საქართველოს ტექნიკური უნივერსიტეტი

ტექნიკური ელექტრონიკის კათედრა

სერგო დადუნაშვილი
მაია ცერცვაძე

მიკროპროცესორული ტექნიკა

(ლექციების კონსპექტი)

თბილისი 2005

PIC16F8x მიკროკონტროლერები

ზოგადი დახასიათება

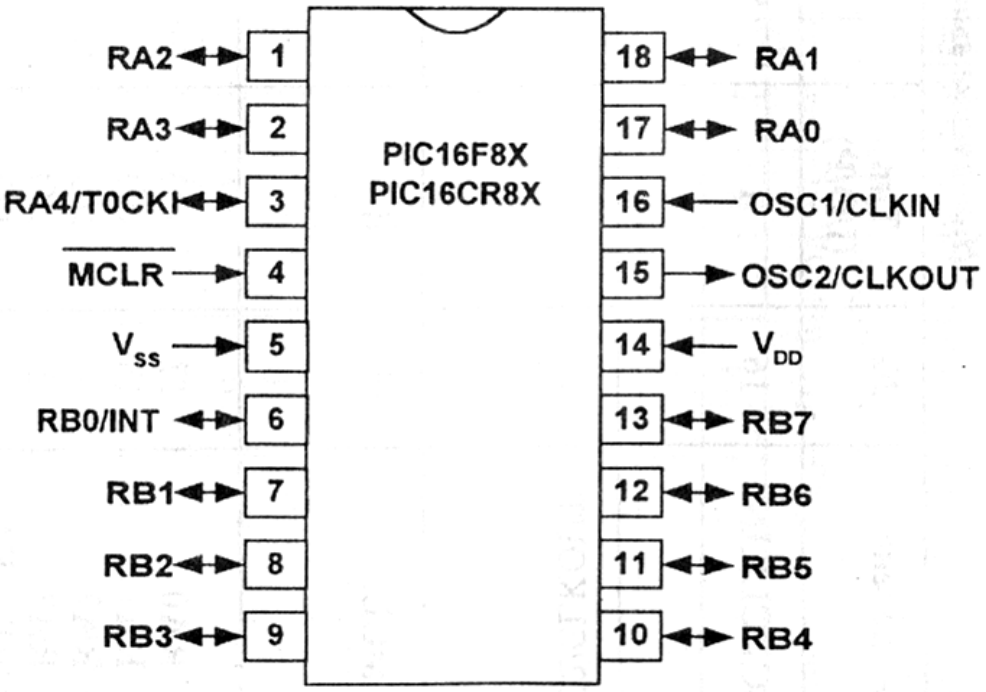
PIC16F8x მიკროკონტროლერები, ისევე როგორც სხვა მიკროკონტროლერები სავაჭრო ნიშნით **PICmicro™** დაფუძნებულია განვითარებულ RISC არქიტექტურაზე. მათ გააჩნიათ ბირთვის გაფართოებული ოფციები, რვადონიანი სტეკი და განსხვავებული შიგა და გარე წყვეტები. ბრძანებების 14-ბიტიანი სიტყვები და 8-ბიტიანი მონაცემები გადაიცემა ერთმანეთისგან დამოუკიდებლად, მეხსიერებისა და მონაცემების განცალკევებული სალტეებით. ბრძანებათა უმეტესობა სრულდება ერთი მანქანური ციკლის განმავლობაში, რომლის ხანგრძლივობაა 400 ნწმ 10 მგჰც ტაქტური სიხშირის დროს (გამონაკლისს შეადგენენ გადასვლის ბრძანებები, რომლებიც სრულდება ორი ციკლის განმავლობაში, 800 ნწმ 10 მგჰც-ის დროს). ბრძანებათა კრებული შედგება 35 ინსტრუქციისაგან (ბრძანებისაგან) ინტუიციურად განვითარებული მნემონიკით.

PIC16F8x მიკროკონტროლერები ჩვეულებრივ იძლევიან საშუალებას შემცირდეს პროგრამის მოცულობა თანაფარდობით 2:1 და გაიზარდოს მისი შესრულების სისწრაფე თანაფარდობით 4:1 ამ კლასის 8-ბიტიანი სხვა მიკროპროცესორების უმრავლესობასთან შედარებით.

კრისტალზე განლაგებულია კონსტანტების EEPROM მეხსიერების 64 ბაიტი მონაცემების შენახვის გარანტირებული 40 წელზე მეტი ვადით გამორთული კვების დროს, მონაცემების მეხსიერების (მონაცემების შენახვის მუშა რეგისტრები) 64 ბაიტამდე. კრისტალები მზადდება 4 მგჰც და 10 მგჰც მაქსიმალური ტაქტური სიხშირეებით, გააჩნიათ შეტანა/გამოტანის 13 პორტი, ჩადგმული ტაიმერ/მთვლელი TMR0, მოთვალთვალე ტაიმერი WDT, ჩაძინების ეკონომიური რეჟიმი SLEEP.

ტაქტური სიხშირე შეიძლება მიეწოდოს RC წრედის საშუალებით, იაფი კერამიკული რეზონატორის ან კვარცის რეზონატორის საშუალებით. შეიძლება აგრეთვე ტაქტური სიხშირის გარე გენერატორის მიერთებაც. დაუშვებელია პროცესორის მატაქტირებელი ჩადგმული გენერატორის გამოყენება. კრისტალში ჩადგმული RC გენერატორიდან შეიძლება მხოლოდ TMR0 ტაიმერ/მთვლელის ან მოთვალთვალე ტაიმერის ტაქტირება არჩევანის მიხედვით.

PIC16F8x მიკროკონტროლერების პროგრამირება შეიძლება უშუალოდ მოწყობილობის პლატაზე, რაც საშუალებას გვაძლევს გავაწყოთ პროგრამა ან ჩავწეროთ კონსტანტები და დაკალიბრების მონაცემები. პლატაზე პროგრამირებისათვის აუცილებელია მაქსიმუმ ხუთი მავთულის არსებობა: 5ვ კვებისთვის, პროგრამირების ძაბვისთვის, მიმდევრობითი მონაცემებისთვის, მატაქტირებელი იმპულსებისა და მიწისთვის. პროგრამების მეხსიერება მხოლოდ ჩადგმულია.



ნახ. 1. **PIC16F8x** მიკროკონტროლერის გამოსასვლელთა განლაგება.

პარამეტრი	PIC16F83	PIC16CR83	PIC16F84	PIC16CR84
მაქს. ტაქტური სიხშირე, მჰც	10	10	10	10
პროგრამების ფლეშ-მეხსიერება	512	-	1K	-
პროგრამების EEPROM მეხსიერება	-	-	-	-
პროგრამების ROM მეხსიერება	-	512	-	512
მონაცემების მეხსიერება, ბაიტი	36	36	68	68
მონაცემების EEPROM მეხსიერება, ბაიტი	64	64	64	64
ტაიმერის მოდული	TMR0	TMR0	TMR0	TMR0
წყვეტის წყაროები	4	4	4	4
შეტანა/გამოტანის ხაზები	13	13	13	13
მკვებავი ძაბვების დიაპაზონი	2.0-6.0	2.0-6.0	2.0-6.0	2.0-6.0

გამომყვანების განლაგება და დანიშნულება

განსახილველი კონტროლერების გამომყვანთა განლაგება ნაჩვენებია ნახაზზე 1-1. კორპუსის ვარიანტებია 18-pin DIP, SOIC.

გამომყვანების დანიშნულება მოკლედ აღწერილია ცხრილში 1-1.

აღნიშვნები. *I-შესასვლელი, O-გამოსასვლელი, P-კვების გამომყვანი, TTL -სტანდარტული TTL შესასვლელი, ST-შესასვლელი შმიტის ტრიგერით*

შენიშვნები.

1. შმიტის ტრიგერი შესასვლელზე გამოიყენება მხოლოდ მაშინ, თუ გამოსასვლელი კონფიგურირებულია, როგორც გარე წყვეტის შესასვლელი.

2. შმიტის ტრიგერი შესასვლელზე გამოიყენება მხოლოდ კრისტალის მიმდევრობითი პროგრამირების რეჟიმში.

3. შმიტის ტრიგერი შესასვლელზე გამოიყენება მხოლოდ მაშინ, თუ ჩართულია RC გენერატორის რეჟიმი, სხვა შემთხვევებში როგორც SMOS შესასვლელი.

აღნიშვნა	**№**	**ტიპი (I/O/P)**	**ბუფერის ტიპი**	**დანიშნულება**
OSC1/CLKIN	16	I	ST/CMOS(3)	შესასვლელი კვარცული რეზონატორის, ან RC წრედის მისაერთებლად ან გარე ტაქტური გენერატორის შესასვლელი
OSC2/CLKOUT	15	O	-	გამოსასვლელი კვარცული რეზონატორის მისაერთებლად კვარცთან მუშაობის რეჟიმში, RC გენერატორის რეჟიმში გამოსასვლელზე ადგილი აქვს იმპულსებს სიხშირით 1/4-დან OSC1-მდე
MCLR	4	I/P	ST	ჩამოყრა დაბალი დონის მიხედვით შესასვლელზე. კრისტალის პროგრამირებისას– პროგრამირების ძაბვის შესასვლელი
RA0 RA1 RA2 RA3 RA4/T0CKI	17 18 1 2 3	I/O I/O I/O I/O I/O	TTL TTL TTL TTL ST	ორმხრივმიმართული A პორტის გამომყვანები RA4/T0CK1 შეიძლება იყოს აწყობილი როგორც იმპულსების შესასვლელი TMR0 ტაიმერ-

				მთვლელისთვის. გამოსასვლელი ღია შესართავით
RB0/INT RB1 RB2 RB3 RB4 RB5 RB6 RB7	6 7 8 9 10 11 12 13	I/O I/O I/O I/O I/O I/O I/O I/O	TTL/ST(1) TTL TTL TTL TTL TTL TTL/ST(2) TTL/ST(2)	ორმხრივმიმართული B პორტის გამომყვანები. პორტის გამომყვანებთან პროგრამულად შეიძლება მიერთებული იქნენ გარე რეზისტორები Vdd-დან. RB0/INT შეიძლება ასევე პროგრამულად იყოს აწყობილი როგორც გარე წყვეტის შესასვლელი. RB4…RB7 შეიძლება ასევე პროგრამულად იყოს აწყობილი როგორც წყვეტის შესასვლელები ამ შესასვლელებიდან ნებისმიერზე დონის ცვლილების მიხედვით. ცვლილების მიმართულება მოიცემა პროგრამულად. კრისტალის პროგრამირებისას შესასვლელი RB6 გამოიყენება როგორც ტაქტური, RB7 როგორც მონაცემების შესასვლელ/გამოსასვლელი
Vss	5	P	-	საერთო სადენი
Vdd	14	P	-	კვების დადებითი ძაბვა

ელექტრული პარამეტრები

ძირითადი პარამეტრები, რომლების ცოდნაც ჩვეულებრივ აუცილებელია PIC16F8x მიკროკონტროლერების ბაზაზე აგებული მოწყობილობების დამუშავებისა და ექსპლუატაციისათვის, შემდეგია:

პარამეტრი	მნიშვნელობა	შენიშვნა
მუშა ტემპერატურა T_A, °C	0 . . . +40 -40 . . . +85	კომერციული შესრულება ინდუსტრიული შესრულება
მაქსიმალური ტაქტური სიხშირე F_{osc}, მჰც	4 10	PIC16F83/84-4 PIC16F83/84-10
კვების ძაბვა V_{DD},ვ	2.0 . . . 6.0 4.0 . . . 6.0	PIC16LF83/84 PIC16F83/84-10
მოხმარებული დენი სტანდარტულ რეჟიმში I_{DD},მა PIC16F83/84-ისთვის	1.8 . . . 4,5 5 . . . 10	F_{osc}=4 მჰც, V_{DD}=5.5ვ F_{osc}=10 მჰც, V_{DD}=5.5ვ
მოხმარებული დენი სტანდარტულ რეჟიმში I_{DD},მა PIC16LF83/84-ისთვის	1 . . . 4 15 . . 45	F_{osc}=2 მჰც, V_{DD}=5.5ვ F_{osc}=32 კჰც, V_{DD}=2ვ WDT გამორთულია
მოხმარებული დენი SLEEP რეჟიმში I_{PD},მკა PIC16F83/84-ისთვის	7 . . . 28 1 . . . 16 1 . . . 14	V_{DD}=4ვ WDT ჩართ., ინდ. V_{DD}=4ვ WDT გამორთ., ინდ. V_{DD}=4ვ WDT გამორთ., კომმ
მოხმარებული დენი SLEEP რეჟიმში I_{PD},მკა PIC1LF83/84-ისთვის	3 . . . 16 0.4 . . . 9 0.4 . . . 7	V_{DD}=2ვ WDT ჩართ., ინდ. V_{DD}=2ვ WDT გამორთ., ინდ. V_{DD}=2ვ WDT გამორთ., კომმ
მაქსიმალური ჩამდინარე დენი ნებისმიერი გამომყვანისათვის, მა	25	შუქდიოდების მართვა დამატებითი ბუფერის გარეშე (მაგრამ რეზისტორით!)

მაქსიმალური გამომდინარე დენი ნებისმიერი გამომყვანისათვის, მა	20	შუქდიოდების მართვა დამატებითი ბუფერის გარეშე (მაგრამ რეზისტორით!)
წაშლა/ჩაწერის ციკლების რაოდენობა პროგრამების ფლეშ-მეხსიერებისათვის, არა უმეტეს	1000	
წაშლა/ჩაწერის ციკლების რაოდენობა პროგრამების მონაცემების მეხსიერებისათვის EEPROM, არა უმეტეს	10.000.000	

მეხსიერების ორგანიზაცია

PIC16F8x მიკროკონტროლერებში არსებობს მეხსიერების ორი ბლოკი – პროგრამების მეხსიერება და მონაცემების მეხსიერება. თითოეულ ბლოკს გააჩნია საკუთარი სალტე, ამგვარად თითოეულ ბლოკთან წვდომა ხდება ერთდროულად.

მონაცემების მეხსიერება, თავის მხრივ, გაყოფილია სპეციალურ რეგისტრებად და საერთო დანიშნულების რეგისტრებად (მომხმარებლის ოდმ). სპეციალური რეგისტრების გამოყენება ხდება მდგომარეობის ბიტების შესანახად, რომლებიც განსაზღვრავენ შეტანა/გამოტანის პორტების, ტაიმერების და კონტროლერის სხვა პერიფერიული მოდულების მუშაობას. დაწვრილებით თითოეული სპეციალური რეგისტრი აღწერილი იქნება შესაბამისი მოდულის განხილვისას.

ოდმ-ისა და სპეციალური რეგისტრების გარდა, მონაცემების მეხსიერების სივრცე შეიცავს უჯრედს EEPROM. მეხსიერების ეს არე არ შეიძლება დამისამართებული იქნას უშუალოდ, და მასთან წვდომა ხორციელდება ირიბი ადრესაციის EEADR რეგისტრის საშუალებით, რომელშიც ჩაიწერება უჯრედის რიგითი ნომერი. EEPROM-ის 64 ბაიტს აქვს ნომრები 00h-დან 3Fh-მდე. EEPROM ჩვეულებრივ გამოიყენება მუდმივების შესანახად, რომელთა მნიშვნელობები არ უნდა დაიკარგოს კვების გამორთვის დროს, მაგალითად მართვის კოდები, ინდივიდუალური ნომრები და სხვ. EEPROM-ის მნიშვნელოვან ღირსებას წარმოადგენს ის, რომ მისი მონაცემები შეიძლება შეცვლილ იქნას პროგრამის შეტანის შემდეგ ერთჯერადად პროგრამირებად კრისტალშიც კი.

მიუხედავად იმისა, რომ EEPROM გარანტირებულად უშვებს წაშლა-ჩაწერის 10 000 000 ციკლს, დამამზადებელი არ იძლევა მისი უჯრედის გამოყენების რეკომენდაციას ხშირად ცვლადი მნიშვნელობების შესანახად, ვინაიდან ამ დროს შეიძლება სწრაფად ამოიწუროს მისი რესურსი.

პროგრამული ცვლადების შესანახად უნდა გამოვიყენოთ ოდმ (საერთო დანიშნულების რეგისტრები).

პროგრამების მეხსიერება

PIC16F8x ჯგუფის მიკროკონტროლერებს გააჩნიათ 13-ბიტიანი პროგრამული მთვლელი, რომელიც იძლევა პროგრამების 8Kx14 მეხსიერების დამისამართების საშუალებას.. მიკროკონტროლერებში PIC16F83 და PIC16CR83 ფიზიკურად ხდება მეხსიერების მხოლოდ პირველი 512 (0000h-00FFh) უჯრედის დამისამართება. PIC16F84-სა და PIC16CR84-ში მიღწევადია მეხსიერების პირველი 1024 (0000h-03FFh) უჯრედი. უფროს უჯრედებთან მიმართვა, რომლებიც აღნიშნული დიაპაზონის ზღვარს გარეთ მდებარეობენ, თანაბარძალოვანია დიაპაზონის შიგნით შესაბამის მისამართებთან მიმართვისა, მაგალითად, PIC16F84-ისათვის მისამართები 30h, 430h, 830h, C30h, 1030h, 1430h, 1830h და 1C30h თანაბარძალოვანია და ახდენენ ერთი და იგივე ბრძანების ადრესაციას.

ჩამოყრის სტარტი ხდება 0000h მისამართიდან, წყვეტის ვექტორი ერთია და მოთავსებულია მისამართზე 0004h. ჩვეულებრივ 0004h მისამართზე ათავსებენ წყვეტის ამოცნობისა და დამუშავების ქვეპროგრამას, ხოლო 0000h მისამართზე – იმ ჭდეზე გადასვლის ბრძანებას, რომელიც მოთავსებულია წყვეტის დამუშავების ქვეპროგრამის შემდეგ.

მონაცემების მეხსიერება

PIC16F8x მიკროკონტროლერებში მონაცემების მეხსიერება გაყოფილია ორ ნაწილად – სპეციალური რეგისტრები და საერთო დანიშნულების რეგისტრები (მომხმარებლის ოდმ). ამის გარდა, მონაცემების მეხსიერება გაყოფილია ორ ბანკად. მონაცემების მეხსიერების ორგანიზაცია გრაფიკულად ნაჩვენებია ნახ. 1-2-ზე.

PIC16F83/CR83

მისამართი			მისამართი
00h	ირიბი მისამართი	ირიბი მისამართი	80h
01h	TMRO	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	მიუწვდომელია	მიუწვდომელია	87h

08h	EEDATA	EECON1	88h
09h	EEADR	EECON2	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	საერთო დანიშნულების 36 რეგისტრი (SRAM)	აისახება 0 ბანკის სივრცეზე	8Ch
2Fh			AFh
30h			B0h
7Fh	ბანკი 0	ბანკი 1	FFh

PIC16F84/CR84

მისამართი			მისამართი
00h	ირიბი მისამართი	ირიბი მისამართი	80h
01h	TMRO	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	მიუწვდომელია	მიუწვდომელია	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	საერთო დანიშნულების 68 რეგისტრი (SRAM)	აისახება 0 ბანკის სივრცეზე	8Ch
4Fh			CFh
50h			D0h
7Fh	ბანკი 0	ბანკი 1	FFh

ბანკების გადართვა ხდება STATUS რეგისტრის მე-5 ბიტის მოცემის საშუალებით. თუ ბიტში მოთავსებულია 0, ხდება ნულოვანი ბანკის დამისამართება, ხოლო თუ 1, შესაბამისად, პირველის. ზოგიერთი სპეციალური რეგისტრის ადრესაცია ხდება ჩართული ბანკისგან დამოუკიდებლად. მაგალითად, 03h მისამართზე მიმართვა ჩართული 0 ბანკის დროს და 83h მისამართზე მიმართვა ჩართული 1 ბანკის დროს ერთმნიშვნელოვნად გვაძლევს სპეციალურ STATUS რეგისტრთან მიღწევას. საერთო დანიშნულების რეგისტრები წარმოადგენენ სტატიკურ ოდმ-ს და შეიძლება დამისამართებული იქნენ უშუალოდ ან ირიბად, ირიბი ადრესაციის FSR რეგისტრის გამოყენებით. *ადრესაცია არ არის დამოკიდებული ბანკზე. ასე მაგალითად, მისამართები 0Eh და 8Eh ახდენს ოდმ-ის ერთი და იმავე უჯრედის ადრესაციას.*

სპეციალური რეგისტრები

სპეციალური რეგისტრები წარმოადგენენ სტატიკურ ოდმ-ს, რომლებშიც გაჩუმების მიხედვით ზოგიერთი ბიტი ყენდება ჩამოყრის დროს(იხ. ცხრილი 1-2).

