בתוך הרכיב יכולים לשמש למטרה כללית - GPIO . **בכתיבה** לפורטים 0 עד 3 ניגשים בעזרת רגיסטרים ב SFR גם בעזרת פקודות הפונות לביית – מיעון ביית - וגם במיעון ביט. פורט 4 (קיים רק ב C8051F380/2/4/6) נמצא ב SFR וניתן לגשת אליו במיעון ביית בלבד.כאשר כותבים לפורט כלשהו הערך הנכתב ל SFR ננעל כדי שהערך יישמר בכל הדק חיצוני.

בזמן קריאה, המצב של ההדקים החיצוניים מתקבל בפורט שב SFR ללא קשר לתכנות הקרוסבר (אפילו שההדק הוקצה לרכיב פריפריאלי פנימי) כך שהוא מתאר מצב אמיתי בהדקים. כל זה נכון אלא אם כן שינינו את מצב הפורט עם פקודות ANL,ORL,XRL,JBC.CPL,INC,DEC,DJNZ ,MOV,CLR,SETB כאשר היעד הוא ביט.

הערה : אם תכנות הקרוסבר נראה מורכב - אפשר להירגע. יש דרך נוספת לתכנות הקרוסבר ללא צורך לדעת איך בנוי כל רגיסטר קרוסבר ומה לרשום אליו. חברת סיליקון לאב מצרפת תוכנה שנקראת Configuration Wizard או בקיצור WIZARD ובעברית **אשף התקנה. זוהי תוכנה גרפית שבה מסמנים ב X אילו רכיבים פריפריאליים רוצים לחבר להדקי הרכיב ומקבלים מה לרשום לכל רגיסטר. נרחיב על התוכנה בהמשך.**

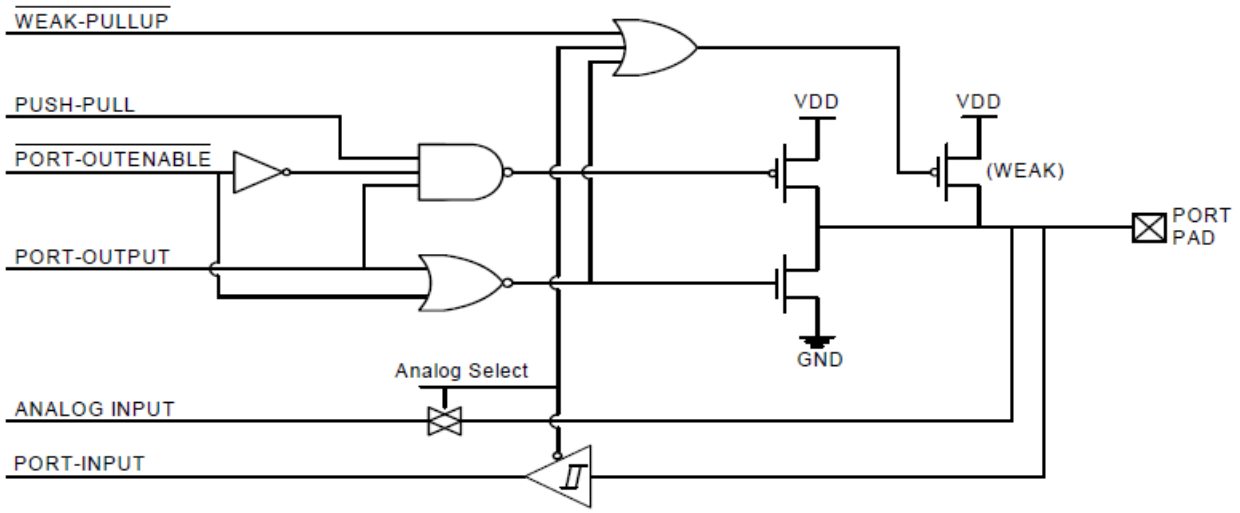
פרק 3 : הכרת מבנה הדק IO

הזכרנו שלרכיבים C8051F380/2/4/6/8 יש 48 הדקים של כניסות או יציאות (המתכנת קובע את מצב ההדק לפי היישום) ולרכיבים C8051F381/3/5/7 יש 32 הדקים בלבד.לרכיב 5 פורטים מפורט 0 עד 4 . ניתן לכתוב או לקרוא ביית מכל אחד מהפורטים.

לפורטים 0 1 2 ו 3 ניתן לפנות במיעון סיבית, כלומר לביט רצוי בפורט.

3.1 : המבנה של הדק IO

האיור הבא מתאר כיצד נראה המבנה של כל הדק של הרכיב (הכוונה במילה הדק היא רגל של הג'וק).



איור 25 : מבנה של הדק של הרכיב

ההדק נמצא בצד ימין ונקרא PORT PAD . ההדק יכול להיות קלט או פלט. בצד שמאל מגיעים אותות בקרה מביטים של רגיסטרים ב SFR שקובעים כיצד יתפקד ההדק. בינהם יש מעגלי אלקטרוניקה המקבלים את אותות הבקרה ופועלים בהתאם לאותות שהתקבלו. ההדק יכול לשמש כהדק קלט או כהדק פלט. הדק קלט יכול להיות קלט דיגיטאלי או קלט אנאלוגי. גם להדק הפלט הדיגיטאלי ישנן שלוש אפשרויות עבודה: א. PUSH PULL ב. OPEN DRAIN ג. OPEN DRAIN עם נגד משיכה למעלה חלשה (WEAK PULL UP) שהתנגדותו כ 100 קילו אוהם. הסבר מפורט על פעולתו בהמשך.

מספר רגיסטרים הנמצאים ב SFR שולטים על ההדק ואלו הם (התו n שמופיע בכל רגיסטר הוא מספר הפורט : 0 עד 3):

- רגיסטר PnMDIN – Port Input Mode Data IN – אופן קלט של פורט הכנסת נתון - בעזרתו קובעים האם ההדק יהיה קלט אנאלוגי או דיגיטאלי.
- רגיסטר PnMDOUT – Port Output Mode Data OUT – אופן פלט של פורט הוצאת נתון בעזרתו קובעים האם היציאה תהיה PUSH PULL או OPEN DRAIN .
- רגיסטר PnSKIP – Port Skip – דילוג פורט – האם לדלג או לחבר את הפורט דרך הקרוסבר.
- הקצאה של ההדק הרצוי לרכיב הפריפריאלי המתאים עם אחד מרגיסטרי הקרוסבר.
- אפשור הקרוסבר בעזרת ביט XBARE=1 ברגיסטר XBR1 .

3.1.1. : קביעת התצורה של ההדק עבור מצבים אנלוגיים

את המצב של כל הדק , אנאלוגי או דיגיטאלי, קובעים בעזרת רגיסטר PnMDIN . **במצב אנאלוגי ההדק מחובר עם נגד משיכה למעלה חלשה - weak Pullup** . במצב זה לא ניתן לכתוב או לקרוא מההדק. המצב הזה חוסך הספק בקרוסבר ומקטין רעש בהדק האנאלוגי. הדקים שהוגדרו כקלט דיגיטלי יכולים להתחבר גם לרכיב היקפי אנלוגי אבל זה לא מומלץ.

הדק שהוגדר כאנאלוגי ונבצע פעולת קריאה של ההדק ייתן תמיד 0 !!

כדי להגדיר הדק כאנלוגי, יש לנקוט בצעדים הבאים:

1. לאפס את הביט השייך להדק ברגיסטר PnMDIN . פעולה זו קובעת את ההדק כאנלוגי.
2. לשים 1 בביט השייך להדק ברגיסטר הפורט Pn .
3. לשים '1' בביט השייך להדק באוגר PnSKIP כדי לדלג על ההדק ועל ידי כך להבטיח שהקרוסבר לא יקצה להדק רכיב פריפריאלי פנימי.

האיור הבא מתאר את הרגיסטר של פורט 0 הנקרא P0MDIN . הרגיסטר קובע האם הדק מסוים של אחד הפורטים יהיה הדק אנאלוגי או הדק דיגיטאלי.

Bit	7	6	5	4	3	2	1	0
Name	P0MDIN[7:0]							
Type	R/W							
Reset	1	1	1	1	1	1	1	1

SFR Address = 0xF1; SFR Page = All Pages

Bit	Name	Function
7:0	P0MDIN[7:0]	**Analog Configuration Bits for P0.7–P0.0 (respectively).** Port pins configured for analog mode have their weak pullup, digital driver, and digital receiver disabled. 0: Corresponding P0.n pin is configured for analog mode. 1: Corresponding P0.n pin is not configured for analog mode.

איור 26 : מבנה רגיסטר PnMDIN

אם שמים בביט מסוים 0 אז ההדק יהיה אנאלוגי. אם שמים בביט 1 ההדק הוא דיגיטאלי. **אחרי פעולת RESET ההדקים הם דיגיטאליים (יש בהם '1').** לדוגמה : כדי לקבוע עבור פורט 0 ש 4 הדקים נמוכים שלו (P0.0 עד P0.3) הם כניסות אנלוגיות ו 4 הדקים גבוהים (P0.4 עד P0.7) יהיו דיגיטאליים נרשום : P0MDIN = 0xF0; .

נתאר כיצד נראה רגיסטר PnMDOUT . לדוגמה נציג את P1MDOUT שנראה באיור הבא:

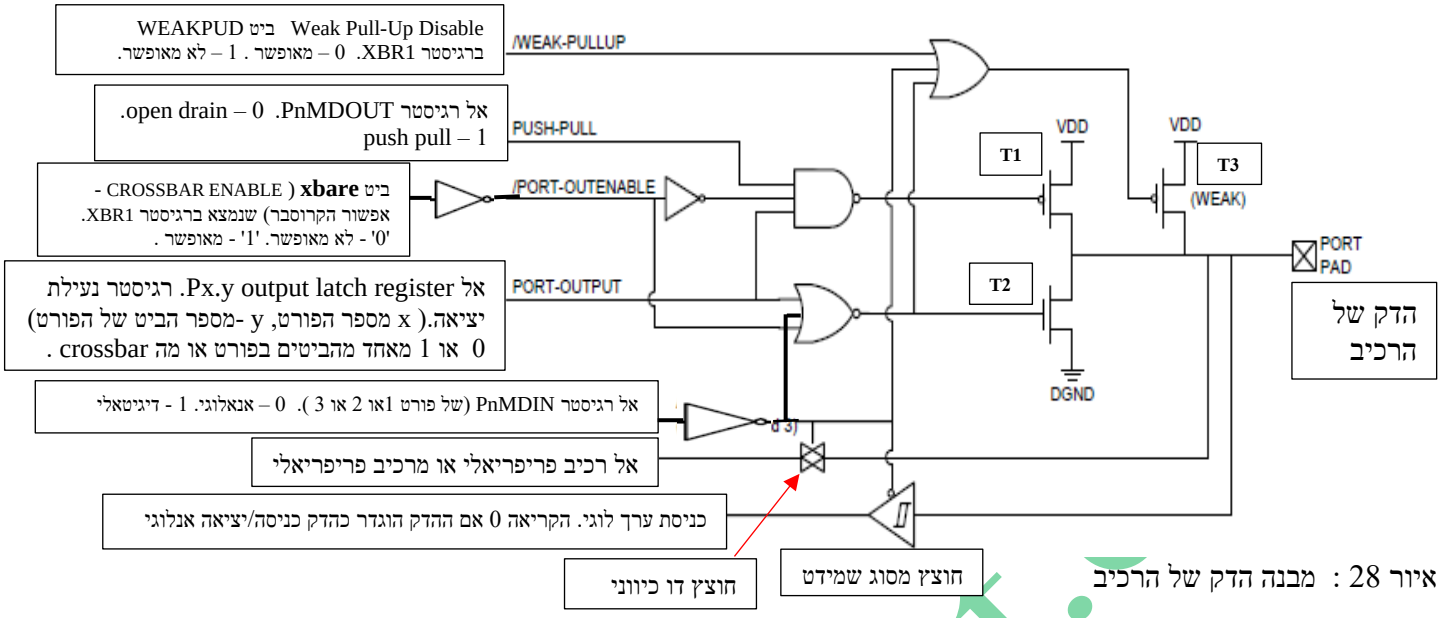
Bit	7	6	5	4	3	2	1	0
Name	P1MDOUT[7:0]							
Type	R/W							
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xA5; SFR Page = All Pages

Bit	Name	Function
7:0	P1MDOUT[7:0]	**Output Configuration Bits for P1.7–P1.0 (respectively).** These bits are ignored if the corresponding bit in register P1MDIN is logic 0. 0: Corresponding P1.n Output is open-drain. 1: Corresponding P1.n Output is push-pull.

איור 27 : רגיסטר P1MDOUT.

בביט ששמים '0' קובעים שההדק יהיה open drain . בביט ששמים '1' ההדק יהיה push pull . **אחרי RESET ההדקים הם open drain** . באיור הבא מתואר מבנה של הדק אחד של הרכיב ואפשרויות התכנות שלו.



איור 28 : מבנה הדק של הרכיב

בטבלה שבהמשך נציג את מצב הדק PORT PAD עבור האפשרויות השונות של הכניסות. ניעזר במצב הטרנזיסטורים T1, T2 ו T3.
מצב OFF אומר שהטרנזיסטור בקטעון. מצב ON אומר שהטרנזיסטור עובד באזור האוהמי שלו (בטרנזיסטור "רגיל" זה נקרא רוויה) ומפל המתח עליו קרוב ל 0 . טרנזיסטור T3 עובד בין 2 מצבים. א. קטעון ב. עבודה באזור הפעיל שלו והוא מייצג התנגדות אוהמית של מאות קילו אוהם.

T1 ו T2 הם טרנזיסטורי MOS FET מסוג העשרה T1 הוא עם אפיק P - (P channel) ואילו T2 עם אפיק (N - N channel).
T1 מחובר כך : השער שלו – GATE - מחובר ליציאת שער ה NAND , המקור -SOURCE- מחובר ל V_{DD} והמפק – DRAIN מחובר למפק של טרנזיסטור T2 . הטרנזיסטור T2 מחובר כך : שהשער שלו – GATE - מחובר ליציאת שער ה NOR , המקור -SOURCE- מחובר לאדמה והמפק – DRAIN – מחובר למפק של T1 . באיור שמו בכניסת השער של T1 עיגול מציין היפוך וכאן הכוונה שכאשר בשער שלו יש 0 הוא במצב ON.
כש T1 מקבל בשער 0 הוא ב ON . כאשר T1 מקבל בשער 1 הוא ב OFF .
כש T2 מקבל בשער 0 הוא ב OFF . כאשר T2 מקבל בשער 1 הוא ב ON .
כש T3 מקבל בשער 0 הוא בהולכה (באזור הפעיל) . כאשר T3 מקבל בשער 0 הוא ב OFF .
המצב Weak Pull Up הוא מצב שבו T3 לא בקטעון אלא עובד באזור הפעיל שלו והוא משמש כנגד בעל ערך של כ 100 קילו אוהם.
המצב קיים רק ב 2 התנאים הבאים : א. הביט WEAKPUD ברגיסטר XBR1 הוא 0 . ב. ההדק באופן דיגיטאלי ונקבע ל open drain וכתבנו 1 לנועל הפורט - port latch - שבו נמצא ההדק. מצב זה נבחר בדרך כלל כאשר ההדק נקבע כהדק קלט.

נתאר את מצב הדק היציאה לפי המצבים השונים האפשריים בטבלה הבאה :

האם ההדק מאופשר/מצב עבודה /מצב היציאה	T2	T1	יציאת שער NOR	יציאת שער NAND	יציאת שער המהפך השני	Port-output Px.y - output latch register	PUSH-PULL PnMDOUT	ביט XBARE ברגיסטר XBR1
ההדק לא מאופשר דרך הקרוסבר.	OFF	OFF	1	1	0	0	0	**0**
ההדק לא מאופשר דרך הקרוסבר.	OFF	OFF	0	1	0	1	0	**0**
ההדק לא מאופשר דרך הקרוסבר.	OFF	OFF	0	1	0	0	1	**0**
ההדק לא מאופשר דרך הקרוסבר.	OFF	OFF	0	0	0	1	1	**0**
ההדק מאופשר, push pull , ביציאה 0 .	ON	OFF	1	1	1	0	1	**1**
ההדק מאופשר, push pull , ביציאה 1 .	OFF	ON	0	1	1	1	1	**1**
ההדק מאופשר, open drain , ביציאה 0 .	ON	OFF	1	1	1	0	0	**1**
ההדק מאופשר, open drain, עכבה גבוהה HIGH Z, **קלט דיגיטאלי**	OFF	OFF	0	1	0	1	0	**1**

טבלה 5 : מצב ההדק במצבי התכנות השונים.

נסכם את הטבלה:

א. אם לא איפשרנו את רגיסטר הקרוסבר – כלומר XBARE=0 אז T1 ו T2 בקטעון והרגל נמצאת "באוויר" . זהו מצב שנקרא HIGH Z - עכבת גבוהה. המצב מתואר בצבע אדום.

ב. אם יש חיבור להדק דרך הקרוסבר (XBARE = 1) אז במצב push pull מה שנשים בביט של הפורט ייצא בהדק החיצוני של הרכיב המתאים לפורט. מתואר בצבע ירוק. במצב זה הרכיב יוציא במצב של '1' ביציאה את מתח ה V_{DD} (מצב זה נקרא SOUCE כי הרכיב שלנו משמש להוצאת זרם החוצה.) ובמצב של 0 הטרנזיסטור T2 יהיה ב ON (T1 ב OFF) וניתן יהיה לסגור דרכו זרם לאדמה (המצב נקרא SINK כי מטביעים/סוגרים זרם דרך הטרנזיסטור לאדמה). המצב מתואר בצבע ירוק.

ג. המצב בצבע כחול מתאר מצב שנקרא open drain . אם נוציא לפורט 0 הטרנזיסטור T2 יהיה ב ON וניתן יהיה לסגור דרכו זרם לאדמה . אם נשים בביט של הפורט '1' אז שני הטרנסיטורים יהיו ב OFF .

יש לשים לב :

- כדי לקבוע שהדק יהיה **קלט דיגיטאלי** יש לשים '0' בביט המתאים ב PnMDOUT ו '1' בנועל הפורט המתאים Pn.
- להדקים שמוגדרים **כניסות אנלוגיות** (בעזרת '0' בביט הרצוי ב PnMDIN) יש לשים דילוג skip בקרוסבר בביט המתאים ב PnSKIP.
- יש **לאפשר את הקרוסבר** כדי להשתמש בפורטים P0 P1 P2 P3 כהדקי I/O סטנדרטיים במצב של פלט. פורט 4 תמיד מתפקד כ GPIO (קלט פלט לשימוש כללי).

3.1.2 - הרחבה על המצבים השונים של ההדק

כל הדק יכול לעבוד באחד מארבעה מצבי העבודה הבאים :

- יציאה עם מרזב (או מפק) פתוח– open drain .
- יציאה עם מרזב (מפק) פתוח עם נגד משיכה למעלה – pull up resistor .
- יציאה עם push pull .
- כניסה אנאלוגית.

3.1.3 סיכום : אתחול/קביעת מצב של הדק I/O .

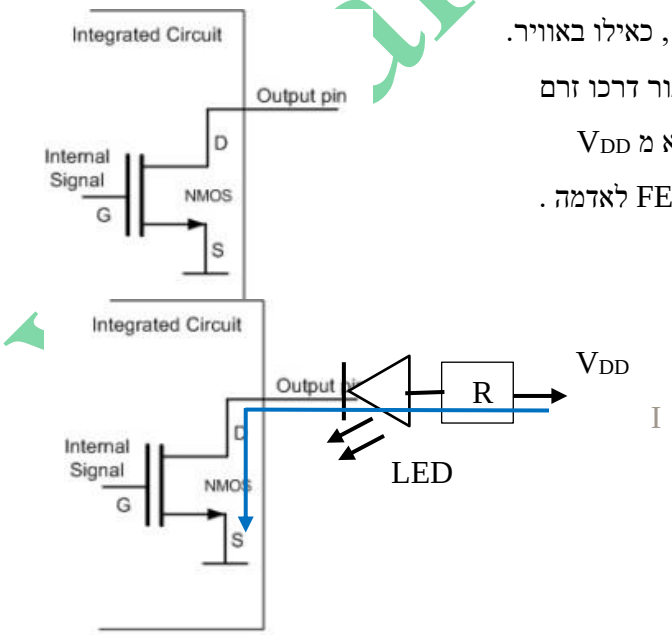
אתחול של פורט כקלט/פלט מורכב מהצעדים הבאים:

1. בחירת אופן הכניסה (אנאלוגי או דיגיטאלי) לכל הדקי הפורט בעזרת רגיסטר PnMDIN (Port Input Mode register – רגיסטר אופן כניסה של הפורט). **אחרי RESET ההדקים דיגיטאליים.**
2. בחירת אופן יציאה (open drain או push pull) לכל הדקי הפורט בעזרת PnMDOUT (Port Output Mode register – רגיסטר אופן יציאה של הפורט). **אחרי RESET כל ההדקים open drain** .
3. בחירת הדקים רצויים לדילוג (שלא נשתמש בהם) עם הקרוסבר בעזרת רגיסטר PnSKIP (Port Skip register – רגיסטר דילוג פורט).
4. שיוך הדקי פורט לרכיבים ההיקפיים הרצויים בעזרת XBR2 XBR1 XBR0 .
5. אפשור הקרוסבר בעזרת ביט XBARE - (XBARE=1) . אם לא מאפשרים את הקרוסבר אז הפורטים או הרכיבים ההיקפיים אינם מתחברים אל ההדקים החיצוניים והם נמצאים "באוויר" . מצב זה נקרא HIGH Z – עכבה גבוהה.

3.1.4 : יציאה עם open drain

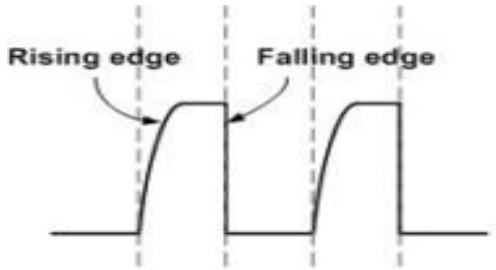
במצב של Open Drain הדק ה D (Drain) של ה FET מתחבר להדק היציאה. אם בשער ה FET ניתן '0' אז ה FET בקטעון והדק היציאה "צף" – float , כאילו באוויר.

אם ניתן בשער ה FET - '1' אז ה FET במצב ON וניתן לסגור דרכו זרם לאדמה כפי שרואים באיור הבא בחלק התחתון. סגירת הזרם היא מ V_{DD} דרך נגד R , דרך הלד ודרך הדקי Drain --> Source של ה FET לאדמה .



איור 29 : מצבי open drain

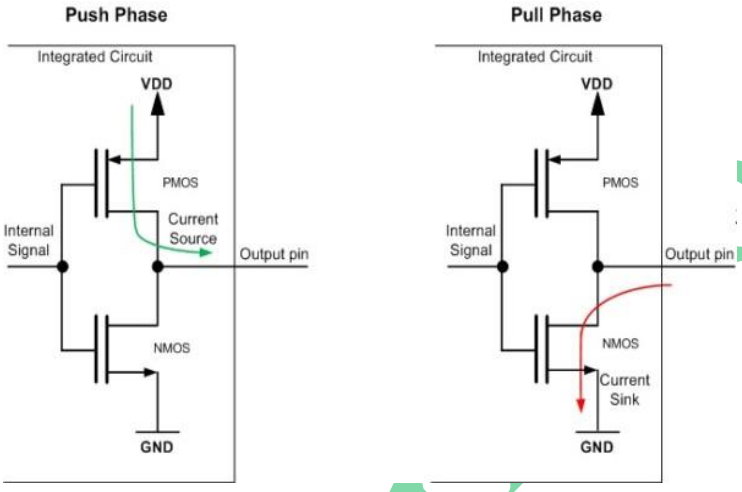
יציאות **open drain** משמשות בדרך כלל בממשקי תקשורת שבה מספר רכיבים מחוברים באותו קו (למשל I2C, One-Wire וכו ').
כאשר כל יציאות הרכיבים המחוברים לקו כלשהו נמצאים במצב Hi-Z – עכבה גבוהה, הקווים "צפים" אבל מחברים **נגד היצוני** pullup שגורם לרמה של '1' . כל התקן יכול למשוך את הקו ללוגיקה 0 ללא חשש שמישהו ישים '1' ותיווצר בעיה כי ב '1' ההדקים "צפים".
נגד המשיכה למעלה החיצוני גורם ביחד עם קיבול היציאה למסננת LPF (מסננת לתדר נמוך) ולהגדלת זמני המעבר בעלייה במעבר מ 0 ל 1 . הדבר נראה באיור הבא:



איור 30 : זמני עלייה יחסית לזמני ירידה ב open drain

3.1.5 : מצב PUSH PULL :

באיור הבא מתואר מצב push pull של יציאת הדק.



איור 31 : מצב push pull

מצד שמאל באיור רואים מצב Push. הטרנזיסטור העליון מוליך והתחתון בקטעון. במצב זה יוצא מתח '1' בהדק. מצב זה נקרא SOURCE (מקור זרם). ההדק ברכיב שלנו דוחף זרם החוצה עד 10 מילי אמפר. **I_{OH}= -10mA**.(הסימן מינוס כי "הפסדנו" זרם).
באיור הימני יש את מצב ה Pull שבו הטרנזיסטור העליון בקטעון והתחתון ב ON ויש סגירת זרם מבחוץ דרך ה FET לאדמה. במצב זה נקבל 0 ביציאה. מצב זה נקרא SINK (הטבעת זרם). ברכיב שלנו הזרם הנכנס המקסימלי הוא 25 מילי אמפר. **I_{OL} =25mA**
סוג זה של פלט אינו מאפשר חיבור של כמה רכיבים יחד כמו בפס - BUS . בפס יכול לקרות מצב שבו רכיב אחד מוציא 1 ורכיב אחר מוציא 0 על אותו קו , כך שיכולה להיווצר בעיה. התצורה של push pull - משיכה דחיפה היא הנפוצה ביותר בממשקים בהם הקווים חד כיווניים (השידור על הקו הוא רק בכיוון אחד – SPI, UART וכו ').

יתרונות :

* רכיבים בחיבור push pull נותנים ביצועים טובים יותר כשמדובר בזמני מעבר מרוויה לקטעון ולהיפך .
* במצב של '1 ביציאה הוא יכול להוציא זרם החוצה.
* ב open drain תצרוכת הספק גבוהה יותר במעברים מ 0 ל 1 ולהפך בגלל נגד המשיכה למעלה החיצוני שמוסיפים.

3.2 נתונים טכניים של הרכיב ושל הדק I/O

בטבלה הבאה מוצגים הערכים הנקובים המקסימליים - Absolute Maximum Ratings :

Parameter	Conditions	Min	Typ	Max	Units
Junction Temperature Under Bias		–55	—	125	°C
Storage Temperature		–65	—	150	°C
Voltage on $\overline{RST}$, VBUS, or any Port I/O Pin with Respect to GND	$V_{DD} \geq 2.2$ V $V_{DD} < 2.2$ V	–0.3 –0.3	— —	5.8 V_{DD} + 3.6	V V
Voltage on V_{DD} with Respect to GND	Regulator1 in Normal Mode Regulator1 in Bypass Mode	–0.3 –0.3	— —	4.2 1.98	V V
Maximum Total Current through V_{DD} or GND		—	—	500	mA
Maximum Output Current sunk by $\overline{RST}$ or any Port Pin		—	—	100	mA

טבלה 6 : ערכים נקובים מקסימליים.

מה אומרת כל שורה בטבלה ?

שורה הראשונה - תחום הטמפרטורה בצמתי ה PN ברכיב כאשר עובדים עם מתח.

שורה שנייה - תחום טמפרטורת האחסנה (הרכיב איננו עובד).

שורה השלישית – תחום מתחים המותר בהדקים יחסית לאדמה. (רואים שניתן לחבר רכיבים שעובדים עם 5 וולט אבל עד 5.8v).

שורה רביעית – מתח הספק בהדק ה V_{DD} יחסית לאדמה (מקסימום 4.2 וולט).

שורה חמישית – זרם מקסימלי בהדק ה V_{DD} או האדמה (500mA) .

שורה אחרונה – מה הזרם הקסימלי בהדק של פורט או הדק ה RESET במצב הטבעת זרם – SINK (100mA).

מתח ספק הכוח האופיני הוא 3.3 וולט והמקסימאלי 3.6 וולט.

Digital Supply Voltage[1]		V_{RST}[1]	3.3	3.6	V

הטבלה הבאה מתארת מאפיינים טכניים של הדק IO של הרכיב:

V_{DD} = 2.7 to 3.6 V, –40 to +85 °C unless otherwise specified.

Parameter	Test Condition	Min	Typ	Max	Unit
Output High Voltage	I_{OH} = –3 mA, Port I/O push-pull I_{OH} = –10 µA, Port I/O push-pull I_{OH} = –10 mA, Port I/O push-pull	V_{DD} – 0.7 V_{DD} – 0.1 —	— — V_{DD} – 0.8	— — —	V
Output Low Voltage	I_{OL} = 8.5 mA I_{OL} = 10 µA I_{OL} = 25 mA	— — —	— — 1.0	0.6 0.1 —	V
Input High Voltage		2.0	—	—	V
Input Low Voltage		—	—	0.8	V
Input Leakage Current	Weak Pullup Off Weak Pullup On, V_{IN} = 0 V	— —	— 15	±1 50	µA

טבלה 7 : מאפיינים של הדק IO

המאפיינים נכונים עבור מתח ספק בין 2.7 וולט ועד 3.6 וולט ובתחום טמפרטורה בין מינוס 40 לפלוס 85 מעלות צלסיוס.

שורה ראשונה – מתח היציאה במצב גבוה : ההדק הוגדר למצב push pull . הסימן מינוס מראה שהרכיב מוציא זרם החוצה (מצב **SOURCE**). יש 3 מצבי זרם . במצב שהרכיב מוציא החוצה 3 מילי אמפר המתח יהיה במצב הכי גרוע (המתח לא יהיה פחות) מ

V_{DD} - 0.7 וולט. בזרם של 10 מילי אמפר זה לא יהיה פחות מ V_{DD} - 0.8 וולט. כדאי לשים לב שהזרם מסומן במינוס כי במצב זה יוצא זרם מהרכיב אל מערכת חיצונית.

שורה שנייה – מתח יציאה במצב נמוך כלומר מצב הטבעת זרם שבו נסגר זרם ממעגל חיצוני דרך הטרנזיסטור שבג'וק : לא מצוין כאן מצב העבודה (push pull או open drain) כי מדובר באותו טרנזיסטור . גם כאן 3 מצבי זרם. ב 8.5 מילי אמפר המתח ביציאה לא יהיה גבוה מ 0.6 וולט . בזרם יציאה של 25 מילי אמפר מתח היציאה האופייני 1 וולט. לא מצוין מה המתח המקסימלי . הזרם מסומן בפלוס כי נכנס זרם מבחוץ אל הרכיב. במצב **source** יוצא זרם מהרכיב ולכן הוא מסומן במינוס.

שורה שלישית – מתח כניסה במצב גבוה - I_{OH} : מ 2 וולט המתח נחשב לגבוה (כ 2/3 מה V_{DD}). זה נקרא Vi_{HMIN}.

שורה רביעית - מתח כניסה במצב נמוך - I_{OL} : מקסימום 0.8 וולט. מתח מעל זה לא נחשב '0' . זה נקרא Vi_{LMAX}

מתח כניסה בין 0.8 ל 2 וולט לא מוגדר כ 0 ולא כ 1 . בדרך כלל ה FET עובד כאן בתחום הפעיל שלו ולא בין ON ל OFF והיצרן לא מבטיח מה יהיה מצב היציאה.

שורה חמישית – זרם זליגה בכניסה : (למעשה מדובר כאן על הדק IO המוגדר ככניסה). ללא נגד weak pull up הזרם פלוס מינוס 1 מיקרו אמפר. אם הכניסה הוגדרה עם weak pull up=on אז המקסימום יהיה 50 מיקרו אמפר.

נתונים עבור שעון המערכת.

ניתן לעבוד עם מתנד פנימי בתדר גבוה ואז היצרן מבטיח שהתדר האופיני הוא 48MHz עם אפשרות סטיה של 0.7MHz .

Parameter	Test Condition	Min	Typ	Max	Unit
Oscillator Frequency	IFCN = 11b	47.3	48	48.7	MHz

IFCN הוא קיצור של Internal Frequency CoNtrol bits - ביטים של בקרת תדר פנימית. אלו 2 ביטים ברגיסטר OSCICN השולט על מערכת התדר הגבוה של הרכיב.

אם עובדים עם מתנד פנימי בתדר נמוך אז התדר האופייני 80 קילו הרץ עם סטייה של 5 קילו הרץ .

Parameter	Test Condition	Min	Typ	Max	Unit
Oscillator Frequency	OSCLD = 11b	75	80	85	kHz

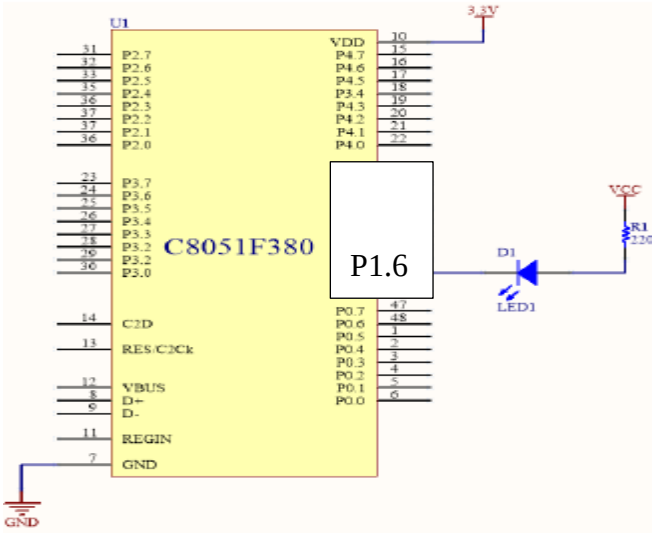
גם OSCLD הם 2 ביטים ברגיסטרי בקרה של שעון המערכת.

אם עובדים עם גביש חיצוני אז התדר מ 0.02 מגה עד 30 מגה הרץ. אם נחבר מתנד חיצוני מ CMOS אז ניתן לעבוד עם תדר שעון בין 0 ל 48 מגה הרץ.

Parameter	Test Condition	Min	Typ	Max	Unit
External Crystal Frequency		0.02	—	30	MHz
External CMOS Oscillator Frequency		0	—	48	MHz

3.3 חיבור התקן פלט וקלט פשוטים - חיבור לד ומפסק

נתחיל עם חיבור לד למיקרו C8051F380. באיור הבא מתוארת לד המתחברת אל הדק P1.6 של המיקרו בקר.



איור 32 : חיבור לד למיקרו להדק P1.6

דוגמה 1 : התוכנית הבאה מדליקה ומכבה לד 10 פעמים ואז מסתיימת.

הלד תידלק כאשר נוציא 0 ביציאת ההדק ותכבה כאשר נוציא 1 . זהו מצב של הטבעת זרם (**SINK**) שבו זורם זרם מה V_{CC} דרך נגד העבודה R1 של 220 אוהם, דרך הלד LED1 ודרך ה FET שנמצא בהדק P1.6 לאדמה.

נחשב כיצד נבחר הנגד R של 220 אוהם. (נניח $V_{CC} = 3.3V$).

$$V_{CC} = V_{R1} + V_{LED} + V_{OLmax}$$

כאשר V_{OLmax} הוא המתח המקסימלי ביציאת ההדק במצב של 0 .

בשורה הבאה רשום מהו המתח V_{OLmax} במצב שבהדק יש 0 . רואים שעבור זרם של 8.5 מילי אמפר המתח לא יעלה על 0.6 וולט.

במצב של זרם 10µA המתח 0.1 וולט וכאשר הזרם ביציאה 25mA המתח יכול להגיע ליותר מ 1 וולט.

			MIN	TYP	MAX	
Output Low Voltage	I_{OL} = 8.5 mA	—	—	0.6	V	
	I_{OL} = 10 µA	—	—	0.1		נשנה
	I_{OL} = 25 mA	—	1.0	—		את

המשוואה על ידי הוספת הזרם במעגל ונקבל:

$$V_{CC} = I*R1 + V_{LED} + V_{OLmax}$$

נעביר אגפים ונקבל :

$$R1 = (V_{CC} - V_{LED} - V_{OLm}) / I$$

כאשר לד בצבע אדום בהולכה נופל עליה מתח של בין 1.2 ל 2.2 וולט (תלוי בסוג ובצבע של הלד. אנחנו נבחר 1.6v). נבחר גם זרם של 10mA שבו הלד מאירה בצורה סבירה בחדר (לא באור יום מחוץ לחדר). ניקח V_{Olmax} =0.7 כי הזרם בחרנו זרם גדול מזה שבשורה ונקבל:

$$\mathbf{R1 = (3.3 – 1.7 – 0.7) / 10*10^{-3} = 90\Omega}$$

בחרנו נגד של 100 אוהם כי הוא נפוץ יותר מנגד של 90 אוהם. הזרם יקטן מעט אבל לא משמעותי . נחשב את הזרם שיזרום דרך הלד :

$I = (V_{CC} - V_{LED} - V_{OLmax}) / R1 = (3.3 - 1.7 - 0.7) / 100 = 9mA$

גם בזרם כזה הלד דולקת באור חדר בצורה יפה.

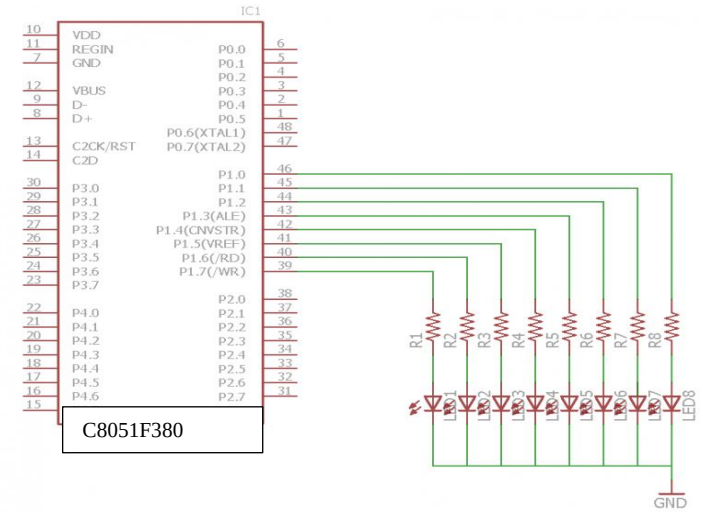
התוכנית הבאה מהבהבת את הלד :

הערות: **1**. נכון שעדיין לא למדנו תוכנה של ה 51 אבל נצא מההנחה ששפה כמו C או C# מוכרת. **2**. לא הפעלנו בתוכנית את שעון המערכת (שעליו נדבר בהמשך). **3**. לא ביטלנו את ה watchdog – כלב שמירה (נזכיר בהמשך).

```
#include "REG51F380.h"      // הנחייה לקומפיילר לכלול קובץ כותר המכיל את ההגדרות של הרגיסטרים והביטים
// include <REG51F380.h>  : אם קובץ הכותר נמצא בספריה הסטנדרטית של התוכנה ולא בספריה של התוכנית הנוכחית נרשום
sbit LED=P1^6;              // הנחייה לקומפיילר שבכל מקום שנרשום LED ירשום P1.6

void delay(long delayTime)  //  פונקציית השהייה
{
        long x;
        for(x=0;x<delayTime;x++);
}
// -----  התוכנית הראשית ------
void main()
{
  int i;
//   אין צורך לקבוע מצב P1MDIN כי ברירת המחדל שלו זה כניסות דיגיטאליות
// P1.6  נקבע שביט P1.6 יתחבר דרך הקרוסבר אל הדק יציאה של הרכיב
   P1MDOUT = 0x40;   //  connected to led P1.6 as push pull
   XBR1 = 0x40;   // weak pull-ups  אפשור הקרוסבר עם
        for(i=0;i<10;i++)  //  מראה כמה פעמים לבצע את הלולאה
        {
                LED=0;            // led ON  לד דולקת
                delay(250000);   // מספר שנבחר לפי ניסוי וטעיה . 250000 זימון פונקציה שמשהה לזמן של
                LED=1;                // led OFF  לד בחושך
                delay(250000);
        }
        while (1);  // בלולאה אין סופית בסיום התוכנית "נתקעים"
}
```

דוגמה 2 : הדלקה של 8 לדים המחוברים לפורט 1 .

באיור הבא מחוברות 8 לדים לפורט 1 . הלדים יידלקו כאשר נוציא 1 לפורט וייכבו כאשר נוציא 0 . מצב עבודה כזה נקרא מצב **SOURCE** – מקור זרם.



איור 33 : הבהוב 8 לדים המחוברים לפורט 1 .

בשורה הבאה מדף הנתונים רואים שכאשר מוציאים 1 בזרם של 3mA מתח היציאה שנקרא V_{OHmin} איננו V_{DD} שהוא בדרך כלל 3.3 וולט אלא $V_{OHmin} = V_{DD} - 0.7$. **צריך לזכור שאסור לעבור זרם של 10mA** .

Output High Voltage	I_{OH} = –3 mA, Port I/O push-pull I_{OH} = –10 µA, Port I/O push-pull I_{OH} = –10 mA, Port I/O push-pull	V_{DD} – 0.7 V_{DD} – 0.1 —	— — V_{DD} – 0.8	— — —	V

נבדוק מהו ערך הנגדים R1 עד R8 שיש לחבר בטור ללדים. נשתמש בנתונים ממקודם : VLED=1.7 ועם הזרם ניזהר לא להגיע ל 10 מילי אמפר אלא נבחר 7 מילי אמפר.

$V_{OHmin} = V_{R1} + V_{LED} = I*R1 + V_{LED}$

$R1 = (V_{OHmin} - V_{LED}) / I = (V_{DD} - 0.7 - 1.7) / 7 * 10^{-3} = (3.3 - 0.7 - 1.7) / 7 * 10^{-3} = 128.5\Omega$

במקרה הזה נבחר נגד של 130 אוהם שהוא נגד נפוץ שערכו קרוב לערך המחושב.

הערה: כדאי לעבוד במצב SINK כי במצב זה יש אפשרות לזרם גדול יותר ביציאת ההדק.

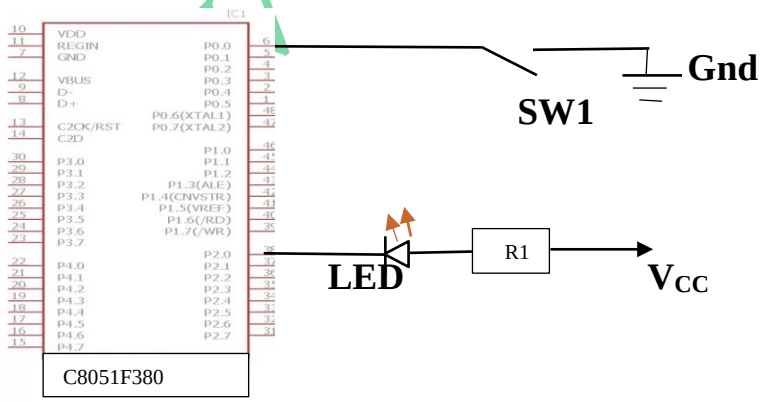
התוכנית : גם כאן לא הפעלנו בתוכנית את שעון המערכת ולא ביטלנו את ה watchdog – כלב שמירה (נזכיר בהמשך).

```
//#include <REG8051F380.h>          // SFR declarations
void PORT_Init (void)      //   אתחול הפורטים
{
  P1MDIN = 0xFF;        //  כל הדקי פורט 1 דיגיטאליים ולא אנאלוגים
  P1MDOUT = 0xFF;     // push pull   היציאות של פורט 1 הן
```

```
  XBR1    = 0x40;          // weak pull-ups   אפשור הקרוסבר  עם                    //
}
// Delay     פונקציית השהיה
void delay(unsigned int t)
{
   unsigned int i,j;
//   נעשה לולאה בתוך לולאה
   for(i=0;i<t;i++)
      for(j=0;j<250;j++);
}
//-----------------------------------------------------------------------------
//   הפונקציה הראשית
//-----------------------------------------------------------------------------
 void main (void)
{
   PORT_Init();                    //  זימון פונקציה לאתחול הפורטים
  while (1)
  {

      P1 = 0xFF;                  //  הדלקת הלדים
      delay(1000);
      P1 = 0x00;                  //  כיבוי הלדים
      delay(1000);
  }
}
//-----------------------------------------------------------------------------
```

דוגמה 3 : חיבור מפסק ולד

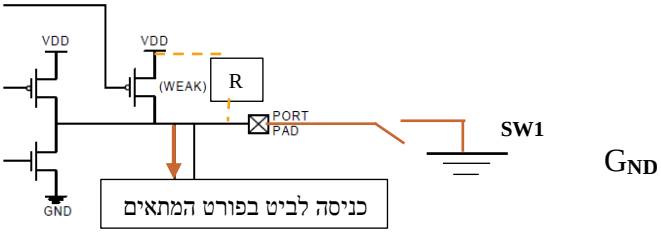
באיור הבא מתואר חיבור של מפסק ולד. המפסק – SW1 – מתחבר ל P0.0 . לד מחוברת להדק P2.0 .



איור 34 : חיבור מפסק ולד.

נרשום תוכנית שכאשר לוחצים על המפסק הלד נדלקת לזמן כלשהו ולאחריו היא נכבית (בדומה לאור בחדר מדרגות בבניין).

המפסק מחובר מצד אחד ל P0.0 ומצידו השני לאדמה. את ההדק P0.0 נגדיר בתוכנית כהדק קלט עם נגד משיכה למעלה חלשה - weak pullup . במצב זה מחובר בתוך הרכיב FET שהתנגדות ה R_{DS} שלו כ 100 קילו אוהם , צד אחד שלו מחובר לאדמה וצד שני שלו מחובר להדק היציאה. הדבר מתואר באיור הבא והחיבור המקווקו עם הנגד R מדמה את התנגדות ה R_{DS} של ה FET :



איור 35 : חיבור עם נגד משיכה למעלה

במצב שההדק בקלט - שני הטרנזיסטורים השמאליים בקטעון ולמעשה מנותקים מהדק היציאה.

כאשר המפסק SW1 פתוח – ההתנגדות R_{DS} "מושכת למעלה" את המתח בהדק ל V_{DD} ונקבל בהדק '1' לוגי .

כאשר המספק SW1 סגור - האדמה עוברת דרך צד ימין של המפסק לצד שמאל שלו ואל הדק הפורט ונקבל בהדק '0' .

הלד נדלקת כאשר נוציא ל P2.0 '0' .

הלד נכבית כאשר נוציא ל P2.0 '1' . התוכנית תיראה כך :

```
#include "REG51F380.h"   //  הנחייה לקומפיילר לכלול את קובץ הכותר. יש בו הצהרות על כתובות כל הרגיסטרים.
sbit LED_pin = P2^0;  // שיוך השם LED_Pin לביט P2.0 שנמצא ב SFR . להדק זה מחוברת הלד.
sbit switch_pin = P0^0;  // שיוך השם switch_Pin לביט P0.0 שנמצא ב SFR . להדק זה מחובר המפסק.
void Delay(int);          // הצהרה על פונקציית השהיה delay
void main (void)          //  הפונקציה הראשית
{
  P0MDIN = 0xFF;   //  כל הדקי פורט 0 כניסות דיגיטאליות ולא אנאלוגיות
  P2MDOUT = 0xFF;                  // היציאות של פורט 2 הן push pull
  XBR1    = 0x40;                  // אפשור הקרוסבר ו weak pull-ups
  switch_pin = 1;   //  קביעת ההדק כקלט
  LED_pin=1;      //כיבוי הלד בהתחלת התוכנית
 while(1)     // לולאה אין סופית
  {
    if(switch_pin == 0 )     // האם המפסק לחוץ ?
    {
            LED_pin = 0;     // הדלקת הלד
            Delay(30000);      // זימון פונקציית השהיה
             LED_pin = 1; //   כיבוי הלד
    }
  }
}
void Delay(int k)  //  פונקציית ההשהיה
{
   int j;
   int i;
   for(i=0;i<k;i++)
       for(j=0;j<100;j++);
 }
```

פרק 4 תוכנה – אסמבלי - שפת סף

הדור הראשון של שפות תוכנה היה שפת מכונה. בשפה זו הכניסו את הקוד הבינארי של הפקודה לכל כתובת רצויה. תהליך כתיבת תוכנית היה ארוך, מייגע ובלתי הגיוני.

