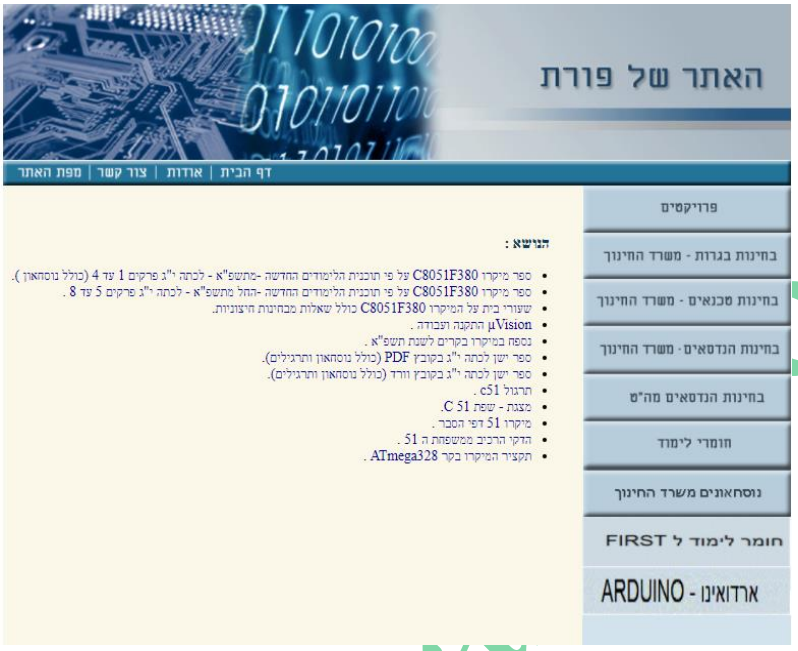
מיקרו בקר C8051F380 של SILICON LABS

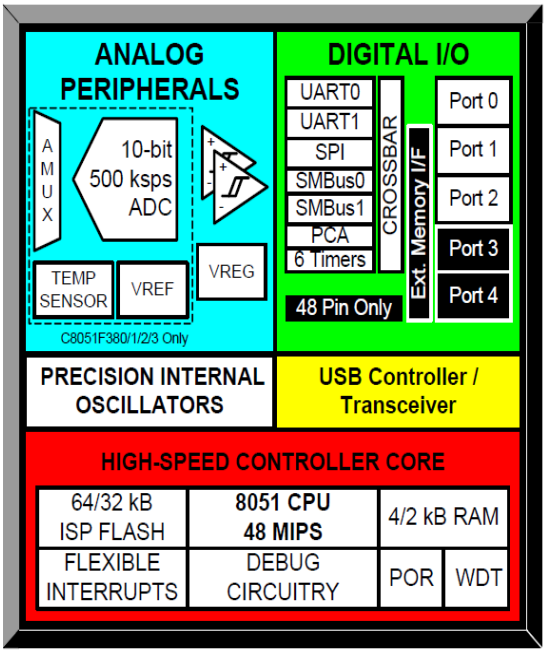
כתב וערך : אריה פורת

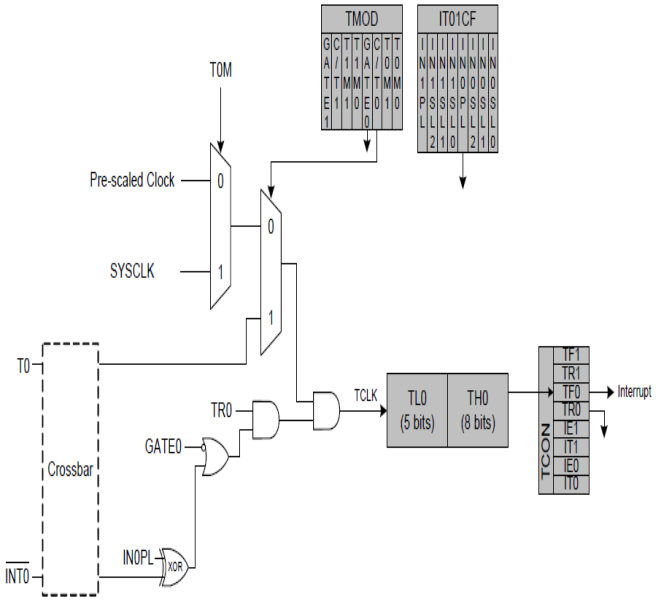
ספר לימוד לכתה יג : פרקים 5, 6, 7, 9 (על פי תוכנית הלימודים החדשה במיקרו בקרים החל מתשפ"ב)

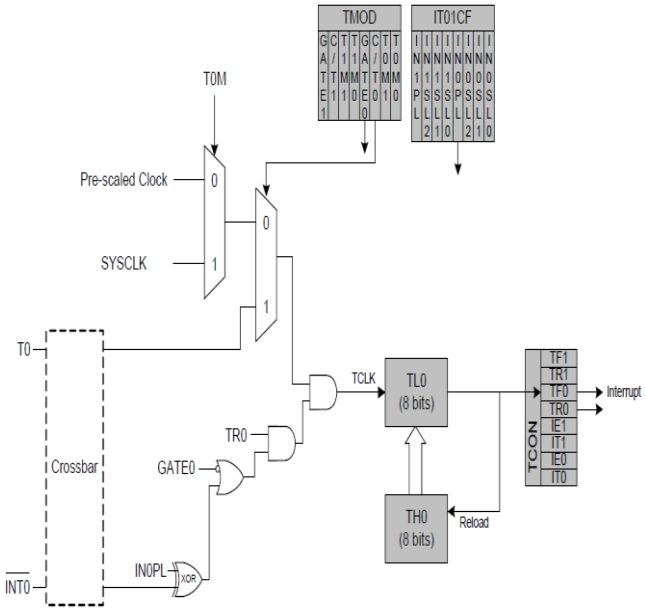
C8051F380/1/2/3/4/5/6/7/C

Full Speed USB Flash MCU Family









תוכן העניינים

פרק 5 : מבוא לשפת C וכתיבת תוכניות בסביבת µVision (פרק 4 בתוכנית הלימודים מתשפ"ב)

5.1 מבוא 5
5.2 מבנה השפה 5
5.3 טיפוסי משתנים 5
5.4 סוגי זיכרון 7
5.4.1 מודולים של זיכרון 8
5.5 טיפוסי משתנים בזיכרון 9
5.5.1 קביעה הזיכרון שבו יימצא המשתנה 9
5.5.2 משתנים נוספים שלא קיימים ב ANSI C 12
5.5.3 הגדרות טיפוסים נוספים 14
5.5.4 אופרטורים ועדיפות PRIORITY 14
5.5.5 מילים שמורות 16
5.6 משפטים בשפת C 16
5.6.1 משפטי השמה 16
5.6.2 תרשימי זרימה 16
5.6.3 משפטי תנאי 17
5.7 לולאות 24
5.7.1 לולאת while 24
5.7.2 לולאת do-while 25
5.7.3 לולאת for 26
5.7.4 תרגול 27
5.8 מערכים ARRAYS 28
5.9 מחרוזות STRINGS 28
5.10 פונקציה 29
5.10.1 הצהרה על פונקציה 30
5.10.2 העברת פרמטרים אל פונקציה 31
5.11 אפיון משתנים 31
5.11.1 משתנה אוטומטי 31
5.11.2 משתנה גלובלי 32
5.11.3 הכרזה על משתנה חיצוני 32
5.11.4 משתנה סטטי 32
5.12 מצביעים 32

5.13 פסיקות 33

5.14 כתיבת קוד אסמבלי בתוך שפת C 34

5.15 ההנחיה #include 35

5.15.1 – שיוך שם לכתובת 36

5.16 – קובץ Init_Device ותוכנת Configuration Wizard..... 37

5.16.1 המלצות עבור תוכניות שנרשום בהמשך..... 37

5.17 דוגמאות 38

5.18 חיבור מיקרו בקר אל תצוגת 7 מקטעים 44

5.18.1 מהי תצוגת 7 מקטעים ובאנגלית 7 Segments ? 44

5.18.2 2 אופני החיבור – אנודה משותפת וקטודה משותפת -C.A ו C.C 45

5.18.3 ממשק בין המיקרו לתצוגת 7 המקטעים 46

5.18.4 חישוב הזרם בכל מקטע 47

5.18.5 דוגמאות 48

15.18.6 חיבור תצוגות 7 מקטעים בריבוב TDM 50

15.18.7 חיבור 4 תצוגות 7 מקטעים בריבוב TDM 52

פרק 6 - פסיקות Interrupts (פרק 5 בתוכנית הלימודים מתשפ"ב)

6.1 פסיקה – הסבר כללי והשוואה לשאילתה (Polling) 57

6.2 חסימה / אפשור פסיקות ועדיפות פסיקות 58

6.3 סוגי פסיקות במיקרו בקר - מקורות פסיקה 59

6.4 הרגיסטרים שמטפלים בפסיקות 61

6.4.1 רגיסטרים לאפשור פסיקות 61

6.4.2 רגיסטרים לעדיפות הפסיקות IP Interrupt Priority 65

6.4.3 פסיקות חיצוניות ורגיסטר TCON 67

6.4.4 – רגיסטר IT01CF – לקביעת קוטביות המתח בפסיקות חיצוניות ... 68

6.5 תהליך הענות לפסיקה 70

6.6 – דוגמאות 72

פרק 7 : טיימרים/קאונטרים (פרק 6 בתוכנית הלימודים מתשפ"ב)

7.1 – מהו טיימר/קאונטר ? 80

7.1.1 : ההבדל בין טיימר לקאונטר 81

7.2 רגיסטרי הבקרה של הטיימרים/קאונטרים 82

7.2.1 רגיסטר ה TMOD 83

7.2.2 רגיסטר Timer CONtrol – TCON - בקרת הטיימר 84
7.2.3 רגיסטר ClocK CONtrol – CKCON בקרת שעון 85
7.3 אופני העבודה של הטיימרים 87
7.3.1 Mode 0 - אופן 0 88
7.3.1.1 דוגמאות 89
7.3.2 MODE 1 - אופן עבודה 1 90
7.3.2.1 דוגמאות 90
7.3.3 MODE 2 - אופן עבודה 2 91
7.3.1 דוגמה 91
7.3.4 MODE 3 - אופן עבודה 3 93
7.3.5 עבודה עם הטיימר כקאונטר (מונה) 94
7.3.6 דוגמאות 95

פרק 8 : תקשורת טורית – UART (פרק 9 בתוכנית הלימודים מתשפ"ב)

8.1 מבוא 99
8.1.1 אופני העברת נתונים 99
8.1.2 מהו UART ? 99
8.1.3 תקשורת אסינכרונית וסינכרונית 100
8.1.4 שידור נתון טורי 100
8.1.5 מאפייני השידור 101
8.2 התקשורת הטורית במיקרו C8051F380 102
8.2.1 UART0 102
8.2.2 סכמה מלבנית של UART0 103
8.2.3 רגיסטר בקרת התקשורת הטורית SCON0 105
8.2.4 עבודה בסביבה מרובת מעבדים 106
8.2.5 מחולל קצב הבאוד של ה UART 107
8.2.6 שגיאת תקשורת בגלל הבדל תדר בין המשדר למקלט 109
8.3 תוכנה 111
8.1.1 קליטה של תו 111
8.1.2 שידור של תו 112
8.4 תרגילי דוגמה 113
8.4.1 תרגיל 1 113
8.4.2 תרגיל 2 113
8.5 UART1 117

פרק 5 : מבוא לשפת C וכתיבת תוכניות בסביבת µVision

הערה : על התקנת סביבת העבודה µVision5 ניתן למצוא בקישור :

http://www.arikporat.com/lectures/micro/%b5Vision%20install%20and%20work.pdf

5.1 מבוא

את התוכניות למיקרו בקר ממשפחת ה 51 רשמו תחילה בשפת אסמבלי אבל עם השנים התוכניות נכתבו בשפת C או ליתר דיוק בשפת C51 . השפה C51 מבוססת על שפת C אבל מתאימה למיקרו בקרים ממשפחת ה 51 .

שפת C היא שפת על , ידידותית למשתמש, עובדת על סוגי מחשבים שונים , על מערכות הפעלה שונות, על מחשבים של יצרני מחשב שונים והיא משמשת בסיס לשפות מחשב אחרות. לשפה יש ספריות רבות שהמשתמש יכול להיעזר בהן כמו של קלט פלט מתמטיקה, גרפיקה , עבודה עם מערכים ומחרוזות ועוד.

כאשר כותבים תוכנית בשפת על ומבצעים תרגום של התוכנית לשפת מכונה (אפסים ואחדים) , התוכנית עוברת בשלב התרגום לקובץ אסמבלי וממנו לשפת מכונה. שפה עילית "חוסכת" מהמשתמש להכיר את מבנה המיקרו על שלל הרגיסטרים והזיכרונות שלו. שפת C עובדת בהרבה מקרים עם מעבדים ואילו C51 עובדת עם מיקרו בקרים ממשפחת ה 51 . בשפת אסמבלי המשתמש חייב להכיר את המבנה הפנימי של המעבד או המיקרו בקר. בשפת על המתכנת בהרבה מקרים לא יודע עם איזה מעבד הוא עובד ואילו זיכרונות מתחברים אליו.

5.2 מבנה השפה

כמו כל שפה , גם ב C51 יש טיפוסי משתנים , קבועים , משפטים של השפה, לולאות , מערכים ומחרוזות ושאר כללים שצריך ללמוד . ככל שנשלוט בכללים טוב יותר נוכל לפתח תוכניות חכמות ומהירות יותר.

את התוכנית אנחנו כותבים עם מעבד תמלילים/עורך מסך . הקובץ שנוצר הוא קובץ טקסט (Text File) . קובץ זה נשמור עם סיומת C והוא נקרא **קובץ מקור** (Source File) . כמובן שחייבים לרשום את הקובץ על פי כללי השפה כדי שהקומפיילר ידע לתרגם אותו. לקומפיילר קוראים בעברית **מהדר**.

קומפיילר "רגיל" לא יוכל לתרגם את קובץ המקור שלנו. רק קומפיילר שיודע לתרגם לשפת מכונה של מיקרו ממשפחת ה 51 יוכל לתרגם את הקובץ. קיימים 2 קומפיילרים נפוצים. האחד של KEIL אותו הכרנו בפרק הקודם ושהוי נקרא SDCC – Small Device C Compiler - קומפיילר ל C לרכיב קטן. שניהם טובים ויש הבדלים קטנים מאוד ביניהם. ה SDCC הוא בחינם וגם ה KEIL היא תוכנה חינמית עד 2 קילו בתים של תוכנית (קוד). משרד החינוך בחר להשתמש בתוכנת ה µVision של KEIL ואנחנו נסביר את הכללים על פי התוכנה הזו.

הערה: נצא מתוך הנחה ששפת C מוכרת לקורא. נעשה חזרה קצרה על השפה עם הרחבת מושגים הקשורים לשפת C51 .

5.3 - טיפוסי משתנים

למהדר – קומפיילר – ל Cx51 מספר טיפוסי נתונים בסיסיים שבהם משתמשים בתוכניות בשפת C. המהדר תומך בטיפוסי הנתונים הסטנדרטיים C, כמו גם במספר סוגי נתונים ייחודיים לפלטפורמת Cx51.

בטבלה הבאה מתוארים טיפוסי המשתנים השונים בשפת C51 .

Data Types/ טיפוס	Bits	Bytes	Value Range/ תחום ערכים	שם בעברית
bit	1		0 to 1	ביט
signed **char**	8	1	-128 — +127	תווי מסומן
unsigned **char**	8	1	0 — 255	תווי לא מסומן
enum	8 / 16	1 or 2	-128 — +127 or -32768 — +32767	קבוצה של קבועים
signed short **int**	16	2	-32768 — +32767	שלם קצר מסומן
unsigned short **int**	16	2	0 — 65535	שלם קצר לא מסומן
signed **int**	16	2	-32768 — +32767	שלם מסומן
unsigned **int**	16	2	0 — 65535	שלם לא מסומן
signed long **int**	32	4	-2147483648 — +2147483647	שלם ארוך מסומן
unsigned long **int**	32	4	0 — 4294967295	שלם ארוך לא מסומן
float	32	4	±1.175494E-38 — ±3.402823E+38	ממשי
double	32	4	±1.175494E-38 — ±3.402823E+38	ממשי כפול
sbit	1		0 or 1	ביט באזור ה SFR
sfr	8	1	0 — 255	רגיסטר של 8 ביטים ב SFR
sfr16	16	2	0 — 65535	רגיסטר של 16 ביטים ב SFR

טבלה 5.1 : טיפוסי משתנים ב c51 .

הערה : bit , sbit , sfr , sfr16 לא נמצאים בשפת C לפי ANSI C (American National Standard Institute – מכון הסטנדרטים האמריקאי הלאומי) אלא ייחודיים לקומפיילר של C51 .

הגדרה של משתנה בשפת C51 דומה להגדרת משתנה בשפת C רגילה :

טיפוס המשתנה שם המשתנה ;

דוגמאות :

unsigned char score; // הגדרת משתנה בשם score מטיפוס תווי לא מסומן

int counter; // הגדרה של משתנה בשם counter מטיפוס שלם

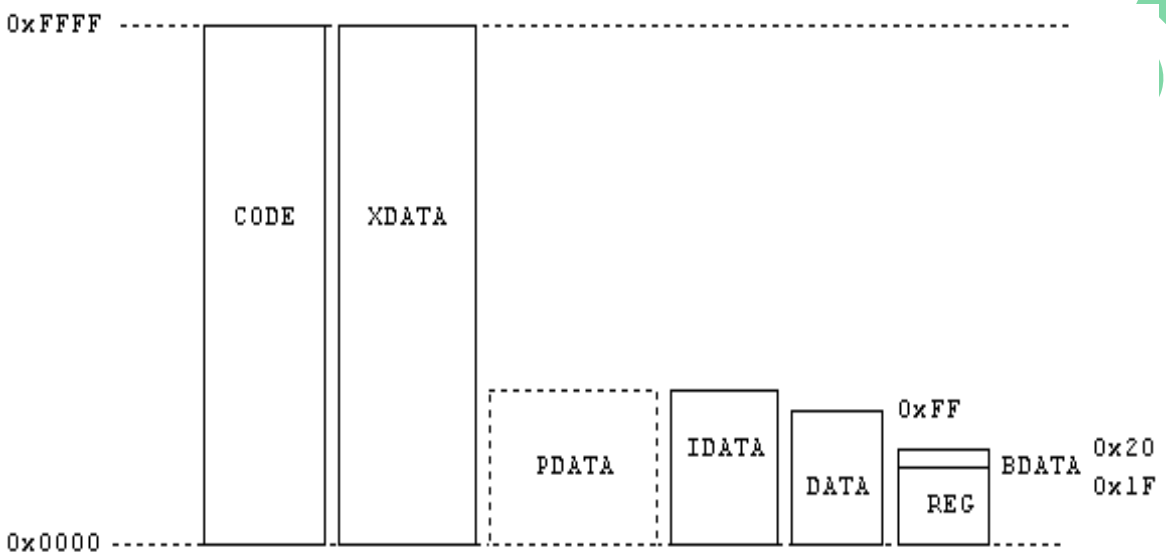
float salary; // הגדרת משתנה בשם salary מטיפוס ממשי

ב c51 מומלץ להשתמש - **במידת האפשר** - במשתנה מטיפוס unsigned char כי המיקרו בקר ממשפחת ה 51 הוא מיקרו של 8 ביט. משתנים מטיפוס אחר מגדילים את שורות הקוד ומאטים את מהירות התוכנית.

5.4 סוגי זיכרון

בשפת C "רגילה" , למשתמש אין אפשרות לקבוע לקומפיילר באיזה זיכרון ובאיזו כתובות יישמרו משתני התוכנית. מערכת ההפעלה היא זו שמקצה כתובות לכל משתנה (או קבוע) ולמתכנת אין שליטה בנושא. לעומת זאת בשפת C51 המשתמש יכול לקבוע באיזה סוג זיכרון יהיה המשתנה . (בקומפיילר SDCC ניתן גם לקבוע את הכתובת הרצויה בזיכרון הרצוי) . נכיר את סוגי הזיכרון השימושיים בשפת C51 .

למיקרו בקר ממשפחת ה 51 ישנם מספר סוגי זיכרון הנראים באיור 5.1 :



איור 5.1 : סוגי הזיכרון המיקרו בקר ממשפחת ה 51

code - זיכרון התוכנית - הקוד. יכול להגיע ל 64KB כאשר בחלק מהרכיבים יש 4KB או 8KB בתוך הרכיב (ב C8051F380 יש 64K בתוך הרכיב !) . היום יש רכיבים שהזיכרון אפילו גדול מ 64KB ומגיע ל 24 ביטים של כתובת כלומר $2^{24} = 2^4*2^{20} = 16M$ בתים. בזיכרון זה נמצאת התוכנית וקבועים שלא ניתן לשנות אותם. כשהקומפיילר מתרגם את הפקודה בשפת C הפונה למשתנה או קבוע בזיכרון הזה הוא ניגש עם פקודות movc .

xdata - זיכרון נתונים חיצוני בנפח של עד 64KB. ברכיבים הראשונים במשפחת ה 51 כל זיכרון הנתונים xdata שנקרא גם XRAM היה חיצוני. עם הזמן הכניסו גם חלק מזיכרון הנתונים XRAM לתוך הרכיב. ב C8051F380 יש 4 קילו בתים בתוך הרכיב. כיום יש רכיבים עם 24 ביטים של כתובות. כשהקומפיילר מתרגם את הפקודה בשפת C הפונה למשתנה כזה הוא ניגש עם פקודות movx בעזרת רגיסטר ה dptr.

pdata זיכרון נתונים חיצוני בן 256 בתים. הוא נחשב כמו xdata. למשתנים בשפת C51 המוגדרים באזור זה הפקודות לאסמבלי מתורגמות לפקודות movx עם @R0 או @r1. (היום זה פחות שימושי)

idata זיכרון נתונים פנימי (ברכיבים המסתיימים בספרה 2 או 3 – כדוגמת 89S52 וכמובן ברכיב C8051F38x) בתחום הכתובות מ 80h ועד ffh (128 עד 255) . הגישה למשתנים בשפת C51 המוגדרים באזור זה היא **במיעון עקיף** בעזרת הרגיסטרים r0 או r1 עם פקודות @r0 או @r1.

data זיכרון הנתונים הפנימי בתחום הכתובות מ 0 עד 7fh (0 עד 127). זוהי **ברירת המחדל** של הקומפיילר למיקום משתנים אם המשתמש לא הגדיר באיזה זיכרון יהיו המשתנים. כאשר הקומפיילר מתרגם פקודה בשפת C למשתנה באזור ה data הפקודה יכולה להיות גם ישירה וגם עקיפה. לדוגמה : mov 56h,#34h או עקיפה בעזרת @r0 או @r1 .

bdata - משתנים המוצהרים כ bdata ממוקמים בזיכרון ה data בתחום הכתובות מ 20h עד 2fh. זהו אזור של 16 בתים המחולק לביטים . באזור זה ניתן לפנות גם לבית וגם לביט מסוים.

reg 32 הבתים הראשונים בזיכרון הנתונים הפנימי. משתנים בשפת C51 המוגדרים באזור זה יישמרו באזור 4 הבנקים בתחום הכתובות 0 עד 31 . בכל בנק 8 רגיסטרים מ r0 ועד r7 .

far - במיקרו בקרים מסוימים ניתן לחבר זיכרונות בנפח גדול מ 64K שהיה במיקרו 51 המקורי. לכתובות של 24 ביט ניתן להגדיר משתנים באזור זה .

5.4.1 מודלים של זיכרון

קומפיילר של c51 תומך ב 3 מודלים של זיכרון: א. Small (קטן) ב. Compact (קומפקטי -דחוס/מרוכז) ג. Large - (רחב) .
לקוראים שלא מכירים שפת C - ניתן לקפוץ על פיסקה זו. יש כאן מושגים שטרם למדנו.
מודל הזיכרון קובע את סוג הזיכרון שאיתו נעביר ארגומנטים לפונקציה (הפרמטרים האקטואליים) , שבו יהיו המשתנים האוטומטיים של התוכנית והצהרות שלא הוגדר בהן במפורש סוג הזיכרון.
בטבלה 5.2 מתוארים 3 המודלים של הזיכרון ובאיזה זיכרון משתמשים במודל.

מודל הזיכרון	פרמטרים ומשתנים אוטומטים	ברירת המחדל משתנים גלובליים	ברירת המחדל קבועים ומשתנים	ברירת המחדל הגדרת המצביע	ברירת המחדל של המצביע
Memory Model	**Parameters & Automatic Variables**	**Default Global Variables**	**Default Constant Variables**	**Default Pointer Definitions**	**Default Pointer Size**
SMALL	data	data	data	*	3 bytes
COMPACT	pdata	pdata	pdata	*	3 bytes
LARGE	xdata	xdata	xdata	*	3 bytes

טבלה 5.2 : 3 המודלים של הזיכרון .

ברוב התוכניות שנשתמש מודל small הוא מספיק טוב.

5.5 טיפוסי משתנים בזיכרון - STORAGE CLASS LANGUAGE EXTENSIONS

להבדיל משפת C "רגילה" , ב C51 המשתמש יכול לקבוע באיזה זיכרון יימצא המשתנה.

הגדרת המשתנים תתבצע כך :

< **טיפוס המשתנה** > < **סוג הזיכרון**> < **שם המשתנה** > ;

דוגמה :

unsigned char data sensor; // הגדרת משתנה בשם sensor מטיפוס תווי לא מסומן שנמצא באזור ה data
int xdata timer; // הגדרה של משתנה בשם timer מטיפוס שלם
float code PIE=3.1428 // הגדרה של קבוע (כי הוא בזיכרון הקוד ולא ניתן לשנות אותו) בשם PIE מטיפוס ממשי

קומפיילר של KEIL או SDCC יקבל הגדרה של משתנה ב 2 האפשרויות הבאות :

<**טיפוס משתנה**> <**סוג הזיכרון**> <**שם משתנה**> ; **או** < **טיפוס משתנה** > <**סוג הזיכרון**> <**שם משתנה**>;

לדוגמה:

xdata unsigned char external_var1;

unsigned char xdata var2;

5.5.1 קביעה הזיכרון שבו יימצא המשתנה :

איזור הנתונים הפנימי מתחלק ל 3 תת אזורים : א. data ב. idata ג. bdata

5.5.1.1 Data (או near) : משתנים שהוגדרו באזור זה נמצאים ב RAM הפנימי בין הכתובות 0 עד 127 (0 עד 7fH).

הגישה אליהם בתרגום לאסמבלי היא עם מיעון ישיר או עקיף.

דוגמה : unsigned char data myGrade;

5.5.1.2 idata משתנים שהוגדרו עם המילה idata ימוקמו בחלק העליון של ה RAM הפנימי. (הכתובות מ 128 עד 255) . כידוע, הגישה לכתובות אלו תהיה במיעון עקיף עם רגיסטר R0 או R1 .

דוגמה: int idata x;

5.5.1.3 bdata (עובד עם קומפיילר של KEIL)

הגדרה של משתנה באזור ה RAM הפנימי בין הכתובות 0x20 עד 0x2f (סה"כ 16 כתובות). אזור זה מחולק לביטים מ0 ועד 0x7f (סה"כ 128 ביטים). ניתן להתייחס אליהם כביט או כמילה.

הגדרה : **<טיפוס המשתנה> bdata <שם המשתנה> ;**

דוגמה:

char bdata test ; // הגדרה של משתנה תווי בשם test שיהיה באזור ה bdata בין הכתובות 20h עד 2fh

5.5.1.4 אזור הנתונים החיצוני XRAM שגודלו יכול להיות 64Kbytes (ב C80C51F380 יש 4KBytes פנימי וניתן להוסיף עוד 60KB חיצוני) מתחלק ל 2 חלקים.

5.5.1.5 xdata - הגדרה של משתנה באזור זה מאפשרת גישה לזיכרון ה RAM החיצוני .

בתרגום לאסמבלי של משתנה מאזור זה משתמשים ברגיסטר ה dptr כמצביע לכתובות ובאקומולטור.

xdata <טיפוס משתנה> <שם משתנה> ; או < טיפוס משתנה> xdata <שם משתנה >;

לדוגמה :

xdata int counter ; // xram הגדרה של משתנה בשם counter שנמצא באזור ה

5.5.1.6 pdata (Paged xdata) זוהי הגדרה המאפשרת גישה למשתנים בזיכרון הנתונים החיצוני בעזרת R0 או R1 (ולא כמו שבדרך כלל עם ה dptr !!). בדרך כלל מיקום המשתנים הוא בתחילת זיכרון ה RAM החיצוני ויש אפשרות למקסימום 256 בתים. בעזרת R0 או R1 יכולים לגשת ל 256 כתובות בלבד (החלק הנמוך של הכתובות) והחלק הגבוה תלוי בפורט 2 .

pdata <טיפוס משתנה> <שם משתנה> ; או < טיפוס משתנה> pdata <שם משתנה >;

דוגמא pdata unsigned char p_var;

5.5.1.7 code . משתנים המוגדרים עם המילה code ממוקמים בזיכרון התוכנית. בדרך כלל משמשים לטבלאות המרה וקבועים.

< טיפוס משתנה > code < שם משתנה > = < קבוע > ;

או code < טיפוס משתנה > < שם משתנה > = < קבוע > ;

דוגמא unsigned char code c_var= 'a';

יש לאתחל את המשתנה (לא ניתן לכתוב אליו ערך מאוחר יותר !). משתנה כזה ניתן רק קריאה. לדוגמא - הגדרה של מחרוזת באזור זה :

char code str[ ]="hello";

5.5.1.8 far במשפחת ה 51 יש נגזרות של חברות שונות שבהן זיכרון הקוד והנתונים יכולים להיות 16Mbytes (חברות כמו פיליפס ודאלאס) . הקומפיילר של KEIL תומך גם בזיכרונות אלו .

far <טיפוס משתנה> <שם משתנה> ; או < טיפוס משתנה> far <שם משתנה >;

דוגמה :

Char far test[ ] = "Enter a number : " ; // far הגדרה של מחרוזת בשם test ואתחול המחרוזת. המחרוזת נמצאת בזיכרון

5.5.1.9 נסכם את המשתנים באזורי הזיכרון השונים בעזרת האיור הבא . ניתן לראות עם איזה רגיסטרים פונים אליהם באסמבלי.

טיפוס הזיכרון	תיאור
code	Program memory (64 KBytes); accessed by opcode MOVC @A+DPTR.
data	Directly addressable internal data memory; fastest access to variables (128 bytes).
idata	Indirectly addressable internal data memory; accessed across the full internal address space (256 bytes).
bdata	Bit-addressable internal data memory; supports mixed bit and byte access (16 bytes).
xdata	External data memory (64 KBytes); accessed by opcode MOVX @DPTR.
far	Extended RAM and ROM memory spaces (up to 16MB); accessed by user defined routines or specific chip extensions (Philips 80C51MX, Dallas 390).
pdata	Paged (256 bytes) external data memory; accessed by opcode MOVX @R*n*.

איור 5.2 : סוגי הזיכרון ובאילו רגיסטרים פונים למשתנים ב C51 בתרגום לאסמבלי.

דוגמאות נוספות להגדרת משתנים :

```
char data var1;
char code odaa[ ] = "Hello World";
unsigned long xdata arr[20];
int xdata myArray[8] [5] [5];
char bdata myFlags;
```

5.5.1.10 כתובות מוחלטות - השימוש במילה _at_

בקומפיילר SDCC ניתן להגדיר את המיקום (הכתובת שנרצה שהמשתנה יהיה בו) על ידי שימוש במילה **at** האומרת לקומפיילר באיזה כתובת לשים את המשתנה. גם קומפיילר של **KEIL** יקבל את ההגדרה אבל יש לרשום **_at_** (עם קו תחתון לפני ואחרי) והמשתנה חייב להיות גלובלי **ולא ניתן לאתחל אותו בערך ראשוני**. לביטים ופונקציות לא ניתן לקבוע כתובת. :

<טיפוס המשתנה> <סוג הזיכרון> <שם המשתנה> _ at_ < כתובת המשתנה > ;

דוגמאות:

int xdata myVar _at_ 0x8000; // myVar ושמו 8000h,8001h הגדרה של משתנה מטיפוס שלם שיהיה בכתובת

unsigned char data array1[0x20] _at_ 0x40 ; // הפנימי RAM הגדרת מערך בגודל 20 הקסה איברים ב

קומפיילר SDCC יקבל את 3 ההגדרות הבאות:

<סוג הזיכרון> < טיפוס המשתנה > at < כתובתו של המשתנה > < שם המשתנה > ;

<טיפוס המשתנה> <טיפוס הזיכרון> at < כתובתו של המשתנה > < שם המשתנה > ;

<סוג הזיכרון> at < כתובתו של המשתנה > < טיפוס המשתנה > < שם המשתנה > ;

5.5.2 משתנים נוספים שלא קיימים ב ANSI C

5.5.2.1 bit

משתנה המוגדר כביט ממוקם בזיכרון מיעון הביטים ב RAM הפנימי בכתובות 32 עד 47 (20h-2fh). סה"כ יש 128 ביטים מ 0 ועד 127 (מ 0 ועד 7fh) .

הגדרה: **bit <שם המשתנה>**

דוגמא : bit test ;

אם רוצים לשים בביט זה '1' נרשום : test=1;

אם רוצים להפוך את מצב הביט : test = ! test

הערות:

א. לא ניתן להגדיר ביט כמצביע (פוינטר).

ב. לא ניתן להגדיר מערך של ביטים.

ג. ניתן שהערך המוחזר מפונקציה יהיה ביט אבל לא ניתן להחזיר ערך של ביט מפונקציה שהוגדרה בעזרת המילה using (שאומרת עם איזה בנק לעבוד) או פונקציות פסיקה שנחסמו עם המילים #pragma disable .

לדוגמה פונקציה המקבלת ערכי ביטים ומחזירה ביט :

```
bit bitFlagTest (bit flag1,bit flag2)
 {
   if(flag1 && flag2)
         return (1);
   else
         return (0);
}
```

5.5.2.1 sbit

הגדרה של ביט באחד הרגיסטרים באזור הרגיסטרים לתפקידים מיוחדים sfr (כתובות 80H עד ffH). ישנן 3 אפשרויות להגדרה (כמשתנים גלובליים) :

א. sbit <שם המשתנה> = <שם הרגיסטר> ^ <מיקום הביט>;

לדוגמה :

sbit led = P1 ^ 6 ; // (P1.6) הגדרה של משתנה בשם led המחובר אל המיקום 6 של פורט 1 או בעברית פשוטה:

ב. sbit <שם המשתנה> = <הכתובת של הרגיסטר> ^ <מיקום הביט>

דוגמאות : sbit mafsek = 0x90 ^ 1 ; // הגדרה של משתנה בשם מפסק המתחבר ל P1.1

sbit EA = 0xA8^7; //(IE רגיסטר) 0xA8 הגדרת ביט בשם EA שנמצא בביט מספר 7 של הרגיסטר שנמצא בכתובת

ג. **sbit < שם המשתנה > = <כתובת הביט>;**

דוגמה : // שיוך משתנה בשם led לביט בכתובת 90 הקסה ב SFR וזו כתובת sbit led=0x90;// P1.0

שיוך המילה CY לביט בכתובת d7H באזור ה SFR . זו למעשה כתובת ביט ה sbit CY = 0xd7; // CarrY

שיוך משתנה בשם stam לביט באזור ה SFR שכתובתו sbit stam = 0xd6; // d6H

הערה: לא לכל הרגיסטרים באזור ה sfr יש ביטים הניתנים למיעון .רק לאלו שכתובתם מתחלקת ב 8 !!

ביט לא יכול להיות מוגדר כמצביע (pointer) או כמערך !

5.5.2.2 sfr

הגדרה של אחד הרגיסטרים באזור הרגיסטרים לתפקידים מיוחדים – sfr . זהו האזור בין הכתובות 0x80 עד 0xff בזיכרון ה RAM הפנימי. משתנים מסוג זה יהיו גלובליים בלבד. ההגדרה היא כמו של כל משתנה בשפת C רק במקום int או char רושמים את המילה sfr. כמו כן יש לציין מספר ולא שם משתנה או פעולה חשבונית או לוגית.

התחביר :

sfr < שם המשתנה > = < כתובת הרגיסטר >;

דוגמאות:

sfr P1 = 0x90 ;

sfr IE = 0xA8;

שיוך השם P1 (פורט 1) לכתובת 0x90 באזור ה sfr ושל המילה IE לכתובת 0xa8 שהיא כתובת רגיסטר אפשור הפסיקות IE .

5.5.2.3 sfr16

הגדרות ה sfr הן טיפוסי נתונים המתארים את הרגיסטרים שלהם 2 בתים (כמו הטיימרים ה dptr וכו') שבאזור הרגיסטרים לתפקידים מיוחדים SFR – Special Function Registers .

הגדרה: (עובד עם הקומפיילר של **KEIL**)

sfr16 < שם המשתנה > = < כתובת > ;

דוגמא:

sfr16 T2 = 0xCC; // Timer 2: T2L 0CCh, T2H 0CDh

עדיף לא להשתמש ב sfr של 16 ביט ו 32 ביט הדורשים סדר גישה מסוים למרות שקומפיילר sdcc בדרך כלל ניגש קודם לביית ה LSB ואח"כ לביית ה MSB .

5.5.3 הגדרות טיפוסים נוספים

5.5.3.1 הגדרת קבוע - const

הגדרת ערך קבוע שלא ניתן לגשת אליו ולשנות ערכו.

הגדרה : **const < טיפוס המשתנה > שם המשתנה = < ערך > ;**

דוגמה : const float pi = 3.14;

5.5.3.2 Volatile

יש להגדיר משתנה כ volatile כאשר הערך שלו יכול להשתנות בצורה לא צפויה.

הגדרה : **volatile < טיפוס המשתנה > < שם המשתנה > ;**

לדוגמא נסתכל בקוד :

```
int i=0
while( i < 3  ) ;
```

בקטע הזה אתחלנו את המשתנה i בערך 0 . הקומפיילר "רואה" שהמשתנה הוא 0 ואין פקודה "בסביבה" שמשנה אותו ולכן הוא יחשוב שמדובר בלולאה אין סופית וירשום במקום הפקודה הזו פקודות המתאימות ל while (1) . (למעשה האופטימייזר של הקומפיילר ירשום פקודה זו). המילה volatile לפני הגדרת המשתנה אומרת לקומפיילר לא לעשות אופטימיזציה אלא לבצע תרגום מדויק של הפקודה. מה ההיגיון לרשום volatile ? הרי לכאורה המשתנה הוא אפס ולא נראה שהוא ישנה את מצבו ?

כשעובדים עם מיקרו בקרים יכולים לעבוד עם פסיקות ובפסיקה ניתן לשנות את ערכו של i. נראה דוגמה שבה הערך i יכול להשתנות. בתוכנית הפסיקה ניתן לרשום שיש להגדיל את המשתנה i . אחרי 3 פעמים שנהיה בתוכנית הפסיקה I יהיה 3 ונעזוב את לולאת ה while .

5.5.4 אופרטורים

אופרטור הוא פעולה חשבונית או לוגית. הפעולה מתבצעת על אופרנדים שהם משתנים או קבועים. לדוגמה : c=a+10; . ה + הוא האופרטור ו a ו 10 הם האופרנדים. הטבלה הבאה מראה את האופרטורים השונים.

אופרטור מתמטי	תיאור	אופרטור להשוואה	תיאור (התשובה היא 0 או 1 – TRUE / FALSE)
+	חיבור	= =	האם שווה (**שני** סימני שווה)
-	חיסור	! =	לא שווה
*	כפל	>	גדול מ-
/	חילוק	<	קטן מ-
%	שארית (מודולו)	> =	גדול או שווה ל-
		< =	קטן או שווה ל-

אופרטור לוגי	תיאור : עבודה על בתים או מילים (התשובה 0 או 1)	אופרטורים העובדים על ביטים	תיאור
!	היפוך	!	היפוך הביט
&&	וגם (and)	&	AND בין הביטים של שני האופרנדים
||	או (or)	|	OR בין הביטים של שני האופרנדים
		^	XOR בין הביטים של שני האופרנדים
++	קידום ב 1	>>	הזזה ימנית
- -	הפחתה ב 1	<<	הזזה שמאלית

טבלה 5.3 : אופרטורים מתמטיים לוגיים והשוואה.

5.5.4.1 קדימות – עדיפות – PRIORITY

כאשר מבצעים פעולות חשבוניות או לוגיות, קיימת עדיפות של סדר פעולות. העדיפות נתונה בטבלה הבאה. רואים שפעולות חשבוניות בעדיפות גבוהה מלוגיות. לאופרטורים - + * בצורה האונרית (פעולה על אופרנד אחד כמו x++) יש עדיפות על השימוש בהם בצורה הבינארית (פעולות על שני אופרנדים).

סדר ביצוע	אופרטורים (פעולות חשבוניות/לוגיות/השוואה)
משמאל לימין	( ) [ ] . -> n++ n--
מימין לשמאל	++ n -- n ! ~ sizeof (type) + - & *
משמאל לימין	* / %
משמאל לימין	+ -
משמאל לימין	<< >>
משמאל לימין	< <= > >=
משמאל לימין	== !=
משמאל לימין	&
משמאל לימין	^
משמאל לימין	|
משמאל לימין	&&
משמאל לימין	||
מימין לשמאל	? :
מימין לשמאל	= += *= <<= >>= /= %=
משמאל לימין	,

טבלה 5.4 סדר עדיפות בפעולות חשבוניות ולוגיות.

5.5.5 מילים שמורות

מילים שמורות (reserved words או key words) הן מילים ששייכות לדקדוק/לתחביר של השפה והן "שמורות" כאבן בנין לשפה. אסור למשתמש לקרוא במילים אלו לשמות של משתנים , קבועים , פונקציות וכו'.

הטבלה הבאה מתארת מילים שמורות בשפת C51 לקומפיילר של KEIL . כמובן שמילים שמורות אלו נוספות למילים השמורות של שפת C "רגילה" (ANSI C).

at (SDCC)	_at_ (KEIL)	bdata	bit	code	Compact
data	far	idata	interrupt	large	Pdata
priority	reentrant	sbit	sfr	sfr16	Small
task	using	xdata			

טבלה 5.5 : מילים שמורות לקומפיילר KEIL .

5.6 משפטים בשפת C

1. כל משפט בשפת C מסתיים בנקודה פסיק (;).
2. אין חשיבות לכמות הרווחים בין מילים או משפטים. לדוגמה : a=10; הוא כמו a = 10 ; .
3. ניתן לרשום מספר משפטים בשורה אחת. לדוגמה: a=10; b=20; c=a+b; .
4. ניתן לתת למשתנה שם עם אותיות גדולות או קטנות.
5. שמות של משתנים לא יכולים להתחיל בספרה. לדוגמה: int 7Seg; היא הגדרה לא חוקית.
6. לא ניתן לרשום שם של משתנה עם רווח בין מילים. לדוגמה : int lcd data; הוא שם לא חוקי. int lsdData; הוא חוקי.
7. ניתן לרשום קו תחתון בשם של משתנה. int lcd_data; הוא שם חוקי.
8. כדאי לתת למשתנה שם עם משמעות לקריאות טובה יותר של התוכנית. לדוגמה אם משתנה משמש כמונה int counter;
9. מומלץ לתת למשתנה עם שם המורכב ממספר מילים תו באותיות גדולות בתחילת כל מילה חדשה. לדוגמה : int counterOfWords; הוא חוקי.

5.6.1 – משפטי השמה

בשפת C רושמים משפט השמה בצורה כזו שמתבצעת השמה (העברה) מהאופרנד הימני אל השמאלי . לדוגמה :

int a=10, b =20,c; // ביצוע השמה / העברה למשתנה a את הערך 10 ולמשתנה b את הערך 20

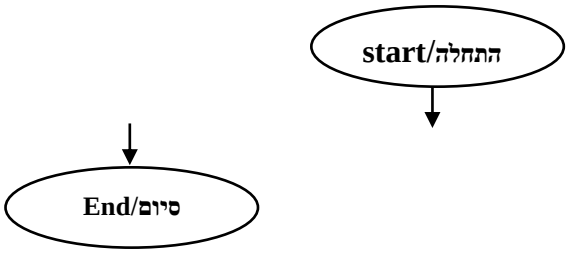
c=a+b; // ביצוע השמה/העברה של תוצאת החיבור בין הערך ב a והערל ב b למשתנה c

5.6.2 תרשימי זרימה

תרשים זרימה של תוכנית הוא שפה גרפית לתאר את מהלך ריצת התוכנית.

ישנם מספר סמלים נפוצים בתרשים הזרימה . החיצים מראים את כיוון זרימת התוכנית.

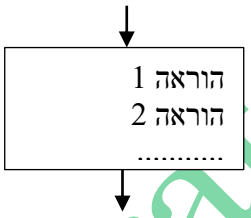
5.6.2.1 אליפסה : התחלה או סיום .



אפשר לרשום בעברית או אנגלית.

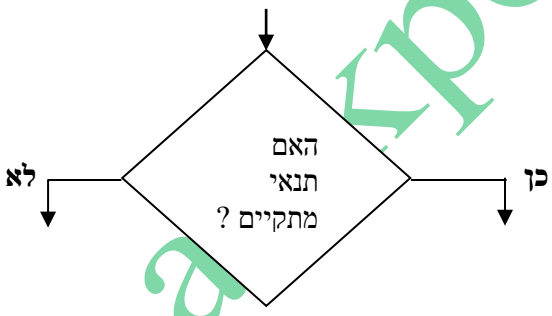
איור 5.3 : תרשים זרימה של התחלה וסיום

5.6.2.2 מלבן : בלוק הוראות - הוראה או סדרה של הוראות.



איור 5.4 : תרשים זרימה של מלבן הוראות

5.6.2.3 מעוין : נקודות החלטה / תנאי



איור 5.5 : תרשים זרימה של נקודת החלטה

בנקודת החלטה בודקים האם תנאי מתקיים . אם כן פונים לכיוון אחד ואם לא לכיוון ההפוך. רושמים כן או לא על כל זרוע של כיוון כדי שנדע לאן התוכנית עוברת. ניתן לרשום רק כן או רק לא על כיוון מסוים ואז יודעים שהכיוון השני הוא ההפך.

דוגמה לתנאי יכול להיות if (a!=0) כדי לדעת האם לחלק ב a או לא.

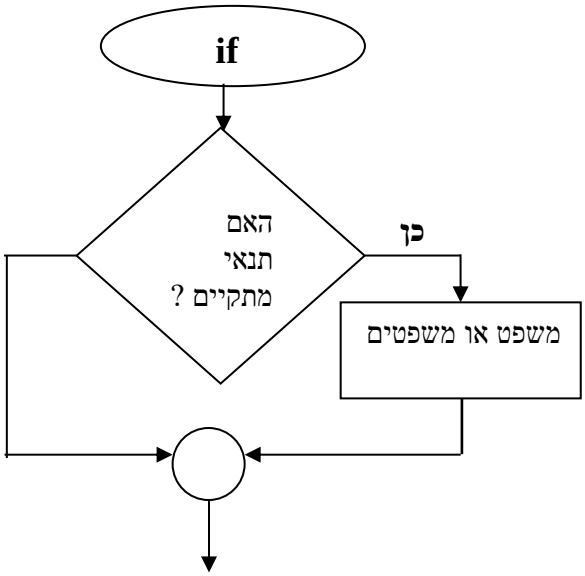
ישנם סמלים נוספים אבל בספר זה נשתמש בסמלים אלו בלבד.