აღნიშვნები: x – უცნობია, u – არ იცვლება, - -აპარატულად ყოველთვის იკითხება, როგორც “0”, q – მნიშვნელობა განისაზღვრება კონტექსტით.

შენიშვნა 1. პროგრამული მთვლელის უფროსი ბაიტი მიუწვდომელია პირდაპირ. PCLATCH-ის შიგთავსი შეიძლება გადატანილი იყოს პროგრამული მთვლელის უფროდ ბაიტში, მაგრამ პროგრამული მთვლელის შიგთავსი არასოდეს არ გადაიტანება PCLATCH-ში.

შენიშვნა 2. STATUS რეგისტრის TO და PD ბიტები არ იცვლება MCLR სიგნალის მიხედვით ჩამოყრისას.

შენიშვნა 3. ჩამოყრის სხვა ვარიანტებს განეკუთვნებიან გარე ჩამოყრა MCLR-ის გავლით და ჩამოყრა მოთვალთვალე ტაიმერის გადავსების მიხედვით.

რეგისტრი STATUS (მისამართი 03H, 81H)

რეგისტრი STATUS ინახავს ალმ-ის ალმებს, ინფორმაციას ჩამოყრის შესახებ და მონაცემთა მეხსიერების ბანკის რჩევის ბიტს. ისევე როგორც სხვა ნებისმიერი რეგისტრი, STATUS შეიძლება წარმოადგენდეს დანიშნულების რეგისტრს ნებისმიერი ოპერაციისათვის. მაგრამ თუ STATUS რეგისტრზე სრულდება ოპერაცია, რომელმაც შეიძლება გამოიწვიოს Z, DC და C ბიტების მდგომარეობის ცვლილება, მაშინ ხდება ამ სამ ბიტში ჩაწერის ბლოკირება. მათი მდგომარეობა იცვლება მხოლოდ აპარატურულად ალმ-ის მდგომარეობის მიხედვით. უფრო მეტიც, ბიტები TO და PD ასევე მიუწვდომელია წვდომისთვის. შედეგად STATUS რეგისტრთან ოპერაციის შედეგი მიმღების როლში შეიძლება აღმოჩნდეს არა ისეთი, როგორსაც მოველოდით. მაგალითად, ბრძანება CLEAR STATUS (მთლიანად გასუფთავდეს რეგისტრი STATUS) სინამდვილეში გაასუფთავებს

მხოლოდ სამ უფროს ბიტს და დააყენებს 1-ში Z ბიტს. ამიტომ რეგისტრ STATUS-თან უნდა გამოვიყენოთ მხოლოდ ბრძანებები BCF, BSF, SWAPF და MOVWF, იმიტომ რომ ისინი არ ცვლიან მდგომარეობის სხვა ბიტებს.

განვიხილოთ დაწვრილებით STATUS რეგისტრის ბიტების დანიშნულება:

bit7 **IRP** – მეხსიერების ბანკის არჩევის რეგისტრი, რომელიც გამოიყენება ირიბი ადრესაციის დროს. ეს ბიტი არ გამოიყენება PIC16F8x-ში და ყოველთვის უნდა დარჩეს ჩამოგდებული. იმისათვის, რომ მიღწეულ იქნას თქვენი პროგრამის შეთავსებადობა უფრო ახალ კონტროლერებთან, ნურასოდეს გამოიყენებთ ამ ბიტს თქვენს პროგრამაში.

0=bank 0,1 (00h-FFh)

1=bank 2,3 (100h-1FFh)

bit6-5 **RP1, RP0** – მეხსიერების ბანკის არჩევის რეგისტრი, რომელიც გამოიყენება პირდაპირი ადრესაციის დროს.

0=bank 0 (00h-7Fh)

01=bank 1 (80h-FFh)

10=bank 2 (100h-17Fh)

11=bank 3 (180h-1FFh)

ყოველი ბანკი შეიცავს 128 ბაიტს. ვინაიდან PIC16F8x-ში გამოიყენება მხოლოდ ორი ბანკი, 0 და 1, ამიტომ ბიტი **RP1** უნდა იყოს ყოველთვის ჩამოგდებული, ხოლო ბანკის ასარჩევად გამოიყენება ბიტი **RP0.**

bit4 **TO** – მოთვალთვალე ტაიმერის ამოქმედების ალამი. ყენდება 1 მდგომარეობაში კვების ჩართვისას ბრძანებებით CLRWDT და SLEEP. ყენდება 0-ში მოთვალთვალე ტაიმერის დაყოვნების დასრულებისას.

bit3 **PD** – მონაცემების შენახვის რეჟიმი. ყენდება 1 მდგომარეობაში კვების ჩართვისას ან ბრძანებით CLRWDT. ყენდება 0-ში ბრძანებით SLEEP.

bit2 **Z** – ნულოვანი შედეგის ალამი. ყენდება 1 მდგომარეობაში, თუ არითმეტიკული ან ლოგიკური ოპერაციის შედეგი ნულის ტოლია. ინარჩუნებს თავის მნიშვნელობას შემდეგ ოპერაციამდე.

bit1 **DC** – ათობითი გადატანის ალამი. გამოიყენება ბრძანებებისათვის ADDWF, ADDLW, SUBWF და SUBLW. ხდება შედეგის მეოთხე თანრიგიდან გადატანის თვალთვალი.

1=მოხდა გადატანა შეკრებისას

0=არ მოხდა გადატანა შეკრებისას

გამოკლება ალმ-ში ხორციელდება პირველი ოპერანდის შეკრებით მეორე ოპერანდის დამატებით კოდთან. ბიტის მნიშვნელობა კონკრეტულად დამოკიდებულია იმაზე, თუ რა ოპერაცია შესრულდა. გამოკლების ოპერაციისთვის ბიტის მნიშვნელობები ინვერტირებულია.

bit0 **C** – გადატანის ალამი. გამოიყენება ბრძანებებისათვის ADDWF, ADDLW, SUBWF და SUBLW. ხდება უფროსი თანრიგიდან ბიტში გადატანის თვალთვალი შეკრებისას.

1=მოხდა გადატანა შეკრებისას

0=არ მოხდა გადატანა შეკრებისას

გამოკლება ალმ-ში ხორციელდება პირველი ოპერანდის შეკრებით მეორე ოპერანდის დამატებით კოდთან. ბიტის მნიშვნელობა კონკრეტულად დამოკიდებულია იმაზე, თუ რა ოპერაცია შესრულდა. გამოკლების ოპერაციისთვის ბიტის მნიშვნელობები ინვერტირებულია. ასე მაგალითად,

00h-01h გამოკლების ოპერაციის შესრულებით ვღებულობთ FFh და C ბიტს, რომელიც დაყენებულია 1-ში. შედეგად ოპერაციის შედეგი უარყოფითია (C ბიტი გამოდის ნიშნის თანრიგის როლში და რიცხვი FFh დაყენებული ნიშნის თანრიგით ორობითი თანრიგის კანონებით წარმოადგენს მინუს ერთიანს). 01h-00h გამოკლების ოპერაციის შესრულებით ვღებულობთ 01h და C ბიტს, რომელიც დაყენებულია 0-ში. შედეგი დადებითია. FFh+02h შეკრების ოპერაციის შესრულებით ვღებულობთ 01h და C =1 ბიტს. მოხდა გადატანა, ე.ი. რეალური შედეგია 101 h.

ძვრის RRF და RLF ბრძანებების შესრულებისას ამ ბიტში ჩაიტვირთება შესაბამისად წყარო-რეგისტრის უმცროსი ან უფროსი ბიტი.

TO და **PD** ალმებით შეიძლება განისაზღვროს, რით იქნა გამოწვეული ჩამოყრა.

- x ბიტების მდგომარეობა არ შეცვლილა. ჩამოყრა MCLR შესასვლელის მიხედვით ჩვეულებრივ რეჟიმში არ ცვლის **TO** და **PD** ბიტების მიმდინარე მნიშვნელობებს.

TO	PD	**მოვლენები, რომლებიც იწვევენ მდგომარეობას "ჩამოყრა"**
1	1	ჩამოყრა კვების ჩართვის მიხედვით
0	1	ამუშავდა მოთვალთვალე ტაიმერი (მაგრამ არა რეჟიმში SLEEP)
1	0	ჩამოყრა MCLRშესასვლელის მიხედვით SLEEP რეჟიმში ან გამოსვლა SLEEP-დან გარე წყვეტის მიხედვით
0	0	გამოსვლა SLEEP-დან მოთვალთვალე ტაიმერის სიგნალის მიხედვით
x	x	ჩამოყრა MCLRშესასვლელის მიხედვით ჩვეულებრივ რეჟიმში

რეგისტრი OPTION_REG(მისამართი 81H)

სპეციალური რეგისტრი **OPTION_REG** წარმოადგენს ჩაწერისა და წაკითხვისათვის მთლიანად მისაწვდომ რეგისტრს, რომელშიც იმყოფებიან წინასწარი გაყოფის, გარე წყვეტების წყაროების, TMR0 ჩადგმული ტაიმერის და B პორტის მომჭერი რეზისტორების მმართველი ბიტები.

OPTION_REG რეგისტრის ბიტების დანიშნულება:

bit7 **RBPU**– B პორტის ჩადგმული დატვირთვის ჩართვა

1=დატვირთვა გამორთულია

0=დატვირთვა ჩართულია

bit6 **INTEDG** – წყვეტადი სიგნალის ფრონტის არჩევა

1=წყვეტა სიგნალის ზრდის მიხედვით გამოსასვლელზე RB0/INT

0= წყვეტა სიგნალის ვარდნის მიხედვით გამოსასვლელზე RB0/INT

bit5 **T0CS** – ტაქტირების წყაროს არჩევა ტაიმერისათვის TMR0

1=იმპულსები შესასვლელიდან RA4/T0CKI

0=შიდა ტაქტური სიხშირე (CLKOUT)

bit4 **T0SE** – ფრონტის სიგნალის არჩევა ტაიმერისათვის TMR0, თუ წყაროდ არჩეულია შესასვლელი RA4/T0CKI (T0CS=1)

1=ინკრემენტი ვარდნის მიხედვით გამოსასვლელზე RA4/T0CKI

0=ინკრემენტი ზრდის მიხედვით გამოსასვლელზე RA4/T0CKI

bit3 **PSA** – ბიტი, რომელიც მართავს წინასწარი გამყოფის მიერთებას

1=წინასწარი გამყოფი მიერთებულია WDT-სთან

0=წინასწარი გამყოფი მიერთებულია TMR0-სთან

bit2-0 **PS2-PS0** – წინასწარი გამყოფის გაყოფის კოეფიციენტის მართვა მიერთებისდა მიხედვით

ბიტები	TMR0-სათვის	WDT-სათვის
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

რეგისტრი INTCON (მისამართი 0BH, 8BH)

სპეციალური რეგისტრი **OPTION_REG** ჩაწერისა და წაკითხვისათვის მთლიანად მისაწვდომ რეგისტრია, რომელშიც იმყოფებიან წყვეტის სხვადასხვა წყაროების მმართველი ბიტები.

bit7 **GIE** – წყვეტების გლობალური აკრძალვის ბიტი

1=დაშვებულია ყველა არაშაბლონირებადი წყვეტა

0=აკრძალულია ყველანაირი წყვეტა

bit6 **EEIE** – წყვეტის ნებართვა EEPROM-ში ჩაწერის დასრულების შემდეგ

1=დაშვებულია წყვეტა ჩაწერის დამთავრების შემდეგ

0=აკრძალულია წყვეტა ჩაწერის დამთავრების შემდეგ

bit5 **T0IE** – წყვეტის ნებართვა TMR0-ის გადავსების მიხედვით

1=დაშვებულია წყვეტა

0=აკრძალულია წყვეტა

bit4 **INTE** – წყვეტის ნებართვა RB0/INT შესასვლელის მიხედვით

1=დაშვებულია წყვეტა

0=აკრძალულია წყვეტა

bit3 **RBIE** – წყვეტის ნებართვა B პორტის (ხაზები RB7/RB4) შესასვლელებზე მდგომარეობის ცვლილების მიხედვით

1=დაშვებულია წყვეტა

0=აკრძალულია წყვეტა

bit2 **T0IF** – წყვეტის ალამი TMR0 ტაიმერ-მთვლელის გადავსების მიხედვით

1= TMR0 გადავსებულ იქნა (უნდა ჩამოიყაროს პროგრამულად)

0= TMR0 არ იქნა გადავსებული

bit1 **INTF** – წყვეტის ალამი RB0/INT შესასვლელის მიხედვით

1=ადგილი ჰქონდა წყვეტას RB0/INT შესასვლელის მიხედვით (უნდა ჩამოიყაროს პროგრამულად)

0= ადგილი არ ჰქონდა წყვეტას RB0/INT შესასვლელის მიხედვით

ალამი გამოიყენება წყვეტის წყაროს განსაზღვრისათვის.

bit0 **RBIF** – წყვეტის ალამი RB7-RB4 შესასვლელებზე მდგომარეობის ცვლილების მიხედვით

1= RB7-RB4-ის ერთ-ერთ გამომყვანზე მოხდა დონის ცვლილება (უნდა ჩამოიყაროს პროგრამულად)

0= არ მოხდა წყვეტა დონის ცვლილების მიხედვით

ალამი გამოიყენება წყვეტის წყაროს განსაზღვრისათვის.

პროგრამული მთვლელი

მიკროკონტროლერის პროგრამული მთვლელი (პმ) შეიცავს 13 თანრიგს. მთვლელის უმცროსი ბაიტი წარმოსდგენს ჩაწერისა და წაკითხვისათვის მთლიანად მისაწვდომ PCL რეგისტრს (მისამართი 02h, 82h). მთვლელის ხუთი უფროსი თანრიგი უშუალოდ მიუწვდომელია ჩაწერისა და წაკითხვისათვის. მათთან მიმართვა ხორციელდება PCLATCH რეგისტრის საშუალებით (მისამართი 0Ah, 8Ah), რომელიც წარმოადგენს ბუფერ-საკეტს მთვლელის უფროსი ბიტებისათვის. PCLATCH-ის შიგთავსი გადაიტანება პმ-ის უფროს თანრიგებში, როცა ხდება პროგრამულ მთვლელში ახალი მნიშვნელობის ჩაწერა. ეს ხდება, როცა სრულდება ბრძანებები CALL, GOTO ან PCL რეგისტრი წარმოადგენს დანიშნულების რეგისტრს არითმეტიკული ოპერაციის შედეგისათვის.

არითმეტიკული ოპერაციის უშუალოდ პროგრამულ მთვლელზე შესრულების შესაძლებლობა საშუალებას გვაძლევს საკმაოდ მარტივად და ეფექტურად, მხოლოდ ორი ბრძანებით, შევასრულოთ მნიშვნელობების ცხრილური გადაკოდირება.. საჭიროა მხოლოდ საწყისი მნიშვნელობის კონტროლი იმისათვის, რომ პროგრამული მთვლელი არ გამოვიდეს ცხრილის საზღვრებიდან, თუ მასში გამოყენებულია 256 ჩანაწერზე ნაკლები.

მიკროკონტროლერებს PIC16F83 და PIC16CR83 გააჩნიათ პროგრამული მეხსიერების 512 სიტყვა, მიკროკონტროლერებს PIC16F84 და PIC16CR84 – 1K პროგრამული მეხსიერება. ბრძანებები CALL და GOTO იყენებენ 11 ბიტს მისამართის მისათითებლად, რაც საშუალებას იძლევა გამოვიყენოთ 2K სამისამართო მეხსიერება. PIC16F8x-ის პროგრამული მეხსიერების შემდგომი გაფართოებისათვის საჭირო იქნება კიდევ ორი ბიტი, რომლებიც განსაზღვრავს მის გვერდებს. ამ ბიტებად გამოყენებული იქნება PCLATCH რეგისტრის 4 და 3 თანრიგები. პროგრამაში CALL და GOTO ბრძანებების გამოყენებისას დარწმუნებული უნდა ვიყოთ, რომ PCLATCH რეგისტრის 4 და 3 ბიტები დაპროგრამირებულია სწორად და მოხდება გადასვლა მეხსიერების საჭირო გვერდზე. *ამჟამად 4 და 3 ბიტები არ გამოიყენება, თუმცა "ქვემოდან ზემოთ" შეთავსებადობის უზრუნველსაყოფად პროგრამებში საერთო დანიშნულების ბიტებად მათი გამოყენება (მაგალითად, ალამთა შენახვისათვის) არ არის რეკომენდირებული.*

სტეკი

მიკროკონტროლერებს PIC16Fxx გააჩნია 8-დონიანი 13-ბიტიანი აპარატურული სტეკი. სტეკი არ წარმოადგენს მონაცემების ან პროგრამის მეხსიერების ნაწილს და სტეკის მაჩვენებელი არ არის მისაწვდომი წაკითხვისა და შენახვისათვის.

13-ბიტიანი პროგრამული მთვლელის შიგთავსი იტვირთება სტეკში CALL ბრძანების შესრულებისას ან წყვეტის ქვეპროგრამაზე გადასვლისას. RETURN, RETLW და RETFIE ბრძანებების შესრულებისას პროგრამული მთვლელის შიგთავსი აღდგება.

სტეკი ორგანიზებულია, როგორც ციკლური ბუფერი. ეს ნიშნავს, რომ მას შემდეგ, რაც სტეკში ჩაწერილია მერვე მნიშვნელობა, კვლავ მოსული მეცხრე მნიშვნელობა ჩაიწერება პირველ ადგილზე, მეათე მეორის ადგილზე და ა.შ. და პირიქით, თუ ხდებოდა პროგრამული მთვლელის აღდგენა რვაჯერ, მაშინ მეცხრეჯერ მასში ჩაიტვირთება პირველი მნიშვნელობა.

სტეკს არ გაჩნია ალმის ბიტი, რომელიც აჩვენებს გადავსებას ან გადაჭარბებულ დატვირთვას, ამიტომ მომხმარმელმა დამოუკიდებლად უნდა ადევნოს თვალი, რომ ქვეპროგრამების შეთავსების დონემ ნებისმიერ სიტუაციაში არ გადააჭარბოს რვას.

ირიბი ადრესაცია: რეგისტრები INDF და FSR

რეგისტრი INDF არ წარმოადგენს ფიზიკურ რეგისტრს. ამ რეგისტრთან მიმართვისას სინამდვილეში ხდება იმ რეგისტრის ადრესაცია, რომლის მისამართი ნაჩვენებია რეგისტრში FSR (ე. ი. FSR წარმოადგენს მიმთითებელს). ასეთი ადრესაცია წარმოადგენს ირიბს.

ირიბი ადრესაციის მაგალითი:

- საერთო დანიშნულების რეგისტრი 0Ch რეგისტრი მისამართზე შეიცავს მნიშვნელობას 01h
- საერთო დანიშნულების რეგისტრი 0Dh რეგისტრი მისამართზე შეიცავს მნიშვნელობას B8h
- FSR რეგისტრში ვტვირთავთ მნიშვნელობას 0Ch
- ვკითხულობთ INDF რეგისტრის მნიშვნელობას და ვღებულობთ მნიშვნელობას 01h
- ვახდენთ FSR (FSR =0Dh) რეგისტრის მნიშნელობის ინკრემენტს ერთით
- ვკითხულობთ მნიშვნელობას INDF რეგისტრიდან და ვღებულობთ მნიშვნელობას B8h

თვით FSR (FSR =0Dh) რეგისტრის მნიშვნელობის წაკითხვის ცდისას ყოველთვის ბრუნდება მნიშვნელობას 00h. INDF რეგისტრში ირიბი ჩაწერის ცდა სრულდება როგორც ცარიელი ოპერაცია (მიუხედავად იმისა, რომ STATUS რეგისტრის ბიტები შეიძლება შეიცვალოს).

ქვემოთ მოყვანილი პროგრამის ფრაგმენტი, რომელიც იყენებს ირიბ ადრესაციას, ასუფთავებს საერთო დანიშნულების რეგისტრებს (RAM), რომლებიც განლაგებულია მისამართებზე 20h-2Fh:

```
. . .
. . .

MOVLW 0x20        ; ვაყენებთ მიმთითებელს
MOVWF FSR         ; RAM-ის სასტარტო მისამართზე
NEXT CLRF INDF        ; ვასუფთავებთ რეგისტრს INDF
     INCF FSR         ; ვახდენთ მიმთითებლის ინკრემენტს
     BTFSS FSR, 4     ; დაამთავრეთ გასუფთავება?
     GOTO  NEXT       ; არა, გასუფთავდეს შემდეგი უჯრედი
                      ; თუ კი, გაგრძელდეს პროგრამა
```

შეტანა-გამოტანის პორტები

კონტროლერებს PIC16F8x გააჩნიათ შეტანა-გამოტანის ორი პორტი **PORTA** და **PORTB.** პორტის თითოეული გამომყვანი შეიძლება დაპროგრამირდეს შესასვლელზე ან გამოსასვლელზე **TRISA** და **TRISB** რეგისტრებში შესაბამისი ბიტის დაყენებით. გამოსაყვანი მნიშვნელობები ფიქსირდება **PORTA და PORTB** რეგისტრ-საკეტებში. შეტანა-გამოტანის მიმართულება შეიძლება შეიცვალოს დროის ნებისმიერ მომენტში. ამას გარდა, ზოგიერთ გამომყვანს მინიჭებული აქვს დამატებითი ფუნქციები.