הדור השני של השפות היה שפת סף . בשפה זו הפקודות קרובות לשפת האדם, על סף הכניסה לשפת האדם. פקודות MOV , ADD , MUL וכו' הן פקודות המבצעות העברה , חיבור וחילוק בהתאמה. שפה זו נקראת גם שפת אסמבלי והתוכנה המתרגמת משפת סף לשפת מכונה נקראת אסמבלר. בשפת אסמבלי כל פקודה מתורגמת לפקודת מכונה. פקודת אסמבלי "תופסת" בין 1 ל 3 בתים בשפת מכונה.

בהמשך הגיע שפת דור שלישי שהיא שפות על. שפת C נפוצה מאוד בין שפות העל והיא גם הבסיס לשפות נוספות. בשפה זו יש פקודות הדומות לשפת האדם כמו if , while , for וכו'.

הדור הרביעי של שפה היא תכנות מונחה עצמים – Object Oriented .

בשפות על יש לשים לב שהפקודות נרשמות באותיות אנגלית קטנות. באסמבלי ניתן לכתוב באותיות קטנות או גדולות.

4.1 : תכנות ותמיכה ב debugging

תכנות זיכרון התוכנית - ה FLASH והתקשורת עם תוכנת התמיכה של ה debug שב FLASH מושגת בעזרת ממשק הפיתוח של Silicon Labs שנקראת 2-Wire (או בקיצור C2).

לוגיקת התמיכה לאיתור באגים הנמצאת בזיכרון הג'וק מאפשרת איתור באגים במהירות רבה, עם אפשרות של הגדרת חומרה, נקודות עצירה – Break Points, הפעלה – Run , עצירה - Stop , ביצוע צעד יחיד – Single Step (כולל בפסיקות ורוטינות), בדיקת המחסנית וקריאה/כתיבה של תוכן הבנקים והזיכרון.

CIP-51 נתמך על ידי כלי פיתוח ו IDE - Integrated Development Environment - סביבת פיתוח משולבת (גם מעבד תמלילים , שמירת תוכניות , טעינת תוכניות) וגם debugger של Silicon Labs וגם ספקי צד שלישי כמו MicroVision של KEIL.

4.2 : סט ההוראות

סט ההוראות של CIP 51 תואם באופן מלא לתקן MCS-51 המקורי של intel . כלי פיתוח סטנדרטיים לפיתוח תוכנה של 8051 מתאים גם ל CIP-51 . לכל ההוראות אותו קוד בינארי , אותם סוגי מיעון ואותה השפעה על רגיסטר PSW והדגלים (שנדבר עליו בפרק הזה). מהירות ביצוע הפקודות ב cip-51 שונה (יכולה להיות מהירה בהרבה).

4.3 : הוראות וזמני CPU

במיקרו 51 המקורי הייתה הבחנה בין מחזורי שעון ומחזורי מכונה. 12 מחזורי שעון נחשבו מחזור מכונה אחד. פקודה הייתה מבוצעת בין מחזור מכונה אחד ועד 4 מחזורי מכונה בפקודות כפל וחילוק (מ 12 מחזורי שעון ועד 48 מחזורי שעון). ב Cip-51 מדובר רק במחזורי שעון בגלל ארכיטקטורת pipeline – קו צינור – שלו. רוב הפקודות מבוצעות באותו מספר מחזורי שעון. הוראות הסתעפות (קפיצה מותנית - לפי תנאי – לדוגמה קפיצה אם המספר אפס) לוקחות עוד 2 מחזורי שעון נוספים אם יש קפיצה על זה שאין קפיצה.

בתדר שעון של 48MHz יש אפשרות לבצע 48 MIPS . המושג MIPS הוא Mega Instructions Per Second – מיליון הוראות בשנייה . סה"כ יש 109 הוראות והחלוקה שלהן למהירות ביצוע נראית בטבלה הבא :

Clocks to Execute	1	2	2/4	3	3/5	4	5	4/6	6	8
Number of Instructions	26	50	5	10	6	5	2	2	2	1

טבלה 8 : זמני ביצוע של פקודות

השורה הראשונה בטבלה מתארת כמה פולסי שעון לוקח לפקודה להתבצע. השורה השנייה מתארת את מספר הפקודות.
מהטבלה רואים שיש 26 פקודות שמבוצעות במחזור שעון 1 , 50 פקודות ב 2 מחזורי שעון וכך הלאה עד לפקודה אחת שנמשכת 8 מחזורי שעון (פקודת חלוקה בין רגיסטרים A ו B).

4.4 : פקודות אסמבלי ושיטות מיעון :

כל פקודת אסמבלי תופסת 1 עד 3 בתים בזיכרון התוכנית. הפקודה מורכבת מפעולה (OPERATION) ו 0 עד 3 אופרנדים המופרדים ביניהם על ידי פסיק. (אופרנד הוא קבוע או משתנה עליו פועלת הפעולה). בפקודות שיש מקור ויעד אז היעד מצוין ראשון ולאחריו המקור. לדוגמא : MOV A,R2 אז A הוא היעד ו R2 הוא המקור. הפקודה אומרת להעביר את הנתון שברגיסטר R2 אל האקומולטור.

4.4.1 שיטות מיעון - Addressing Modes

שיטות מיעון מתארות את אופן העברת הנתון בין המיקרו בקר ובין הכתובות בזיכרונות המחוברים אל המיקרו בקר או אל הרגיסטרים השונים שלו. המיקרו בקר תומך ב 8 סוגים של מיעון (addressing):

- מיעון ישיר – Direct Addressing
- מיעון עקיף – Indirect Addressing
- מיעון מיידי – Immediate constant Addressing
- מיעון רגיסטר - Register Addressing
- מיעון אינדקס - Index Addressing
- מיעון יחסי – Relative Addressing
- מיעון אבסולוטי (מוחלט) – Absolute Addressing
- מיעון רחוק - Long Addressing

4.4.1.1 – Direct Addressing - מיעון ישיר .

במיעון זה האופרנד (כתובת) מצוין בפקודה. רק ה RAM פנימי (128 בתים בלבד ! – בין 0 ל 127) וה SFR (כתובות בין 128 ו 255)משתתפים במיעון זה .

דוגמה : MOV 70H, #55H שאומר להעביר את הנתון 50H לתא זיכרון שכתובתו 70H . הכתובת 70H מצויה בפקודה עצמה. (הסימן # משמעותן היא "את הנתון").

4.4.1.2 - Indirect Addressing - מיעון עקיף

במיעון זה הכתובת נמצאת באחד הרגיסטרים. נעזרים באוגרים R0 , R1 וברגיסטר ה DPTR שהערך שיש בהם הוא למעשה כתובת. הערך ברגיסטרים R0 או R1 יכול להיות בין 0 ל 255 ומכאן שהכתובת יכולה להיות בין 0 ל 255. כאשר הכתובת בין 128 ל 255 פונים ל RAM הנתונים הגבוה (ולא ל SFR) . הערך ברגיסטר ה DPTR יכול להיות בין 0 ל 65535 ובעזרתו ניתן לפנות ל 64K הכתובות של זיכרון הנתונים החיצוני.

גם הפקודות PUSH ו POP נעזרות במיעון זה ומסתמכות על מצביע המחסנית SP – Stack Pointer .

דוגמה: נניח שהתוכן של R0 הוא 70H . הפקודה MOV @R0,#55H אומרת להעביר את הנתון 55H אל כתובת מסוימת בזיכרון. לאיזו כתובת ? לכתובת שהתוכן של R0 מצביע עליה. היות ו R0 = 70H , יש להעביר את הנתון 55H לכתובת 70H .

הערה: הפקודה מבצעת את אותה הפעולה כמו זו שהייתה במיעון ישיר. מתי כדאי להשתמש בכל אחד מסוגי המיעון נראה בהמשך.

דוגמה נוספת : MOV A,@DPTR שבה אומרים להעביר אל האקומולטור נתון מהכתובת שה DPTR מצביע עליה.

4.4.1.3 Immediate Addressing - מיעון מיידי

בפקודה זו מעבירים קבוע אל היעד. הנתון נמצא בפקודה עצמה. לדוגמא : MOV A,#100 .

העבר אל האקומולטור את הנתון העשרוני 100 . ניתן היה לרשום גם MOV A,#64H או MOV A,#01100100B .

4.4.1.4 Register Addressing - מיעון רגיסטר

גישה ל אחד מ 8 רגיסטרי העבודה (R0 עד R7). כאמור , בזיכרון הנתונים הפנימי 4 בנקים, מבנק 0 ועד בנק 3 . בכל בנק יש 8 בתים, אוגרים, רגיסטרים המסופרים מ R0 עד R7 . הביטים RS1 RS0 ברגיסטר ה PSW קובעים עם איזה רגיסטרים עובדים. רגיסטרים נוספים למיעון זה הם האקומולטור (ACC), רגיסטר B , ה DPTR ודגל ה CY. ניתן לגשת לרגיסטרים אלו בעזרת פקודות מסוימות שתופסות פחות מקום בזיכרון.

דוגמה: **MOV R0,#90H**

הפקודה אומרת להעביר את הנתון 90H אל R0 . הפקודה תופסת 2 בתים בזיכרון.

הערה: נניח שעובדים עם בנק 0 ואז הפקודה (במיעון ישיר) MOV 0,#90H עושה בדיוק את אותה הפעולה אבל היא תופסת 3 בתים בזיכרון התוכנית.

4.4.1.5 Index Addressing מיעון אינדקס

מיעון זה הוא לקריאה מזיכרון התוכנית של טבלאות המרה - Look Up tables . טבלאות אלו יכולות להמיר ממעלות פרנהייט לצלסיוס או בין מידות של מטר לרגל , המרה בין מטבעות וכו. רגיסטר ה DPTR או ה PC מצביעים על בסיס הטבלה והאקומולטור מראה את ההיסט מהבסיס . דוגמא לפקודה זו היא : MOV A ,@A+DPTR . קיימת גם פקודת קפיצה במיעון זה והיא : JMP @A+PC המחברת את תוכן האקומולטור עם תוכן ה DPTR והסכום משמש לכתובת בזיכרון אליה עוברת התוכנית.

4.4.1.6. Relative Addressing מיעון יחסי

מיעון יחסי עוסק בקפיצות . קפיצה היא מעבר של התוכנית לכתובת בזיכרון שאיננה הכתובת העוקבת לזו שנמצאים בה עכשיו. בשיטה זו טווח הקפיצה הוא 127 כתובות קדימה או 128 כתובות אחורה.הקפיצות הקצרות sjmp והקפיצות המותנות הן קפיצות יחסיות.

4.4.1.7 Absolute Addressing - מיעון אבסולוטי

גם מיעון זה עוסק בקפיצה או בקריאה לפרוצדורה. טווח הקפיצה כאן הוא פלוס מינוס 2 קילו כתובות . הפקודה היא בת 2 בתים כאשר 5 ביטים הם קוד הפעולה ועוד 11 ביטים של הכתובת בתחום של 2 קילו שצריך לקפוץ אליה. הפקודה AJMP או ACALL הן קפיצות אבסולוטיות.

4.4.1.8 Long Addressing – מיעון רחוק

גם מיעון זה עוסק בקפיצה או קריאה לפרוצדורה. טווח הקפיצה הוא לכל כתובת במרחב הזיכרון של התוכנית כלומר במרחב 64 קילו כתובות. הפקודה בת 3 בתים. בית אחד הוא קוד הפעולה ועוד 2 בתים של כתובת. LJMP או LCALL הן פקודות במיעון רחוק.

4.4.2 חלוקת הפקודות לפי סוגים

חלוקה נוספת היא לפי אופי הפקודות. הפקודות מתחלקות ל 5 הקבוצות הבאות :

- פקודות העברת נתונים.
- פקודות אריתמטיות (חשבוניות) – חיבור חיסור כפל וחילוק.
- פקודות לוגיות – NOT AND OR XOR .
- פקודות על משתנים בוליאניים.
- פקודות מכונה והסתעפות.

4.5 סט ההוראות

סט - אוצר - ההוראות של המיקרו בקר ממשפחת ה 51 וה cip-51 מאפשר לפעול על בתים או ביטים בודדים.

ניעזר באיורים שבהמשך הפרק . אלו תקצירי הפקודות. "תקציר" הוא קיצור, והכוונה לקיצור אוצר הפקודות על ידי הענקת שמות משותפים לכתובות שונות או רגיסטרים שונים .

לדוגמה: אם נרצה להעביר את תוכן רגיסטר A אל אחת מ 128 הכתובות בזיכרון הנתונים הפנימי נצטרך 128 פקודות כמו :

```
MOV 0,A        ; העברת הנתון שבאקומולטור לכתובת 0 ב RAM הנתונים הפנימי
MOV 1,A        ; העברת הנתון שבאקומולטור לכתובת 1 ב RAM הנתונים הפנימי
⋮
MOV 127,A      ; העברת הנתון שבאקומולטור לכתובת 127 ב RAM הנתונים הפנימי
```

ניתן לקרא לכל 128 הכתובות בשם הכולל **direct** ואז במקום 128 פקודות שונות ניתן לקצר ולרשום : **MOV direct,A** .

יש לזכור שבמעבדי INTEL ושאר התואמים הפקודה בנויה כך :

OPCODE DESTINATIO , SOURCE

קוד פעולה יעד , מקור

כלומר קוד הפעולה (העברה או פעולה אריתמטית או לוגית) ואז קודם את היעד ואחר כך את המקור בניגוד למחשבה שלנו "שחושבים הפוך" . ישנם מעבדים שאצלם בפקודות מופיע קודם המקור ואחריו היעד.

שמות כלליים בתקציר ההוראות :

4.5.1 Rn

n מציין את אחד מ 8 הרגיסטרים R0 עד R7 של הבנק הנוכחי (לפי RS1 RS0 ברגיסטר PSW). הפקודה MOV A,Rn מחליפה 8 פקודות כמו :

```
MOV A,R0
MOV A,R1
   ⋮
MOV A,R7
```

4.5.2 Direct

128 כתובות בזיכרון הנתונים הפנימי + הכתובות ב SFR . לדוגמה : mov 50h,60h אומר להעביר את הנתון שיש בכתובת 60h אל כתובת 50h . התוכן הקודם של כתובת 50h יימחק והוא יקבל את הערך שיש ב 60h . כתוב 60h עצמה איננה משתנה. כעת יהיה בכתובת 50h ובכתובת 60h אותו ערך (זה שהיה ב 60h).

4.5.3 @Ri

זהו מיעון עקיף . i מציין את אחד מהרגיסטרים R0 או R1 בלבד. הסימן שטרודל @Ri אומר " כתובת עליה מצביע Ri" (R0 או R1). הכתובת היא באזור RAMהנתונים הפנימי. נניח שב R0 יש את הערך 60H . @R0 אומר בכתובת עליה מצביע R0 כלומר כתובת 60H באזור RAM הנתונים הפנימי. לדוגמה : mov @r0,#55H אומר להעביר לכתובת ש r0 מצביע עליה (כתובת 60H) את הנתון 55H . אחרי הפקודה יהיה בכתובת 60H ב RAM הנתונים הפנימי את הערך 55H .

4.5.4 #data

הסימן # אומר "מיידית את הנתון." data הוא נתון בן 8 סיביות מ 00000000 ועד 11111111 . לדוגמה : mov r5,#89H אומר להעביר ל r5 את הנתון 89 בהקסה דצימלי (8 ביטים בינאריים).

4.5.4 #data16

קבוע בן 16 סיביות. הוא מופיע בביית ה 2 והשלישי של פקודה . לדוגמא :mov dptr,#1239H אומר להעביר ל dptr את הקבוע 1239 בהקסה דצימלי (16 ביטים בינאריים).

4.5.5 bit

אחד הביטים שבתוך אזור ה RAM הפנימי שמחולק לביטים (כתובות 20H עד 2FH) וכמו כן לאזור הרגיסטרים SFR שגם כאן ניתן לפנות לכל ביט בנפרד. ניתן לשים בכל ביט ערך 0 או 1 כרצוננו וניתן לבצע קפיצה מותנית במצב הביט. לדוגמה : setb 5 אומר לשים בביט מספר 5 את הערך 1 . clr 7 אומר לשים 0 בביט מספר 7 . (2 הביטים נמצאים בכתובת 20H ב RAM הפנימי.

4.5.6 Add16

כתובת יעד בכל אחת מ 64K כתובות בזיכרון. לדוגמה : ljmp 9876H . הפקודה מקפיצה את התוכנית לכתובת 9876H שהיא כתובת בת 16 הביטים.

4.5.7 Add11

כתובת יעד במרחק של 2K כתובות מהכתובת הנוכחית. Add11 יבוא עם הפקודה acall add11 או ajmp add11 .

4.5.8 rel

כתובת יעד במרחק 127+ כתובות קדימה או 128- כתובות לאחור. הקפיצה היא יחסית למיקום שבו נמצאת הפקודה.

4.6 חלוקת ההוראות לקבוצות :

באיורים שבהמשך ישנם מספר קיצורים שבהם נשתמש. להלן הסבר הקיצורים :

src הוא קיצור של source – מקור , ו **dest** קיצור של destination - יעד.

המקור או היעד יכולים להיות כתובות באזור הנתונים הפנימי מכתובות 0 עד 127 והכתובות ב SFR (הכתובות מ 80H עד FFH).

direct - אחת מהכתובות באזור הנתונים הפנימי (מכתובות 0 עד 127) או מהכתובות ב SFR (הכתובות מ 80H עד FFH).

כאשר רושמים (direct) מתכוונים לתוכן של הכתובת , לדוגמה (70h) אומר תוכן כתובת 70h .

בדוגמאות שבהמשך נצא מההנחה :

א. A = 55H

ב. DPTR = 1000H ובכתובת זו נמצא הנתון 22H .

ג. בכתובת 70H בזיכרון הנתונים הפנימי נמצא הנתון 64H (100 עשרוני) .

ד. R5=99H

כמו כן זמן ביצוע פקודה חושב לפי גביש של 48MHz .

4.6.1 הוראות העברת נתונים

הטבלה הבאה מתארת תקציר של פקודות העברת הנתונים:

Data Transfer		בתים	מחזורי שעון
MOV A, Rn	Move Register to A	1	1
MOV A, direct	Move direct byte to A	2	2
MOV A, @Ri	Move indirect RAM to A	1	2
MOV A, #data	Move immediate to A	2	2
MOV Rn, A	Move A to Register	1	1
MOV Rn, direct	Move direct byte to Register	2	2
MOV Rn, #data	Move immediate to Register	2	2
MOV direct, A	Move A to direct byte	2	2
MOV direct, Rn	Move Register to direct byte	2	2
MOV direct, direct	Move direct byte to direct byte	3	3
MOV direct, @Ri	Move indirect RAM to direct byte	2	2
MOV direct, #data	Move immediate to direct byte	3	3
MOV @Ri, A	Move A to indirect RAM	1	2
MOV @Ri, direct	Move direct byte to indirect RAM	2	2
MOV @Ri, #data	Move immediate to indirect RAM	2	2
MOV DPTR, #data16	Load DPTR with 16-bit constant	3	3
MOVC A, @A+DPTR	Move code byte relative DPTR to A	1	3
MOVC A, @A+PC	Move code byte relative PC to A	1	3
MOVX A, @Ri	Move external data (8-bit address) to A	1	3
MOVX @Ri, A	Move A to external data (8-bit address)	1	3
MOVX A, @DPTR	Move external data (16-bit address) to A	1	3
MOVX @DPTR, A	Move A to external data (16-bit address)	1	3
PUSH direct	Push direct byte onto stack	2	2
POP direct	Pop direct byte from stack	2	2
XCH A, Rn	Exchange Register with A	1	1
XCH A, direct	Exchange direct byte with A	2	2
XCH A, @Ri	Exchange indirect RAM with A	1	2
XCHD A, @Ri	Exchange low nibble of indirect RAM with A	1	2

טבלה 9 : תקציר פקודות העברת נתונים

בטבלה הבאה יש הסברים לתקציר הפקודות : בעמודה קוד מנמוני הכוונה לאמצעי עזר, סמל או סימן רומז, המסייעים לזכור דברים.

הקוד המנמוני	הפעולה	הסבר	דוגמה	כמות מחזורי השעון
Mov A,Rn	(A) ← (Rn)	העבר ל A(לאקומולטור) את הנתון שב Rn	Mov A,R5 העבר ל A את הנתון שב R5 , כלומר : A=99H	1
MOV A, direct	(A) ← (direct)	העבר ל A את תוכן הכתובת direct	Mov a,70H תוכן A יהיה 64H	2

MOV A,@Ri	(A) ← (@Ri)	העבר ל A את הנתון שנמצא בכתובת עליה מצביע Ri .	Mov a,@r0 העבר ל A נתון שנמצא בכתובת עליה מצביע r0 . אם r0=40h אז מעבירים ל A את הנתון שנמצא בכתובת 40h .	2
Mov a,#data	(A) ← #data	העבר מיידית לאקומולטור את הנתון.	Mov a,#56H בסיום הפקודה : A=56H	2
Mov Rn,a	(Rn) ← (A)	העבר ל Rn את הנתון שבאקומולטור	Mov R6,A העבר ל R6 את הנתון שבאקומולטור. R6=55H	1
Mov Rn,direct	(Rn) ← (direct)	העבר את הנתון שבכתובת direct אל Rn	Mov R3,70H העבר את הנתון שבכתובת 70H ב RAM הנתונים הפנימי אל רגיסטר R3	2
Mov Rn,#data	(Rn) ← #data	העבר מיידית ל Rn את הנתון data	Mov r6,#78h העבר מיידית ל R6 את הנתון 78H . לאחר הפקודה R6=78H .	2
MOV direct,A	(direct) ← (A)	העבר את הנתון שב A אל הכתובת direct	Mov 60H,a העבר את הנתון שבאקומולטור לכתובת 60H ב direct . תוכן כתובת 60H יהיה 55H	2
Mov direct,Rn	(direct) ← (Rn)	העבר את הנתון שברגיסטר Rn אל כתובת direct ב RAM הפנימי	Mov 60H,R5 הנתון שברגיסטר R5 מועבר לכתובת 60H ב RAM הפנימי.	2
MOV direct,direct	(direct) ← (direct)	העבר את הנתון שבמקור אל היעד.	Mov 50H,70h העבר את הנתון שבכתובת 70h לכתובת 50h . בכתובת 50h יהיה 64h	3
Mov direct,@Ri	(direct) ← (@Ri)	העבר לכתובת direct את הנתון שנמצא בכתובת ש	Mov 50H,@R0 אם R0=70H אז מעבירים לכתובת 50H ב RAM הפנימי	2

	את הנתון שנמצא בכתובת 70H ב RAM הפנימי.	Ri מצביע עליה ב RAM הפנימי		
3	Mov 40H,#99H העבר מיידית את הנתון 99H לכתובת 40H ב RAM הפנימי.	העבר מיידית לכתובת direct ב RAM הפנימי את הנתון data	(direct)←#data	Mov direct,#data
2	Mov @r1,a העבר את הנתון שב A אל הכתובת עליה מצביע R1 . אם R1=50h אז מעבירים לכתובת 50h את הנתון 64h .	העבר את הנתון שב A אל הכתובת ש Ri מצביע עליה.	(@Ri) ← (A)	MOV @Ri,A
2	Mov @r0,70H העבר את הנתון שבכתובת 70H לכתובת ש R0 מצביע עליה	העבר את הנתון שבכתובת direct אל הכתובת ש Ri מצביע עליה	(@Ri)←(direct)	Mov @Ri,direct
2	Mov @r1,#55H העבר מיידית את הנתון 55H לכתובת עליה מצביע R0 .	העבר מיידית את הנתון לכתובת עליה מצביע Ri	(@Ri)←#data	Mov @Ri,#data
3	Mov dptr,#1000h העבר את הנתון 1000h אל ה DPTR .	העבר ל DPTR את הנתון בן 16 הביטים.	(DPTR)←#data 16	MOV DPTR,#data 16
3	אם dptr=1000h ו A=10H מביאים נתון מכתובת 1010h בזיכרון התוכנית אל A (שימושי ב look up tables).	העבר ל A נתון מזיכרון התוכנית מהכתובת שהיא הסכום של הנתונים ב A+DPTR	A ← ((A)+(DPTR))	MOVC A,@A+DPTR
3	אם PC=1000h ו A=90H מביאים נתון מכתובת 1090h בזיכרון התוכנית אל A (שימושי ב look up tables).	העבר ל A נתון מזיכרון התוכנית מהכתובת שהיא הסכום של הנתונים ב A+PC	A ← ((A)+(PC))	MOVC A,@A+PC
3	אם R0=50h אז מעבירים את הנתון בכתובת 50h בזיכרון ה XRAM אל A .	העבר ל A נתון מזיכרון הנתונים XRAM (שנמצא בתוך הרכיב)	A ← @Ri	MOVX A,@Ri

		מהכתובת שמצביע עליה הרגיסטרים R0 או R1.		
MOVX @Ri,A	@Ri ← A	הנתון שבאקומולטור עובר אל הכתובת עליה מצביע הרגיסטרים R0 או R1 בזיכרון ה XRAM שברכיב.		3
MOVX A,@DPTR	(A)← (@DPTR)	העבר ל A את הנתון מהכתובת ש DPTR מצביע עליה. הכתובת היא מזיכרון הנתונים החיצוני !	Movx a,@dptr העבר את הנתון מכתובת 1000h אל האקומולטור. A=22h .	3
MOVX @DPTR,A	(@DPTR) ←(A)	העבר את הנתון שבאקומולטור אל הכתובת שה DPTR מצביע עליה.	Movx @dptr,a העבר את הנתון שב Aאל כתובת 1000h בזיכרון הנתונים החיצוני. בכתובת 1000h יהיה 55h .	3
PUSH direct	(SP)=(SP)+1 (@SP) ← (direct)	דחוף אל המחסנית את הנתון שבכתובת src (כתובת ב direct).	Push 70h מצביע המחסנית SP מתקדם ב 1 ותוכן תא 70h מועבר אל הכתובת עליה מצביע ה SP. לדוגמא : אם sp=30h אז בכתובת 31h יהיה 64h .	2
POP direct	(direct)← (@SP) , (SP)=(SP)-1	משוך מהמחסנית נתון והעבר אותו ליעד (כתובת ב direct)	Pop 60h מעבירים את הנתון מהכתובת עליה מצביע SP אל כתובת 60h ב direct ו SP קטן ב 1.	2
XCH A, direct	(A) ↔(direct)	החלף בין הנתון שב A עם הנתון direct.	Xch a,70h הנתון שב A מתחלף עם הנתון שבכתובת 70h . אחרי ביצוע הפקודה A=64h ובכתובת 70h יהיה 55h .	1
XCH a,Rn	(A)↔ Rn	החלף בין הנתון שב A עם הנתון שב Rn	XCH A,R5	1

	אם הנתון ב R5=50H ו A=30H אז אחרי הפקודה : A=50h ו R5=30h .			
1	נניח R1=50h ובכתובת 50h יש את הנתון 22h ו A=10h. אחרי הפקודה יהיה בכתובת 50h את הנתון 10h ו A=22h .	החלף את הנתונים בין A והנתון בכתובת ש Ri מצביע עליה	(A) ↔ (@Ri)	XCH A,@Ri
1	xchg a,@r0 אם r0=70h אז בסיום הפקודה A=54h ובכתובת 70h יהיה 65h .	החלף בין 4 הביטים הנמוכים של A ו 4 הביטים הנמוכים בכתובת ש Ri מצביע עליה.	$(A_{3-0}) \leftrightarrow ((@Ri_{3-0}))$ A and @Ri exchange low nibble	XCHD A,@Ri

טבלה 10 : הוראות העברת נתונים

ניתן מספר תוכניות דוגמה בשפת אסמבלי. **בשפת אסמבלי אין הבדל בין אותיות גדולות וקטנות** . נקודה פסיק ; היא שורת הערה.

דוגמה 1: רשום תוכנית שממלאת בלוק כתובות באזור ה RAM הפנימי (DIRECT) מכתובת 60h ועד 6fh בנתון 64H .
הערה: היות ומדובר באזור ה RAM הפנימי ניתן להשתמש ברגיסטרים r0 או r1 כמצביעים. בדוגמה כאן השתמשנו ב r0 .
התוכנית :

```
        Mov a,#64h     ; 64h העבר לאקומולטור מיידית את הנתון
        Mov r0,#60h    ; העבר לרגיסטר r0 מיידית את הנתון 60 הקסה. הוא מצביע על תחילת כתובת בלוק הנתונים
        Mov r7,#10h    ; מונה התוכנית. מראה כמה פעמים לבצע את לולאת העברת הנתון שבהמשך.
Shoov: mov @r0,a       ; r0 העבר את הנתון שבאקומולטור אל הכתובת עליה מצביע הנתון שב
        Inc r0         ; r0 הגדל ב 1 את הנתון שברגיסטר
        Djnz r7,shoov  ; חסר אחד מרגיסטר r7 ואם הוא לא 0 קפוץ לכתובת shoov
        End            ; סיום התוכנית
```

דוגמה 2 : רשום תוכנית שתמלא בלוק כתובות באזור ה XRAM החל מכתובת 200h ועד 20fh בנתון ההולך וגדל מ 0 ועד 0fh .
הערה : היות ומדובר על זיכרון ה XRAM נשתמש ב DPTR כמצביע לבלוק זה.

```
Mov a,#0          ;  0 העבר לאקומולטור את הנתון
Mov dptr,#200h    ; XRAM משמש כמצביע על הכתובות באזור dptr .200h העבר ל dptr את הנתון
Mov r6,#10h       ; מונה התוכנית. מראה כמה פעמים יש לבצע את העברת הנתון
```

Again: movx @dptr,a ; dptr העבר את הנתון שבאקומולטור לכתובת עליה מצביע רגיסטר

Inc dptr ; הגדל ב 1 את ערך הנתון שב dptr . אם בהתחלה יש בו 200 הקסה אז אחרי הפקודה יהיה בו 201 הקסה

Inc a ; הגדל ב 1 את הערך שבאקומולטור

Djnz r6,again ; again (תווית) חסר 1 מרגיסטר r6 ואם הוא לא 0 קפוץ לכתובת

End

דוגמה 3 : יש לבצע החלפה בין הנתונים שבבלוק שיש ב RAM הפנימי שבדוגמה 1 ובין הנתונים ב XRAM שבדוגמה 2.

כאן נשתמש ברגיסטר r1 (במקום r0 בדוגמה 1) שיצביע על הבלוק ב RAM הנתונים הפנימי. ה dptr מצביע על אזור ה XRAM .

mov r1,#60h ; העבר ל r1 את הנתון 60 הקסה. הרגיסטר ישמש כמצביע לאזור הנתונים הפנימי

mov dptr,#200h ; העבר ל dptr את הנתון 200h . הרגיסטר ישמש מצביע לכתובות ב XRAM

mov r5,#10h ; מונה התוכנית

again: movx a,@dptr ; dptr העבר נתון לאקומולטור מהכתובת עליה מצביע הרגיסטר

mov b,a ; שמור את הנתון ברגיסטר b

mov a,@r1 ; העבר אל האקומולטור נתון מהכתובת עליה מצביע רגיסטר r1 באזור הפנימי

movx @dptr,a ;dptr את הנתון שהבאנו מהאזור הפנימי בעזרת r1 העבר אל הכתובת ב XRAM עליה מצביע

mov a,b ; a את הנתון שהבאנו מהאזור החיצוני ושמרנו ב b מחזירים ל

mov @r1,a ; מעבירים את הנתון שב a לכתובת בזיכרון הפנימי עליה מצביע r1

inc dptr ; הגדלה ב 1 של הנתון שב dptr כדי שיצביע על הכתובת הבאה

inc r1 ; הגדלה ב 1 את הנתון שב r1 כדי שיצביע על הכתובת הבאה

djnz r5,again ; again חסר 1 מהמונה ואם הוא לא 0 קפוץ לכתובת

end

4.6.2 הוראות אריתמטיות

הפקודות החשבוניות כוללות חיבור חיסור כפל וחילוק. כל הפעולות מתבצעות ביחידה האריתמטית לוגית – ALU. בחיבור וחיסור הדגלים שברגיסטר PSW מושפעים מהתוצאה.

4.6.2.1 הרגיסטר PSW – Program Status Word – מילת מצב תוכנית

הרגיסטר נקרא גם רגיסטר הדגלים והוא מתואר באיור הבא.

Bit	7	6	5	4	3	2	1	0
Name	CY	AC	F0	RS[1:0]		OV	F1	PARITY
Type	R/W	R/W	R/W	R/W		R/W	R/W	R
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xD0; SFR Page = All Pages; Bit-Addressable

איור 36 : מבנה רגיסטר PSW

נתאר את תפקיד הביטים מהביט הנמוך – LSB אל הגבוה MSB (מביט 0 עד ביט 7) .

Parity - ביט הזוגיות : מקבל 1 כאשר כמות ה '1' באקומולטור הוא **אי זוגי** - זוגיות הנקראת Even Parity - כלומר כמות ה 1 באקומולטור ועוד מצב הדגל נותנים כמות זוגית של אחדים. אחרת מקבל 0. השימוש העיקרי שלו הוא בתקשורת טורית. ביט זה הוא R (לקריאה ולא ניתן לכתיבה). אפשר לנסח את זה גם כך : הביט מקבל 1 כאשר הסכום של 8 הביטים של האקומולטור הוא אי זוגי . הוא מקבל 0 אם הסכום של 8 הביטים הוא זוגי.

F1 : ביט מספר 1 – ללא תפקיד מיוחד. ניתן לקרוא את ערכו ולכתוב אליו.

OV - OVerflow - דגל הגלישה - בספרים מגדירים אותו כך : הדגל מקבל 1 כאשר משתנה היעד אינו מספיק רחב כדי להכיל את התוצאה. ההגדרה "קצת" קשה להבנה. ננסה להסביר אותה . הדגל מקבל 1 כאשר יש גלישה במקרים הבאים :

א. בפעולות חיבור או חיסור יש גלישה דבר הגורם לשינוי בביט הסימן.

ב. בפעולת כפל יש גלישה והתוצאה גדולה מ 255 (גדולה מביית)

ג. בפעולת חלוקה יש גלישה כי הייתה חלוקה ב 0 .

דוגמה : אם נחבר את שני המספרים 01000000 + 01000001 התוצאה שנקבל היא 10000001 . חיברנו 2 מספרים חיוביים (בביט ה MSB של המספר יש 0) והתוצאה שהתקבלה שלילית (בביט ה MSB יש 1). הדגל יהיה 1 ויגיד למתכנת שיש "בעיה" – גלישה - בתוצאה. דבר דומה קיים גם אם נחבר את 2 המספרים השליליים 10000000 + 10000010 התוצאה המתקבלת היא חיובית : 00000010 . הדגל יהיה ב 1 ויגיד למתכנת שהייתה גלישה והתוצאה שגויה.

בכל שאר המקרים חוץ מ 3 המקרים שציינו בדגל יהיה 0 .

בפעולת איפוס RESET הדגלים מתאפסים.

RS1 RS0 - **R**egister bank **S**elect bits - 2 ביטים אלו קובעים עם איזה בנק עובדים (לפי איור 8 הבא).

00: Bank 0, Addresses 0x00-0x07
01: Bank 1, Addresses 0x08-0x0F
10: Bank 2, Addresses 0x10-0x17
11: Bank 3, Addresses 0x18-0x1F

איור 37 : בחירת הבנק בעזרת הביטים RS1 RS0

F0 - Flag 0 - ללא תפקיד . ניתן לשימוש כללי.

AC - Auxiliary Carry Flag – דגל נשא עזר - כאשר בפעולת חיבור יש Carry – נשא - מביט D3 אל ביט D4 או בפעולת חיסור יש borrow – השאלה – מביט 4 אל ביט 3 . לביט זה תפקיד בפעולות עם מספרים המיוצגים ב BCD .

CY - CarrY Flag – דגל נשא - ביט המשמש בפעולות חשבוניות ופעולות הזזה. הביט מקבל 1 אם בפעולה חשבונית היה נשא מביט ה MSB החוצה או בפעולת חיסור יש השאלה מבחוץ אל ביט ה MSB . בשאר המקרים בביט יהיה 0 .

דוגמה : נניח שמחברים 28H+29H . התשובה תהיה : 51H . מצב הדגלים יהיה : **CY**=0, **P**=1, **AC**=1, **OV**=0

דוגמא נוספת : 11010111+10001000 . התשובה : 01011111 . מצב הדגלים יהיה : **CY**=1, **P**=0, **AC**=0 , **OV**=1

הטבלה הבאה היא תקציר ההוראות האריתמטיות :

הקוד המנמוני	תיאור הפקודה	בתים	מחזורי שעון
Mnemonic	**Description**	**Bytes**	**Clock Cycles**
Arithmetic Operations			
ADD A, Rn	Add register to A	1	1
ADD A, direct	Add direct byte to A	2	2
ADD A, @Ri	Add indirect RAM to A	1	2
ADD A, #data	Add immediate to A	2	2
ADDC A, Rn	Add register to A with carry	1	1
ADDC A, direct	Add direct byte to A with carry	2	2
ADDC A, @Ri	Add indirect RAM to A with carry	1	2
ADDC A, #data	Add immediate to A with carry	2	2
SUBB A, Rn	Subtract register from A with borrow	1	1
SUBB A, direct	Subtract direct byte from A with borrow	2	2
SUBB A, @Ri	Subtract indirect RAM from A with borrow	1	2
SUBB A, #data	Subtract immediate from A with borrow	2	2
INC A	Increment A	1	1
INC Rn	Increment register	1	1
INC direct	Increment direct byte	2	2
INC @Ri	Increment indirect RAM	1	2
DEC A	Decrement A	1	1
DEC Rn	Decrement register	1	1
DEC direct	Decrement direct byte	2	2
DEC @Ri	Decrement indirect RAM	1	2
INC DPTR	Increment Data Pointer	1	1
MUL AB	Multiply A and B	1	4
DIV AB	Divide A by B	1	8
DA A	Decimal adjust A	1	1

טבלה 11 : תקציר פקודות אריתמטיות (חשבוניות)

הטבלה הבאה היא הסבר של תקציר הפקודות :

הקוד המנמוני	**הפעולה**	**הסבר**	**דוגמה**	**מחזורי שעון**
add a,Rn	(A)←(A)+(Rn)	חבר את הנתון שבאקומולטור עם הנתון שב Rn . התוצאה מועברת	Add a,R5	1

		לאקומולטור. הדגלים מושפעים.		
add a,direct	(A)←(A)+(direct)	חבר את הנתון שבאקומולטור עם הנתון שבכתובת direct . התוצאה תהיה באקומולטור והדגלים מושפעים.	Add a,50H חבר את תוכן האקומולטור עם הנתון שנמצא בכתובת 50H ב RAM הנתונים הפנימי. התוצאה עוברת לאקומולטור. הדגלים מושפעים.	2
add a,@Ri	(A)←(A)+(@Ri)	חבר את הנתון שבאקומולטור עם הנתון בכתובת שרגיסטר Ri מצביע עליו. התוצאה תהיה באקומולטור והדגלים מושפעים.	Add a,@r1 נניח ש R1=60H . מתבצעת פעולת חיבור בין הנתון שבאקומולטור עם הנתון בכתובת 60H (הכתובת ש R1 מצביע עליה). התוצאה באקומולטור והדגלים מושפעים.	2
add a,#data	(A)←(A)+data+C	חבר את הנתון שבאקומולטור מידית עם הנתון שבפקודה. התוצאה תהיה באקומולטור והדגלים מושפעים.	Add a,#56h חבר את הנתון שבאקומולטור עם הנתון 56H . התוצאה תהיה באקומולטור והדגלים מושפעים.	2
addc a,Rn	(A)←(A)+(Rn)+C	חבר את הנתון שבאקומולטור עם הנתון שב Rn ועם דגל הנשא . התוצאה מועברת לאקומולטור. הדגלים מושפעים.	Addc a,R5 חבר את הנתון שבאקומולטור עם הנתון שב R5 +דגל הנשא . התוצאה באקומלטור והדגלים מושפעים	1
addc a,direct	(A)←(A)+(direct)+C	חבר את הנתון שבאקומולטור עם הנתון שבכתובת direct ועם דגל הנשא. התוצאה תהיה באקומולטור והדגלים מושפעים.	Addc a,50H חבר את תוכן האקומולטור עם הנתון שנמצא בכתובת 50H ב RAM הנתונים הפנימי + דגל הנשא. התוצאה עוברת לאקומולטור. הדגלים מושפעים.	2

addc a,@Ri	(A)←(A)+(@Ri)+C	חבר את הנתון שבאקומולטור עם הנתון בכתובת שרגיסטר Ri מצביע עליו ועם דגל הנשא. התוצאה תהיה באקומולטור והדגלים מושפעים.	Addc a,@r1 נניח ש R1=60H . מתבצעת פעולת חיבור בין הנתון שבאקומולטור עם הנתון בכתובת 60H (הכתובת ש R1 מצביע עליה)+ דגל הנשא. התוצאה באקומולטור והדגלים מושפעים.	2
addc a,#data	(A)←(A)+data+C	חבר את הנתון שבאקומולטור מידית עם הנתון שבפקודה ועם דגל הנשא. התוצאה תהיה באקומולטור והדגלים מושפעים.	Addc a,#56h חבר את הנתון שבאקומולטור עם הנתון 56H+ דגל הנשא. התוצאה תהיה באקומולטור והדגלים מושפעים.	2
subb a,Rn	(A)←(A)-(Rn)-C	חסר את הנתון שבאקומולטור עם הנתון שב Rn וחסר גם את דגל הנשא. התוצאה מועברת לאקומולטור. הדגלים מושפעים.	subb a,R5 חסר את הנתון שבאקומולטור עם הנתון שב R5 +דגל הנשא . התוצאה באקומלטור והדגלים מושפעים	1
subb a,direct	(A)←(A)-(direct)-C	חסר את הנתון שבאקומולטור עם הנתון שבכתובת direct וחסר גם את דגל הנשא . התוצאה תהיה באקומולטור והדגלים מושפעים.	subb a,50H חסר את תוכן האקומולטור עם הנתון שנמצא בכתובת 50H ב RAM הנתונים הפנימי + דגל הנשא. התוצאה עוברת לאקומולטור. הדגלים מושפעים.	2
subb a,@Ri	(A)←(A)-(@Ri)-C	חסר את הנתון שבאקומולטור עם הנתון בכתובת שרגיסטר Ri מצביע עליו וחסר גם את דגל הנשא. התוצאה תהיה באקומולטור והדגלים מושפעים.	subb a,@r1 נניח ש R1=60H . מתבצעת פעולת חיסור בין הנתון שבאקומולטור עם הנתון בכתובת 60H (הכתובת ש R1 מצביע עליה) מינוס דגל הנשא. התוצאה באקומולטור והדגלים מושפעים.	2
subb a,#data	(A)←(A)-data-C	חסר את הנתון שבאקומולטור מידית עם	subb a,#56h	2

		הנתון שבפקודה וחסר גם את דגל הנשא. התוצאה תהיה באקומולטור והדגלים מושפעים.	חסר את הנתון שבאקומולטור עם הנתון 56H וחסר גם את דגל הנשא. התוצאה תהיה באקומולטור והדגלים מושפעים.	
inc a	(A)←(A)+1	הגדל ב1 את תוכן האקומולטור	Inc a אם הנתון באקומולטור לפני הפקודה הוא 10 אז אחרי הפקודה יהיה באקומולטור את הנתון 11	1
inc Rn	(Rn) ← (Rn)+1	הגדל ב 1 את תוכן Rn	Inc r5 אם ב R5 היה לפני הפקודה את הנתון 10 אז לאחר הפקודה יהיה בו 11	1
inc direct	(direct)←(direct)+1	הגדל את הנתון שבכתובת direct ב 1	Inc 60H הגדל את הנתון שבכתובת 60H ב 1 .	2
inc @Ri	(@Ri)←(@Ri)+1	הגדל את הנתון בכתובת ש Ri מצביע עליה ב 1	Inc @R1 נניח R1=70H . מגדילים ב 1 את הנתון שבכתובת 70H .	2
dec a	(A) ← (A) – 1	הקטן ב 1 את תוכן האקומולטור	dec a אם הנתון באקומולטור לפני הפקודה הוא 10 אז אחרי הפקודה יהיה באקומולטור את הנתון 9	1
dec Rn	(Rn) ← (Rn)-1	חסר 1 מהנתון שב Rn	dec r5 אם ב R5 היה לפני הפקודה את הנתון 10 אז לאחר הפקודה יהיה בו 9	1
dec direct	(direct)←(direct)-1	הקטן את הנתון שבכתובת direct ב 1	dec 60H הקטן את הנתון שבכתובת 60H ב 1 .	2
dec @Ri	(@Ri)←(@Ri)-1	הקטן את הנתון בכתובת ש Ri מצביע עליה ב 1	dec @R1 נניח R1=70H . מקטינים ב 1 את הנתון שבכתובת 70H .	2
inc dptr	(dptr)←(dptr)+1	הגדל ב1 את הנתון ברגיסטר ה dptr	Inc dptr אם הנתון ב dptr היה 1234h אז אחרי הפקודה יהיה בו 1235h .	2
mul ab	(A)7-0← (A) X (B) (B)15-8	בצע כפל בין הנתון שבאקומולטור לנתון	Mul ab	4

		שברגיסטר B . 8 הביטים הנמוכים של התוצאה נכנסים לאקומולטור ו 8 הביטים הגבוהים של תוצאת הכפל ל B .	1. אם A=5 ו B=4 אז אחרי הפקודה A=20=14H ו B=0 . 2. אם A =16 וגם B=16 אז מכפלתם נותנת 256. לכן A=00000000 ו B=00000001	
div ab	(A)15-8 ↖ (B)7-0 ← (A) / (B)	חלק את הנתון שב A עם הנתון שב B . התוצאה נכנסת ל A והשארית ל B .	Div ab א. אם A=10 ו B=7 אז אחרי הפקודה: התוצאה A=1 והשארית B=3 . ב. אם A=100 ו B=11 אז אחרי החלוקה : A=9 ו B=1 .	8
da a	If (A3~A0) > 9 or AC=1 then A=A+6. If (A7~A4)>9 or CF=1 then A=A+60H and CF=1	בצע תיקון עשרוני לאקומולטור. השימוש בפקודה הוא כשעובדים עם מספרים המיוצגים ב BCD מצופף (packed BCD). כל 4 ביטים ספרה אחת. באקומולטור מופיעות 2 ספרות ב BCD . כל ספרה בין 0000 עד 1001 ..הספרות A עד F לא מופיעות ב BCD.	DA A	1

טבלה 12 : הסבר הוראות אריתמטיות

4.6.3 הוראות לוגיות

הפקודות הלוגיות שמשמשות אותנו הן AND OR XOR . ישנן 6 פקודות לכל פעולה לוגית. הפקודות דומות ומספיק להסביר את הפקודות AND . פעולת OR או XOR מתבצעות בצורה דומה.

Logical Operations			
ANL A, Rn	AND Register to A	1	1
ANL A, direct	AND direct byte to A	2	2
ANL A, @Ri	AND indirect RAM to A	1	2
ANL A, #data	AND immediate to A	2	2
ANL direct, A	AND A to direct byte	2	2
ANL direct, #data	AND immediate to direct byte	3	3
ORL A, Rn	OR Register to A	1	1
ORL A, direct	OR direct byte to A	2	2
ORL A, @Ri	OR indirect RAM to A	1	2
ORL A, #data	OR immediate to A	2	2
ORL direct, A	OR A to direct byte	2	2
ORL direct, #data	OR immediate to direct byte	3	3
XRL A, Rn	Exclusive-OR Register to A	1	1
XRL A, direct	Exclusive-OR direct byte to A	2	2
XRL A, @Ri	Exclusive-OR indirect RAM to A	1	2
XRL A, #data	Exclusive-OR immediate to A	2	2
XRL direct, A	Exclusive-OR A to direct byte	2	2
XRL direct, #data	Exclusive-OR immediate to direct byte	3	3
CLR A	Clear A	1	1
CPL A	Complement A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through Carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through Carry	1	1
SWAP A	Swap nibbles of A	1	1

טבלה 13 : תקציר פקודות לוגיות.