5.6.3 משפטי תנאי

משפט תנאי הוא משפט עם תנאי. אם התנאי מתקיים מבצעים משפט/משפטים מסוימים. משפט תנאי מחזיר או TRUE (נחשב כ 1) או FALSE (נחשב כ 0).

5.6.3.1 תבנית הפקודה if :

במשפט if בודקים האם תנאי מתקיים. אם כן מבצעים משפט / משפטים. אם התנאי לא מתקיים לא מבצעים שום דבר.



איור 5.6 תרשים זרימה של תנאי if

התחביר :

```
if  ( תנאי )
        משפט ;
```

אם התנאי מתקיים עוברים למשפט הבא. אם התנאי לא מתקיים עוברים לשורה אחרי המשפט.

אם יש מספר פקודות לביצוע תוחמים אותן בעזרת סוגריים מסולסלים:

```
if ( תנאי )
        {
                1 משפט ;
                2 משפט ;
                ……..
        }
```

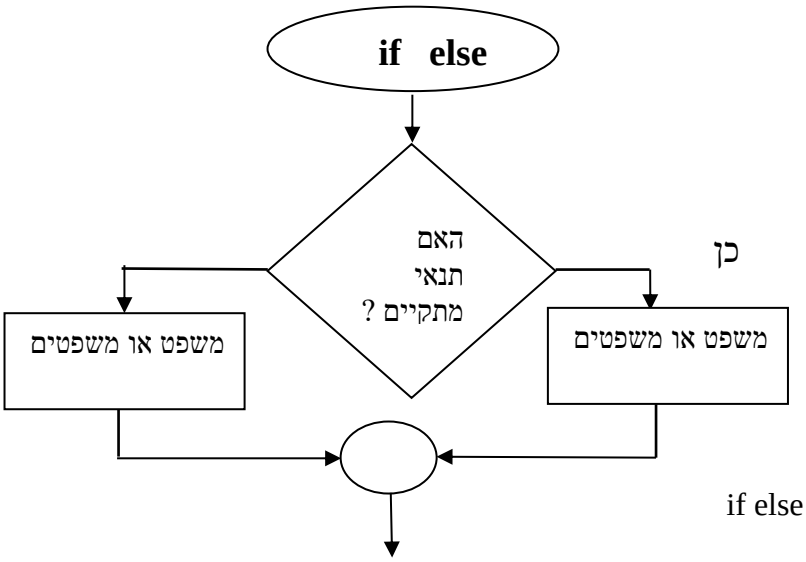
לדוגמה:

```
if ( a>b)
        printf("a is bigger than b");
```

התנאי בדק האם a>b . אם כן מדפיסים ש a גדול מ b . אם b יותר גדול או שווה ל a ממשיכים הלאה בתוכנית.

5.6.3.2 if else

משפט if else הוא משפט שבו בודקים האם תנאי מתקיים. אם כן מבצעים משפט/משפטים. אם התנאי לא מתקיים מבצעים משפט/משפטים אחרים.



איור 5.7 : תרשים זרימה של if else

התחביר :

```
if (תנאי)
        משפט ;
else
        משפט ;
```

דוגמה:

```
if ( a>b)
        printf("a is bigger than b")
else
        printf("b is bigger or equal to a");
```

אם יש מספר פקודות לביצוע :

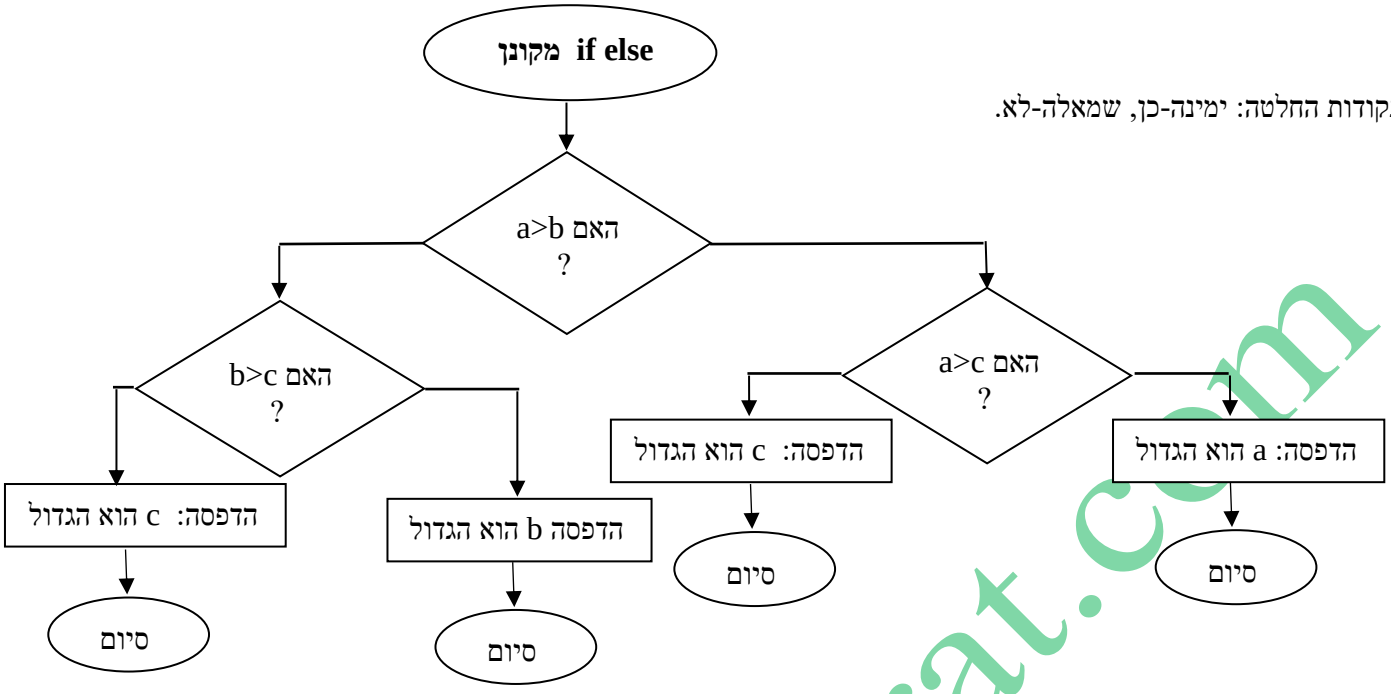
```
if <תנאי>
        {
                משפט 1 ;
                משפט 2 ;
                ……..
        }
else
        משפט ;
```

5.6.3.3 if-else מקונן

המילה מקונן באה מהמילה קן (של ציפורים). זהו משפט אם מספר if ומספר else .

דוגמה : נניח שיש לנו 3 מספרים לא שווים הנמצאים במשתנים a, b ו c . נמצא מיהו המספר הגדול משלושתם.

בנקודות החלטה: ימינה-כן, שמאלה-לא.



איור 5.8 : דוגמה לתרשים זרימה של if else מקונן של מציאת מספר גדול מבין 3 מספרים.

ובשפת C :

```
if (a>b)
        if(a>c)
                printf("a is the biggest number);
        else
                printf("c is the biggest number);
else if (b>c)
        printf("b is the biggest number);
else
        printf("c is the biggest number);
```

5.6.3.4 משפט הצבה מותנה

תחביר **(תנאי) ? משפט 1 : משפט 2 ;**

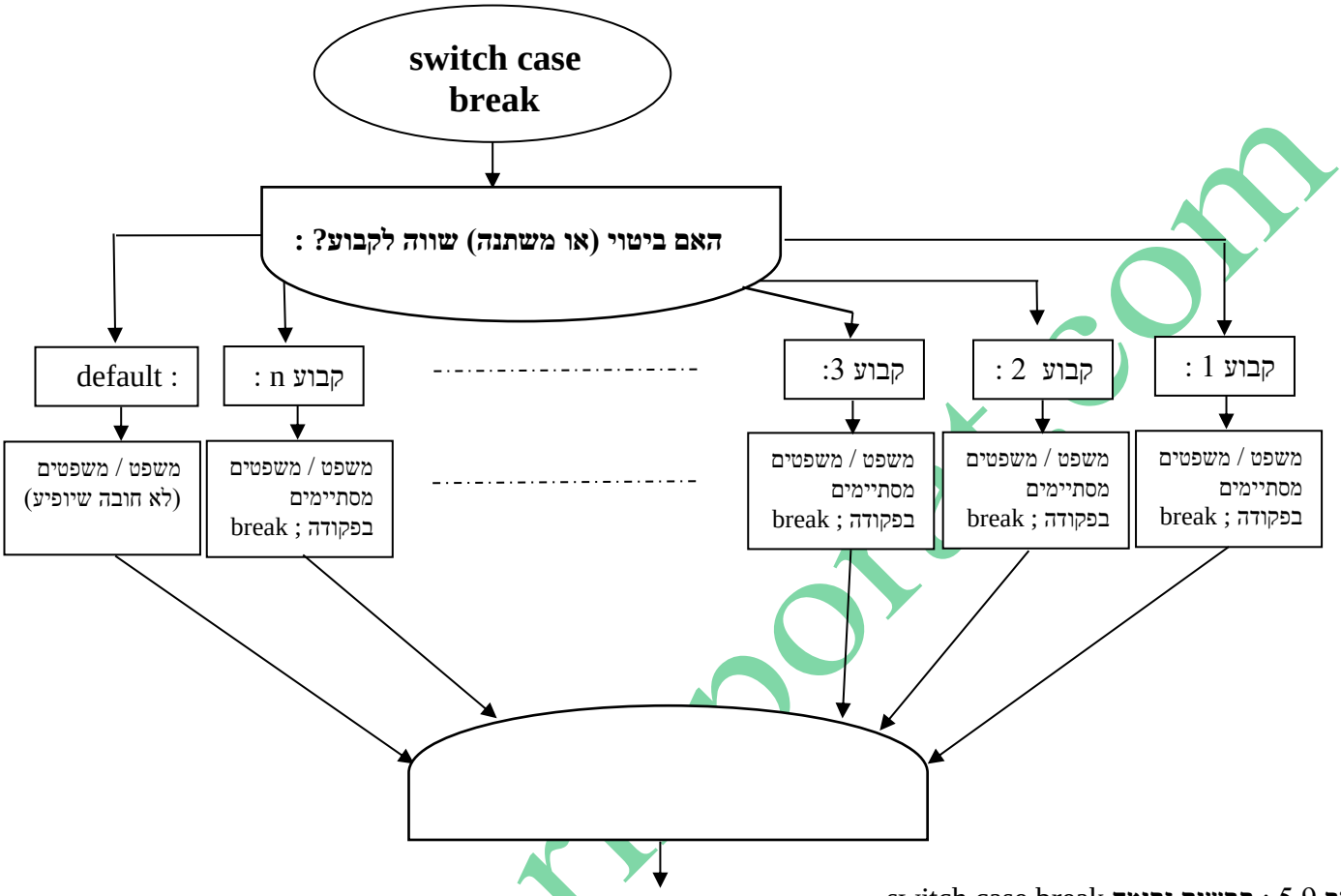
אם התנאי מתקיים מתבצע משפט 1 . אם התנאי לא מתקיים מתבצע משפט 2 .

דוגמה :

```
(a>b) ? printf("a is bigger than b") : printf ("b is bigger or equal to a")
```

switch - case - break 5.6.3.5

כאשר יש הרבה משפטי if else מקוננים ניתן להשתמש במשפט הצבה מותנה שמחליף את כל ה else if .



איור 5.9 : תרשים זרימה switch case break

התחביר :

```
switch ( ביטוי )
        {
                case  1 קבוע :
                        משפט ;
                        משפט ;
                        ……..
                break ;
                case  2 קבוע :
                        משפט ;
                        משפט ;
                        ……..
                break ;
```

```
        case   3 קבוע :
                 משפט ;
                 משפט ;
                ……..
        break ;

        ┆
        ┆
        default :
                 משפט ;
                 משפט ;
    }
```

דוגמה :

```
int a=10,b=20,choice;  //  הגדרה של 3 משתנים מטיפוס שלם ואתחול של 2 המשתנים הראשונים
printf("\t*** MENU***\n1.Add\n2.Subtract\n3.Multiply\n4.Divide"); //  הדפסת מסך תפריט
printf("\nPlease enter your choice (1 – 4 ): "); // הדפסה : נא הכנס בחירתך
scanf("%d",&choice); //  קליטה של מספר
switch(choice):  //  ? שווה לקבוע הבא choice  האם
{                                                   //      ***MENU***
case 1: // ? 1 שווה ל choice  האם                   //   1. Add
        printf("a+b = %d+%d = %d",a,b,a+b);         //   2.Subtract
break;                                              //   3.Multiply
case 2: // ? 2 שווה ל choice האם                    //   4. Divide
        printf("a-b = %d -%d = %d",a,b,a-b);        //   Please enter your choice (1 – 4) :
break;                                              //  a-b = 10 – 20 = -10
case 3:  // ?  3 שווה ל choice האם
        printf("a*b = %d*%d = %d",a,b,a*b);
break;
case 4:   // ?  4 שווה ל choice האם
        printf("a/b = %d/%d = %f",a,b,(float)a/b);   // 10/20 = 0.500
break;
                                //        casting
default :
        printf("Wrong Choice");
}
```

5.6.3.6 משפט goto

מעבר אל קטע תוכנית אחר .

תחביר : goto < תווית > ;

דוגמא :

```
goto sof ;
    ⋮
sof:
    ⋮
```

5.6.3.7 משפט break

יציאה מתוך משפט הבקרה והפסקת הלולאה שבה נמצאים.

התחביר : break ;

5.6.3.8 משפט continue

הפסקת ביצוע משפטי הלולאה ודילוג לראש הלולאה לביצוע נוסף.

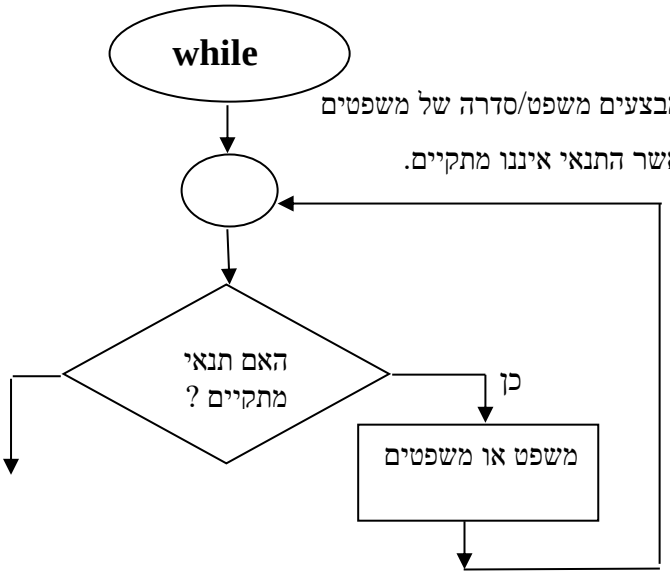
התחביר : continue ;

5.7 לולאות

לולאה הוא קטע של תוכנית החוזר על עצמו כל עוד תנאי מתקיים.

5.7.1 לולאת while

בלולאת while בודקים האם תנאי מתקיים ואם כן מבצעים משפט/סדרה של משפטים ושוב חוזרים לבדוק את התנאי. סיום הלולאה הוא כאשר התנאי איננו מתקיים.



איור 5.10 : תרשים זרימה לולאת while .

התחביר :

```
while ( תנאי )
        משפט ;
```

אם יש מספר משפטים שיש לבצע אם התנאי מתקיים יש לרשום אותם בין סוגריים מסולסלים.

התחביר :

```
while ( תנאי )
        {
                1 משפט ;
                 2 משפט ;
                .......
        }
```

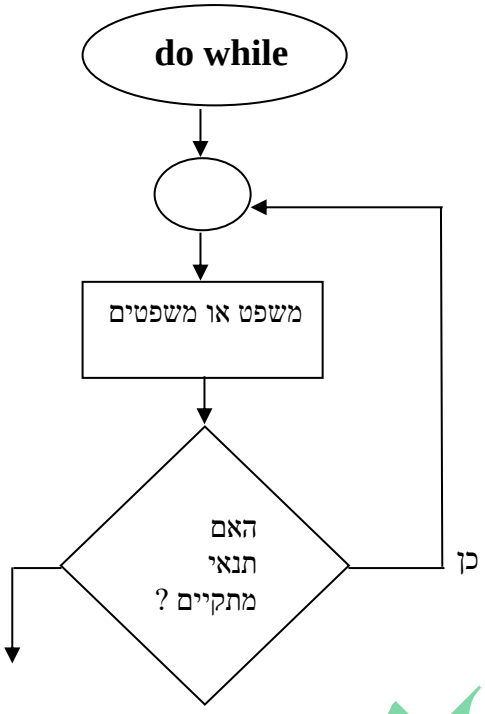
דוגמה:

```
int a=5, factorial=1,x=1;
while(x<=a)
{
        factorial = factorial*x;
        x++;
}
```

התוכנית מחשבת את העצרת של המספר 5

5.7.2 לולאת do while

בלולאת do-while מבצעים סדרת פעולות ואז בודקים אם תנאי מתקיים . אם כן חוזרים על סדרת הפעולות. אם התנאי לא מתקיים – הלולאה מסתיימת.



איור 5.11 : תרשים זרימה של לולאת do while .

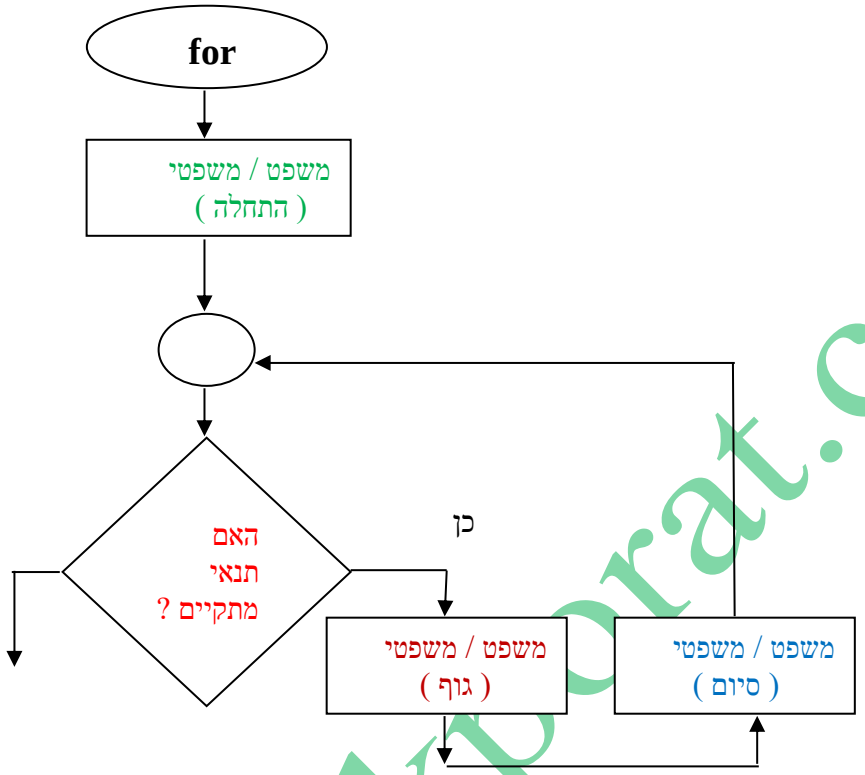
התחביר:

```
do
    משפט;
while(תנאי) ;
```

כאשר רוצים לבצע מספר משפטים התחביר יהיה:

```
do
{
1משפט;
2משפט;
….
}
while(תנאי);
```

5.7.3 לולאת for



איור 5.12 : תרשים זרימה של לולאת for .

התחביר (יש לשים לב לנקודה פסיק בין כל חלק בסוגריים).

for (משפט / משפטי התחלה ; תנאי ; משפט / משפטי סיום)

משפט גוף ;

אם יש כמה משפטי גוף :

for (משפט כניסה / משפטי ; תנאי ; משפט / משפטי סיום)

{

משפט גוף 1 ;

משפט גוף 2 ;

.....

}

אם יש משפטי כניסה הם מופרדים ביניהם בעזרת פסיק. אותו הדבר לגבי משפטי סיום . ניתן לא לרשום את אחד או יותר מהמשפטים (כניסה, גוף , סיום) וגם לא את התנאי אבל יש להמשיך ולסמן את הנקודה פסיק (;) .

דוגמא : ללא משפט כניסה ועם 2 משפטי סיום.

```
for( ; i<10;i++,j--)
{
  ⋮
}
```

5.7.4 תרגול

5.7.4.1 : מה עושה התוכנית הבאה ? (הנח ש ל 8 הדק PORT1 מחוברות 8 לדים המופעלות על ידי '1').

```
#include <REG51F380.h>
void delay(long del)   // פונקציית השהייה
{
        while(del--);
}

void main()
{
        P1=0x01;
        while(1)
        {
                delay(500000);
          P1=P1<<1;    // הזז את הנתון בפורט 1 פעם אחת שמאלה
                if(P1==0)
                        P1=0x1;
        }
}
```

תשובה : התוכנית מזיזה את הנתון שבפורט 1 פעם אחת שמאלה עם השהייה בין הזזה להזזה. ההשהיה נקבעת על ידי הערך שאנחנו שולחים לפונקציה delay . בדוגמה בתרגיל שולחים את הערך 500000 שגורם להשהייה של פחות משנייה אחת. אם היינו מחברים ל 8 הדקי PORT1 8 לדים אז היינו רואים שבתחילת התוכנית נדלקת הלד המתחברת להדק P1.0 ולאחר קצת פחות משנייה נדלקת הלד שמחוברת ל P1.1 ולאחריה הלד ב P1.2 וכך הלאה. למעשה הרצנו אור בלדים.

5.7.4.2 מה היה קורה אם לא היינו רושמים את השורות : P1=0x01; if(P1==0) ?

תשובה : האור היה רץ שמאלה עד שהלד ב P1.7 הייתה נדלקת. בהזזה השמאלית הבאה P1=00000000 ואף לד לא תידלק למרות שהתוכנית ממשיכה לרוץ.

5.8 מערכים Arrays

מערך הוא רצף של משתנים המסודרים אחד אחרי השני בזיכרון , כולם מאותו הטיפוס ולכולם אותו שם. הם נבדלים אחד מהשני במיקום שלהם במערך.

העבודה במערכים מאפשרת עבודה קלה, חסכון בזמן ויצירת ארגון וסדר בתוכנית כאשר אנו משתמשים במספר רב של משתנים שנועדו לאותה המטרה. כל משתנה במערך נקרא איבר

תחביר להגדרת מערך:

< טיפוס איברי המערך > < שם המערך > [כמות האיברים במערך] ;

דוגמאות להגדרה של מערכים:

```
int arr[10];
char string[8];
float f[5];
```

ניתן גם לאתחל את המערך בזמן ההגדרה :

```
int arr[ ] = { 10,-20,30,50};
```

במקרה הזה, למרות שלא הגדרנו את כמות האיברים, הקומפיילר יפתח מערך של 4 איברים.

5.9 מחרוזות STRINGS

מחרוזת היא מערך מטיפוס תווי - char .

הגדרת מחרוזת דומה להגדרת מערך.

הגדרה :

```
char < שם המחרוזת > [ כמות האיברים במחרוזת ] ;
```

דוגמה : הגדרה של מחרוזת בת 10 אברים :

```
char string1[10];
```

כאשר מגדירים מחרוזת הקומפיילר מכניס למחרוזת את התו \0 (NULL) . לכן יש לדאוג לאיבר נוסף עבור תו זה.

דוגמה:

```
char str2[ ] = ="Hello" ;
```

במקרה הזה הקומפיילר ייפתח מערך – מחרוזת – של 6 איברים (האיבר השישי הוא ה NULL).

כאשר הקומפיילר מגדיר מערך או מחרוזת הוא פותח גם מצביע הנקרא בשם של המערך או המחרוזת, ומכניס אליו את הכתובת של האיבר הראשון. מכאן **שם של מערך או מחרוזת הוא הכתובת של האיבר הראשון של המערך או המחרוזת.**

5.10 פונקציה

קטע תוכנית המקבל נתונים, מעבד אותם ומחזיר ערך אחד בלבד.

הגדרת פונקציה:

```
< טיפוס הערך המוחזר מהפונקציה >  < שם הפונקציה > ( פרמטרים שהפונקציה מקבלת)
{
    הגדרת משתנים ;
    משפטי התוכנית ;
   return( ערך מוחזר ) ;
}
```

דוגמא :

```
float div (int a, float b) //הגדרה של פונקציה בשם div שמחזירה ערך מטיפוס float. הפונקציה מקבלת משתנים מטיפוס שלם ומטיפוס ממשי
{
        float c;
        c=a/b;
        return ( c ) ;
}
```

ברשימת הפרמטרים – משתנים - שהפונקציה מקבלת , הנקראים **פרמטרים פורמאליים,** יש לציין לכל משתנה את הטיפוס שלו , אפילו אם הם מאותו הטיפוס ! .

ניתן גם להגדיר בדרך נוספת :

```
< טיפוס הערך המוחזר >  <שם הפונקציה>  (שמות הפרמטרים שהפונקציה מקבלת)
 < טיפוס פרמטר 1 > < שם פרמטר 1> , < טיפוס פרמטר 2 > < שם פרמטר 2> ...;
{
    הגדרת משתנים ;
    משפטי התוכנית ;
   return ( ערך מוחזר ) ;
}
```

דוגמא : ניתן להגדיר פונקציה גם בצורה הבאה:

```
float add (a,b)
int a, float b
{
        float c;
```

```
        c=a/b;
        return ( c ) ;
}
```

הפונקציה מקבלת פרמטרים/ארגומנטים מתוכנית אחרת הקוראת לה. הפרמטרים שנשלחים אל הפונקציה נקראים **פרמטרים אקטואליים** . הפרמטרים המקבלים בפונקציה נקראים **פרמטרים פורמליים**. הכמות והטיפוסים של הפרמטרים בשורה הקוראת לפונקציה – פרמטרים אקטואליים חייבים להיות זהים לכמות וטיפוסי הפרמטרים המקבלים בפונקציה – פרמטרים פורמליים.

5.10.1 הצהרה על פונקציה

את הפונקציה רושמים **מעל** התוכנית הקוראת או שמכריזים עליה - **הצהרה** - לפני כן.

הסבר: נתונה התוכנית הבאה שבה קוראים בשורה 4 לפונקציה add . הפונקציה add נמצאת בשורה 6.

```
1       void main( )
2       {
3          int a=10,b=20;
4          add(a,b);
5        }
6       void add (int x, int y)
7        {
8         printf("a+b = %d + %d = %d",x,y,x+y);
9        }
```

הקומפיילר של שפת C מתרגם משורה מספר 1 כלפי מטה. הוא מגיע לשורה 4 ורואה שמזמנים פונקציה שהוא לא מכיר.

הקומפיילר נותן הודעת שגיאה שהוא לא מכיר את הפונקציה add ולא יוצר קובץ ריצה או קובץ HEX.

כדי למנוע שגיאה זו יש **להצהיר** על הפונקציה לפני שקוראים לה. **את ההצהרה שמים לפני שורה 1 של התוכנית.**

הצהרה על פונקציה :

< טיפוס הערך המוחזר > < שם הפונקציה > (רשימת פרמטרים שהפונקציה מקבלת) ;

ובמקרה שלנו : void add (int x, int y) ; . יש לשים לב לנקודה – פסיק בהצהרה.

הצהרה זו "עושה הכרה" לקומפיילר ואומרת לו שגם אם הוא "נפגש" בשורה הקוראת לפונקציה add שהוא כרגע איננו מכיר, לא לתת הודעת שגיאה אלא להמשיך לתרגם את התוכנית. אומרים לו הפונקציה נקראת add והיא לא מחזירה ערך והיא מקבלת 2 פרמטרים מטיפוס שלם בשם x ו y .

ניתן לרשום את הפונקציה add לפני הפונקציה main ואז אין צורך בהצהרה כי הקומפיילר עובר על התוכנית מלמעלה כלפי מטה והוא מתרגם את הפונקציה add וכבר מכיר אותה.

ברשימת הפרמטרים ניתן לרשום את טיפוס המשתנים ושמם לפי הסדר שיהיה בפונקציה. ניתן גם להזניח את שם המשתנים ולרשום רק את הטיפוס של כל משתנה . חלק מהקומפיילרים יקבלו את ההצהרה גם ללא רשימת הפרמטרים הפורמאליים.

5.10.2 העברת פרמטרים אל פונקציה

כאשר מעבירים פרמטרים אקטואליים אל פרמטרים פורמאליים של פונקציה ישנם שני סוגי העברה.

א. העברת ערך - Call By Value

העברת **הערך** של פרמטר אקטואלי אל פונקציה. הערך מועבר אל פרמטר פורמאלי. שינוי הערך של הפרמטר (משתנה) הפורמאלי בפונקציה לא ישנה את הערך המקורי של הפרמטר האקטואלי בשורה הקוראת.

ב. העברת כתובת – Call By Reference

העברת כתובת של משתנה. שינוי תוכן הכתובת ישנה את המשתנה בפונקציה הקוראת ! . כאשר מעבירים מערך או מחרוזת מעבירים את הכתובת של המערך או המחרוזת. שינוי של המערך או המחרוזת בפונקציה, משנה את ערכי המערך או המחרוזת המקוריים.

5.10.2.1 קבלת פרמטר פורמלי שהוא מערך :

פונקציה המקבלת מערך מוגדרת כך:

```
< טיפוס הערך המוחזר >  < שם הפונקציה > ( <טיפוס המערך> < שם מערך > [ ] )
{
  פקודות הפונקציה;
        ⋮
  return ( ערך מוחזר ) ;
}
```

אין צורך לרשום את כמות האיברים של המערך !

5.11 אפיון משתנים (סוגי משתנים)

משתנים מאופיינים ע"י 2 מושגים : **טווח הכרה (scope) ואורך חיים (life time)**.

טווח הכרה - הפונקציות שבהם יוכר משתנה ומאילו פונקציות ניתן להתייחס אליו.

אורך חיים - באיזו נקודה בתוכנית נוצר המשתנה ועד מתי ניתן להתייחס אל הערך שבו.

5.11.1 משתנה אוטומטי (מקומי, פנימי, לוקלי)

המשתנים המוגדרים בתוך פונקציה, כולל הפרמטרים הפורמאליים שהפונקציה מקבלת.

תחביר :

< טיפוס המשתנה > < שם המשתנה > ;

טווח הכרה – בפונקציה שבה הוגדרו. **אורך חיים** - כל זמן שהפונקציה מתבצעת.

5.11.2 משתנה גלובלי (כללי)

המשתנים המוגדרים מעל הפונקציה הראשונה בתוכנית או בין פונקציות.

תחביר

< טיפוס המשתנה > < שם המשתנה > ;

טווח הכרה – בכל הפונקציות מנקודת ההגדרה ועד סוף התוכנית. **אורך חיים** – כל זמן ריצת התוכנית.

5.11.3 הכרזה על משתנה חיצוני

כאשר רוצים להשתמש בקובץ מקור כלשהו במשתנה שהוגדר בקובץ מקור אחר (המשתנה בקובץ האחר הוגדר כגלובלי !) יש לבצע **הכרזה**. ההכרזה אומרת שהמשתנה הוגדר בקובץ מקור אחר.

תחביר :

extern < טיפוס המשתנה > <שם המשתנה> ;

5.11.4 משתנה סטטי

משתנה אוטומטי עם טווח הכרה בפונקציה שהוגדר אבל עם אורך חיים לכל התוכנית.

תחביר :

static < טיפוס המשתנה > < שם המשתנה > ;

טווח הכרה – בפונקציה שבה הוגדר המשתנה. **אורך החיים** - כל זמן ריצת התוכנית.

ניתן גם להגדיר משתנה סטטי גלובלי. משתנה כזה לא יוכר על ידי קבצי מקור אחרים.

5.12 מצביעים - POINTERS

מהדר Cx51 של KEIL תומך בהצהרה של משתנה מצביע באמצעות התו * (כוכבית). המצביעים משמשים לביצוע כל הפעולות האפשריות ב- C רגיל. עם זאת, בשל הארכיטקטורה הייחודית של ה- 8051 המהדר תומך בשני סוגים של מצביעים:

א. מצביע כללי - generic pointer

ב. מצביע ספציפי לזיכרון - Memory specific pointer

הגדרת מצביע ספציפי לזיכרון:

< טיפוס המצביע> < הזיכרון עליו מצביע > * < שם המצביע> ;

דוגמאות:

char idata *ip; **(המצביע בגודל של ביית)**

char xdata *xp; **(המצביע בגודל 2 בתים)**

הגדרת מצביע כללי :

< שם המצביע> * <טיפוס המצביע > ;

דוגמה : char *p ;

מצביע כללי הוא בן 3 בתים. א. ביית ה MSB מציין את סוג הזיכרון. ב. החלק הגבוה של הכתובת ג. החלק הנמוך של הכתובת.

מגודל המצביע ניתן להבין שהשימוש במצביע ספציפי יעיל יותר, תופס פחות שורות קוד ומאיץ את ריצת התוכנית.

5.13 : פסיקות

מתייחסים אל פסיקה כמו לפונקציה בתוספת שימוש במילים השמורות interrupt ו using . פסיקה בדרך כלל איננה מקבלת ערכים ואיננה מחזירה ערך כי איננו יודעים באיזה שלב בתוכנית היא תופיע (הרחבה על נושא הפסיקות – בפרק הבא).

הגדרה :

```
void  < שם הפסיקה > (void )   interrupt <מספר הפסיקה >  using  < מספר הבנק >
{
 משפטי הפונקציה ;
}
```

שם הפסיקה יכול להיות כל שם של פונקציה. רצוי לתת שם בעל משמעות להבנה טובה יותר .

מספר הפסיקה הוא מספר בין 0 ל 4 (או 5 ברכיבים המסתיימים ב 2) לפי הטבלה שבהמשך.

מספר הבנק הוא מספר בין 0 ל 3 האומר עם איזה בנק ממליצים לקומפיילר להשתמש בפסיקה. לא חובה לרשום זאת !

דוגמא:

```
void timer0 (void) interrupt 1  using 2
 {

   ……..;
  }
```

הפונקציה נקראת timer0 (או כל שם אחר), היא איננה מחזירה ערך וגם לא מקבלת פרמטרים. המילה interrupt 1 אומרת שזוהי פסיקה מספר 1 (פסיקה 0 היא הפסיקה החיצונית int0) והמילה using 2 אומרת להשתמש בבנק 2 במהלך ביצוע הפסיקה.

טבלת וקטור הפסיקות - מספרי הפסיקה וכתובתם בזיכרון התוכנית נראים בטבלה הבאה:

Interrupt #	Description	Vector Address
0	External 0	0x0003
1	Timer 0	0x000b
2	External 1	0x0013
3	Timer 1	0x001b
4	Serial	0x0023
5	Timer 2 (8052)	0x002b
...		...
n		0x0003 + 8*n

טבלה 5.6 : טבלת וקטור הפסיקות

בטבלה רואים את מספר הפסיקה בעמודה השמאלית. העמודה המרכזית מתארת את הפסיקה. העמודה הימנית מראה את המיקום של הפסיקה בזיכרון התוכנית. הטבלה נכונה עבור הרכיב 51 המקורי. עבור C8051F380 יש טבלה אחרת שנראה בהמשך.

5.14 כתיבת קוד אסמבלי בתוך שפת C

כדי לכתוב בתוכנית בשפת C פקודות של אסמבלי נרשום את המילה _asm ואח"כ את כל שורות הקוד באסמבלי ובסוף יש לרשום _endasm יש לרשום את הנתונים ההקסה דצימליים כמו בשפת C ולא כמו באסמבלי רגיל !

```
_asm
mov a,#0x55
mov b,a
mov dptr,#0x1000
movx @dptr,a
ret
_endasm;.          //   לשים לב לנקודה פסיק
```

כאשר רוצים שקטעי קוד בשפת אסמבלי הנמצאים בתוכנית בשפת C ישתמשו במשתנים שהוגדרו בשפת C יש לרשום את שם המשתנה עם קו תחתון לפניו (under score) .

דוגמא :

```
unsigned char var1;
void main()
 {
        _asm
        mov a,_var1
        _endasm;
```

5.15 ההנחיה #include

כדי לעבוד עם רכיב ממשפחת ה 51 יש לרשום בתחילת התוכנית את ההנחיה **#include < שם קובץ כותר של הרכיב.h >** כאשר הסיומת h היא קיצור של header (כותר או כותרת). ההנחיה דומה ל #include <stdio.h> שרושמים בשפת C כאשר רוצים לעבוד עם הפונקציות printf או scanf וכו'.

כאשר נרצה לעבוד עם הרכיב C8051F380 נרשום : **#include <REG51F380.h>** . ההנחיה אומרת לקומפיילר לכלול את קובץ הכותר REG8051F380 . בקובץ הזה יש הצהרות המקשרות את כל הזיכרונות, הרגיסטרים והרכיבים הפריפריאליים של המיקרו בקר C8051F380 לשימוש עם שפת C51 . מרגע ההנחיה נוכל לרשום P1=0x55; והקומפיילר יודע להעביר את הנתון 55 בהקסה דצימלי לפורט 3 . בקובץ הכותר מוצהר כי כתובת פורט 1 היא 90H . כאשר הפקודה תתורגם לאסמבלי היא תהיה : mov 90h,#55h .

קובץ הכותר משייך שם לכתובת של אותו רכיב (רגיסטר או זיכרון וכו').

קבצי הכותר של הרכיבים עבור כל הנגזרות של רכיבי ה 51 של המשפחות השונות נמצא בספריה : C:\Keil_v5\C51\INC . בספריה זו יש יצרנים שונים של רכיבים ממשפחת ה 51 . הרכיב C8051F380 נמצא בספריה SiLABS וכאשר ניכנס לספריה נראה מספר רב של קבצי כותר עבור הרכיבים השונים של היצרן.

באיור שבהמשך נראה 2 דפים של קובץ הכותר REG51F380 מתוך מספר דפים. השמאלי הוא הצהרות על רגיסטרים והימני על ביטים. ניתן לראות שההצהרות על ביטים משתמשות בתו ^ שמציין את הביט של הרגיסטר . לדוגמה הרגיסטר TCON נמצא בכתובת 88H . הביט TCON^0 נקרא IT0 . ניתן לראות זאת בעזרת החץ הירוק. בחלק השמאלי אמרנו לקומפיילר שכתובת TCON היא 88H ואז בהמשך כאשר נצהיר על אחד הביטים שלו הקומפיילר יודע באיזו כתובת נמצא כל ביט. ליתר דיוק ה SFR בנוי בצורה כזו שרק רגיסטרים הנמצאים בכתובות שמחולקות ב 8 ללא שארית הם רגיסטרים שניתן לפנות לכל ביט שלהם. שאר הרגיסטרים הם ללא חלוקה לביטים ולא ניתן לגשת ישירות לביט מסוים שלהם אלא רק לכל הרגיסטר. צורה זו מאפשרת להבין שאם פורט 1 נמצא בכתובת 90H אז ביט 0 שלו - P1^0 - נמצא בכתובת 90H במיעון ביט , ביט 1 שלו – P1^1 נמצא בכתובת 91H וכך הלאה עד P1^7 הנמצא בכתובת 97H במיעון ביט. הפקודה setb 97h (או בשפת C51 : P1^7=1) תשים '1' בביט ה MSB של פורט 1 . הפקודה clr 92h (או בשפת C51 : P1^2=0) תשים '0' בביט השלישי של פורט 1 (הביטים מתחילים מ 0).

```
REG51F380.H
1   /*-----------------------------------------------------------------------------
2   REG51F380.H
3
4   Header file for Silabs C8051F380/1/2/3/4/5/6 and C8051F387 devices.
5   Copyright (c) 2011 ARM Ltd and ARM Germnay GmbH.
6   All rights reserved.
7   -----------------------------------------------------------------------------*/
8
9
10  #ifndef __REG51F380_H__
11  #define __REG51F380_H__
12
13  /* SFRPage 0x00 and SFRPage 0x0F Registers */
14  sfr P0          = 0x80;         /* Port 0 Latch                          */
15  sfr SP          = 0x81;         /* Stack Pointer                         */
16  sfr DPL         = 0x82;         /* Data Pointer Low                      */
17  sfr DPH         = 0x83;         /* Data Pointer High                     */
18  sfr EMI0TC      = 0x84;         /* External Memory Interface Timing      */
19  sfr EMI0CF      = 0x85;         /* External Memory Interface Configuration */
20  sfr OSCLCN      = 0x86;         /* Internal Low-Frequency Oscillator Control*/
21  sfr PCON        = 0x87;         /* Power Control                         */
22  sfr TCON        = 0x88;         /* Timer/Counter Control                 */
23  sfr TMOD        = 0x89;         /* Timer/Counter Mode                    */
24  sfr TL0         = 0x8A;         /* Timer/Counter 0 Low                   */
25  sfr TL1         = 0x8B;         /* Timer/Counter 1 Low                   */
26  sfr TH0         = 0x8C;         /* Timer/Counter 0 High                  */
27  sfr TH1         = 0x8D;         /* Timer/Counter 1 High                  */
28  sfr CKCON       = 0x8E;         /* Clock Control                         */
29  sfr PSCTL       = 0x8F;         /* Program S
30  sfr P1          = 0x90;         /* Port 1 La
31  sfr TMR3CN      = 0x91;         /* Timer/Cou
32  sfr TMR3RLL     = 0x92;         /* Timer/Counter 3 Reload Low    (page 0) */
33  sfr TMR3RLH     = 0x93;         /* Timer/Counter 3 Reload High   (page 0) */
34  sfr TMR3L       = 0x94;         /* Timer/Counter 3 Low                   */
35  sfr TMR3H       = 0x95;         /* Timer/Counter 3 High                  */
36  sfr USB0ADR     = 0x96;         /* USB0 Indirect Address Register        */
37  sfr USB0DAT     = 0x97;         /* USB0 Data Register                    */
38  sfr TMR4CN      = 0x91;         /* Timer/Counter 4 Control       (page F) */
39  sfr TMR4RLL     = 0x92;         /* Timer/Counter 4 Reload Low    (page F) */
40  sfr TMR4RLH     = 0x93;         /* Timer/Counter 4 Reload High   (page F) */
41  sfr SCON0       = 0x98;         /* UART0 Control                         */
42  sfr SBUF0       = 0x99;         /* UART0 Data Buffer                     */
43  sfr CPT1CN      = 0x9A;         /* Comparator 1 Control                  */
44  sfr CPT0CN      = 0x9B;         /* Comparator 0 Control                  */
45  sfr CPT1MD      = 0x9C;         /* Comparator 1 Mode                     */
46  sfr CPT0MD      = 0x9D;         /* Comparator 0 Mode                     */
47  sfr CPT1MX      = 0x9E;         /* Comparator 1 Mux                      */
48  sfr CPT0MX      = 0x9F;         /* Comparator 0 Mux                      */
49  sfr P2          = 0xA0;         /* Port 2 Latch                          */
50  sfr SPI0CFG     = 0xA1;         /* SPI0 Configuration                    */
51  sfr SPI0CKR     = 0xA2;         /* SPI0 Clock rate control               */
52  sfr SPI0DAT     = 0xA3;         /* SPI0 Data Buffer                      */
53  sfr P0MDOUT     = 0xA4;         /* Port 0 Output Mode                    */
54  sfr P1MDOUT     = 0xA5;         /* Port 1 Output Mode                    */
55  sfr P2MDOUT     = 0xA6;         /* Port 2 Output Mode                    */
56  sfr P3MDOUT     = 0xA7;         /* Port 3 Output Mode                    */
100 %   No issues found
```

כתובות הפורטים ב SFR

```
REG51F380.H
176
177 /* Bit Definitions for SFRPage 0x00 and SFRPage 0x0F Registers */
178 /* TCON 0x88 */
179 sbit TF1     = TCON^7;          /* Timer 1 Overflow Flag                 */
180 sbit TR1     = TCON^6;          /* Timer 1 On/Off Control                */
181 sbit TF0     = TCON^5;          /* Timer 0 Overflow Flag                 */
182 sbit TR0     = TCON^4;          /* Timer 0 On/Off Control                */
183 sbit IE1     = TCON^3;          /* Ext. Interrupt 1 Edge Flag            */
184 sbit IT1     = TCON^2;          /* Ext. Interrupt 1 Type                 */
185 sbit IE0     = TCON^1;          /* Ext. Interrupt 0 Edge Flag            */
186 sbit IT0     = TCON^0;          /* Ext. Interrupt 0 Type                 */
187
188 /* SCON0 0x98 */
189 sbit S0MODE  = SCON0^7;         /* Serial Port 0 Operation Mode          */
190                                 /* Bit 6 unused                          */
191 sbit MCE0    = SCON0^5;         /* Multiprocessor Communication Enable   */
192 sbit REN0    = SCON0^4;         /* UART0 RX Enable                       */
193 sbit TB80    = SCON0^3;         /* Ninth Transmission Bit                */
194 sbit RB80    = SCON0^2;         /* Ninth Receive Bit                     */
195 sbit TI0     = SCON0^1;         /* UART0 TX Interrupt Flag               */
196 sbit RI0     = SCON0^0;         /* UART0 RX Interrupt Flag               */
197
198 /* IE 0xA8 */
199 sbit EA      = IE^7;            /* Global Interrupt Enable               */
200 sbit ESPI0   = IE^6;            /* SPI0 Interrupt Enable                 */
201 sbit ET2     = IE^5;            /* Timer 2 Interrupt Enable              */
202 sbit ES0     = IE^4;            /* UART0 Interrupt Enable                */
203 sbit ET1     = IE^3;            /* Timer 1 Interrupt Enable              */
204 sbit EX1     = IE^2;            /* External Interrupt 1 Enable           */
205 sbit ET0     = IE^1;            /* Timer 0 Interrupt Enable              */
206 sbit EX0     = IE^0;            /* External Interrupt 0 Enable           */
207
208 /* IP 0xB8 */
209                                 /* Bit 7 unused                          */
210 sbit PSPI0   = IP^6;            /* SPI0 Interrupt Priority               */
211 sbit PT2     = IP^5;            /* Timer 2 Priority                      */
212 sbit PS0     = IP^4;            /* UART0 Priority                        */
213 sbit PT1     = IP^3;            /* Timer 1 Priority                      */
214 sbit PX1     = IP^2;            /* External Interrupt 1 Priority         */
215 sbit PT0     = IP^1;            /* Timer 0 Priority                      */
216 sbit PX0     = IP^0;            /* External Interrupt 0 Priority         */
217
218 /* SMB0CN 0xC0 SFR Page=0 */
219 sbit MASTER0  = SMB0CN^7;       /* SMBus0 Master/Slave Indicator         */
220 sbit TXMODE0  = SMB0CN^6;       /* SMBus0 Transmit Mode Indicator        */
221 sbit STA0     = SMB0CN^5;       /* SMBus0 Start Flag                     */
222 sbit STO0     = SMB0CN^4;       /* SMBus0 Stop Flag                      */
223 sbit ACKRQ0   = SMB0CN^3;       /* SMBus0 Acknowledge Request            */
224 sbit ARBLOST0 = SMB0CN^2;       /* SMBus0 Arbitration Lost Indicator     */
225 sbit ACK0     = SMB0CN^1;       /* SMBus0 Acknowledge                    */
226 sbit SI0      = SMB0CN^0;       /* SMBus0 Interrupt Flag                 */
227
228 /* SMB1CN 0xC0 SFR Page=F */
229 sbit MASTER1  = SMB1CN^7;       /* SMBus1 Master/Slave Indicator         */
230 sbit TXMODE1  = SMB1CN^6;       /* SMBus1 Transmit Mode Indicator        */
231 sbit STA1     = SMB1CN^5;       /* SMBus1 Start Flag                     */
100 %   No issues found
```

איור 5.13 : 2 דפים מקובץ REG51F380. השמאלי הגדרות של רגיסטרים ב SFR ובימני הגדרות של ביטים.