ორმხრივმიმართული პორტი A, რეგისტრები PORTA და TRISA

PORTA წარმოადგენს 5-ბიტიან საკეტს. **RA4** ხაზს გააჩნია შმიტის ტრიგერი შესასვლელზე შეტანის რეჟიმში და გახსნილი შესართავით გამოსასვლელის რეჟიმში. პორტ **A**-ს დანარჩენი ხაზები შესასვლელის მიხედვით მუშაობენ სტანდარტულ **TTL** დონეებთან. გამოსასვლელები უერთდება *კომპლემენტარულ* გამოსასვლელ **CMOS** დრაივერებს. **RA4** ხაზი გამოიყენება აგრეთვე როგორც ტაქტური იმპულსების შესასვლელი **TMR0** ტაიმერისთვის.

მონაცემების გადაცემის მიმართულება თითოეული ხაზისთვის პროგრამირდება ცალკე **TRISA** რეგისტრის bit0…bit4 ბიტების დაყენებით ან ჩამოყრით. ბიტის 1-ში დაყენება ააწყობს შესაბამის ხაზს შესასვლელზე. გამოსასვლელი დრაივერი ამ დროს გადადის მაღალიმპედანსურ მდგომარეობაში. ბიტის 0-ში დაყენება ააწყობს პორტის ხაზს გამოსასვლელზე და გამოიყვანს მასზე **PORTA**-ს სალტის შესაბამისი ბიტის შიგთავსს. გაჩუმების მიხედვით კვების ჩართვისას ყველა ხაზი აწყობილია შესასვლელზე. **A** პორტის კითხვისას ყოველთვის ხდება გამოსასვლელებზე ნამდვილი ლოგიკური მნიშვნელობების წაკითხვა, იმისდა მიუხედავად

ცალკეული თანრიგები დაპროგრამირებულია როგორც შესასვლელები თუ როგორც გამოსასვლელები.

პორტში ჩაწერის ყველა ოპერაცია სრულდება როგორც წაკითხვა-მოდიფიკაცია-ჩაწერა. მაგალითად, როცა სრულდება ბრძანება **BSF PORTA, 3 (A** პორტის ბიტ 3-ის დაყენება ერთიანში), მაშინ **A** პორტის ყველა ხაზიდან ხდება იმ რეალური მნიშვნელობებ წაკითხვა, რომლებიც ამ მომენტში არიან გამოსასვლელებზე, მოდიფიცირდება მესამე ბიტი და ყველა მიღებული მნიშვნელობა ჩაიწერება რეგისტრ-საკეტ **PORTA**-ში. ამგვარად, საკეტ **PORTA**-ს ზოგიერთ ბიტს შეიძლება ჰქონდეს გაუთვალისწინებელი მნიშვნელობა. თუ პროგრამის შესრულების პროცესში **A** პორტის ხაზები წაკითხვის შემდეგ გადაწყობილი იქნება გამოსასვლელზე, მაშინ ზოგიერთ მათგანზე გამოყვანილი იქნება გაუთვალისწინებელი მნიშვნელობები, რამაც შეიძლება მიგვიყვანოს მოულოდნელ შედეგებამდე მოწყობილობის მუშაობისას. *პროგრამის წერის პროცესში, პორტის ხაზების აწყობის ცვლილების წინ, რეკომენდირებულია ცხადად მოიცეს მნიშვნელობები საკეტ **PORTA**-ში კრიტიკული ხაზებისათვის.*

ასევე უნდა იქნას გათვალისწინებული შიგა წრედების ინერციულობა პორტის ხაზებზე სიგნალის ცვლილებისას. მაგალითად პორტის ხაზებიდან ერთ-ერთი აწყობილი იყო გამოსასვლელზე, და მასზე არსებობდა ლოგიკური ნულის დონე. შემდეგ ეს ხაზი გადააწყეს შესასვლელზე, და მოწყობილობის სქემის შესაბამისად, გარედან ხაზზე მოდის მაღალი დონის სიგნალი რეზისტორიდან, რომელიც მიერთებულია მკვებავ სალტესთან. მონტაჟის პარაზიტული ტევადობისა და სხვა კომპონენტების საკუთარი ტევადობების გამო მაღალი დონე ხაზზე მიეწოდება გარკვეული დაყოვნებით, და წაკითხვისას შეიძლება მიღებულ იქნას გაუთვალისწინებელი მნიშვნელობა. პორტის გამოსასვლელ დრაივერებს ასევე ახასიათებთ გარკვეული ინერციულობა. ამიტომ *არასასურველია, რომ პორტის წაკითხვის ბრძანება სწრაფადვე მოჰყვეს პორტის გამოსასვლელიდან შესასვლელზე გადაწყობის ბრძანებას.* შეიძლება გამოვიყენოთ ბრძანება **NOP.**

ქვემოთ მოყვანილია **A** პორტის ინიციალიზაციის მაგალითი.

```
CLRF        PORTA          ; ვანულებთ ყველა ბიტს საკეტში
BSF         STATUS, RP0    ; ვირჩევთ ბანკს 1
    MOVLW       0x1E           ; მნიშვნელობა, რომელიც ბიტების მიხედვით
        ; განსაზღვრავს შესასვლელ/გამოსასვლელის
        ; მიმართულებას თითოეული ხაზისათვის:
        ; RA0 - გამოსასვლელი,
        ; RA1 – RA4 შესასვლელი
  MOVWF     TRISA          ; მმართველ რეგისტრში (მისამართი 85ჰ,
                           ; ბანკი 1) მნიშვნელობის ჩაწერა
```

BSF STATUS, RP0 ; ვირჩევთ ბანკს 0

A პორტის ბლოკ-სქემა **RA0… RA3** და **RA4** ხაზებისათვის მოყვანილია ნახ. 1-3.

ორმხრივმიმართული პორტი B, რეგისტრები PORTB და TRISB

PORTB წარმოადგენს 8-ბიტიან ორმხრივმიმართულ პორტს. პორტის სტრუქტურული სქემა წარმოდგენილია ნახაზზე 1-4. გამოსასვლელი მნიშვნელობები ჩაიწერება რეგისტრ-საკეტ **PORTB**-ში. შესასვლელ-გამოსასვლელის მიმართულება განისაზღვრება **TRISB** რეგისტრის ბიტების დაყენებით ან ჩამოყრით. ბიტის 1-ში დაყენება ააწყობს შესაბამის ხაზს შესასვლელზე. გამოსასვლელი დრაივერი ამ დროს გადადის მაღალიმპედანსურ მდგომარეობაში. ბიტის 0-ში დაყენება ააწყობს პორტის ხაზს გამოსასვლელზე. კვების ჩართვისას ყველა ხაზი გაჩუმების მიხედვით აწყობილია შესასვლელზე. ისევე როგორც პორტ **A**-სათვის, პორტ **B-**ს წაკითხვა ყოველთვის აბრუნებს ნამდვილ მნიშვნელობებს გამოსასვლელებზე, თითოეული გამომყვანისთვის მონაცემების გადაცემის მიმართულებისაგან დამოუკიდებლად.

პორტ **B-**ს ყველა გამომყვანს აქვს ჩადგმული გამორთვადი დატვირთვა რეზისტორების სახით, რომლებიც მიერთებულია კვების სალტესთან (მომჭერი რეზისტორები). დატვირთვა ჩაირთვება ან გამოირთვება ერთდროულად ყველა გამოსასვლელისთვის ერთდროულად **OPTION_REG** რეგისტრის **RBPU** ბიტის საშუალებით. კვების ჩართვისას **RBPU=1** და დატვირთვა გამორთულია. **RBPU** ბიტის პროგრამული განულება ჩართავს დატვირთვას, მაგრამ გამოსასვლელზე აწყობილი ხაზებისათვის დატვირთვა ავტომატურად გამოირთვება.

RB4…RB7 ხაზები შეიძლება გამოვიყენოთ როგორც *წყვეტის* შესასვლელები *დონის ცვლილების მიხედვით.* ამ რანგში შეიძლება გამოვიყენოთ მხოლოდ შესასვლელზე აწყობილი ხაზები. თითოეულ ბრძანებით ციკლში ხდება გამოსასვლელებზე მიმდინარე მნიშვნელობების შედარება წინა მნიშვნელობებთან, რომლებიც დაფიქსირებულია სპეციალურ საკეტში. თუ ამ გამოსასვლელებიდან ერთ-ერთზე მაინც მოხდა დონის ცვლილება, ფორმირდება წყვეტა. იმპულსის ხანგრძლივობა, რომელიც გაიგება, როგორც დონის ცვლილება, არ უნდა იყოს ტაქტური სიხშირის 4 პერიოდზე ნაკლები. პროგრამული ამოცნობა იმისა, თუ **RB4…RB7** ხაზებიდან რომლიდან მოხდა წყვეტა, შეუძლებელია.

ამ წყვეტას კონტროლიორი გამოყავს **SLEEP** მდგომარეობიდან. წყვეტის დამუშავების ქვეპროგრამაში მომხმარებელმა უნდა ჩამოყაროს წყვეტა ორიდან ერთი საშუალებით:

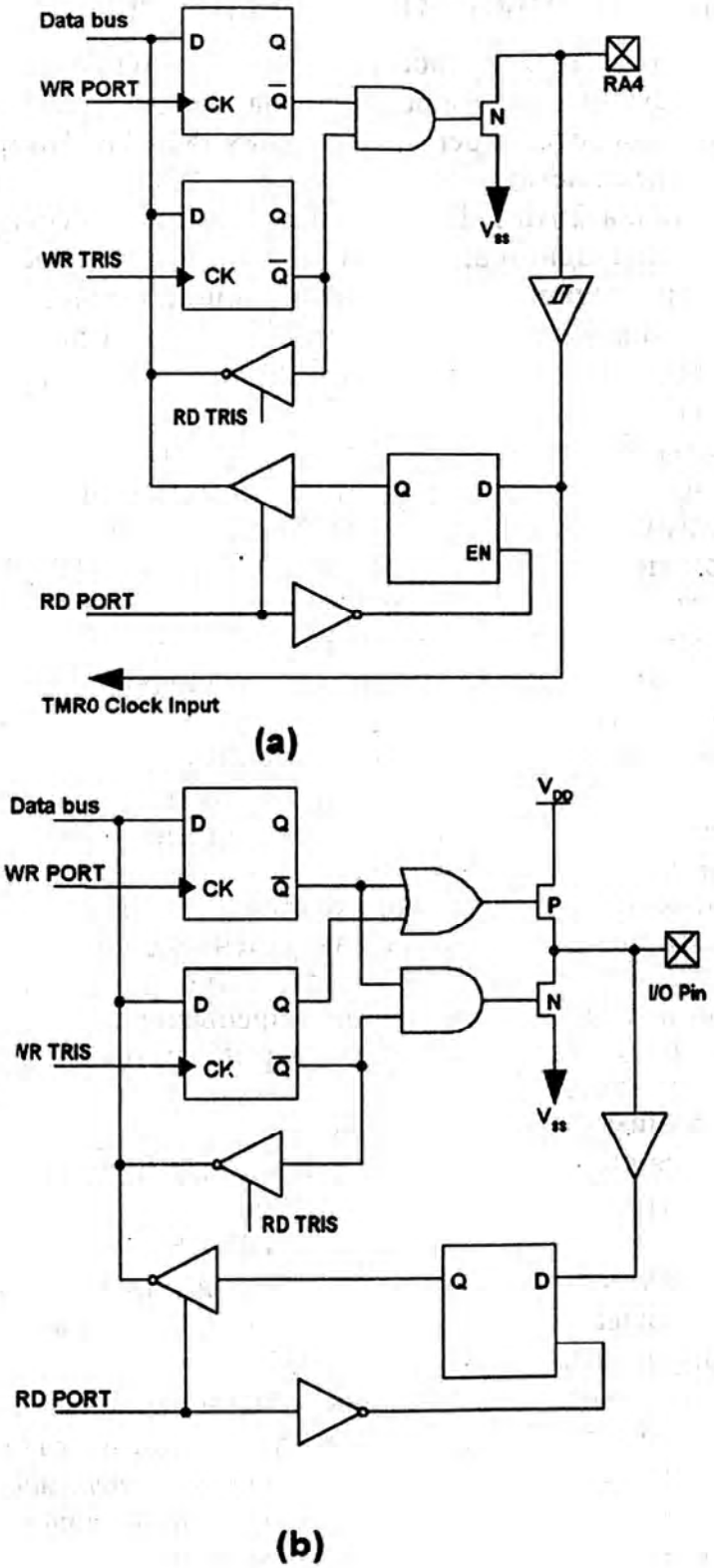
– წაკითხვისა და ჩაწერისათვის მიმართოს პორტ **B-**ს, ამ დროს **INTCON** რეგისტრის ბიტ-ალამი **RBIF** განულდება,

– უშუალოდ გაანულოს ბიტ-ალამი **RBIF**

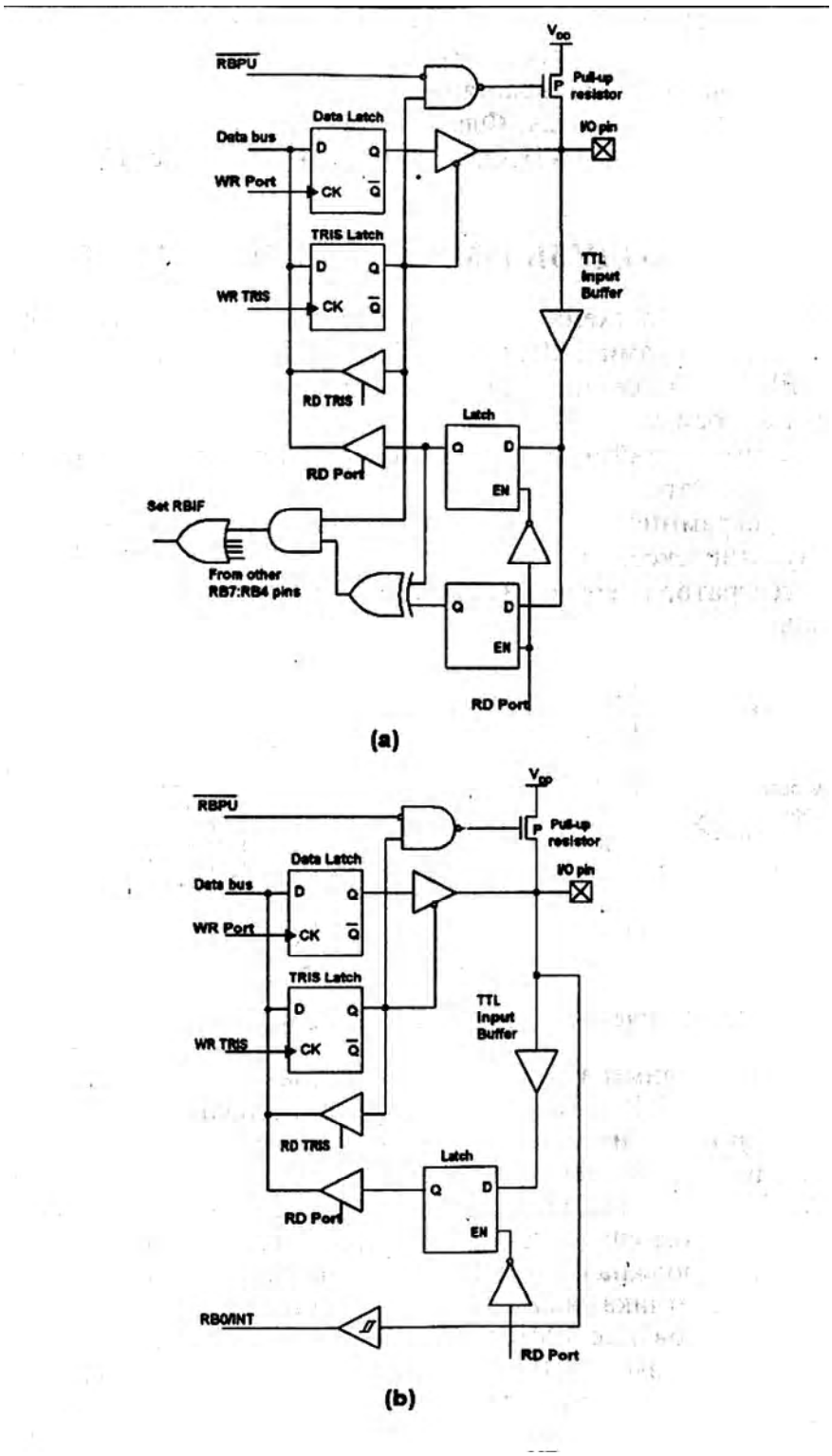
თუ არ გავანულებთ ბიტს **RBIF**, მაშინ წყვეტის დამუშავების ქვეპროგრამიდან გამოსვლის შემდეგ კონტროლიორი კვლავ აღმოაჩენს წყვეტას და გადავა მისი დამუშავების ქვეპროგრამაზე. პროგრამა ჩაიციკლება.

დონის ცვლილების მიხედვით წყვეტას ჩვეულებრივ იყენებენ კონტროლიორის გამოსაყვანად **SLEEP** რეჟიმიდან კლავიატურის ღილაკების დაჭერით. *როცა ნებადართულია დონის ცვლილების მიხედვით წყვეტა, არ არის რეკომენდირებული პორტ **B**-ს **RB4**…**RB7** ხაზების აწყობა გამოსასვლელზე და მათზე დონის ცვლილება, ვინაიდან მსგავსი ცვლილება შეიძლება გაგებულ იქნას, როგორც წყვეტის მოთხოვნა.*

გამოსასვლელი **RB0/INT** გამოიყენება როგორც გარე წყვეტის შესასვლელი. თუ **OPTION_REG** რეგისტრის **INTEDG** ბიტი დაყენებულია 1-ში, მაშინ შესასვლელზე **RB0/INT** ადგილი აქვს დონის ზრდის მიხედვით წყვეტას, ხოლო თუ **INTEDG**=0, – დონის კლების მიხედვით წყვეტას. ამ წყვეტისათვის ბიტ-ალამს წარმოადგენს **INTCOM** რეგისტრის **INTF** ბიტი. ამ ბიტის ჩამოყრა იძულებით არ მოითხოვება.



ნახ. 2. A პორტის, RA (a) და RA0-RA3 (ბ) ხაზების ბლოკ-სქემა.



ნახ. 3. B პორტის, RB7-RB4 (a) და RB3-RB0 (b) ხაზების ბლოკ-სქემა.

მოდული TIMER0 და რეგისტრი TMR0

შიგა მოდულის **TIMER0** სტრუქტურული სქემა და მისი ურთიერთკავშირი **TMR0** და **OPTION_REG** რეგისტრებთან ნაჩვენებია ნახ. 1-5. **TIMER0** წარმოადგენს ტაიმერ-მთვლელის პროგრამირებად მოდულს. იგი შეიცავს:

- 8-ბიტიან ტაიმერ-მთვლელს **TMR0**, რომელიც მისაწვდომია როგორც რეგისტრი წაკითხვისა და ჩაწერისათვის,
- პროგრამირებად წინასწარ გამყოფს,

შესასვლელი სიგნალის მულტიპლექსორს

- **TMR0** რეგისტრის **FFh**-დან **00h**-ში გადავსების მიხედვით წყვეტის გენერატორს.

ტაიმერის რეჟიმის მოცემა ხდება **OPTION_REG** რეგისტრის **T0CS** ბიტის ჩამოყრით. ამ რეჟიმში **TMR0**-ის ინკრემენტირება ხდება ტაქტური სიხშირის *შიდა* წყაროდან. თუ წინასწარი გამყოფი არ არის მიერთებული, ინკრემენტირება ხდება თითოეულ მანქანურ ციკლში. თუ ხდებოდა ჩაწერა **TMR0** რეგისტრში, მისი ინკრემენტირება დაიწყება ორი ბრძანების ციკლის შემდეგ, ამიტომ ჩაწერადი ცვლადი უნდა იქნეს შესაბამისად კორექტირებული.