הטבלה הבאה היא הסבר לתקציר הפקודות:

הקוד המנמוני	הפעולה	הסבר	דוגמה	מחזורי שעון
anl a,Rn	(A) ← (A)and(Rn)	פעולת AND לוגי בין התוכן באקומולטור ותוכן רגיסטר Rn (בין הביטים בהתאמה). התוצאה תהיה באקומולטור	Anl a,r5 אם לפני הפקודה A=66h ו R5=aah אז 01100110 and 10101010 A = 00100010 אחרי הפקודה יהיה באקומולטור 22h .	1
anl a,direct	(A)←(a)and(direct)	פעולת AND לוגי בין תוכן האקומולטור ותוכן אחת הכתובות באזור ה	Anl a,50h	2

		direct . התוצאה תעבור לאקומולטור.	אם לפני הפקודה a=55h ותוכן תא 50h הוא 5bh אז התוצאה שהיא 51h עוברת לאקומולטור	
anl a,@Ri	(A)←(a) and((Ri))	פעולת AND לוגי בין תוכן האקומולטור ותוכן הכתובת שרגיסטר Ri מצביע עליה. התוצאה האקומולטור.	Anl a,@r0 אם לפני הפקודה a=55h ותוכן תא 50h הוא 5bh ו R0=50h אז מתבצעת פעולת AND בין A ותוכן כתובת 50h . התוצאה שהיא 51h עוברת לאקומולטור.	2
anl a,#data	(A)←(a) and #data	פעולת and לוגי בין תוכן האקומולטור והנתון שבפקודה. התוצאה באקומולטור	Anl a,#6fh אם לפני הפקודה a=55h אז אחרי הפקודה a=45h	2
anl direct,a	(direct)←(direct)and(A)	פעולת AND לוגי בין התוכן של כתובת direct ותוכן האקומולטור. התוצאה בכתובת direct	Anl 50h,a אם לפני הפקודה a=55h ותוכן תא 50h הוא 5bh אז התוצאה שהיא 51h עוברת לכתובת 50h .	2
anl direct,#data	(direct)←(direct)and #data	פעולת and לוגי בין תוכן כתובת directוהנתון שבפקודה. התוצאה ב direct	Anl 50h,#6fh אם לפני הפקודה תוכן כתובת 50h הוא 55h אז אחרי הפקודה בכתובת 50h יהיה 45h .	3
orl a,Rn	(A) ← (A)or(Rn)	פעולת OR לוגי בין התוכן באקומולטור ותוכן רגיסטר Rn (בין הביטים בהתאמה). התוצאה תהיה באקומולטור	Orl a,r5 Anl a,r5 אם לפני הפקודה A=66h ו R5=aah אז 01100110 or 10101010 A = 11101110 אחרי הפקודה יהיה באקומולטור eeh .	1
orl a,direct	(A)←(a)or(direct)	פעולת OR לוגי בין תוכן האקומולטור ותוכן אחת	Orl a,50h	2

		הכתובות באזור ה direct . התוצאה תעבור לאקומולטור.	אם לפני הפקודה a=55h ותוכן תא 50h הוא 5bh אז התוצאה שהיא 5fh עוברת לאקומולטור	
orl a,@Ri	(A)←(a) or((Ri))	פעולת OR לוגי בין תוכן האקומולטור ותוכן הכתובת שרגיסטר Ri מצביע עליה. התוצאה האקומולטור.	Orl a,@r0 אם לפני הפקודה a=55h ותוכן תא 50h הוא 5bh ו R0=50h אז מתבצעת פעולת OR בין האקומולטור ותוכן כתובת 50h . התוצאה שהיא 5fh עוברת לאקומולטור	2
orl a,#data	(A)←(a) or #data	פעולת OR לוגי בין תוכן האקומולטור והנתון שבפקודה. התוצאה באקומולטור	Orl a,#6fh אם לפני הפקודה a=55h אז אחרי הפקודה a=7fh	2
orl direct,a	(direct)←(direct)or(A)	פעולת OR לוגי בין התוכן של כתובת direct ותוכן האקומולטור. התוצאה בכתובת direct	Orl a,50h אם לפני הפקודה a=55h ותוכן תא 50h הוא 6dh אז התוצאה שהיא 6dh עוברת לכתובת 50h .	2
orl direct,#data	(direct)←(direct)or #data	פעולת OR לוגי בין תוכן כתובת directוהנתון שבפקודה. התוצאה ב direct	Orl 50h,#6fh אם לפני הפקודה תוכן כתובת 50h הוא 55h אז אחרי הפקודה בכתובת 50h יהיה 7fh .	3
xrl a,Rn	(A) ← (A)xor(Rn)	פעולת XOR לוגי בין תוכן האקומולטור ותוכן רגיסטר Rn (בין הביטים בהתאמה). התוצאה תהיה באקומולטור	Xrl a,r5 אם לפני הפקודה A=66h ו R5=aah אז 01100110 xor 10101010 A =11001100 אחרי הפקודה יהיה באקומולטור cch .	1
xrl a,direct	(A)←(a)xor(direct)	פעולת XOR לוגי בין תוכן האקומולטור ותוכן	Xrl a,50h	2

		אחת הכתובות באזור ה direct . התוצאה תעבור לאקומולטור.	אם לפני הפקודה a=55h ותוכן תא 50h הוא 5bh אז התוצאה שהיא 0eh עוברת לאקומולטור	
xrl a,@Ri	(A)←(a)xor((Ri))	פעולת XOR לוגי בין תוכן האקומולטור ותוכן הכתובת שרגיסטר Ri מצביע עליה. התוצאה האקומולטור.	Xrl a,@r0 אם לפני הפקודה a=55h ותוכן תא 50h הוא 5bh ו R0=50h אז מתבצעת פעולת XOR בין A ותוכן כתובת 50h . התוצאה שהיא 5fh עוברת לאקומולטור	2
xrl a,#data	(A)←(a) xor #data	פעולת XOR לוגי בין תוכן A והנתון שבפקודה. התוצאה באקומולטור	Xrl a,#6fh אם לפני הפקודה a=55h אז אחרי הפקודה a=3ah	2
xrl direct,a	(direct)←(direct)xor(A)	פעולת XOR לוגי בין התוכן של כתובת direct ותוכן A. התוצאה בכתובת direct	Xrl 50H,A אם לפני הפקודה a=55h ותוכן תא 50h הוא 6dh אז התוצאה שהיא 38h עוברת לכתובת 50h .	2
xrl direct,#data	(direct)←(direct)xor #data	פעולת XOR לוגי בין תוכן כתובת directוהנתון שבפקודה. התוצאה ב direct	Xrl 50h,#6fh אם לפני הפקודה תוכן כתובת 50h הוא 55h אז אחרי הפקודה בכתובת 50h יהיה 38h .	3
Clr a	(A)← 0	נקה (אפס) את כל הסיביות באקומולטור	Clr a אם תוכן האקומולטור היה כלשהו אז אחרי הפקודה A=00000000	1
Cpl a	(A) = (A')	בצע השלמה ל 1 של כל אחד מהביטים באקומולטור. (הפוך כל ביט)	Cpl a אם לפני הפקודה A=01101010 אז אחריה: A=10010101	1
RL A	(An+1)←(An) n=0–6 , (AO)← (A7)	סובב את הנתון באקומולטור פעם אחת שמאלה. כל ביט זז שמאלה פעם אחת וביט A7 נכנס ל A0	RL A אם לפני הפקודה A=11000011 אז אחרי הפקודה A=10000111	1

RLC A	(An+1)←(An) n=0–6 (AO)← (CY) (CY)←(A7)	סובב את הנתון שבאקומולטור שמאלה דרך דגל ה CY (נשא)	RLC A אם לפני הפקודה CY=0 ו A=11000011 אז אחרי הפקודה : CY=1 A=10000110	1
RR A	(An)←(An+1) n=0–6 (A7)← (A0)	סובב את הנתון באקומולטור פעם אחת ימינה. כל ביט זז ימינה פעם אחת וביט A0 נכנס ל A7	RL A אם לפני הפקודה A=11000011 אז אחרי הפקודה A=11100001	1
RRC A	(An)←(An+1) n=0–6 (A7)← (CY) (CY)←(A0)	סובב את הנתון שבאקומולטור ימינה דרך דגל ה CY (נשא). כל ביט זז ימינה פעם אחת. ביט CY נכנס ל A7 וביט A0 נכנס ל CY .	RLC A אם לפני הפקודה CY=0 ו A=11000011 אז אחרי הפקודה : CY=1 A=01100001	1
Swap a	(A3-0) ↔ (A7-4)	מחליפים את 4 הביטים הנמוכים של האקומולטור עם 4 הביטים הגבוהים (החלפה בין הניבל הגבוה והנמוך).	Swap a אם A=18H אז אחרי הפקודה : A=81H	1

טבלה 14 : הסבר הפקודות הלוגיות

תרגיל דוגמה: בהנחה שבפורט 1 יש 8 לדים . רשום תוכנית שתריץ אור בלדים אחד אחרי השני. בהתחלה תידלק הלד שב P1.0 אחרי זמן מסוים תידלק הלד ב P1.1 והלד שדלקה ב P1.0 נכבית. אחרי זמן מסוים נדלקת הלד ב P1.2 בלבד וכך הלאה. אחרי זמן מסוים שהלד ב P1.7 דולקת היא תגבה והלד שב P1.0 דולקת שוב וכך הלאה בלולאה אין סופית.

הערה : נניח שהלד דולקת כאשר בהדק הפורט יש 1 . כמו כן נניח שהוגדרו ההדקים כפלט עם push pull והקרוסבר פעיל.

פתרון :

```
        mov a,#1      ; P1.0 העבר אל האקומולטור מיידית את הנתון 1 . 00000001B ערך זה מדליק רק את הלד שב
aga:    mov p1,a      ; P1 העבר את הנתון שבאקומולטור אל
        lcall delay   ; .קריאה לתוכנית השהייה כדי שהעין תבחין שהלד המסוימת דולקת ושאר הלדים כבויות
        rl a          ; . סיבוב האקומולטור שמאלה
        sjmp aga      ; aga קפוץ לכתובת
```

```
; ----- תוכנית השהייה של לולאה בתוך לולאה בתוך לולאה. ------
delay:  mov r5,#0
del0:   mov r4,#0
del1:   mov r3,#0
        djnz r3,$          ; djnz r3,$ חסר 1 מרגיסטר r3 ואם הוא לא 0 קפוץ לכתובת של הפקודה הנוכחית
        djnz r4,del1
        djnz r5,del0
        ret
        end
```

4.6.4 הוראות הפעולות עם משתנים בוליאניים

אחד מיתרונותיו של המיקרו בקר הוא האפשרות לפעול על ביט בודד. רוב המיקרו מעבדים אינם יכולים לפעול ישירות על ביט בודד. ב RAM הנתונים הפנימי בין הכתובות 20H ועד 2FH יש 16 בתים המחולקים לביטים. בכל כתובת 8 ביטים ולכן יש לנו 8*16=128 ביטים, מביט מספר 0 ועד ביט 7fH (127). ניתן לפנות לכל כתובת באזור זה כביית או לגשת לביט בודד באחת מכתובות אלו ולקבוע האם יהיה בו 0 או 1 . כמו כן ניתן לקבוע את מצב דגל הנשא – CY ולעשות עליו פעולות AND או OR עם אחד הביטים 0 עד 127 .

גם אזור הרגיסטרים המיוחדים ה SFR מחולק לביטים. לא כל הרגיסטרים באזור זה מחולקים לביטים אלא רק הרגיסטרים הנמצאים בכתובת המתחלקת ב 8 ללא שארית (שארית 0). לדוגמא P1 נמצא באזור זה בכתובת 90H . הביטים של הפורט מתחברים להדקים של הרכיב (הדקים 1 עד 8 במבנה DIP). הביטים של הפורט הם בעלי הכתובות מ 90h ל P1.0 ועד 97h ל P1.7 . בפקודות בפרק זה ניתן לפנות גם לביטים של פורט 1 ולכל שאר הביטים באזור הרגיסטרים המיוחדים.

Boolean Manipulation	תיאור הפקודה	בתים	מחזורי שעון
CLR C	Clear Carry	1	1
CLR bit	Clear direct bit	2	2
SETB C	Set Carry	1	1
SETB bit	Set direct bit	2	2
CPL C	Complement Carry	1	1
CPL bit	Complement direct bit	2	2
ANL C, bit	AND direct bit to Carry	2	2
ANL C, /bit	AND complement of direct bit to Carry	2	2
ORL C, bit	OR direct bit to carry	2	2
ORL C, /bit	OR complement of direct bit to Carry	2	2
MOV C, bit	Move direct bit to Carry	2	2
MOV bit, C	Move Carry to direct bit	2	2

טבלה 15 : תקציר פקודות הפועלות על ביטים

הטבלה הבאה היא הסבר לתקציר הפקודות הפועלות על משתנים בוליאניים.

מחזורי שעון	דוגמה	הסבר	הפעולה	הקוד המנמוני
1	CLR C לא משנה מה היה הערך בדגל ה CY. אחרי הפקודה יש בו 0 .	אפס את דגל הנשא - CY	(CY) ← 0	CLR C
2	1. Clr 70h אפס את ביט 70h . 2. Clr P1.0 שים 0 בהדק P1.0	אפס את הביט	(bit) ← 0	CLR bit
1	SETB C אחרי הפקודה (CY) = 1	שים 1 בביט הנשא CY	(CY) ← 1	SETB C
2	1. Setb 78h שים 1 בביט 78h . SETB P1.5 שים 1 בהדק P1.5 .	שים 1 בביט	(bit) ← 1	SETB bit
1	CPL C אם דגל CY לפני הפקודה היה 1 אז אחרי הפקודה יהיה בו 0 .	השלם ל 1 (הפוך) את מצב דגל CY	(CY) ← (CY)'	CPL C
2	1. CPL 60h הפוך את מצב ביט מספר 60h . אם היה בו 0 אז אחרי הפקודה יהיה בו 1 ולהפך. 2. CPL P1.1 הפוך את מצב הדק P1.1.	השלם ל 1 (הפוך) את הערך בביט	(bit) ← (bit)'	CPL bit
2	ANL C,74h פעולת AND לוגי בין דגל CY ובין הערך בביט 74h . התוצאה תעבור ל CY	פעולת AND בין ערך דגל ה CY והערך בביט. התוצאה תעבור ל CY	(CY)←(CY) and(bit)	ANL C,bit
2	ANL C,/70h נניח CY=1 וביט 70h=0. הפקודה מבצעת AND בין ה 1 שב CY ובין המצב ההפוך לזה שבביט 70h (ההפך מ 0 כלומר 1). התוצאה – 1- תהיה ב CY	פעולת AND בין דגל CY והמצב ההפוך בביט. התוצאה ב CY.	(CY)← (CY)and(bit)'	ANL C,/bit

2	ORL C,74h פעולת OR לוגי בין דגל CY ובין הערך בביט 74h . התוצאה תעבור ל CY	פעולת OR בין ערך דגל ה CY והערך בביט. התוצאה תעבור ל CY	(CY)←(CY) or(bit)	ORL C,bit
2	ORL C,/70h נניח CY=0 וביט 70h=0 . הפקודה מבצעת פעולת OR בין ה 0 שב CY ובין המצב ההפוך לזה שבביט 70h (ההפך מ 0 כלומר 1). התוצאה – 1- תהיה ב CY	פעולת OR בין דגל CY והמצב ההפוך בביט. התוצאה ב CY.	(CY)← (CY)or(bit)'	ORL C,/bit
2	MOV C,60h אם בביט 60h יש 0 אז אחרי הפקודה CY=0	העבר את הערך שבביט אל ה CY.	(CY) ← (bit)	MOV C,bit
	MOV 60H,C אם ב CY יש 0 אז אחרי הפקודה יהיה בביט 60h יהיה 0.	העבר את הערך שב CY אל הביט.	(bit) ← (CY)	MOV bit,C

טבלה 16 : פקודות על משתנים בוליאניים

תרגיל דוגמא: נתון קטע התוכנית הבא . הנח שהוגדרו חיבורי הקרוסבר המתאימים.

```
Again :  Cpl P1.0
         Sjmp again  // again   קפוץ לכתובת
```

מהו הגל המופק בהדק P1.0 אם ידוע שתדר הגביש של המיקרו הוא 6MHz ?

פתרון

הפקודה cpl P1.0 הופכת את מצב ביט P1.0 והפקודה sjmp (Short JuMP) מקפיצה את התוכנית חזרה לכתובת again ומכאן שהתוכנית מבצעת לולאה אין סופית שבה הופכים כל פעם את מצב הביט אז בהדק P1.0 מקבלים 0 ו 1 לסירוגין. 0 מייצג מתח של כ 0 וולט ו 1 מייצג מתח של כ 3 וולט. מכאן שמקבלים גל מרובע בין 0 ל 3 וולט בתדר שנקבע על ידי מהירות העבודה של המיקרו (שנקבעת על ידי תדירות הגביש).

נחשב זמנים . הפקודה cpl P1.0 תופסת מחזור 2 מחזורי שעון והפקודה sjmp תופסת 4 מחזורי שעון (נלמד קפיצות בפרק הבא) אז ביחד נקבל 6 מחזורי שעון.

מחזור שעון הוא : $1/(6*10^6)$. נכפיל 6 מחזורי שעון בזמן מחזור שעון ונקבל : $6*1/(6*10^6)=1*10^{-6}$ שהם 1 מיקרו שניות. מכאן שזמן של כל מצב בהדק P1.0 הוא 1 מיקרו שנייה.

זמן המחזור מורכב מזמן שבו הגל ב 0 ועוד הזמן שב 1 . סה"כ 2 מיקרו שניות.

מכאן שתדר הגל המרובע בהדק P1.0 יהיה : $f = 1/ T = 1/2*10^{-6}=500KHz$.

4.6.5 פקודות בקרת תכנית ומכונה

פקודות אלו כוללות קפיצות לא מותנות, קפיצות מותנות (לפי תנאי - לדוגמא "קפוץ אם באקומולטור יש 0" או "קפוץ אם יש 1 בדגל ה CY" וכו'), קריאה לפרוצדורות וחזרה ממנה, חזרה מפסיקות וכו'.

4.6.5.1 קפיצות במיקרו בקר

קפיצה היא מעבר של התוכנית לפקודה שאיננה נמצאת בכתובת העוקבת לפקודה הנוכחית. לדוגמא התוכנית נמצאת בכתובת 200h ורוצים לקפוץ לכתובת 300h . הקפיצה מתבצעת על ידי טעינה של הכתובת החדשה למונה התוכנית – PC .

ישנן 3 סוגי קפיצה : **1. קצרה 2. אבסולוטית 3. ארוכה.** ההבדל בין הקפיצות הוא טווח הקפיצה.

4.6.5.1.1 <u>קפיצה קצרה</u> – sjmp rel

הקפיצה היא יחסית **rel**ative . טווח הקפיצה הוא 127 כתובות קדימה או 128 כתובות אחורה, יחסית למקום שבו נמצאים. הפקודה תופסת 2 בתים בזיכרון. הביית הראשון הוא קוד הפעולה של הפקודה והבית השני הוא הטווח או ההיסט (offset) . אם בבית השני, בסיבית ה MSB, יש 0 אז הקפיצה קדימה ושאר 7 הביטים מראים את הטווח. אם בסיבית ה MSB יש 1 אז הקפיצה אחורה ו 7 הביטים האחרים מראים (בשיטת המשלים ל 2) את הטווח אחורה.
הקפיצה נקראת יחסית כי הקפיצה היא לא לכתובת מוחלטת, אלא מראה כמה כתובות קדימה או אחורה יש לקפוץ יחסית למיקום מונה התוכנית.

דוגמא 1 : נניח שהיינו בכתובת 500h ונתקלנו בפקודה sjmp label1 . היות והפקודה "תופסת" 2 בתים, אז בסיום הבאת הפקודה מונה התוכנית PC שווה 502h . בכתובת 500h היה את הקוד 80h שהוא קוד הפעולה של sjmp ונניח שבכתובת 501h יש את המספר 25h (שהוא ההיסט – טווח הקפיצה). היות ו 25h = 00100101 ובסיבית ה MSB יש 0 אז הקפיצה קדימה. המיקרו בקר מחבר לתוכן ה PC את המספר 25h ומקבל PC = 502h+25h=527h והפקודה הבאה תבוא מכתובת זו. כלומר קפצנו לכתובת 527h הנמצאת במרחק –offset – של 25h כתובות יחסית לכתובת שהייתה ב PC .
דוגמא 2 : נניח שהיינו בכתובת 600h ונתקלנו בפקודה sjmp label2 . היות והפקודה "תופסת" 2 בתים, אז בסיום הבאת הפקודה מונה התוכנית PC שווה 602h . בכתובת 600h היה את הקוד 80h שהוא קוד הפעולה של sjmp ונניח שבכתובת 601h יש את המספר e5h (שהוא ההיסט – טווח הקפיצה). היות ו e5h = 11100101 ובסיבית ה MSB יש 1 אז הקפיצה אחורה והטווח הוא המשלים ל 2 של המספר. נהפוך כל ביט של 0 ל 1 וכל ביט של 1 ל 0 ונקבל : 00011010 ונוסיף 1 ונקבל : 00011011 . זהו המספר 1bh=27 , כלומר יש לקפוץ 27 כתובות אחורה. 602h+(-1bh)=5e7h.

הערה : הקפיצות המותנות הן קפיצות קצרות.

4.6.5.1.2 קפיצה אבסולוטית Ajmp add11

זוהי קפיצה בתחום של ±2K כתובות . הפקודה תופסת 2 בתים. בביית הראשון מציינים 5 הביטים "הנמוכים" את קוד הפעולה ו 3 הביטים "הגבוהים" של הבית ועוד 8 הביטים של הבית השני מציינים כתובת בתחום של 2048 =2K כתובות.

מ A0 ועד A10 יש 11 ביט ומכאן שהטווח יכול להיות 2^{11}=2048 כתובות.

A10	A9	A8	0	0	0	0	1	╳	A7	A6	A5	A4	A3	A2	A1	A0

הבית הראשון של הפקודה המכיל קוד פעולה (5 ביטים נמוכים) ואת 3 הביטים "הגבוהים" של הכתובת

הבית השני של הפקודה: החלק ה"נמוך" של הכתובת

איור 38 : מבנה הפקודה בקפיצה אבסולוטית

4.6.5.1.3 קפיצה רחוקה Ljmp add16

קפיצה לכל כתובת במרחב 64K הכתובות של זיכרון התוכנית. הבית הראשון הוא קוד הפעולה (02h) ו 2 הבתים הבאים הם הכתובת הפיזית. הבית השני הוא 8 הביטים הגבוהים של הכתובת והבית השלישי הוא 8 הביטים הנמוכים של הכתובת.

4.6.5.2 פרוצדורות במיקרו בקר

פרוצדורה נקראת גם שגרה , רוטינה, תת שגרה (Routine). זוהי קבוצה של הוראות לביצוע מטלה כלשהי. לפרוצדורה קוראים מכתובת כלשהי בתוכנית, קופצים לכתובת שבה רשומה הפרוצדורה , מבצעים את ההוראות הכתובות בפרוצדורה ובסיום תכנית הפרוצדורה מגיעה פקודת ret שהיא קיצור של return (חזור) ואז חוזרים אל השורה ממנה קראנו לפרוצדורה.

השימוש בפרוצדורה נדרש בפיתוח וכתיבה של תכנית גדולה , קל יותר לפרק את התוכנית הגדולה למשימות קטנות שכל אחת מהן מוגדרת כפרוצדורה (שגרה) . גישה זו מאפשרת להתמקד כל פעם בכתיבה ובדיקה של שגרה אחת בנפרד ובסיום מרכיבים את התוכנית כולה מן השגרות השונות שנכתבו .לאחר ששגרה נבדקה ,נוכל להשתמש בה שוב ושוב באותה תכנית או אפילו בתוכניות אחרות וכך לחסוך חלק מן הזמן הדרוש לכתיבת התוכנית. יתרונות נוספים לפרוצדורה הם חיסכון במקום בזיכרון, אחזקה וניפוי (debug) קלים יותר.

במיקרו בקר ממשפחת ה 51 יש 2 סוגי קריאה לפרוצדורה: א. קריאה לפרוצדורה אבסולוטית. ב. קריאה לפרוצדורה רחוקה. ההבדל ביניהן הוא בטווח שבו נמצאת הפרוצדורה מהשורה הקוראת.

4.6.5.2.1 ACALL Addr11 ב - קריאה לפרוצדורה אבסולוטית בתחום של ±2K כתובות . הפקודה תופסת 2 בתים. בביית הראשון מציינים 5 הביטים "הנמוכים" את קוד הפעולה ו 3 הביטים "הגבוה ים" של הבית ועוד 8 הביטים של הבית השני מציינים כתובת בתחום של 2048 =2K כתובות. מ A0 ועד A10 יש 11 ביט ומכאן שהטווח יכול להיות 2^{11}=2048 כתובות.

A10	A9	A8	1	0	0	0	1		A7	A6	A5	A4	A3	A2	A1	A0

הבית הראשון של הפקודה המכיל קוד פעולה (5 ביטים נמוכים) ואת 3 הביטים "הגבוהים" של הכתובת

הבית השני של הפקודה :החלק הנמוך של הכתובת

איור 39 : מבנה פקודת קריאה לפרוצדורה אבסולוטית.

4.6.5.2.2 LCALL Addr16 - קריאה לפרוצדורה רחוקה. הפרוצדורה יכולה להימצא בכל אחת מהכתובות במרחב זיכרון התוכנית. הפקודה "תופסת" 3 בתים. הבית הראשון הוא קוד הפעולה (12h) ו 2 הבתים הבאים הם כתובת הפרוצדורה, כאשר הבית השני הוא 8 הביטים הגבוהים של הכתובת והבית השלישי הם 8 הביטים הנמוכים של הכתובת.

4.6.5.3 תהליך קריאה לפרוצדורה וחזרה מפרוצדורה.

נניח שנמצאים בכתובת 1000h ורשומה בה הפקודה : LCALL sub1 .

כתובת הפרוצדורה sub1 היא 5678h . כמו כן נניח שמצביע המחסנית SP=7

המיקרו בקר מביא את הפקודה מכתובת 1000h בעזרת מונה התוכנית (PC) ומפענח (לפי הקוד 12h) שהפקודה היא "קרא לפרוצדורה רחוקה". המיקרו "יודע" שהוא צריך להביא את הנתונים ב 2 הכתובות הבאות (1001h ו 1002h) שם רשומים כתובת הפרוצדורה sub1 (56h ו 78h בהתאמה) . ניעזר בקטע ממפת הזיכרון הבאה :

זיכרון התוכנית

הערה/הסבר	ערך בהקסה דצימלי	כתובת ב הקסה דצימלי
קרא לפרוצדורה רחוקה	12	1000
8 ביט גבוהים של sub1	56	1001
8 ביט נמוכים של sub1	78	1002
כתובת החזרה		1003
	⋮	⋮
	⋮	⋮
	⋮	⋮
כתובת התחלת :sub1		5678
		5679
	⋮	⋮
קוד פעולה של RET	22	5698
		5699

איור 40 : קטע ממפת זיכרון התוכנית וזיכרון RAM הנתונים הפנימי בקריאה לפרוצדורה

מתוך השרטוט ניתן לראות שבסיום שלב הבאת הפקודה מונה התוכנית נמצא בכתובת 1003h . כעת על המיקרו בקר לעבור לפרוצדורה. לפני שהוא עובר לפרוצדורה עליו לבצע את השלבים הבאים :

1. לשמור את הכתובת 1003h שהיא "כתובת החזרה" במחסנית כדי שבסיום הפרוצדורה הוא יחזור לכתובת זו.
2. לטעון את מונה התוכנית (PC) לכתובת sub1 שהיא 5678h ואז הפקודה הבאה תבוא מכתובת זו וכך הוא מתחיל לבצע את הפקודות של הפרוצדורה.

2 השלבים מתוארים בצורה הבאה:

1. (SP)← (SP) + 1 =7+1=8 — SP - מצביע המחסנית מתקדם ב 1
2. ((SP))← (PC_{7-0}) — לכתובת 8 ייכנסו ביטים 0 עד 7 של מונה התוכנית (03הקסה)
3. (SP)← (SP) + 1=8+1=9 — מצביע המחסנית גדל שוב ב 1
4. ((SP)) ← (PC_{15-8}) — לכתובת זו ייכנסו הביטים 8 עד 15 של מונה התוכנית (10הקסה)
5. (PC) ← addr $_{15-0}$ — כתובת הפרוצדורה (5678 הקסה) נכנסת למונה התוכנית
6. הפקודה הבאה תבוא מהכתובת החדשה שב PC וכך עברנו לכתובת בה נמצאת הפרוצדורה

זיכרון RAM הנתונים הפנימי

הערה/הסבר	ערך בהקסה דצימלי	כתובת ב הקסה דצימלי
		0
		⋮
SP =7 לפני הקריאה לפרוצדורה		7
בכתובות 8 ו 9 נשמרת כתובת החזרה שהיא 1003h	03	8
SP = 9 במעבר לפרוצדורה	10	9
		A
		B

איור 41 : זיכרון RAM הנתונים הפנימי

4.6.5.3.1 תהליך החזרה מפרוצדורה

לאחר שמונה התוכנית נטען לכתובת הפרוצדורה המיקרו מבצע את הפקודות הרשומות בפרוצדורה עד שהוא מגיע לכתובת 5698h ומגיעה הפקודה RET שהיא קיצור של RETURN – חזור. עכשיו המיקרו צריך "לשלוף" מהמחסנית את כתובת החזרה ולחזור אל הכתובת ממנה יצאנו לפרוצדורה (1003h בדוגמא שלנו). מצב ה PC לאחר שהביא את הפקודה מכתובת 5698h (הוא נמצא בכתובת 5699h):

תהליך השליפה מתבצע לפי השלבים הבאים :

1. **(PC_{15-8}) ← ((SP))**

כשסיימנו את הקריאה לפרוצדורה SP=9 . (סעיף 3 בתהליך בעמוד הקודם). מכתובת זו מעבירים את הנתון 10h לביטים 8 עד 15 של מונה התוכנית PC .

2. (SP) ← (SP) – 1 9-1=8 : מצביע המחסנית קטן ב 1

3. **(PC_{7-0}) + ((SP))**

מכתובת 8 מעבירים את הנתון 03h אל 8 הביטים הנמוכים של מונה התוכנית

4. **(SP) + (SP) -1** מצביע המחסנית קטן ב 1 והוא יהיה 7

מונה התוכנית יהיה 1003h והפקודה הבאה תבוא מכתובת זו . חזרנו מהפרוצדורה לאותה כתובת ממנה יצאנו אליה. גם מצביע המחסנית, SP חזר לערך הראשוני שלו (7) שהיה לפני הקריאה לפרוצדורה.

4.6.5.4 פסיקות

נושא הפסיקות במיקרו בקר מוסבר בפרוט בפרקים הבאים.

4.6.5.5 תקציר פקודות קפיצה, קריאה לפרוצדורה, חזרה מפרוצדורה וחזרה מפסיקה

JC rel	Jump if Carry is set	2	2/4
JNC rel	Jump if Carry is not set	2	2/4
JB bit, rel	Jump if direct bit is set	3	3/5
JNB bit, rel	Jump if direct bit is not set	3	3/5
JBC bit, rel	Jump if direct bit is set and clear bit	3	3/5
ACALL addr11	Absolute subroutine call	2	4
LCALL addr16	Long subroutine call	3	5
RET	Return from subroutine	1	6
RETI	Return from interrupt	1	6
AJMP addr11	Absolute jump	2	4
LJMP addr16	Long jump	3	5
SJMP rel	Short jump (relative address)	2	4
JMP @A+DPTR	Jump indirect relative to DPTR	1	4
JZ rel	Jump if A equals zero	2	2/4
JNZ rel	Jump if A does not equal zero	2	2/4
CJNE A, direct, rel	Compare direct byte to A and jump if not equal	3	4/6
CJNE A, #data, rel	Compare immediate to A and jump if not equal	3	3/5
CJNE Rn, #data, rel	Compare immediate to Register and jump if not equal	3	3/5
CJNE @Ri, #data, rel	Compare immediate to indirect and jump if not equal	3	4/6
DJNZ Rn, rel	Decrement Register and jump if not zero	2	2/4
DJNZ direct, rel	Decrement direct byte and jump if not zero	3	3/5
NOP	No operation	1	1

טבלה 17 : תקציר פקודות קפיצה ובקרת מכונה

4.6.5.5.1. הסבר :

מחזורי שעון	דוגמה	הסבר	הפעולה	הקוד המנמוני
4	Acall sub_a קרא לפרוצדורה sub_a . כתובת החזרה נשמרת במחסנית ו 11 הביטים שבפקודה (3 הביטים הגבוהים בבית הראשון של הפקודה ועוד 8 הביטים בבית השני) נטענים ל PC.	קריאה לפרוצדורה אבסולוטית בטווח של עד ±2K	**(PC)← (PC)+** 2 **(SP)** ← (SP) + 1 ((SP)) ← (PC7-0) **(SP)** ← (SP) + 1 ((SP)) ← (PC15-8) (PC 1o-o)← page address	acall Addr1
5	Lcall lsub	קריאה לפרוצדורה רחוקה היכולה להיות בכל כתובת במרחב 64K הכתובות של זיכרון התוכנית	**(PC)** ← **(PC)**+3 (SP)← (SP) + 1 ((SP)) ← (PC_{7-0}) (SP)← (SP) + 1 ((SP))← (PC_{15-8}) (PC) ← addr15-0	lcall addr16
6	Ret חזור. כתובת החזרה "נשלפת" מהמחסנית ומועברת ל PC (מצביע המחסנית מחוסר ב 2) .	חזור מפרוצדורה. שליפת כתובת החזרה מהמחסנית וחזרה לכתובת ממנה יצאנו לפרוצדורה	(PC_{15-8}) ← ((SP)) (SP) ←(SP) – 1 (PC_{7-0}) ← ((SP)) (SP) ← (SP) -1	ret
6	Reti חזור מפסיקה. כתובת החזרה "נשלפת" מהמחסנית ומועברת ל PC (מצביע המחסנית מחוסר ב 2) . לוגיקת הפסיקות מאופשרת.	חזור מפסיקה. כתובת החזרה "נשלפת" מהמחסנית ונטענת ל PC. מצביע המחסנית קטן 2 ולוגיקת הפסיקות מאופשרת בחזרה.	(PC_{15-8})← ((SP)) (SP)← (SP) -1 (PC_{7-0}) ← ((SP)) (SP) ←(SP) -1	reti
4	Ajmp lab_a קפוץ לתווית lab_a הקפיצה מתבצעת ע"י כך ש 11 הביטים שבפקודה (3 הביטים הגבוהים בבית הראשון של הפקודה ועוד 8 הביטים בבית השני) נטענים ל PC.	קפוץ קפיצה אבסולוטית בתחום של ±2K . הסיביות 0 עד 10 של מונה התוכנית מקבלות את 10 הביטים של הכתובת שבפקודה	**(PC) ← (PC)+2** (PClo.o)← page address	ajmp Addr11

5	Ljmp label_f נניח ש label_f נמצא בכתובת 6789h אז PCמקבל את הכתובת הזו וקופצים לבצע את הפקודה שבכתובת זו	קפוץ קפיצה רחוקה לכתובת שבשני הבתים האחרונים של הפקודה	(PC)←$addr_{15-0}$	ljmp Addr16
4	Sjmp label_s קפוץ לתווית label_s הנמצאת בטווח של עד 127 קדימה ומינוס 128 אחורה. הקפיצה מתבצעת ע"י חיבור תוכן ה PC עם הטווח עד label_s .	קפוץ קפיצה קצרה יחסית בטווח של פלוס 127 מינוס 128 .	(PC)← (PC) + 2 (PC)←(PC)+ rel	sjmp rel
4	Jmp @A+dptr אם A=20H ו dptr=500h אז PC=520h וקופצים לכתובת זו.	קפוץ לכתובת שהיא הסכום של הנתון (בן 8 הביטים הלא מסומן) שב A וב dptr . הסכום נכנס למונה התוכנית	(PC)← (a)+(dptr)	jmp @A+dptr

טבלה 18 –פקודות קפיצה לא מותנות

4.6.5.5.2 פקודות קפיצה מותנות

בפקודות אלו הקפיצה מותנית במצבו של האקומולטור, או של דגל ה CY או של ביט כלשהו. במידה והתנאי מתקיים אז המיקרו קופץ לכתובת שרשומה בפקודה. אם התנאי איננו מתקיים המיקרו בקר עובר להמשך הביצוע של הפקודה הבאה.

מחזורי שעון	דוגמה	הסבר	הפעולה	הקוד המנמוני
2/4	Jz label_z קפוץ לכתובת label_z אם 8 הביטים באקומולטור הם 0 . אחרת המשך לפקודה הבאה.	קפוץ לכתובת היחסית (בטווח של פלוס 127 מינוס 128) אם 8 הביטים באקומולטור הן 0 . אחרת המשך בתוכנית	(PC)←(PC)+ 2 IF (A) = O THEN (PC)←(PC)+ rel	jz rel
2/4	Jnz label_nz	קפוץ לכתובת היחסית (בטווח של פלוס 127 מינוס 128) אם לפחות באחד הביטים באקומולטור יש 1 .	(PC)←(PC)+ 2 IF (A) ≠ O THEN (PC)←(PC)+ rel	jnz rel

		אחרת המשך בפקודה הבאה.		
jc rel	(PC)← (PC)+ 2 IF (C) = 1 THEN (PC)←(PC)+ rel	קפוץ לכתובת היחסית (בטווח של פלוס 127 מינוס 128) אם בדגל הנשא (CY) יש 1 . אחרת המשך בפקודה הבאה.	Jc label1	2/4
jnc rel	(PC)← (PC) + 2 IF (C) = O THEN (PC)←(PC)+ rel	קפוץ לכתובת היחסית (בטווח של פלוס 127 מינוס 128) אם בדגל הנשא (CY) יש 0 . אחרת המשך בפקודה הבאה.	Jnc label2	2/4
jb bit,rel	(PC)← (PC)+ 3 IF (bit) = 1 THEN (PC)←(PC)+ rel	קפוץ לכתובת היחסית (בטווח של פלוס 127 מינוס 128) אם בביט המצוין בפקודה יש 1 . אחרת המשך בפקודה הבאה.	Jb 70h,label_b קפוץ לתווית label_b אם בביט מספר 70h יש 1 . אחרת המשך בפקודה הבאה.	3/5
jnb bit,rel	(PC)← (PC)+ 3 IF (bit) = 0 THEN (PC)←(PC)+ rel	קפוץ לכתובת היחסית (בטווח של פלוס 127 מינוס 128) אם בביט המצוין בפקודה יש 1 . אחרת המשך בפקודה הבאה.	Jnb acc.0,zugi קפוץ לתווית zugi אם בביט LSB של האקומולטור יש 0 (אם המספר זוגי). אחרת המשך בפקודה הבאה.	3/5
jbc bit,rel	(PC) ← (PC)+ 3 IF (bit) = 1 THEN (bit)← O (PC)←(PC)+ rel	קפוץ לכתובת היחסית (בטווח של פלוס 127 מינוס 128) אם בביט המצוין בפקודה יש 1 וגם אפס את הביט. אחרת המשך בפקודה הבאה.	Jbc 60h,lab_bc קפוץ לכתובת lab_bc אם בביט מספר 60h יש 1 וגם הפוך את הביט ל 0 . אחרת (אם בביט היה 0) המשך בפודה הבאה.	3/5

cjne a,direct,rel	זהו קיצור הפקודה :
	Compare and **J**ump if **N**ot **E**qual.
	הפקודה מבצעת :
	(PC) ← (PC)+ 3
	IF (A) < > *(direct)*
	THEN
	(PC)← (PC)+
	relative offset
	IF (A) < *(direct)*
	THEN
	(c) ←1
	else
	(c) ← o
		השווה בין הנתון ב A והנתון שב direct . אם הם **לא** שווים אז קפוץ לכתובת היחסית. אם הם כן שווים – המשך בפקודה הבאה.
		פעולת ההשוואה משפיעה על דגל ה CY.
		אם a<(direct) אז CY←1 , אחרת 0 .
		הנתונים ב A וב direct הם לא מסומנים !
			Cjne a,50h,label_d
			השווה בין הנתון שב Aובין הנתון שבכתובת 50h בזיכרון הנתונים הפנימי. אם הם לא שווים קפוץ לכתובת label_d . אם הם שווים המשך בפקודה הבאה. אם a < (50h) אז CY←1 , אחרת 0 . הנתון ב A וב direct הם לא מסומנים !
				4/6

cjne a,#data,rel	זהו קיצור הפקודה :
	Compare and **J**ump if **N**ot **E**qual.
	הפקודה מבצעת :
	(PC)← (PC) + 3
	IF (A) < > data
	THEN
	(PC)←(PC)+
	Relative offset
	IF (A) < *data*
	THEN
	(C) ← 1
	else
	(C) ← 0
		השווה בין הנתון ב A והנתון #data . אם הם **לא** שווים אז קפוץ לכתובת היחסית. אם הם כן שווים – המשך בפקודה הבאה.
		פעולת ההשוואה משפיעה על דגל ה CY.
		אם a<data אז CY←1 , אחרת 0 .
		הנתונים ב A וב direct הם לא מסומנים !
			Cjne a,#50h,label_d
			השווה בין הנתון שב Aובין הנתון 50h .
			אם הם לא שווים קפוץ לכתובת label_d .
			אם הם שווים המשך בפקודה הבאה. אם a < 50h אז CY←1 , אחרת 0 . הנתונים לא מסומנים !
				3/5

cjne Rn,#data,rel	זהו קיצור הפקודה :	השווה בין הנתון ב Rn והנתון #data . אם הם **לא** שווים אז קפוץ	Cjne R2,#50h,label_e השווה בין הנתון שב R2 ובין הנתון 50h . אם הם לא שווים קפוץ לכתובת label_e .	3/5

	Compare and **J**ump if **N**ot **E**qual.
	הפקודה מבצעת :
	(PC)← (PC) + 3
	IF (Rn) < > data
	THEN
	(PC)←(PC)+
	Relative offset
	IF (Rn) < *data*
	THEN
	(C) ← 1
	else
	(C) ← 0
		לכתובת היחסית. אם הם כן שווים – המשך בפקודה הבאה.
		פעולת ההשוואה משפיעה על דגל ה CY.
		אם (Rn)<#data אז CY←1 , אחרת 0 .
		הנתונים ב A וב direct הם לא מסומנים !
			אם הם שווים המשך בפקודה הבאה. אם (R2) < 50h אז CY←1 , אחרת 0 .
			הנתונים לא מסומנים !	

cjne @Ri,#data,rel	זהו קיצור הפקודה :
	Compare and **J**ump if **N**ot **E**qual.
	הפקודה מבצעת :
	(PC)← (PC) + 3
	IF ((Ri)) < > data
	THEN
	(PC)←(PC)+
	Relative offset
	IF ((Ri)) < *data*
	THEN
	(C) ← 1
	else
	(C) ← 0
		השווה בין הנתון בכתובת ש Ri מצביע עליה והנתון #data .
		אם הם **לא** שווים אז קפוץ לכתובת היחסית.
		אם הם כן שווים – המשך בפקודה הבאה.
		פעולת ההשוואה משפיעה על דגל ה CY.
		אם ((Ri))<#data אז CY←1 , אחרת 0 .
		הנתונים ב A וב direct הם לא מסומנים !
			Cjne @R1,#50h,label_f
			נניח R1=60h
			השווה בין הנתון בכתובת ש R1 מצביע עליה (הנתון בכתובת 60h) ובין הנתון 50h . אם הם לא שווים קפוץ לכתובת label_f .
			אם הם שווים המשך בפקודה הבאה. אם (60) < 50h אז CY←1 , אחרת 0 .
			הנתונים לא מסומנים !
				4/6

djnz Rn,rel	הפקודה היא קיצור: **D**ecrement and **J**ump if **N**ot **Z**ero	חסר 1 מרגיסטר Rn ואם הוא לא 0 קפוץ לכתובת היחסית (בטווח של עד 127 כתובות	Djnz r5,label_x חסר 1 מרגיסטר R5 ואם הוא לא 0 קפוץ לכתובת label_x .	2/4

	הפקודה מבצעת: (PC) ← (PC)+2 (Rn) ←(Rn) – 1 IF (Rn) > 0 or (Rn) < 0 THEN (PC)←(PC)+ rel	קדימה או עד 128 כתובות אחורה).		
djnz direct,rel	הפקודה היא קיצור: **D**ecrement and **J**ump if **N**ot **Z**ero הפקודה מבצעת: (PC) ← (PC)+2 (direct) ←(direct) – 1 IF (direct) > 0 or (direct) < 0 THEN (PC)←(PC)+ rel	חסר 1 מהתוכן של הכתובת direct ואם התוכן הוא לא 0 קפוץ לכתובת היחסית (בטווח של עד 127 כתובות קדימה או עד 128 כתובות אחורה).	Djnz 75h,label_x חסר 1 מהנתון בכתובת 75h ואם הוא לא 0 קפוץ לכתובת label_x .	3/5
nop	(PC)← (PC) + 1	הפקודה איננה מבצעת דבר. משמשת להשהיה קצרה או לשמירת מקום לפקודות נוספות.	nop	1

טבלה 19 – פקודות קפיצה מותנות

4.6.6 תרגילי דוגמא וחישובי זמן:

תרגיל 1

מה עושה הפרוצדורה (שגרה) הבאה ? למה היא יכולה לשמש ?

```
Mov R5,#10
Djnz r5,$      ;        again: djnz r5,again   : פקודה זהה לפקודה
Ret
```

פתרון תרגיל 1

בשורה הראשונה מעבירים ל r5 את הנתון 10 .

בשורה השנייה מחסרים 1 מ r5 ואם r5 לא אפס חוזרים לאותה השורה פעם נוספת. הפקודה djnz r5,$ שקולה לפקודה again: djnz r5,again . בדוגמא שלנו אחרי חיסור 1 מ r5 יש בו 9 , כלומר הוא לא 0 וחוזרים שוב לאותה השורה. פעם נוספת מחסרים אחד מ r5 (עכשיו יש בו 8) והיות ואין בו 0 שוב חוזרים לאותה השורה. כך נשארים בשורה הזו 10 פעמים , עד ש r5 הוא אפס ואז המיקרו מבצע את פקודת ה ret ומסיים את הפרוצדורה.

אם היינו שמים ב r5 את הנתון 20 אז היינו מבצעים את השורה djnz r5,$ עשרים פעמים. מכאן שהמספר שנטעין את r5 קובע כמה פעמים נעשה את השורה djnz r5,$ ולמעשה המספר קובע את הזמן שבו נשהה בפרוצדורה. תכנית כזו משמשת אותנו להשהיה.

4.6.6.1 חישובי זמן

מהירות הביצוע של כל פקודה תלויה במהירות המיקרו בקר. מהירות זו תלויה בגביש שבתוך המיקרו בקר או בגביש חיצוני שאותו מחברים בין ההדקים P0.2 ו P0.3 של המיקרו בקר. תדר הגביש נקרא $f_{crystal}$. זמן המחזור של תדר הגביש נקרא מחזור שעון ומסומן כ Tclk .

מחזור שעון תלוי בגביש. לדוגמא : עבור גביש של $f_{crystal}$ = 48MHz יחושב זמן מחזור שעון כך:

$$\mathbf{Tclk = 1/f_{crystal} \rightarrow Tclk = 1/(48*10^{6)} = 1/48 * 10^{-6} = 1/48\ \mu Sec = 0.0208333333333\ \mu Sec\ .}$$

אנחנו נקרב את הערך ל : 0.021 מיקרו שנייה.

כל פקודה מאוצר ההוראות של המיקרו בקר מתבצעת בין 1 עד 6 מחזורי שעון . עבור תדר של 48MHz – זמן ביצוע הפקודה יהיה : 6*0.021**µSec=0.126µSec** .

העתקנו את התרגיל הקודם והוספנו בסוגריים את כמות מחזורי המכונה שהפקודה "תופסת".

```
Mov R5,#10    ;  (2)
Again : Djnz r5,again     ;   (2/4)
Ret           ;  (6)
```

רואים שהפקודה mov r5,#10 מבוצעת במחזור שעון אחד. הפקודה djnz r5,$ מבוצעת בין 2 ל 4 מחזורי שעון, כלומר כאשר מסיימים את הפקודה שבכתובת again , מונה התוכנית נמצא בכתובת (again+2) . 9 פעמים חוזרים לכתובת again לפקודה Djnz R5,again ופעם אחת (כש R5 שווה 0) לא צריך לחזור . 9 פעמים הפקודה מבוצעת ב 4 מחזורי שעון ופעם אחת ב 2 מחזורי שעון. הפקודה ret מתבצעת ב 6 מחזורי מכונה. היות והפקודה djnz r5,$ מתבצעת 10 פעמים אז סך מחזורי המכונה (נסמן ב N) יהיה:

N = 2 + 9*4 + 1*2 + 6 =46 clock cycles

היות וחישבנו שעבור גביש של 48 מגה הרץ, זמן שעון הוא 0.021 מיקרו שנייה ואז הזמן שבו תבוצע הפרוצדורה יהיה כמות מחזורי השעון להכפיל בזמן שעון.

t = 46 * 0.021 USec = 0.966µSec

תרגיל 2

נתון מיקרו בקר עם גביש חיצוני של 6Mhz והפרוצדורה הקודמת. בשורה הראשונה רשמנו mov r5,#100 במקום mov r5,#10 . מה יהיה זמן ההשהיה של הפרוצדורה ?