5.15.1 – שיוך שם לכתובת .

בהסבר על הגדרות המשתנים בתחילת הפרק הראנו שפעמים רבות רוצים לשייך שם לכתובת באזור ה SFR . דוגמה למקרה זה יכולה להיות כאשר דיודת לד מחוברת לאחד מהדקי המיקרו בקר. נניח שהיא מחוברת להדק P1.0 . במקרה זה נוכל לרשום :

```
sbit led=0x90;
```

במשפט הזה אומרים שנשייך את המילה led לביט באזור ה sfr שנמצא בכתובת 90H (זוהי הכתובת של הביט P1.0). אחר כך נרשום בתוכנית את המילה led ללא צורך לזכור לאיזה הדק היא מחוברת. נוכל לרשום : led=1; או led=0; להדליק או לכבות את הלד.

ניתן לא להשתמש בהנחיה #include ולרשום את ההצהרות בצורה הבאה :

```
sfr P1=0x90;
```

במשפט הזה אומרים לקומפיילר שהשם P1 שייך לאזור ה SFR לכתובת 90H . ניתן גם לומר זאת בדרך נוספת : השם P1 משויך לכתובת 90H באזור ה SFR . אם נרשום בשפת C51 את הפקודה P1=0x55; אז התרגום לאסמבלי יהיה mov 90h,#55h .
אם נרשום P1^5=1; אז התרגום לאסמבלי יהיה setb 95h .

ניתן גם לרשום את המשפט הבא :

```
sbit P1_2 = 0x92;
```

כאן אומרים שנשייך את המילה P1_2 לביט שנמצא ב sfr בכתובת 92H (ביט P1.2).

5.16 – קובץ Init_Device ותוכנת Configuration Wizard .

כאשר כותבים תוכנית במערכות עם מיקרו בקרים נוהגים בתחילת התוכנית לאתחל הדקים (לקבוע מי קלט ומי פלט) , לאתחל קצב של תקשורת ל baud הרצוי ולאתחל מודולים של חומרה. האתחולים מתבצעים פעם אחת בלבד. בדרך כלל נרשום פונקציית אחת שבה יהיו כל האתחולים הרצויים. הדבר דומה לפונקציית ה setup( ) בארדואינו שלאחריה עובדת פונקציית ה loop( ).

לחברת SILABS תוכנה הנקראת Configuration Wizard - ובעברית - אשף התצורה . זוהי תוכנה גרפית באמצעותה ניתן לקבוע את התצורה של המעגל שלנו. התוכנה חוסכת למשתמש לזכור את המבנה הפנימי של כל הרגיסטרים של המיקרו בקר וכיצד לתכנת אותם. אפשרות זו נוחה עושה למשתמש "חיים קלים יותר". כמובן שעל המשתמש לשלוט בתוכנה ולדעת מה האפשרויות הקימות !! . התוכנה היא גרפית שבה יש לסמן במקומות המתאימים את הדקי הפורטים (קלט או פלט) , לאילו הדקים יתחברו הרכיבים הפריפריאליים וחיבור הקרוסבר, עם איזה תדר לעבוד ,האם להפעיל watch dog , וכו' . בסיום , התוכנה יוצרת קוד עבור הרגיסטרים המתאימים. הקוד נשמר בקובץ בשם **void Init_Device(void);** .

בכל תוכנית שרצה בשפת C או C51 קיימת פונקציה ראשית הנקראת main( ) . בתחילת המשפטים של הפונקציה נזמן את הפונקציה Init_Device( ); לאתחול הרכיבים בצורה הבאה:

```
void main( )
{
Init_Device( );
משפטי הפונקציה ;
  ¦
  ¦
  ¦
}
```

5.16.1 המלצות עבור תוכניות שנרשום בהמשך.

להלן מספר המלצות של מר אבי חיון לעבודה עם המיקרו בקר C8051F380 עם תוכנת KEIL במיקרו ויז'ן .

כדי לעבוד בצורה אחידה עם שמות זהים של הרגיסטרים וטיפוסים של משתנים, יש להצהיר על ספריות הבאות:

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

אחת הספריות כוללת קיצור להצהרת משתנים, בדוגמאות נשתמש בקיצור:

unsigned char - U8

unsigned int - U16

בפרקים הבאים של הספר נצטרך להשתמש בפונקציות המבצעות השהייה . לשם כך נשתמש בפונקציות הנקראות :

void delay_ms(int ms)

הפונקציה מבצעת השהייה באלפיות שנייה. הפקודה ;delay_ms(1000) מבצעת השהייה של 1000 מילי שניות , כלומר 1 שנייה.

פונקציה נוספת שנשתמש בהמשך :

void delay_us(int us)

הפונקציה מבצעת השהייה במיליוניות שנייה. הפקודה ;delay_us(500) מבצעת השהייה של 500 מיקרו שניות , כלומר 0.5 אלפיות שנייה.

הערה : פונקציות ההשהיה אינן קיימות באמת בספריה של המיקרו בקר. אם נרשום אותן בתוכניות עם µVision שנרשום במחשב שלנו נקבל הודעת שגיאה !! הן עוזרות לנו בדוגמאות שניתן בהמשך הספר.

הערות לתרגילים המשך הספר :

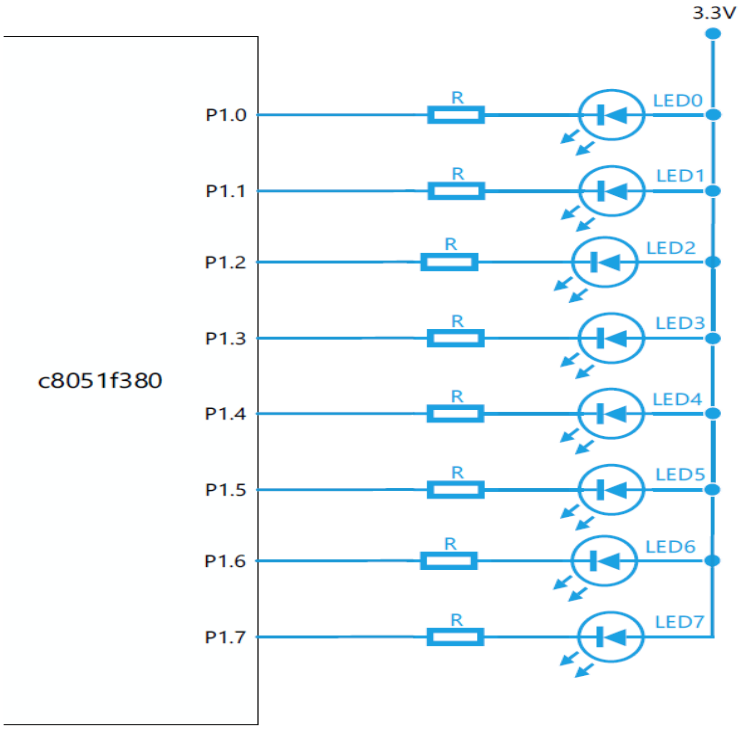
1. בחלק גדול של התרגילים נניח שהקובץ ;Init_Device( ) הוגדר כראוי וכלול בתוכנית.
2. בחלק מהתרגילים נשתמש ב 2 פונקציות ההשהיה שרשמנו בדף זה כדי לבצע השהיות רצויות.

5.17 דוגמאות

5.17.1

דוגמה 1 : האיור בעמוד הבא מתאר חיבור של מיקרו בקר C8051F380 ל 8 נורות לד בעזרת פורט 1 .

ניתן להניח שהוגדרו ההדקים של פורט 1 כפלט ב push pull והקרוסבר חובר. הנגד R הוא של 150Ω . מתח היציאה במצב '0' מהדק הפורט הוא כ - 0.3 וולט (Vo_L - output low level) בזרם יציאה בין 5 מילי אמפר ל 10 מילי אמפר . V_{LED} =1.7v .



איור 5.14 : חיבור 8 לדים אל פורט 1 .

א. חשב את הזרם דרך הלד כאשר היא בהולכה.

פתרון א : משוואת המתח בעזרת חוק קירכהוף היא :

$3.3 = V_{LED} + V_R + V_{OL}$

$3.3 = 1.7 + I * 150 + 0.3$

$I = (3.3 - 1.7 - 0.3) / 150 = 8.666$ mA .

ב. מה עושה התוכנית הבאה ?

```
1.  #include <REG8051F380.h>  // הכללה של קובץ הכותר. יש בו הצהרות על כל הרגיסטרים והביטים והכתובות
    שלהם
2.  void main ( )   // התוכנית הראשית
3.  {
4.  P1 = 0;     //  הדלקת כל 8 הלדים
5.  while(1)        //   לולאה אין סופית
6.  {
7.      delay_ms(500);  //  חצי שנייה -  השהיה של 500 מילי שניות
8.      P1= ~ P1;          //  הפיכת מצב הביטים של פורט . אם הלדים דלקו הן יעברו לחושך ואם היו בחושך הן ידלקו
9.    }
10. }
```

פתרון ב : התוכנית מהבהבת את 8 הלדים בלולאה אין סופית בקצב של חצי שנייה ON ו חצי שנייה OFF .

ג. מה היה קורה אם בשורה 4 במקום P1=0 היינו רושמים P1=0x0f; ?

פתרון ג : הלדים שמספרן 0 עד 3 היו דולקות והלדים שמספרן 4 עד 7 כבויות למשך 1 שנייה. אחר כך היה מתהפך המצב והלדים מ 0 עד 3 היו כבויות והלדים 4 עד 7 דלוקות למשך 1 שנייה והתוכנית מתבצעת בלולאה אין סופית.

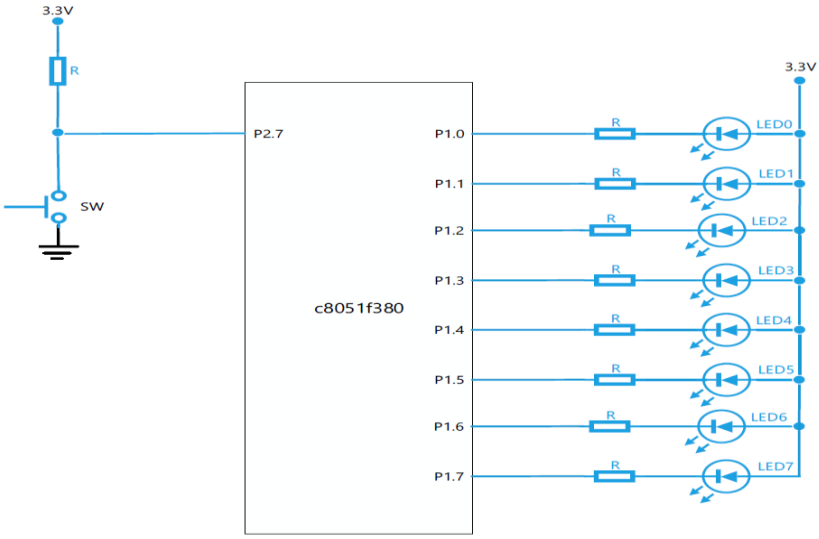
ד. מה היה קורה אם בשורה 4 היינו רושמים P1=0xaa; ? **פתרון ד** : (נסו לפתור לבד...).

5.17.2 המשך תרגילי דוגמה:

התרגילים הבאים הם תרגילי דוגמה שכתב ופתר מר אבי חיון. הוספתי הערות הסבר ליד הפתרון. כדאי לשים לב לדרך הפתרון ולשימוש בפונקציות Init_Device( ); ופונקציות ההשהיה אפילו שהן לא נתונות בפתרונות. המטרה היא לתת כיוון לפתרון התרגילים שיופיעו בבחינה החיצונית.

שאלת דוגמה 1 :

נתונה המערכת הבאה הכוללת מיקרו-בקר C8051F380, שמונה נורות LED ולחצן.



כתבו תוכנית המשמשת להדלקת נורת LED מתוך שמונה נורות לפי התנאים הבאים:

א. כאשר הלחצן לא לחוץ, תהיה הרצת נורות ה- LED כל שנייה מכיוון LED0 עד LED7 במחזוריות (LED0 בלבד דולק, לאחר מכן LED1 בלבד וכך הלאה עד LED7 וחוזר ל- LED0)

ב. כאשר הלחצן לחוץ, תהיה הרצת נורת ה-LED בכיוון ההפוך מנורת LED7 עד LED0 במחזוריות

לצורך יצירת השהיות בין הדלקות נתונה פונקציית השהייה במילי שניות:

```
delay_ms(int ms)
```

להלן מבנה בסיסי של התוכנית הכוללת אתחול תדר השעון וביטל פעולת WDT הוסיפו לקוד זה את פתרון לסעיף א ו- ב.

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

void Init_Device(void);
void delay_ms(int ms);

void main(){
      Init_Device();
}
```

פתרון :

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

void Init_Device(void);
void delay_ms(int ms);

sbit SW=P2^7;

void main(){
      U8 port;
      Init_Device();
      port=0x01;
      P1=~port;

      while(1){
            delay_ms(1000);
            if(SW)
            {
                  if (port==0x80) port=0x01;
                  else port=port<<1;
            }
            else
            {
                  if (port==0x01) port=0x80;
                  else port=port>>1;
            }
            P1=~port;
      }
}
```

הכללה של 2 קבצי כותר המוזכרות בסעיף 5.14.1 . הקבצים האלו נמצאים בספריה שבה נמצא הקובץ הזה של הפתרון. אם הקבצים נמצאים בספריה הסטנדרטית היינו רושמים: #include < name.h >

הצהרה על 2 פונקציות של אתחול הרכיב שנוצרת על ידי אשף התצורה ועל פונקציית השהייה במילי שניות. נניח ששתי הפונקציות קיימות.

שיוך השם sw לביט P2^7 באזור ה SFR

הפונקציה הראשית

הגדרת משתנה מטיפוס לא מסומן של 8 ביט בשם port. כמו unsigned

קריאה – זימון לפונקציה Init_Device שבה יש את כל האתחולים

הפיכת מצב הביטים של המשתנה port והוצאה ל P1 . רק הנורה LED0 נדלקת. port=11111110B

לולאה אין סופית

זימון פונקציה שמבצעת השהייה של 1 שנייה

האם המפסק פתוח ויש 1 בכניסה ? (יש נגד pull up) ? if sw==TRUE

האם מצב port מציין שלד ב P1.7 דולקת ? אם כן port=0x1 להדליק הלד ב P1.0 .

הזז את מצב port פעם אחת שמאלה כדי להדליק את הלד הבאה

המפסק סגור

האם מצב port מציין שלד ב 0P1. דולקת ? אם כן port=0x80 להדליק הלד ב P1.7 .

הפוך את מצב הביטים של המשתנה port (הלדים נדלקים כשיש '0') והעבר לפורט 1.

הזז את מצב port פעם אחת ימינה כדי להדליק את הלד הבאה

תרגיל למחשבה : המפסק בשאלה הקודמת הוא מפסק לחצן שנקרא toggle – מחליף בין מצבים. יש לו 2 מצבים מצב פתוח (כמו שנראה באיור לשאלה הקודמת) או סגור בדומה למפסק המדליק/מכבה מנורה בבית. קיים מפסק **push button** – מפסק לחיצה. זהו מפסק עם קפיץ. למפסק יש מצב שבו הוא נמצא בדרך כלל. כאשר לוחצים על המפסק הוא עובר למצבו השני וכאשר מפסיקים ללחוץ עליו הוא חוזר למצבו הרגיל. אם המפסק בדרך כלל פתוח וכאשר לוחצים עליו הוא נסגר אז הוא נקרא N.O – Normally Open – פתוח בדרך כלל . אם הוא בדרך כלל סגור וכאשר לוחצים עליו הוא נפתח הוא נקרא N.C – Normally Close – בדרך כלל סגור. הפסקת הלחיצה תחזיר אותו למצבו הסגור. דוגמה למפסק push button הוא המפסק בחדר מדרגות. המפסק הוא N.O .

יש לכתוב את אותה התוכנית כמו בתרגיל הקודם רק שהמפסק איננו לחצן רגיל אלא push button מסוג N.O . כל לחיצה על המפסק משנה את כיוון הדלקת הלדים . רמז : הוסף משתנה (מטיפוס ביט או מאיזה טיפוס שתרצו) שמראה שהייתה לחיצה במפסק . כל לחיצה תשנה את מצב המשתנה. בהתאם למצב המשתנה נקבע האם להזיז ימינה או שמאלה.

שאלה דוגמה מס' 2

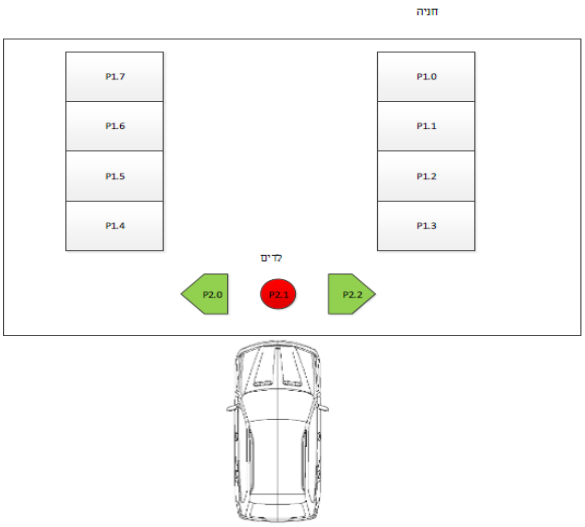
במגרש חניה ממוקמים שמונה חיישנים למיקום רכב. המגרש מחולק לארבע חניות בצידו הימני של המגרש וארבע חניות בצידו השמאלי של המגרש. ארבע החיישנים של הצד הימני מחוברים להדקים P1.0- P1.3 וארבע החיישנים של הצד השמאלי מחוברים להדקים P1.4 – P1.7.

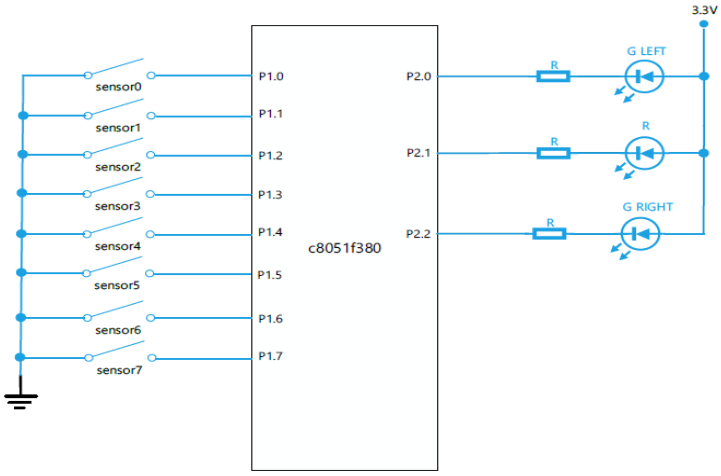
כאשר רכב נמצא בחנייה החיישן סגור. בנוסף במגרש החניה קיימים שלוש נורות LED: שתי נורות LED בצבע ירוק ונורה LED בצבע אדום. נורות ה-LED מחוברים להדקים P2.0-P2.2 המשמשים לחיווי לפי המצבים הבאים:

- נורת LED הימנית בצבע ירוק המחוברת להדק P2.2- תדלק כאשר יש מקום חניה פנוי בצד הימני של המגרש.
- נורת LED השמאלית בצבע ירוק המחוברת להדק P2.0 תדלק כאשר יש מקום חניה פנוי בצד השמאלי של המגרש.
- נורת LED בצבע אדום המחוברת להדק P2.1 מהבהב כל שניה כאשר אין מקום בחניה.

לצורך יצירת השהיות בין הדלקות נתונה פונקציית השהייה במילי שניות:

```
delay_ms(int ms)
```





הנח שהתוכנית כוללת אתחול תדר השעון וביטול פעולת WDT.

א. כתבו תוכנית המפעילה את נורות ה-LED בהתאם למצבים שהוגדרו.

ב. כתוב פונקציה בשם NumberParking() שתספור את מספר המקומות הפנויים ותחזיר ערך זה.

פתרון 2

א.

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

void Init_Device(void);
void delay_ms(int ms);

sbit G_LEFT=P2^0;
sbit R=P2^1;
sbit G_RIGHT=P2^2;

void main(){
      Init_Device();
      // LEDS OFF
      G_LEFT=1;
      R=1;
      G_RIGHT=1;

      while(1){
            if((P1&0x0F) != 0)
                  G_RIGHT=0;
            else
                  G_RIGHT=1;

            if((P1&0xF0) != 0)
                  G_LEFT=0;
            else
                  G_LEFT=1;

            if(P1 == 0){
                  R=!R;
                  delay_ms(1000);
            }
            else
                  R=1;
      }
}
```

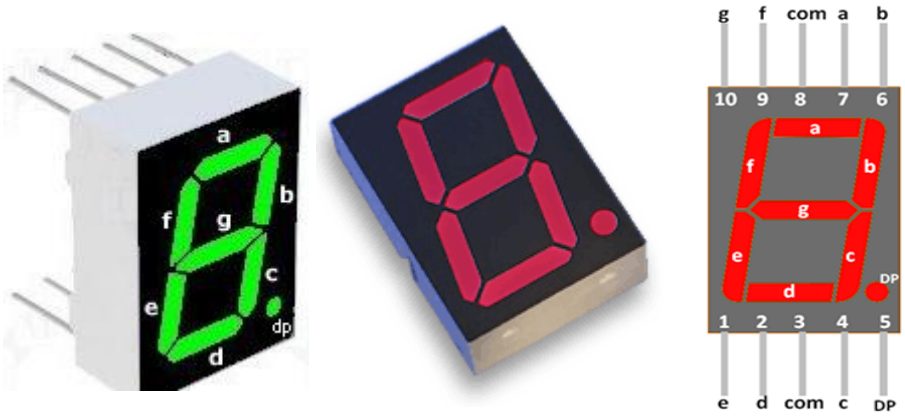
ב.

```
U8 NumberParking(){
      U8 i,cnt=0;
      for(i=0;i<8 ;i++) {
          if(P1 & (1<<i) )
             cnt++;
          }
    return cnt;
}
```

5.18 - חיבור מיקרו בקר אל תצוגת 7 מקטעים

5.18.1 מהי תצוגת 7 מקטעים ובאנגלית 7 Segments ?

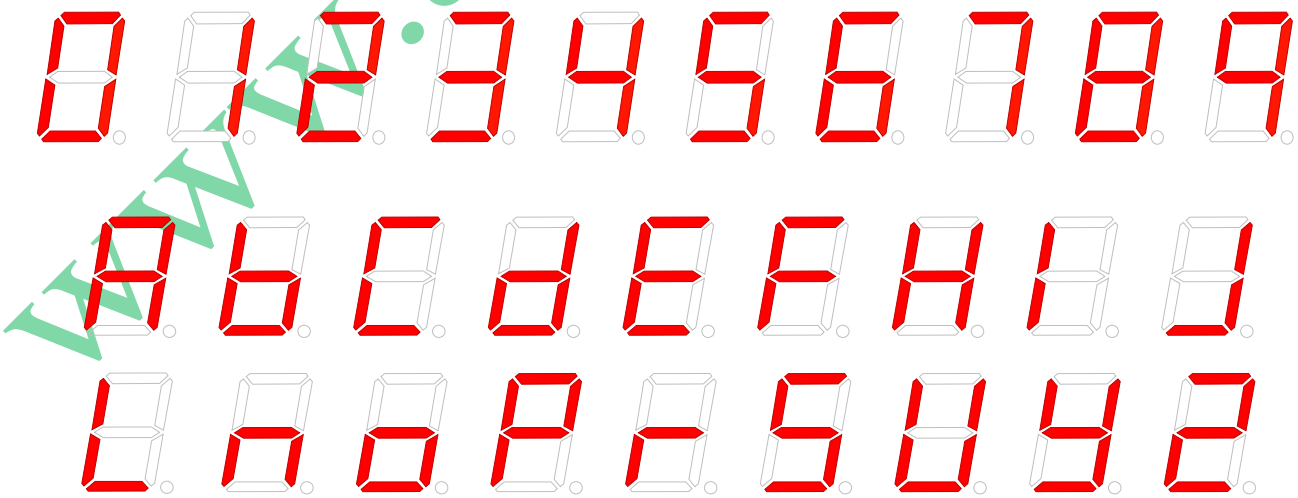
תצוגת 7 מקטעים ובאנגלית 7 segments נפוצה ביותר להצגת ספרות ואותיות . באיור הבא רואים 2 רכיבי תצוגת 7 מקטעים בשני צבעים שונים (כאן רואים אדום וירוק - השניים מצד שמאל. ישנם צבעים נוספים) ואת חיבורי ההדקים שלו (מצד ימין).



איור 5.15 : רכיב תצוגת 7 מקטעים .

כל מקטע – סיגמנט - הוא דיודת לד led. ברכיב יש 7 לדים מ a ועד g ועוד לד המציינת נקודה עשרונית D.P - Decimal Point . כדי להציג את הספרה 8 יש להדליק את כל הלדים (ללא הנקודה העשרונית) . כדי להציג את הספרה 0 יש להדליק את הלדים מ a ועד f (הלדים g ו D.P לא דולקות) . כדי להציג את הספרה 1 יש להדליק את b ו c בלבד וכך הלאה. התצוגה מגיעה בדרך כלל בזיווד (אריזה) של 10 פינים והספירה של הפינים היא כמו בג'וק רגיל באריזת DIP .

ניתן להציג בתצוגה ספרות מ 0 ועד 9 ואפילו את רוב התווים באנגלית כפי שנראה באיור הבא: (ניתן להציג גם חלק מעברית)

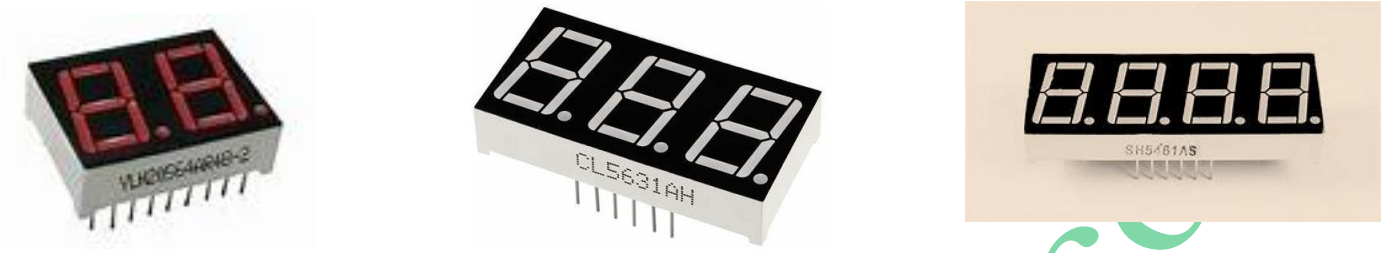


איור 5.16 : הצגת ספרות ואותיות בתצוגת 7 מקטעים.

התצוגות מגיעות בגדלים ובצבעים שונים. ישנן תצוגות בגודל של פחות מ 1 ס"מ וישנן גדולות בגודל של כ 10 ס"מ ואולי יותר.

בתצוגות "גדולות" מקטע אחד יכול להיות מורכב מ 2 או יותר לדים בטור.

קיימות תצוגות 7 מקטעים שבהן יש 2 תצוגות 7 מקטעים או 3 תצוגות ואפילו 8 מקטעים בתוך אריזה אחת כמתואר באיור :



איור 5.17 : 2, 3, ו 4 תצוגות 7 מקטעים באריזה אחת.

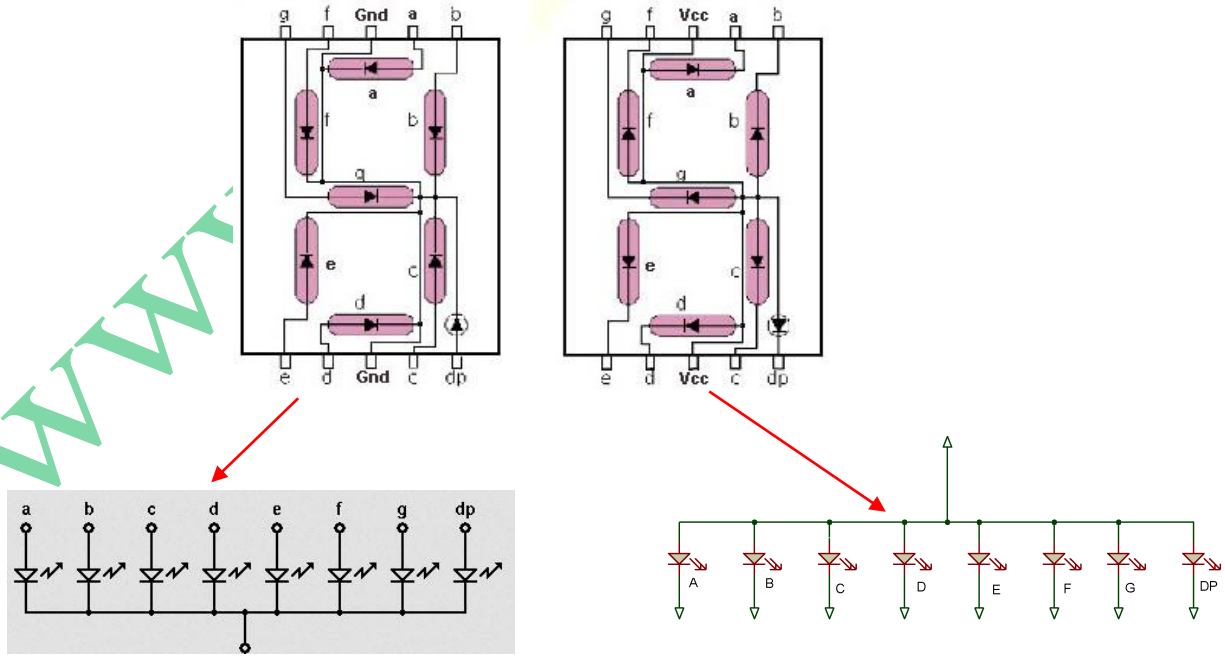
5.18.2 2 אופני החיבור – אנודה משותפת וקטודה משותפת - C.A ו C.C

לתצוגה 2 סוגי חיבור. **א. אנודה משותפת C.A – Common Anode** ב. **קטודה משותפת - C.C – Common Catode.**

המושג "אנודה משותפת" אומר שכל האנודות של המקטעים מקוצרות יחד בתוך הרכיב ומחוברות אל רגל אחת בג'וק ומחברים אליה מתח חיובי (לדוגמה 5 וולט). כדי להדליק את אחד המקטעים יש לתת בקתודה המתאימה 0 .

המושג "קטודה משותפת " " אומר שכל הקתודות של המקטעים מקוצרות יחד בתוך הרכיב ומחוברות אל רגל אחת בג'וק ומחברים אותה בדרך כלל לאדמה. כדי להדליק את אחד המקטעים יש לתת באנודה המתאימה מתח חיובי (5v לדוגמה) .

האיור הבא מתאר את 2 המבנים :

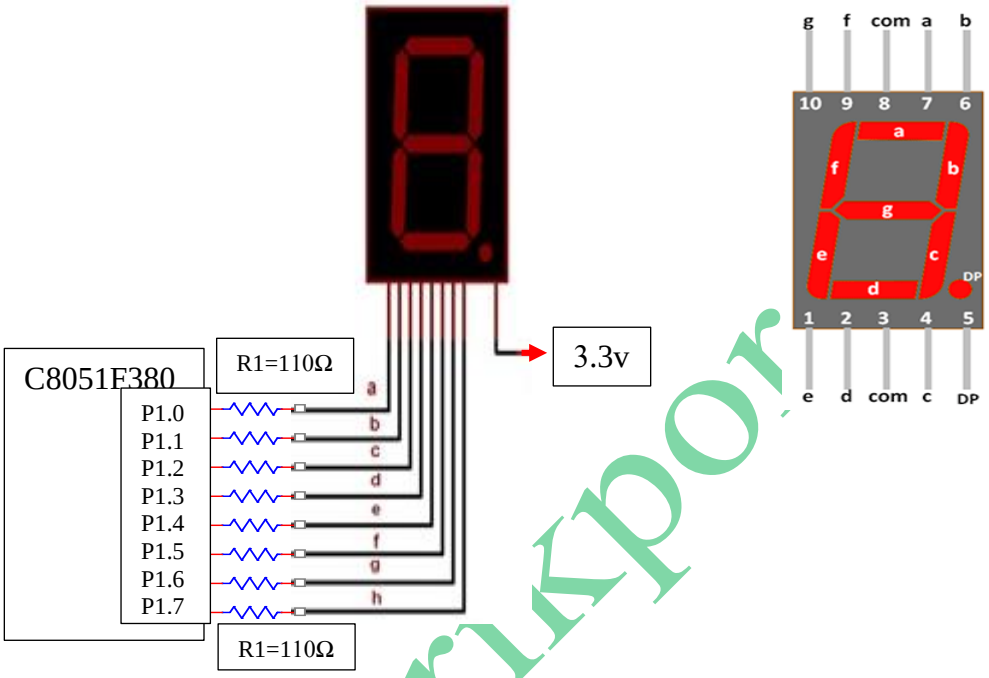


איור 5.16 : 2 סוגי חיבור: אנודה משותפת מימין C.A וקטודה משותפת משמאל C.C . מבנה פנימי ושרטוט השמלי.

5.18.3 ממשק בין המיקרו לתצוגת 7 המקטעים

באיור הבא רואים מממשק בין מיקרו בקר לתצוגת 7 מקטעים . התצוגה מתחברת בחיסור אנודה משותפת Common Anode .

כיצד יודעים שהחיבור הוא אנודה משותפת ? רואים שלתצוגה מחובר מתח של 5 וולט ואילו שאר ההדקים מ a ועד h) h הוא הנקודה העשרונית - DP) מתחברים להדקי המיקרו בקר. כל האנודות של התצוגה מקוצרות בתוך הרכיב וכדי להדליק מקטע מסוים יש לתת '0' באחת מהרגליים של פורט 1 המתחברות למקטעים a עד h. אם החיבור היה קטודה משותפת C.C אז במקום 5v היה רשום Gnd (אדמה) ואז יש לתת "1' להדק המתאים בפורט 1 . האיור הבא מתאר חיבור תצוגת 7 מקטעים אל פורט 1 :



איור 5.17: חיבור תצוגת 7 מקטעים למיקרו בקר עם נגד אחד (מימין) ועם 8 נגדים באמצע. משמאל : מבנה המקטעים בתצוגה.

כדי להדליק את הספרה 8 יש להאיר את כל הלדים מ a ועד f ולא להאיר את h ואת g . בטבלה הבאה מתואר מה יש להוציא להדקי פורט 1 כדי להדליק את הספרה הרצויה בצורת חיבור של אנודה משותפת וקטודה משותפת.

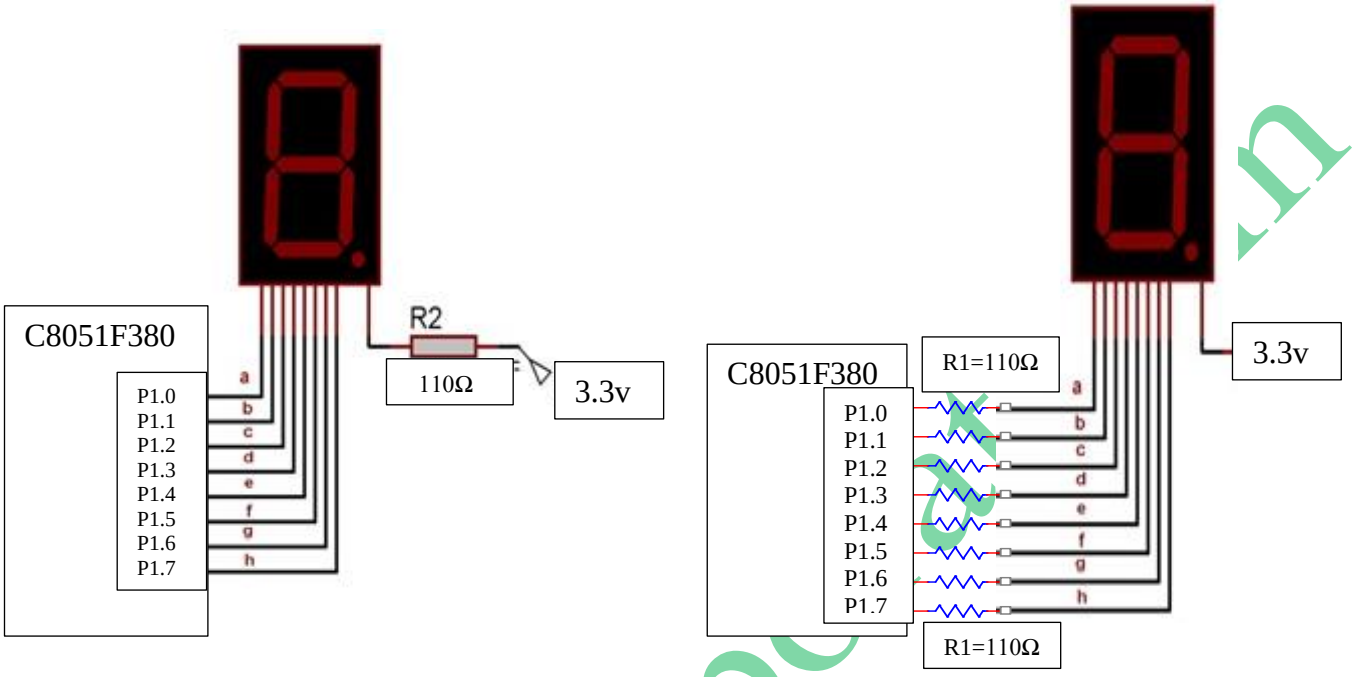
לתצוגת אנודה משותפת		
הספרה	h g f e d c b a	הערך ב HEX
0	1 1 0 0 0 0 0 0	C0
1	1 1 1 1 1 0 0 1	F9
2	1 0 1 0 0 1 0 0	A4
3	1 0 1 1 0 0 0 0	B0
4	1 0 0 1 1 0 0 1	99
5	1 0 0 1 0 0 1 0	92
6	1 0 0 0 0 0 1 0	82
7	1 1 1 1 1 0 0 0	F8
8	1 0 0 0 0 0 0 0	80
9	1 0 0 1 0 0 0 0	90

לתצוגת קטודה משותפת		
הספרה	h g f e d c b a	הערך ב HEX
0	0 0 1 1 1 1 1 1	3F
1	0 0 0 0 0 1 1 0	06
2	0 1 0 1 1 0 1 1	5B
3	0 1 0 0 1 1 1 1	4F
4	0 1 1 0 0 1 1 0	66
5	0 1 1 0 1 1 0 1	6D
6	0 1 1 1 1 1 0 1	7D
7	0 0 0 0 0 1 1 1	07
8	0 1 1 1 1 1 1 1	7F
9	0 1 1 0 1 1 1 1	6F

טבלה 5.7 : הפעלת המקטעים בחיבור אנודה משותפת בטבלה הימנית ועבור קטודה משותפת בטבלה השמאלית.

5.18.4 חישוב הזרם בכל מקטע

נתונים 2 המעגלים שבאיור הבא . שניהם מחוברים בחיבור אנודה משותפת . באיזה מהשניים נבחר ?



איור 5.18 : שני מעגלים המחוברים בחיבור אנודה משותפת. במי נבחר ?

באיור הימני רואים שכל המקטעים מחוברים דרך נגד R1 של 100 אוהם להדקי פורט 1 . באיור השמאלי יש נגד אחד . בהמשך נסביר עם מי מהשניים כדאי לעבוד. כרגע נחשב את הזרם בכל מקטע. תפקיד הנגדים לקבוע את נקודת העבודה של הלדים, כלומר לקבוע את הזרם דרך כל לד. אם נשים נגד גדול מידי יזרום דרך הלד זרם קטן ולא נראה שהיא דולקת או שהיא תאיר באור חלש מאוד. אם נשים נגד קטן מידי נגרום לנזק גם ללד וגם למיקרו . אסור לחבר את הלד ללא נגד עבודה כי נגרום נזק גם ללד וגם למיקרו. נמצא מהו הזרם הזורם דרך כל מקטע :

בעזרת חוק קירכהוף :

$3.3 = V_{LED} + V_R + V_{OL}$

V_{OL} – המתח ביציאת הדק פורט 1 במצב של '0' . אנחנו נניח שזה כ 0.5 וולט. המתח על לד בצבע אדום בהולכה הוא כ 1.7 וולט.

$3.3 = 1.7 + I*110 + 0.5 \rightarrow I = (3.3 - 1.7 - 0.5) / 110 = 1.1 / 110 = 10$ mA.

זהו זרם שגורם ללד להאיר בצורה סבירה באור חדר. באור יום (בשמש מחוץ לחדר) יש להזרים דרך הלד זרם כפול ואפילו משולש . בכל מקרה אסור לעבור את ה 50mA כי אז ייגרם לה נזק.

נבדוק עם איזה מעגל מ 2 המעגלים באיור למעלה כדאי לעבוד ? האם נעבוד עם נגד אחד במקום 8 נגדים שמוזיל את עלות המעגל במעגל עם 8 הנגדים ? (עלות המעגל עם נגד אחד זולה יותר כי יש פחות 7 נגדים, המעגל מודפס קטן יותר ויש גם פחות חורים במעגל המודפס או פחות הלחמות גל בטכנולוגית SMT) .

ובכן ... במקרה המסוים הזה, נעדיף דווקא את המעגל "היקר"... מפתיע ... ? הסיבה היא שבמעגל עם 8 הנגדים זורם דרך כל מקטע אותו זרם של 9.3 מילי אמפר . כאשר הספרה העשרונית שרוצים לראות היא 1 שבה דולקים המקטעים b ו c זורם דרך כל מקטע 9.3 מילי אמפר. גם כאשר הספרה היא 8 זורם דרך כל מקטע 9.3 מילי אמפר וכולן מאירות בעוצמת אור שווה.
כאשר נעבוד עם המעגל עם הנגד היחיד אז כאשר נרצה לראות את הספרה העשרונית 1 יזרום דרך הנגד זרם של 9.3 מילי אמפר אבל הוא מתחלק בין המקטעים b ו c ולכן בכל מקטע יהיה זרם של 9.3 / 2 = 4.65 מילי אמפר. מכאן שעוצמת האור תהיה חלשה יותר מזו שהייתה במעגל עם 8 הנגדים. כאשר נציג את הספרה 8 אז הזרם דרך הנגד מתחלק בין 7 מקטעים ובכל מקטע יהיה זרם של : 9.3 / 7 = 1.32 מילי אמפר . כל מקטע יאיר בעוצמה חלשה בהרבה מזו שבמעגל עם 8 הנגדים. עוצמת האור בכל מקטע במעגל עם הנגד האחד תלויה בכמות המקטעים הדולקים וברור שנעדיף במקרה זה את המעגל "היקר".

5.18.5 דוגמאות

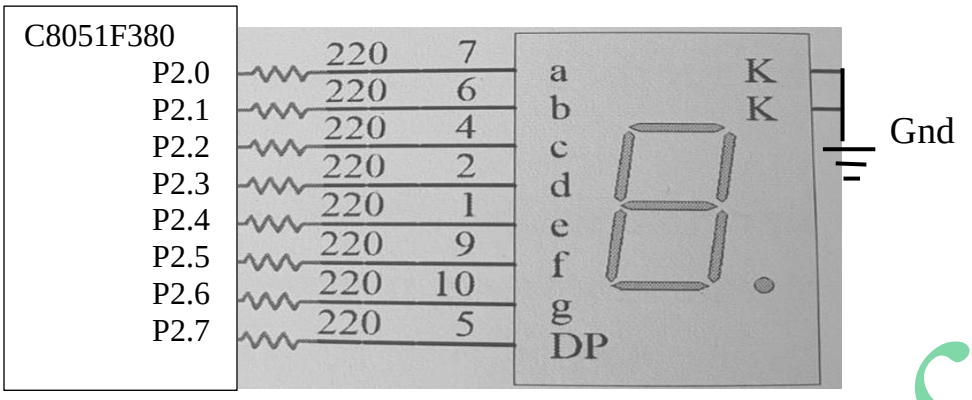
דוגמה 1 :

רשום תוכנית שתדליק את הספרות מ 0 ועד 9 בתצוגת 7 המקטעים במעגל הימני באיור הקודם בלולאה. בסיום כל ספירה נחזור לספרה 0 ונספור שוב. הנח שנתונות הפונקציות Init_Device ו delay_ms .

```
#include <REG8051F380.h>
// RAM  כדי לא לבזבז מקום ב FLASH     חיבור אנודה משותפת. שמירת המערך בזיכרון התוכנית
//                                 0     1     2     3     4     5     6     7     8
9
unsigned char code CA[10]= {0×C0, 0xF9, 0×A4, 0×B0, 0×99, 0×92, 0×82, 0xF8, 0x80, 0×90};
void main( )
{
unsigned int i,j;
Init_Device( ); // וחיבור הקרוסבר   push pull     פונקציה לאתחול  פורט 1 כפלט
while(1)  //   לולאה אין סופית
  {
   for(i=0;i<10;i++) // 9  ועד 0 לולאה של 10 פעמים כדי להציג את הספרות מ
    {
     P1=CA[i];  // i במיקום  CA מוציאים לפורט 1 את הערך שיש במערך
    delay_ms(1000);  //  שנייה 1 -  השהיה של 1000 מילי שניות
    }

  }
}
```

15.18.5.1 חיבור תצוגת 7 מקטעים בחיבור קטודה משותפת.

האיורים הקודמים הראו חיבור של תצוגת 7 מקטעים בחיבור אנודה משותפת . באיור הבא נתאר חיבור של תצוגת 7 מקטעים בחיבור קטודה משותפת.



איור 5.19 : תצוגת 7 מקטעים בחיבור קטודה משותפת.

במעגל הזה חיברנו את הדק המשותף של כל הקטודות לאדמה Gnd (GrouND) . ההדקים של המקטעים מחוברים אל פורט 2 . במקרה הזה יש לתת בהדק המתאים של פורט 2 '1' (זה בערך 3.3 וולט בג'וק C8051F380).

כדי להציג את הספרות 0 עד 9 בלולאה בדומה לתרגיל הקודם, יש לשנות את טבלת הקבועים ואת הפורט אליו מוציאים את הנתון.

סימנו את השינוי בכתב מודגש. הנח שנתונות הפונקציות Init_Device ו delay_ms .