მთვლელის რეჟიმის მოცემა ხდება **T0CS** ბიტის 1-ში დაყენებით. ამ შემთხვევაში **TMR0**-ის ინკრემენტირება ხდება გარე წყაროდან, რომელიც მიერთებულია **TMR** გამოსასვლელთან. შესასვლელი იმპულსების აქტიური ფრონტი შეიძლება მოიცეს პროგრამულად. თუ **OPTION_REG** რეგისტრის **T0SE** ბიტი ჩამოყრილია, აქტიურს წარმოადგენს სიგნალის ზრდა.

წინასწარი გამყოფი

წინასწარი გამყოფი წარმოადგენს 8-ბიტიან მთვლელს, რომელიც ასევე შეიძლება გამოყენებული იქნას, როგორც მოთვალთვალე ტაიმერის გამოსასვლელი გამყოფი (პოსტგამყოფი). თუ წინასწარი გამყოფი მიერთებულია **TMR0** მოდულთან, მაშინ ის არ შეიძლება გამოყენებული იყოს მოთვალთვალე ტაიმერთან და პირიქით. **OPTION_REG** რეგისტრის **PSA** ბიტი განსაზღვრავს წინასწარი გამყოფის მიერთებას. თუ **PSA=0,** წინასწარი გამყოფი მიერთებულია **TMR0** ტაიმერ/მთვლელთან. ბიტები **PS0… PS2** განსაზღვრავენ გაყოფის კოეფიციენტს, ცხრილი მოყვანილია **OPTION_REG** რეგისტრის აღწერილობაში.

როცა წინასწარი გამყოფი მიერთებულია ტაიმერ/მთვლელთან, ყველა ბრძანება, რომელიც იყენებს **TMR0** რეგისტრში ჩაწერას, ანულებს წინასწარ გამყოფს. თუ წინასწარი გამყოფი მიერთებულია მოთვალთვალე ტაიმერთან, ისინი ნულდება ერთობლივად, CLRWDT ბრძანებით. წინასწარი გამყოფი მიუღწევადია პირდაპირი ჩაწერისა და წაკითხვისათვის.

წინასწარი გამყოფის მიერთება შეიძლება შეიცვალოს ძალიან ადვილად, ანუ პროგრამის შესრულების დროს. *იმისათვის, რომ ავიცილოთ კონტროლერის წინასწარ განუზრახელი ჩამოყრა,*

რომელიც შეიძლება მოხდეს ამ დროს, დამამზადებელი იძლევა რეკომენდაციას მოვახდინოთ წინასწარი გამყოფის გადართვა ბრძანებათა მკაცრად განსაზღვრული თანმიმდევრობით:

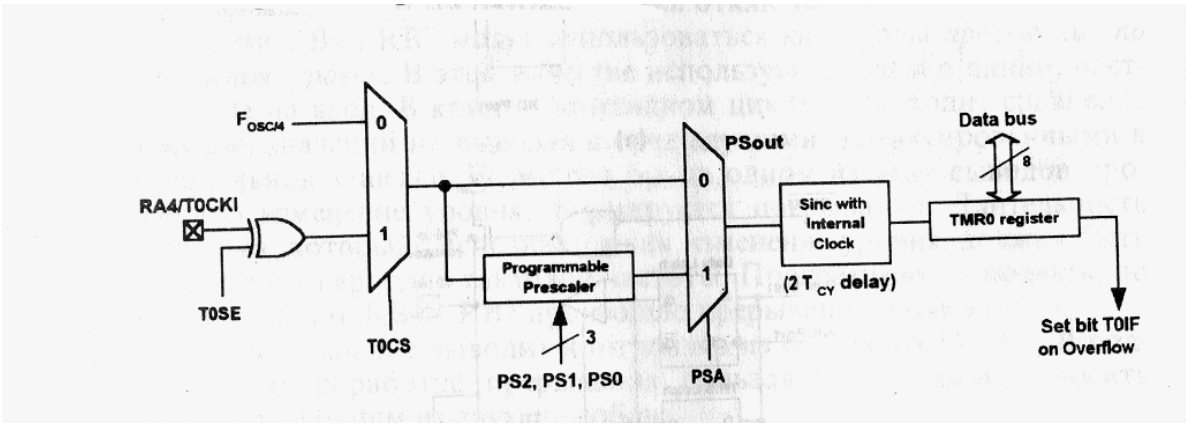
*1. წინასწარი გამყოფის გადართვა **TMR0**-დან **WDT**-ში*

```
BCF        STATUS, RP0     ; ბანკი 0
CLRF       TMR0            ; TMR0 მთვლელის და წინასწარი გამყოფის
                           ; გასუფთავება
BSF        STATUS, RP0     ; ბანკი 1
CLRWDT                     ; მოთვალთვალე ტაიმერის გასუფთავება
 MOVLW     b'xxxx1xxx'     ;წინასწარი გამყოფის მნიშვნელობის
 ;დაყენება,
 ;ბიტები PS0… PS2
 MOVWF     OPTION_REG      ; მიერთება WDT-სთან
BSF        STATUS, RP0     ; ბანკი 0
```

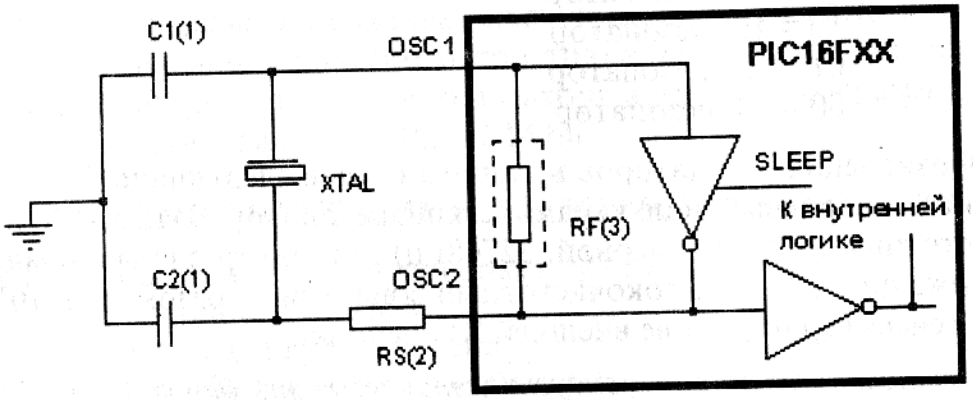
*2. წინასწარი გამყოფის გადართვა **WDT**-დან **TMR0**-ში*

```
CLRWDT                     ; მოთვალთვალე ტაიმერის და
                           ;წინასწარი გამყოფის გასუფთავება
BSF        STATUS, RP0     ; ბანკი 1
MOVLW      b'xxxx0xxx'     ; TMR0-თან მიერთება,
                           ;წინასწარი გამყოფის ბიტების დაყენება
MOVWF      OPTION_REG      ; და ტაქტირების წყაროსთან მიერთება
BSF        STATUS, RP0     ; ბანკი 0
```

აქ სიმბოლო "x" აღნიშნავს ბიტებს, რომელთა მოცემა ხდება პროგრამისტის მიერ კონკრეტულად, კონკრეტული სიტუაციისდა მიხედვით.

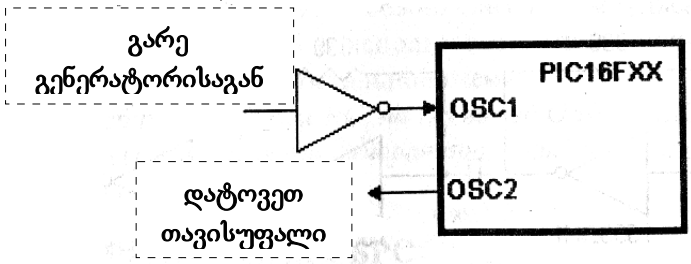


ნახ. 4. **TIMER0** მოდულის სტრუქტურული სქემა.



1. **C1** და **C2** კონდენსატორების ნომინალები იხილე ცხრილში.
2. **RS** მიმდევრობითი რეზისტორი შეიძლება დაგვჭირდესტიპის **AT** ნტიპის ჭრილის მქონე კრისტალის გამოყენებისას.
3. **RF** შიგა წინაღობა მიახლოებით 10 მომ ტოლია.

ნახ 5, კვარცული რეზონატორის ჩართვის სქემა.



ნახ. 6. გარე გენერატორის ჩართვის სქემა.

გარე წყაროდან ტაქტირების თავისებურებანი

გარე წყაროდან ინკრემენტირებას გააჩნია რიგი თავისებურებები. გარე სიგნალის სტრობირება ხდება სინქრონიზაციის შიგა იმპულსებით. შესასვლელი სიგნალის ფრონტსა და იმ ფრონტს შორის, რის მიხედვითაც ხდება რეალურად ხდება **TMR0** მთვლელის ინკრემენტირება, წარმოიქმნება გარკვეული დაყოვნება. ამ დაყოვნების ვარირება ხდება ტაქტური გენერატორის რხევების 3-დან 7-მდე პერიოდის ზღვრებში. მაგალითად, კონტროლიორის ტაქტირებისას 10მჰც კვარცული გენერატორიდან დაყოვნება შეადგენს 300-დან 700ნწმ-მდე. შესაბამისად, გარე მოვლენებს შორის დროითი ინტერვალის გაზომვის ცდომილება შეადგენს ±400ნწმ.

სტრობირება ხდება წინასწარი გამყოფის გამოსასვლელზე, ორჯერ თითოეული ბრძანებითი ციკლის განმავლობაში იმისათვის, რომ განისაზღვროს მატაქტირებელი იმპულსის ზრდა და ვარდნა. შედეგად მაღალი დონის (Tმ) და დაბალი დონის (Tდ) მდგომარეობების ხანგრძლივობა წინასწარი გამყოფის გამოსასვლელზე არ უნდა იყოს სინქრონიზაციის ორ პერიოდზე ნაკლები, პლუს 20ნწმ დამყარებაზე.

თუ წინასწარი გამყოფი არ არის მიერთებული, შესასვლელი სიგნალი ეწოდება პირდაპირ და მას უნდა ჰქონდეს შემდეგი პარამეტრები:

Tმ>=2* Tosc+20ნწმ

Tდ>=2* Tosc+20ნწმ

თუ წინასწარი გამყოფი მიერთებულია, სიგნალს მის გამოსასვლელზე ყოველთვის აქვს მეანდრის ფორმა. სიგნალის მაღალი და დაბალი დონეების ხანგრძლივობები წინასწარი გამყოფის შესასვლელზე არ უნდა იყოს 10ნწმ-ზე ნაკლები. შედეგად, შესასვლელ სიგნალს RA4/T0CK გამოსასვლელზე მიერთებული წინასწარი გამყოფის დროს უნდა ჰქონდეს პარამეტრები:

Tმ>=10ნწმ

Tდ>=10ნწმ.

მუშაობა EEPROM-თან

EEPROM-ის მონაცემთა მეხსიერება ხელმისაწვდომია წაკითხვისა და ჩაწერისათვის მკვებავი ძაბვის მთელ სამუშაო დიაპაზონში და დანიშნულია 8-ბიტიანი მნიშვნელობების შენახვისათვის. ხალი მნიშვნელობის ჩაწერის წინ იშლება წინა მნიშვნელობა. PIC16F8x მიკროკონტროლერებს გააჩნიათ 64 ბაიტი **EEPROM** მისამართებით 00h-დან 3Fh-მდე, მაგრამ ეს უჯრედები მიუწვდომელია პირდაპირი ადრესაციის გზით მიკროკონტროლერის სივრცეში. მათთან მისაწვდომად გამოიყენება ირიბ რეგისტრული ადრესაცია სპეციალური რეგისტრების გზით. მთლიანად **EEPROM**-თან მუშაობისას გამოიყენება ოთხი სპეციალური რეგისტრი:

- **EECON1**

- **EECON2**
- **EEDATA**
- **EEADR**

გაცვლის რეგისტრი **EEDATA** შეიცავს წაკითხვა-ჩაწერის 8-ბიტიან მონაცემებს. **EEADR** ინახავს უჯრედის მისამართს, რომელთანაც ხდება მიმართვა. *მიუხედავად იმისა, რომ **PIC16F8**x-ში ფიზიკურად არსებობს **EEPROM**-ის 64 ბაიტი, ხდება ყველა ბიტის დეკოდირება. ამიტომ აუცილებელია თვალი ვადევნოთ მნიშვნელობას **EEADR**-ში იმისათვის, რომ არ გამოვიდეთ სამისამართო სივრცის საზღვრებს გარეთ. მისამართის ორი უფროსი ბიტი ყოველთვის უნდა უდრიდეს ნულს. შესაძლებელია, ისინი გამოყენებული იქნენ შემდგომ მოდიფიკაციებში, მოცულობის გაზრდილი მოცულობით.*

EEPROM-ში ჩაწერისას მოითხოვება მკაცრად შევინარჩუნოთ დროითი ინტერვალი, რომელიც კონტროლირდება ჩადგმული ტაიმერით. ჩაწერის დრო შეიძლება იცვლებოდეს კრისტალიდან კრისტალამდე, ასევე მკვებავი ძაბვისა და ტემპერატურეს მიხედვით.

როდესაც მიკროკონტროლერში დაყენებულია კოდის დაცვის ბიტი, პროცესორს შეუძლია წაიკითხოს და ჩაწეროს **EEPROM**-ში, მაგრამ პროგრამატორისთვის ეს მეხსიერება ხდება მიუწვდომელი. ერთჯერადად პროგრამირებად მიკროკონტროლერებს აქვთ დაცვის ორი ბიტი – ერთი იცავს პროგრამების მეხსიერებას, მეორე – მონაცემების მეხსიერებას **EEPROM**-ს. მსგავსი მიდგომა საშუალებას იძლევა დაპროგრამირდეს წინასწარ კონტროლერების რაღაც რაოდენობა და დაცული იქნეს პროგრამული კოდი ისე, რომ არ შევიტანოთ კონსტანტები **EEPROM**-ში. შემდეგ, უშუალოდ პლატაზე მონტაჟის წინ, ყოველ კრისტალში შეაქვთ აუცილებელი ინდივიდუალური კონსტანტები და აყენებენ დაცვის მეორე ბიტს.

რეგისტრები EECON1 და EECON2

რეგისტრი EECON1 წარმოადგენს საკონტროლო რეგისტრს, რომლისთვისაც ფიზიკურად მისაწვდომია ხუთი უმცროსი ბიტი. სამი უფროსი ბიტი არამოქმედია და ყოველთვის წაიკითხება, როგორც “0”. რეგისტრის ბიტების დანიშნულება მოყვანილია ქვემოთ:

bit 7-5 **ფიზიკურად მიუწვდომელია**, ყოველთვის იკითხება, როგორც “0”

bit4 **EEIF** – წყვეტის ალამი ჩაწერის დასრულების მიხედვით

1=ჩაწერა დასრულებულია (ჩამოყრილი უნდა იქნეს პროგრამულად)

0=ჩაწერა დასრულებული არ არის ან არ დაწყებულა

bit3 **WRERR – EEPROM**-ში ჩაწერის შეცდომის ალამი

1=ჩაწერა შეწყვეტილია დროზე ადრე

0=ჩაწერა განხორციელდა წარმატებულად

bit2 **WREN – EEPROM**-ში ჩაწერის ნებართვა

1=ნებადართულია ჩაწერის ციკლი

0=აკრძალულია ჩაწერა

bit1 **WR** – ჩაწერის მართვის ბიტი

1=დაიწყოს ჩაწერის ციკლი. ბიტის პროგრამული დაყენება წარმოადგენს ჩაწერის ციკლის დაწყების ბრძანებას. ამ ბიტის ჩამოყრა ხდება მხოლოდ აპარატულად, როცა ჩაწერის ციკლი დასრულებულია.

0= დასრულებულია მონაცემების ჩაწერის ციკლი

bit0 **RD** – წაკითხვის მართვის ბიტი

1=დაიწყოს მონაცემების წაკითხვა **EEPROM**-დან. წაკითხვა იკავებს ერთ საბრძანებო ციკლს. ბიტის პროგრამული დაყენება წარმოადგენს წაკითხვის ციკლის დაწყების ბრძანებას. ბიტის ჩამოყრა ხდება მხოლოდ აპარატულად.

0=წაკითხვა არ დაწყებულა

იმის შეუძლებლობა, რომ პროგრამულად ჩამოიყაროს **WR** ბიტი, გვიცავს ჩაწერის ციკლის შემთხვევითი წინასწარი წყვეტისაგან, რადგან ეს ციკლი იკავებს რამდენიმე მანქანურ ტაქტს.

WREN ბიტი კვების ჩართვისას ჩამოყრილია, რაც გვიცავს შემთხვევითი ჩაწერისაგან. **WREN** ბიტის დაყენება ხდება, როდესაც ჩაწერის ოპერაცია შეწყვეტილია ჩამოყრით **MCLR** შესასვლელის მიხედვით ან მოთვალთვალე ტაიმერის გადავსების მიხედვით ჩამოყრით. ამ შემთხვევაში, განმეორებითი სტარტის დროს, მომხმარებელს შეუძლია შეამოწმოს **WRERR** ბიტი და, აუცილებლობის შემთხვევაში გაიმეოროს ჩაწერა. ჩამოყრის დროს მონაცემები და მოსამართი რეგისტრებში **EEDATA** და **EEADR** არ იკარგება.

რეგისტრი **EECON2** არ წარმოადგენს ფიზიკურ რეგისტრს და გამოიყენება ჩაწერისას განსაკუთრებით დამხმარე რეგისტრის სახით. ამ რეგისტრის წაკითხვა ყოველთვის აბრუნებს მნიშვნელობას “0”.

მონაცემების წაკითხვა EEPROM-იდან

მონაცემების წაკითხვისათვის აუცილებელია ჩაიწეროს მისამართი რეგისტრში **EEADR** და დაყენდეს **EECON1** რეგისტრის RD ბიტი. შემდგომ ციკლში მონაცემები უკვე მისაწვდომია წაკითხვისათვის რეგისტრიდან **EEDATA.** წაკითხული მონაცემები ინახება ამ რეგისტრში, სანამ არ იქნება წაკითხული ახალი მონაცემები ან სანამ მასში არ იქნება შეტანილი მონაცემები ჩაწერისათვის. ქვემოთ მოყვანილია მონაცემების წაკითხვის მაგალითი, **EEDATA**-დან წაკითხული მონაცემები იწერება აკუმულატორში შემდგომი ოპერაციებისათვის.

```
BCF     STATUS, RP0     ; ბანკი 0
```

```
MOVLW   CONFIG_ADDR   ; რაღაც მისამართი CONFIG_ADR
MOVWF   EEDR          ; ვწერთ, როგორც მისამართს წაკითხვისათვის
        BSF     STATUS, RP0     ; ბანკი 1
        BSF     EECON1, RD      ; ვკითხულობთ EEPROM
BCF     STATUS, RP0     ; ბანკი 0
MOVF    EEDATA, W       ; გადავწერთ მონაცემებს W-ში
```

მონაცემების ჩაწერა EEPROM-ში

EEPROM-ში მონაცემების ჩასაწერად აუცილებელია ჯერ ჩაიწეროს მისამართი რეგისტრში **EEADR** და მონაცემები ჩაწერისათვის რეგისტრში **EEDATA,** ხოლო *შემდეგ შევასრულოთ ბრძანებების აუცილებელი თანმიმდევრობა, რომელიც რეკომენდირებულია მწარმოებლის მიერ (გამოყოფილია მუქი შრიფტით):*

```
        BSF     STATUS, RP0     ; ბანკი 1
        BCF     INCON, GIE      ; ყველა წყვეტის აკრძალვა
BSF     EECON1, WREN    ; ბანკი 0

MOVLW   55h             ;
MOVWF   EECON2          ; ვწერთ 55h
MOVLW   AAh             ;
MOVWF   EECON2          ; ვწერთ AAh
BSF     EECON1, WR      ; მონაცემების ჩაწერის სტარტი

BSF     EECON1, GIE     ; წყვეტების ნებართვა
```

*ჩაწერის პროცესი არ იქნება ინიცირებული, თუ არ იქნება შესრულებული 55h-ის და AAh-ის რიგრიგობითი ჩაწერა **EECON2** რეგისტრში მანამ, სანამ დავაყენებთ ბიტს **WR**. მწარმოებელი დაბეჯითებით იძლევა რეკომენდაციას აიკრძალოს ყველა წყვეტა პროგრამის ამ ფრაგმენტის შესრულების მომენტში.* თუ წყვეტები მოწყობილობის მუშაობაში საერთოდ არ გამოიყენება, მაშინ არ არსებობს წყვეტების აკრძალვისა და შემდგომი ნებართვების საჭიროება, ვინაიდან კვების ჩართვის მიხედვით ყველა წყვეტა აკრძალულია.