פתרון תרגיל 2

נחשב את זמן מחזור שעון:

$Tclk = 1 / (6*10^6) = 0.166\ \mu Sec$

ההבדל בתוכנית עכשיו מזו הקודמת הוא שפעולת החיסור מ r5 מבוצעת 100 פעמים ולא 10 . 99 פעמים קופצים חזרה ל again ופעם אחת לא חוזרים.

נחשב את סך מחזורי השעון של הפרוצדורה:

$N = 2 + 99*4 + 1*2 + 6 = 406$ מחזורי שעון.

נחשב את הזמן (ההשהיה) הכוללת :

$t = 406 * 0.166\ USec = 67.396\mu Sec$

תרגיל 3

ידוע שתדר הגביש 48MHz . נניח שהקרוסבר הוגדר . מה עושה קטע התוכנית הבא:

```
Again: cpl  P1.0        (1)
       Lcall delay      (5)
       sjmp again       (4)
        ¦
delay: mov r5,#10
       Djnz r5,$        0.966µSec  : 1 מתרגיל
       Ret
```

פתרון תרגיל 3

הפקודה cpl p1.0 הופכת את מצב ההדק של p1.0. אם בהדק היה 0 (מתח קרוב ל 0 וולט) אז הוא עובר ל 1 (מתח קרוב ל 3 וולט) ולהפך. פקודה זו מבוצעת במחזור שעון אחד וראינו שעבור הגביש של 48 מגה הרץ , מחזור שעון הוא 0.021 מיקרו שניות.

הפקודה lcall delay נמשכת 5 מחזורי שעון – 5*0.021µSec וקוראת לפרוצדורת השהייה delay שנמשכת 0.966USec .

הפקודה sjmp again מבוצעת ב 4 מחזורי שעון , כלומר 4*0.021 מיקרו שניות.

סה"כ מעבר של התוכנה על כל הפקודות נמשך :

$T= 0.021 + 5*0.021 + 0.966 + 4*0.021 = 1.176\ \mu Sec$

cpl P1.0 + Lcall delay + delay + sjmp again

אחרי ביצוע הפקודה sjmp again חוזרים לכתובת again ושוב הופכים את מצב הדק P1.0 , קוראים לפרוצדורת ההשהיה וכאשר חוזרים מהפרוצדורה שוב קופצים לכתובת again וחוזר חלילה. ב P1.0 נקבל גל מרובע באמפליטודה שבין 0 ל 3 וולט . זמן המחזור של הגל המורכב מהזמן שהגל ב 0 ועוד הזמן שהגל ב 1 והוא $2*1.176\ \mu Sec = 2.352\ \mu Sec$. ומכאן שתדר הגל המרובע בהדק P1.0 יהיה :

$f_{p1.0} = 1 / (2.352 *10^{-6)} = 425,170.068\ Hz$

4.7 הכרת סביבת הפיתוח µVision

4.7.1 כללי

כדי לכתוב ולהריץ תוכנית באסמבלי או בשפת על כמו C צריך להתקין במחשב סביבת פיתוח הנקראת **IDE** . זהו קיצור של Integrated Development Environment - ובעברית סביבת פיתוח משולבת. בסביבה כזו יש מספר קבצים המכילים בין היתר את המרכיבים הבאים :

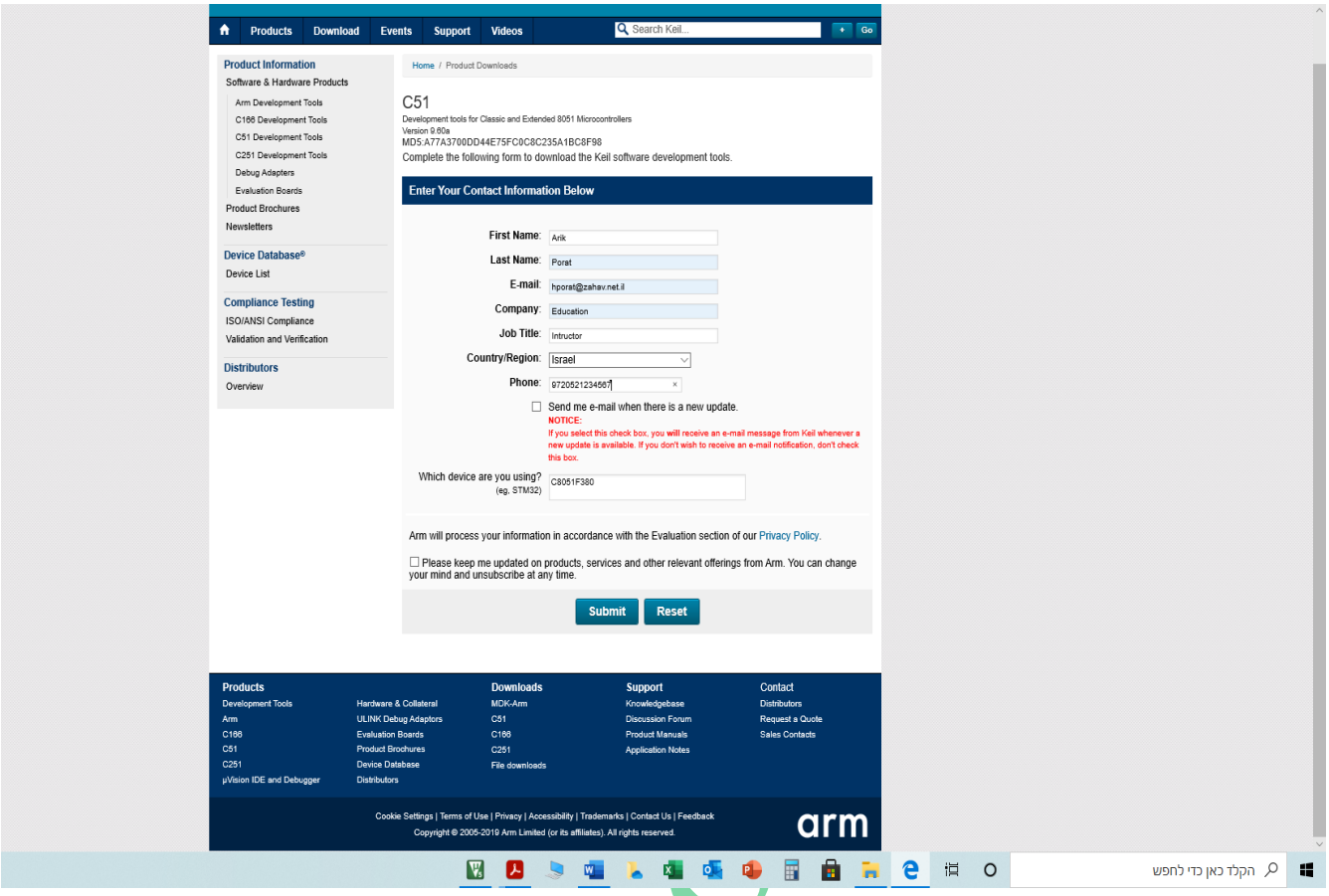
- מעבד תמלילים (כמו word או qtext או notepad) שבעזרתו כותבים את התוכנית. מעבד התמלילים מכיל את האפשרויות של כתיבה, מחיקה, העתקה, הדבקה וטיפול בקבצים (טעינה ושמירה) . את התוכנית שכותבים יש לשמור עם סיומת המתאימה . לדוגמה אם הקובץ באסמבלי של ה 51 הוא ייקרא name.a51 . אם הוא ב C הוא ייקרא name.c .
- לאחר כתיבת התוכנית יש לבדוק האם התוכנית נכתבה עם שגיאות תחביר -Syntax . אם כן התוכנית איננה מתורגמת ומודיעה למשתמש היכן הטעויות כדי שיוכל לתקן. אחרי שהמשתמש תיקן את טעויות התחביר התוכנה מתרגמת את התוכנית לשפת מכונה. התוכנה שעושה את 2 הדברים נקראת אסמבלר עבור תוכנית באסמבלי או קומפיילר אם התוכנית רשומה בשפת על. הקובץ שנוצר נקרא קובץ HEX עבור מיקרו בקרים ו קובץ exe במערכות מחשב.
- לאחר שהקומפיילר תירגם את התוכנית יש צורך לבדוק האם התוכנית עובדת כראוי. לשם כך יש תוכנה הנקראת Debug – ניפוי - שמאפשרת למשתמש להריץ את התוכנית ובמידה ואיננה עובדת כראוי יש אפשרות לעבור פקודה אחר פקודה step one line או להריץ את התוכנית עד פקודה רצויה אותה מסמנים עם העכבר Go to Cursor , או ליצור נקודות שבירה Break points ועוד.
- קיים גם מנהל פרויקטים אם רוצים לשלב מספר תוכנות שרצות תחת תוכנית אחת. המנהל מבצע קישור (LINK) לכל הקבצים שנמצאים בפרויקט.

ניתן להריץ תוכניות שכתבנו למיקרו בקר עם מספר סביבות עבודה. לחברת Silicon Labs עצמה יש סביבת העבודה משלה הנקראת Simplicity Studio אבל משרד החינוך ממליץ לעבוד עם תוכנה הנקראת µVision של חברת Keil .

µVision היא IDE - סביבת פיתוח משולבת - מבוססת חלונות המשלבת מעבד תמלילים/עורך חזק ומודרני עם מנהל פרויקטים. הוא משלב את כל הכלים הדרושים לפיתוח יישומים משובצים, כולל מהדר C/C++ , מקשר ויוצר קבצי µVision וקבצי מכונה עם סיומת HEX ומסייע לזרז את תהליך הפיתוח של יישומים. הוא מאפשר סימולציה (הרצת התוכנית בהדמייה במחשב ולא ממש עם מיקרו בקר) וגם צריבה של זיכרון התוכנית מסוג FLASH של המיקרו בקר. היא עובדת גם עם מיקרו בקרים מסוגים שונים של חברות אחרות.

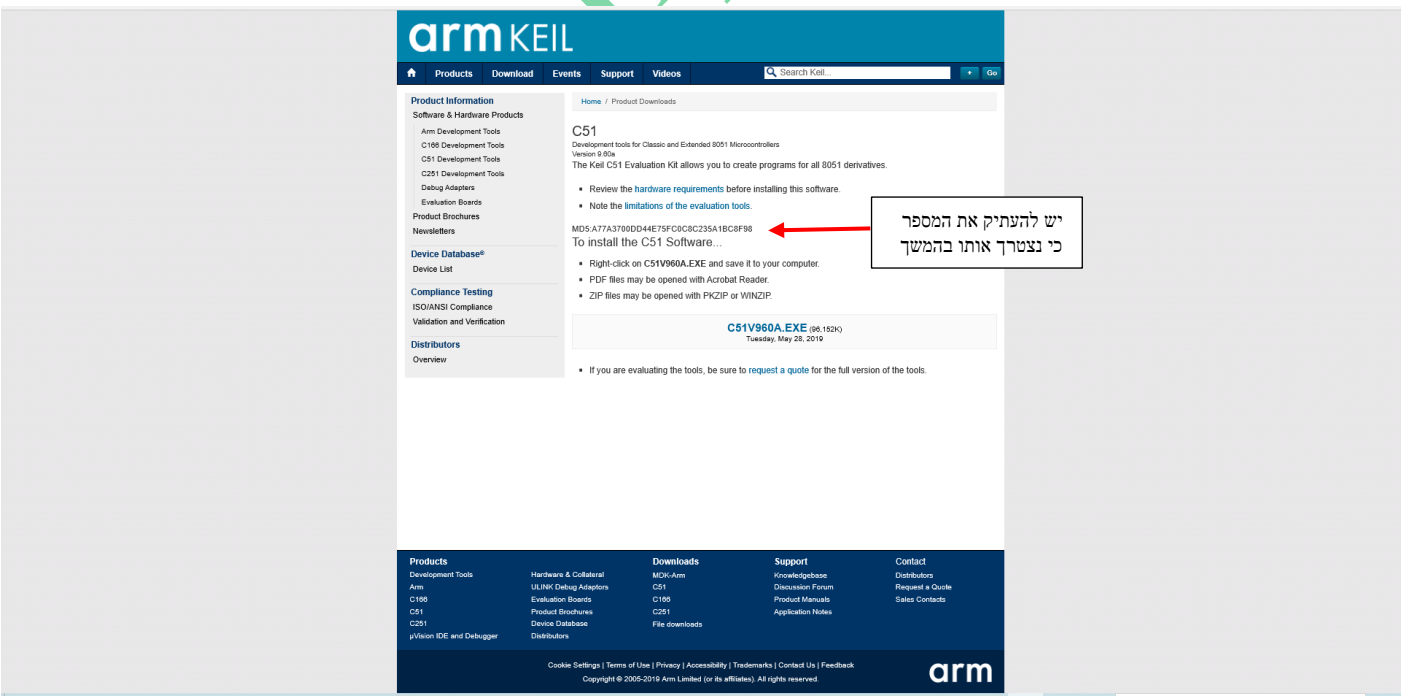
4.7.2 הורדת סביבת הפיתוח µVision5 למחשב

את תוכנת µVision5 ניתן להוריד בקישור : https://www.keil.com/demo/eval/c51.htm . במסך שייפתח שנראה באיור הבא יש למלא שאלון . יש לרשום את השם הפרטי, שם המשפחה ה Email , החברה שבה אתה עובד (אני רשמתי חינוך – Education) , תפקיד (אני רשמתי instructor . אם את/ה סטודנט רשום student), אחר כך יש להכניס את הארץ ואת הטלפון. למטה יותר, אחרי הכתב באדום יש אפשרות לסמן במלבן האם לקבל מייל כאשר יש עדכונים חדשים (לא חובה) , מתחת יש מלבן שבו יש לרשום באיזה רכיב מדובר (אני רשמתי C8051F380) ואז לוחצים למטה על Submit .



איור 4.7.1 : מסך הרשמה לתוכנה של Keil .

לאחר הלחיצה על Submit מקבלים את המסך הבא :



איור 4.7.2 : מסך הורדת התוכנה

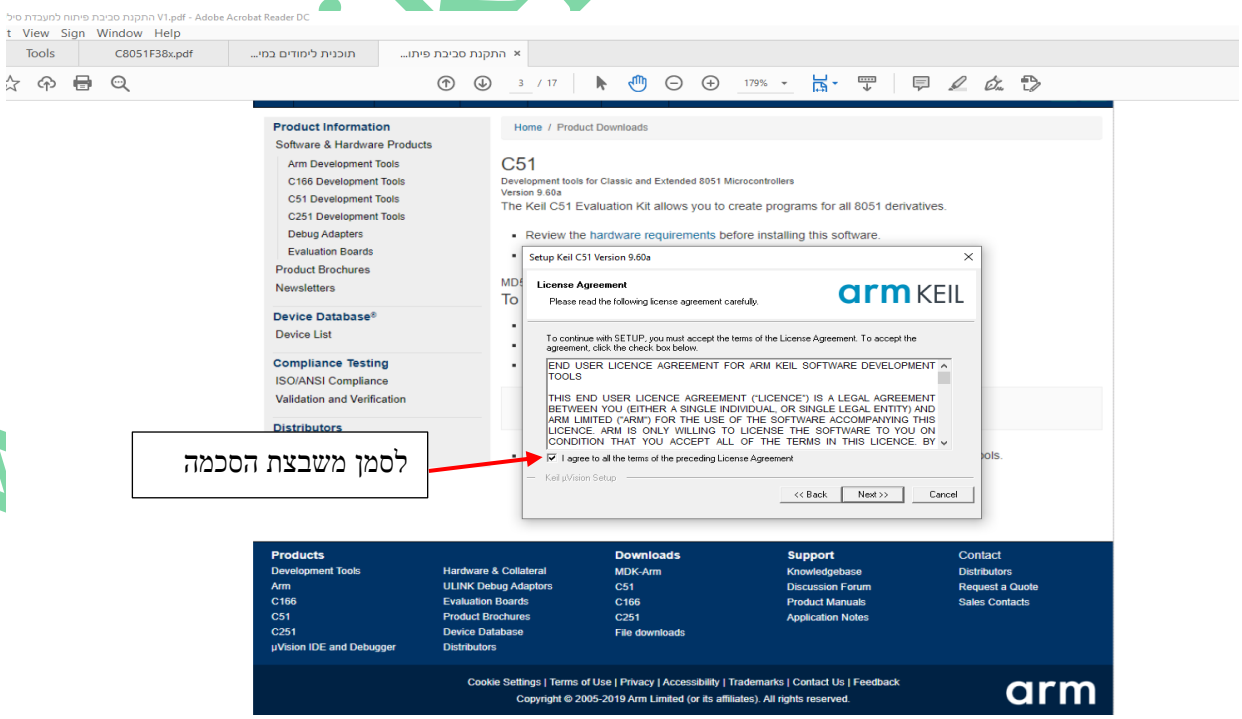
המספר שהעתקנו הוא : MD5:A77A3700DD44E75FC0C8C235A1BC8F98

מביאים את הסמן עם העכבר על הכיתוב C51V960A.EXE ולוחצים כדי להוריד את התוכנה. במחשב שלי הופיעה השאלה האם לאפשר לתוכנה לבצע שינויים במחשב ועניתי לאפשר ואחר כך הופיעה השאלה האם להפעיל או לשמור ורשמתי להפעיל.

המסך החדש שהתקבל נראה כך :

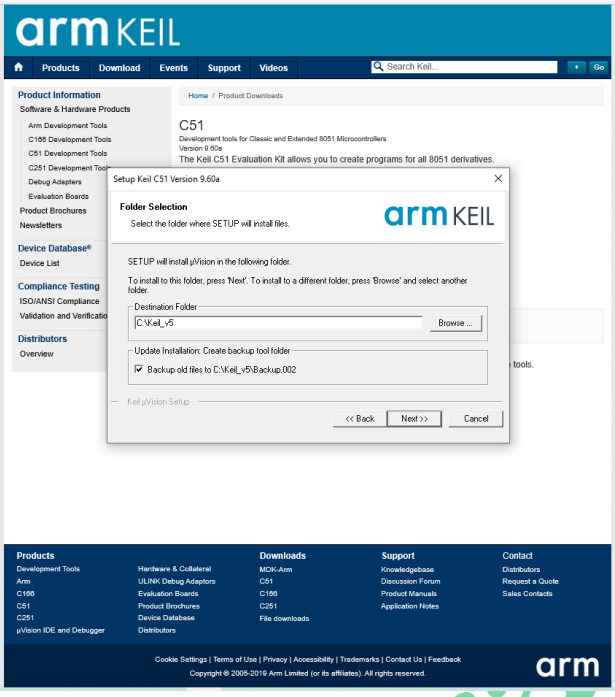
איור 4.7.3 : תוכנית ההתקנה

לחיצה על Next ונקבל את מסך רישיון ההסכמה.



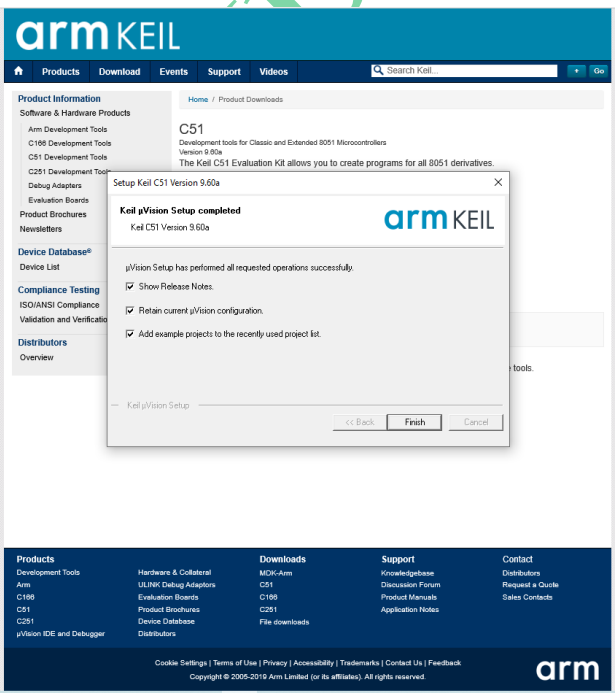
איור 4.7.4 :מסך רישיון ההסכמה

לאחר לחיצה על Next נקבל את המסך :



איור 4.7.5 : מיקום התוכנית בכונן C בספרייה Keil_v5

לחיצה נוספת על Next תראה לך את הפרטים שרשמת ברישום ותיתן לך אפשרות לשנות. לחץ שוב על Next ותתחיל טעינה של התוכנית. בסיום נקבל את המסך :



איור 4.7.6 : מסך סיום טעינת התוכנית.

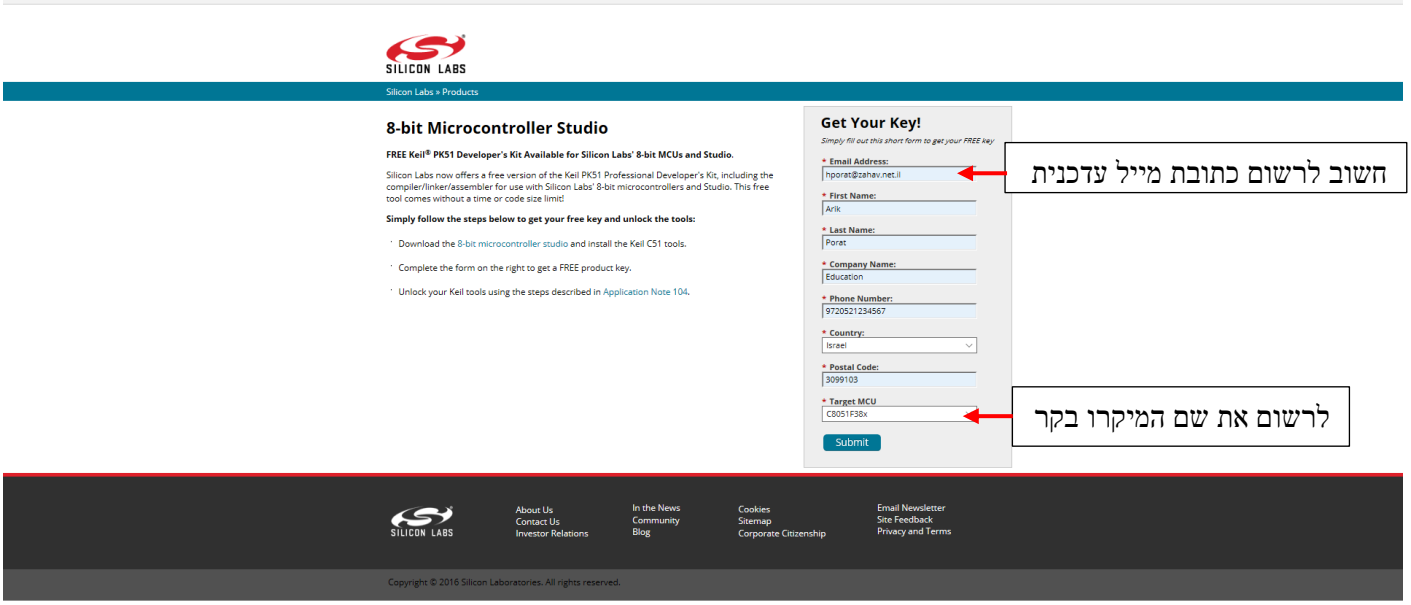
לחיצה על Finish תסיים את טעינת התוכנית. **מכאן ניתן להתחיל לעבוד עם התוכנה** ! אין צורך לעבור לשלבים הבאים שמטרתם לקבל מפתח מוצר ולעבוד כאדמיניסטרטור. נמשיך לשלב הבא.

4.7.3 – קבלת מפתח מוצר

כדי לקבל רישיון שנקרא מפתח מוצר product key או מספר סידורי של המוצר ניכנס לקישור :

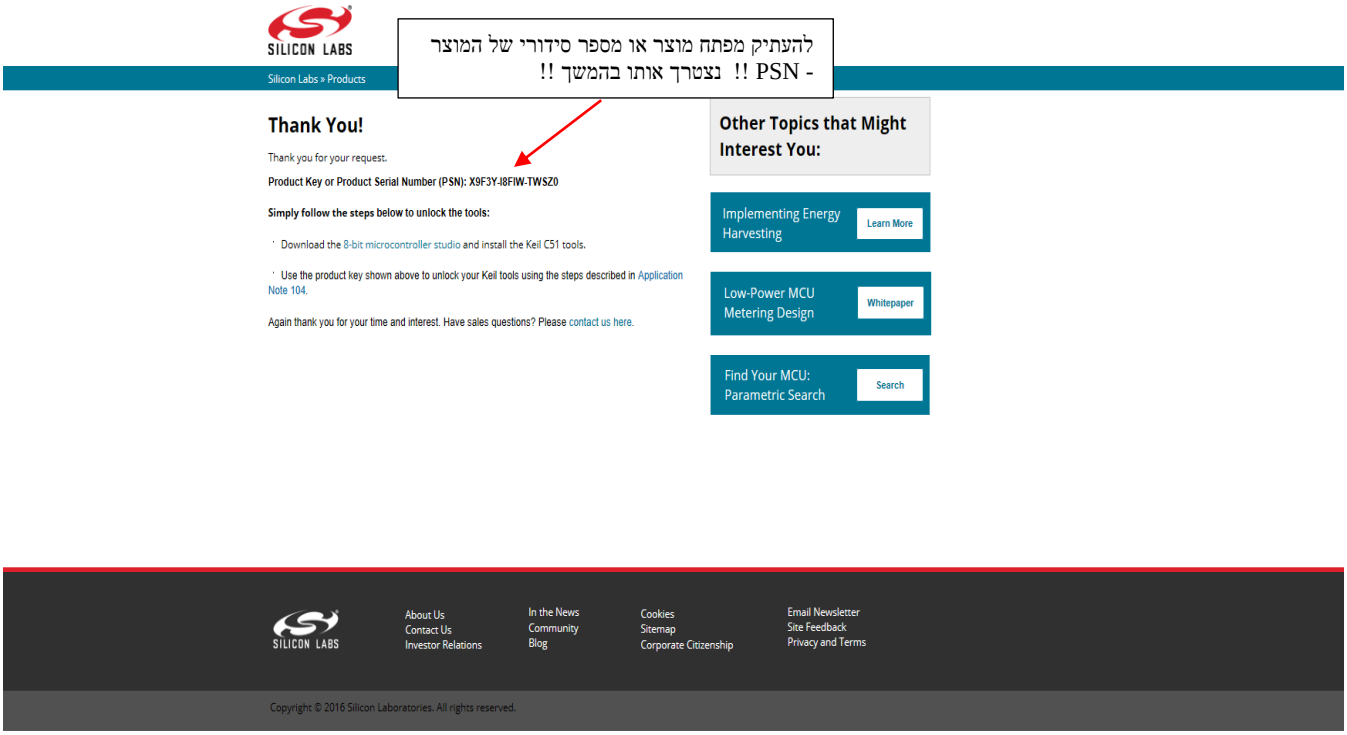
https://pages.silabs.com/lp-keil-pk51.html

נקבל מסך עם האיור הבא :



איור 4.7.7 : מסך לקבלת מפתח מוצר

לחיצה על Submit תעביר אותנו למסך הבא :



איור 4.7.8 : מסך סיום ההרשמה למפתח המוצר

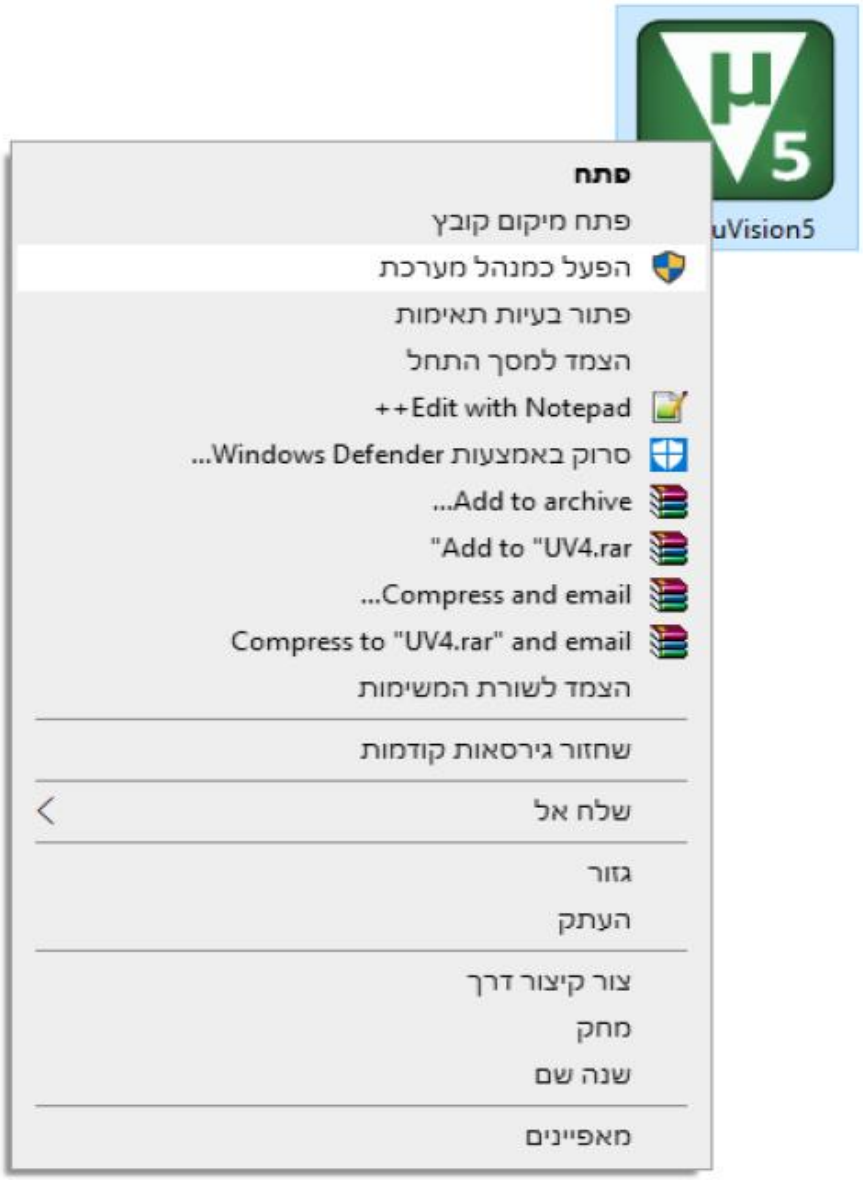
מפתח המוצר במקרה שלי הוא (אתם תקבלו מספר אחר ויש להעתיק אותו ולשמור לשלב הבא) :

Product Key or Product Serial Number (PSN): X9F3Y-I8FIW-TWSZ0

PSN – Product Serial Number – מספר סידורי של המוצר. כאן סיימנו את קבלת מפתח המוצר/מספר סידורי מוצר .

4.7.3.1 – הפעלה כמנהל (אדמיניסטרטור)

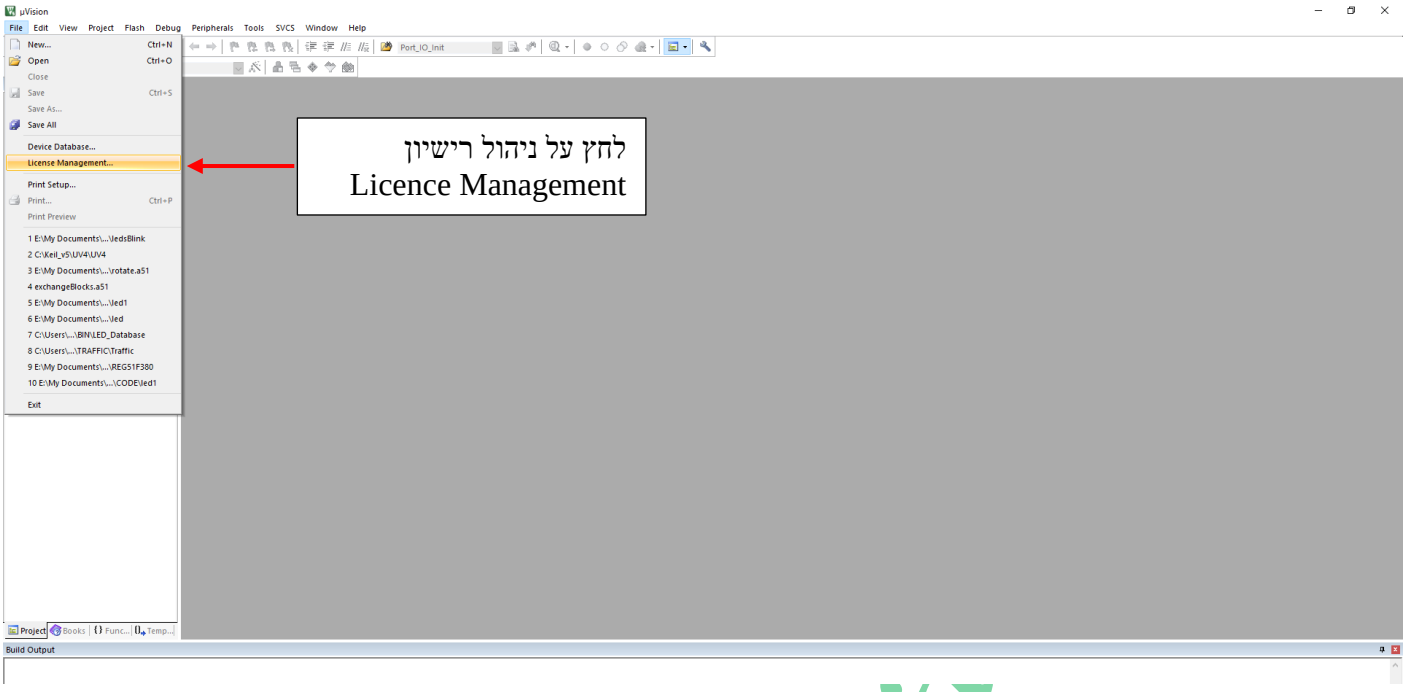
אחרי שקיבלנו את מפתח המוצר נפעיל את התוכנה כמנהל המערכת - אדמיניסטרטור. לשם כך נעבור למסך הבית - desk top ונלחץ עם המפסק הימני בעכבר על ICON (צלמית) לקיצור הדרך לתוכנית **µVision5** ונקבל.



איור 4.7.9: הפעלה כאדמיניסטרטור

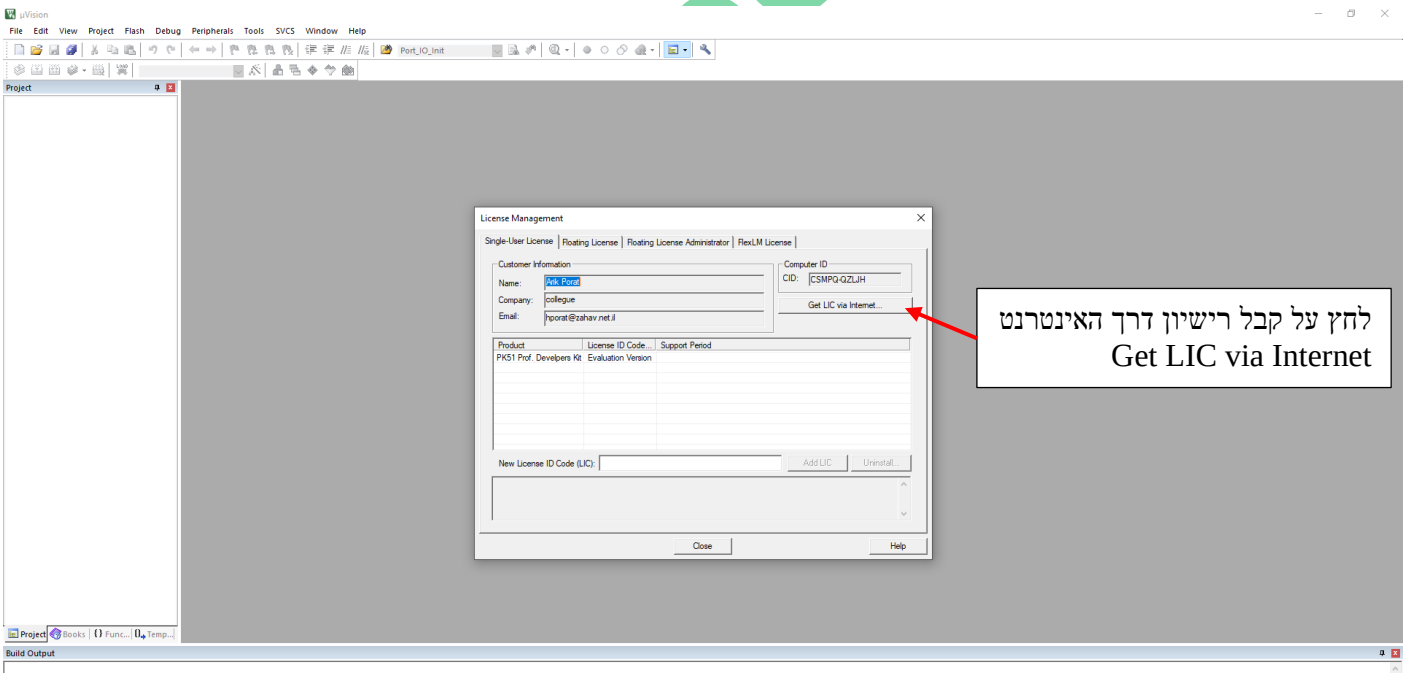
4.7.3.2 הפעלה כמנהל המערכת - אדמיניסטרטור

נלחץ על "הפעל כמנהל המערכת" ונקבל את המסך שבאיור הבא . זוהי התוכנה **µVision5** שנעבוד איתה. נלחץ על file בתפריט ונקבל:



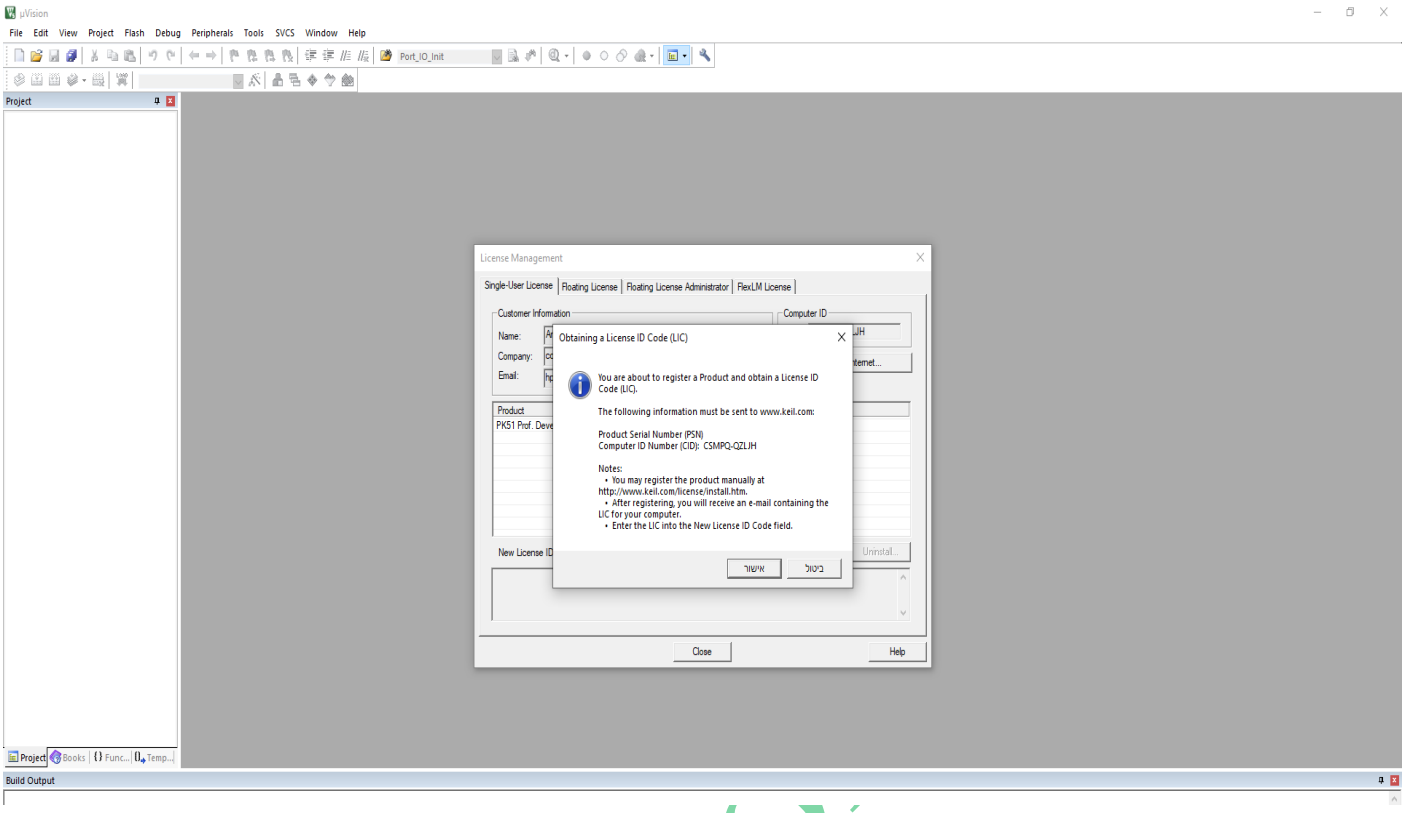
איור 4.7.10 : מסך התוכנה שבו נפעיל את "ניהול רישיון" עם מפתח המוצר/המספר הסידורי .

לחיצה על License Management תיתן לנו את המסך הבא:



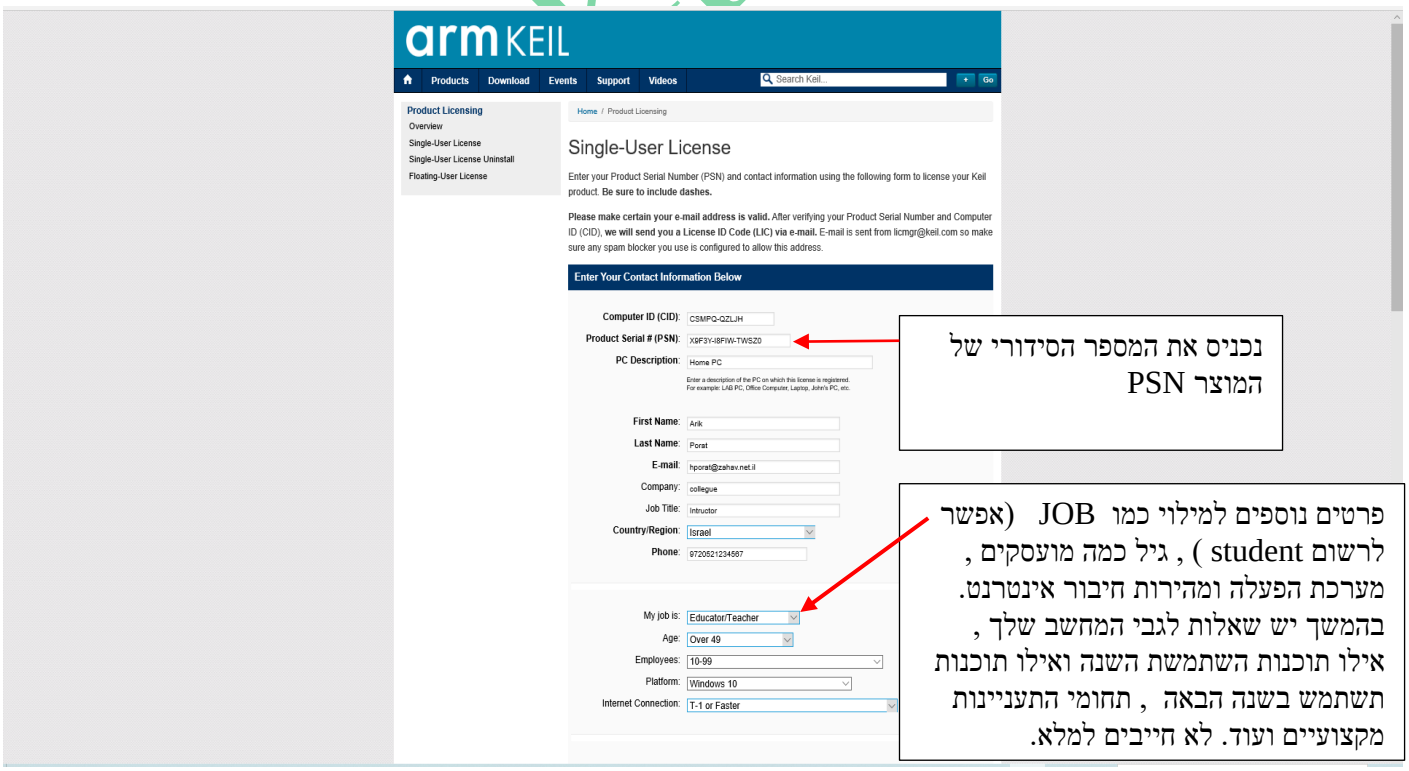
איור 4.7.11: קבלת רישיון דרך האינטרנט

לחיצה על Get LIC via Internet ונקבל את המסך באיור הבא שמודיע לך שאתה עומד לשלוח את מספר המחשב והמספר הסידורי של המוצר – PSN - Product Serial Number לחברת Keil :



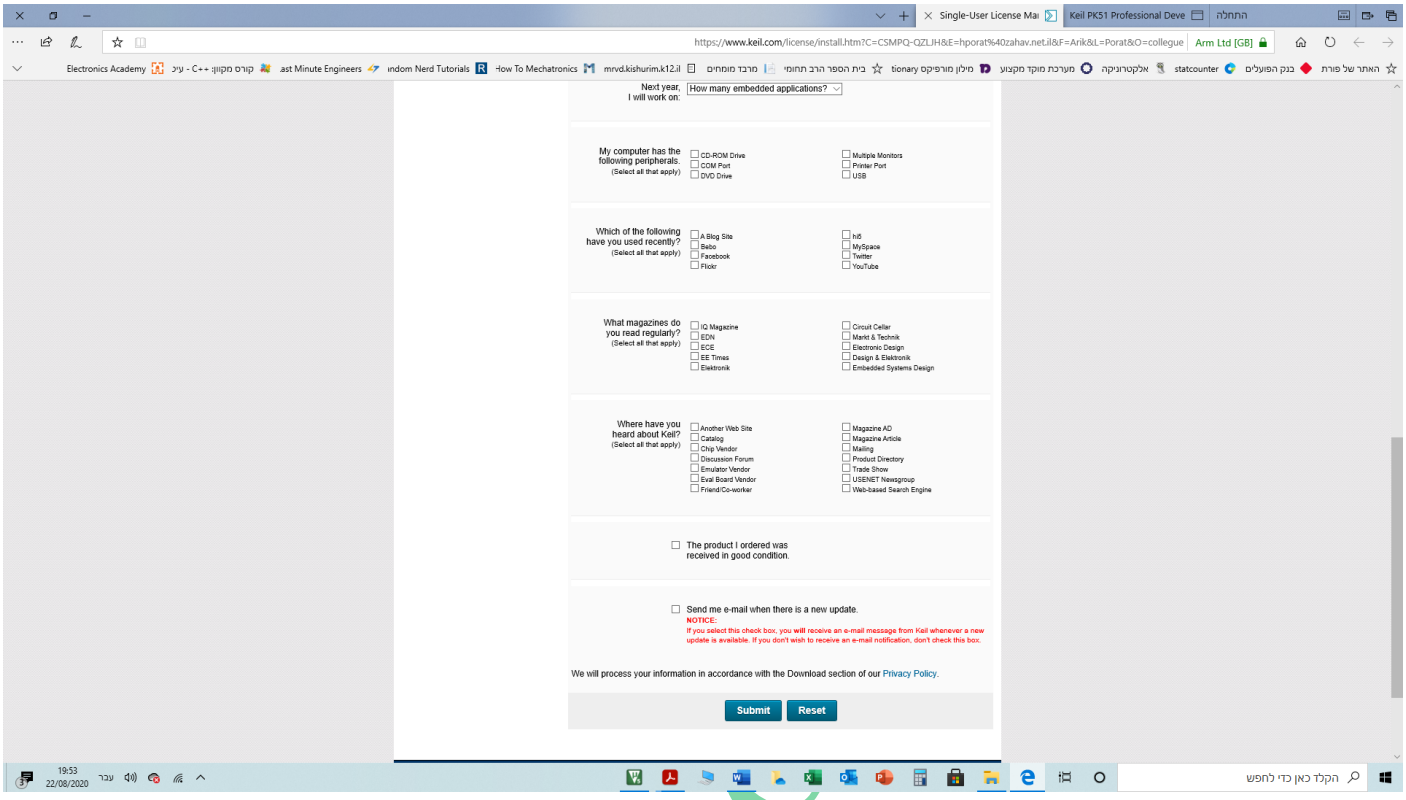
איור 4.7.12: מסך הודעה על שליחת מספר המחשב ומספר סידורי של המוצר

לחיצה על אישור ונקבל :



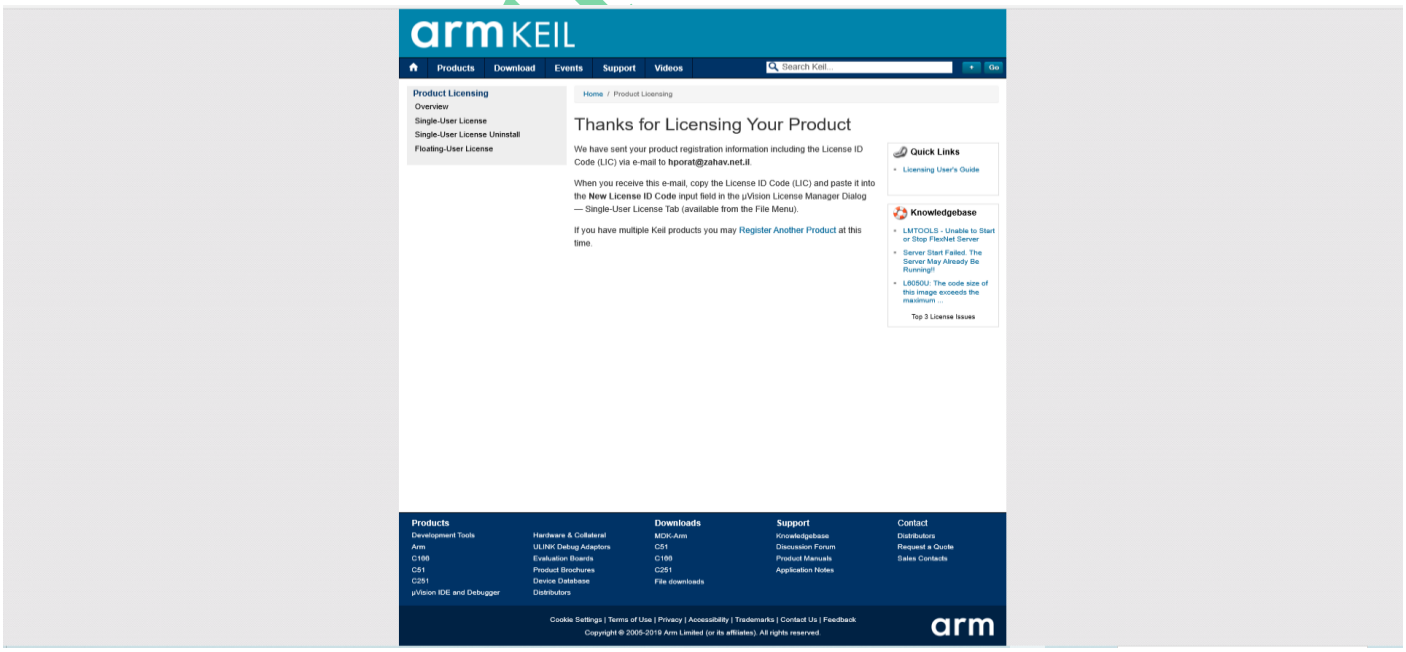
איור 4.7.13 : מסך הכנסת מספר סידורי של המוצר.

