

# Tratarea întreruperilor la dsPIC33

## a) Scopul lucrării

Familiarizarea cu modul în care sunt tratate întreruperile de către controlerul de întreruperi al familiei dsPIC33. Drept exemplu, se va lucra cu întreruperea externă INT0.

## b) Introducere

O întrerupere reprezintă un semnal sincron sau asincron, care semnalizează apariția unui eveniment ce trebuie tratat de microcontroler.

Tratarea întreruperii este de fapt suspendarea execuției normale a programului principal și lansarea **rutinei de tratare a întreruperilor (RTI)** sau din engleză a ***interrupt service routine (ISR)***. După tratarea unei întreruperi, programul revine în punctul în care a fost suspendată execuția și reia firul normal de execuție (vezi Figura 1.1).

### 1. *Flagul de întrerupere*

Mecanismul de întreruperi trebuie activat explicit, acest lucru este realizat cu ajutorul unui bit de control, numit *flag* de întrerupere. La apariția unei întreruperi (Figura 1.1 *int(1)*) *flagul* de întreruperi e setat automat pe valoarea 1 și e de datoria programatorului ca după tratarea întreruperii să îl seteze înapoi pe valoarea 0 (vezi *isr\_flag* din Figura 1.1).

Rutina de tratare a întreruperii întârzie orice altă activitate din main și inhibă execuția altor întreruperi, de aceea se dorește ca timpul de execuție al ei să fie cât mai mic. În cazul în care acest lucru nu este posibil, în interiorul ISR se va inițializa o variabilă de tip *flag* cu o valoare care să ne ateste apariția întreruperii (exemplu: *var\_flag = 1*, vezi Figura 1.2), iar întreruperea va fi tratată mai târziu în cod.

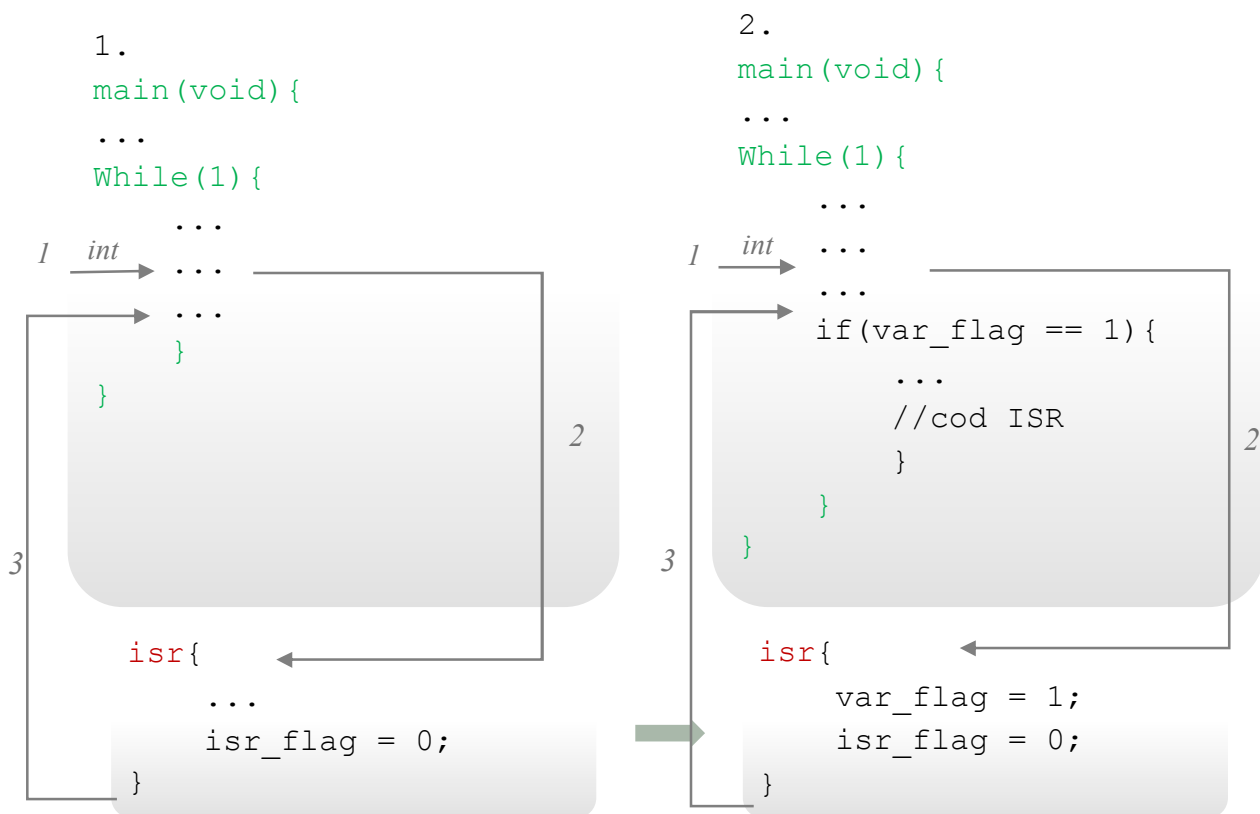


Figura 1. Tratarea ISR-urilor

## 2. Bitul *Enable*

La majoritatea microcontrolerelor, inclusiv la dsPIC33, vom găsi un bit de control *Enable*, prin setarea căruia permitem sau nu lucrul cu întreruperile.