```
#include <REG8051F380.h>
// FLASH     חיבור אנודה משותפת. שמירת המערך בזיכרון התוכנית
//טבלת המרה מעשרוני ל 7 מקטעים   0     1     2     3     4     5     6     7     8
9
unsigned char code CC[10]= {0×3F, 0x06, 0×5B, 0×4F, 0×66, 0×6D, 0×7D, 0x07, 0x7F, 0×6F};
void main( )
{
unsigned int i,j;
Init_Device( ); // פונקציה לאתחול פורט 2 כפלט push pull וחיבור הקרוסבר
while(1)   //  לולאה אין סופית
  {
  for(i=0;i<10;i++)   //  לולאה של 10 פעמים כדי להציג את הספרות מ 0 ועד 9
    {
     P2=CC[i];  // מוציאים לפורט 2 את הערך שיש במערך CC במיקום i
     delay_ms(1000);  //  השהיה של 1000 מילי שניות - 1 שנייה
    }
  }
```

הזרם בכל סיגמנט/מקטע הוא (בהנחה שמתח '1' הוא 3.3 וולט והמתח על מקטע לד 1.5v :

$3.3 = V_R + V_{LED} = I*220 + 1.5 \rightarrow I = (3.3 - 1.5) / 220 = 8.18mA$

15.18.5.2 – באיזו צורת חיבור כדאי להשתמש (האם אנודה משותפת או קטודה משותפת ?) .

רואים שבשתי התוכניות כמעט ואין הבדל חוץ מהטבלה המתאימה.

ישנם מיקרו בקרים בטכנולוגיה "ישנה" שבמצב של '1' הוציאו כמה עשרות מיקרו אמפר (SOURCE) ובמצב SINK יכלו לספוג זרם של עשרות בודדות של מילי אמפר. במיקרו בקרים כאלו נעבוד עם תצוגות 7 מקטעים בחיבור אנודה משותפת. האנודות מקבלות 5 וולט וסגירת הזרם היא דרך הסיגמנט/מקטע , דרך הדק המיקרו בקר לאדמה.

ישנם מיקרו בקרים שהזרם במצב של '1' (מצב SOURCE) והזרם במצב '0' (מצב SINK) הוא אותו זרם כמו לדוגמה בארדואינו שמוציא בהדק 40 מילי אמפר (SOURCE) או יש הטבעת זרם דרך ההדק של 40 מילי אמפר במצב SINK .

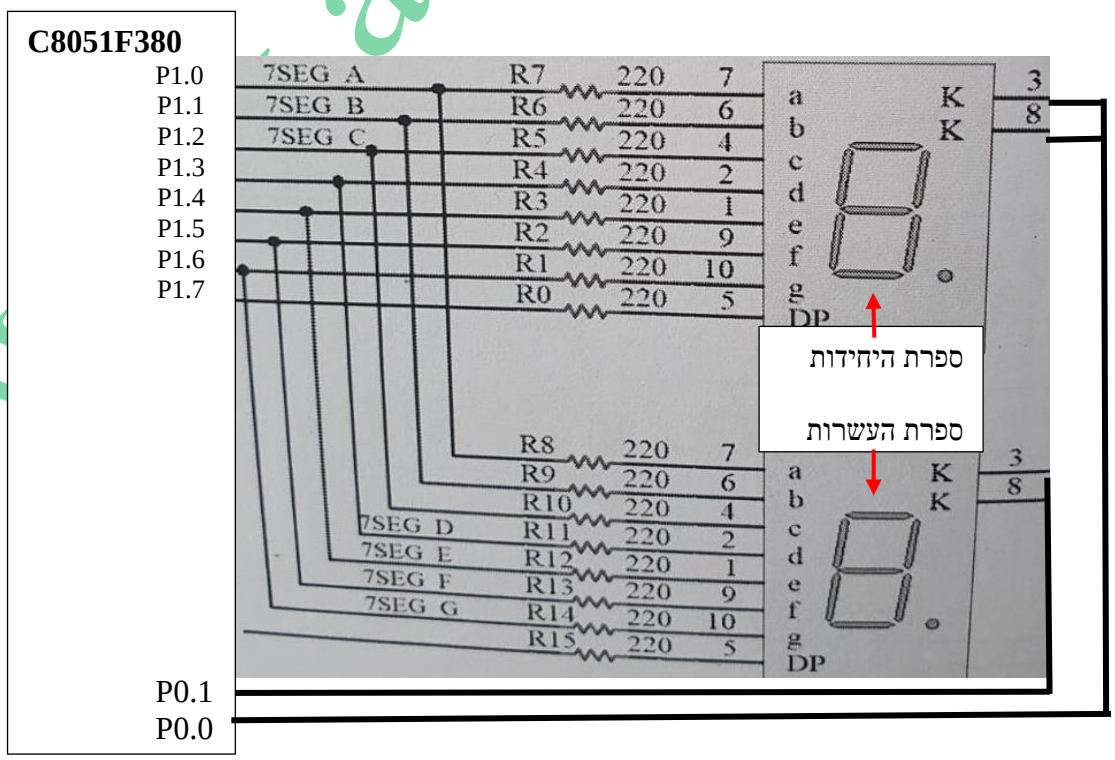
במיקרו C8051F380 הזרם במצב SINK הוא מקסימום 100mA כפי שמתואר בדפי הנתונים בשורה הבאה שנלקחה מטבלת ערכים נקובים מקסימליים – Absolute Maximum Rating:

Maximum Output Current sunk by RST or any Port Pin		—	—	100	mA

במצב SOURCE הזרם המקסימלי הוא 10mA. כאן אין הבדל בין 2 החיבורים כי לא מזרימים זרם גבוה מ 10mA בכל מקטע. כאשר נרצה זרמים גבוהים יותר בכל מקטע נעדיף חיבור אנודה משותפת כי סגירת הזרם היא דרך ההדק לאדמה וזה מצב SINK.

15.18.6 חיבור תצוגות 7 מקטעים בריבוב Time Division Multiplexing – TDM -ריבוב חלוקת זמן

כאשר נרצה לחבר מספר תצוגות 7 מקטעים יחד כדי להציג מספר ספרות , היינו צריכים באופן תאורטי כמות הדקים גדולה. לדוגמה , עבור 2 תצוגות 7 מקטעים היינו צריכים כ 16 הדקים (8 הדקים עבור ההדקים a עד h של כל תצוגה). עבור 4 תצוגות היינו צריכים 32 הדקים. במקום לחבר כמות גדולה כזו של הדקים ניתן לחבר את התצוגות במקביל ולעבוד בשיטת הריבוב. האיור הבא מתאר את המעגל:



איור 5.20 : ריבוב עם 2 תצוגות 7 מקטעים.

רעיון הריבוב הוא :

א. מדליקים את תצוגת ספרת היחידות ותצוגת ספרת העשרות כבויה.

ב. מכבים את ספרת היחידות ומדליקים את ספרת העשרות.

ג. חוזרים בלולאה אין סופית על הסעיפים א ו ב . אם עושים זאת מספיק מהר (כמה עשרות פעמים בשנייה), העין "רואה" את שני המספרים יחד. המצב נקרא "התמדה של ראייה" ובאנגלית **POV- Persistence of vision** .

ד. התהליך מתבצע בשלבים הבאים :

1. מכבים את שתי התצוגות נותנים ב P0.0=P0.1=1 ואז אין לתצוגות מסלול סגירת זרם לאדמה.
2. כדי להדליק את ספרת היחידות בלבד יש לתת בפורט 1 ערך המדליק את הספרה הרצויה ואז שמים P0.0=0 . הדבר מדליק את הספרה הרצויה בתצוגת 7 המקטעים של היחידות.
3. מבצעים השהייה קצרה כדי שהסגמנטים יידלקו והעין תבחין בספרה.
4. שמים P0.0=1 ואז 2 התצוגות בחושך.
5. שמים בפורט 1 ערך המתאים לספרה הרצויה ואז P0.1=0 . נדלקת הספרה הרצויה בתצוגת העשרות.
6. מבצעים השהייה קצרה כדי שהסגמנטים יידלקו והעין תבחין בספרה.
7. חוזרים לסעיף 1.

ריבוב כזה נקרא ריבוב חלוקת זמן - TDM - כי בכל פרק זמן מדליקים תצוגה אחרת.

נרשום תוכנית להדלקת הספרות מ 0 ועד 99 בלולאה .

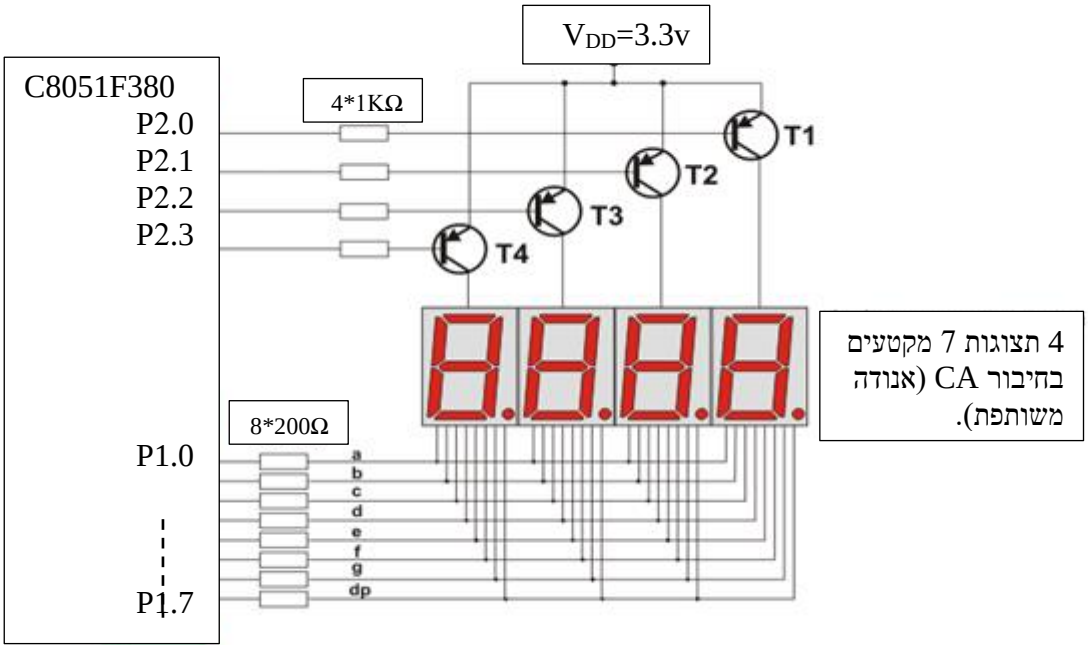
דוגמה 2 : נתונות הפונקציות Init_Device ו delay_ms .

```
#include <REG51F380.h>
// FLASH שמירת הטבלה בזיכרון ה  0    1    2    3    4    5    6    7    8    9
unsigned char code CC[ ] = { 0x3f, 0x06, 0x5b, 0x4f, 0x66 , 0x6d, 0x7d, 0x07, 0x7f, 0x6f};
sbit seg0 = P0^0;
sbit seg1 = P0^1;
void main( )
{
        unsigned char  x,y ;  // הגדרה של משתנים שנמצאים בזיכרון data ram הפנימי
        Init_Device( ); // פונקציה לאתחול פורטים 0 ו 1 כפלט push pull וחיבור הקרוסבר
        seg0 = 1 ;  // כיבוי ספרת היחידות
        seg1=1; //  כיבוי ספרת העשרות
  while (1)
        {
```

```
        for(x=0;x<100;x++)// לולאה הסופרת מ 0 ועד 99
          for(y=0;y<25;y++) // ריבוב התצוגות. מדליקים לחילופין את ספרת היחידות ואת ספרת העשרות 25 פעמים
            {
              P1= CC[x/10];  // ספרת העשרות
              seg1=0;  // אפשרות סגירת זרם לתצוגת העשרות
              delay_ms(10);  // השהייה של חצי שנייה
              seg1=1; //  כיבוי ספרת העשרות
              P1=CC[x%10];
              seg0=0;  // אפשרות סגירת זרם לתצוגת היחידות
              delay_ms(10);  // השהייה של חצי שנייה
              seg0=1; // כיבוי ספרת היחידות
            }
        }
}
```

15.18.7 חיבור 4 תצוגות 7 מקטעים בריבוב TDM

האיור הבא מתאר חיבור של 4 תצוגות 7 מקטעים בריבוב .



איור 5.21 : חיבור 4 תצוגות 7 מקטעים בריבוב TDM.

בדוגמה שבאיור התצוגות מחוברות בחיבור אנודה משותפת. הטרנזיסטורים משמשים גם כמתגים וגם כמגברי זרם . יש לזכור שהטרנזיסטור מסוג PNP . '0' באחד מהדקי P2 המתחבר לבסיס הטרנזיסטור יגרום לרווית הטרנזיסטור ומעבר מתח של :

V_{DD}-V_{CESAT}= 3.3-0.2=2.9v לקולט ולאנודה המשותפת של התצוגה והסיפרה שתידלק היא לפי הערך שנכניס ב P1. '1' באחד מהדקי P2 המתחבר לבסיס הטרנזיסטור יגרום לקטעון של הטרנזיסטור והמתח V_{DD} לא עובר לאנודה ולכן התצוגה הזו לא תאיר.

כאשר רוצים להפעיל את התצוגה הימנית יש לתת P2.0 = 0 ו P2.1=P2.2 = P2.3 = 1 . הטרנזיסטור T1 נכנס לרוויה ואילו T2 T3 T4 בקטעון. 3.3v וולט עובר דרך T1 מהאמיטר אל הקולט (נכון יותר 3.1v פחות מתח אמיטר קולט ברוויה – 0.2v - שזה כ 2.9v) לאנודה המשותפת של התצוגה הימנית. בהתאם לערך שנשים ב P1 יידלק המספר המתאים.

מדוע צריך את הטרנזיסטורים ? הסיבה לכך היא שאם היינו מחברים את הדקי P2 ישירות לאנודות המשותפות אז כדי להפעיל תצוגה כלשהי היינו נותנים בהדק המתחבר לאנודה המשותפת של התצוגה '1' . במצב של '1' ביציאה הזרם המקסימלי האפשרי הוא 10mA . אם היינו מדליקים את כל 7 המקטעים בתצוגה הזרם בכל מקטע היה 10/7=1.4mA . זרם קטן כזה, היה מדליק אור חלש במקטעים.

בכל פרק זמן קצר מבצעים את פעולת הריבוב. מדליקים ספרה רצויה בתצוגה אחת. אחר כך מכבים את הראשונה ומדליקים לזמן קצר את הספרה הרצויה בתצוגה השנייה . אחר כך מכבים את השנייה ומציגים את הספרה הרצויה בתצוגה השלישית. מכבים את השלישית ומציגים את הספרה הרצויה בתצוגה הרביעית. מיד לאחר הדלקת התצוגה הרביעית מכבים אותה וחוזרים להציג בלולאה שוב את הראשונה וכך הלאה. אם החזרה על הלולאה מתבצעת בקצב מהיר של כמה עשרות פעמים בשנייה , העין לא מבחינה שאנחנו מדליקים תצוגה ומכבים אותה אלא העין "רואה" שכל 4 התצוגות דולקות יחד.

דוגמה 3:

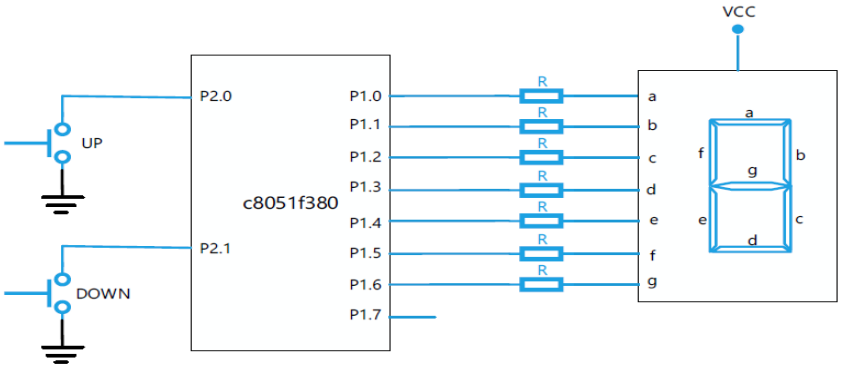
מה עושה התוכנית הבאה ? ניתן להניח שנתונות הפונקציות Init_Device ו delay_ms .

```
#include <REG51F380.h>

// שמירת הטבלה בזיכרון FLASH   0   1   2   3   4   5   6   7   8   9

unsigned char code CA[ ] = { 0xc0, 0xf9, 0xa4, 0xb0, 0x99 , 0x92, 0x82, 0xf8, 0x80, 0x90};

void main( )

{

        int  x,y ;  // הגדרה של משתנים שנמצאים בזיכרון ה xdata

        Init_Device( ); // פונקציה לאתחול פורטים 0 ו 1 כפלט push pull וחיבור הקרוסבר

        P2=0xff;  // כיבוי כל התצוגות

  while (1)

        {
```

```
        for(x=0;x<10000;x++)          // לולאה הסופרת מ 0 ועד 9999
          for(y=0;y<25;y++) // ריבוב התצוגות. מדליקים לחילופין את ספרת האלפים המאות העשרות והיחידות 25 פעמים
           {
                P1=CA[x/1000];
                P2=0xf7; // הדלקת ספרת האלפים בלבד
                delay_ms(2);  //השהייה של 2 אלפיות שנייה
                P2=0xff;  // כיבוי כל התצוגות
                P1=CA[x/100%10]; // ספרת המאות
                P2=0xfb; // הדלקת ספרת המאות בלבד
                delay_ms(2);  //השהייה של 2 אלפיות שנייה
                P2=0xff;  // כיבוי כל התצוגות
                P1= CA[x/10%10];  // ספרת העשרות
                P2=0xfd;  // הדלקת ספרת העשרות בלבד
                delay_ms(2);  //השהייה של 2 אלפיות שנייה
                P2=0xff;  // כיבוי כל התצוגות
                P1=CA[x%10]; // ספרת היחידות
                P2=0xfe;  // אפשרות סגירת זרם לתצוגת היחידות
                delay_ms(2);  //השהייה של 2 אלפיות שנייה
                P2=0xff; // כיבוי כל התצוגות
        }
    }
}
```

דוגמה 4 :

נתונה המערכת הבאה הכוללת מיקרו-בקר C8051F380, תצוגה 7 Segment ושני לחצנים (ללא ריטוטים).



להלן קוד התוכנית

```
#include "compiler_defs.h"
#include "C8051F380_defs.h"
void Init_Device(void);
code unsigned char table[10]={ 0x3f, 0x06, 0x5b, 0x4f, 0x66,
                               0x6d, 0x7d, 0x07, 0x7f, 0x6f };
sbit UP=P2^0;
sbit DOWN=P2^1;
void main(){
      int cnt=0;
      Init_Device();
      P1=~table[0];
      while(1)     {
            if(!UP){
                  if (cnt<9) cnt++;
                  P1=~table[cnt];
                  while(!UP);
            }

            if(!DOWN){
                  if (cnt>0) cnt--;
                  P1=~table[cnt];
                  while(!DOWN);
            }
      }
}
```

א. הסבר מה מבצעת התוכנית.

ב. מה יוצג בתצוגת 7segment בזמן אתחול הבקר.

ג. הסבר את הפקודה : P1=~table[cnt];

ד. למה נועדו הפקודות : while(!UP); ו- while(!DOWN);

ה. שנה את התוכנית כך שלחיצה אחת על לחצן UP תקדם את המונה כל שנייה מהערך הקיים עד 9 ותעצור ולחיצה אחת על DOWN תמנה מטה כל שנייה עד לערך 0 ותעצור.

לצורך יצירת השהיות בין הדלקות נתונה פונקציית השהייה במילי שניות :

```
delay_ms(int ms)
```

פתרון 4 :

א. התוכנית משמשת כמונה בתחום 9- 0. קידום ערך המונה באחד מתבצע באמצעות לחצן UP, חיסור אחד מערך המונה מתבצע באמצעות לחצן DOWN . תוצאת המניה תוצג על גבי תצוגה 7 Segment .

ב. בזמן אתחול הבקר והפקודה P1=~table[0]; תגרום להצגת הספרה 0 בתצוגה.

ג. הפקודה P1=~table[cnt]; תשלח ל-P1 ערך שיגרום להצגת הערך של cnt בתצוגה

הסבר : cnt מצביע על מיקום במערך מ-0 עד 9 , הערך מהטבלה עובר היפוך ~ וגורם לנורות ה-LED המקבלים '0' לוגי לדלוק (אנודה משותפת)

לדוגמה – עבור P1 cnt=1 יקבל את ההיפוך של 0x06 כלומר 11111001 לכן נורות b LED ו- c יפעלו ונקבל את הערך 1 בתצוגה.

ד. הפקודות נועדו להמתנה לשחרור הלחצן , אחרת המונה היה מתקדם במהירות לפי זמן הלולאה.

```
#include "compiler_defs.h"
#include "C8051F380_defs.h"
void Init_Device(void);
void delay_ms(int ms);
code unsigned char table[10]= {0x3f, 0x06, 0x5b, 0x4f, 0x66,
                               0x6d, 0x7d, 0x07, 0x7f, 0x6f };
sbit UP=P2^0;
sbit DOWN=P2^1;
void main(){
      int cnt=0;
      Init_Device();
      P1=~table[0];
      while(1)    {
            if(!UP){
                  while(cnt<9){
                        delay_ms(1000);
                        cnt++;
                        P1=~table[cnt];
                  }
            }

            if(!DOWN){
                  while(cnt>0){
                        delay_ms(1000);
                        cnt--;
                        P1=~table[cnt];
                  }
            }
      }
}
```

6. פסיקות Interrupts

6.1 פסיקה – הסבר כללי והשוואה לשאילתה (Polling)

המילה interrupt באנגלית היא "הפרעה, קטיעה, שיסוע (באמצע דיבור)" בעברית. לדוגמא היינו עסוקים בקריאת ספר והטלפון מצלצל וקוטע את הקריאה ומפריע לנו . אנחנו נשים סימניה בשורה אליה הגענו בספר, נענה לטלפון ונטפל בהפרעה. בסיום הטיפול נחזור לקרוא את הספר , ובעזרת הסימנייה , נחזור בדיוק לשורה שבה היינו לפני שענינו לטלפון.

הפסיקה במערכות מחשבים מתארת מצב שבו המחשב מבצע תכנית מסוימת ובזמן כלשהו , אחת המערכות או הרכיבים המתחברים למחשב מפריעים/מפריעות למהלך התוכנית ומבקשות טיפול. על המחשב לטפל במערכת/רכיב שהפריעו וביקשו טיפול ובסיום הטיפול צריך לחזור לאותה השורה בתוכנית שהמחשב היה.

הערה חשובה: פסיקה –הפרעה – נחשבת בחיי היום יום כפעולה שלילית. במעבדים ומיקרו בקרים היא דרך עבודה מקובלת ולא מתייחסים אליה בצורה שלילית . התקני הקלט והפלט במחשב האישי מתחברים בעזרת פסיקות המחוברות כולן אל בקר פסיקות.
במחשב נקראות בקשות הפסיקה IRQ קיצור של Interrupt ReQuest או דרישת פסיקה.
תפקיד המעבדים/מיקרו בקרים הוא לנהל/לשלוט על כל מה התהליכים הקורים במחשב . ומכאן הם התייחסו למעבדים/מיקרו בקרים בהקבלה למנהל במפעל או מנהל חברה או מנהל מוסד חינוכי. בדמיונם הם תיארו 2 סוגי מנהלים :

1. **מנהל העובד עם פסיקות.**

   זהו מנהל אשר יושב במשרדו ועסוק בעבוד ניהול (מטרות, שיווק, משכורות וכו'). נניח שהמנהל קורא מסמך המתאר את מצבה הפיננסי של החברה. אחד העובדים רוצה לדבר או להתייעץ עם המנהל. העובד בא לחדרו של המנהל , דופק על הדלת ומפריע למנהל בעבודתו . תהליך הטיפול של המנהל בהפרעה יהיה לפי השלבים הבאים :

   א. המנהל מסיים לקרא את המשפט האחרון שאליו הגיע .
   ב. המנהל מסמן בצורה כלשהי (אולי בעזרת סימנייה ...) את הנקודה שבה הפסיק את עבודתו, כלומר הוא מסמן את המשפט הבא במכתב אותו יצטרך לקרא אחרי הטיפול בהפרעה.
   ג. המנהל מטפל בעובד המפריע (לכל עובד יש תוכנית טיפול/פתרון אישי).
   ד. בסיום הטיפול בעובד , המנהל חוזר למכתב שקרא , לאותו המשפט שצריך להמשיך ממנו בעזרת הסימון שעשה מקודם.

2. **מנהל העובד עם שאילתות Polling .**

   מנהל כזה איננו עובד עם פסיקות . הוא איננו מוכן לקבל הפרעות בזמן שהוא עסוק וממוקד בקריאת המסמך. הוא שם שלט על הדלת : "נא לא להקיש על הדלת. נא רשום שמך על דף ברשימה שבהמשך... " . המנהל קורא את המסמך וכל פרק זמן מסוים (כאשר הוא מסיים פרק במסמך לדוגמה) הוא עוצר את העבודה ובודק ברשימה האם מישהו ביקש טיפול. במידה וכן הוא מטפל בעובד/ים הרשומים בדף. במידה ולא הוא ממשיך בקריאת המסמך עד הפרק הבא ושוב התהליך חוזר על עצמו.

ההבדל בין 2 שיטות העבודה הוא שבעבודה עם פסיקות העובדים הם היוזמים את בקשת הפסיקה (או הרכיבים הפריפריאליים המתחברים למיקרו) ובעבודה עם שאילתה המנהל(או המיקרו בקר) הוא היוזם והוא שואל את העובדים האם הם צריכים טיפול.
כמובן שיש מנהלים המשלבים בין 2 השיטות . גם במיקרו ניתן לעבוד באחת השיטות הרצויות או לשלב בין 2 השיטות.
חסרון ה POLLING הוא בזבוז זמן מעבד שצריך לפנות כל פרק זמן לרכיבים.

6.2 חסימה / אפשור פסיקות ועדיפות פסיקות.

מנהל העובד עם פסיקות, יכול לקבוע את אפשרויות העבודה הבאות :

1. כאשר הוא עסוק בנושא מהותי/חשוב/דורש התמקדות , הוא יכול לקבוע שהוא איננו נענה / מסכים לקבל הפרעות. אנשים יכולים לדפוק על הדלת , לצלצל אליו לטלפון או לנסות כל פעולה אחרת. המנהל נועל את הדלת ומנתק את הטלפון. הוא איננו מטפל באף הפרעה. בצורה כזו הוא חסם את כל הפסיקות.
2. המנהל יכול לקבוע באיזו הפרעה הוא מוכן לטפל ואיזו לא. לדוגמא : אם מדובר באחד המנהלים הבכירים הוא מוכן לטפל ואם מדובר בעובד זוטר הוא איננו מוכן לטפל בו כרגע. למעשה הוא קובע איזו פסיקה/הפרעה הוא מאפשר ואיזו הוא חוסם.
3. המנהל גם יכול לקבוע סדר עדיפות – Priority . במידה ומגיעות מספר פסיקות/הפרעות יחד – כמה עובדים מגיעים יחד - המנהל קובע באיזו הפרעה הוא יטפל ראשונה , מי אחריה וכך הלאה. כיצד יכולות להגיע כמה הפרעות בו זמנית ? ניקח דוגמא שבה המנהל חסם את כל ההפרעות והגיעו מספר עובדים וכולם רוצים טיפול והם ממתינים ליד חדר המנהל. ברגע שהמנהל יאפשר פסיקות הוא רואה שיש מספר פסיקות בו זמנית והוא יטפל בהן לפי סדר עדיפות שהוא קובע.

נתאר בעזרת הטבלה הבאה תהליך הענות לפסיקה במיקרו בקר ממשפחת ה 51 . תיארנו מקודם את השלבים בטיפול בפסיקה של מנהל מפעל/חברה/מוסד חינוכי. נראה את הפעולות המקבילות במיקרו : נצא מתוך ההנחות הבאות :

המנהל קורא את משפט/סעיף מספר 100 במסמך. המשפט/סעיף הבא הוא משפט מספר 101 .

המיקרו מבצע את הפקודה בכתובת (שורה) 100 בתוכנית. הפקודה הבאה נמצאת בכתובת 101 .

ועכשיו מגיעה בקשת פסיקה – על ידי עובד במפעל או פסיקה במיקרו :

השלב בהיענות לפסיקה	מנהל המפעל/חברה	מיקרו בקר	הערות
1	המנהל מסיים את המשפט/סעיף במסמך שהוא קורא (משפט 100).	המיקרו מסיים את הפקודה/שורה 100	המנהל מסיים לקרא את משפט 100 והמיקרו מסיים לבצע את הפקודה שבכתובת 100
2	המנהל שומר בעזרת סימנייה את מיקום המשפט/סעיף אליו הגיע (משפט מספר 101 במסמך).	המיקרו שומר בזיכרון במקום שנקרא "מחסנית" את הכתובת/שורה של הפקודה אליה הגיע ("כתובת החזרה שהיא 101").	המנהל שם את הסימנייה על כתובת 101 . המיקרו מכניס למחסנית את כתובת 101 .
3	המנהל מכניס את העובד ומטפל בו. לכל עובד יש טיפול/פתרון אישי. בזמן הזה הוא איננו מקבל	המיקרו עובר לתוכנית שמטפלת במפריע. לכל מפריע תוכנית (כתובת) טיפול משלו. המיקרו שם בביט EA ברגיסטר IE את הערך 0 כדי לחסום את כל הפסיקות ולא	

	פסיקות נוספות ומקדיש את עצמו לעובד בלבד.	לקבל פסיקות נוספות אלא רק לטפל במפריע המסוים.	
4	בסיום הטיפול הוא אומר לעובד "שלום, ביי..." וחוזר בעזרת הסימנייה אל המשפט/סעיף במסמך שאליו הגיע (בעזרת הסימניה). עכשיו הוא יכול לקבל פסיקות נוספות.	בסיום תוכנית הטיפול מופיעה פקודת reti (קיצור של RETurn from Interrupt - חזור מפסיקה) . המיקרו מושך מהמחסנית את כתובת החזרה וחוזר אל הפקודה אליו הגיע. הוא מחזיר לביט EA את הערך 1 כדי לאפשר קבלת פסיקות נוספות.	המנהל חוזר למשפט מספר 101 (בעזרת הסימנייה) וממשיך לקרא את המסמך ממשפט זה. המיקרו מושך מהמחסנית את כתובת 101 וממשיך את התוכנית מכתובת זו.

טבלה מספר 6.א – תהליך השוואה בין הענות לבקשת פסיקה במיקרו לבקשת טיפול של עובד

ברכיבים במשפחת ה 51 יש 2 רמות עדיפות : גבוהה ונמוכה . אם 2 בקשות פסיקה מגיעות יחד אז קודם תטופל הפסיקה עם העדיפות הגבוהה. אם מגיעות 2 פסיקות עם אותה רמת עדיפות אז יש עדיפות "טבעית" אותה קבע היצרן והפסיקה שתטופל ראשונה תהיה על פי סדר העדיפות הטבעי. הסבר מפורט יותר יהיה כאשר נדבר על רגיסטר עדיפות פסיקה.

זמן התגובה לבקשת פסיקה איננו קבוע ותלוי במצב המעבד. הזמן המהיר ביותר הוא 6 מחזורי שעון. מחזור אחד כדי לזהות את סוג הפסיקה ועוד 5 מחזורי שעון כדי להשלים את ה lcall לכתובת של וקטור הפסיקה. זמן מקסימלי הוא 20 מחזורי שעון וזה כאשר יש פקודת div הבאה אחרי reti . מחזור אחד כדי לזהות את סוג הפסיקה, 6 מחזורי שעון של חזרה מפסיקה (reti) , 8 מחזורי שעון של div ועוד 5 מחזורי שעון כדי להשלים את ה lcall לכתובת של וקטור הפסיקה

6.3 סוגי פסיקות במיקרו בקר - מקורות פסיקה.

ברכיב ממשפחת ה 51 המקורי היו 5 מקורות פסיקה (ברכיבים המסתיימים ב 2 ו 3 יש 6 מקורות פסיקה ואפילו יותר). במילים "מקורות פסיקה" הכוונה היא מאיפה יכולה להגיע הפסיקה או מי יכול לבקש פסיקה. ברכיב C8051F380 יש 22 מקורות פסיקה. במיקרו בקר ממשפחת ה 51 המקורי מקורות הפסיקה היו :

1. 2 פסיקות חיצוניות הנכנסות בהדקים **$\overline{\text{INT0}}$** ו **$\overline{\text{INT1}}$** של המיקרו. פסיקות אלו מתקבלות כאשר המתח בהדק ירד מגבוה לנמוך, כלומר מ 1 ל 0 . הגג מעל ההדקים אומר שהפסיקות פעילות בנמוך. כלומר כאשר בהדק יהיה 0 מתבקשת פסיקה.
2. פסיקות טיימרים. כאשר טיימר 0 או טיימר 1 מסיימים את הספירה שלהם הם נותנים בקשת פסיקה. ברכיבים המסתיימים ב 2 ו 3 יש גם את טיימר 2 . גם הוא נותן פסיקה בסיום הספירה שלו.
3. פסיקת תקשורת טורית הנקראת UART או SERIAL . כאשר עובדים עם מעגל התקשורת הטורית והסתיים שידור ביית טורי, מעגל התקשורת הטורית מודיע על סיום השידור בעזרת פסיקה. דבר דומה קורה כאשר מסתיימת קליטה של ביית טורי. בסיום הקליטה מעגל התקשורת הטורית מודיע בעזרת פסיקה על סיום קליטת ביית טורי.

סיכום : במיקרו בקר mcs51 של אינטל היו : 2 פסיקות חיצוניות + 2 פסיקות טיימרים + פסיקת תקשורת טורית (סיום שידור טורי וסיום קליטה טורית נותנים אותה פסיקה) : סה"כ 5 מקורות פסיקה.

הערה : נהוג לכלול את מצב האתחול RESET כסוג של פסיקה כך שיש להוסיף פסיקה זו לכמות הפסיקות שהזכרנו.

במיקרו בקר C8051F380 יש 22 מקורות פסיקה (כולל RESET ועוד פסיקה ללא שימוש (רזרבה)). לכל פסיקה יש תוכנית טיפול משלה הנמצאת בכתובת קבועה וידועה שהחליטו יצרני הרכיב. כתובת זו נקראת בשפה מקצועית וקטור פסיקה ותוכנית הטיפול בפסיקה נקראת : **ISR** – Interrupt Service Routine - שגרת הטיפול בפסיקה.

הטבלה הבאה מסכמת את נושא הפסיקות :

מקור הפסיקה	וקטור הפסיקה	סדר העדיפות	דגל (ביט) שמראה שיש בקשת פסיקה	מיעון ביט ?	מאופס על ידי חומרה ?	ביט אפשור	ביט קביעת עדיפות
Reset	0x0000	Top	None	N/A	N/A	Always Enabled	Always Highest
External Interrupt 0 (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 (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 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)
Programmable Counter Array	0x005B	11	CF (PCA0CN.7) CCFn (PCA0CN.n)	Y	N	EPCA0 (EIE1.4)	PPCA0 (EIP1.4)
Comparator0	0x0063	12	CP0FIF (CPT0CN.4) CP0RIF (CPT0CN.5)	N	N	ECP0 (EIE1.5)	PCP0 (EIP1.5)
Comparator1	0x006B	13	CP1FIF (CPT1CN.4) CP1RIF (CPT1CN.5)	N	N	ECP1 (EIE1.6)	PCP1 (EIP1.6)
Timer 3 Overflow	0x0073	14	TF3H (TMR3CN.7) TF3L (TMR3CN.6)	N	N	ET3 (EIE1.7)	PT3 (EIP1.7)
VBUS Level	0x007B	15	N/A	N/A	N/A	EVBUS (EIE2.0)	PVBUS (EIP2.0)
UART1	0x0083	16	RI1 (SCON1.0) TI1 (SCON1.1)	N	N	ES1 (EIE2.1)	PS1 (EIP2.1)
Reserved	0x008B	17	N/A	N/A	N/A	N/A	N/A
SMB1	0x0093	18	SI (SMB1CN.0)	Y	N	ESMB1 (EIE2.3)	PSMB1 (EIP2.3)
Timer 4 Overflow	0x009B	19	TF4H (TMR4CN.7) TF4L (TMR4CN.6)	N	N	ET4 (EIE2.4)	PT4 (EIP2.4)
Timer 5 Overflow	0x00A3	20	TF5H (TMR5CN.7) TF5L (TMR5CN.6)	Y	N	ET5 (EIE2.5)	PT5 (EIP2.5)

טבלה 6.ב : טבלה מסכמת - פסיקות

העמודות בטבלה המסכמת הן (משמאל לימין) :

- מהו **מקור הפסיקה** (מי מבקש את הפסיקה) ?
- **וקטור הפסיקה** (באיזו כתובת נמצאת תוכנית הפסיקה). בין וקטור אחד לשני 8 כתובות. ברוב המקרים נמצא כאן פקודת lcall או ljmp לכתובת כלשהי במרחב זיכרון הכתובות כי 8 כתובות בדרך כלל לא מספיקות לכתיבת תוכנית פסיקה.
- **סדר העדיפות** של היצרן. העדיפות הגבוהה ביותר - 0 (של פסיקה חיצונית 0) והעדיפות הנמוכה ביותר– 20 לטיימר 5 . **סדר העדיפות הוא גם מספר הפסיקה כשכותבים פונקציה בשפת C51** .
- איזה **ביט** מראה שיש **פסיקה** (בסוגריים – הרגיסטר שבו הוא נמצא). הביטים שימושיים באופן עבודה – שאילתה - POLLING.
- **מיעון ביט ?** האם ניתן לפנות לביט במיעון ביט ? פקודה לדוגמה : jb ie0, address_x או if(IE0) בשפת C51.
- **מאופס על ידי חומרה ?** האם כאשר נענים לפסיקה וחוזרים לתוכנית הראשית אחרי reti הביט מאופס על ידי החומרה ?
- **ביט אפשור .** שם בביט המאפשר/חוסם את הפסיקה ובסוגריים באיזה רגיסטר נמצא הביט.
- **ביט קביעת עדיפות**. איך נקרא הביט שמאפשר קביעת עדיפות נמוכה (0) או גבוהה (1) ובאיזה רגיסטר נמצא הביט.

6.4 הרגיסטרים שמטפלים בפסיקות

במיקרו המקורי ממשפחת ה 51 היו 3 רגיסטרים המבקרים על מערכת הפסיקות ובמיקרו C8051F380 יש 4 רגיסטרים .

א. IE – Interrupt Enable – רגיסטר אפשור הפסיקות.

ב. IP – Interrupt Priority – רגיסטר עדיפות הפסיקות.

ג. TCON - Timer CONtrol - רגיסטר בקרת הטיימרים - שבו 4 הסיביות הנמוכות שייכות **לפסיקות חיצוניות בלבד** (ולא לשאר הפסיקות) ו 4 הסיביות הגבוהות שייכות לבקרה על הטיימרים (ומכאן שמו של הרגיסטר).

ד. במיקרו בקר C8051F380 הוסיפו רגיסטר IT01CF

במילים " פסיקות חיצוניות " הכוונה שרכיב או מערכת מחוץ למיקרו מבקש פסיקה. שאר הפסיקות מגיעות מהרכיבים הפריפריאליים הפנימיים שבמיקרו כמו הטיימרים המשווים , התקשורות וכו'. למיקרו יש 2 פסיקות חיצוניות הנקראות פסיקה חיצונית 0 ופסיקה חיצונית 1 . רכיב חיצוני שרוצה טיפול צריך להתחבר להדק $\overline{INT0}$ או $\overline{INT1}$ ולהוריד את המתח בהדק ל 0 (הפסיקות ב 51 המקורי הופעלו בנמוך).

למיקרו C8051F380 22 מקורות פסיקה וגם אם נוריד את ה RESET אז יש 21 פסיקות ומכאן שיש להוסיף עוד רגיסטרים לאלו שלמעלה כדי לשלוט על כמות כזו של פסיקות. מכאן שהוסיפו 2 רגיסטרים של אפשור פסיקות **EIE1** ו **EIE2** (Extended Interrupt Enable – הרחבת אפשור פסיקות 1 ו 2), עוד 2 רגיסטרים של עדיפות פסיקות EIP1 ו EIP2 (גם כאן התו E בהתחלה הוא Extended – הרחבה) ועוד רגיסטר בשם **IT01CF** - InT**0**/intr**1** ConFiguration – תצורת הפסיקות החיצוניות 0 ו 1 . בעזרת הרגיסטר קובעים האם הפסיקה תתקבל כאשר יש 0 או כאשר יש 1 או כאשר יש ירידה מ 1 ל 0 או כאשר יש עליה מ 0 ל 1 . כמו כן בעזרתו קובעים אילו הדק/ים של פורט 0 יהיה הדק/י הפסיקה החיצונית 0 ו 1 .

6.4.1 רגיסטרים לאפשור פסיקות

בעזרת הרגיסטרים "אפשור פסיקה" המתכנת קובע אילו פסיקות הוא מאפשר ואילו פסיקות הוא חוסם.

בעזרת הרגיסטרים "עדיפות פסיקה" המתכנת קובע את רמת העדיפות של כל פסיקה (נמוכה או גבוהה).

הרגיסטר IE - Interrupt Enable – אפשור פסיקות - הוא רגיסטר שהיה גם ב 8051 המקורי. במיקרו C8051F380 הוסיפו עוד 2 רגיסטרים הנקראים EIE1 ו EIE2 שהם קיצור של **E**xtended **E**nable **I**nterrupt - ובעברית - הרחבת אפשור פסיקות. עם שני הרגיסטרים הנוספים האלה שולטים על שאר הפסיקות שנוספו.

הרגיסטר IP - Interrupt Priority – עדיפות פסיקה – גם הוא היה ב 8051 המקורי. במיקרו C8051F380 הוסיפו עוד 2 רגיסטרים הנקראים EIP1 ו EIP2 שהם קיצור של Extended Interrupt Priority – ובעברית - הרחבת עדיפות פסיקה. עם שני הרגיסטרים הנוספים האלו שולטים על עדיפות הפסיקות של שאר הפסיקות שנוספו.

הרגיסטר TCON – לרגיסטר הזה 8 ביטים. 4 הביטים הנמוכים שולטים על הפסיקות החיצוניות הנקראות פסיקה חיצונית 0 ופסיקה חיצונית 1. מתוך 4 הביטים הנמוכים 2 ביטים משמשים כדגלים ומציינים שהייתה בקשת פסיקה חיצונית 0 או 1 בהדקים $\overline{INT0}$ ו $\overline{INT1}$ בהתאמה ובעזרת 2 ביטים נוספים IT0 עבור פסיקה חיצונית 0 ו IT1 עבור פסיקה חיצונית 1 הקובעים האם הפסיקה תופעל על רמה לוגית – LEVEL - (0 לוגי או 1) או על שפה – EDGE (ירידה מ 1 ל 0 או עלייה מ 0 ל 1) .

למיקרו C8051F380 יש רגיסטר נוסף ששמו **IT01CF** שבו ניתן גם לקבוע בעזרת ביט קוטביות POLARITY שנקרא IN0PL לפסיקה חיצונית 0 או IN1PL לפסיקה חיצונית 1 שהפסיקה תתקבל בעזרת 4 אפשרויות אלו :

א. רמת מתח של '0' מתח המוגדר כ 0 לוגי (בין 0 וולט ל 0.8 וולט). לדוגמה עבור פסיקה חיצונית 0 : IT0=0 IN0PL=0

ב. רמת מתח של '1' המוגדרת בין 2.8 וולט ל 3.3 וולט. לדוגמה עבור פסיקה חיצונית 0 : IT0=0 =1IN0PL

ג. ירידה מ 1 ל 0 . לדוגמה עבור פסיקה חיצונית 0 : IT0= 1 IN0PL=0

ד. עלייה מ 0 ל 1 . לדוגמה עבור פסיקה חיצונית 0 : IT0=1 IN0PL=1

נסביר את הרגיסטרים של מערכת הפסיקות ונתחיל עם **רגיסטר IE** – Interrupts Enable – אפשור פסיקות המתואר באיור הבא:

Bit	7	6	5	4	3	2	1	0
Name	EA	ESPI0	ET2	ES0	ET1	EX1	ET0	EX0
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 = 0xA8; SFR Page = All Pages; Bit-Addressable

Bit	Name	Function
7	EA	**Enable All Interrupts.** Globally enables/disables all interrupts. It overrides individual interrupt mask settings. 0: Disable all interrupt sources. 1: Enable each interrupt according to its individual mask setting.
6	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.
5	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.
4	ES0	**Enable UART0 Interrupt.** This bit sets the masking of the UART0 interrupt. 0: Disable UART0 interrupt. 1: Enable UART0 interrupt.
3	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.
2	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.
1	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.
0	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.

איור 6.א : רגיסטר איפשור הפסיקות IE

השורות בחלק העליון באיור :

שורה ראשונה - מספר הביט ברגיסטר. **שורה שנייה** - שם הביט. **שורה שלישית** – האם הביט לקריאה או כתיבה Read/Write ?

שורה רביעית – מה מצב הביטים אחרי RESET . כל הביטים ב 0 וזה אומר שכל הפסיקות ממוסכות, כלומר חסומות ולא מאופשרות.

השורה הבאה מראה שכתובת הרגיסטר באזור ה SFR הוא A8H . הוא נמצא בכל דפי ה SFR וניתן לפנות אליו במיעון ביט .

נסביר את הביטים ברגיסטר IE מביט 7 ועד ביט 0 .

הערה : אם שמים בביט מסוים 0 זה חוסם את הפסיקה ו 1 מאפשר את הפסיקה .

EA - Enable All - אפשור כללי - אם נשים 0 בביט זה, חוסמים את כל הפסיקות ללא תלות בערך הקיים בביטים מימין. אם נשים 1 בביט אז נאפשר את הפסיקות הרשומות מימין ובתנאי שיש בביט שלהן 1 .

ESPI0 - Enable Serial Peripheral Interface 0 –אפשור פסיקה ממשק טורי היקפי מספר 0 . חסימה - 0 , אפשור 1 .
הערה : (למעשה יש רק תקשורת SPI אחת. אולי זה הכנה לרכיב שיהיו בו 2 תקשורות SPI ??)

ET2 – Enable Timer 2 **-** אפשור פסיקת טיימר מספר 2 . 0 לא מאפשר פסיקה מטיימר 2 . 1 – מאפשר.

ES0 - Enable Serial 0 - אפשור טורי 0. אם נשים 1 בביט מאפשרים קבלת פסיקה טורית של UART0. אם נשים 0 – חוסמים קבלת פסיקה טורית של UART0. ברכיב יש 2 תקשורות טוריות של UART שמספרן 0 ו 1 . הביט הזה מאפשר או חוסם פסיקת UART מספר 0 .

ET1 - Enable Timer 1 - אפשור טיימר 1 - אם נשים בביט 1 - מאפשרים קבלת פסיקת Timer1. 0 בביט לא מאפשר קבלת פסיקה מ Timer1 .

EX1 - Enable eXternal 1 – אפשר חיצוני 1 – אם שמים 1 בביט זה – מאפשרים קבלת פסיקה חיצונית 1 . 0 בביט זה חוסם קבלת פסיקה חיצונית 1 (הפסיקה מגיעה מהדק $\overline{INT1}$ של הרכיב).

ET0 - Enable Timer 0 - אפשור טיימר 0 - כמו ההסבר ל ET1 אבל לגבי Timer0 .

EX0 - Enable eXternal 0 – אפשר חיצוני 0 – אם שמים 1 בביט זה – מאפשרים קבלת פסיקה חיצונית מספר 0 . אם שמים 0 בביט זה חוסמים קבלת פסיקה חיצונית 0 (הפסיקה מגיעה מהדק $\overline{INT0}$ של הרכיב).
ישנם 2 רגיסטרים לאפשור פסיקות נוספים EIE1 ו EIE2 לשאר הפסיקות. בספר הזה נתמקד על סוגי הפסיקה שברגיסטר IE .
באיור הבא נתאר את מבנה 2 הרגיסטרים הנוספים לאפשור פסיקות EIE1 ו EIE2 .

Bit	7	6	5	4	3	2	1	0
Name	ET3	ECP1	ECP0	EPCA0	EADC0	EWADC0	EUSB0	ESMB0
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 = 0xE6; SFR Page = All Pages

Bit	Name	Function
7	ET3	**Enable Timer 3 Interrupt.** This bit sets the masking of the Timer 3 interrupt. 0: Disable Timer 3 interrupts. 1: Enable interrupt requests generated by the TF3L or TF3H flags. אפשור פסיקת טיימר 3
6	ECP1	**Enable Comparator1 (CP1) Interrupt.** This bit sets the masking of the CP1 interrupt. 0: Disable CP1 interrupts. 1: Enable interrupt requests generated by the CP1RIF or CP1FIF flags. אפשור פסיקת משווה 1
5	ECP0	**Enable Comparator0 (CP0) Interrupt.** This bit sets the masking of the CP0 interrupt. 0: Disable CP0 interrupts. 1: Enable interrupt requests generated by the CP0RIF or CP0FIF flags. אפשור פסיקת משווה 0
4	EPCA0	**Enable Programmable Counter Array (PCA0) Interrupt.** This bit sets the masking of the PCA0 interrupts. 0: Disable all PCA0 interrupts. 1: Enable interrupt requests generated by PCA0. אפשור פסיקת מערך מונים מתוכנת 0
3	EADC0	**Enable ADC0 Conversion Complete Interrupt.** This bit sets the masking of the ADC0 Conversion Complete interrupt. 0: Disable ADC0 Conversion Complete interrupt. 1: Enable interrupt requests generated by the AD0INT flag. אפשור פסיקת ADC0
2	EWADC0	**Enable Window Comparison ADC0 Interrupt.** This bit sets the masking of ADC0 Window Comparison interrupt. 0: Disable ADC0 Window Comparison interrupt. 1: Enable interrupt requests generated by ADC0 Window Compare flag (AD0WINT). אפשור פסיקת חלון השוואה ADC0
1	EUSB0	**Enable USB (USB0) Interrupt.** This bit sets the masking of the USB0 interrupt. 0: Disable all USB0 interrupts. 1: Enable interrupt requests generated by USB0. אפשור פסיקת USB0
0	ESMB0	**Enable SMBus0 Interrupt.** This bit sets the masking of the SMB0 interrupt. 0: Disable all SMB0 interrupts. 1: Enable interrupt requests generated by SMB0. אפשור פסיקת SMBus 0 (I^2C0)

Bit	7	6	5	4	3	2	1	0
Name			ET5	ET4	ESMB1		ES1	EVBUS
Type	R	R	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xE7; SFR Page = All Pages

Bit	Name	Function
7:6	Unused	Read = 00b, Write = Don't Care. ללא שימוש
5	ET5	**Enable Timer 5 Interrupt.** This bit sets the masking of the Timer 5 interrupt. 0: Disable Timer 5 interrupts. 1: Enable interrupt requests generated by the TF5L or TF5H flags. אפשור פסיקת טיימר 5
4	ET4	**Enable Timer 4 Interrupt.** This bit sets the masking of the Timer 4 interrupt. 0: Disable Timer 4interrupts. 1: Enable interrupt requests generated by the TF4L or TF4H flags. אפשור פסיקת טיימר 4
3	ESMB1	**Enable SMBus1 Interrupt.** This bit sets the masking of the SMB1 interrupt. 0: Disable all SMB1 interrupts. 1: Enable interrupt requests generated by SMB1. אפשור פסיקת SMBus 1 (I^2C 1)
2	Reserved	Must Write 0b. ביט רזרבה. יש לשים בו 0
1	ES1	**Enable UART1 Interrupt.** This bit sets the masking of the UART1 interrupt. 0: Disable UART1 interrupt. 1: Enable UART1 interrupt. אפשור פסיקת UART 1
0	EVBUS	**Enable VBUS Level Interrupt.** This bit sets the masking of the VBUS interrupt. 0: Disable all VBUS interrupts. 1: Enable interrupt requests generated by VBUS level sense. אפשור פסיקת VBUS

איור 6.ב רגיסטרים EIE1 משמאל ו EIE2 מימין.

באיור רשמנו לצד כל ביט את תפקידו בעברית. ב 2 רגיסטרים אלו נשתמש בצורה מעטה יחסית לרגיסטר ה IE .

דוגמאות : בדוגמאות הבאות הפקודה הראשונה בכל דוגמה היא באסמבלי. הדוגמה השנייה היא בשפת C51 בהנחה שרשמנו #include <REG51F380.h> .

דוגמא 1: רשום פקודה שתאפשר פסיקות חיצוניות בלבד.