ბიტი **WREN** ჩამოიყრება აპარატურულად, ამიტომ ის აუცილებელია ჩამოიყაროს პროგრამულად ყველა მონაცემის ჩაწერის დამთავრების შემდეგ. ეს ბიტი იცავს **EEPROM**-ში შემთხვევითი მონაცემების ჩაწერისაგან, მაგალითად პროგრამის მტყუნებისას აუცილებელია

გულმოდგინედ ვადევნოთ თვალი, რომ ეს ბიტი ჩამოყრილი იყოს ყოველთვის, როცა არ ხდება მონაცემების ჩაწერა.

ბიტის **WREN** ჩამოყრა ჩაწერის დაწყებული ციკლის დროს არ იქონიებს გავლენას მის წარმატებულ დასრულებაზე.

*პროგრამორების გავრცელებულ შეცდომას წარმოადგენს ცდა დაიწყოს ჩაწერა **EEPROM**-ის ახალ უჯრედში მანამ, სანამ არ დასრულებულა ჩაწერა წინამდებარეში. შემდგომი მნიშვნელობა უბრალოდ არ იქნება ჩაწერილი. აუცილებელია თავიდან იმის დადასტურება, რომ წინა ციკლი დასრულდა. როცა მთავრდება ჩაწერის ციკლი, აპარატულად ჩამოიყრება ბიტი **WR** და დგება წყვეტის **EEIF** ალამი. მომხმარებელს შეუძლია გამოიყენოს ამ ბიტებიდან ნებისმიერი ჩაწერის დასასრულის მომენტის განსაზღვრისათვის. შედეგად აუცილებელია **EEIF** ბიტი პროგრამულად ჩამოიყაროს.*

ზოგჯერ, ამოცანისდა მიხედვით, შეიძლება მოითხოვებოდეს **EEPROM**-ში მონაცემების ჩაწერის სისწორის შემოწმება. მაგალითად, თუ თქვენ შეიტანეთ სიგნალიზაციაში მიწვდომის ახალი კოდი, ხოლო ის ჩაიწერა არასწორად, მაშინ სამუშაო რეჟიმში გადასვლის შემდეგ სიგნალიზაცია სამუდამოდ გახდება მიუწვდომელი მართვისათვის, რამდენადაც ჩვენი კოდის ნაცვლად შენახულ იქნება შემთხვევითი რიცხვი. ყველაზე ხშირად ხდება ე.წ. ბიტის გაჟონვა, როცა ბიტი, ჩაწერილი როგორც “1”, შემდგომში წაიკითხება, როგორც “0”. შემოწმებისათვის, ჩაწერის შემდეგ სწრაფადვე კითხულობენ შესამოწმებელ უჯრედს და რეგისტრ **EEDATA**-ს შიგთავსს ადარებენ სწორ მნიშვნელობას. თუ მნიშვნელობები არ დაემთხვა, ან ახდენენ ჩაწერის ცდას გარკვეულ რიცხვჯერ ან სწრაფად გამოაქვთ შეტყობინება შეცდომის შესახებ.

კონფიგურაციის სიტყვა CPU

კონფიგურაციის სიტყვა განლაგებულია მისამართზე 2007h. ეს მისამართი იმყოფება პროგრამების სამომხმარებლო მეხსიერების ფარგლებს გარეთ და შედის სპეციალური სამისამართო სივრცის შემადგენლობაში (2000h – 3FFFh), რომელიც მიღწევადია მხოლოდ პროგრამატორისათვის პროგრამირების დროს. კონფიგურაციის სიტყვა შეიცავს 14 ბიტს.

კონფიგურაციის სიტყვა PIC16CR83-ისა და PIC16CR84-ისათვის

bit 13-8 **CP** – პროგრამული კოდის დაცვის ბიტი

1=დაცვა ამორთულია

0=დაცვა დაყენებულია

bit7 **DP**– მონაცემების მეხსიერების(**EEPROM**) დაცვის ბიტი

1=დაცვა ამორთულია

0=დაცვა დაყენებულია

bit6-4 **CP** – პროგრამული კოდის დაცვის ბიტი

1=დაცვა ამორთულია

0=დაცვა დაყენებულია

bit3 **PWRTE** – დაყოვნების ნებართვის ბიტი კვების ჩართვისას

1=დაცვა ამორთულია

0=დაცვა დაყენებულია

bit 2 **WDTE** – მოთვალთვალე ტაიმერის ჩართვის ბიტი

1=მოთვალთვალე ტაიმერი ჩართულია

0=მოთვალთვალე ტაიმერი გამორთულია

bit1-0 **FOSC1-FOSC0** – ტაქტური გენერატორის რეჟიმის არჩევის ბიტი

11= **RC**-გენერატორი

10= **HS** რეზონატორი

01= **XT** რეზონატორი

00= **LP** რეზონატორი

კონფიგურაციის სიტყვა PIC16F83-ისა და PIC16F84-ისათვის

bit 13-4 **CP** – პროგრამული კოდის დაცვის ბიტი

1=დაცვა ამორთულია

0=დაცვა დაყენებულია

bit3 **PWRTE** – დაყოვნების ნებართვის ბიტი კვების ჩართვისას

1=დაცვა ამორთულია

0=დაცვა დაყენებულია

bit 2 **WDTE** – მოთვალთვალე ტაიმერის ჩართვის ბიტი

1=მოთვალთვალე ტაიმერი ჩართულია

0=მოთვალთვალე ტაიმერი გამორთულია

bit1-0 **FOSC1-FOSC0** – ტაქტური გენერატორის რეჟიმის არჩევის ბიტი

11= **RC**-გენერატორი

10= **HS** რეზონატორი

01= **XT** რეზონატორი

00= **LP** რეზონატორი

რეზონატორების აღნიშვნები ამ შემთხვევაში შემდეგია: **XT** – სტანდარტული კვარცული ან კერამიკული რეზონატორი 4 მჰც, **LP** – დაბალსიხშირული (ჩვეულებრივ საათის, 32768 ჰც) რეზონატორი ეკონომიკური ამოცანებისათვის, **HS** – მაღალსიხშირული კვარცული 10 მჰც, **RC**-გენერატორი გარე **RC**-წრედის საფუძველზე.

მწარმოებელი არ იძლევა რეკომენდაციას დაყენებულ იქნას კოდის დაცვის ბიტები დაუპროგრამირებელ კონტროლერზე.

იდენტიფიკატორები

2000h-2003h მისამართებზე განლაგებულია ოთხი სიტყვა, რომელთა დანიშნულებაა კრისტალის ინდივიდუალური საიდენტიფიკაციო კოდის, პროგრამის საკონტროლო ჯამის და სხვა ინფორმაციის შენახვა. ისინი შეიძლება წაკითხული და ჩაწერილნი იყვნენ პროგრამატორის საშუალებით. როცა დაყენებულია პროგრამული კოდის დაცვის ბიტი, მაშინ თითოეული სიტყვის უმცროსი შვიდი ბიტი რჩება მისაწვდომი, ხოლო უფროსები იკითხება, როგორც 0, ამიტომ რეკომენდირებულია შვიდბიტიანი იდენტიფიკატორების გამოყენება, რომელთა წაკითხვა შეიძლება დაცულ ვარიანტში.

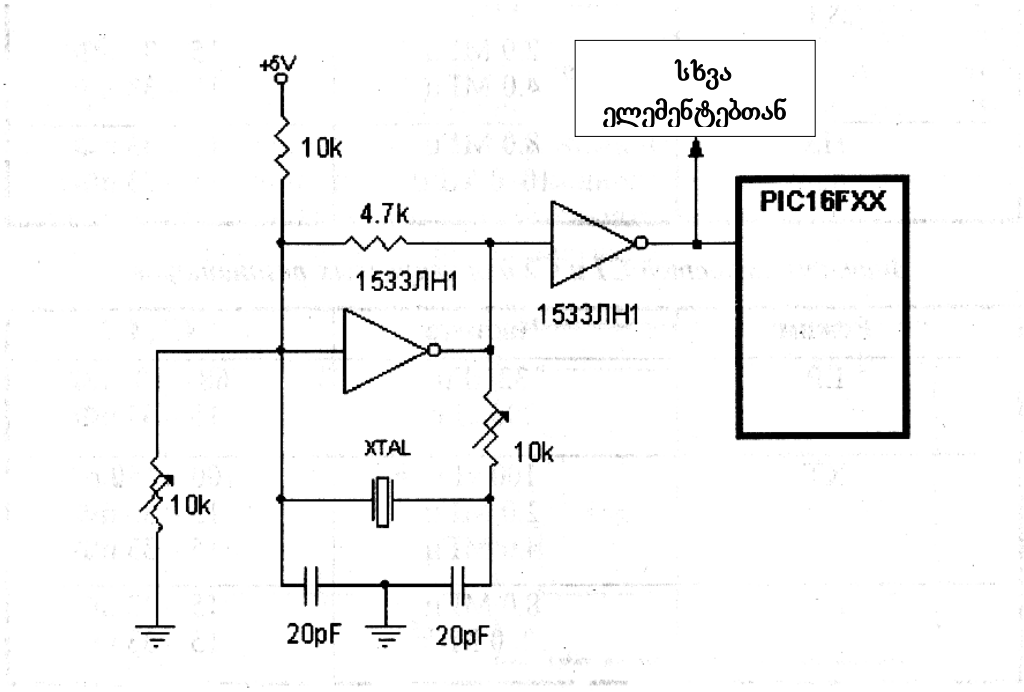
ტაქტური სიხშირის გარე წყაროები

მიკროკონტროლერებს **PIC16F8x** არ გააჩნიათ ჩადგმული ტაქტური გენერატორი, რომელიც მუშაობს გარე ელემენტების გარეშე. ტაქტირებისათვის აუცილებელია ან რეზონატორი, კვარცული ან კერამიკული, ან დამოუკიდებელი ტაქტური გენერატორი.

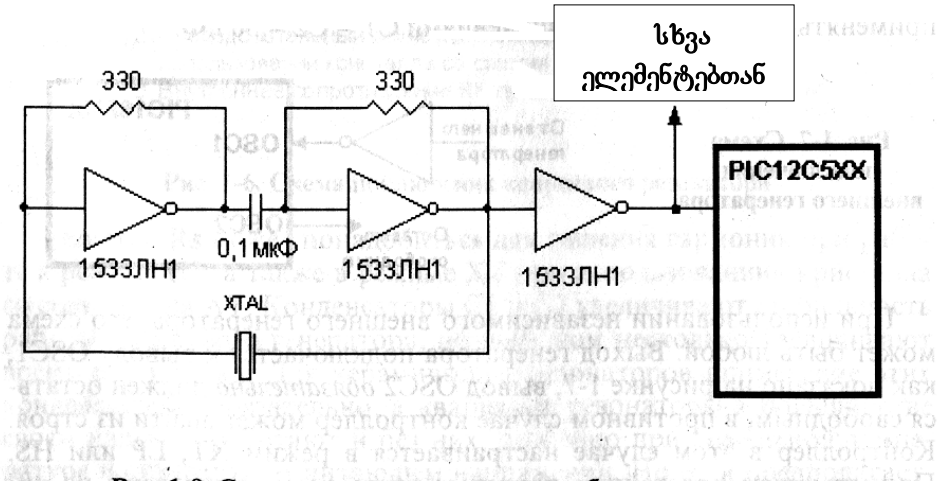
გენერატორი, რომელიც აწყობილია კვარცული ან კერამიკული რეზონატორის საფუძველზე, მოითხოვს გარკვეულ დროს რხევების დასამყარებლად. გენერატორის რეჟიმის დამყარების დროს პროცესორი არ უნდა მუშაობდეს. ამ მიზნისათვის არსებობს გაშვების ჩადგმული ტაიმერი, რომელიც **MCLR** გამოსასვლელზე ლოგიკური ერთიანის დონის გაჩენის და კვების ჩართვის ტაიმერის დაყოვნების გასვლის შემდეგ გარე გენერატორის 1024 ტაქტის განმავლობაში აჩერებს პროცესორს ჩამოყრის მდგომარეობაში. ეს დაყოვნება აუცილებელია ტაქტური გენერატორის სტაბილიზაციისათვის. გაშვების დაყოვნება არ გენერირდება **RC**-წრედის რეჟიმისათვის.

საკუთარი გენერატორის ნორმალური მუშაობისათვის საჭიროა კვარცი, რომელიც მუშაობს პარალელური რეზონანსის სიხშირეზე.

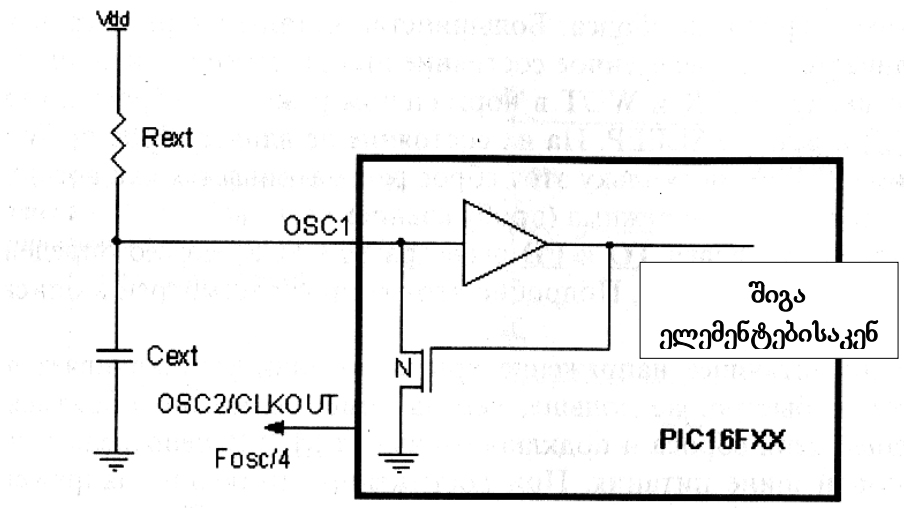
ჩართვის სქემა ნაჩვენებია ნახაზზე 1-6.



ნახ. 7. კვარცის პარალელურ რეზონანსის სიხშირეზე მომუშავე გარე გენერატორის სქემა.



ნახ. 8. კვარცის მიმდევრობითი რეზონანსის სიხშირეზე მომუშავე გარე გენერატორის სქემა.



ნახ. 9. გარე RC გენერატორის სქემა.

რეზისტორი R_S შეიძლება საჭირო გახდეს ჰარმონიკების ჩასახშობად HS რეჟიმში, ასევე XT რეჟიმში კრისტალის გამოყენებისას AT ტიპის ჭრილით. C1 და C2 კონდენსატორები ზრდიან კვარცული გენერატორის მუშაობის სტაბილურობას, მაგრამ ამ დროს რამდენადმე ზრდიან მისი გაშვების დროს. კერამიკული რეზონატორებისათვის ამ კონდენსატორების გამოყენება აუცილებელია, ხოლო კვარცული რეზონატორები უმეტეს შემთხვევებში საიმედოდ მუშაობენ მათ გარეშეც, განსაკუთრებით ოთახის ტემპერატურაზე და სტანდარტული კვების ძაბვის დროს. მაგრამ თუ სავარაუდოა მოწყობილობის მუშაობა უარყოფითი ტემპერატურების ან კვების დაწეული ძაბვის დროს, აუცილებელია C1 და C2 კონდენსატორების დაყენება.

C1 და C2 კონდენსატორების მნიშვნელობები კერამიკული რეზონატორებისათვის

ღეჟიმი	**სიხშირე**	**C1, C2**
XT	455 კჰც 2.0 მჰც 4.0 მჰც	47 - 100 პფ 15 – 33 პფ 15 – 33 პფ
HS	8.0 მჰც 10.0 მჰც	15 – 33 პფ 15 – 33 პფ

C1 და C2 კონდენსატორების მნიშვნელობები კვარცული რეზონატორებისათვის

ღეჟიმი	**სიხშირე**	**C1, C2**
LP	32 კჰც	68 - 100 პფ
	200 მჰც	15 – 33 პფ
XT	100 კჰც	100 - 150 პფ
	2.0 მჰც	15 – 33 პფ
	4.0 მჰც	15 – 33 პფ
HS	8.0 მჰც	15 – 33 პფ
	10.0 მჰც	15 – 33 პფ

4,5ვ-ზე მეტი ძაბვის დროს მწარმოებელი იძლევა რეკომენდაციას გამოვიყენოთ კონდენსატორები მნიშვნელობებით: C1=C2=30 პფ.

დამოუკიდებელი გარე გენერატორის გამოყენებისას, მისი სქემა შეიძლება იყოს ნებისმიერი. გენერატორის გამოსასვლელი უერთდება გამომყვანს **OSC1**, როგორც ნაჩვენებია ნახაზზე 1-7, გამომყვანი **OSC2** აუცილებლად უნდა დარჩეს თავისუფალი, წინააღმდეგ შემთხვევაში კონტროლერი შეიძლება გამოვიდეს მწყობრიდან. კონტროლერი ამ შემთხვევაში აიწყობა XT, LP ან HS რეჟიმში. უმარტივესი ტაქტური გენერატორების სქემის მაგალითები მოყვანილია ნახაზებზე 1-8 და 1-9.

თუ მოხმარება არაკრიტიკულია ტაქტური სიხშირის მნიშვნელობისა და სტაბილობის მიმართ, შეიძლება გამოვიყენოთ იაფი RC-გენერატორი. რეზისტორი და კონდენსატორი წარმოადგენენ გარე ელემენტებს და უერთდებიან ისე, როგორც ნაჩვენებია ნახაზზე 1-10.

რეზისტორის რეკომენდირებული ნომინალი მდებარეობს ზღვრებში 5კომ-დან 100 კომ-მდე. 4კომ-ზე ნაკლები რეზისტორის გამოყენებისას გენერაცია შეიძლება იყოს არასტაბილური ან საერთოდ არ განხორციელდეს.

საკმაოდ დიდი – 1 მომ და მეტი ნომინალის დროს გენერატორის მუშაობაზე გავლენას ახდენენ სქემის გარე აწყობა და ხმაური, ასევე მონტაჟის ტევადობა და პლატის ტენიანობა. მიუხედავად იმისა, რომ გენერატორს შეუძლია მუშაობა საერთოდ გარე კონდენსატორის გარეშეც, რეკომენდირებულია გამოვიყენოთ კონდენსატორი 20 პფ ტევადობით გენერატორის სტაბილურობისა და ხელშეშლებისადმი მდგრადობის გაზრდისათვის.

ტაქტური გენერატორის სიხშირე დამოკიდებულია კვების ძაბვაზე, რეზისტორისა და კონდენსატორის ნომინალებზე და იცვლება კრისტალიდან კრისტალამდე.

RC რეჟიმში **OSC2/CLKOUT** გამომყვანიდან შეიძლება მოიხსნას იმპულსები ტაქტური სიხშირის ერთი მეოთხედი სიხშირით და გამოვიყენოთ ეს იმპულსები დანარჩენი სქემის ტაქტირებისა და სინქრონიზაციისათვის.

*როცა კონტროლერი აწყობილია RC რეჟიმში, მის გამომყვანზე **OSC1/CLKIN** არ შეიძლება გარე გენერატორიდან იმპულსების მიწოდება, ვინაიდან ამან შეიძლება გამოიწვიოს კრისტალის მწყობრიდან გამოსვლა.*

ჩამოყრის ორგანიზაცია

PIC16F8x-სათვის დასაშვებია ჩამოყრის შემდეგი ხუთი ვარიანტი:

- ჩამოყრა კვების ჩართვის მიხედვით
- ჩამოყრა **MCLR** შესასვლელის მიხედვით ნორმალური მუშაობის დროს
- ჩამოყრა **MCLR** შესასვლელის მიხედვით **SLEEP** რეჟიმში
- ჩამოყრა მოთვალთვალე ტაიმერის **(WDT)** გადავსების მიხედვით ნორმალური მუშაობის დროს
- ჩამოყრა მოთვალთვალე ტაიმერის **(WDT)** გადავსების მიხედვით **SLEEP** რეჟიმში

MCLR შესასვლელზე გვაქვს შმიტის ტრიგერი, რომელიც გვიცავს შემთხვევითი მცირე ხანგრძლივობის იმპულსებისაგან, რომლებიც გამოწვეულია სქემის ხმაურით ან გარე აწყობით .

ზოგიერთ რეგისტრზე ჩამოყრის მდგომარეობა გავლენას არ ახდენს. მისი შიგთავსი არ არის განსაზღვრული კვების ჩართვისას და არ იცვლება ჩამოყრის დანარჩენი ვარიანტების დროს. დანარჩენი რეგისტრების უმრავლესობა ყენდება განსაზღვრულ მდგომარეობაში კვების ჩართვისას, **MCLR** და **WDT**-ს შესასვლელების ნორმალურ რეჟიმში ჩამოყრის მიხედვით და **MCLR** შესასვლელის ჩამოყრის მიხედვით რეჟიმში **SLEEP**. მათ მდგომარეობაზე არ მოქმედებს ჩამოყრა **WDT**-ის მიხედვით რეჟიმში **SLEEP**, რამდენადაც ეს ჩამოყრა განიხილება როგორც ნორმალური რეჟიმის აღდგენა (გაღვიძება).