נמשיך ונרד עם המסך עד למטה. כאן יש למלא נתונים שאלון לגבי היישומים שצריך ,



איור 4.7.14: מסך הרשמה בסוף הכנסת המספר הסידורי של המוצר.

בסיום ההרשמה נקבל את המסך הבא :



איור 4.7.15 : סיום מסך הרשמה של המספר הסידורי של המוצר.

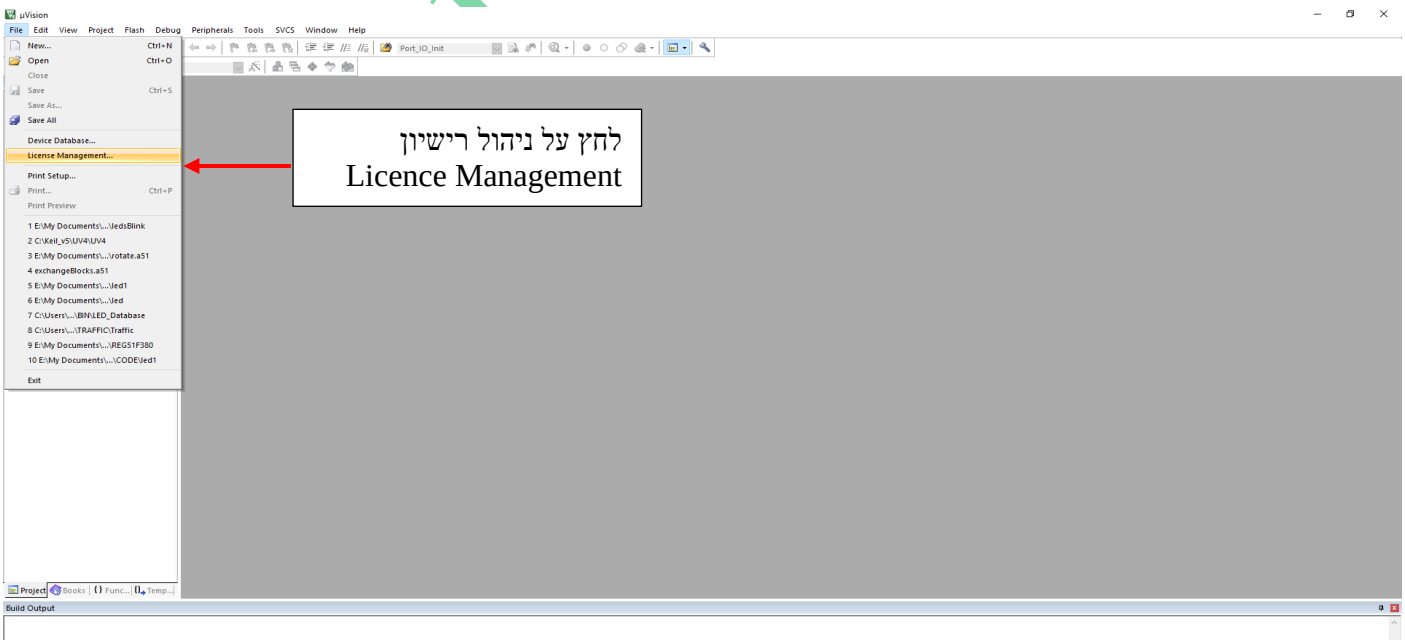
בסיום מסך ההרשמה נשלח אליך מייל מהתמיכה הטכנית של חברת Keil אל המייל שרשמת בסעיפים הקודמים שנראה כך (את ההדגשה בצבע ירוק וצהוב אני הוספתי לנוחיות ההסבר...):

```
Thank you for licensing your Keil product.  Your License ID Code (LIC) is printed below.
Print a copy of this e-mail to keep for your records.

PK51 Professional Developer's Kit
For Silicon Labs Devices Only
Support Ends 28 Feb 2022

PC Description       : HOME PC
Computer ID     (CID): CSMPQ-QZLJH

License ID Code (LIC): HN910-BRXK0-YNR7V-276IS-62JWP-CHLZ6


To activate your Keil product, copy the License ID Code (LIC) and paste it into the New
License ID Code input field on the Single-User License Tab in the uVision4 License Manager
Dialog (available from the File menu).

*** DO NOT REPLY TO THIS EMAIL:  For licensing problems or questions, please contact Keil
Technical Support.

Thank You,
Technical Support
```

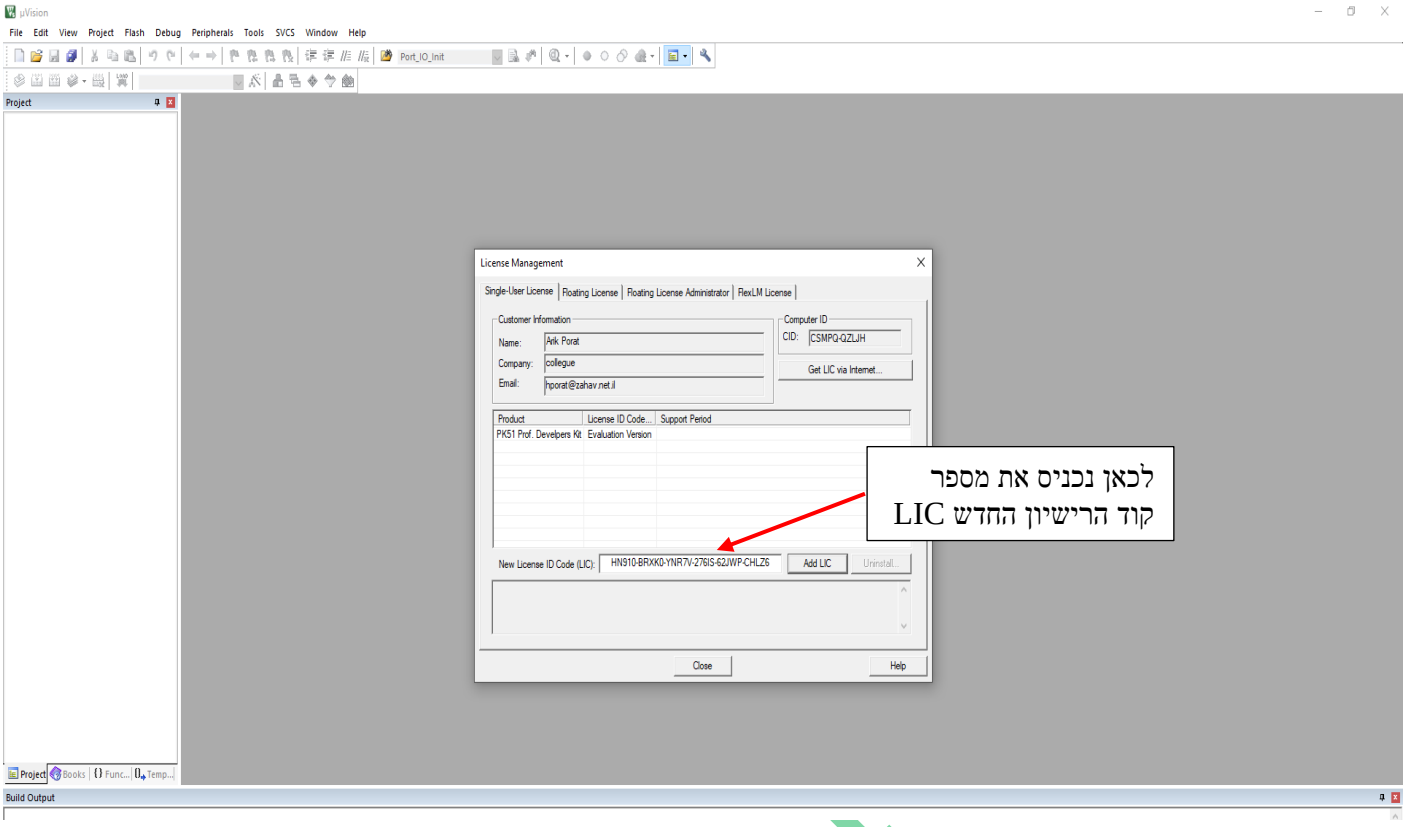
הרישיון שמקבלים הוא עד 28 Feb 2022 (28 בפברואר 2022) מודגש בצבע ירוק.

לסיום יש להכניס את מספר הרישיון החדש שמקבלים מודגש בצהוב : HN910-BRXK0-YNR7V-276IS-62JWP-CHLZ6

נחזור לתוכנה של **µVision5** ל License Management ונכניס את הרישיון החדש :



איור 4.7.16 : מסך ניהול רישיון כדי להכניס מספר קוד רישיון חדש License Id Code - LID



איור 4.7.17 : הכנסת קוד רישיון LIC

נלחץ על Close וכאן הסתיימה העלאת התוכנה ורישום חוקי למערכת.

4.8 עבודה עם KEIL בסביבת µVision

4.8.1 כללי

חברה שמוכרת תוכנה כלשהי מספקת סביבת עבודה הנקראת IDE – Integrated Development Environment – סביבת פיתוח משולבת - המורכבת מ 3 חלקים :

א. תוכנה לכתיבת התוכנית הנקרא **קובץ מקור** - Source File - בעזרת תוכנת עריכה כמו notepad, edlin או מעבד תמלילים כמו word , Qtext וכו'. קובץ המקור הוא קובץ טקסט (נקרא גם קובץ אסקי ASCII) . אם כותבים את התוכנית בשפת סי C אז הקובץ מסתיים עם C לדוגמה : myProg.c אם הקובץ הוא באסמבלי של 80x86 הוא מסתיים בשם asm לדוגמה : myProg.asm . אם הקובץ הוא באסמבלי של מיקרו בקר ממשפחת ה 51 הוא ייקרא : myProg.a51 . אם הקובץ הוא בויזואל בייסיק הוא ייקרא : myProg.vb וכך הלאה.

ב. תרגום התוכנית (קומפילציה). התוכנה המתרגמת את קובץ המקור שנוצר בסעיף הקודם לשתף מכונה שהיא אפסים ואחדים שהמעבד או המיקרו בקר מבין . אם התוכנה מתרגמת מקובץ אסמבלי לשפת מכונה אז התוכנה נקראת אסמבלר. אם היא מתרגמת קובץ מקור בשפת על או בשפה מונחית עצמים התוכנה המתרגמת נקראת קומפיילר או אינטרפרטר – INTERPRETER כמו בשפת פייתון. הקובץ שנוצר נקרא קובץ מטרה – Object File . קובץ המטרה מורכב מאפסים ואחדים שהם שפת המכונה של המעבד או המיקרו בקר. תוכנה שיודעת לתרגם אסמבלי של מעבד 80x86 יודעת לתרגם קבצים המסתיימים ב asm . תוכנה שמתרגמת קובץ שמסתיים ב a51 מתרגמת את קובץ המקור לשפת מכונה של מיקרו בקר ממשפחת ה 51 . התוכנה המתרגמת איננה יודעת מה מטרת התוכנית או מה המתכנן מתכוון שהתוכנית תעשה. היא בודקת שבקובץ המקור אין שגיאות תחביר (SYNTAX). לדוגמה שהפקודה שכתבנו היא mov al,bl ולא כתבנו move al,bl (התו e מיותר) . ליד כל שגיאה גם רשומה השורה שבה יש שגיאה כדי שלמתכנת יהיה קל למצא את השגיאה ולתקן אותה. אם אין שגיאות קומפילציה הקובץ מתורגם לקובץ מטרה OBJ. לפעמים מקבלים הודעה שאין שגיאות אבל יש warning – אזהרה – משהו לא הגיוני – לוגי – באחת השורות בתוכנית. במקרה כזה התוכנית אמנם מתורגמת לקובץ מטרה אבל מומלץ להבין את האזהרה ולתקן זאת במידת הצורך.

ג. הרצה של התוכנית עם תוכנת ניפוי – Debugger - המאפשרת תוך כדי הרצה גם לראות מצב זיכרון , מצב של משתנים , מצב רגיסטרים ועוד . בהמשך הפרק נבין את אפשרויות הניפוי.

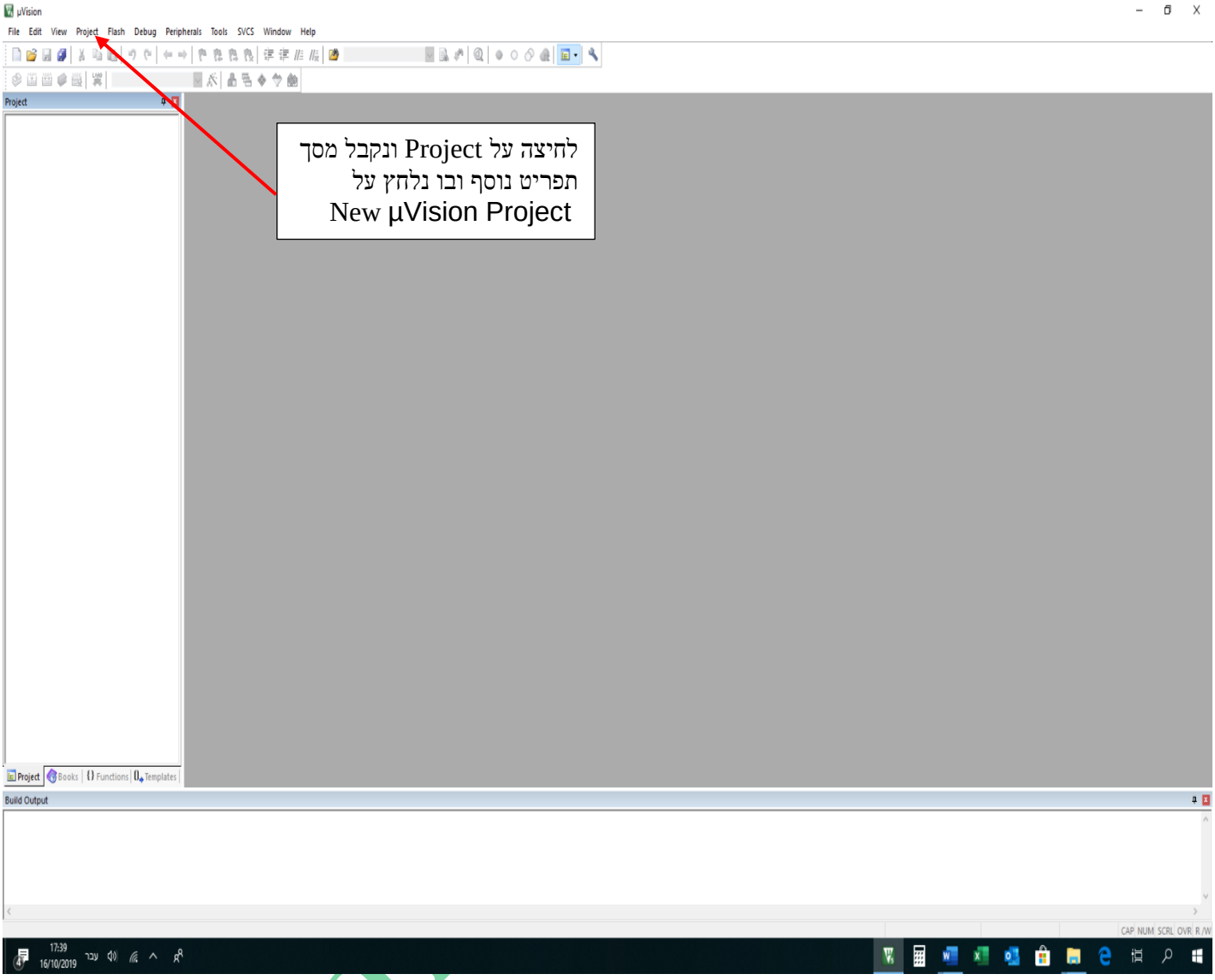
ד. בהרבה מקרים כאשר בונים פרויקט הוא מורכב ממספר קבצי מקור. כל קובץ מקור יכול להיכתב על ידי משתמש אחר . כל קובץ מקור כזה מתורגם לקובץ מטרה (OBJ) ולבסוף התוכנה מקשרת – link - בין כל קבצי המטרה השונים.

הערה: תוכנית ה **µVision** היא עשירה ביותר עם הרבה אפשרויות עבודה ולא ניתן ללמד "הכול" בפעם אחת. תרגול ועוד פעם תרגול ישפרו את הידע בתוכנה ויקנו בטחון למשתמש.

4.8.2 התחלת העבודה – כתיבת קובץ מקור ושמירתו בתוך פרויקט

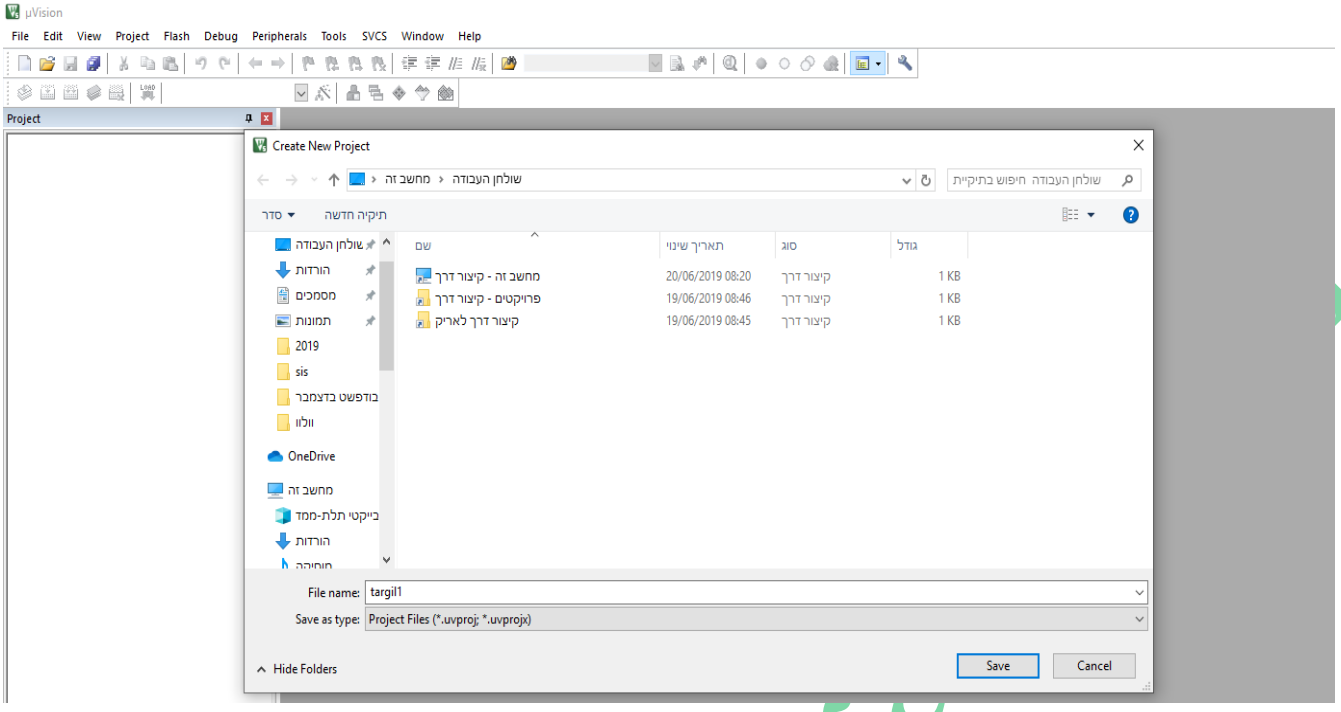
התוכנה של KEIL בסביבת **µVision** היא סביבת עבודה IDE והיא מתאימה למשפחות של מיקרו בקרים של יצרנים שונים . אנחנו נסביר את התוכנה ללא חקירה מעמיקה לכל האפשרויות אבל במידה מספקת להרצת תוכניות מעבדה ופרויקטים . הסבר באנגלית על תוכנת **µVision** גרסה 4 ניתן למצוא בקישור : https://www.keil.com/product/brochures/uv4.pdf

4.8.2.1 כאשר נלחץ על קיצור הדרך להפעלת התוכנה נקבל את המסך המתואר באיור 4.1 :



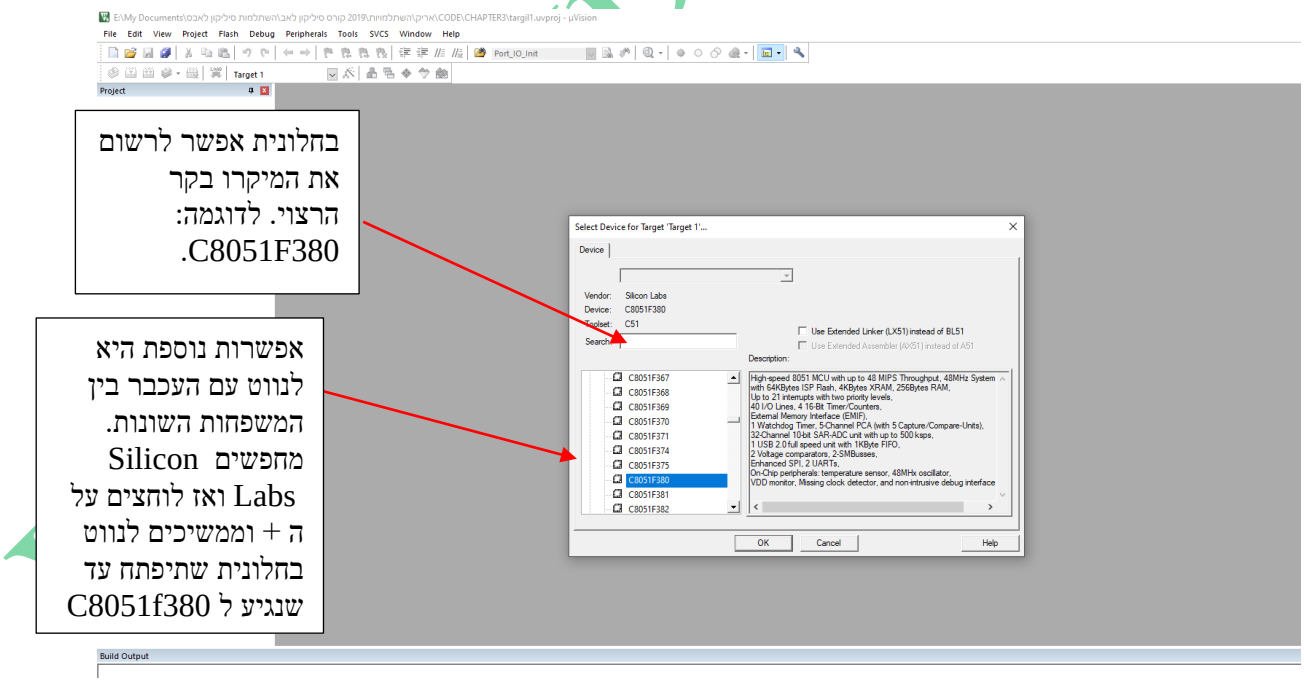
איור 4.1 : מסך התוכנה µV5 – KEIL

4.8.2.2 נפתח פרויקט ונקרא לו תרגיל 1 . במסך התפריט למעלה נלחץ על project ונקבל תפריט משנה ונלחץ על NEW µVISION PROJECT. ייפתח מסך שבו נקבע את מיקום קובץ הפרויקט שנכתוב. ראה איור 4.2: כאן שמרנו את הקובץ בשולחן העבודה וקראנו לו בשם targil1 . לחץ על SAVE . ברירת המחדל של סיומת הקובץ היא µVPROJ.



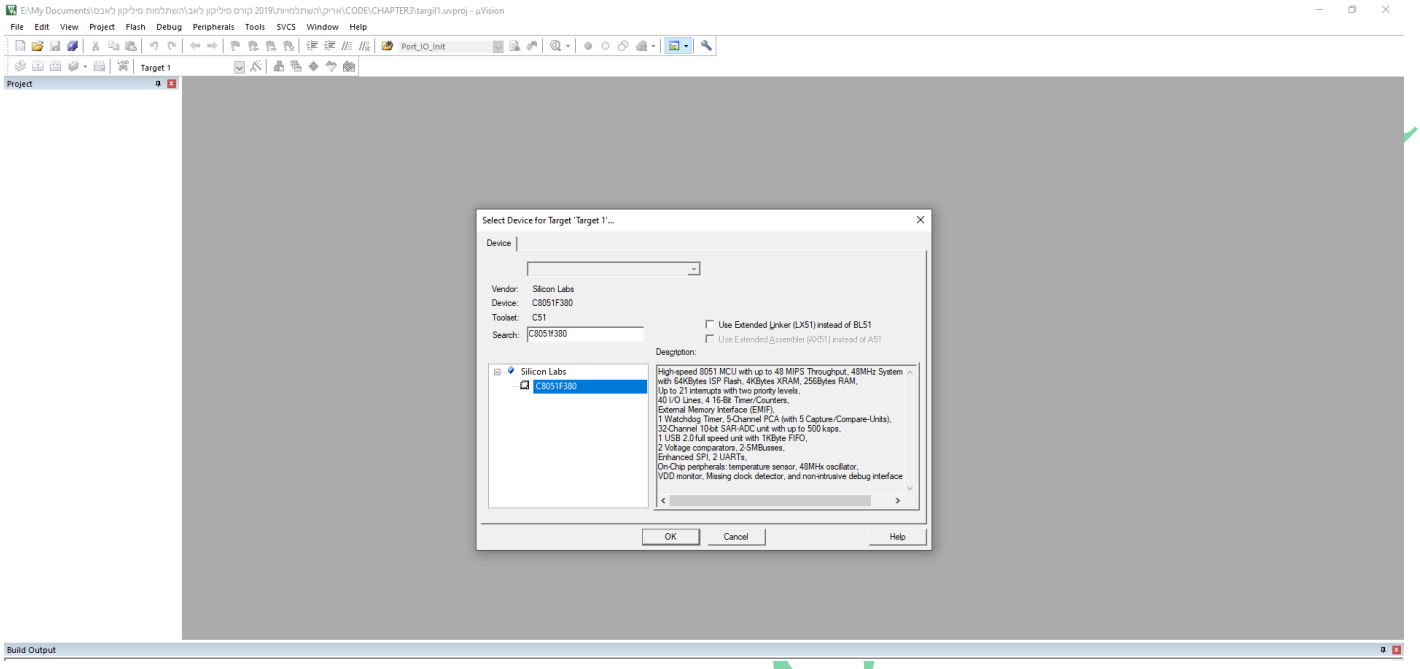
איור 4.2 : שם הפרויקט שפתחנו הוא targil1. µVPROJ .

4.8.2.3 לחיצה על save פתחה את מסך התפריט CPU שבאיור 3 שהוא מסך שבו בוחרים את המיקרו בקר שאיתו רוצים לעבוד.



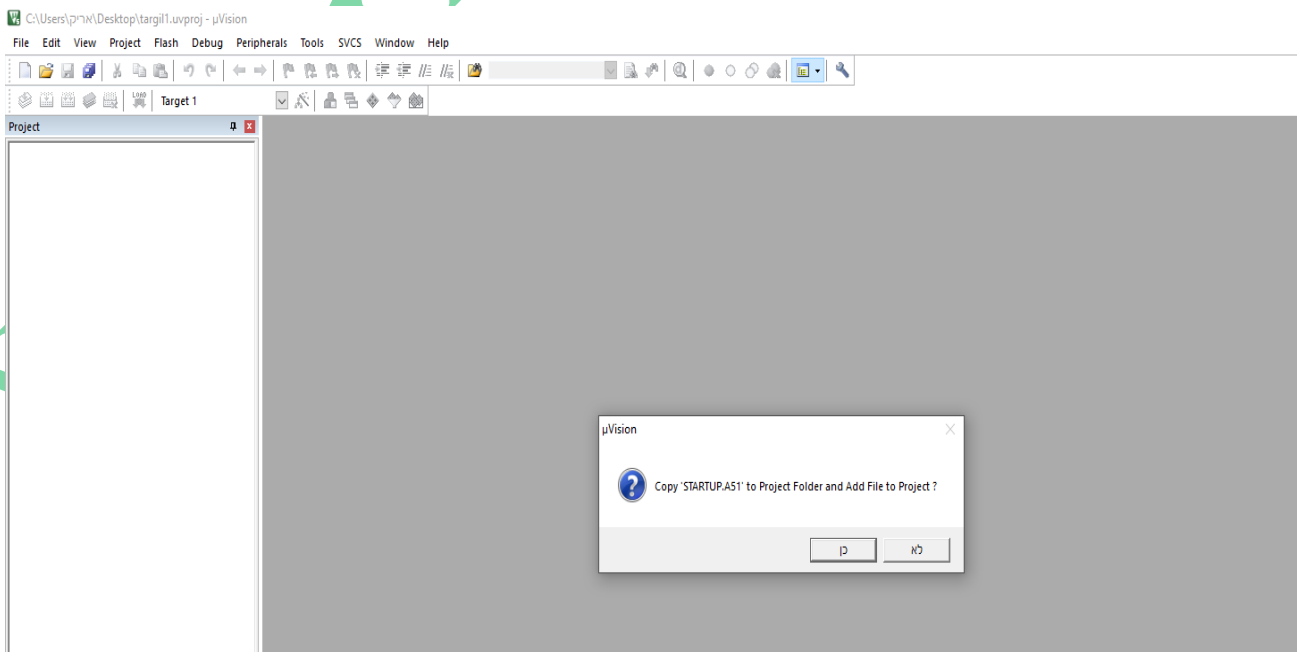
איור 4.3 : בחירת המיקרו בקר שאיתו רוצים לעבוד.

כאן יש לנו 2 אפשרויות. 1. לרשום בחלון ה SEARCH את שם המיקרו בקר , לדוגמה C8051F380 2. לעבור בצורה ידנית על כל היצרנים והרכיבים שלהם. לעבודה רגילה נרשום בשורת ה search : C8051F380 של חברת Silicon Labs ונקבל את המסך שבאיור 4 . נסמן עם העכבר את השורה שבה נמצא המיקרו בקר הרצוי (צבע כחול) ונקבל בצד ימין מספר מאפיינים שונים לגבי המיקרו בקר הזה. נלחץ על OK למטה ונעבור לסעיף הבא:



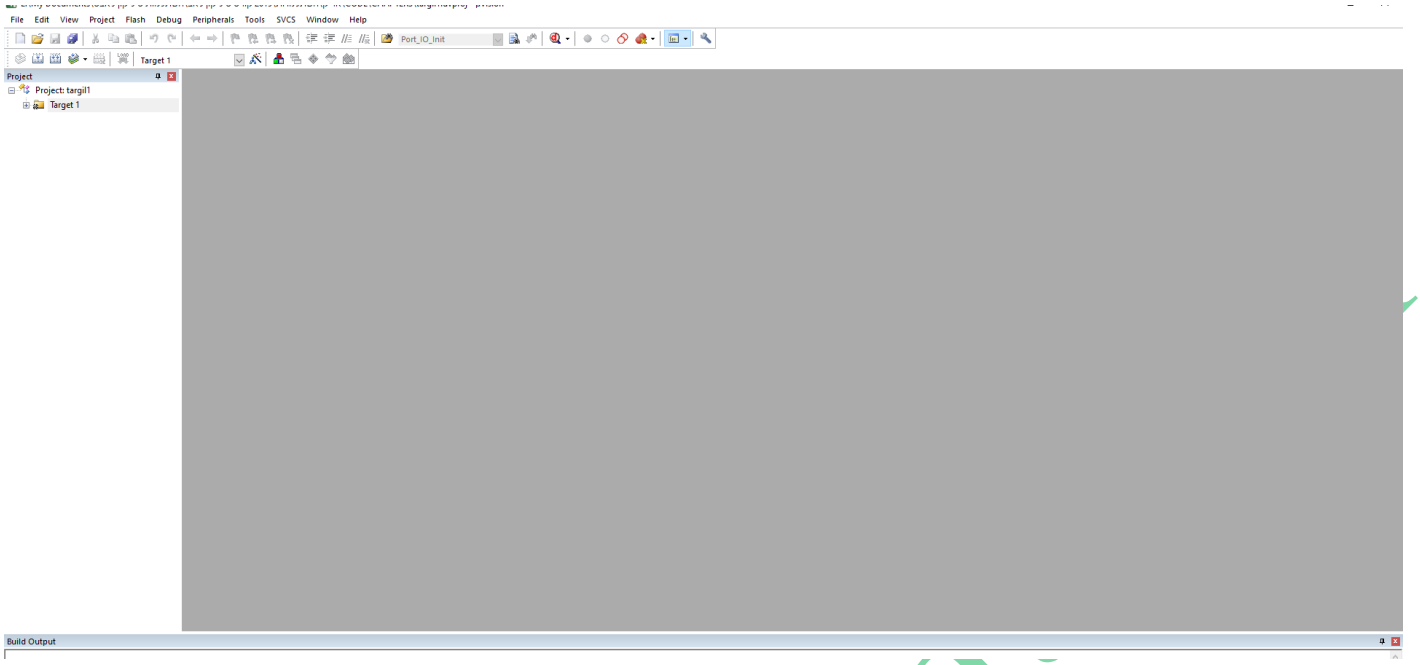
איור 4.4 : בחירת המיקרו בקר הרצוי.

4.8.2.4 במסך זה שואלים האם רוצים להוסיף קובץ startup.a51 . הכוונה היא שהקומפיילר יירשום תוכנית אתחול שבה הוא רושם בכתובת 0 קפיצה לכתובת שהיא אחרי הכתובות של וקטורי הפסיקה. לדוגמה לכתובת 100H



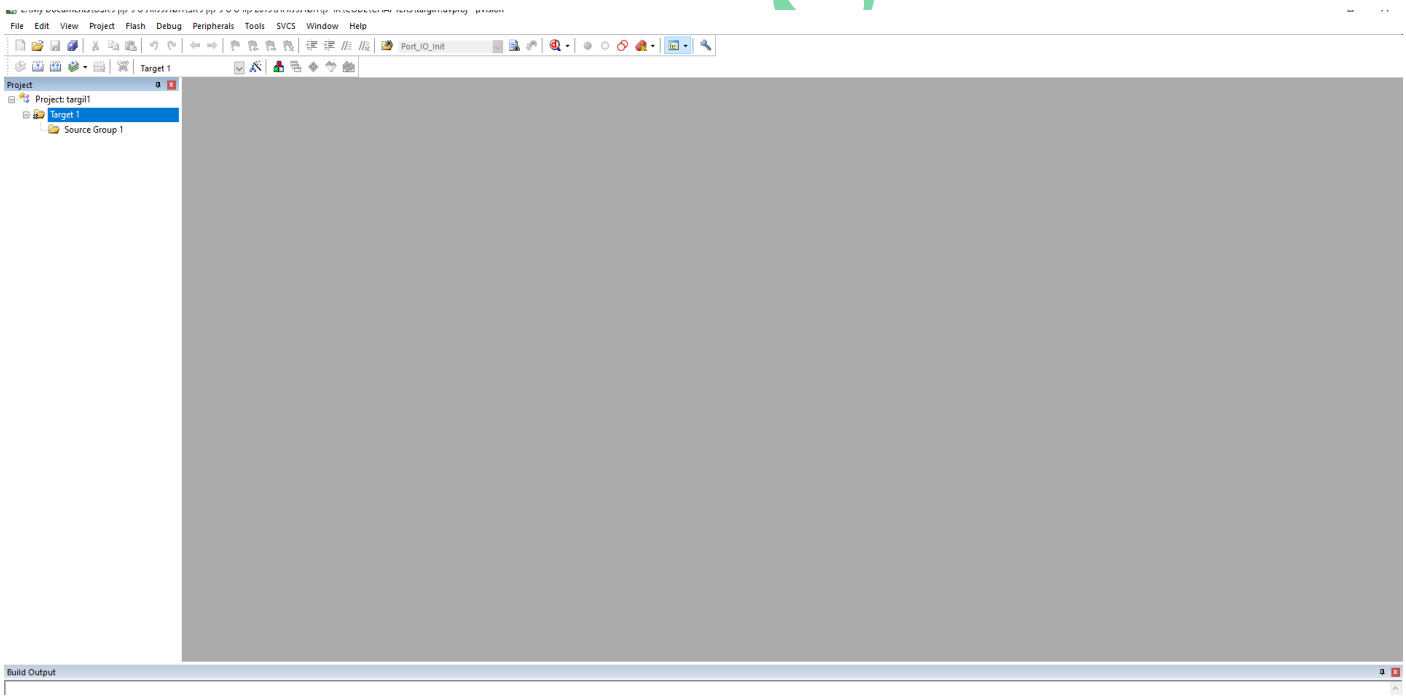
איור 4.5 : מסך בחירה האם להוסיף startup.a51

כאן ניתן ללחוץ על "כן" או "לא". נניח שנלחץ על "לא" ואז נקבל את המסך שבאיור 6 :



איור 4.6 : מסך הפרויקט

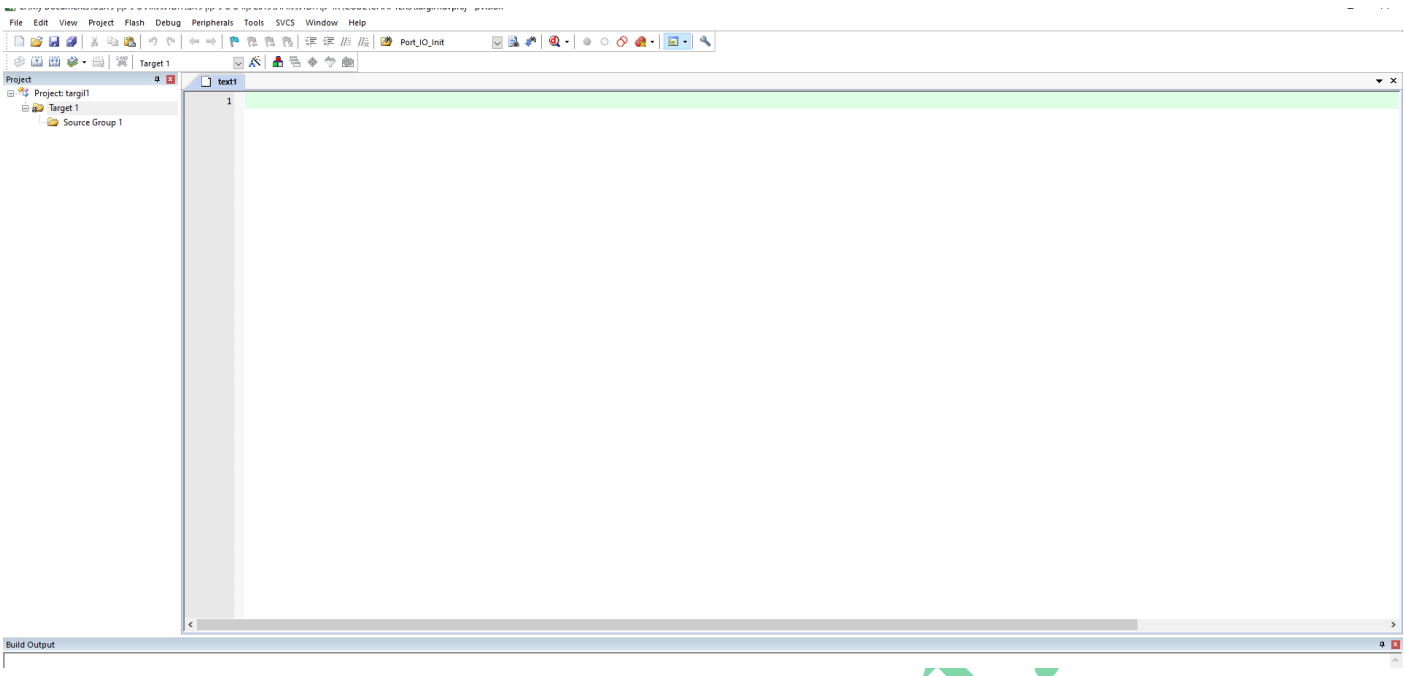
בצד שמאל קיבלנו את מסך הפרויקט. הוא נקרא target 1 . מתחתיו יש את Target 1 לחיצה על ה + משמאלו ונקבל את המסך באיור 4.7.



איור 4.7 : מסך הפרויקט ובו Source Group 1 ריק.

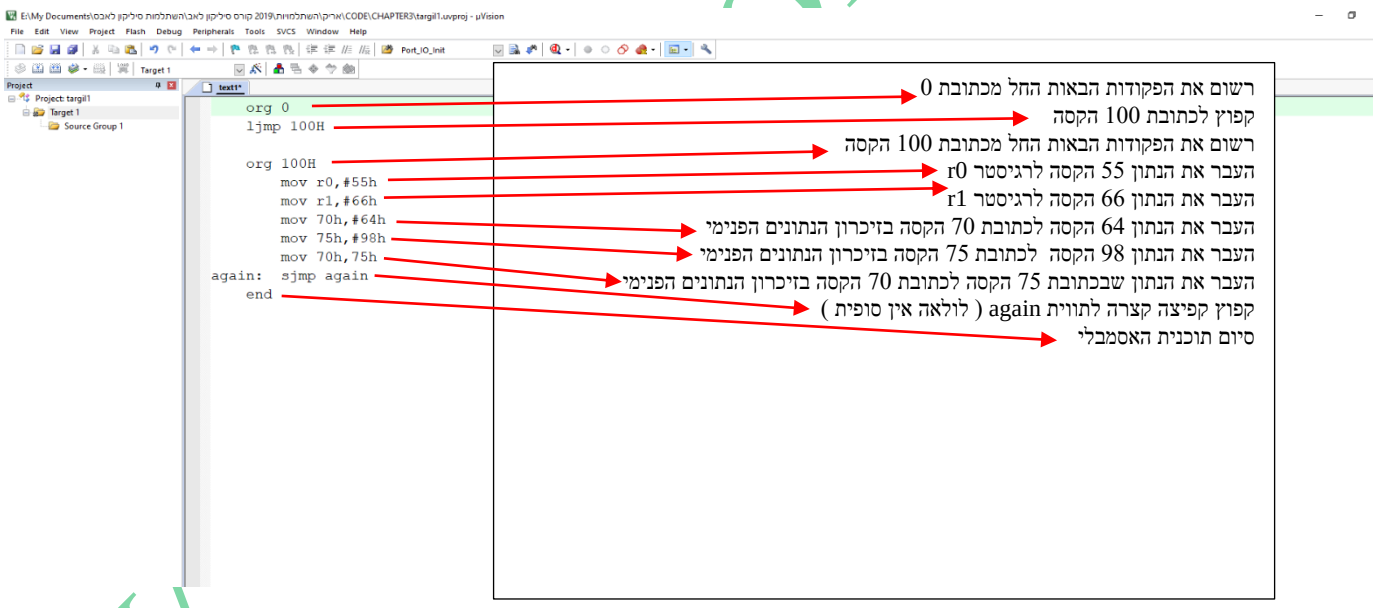
מתחת ל Source Group 1 לא רשום כלום כי עדיין לא הכנסנו לפרויקט שלנו קובץ תכנית שאותו נרצה להריץ.

נכתוב כעת תוכנית שאותה נרצה להריץ. לשם כך נבחר במסך התפריט שלמעלה FILE ואז במסך החדש שנפתח נלחץ על NEW ונקבל את המסך שבאיור 4.8 עם מסך TEXT1. במסך זה נכתוב את התוכנית.