```
Mov ie,#10000101b  ;  85h
IE=0x85;
```

דוגמא 2 : רשום פקודה שתאפשר פסיקה חיצונית 1 ופסיקת תקשורת טורית 0 :

```
Mov ie,#10010100b  ;  94h
IE=0x94;
```

דוגמא 3 : רשום פקודה לחסימת כל הפסיקות :

```
mov ie,#0   או   clr ea
EA=0;       or   IE=0;
```

6.4.2 רגיסטרים לעדיפות הפסיקות IP Interrupt Priority

בעזרת רגיסטר זה קובעים את סדר העדיפות בטיפול בפסיקות במידה והגיעו מספר פסיקות יחד. העדיפות אומרת מה סדר הטיפול בפסיקות , כלומר באיזו פסיקה המיקרו מטפל ראשונה, באיזו שנייה וכך הלאה. בנוסף לרגיסטר ה IP שהיה ב 51 המקורי יש כאן עוד 2 רגיסטרים של עדיפות הנקראים EIP1 ו EIP2 עבור העדיפות לשאר הפסיקות. **EIP -E**xtended **I**nterrupt **P**riority (הרחבת עדיפות פסיקות).

ישנן 2 רמות של עדיפות : גבוהה ונמוכה כאשר בביט ששמים 0 אומרים שעדיפות פסיקה זו נמוכה ובביט ששמים 1 קובעים שעדיפות הפסיקה – גבוהה. קיימת עדיפות "טבעית" אותה קבע היצרן. אם 2 פסיקות נמצאות באותה העדיפות אז העדיפות תיקבע לפי עדיפות "טבעית" שמתוארת בטבלה 6.ב בעמודה השלישית.

האיור הבא מתאר את **רגיסטר עדיפות הפסיקות IP** .

Bit	7	6	5	4	3	2	1	0
Name		PSPI0	PT2	PS0	PT1	PX1	PT0	PX0
Type	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	1	0	0	0	0	0	0	0

SFR Address = 0xB8; SFR Page = All Pages; Bit-Addressable

Bit	Name	Function
7	Unused	Read = 1b, Write = Don't Care.
6	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.
5	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 interrupt set to high priority level.
4	PS0	**UART0 Interrupt Priority Control.** This bit sets the priority of the UART0 interrupt. 0: UART0 interrupt set to low priority level. 1: UART0 interrupt set to high priority level.
3	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 interrupt set to high priority level.
2	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.
1	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.
0	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.

איור 6.ג : רגיסטר עדיפות הפסיקות IP

האות P בכל התחלת ביט אומרת Priority - עדיפות. סדר הביטים דומה לסדר הביטים ברגיסטר IE . משמאל פסיקה חיצונית 0 , הביט הבא הוא טיימר 0 וכך הלאה.

הביטים **PX0** ו **PX1** - עדיפות פסיקה חיצונית 0 ופסיקה חיצונית 1 בהתאמה.

הביטים **PT0** **PT1** ו **PT2** - עדיפות פסיקת טיימר 0 טיימר 1 וטיימר 2 בהתאמה.

הביט PS0 - עדיפות פסיקת תקשורת טורית מספר 0. הביט SPI0 – עדיפות פסיקה של ה SPI (ממשק טורי היקפי).

קיימת עדיפות "טבעית" במשפחת ה 51 , עדיפות שאותה קבע יצרן הג'וק. העדיפות היא מימין לשמאל בביטים. כלומר לפסיקה חיצונית 0 העדיפות הגבוהה ביותר , אחריה טיימר 0 אחריה פסיקה חיצונית 1 אחר כך פסיקה טיימר 1 וכך הלאה. בתוך העדיפות הזו המתכנת יכול לקבוע עדיפות משלו על ידי מתן עדיפות נמוכה (שמים 0) בביט המתאים או עדיפות גבוהה (שמים 1). אם למספר פסיקות המתכנת נתן אותה עדיפות אז סדר הטיפול יהיה לפי העדיפות "הטבעית".

נוסיף ונתאר את 2 רגיסטרי העדיפות הנוספים הנקראים EIP1 ו EIP2 הנראים באיור הבא :

Bit	7	6	5	4	3	2	1	0
Name	PT3	PCP1	PCP0	PPCA0	PADC0	PWADC0	PUSB0	PSMB0
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 = 0xF6; SFR Page = All Pages

Bit	Name	Function	
7	PT3	**Timer 3 Interrupt Priority Control.** This bit sets the priority of the Timer 3 interrupt. 0: Timer 3 interrupts set to low priority level. 1: Timer 3 interrupts set to high priority level.	עדיפות פסיקת טיימר 3
6	PCP1	**Comparator1 (CP1) Interrupt Priority Control.** This bit sets the priority of the CP1 interrupt. 0: CP1 interrupt set to low priority level. 1: CP1 interrupt set to high priority level.	עדיפות פסיקת משווה 1
5	PCP0	**Comparator0 (CP0) Interrupt Priority Control.** This bit sets the priority of the CP0 interrupt. 0: CP0 interrupt set to low priority level. 1: CP0 interrupt set to high priority level.	עדיפות פסיקת משווה 0
4	PPCA0	**Programmable Counter Array (PCA0) Interrupt Priority Control.** This bit sets the priority of the PCA0 interrupt. 0: PCA0 interrupt set to low priority level. 1: PCA0 interrupt set to high priority level.	עדיפות פסיקת מערך מונים מתוכנת 0
3	PADC0	**ADC0 Conversion Complete Interrupt Priority Control.** This bit sets the priority of the ADC0 Conversion Complete interrupt. 0: ADC0 Conversion Complete interrupt set to low priority level. 1: ADC0 Conversion Complete interrupt set to high priority level.	עדיפות פסיקת ADC0
2	PWADC0	**ADC0 Window Comparator Interrupt Priority Control.** This bit sets the priority of the ADC0 Window interrupt. 0: ADC0 Window interrupt set to low priority level. 1: ADC0 Window interrupt set to high priority level.	עדיפות פסיקת חלון השוואה
1	PUSB0	**USB (USB0) Interrupt Priority Control.** This bit sets the priority of the USB0 interrupt. 0: USB0 interrupt set to low priority level. 1: USB0 interrupt set to high priority level.	עדיפות פסיקת USB0
0	PSMB0	**SMBus0 Interrupt Priority Control.** This bit sets the priority of the SMB0 interrupt. 0: SMB0 interrupt set to low priority level. 1: SMB0 interrupt set to high priority level.	עדיפות פסיקת SMBus 0 (I^2C 0)

Bit	7	6	5	4	3	2	1	0
Name			ET5	ET4	ESMB1		ES1	EVBUS
Type	R	R	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

SFR Address = 0xE7; SFR Page = All Pages

Bit	Name	Function	
7:6	Unused	Read = 00b, Write = Don't Care.	ללא שימוש
5	ET5	**Enable Timer 5 Interrupt.** This bit sets the masking of the Timer 5 interrupt. 0: Disable Timer 5 interrupts. 1: Enable interrupt requests generated by the TF5L or TF5H flags.	עדיפות פסיקת טיימר 5
4	ET4	**Enable Timer 4 Interrupt.** This bit sets the masking of the Timer 4 interrupt. 0: Disable Timer 4interrupts. 1: Enable interrupt requests generated by the TF4L or TF4H flags.	עדיפות פסיקת טיימר 4
3	ESMB1	**Enable SMBus1 Interrupt.** This bit sets the masking of the SMB1 interrupt. 0: Disable all SMB1 interrupts. 1: Enable interrupt requests generated by SMB1.	עדיפות פסיקת SMBus 1 (I^2C 1)
2	Reserved	Must Write 0b.	רזרבה. יש לשים בו 0 .
1	ES1	**Enable UART1 Interrupt.** This bit sets the masking of the UART1 interrupt. 0: Disable UART1 interrupt. 1: Enable UART1 interrupt.	עדיפות פסיקת UART 1
0	EVBUS	**Enable VBUS Level Interrupt.** This bit sets the masking of the VBUS interrupt. 0: Disable all VBUS interrupts. 1: Enable interrupt requests generated by VBUS level sense.	עדיפות פסיקת VBUS

איור 6.ד : רגיסטרי הרחבת עדיפות הפסיקות EIE1 משמאל ו EIE2 מימין.

דוגמאות :

דוגמא 1 : רשום 2 פקודות שמאפשרות פסיקה טיימר 0 ופסיקת תקשורת טורית 0 אבל העדיפות תהיה של תקשורת טורית 0.

Mov ie,#10010010b ; אפשור פסיקת טיימר 0 ותקשורת טורית

Mov ip,#00010000b ; עדיפות גבוהה יותר לפסיקת תקשורת טורית על פסיקת טיימר 0

ובשפת C51 - בהנחה שרשמנו את שורת ההכללה ל REG51F380 – הפקודות הן :

IE=0x92;

IP=0x10;

דוגמה 2 : רשום פקודות לאפשור כל פסיקות הטיימרים ושסדר העדיפות יהיה שטיימר 1 עם עדיפות גבוהה משל שאר הטיימרים.

mov ie,#10101010b ; אפשור פסיקות טיימר 0, טיימר 1 וטיימר 2

mov eie1,#10000000b ; אפשור פסיקת טיימר 3

mov eie2,#01100000b ; אפשור פסיקות טיימר 4 ו טיימר 5

mov IP,#00001000b ; קביעת סדר עדיפות גבוהה לטיימר 1 ונמוכה לטיימר 0 וטיימר 2

אין צורך לקבוע עדיפות ל EIP1 ו EIP2 כי יש בהם 0 (עדיפות נמוכה) בפעולת ה RESET .

ובשפת C51 : IE=0xAA; EIE1=0x80; EIE2=0x60; IP=0x8 ;

שאלה : האם ניתן לקבוע סדר עדיפות רצוי עבור כל אפשרויות עדיפות פסיקות ?

תשובה : לא ! יש מוגבלות מבחינת האפשרויות . לדוגמא סדר עדיפות שלא נצליח לממש:

תרגיל : יש לאפשר 3 פסיקות. פסיקה חיצונית 0 ו 1 ופסיקת טיימר 1 . רוצים שסדר העדיפות יהיה : א. טיימר 1 בעדיפות הגבוהה ביותר ב. פסיקה חיצונית 1 בעדיפות הבאה ג. בעדיפות הנמוכה פסיקה חיצונית 0 .

mov ie,#10001101b ; אפשור פסיקות חיצוניות 0 ו 1 וטיימר 1

mov ip,# ????????

נסו ותיווכחו שלא ניתן לממש סדר עדיפות כזה.

6.4.3 פסיקות חיצוניות ורגיסטר TCON (בקרת טיימרים - Timers CONtrol)

למרות ששם הרגיסטר איננו קשור לפסיקות , אלא לבקרת הטיימרים, 4 הסיביות "הנמוכות" ברגיסטר ה TCON קשורות למערכת הפסיקות החיצוניות. את מבנה הרגיסטר ניתן לראות באיור הבא :

Bit	7	6	5	4	3	2	1	0
Name	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
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 = 0x88; SFR Page = All Pages; Bit-Addressable

Bit	Name	Function
7	TF1	**Timer 1 Overflow Flag.** Set to 1 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.
6	TR1	**Timer 1 Run Control.** Timer 1 is enabled by setting this bit to 1.
5	TF0	**Timer 0 Overflow Flag.** Set to 1 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.
4	TR0	**Timer 0 Run Control.** Timer 0 is enabled by setting this bit to 1.
3	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 in edge-triggered mode.
2	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 16.7). 0: $\overline{INT1}$ is level triggered. 1: $\overline{INT1}$ is edge triggered.
1	IE0	**External Interrupt 0.** 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 0 service routine in edge-triggered mode.
0	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 16.7). 0: $\overline{INT0}$ is level triggered. 1: $\overline{INT0}$ is edge triggered.

איור 6.ב - רגיסטר ה TCON

את 4 הביטים הגבוהים השייכם לטיימרים נסביר בפרק על הטיימרים. כאן נסביר את תפקיד 4 הביטים הנמוכים.

ב 8051 המקורי קבעו 4 ביטים נמוכים אלו 2 אפשרויות של אופני עבודה והם נקראים **אופן הדרבון**:

א. עבודה על רמה -**LEVEL** - של מתח , הווה אומר שרכיב שמבקש פסיקה צריך לשים 0 בהדק הפסיקה החיצונית ובמצב 1 אין בקשת פסיקה (כלומר פעיל בנמוך – Active Low) . 0 היה 0 וולט ו 1 היה 5 וולט.

ב. עבודה על שפה (או קצה) – **EDGE** – הפסיקה הייתה מתקבלת במעבר מ 1 ל 0 כלומר בירידת המתח מ 5 וולט ל 0 וולט .

במיקרו C8051F380 יש ל 4 הביטים את אותו התפקיד של בחירת אופני עבודה של רמה או שפה אבל הוסיפו רגיסטר נוסף הנקרא **IT01CF** שבעזרתו ניתן לקבוע שבמידה ונבחר אופן עבודה LEVEL – רמה - ברגיסטר ה TCON ניתן לקבוע האם בקשת הפסיקה תהיה ברמה של 0 (0 וולט) או ברמה של 1 (רמה של 1 היא 3.3 וולט שהוא מתח הספק של המיקרו C8051F380). כמו כן אם נבחר אופן עבודה של EDGE – שפה – ברגיסטר ה TCON אז נוכל לקבוע האם הפסיקה תתקבל על ידי מעבר של רמה מ 0 ל 1 או מ 1 ל 0 .

נסביר את 4 הדקי ה TCON הנמוכים.

ביט IT0 - בעזרת ביט זה המתכנת קובע האם הפסיקה החיצונית מספר 0 המגיעה בהדק $\overline{INT0}$, תעבוד על רמה LEVEL או על שפה EDGE . אם המתכנת שם בביט 0 אז הפסיקה עובדת על רמה. אם המתכנת שם בביט 1 אז עובדים על שפה/קצה.

בעבודה עם רמה LEVEL (המתכנת שם 0 בביט IT0) , הכוונה היא שהמערכת שמבקשת טיפול חייבת להחזיק את ההדק החיצוני - **INT0** ב 0 עד שהמיקרו נענה ועובר לטפל בבקשה, אחרת היא לא תקבל טיפול. לדוגמא : נניח שבקטע תוכנית כלשהו, המיקרו חוסם את כל הפסיקות. מערכת המתחברת להדק הפסיקה INT0 ביקשה פסיקה על ידי הורדת ההדק ל 0 . המערכת החזירה את המתח בהדק הפסיקה ל 1 לפני שקיבלה טיפול. כאשר המיקרו יתפנה לטפל בפסיקה הוא "רואה" שאין בקשת פסיקה בהדק ולכן הוא לא יתפנה לטפל בפסיקה.

בעבודה עם שפה EDGE המתכנת שם בביט IT0=1 . שוב נניח שבקטע תוכנית מסוים המיקרו חסם את כל בקשות הפסיקה. המערכת שמבקשת פסיקה מורידה את הדק בקשת הפסיקה החיצונית **$\overline{INT0}$** ל 0 ויכולה לחזור ל 1. בתוך מערכת הפסיקות קיים דלג-לג (F.F – Flip Flop) ש"זוכר" את בקשת הפסיקה. כאשר המיקרו יתפנה לקבל פסיקות הוא ייראה שיש בקשת פסיקה (בגלל ה FF) ויטפל בפסיקה.

ביט **IE0** - הביט עולה ל 1 על ידי החומרה כאשר יש בקשת פסיקה בהדק **$\overline{INT0}$** על ידי שינוי רמת המתח בהדק הפסיקה מ 1 ל 0 (או מ 0 ל 1 – תלוי במה שהמשתמש קבע ברגיסטר **IT01CF**) . הביט מוחזר ל 0 בסיום תוכנית הטיפול בפסיקה על ידי החומרה. אחד השימושים של הביט יכול להיות עבור המנהל העובד בשיטת השאילתות ולא הפסיקות. המיקרו בודק את מצב הביט. אם יש בו 1 הוא יודע שהייתה בקשת פסיקה ומטפל בה.

הביטים **IT1** ו **IE1** - כמו הביטים IT0 ו IE0 אבל לגבי פסיקה חיצונית 1 .

6.4.4 – רגיסטר IT01CF – לקביעת קוטביות המתח בפסיקות חיצוניות.

האיור הבא מתאר את רגיסטר **IT01CF** שהוא קיצור של **C**on**F**iguration **IT1** / $\overline{INT1}$ **$\overline{INT0}$** – תצורת הפסיקות.

Bit	7	6	5	4	3	2	1	0
Name	IN1PL	IN1SL[2:0]			IN0PL	IN0SL[2:0]		
Type	R/W	R/W			R/W	R/W		
Reset	0	0	0	0	0	0	0	1

SFR Address = 0xE4; SFR Page = 0

Bit	Name	Function
7	IN1PL	**$\overline{\text{INT1}}$ Polarity.** 0: $\overline{\text{INT1}}$ input is active low. 1: $\overline{\text{INT1}}$ input is active high. קוטביות המתח בהדק $\overline{\text{INT1}}$: אם שמנו ברגיסטר ה TCON בביט IT1=0 - עבודה עם רמה – LEVEL - אז אם נשים IN1PL=0 אז נקבל פסיקה ברמת של 0. אם נשים IN1PL=1 נקבל פסיקה ברמה של 1 . אם נשים IT1=1 - EDGE - שפה אז במצב 0 -ירידה, ב 1 – עליה.
6:4	IN1SL[2:0]	**$\overline{\text{INT1}}$ 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. 000: Select P0.0 001: Select P0.1 010: Select P0.2 011: Select P0.3 100: Select P0.4 101: Select P0.5 110: Select P0.6 111: Select P0.7 3 ביטים הקובעים איזה הדק של פורט 0 מוקצה להדק $\overline{\text{INT1}}$. הקצאת הדק זו איננה תלויה בקרוסבר . ההדק יבדוק מה מצב ההדק ללא הפרעה לרכיב הפריפריאלי שהוקצה להדק על ידי הקרוסבר. הקרוסבר לא יקצה את ההדק לרכיב פריפריאלי אם הוא הוגדר לדילוג -SKIP .
3	IN0PL	**$\overline{\text{INT0}}$ Polarity.** 0: $\overline{\text{INT0}}$ input is active low. 1: $\overline{\text{INT0}}$ input is active high. קוטביות המתח בהדק $\overline{\text{INT0}}$: אם שמנו ברגיסטר ה TCON בביט IT0=0 - עבודה עם רמה – LEVEL - אז אם נשים IN0PL=0 אז נקבל פסיקה ברמת של 0. אם נשים IN0PL=1 נקבל פסיקה ברמה של 1 . אם נשים IT0=1 - EDGE - שפה אז במצב 0 -ירידה, ב 1 – עליה.
2:0	IN0SL[2:0]	**$\overline{\text{INT0}}$ 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. 000: Select P0.0 001: Select P0.1 010: Select P0.2 011: Select P0.3 100: Select P0.4 101: Select P0.5 110: Select P0.6 111: Select P0.7 3 ביטים הקובעים איזה הדק של פורט 0 מוקצה להדק $\overline{\text{INT0}}$. הקצאת הדק זו איננה תלויה בקרוסבר . ההדק יבדוק מה מצב ההדק ללא הפרעה לרכיב הפריפריאלי שהוקצה להדק על ידי הקרוסבר. הקרוסבר לא יקצה את ההדק לרכיב פריפריאלי אם הוא הוגדר לדילוג -SKIP .

איור 6.ג רגיסטר IT01CF - תצורת הפסיקות החיצוניות.

הרגיסטר קובע את קוטביות המתח עבור בקשות הפסיקה החיצוניות ואיזה הדק של פורט 0 של הרכיב הוקצה לבקשת הפסיקה.

בעזרת הטבלה הבאה יהיה קל להבין את הקשר בין הביטים ברגיסטר ה TCON ובין הביטים ברגיסטר כאן IT01CF :

IT1	IN1PL	$\overline{\text{INT1}}$ Interrupt	ביט IT1 ברגיסטר ב TCON שקובע רמה (0) או EDGE (1) . ביט IN1P ברגיסטר IT01CF קובע POLARITY - קוטביות.
1	0	Active low, edge sensitive	עבודה ב EDGE - שפה, קצה . פסיקה במעבר מ 1 ל 0
1	1	Active high, edge sensitive	עבודה ב EDGE - שפה, קצה . פסיקה במעבר מ 0 ל 1
0	0	Active low, level sensitive	עבודה ב LEVEL - רמה. פסיקה ב '0'
0	1	Active high, level sensitive	עבודה ב LEVEL - רמה. פסיקה ב '1' .

טבלה 6.ג : התצורות האפשריות עבור פסיקה חיצונית 1 בהדק INT1 .

הטבלה מתארת תצורות אפשריות עבור פסיקה חיצונית 1 . טבלה דומה קיימת גם עבור תצורות אפשריות של פסיקה חיצונית 0 .

לכל פסיקה יש וקטור פסיקה – כתובת אליה עוברת התוכנית – כאשר המיקרו מטפל בפסיקה. בטבלה הבאה ריכזנו רק את הפסיקות המתאימות לפסיקות חיצוניות, לפסיקות טיימרים 0 ו 1 ולתקשורת טורית UART. רואים את הקשר בין מקור הפסיקה, איזה ביטים מבקשים את הפסיקה, האם הבקשה "נמחקת" כאשר נענים לפסיקה ולאיזו כתובת פונים כאשר נענים לפסיקה.

מקור הפסיקה | שם הביט שמציין שיש פסיקה | האם הביט מתאפס על ידי החומרה בסיום תוכנית הפסיקה? | כתובת תכנית הפסיקה

Interrupt Source	Interrupt Request Bits	Cleared by Hardware	Vector Address
$\overline{INT0}$	IE0	No (level) Yes (trans.)	0003H
Timer 0	TF0	Yes	000BH
$\overline{INT1}$	IE1	No (level) Yes (trans.)	0013H
Timer 1	TF1	Yes	001BH
Serial Port	RI, TI	No	0023H
System Reset	RST		0000H

טבלה 6.ד : טבלת ווקטור הפסיקות עבור הפסיקות ברגיסטר IE

בטבלה קיימות 4 עמודות ו 6 שורות. כל שורה מתארת פסיקה כלשהי. בשורה האחרונה רואים שפעולת האיפוס – RESET - נחשבת כפסיקה וכאשר היא מתקבלת עוברים לכתובת 0 , כלומר לכתובת הראשונה בזיכרון התוכנית ומכאן התוכנית מתחילה. העמודה השמאלית מראה מהו מקור הפסיקה. העמודה מימינה מראה אילו ביטים מראים שיש בקשת פסיקה. העמודה הבאה אומרת האם בסיום הטיפול בפסיקה הביט מתאפס על ידי החומרה. העמודה הימנית מראה באיזו כתובת נמצאת התוכנית המטפלת בפסיקה.

מהטבלה ניתן לראות שכאשר יש בקשת פסיקה חיצונית 0 , עוברים לכתובת 3 ומטפלים בפסיקה. אם יש בקשת פסיקה חיצונית 1 המיקרו יקפוץ לכתובת 13H . בכתובת זו נרשום תוכנית המטפלת במערכת המתחברת לפסיקה חיצונית 1 . היות ובין פסיקה לפסיקה יש רק 8 כתובות, נהוג לרשום פקודת קפיצה לכתובת כלשהי בזיכרון התוכנית כדי לא "לדרוס" את הפסיקה הבאה.

6.5 תהליך הענות לפסיקה, הכנסת כתובת החזרה למחסנית, מעבר לתוכנית טיפול וחזרה על ידי שליפת כתובת החזרה

תהליך הענות לפסיקה נקרא ISR – In Service Routine שהתרגום שלו הוא שגרת שרות.

ניתן דוגמא מספרית המתארת את תהליך ההיענות לפסיקה.

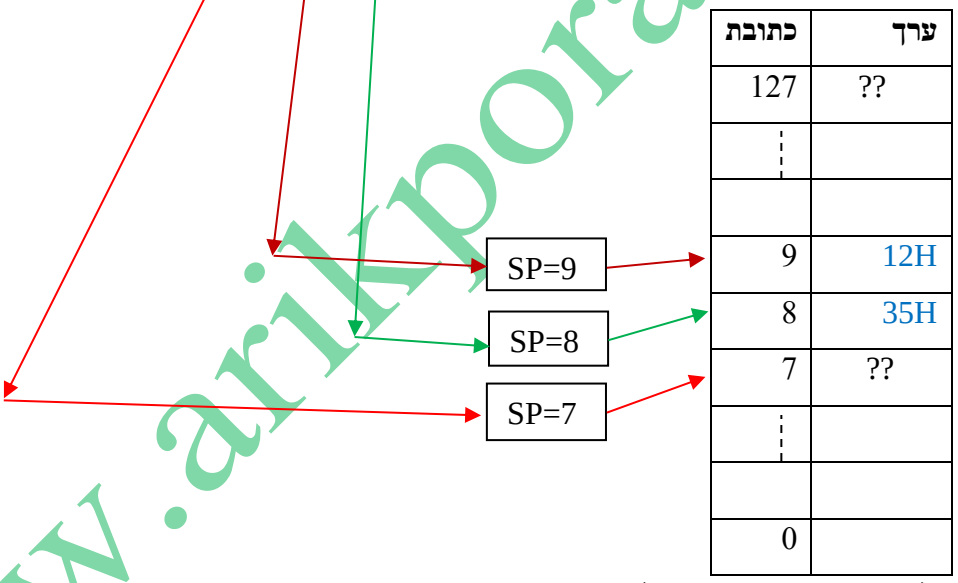
א. נניח שהמיקרו מבצע את הפקודה שבכתובת 1234H . במונה התוכנית שלו יש את הכתובת 1235H שהיא הכתובת של הפקודה הבאה אותה הוא אמור לבצע.

ב. בזמן ביצוע הפקודה הגיעה בקשת פסיקה חיצונית בהדק $\overline{INT1}$. רכיב המתחבר להדק זה הוריד את ההדק ל 0 והעלה את ההדק חזרה ל 1 (כך קבענו בעזרת הרגיסטרים TCON ו IT0CF) .

ג. בהנחה שהמשתמש איפשר פסיקה חיצונית 1 (בעזרת הפקודה mov ie,#84H) וגם בביט IT1 שברגיסטר ה TCON הוא שם 1 (בעזרת הפקודה setb it1), אז המיקרו מסיים לבצע את הפקודה הנוכחית, ומכניס למחסנית את כתובת החזרה – 1235H . כתובת זו מורכבת מ 8 ביט נמוכים (35H) ו 8 ביט גבוהים (23H) . **תהליך ההכנסה** הוא כך :

ד. בהנחה שבמצביע המחסנית יש את הערך 7 , כלומר sp=7 מתבצע התהליך הבא:

1. המיקרו מגדיל את SP ב 1 , כלומר sp=sp+1=7+1=8 .
2. לכתובת זו הוא מכניס את החלק הנמוך של כתובת החזרה, כלומר בכתובת 8 ב RAM הנתונים הפנימי יש 35H .
3. המיקרו מגדיל שוב את SP ב 1 , כלומר sp=sp+1=8+1=9 .
4. לכתובת זו נכנס החלק הגבוה של כתובת החזרה. כלומר בכתובת 9 יהיה הנתון 12H .



טבלה 6.ה : תהליך הכנסת כתובת החזרה למחסנית.

ה. עכשיו המיקרו יכול לטפל במבקש הפסיקה. היות והפסיקה היא פסיקה חיצונית 1 מספר 1 אז מטבלת ווקטור הפסיקות רואים שכתובת הפסיקה היא 0013H . המיקרו טוען את מונה התוכנית בערך זה ואז התוכנית עוברת לכתובת זו. כאן רושמים את הכתובת המטפלת בפסיקה. בסיום תכנית הטיפול רשומה הפקודה ret שהיא קיצור של return (חזור).

ו. כאשר המעבד הגיע לפקודת ה reti (חזור מפסיקה), הוא צריך לשלוף את כתובת החזרה מהמחסנית וכך יוכל חזור לכתובת בתוכנית .

ז. **תהליך השליפה של כתובת החזרה** מתבצע כך :

1. מהכתובת עליה מצביע ה SP מעבירים את הנתון לחלק הגבוה של מונה התוכנית. היות ו SP שווה 9 ובכתובת זו נמצא 12H אז החלק הגבוה של מונה התוכנית שווה 12H .

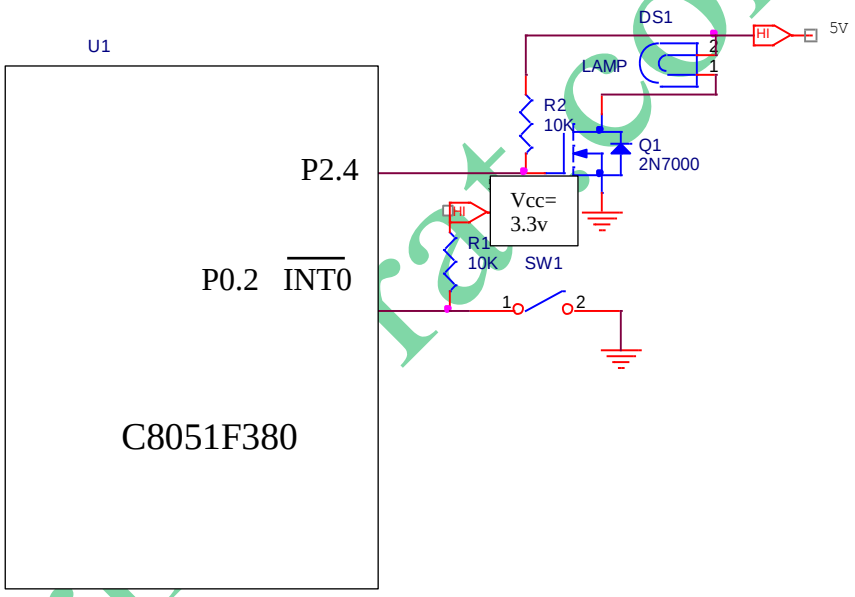
2. המיקרו מחסר 1 מ SP , כלומר SP=SP+1=9-1=8
3. מכתובת זו המיקרו מעביר את הנתון (35H) לחלק הנמוך של מונה התוכנית.
4. המיקרו מחסר 1 מ SP כלומר SP=SP-1=8-1=7 .

ח. במונה התוכנית יש את הערך 1235H . התוכנית חוזרת לפקודה ממנה יצאנו לטפל בפסיקה.

6.6 – דוגמאות

הערה : בחלק מהדוגמאות לא צוינו הגדרות האתחול של ההדקים וחבור הקרוסבר אלא רק את פעולת התוכנית הרצויה.

דוגמה 1 : רשום תוכנית המדמה הפעלת נורה בחדר מדרגות. נתון המעגל הבא שבו רואים רק החיבורים הנדרשים לתוכנית כאן :



איור 6.ד : פסיקה להפעלת נורה לזמן מסוים.

במעגל המשורטט רואים :

1. מיקרו בקר ממשפחת ה 51 .
2. מפסק המתחבר להדק P0.2 שישמש כהדק $\overline{INT0}$ של המיקרו בקר.
3. נגד R1 של 10K - נגד משיכה למעלה - Pull Up - דואג שכאשר המפסק פתוח יש 1 בהדק הפסיקה . לחיצה על המפסק תעביר את האדמה מצד ימין של המפסק לצד שמאל ותיתן 0 בהדק כניסת פסיקה חיצונית 0 ועל ידי כך מתקבלת בקשת פסיקה.
4. נורה המאירה כאשר זורם דרכה זרם. הנורה להספק של 1 וואט , כלומר ב 5 וולט זורם דרכה 200 מילי אמפר. זרם כזה המיקרו לא יכול לספק.
5. טרנזיסטור FET המשמש כמתג. כאשר הוא מקבל בשער שלו 0 הוא בקטעון והוא כמו מתג פתוח שאיננו מזרים זרם. במצב שניתן 1 בשער שלו הוא ברוויה (או ליתר דיוק באזור האוהמי שלו) ואז הוא כמו מתג סגור המזרים דרכו זרם.

שמנו טרנזיסטור FET בגלל שהנורה צורכת זרם של 200 מילי אמפר שאותו המיקרו בקר לא יכול לתת.

דוגמה 1 :

יש לרשום תכנית בשפת סף שתדאג להדליק את הנורה לפרק זמן של חצי דקה ולאחר מכן לכבות אותה עד הלחיצה הבאה. הדבר דומה ללחיצה על מפסק בחדר מדרגות שבעזרתה רוצים להדליק נורה לפרק זמן של כחצי דקה ואז היא נכבית. הפעלת המנורה תהיה על ידי הוצאת 1 להדק P2.4 . טרנזיסטור ה FET ייכנס לרוויה (יעבוד בחלק האוהמי של האופיין שלו) , יזרום זרם דרך הנורה ודרך ה FET והנורה תאיר. 0 בהדק P2.4 יגרום לכך שה FET בקטעון והנורה כבויה.

פתרון :

בתחילת התוכנית נכבה את הנורית ואז נפעיל את הרגיסטרים של הפסיקה :

א. בעזרת רגיסטר IE המאפשר פסיקות נאפשר פסיקה חיצונית 0 ונשים גם בביט EA=1 .

ב. נקבע בעזרת ביט IT0 ברגיסטר ה TCON עבודה עם הפסיקה על EDGE - שפה (מעבר בין רמות מתח) .

ג. בעזרת רגיסטר IT01CF נקבע שהמעבר של הרמות יהיה מ 1 ל 0 ושהדק פסיקה חיצונית 0 יהיה הדק P0.2 של המיקרו.

התוכנית :

```
      org  0          ;  0 - הכתובת ההתחלתית של התוכנית
   ljmp main          ;   main  קפוץ אל התווית

      org  3          ;  מתחילים לכתוב את התוכנית החל מכתובת 3 המתאימה לפסיקה חיצונית 0
    setb P2.4         ;  שים 1 בהדק והכנס את הטרנזיסטור לרוויה
   ; קריאה לפרוצדורה שתבצע השהייה לחצי דקה
    clr P2.4
    reti

    org 100H
main:
    clr P2.4   ;  כבוי הנורה
    mov ie,#81H   ;  לאפשור פסיקה חיצונית 0
    setb IT0  ;  EDGE  הפעלת הפסיקה על
    mov IT01CF,#2  ; INT0  כהדק פסיקה  P0.2  הקצאת הדק . 0 ל 1 מ ירידה על עבודה. IN0PL=0,IN0SL=2
    sjmp $   ;  חזור לפקודה הנוכחית
```

הפקודה Org 0 אומרת לקומפיילר להציב את כל הפקודות שנמצאות אחרי שורה זו החל מכתובת 0 והלאה.

בכתובת 3 רשומה תכנית השרות של פסיקה חיצונית 0 .לכאן נגיע אם המשתמש יילחץ על המפסק.

בכתובת 100H רשומה הפקודה מכבה את הנורה.

השורה הבאה מאפשרת קבלת פסיקה חיצונית 0 .

השורה הבאה אומרת שהפסיקה תפעל על EDGE – שפה ,קצה, כלומר על מעבר מרמה לוגית אחת לאחרת.

השורה הבאה אומרת שהמעבר יהיה מרמה לוגית גבוהה (1) לרמה נמוכה (0) כלומר על ירידה מ 1 ל 0 .

השורה האחרונה מכניסה את התוכנית ללולאה אין סופית סביב הפקודה עצמה. סימן ה $ בפקודה אומר לקפוץ לשורה שבה נמצאים. זה כמו הפקודה : again: sjmp again .

מהלולאה הזו יוצאים רק כאשר מישהו לוחץ על המפסק. הלחיצה גורמת לירידה מ 1 ל 0 בהדק הפסיקה החיצונית מספר 0 . המיקרו מסיים לבצע את הפקודה sjmp וחוזר לכתובת של אותה השורה. הוא מכניס את הכתובת של הלולאה (כתובת החזרה) למחסנית (כדי לדעת לאיזו כתובת לחזור) ורק אז עובר לכתובת 3 שהיא וקטור הפסיקה של פסיקה חיצונית . המיקרו מבצע את התוכנית הרשומה שם. הוא מדליק את הנורה על ידי הוצאת 1 בהדק P2.4 . לאחר מכן עוברים לפרוצדורה שמבצעת השהייה של חצי דקה (תוכנית ההשהיה לא מוצגת כאן..) וכאשר חוזרים מתוכנית ההשהיה מכבים את הנורה. פקודת ה reti מחזירה את התוכנית לפקודה sjmp $. שוב ניכנס כאן ללולאה עד הלחיצה הבאה על המפסק שבה נדליק שוב את הנורה וחוזר חלילה.

הערה : כדאי לשים לב שאם בביט IT0 היינו שמים 0 , כלומר היינו עובדים על רמה – Level - אז במידה והמפסק היה נלחץ קבוע (הכנסת קיסם להחזקת המפסק לחוץ באופן קבוע, כנהוג במחוזותינו...), אז הנורית הייתה נדלקת לחצי דקה, בפקודה clr p2.4 היא נכבית וכאשר חוזרים לפקודה sjmp $ רואים שיש 0 בהדק הפסיקה ושוב קופצים לתוכנית הפסיקה . הנורה הייתה נכבית לזמן קצר ביותר , שוב הייתה נדלקת לחתי דקה ושוב נכבית לזמן קצר מאוד ושוב נדלקת. באופן מעשי העין הייתה רואה שהנורה דולקת כל הזמן כי בגלל מהירות המיקרו העין לא מצליחה להבחין בכיבוי הנורה . בעבודה עם EDGE שפה-קצה - ירידה מ 1 ל 0 - הנורה נדלקת פעם אחת בלבד ונכבית. כדי להדליק את המנורה שוב היה צריך להוציא את הקיסם , המפסק ייפתח וההדק יקבל 1 ואז אם נלחץ – נקבל פסיקה כי יש ירידה מ 1 ל 0 .

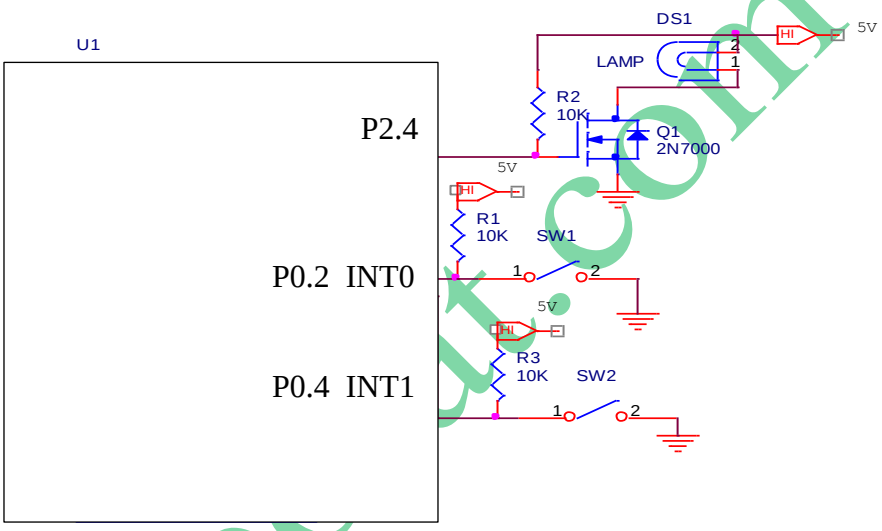
נרשום את אותה התוכנית בשפת C51 :

```
#include <REG51F380>
void myExt0Int ( ) interrupt 0          // פונקציית הפסיקה . שם הפונקציה myExt0Int. זוהי פסיקה מספר 0 .
{
        P0^2=1;
        //  זימון פונקציית השהייה לחצי דקה
        P0^2=0;
}
void main( )
{
    P2^4=0;      //    כבוי הנורה
    IE=0x81;    //  אפשור פסיקה חיצונית 0
    IT0=1;       // EDEGE  עבודה עם פסיקה חיצונית 0  על
    IT01CF=2; //  עבודה על ירידה מרמת מתח גבוהה לנמוכה . הדק P2.0 הוא הדק פסיקה חיצונית 0
    while(1); //לולאה אין סופית. יציאה מהלולאה כאשר יש לחיצה על המפסק ומקבלים פסיקה ואז עוברים לפונקציית הפסיקה
}
```

דוגמא 2 : הפעלה וכיבוי נורה בעזרת 2 מפסקים.

במעגל הבא 2 מפסקים. כל לחיצה על מפסק אמורה להחליף את מצב הנורה. אם היא דלקה - יש לכבות אותה ולהפך. לא משנה מאיזה מפסק הגיעה הפקודה. במעגל שורטטו הרכיבים הנחוצים לתרגיל בלבד.

כאן נאפשר את פסיקה חיצונית 0 ו 1 . כל לחיצה תהפוך את מצב הנורה.



איור 6.ה : הפעלה וכיבוי נורה בעזרת 2 מפסקים.

פתרון :

```
    Org   0 ;   0 - הכתובת ההתחלתית של התוכנית
ljmp main1 ;    main קפוץ אל התווית

    Org  3          ;  0 הפקודות הבאות נכתבות החל מכתובת 3 המתאימה לפסיקה חיצונית
 cpl P2.4   ;  הפוך את מצב הביט. אם היה בו 0 הפוך ל 1 ולהפך. בצורה כזו משתנה מצב הנורה
    ; Debouncing - השהייה כדי להתגבר על ניתור המגעות
 RETI

org 13h   ; 1 הפקודות הבאות נכתבות החל מכתובת 13 הקסה המתאימה לפסיקה חיצונית
cpl P2.4   ;  הפוך את מצב הביט. אם היה בו 0 הפוך ל 1 ולהפך. בצורה כזו משתנה מצב הנורה
    ; Debouncing - השהייה כדי להתגבר על ניתור המגעות
 RETI
    Org 100H
main1:
    clr P2.4     ;  כבה את הנורה
```

```
mov ie,#85H   ;   אפשור פסיקה חיצונית 0 ופסיקה חיצונית 1
setb IT0  ;  EDGE  הפעלת פסיקה חיצונית 0 על
setb  IT1  ;   EDGE הפעלת פסיקה חיצונית 0 על
mov IT01CF,#42H  ; הפעלת 2 הפסיקות על ירידות ( מעבר מרמה גבוהה לנמוכה). פסיקה חיצונית 0 מוקצית ל P0.2  ופסיקה חיצונית 1 להדק P2.4
sjmp $  ;   חזור לפקודה הנוכחית
```

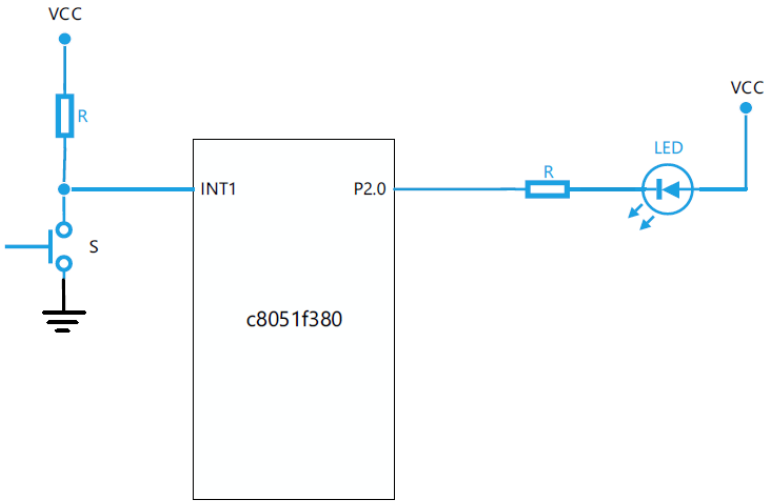
השינוי בתוכנית מזו של קודם הוא שאפשרנו גם פסיקה חיצונית 1 וגם אותה הפעלנו על ירידות. 2 תוכניות הפסיקה זהות. בכל תכנית אנו הופכים את מצב הדק P2.4 ועל ידי כך הופכים את מצב מתג הטרנזיסטור ומכאן שאת מצב הנורה. בכל תכנית פסיקה עשינו תכנית השהייה המתגברת על מצב של ניתור מגעות - debouncing - של המפסק. על ניתור מגעות נדבר בהמשך הספר.

נכתוב את התוכנית גם בשפת C51 :

```
#include <REG51F380>
void myExt0Int ( ) interrupt 0          // . 0 מספר הפסיקה הוא .myExt0Int פונקציית הפסיקה . שם הפונקציה
{
        P0^2=1;
        //  זימון פונקציית השהייה לחצי דקה
        P0^2=0;
}
void myExt1Int ( ) interrupt 2          // . 2 מספר הפסיקה הוא .myExt1Int פונקציית הפסיקה . שם הפונקציה
{
        P0^2=1;
        //  זימון פונקציית השהייה לחצי דקה
        P0^2=0;
}
void main( )
{
   P2^4=0;     //    כבוי הנורה
   IE=0x85;    //  אפשור פסיקה חיצונית 0
   IT0=1;      // EDEGE  עבודה עם פסיקה חיצונית 0 על
   IT1=1;      // EDEGE  עבודה עם פסיקה חיצונית 1 על
   IT01CF=0x42;      //עבודה על ירידה מרמת מתח גבוהה לנמוכה . הדק P2.0 הוא הדק פסיקה חיצונית 0 והדק P0.4 הוא הדק פסיקה חיצונית 1
   while(1); //לולאה אין סופית. יציאה מהלולאה כאשר יש לחיצה על המפסק ומקבלים פסיקה ואז עוברים לפונקציית הפסיקה
}
```

דוגמה 3 :

נתונה המערכת הבאה הכוללת מיקרו-בקר C8051F380, נורת LED המחוברת להדק P2.0 ולחצן ללא ריטוטים המחובר להדק הפסיקה .



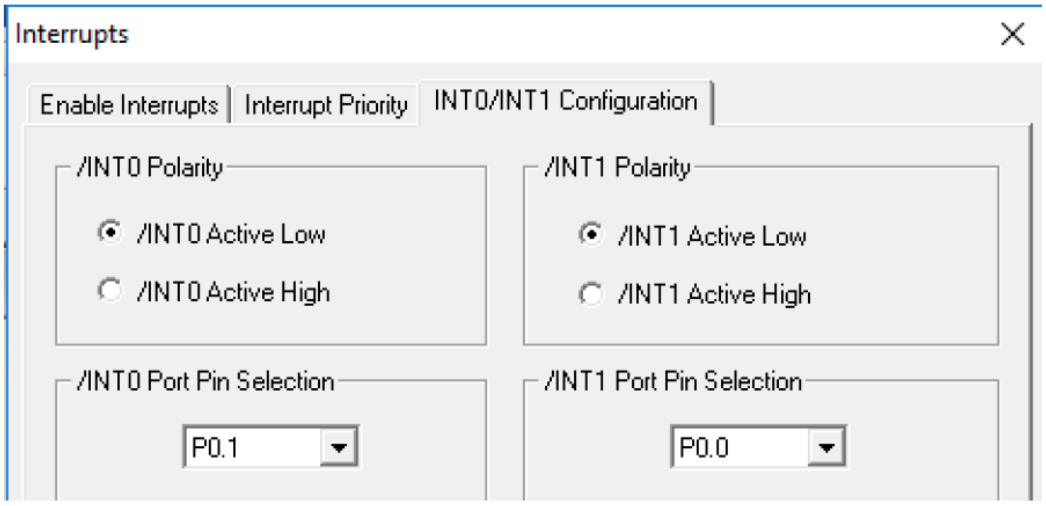
לפניכם חלק מפונקציית main ופונקציית הפסיקה :

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

void Init_Device(void);

sbit P2_0 = P2 ^ 0;

void main(void)
{
      Init_Device();
        P2_0 = 1;
      IE = 0x84;
      TCON = 0x04;

      while (1)
      {
      }
}

void int1() interrupt 2
{
}
```

א. בפונקציית main , הסבירו את שתי מילות הבקרה

```
IE = 0x84;
TCON = 0x04;
```

ב. נתון חלון הדקי הפסיקות:



כתבו לאיזה הדק חובר הלחצן, האם הפסיקה מתבצעת בזמן לחיצה או שחרור הלחצן.