STATUS რეგისტრის **TO** და **PD** ალმების გამოყენებით შეიძლება განისაზღვროს, რითი იქნა გამოწვეული ჩამოყრა. დაწვრილებით ეს საკითხი აღწერილია რეგისტრის აღწერილობაში.

თუ მკვებავი ძაბვა ჩართვისას მყარდება საკმაოდ სწრაფად, არა უმეტეს 70მწმ-ის განმავლობაში, მაშინ შეიძლება ავიცილოთ ჩამოყრის გარე წრედი და **MCLR** შესასვლელი მივაერთოთ კვების პლუსურ სალტესთან. კვების ძაბვის მიერ 1.2-1.7ვ დონის მიღწევისას ფორმირდება ჩამოყრის შიგა სიგნალი და დაიწყება დროითი დაყოვნების ათვლა სპეციალური შიდა ტაიმერით **PWRT** (Power-up timer). ამ დროის განმავლობაში მკვებავმა ძაბვამ უნდა მიაღწიოს ნორმალურ სამუშაო დონეს. ტაიმერი **PWRT** მუშაობს დამოუკუდებელი ჩადგმული **RC**-გენერატორის გარეშე, დაყოვნების დრო შეადგენს 72 მწმ და შეიძლება რამდენადმე იცვლებოდეს

კრისტალიდან კრისტალამდე, ასევე ტემპერატურისდა მიხედვით. **PWRT** ტაიმერის დაყოვნების დასრულების შემდეგ ირთვება ძირითადი ტაქტური გენერატორის გაშვების ტაიმერი, მაგრამ მისი ტაქტირება ხდება უშუალოდ ამ გენერატორიდან და ითვლის 1024 იმპულსს.

ტაიმერი **PWRT** შეიძლება ჩართული ან გამორთული იყოს **PWRTE** ბიტის ცვლილებით კონფიგურაციის სიტყვაში.

თუ შესაძლებელია სიტუაცია, როცა მკვებავი ძაბვა მდორედ მცირდება სამუშაო დონის ქვემოთ, მაგრამ არ აღწევს ნულს, ხოლო შემდეგ აღდგება, ჩამოყრის დონის საიმედო ფორმირებისათვის აუცილებელია სხვა სქემის გამოყენება. ნახაზზე 1-12 ნაჩვენებია სქემა სტაბილიტრონით. ჩამოყრა ფორმირდება, როცა მკვებავი ძაბვა მცირდება V_d+0.7 დონემდე, სადაც V_d სტაბილიტრონის სტაბილიზაციის ძაბვაა.

ნახაზზე 1-13 მოყვანილია ანალოგიური სქემა, მაგრამ სტაბილიტრონის გარეშე, რომელიც ხასიათდება რეაქციის უფრო დაბალი სიზუსტით მკვებავი ძაბვის დაწევაზე. ტრანზისტორი იკეტება, როცა კვების ძაბვა მცირდება დონემდე, რომელიც გამოითვლება ფორმულით. შეიძლება აგრეთვე სპეციალური მიკროსქემის –კვების სუპერვაიზერის გამოყენება, რომელიც აფორმირებს ჩამოყრის იმპულსს კვების ძაბვის დაწევისას.

წყვეტის ორგანიზაცია

PIC16F8x-მიკროკონტროლერებს გააჩნიათ წყვეტის ოთხი წყარო:

ა) გარე **RB0/INT** შესასვლელის მიხედვით

ბ) გარე პორტი **B**-ს **RB4-RB7** ხაზებიდან ერთერთის მდგომარეობის ცვლილების მიხედვით

გ) შიგა **TMR0** ტაიმერი/მთვლელის გადავსების მიხედვით

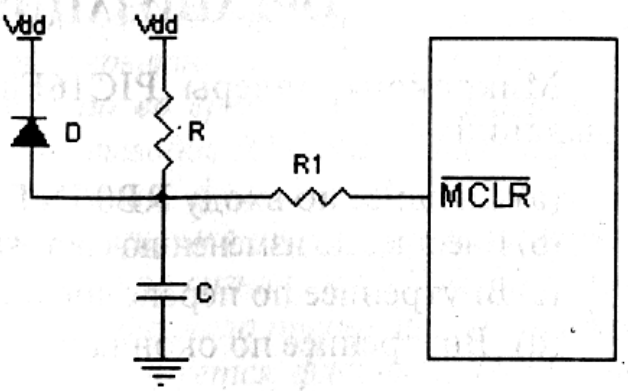
დ) შიგა **EPROM**-ში ჩაწერის დასრულების მიხედვით.

თითოეული ა), ბ) და გ) წყვეტისათვის არსებობს ალამ-ბიტი რეგისტრში **INTCON**, რომელიც იძლევა სიგნალს მოთხოვნის კონკრეტული სახის მოსვლის შესახებ. დ) წყვეტისათვის ალამ-ბიტი ინახება რეგისტრში **EECON1**. აანალიზებს რა წყვეტის დამუშავების ქვეპროგრამის ბიტებს, 0004h მისამართიდან დაწყებული, განსაზღვრავს წყვეტის წყაროს. ამას გარდა, რეგისტრში **INTCON** მოთავსებულია გლობალური წყვეტის ბიტი **GIE** და თითოეული წყვეტის ინდივიდუალური აკრძალვის ბიტები. როცა ბიტი **GIE** ჩამოყრილია, ყველა წყვეტა აკრძალულია. კვების ჩართვისას ბიტი **GIE** გაჩუმების მიხედვით ჩამოყრილია.

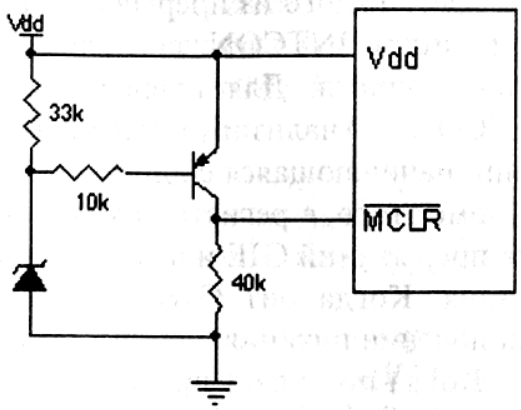
როდესაც მოდის წყვეტა, ბიტი **GIE** ჩამოიყრება იმისათვის, რომ არ დავუშვათ ახალი წყვეტა *(რამდენადაც **PIC16F8x**-მიკროკონტროლერისთვის არსებობს წყვეტის მხოლოდ ერთი ვექტორი)*, დაბრუნების მისამართი იტვირთება სტეკში, ხოლო პროგრამულ მთვლელში იტვირთება ვექტორ-

მისამართი 0004h. წყვეტის დამუშავების **RETFIE** ქვეპროგრამიდან დაბრუნების ინსტრუქცია აყენებს ბიტს **GIE** ერთიანში, რითაც იძლევა წყვეტის ნებართვას.

გარე წყვეტის მოთხოვნის დამუშავებამ შეიძლება დაიკავოს პროცესორის სამი ან ოთხი ბრძანებითი ციკლი, იმიდა მიხედვით, თუ დროის რა მომენტშია აღმოჩენილი მოთხოვნა.



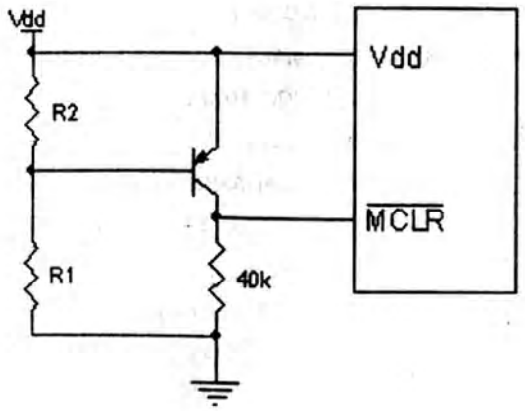
ნახ. 10. ჩამოყრის გარე წრედი.



ნახ. 11. სიგნალ "ჩამოყრა"-ს მაფორმირებელი სქემა, კვების ძაბვის შემცირებისას სტაბილიტრონის გამოყენებით.

წყვეტის ყველა ალამ-ბიტი ჩამოყრილი უნდა იყოს პროგრამულად მანამ, სანამ წყვეტები კვლავ იქნება დაშვებული **GIE** *ბიტის დაყენებით. წინააღმდეგ შემთხვევაში შეიძლება ადგილი ჰქონდეს წყვეტაში განმეორებით შესვლას და პროგრამის ჩაციკლვას. ასევე უნდა გვახსოვდეს, რომ წყვეტების ალამ-ბიტები ყენდება წყვეტის მოვლენების შესაბამისად იმისდა მიუხედავად,*

დაშაბლონებულია ეს წყვეტები თუ არა. ამ ფაქტების იგნორირებას მივყავართ პროგრამირების საკმაოდ ცნობილ შეცდომებამდე, რომლებმაც შეიძლება თავი იჩინონ მოგვიანებით.



ნახ. 12. სიგნალ "ჩამოყრა"-ს მაფორმირებელი სქემა, კვების ძაბვის შემცირებისას სტაბილიტრონის გარეშე.

მაგალითად, დაგეგმილია, რომ წყვეტა ***RB0/INT*** *შესასვლელის მიხედვით დაშვებულია ყოველთვის, ხოლო წყვეტა* ***RB4-RB7*** *შესასვლელებზე მდგომარეობის ცვლილების მიხედვით ნებადართული ხდება პერიოდულად, პროგრამის ალგორითმის მიხედვით. მაგრამ პროგრამირების შეცდომების შედეგად წყვეტის დამუშავების ქვეპროგრამა არ ითვალისწინებს პროგრამის მუშაობის მიმდინარე კონტექსტს, არ აანალიზებს დაშვებულია თუ არა მოცემულ მომენტში წყვეტა* ***RB4-RB7****-ის მიხედვით, ახდენს მხოლოდ ალამ-ბიტების ტესტირებას. წარმოვიდგინოთ, რომ პროგრამის მუშაობის დროს მოხდა დონის ცვლილება რომელიმე* ***RB4-RB7*** *გამომყვანზე. მიუხედავად იმისა, რომ ეს წყვეტა იქნა შენიღბული, დაყენებულ იქნება* ***RBIF*** *ალამ-ბიტი. თუ წყვეტის დამუშავების ქვეპროგრამა ახდენს პირველად* ***RBIF*** *ბიტის ტესტირებას, მაშინ* ***RB0/INT****-ზე სიგნალის მიხედვით წყვეტაში შესვლისასაც კი მიღებული იქნება მცდარი გადაწყვეტილება რომ მოხდა წყვეტა* ***RB4-RB7*** *შესასვლელების მიხედვით. და პირიქით, თუ თავდაპირველად ხდება* ***INTE*** *ალამ-ბიტის ტესტირება იმაზე, მაშინ შეიძლება მიღებულ იქნეს მცდარი გადაწყვეტილება* ***RB0/INT****-ის მიხედვით წყვეტის შესახებ, როცა რეალურად წყვეტა გამოწვეულია* ***RB4-RB7*** *შესასვლელების მდგომარეობის ცვლილებით. ამ შეცდომის თავიდან ასაცილებლად, საჭიროა არა მარტო ალამ-ბიტების, არამედ შესაბამისი შემნიღბავი ბიტის ტესტირებაც.*

მეორეს მხრივ, ხანდახან მოსახერხებელია გამოვიყენოთ ალამ-ბიტები იმ მოვლენების პროგრამული აღმოჩენებისათვის, რომლებიც არ მოითხოვენ წყვეტას მათი დასრულების

მომენტისათვის. ამ შემთხვევაში ახდენენ წყვეტების დაშაბლონებას და პერიოდულად, საჭირო მომენტში ახდენენ ალამ-ბიტის ტესტირებას.

წყვეტა **RB0/INT** შესასვლელის მიხედვით შეიძლება მოხდეს შესასვლელი სიგნალის ზრდის მიხედვით, თუ **OPTION_REG** რეგისტრის **INTEDG** ბიტი დაყენებულია ან ვარდნის მიხედვით, თუ **INTEDG** ჩამოყრილია. როცა ხდება დონის უკანა გადასვლა, ყენდება **INTCON** რეგისტრის **INTF** ბიტი. წყვეტა შეიძლება დაშაბლონდეს **INTE** ბიტის ჩამოყრით.

*ასეთმა წყვეტამ კონტროლერი შეიძლება გამოიყვანოს **SLEEP** რეჟიმიდან, თუ ამ რეჟიმში შესვლამდე **INTE** ბიტი არ იყო დაყენებული 1-ში. **INTE** ბიტის მდგომარეობა განსაზღვრავს, გადავა პროცესორი ვექტორ-მისამართზე 0004h გაღვიძების შემდეგ თუ პროგრამის შესრულება გაგრძელდება მიმდინარე მისამართიდან.*

RB4-RB7 გამომყვანებიდან ერთ-ერთზე დონის ცვლილების აღმოჩენისათვის და წყვეტაზე მოთხოვნის ფორმირებისათვის აუცილებელია, რომ იმპულსის ხანგრძლივობა არ იყოს პროცესორის ერთ ბრძანებით ციკლზე ნაკლები.

PIC16F8x-მიკროკონტროლერებს არ გააჩნიათ სპეციალური ბრძანებები **PUSH** და **POP** აკუმულატორისა და სპეციალური რეგისტრების შიგთავსის შესანახად წყვეტის დამუშავების დროს მათი შემდგომი აღდგენით. ავტომატურად ინახება მხოლოდ პროგრამული მთვლელის მნიშვნელობა ქვეპროგრამიდან დასაბრუნებლად. ამიტომ მომხმარებელმა პროგრამულად უნდა შეინახოს აკუმელატორის (**W** სამუშაო რეგისტრის) და **STATUS** რეგისტრის მნიშვნელობები. პროგრამის მაგალითი მოყვანილია ქვემოთ:

```
PUSH  MOVWF  W_TEMP    ; შევინახეთ W-ს მნიშვნელობა W_TEMP-ში
      SWAPF STATUS,W   ; ჩავტვირთეთ STATUS რეგ. მნიშვნელობა
      MOVWF  ST_TEMP   ; W-ში და შევინახეთ W რეგისტრში
                       ; ST_TEMP-ში
      INTR             ; აქ განლაგდება წყვეტის დამუშავების თქვენი
                       ; ქვეპროგრამის სხეული. თუ საჭიროა,
                       ; შეიძლება შეცვალოთ მეხსიერების
                       ; მიმდინარე ბანკი
POP   SWAPF  ST_TEMP,W ; ჩავტვირთეთ შენახული მნიშვნელობა
      MOVWF STATUS     ; და აღვადგინეთ იგი რეგისტრში STATUS
                       ; ამ დროს აღდგენილია წინა ბანკი
      SWAPF  W_TEMP,f  ; აღდგენილია W რეგისტრის მნიშვნელობა
      SWAPF  W_TEMP,W  ; ST_TEMP-ში
```

SWAPF ბრძანების გამოყენება მნიშვნელობების ჩასატვირთად და ამოსატვირთად განპირობებულია იმით, რომ ეს ბრძანება არ ცვლის **STATUS** რეგისტრის ნულოვანი შედეგის **Z** ალამ-ბიტის მდგომარეობას. **MOVF** ბრძანების გამოყენება შეამოკლებდა პროგრამულ კოდს, მაგრამ მაშინ აკუმულატორის მნიშვნელობის შენახვისას შეიძლება შეცვლილ იქნას **Z** ბიტის მდგომარეობა, რაც ზოგად შემთხვევაში დაუშვებელია. კორექტული პროგრამირება მოითხოვს, რომ წყვეტის ქვეპროგრამიდან დაბრუნებისას **W** და **STATUS** რეგისტრების მნიშვნელობები აღდგენილ იქნენ აბსოლუტურად ზუსტად.

მოთვალთვალე ტაიმერი

მოთვალთვალე ტაიმერი წარმოადგენს ჩადგმული **RC**-გენერატორისა და მთვლელის კომბინაციას, რომლის გადავსების დროს ხდება პროცესორის ჩამოყრა. ჩადგმული გენერატორი არ მოითხოვს შიდა წრედებს და მუშაობს მაშინაც კი, როცა პროცესორის ტაქტური გენერატორი გაჩერებულია რეჟიმში **SLEEP**. გენერატორსა და მთვლელს შორის შეიძლება ჩართული იყოს წინასწარი გამყოფი, რომლის პარამეტრები მოიცემა **OPTION_REG** რეგისტრის ბიტებით.

მოთვალთვალე ტაიმერის გაძლების დრო დამოკიდებულია ტემპერატურაზე, კვების ძაბვასა და წინასწარი გამყოფის გაყოფის კოეფიციენტზე. მიერთებული წინასწარი გამყოფითა და მაქსიმალური გაყოფის კოეფიციენტით 1:128 გაძლებამ შეიძლება მიაღწიოს 2,5 წმ. ნომინალური დაყოვნება წინასწარი გამყოფის გარეშე შეადგენს დაახლოებით 18 მწმ.

*როცა ამუშავდება მოთვალთვალე ტაიმერი, მას არ გადაჰყავს **MCLR** გამომყვანი დაბალ დონეზე. ჩამოყრა ხორციელდება მხოლოდ მიკროკონტროლერის შიგა წრედების მიხედვით.*

მოთვალთვალე ტაიმერის მთავარ დანიშნულებას წარმოადგენს მოწყობილობის აპარატურულ ამოვარდნებთან ბრძოლა. მოწყობილობებში, რომლებიც მიდრეკილნი არიან იმპულსური ხელშეშლებისა და მაღალსიხშირული აწყობის დიდ დონეებთან, განსაკუთრებით ხშირად ზიანდება **OPTION_REG** რეგისტრის შიგთავსი. ამის გარდა, შესაძლებელია გაუთვალისწინებელი ამოვარდნები მიკროკონტროლერის სხვა მოდულებისა და მოწყობილობის პერიფერიული ნაწილის მუშაობაში. შედეგად შეიძლება მოხდეს პროგრამის "დაკიდება". თუ მოთვალთვალე ტაიმერი ჩართულია, მაშინ ნორმალურად მომუშავე პროგრამამ პერიოდულად უნდა გაანულოს მოთვალთვალე ტაიმერის მთვლელი ისე, რომ არ დაუშვას პროცესორის ჩამოყრა. როცა ხდება ამოვარდნა, მოთვალთვალე ტაიმერი წყვეტს პროგრამულად განულებას, მისი გადავსების შემდეგ ხდება პროცესორის ჩამოყრა და მოწყობილობის ხელახალი ინიციალიზაცია. ამოვარდნებისაგან მაქსიმალური დაცულობის მისაღწევად რეკომენდირებულია შესრულდეს შემდეგი პირობები:

- ავირჩიოთ მოთვალთვალე ტაიმერის შესაძლო მცირე ინტერვალი და ჩავაგდოთ პროგრამის სხვადასხვა ადგილებში;

- მოწყობილობის აპარატურული ნაწილი მთლიანად უნდა ინიციალიზირდებოდეს პროცესორის ნებისმიერი ჩამოყრის დროს, და არა მხოლოდ კვების ჩართვისას და **"RESET"** ღილაკის დაჭერისას.

ბრძანებები **CLRWDT** და **SLEEP** ანულებენ მოთვალთვალე ტაიმერის მთვლელს და წინასწარი გამყოფის მთვლელს, თუ ის მიერთებულია მოთვალთვალე ტაიმერთან. ამგვარად თავიდან ხდება პროცესორის ჩამოგდების აცილება და იწყება ახალი დროითი ინტერვალის ფორმირება.

მოთვალთვალე ტაიმერს ასევე შეუძლია გამოიყვანოს პროცესორი **SLEEP** რეჟიმიდან. ამ დროს მოხდება არა ჩამოყრა, არამედ პროგრამის ნორმალური შესრულების გაგრძელება.

მოთვალთვალე ტაიმერის მუშაობა დაშვებული და აკრძალული შეიძლება იყოს მხოლოდ პროგრამატორის საშუალებით, WDTE კონფიგურაციის ბიტის დაყენებით ან ჩამოგდებით.

ენერგოდამზოგი რეჟიმი SLEEP

მიკროკონტროლერი გადადის რეჟიმში **SLEEP** სპეციალური **SLEEP** ბრძანების შესრულებისას. თუ მოთვალთვალე ტაიმერი ჩართულია, მაშინ იგი ნულდება და იწყებს დაყოვნების ათვლას თავიდან. **STATUS** რეგისტრში ხდება **PD** ბიტის ჩამოყრა და ყენდება ბიტი **TO**. ტაქტური გენერატორი გამოირთვება. პორტების გამომყვანები ინარჩუნებენ მდგომარეობას, რომელიც არსებობდა უშუალოდ **SLEEP** ბრძანების შესრულებამდე.