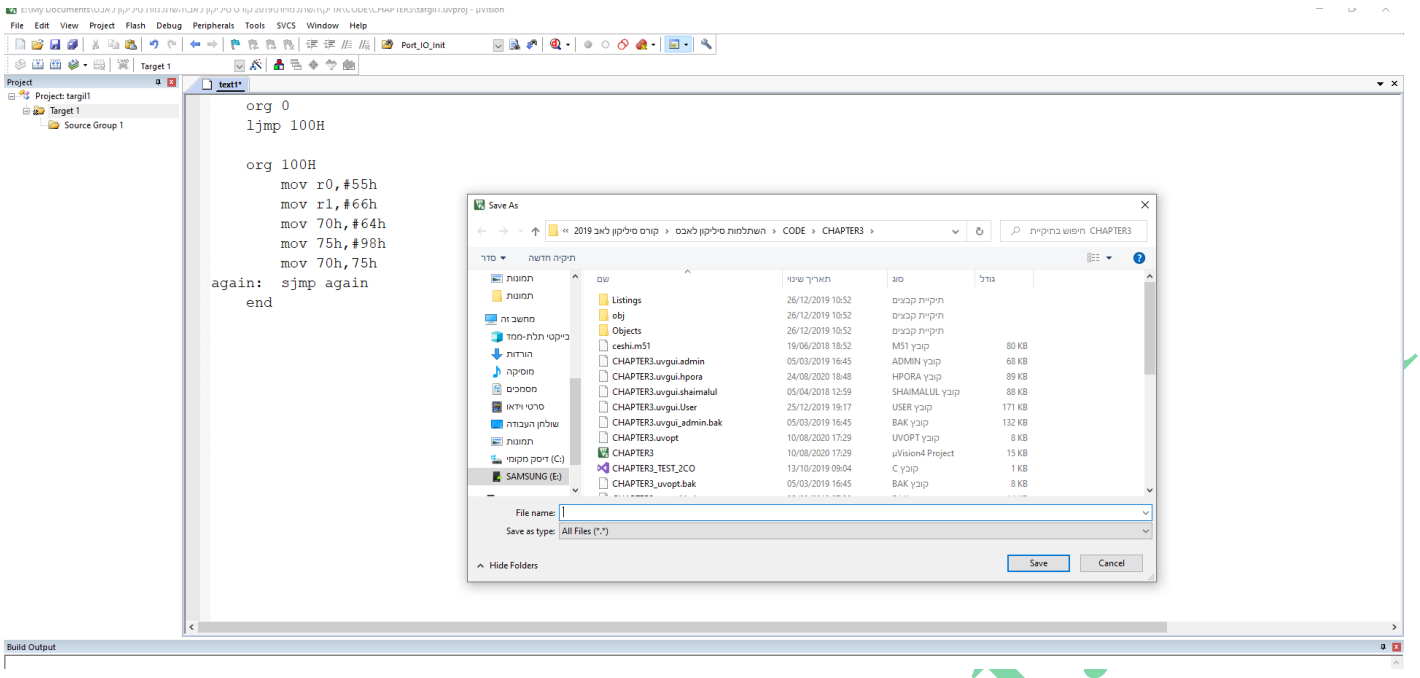
איור 4.8 : מסך כתיבת התוכנית

4.8.2.5 נכתוב את תוכנית האסמבלי הראשונה שבאיור 4.9 :



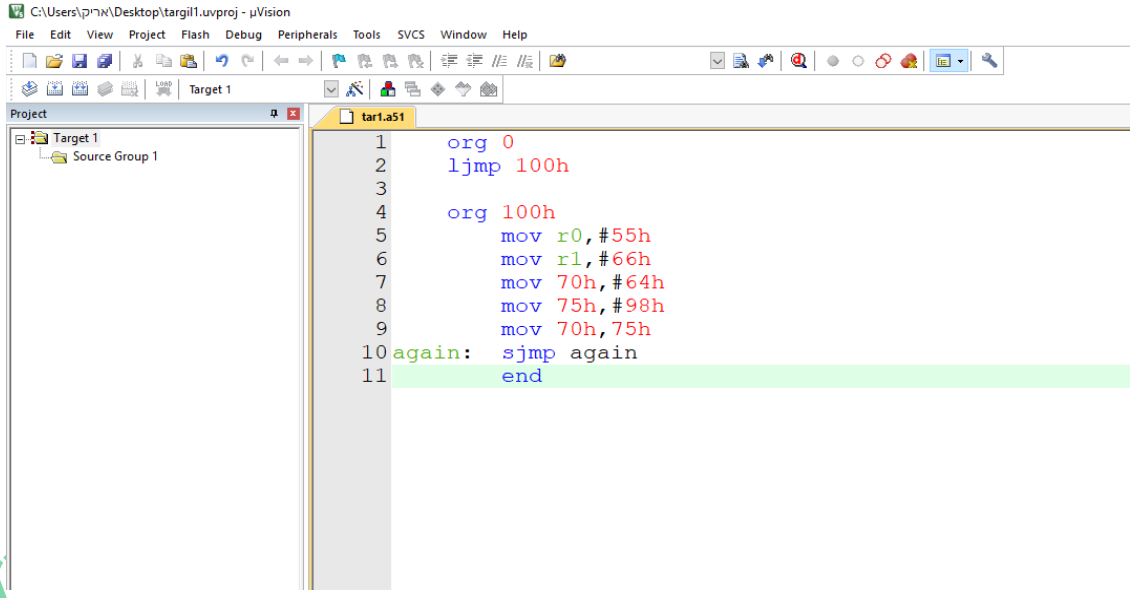
איור 4.9 : כתיבת תכנית ראשונה

4.8.2.6 את התוכנית שרשמנו נשמור כקובץ אסמבלי. לשם כך נלחץ על התפריט FILE ובמסך שייפתח נלחץ על SAVE AS ונקבל את המסך שבאיור 4.10 :



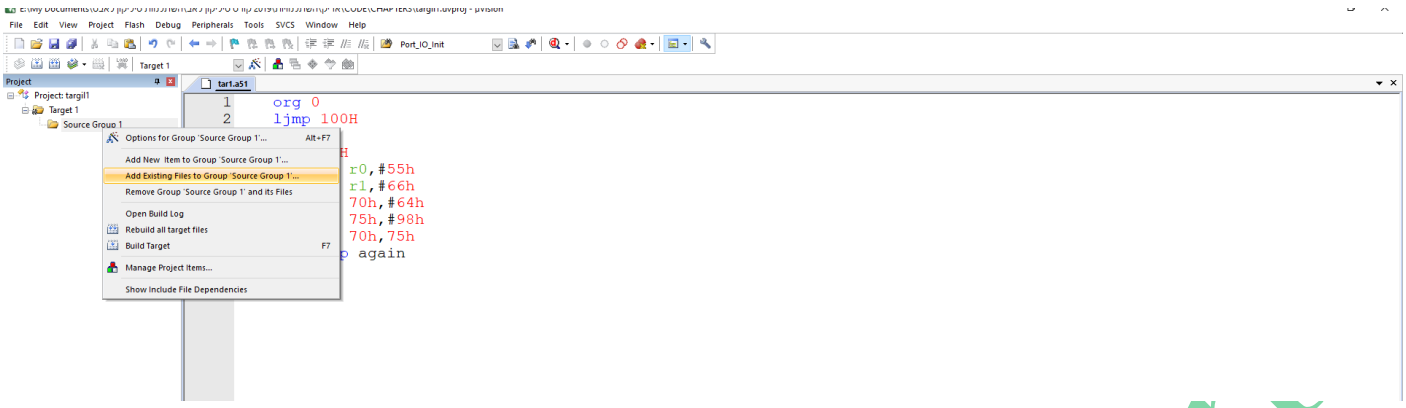
איור 4.10 : שמירת הקובץ tar1.a51

יש לשים לב לרשום את הקובץ עם סיומת a51 כדי שהקומפיילר ידע לתרגם אותו. אנחנו קראנו לקובץ tar1.a51 . נלחץ ע SAVE ונראה ששם הקובץ השתנה מ TEXT 1 ל tar1 והפקודות רשומות בצבע כמו באיור 4.11 :



איור 4.11 : התוכנית tar1.a51 .

4.8.2.7 כעת יש לצרף את הקובץ tar1.a51 לפרויקט שלנו. לשם כך נביא בעזרת העכבר את הסמן אל Source Group 1 (בחלון הפרויקט בצד שמאל למעלה) ונלחץ על המפסק הימני בעכבר. נקבל את המסך שבאיור 4.12:



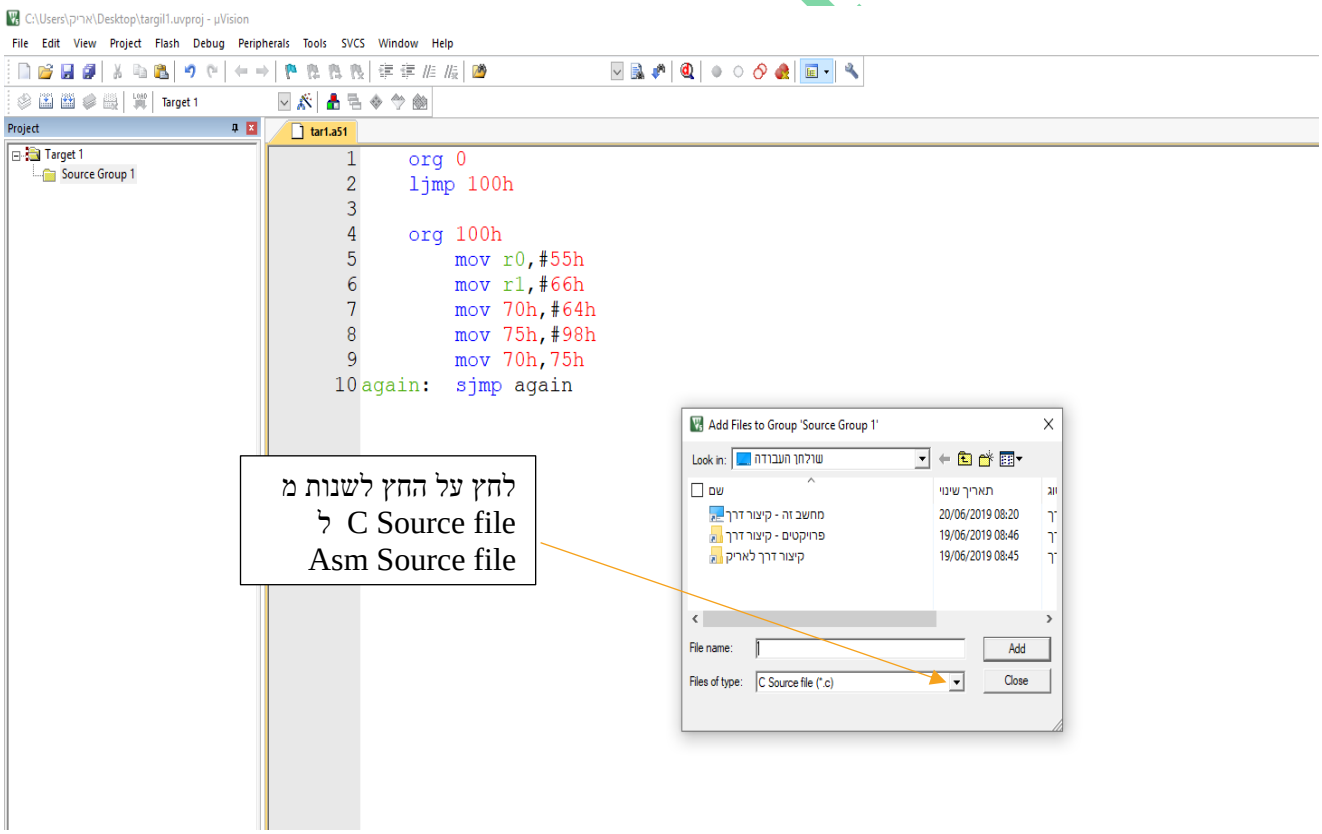
איור 4.12 : הוספת הקובץ tar1.a51 לפרויקט.

נלחץ על ...'Add Existing file to Group 'Source Group 1 ואז נעבור למסך באיור 13 .

4.8.2.8 במסך זה נבחר את הקובץ שנרצה להוסיף לפרויקט. ניעזר באיור 4.13 .

נקליק עם העכבר פעמיים על ...'Source Group 1' (או נלחץ עם המפסק הימני בעכבר על Source Group 1 ואז

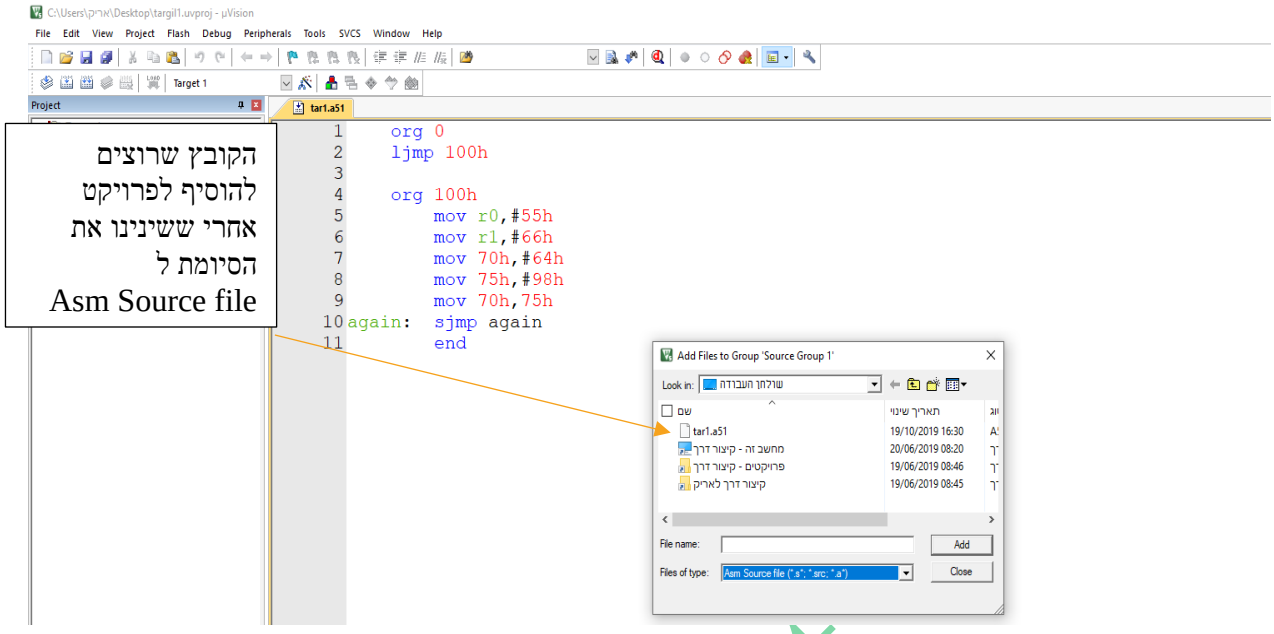
נקבל את מסך 1 נלחץ בעכבר בתפריט החדש שייפתח על : "Add existing files to group "source Group 1 ונקבל :



איור 4.13 : הוספת קובץ tar1.a51 לפרויקט .

לא רואים את הקובץ tar1.a51 כי כרגע מוצג בחלון קבצי C (כרגע אין וכאשר נכתוב בשפת C נקבל את הקבצים שנרשום) .

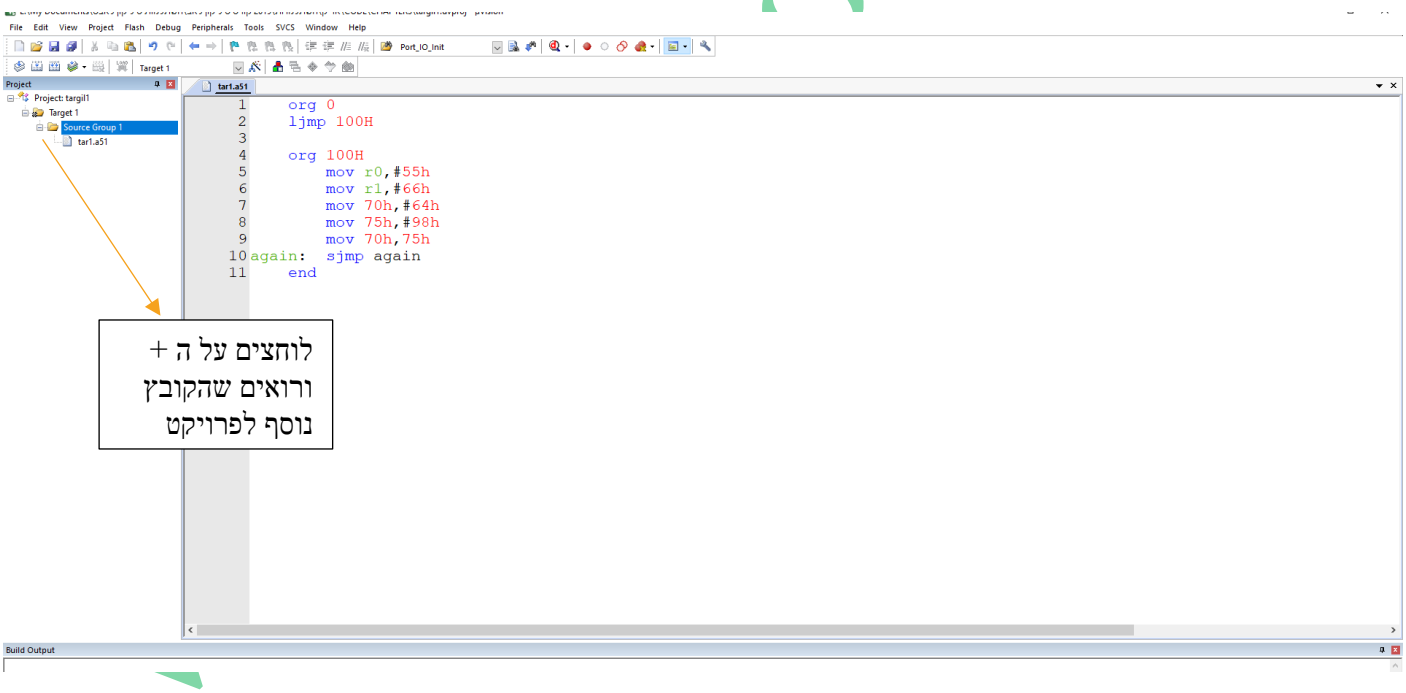
נשנה את החלון ל Asm Source file ונקבל את איור 4.14 שבו רואים את הקובץ tar1.a51 .



איור 4.14 : בחירת הקובץ להוספה לפרויקט.

נלחץ בעזרת העכבר על הקובץ tar1.a51 פעמיים ואז נראה את המסך באיור 4.15 שבו רואים שהקובץ התווסף לפרויקט.

באיור 4.15 רואים שהקובץ התווסף לפרויקט.

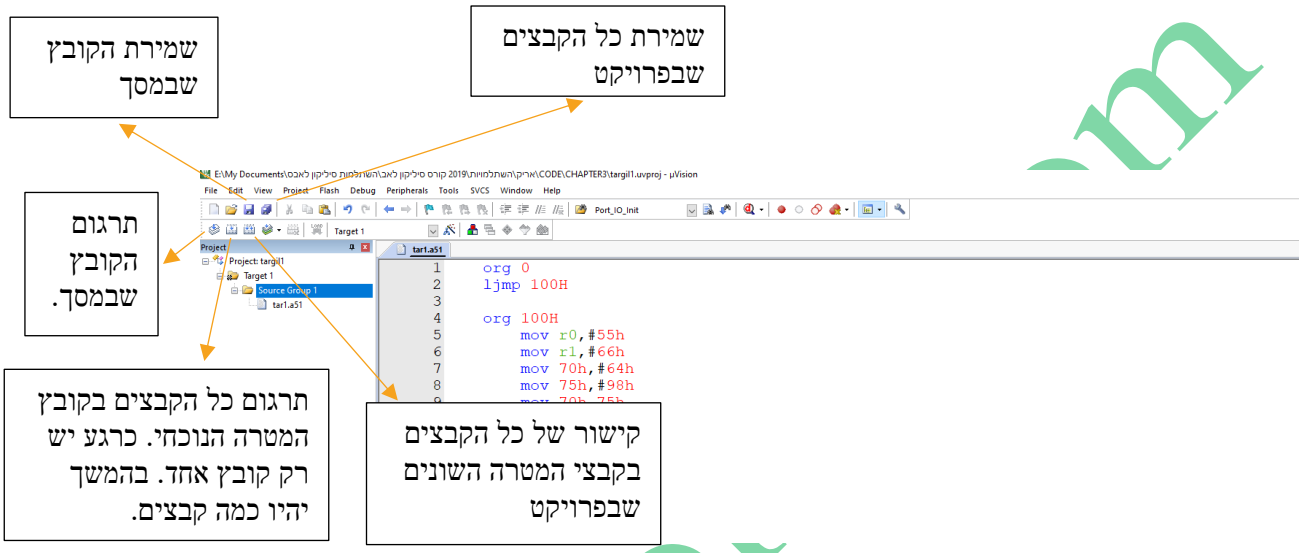


איור 4.15 : לוחצים על ה + ליד Source Group 1

עד כאן כתבנו תוכנית והוספנו אותה לפרויקט. עכשיו נבדוק שאין לנו טעויות ונריץ את התוכנית.

4.9 תרגום התוכנית – קומפילציה

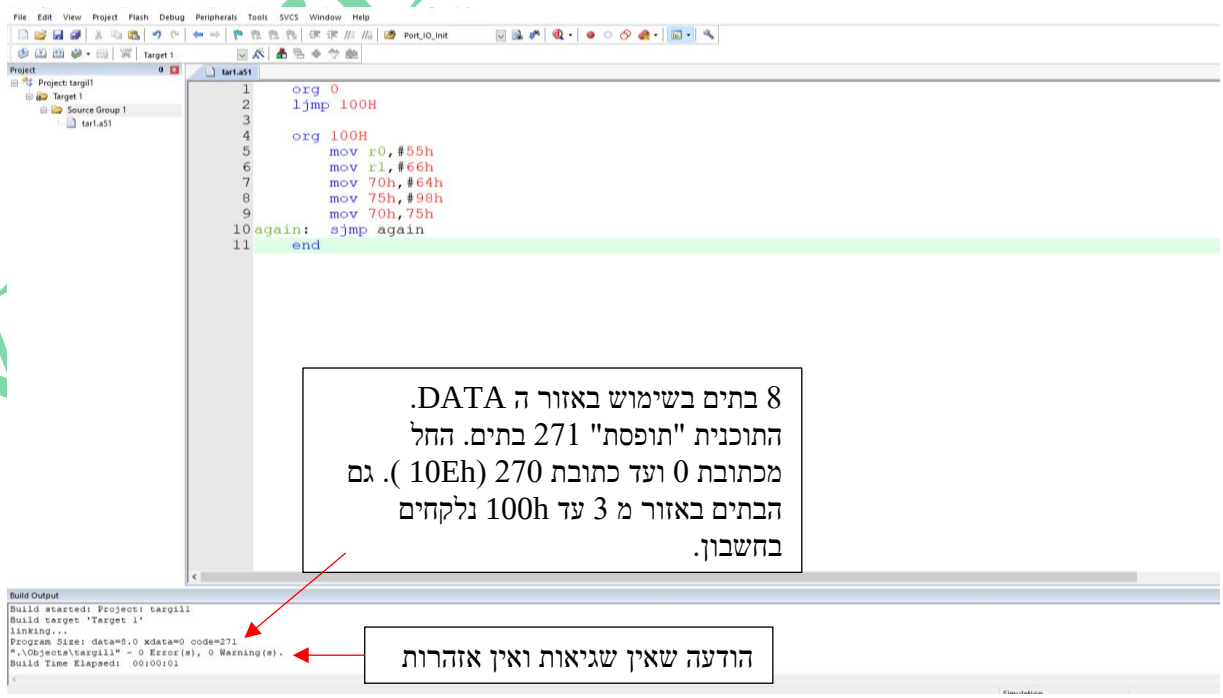
כאשר כותבים תכנית משתמשים בתוכנית שמקשרת בין המשתמש, המקלדת והתצוגה (מסך). התוכנית נקראת מעבד תמלילים או עורך מסך. הקובץ נשמר כקובץ טקסט ואנחנו קראנו לו tart1.a51 . את קובץ הטקסט יש לתרגם לאפסים ואחדים שהמיקרו בקר מכיר – שפת מכונה. התוכנית המתרגמת נקראת אסמבלר (עבור שפה עילית זה נקרא קומפיילר). עד עכשיו כתבנו את התוכנית . בפרק זה נתרגם את התוכנית ונבדוק שאין טעויות תחביר. ניעזר באיור 4.16 :



איור 4.16 : תפקיד הצלמיות בתפריט

באיור רשמנו מה עושה כל צלמית שבתפריט. המלצה שלנו היא לעבור עם העכבר על כל אחת מ 5 הצלמיות אחת אחרי השנייה בסדר הבא : א. שמירת הקובץ שבמסך. ב. שמירת כל הקבצים שבפרויקט. ג. תרגום של הקובץ שבמסך. ד. תרגום כל הקבצים בקובץ המטרה הנוכחי ה. תרגום כל הקבצים בקבצי המטרה השונים.

4.9.1 במידה ואין שגיאות נקבל את המסך שבאיור 4.17 :



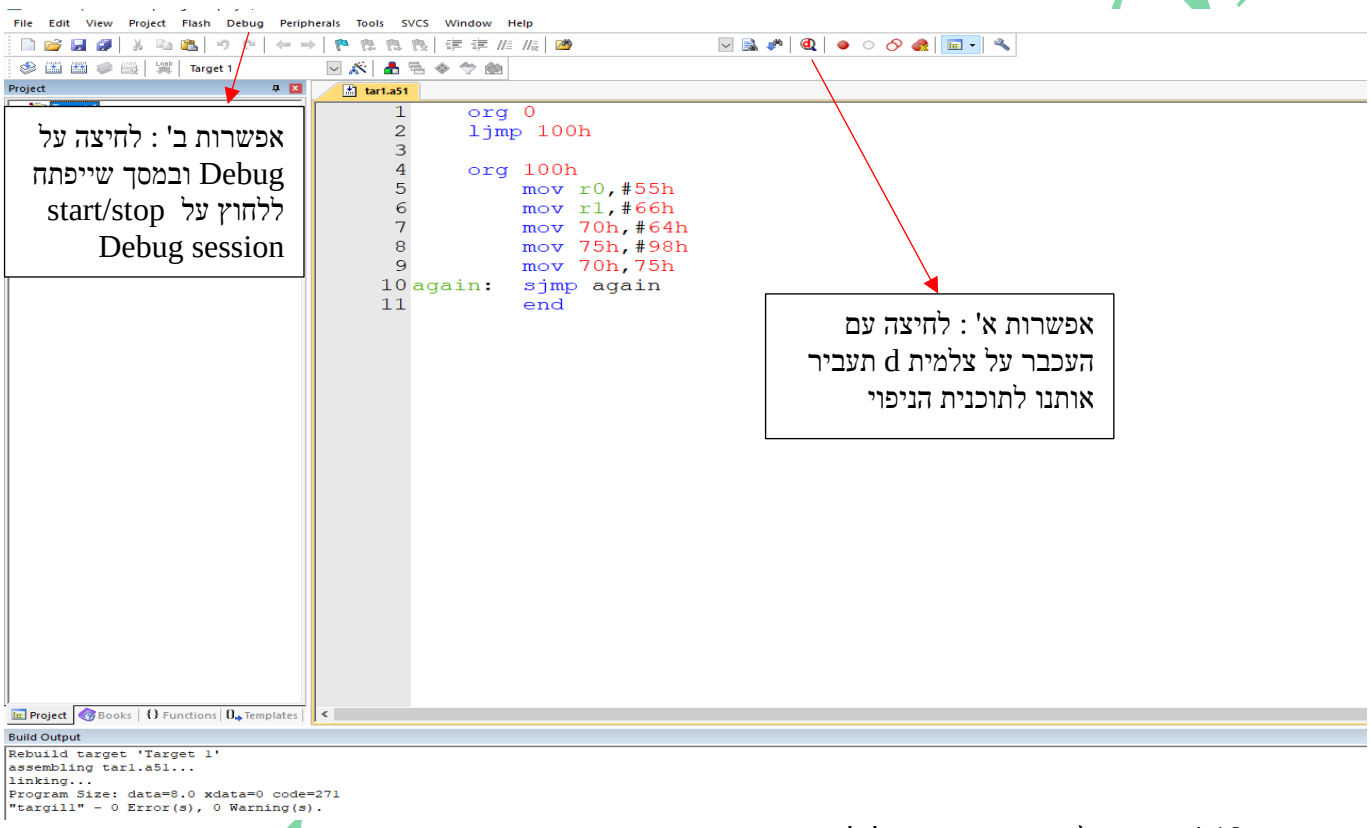
איור 4.17 : המסך המתקבל אחרי קומפילציה במידה ואין שגיאות

המסך המתקבל באיור 4.17 הוא כאשר אין שגיאות בתחביר. זה עדיין לא אומר שהתוכנית תעשה את מה שרצינו. כרגע עברנו רק את שלב כתיבת התוכנית ותרגומה. עכשיו יש להריץ אותה ולראות האם היא עושה את מה שרצינו.

4.10 הרצה עם תוכנית ניפוי

בפרקים ב' ו ג' כתבנו את התוכנית ותרגמנו אותה. כעת נבדוק מה התוכנית עושה. התוכנית שכתבנו איננה תוכנית "רצינית" וכל מטרתה היא להראות איך עובדים עם תוכנית הניפוי.

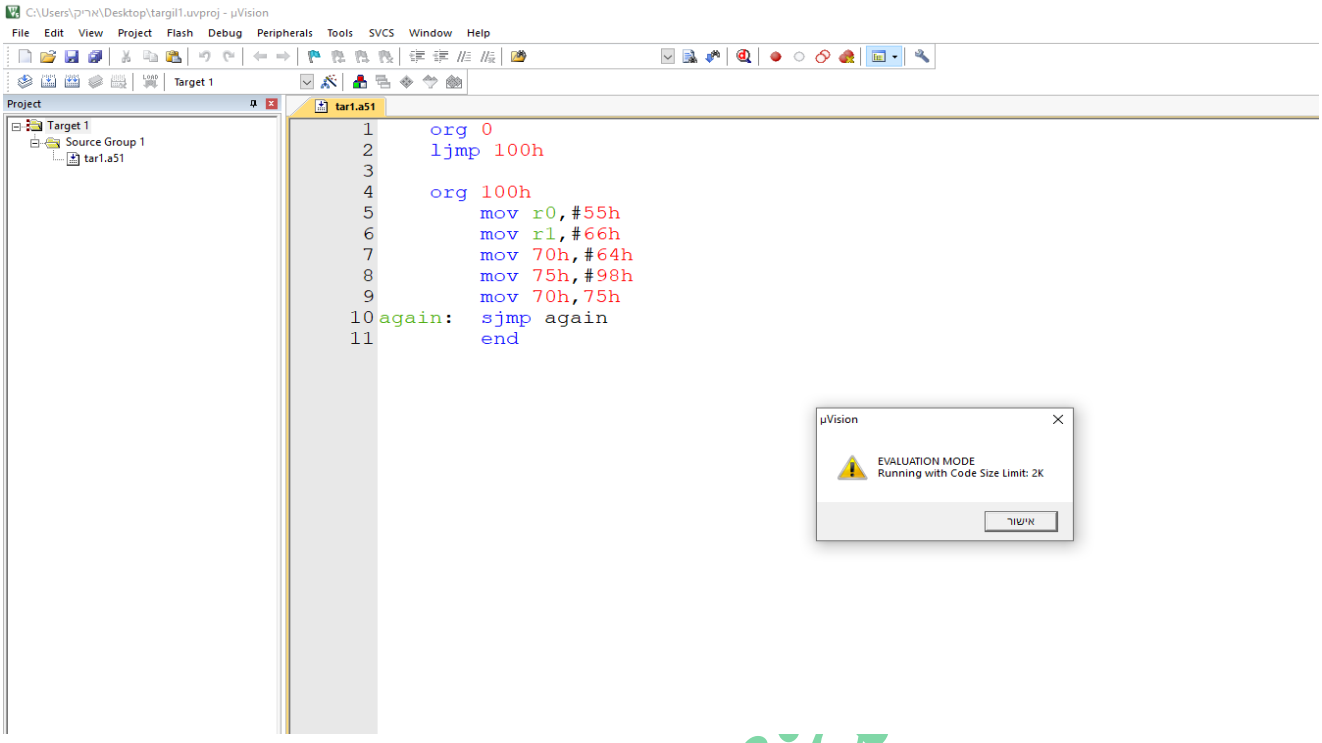
א. ניעזר באיור 4.18 כדי לפתוח את תוכנית הניפוי debug :



איור 4.18 : כניסה לתוכנית הניפוי debug

ב. המסך שנקבל אחרי הפעלת אחת מהאפשרויות באיור 4.18 נראה באיור 4.19 .

ג. קיבלנו מסך שמודיע לנו שהתוכנה של KEIL שאנחנו משתמשים היא להערכה והיא מוגבלת לעד 2K בתים של קוד. "בעברית פשוטה" זה אומר שעד כאן התוכנית היא ללא כסף. עבור תוכניות המעבדה זה מספיק. לפרויקט רציני זה לא מספיק וזה ידרבן אותנו לקנות גרסה מלאה מהחברה.



איור 4.19 : מסך המגביל את התוכנית שלנו לעד 2K בתים של קוד

ד. לחיצה על אישור במסך 4.19 ונקבל את המסך שבאיור 4.20 .

באיור ניתן לראות את האזורים השונים שבמסך הניפוי הכוללים את אזור הקוד, אזור האסמבלי, אזור הרגיסטרים של המעבד וערכם, אזור המחסנית ואזור זיכרון 1 .

באזור הרגיסטרים נראים הערכים בכניסה לתוכנית. מצב הרגיסטרים בכניסה לתוכנית מדמה פעולת reset . רואים שכל הרגיסטרים התאפסו חוץ ממצביע המחסנית SP שקיבל את הערך 7 . הבנק הנוכחי שעובדים איתו הוא 0 כי רגיסטר ה PSW = 0 . ברגיסטר זה ישנן 2 ביטים שבעזרתם קובעים עם איזה בנק עובדים (RS0 = RS1 = 0) .

ה. יציאה ממסך הניפוי וחזרה לתוכנית העריכה/כתיבה על ידי לחיצה חוזרת על הצלמית d שבעזרתה הגענו למסך הניפוי.

אזור הקוד: בכתובות 0, 1, 2 רשום בהתאמה 02 01 00 . הקוד 02 הוא קוד הפעולה של ljmp
והמספרים 0100 אומרים מהי הכתובת אליה רוצים לקפוץ (0100 הקסה). זהו תרגום של
המשפטים org 0 ו ljmp 100h שבתוכנית האסמבלי שלנו לשפת מכונה .
בכתובת 3, 4... רשום 00 שכוונתו NOP שאומר No OPeration (ללא פעולה)

אזור תוכנית
האסמבלי
שכתבנו

כרגע מראה את אזור
הרגיסטרים. ניתן ללחוץ על
Project (משמאלו) ולראות
את מבנה הפרויקט

אזור המחסנית

אזור זיכרון 1 (בהמשך
נראה איך משתמשים בו)

מסך 4.20 : מסך תוכנית הניפוי debug

כל אחת מהפעולות של הניפוי יכולות להתבצע בשני אופנים. או על ידי מקש של המקלדת או בעזרת צלמית בתפריט (נסביר באיור 21). באיור 4.21 נראים הצלמיות לפעולות הניפוי ותפקידן מוסבר מתחת לאיור.

איור 4.21 מראה את הצלמיות עבור כל פעולת ניפוי ללא צורך לזכור את המקשים המבצעים פעולות אלו .

Run F5
to next
breakpoint

Stop- Halt
program execution

Reset – Resets
the
microcontroller
CPU or simulator
while debugging

Step F11
Step one line

Step over F10
Step over the
current line

Step out (Ctrl+f11)
Step out of the
current function

Run to cursor line (Ctrl+f10)
Run to the current cursor line

חלון ה
disassembly
הצג/הסתר

חלון הפקודה
הצג/הסתר

חלון
הרגיסטרים
הצג/הסתר

חלון הסמלים
symbols
הצג/הסתר

חלון קריאה
למחסנית
הצג/הסתר

חלון
עקיבה/משתנים
watch
הצג/הסתר

חלון זיכרון
הצג/הסתר

חלון אנלייזר
הצג/הסתר

חלון טורי
או serial
printf
הצג/הסתר

איור 4.21 : פעולות ניפוי בסיסיות

ו. 5 פעולות ניפוי בסיסיות (עם דוגמאות בהמשך):

1. **Step one line** – צעד שורה אחת - הרצת התוכנית שורה אחרי שורה בעזרת מקש **F11** . במצב זה התוכנית מבצעת את השורה בה נמצא הסמן ועוצרת בשורה הבאה. המשתמש יכול לבדוק מה קרה בזיכרון או ברגיסטרים (או במשתנים) אחרי שהפקודה התבצעה. אם יש פקודה כמו lcall delay (קרא לפרוצדורה הרחוקה delay) עוברים לשורה הראשונה של הפרוצדורה ועוצרים בשורה זו. אם כותבים בשפה עילית וקוראים לפונקציה עוברים לשורה הראשונה של הפונקציה ועוצרים בה.
2. **Step over** - צעד נוסף בעזרת המקש **F10**. אם יש פקודה כמו lcall delay (קרא לפרוצדורה הרחוקה delay) אז התוכנית עוברת לפרוצדורה delay , מבצעת את כל פקודות הפרוצדורה וכאשר התוכנית מגיעה לפקודה ret - חוזרים לשורה אחרי הפקודה lcall delay ואז התוכנית נעצרת והמשתמש יכול לבדוק מה קורה בזיכרון התוכנית, ברגיסטרים או במשתני התוכנית.
3. **Step out** - יציאה בעזרת צמד המקשים **CTRL+F11** . מצב זה עוזר כאשר נמצאים בפרוצדורה (או בפונקציה) ורוצים לסיים אותה ולחזור לשורה אחרי השורה שקראה לפרוצדורה או לפונקציה .
4. **Run to Cursor Line** – רוץ לשורה של הסמן - בעזרת המקשים **CTRL+F10** . במצב זה ניתן להעביר את הסמן לשורה רצויה ואז בעזרת המקשים CTRL+F10 התוכנית תרוץ וכאשר תגיע לשורה שבה נמצא הסמן היא תעצור.
5. **Break Point** - נקודת עצירה – בעזרת מקש **F9.** כאשר תוכנית רצה ורוצים שכאשר היא תגיע לשורה מסוימת היא תעצור בשורה זו אז מביאים את הסמן לשורה הרצויה ולוחצים על מקש F9 . השורה נצבעת בצבע ירוק עם סימון של עיגול בצבע אדום משמאל למספר השורה . בכל פעם שהתוכנית מגיעה לשורה המסומנת היא תעצור והמשתמש יוכל לבדוק את מצב הרגיסטרים, זיכרון וכו'.

דוגמה: נתונה התוכנית :

כתובת	**הפקודה**
110h	Mov a,#64h
112h	Lcall delay ; הפרוצדורה delay בכתובת 200h
115h	mov r0,#55h
⋮	
200h	mov r5,#98h
202h	djnz r5,$
204h	ret
205h	

נניח שאנחנו בכתובת 112h בפקודה lcall delay .

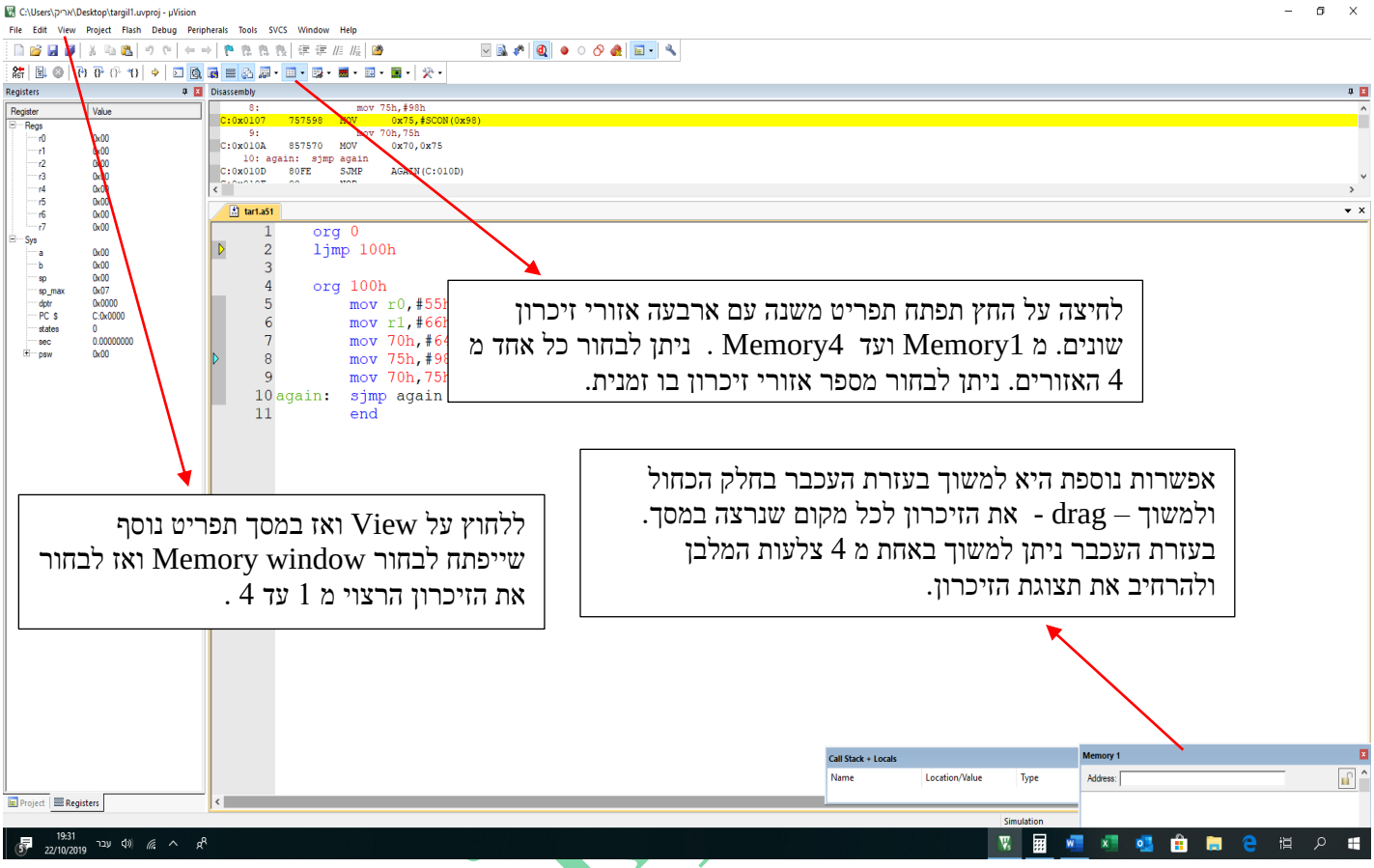
אם נלחץ F11 (step one line) נעבור לשורה 200h ונעצור בשורה זו.

אם נלחץ F10 (step over) התוכנית תעצור בשורה 115h **אחרי שביצעה את הפרוצדורה** delay .

נניח שאנחנו בשורה 202h . בשורה זו מבצעים חיסור של 1 מרגיסטר r5 והיות והוא איננו 0 חוזרים לשורה 202h . אם נלחץ על F11 או F10 נבצע פקודה זו 98h פעמים (252 עשרוני) עד שמגיעים לפקודה ret . לעומת זאת אם נלחץ על **CTRL+F11** (Step out) התוכנית תעבור לשורה 115h ותעצור בשורה זו.

ז. הצגת אזורי זיכרון הנתונים הפנימי והחיצוני

כאשר נמצאים בתוכנית הניפוי ורוצים להציג אזור בזיכרון הפנימי או החיצוני ניתן לעשות זאת ב 3 דרכים. ניעזר באיור 4.22 :



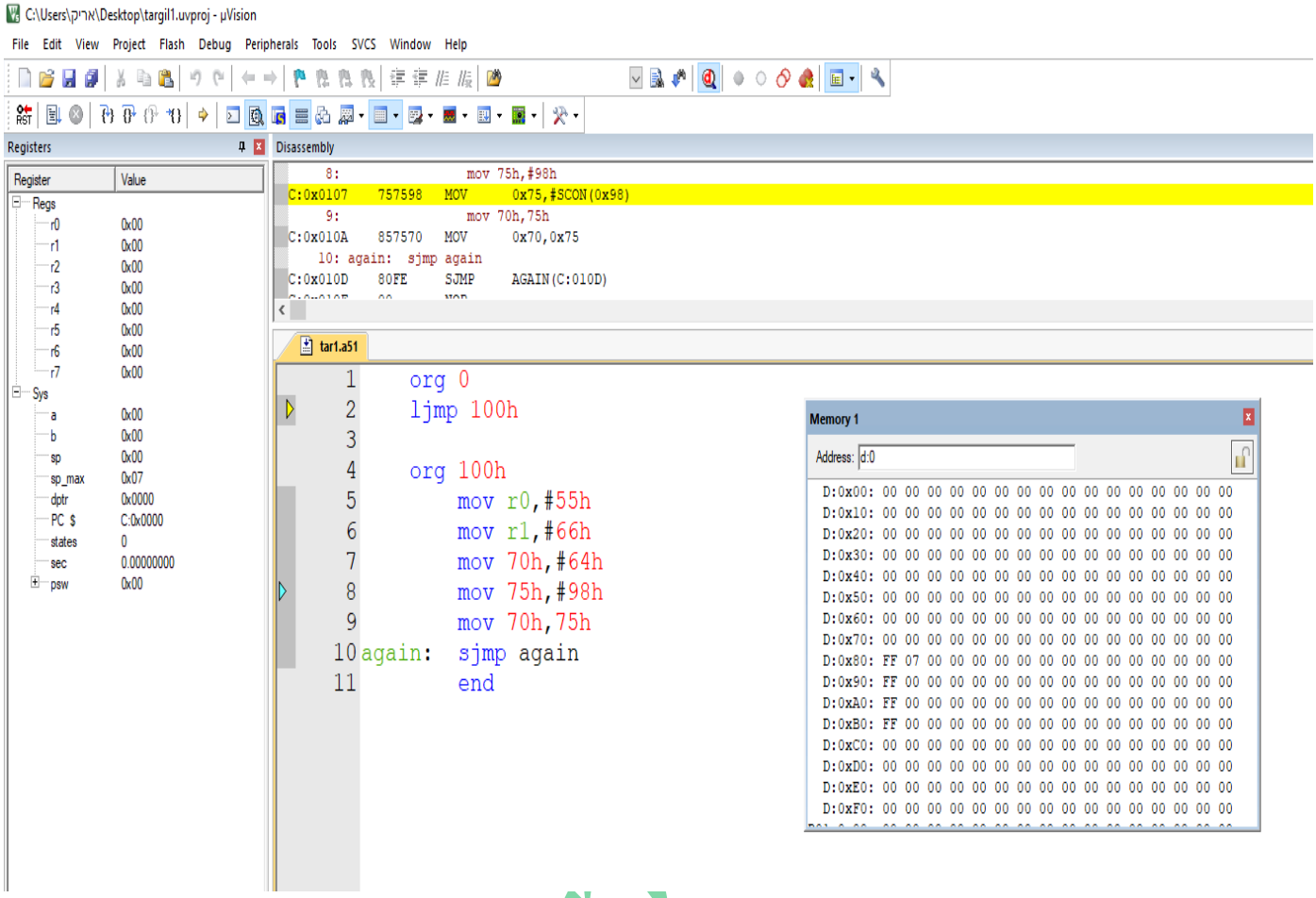
איור 4.22 : 3 אפשרויות להצגת אזור זיכרון

לאחר שבוחרים אזור זיכרון נוכל לבחור איזה זיכרון רוצים לראות, כלומר האם פנימי או חיצוני ומאיזו כתובת להציג את הזיכרון. איור 4.23 מראה כיצד מציגים את אזור הזיכרון הרצוי. באיור בחרנו את Memory1 (באחת מ 3 האפשרויות) , הבאנו את הזיכרון למיקום הרצוי במסך והרחבנו את התמונה של הזיכרון בצורה נוחה לנו.

כדי לראות אזור RAM נתונים פנימי נרשום : D: Address . D מראה RAM - Data נתונים פנימי ו Address היא הכתובת שממנה רוצים להתחיל לראות . (לא לשכוח את הנקודתיים ביניהם). הכתובת היא עשרונית או אם רושמים h אז היא הקסה.

כדי לראות את זיכרון ה RAM החיצוני XRAM יש לרשום x:Address .