ג. כתוב את תוכנית הפסיקה, אשר תגרום ללד לדלוק ולאחר שנייה להיכבות, עבור 5 הפסיקות הראשונות ועבור - 5 הפסיקות הבאות לדלוק למשך 2 שניות. לאחר 10 פסיקות תיחסם הפסיקה.

לצורך יצירת השהיות בין הדלקות נתונה פונקציית השהייה במילי שניות :

```
delay_ms(int ms)
```

פתרון דוגמה 3 :

א. כדי לאפשר את פסיקה חיצונית מספר 1 יש לכתוב לרגיסטר IE
זהו מבנה הרגיסטר IE:

Bit	7	6	5	4	3	2	1	0
Name	EA	ESPI0	ET2	ES0	ET1	EX1	ET0	EX0
Value	1	0	0	0	0	1	0	0

מהרגיסטר ניתן לראות שסיבית אפשור קבלת הפסיקות פעילה EA וסיבית אפשור פסיקה חיצונית 1 גם פעילה.

רגיסטר TCON מאפשר את אופן דרבון הפסיקה זהו מבנה הרגיסטר:

Bit	7	6	5	4	3	2	1	0
Name	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
value	0	0	0	0	0	1	0	0

מהרגיסטר ניתן לראות שערך סיבית IT1 '1' לוגי כך שהפסיקה תהיה פעילה בדרבון קצה.

ב. הלחצן מחובר להדק P0.0 (int1) והפסיקה מתבצעת בזמן הלחיצה(Active Low) ניתן לראות זאת באמצעות רגיסטר IT01CF.

זהו מבנה הרגיסטר:

Bit	7	6	5	4	3	2	1	0
Name	IN1PL	IN1SL[2:0]			IN0PL	IN0SL[2:0]		
value	0	0	0	1	0	0	0	0

ג.

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

void Init_Device(void);
void delay_ms(int ms);

sbit P2_0 = P2 ^ 0;

int cnt=0;

void main(void){
      Init_Device();
      P2_0 = 1;
      IE = 0x84;
      TCON = 0x04;

      while (1);
}


void int1() interrupt 2 {
      if(cnt<5)
      {
            cnt++;
            P2_0=0;
            delay_ms(1000);
            P2_0=1;
      }
      else if(cnt<10)
      {
            cnt++;
            P2_0=0;
            delay_ms(2000);
            P2_0=1;
      }
      else
            IE = 0x80;
}
```

פרק 7 : טיימרים/קאונטרים

7.1 – מהו טיימר/קאונטר ?

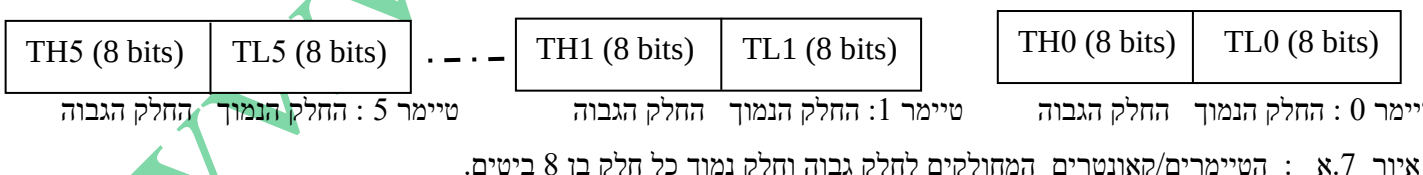
טיימר/קאונטר היא מערכת ספירה. היא מורכבת מ 2 המילים טיימר (בעברית - זמנן) וקאונטר (בעברית - מונה) . המערכת מקבלת בכניסה שלה פולסי ספירה וכל פולס הנספר מגדיל את ערך המונה ב 1 – מונה מעלה Up Counter. כאשר כל התאים של המונה מתמלאים ב '1' ומגיע פולס ספירה נוסף המערכת מתאפסת ומתחילה ספירה חדשה. הדבר דומה לספידומטר של מכונית. בספידומטר הספירה עשרונית ואילו בטיימר/קאונטר היא בינארית. בספידומטר עם 6 ספרות , כאשר מגיעים לערך 999999 ק"מ ונוסעים עוד 1 קילומטר נוסף הספידומטר יראה 000000 ק"מ. בטיימר/קאונטר כאשר הוא מגיע לערך 1111111111111111 (16 ביטים של '1') ומגיע פולס ספירה נוסף נקבל 0000000000000000 (16 ביטים של 0) והטיימר/קאונטר "מודיע" למיקרו בקר שהוא סיים ספירה. ההודעה היא בעזרת בקשת פסיקה והעברת '1' לביט מתאים המשמש כדגל כאשר עובדים בשאילתה.

הפולסים לספירה – דפקים של ספירה - יכולים להגיע ממערכת חיצונית למיקרו בקר המתחברת לאחד ההדקים של המיקרו או ממערכת של פולסי שעון פנימיים שהזמן בין כל פולס לפולס הוא קבוע וידוע. במצב של פולסי ספירה המגיעים ממערכת חיצונית אומרים שהמערכת משמשת כקאונטר וכאשר הפולסים מגיעים ממערכת פולסי שעון בתוך המיקרו אומרים שעובדים במצב של טיימר.

בהמשך הספר, למען הקיצור, נשתמש במילה טיימר בלבד במקום טיימר/קאונטר. בהמשך נסביר בהרחבה את ההבדל בין טיימר וקאונטר ובמקום שצריך הפרדה נקרא לפעולת הספירה בשם הנכון.

השימוש בטיימר הוא יצירת השהיה מדויקת, מדידת מרווחי זמן , מדידת זמן של פולס (רוחב דופק) , ליצור בקשות פסיקה כל פרק זמן רצוי ועוד. השימוש כקאונטר הוא לספור אירועים חיצוניים, מדידת תדר של מערכת חיצונית ועוד.

כל אחד מהטיימרים הוא למעשה מונה מעלה - UP COUNTER בן 16 ביטים. כל טיימר מורכב מ 2 חלקים. החלק הגבוה של הטיימר/קאונטר והחלק הנמוך . כל חלק הוא של 8 ביט ולכן לכל טיימר/קאונטר יש 16 ביט. הם נמצאים ב SFR . האיור הבא מראה את המבנה של טיימר 0 וטיימר 1 .



איור 7.א : הטיימרים/קאונטרים המחולקים לחלק גבוה וחלק נמוך כל חלק בן 8 ביטים.

כאשר הטיימר סופר ומגיע למספר המקסימאלי , כל הביטים שלו נמצאים ב 1 . כאשר הוא מקבל פולס ספירה נוסף אז מתבצעת **גלישה - OVERFLOW** , כל הביטים שלו מתאפסים ונשלחת בקשת פסיקה אל מערכת הפסיקות. הטיימר מתחיל לספור שוב מ 0 ואם המתכנת מאפשר פסיקת טיימר מתקבלת פסיקת טיימר. לכל טיימר יש כתובת פסיקה משלו . לדוגמה " כתובת פסיקה של טיימר 0 היא כתובת 0BH בזיכרון התוכנית , או כתובת 1BH עבור בקשת פסיקת מטיימר 1 וכך הלאה.

7.1.1 : ההבדל בין טיימר לקאונטר

הכניסה של דפקי הספירה אל כל אחד מהטיימרים יכולה להגיע משני מקורות, והמשתמש קובע מאיזה מקום – מקור - ייכנסו הדפקים לספירה :

א. דפקי שעון המגיעים מהגל הריבועי של שעון המערכת.

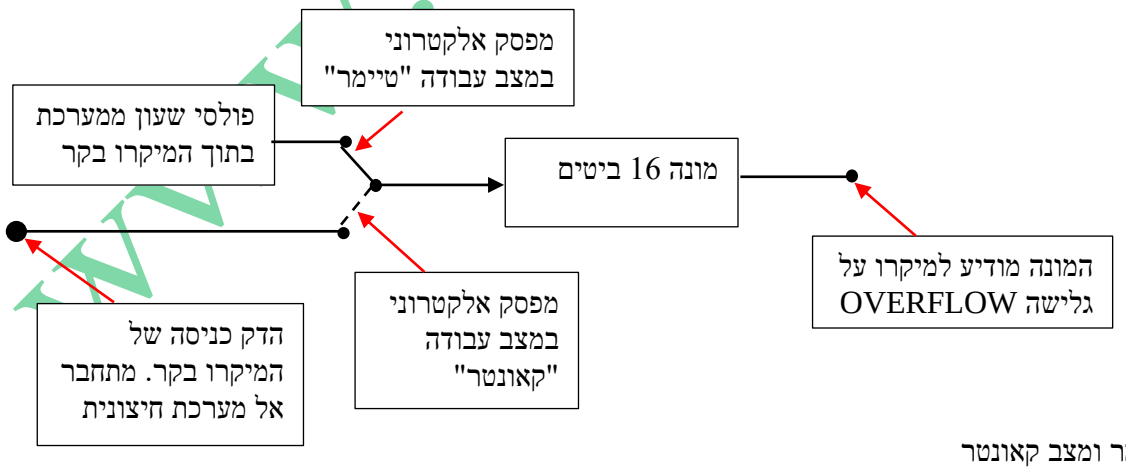
ב. דפקים המגיעים לספירה באחת מהרגלים של המיקרו. ברגל T0 של המיקרו נכנסים דפקים לספירה לטיימר 0 וברגל T1 של המיקרו נכנסים דפקים לספירה לטיימר 1 . כאשר המתכנת קובע שהדפקים שיגיעו הם פולסי שעון של המערכת אז מצב עבודה זה נקרא **"טיימר" – זמנן** . כאשר המתכנת קובע שהדפקים שיגיעו לספירה הם חיצוניים – המצב נקרא עבודה כ**"קאונטר" - מונה**.

המצב שנקרא **"טיימר" – זמנן** , הוא בגלל שניתן במצב זה **לספור זמן**. לדוגמא : אם הפולסים המגיעים לספירה הם בתדר של 1MHz , כלומר במרווחי זמן של 1/1MHz = 1μSec , כלומר כל מיליונית שנייה. אם כל הביטים בטיימר הם '0' - (כלומר המספר ההתחלתי של המונה הוא 0) והוא מקבל פקודה לספור, ואחרי זמן מסוים עצרנו את הספירה וראינו שהטיימר נמצא על 1000 נדע שספרנו 1000 פולסי שעון ולכן הזמן שעבר בין תחילת הספירה לעצירתה הוא :

1000*1μSec = 1000Usec=1msec

כלומר עברה אלפית שנייה. מכאן שבמצב טיימר מודדים זמן ומכאן שמו זמנן.

מצב "קאונטר" הוא ספירת דפקים המגיעם ממערכת חיצונית. לדוגמא : נניח שרוצים לספור בעזרת טיימר 0 כמה צופים נכנסו להיכל הכדורסל ביד אליהו. מפעילים את הטיימר במצב עבודה "קאונטר" . כל אדם הנכנס להיכל עובר דרך כניסה עם חיישן היוצר פולס. הפולסים נכנסים בהדק T0 של המיקרו. כל כניסת אדם, נותנת פולס בכניסת T0 והמונה סופר אותו ומתקדם ב 1 . בסיום כניסת הקהל נקרא את מצב המונה ונדע כמה צופים נכנסו. בגלל מצב הספירה של המונה המצב נקרא **"קאונטר"** .
דוגמה נוספת למצב קאונטר היא בית הדפוס של העיתון "ידיעות אחרונות" . כל עיתון שמודפס ועובר למסוע מייצר פולס ספירה. הקאונטר מראה את כמות העיתונים המודפסים. האיור הבא מראה את ההבדל בין מצב "טיימר" למצב "קאונטר" :



איור 7.ב מצב טיימר ומצב קאונטר

למפסק האלקטרוני (עליו שולט המשתמש) יש 2 מצבי עבודה. המצב המשורטט הוא טיימר ואז הפולסים לספירה מגיעים ממערכת בתוך המיקרו בקר. במצב השני של המפסק (הקו המקווקו) הפולסים לספירה מגיעים ממערכת חיצונית.

7.2 רגיסטרי הבקרה של הטיימרים/קאונטרים

בליבה הבסיסית של המיקרו בקר ממשפחת ה 51 היו 2 טיימרים הנקראים טיימר 0 וטיימר 1 שניהם של 16 ביט. ברכיבים המסתיימים ב 2 או 3 (8052, 8053) יש 3 טיימרים . הטיימר הנוסף נקרא טיימר 2 .
במיקרו בקר C8051F380 יש 6 טיימרים/קאונטרים. שניים (טיימר 0 וטיימר 1) תואמים ל 8051 הסטנדרטי . 4 הנוספים גם הם של 16 ביטים ועובדים במצב הנקרא טעינה אוטומטית – Auto Relaod - לשימוש עם SMBUS או לשימוש כללי.

במיקרו בקר הסטנדרטי של ה 8051 היו שני רגיסטרים השולטים על טיימר 0 וטיימר 1 : (כל טיימר מופעל באופן אישי ולא תלוי בשני). במיקרו בקר C8051F380 יש 4 רגיסטרים .

א. TMOD - קיצור של **(T**imers **MOD**es**)** (אופני טיימרים)
ב. TCON קיצור של **(T**imers **CON**trol **)** - (בקרת הטיימרים).
ג. CKCON - קיצור של ClocK CONtrol – בקרת שעון .
ד. CKCON1 - קיצור של ClocK CONtrol 1 – בקרת שעון 1 .

2 הרגיסטרים הראשונים (בסעיף א ו ב למעלה) היו במיקרו הסטנדרטי 8051. השניים הנוספים קיימים במיקרו בקר C8051F380 .

הרגיסטר **TMOD** קובע האם הטיימרים 0 ו 1 עובדים כטיימרים או קאונטרים, האם מאפשרים או לא מאפשרים להם ספירה וכמו כן את אחד מ 4 אופני (אפשרויות) הפעולה שלהם. (ל 4 הטיימרים הנוספים – טיימרים 2 עד 5 יש רק 2 אופני עבודה).

בעזרת רגיסטר ה **TCON** נותנים לטיימרים פקודת "רוץ" – RUN להתחיל את הספירה או STOP לעצור את הספירה.

בעזרת רגיסטר **CKCON** בקרת השעון קובעים את תדר פולסי השעון המגיע לטיימרים 0,1,2,3

בעזרת רגיסטר **CKCON1** בקרת השעון 1 קובעים את תדר פולסי השעון המגיע לטיימרים 4 ו 5 .

לכל אחד מהטיימרים 2 עד 5 יש עוד 3 רגיסטרים :

1. **TMRnCN** קיצור של TiMeR n CoNtrol – בעברית – בקרת טיימר n (n הוא מספר בין 2 ל 5) .
2. **TMRnRLL** קיצור של TiMeR n ReLoad register Low byte – הביית הנמוך של רגיסטר טעינה חוזרת של טיימר n .
3. **TMRnRLH** קיצור של TiMeR n ReLoad register High byte – הביית הגבוה של רגיסטר טעינה חוזרת של טיימר n .

בפרק זה נתעכב במיוחד על טיימרים 0 ו 1 שהם כמעט זהים במבנה ובאופני ההפעלה.

7.2.1 רגיסטר ה TMOD

רגיסטר TMOD - קיצור של Timer MODe ובעברית אופן הטיימר - קובע האם הטיימרים 0 ו 1 עובדים כטיימרים או קאונטרים, האם מאפשרים או לא מאפשרים להם ספירה וכמו כן את אחד מ 4 אופני (אפשרויות) הפעולה שלהם.

באיור הבא נראה מבנה רגיסטר ה TMOD

Bit	7	6	5	4	3	2	1	0
Name	GATE1	C/T1	T1M[1:0]		GATE0	C/T0	T0M[1:0]	
Type	R/W	R/W	R/W		R/W	R/W	R/W	
Reset	0	0	0	0	0	0	0	0

SFR Address = 0x89; SFR Page = All Pages

Bit	Name	Function
7	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 IT01CF (see SFR Definition 16.7).
6	C/T1	**Counter/Timer 1 Select.** 0: Timer: Timer 1 incremented by clock defined by T1M bit in register CKCON. 1: Counter: Timer 1 incremented by high-to-low transitions on external pin (T1).
5:4	T1M[1:0]	**Timer 1 Mode Select.** These bits select the Timer 1 operation mode. 00: Mode 0, 13-bit Counter/Timer 01: Mode 1, 16-bit Counter/Timer 10: Mode 2, 8-bit Counter/Timer with Auto-Reload 11: Mode 3, Timer 1 Inactive
3	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 IT01CF (see SFR Definition 16.7).
2	C/T0	**Counter/Timer 0 Select.** 0: Timer: Timer 0 incremented by clock defined by T0M bit in register CKCON. 1: Counter: Timer 0 incremented by high-to-low transitions on external pin (T0).
1:0	T0M[1:0]	**Timer 0 Mode Select.** These bits select the Timer 0 operation mode. 00: Mode 0, 13-bit Counter/Timer 01: Mode 1, 16-bit Counter/Timer 10: Mode 2, 8-bit Counter/Timer with Auto-Reload 11: Mode 3, Two 8-bit Counter/Timers

איור 7.ג : מבנה רגיסטר ה TMOD .

הרגיסטר מחולק ל 2 חלקים דומים. 4 הביטים הנמוכים (ביטים 0 עד 3) שולטים על טיימר 0 ו 4 הביטים הגבוהים (ביטים 4 עד 7) שולטים על טיימר 1 . נסביר את 4 הביטים הנמוכים עבור טיימר 0 . הסבר דומה יהיה לגבי 4 הביטים הגבוהים רק לגבי טיימר 1 .

ביט 3 - **GATE0** (שער) – בקרת שער 0. אם בביט נשים '0' - מאפשרים למונה לספור ללא התחשבות בהדק $\overline{INT0}$ (שהזכרנו בפרק על פסיקות). אם נשים בו '1' אז הספירה תלויה בהדק הכניסה החיצוני $\overline{INT0}$. המונה יוכל לספור אם ביט TR0=1 . הביט נמצא ברגיסטר ה TCON שיוסבר בסעיף הבא.

ביט 2 - C/T0 - **Counter/Timer 0 select** – בחירת טיימר או קאונטר . כאשר המתכנת קובע בביט זה 0 עובדים במצב "טיימר" והדפקים המגיעים לספירה הם פולסי השעון של המערכת המגיעים בתדר הנקבע על ידי ביט T0M ברגיסטר CKCON . אם המתכנת שם בביט זה '1' עובדים במצב "קאונטר" כלומר הדפקים לספירה מגיעים מהדק T0 של המיקרו (בירידה מ 1 ל 0 בכניסה זו).

ביטים 0 ו 1 - T0M[1:0] - בחירת אופן העבודה של טיימר 0 - **Timer 0 Mode select** - קובעים את אחד מ 4 אופני עבודה של הטיימר. הטבלה הבאה קובעת את מצבי העבודה עבור טיימר 0 . מצב דומה קיים לטיימר 1 :

	אופן עבודה	תיאור הפעולה
0 0	0	הטיימר/קאונטר עובד כמונה בעל 13 ביט בלבד. TH0 - של 8 ביט , ו TL0 -של 5 ביט (שניתן לטעון אותו למספר התחלתי) . זהו מצב תאימות למיקרו בקר 8048 שהיה דור קודם למיקרו ממשפחת ה 51.
0 1	1	הטיימר/קאונטר עובד כמונה בן 16 ביט.
1 0	2	הטיימר/קאונטר עובד במצב "טעינה אוטומטית" - Auto Reload. במצב זה רק TL0 משמש לספירה. TH0 לא סופר. הוא מקבל ערך התחלתי מהמשתמש ובסיום כל ספירה של TL0 (TL0=0xFF) והוא מקבל פולס ספירה נוסף הוא לא מתאפס אלא נטען למספר שנמצא ב TH0 וממשיך לספור ממספר זה.
1 1	3	TL0 משמש כטיימר/קאונטר ונשלט על ידי הביטים השולטים על טיימר 0 . TH0 משמש כטיימר בלבד **ונשלט על ידי הביטים של טיימר 1 . טיימר 1 איננו סופר במצב זה !**

טבלה 7.א : 4 אופני העבודה של הטיימרים.

7.2.2 רגיסטר TCON – Timer CONtrol - בקרת הטיימר

4 הביטים הנמוכים של הרגיסטר שייכים למערכת הפסיקות והוסברו בפרק על מערכת הפסיקות. 4 הביטים הגבוהים שייכים למערכת הטיימרים 0 ו 1 . 2 ביטים מהארבעה שייכים לטיימר 0 ו 2 שייכים לטיימר 1. נסביר את 2 הביטים השייכים לטיימר 0 . 2 הביטים השייכים לטיימר 1 פועלים בצורה זהה אבל על טיימר 1 . האיור הבא מתאר את רגיסטר ה TCON .

Bit	7	6	5	4	3	2	1	0
Name	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
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 = 0x88; SFR Page = All Pages; Bit-Addressable

Bit	Name	Function
7	TF1	**Timer 1 Overflow Flag.** Set to 1 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.
6	TR1	**Timer 1 Run Control.** Timer 1 is enabled by setting this bit to 1.
5	TF0	**Timer 0 Overflow Flag.** Set to 1 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.
4	TR0	**Timer 0 Run Control.** Timer 0 is enabled by setting this bit to 1.
3	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 in edge-triggered mode.
2	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 16.7). 0: $\overline{INT1}$ is level triggered. 1: $\overline{INT1}$ is edge triggered.
1	IE0	**External Interrupt 0.** 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 0 service routine in edge-triggered mode.
0	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 16.7). 0: $\overline{INT0}$ is level triggered. 1: $\overline{INT0}$ is edge triggered.

איור 7.ד : רגיסטר ה TCON .

לשם שינוי נסביר כאן את 2 הביטים השייכים לטיימר 1 . ביטים אלו הם ביטים 6 ו 7 ברגיסטר. 2 הביטים 4 ו 5 שייכים לטיימר 0 ופועלים בדומה לאלו של טיימר 1 .

TF1 – Timer 1 overFlow – גלישת טיימר 1 : הביט עולה ל 1 על ידי החומרה כאשר בטיימר 1 יש גלישה, כלומר כל הביטים של הטיימר היו ב 1 והטיימר קיבל דופק ספירה נוסף. ניתן לאפס את הדגל בתוכנה עם הפקודה באסמבלי clr TF1 או TF1=0 ב C51 אבל הוא מאופס אוטומטית כשהמיקרו עובר לטפל בשגרת הפסיקה של טיימר 1 (כתובת 1BH בזיכרון התוכנית) . מתכנת העובד בשיטת השאילתה (POLLING) יכול לבדוק את מצב הביט ולדעת האם הטיימר סיים ספירה ומתחיל ספירה חדשה.

TR0 - Timer 1 Run - הביט נשלט בתוכנה על ידי המשתמש. אם המתכנת שם בביט 0 הספירה נעצרת. אם המתכנת שם בביט 1 אז הטיימר מתחיל לספור (תנאי נוסף לספירה היא לשים ב GATE1=1 או הדק INT1=0 כמו שנראה בהסבר שבהמשך).

7.2.3 רגיסטר ClocK CONtrol – CKCON בקרת שעון

רגיסטר בקרת השעון קובע איזה תדר שעון מגיע לטיימרים 0 1 ו 2 .

המבנה שלו נראה באיור הבא :

Bit	7	6	5	4	3	2	1	0
Name	T3MH	T3ML	T2MH	T2ML	T1M	T0M	SCA[1:0]	
Type	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 = 0x8E; SFR Page = All Pages

Bit	Name	Function
7	T3MH	**Timer 3 High Byte Clock Select.** Selects the clock supplied to the Timer 3 high byte (split 8-bit timer mode only). 0: Timer 3 high byte uses the clock defined by the T3XCLK bit in TMR3CN. 1: Timer 3 high byte uses the system clock.
6	T3ML	**Timer 3 Low Byte Clock Select.** Selects the clock supplied to Timer 3. Selects the clock supplied to the lower 8-bit timer in split 8-bit timer mode. 0: Timer 3 low byte uses the clock defined by the T3XCLK bit in TMR3CN. 1: Timer 3 low byte uses the system clock.
5	T2MH	**Timer 2 High Byte Clock Select.** Selects the clock supplied to the Timer 2 high byte (split 8-bit timer mode only). 0: Timer 2 high byte uses the clock defined by the T2XCLK bit in TMR2CN. 1: Timer 2 high byte uses the system clock.
4	T2ML	**Timer 2 Low Byte Clock Select.** 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.
3	T1	**Timer 1 Clock Select.** Selects the clock source supplied to Timer 1. Ignored when C/T1 is set to 1. 0: Timer 1 uses the clock defined by the prescale bits SCA[1:0]. 1: Timer 1 uses the system clock.
2	T0	**Timer 0 Clock Select.** Selects the clock source supplied to Timer 0. Ignored when C/T0 is set to 1. 0: Counter/Timer 0 uses the clock defined by the prescale bits SCA[1:0]. 1: Counter/Timer 0 uses the system clock.
1:0	SCA[1:0]	**Timer 0/1 Prescale Bits.** These bits control the Timer 0/1 Clock Prescaler: 00: System clock divided by 12 01: System clock divided by 4 10: System clock divided by 48 11: External clock divided by 8 (synchronized with the system clock)

איור 7.ב : מבנה רגיסטר CKCON .

ביטים 0 ו 1 : SCA[1:0] - Timer 0/1 preSCAle bits – הביטים של טיימרים 0 ו 1 לחלוקת שעון המערכת . 2 ביטים אלו מבקרים בכמה נחלק את תדר שעון המערכת שיגיעו לטיימר 0 ו טיימר 1 .

הערה : שעון המערכת הנקרא SYSCLK (קיצור של SYSTEM CLOCK) הוא גל מרובע המגיע מיחידת מתנד השעון. שעון המערכת מגיע מיחידת מתנד השעון (מתנד הוא מעגל אלקטרוני היוצר תנודות . במיקרו התנודות הן גל ריבועי). ביחידה זו 4 רגיסטרים הקובעים האם המתנד יהיה פנימי (בתוך המיקרו) או חיצוני, בכמה יחולק תדר המתנד ועוד. הגל הריבועי שיוצא מיחידת המתנד נקרא שעון מערכת. ברירת המחדל של המיקרו בקר היא עבודה עם מתנד פנימי של 12 מגה הרץ. ניתן לעבוד גם עם תדר מתנד של 48 מגה הרץ בעזרת פקודות שנשלח ל 4 הרגיסטרים ביחידת המתנד.

קיימים 4 מצבי חלוקה של שעון המערכת (ביטים 0 ו 1):

00 : שעון המערכת מחולק ב 12 . בעבודה עם SYSCLK של 12 מגה הרץ אז התדר שיגיע לטיימרים 0, 1 ו 2 יהיה 1 מגה הרץ.

01 : שעון המערכת מחולק ב 4 . בעבודה עם SYSCLK של 12 מגה הרץ אז התדר שיגיע לטיימרים 0, 1 ו 2 יהיה 3 מגה הרץ.

10 : שעון המערכת מחולק ב 48 . בעבודה עם SYSCLK של 12 מגה הרץ אז התדר שיגיע לטיימרים 0, 1 ו 2 יהיה 0.25 מגה הרץ. אם קבענו שעון מערכת של 48 מגה הרץ (התדר המגיע מיחידת המתנד) אז לטיימר 0 ו 1 ו 2 יגיע תדר של 1 מגה הרץ.

11 : פולסי שעון חיצוניים מחולקים ב 8 (מסונכרנים עם תדר שעון המערכת).

ביט 2 : **T0** - הוא **T**imer **0** clock select - בחירת מקור טיימר 0 , כלומר מאיפה יגיעו פולסי הספירה לטיימר 0 . אם המתכנת שם בביט 0 אז הפולסים מגיעים לאחר חלוקה של שעון המערכת לפי הביטים 0 ו 1 . אם שמים בביט 1 אז הפולסים הם שעון המערכת ללא חלוקה . כמובן שבמצב קאונטר (ספירת ארועים חיצוניים) אין לביט זה משמעות.

ביט 3 : **T1** - כמו ביט 2 אבל עבור טיימר 1 .

ביט 4 : **T2ML** – קיצור של **T**i**M**er **2** Low byte clock select – בחירת השעון שיגיע לביית הנמוך של טיימר 2 . אם טיימר 2 מוגדר באופן עבודה של טיימר 8 ביט מפוצל (מחולק ל 2 טיימרים של 8 ביט) אז הביט בוחר את השעון המסופק ל 8 הביטים הנמוכים של הטיימר. אם שמים בביט 0 : טיימר 2 משתמש בשעון שהוגדר על ידי ביט T2XLCK ברגיסטר TMR2CN . אם שמים בביט 1 : הביית הנמוך של טיימר 2 משתמש בשעון המערכת.

ביט 5 : **T2MH** – קיצור של **T**i**M**er **2** **H**igh byte clock select- בחירת השעון שיגיע לביית הגבוה של טיימר 2 . אם טיימר 2 מוגדר באופן עבודה של טיימר 8 ביט מפוצל (מחולק ל 2 טיימרים של 8 ביט) אז הביט בוחר את השעון המסופק ל 8 הביטים הנמוכים של הטיימר. אם שמים בביט 0 : טיימר 2 משתמש בשעון שהוגדר על ידי ביט T2XLCK ברגיסטר TMR2CN . אם שמים בביט 1 : הביית הנמוך של טיימר 2 משתמש בשעון המערכת.

ביטים 7 ו 8 **T3ML** ו **T3MH** מבצעים פעולה דומה לביטים 4 ו 5 אבל עבור טיימר 3 .

7.3 אופני העבודה של הטיימרים :

לכל אחד מהטיימרים 4 אופני עבודה הנקבעים על ידי הביטים M1 M0 של כל טיימר הנמצאים ברגיסטר TMOD אופני העבודה.

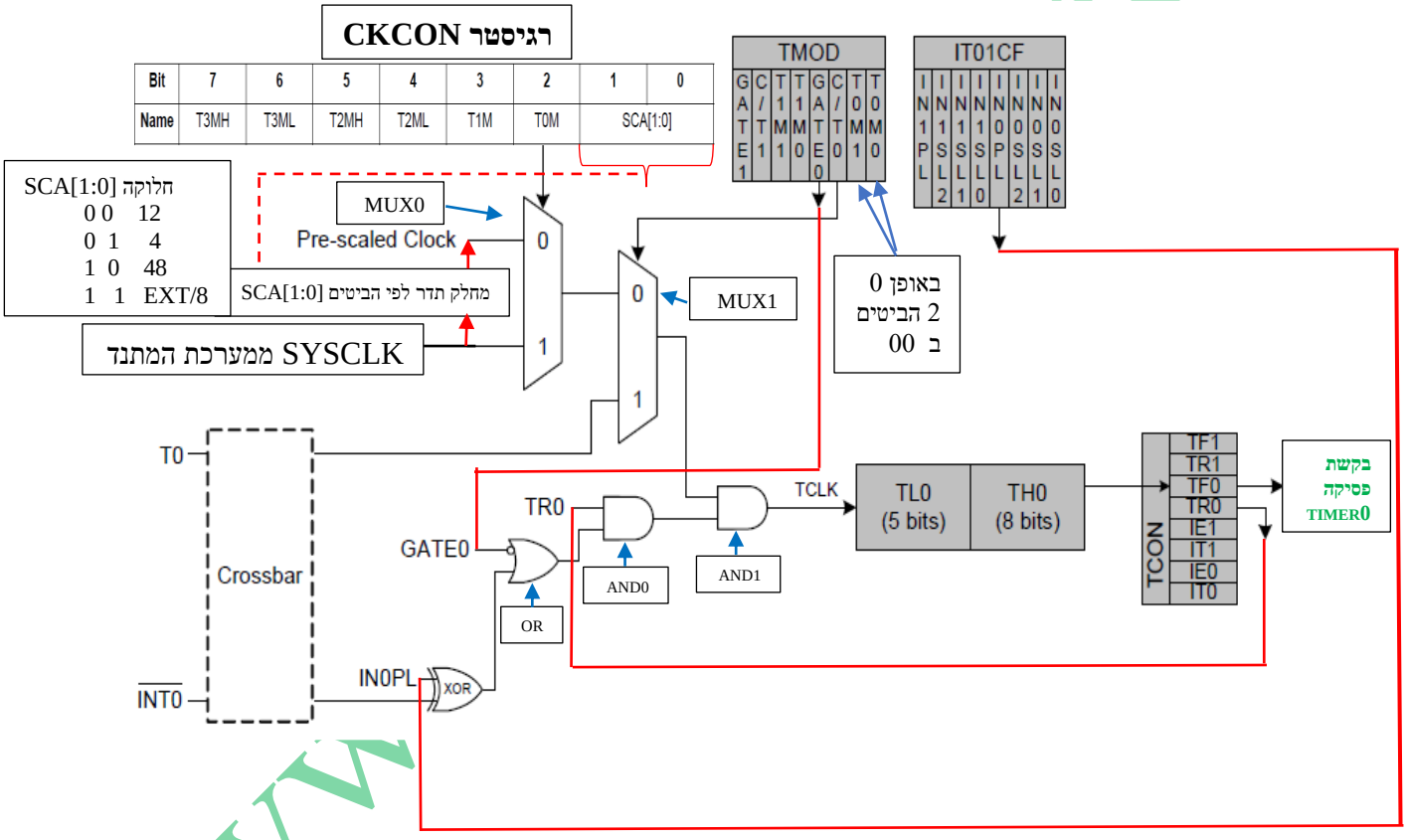
כל המעגלים מסביב לטיימר זהים חוץ מהטיימר עצמו שפעם הוא 13 ביטים או 16 ביטים או 8 ביטים לפי אופן העבודה.

נתאר את 4 אופני העבודה :

7.3.1 Mode 0 - אופן 0

הטיימר TL ו TH (של טיימר 0 או 1) משמשים כטיימר או מונה (counter) בן 13 סיביות (החלק הנמוך כמונה בן 5 סיביות והחלק הגבוה כמונה בן 8 סיביות). אופן 0 הוא תאימות עם מיקרו בקר מדור קודם של INTEL שנקרא 8048 .

באיור הבא מתואר אופן 0 עבור טיימר 0 והביטים השולטים עליו ברגיסטרים המתאימים.



איור 7.ה : טיימר 0 באופן 0

במרכז האיור רואים את שני חלקי טיימר 0 שהם TL0 ו TH0 . באופן 0 משתמשים ב 8 ביטים של TH0 ורק ב 5 ביטים (מתוך ה 8) של TL0 כך שהטיימר הוא של 13 ביטים. כאשר הטיימר מסיים לספור (13 הביטים עברו ממצב שכולם ב 1 למצב שכולם ב 0) מתקבלת גלישה overflow ביציאת הטיימר שגורמת לביט TF0 לעלות ל 1 ולמערכת הפסיקות יוצאת בקשת פסיקה של טיימר 0 .

לטיימר מגיעים פולסים לספירה הנקראים Timer CloCK – TCLK . הם מגיעים מיציאת שער AND ב 2 תנאים :

1. הכניסה התחתונה של שער ה AND צריכה לקבל 1 .
2. הכניסה העליונה של שער ה AND צריכה לקבל את פולסי הספירה.

כדי לקבל '1' בכניסת שער AND1 יש לקבל '1' בשתי כניסות שער AND0 .

הכניסה העליונה מקבלת '1' כאשר המשתמש ירשום לביט TR0=1 (Timer 0 Run – טיימר 0 "רוץ") ברגיסטר ה TCON .

הכניסה התחתונה מקבלת '1' משער OR (שהכניסה העליונה שלו היא מהפכת) .

ביציאת שער ה OR נקבל 1 באפשרויות הבאות :

1. כאשר המתכנת ירשום GATE0=0 אז היות ובכניסת ה OR יש מהפך (העיגול בכניסה) נקבל בתוך השער '1' וביציאה '1' ואז לא משנה מה יש בכניסה משער ה XOR.
2. כאשר יגיע '1' משער ה XOR .

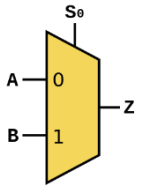
אם המתכנת רוצה שהספירה תהיה על ידי אירוע מבחוץ הוא שם GATE0=1 ואז שער ה OR יוציא '1' רק כאשר הוא יקבל '1' משער ה XOR . דוגמה למצב זה הוא מדידת מרחק עם חיישן SRF04 או SRF05 .

שער XOR מוציא '1' כאשר 2 הכניסות שלו הפוכות אחת לשנייה. הכניסה התחתונה שלו מתחברת להדק $\overline{INT0}$ (שיש לאפשר אותו בעזרת הקרוסבר) והעליונה מתחברת לביט IN0PL (קובע את קוטביות PoLarity של הדק הפסיקה $\overline{INT0}$) ברגיסטר IT01CF.

אם המתכנת קבע IN0PL=0 אז שער ה XOR יקבל '1' כאשר בהדק $\overline{INT0}$ יש '1' . ואם המתכנת קבע IN0PL=1 אז ביציאת שער ה XOR נקבל '1' כאשר בהדק $\overline{INT0}$ יש '0' .

פולסי הספירה לכניסה העליונה של שער AND1 מגיעים ממעגל הנקרא MUX1 – קיצור של MUltipleXer – ובעברית מרבב. זהו מרבב מ 2 ל 1 , כלומר יש לו 2 כניסות נתון (מצד שמאל שלו) , כניסה של קו בחירה (בחלק העליון) ויציאה (בצד ימין שלו). בעזרת קו הבחירה בוחרים איזו כניסה תעבור ליציאה. יש מרבבים מ 4 ל 1 שפרושו 4 כניסות של נתון, 2 כניסות בחירה ויציאה אחת. המספר הדיגיטאלי בהדק/י הבחירה קובע איזו כניסה תעבור ליציאה. יש מרבב מ 8 ל 1 שפרושו 8 כניסות נתון , 3 כניסות בחירה ויציאה וכך הלאה.

האיור הבא מסביר את פעולת המרבב :



איור 7.ו מרבב וטבלת האמת שלו

בחירה S0	יציאה Z
0	A
1	B

בעזרת S0 קובעים איזו כניסה (A או B) תועבר ליציאה Z . כאשר S0=0 הכניסה A מועברת אל היציאה. כאשר S0=1 כניסה B עוברת ליציאה.

באיור של טיימר 0 רואים שקו הבחירה מגיע מביט C/T0 ברגיסטר ה TMOD. אם שמנו בביט C/T0=0 הכניסה העליונה עוברת ליציאה וזהו מצב עבודה של "טיימר" . אם C/T0=1 אז הכניסה התחתונה של המרבב עוברת ליציאה וזהו מצב עבודה של קאונטר.

הכניסה העליונה למרבב מגיעה ממרבב נוסף MUX0 . המרבב נשלט על ידי ביט TOM ברגיסטר CKCON . הרגיסטר קובע בכמה נחלק את התדר מהמתנד והביט קובע במצב TOM=0 שהכניסה העליונה – תדר ה SYSCLK המגיע ממערכת המתנד מחולק במספר שנקבע על ידי הביטים SCA[1:0] של המרבב תעבור ליציאה ובמצב TOM=1 הכניסה התחתונה במרבב עוברת ליציאה , כלומר תדר ה SYSCLK ללא חלוקה.

כמובן שטיימר 1 פועל בדיוק באותה צורה כמו טיימר 0 .

ניתן לסכם את כל האפשרויות להפעלה או הפסקה של הרצת טיימר 0 (ללא קשר למקור פולסי הספירה) בעזרת הטבלה הבאה :

ביט ב TCON	ביט ב TMOD	הדק הפסיקה	הטיימר מאופשר או לא
TR0	**GATE0**	**$\overline{\text{INT0}}$**	**Counter/Timer**
0	X	X	Disabled
1	0	X	Enabled
1	1	0	Disabled
1	1	1	Enabled
Note: X = Don't Care			

טבלה 7.ב : אפשרויות הפעלה/הפסקת ריצת הטיימר

7.3.1.1 דוגמאות

לכל הדוגמאות שבהמשך ניתן להניח שתדר שעון המערכת SYSCLK ממערכת המתנד היא 48MHz .

בחלק משורות התוכנה מופיעה פקודה באסמבלי ומימינה הפקודה בשפת C51 .

1. יש לרשום פקודות להפעלת טיימר 0 באופן 0 במצב "טיימר". פולסי הספירה לטיימר - TCLK - יהיו בתדר 1MHz .

```
mov ckcon,#6 ;      CKCON=6;  //  T0M=1, SCA[1:0]=10  ;  פולסי ספירה עם חלוקה של תדר השעון ב 48
mov tmod,#0  ;      TMOD=0;  //  טיימר 0 באופן 0, כטיימר ומאפשרים ספירה
setb tr0     ;      TR0=1;   //  שים 1 בביט TR0 - טיימר 0 מתחיל לספור
```

2. רשום פקודות להפעלת טיימר 1 כטיימר באופן 0 . פולסי הספירה יהיו 1MHz .