იმისათვის, რომ ამ რეჟიმში მოხმარებული დენი იყოს მინიმალური, გამოსასვლელზე აწყობილ პორტებს უნდა ჰქონდეთ მნიშვნელობები, რომელთა დროსაც მათში არ გაივლის დენი გარე წყაროებიდან. პორტების ხაზები, რომლებიც აწყობილია შესასვლელზე და გააჩნიათ მაღალი შესასვლელი წინაღობა, შეერთებული უნდა იყოს რიგის გარე რეზისტორების გავლით კვების სალტესთან ან საერთო სადენთან იმისათვის, რომ გამოირიცხოს გადართვის შიდა დენები, რომლებიც გამოწვეულია შესასვლელებზე ქაოტური აწყობით. ასევე უნდა მოვიქცეთ შესასვლელთან **RA4/T0CK1**. MCLR გამოსასვლელზე უნდა იყოს მაღალი დონე.

SLEEP რეჟიმის გაღვიძება

.

რეჟიმიდან პროცესორი შეიძლება გამოყვანილ იქნას სამი სხვადასხვა საშუალებით:

ა) გარე ჩამოყრით MCLR შესასვლელის მიხედვით;

ბ) მოთვალთვალე ტაიმერის გადავსების დროს (თუ ის ჩართულია);

გ) წყვეტით **RB0/INT** შესასვლელის მიხედვით, **RB4-RB7** შესასვლელების მდგომარეობის ცვლილების მიხედვით და **EEPROM**-ში ჩაწერის დასრულების მიხედვით.

ა) მოვლენა იწვევს პროცესორის ჩამოყრას და საწყისი მისამართიდან პროგრამის შესრულებას. დანარჩენ ორ მოვლენას მივყავართ პროგრამის შესრულების გაგრძელებასთან. **SLEEP** ბრძანების შესრულების დროს პროცესორი ტვირთავს ბრძანებების ბუფერში შემდეგ ბრძანებას **(PC+1)**. იმისათვის, რომ პროცესორი გამოვიდეს **SLEEP** რეჟიმიდან წყვეტის მიხედვით, ის დაშვებული უნდა იქნეს შესაბამისი ბიტებით.

*წყვეტების მიხედვით გაღვიძების თანმიმდევრობა დამოკიდებულია მხოლოდ **GIE** ბიტის მდგომარეობაზე. თუ ეს ბიტი ჩამოყრილია, მაშინ მისი გაღვიძების შემდეგ სრულდება ბრძანება, რომელიც მოყვება ბრძანება **SLEEP**-ს (და უკვე ჩატვირთულია ბუფერში) და შემდეგ უკვე რიგის მიხედვით. თუ **GIE** ბიტი დაყენებულია 1-ში, მაშინ თავდაპირველად სრულდება ბრძანება, რომელიც ჩატვირთულია ბუფერში, ხოლო შემდეგ პროცესორი გადადის წყვეტის მისამართ-ვექტორზე 0004h. თუ ბრძანების შესრულება, რომელიც მოსდევს **SLEEP**-ს, წყვეტის მიხედვით გამოსვლისას არასასურველია, მაშინ **SLEEP** ბრძანებას უმალვე უნდა მოყვეს ბრძანება **NOP**.*

როცა წყვეტების გლობალური ნებართვის ბიტი **GIE** ჩამოყრილია და ჩნდება სიტუაცია, რომლის დროსაც წყვეტებიდან ნებისმიერის დროს ერთდროულად დაყენებულია ნებართვის ბიტი და ალამ-ბიტი, ადგილი აქვს ერთ-ერთ შემდეგ მოვლენათაგანს:

თუ ადგილი ჰქონდა წყვეტის ნებართვას, მაშინ მოწყობილობა ნელ-ნელა იღვიძებს **SLEEP** რეჟიმიდან. ბრძანება **SLEEP** მუშავდება მთლიანად იმ შემთხვევაშიც კი, როცა წყვეტას ადგილი ჰქონდა მოქმედების მომენტში. მოთვალთვალე ტაიმერი და წინასწარი გამყოფი სუფთავდება, ყენდება ბიტი **TO** და ჩამოიყრება ბიტი **PD**.

მაშინაც კი, თუ მოვახდენთ ალამ-ბიტების ტესტირებას **SLEEP** ბრძანების შესრულების დასაწყისამდე, შესაძლებელია სიტუაცია, როცა რაღაც ალამ-ბიტი დაყენდება მანამდე, სანამ დასრულდება **SLEEP** ბრძანება. იმისათვის, რომ დავრწმუნდეთ, შესრულდა თუ არა **SLEEP** ბრძანება, მოახდინეთ **PD** ბიტის ტესტირება. თუ **PD** ბიტი დაყენებულია, ე.ი. SLEEP ბრძანება შესრულებულ იქნა.

ბრძანებების სისტემა

PIC16Fxx მიკროკონტროლერების ყოველი ბრძანება წარმოადგენს 14-ბიტიან სიტყვას, რომელიც შედგება ბრძანების კოდისგან **(OPCODE)** და ერთი ან რამდენიმე ოპერანდისაგან. ბრძანების აღწერისას გამოყენებულია ბრძანების ველების შემდეგი აღნიშვნები, რომლებსაც ჩვეულებრივ იყენებენ მომხმარებლები (იხ. ცხრილი)

ველი	აღნიშვნის განმარტება
f	სპეციალური რეგისტრის ან მომხმარებლის რეგისტრის მისამართი. მნიშვნელობების დიაპაზონი 0x00-დან 0x7F-მდე. ფირმული ასემბლერი უშვებს ციფრული მნიშვნელობების მაგივრად უშუალოდ ადრე განსაზღვრული რეგისტრების სახელების გამოყენებას.
W	სამუშაო რეგისტრი (აკუმულატორი)
b	ბიტური მისამართი, რომელიც გამოიყენება 8-ბიტიან რეგისტრთან, და რომელიც მიუთითებს იმ ბიტზე რეგისტრის შიგნით, რომელთანაც სრულდება ბიტური ოპერაცია. ასემბლერულ ტექსტში აღნიშნავს კონსტანტას, წარმოდგენილს ორობით აღრიცხვაში.
k	ლიტერალი, კონსტანტა ან ჭდე
d	ოპერაციის შედეგის მიმღების მიმთითებელი. თუ d=0, შედეგი ინახებაW-ში, თუ d=1, შედეგი ინახება რეგისტრში, რომელიც მითითებულია ბრძანებაში. გაჩუმების მიხედვით d=1. *არცერთი სხვა რეგისტრი, გარდა იმისა, რაც გამოყენებულია ოპერაციაში, არ შეიძლება გამოყენებულ იქნას მიმღებად.* ფირმული ასემბლერი მეტი თვალსაჩინოებისათვის უშვებს 0 და 1 მნიშვნელობების მაგივრად შესაბამისად w და f სიმბოლოების გამოყენებას.
label	ჭდის სახელი
TOS	Top Of Stack – სტეკის მწვერვალი
PC	პროგრამული მთვლელი
PCLAT CH	პროგრამული მთვლელის უფროსი მნიშვნელობების რეგისტრ-ჩამკეტი
GIE	წყვეტების გლობალური აკრძალვა/დაშვების ბიტი
WDT	WatchDog Timer – მოთვალთვალე ტაიმერი
<u>TO</u>	ტაიმერის გადავსების ალამ-ბიტი
<u>PD</u>	შემცირებული ენერგომოხმარების SLEEP რეჟიმის ალამ-ბიტი
dest	მიმღები. რეგისტრი W ან სხვა ბრძანებაში მითითებული რეგისტრი

[]	დასაშვები მნიშვნელობები
()	შიგთავსი
→	მნიშვნელობის გადაცემა
<>	ბიტის რიგითი ნომერი 8-ბიტიან რეგისტრში, 0-დან 7-მდე
e	"თავის თავში შეიცავს მნიშვნელობას"
italic	მნიშვნელობები ან პირობები, რომლებსაც განსაზღვრავს მომხმარებელი

კონტროლერის ბრძანებების მთელი ჩამონათვალი მოიცავს ბაიტურ, ბიტურ, ლიტერალურ ბრძანებებს და გადასვლის ბრძანებებს. ბაიტური ბრძანებები მოიცავენ ბრძანების კოდს, რეგისტრ-წყარო "f"-ის მისამართის მიმთითებელს და მიმღების მიმთითებელს "d":

13	8 7 6	0
OPCODE	d	f (რეგისტრის მისამართი)

მაგალითად,

```
MYREG     EQU     0x1D
...
...
ADDWF     MYREG, f
```

ამ შემთხვევაში, მომხმარებლის რეგისტრი MYREG, რომელიც განისაზღვრება 1D მისამართის მიხედვით, ემატება აკუმულატორის შიგთავსს და შედეგი თავსდება რეგისტრში MYREG.

ბიტური ბრძანებები ცვლიან ან ამოწმებენ კონკრეტული მოცემული ბიტის მნიშვნელობას რეგისტრში. სამი "b" ბიტის საშუალებით ბრძანებაში უთითებენ "f" რეგისტრის 8 ბიტიდან ერთ-ერთს:

13	10 9	7 6	0
OPCODE	d	f (რეგისტრის მისამართი)	

მაგალითად,

```
MYREG     EQU     0x1D
...
CLRF      MYREG, f
...
BSF       MYREG, 5
```

ამ მაგალითში მომხმარებლის რეგისტრი MYREG თავდაპირველად სუფთავდება (ყველა ბიტი ყენდება 0-ში) ბრძანებით CLRF, შემდეგ ამ რეგისტრის მეხუთე ბიტი ყენდება 1-ში ბრძანებით BSF.

ლიტერალური ბრძანებები და გადასვლის ბრძანებები მოიცავენ ბრძანების კოდს და 8-ან 11-ბიტიან კონსტანტას "k":

13	8 7 0
OPCODE	k

მაგალითად,

```
MOVLW     0xBFh
```

ბრძანება MOVLW (MOV Literal to W) ჩატვირთავს აკუმულატორში თექვსმეტთანრიგიან მნიშვნელობას BFh.

CALL და GOTO ბრძანებებისათვის გამოიყენება ბრძანების სხვა ფორმატი:

13	11 10 0
OPCODE	k

ამ შემთხვევაში k აღნიშნავს პროგრამული მისამართის 11-ბიტიან მნიშვნელობას, რომელიც იტვირთება პროგრამულ მთვლელში ბრძანების შესრულებისას. ასემბლერულ ტექსტში გადასვლის მისამართი აღინიშნება ჭდით, და k-ს მნიშვნელობა გამოითვლება ასემბლერით.

ბრძანებების უმეტესობა სრულდება ერთი მანქანური ციკლის განმავლობაში, გადასვლისა და ქვეპროგრამის გამოძახების ბრძანებების გარდა, რომლებიც მოითხოვენ პროგრამული მთვლელის მნიშვნელობების შეცვლას. ეს ბრძანებები სრულდება ორი მანქანური ციკლის განმავლობაში. ერთი მანქანური ციკლი ხანგრძლივობით უდრის ტაქტური გენერატორის რხევების ოთხ პერიოდს. ასე მაგალითად, კვარცული გენერატორის სტანდარტული 4მჰც სიხშირის დროს მანქანური ციკლის ხანგრძლივობა შეადგენს 1მიკროწამს. გადასვლის ბრძანებები სრულდება 2მწმ-ს განმავლობაში.

პროგრამის ცალკეული უბნების შესრულების დროის გამოთვლისას, რომლებიც განსაკუთრებით კრიტიკულია დროის მიხედვით, უნდა გვახსოვდეს, რომ გადასვლის ბრძანებები პირობის მიხედვით სრულდება ერთი ან ორი ციკლის განმავლობაში, იმისდა მიხედვით, სრულდება თუ არა პირობა. მაგალითად, შემდეგ ფრაგმენტში

```
. . .
BTFSC     MYREG, 5
DECF      MYREG,
ADDWF     MYREG, f
. . .
```

ბრძანება ***BTFSC(Bit Test F, Skip if Clear)****ამოწმებს* ***MYREG*** *რეგისტრის მეხუთე ბიტს და თუ ეს ბიტი 1-ის ტოლია, მაშინვე ამ ბრძანების შემდეგ სრულდება შემდეგი,* ***DECF MYREG****. ბიტის ტესტირების ბრძანება შესრულებულია ერთი მანქანური ციკლის განმავლობაში. თუ MYREG რეგისტრის მეხუთე ბიტი გასუფთავებულია, ხდება ერთ ბრძანებაზე გადახტომა* ***ADDWF MYREG*** *ბრძანებაზე. ამ შემთხვევაში ბიტის ტესტირების ბრძანება სრულდება ორი მანქანური ციკლის განმავლობაში. თუ ტესტირების პროგრამები მრავალჯერ სრულდება ციკლში, მაშინ სხვადასხვა პირობებისას შეიძლება დაგროვდეს მნიშვნელოვანი ცდომილებები პროგრამის ფრაგმენტის შესრულების მოსალოდნელი დროის გამოთვლაში.*

ბრძანებების მოკლე ჩამონათვალი

ოპერანდის მნემონიკა	მნემონიკის გაშიფვრა	ციკლების რაოდენობა	შეცვლადი რეგისტრები	შენიშვნა
		აიტური ოპერაციები რეგისტრებთან		
ADDWF f, d	Add W and F	1	C,DC,Z	1,2
ANDWF f, d	AND W with F	1	Z	1,2
CLRF f	Clear F	1	Z	2
CLRW	Clear W	1	Z	
COMF f, d	Complement F	1	Z	1,2
DECF f, d	Decrement F	1	Z	1,2
DECFSZ f, d	Decrement F, Skip if Zero	1(2)		1,2,3
INCF f, d	Increment F	1	Z	1,2
INCFSZ f, d	Increment F, Skip if Zero	1(2)		1,2,3
IORWF f,d	Inclusive OR W with F	1	Z	1,2
MOVF f,d	Move F	1	Z	1,2
MOVWF f	Move W to F	1		
NOP	No Operation	1		
RLF f, d	Rotate Left F through carry	1	C	1,2
RRF f, d	Rotate Right F through carry	1	C	1,2
SUBWF f, d	Subtract W from F	1	C,DC,Z	1,2
SWAPF f, d	Swap nibbles in F	1		1,2
XORWF f, d	Exclusive OR W with F	1	Z	1,2
		ბიტური ოპერაციები რეგისტრებთან		
BCF f, b	Bit Clear F	1		1,2
BSF f, b	Bit Set F	1		1,2
BTFSC f, b	Bit Test F, Skip if Clear	1(2)		3
BTFSS f, b	Bit Test F, Skip if Set	1(2)		3
		ოპერაციები ლიტერალებსა და გადასვლებთან		
ADDLW k	Add Literal and W	1	C,DC,Z	
ANDLW k	AND Literal with W	1	Z	
CALL k	Call subroutine	2		
CLRWDT	Clear Watchdog Timer	1	TO,PD	
GOTO k	GO To address	2		
IORLW k	Inclusive OR Literal with W	1	Z	

MOVLW k	Move Literal to W	1		
RETFIE	Return From Interrupt	2		
RETLW k	Return with Literal in W	2		
RETURN	Return from subroutine	2		
SLEEP	Sleep into standby mode	1	TO,PD	
SUBLW k	Subtract W from Literal	1	C,DC,Z	
XORLW k	Exclusive OR Literal with W	1	Z	

შენიშვნა:

1) როცა ხდება შეტანა-გამოტანის პორტების წაკითხვა-მოდიფიკაცია-ჩაწერა, ყოველთვის ხდება გამოსასვლელებზე ძაბვების რეალური მნიშვნელობების წაკითხვა, იმისადა მიუხედავად, როგორ არის აწყობილი პორტის ხაზები და რა არის ჩაწერილი ტრიგერ-საკეტში. ოპერაცია სრულდება რეალურ წაკითხულ მნიშვნელობებზე და შედეგი ჩაიწერება საკეტში.

2) როდესაც ეს ინსტქუქცია სრულდება რეგისტრზე TMR0 და d=1, წინასწარი გამყოფი ჩამოიყრება, თუ მიერთებულია მოდულთან Timer0.

3) თუ პროგრამული მთვლელი მოდიფიცირდება ან პირობაზე შემოწმება აბრუნებს ჭეშმარიტ შედეგს, ინსტრუქცია მოითხოვს ორ მანქანურ ციკლს. მეორე ციკლი სრულდება პროცესორის მიერ, როგორც NOP.

ინსტრუქციების აღწერილობა

ADDLW	**Add Literal and W**
სინტაქსი	[*label*] ADDLW k
ოპერანდები	0?k?255
მოქმედება	(W) + k → (W)
გავლენა ალმებზე	C, DC, Z

აღწერილობა W სამუშაო რეგისტრის შიგთავსი ემატება რვაბიტიან კონსტანტას k და შედეგი თავსდება რეგისტრში W

მაგალითი ADDLW 0x15

ADDWF	**Add W and F**
სინტაქსი	[*label*] ADDWF f, d
ოპერანდები	0≤f≤127
	d∈[0,1]
მოქმედება	(W) +(f) → (dest)

გავლენა ალმებზე	C, DC, Z

აღწერილობა W რეგისტრის შიგთავსი იკრიბება f რეგისტრის შიგთავსთან და შედეგი თავსდება რეგისტრში f, თუ d=1 და აკუმულატორში W, თუ d=0. 0 და 1 მნიშვნელობების ნაცვლად, შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f.

მაგალითი	ADDWF MY_REG, 0

ANDLW	**AND Literal with W**
სინტაქსი	[*label*] ANDLW k
ოპერანდები	$0 \le k \le 255$
მოქმედება	(W).AND.(k) → (W)
გავლენა ალმებზე	Z

აღწერილობა W რეგისტრის შიგთავსსა და რვაბიტიან კონსტანტა k-ზე სრულდება ოპერაცია "ლოგიკური და". შედეგი თავსდება რეგისტრში W

მაგალითი	ANDLW 0x3A

ANDWF	**AND W with F**
სინტაქსი	[*label*] ANDWF f, d
ოპერანდები	$0 \le k \le 127$
	$d \in [0,1]$
მოქმედება	(W).AND.(f) → (dest)
გავლენა ალმებზე	Z

აღწერილობა W და f რეგისტრების შიგთავსებს შორის სრულდება ოპერაცია "ლოგიკური და". შედეგი თავსდება რეგისტრში f, თუ d=1 და რეგისტრში W, თუ d=0. 0 და 1 მნიშვნელობების ნაცვლად, შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f.

მაგალითი	ANDWF MY_REG, 1

BCF	**Bit Clear F**
სინტაქსი	[*label*] BCF f, d
ოპერანდები	$0 \le k \le 127$
	$0 \le b \le 7$
მოქმედება	0 → (f<b>)
გავლენა ალმებზე	არა

აღწერილობა აყენებს "0"-ში b ნომერ ბიტს რეგისტრში f.

მაგალითი	BCF MY_REG, 3, დაყენდესს “0”-ში ნომერ 3 ბიტს რეგისტრში MY_REG
	. . .
	BCF PORTA,4 ; დაყენდეს 0-ში A პორტის RA4 ხაზი

BSF	**Bit Set F**
სინტაქსი	[*label*] BSF f, d
ოპერანდები	0≤k≤127
	0≤b≤7
მოქმედება	0 → (f<b>)
გავლენა ალმებზე	არა
აღწერილობა	აყენებს “1”-ში b ნომერ ბიტს რეგისტრში f.
მაგალითი	BCF MY_REG, 3, დაყენდესს “1”-ში ნომერ 3 ბიტს რეგისტრში MY_REG
	. . .
	BCF PORTA,4 ; დაყენდეს 1-ში A პორტის RA4 ხაზი

BTFSC	**Bit Test F Skip if Clear**
სინტაქსი	[*label*] BTFSC f, d
ოპერანდები	0≤k≤127
	0≤b≤7
მოქმედება	skip next command if (f<b>)=0
გავლენა ალმებზე	არა
აღწერილობა	თუ f რეგისტრის b ბიტი უდრის 1, სრულდება რიგის მიხედვით შემდეგი შემდეგი ბრძანება. თუ f რეგისტრის b ბიტი უდრის 0, შემდეგი ბრძანება გამოიტოვება და ინკრემენტირდება პროგრამული მთვლელი. ამ შემთხვევაში შესრულება იკავებს 2 მანქანურ ციკლს.
მაგალითი	BTFSC MY_REG, 2, ვახდენთ 2 ბიტის ტესტირებას რეგისტრში MY_REG
	GOTO LABEL_ONE ;თუ 1, გადასვლა ჭდეზე LABEL_ONE
ADDWF MYREG	; თუ 0, სრულდება შეკრება W-სთან

BTFSS	**Bit Test F Skip if Set**
სინტაქსი	[*label*] BTFSS f, d
ოპერანდები	0≤k≤127

0≤b≤71

გავლენა ალმებზე	არა

აღწერილობა თუ f რეგისტრის b ბიტი უდრის 0, სრულდება რიგის მიხედვით შემდეგი შემდეგი ბრძანება. თუ f რეგისტრის b ბიტი უდრის 1, შემდეგი ბრძანება გამოიტოვება და ინკრემენტირდება პროგრამული მთვლელი. ამ შემთხვევაში შესრულება იკავებს 2 მანქანურ ციკლს.