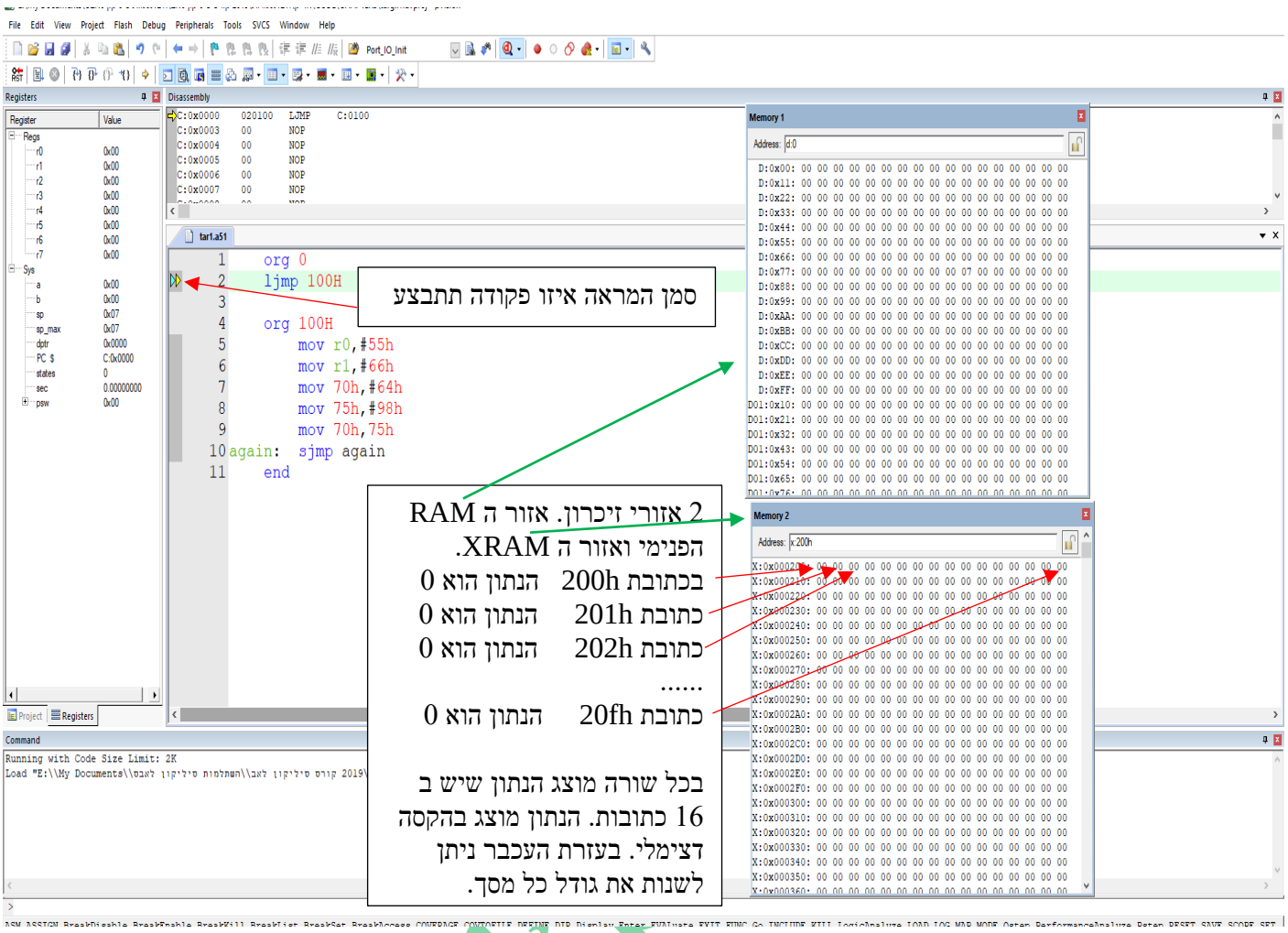
באיורים הבאים מראים את הזיכרונות.



איור 4.23: הצגת אזור זיכרון הנתונים הפנימי

בחלון ה Address רושמים D:0 שאומר שרוצים לראות את אזור ה Data , אזור הנתונים הפנימי , החל מכתובת 0 . הנתונים המוצגים הם בהקסה דצימלי כאשר בצד שמאל מופיעה הכתובת ומימין לה הנתונים בכתובות אלו. בדוגמה כאן בשורה הראשונה רשום D:0x00 ואז מוצגים 16 נתונים (0 בדוגמה כאן) המייצגים את הנתונים בכתובות מ 0 ועד כתובת 0FH . בשורה השנייה רשומה הכתובת D:0x10 (שהיא 16 בעשרוני) ומימין לה 16 הנתונים (0 בדוגמה כאן) שבכתובות מ 10H ועד 1FH וכך הלאה.

כדי להציג את אזור הזיכרון החיצוני החל מכתובת 200H יש לרשום X:200h . ה X מציין שמדובר ב eXternal .
איור 4.24 מראה 2 אזורי זיכרון. האחד פנימי החל מכתובת 0 והשני חיצוני החל מכתובת 200H .



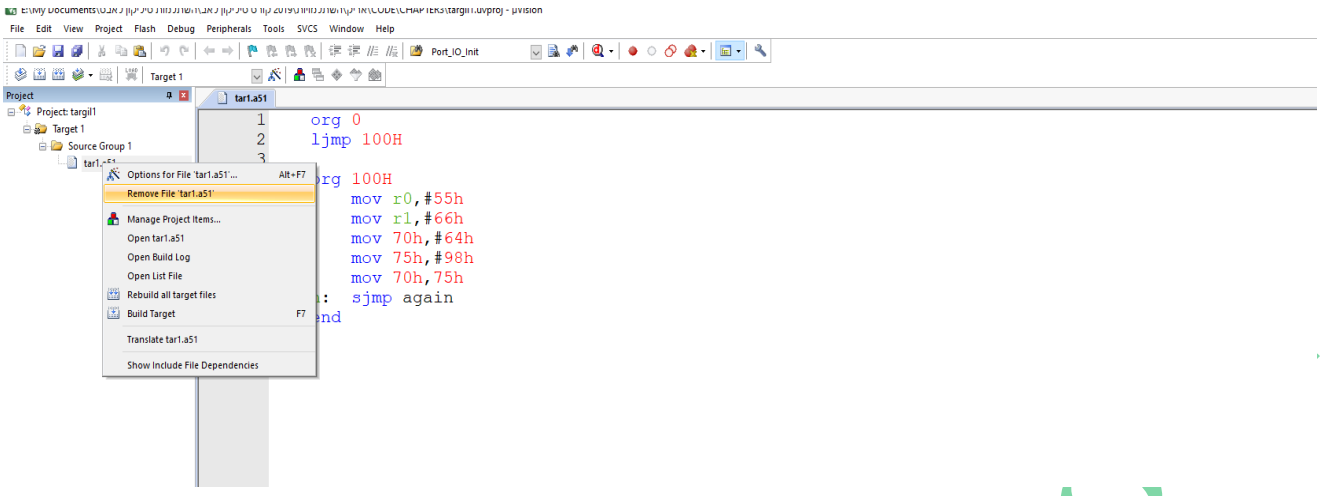
איור 4.24 : 2 אזורי תצוגה. פנימי מכתובת 0 וחיצוני מכתובת 200H .

כעת ניתן להריץ את התוכנית באחת מתוך 5 אפשרויות הניפוי ולראות מה קורה בזיכרון תוך כדי הרצת התוכנית.

4.11 דוגמה של תוכנית עם הרצה וניפוי

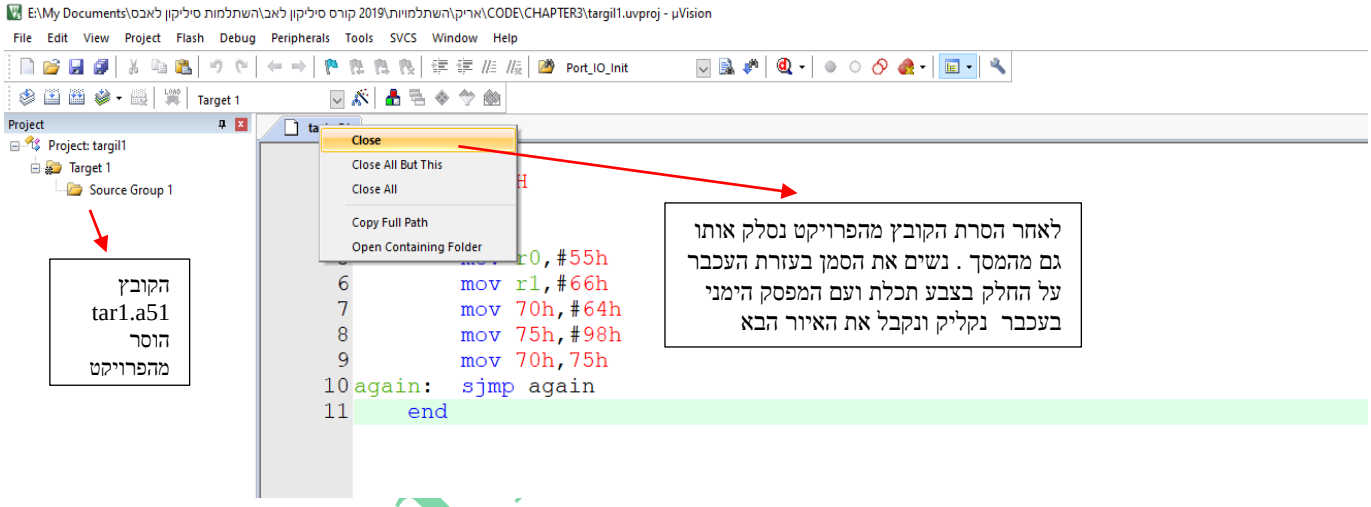
נרשום תוכנית שממלאת בלוק כתובות באזור ה RAM הפנימי החל מכתובת 50h ועד 6fh בנתון ההולך וגדל מ 0 ועד 1fh בהתאמה. לאחר מכן נעביר את הנתון לאזור ה XRAM מכתובת 200h ועד 21fh .

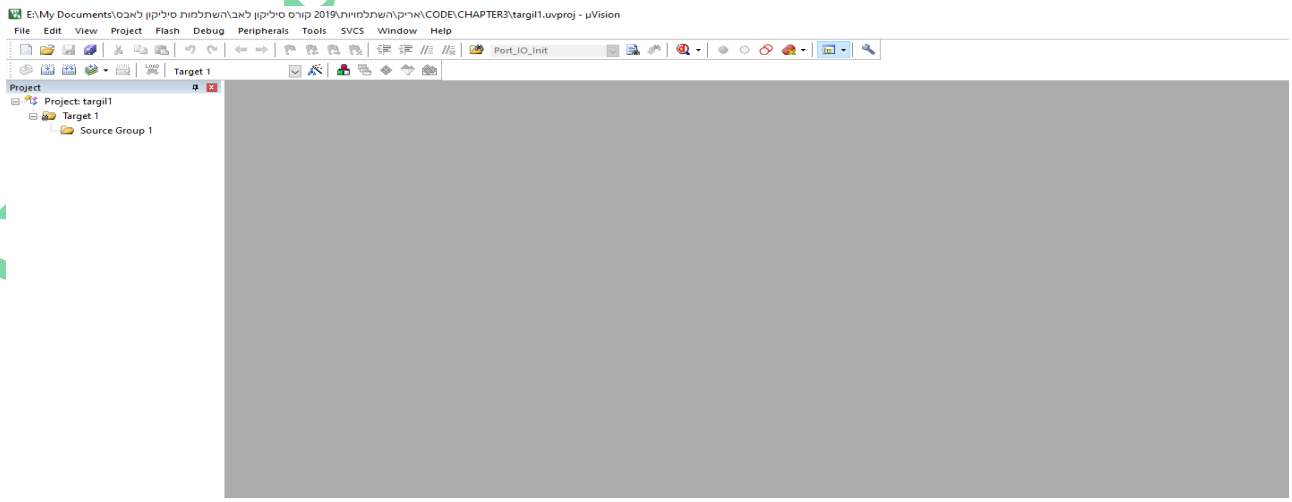
4.11.1 יש אפשרות לפתוח פרויקט חדש עם כל השלבים שהזכרנו בסעיפים קודמים או להשתמש באותו פרויקט הנקרא targil1 ולסלק ממנו את הקובץ tar1.a51 ולהוסיף לו קובץ חדש שנרשום. היות ואנחנו "עצלנים" אז נשתמש באפשרות השנייה ולא בכל תהליך יצירת פרויקט. בהתחלה נסיר את התוכנית tar1.a51 מהפרויקט. לשם כך נשים את העכבר על tar1.a51 בחלון הפרויקט ועם הקלקה על המפסק הימני בעכבר נקבל מסך הנראה באיור 4.25 :



איור 4.25: הסרת הקובץ tar1.a51 מהפרויקט

נקבל מסך שבו השאלה האם אנחנו בטוחים ? ונלחץ על Yes ונקבל את המסך באיור 4.26 .

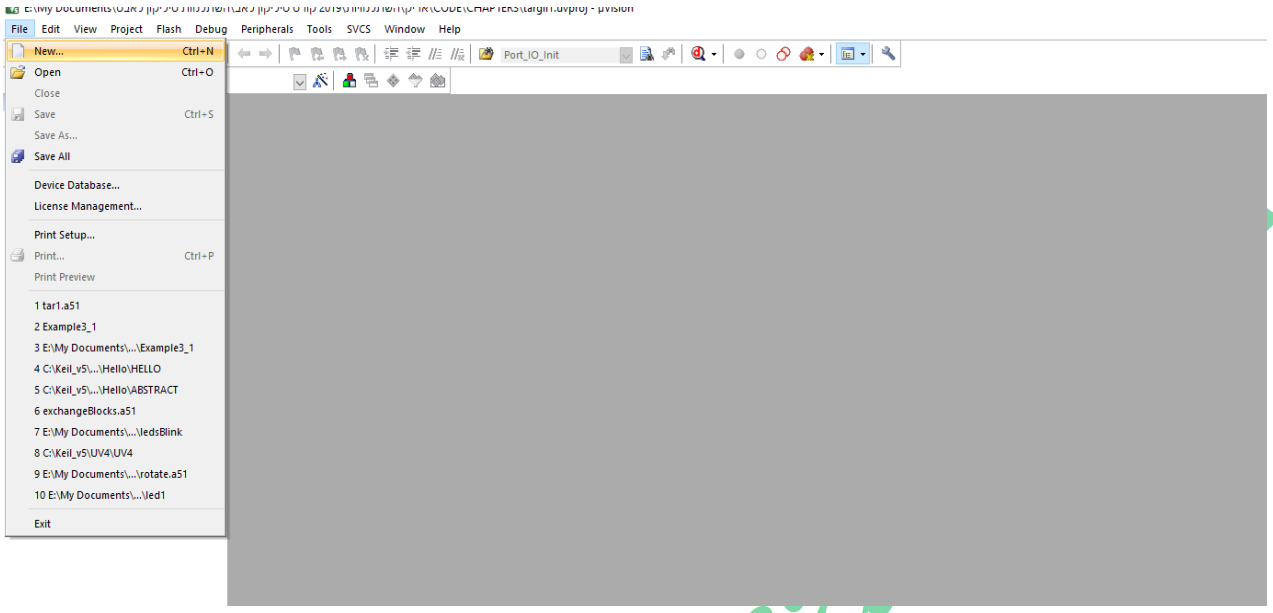




איור 4.26 : המסך לאחר הסרת הקובץ tar1.a51 מהפרויקט .

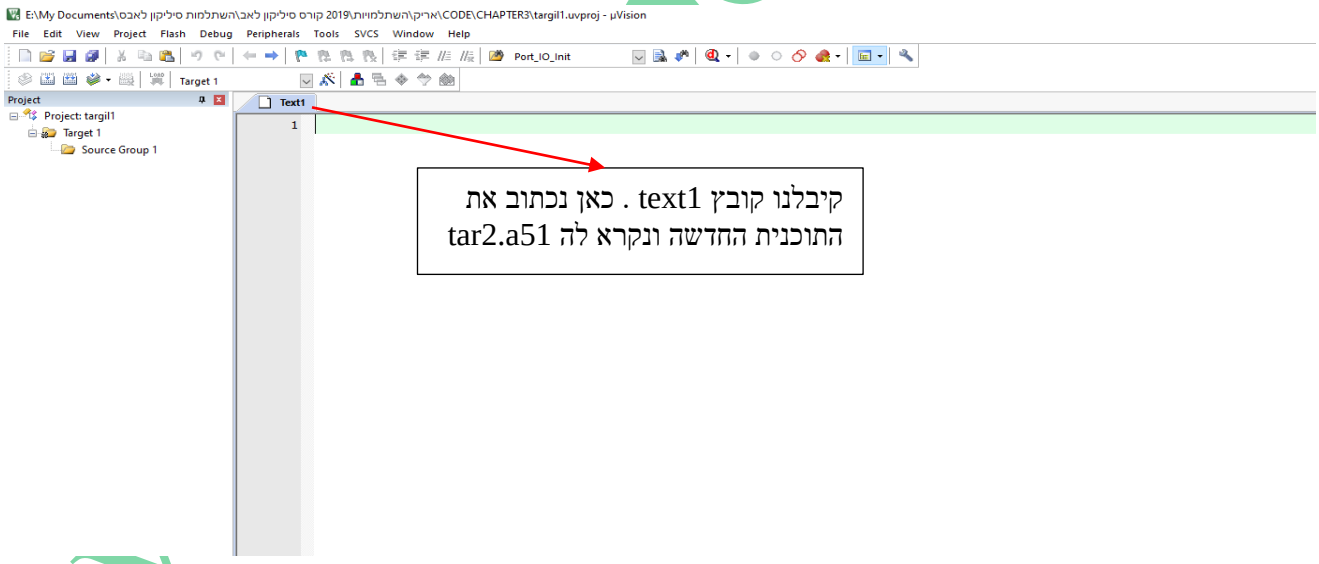
4.11.2 כתיבת התוכנית החדשה

כדי לכתוב תוכנית חדשה יש לחזור על סעיפים החל מ 4.8.2.5 . ניעזר באיור הבא.



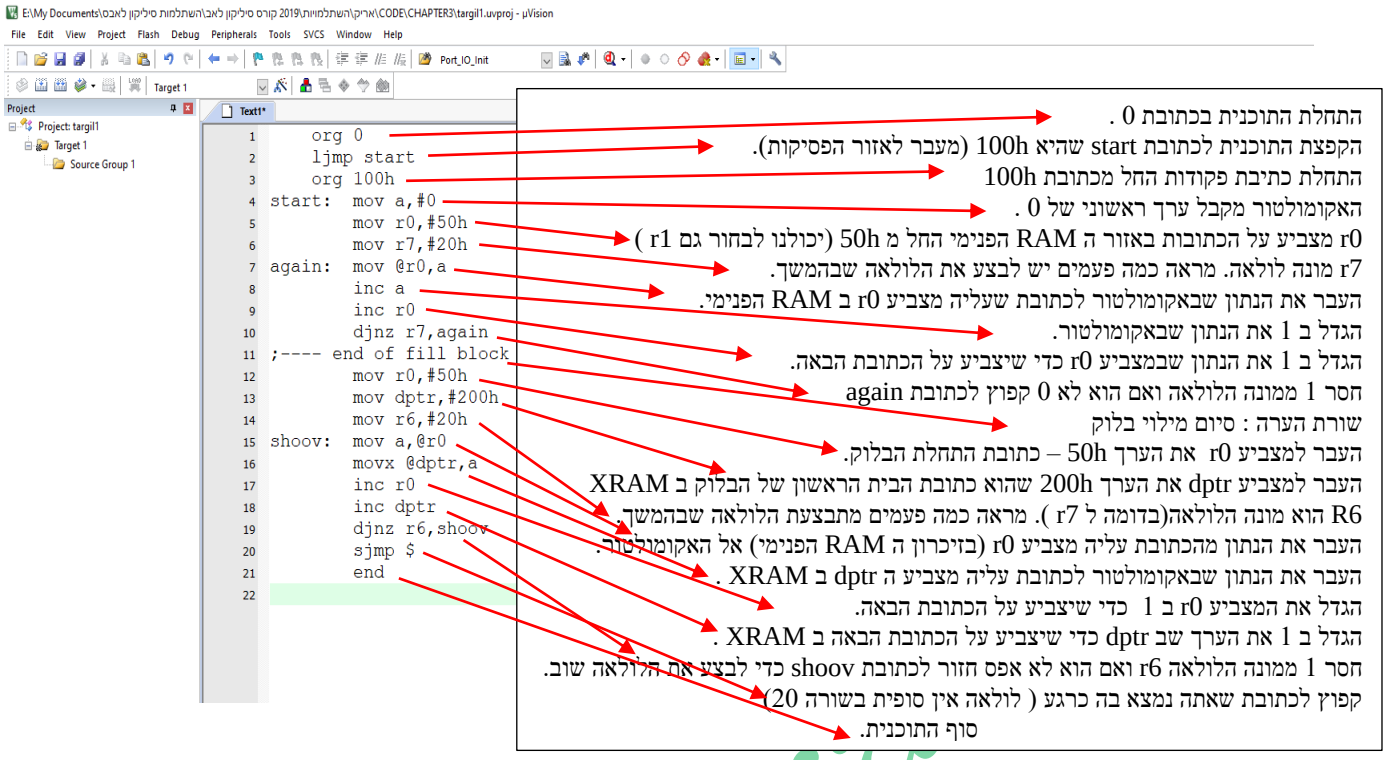
איור 4.27 : מסך כתיבת קובץ חדש

נקיש על File ו New ונקבל



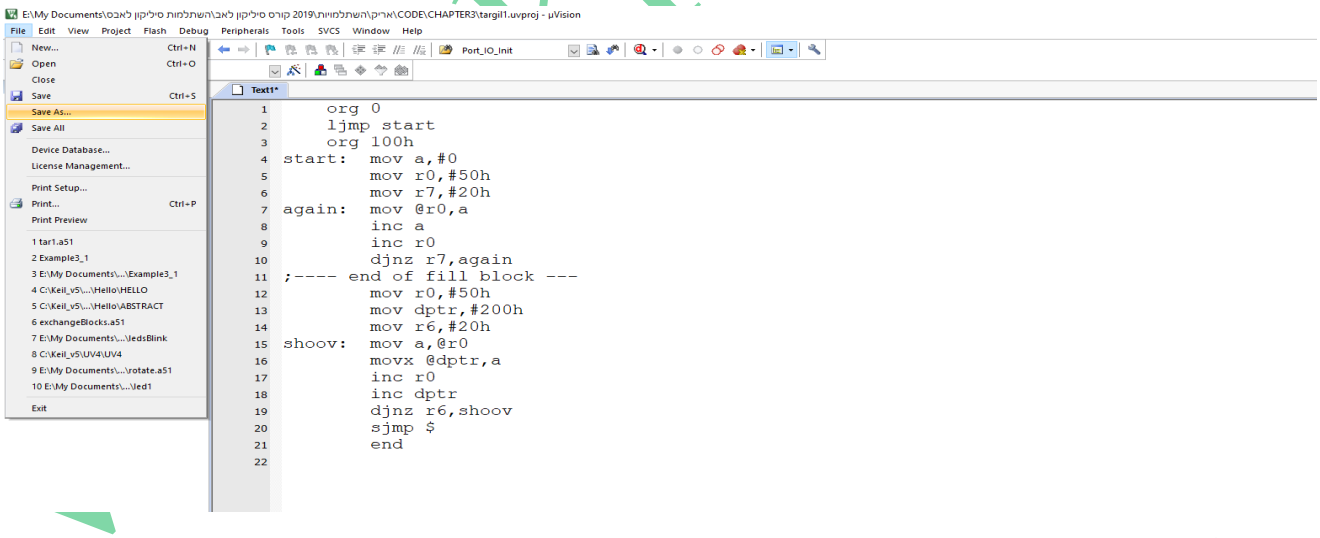
איור 4.28 מסך כתיבת תוכנית חדשה.

נכתוב את התוכנית החדשה . התוכנית נראית באיור 4.29 .



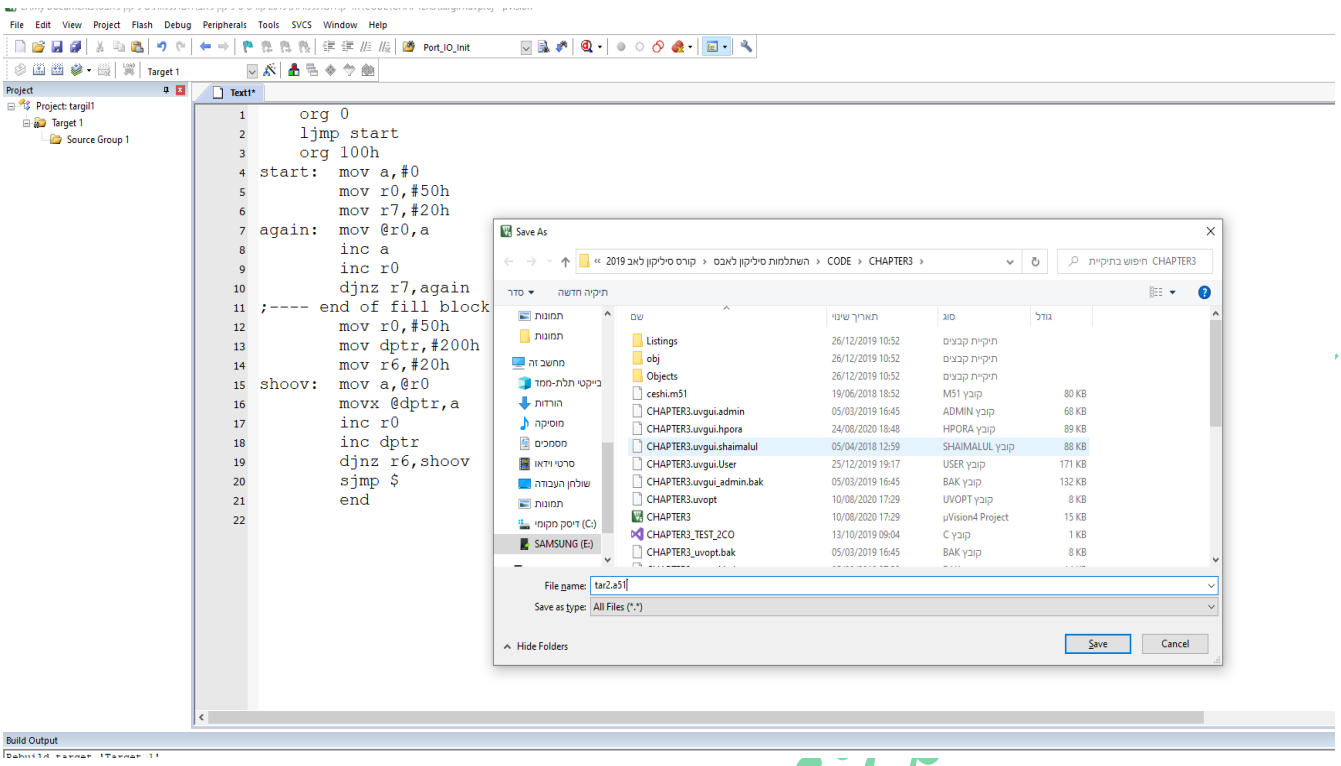
איור 4.29 : התוכנית השנייה. כרגע היא עדיין נקראת text1 .

4.11.3 נשנה את שם התוכנית מ text1 ל tar2.a51 ונוסיף את התוכנית לפרויקט. לשם כך נלחץ על File ובמסך שייפתח נקיש Save As… . כשאי לשים לב שהקובץ רשום עדיין ללא צבעים אלא רק בשחור לבן.



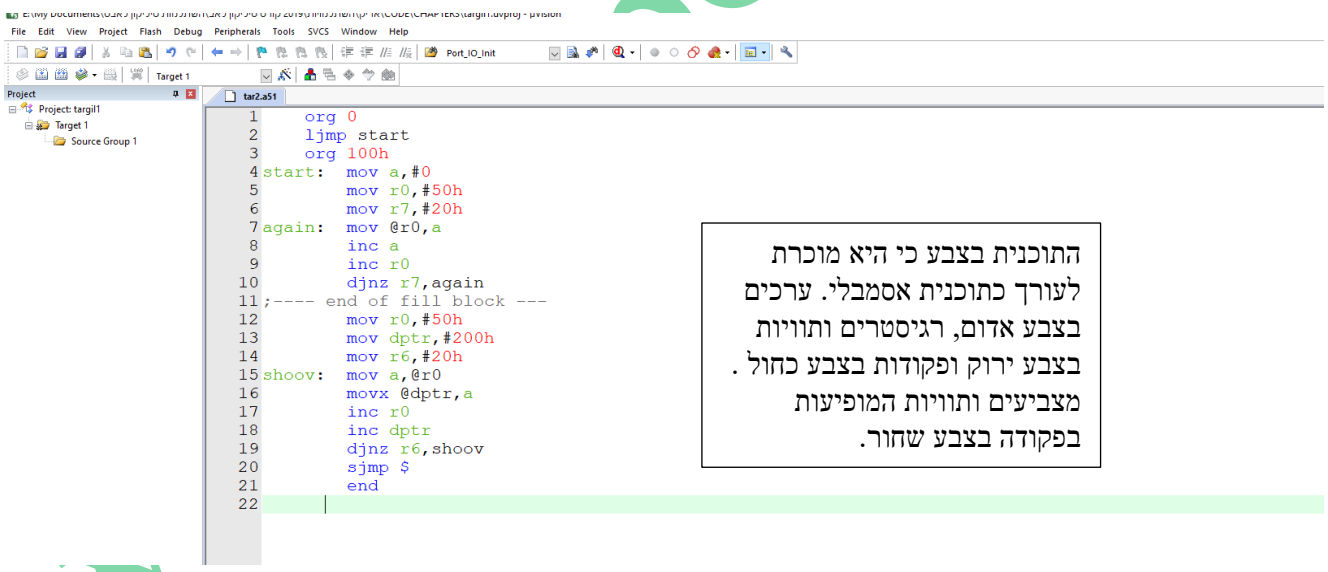
איור 4.30 : מסך שמירת קובץ חדש בשם tar2.a51 .

באיור הבא נרשום את שם התוכנית החדשה ונשמור עליה בספריה הרצויה.



איור 4.31 שמירת הקובץ בשם tar2.a51 בספריה הרצויה .

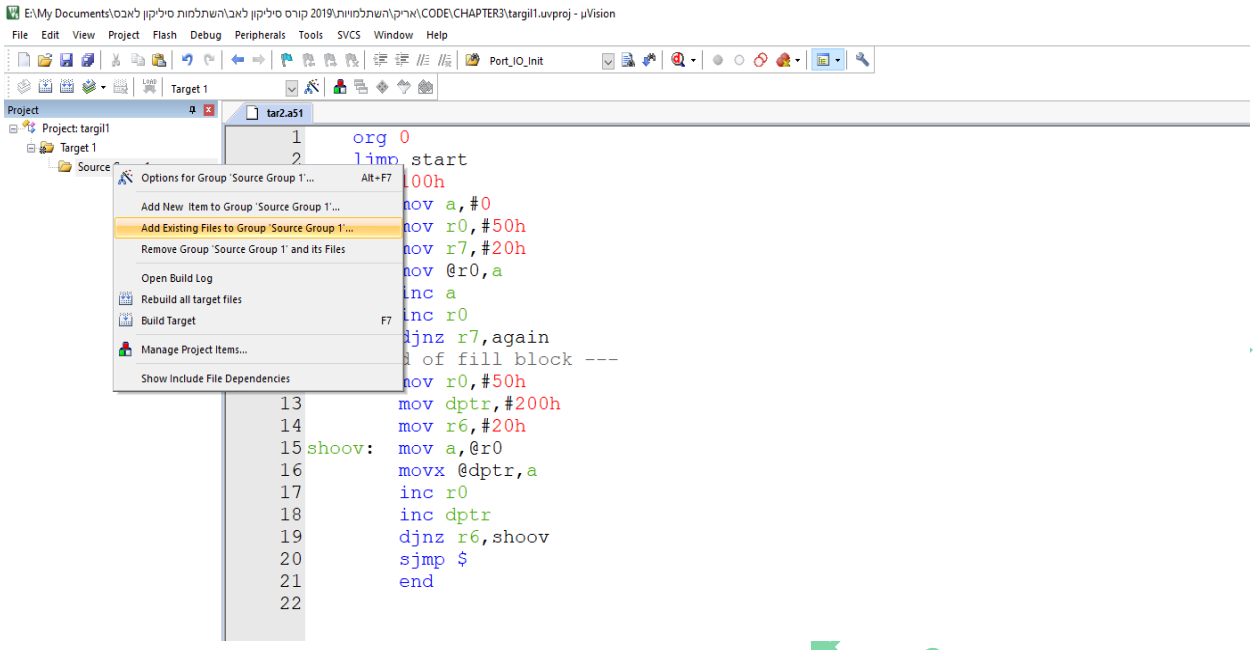
הקשה על Save תשמור את הקובץ בספרייה ונקבל :



איור 4.32 : מסך עם התוכנית tar2.a51 .

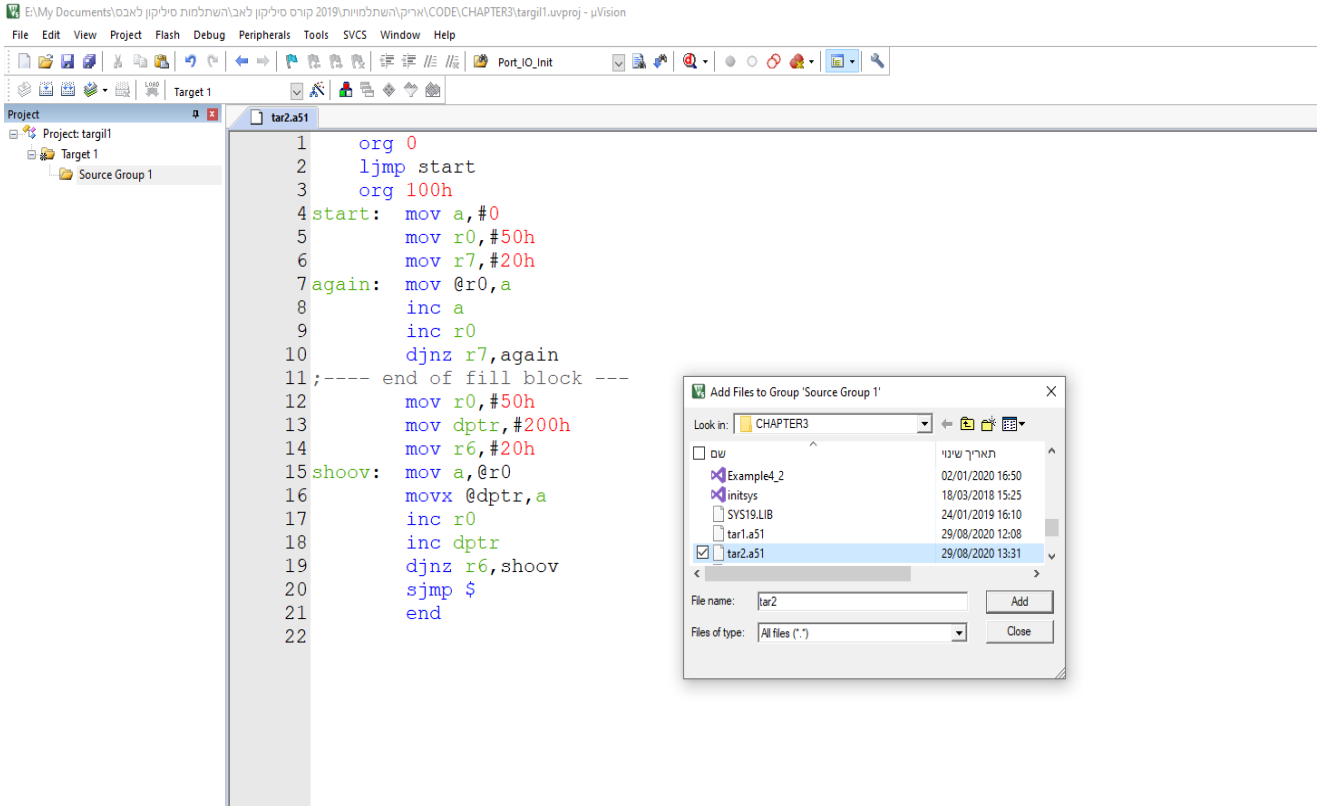
4.11.4 הוספת הקובץ לפרויקט.

כדי להוסיף את הקובץ לפרויקט נשים את העכבר על Source Group 1 החלק של הפרויקט , נקיש על המפסק הימני בעכבר ונקבל :



איור 4.33 : הוספת קובץ קיים לפרויקט .

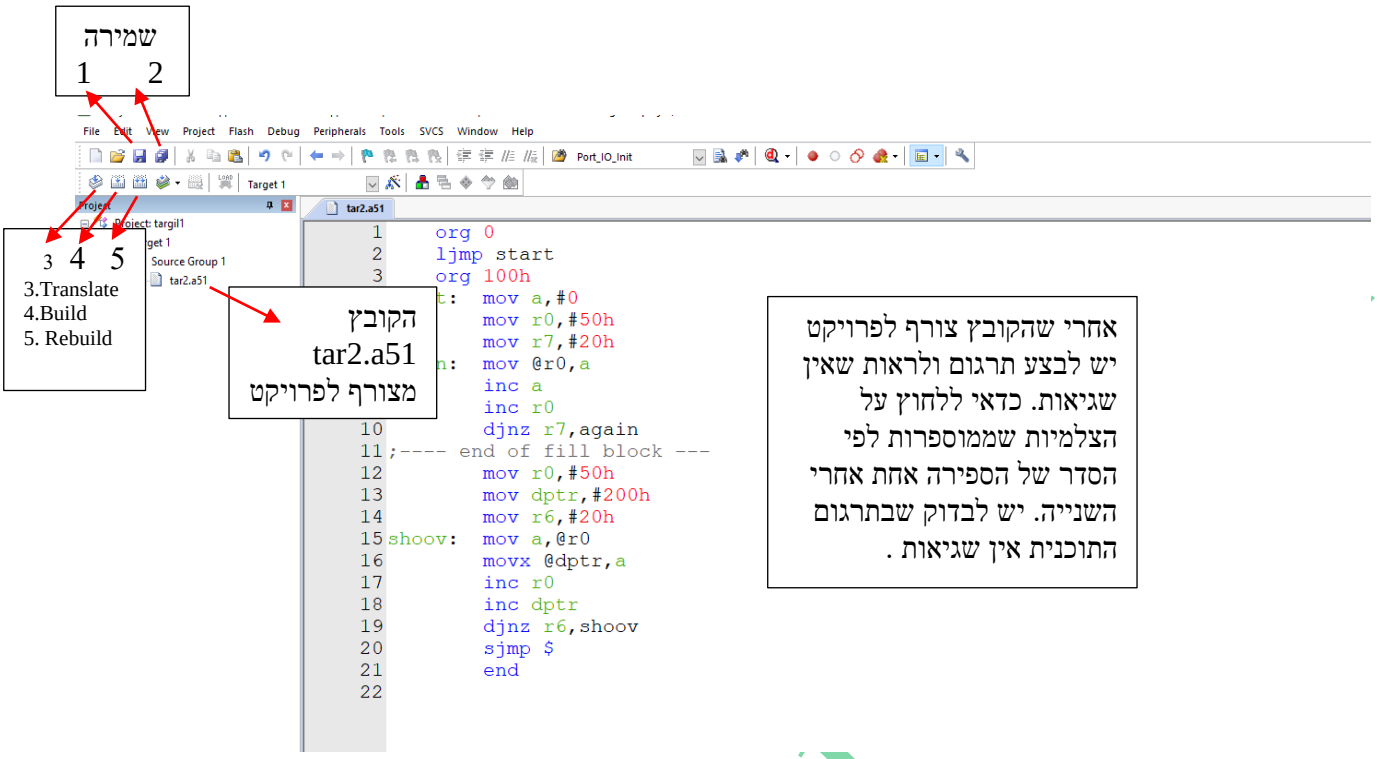
נקיש על Add Existing Files to Group Sourcr Group 1… ונקבל :



איור 4.34 : הוספת הקובץ tar2.a51

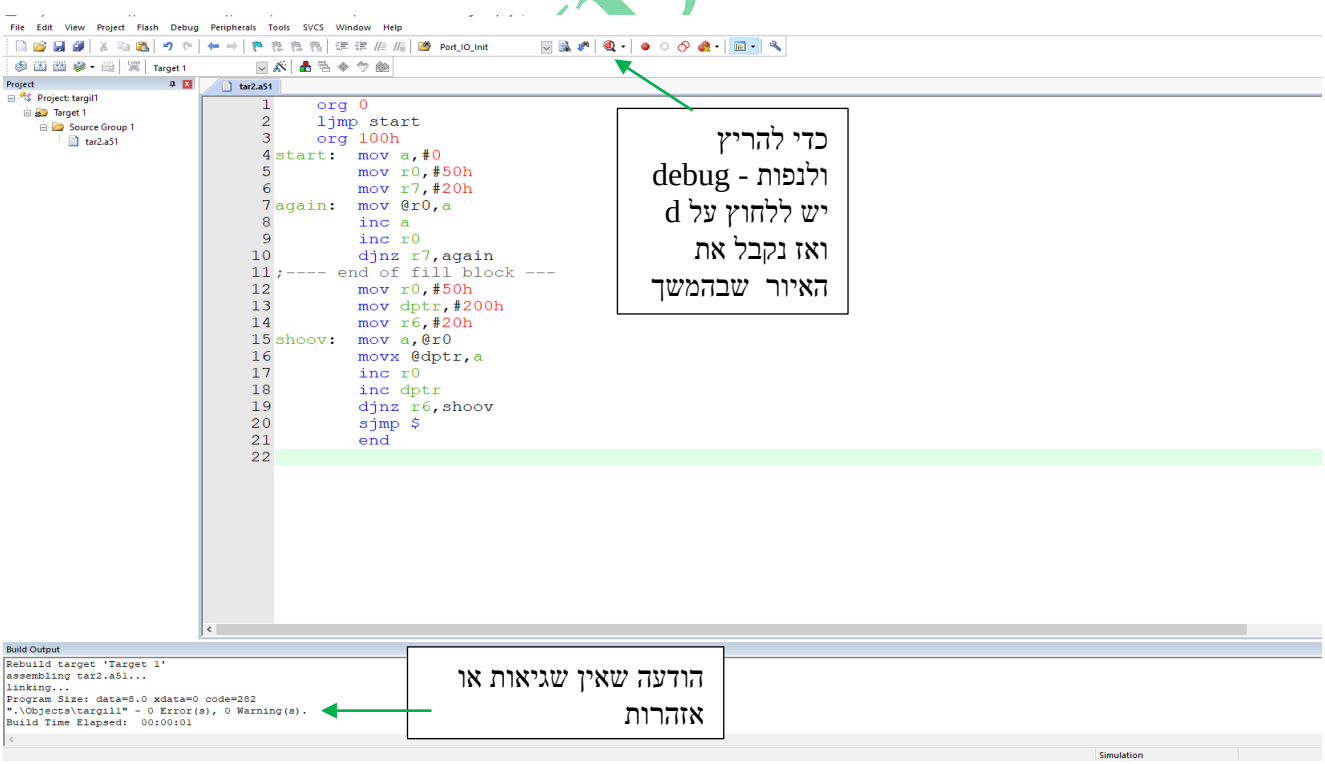
נלחץ על tar2.a51 פעמיים או נסמן אותו עם העכבר ונלחץ על Add והקובץ יצורף לפרויקט.

4.11.5 שמירה ותרגום



איור 4.35 : הקובץ צורף. הגיע שלב השמירה והתרגום.

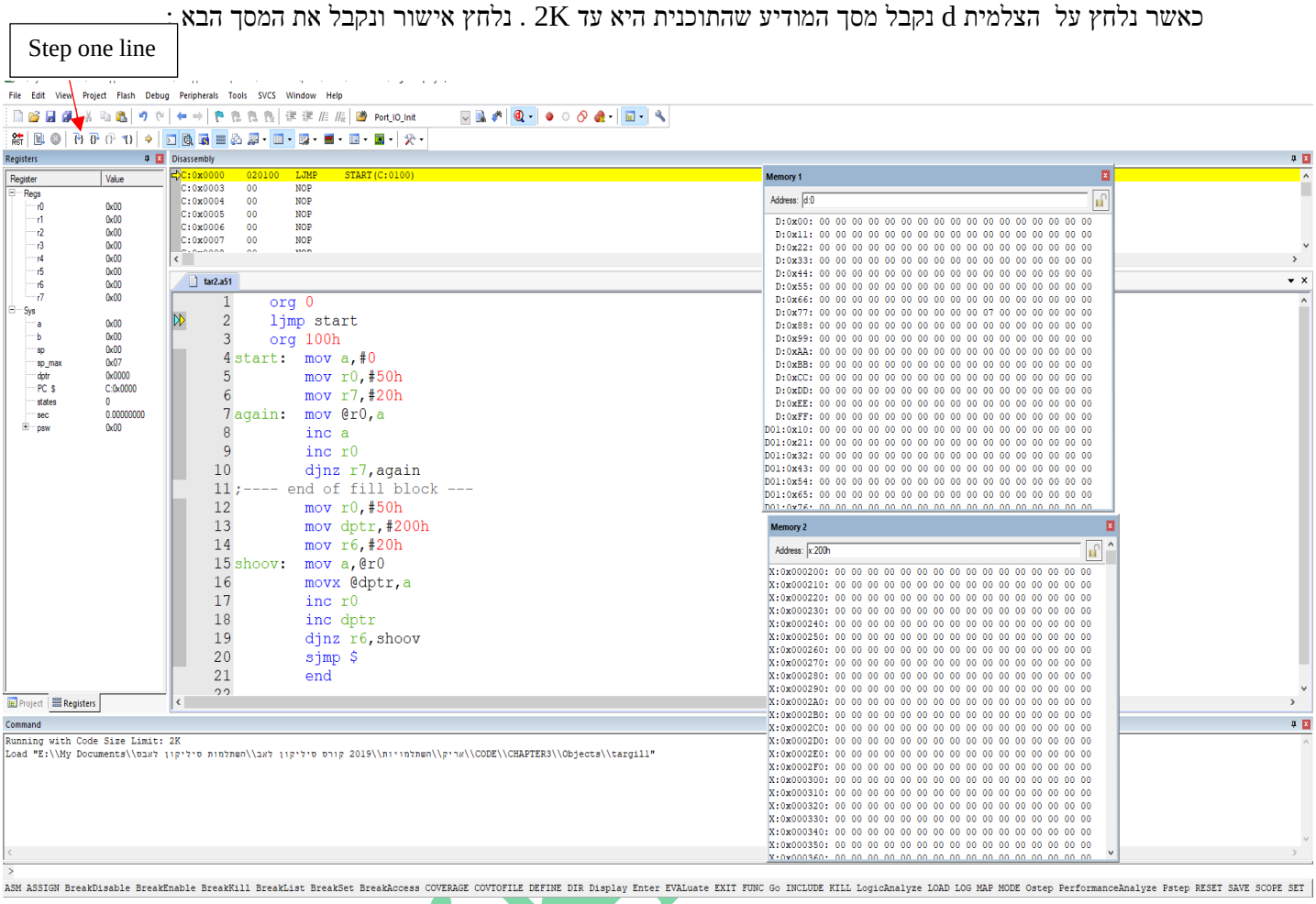
מומלץ ללחוץ על הצלמיות אחת אחרי השנייה לפי הסדר הרשום. במידה שאין שגיאה נקבל :



איור 4.36 : אחרי פעולת תרגום ו build

4.11.6 הרצה ו debug (ניפוי)

כאשר נלחץ על הצלמית d נקבל מסך המודיע שהתוכנית היא עד 2K . נלחץ אישור ונקבל את המסך הבא :

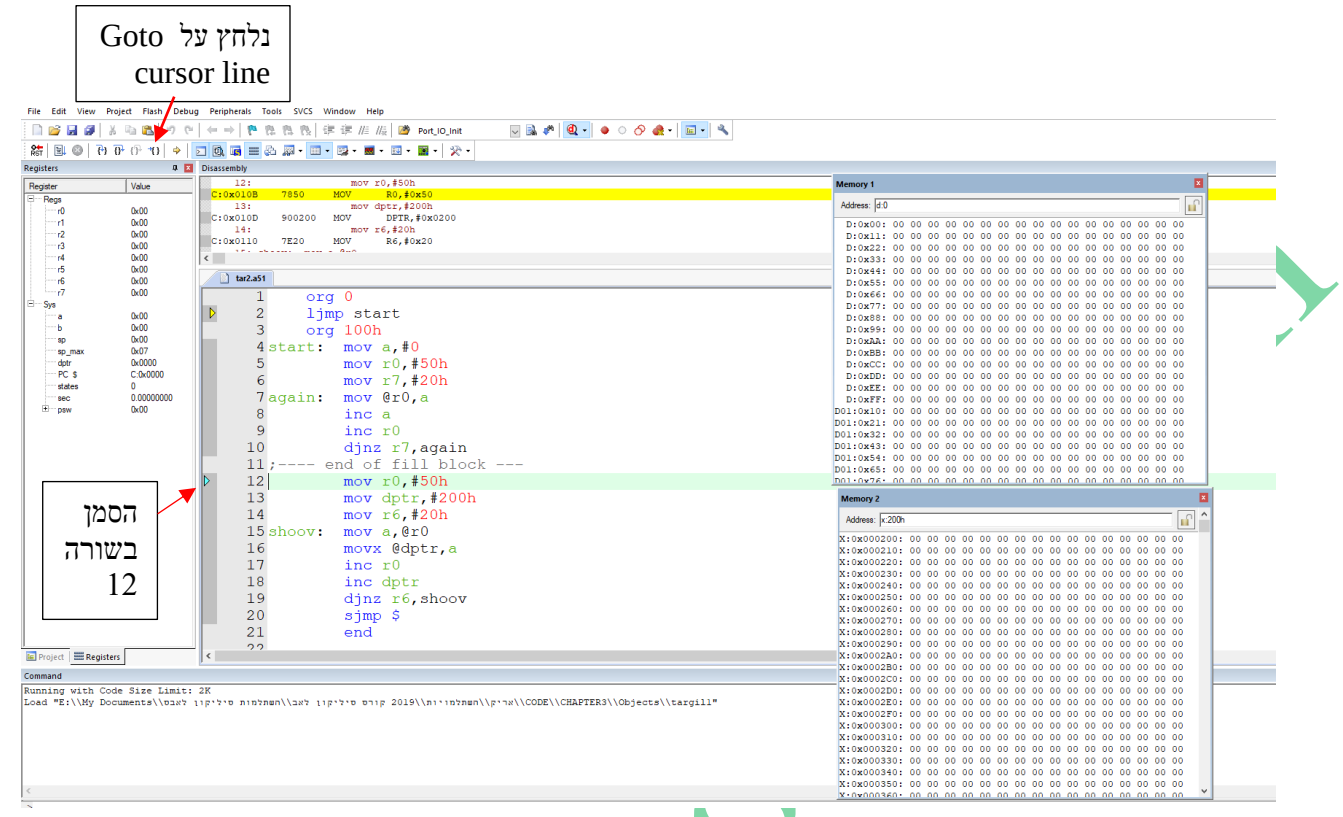


איור 4.37 מסך הרצה וניפוי

באיור רואים 2 אזורי זיכרון. העליון הוא אזור ה RAM הפנימי החל מכתובת 0 והשני אזור ה XRAM החל מכתובת 200h . האיור של המסכים דומה לזה שבאיור 4.24 .