```
mov ckcon,#0ah ;    CKCON=0x0a;  //  T1M=1, SCA[1:0]=10  ;  פולסי ספירה עם חלוקה של תדר השעון ב 48
mov tmod,#0  ;      TMOD=0;  //  טיימר 1 באופן 0, כטיימר ומאפשרים ספירה
setb tr1     ;      TR1=1;   //  שים 1 בביט TR1 - טיימר 1 מתחיל לספור
```

3. מה הפקודה שנרשום כדי לאפשר פסיקת טיימר 1 :

```
mov ie,#10001000b  ;  88H  ;  IE=0x88;
```

4. מהו קצב בקשות הפסיקה המתקבל מהטיימר (יש לזכור שתדר הספירה 1MHZ).

היות והטיימר בן 13 ביטים אז הוא מחלק את התדר (סופר עד המקסימום , מתאפס ושוב מתחיל לספור) :

$2^{13} = 2^5 * 2^{10} = 32*1024=32768$

ומכאן שתדר בקשות הפסיקה יהיה תדר פולסי הכניסה לחלק בכמות הפעמים שהוא מסיים ספירה מתאפס ושוב סופר:

1MHz : 32768 = 30.517578125 פסיקות בשנייה

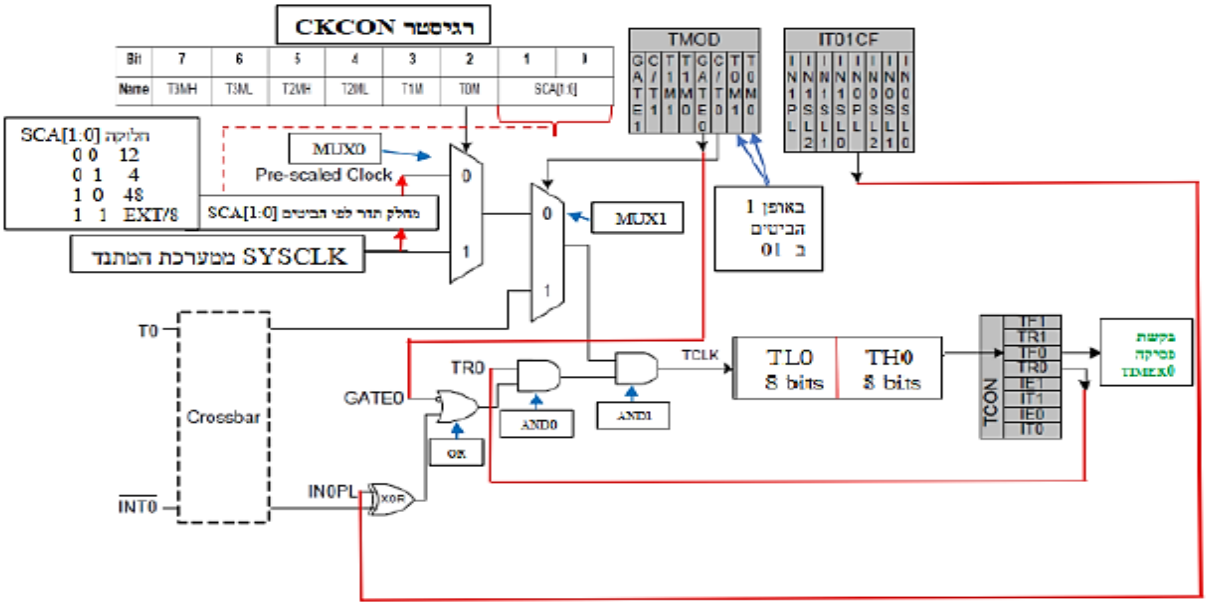
או מקבלים פסיקה כל פרק זמן של : t = 1 / 30.5 = 0.0327868 Sec שניות.

7.3.2 MODE 1 - אופן עבודה 1

הטיימר/קאונטר עובד כמו באופן 0 אבל כטיימר בן 16 סיביות ולא 13 . החלק הנמוך של 8 ביטים וגם החלק הגבוה בן 8 ביטים).
הסבר אופן הפעולה של אופן 1 הוא בדיוק כמו הסבר הפעולה באופן 0 שבסעיף הקודם.

ההבדל בין 2 אופני העבודה מבחינת התוכנה הוא שברגיסטר TMOD יש לרשום את האופן הנכון. אם רשמנו עבור טיימר 0 וגם עבור טיימר 1 לעבודה במצב טיימר באופן 0 את הפקודה MOV TMOD,#0 כדי ישעבדו באופן 0 אז כעת נרשום עבור טיימר 0 בעבודה כטיימר באופן 1 את הפקודה MOV TMOD,#1 , עבור טיימר 1 נרשום MOV TMOD,#10H ועבור שניהם נרשום MOV TMOD,#11H.

האיור הבא מתאר את טיימר 0 באופן 1 :



איור 7.ז : אופן 1 של טיימר 0 .

רואים שהמבנה דומה לזה של אופן 0 (באיור כאן לא ציירנו את הקווים מהרגיסטרים המתאימים אל השערים אבל ניתן לקשר/להבין בעזרת השמות). ההבדל היחיד בין 2 אופני העבודה הוא במבנה של הטיימר עצמו.

7.3.2.1 דוגמאות (בהנחה שתדר השעון 48 מגה הרץ)

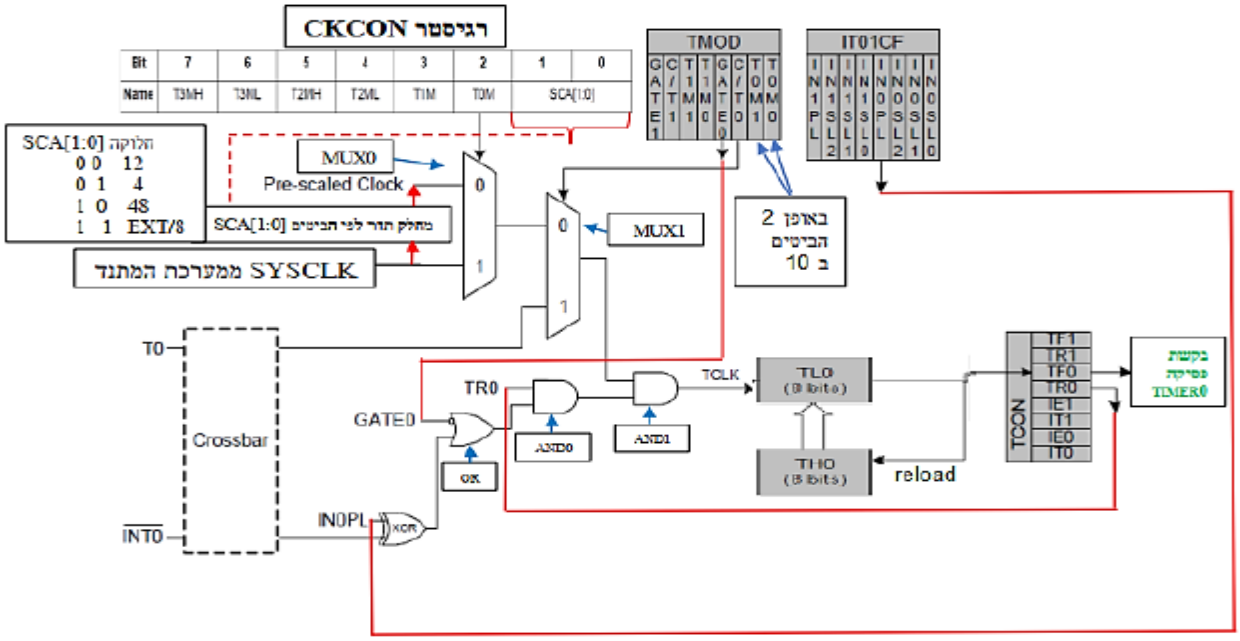
1. יש לרשום פקודות להפעלת טיימר 0 באופן 1 במצב "טיימר". פולסי הספירה לטיימר - TCLK - יהיו בתדר 1MHz . בצד שמאל רשמנו את פקודות האסמבלי ובצד ימין של כל פקודת אסמבלי את הפקודה בשפת C51 המתאימה.

```
mov ckcon,#2 ;      CKCON=2;  //  T0M=1, SCA[1:0]=10  ;  פולסי ספירה עם חלוקה של תדר השעון ב 48
mov tmod,#1  ;      TMOD=1;   //  טיימר 0 באופן 1, כטיימר ומאפשרים ספירה
setb tr0     ;      TR0=1;    //  שים 1 בביט TR0 - טיימר 0 מתחיל לספור
```

2. מהו קצב בקשות הפסיקה באופן זה : פסיקות בשנייה $1MHz / 2^{16} = 15.25$

7.3.3 MODE 2 - אופן עבודה 2

באופן 2 רק החלק הנמוך של הטיימר סופר. החלק הגבוה מקבל מהמתכנת ערך רצוי כלשהו. כאשר TL1 מסיים את הספירה הוא איננו מתאפס אלא נטען למספר שיש ב TH1 .



איור 7.ח : אופן 2 של טיימר 0 .

גם באיור זה לא הראינו את חיבור הביטים למקומות שאליהם הם מחוברים רק ניתן להבין זאת לבד. ניתן לראות שהשינוי היחידי במצב זה הוא מבנה הטיימר. החלק הסופר את פולסי הכניסה הוא החלק הנמוך של הטיימר. כאשר TL0 מסיים ספירה ומתחיל ספירה חדשה הוא פועל כמו ב 2 האופנים הקודמים. הוא שם בביט TF0 '1' המציין שהסתיימה ספירה ומתחילה ספירה חדשה ויוצאת בקשת פסיקה של הטיימר. אבל כאן יש גם דבר נוסף. היציאה מ TL0 מגיעה גם ל TH0 ואומרת לו לטעון מחדש את הערך שיש בו ל TL0. מכאן ש TL0 סופר מהמספר שיש ב TH0 עד 255 , נטען שוב למספר שיש ב TH0 ומתחיל ספירה חדשה.

7.3.1 דוגמה

1. נניח שטענו את TH0 בערך 56. מה קצב הפסיקות המתקבל במצב ש TCLK=1MHz ?

היות ו TL1 מקבל בסיום ספירה את הערך שיש ב TH1 אז הוא סופר מהערך 56 עד 2^8 כלומר עד 255 ובפולס הבא במקום להתאפס הוא חוזר ל 56 ושוב סופר. סה"כ הוא סופר 256-56=200 דפקי שעון. מכאן שקצב הפסיקות יהיה :

פסיקות בשנייה $1*10^6 / 200 = 5000$ = כמות פסיקות בשנייה

מרווחי הזמן בין כל פסיקה לזו שאחריה :

$t = 1 / 5000 = 200*10^{-6} = 200\ \mu Sec$

2. ידוע שתדר TCLK שמגיע לטיימר לספירה הוא 1 מגה הרץ. **בהנחה שהוגדר חיבור הקרוסבר המתאים בפורט 1**. צייר את צורת הגל המתקבלת ב 8 ההדקים של פורט 1 עבור התוכנית הבאה (בצד שמאל אסמבלי ובצד ימין C51):

```
1.      org 0                          #include <REG8051F380>
2.      Ljmp start                     void t0Int ( ) interrupt 1
3.      Org 0bh                        {
4.      Inc p1                                P1++;
5.      Reti                           }
6.      Org 100h                       void main( )
7.  Start: mov tmod,#2                 {
8.      Mov th0,#156                      TMOD=2;
9.      Mov tl0,#156                      TH0=TL0=156;
10.     Mov ie,#82h                       IE=0x82;
11.     mov xbr0,#40h                     XRB0=0x40;
11.     Setb tr0                          TR0=1;
12.     Sjmp $                            while (1);     }
```

פתרון

בהפעלת המערכת, כשהתוכנית מבצעת את שורה מספר 2 העברנו את התוכנית לכתובת 100h . בשורה 7 הפעלנו את טיימר 0 באופן 2 וטענו את החלק הגבוה שלו (וגם את הנמוך) ל 156 . במילים אחרות המונה סופר 156 , 157 , 158 .... עד שמגיע למצב שכל 8 הביטים שלו ב 11111111 (255 עשרוני) ואז בפולס הבא הוא נותן פסיקה ובו בזמן נטען ל 156 ושוב ממשיך לספור 157, 158...וכך הלאה. סה"כ בין פסיקה לפסיקה הוא סופר 100 דפקי שעון.

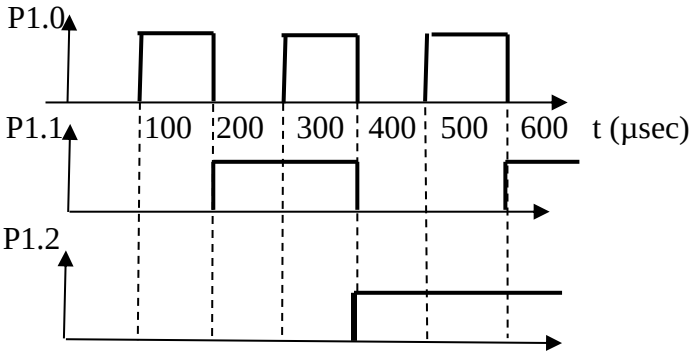
היות ודפקי השעון מגיעים בקצב של 1 מגה הרץ (או זמן של 1 מיקרו שנייה) , והטיימר סופר 100 דפקי שעון אז קצב הפסיקות הוא $1*10^6 / 100 = 10000$ פסיקות בשנייה. בין פסיקה אחת לבאה אחריה יש 100 מיקרו שניות (לפי $T = 1/10000$).

בשורה 10 אפשרנו פסיקת טיימר 0 , בשורה 11 אומרים לטיימר 0 להתחיל לרוץ. בשורה 12 נכנסים ללולאה שחוזרת על עצמה. מהלולאה נצא כל פעם שתתקבל פסיקה מטיימר 0 . נעבור לכתובת 0bh ונבצע את התוכנית שרשומה שם. זוהי תוכנית הפסיקה עבור טיימר 0. בתוכנית זו מגדילים ב 1 את ערך פורט 1 (שורה 4) ובשורה 5 חוזרים לשורה 12 ושוב נכנסים ללולאה סביב שורה 12 עד לפסיקה הבאה של טיימר 0 אחרי 100 מיקרו שניות .

נניח שבפורט 1 היה בהתחלה 00000000 . כל 100 מיקרו שניות נקבל פסיקה וערך הפורט גדל ב 1 .

מצב הפורט בהתחלה	P1 = 0 0 0 0 0 0 0 0
אחרי 100 מיקרו שניות :	P1 = 0 0 0 0 0 0 0 1
אחרי עוד 100 מיקרו	P1 = 0 0 0 0 0 0 1 0
אחרי עוד 100 מיקרו	P1 = 0 0 0 0 0 0 1 1

וכך הלאה... האיור הבא מתאר את צורת הגל 0 (במצב '0' יש מתח של כמעט 0 וולט ואילו במצב '1' יש בסביבות 3 וולט) :



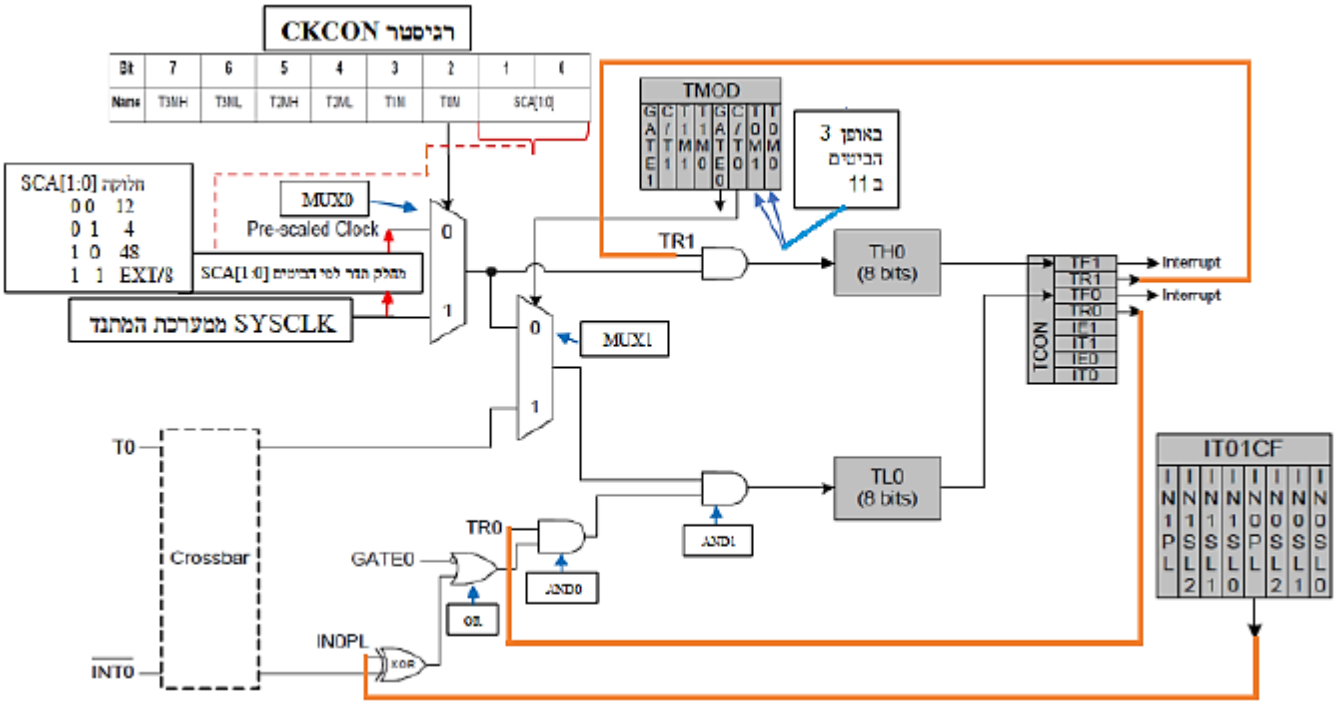
איור 7.ט צורות הגל בהדקי פורט 1 :

ניתן ראות שהגל המרובע שיוצא בהדק P1.0 הוא בתדר של : 1 / 200USec כלומר 5KHz .

בהדק P1.1 יש גל מרובע בתדר פי 2 קטן יותר ולכן יהיה גל מרובע בתדר של 2.5KHz וכך הלאה. קיבלנו 8 גלים מרובעים כשבכל רגל חצי תדירות מזו שמעליה.

7.3.4 MODE3 - אופן עבודה 3

טיימר 0 עובד באופן 3 כשני מונים בני 8 סיביות. **טיימר 1 איננו פעיל באופן 3.** כאשר מפעילים את טיימר 0 באופן 3 משתמשים בביטים TR1 ו TF1 של טיימר 1 . את 8 הביטים הנמוכים של טיימר 0 מפעילים בעזרת הביטים הרגילים של הרגיסטרים ואילו את 8 הביטים הגבוהים של טיימר 0 - TH0 - מפעיל הביט TR1 ! והגלישה של TH0 מפעילה את ביט TF1 ואת הפסיקה של טיימר 1 ! .



איור 7.י : אופן 3 של טיימר 0 .

כאשר טיימר 0 מופעל באופן 3 ניתן לעבוד עם טיימר 1 באופנים 0 , 1 , 2 אבל הוא לא יכול להפעיל את דגל TF1 או לבקש פסיקה . יציאת הגלישה של טיימר 1 יכולה ליצור גל ריבועי לקביעת קצבי תקשורת (כמו לדוגמה עבור תקשורת טורית). אם רוצים לא לעבוד עם טיימר 1 מגדירים לו אופן עבודה 3 .

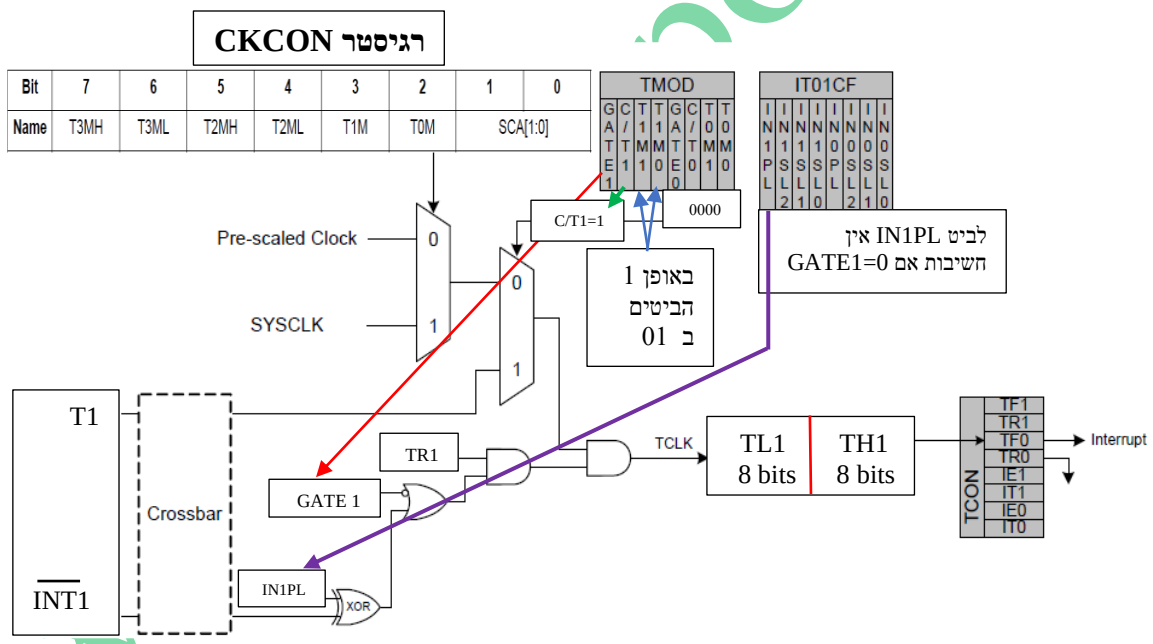
7.3.5 עבודה עם הטיימר כקאונטר (מונה)

הביטים C/T0 ו C/T1 קובעים האם עובדים עם הטיימרים 0 ו 1 כטיימר או קאונטר. במצב שהמתכנת שם בביט 1 הטיימר עובד כקאונטר וסופר פולסי ספירה המגיעים אליו דרך הדק T0 (לטיימר 0) או T1 (לטיימר 1).

דוגמה לשימוש כזה הוא בבית דפוס שבו כל עיתון שמודפס נותן דופק חשמלי. דופק זה ייכנס לספירה בהדק T1 (יש לאפשר אותו בקרוסבר). כאשר נרצה לדעת כמה עיתונים הודפסו ניתן לקרא את הרגיסטרים TL0 TH0 .

דוגמה נוספת לעבודה כקאונטר היא ספירה של כמות אנשים שנכנסת לאצטדיון ספורט. קיים חיישן שדואג לתת פולס חשמלי עבור כל כניסת צופה . בסוף היום נוכל לדעת כמה צופים נכחו באצטדיון.

דוגמה : נעבוד עם טיימר 1 באופן 1, נאפס את החלק הגבוה והנמוך של טיימר 1 , נשים C/T1=1. נשים G= 0 ו TR1=1 ואז המונה סופר את הפולסים המגיעים אליו מחיישן מכניסה לאצטדיון המתחבר להדק T1. בסוף היום נוכל לקרא את מצב TH1 ו TL1 ונדע כמה אנשים נכנסו לאצטדיון .



איור 7.יא : טיימר/קאונטר 1 באופן1 כקאונטר.

פולסי הספירה נכנסים בהדק T1 (יש לקבוע הדק כקלט ולחבר אותו דרך הקרוסבר). בדוגמה הבאה רשמנו רק את תכנות המונה.

בצד שמאל רשמנו את פקודות האסמבלי ובצד ימין של כל פקודת אסמבלי את הפקודה בשפת C51 המתאימה.

```
mov tmod,#50h ;      TMOD=0x50;    // GATE1=0      טיימר 1 באופן 1, כקאונטר ומאפשרים ספירה
setb tr1      ;      TR1=1;        //  שים 1 בביט TR1 - טיימר 1 מתחיל לספור
```

מהי כמות הצופים המקסימלית שניתן למדוד עם מונה 16 ביט ? איך ניתן להגדיל את הכמות במיקרו בקר ?

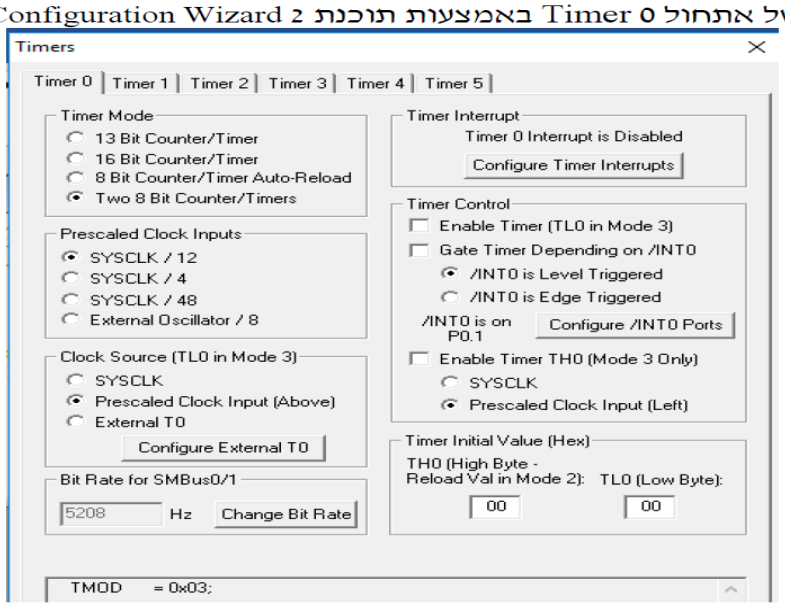
7.3.6 דוגמאות

7.3.6.1 דוגמה 1 :

נתונה התוכנית הבאה המייצרת גל ריבועי בהדק P2.0 בהתאם לערך המתקבל בפורט P1
תדר השעון SYSCLK הוא 12MHz:

```
#include "compiler_defs.h"
#include "C8051F380_defs.h"
void Init_Device(void);
sbit P2_0 = P2 ^ 0;
void main(void){
      Init_Device();
      TMOD = 0x03;
      P2_0 = 0;

      while (1)   {
            if (P1 == 0){
                  TR0 = 0;
                  P2_0 = 0;
            }
            else  {
                  TR0 = 1;
                  TF0 = 0;
                  TL0 = P1;
                  while (TF0 == 0);
                  P2_0 = 1;
                  TF0 = 0;
                  TL0 = 256 - P1;
                  while (TF0 == 0);
                  P2_0 = 0;
            }
      }
}
```

להלן צילום מסך של אתחול Timer 0 באמצעות תוכנת Configuration Wizard 2



א. הסברו את הפקודה TMOD = 0x03; וציין באיזה תדר פועל הטיימר.

ב. צייר עם ערכי הזמן את צורת הגל בפורט P2.0 עבור :

1. P1=0
2. P1=0x80
3. P1=0x20
4. P1=255

הערה : ניתן להזניח את זמני הפקודות .

ג. הצע דרך להגדיל פי 2 את תדר גל המוצא בפורט P2.0

פתרון 1 :

א. אוגר הבקרה TMOD אחראי לפעולת TIMER0/1

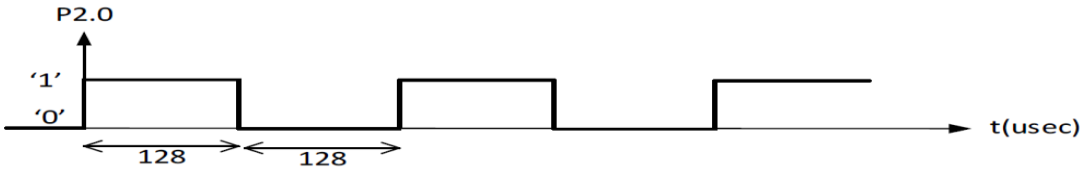
זהו מבנה רגיסטר :

Bit	7	6	5	4	3	2	1	0
Name	GATE1	C/T1	T1M[1]	T1M[0]	GATE0	C/T0	T0M[1]	T0M[0]

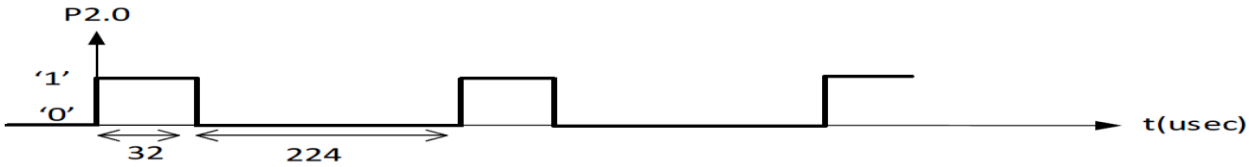
עבור ערך 0x03 המונה 0 פועל במוד 3 בו פועלים 2 מונים של 8 סיביות והמונים פועלים כ-TIMER

ב. 1. נקבל מתח 0 במוצא.

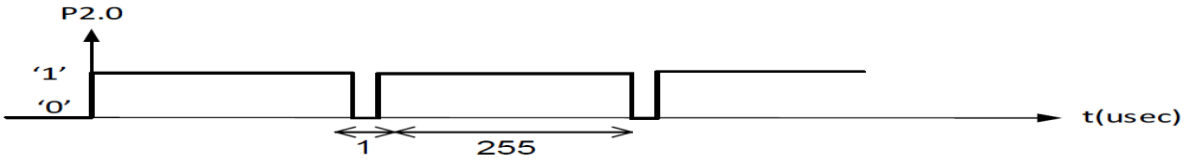
2. המוצא יהיה 0 בזמן שהטיימר ימנה מעלה מהערך 0x80=128 עד שיגיע ל-0 (אחרי הערך 255 ואז הדגל TF0 עולה ל-1) כלומר 128 מחזורי שעון של 1usec (12MHZ/12) , לאחר מכן המוצא יהיה '1' לוגי והמונה סופר מעלה מהערך 256-128=128 עד 0 . לכן נקבל גל מחזורי עם 128usec במצב '0' ו-'1' לוגי .



3. '0' לוגי במוצא במשך 256-0x20= 224usec ו-'1' לוגי במשך 32usec



4. עבור ערך 255 , '0' לוגי במשך 1 מחזור ו-255 מחזורים עבור '1' לוגי



ג. אפשרות אחת היא להגדיל את תדר השעון מ- 12MHz ל-24MHz

7.3.6.2. דוגמה 2 :

נתונה התוכנית הבאה המודדת רוחב פולס חיובי בהדק P2.0 ומאכסנת את הערך הנמדד במשתנה pulseWidth. המיקרו-בקר מאותחל לתדר SYSCLK של 12MHz ומחולק ב- 12 עבור TIMER0

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

void Init_Device(void);

sbit P2_0 = P2 ^ 0;

void main(void){
      U16 pulseWidth;
      Init_Device();
      TMOD = 0x01;

      while (1)  {
            TH0 = 0;
            TL0 = 0;
            while (P2_0 == 0);
            TR0 = 1;
            while (P2_0 == 1);
            TR0 = 0;
            PulseWidth = (TH0 << 8) + TL0;

      }
}
```

א. הסברו את ההוראה TMOD = 0x01;

ב. ציינו את ערך המשתנה pulseWidth , כאשר רוחב הפולס החיובי הוא :

1. 100 usec
2. 12 msec
3. 100 msec

הערה : ניתן להזניח את זמני הפקודות .

ג. מהו רוחב הפולס המינימלי והמכסימלי שניתן למדוד באמצעות תוכנית זו.

ד. הוסיפו/שנו את התוכנית כך שימדוד את זמן המחזור ביחידות מיקרו שניה במשתנה (T) כאשר מסופק לכניסה (P2.0) גל מחזורי ריבועי.

פתרון 2 :

א. אוגר הבקרה TMOD אחראי לפעולת TIMER0/1

זהו מבנה הרגיסטר :

Bit	7	6	5	4	3	2	1	0
Name	GATE1	C/T1	T1M[1]	T1M[0]	GATE0	C/T0	T0M[1]	T0M[0]

עבור ערך 0x01 המונה 0 פועל במוד 1 בו פועל מונה אחד של 16 סיביות ופועל כ- TIMER

ב. המונה של 16 סיביות רץ לפי שעון של 1MHz (12MHz/12, זמן מחזור של 1usec) כאשר P2.0 עולה ל-'1' לוגי ומפסיק כאשר P2.0 יורד ל-'0' לוגי.

1. עבור רוחב פולס של 100usec , PulseWidth = T0 = 100
2. עבור רוחב פולס של 12msec , PulseWidth = T0 = 12000
3. עבור רוחב פולס של 100msec , המונה יכול לספור רק עד 2^{16} כלומר עד רוחב פולס של 65.536msec לכן המונה יגיע לערך זה ויספור שוב בשארית הזמן 100m- 65.536m= 34.464msec ונקבל PulseWidth=34464 .

ג. רוחב פולס מכסימלי 65.536msec ומינימלי 1usec.

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

void Init_Device(void);
sbit P2_0 = P2 ^ 0;