მაგალითი BTFSC MY_REG, 2, ვახდენთ 2 ბიტის ტესტირებას რეგისტრში MY_REG

GOTO LABEL_ONE ;თუ 0, გადასვლა ჭდეზე LABEL_ONE

ADDWF MYREG ; თუ 1, სრულდება შეკრება W-სთან

CALL **Call subroutine**

სინტაქსი	[*label*] CALL k
ოპერანდები	0≤k≤2047
მოქმედება	(PC) + 1 → TOS
	k → PC <10:0>
	(PCLATCH<4:3>) → (PC<12:11>)
გავლენა ალმებზე	არა

აღწერილობა შემდეგი ბრძანების მისამართი იტვირთება სტეკში დაბრუნების მისამართის სახით. ხდება გადასვლა ქვეპროგრამის მისამართზე, რომელიც მითითებულია კონსტანტაში k. ასემბლერში კონსტანტის ნაცვლად გამოიყენება ტექსტური ჭდე.

```
CALL LOADER
. . .          ; ძირითადი პროგრამის ტექსტი
. . .
LOADER ...     ; ქვეპროგრამის ტექსტი
  . . .
  RETURN
```

CLRF **Clear F**

სინტაქსი	[*label*] CLRF f
ოპერანდები	0≤k≤127
მოქმედება	00h → (f)
	1 → Z
გავლენა ალმებზე	Z

აღწერილობა f რეგისტრის შიგთავსი მთლიანად სუფთავდება,

	ყენდება STATUS რეგისტრის Z ალამი.
მაგალითი	CLRF MY_REG

CLRWDT	**Clear WatchDog Timer**
სინტაქსი	[*label*] CLRWDT
ოპერანდები	არა
მოქმედება	00h → WDT
0	→ (წინასწარი გამყოფი WDT)
1 → TO	
1→ PD	
გავლენა ალმებზე	TO, PD
აღწერილობა	ჩამოყრის მოთვალთვალე ტაიმერს და ასუფთავებს წინასწარ გამყოფს, თუ ის მიერთებულია მოთვალთვალე ტაიმერთან.
მაგალითი	CLRWDT

COMF	**Complement F**
სინტაქსი	[*label*] COMF f, d
ოპერანდები	0≤k≤127
	d∈[0,1]
მოქმედება	(f) → (dest)
გავლენა ალმებზე	Z
აღწერილობა	f რეგისტრის შიგთავსის ინვერსია. თუ d=0, შედეგი ნარჩუნდენა W-ში, თუ d=1, შედეგი ინახება რეგისტრში f. 0 და 1 მნიშვნელობების ნაცვლად შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f.
მაგალითი	COMF MY_REG ; MY_REG-ის ინვერსია, შედეგი W-ში

DECF	**Decrement F**
სინტაქსი	[*label*] DECF f, d
ოპერანდები	0≤k≤127
	d∈[0,1]

მოქმედება	(f) - 1→ (dest)
გავლენა ალმებზე	Z
აღწერილობა	f რეგისტრის შიგთავსს აკლდება ერთიანი. თუ d=0, შედეგი ინახება W-ში, თუ d=1, შედეგი ინახება რეგისტრში f. 0 და 1 მნიშვნელობების ნაცვლად შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f.
მაგალითი	DECF MY_REG

DECFSZ	**Decrement F, Skip if Zero**
სინტაქსი	[*label*] DECFSZ f, d
ოპერანდები	0≤k≤127 d∈[0,1]
მოქმედება	(f) - 1→ (dest) skip if result =0
გავლენა ალმებზე	არა
აღწერილობა	f რეგისტრის შიგთავსს აკლდება ერთიანი. თუ d=0, შედეგი ინახება W-ში, თუ d=1, შედეგი ინახება რეგისტრში f. 0 და 1 მნიშვნელობების ნაცვლად შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f. თუ შედეგი არ

უდრის 0, სრულდება რიგის მიხედვით შემდეგი ბრძანება, თუ შედეგი უდრის 0, შემდეგი ბრძანება გამოიტოვება, პროგრამული მთვლელი ინკრემენტირდება. ამ შემთხვევაში შესრულება იკავებს 2 მანქანურ ციკლს.

```
DECFSZ MY_REG, 0
GOTO REZ_ZERO ; გადასვლა, თუ 0
. . .
  .
. . .
REZ_ZERO . . .
```

GOTO	**Goto label (unconditional branch)**
სინტაქსი	[*label*] GOTO f, d

ოპერანდები	$0 \le k \le 2047$
მოქმედება	$k \rightarrow (PC<10:0>)$ $(PCLATCH<4:3>) \rightarrow (PC<12:11>)$
გავლენა ალმებზე	არა
აღწერილობა	უპირობო გადასვლა მისამართზე k. ასემბლერში კონსტანტის ნაცვლად გამოიყენება ტექსტური ჭდე.
მაგალითი	GOTO NEW_ADDR . . . . . . NEW_ADDR . . .

INCF	**Increment F**
სინტაქსი	[*label*] INCF f, d
ოპერანდები	$0 \le k \le 127$ $d \in [0,1]$
მოქმედება	$(\underline{f}) + 1 \rightarrow (dest)$
გავლენა ალმებზე	Z
აღწერილობა	f რეგისტრის შიგთავსს ემატება 1. თუ d=0, შედეგი ინახება W-ში, თუ d=1, შედეგი ინახება რეგისტრში f. 0 და 1 მნიშვნელობების ნაცვლად შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f.
მაგალითი	INCF MY_REG, 0

INCFSZ	**Increment F, Skip if Zero**
სინტაქსი	[*label*] INCFSZ f, d
ოპერანდები	$0 \le k \le 127$ $d \in [0,1]$
მოქმედება	$(\underline{f}) + 1 \rightarrow (dest)$ skip if result =0
გავლენა ალმებზე	არა
აღწერილობა	f რეგისტრის შიგთავსს ემატება ერთიანი. თუ d=0,

შედეგი ინახება W-ში, თუ d=1, შედეგი

ინახება რეგისტრში f. 0 და 1 მნიშვნელობების

ნაცვლად შეიძლება გამოვიყენოთ

შესაბამისად სიმბოლოები W და f. თუ შედეგი არ

უდრის 0, სრულდება რიგის მიხედვით შემდეგი ბრძანება, თუ შედეგი უდრის 0, შემდეგი ბრძანება

გამოიტოვება, პროგრამული მთვლელი ინკრემენტირდება. ამ შემთხვევაში შესრულებს იკავებს 2 მანქანურ ციკლს.

მაგალითი

```
INCFSZ MY_REG, 0
 GOTO REZ_ZERO ; გადასვლა, თუ 0
. . .
  .
. . .
REZ_ZERO . . .
```

IORLW	**Inclusive OR Literal with W**
სინტაქსი	[*label*] IORLW k
ოპერანდები	$0 \le k \le 255$
მოქმედება	(W).OR.(k) → (W)
გავლენა ალმებზე	Z
აღწერილობა	W რეგისტრის შიგთავსსა და რვაბიტიან კონსტანტა k-ზე სრულდება ოპერაცია "ლოგიკური ან". შედეგი თავსდება რეგისტრში W
მაგალითი	IORLW 0x15

IORWF	**Inclusive OR W with F**
სინტაქსი	[*label*] IORWF f, d
ოპერანდები	$0 \le k \le 127$ $d \in [0,1]$
მოქმედება	(W) .OR.(f) → (dest)
გავლენა ალმებზე	Z
აღწერილობა	W რეგისტრის შიგთავსსა და f რეგისტრის შიგთავსს შორის სრულდება ოპერაცია "ლოგიკური ან". თუ d=0, შედეგი ინახება W-ში, თუ d=1, შედეგი

ინახება რეგისტრში f. 0 და 1 მნიშვნელობების ნაცვლად შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f.

მაგალითი IORWF MY_REG, 0

MOVLW **Mov Literal to W**

სინტაქსი [*label*] MOVLW k

ოპერანდები $0 \le k \le 255$

მოქმედება $k \rightarrow (W)$

გავლენა ალმებზე არა

აღწერილობა რვაბიტიანი კონსტანტა k თავსდება W რეგისტრში. თუ k–ს მნიშვნელობა არ არის მითითებული, იგი ასემბლერდება, როგორც 0.

მაგალითი MOVLW 0x15

MOVF **Move F**

სინტაქსი [*label*] MOVF f, d

ოპერანდები $0 \le k \le 127$

$d \in [0,1]$

მოქმედება $(f) \rightarrow (dest)$

გავლენა ალმებზე Z

აღწერილობა მნიშვნელობა f რეგისტრიდან გადაიტანება მიმღებში, რომელიც განისაზღვრება მნიშვნელობით d. თუ d=0, შედეგი ინახება W-ში, თუ d=1, შედეგი ინახება რეგისტრში f. 0 და 1 მნიშვნელობების ნაცვლად შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f. მნიშვნელობა d=1 გამოიყენება f რეგისტრის შიგთავსის ნულზე ტესტირებისათვის, რამდენადაც ამ შემთხვევაში რეგისტრის შიგთავსი არ იცვლება, მაგრამ ალამი Z დაყენდება რეგისტრის მნიშვნელობის შესაბამისად.

მაგალითი MOVF MY_REG, 0 ; MY_REG-ის შიგთავსის გადატანა W –ში.

MOVF MY_REG, 1 ; MY_REG-ის

შიგთავსის ტესტირება.

MOVWF	**Move W to F**
სინტაქსი	[*label*] MOVWF f
ოპერანდები	$0 \le k \le 127$
მოქმედება	(W) → (f)
გავლენა ალმებზე	არა
აღწერილობა	W რეგისტრის შიგთავსი გადაიტანება რეგისტრში F
მაგალითი	MOVWF MY_REG

NOP	**No Operation**
სინტაქსი	[*label*] NOP
ოპერანდები	არა
მოქმედება	არა
გავლენა ალმებზე	არა

აღწერილობა პროცესორის არითმეტიკულ-ლოგიკური მოწყობილობა არ ასრულებს არანაირ ოპერაციას. ტაქტური სიხშირის ოთხი ტაქტის შემდეგ (ერთი მანქანური ციკლი) პროცესორი გადადის შემდეგი ოპერაციის შესრულებაზე.

მაგალითი NOP

RETFIE	**Return From Interrupt**
სინტაქსი	[*label*] RETFIE
ოპერანდები	არა
მოქმედება	(TOS) → (PC) 1 → GIE
გავლენა ალმებზე	არა

აღწერილობა დაბრუნება წყვეტის დამუშავების ქვეპროგრამიდან. დაბრუნების მისამართი ჩაიტვირთება პროგრამულ მთვლელში. წყვეტების გლობალური მართვის ბიტი GIE ყენდება 1-ში.

მაგალითი RETFIE

RETLF	**Return with Literal in W**
სინტაქსი	[*label*] RETLW k
ოპერანდები	0≤k≤255
მოქმედება	(k) → (W) (TOS) → (PC)
გავლენა ალმებზე	არა
აღწერილობა	დაბრუნება ქვეპროგრამიდან კონსტანტით აკუმულატორში. რეგისტრში W ჩაიტვირთება რვაბიტიანი ლიტერალი k. პროგრამულ მთვლელში ჩაიტვირთება დაბრუნების მისამართი სტეკიდან.
მაგალითი	RETLW 0xFE

RETURN	**Return from subroutine**
სინტაქსი	[*label*] RETURN
ოპერანდები	არა
მოქმედება	(TOS) → (PC)
გავლენა ალმებზე	არა
აღწერილობა	დაბრუნება ქვეპროგრამიდან. დაბრუნების მისამართი ჩაიტვირთება პროგრამულ მთვლელში სტეკიდან.
მაგალითი	RETURN

RLF	**Rotate Left F through carry**
სინტაქსი	[*label*] RLF f, d
ოპერანდები	0≤k≤127 d∈[0,1]
მოქმედება	←(C) ← (register) ← ↓ ⟶ ↑
გავლენა ალმებზე	C
აღწერილობა	f რეგისტრის შიგთავსი ბიტების მიხედვით იძვრის მარცხნივ გადატანის C ალმის გავლით. თუ d=0, შედეგი ინახება W-ში, თუ d=1, შედეგი ინახება რეგისტრში f. 0 და 1 მნიშვნელობების ნაცვლად შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f.

მაგალითი	RLF MY_REG, 0

RRF	**Rotate Right F through carry**
სინტაქსი	[*label*] RRF f, d
ოპერანდები	$0 \le k \le 127$ $d \in [0,1]$
მოქმედება	$\rightarrow$ (C) $\rightarrow$ (register) $\rightarrow$ $\uparrow \longleftarrow \downarrow$
გავლენა ალმებზე	C
აღწერილობა	f რეგისტრის შიგთავსი ბიტების მიხედვით იძვრის მარჯვნივ გადატანის C ალმის გავლით. თუ d=0, შედეგი ინახება W-ში, თუ d=1, შედეგი ინახება რეგისტრში f. 0 და 1 მნიშვნელობების ნაცვლად შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f.
მაგალითი	RRF MY_REG, 0

SUBLW	**Subtract W from Literal**
სინტაქსი	[*label*] SUBLW k
ოპერანდები	$0 \le k \le 255$
მოქმედება	k – (W) $\rightarrow$ (W)
გავლენა ალმებზე	C, DC, Z
აღწერილობა	W რეგისტრის შიგთავსი აკლდება რვაბიტიან ლიტერალს k. შედეგი თავსდება რეგისტრში W.

მაგალითები SUBLW 0x02; 2 კონსტანტას უნდა გამოაკლდეს W რეგისტრის შიგთავსი.

<u>მაგალითი 1.</u> შესრულებამდე W=0x01, C=?, Z=?

შესრულების შემდეგ W=0x01, C=1, Z=0

-შედეგი დადებითია.

<u>მაგალითი 2.</u> შესრულებამდე W=0x02, C=?, Z=?

შესრულების შემდეგ W=0x00, C=1, Z=1

-შედეგი ნულის ტოლია.

<u>მაგალითი 3.</u> შესრულებამდე W=0x03, C=?, Z=?

შესრულების შემდეგ W=0xFF, C=0, Z=0

-შედეგი უარყოფითია.

SUBWF	**Subtract W from F**
სინტაქსი	[*label*] SUBWF f, d
ოპერანდები	$0 \le k \le 127$
	$d \in [0,1]$
მოქმედება	(f) – (W) → (dest)
გავლენა ალმებზე	C, DC, Z
აღწერილობა	W რეგისტრის შიგთავსი აკლდება f რეგისტრის მნიშვნელობას. თუ d=0, შედეგი ინახება W-ში, თუ d=1, შედეგი ინახება რეგისტრში f. 0 და 1 მნიშვნელობების ნაცვლად შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f.
მაგალითები	SUBWF MY_REG, 1

<u>მაგალითი 1.</u> შესრულებამდე MY_REG =3, W=2, C=?, Z=?

შესრულების შემდეგ MY_REG =1, W=2, C=1, Z=0

-შედეგი დადებითია.

<u>მაგალითი 2.</u> შესრულებამდე MY_REG=2, W=2, C=?, Z=?

შესრულების შემდეგ MY_REG =0, W=2, C=1, Z=1

-შედეგი ნულის ტოლია.

<u>მაგალითი 3.</u> შესრულებამდე MY_REG=1, W=2, C=?, Z=?

შესრულების შემდეგ MY_REG =0xFF, W=2, C=0, Z=0

-შედეგი უარყოფითია.

SLEEP	**SLEEP into standby mode**
სინტაქსი	[*label*] SLEEP
ოპერანდები	არა
მოქმედება	00h → (WDT)
0→ (WDT prescaler)	
1 → TO	
1→ PD	
გავლენა ალმებზე	TO, PD
აღწერილობა	გადაყავს პროცესორი ენერგოდამზოგ რეჟიმში SLEEP და აჩერებს ტაქტურ გენერატორს
მაგალითი	SLEEP

SWAPF	**Swap nibbles in F**
სინტაქსი	[*label*] SWAPF f, d
ოპერანდები	0≤k≤127 d∈[0,1]
მოქმედება	(f<3:0>) → (dest<7:4>) (f<7:4>) → (dest<3:0>)
გავლენა ალმებზე	არა
აღწერილობა	f რეგისტრის უფროსი და უმცროსი ნახევარბაიტები ადგილს ცვლიან. შედეგი შეიტანება მიმღებში, რომელიც განისაზღვრება მნიშვნელობით d. თუ d=0, შედეგი ინახება W-ში, თუ d=1, შედეგი ინახება რეგისტრში f. 0 და 1 მნიშვნელობების ნაცვლად შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f.
მაგალითი	RLF MY_REG, 0

XORLW	**Exclusive OR Literal with W**
სინტაქსი	[*label*] XORLW k
ოპერანდები	0≤k≤255

მოქმედება	(W) .XOR.k → (W)
გავლენა ალმებზე	Z
აღწერილობა	W რეგისტრის შიგთავსსა და რვათანრიგიან კონსტანტა k-ს შორის სრულდება ოპერაცია “გამომრიცხავი ან”. შედეგი თავსდება რეგისტრში W.
მაგალითი	XORLW 0xCB

XORWF	**Exclusive OR W with F**
სინტაქსი	[*label*] XORWF f, d
ოპერანდები	0≤k≤127 d∈[0,1]
მოქმედება	(W) .XOR.(f) → (dest)
გავლენა ალმებზე	Z
აღწერილობა	W და f რეგისტრების შიგთავსებზე სრულდება ოპერაცია “გამომრიცხავი ან”. თუ d=0, შედეგი ინახება W-ში, თუ d=1, შედეგი ინახება რეგისტრში f. 0 და 1 მნიშვნელობების ნაცვლად შეიძლება გამოვიყენოთ შესაბამისად სიმბოლოები W და f.
მაგალითი	XORWF MY_REG, 0

*შემდეგი ორი ბრძანება **OPTION** და **TRIS** ჯერ არ გამოიყენება კონტროლერებში **PIC16Fxx** და მათი მომსახურება ხდება ასემბლერის მიერ გამაფრთხილებელი შეტყობინების გაცემით. მაგრამ ფირმა **Microchip®**–ის შემდგომ ნამუშევრებში მათი გამოყენება არ მოხდება. “ზემოთ” შეთავსებადობის უზრუნველსაყოფად კონტროლერების შედარებით შემდგომ ვერსიებთან არ უნდა იქნეს გამოყენებული ეს ბრძანებები საკუთარ პროგრამებში. ადრე დაწერილი პროგრამების გადასატანად და მზა ბიბლიოთეკების გამოსაყენებლად აუცილებელია შემოწმდეს მათი ტექსტი და შეიცვალოს ბრძანებები **OPTION** და **TRIS** ბრძანებების ექვივალენტური თანმიმდევრობით, კონტექსტთან შესაბამისობაში.*

OPTION	**Load Option Register**
სინტაქსი	[*label*] OPTION
ოპერანდები	არა
მოქმედება	(W) → OPTION register
გავლენა ალმებზე	არა

აღწერილობა W რეგისტრის შიგთავსი ჩაიტვირთება რეგისტრში OPTION. ინსტრუქცია დახმარებას გაგვიწევს კოდების შექმნისათვის, რომლებიც თავსებადია კონტროლერებთან PIC16C5X. ვინაიდან OPTION რეგისტრი მისაწვდომია წაკითხვისა და ჩაწერისათვის, შესაძლებელია მისი პირდაპირი ადრესაცია.

TRIS	**Load TRIS Register**
სინტაქსი	[*label*] TRIS f
ოპერანდები	$5 \le k \le 7$
მოქმედება	(W) → TRIS register f
გავლენა ალმებზე	არა

აღწერილობა ინსტრუქცია დახმარებას გაგვიწევს კოდების შექმნისათვის, რომლებიც თავსებადია კონტროლერებთან PIC16C5X. ვინაიდან TRIS რეგისტრები მისაწვდომია წაკითხვისა და ჩაწერისათვის, შესაძლებელია მათი პირდაპირი ადრესაცია.