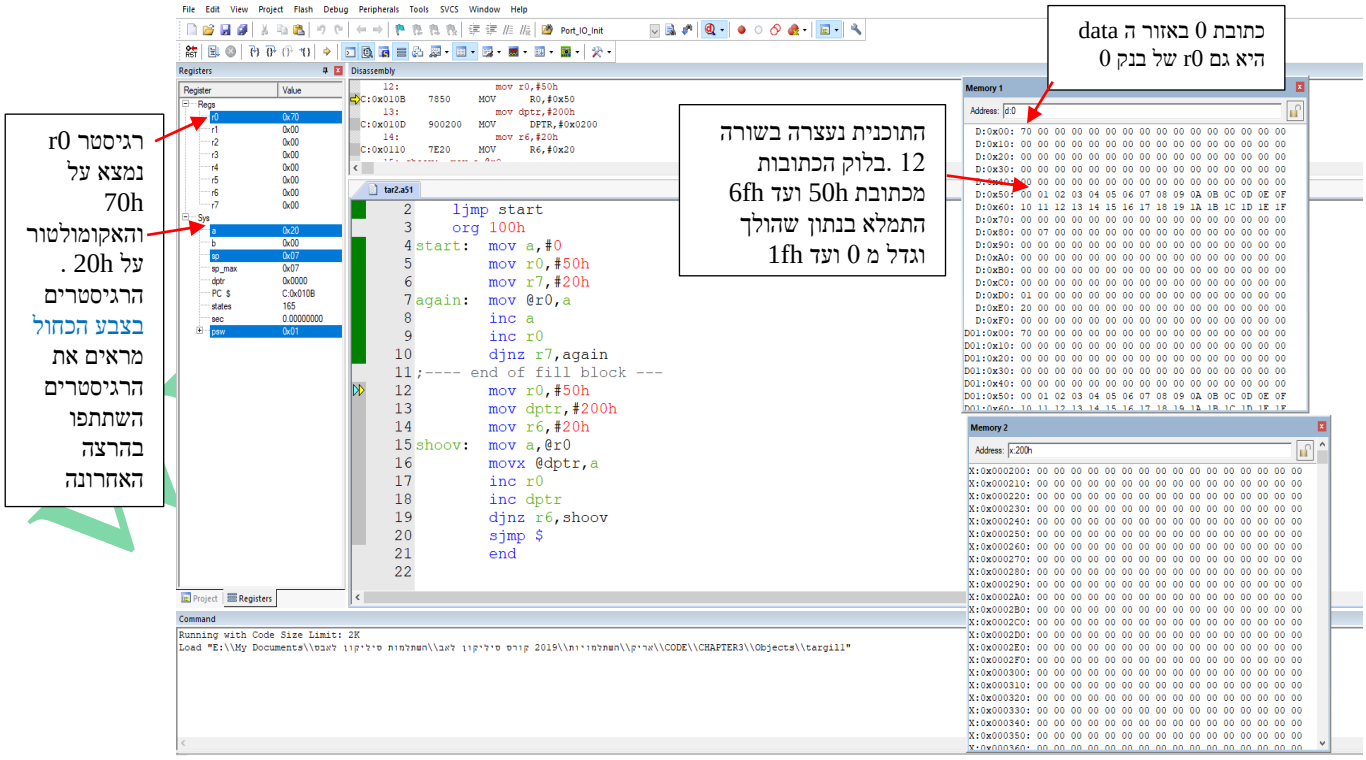
4.11.6.1 – הרצה עם Run to cursor line.

ניתן להריץ את התוכנית צעד שורה אחת - Step one line . ניתן ללחוץ על F11 או על הצלמית המתאימה במסך. במקרה כזה נעבור שורה אחרי שורה על התוכנית ונוכל לראות כיצד משתנים הרגיסטרים שבתוכנית ואזורי הזיכרון. נדגים הרצה עם Run to cursor line (הרץ עד שורת הסמן). לשם כך נביא את הסמן עם העכבר לשורה 12 ונלחץ על מפסק שמאל של העכבר ונקבל את התמונה שבאיור הבא :



איור 4.38 : סימון השורה שרוצים שהתוכנית תרוץ ותיעצר בשורה זו Goto cursor line .

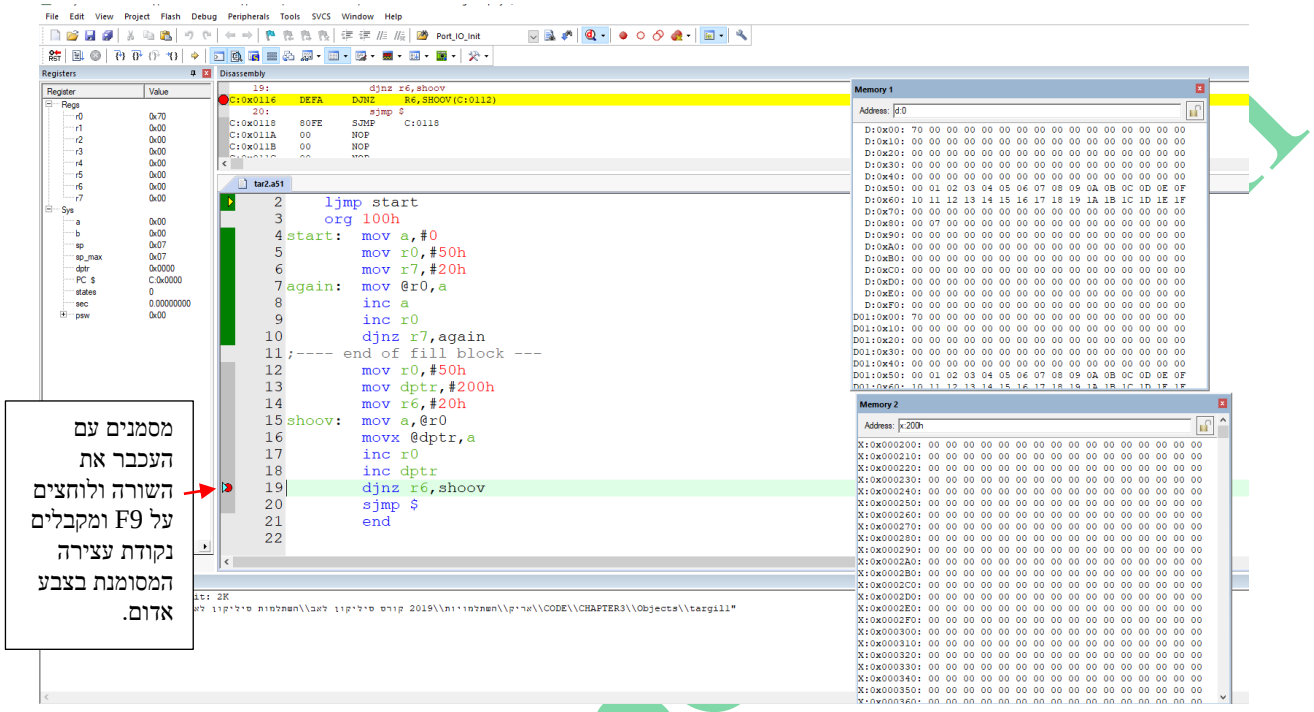
לחיצה על צלמית Goto cursor line (או Ctrl+F10) ונקבל את התמונה הבאה :



איור 4.39 : מצב הזיכרון והרגיסטרים בהרצת התוכנית עד שורה 12 .

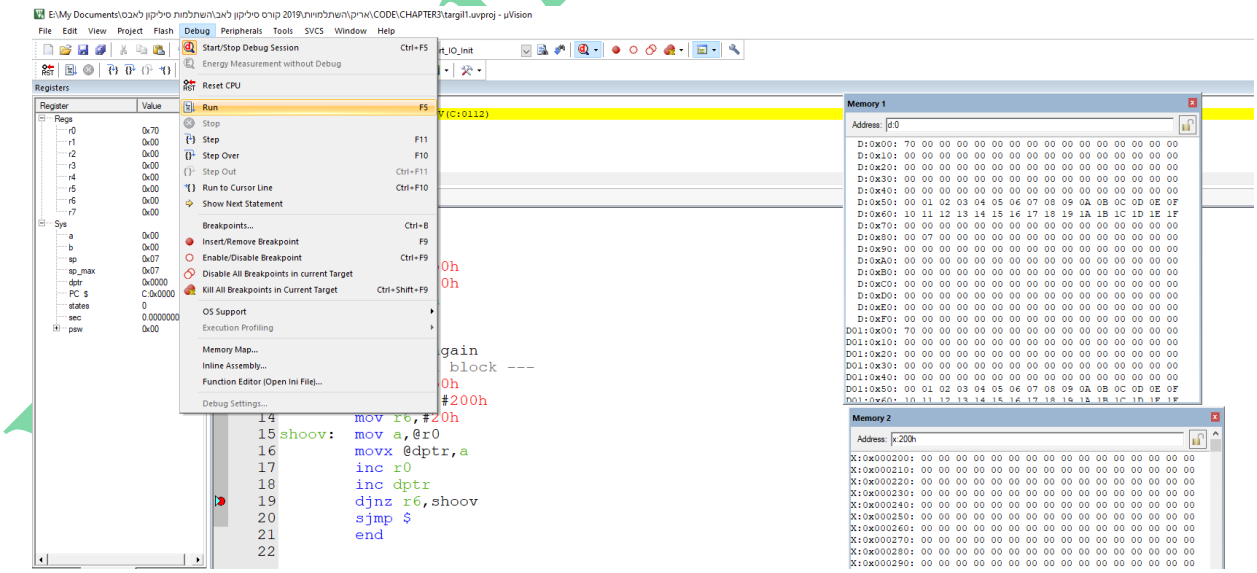
4.11.6.2 הרצה עם נקודת שבירה - Break point

נקודת שבירה היא שורה בתוכנית שבכל פעם שהתוכנית תגיע לשורה זו היא תעצור ונוכל לבדוק מה קורה בזיכרון, ברגיסטרים (או עם משתנים בשפת C). כדי לסמן שורה שבה נרצה לשים נקודת עצירה נסמן עם העכבר את השורה הרצויה ולחץ על F9 ונקבל את המסך הבא:



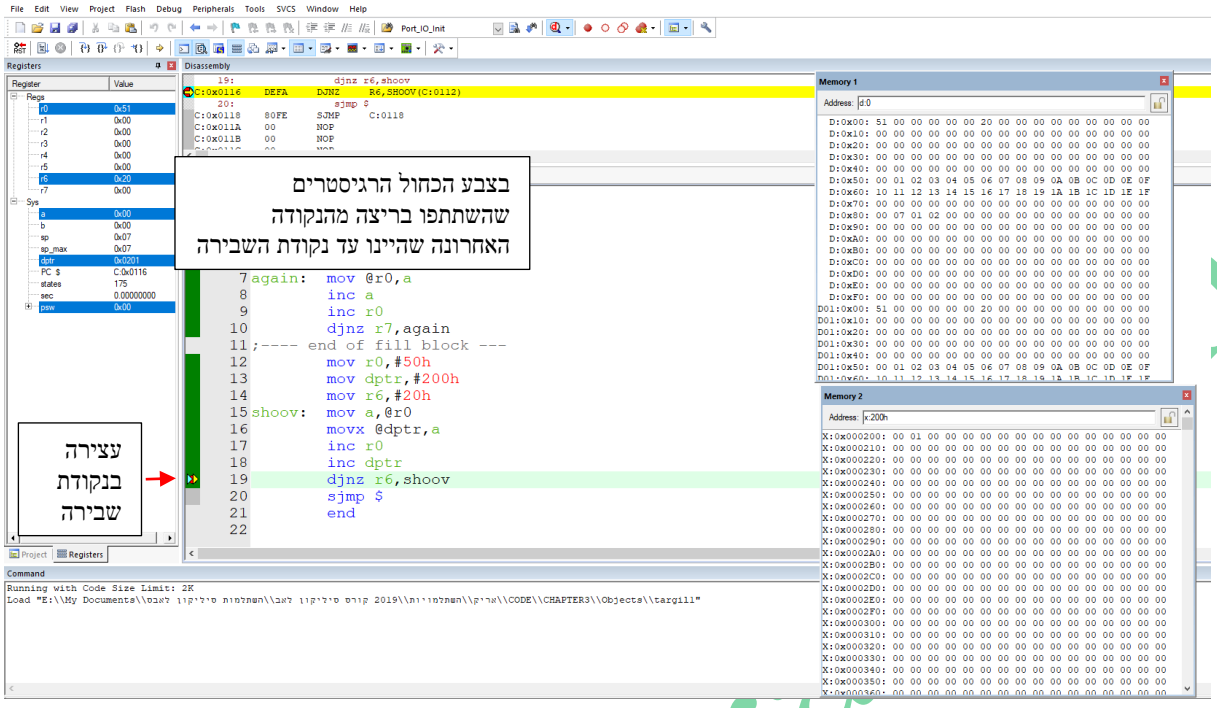
איור 4.40 : נקודת עצירה

נריץ את התוכנית על ידי לחיצה על Debug בתפריט ו Run או על מקש F5 :



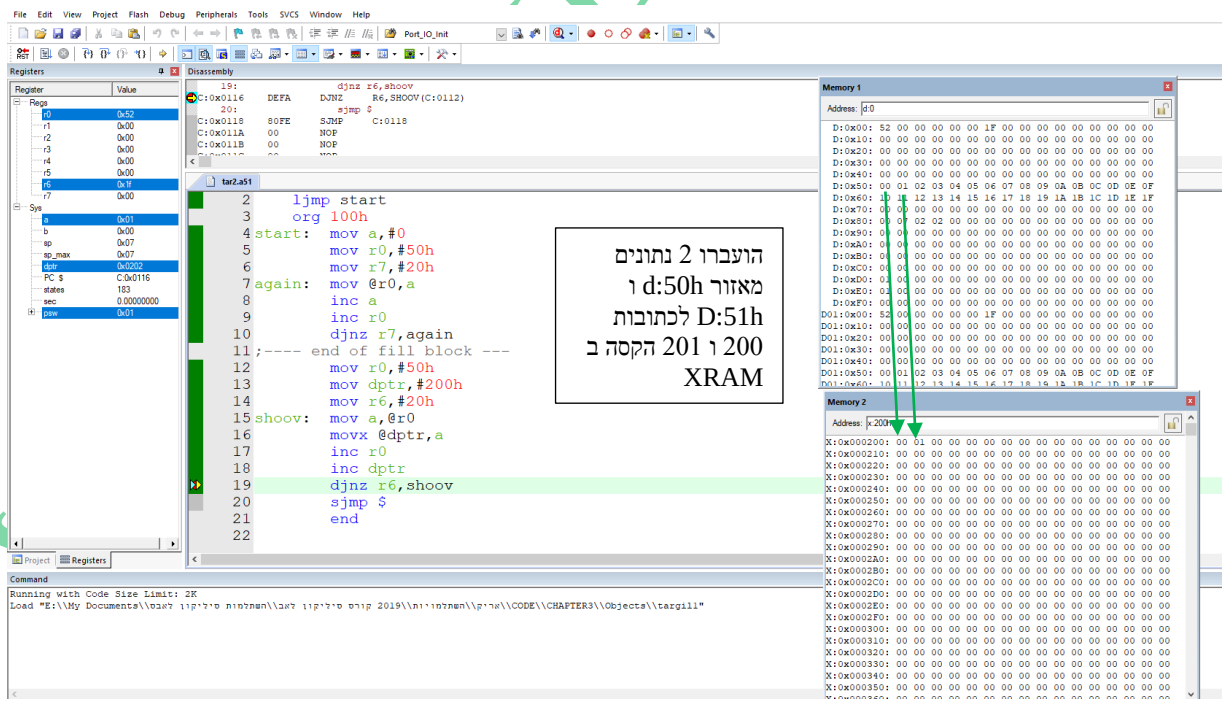
איור 4.41 : הרצת התוכנית עד נקודת העצירה בעזרת Debug ו Run

התוכנית רצה ועוצרת בשורה עם נקודת השבירה :



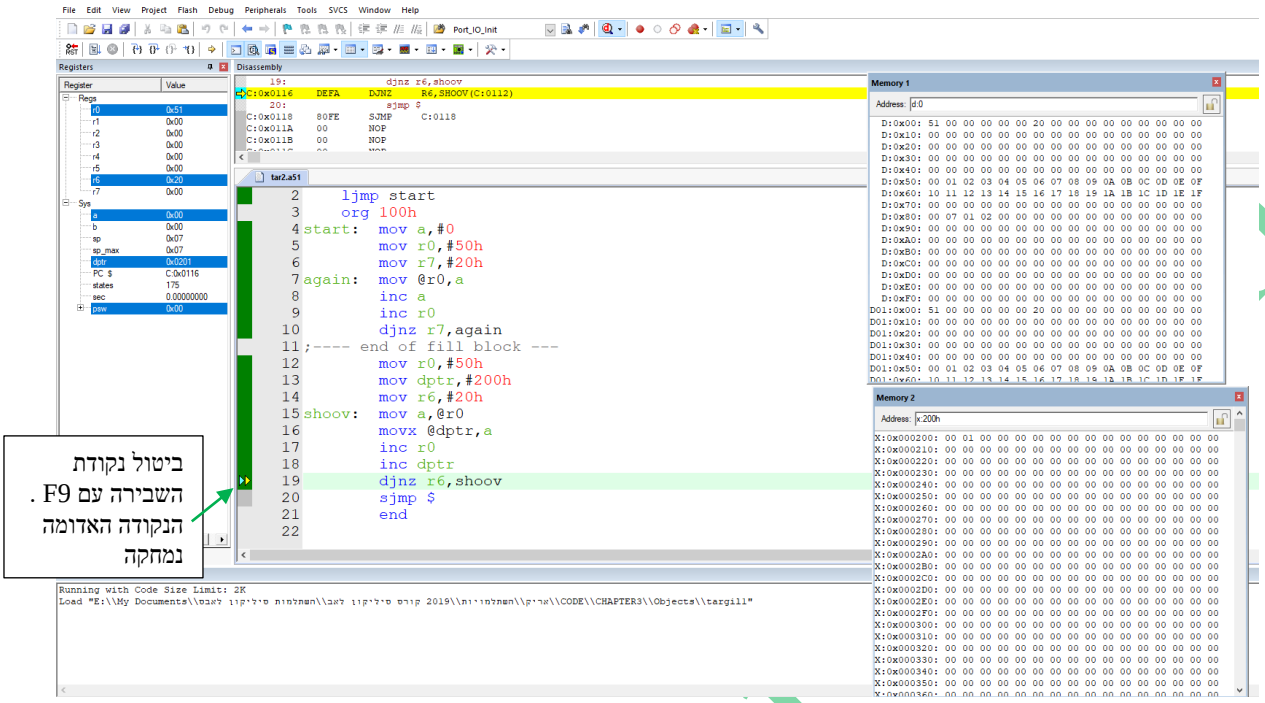
איור 4.42 : הרצה עם נקודת שבירה פעם ראשונה.

הרצה פעם שנייה : שוב נלחץ על F5 או על Debug ואחר כך על Run ונקבל :



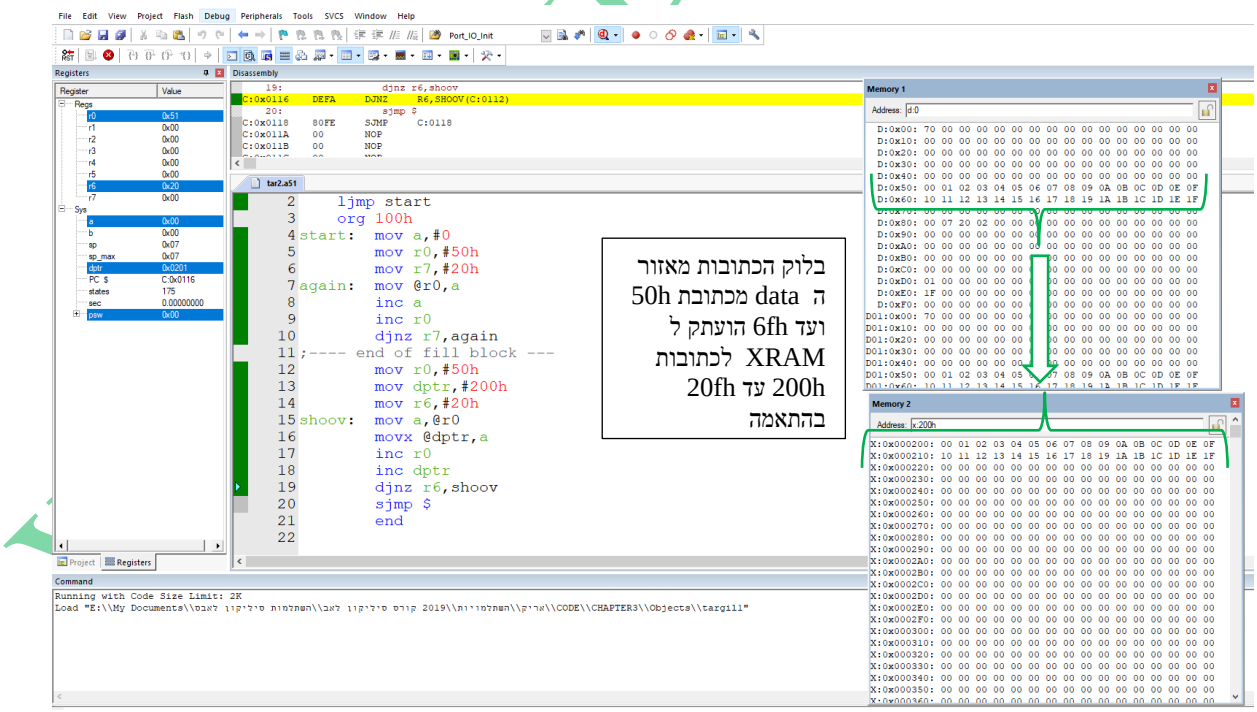
איור 4.43 : מעבר פעמיים עד לנקודת עצירה. העברת 2 נתונים.

לאחר שהבנו מה עושה נקודת השבירה , נמחק את נקודת השבירה ונריץ את התוכנית עד סופה. נבטל את נקודת השבירה על ידי לחיצה על F9 . הסימן האדום של נקודת שבירה ייעלם.



איור 4.44 : ביטול נקודת שבירה על ידי הקשה על F9 .

נריץ את התוכנית בעזרת מקש F5 או נלחץ שוב על Debug ואחר כך על Run ונקבל :



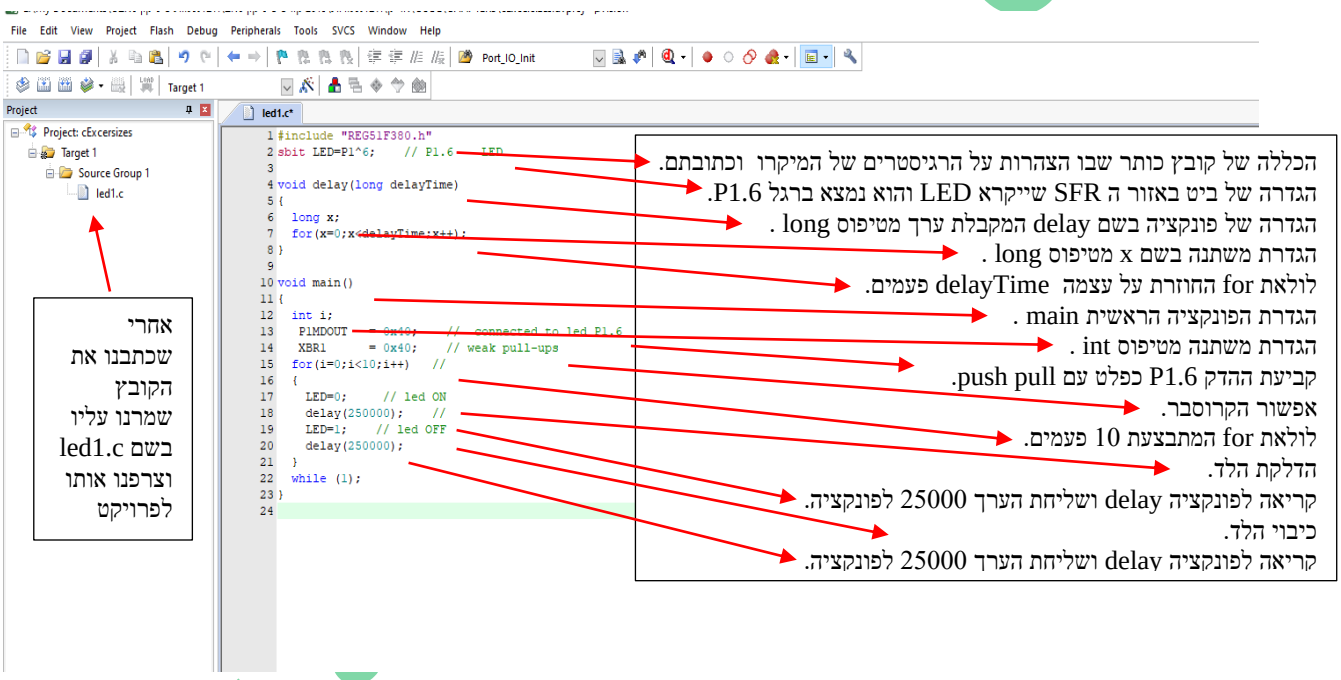
איור 4.45 : מצב הזיכרון והרגיסטרים בסיום התוכנית.

4.12 µVision וכתיבת תוכנית בשפת C51

העבודה עם התוכנה בשפת C דומה לעבודה עם שפת אסמבלי. פתיחת הפרויקט היא כמו פתיחת פרויקט באסמבלי. כתיבת התוכנית בשפת C והכנסתה לפרויקט היא כמו עבור קובץ אסמבלי רק שלקובץ ניתן סיומת c.שם הקובץ . ניתן לחזור על סעיפים 4.8.2.1 עד 4.8.2.6. נכתוב קובץ המהבהב לד 10 פעמים ונקרא לו בשם led1.c

4.12.1 כתיבת התוכנית וצירופו לפרויקט

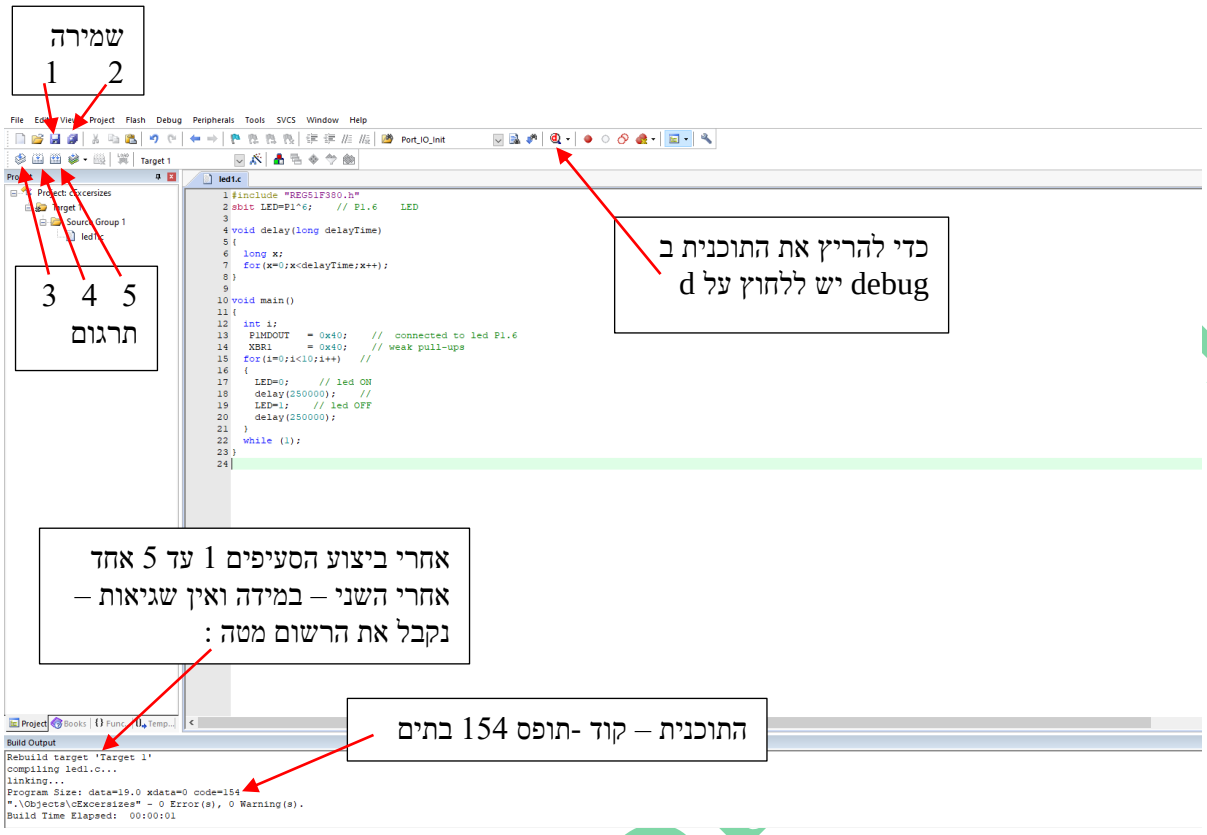
האיור הבא מראה את התוכנית שרשמנו והוספנו לפרויקט. אנחנו נניח שחיברנו לד ל P1.6 של המיקרו בקר. הלד מקבלת באנודה 3 וולט והקתודה מתחברת דרך נגד אל P1.6 . אם נוציא 0 ל P1.6 הלד תידלק ואם נוציא 1 היא כבויה.



איור 4.46 : תוכנית בשפת C להבהוב של לד 10 פעמים.

4.12.2 תרגום (קומפילציה)

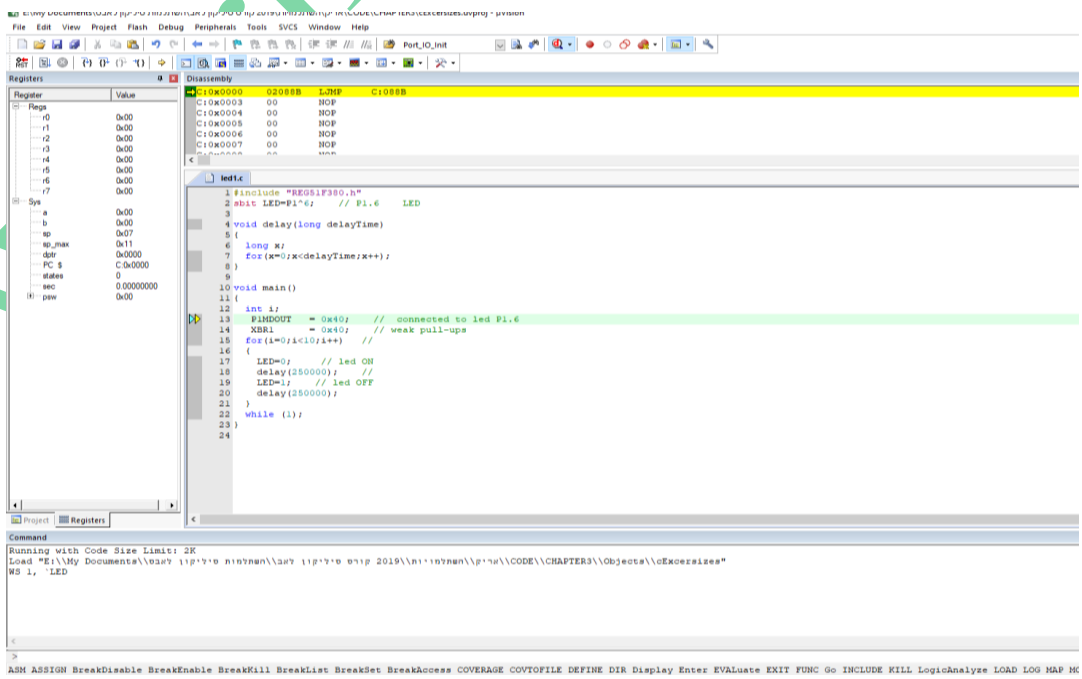
באיור הבא יש לבצע את הסעיפים 1 עד 5 . זאת כמובן בתנאי שאין טעות בקומפילציה.



איור 4.47 : תרגום (קומפילציה)

4.12.3 הרצה

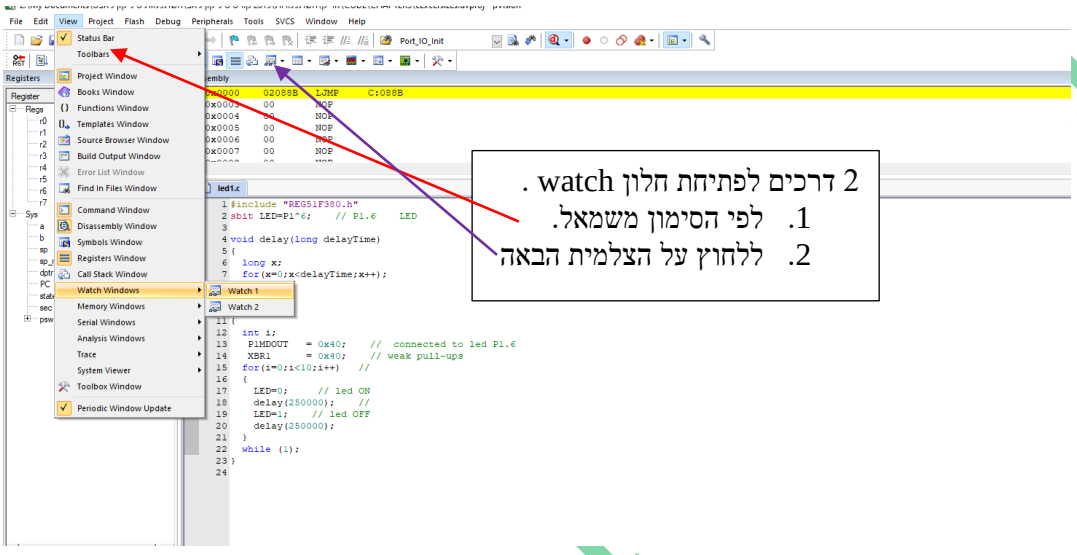
נלחץ על ה d ונעבור לתוכנית ה debug .



איור 4.48 מסך ה debug .

גם כאן נוכל להשתמש בכל אפשרויות ההרצה החל מ Step one line – F11 ועבור דרך Run to cursor line ונקודות שבירה.

בתוכנית זו אנחנו רוצים לראות מה קורה עם P1.6 שם נמצאת הלד. כדי לראות מה קורה עם הלד נפתח חלון watch שמאפשר לנו לראות משתנים של התוכנית.

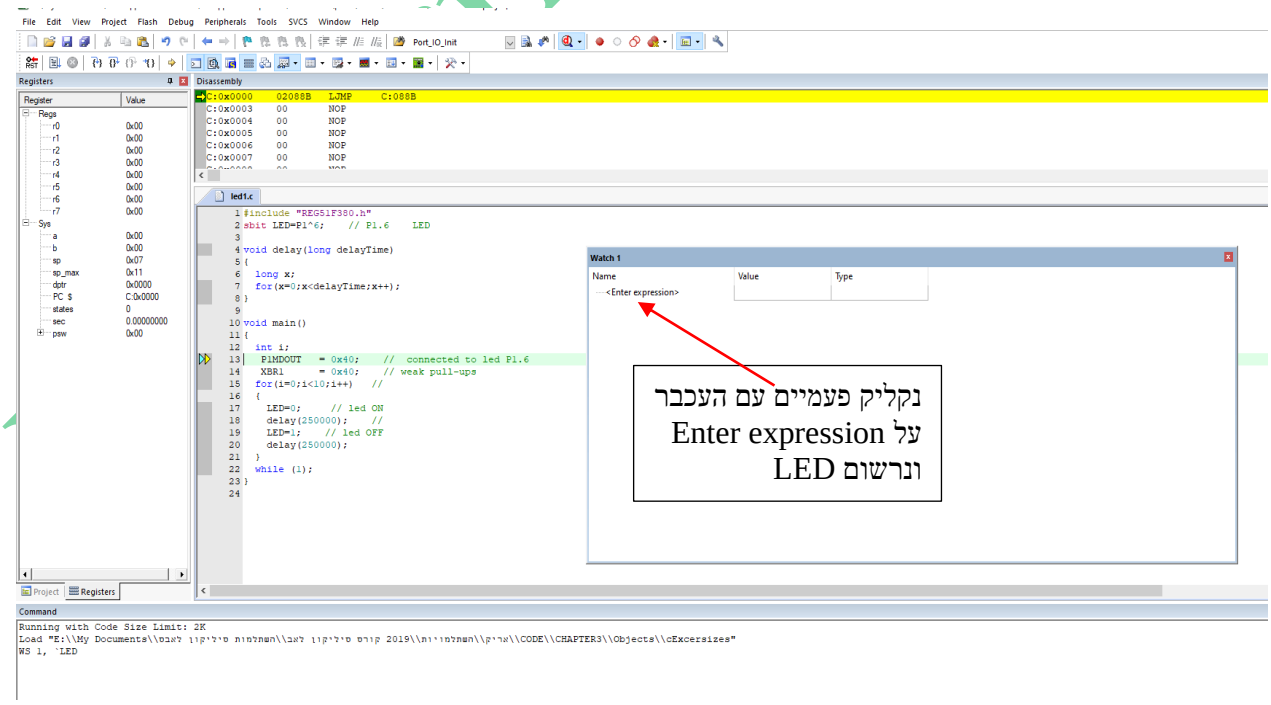


איור 4.49 : כיצד נפתח חלון watch

דרך אחת היא ללחוץ עם העכבר על View בתפריט ואז בתפריט המשנה שנפתח לסמן Watch Windows ובחלון תפריט הנוסף שנפתח ללחוץ על watch 1 .

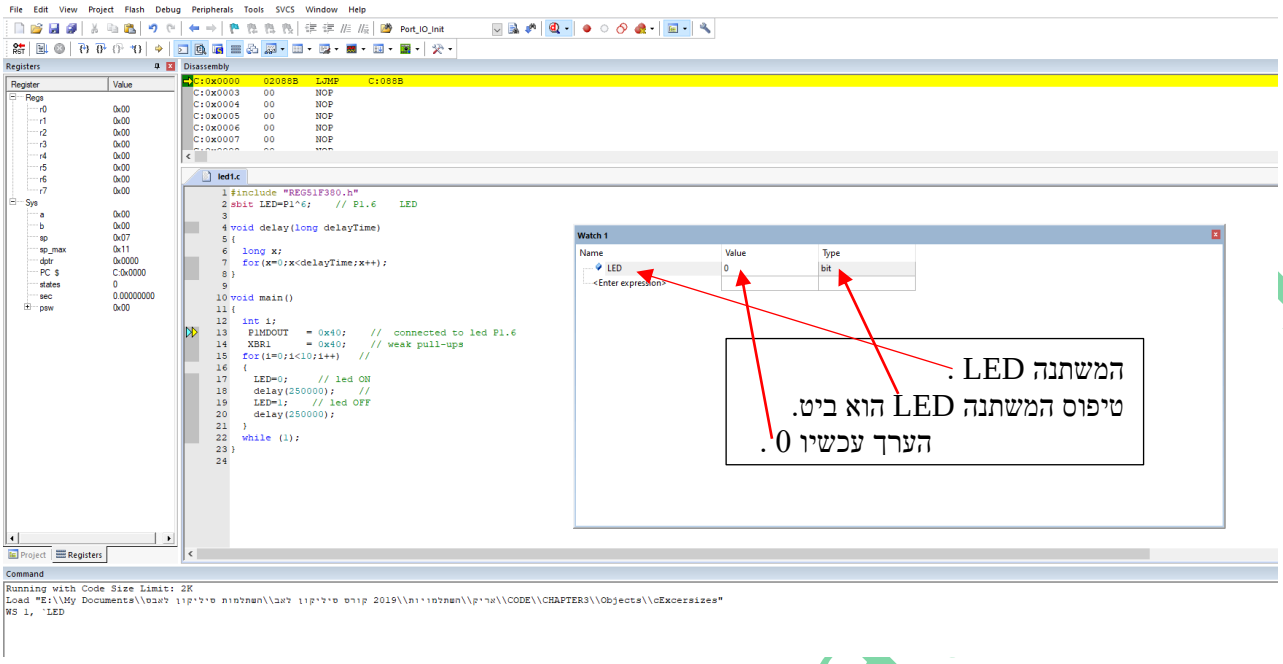
דרך נוספת היא ללחוץ על הצלמית שבאיור.

המסך שנקבל נראה כך :



איור 4.50: מסך watch

לאחר שרשמנו LED נקבל את המסך הבא :



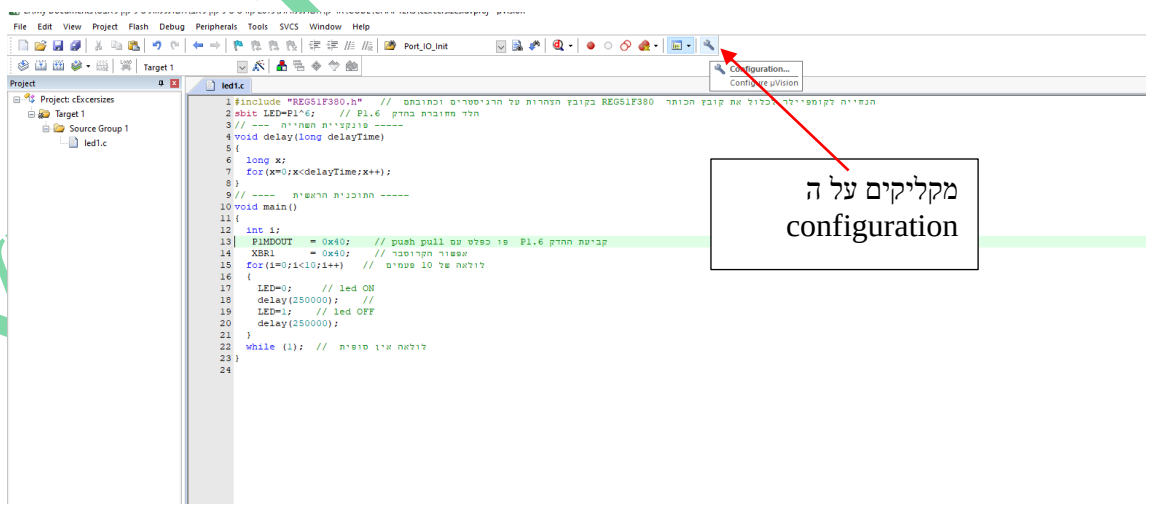
איור 4.51 : המשתנה LED נוסף במסך ה WATCH.

ניתן להוסיף משתנים נוספים על ידי הקלקה כפולה שוב על Enter expression .

עכשיו ניתן להריץ את התוכנית ולראות שהתוכן של ה LED משתנה בין ON ל OFF (0 ו 1) 10 פעמים .

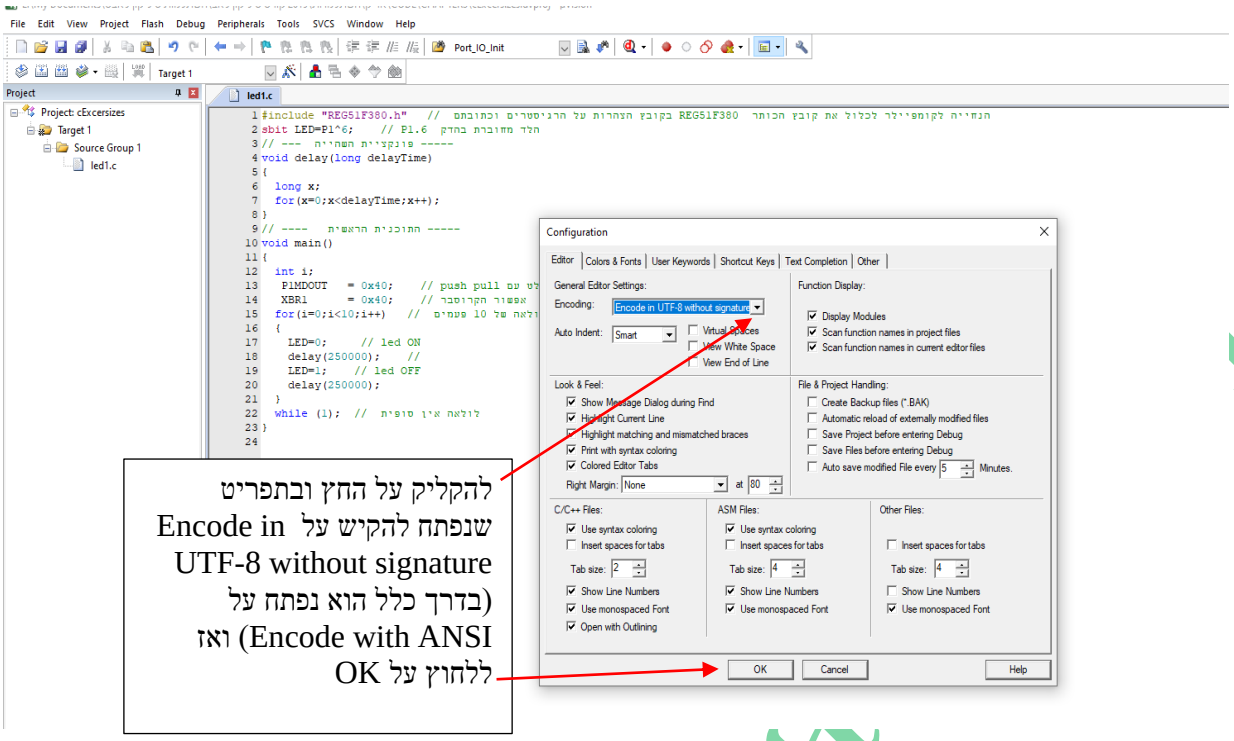
4.13 הוספה של כתיבת עברית

כדי להוסיף הערות בעברית יש לבצע את השלבים הבאים :



איור 4.52 : מעבר למסך התצורה – Configuration

הקלקה על ה configuration ונקבל את המסך הבא :



איור 4.53 : מסך העריכה

זהו מסך העריכה שבו ניתן לקבוע את גודל התווים באסמבלי וב C , את צבע השורות , מרחק השורות מקצוות המסך ועוד.

יש להקליק על החץ שבאיור ובמסך שייפתח (ברירת המחדל היא Encode In ANSI) יש לקבוע Encode in UTF-8 without signature . לאחר מכן ללחוץ על OK . מרגע זה ניתן יהיה לכתוב גם בעברית.