void main(void){
      unsigned int T;
      Init_Device();
      TMOD = 0x01;
      while (1)  {
            TH0 = 0;
            TL0 = 0;
            while (P2_0 == 1);
            while (P2_0 == 0);
            TR0 = 1;
            while (P2_0 == 1);
            while (P2_0 == 0);
            TR0 = 0;
            T = (TH0 << 8) + TL0;
      }
}
```

התוכנית ממתינה לעלייה של '1' לוגי ;(while (P2_0 == 0 (הפקודה לפני כן while (P2_0 == 1); במקרה שהבדיקה מתחילה באמצע הפולס החיובי של הגל)

המונה מתחיל לספור עד P2.0 ירד ל-'0' ;(while (P2_0 == 1 ויעלה שוב ל-'1' while (P2_0 == 0);

פרק 8 : תקשורת טורית - ה UART

8.1 מבוא

שתי שיטות השידור להעברת נתונים בין 2 מערכות הן תקשורת מקבילית ותקשורת טורית. בתקשורת מקבילית משדרים במקביל על 8 חוטים BYTE ביית - על כל חוט ביט. בתקשורת טורית יש רק 2 חוטים ומשדרים את ה BYTE ביית - ביט אחרי ביט. מכאן ברור שיתרון התקשורת המקבילית על התקשורת הטורית הוא במהירות העברת הנתונים ואילו החיסרון בתקשורת המקבילית הוא מחיר (יותר חוטים).

8.1.1 אופני העברת נתונים

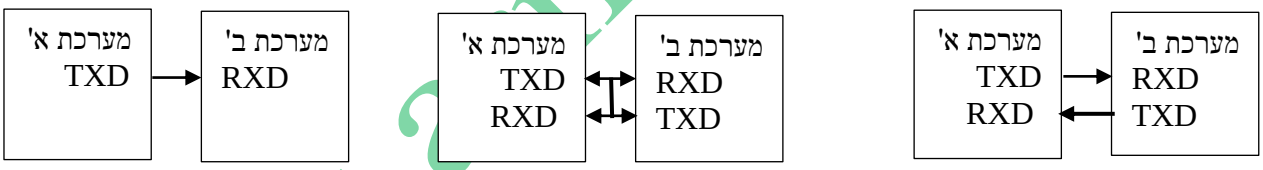
ישנם 3 אופני העברת נתונים : א. SIMPLEX ב. HALF DUPLEX ג. FULL DUPLEX . אופן העברת נתונים מתאר את הכיוון של זרימת האות בין שתי מערכות.

באופן **SIMPLEX** ההעברה של התקשורת היא חד כיוונית. רק מכשיר אחד יכול להעביר את הנתונים לשני. המכשיר השני יכול רק לקלוט . לדוגמה : לוח קשים של מחשב שרק מעביר נתונים למחשב. צג מחשב שרק מקבל נתונים ומציג אותם.

באופן **HALF DUPLEX** - חצי דו כיווני – העברת הנתונים היא דו כיוונית אבל לא באותו זמן. ניתן להעביר נתונים ממערכת א' למערכת ב' ובסיום מערכת ב' מעבירה את הנתונים למערכת א'. לא ניתן לשדר ולקלוט בו זמנית. דוגמאות : מכשירי Walkie - Talkies או מכשירי קשר צבאיים שבהם הצד המשדר והקולט מתחלפים מאחד לשני – "מספר 2 האם שומע ? עבור..." ואז מספר 2 עונה " מספר 2 שומע.. עבור .." והצד המשדר והצד הקולט מתחלפים בינהם .

באופן **FULL DUPLEX** – דו כיווני מלא – כל מערכת יכולה לשדר ולקלוט בו זמנית. לדוגמה: טלפון.

באיור הבא מתוארים 3 אופני העבודה.



SIMPLEX HALF DUPLEX FULL DUPLEX

איור 8.1 3 אופני החיבור של תקשורת טורית.

8.1.2 מהו UART ?

בתוך המיקרו בקר קיימת מערכת לתקשורת טורית הנקראת **UART** - **Universal Asynchronous Receiver Transmitter** - **מערכת משדר מקלט א-סינכרונית אוניברסאלית**. מערכת זו יודעת לקבל ביית מהמיקרו ולשדר אותו בצורה טורית בהדק שנקרא TxD של המיקרו. בסיום השידור של ה ביית , ה UART מודיע למיקרו שהוא סיים לשדר את הביית והוא מוכן לקבל ביית חדש לשידור. ה UART משדר את הביית לפי פרוטוקול תקשורת טורית : קודם את ביט ההתחלה – Start Bit , אח"כ את סיביות הנתון ולבסוף את ביט הזוגיות - PARITY (אם הוחלט שעובדים עם זוגיות) ואת ביט הסיום Stop Bit . כפי שהזכרנו - בסיום שידור התו ה UART מודיעה למיקרו (גם בעזרת פסיקה וגם על ידי דגל TI) על סיום שידור התו. כמובן

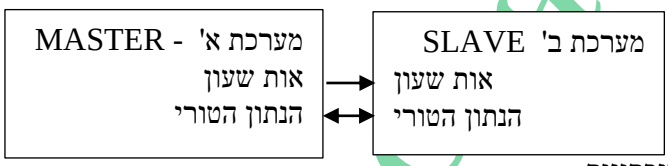
שהתוספת של ביט ההתחלה וביט הסיום מאטים עוד יותר את קצב התקשורת ולכן הומצאו שיטות חדשות יותר (יוסבר בפרקים הבאים של הספר).

בקליטה טורית קורה תהליך הפוך. המיקרו מקבל בהדק שנקרא RxD את הביטים הטוריים אחד אחרי השני. ה UART מזהה את ביט ההתחלה, את הביטים של הנתון , את הביט של הזוגיות (אם הוחלט לעבוד עם זוגיות) ואת ביט הסיום ומודיע למיקרו על ידי פסיקה וגם על ידי דגל RI, על סיום קליטה טורית. המיקרו יקרא את הביית שנקלט וה UART יהיה מוכן לקליטת ביית חדש. באופן עבודה כזה שה UART מבצע את פעולת השידור והקליטה , המיקרו פנוי לטפל בדברים נוספים ולא צריך לנהל את שידור או קליטת הנתונים הטורית.

8.1.3 תקשורת אסינכרונית וסינכרונית

תקשורת טורית – UART - היא תקשורת **אסינכרונית** כי אין לנו תדר שעון מרכזי המסנכרן את מעבר הנתונים.

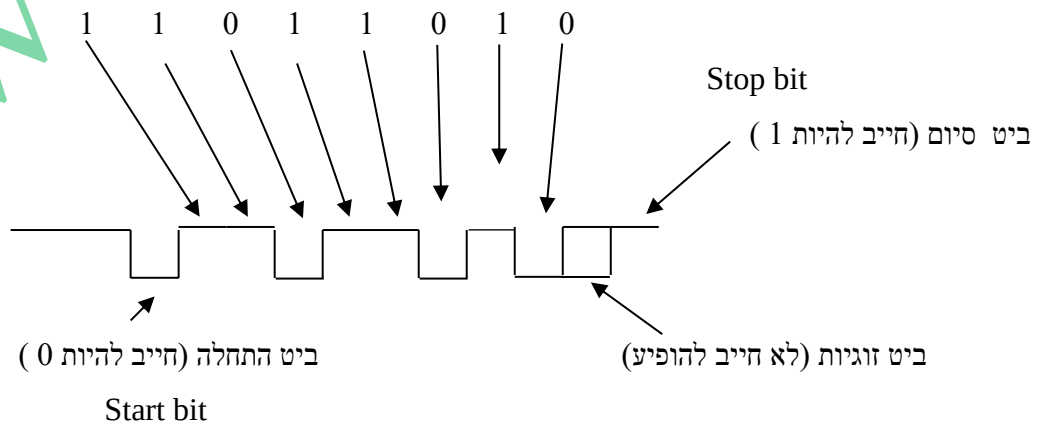
קיימות מערכות תקשורת טורית נוספות כמו I2C או SPI שהן מערכות תקשורת טורית **סינכרוניות** . במערכות אלו קיים אדון ועבד/ים MASTER - SLAVE ויש תדר שעון מרכזי של מערכת האדון MASTER שנותן פולסי שעון לסנכרון לכל מערכות העבד – SLAVE - המתחברות אליו. האיור הבא מתאר תקשורת טורית **סינכרונית**:



איור 8.2 : תקשורת טורית סינכרונית

8.1.4 שידור נתון טורי

נזכיר כיצד נשלח נתון בקו תקשורת טורית. נניח שרוצים לשדר את התו 5BH כלומר 01011011 . התו משודר מהביט הנמוך אל הגבוה (מה LSB אל ה MSB). במצב שאין שידור הקו נמצא בגבוה . התחלת שידור מתבצעת עם הורדת הקו ל 0 לזמן של ביט. ביט זה נקרא ביט התחלה – start bit . לאחר ביט ההתחלה משודרים 8 הביטים של הנתון מהביט הנמוך D0 ועד הביט הגבוה D7 . אם הוחלט לעבוד עם זוגיות אז משודר גם הביט התשיעי (במידה ולא אז אין ביט תשיעי) ולבסוף הקו עולה ל 1 ומציין סיום שידור. ביט זה נקרא ביט סיום/עצירה - stop bit . ביית חדש שישודר מתחיל שוב עם ביט התחלה ואחר כך הביטים של הנתון וביט סיום . האיור הבא מראה שידור של ביית .



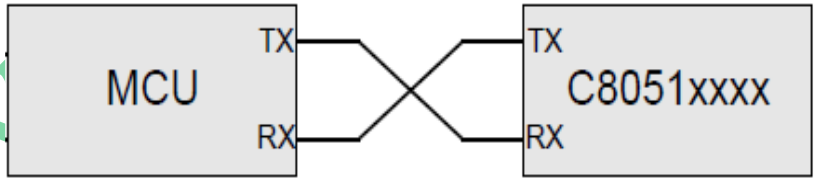
איור 8.3 : שידור הנתון 5Bh בתקשורת טורית .

8.1.5 מאפייני השידור

כמות הביטים הנשלחת בשנייה נקראת **קצב התקשורת או קצב הביטים** - Bit Rate. היחידות הן **סיביות לשנייה או סל"ש.** **באנגלית Bits Per Second – bps** . קצבי שידור מקובלים בתקשורת טורית : 4800, 9600, 19200, 28800, 57600, 115200 ו 230400 ביטים בשנייה.

בתחילת ימי התקשורת הטורית היה מושג נפוץ הנקרא באוד - BAUD - לתיאור של קצב העברת המידע. הוא ייצג את כמות הסמלים בשנייה. סמל הוא יחידת מידע שיכולה לייצג יותר מביט אחד. הבאוד הוחלף במושג קצב ביטים הנמדד בביטים לשנייה.

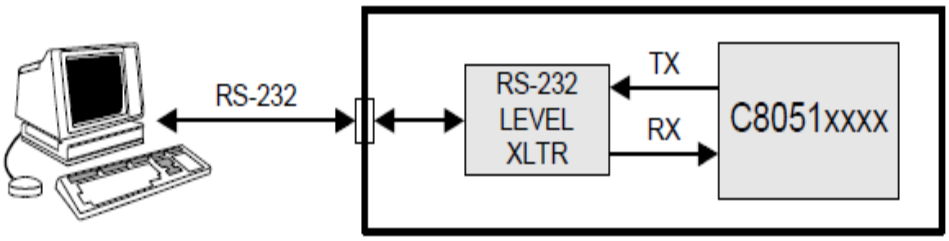
כמות הביטים של הנתון במיקרו שלנו יכולים להיות 8 או 9 ביטים בכל מסגרת (כלומר בין ביט ההתחלה לביט הסיום). ביט הזוגיות הוא ביט שמציין את כמות ה 1 שיש בנתון. בתחילת ימי התקשורת הטורית היו משתמשים בביט הזוגיות כדי שהצד הקולט יידע האם הוא קלט את המידע נכון. לדוגמא : נניח שמשדרים את התו 5bH . בתו יש 5 פעמים 1 . היות וכמות ה 1 איננה זוגית היו שמים בביט הזוגיות 0 . הצד הקולט סופר את כמות ה 1 שקלט. היות והוא קלט 5 פעמים 1 הוא יודע שביט הזוגיות שיקלוט חייב להיות 0 . אם הוא קלט ביט זוגיות 0 מסקנה – הקליטה נכונה. אם אחד הביטים היה מתהפך בדרך אז הצד הקולט היה קולט מספר זוגי של 1 ואז היה רואה שביט הזוגיות הוא 0 והיה מבחין שקלט נתון שגוי. במקרה כזה הוא יכול לבקש מהצד המשדר לשלוח את הנתון פעם נוספת. הוספת ביט הזוגיות גורם לביט טורי נוסף ל 8 הביטים של המידע, דבר שמאט את קצב התקשורת הטורית, הנמוך ממילא. היום יש שיטות חדשות לאיתור שגיאות ואפילו לתיקון שגיאות. ניתן לקבוע את זמן ביט הסיום לאורך זמן של 1, 1.5 או 2 סיביות. סט הפרמטרים הנפוץ ביותר הוא **8N1** שמשמעותו '8' ביטים של מידע , 'N' - ללא סיבית זוגיות, '1' - אורך ביט הסיום היא זמן ביט יחידה. על-מנת לקיים תקשורת תקינה על שני הצדדים לכוון לאותו סט של פרמטרים, למשל 9200 - 8N1 1 (19200 קצב הביטים בשנייה). באיור הבא רואים חיבור של המיקרו C8051Fxx אל מיקרו בקר אחר. החיבור הוא FULL DUPLEX. רואים שהדק השידור של מיקרו אחד מתחברת להדק הקליטה של המיקרו השני ולהפך.



איור 8.4 חיבור בין 2 מיקרו בקרים בתקשורת טורית.

תקן לשידור טורי נקרא גם RS-232 או EIA RS-232 (RS קיצור של Recommended Standard – סטנדרט/תקן מומלץ) והוא משמש להעברת נתונים בין מכשיר DTE – Data Terminal Equipment – ציוד קצה נתונים - כמו מחשב ובין מכשיר DCE – Data Circuit Equipment – ציוד מעגל נתונים – הקולט את המידע. התקן עוסק ברמות מתח של שליחת הנתונים. רמה לוגית של 0 מיוצגת על ידי רמת מתח בין 7 וולט ל 15 וולט ורמה לוגית של '1' מיוצגת על ידי מתח בין 7- עד 15- וולט. הרמה הלוגית 0 גבוהה מזו של ה 1 וזה נקרא לוגיקה שלילית. כמו כן התקן קובע כיצד ייראה המחבר (קונקטור) של התקשורת ועוד.

באיור הבא רואים חיבור בין מחשב ובין מיקרו בקר ממשפחת SILABS עם מתאם רמות מתח מ TTL ל RS232 - המלבן נקרא באיור RS - 232 LEVEL XLTR.



איור 8.5 חיבור בין מיקרו בקר ובין מחשב עם ממיר רמות מתח מ TTL ל RS232 .

בצד הקולט רואים שהקו בגבוה "ומבינים" שהצד השני מחובר . (אם ללא שידור היה 0 לא היינו יודעים האם הצד השני מחובר). ברגע שהצד הקולט מרגיש 0 הוא ממתין זמן של חצי ביט ושוב דוגם את הקו לראות שיש עדיין 0 (להיות בטוח שה 0 הקודם לא היה רעש אקראי) . לאחר מכן הוא דוגם את הקו כל זמן של ביט ואוסף את 8 הביטים של הנתון (ואם יש גם ביט זוגיות אז 9 ביטים) , דוגם את הקו כדי לוודא שיש 1 שזהו ביט הסיום ומודיע למיקרו שנקלט ביית . המיקרו צריך לקרא את הביית שנקלט כדי שהמקלט יוכל לקלוט את הביית החדש שישודר.

8.2 התקשורת הטורית במיקרו C8051F380

במיקרו בקר הבסיסי ממשפחת ה 51 היה רק UART אחד בלבד. ב C8051F380 יש 2 מערכות UART הנקראות UART0 ו UART1 . הן עובדות במקביל וכל אחת עצמאית ללא קשר לשנייה. UART0 דומה בפעולתו ל UART שהיה ב 51 המקורי. UART1 עובד בצורה גדולה יותר, יש לו 6 רגיסטרים ב SFR עם אפשרות לקלוט 3 נתונים ללא קריאה של המיקרו בקר. בספר זה נעסוק רק ב UART0 .

8.2.1 UART0

היחידה UART0 היא מערכת תקשורת טורית , אסינכרונית , FULL DUPLEX , המאפשרת את אופני העבודה 1 ו 3 של המיקרו 8051 המקורי. אופן 1 הוא שידור טורי של 8 ביטים ואופן 3 שידור של 9 ביטים בקצב שידור / קליטה משתנים (קצב הנשלט על ידי המשתמש) . במיקרו C8051F380 יש תמיכה עשירה של מקורות תדר שעון ליצירת קצבי שידור סטנדרטיים (נראה בטבלה בעמודים הבאים) . מערכת חוצצי נתונים מאפשרת להתחיל לקלוט ביית נוסף עוד לפני שהמיקרו קרא את הנתון הקודם.

ל UART0 יש 3 רגיסטרים ב SFR .

א. SCON0 - Serial Control Register 0 - רגיסטר בקרה טורית 0

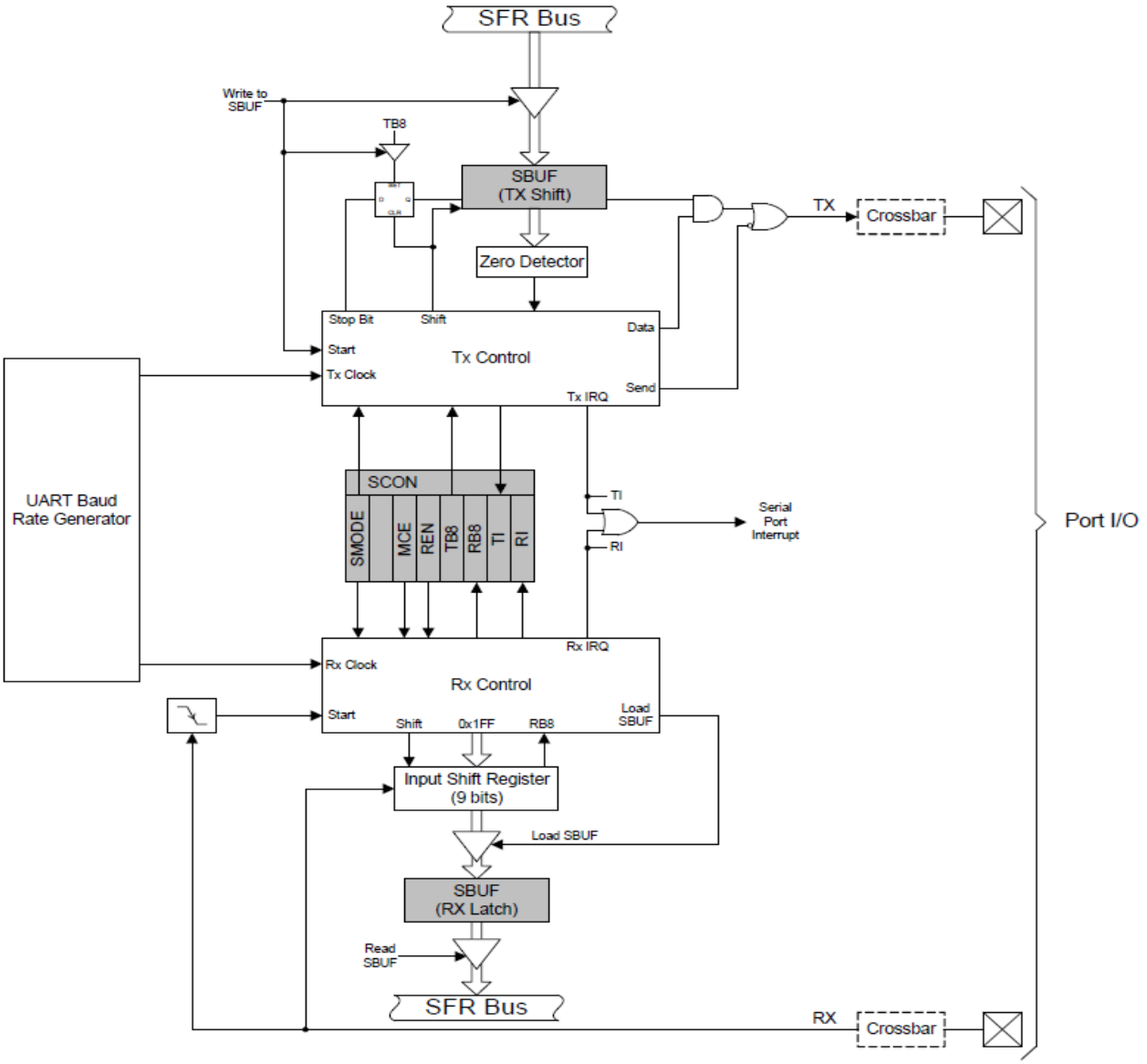
ב. SBUF0 - Serial BUFfer 0 – חוצץ טורי 0 . למעשה יש 2 חוצצים טוריים SBUF0 , אחד לשידור ואחד לקליטה. כאשר המיקרו כותב נתון ל SBUF0 הוא פונה ל SBUF0 של השידור . כאשר המיקרו קורא נתון הוא קורא מה SBUF של הקליטה.

במיקרו 51 המקורי היו 3 רגיסטרים השייכים לתקשורת הטורית. רגיסטר ה SCON השולט על אופן העבודה של התקשורת (כמות סיביות, זוגיות, קצב תקשורת) ושני רגיסטרים של SBUF הנמצאים בכתובת 99H באזור ה SFR. אחד מקבל את הבית לשידור כאשר המיקרו כותב אליו ואז הוא משדר אותו והשני מכיל את הבית שנקלט והמיקרו קורא אותו. ההבדל בגישה אליהם מתבצע בעזרת קווי קריאה /כתיבה פנימיים. לא ניתן לקרוא נתון מה SBUF0 של השידור ולא ניתן לכתוב ל SBUF0 של הקליטה.

כאשר UART0 מסיים לשדר את הנתון הוא מבקש פסיקת תקשורת טורית (SERIAL) וביט TI ב SCON0 עולה ל 1 . דבר דומה קורה בקליטה. כאשר הסתיימה קליטה של נתון יש בקשת פסיקה טורית וביט RI ב SCON0 עולה ל 1 . היות וגם סיום השידור וגם סיום קליטה יוצרים אותה בקשת פסיקה (כתובת 23H - פסיקה מספר 4) , על המתכנת לבדוק את הביטים TI ו RI כדי לדעת מאיזו סיבה יש בקשת פסיקה (האם סיום שידור או סיום קליטה). יש לזכור שכאשר המיקרו פונה לטיפול בפסיקה הביטים RI ו TI אינם מתאפסים על ידי החומרה ויש לאפס אותם בתוכנה.

8.2.2 סכמה מלבנית של UART0

באיור הבא מתוארת הסכמה המלבנית של UART0 .



איור 8.6 הסכמה המלבנית של UART0 .

- החלק העליון באיור, מהמלבן Tx Control ומעלה זהו **המשדר של ה UART**. בחלק העליון שלו נמצא ה SBUF של השידור שהוא רגיסטר הזזה. במלבן Tx Control יש מעגלי אלקטרוניקה השולטים על רגיסטר ההזזה ומוציאים ממנו את הנתון לפי הפרוטוקול הטורי שהסברנו עם ביט התחלה וסיום. תחילת השידור הטורי מתבצעת עם העברת הנתון ל SBUF.

- החלק התחתון של האיור, מהמלבן Rx Control כלפי מטה - זהו **המקלט של ה UART**. בחלק התחתון שלו נמצא ה SBUF של הקליטה. ה Rx Control הוא מערכת בקרה הכולל מעגלי אלקטרוניקה השולטים על רגיסטר הזזה של 9 ביט - Input Shift Register - הקולט את הנתון הטורי ביט אחרי ביט כולל הזזה. הקליטה מגיעה כאשר מזהים ירידה מ 1 ל 0 בהדק RX. ירידה זו נכנסת לרגל START של מלבן ה RX Control . כל ביט שנקלט מקבל הזזה ימנית בעזרת הקו Shift . בסיום קליטת הנתון מערכת הבקרה נותנת פולס Load SBUF לחוצץ (מצויר כמשולש) שמעביר את הנתון שנקלט ברגיסטר ההזזה ל SBUF של הקליטה. כמו כן מוציאה מערכת הבקרה בקו Rx IRQ בקשת פסיקה של תקשורת טורית.

- בין מערכת השידור והקליטה נמצא רגיסטר ה SCON - Serial CONtrol - בקרה טורית אשר מבקר על התקשורת הטורית.

- באמצע משמאל נמצא המלבן UART Baud Rate Generator - מחולל קצב הבאוד של ה UART. זהו המלבן שנותן את תדר השעון להפעלת המשדר והמקלט . אות השעון שהוא נותן למערכת השידור והקליטה נקרא Tx Clock ו Rx Clock בהתאמה.

- אפשר לראות שהגישה ל SBUF של השידור היא בעזרת אות כתיבה - Write to SBUF ואילו הגישה ל SBUF של הקליטה היא בעזרת אות קריאה - Read SBUF .

- בחלק הימני ביותר של האיור יש את קו ה TX (למעלה באיור) ואת קו ה RX (למטה באיור) המתחברים בעזרת הקרוסבר להדק המתאים של הרכיב .

- מימין ל SCON יש שער OR המתחבר אל Tx IRQ ואל Rx IRQ . כאשר ה UART סיים לשדר את הביית שנמצא ב SBUF של השידור הוא מוציא '1' אל הדק Tx IRQ וגם לביט TI ב SCON . ביציאת שער ה OR יש '1'. יציאה זו מתחברת למערכת הפסיקות ויש בקשת פסיקה של UART . דבר דומה קיים במערכת הקליטה של ה UART . כאשר ה UART סיים לקלוט נתון הוא מוציא '1' אל שער ה OR וגם לביט RI ב SCON . ביציאת שער ה OR יהיה '1' ונקבל בקשת פסיקה. אם נאפשר פסיקת תקשורת טורית אז כאשר המיקרו פונה לתוכנית הפסיקה, יש לבדוק האם הפסיקה היא של סיום שידור או סיום פסיקה ובהתאמה לטפל בפסיקה. היות והתקשורת הטורית היא FULL DUPLEX יכול להיות מצב שנקבל פסיקה גם מהמשדר וגם מהמקלט בו זמנית. יש לזכור שבתוכנית הפסיקה יש לאפס את ביט ה TI או/ו ה RI בתוכנה כדי להכין אותם למחזור שידור/קליטה חדשים.בצד ימין של האיור רואים את הדקי השידור Tx והקליטה Rx של יחידת ה UART. כדי שהדקים אלו יצאו אל הדקי הרכיב יש לאפשר את הקרוסבר בעזרת ביט XBARE של xbr1 . הפקודה היא ; XBR1=0x40 . כמו כן יש לאפשר ברגיסטר ה xbr0 שבאיור הבא את הביט URT0E ולשים בו 1 כדי שהדק השידור יתחבר אל P0.4 והדק הקליטה ל P0.5 . הדקים אלו קבועים ולא ניתן לשנות אותם.

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

איור 8.7 : רגיסטר XBR0 : 0 – הדקי התקשורת של ה UART לא מתחברים. 1 – מתחברים להדקי P0.4 (Tx) ו P0.5 (Rx)

8.2.3 רגיסטר בקרת התקשורת הטורית – SCON0

באיור הבא מתואר רגיסטר ה **SCON0** - Serial CONtrol 0 - בקרה טורית 0. בעזרתו שולטים על התקשורת הטורית. איתו קובעים האם עובדים עם 8 או 9 ביט, האם עובדים עם UART בודד או כמה מעבדים עם UART .

נסביר כל ביט בעזרת האיור הבא :

Bit	7	6	5	4	3	2	1	0
Name	S0MODE	-	MCE0	REN0	TB80	RB80	TI0	RI0
Type	R/W	R	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	1	0	0	0	0	0	0

SFR Address = 0x98; SFR Page = All Pages; Bit-Addressable

Bit	Name	Function
7	S0MODE	**Serial Port 0 Operation Mode.** Selects the UART0 Operation Mode. 0: 8-bit UART with Variable Baud Rate. 1: 9-bit UART with Variable Baud Rate. בחירת אופן הפעולה עם פורט טורי 0 (כמות סיביות מידע): 0 – UART של 8 ביט עם קצב תקשורת משתנה. 1 – UART של 9 ביט עם קצב תקשורת משתנה. קצב תקשורת משתנה אומר שהמשתמש בעזרת תוכנה קובע את הקצב.
6	Unused	Read = 1b, Write = don't care. ללא שימוש.
5	MCE0	**Multiprocessor Communication Enable.** The function of this bit is dependent on the Serial Port 0 Operation Mode: **Mode 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. **Mode 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. אפשור תקשורת מרובת מעבדים. 0 – המיקרו מתחבר אל UART אחד. 1 - המיקרו מתחבר אל מספר UART. מצב הקיים רק בעבודה עם 9 ביט
4	REN0	**Receive Enable.** 0: UART0 reception disabled. 1: UART0 reception enabled. אפשור קליטה. 0 – חסימת קליטה. 1 - אפשור קליטה
3	TB80	**Ninth Transmission Bit.** The logic level of this bit will be sent as the ninth transmission bit in 9-bit UART Mode (Mode 1). Unused in 8-bit mode (Mode 0). הביט ה 9 שישודר בעבודה עם UART של 9 ביטים.
2	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. הביט ה 9 שנקלט בעבודה עם UART של 9 ביטים.
1	TI0	**Transmit Interrupt Flag.** הסבר בהמשך 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.
0	RI0	**Receive Interrupt Flag.** הסבר בהמשך 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.

איור 8.8 : רגיסטר SCON0

RI0 Receive Interrupt flag - דגל פסיקת קליטה

הביט עולה ל 1 על ידי החומרה של ה UART כאשר הסתיימה קליטה טורית של ביית. המתכנת יבדוק את הביט . אם יש בו 1 יודעים שנקלט בית חדש. יש למשוך את הבית מה SBUF ולאפס את הביט בתוכנה (CLR RI) כדי להתכונן לקליטת ביית חדש.

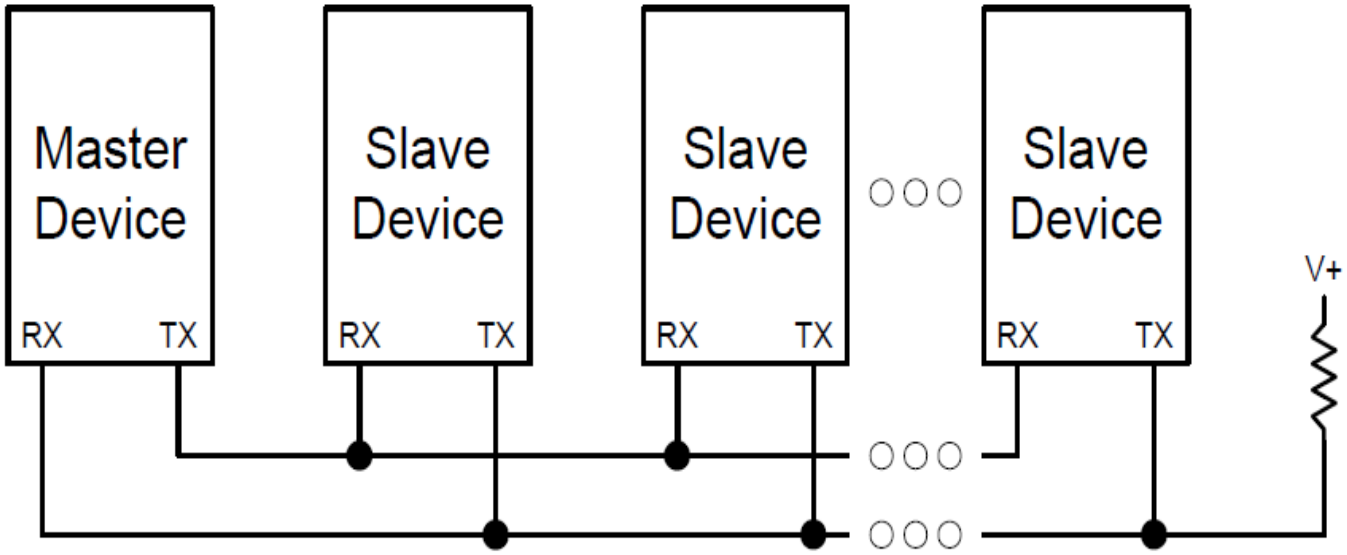
TI0 Transmit Interrupt flag – דגל פסיקת שידור

הביט עולה ל 1 על ידי החומרה של ה UART כאשר הסתיים שידור התו שנמצא ב SBUF . המתכנת בודק את הביט ואם יש בו 1 הוא יודע שהסתיים שידור הבייט ואז הוא שם בביט אפס בעזרת הפקודה CLR TI ואז יכול לשלוח ביית חדש לשידור.

הערה : ניתן להשתמש בבדיקת 2 הביטים כאשר עובדים בשיטת השאילתה – POLLING , אבל גם בשיטת הפסיקה. כאשר מתקבלת פסיקת תקשורת טורית, היא יכולה להתקבל או בסיום שידור ביית או בסיום קליטת ביית. בעזרת 2 הביטים TI0 ו RI0המתכנת יודע האם הפסיקה היא של השידור או של הקליטה.

8.2.4 עבודה בסביבה מרובת מעבדים

האיור הבא מתאר חיבור בסביבה מרובת מעבדים. רואים שמספר יחידות של עבד – SLAVE - מתחברות אל אדון -MASTER אחד.

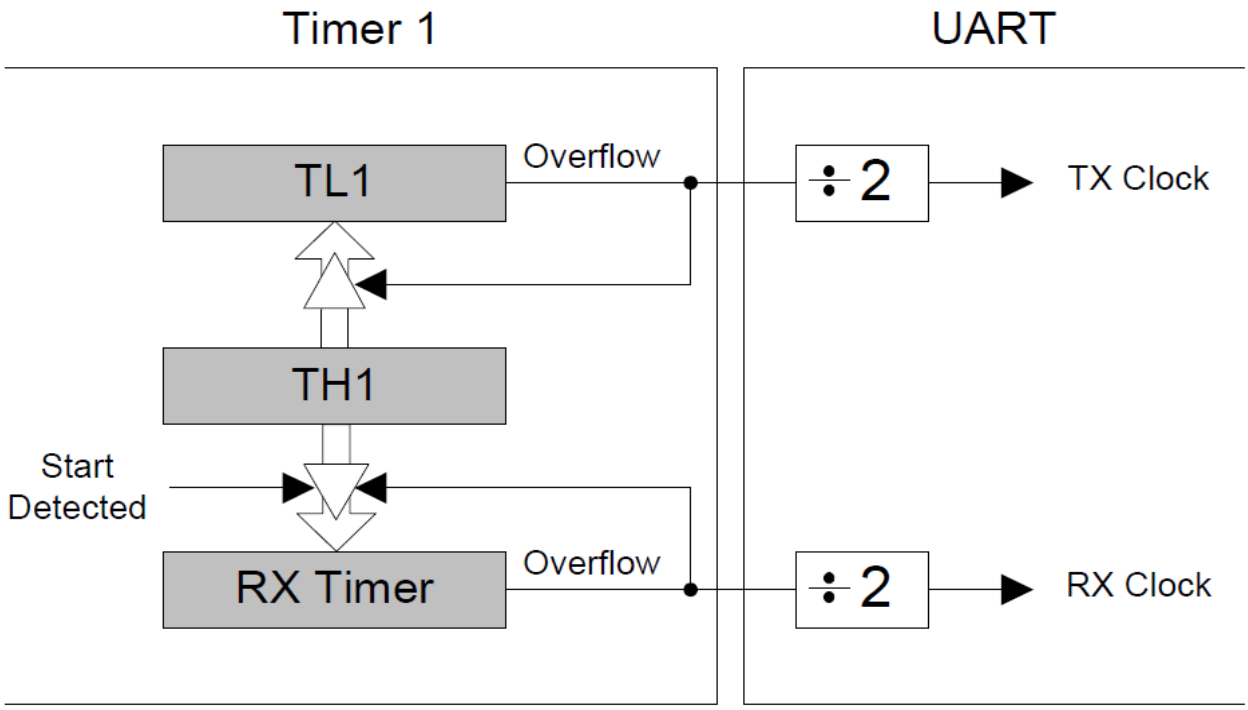


איור 8.9 עבודה בסביבה מרובת מעבדים

לכל רכיב עבד יש כתובת משלו. בעזרת הביט ה 9 (הנקרא TB8 ברגיסטר ה SCON0) המסטר מודיע האם הוא שולח נתון או כתובת. כאשר בביט ה 9 יש '0' זה אומר ש 8 הביטים המשודרים הם נתון. כאשר המסטר שם בביט ה 9 '1' זה אומר ש 8 הביטים הבאים הם של כתובת. לכל עבד יש כתובת משלו . בתחילת התקשורת המסטר שולח כתובת (ביט 9 הוא 1). רק העבד שזוהי הכתובת שלו ייצור קשר עם המסטר.

8.2.5 UART Baud Rate Generator - מחולל קצב הבאוד של ה UART

פולסי השעון הקובעים את קצב התקשורת של ה UART מתקבלים מטיימר 1 שעובד באופן של טעינה חוזרת אוטומטית של 8 ביט. באופן זה טיימר TL1 הוא הסופר ובסיום הספירה הוא נטען לערך שיש ב TH1 . באיור הבא רואים את יצירת קצב הבאוד.



איור 8.10 יצירת קצב הבאוד .

תדר השעון של TX נוצר על ידי TL1 . תדר השעון של RX נוצר על ידי רגיסטר שנקרא באיור RX Timer . זהו רגיסטר שיש בו את התדר שיש ב TL1 . לרגיסטר זה לא ניתן לגשת . גם תדר ה TX וגם ה RX מחולקים ב 2 ויוצרים את קצב התקשורת TX Clock ו RX Clock . הטיימר RX Timer מאופשר ומשתמש באותו ערך של טעינה חוזרת שיש ב TH1 אבל הוא טוען את הערך שיש ב TH1 רק כאשר יש ירידה בהדק ה RX Start Detected – התגלה אפס - כדי להיות מסונכרן עם הירידה הראשונה של קו הקליטה מ 1 ל 0 האומרת שמתחילה קליטה ולא להיות קשור למצב של השידור .

כפי שהזכרנו קודם, טיימר 1 הוא זה שמספק את פולסי ההפעלה ל UART . את טיימר 1 מפעילים באופן 2 שהוא טעינה חוזרת אוטומטית שבה TL1 בלבד הוא המונה ובסיום הספירה הוא נטען לערך שב TH1 . המשתמש יכול לקבוע מאיפה יגיעו פולסי הספירה לטיימר 1 (פנימי או חיצוני) ואת תדר הפולסים. ישנן 6 אפשרויות:

- תדר המתנד הפנימי של שעון המערכת הנקרא SYSCLK.
- תדר SYSCLK אחרי חלוקה ב 4 .
- תדר SYSCLK אחרי חלוקה ב 12 .
- תדר SYSCLK אחרי חלוקה ב 48 .
- תדר שעון ממתנד חיצוני.
- תדר שעון ממתנד חיצוני אחרי חלוקה ב 8 .

יש לטעון את TH1 בערך שייצור גלישה (מעבר של TL1 מ FFH לערך שב TH1) של 2 פעמים קצב התקשורת הרצוי או במילים אחרות קצב התקשורת/באוד צריך להיות חצי מקצב הגלישה . אם נסמן את תדר הפולסים לספירה של TL1 ב $T1_{CLK}$, את קצב הגלישה נסמן כ T1_Overflow_Rate ואת הוא קצב התקשורת/באוד ב UARTBaudRate נקבל את הנוסחאות הבאות:

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

קצב הגלישה של TL1 תלוי בתדר הספירה שלו חלקי כמות הפולסים שהוא סופר בין גלישה לגלישה (256-TH1) כאשר TH1 הוא הערך שטוענים ל TH1 נקבל את קצב הגלישות בעזרת משוואה B :

B) $$T1\_Overflow\_Rate = \frac{T1_{CLK}}{256 - TH1}$$

שינוי נושא בנוסחה שבמשוואה B ייתן לנו את הערך שיש לטעון ל TH1 .

$$\mathbf{256 - TH1 = T1_{CLK} \ / \ T1\_Overflow\_Rate}$$

או :

C) $$\mathbf{TH1 = 256 - T1_{CLK} \ / \ T1\_Overflow\_Rate}$$

אם נציב את **T1_Overflow_Rate** שבמשוואה A בנוסחה C נקבל :

D) $$\mathbf{TH1 = 256 - T1_{CLK} \ / \ (2*UARTBaudRate)}$$

דוגמה:

ידוע שעובדים עם המתנד הפנימי ו SYSCLK=48MHz . נבחר יחס חלוקה של 4 . רוצים קצב תקשורת של 57600 ביטים בשנייה . מהו ערך הטעינה שנטען את TH1 ?

פתרון :

אם בחרנו יחס חלוקה של 4 אז התדר המגיע לספירה ונקרא $T1_{CLK}$ שווה: 48MHz / 4= 12MHz

נציב בנוסחה D את הנתונים ונקבל (המספר לא יוצא שלם ולכן נעגל למספר הקרוב ביותר):

$$TH1 = 256 - 12*10^6 / 2*57600 = 256 - 104.1666 = = \sim 152 = 98H$$

הערה : כדאי לשים לב שלא קיבלנו מספר שלם ועשינו עיגול של התוצאה ויש סטייה קלה מהתדר הרצוי . במקרה כאן במקום קצב של 57600 נשתמש בנוסחה D ונקבל קצב של : $12*10^6/ \ 2* (256 -152) = 57,692.3$ bps .

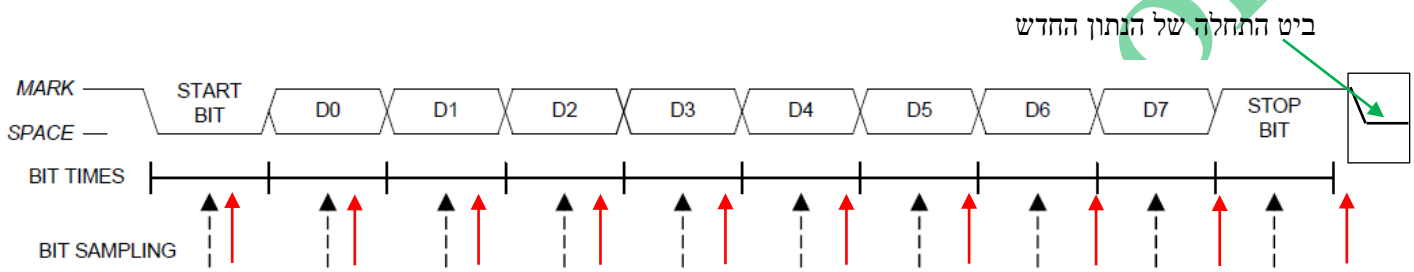
סטיית התדר היחסית כאן היא של : (57692-57600) / 57600 = 0.001597 שהיא 0.1597% .

8.2.6 שגיאות תקשורת בגלל הבדל תדר בין המשדר והמקלט

האם כתוצאה מסטיית התדר הקטנה שבסעיף הקודם בין המשדר למקלט תהיה בעיית תקשורת שגויה ? התשובה היא לא . במידה ותהיה סטיית תדר גדולה מגבולות שנחשב בהמשך - אז כן . עבור סטייה כזו קטנה של כ 0.16% לא תהיה טעות בתקשורת. הסיבה לכך היא שהסטייה איננה מצטברת אלא מתואמת לכל תחילת ביט התחלה חדש. בכל פעם שמגיע ביט סיום יגיע אחריו ביט ההתחלה של הנתון החדש . ביט התחלה מתחיל כאשר יש ירידה בקו מ 1 ל 0 ומרגע זה מתחילה קליטה חדשה.

נסביר זאת בעזרת הדוגמה הבאה ונחשב מהי הסטייה בין קצב השידור וקצב הקליטה המותר כדי שלא נקבל שגיאת תקשורת.

באיור הבא מתואר שידור נתון של 8 ביט. סה"כ משודרים עם ביט התחלה וסיום – 10 ביטים .



איור 8.11 שידור 8 ביטים בתקשורת טורית.

הקליטה מתחילה כאשר יש ירידה של הקו מ 1 ל 0 זהו ה start bit – ביט התחלה. המקלט ממתין זמן של חצי ביט , בודק שוב שיש 0 בקו (בודק שה 0 הקודם לא היה רעש) ומתחיל לדגום את הקו כל זמן של ביט בין דגימה לדגימה. הדגימות הן במרכז הביט כי שם אין תופעות מעבר של רעשים כתוצאה מזמני עלייה או ירידה. נניח שתדר המקלט נמוך מתדר המשדר (זמן בין ביט לביט גדול יותר) . בעזרת החיצים האדומים הראנו שבכל ביט יש תזוזה קלה של הדגימה מהמרכז. כל עוד הבדל התדר לא גורם לשגיאה

בקריאה של ביט הסיום לא תהיה בעיית קליטה כי בביט ההתחלה החדש תתחיל דגימה חדשה . באיור רואים שאם החץ האדום הימני היה זז עוד קצת ימינה אז הדגימה הייתה של 0 ולא של 1 וזו שגיאת קליטה.

כמובן שדבר דומה קיים אם תדר הקליטה קטן מתדר השידור. במקרה כזה החיצים האדומים יופיעו משמאל לחיצים השחורים וגם כאן יש לדאוג שאחרי 10 דגימות לא נדגום את ביט D7 כאילו הוא ביט הסיום.

מכאן ברור שהפרש הזמן – נסמן ב dt בין הביט של השידור שנסמן ב Tx ובין זמן הביט במקלט שנסמן ב Tr כפול 10 ביטים חייב להיות קטן מזמן של חצי ביט שידור.

$(Tx-Tr) * 10 < 0.5*Tx$

$Tx-Tr < 0.05*Tx$

כלומר הפרש זמן (או תדר) קטן מ 5% בין תדר השידור והקליטה יאפשר עדיין קליטה ללא שגיאות.

במידה ויש תקשורת עם 11 ביטים (9 ביטים של נתון + ביט התחלה + ביט סיום) אז ההפרש יהיה :

$(Tx-Tr)*11 < 0.5 *Tx$

$Tx-Tr < 0.045 * Tx$

כלומר הפרש זמן (או תדר) קטן מ 4.5% בין תדר השידור והקליטה יאפשר עדיין קליטה ללא שגיאות.

דוגמה:

תדר השידור הוא 57600 ביטים בשנייה עבור תקשורת של 8 ביטים. מה הם גבולות תדר הקליטה כדי שלא תהיינה שגיאות בגלל התדרים השונים.

פתרון :

ראינו שבשידור של 8 ביטים מותרת שגיאה של 5% . מכאן : 57600 * 5% = ±2,880 כלומר תדר המקלט יכול להיות קטן מ : 57600 + 2880 = **60480** וגדול מ : 57600-2880 = **54720** ועדיין לא נקבל שגיאות קליטה.

בטבלה שבהמשך חוסכים מאתנו את החישובים כדי לדעת איזה ערך טעינה יש להעביר ל TH1 . בנוסף לכך מתארים מה צריך להעביר לרגיסטר CKCON ClocK CONtrol - בקרת השעון כדי שיגיעו פולסים בקצב הרצוי . נזכיר כיצד נראה הרגיסטר:

Bit	7	6	5	4	3	2	1	0
Name	T3MH	T3ML	T2MH	T2ML	T1M	T0M	SCA[1:0]	

T1M=1 – ללא חלוקת תדר. T1M=0 אז חלוקה לפי SCA[1:0] : 00- 12 , 01 - 4 , 10 –48 , 11 – חלוקת תדר היצוני ב 8 .

האיור הבא מראה את הטבלה :

	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

Note: SCA1-SCA0 and T1M define the Timer Clock Source. X = Don't care

איור 8.10 : קביעת ערך הטעינה ל TH1 עבור קצבי שידור סטנדרטיים ועבודה עם המתנד הפנימי SYSCLK .

נסביר את העמודות בטבלה משמאל לימין.

העמודה השמאלית ביותר אומרת מהו תדר המתנד הפנימי SYSCLK .

Target Baud Rate - מתארת את קצב תקשורת המטרה - הרצוי .

Actual Baud Rate – הקצב המעשי שמקבלים (בדוגמה בסעיף קודם ראינו שיש הבדל בין תדר המטרה הרצוי והתדר המעשי שהמערכת מוציאה).

Baud Rate Error – שגיאת קצב הבאוד. – הסטייה באחוזים בין התדר הרצוי – המטרה והתדר המעשי .

Oscillator Devide Factor - מקדם חלוקת תדר המתנד - בכמה לחלק את תדר ה SYSCLK .

Timer Source Clock – מקור השעון של הטיימר – מאיפה מגיעים פולסי הספירה לטיימר .

***SCA1 – SCA0 pre scale Select** - מה צריך לשים בביטים SCA1- SCA0 שברגיסטר בקרת השעון CKCON – ClocK CONtrol שהוסבר בפרק על הטיימרים (7.2.3 - פרק 7 סעיף 2.3) . 2 הביטים קובעים את יחס חלוקת השעון .

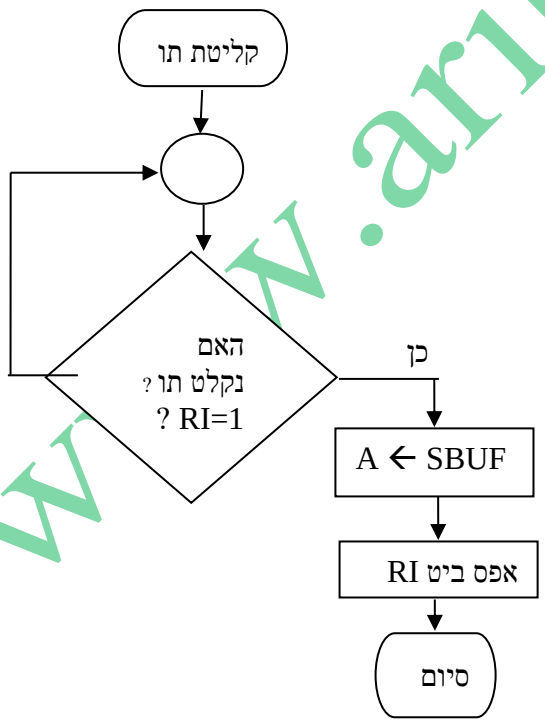
T1M - מה צריך לשים בביט T1M שברגיסטר CKCON - הביט קובע את מקור השעון. .אם המתכנת שם בביט 0 אז הפולסים מגיעים לאחר חלוקה של שעון המערכת לפי הביטים 0 ו 1 (SCA1- SCA0). אם שמים בביט 1 אז הפולסים הם שעון המערכת ללא חלוקה.

Timer 1 Reload Value (hex) - ערך טעינה לטיימר 1 (ליתר דיוק ל TH1) בהקסה דצימלי.

8.3 תוכנה :

8.3.1 קליטה של תו

כדי לבדוק האם נקלט תו נעזר באיור תרשים המלבנים הבא המתאר פרוצדורה / פונקציה לקליטת תו טורי:



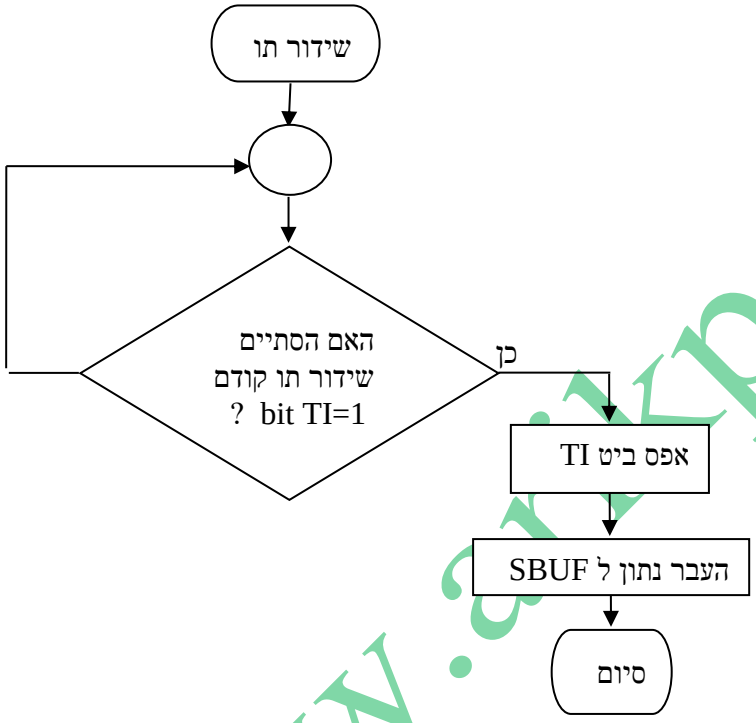
איור 8.12 : תרשים זרימה של קליטת ביית טורי

בודקים את הביט RI המציין שנקלט נתון טורי. אם בביט יש 0 זה אומר שלא נקלט תו. אם בביט יש 1 זה אומר שנקלט תו חדש. במקרה כזה "מושכים" את התו מה SBUF של הקליטה ומאפסים את ביט RI כדי שיעלה ל 1 שיגיע תו חדש נוסף. נרשום תכנית באסמבלי מצד שמאל ובשפת C51 מימין (נניח שהוכללו הספריות המתאימות והוגדר משתנה char myReceivedData) :

```
receiveTav:                          void  recieveTav( ) {
        jnb ri,$                       while (RI==0);
        Mov a, sbuf                    myReceivedData=SBUF;
        Clr ri                         RI=0
        Ret                            }
```

8.3.2 שידור של תו

נבדוק האם הסתיים שידור התו הטורי :



איור 8.13 תרשים זרימה של שידור ביית טורי .

הסבר : בודקים האם הסתיים שידור התו הקודם בעזרת ביט TI. אם לא (TI = 0) חוזרים שוב לבדוק את הביט. ברגע שה UART סיים לשדר את התו הוא מעלה את הביט ל 1 ואז מנקים את הביט ושולחים ל SBUF ביית חדש לשידור . נרשום תכנית באסמבלי מצד שמאל ובשפת C51 מימין (נניח שהוכללו הספריות המתאימות והוגדר משתנה char myTransmittedData) :

פרוצדורת השידור תיראה :

```
sendTav:                             void sendTav ( )      {
        jnb ti,$                       while(TI=0);
        Clr ti                         TI=0;
        Mov sbuf,a                     SBUF=myTranmittedData;
        Ret                            }
```

8.4. תרגילי דוגמה

8.4.1 תרגיל 1 :

רשום פקודות לעבודה בתקשורת טורית בין 2 מערכות מבוקרות מיקרו, בקצב תקשורת של 9600bps , ללא ביט תשיעי. תדר SYSCLK הוא 24MHz . עבוד עם שאילתה (POLLING). הדק Tx0 מתחבר ל P0.4 והדק Rx0 ל P0.5 .

פתרון : נעבוד לפי השורה השישית בטבלה שבפרקים הקודמים עבור תדר שעון SYSCLK =48MHz

9600	9615	0.16%	2496	SYSCLK / 12	00	0	0x98

Bit	7	6	5	4	3	2	1	0
Name	S0MODE	-	MCE0	REN0	TB80	RB80	TI0	RI0

Bit	7	6	5	4	3	2	1	0
Name	T3MH	T3ML	T2MH	T2ML	T1M	T0M	SCA[1:0]	

Bit	7	6	5	4	3	2	1	0
Name	GATE1	C/T1	T1M[1:0]		GATE0	C/T0	T0M[1:0]	

```
Mov scon0,#00010010B;  8 ביט קצב תקשורת משתנה, אפשור קליטה
; שמים 1 ב TI0 עבור תו השידור הראשון. אומרים שתו קודם כבר שודר
Mov ckcon,# 00000000B ;
```

T1M=0 חלוקת התדר לפי SCA[1:0] : 00 - 12 , 01 - 4 , 10 – 48 , 11 – חלוקת תדר חיצוני ב 8 .

```
Mov tmod,#20h;          טיימר 1 באופן 2
Mov th1,#98h ;  9600 לקצב שידור
Mov tl1,#98h
Setb tr1                ;  הרצת הטיימר
Mov xrb0,#00000001B  ; P0.4  P0.5  חיבור הדקי  Tx0 Rx0  להדקי
mov xbr1,#40h; enable crossbar אפשור הקרוס בר
```

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

אותה התוכנית בשפת C51 (בהנחה שהוכללו הספריות המתאימות) :

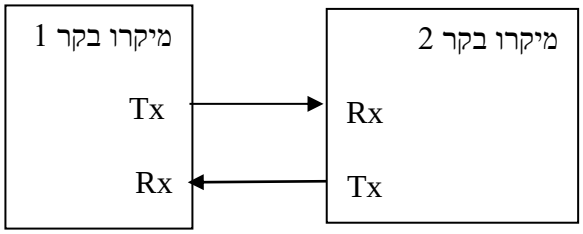
```
SCON0 = 0x10;
CKCON = 8;
TMOD = 0x20;
TH1 = TL1 = 0x98;
TR1=1;
XBR0=1;
XBR1=0x40;
```

8.4.2 תרגיל 2 :

חבר בין 2 מערכות מיקרו בקרים בעזרת תקשורת טורית. התקשורת היא FULL DUPLEX תקשורת של 8 ביט בקצב 115200 סל"ש . SYSCLK = 48MHz בשתי המערכות. א. צייר את החומרה הרלוונטית לתרגיל. ב. יש לרשום פרוצדורה / פונקציה עבור אחד מהמיקרו בקרים שתאתחל את תקשורת ה UART . ג. יש לרשום פרוצדורה / פונקציה שתשדר בלוק/מערך נתונים מכתובת 50H ועד 5FH ד. יש לרשום פרוצדורה / פונקציה שתקלוט את בלוק /מערך התווים ותשמור עלי החל מכתובת 60H ועד 6FH .

פתרון :

א. חיבור בין 2 המיקרו בקרים.



איור 8.13 : חיבור בין 2 המיקרו בקרים בתקשורת טורית.

ב. אתחול התקשורת הטורית . תחילה באסמבלי ואחר כך בשפת C51 .

ניעזר בשורה השנייה שבטבלה עבור SYCLK=48MHz הנראית כך:

115200	115385	0.16%	416	SYSCLK	XX	1	0x30
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)**

באסמבלי :

```
initUart:
        mov xrb0,#00000001B  ;   P0.4  P0.5  להדקי  Tx0 Rx0  חיבור הדקי
        mov xbr1,#40h;     enable crossbar אפשור הקרוס בר
        mov scon0,#0001000B;  8 ביט קצב תקשורת משתנה, אפשור קליטה
        mov ckcon,# 00001000B ;  T1M = 1 ,  SCA1-SCA0  אין חלוקת תדר ולא משנה מה יש בביטים
        mov tmod,#20h;          טיימר 1 באופן 2
        mov th1,#30h  ; 115200  לקצב שידור
        mov tl1,#30h
        setb tr1 ;  1 הרצת טיימר
        ret
```

בשפת C51 (בהנחה שהוגדרו הספריות המתאימות).

```
void initUart( )
{
  XBR0 = 1;
  XBR1 = 0x40;
  SCON0 = 0X10;
  CKCON = 0X8;
```

```
  TMOD = 0x20;
  TH1 = TL1 = 0x30;
  TR1 = 1;
}
```

ג. הפרוצדורה באסמבלי

```
sendBlock:
                mov r0,#50h ;   כתובת תחילת הבלוק
again:          mov a,@r0       ; העברת הנתון מהזיכרון לאקומולטור
                lcall transmit  ; קרא לפרוצדורת השידור
                Inc r0          ;  הגדלת המצביע שיראה על הכתובת הבאה
                Cjne r0,#60h,again     ;  ? האם הגענו לסוף בלוק הנתונים
                ret             ;  לולאה אין סופית
Transmit:       jnb ti0,$               ; ? האם הסתיים שידור תו קודם
                clr ti0         ;  ניקוי הביט שמראה שהסתיים שידור קודם
                mov sbuf0,a     ; העברת הנתון לשידור
                ret             ;  חזור
```

פונקציית השידור בשפת C51 .

```
char data arrayT [0x10] _at_ 0x50 ;     // הגדרת מערך בתים המתחיל בכתובת 50H ב RAM הפנימי
unsigned char  x;  //  משתנה גלובאלי כדי שיוכר בפונקציות שמתחתיו
void transmit( ); // הצהרה על פונקציה שמופיעה מתחת השורה הקוראת לפונקציה כדי שתוכר על ידי הקומפיילר.
void sendArray( )       //   פונקציית שידור המערך
{
  for (x=0;x<0x10;x++)          // לולאה המתבצעת 10H פעמים
          transmit( );      //  זימון/קריאה לפונקציית שידור תו אחד.
}
void transmit( )     //  פונקציית שידור של תו אחד.
{
  while (TI0==0);   //    ממתינים לסיום השידור הקודם. בסיום שידור הנתון הביט עולה ל 1
  TI0=0;  // איפוס ביט TI כדי להכין אותו לשידור לשידור נתון חדש
  SBUF0=arrayT[x];   // שולחים ל SBUF נתון מהמערך והוא מיד משודר על ידי ה UART
}
```

ד. פונקציית הקליטה באסמבלי :

```
ReceiveBlock:
            mov r1,#60h ;   כתובת תחילת הבלוק
again:      lcall receive   ; קרא לפרוצדורת השידור
            mov @r1,a       ; העברת הנתון מהאקומולטור לזיכרון
            inc r1          ;  הגדלת המצביע שיראה על הכתובת הבאה
            cjne r1,#70h,again      ;  ? האם הגענו לסוף בלוק הנתונים
            ret             ;  לולאה אין סופית
Receive :   jnb ri0,$                   ; ?  האם הסתיימה קליטת תו
            mov a,sbuf0             ; העברת הנתון הנקלט לאקומולטור
            clr ri0                 ;  ניקוי הביט שמראה שהסתיים קליטת תו
            ret                     ;  חזור
```

ובשפת C51 :

```
char data arrayR [0x10] _at_ 0x60 ;     // הגדרת מערך בתים המתחיל בכתובת 60H ב RAM הפנימי
unsigned char  x;  //  משתנה גלובאלי כדי שיוכר בפונקציות שמתחתיו
void receive( ); // .(הצהרה על פונקציית קליטת תו שמופיעה מתחת השורה הקוראת לפונקציה (כדי שתוכר על ידי הקומפיילר
void receiveArray( )   //   פונקציית קליטת המערך
{
  for (x=0; x < 0x10 ; x++)     // לולאה המתבצעת 10H פעמים
          receive( );       //  .זימון/קריאה לפונקציית קליטת תו אחד
}
void receive( )    //  .פונקציית קליטה של תו אחד
{
  while (RI0==0);   //    ממתינים לסיום השידור הקודם. בסיום שידור הנתון הביט עולה ל 1
  RI0=0;  // איפוס ביט TI כדי להכין אותו לשידור לשידור נתון חדש
  arrayR[x] = SBUF0; //  .מעבירים את הנתון שנקלט למערך במיקום המתאים
}
```

UART1 8.5

UART1 לא מופיע בתוכנית הלימודים אבל נכתוב עליו מספר משפטים.

- UART1 בדומה ל UART0 הוא פורט טורי , אסינכרוני העובד ב FULL DUPLEX ומציע מגוון אפשרויות של פורמט נתונים. יש לו מחולל קצב תקשורת משלו עם טיימר של 16 ביט ואפשרויות חלוקת תדר שונות כך שניתן לקבל קצבי תקשורת רבים. יש לו FIFO (חוצץ זיכרון העובד לפי העיקרון של הראשון שנכנס יוצא ראשון – First In First Out) היכול לקבל 3 בתים של נתונים לפני שהנתונים אובדים וקורית גלישה. במילים אחרות : הוא יכול לקלוט 3 בתים מבלי שהמיקרו יבצע קריאה שלהם . בביית הנתון הרביעי שייקלט מבלי שהמיקרו "משך" את הנתונים הקודמים נתחיל לאבד את הנתונים הקודמים שנקלטו.
- ל UART1 יש 6 רגיסטרים הנמצאים ב SFR :
  3 משמשים ליצירת קצב התקשורת .
  2 משמשים לפורמי נתונים , בקרה וסטאטוס.
  1 משמש לקליטה או שידור נתון (למעשה זה 2 רגיסטרים אחד לשידור ואחד לקליטה והקריאה או הכתיבה מזהה
  למי מהם פונים (כמו ב UART0) .
- כתיבה ל SBUF1 ניגשת לרגיסטר אחיזת שידור - TRANSMIT HOLDING .
- קריאה מ SBUF1 ניגשת תמיד אל הביית הראשון ב FIFO של הקליטה.
- גם עם UART1 ניתן לעבוד עם פסיקות והן קורות בכל סיום שידור של ביית או בסיום קליטה של ביית. גם כאן יש 2 ביטים TI1 ו RI1 שנמצאים ב SCON1 ומראים שהסתיים שידור או הסתיימה קליטה. כתובת הפסיקה היא 0X83 בזיכרון התוכנית והיא פסיקה מספר 16 (כולל reset).
- גם כאן הביטים TI ו RI אינם מאופסים בחומרה כאשר נענים לפסיקה ויש לאפס אותם בתוכנה.