## 3. Prioritatea întreruperilor

Fiecărei surse de întrerupere i se poate asocia un **nivel de prioritate** de la 0 la 7. Acest nivel poate fi modificat cu ajutorul celor mai puțin semnificativi 3 biți din fiecare grupare de 4 biți (nibble) rezervată fiecărei surse din registrul IPCx. O sursa de întrerupere ce va avea setați cei 3 mai puțin semnificativi biți va avea nivelul 7, deci un grad de prioritate maxim. Dacă acești 3 biți sunt 0 putem considera ca acea sursa de întrerupere este dezactivată.

Unele microcontrolere au o **tabelă a vectorilor de întreruperi(TVI)** implementată. În această tabelă, fiecărei întreruperi îi este asociată adresa rutinei sale de tratare, la care programul va face salt în cazul apariției acesteia. Aceste adrese sunt predefinite și sunt mapate în memoria de program într-un spațiu contiguu care alcătuiește TVI. Adresele întreruperilor în TVI sunt setate în funcție de prioritatea lor: cu cât adresa este mai mică cu atât prioritatea este mai mare.

Controlerul de întreruperi al familie dsPIC33FJ ne pune la dispoziție două tabele ale vectorilor de întreruperi.

### **1. TVI – Tabela vectorilor de întreruperi**

Tabela vectorilor de întreruperi se află în memoria program la adresa 0x000004 și conține 126 de vectori dintre care: 8 vectori pentru trapurile nemascabile și 118 pentru alte surse de întrerupere. În general, fiecărei surse de întrerupere i se va asocia un vector în această tabelă, acesta conținând adresa de 24 de biți a rutinei corespunzătoare acelei surse de întrerupere (ISR- Interrupt Service Routine).

Secvența de RESET care se observă în această tabelă (vezi Figura 2) nu poate fi considerată ca fiind o excepție, întrerupere deoarece modulul de tratare a întreruperilor nu este implicat direct în această fază. Microcontrolerul își resetează toate registrele (deci și registrul PC) și execuția programului va începe de la adresa 0x000000 unde se va afla o instrucțiune GOTO care va redirecționa execuția programului către rutina principală a programului.

### **2. TAVI – Tabela alternativă a vectorilor de întreruperi**

AIVT se afla în memoria program imediat după IVT și utilizarea acestuia este posibilă prin setarea bitului *Enable Alternate Interrupt Vector Table* din registrul 2 de control al întreruperilor (INTCON2<15>). Dacă acest bit este setat, toate întreruperile și excepțiile folosesc această tabelă alternativă pentru obținerea adreselor rutinelor de tratare a întreruperilor. Această tabelă este organizată în aceeași manieră cu cea principală și este folosită în principal pentru **debugging**.

Reset – GOTO Instruction	0x000000
Reset – GOTO Address	0x000002
Reserved	0x000004
Oscillator Fail Trap Vector	
Address Error Trap Vector	
Stack Error Trap Vector	
Math Error Trap Vector	
DMA Error Trap Vector	
Reserved	
Reserved	
Interrupt Vector 0	0x000014
Interrupt Vector 1	
~	
~	
~	
Interrupt Vector 52	0x00007C
Interrupt Vector 53	0x00007E
Interrupt Vector 54	0x000080
~	
~	
~	
Interrupt Vector 116	0x0000FC
Interrupt Vector 117	0x0000FE

Figura 2. Tabela vectorilor de întrerupere (în ordine descrescătoare)

Întreruperile pot fi atât externe, cât și interne. Sursele interne de întreruperi sunt reprezentate de perifericele microcontrolerului:

- Canale de timp,
- Convertoare AD,

iar cele exterioare pot veni de la dispozitive, componente conectate la microcontroler, care pot genera o cerere de întrerupere pe anumiți pini ai microcontrolerului:

- Butoane,
- Senzori digitali,

Pentru tratarea întreruperilor externe, sunt necesare câteva setări suplimentare celor deja menționate:

#### 4. Setarea pinului ca intrare

În cazul în care se va utiliza butonul de pe placă (K2) ca sursă de întrerupere, va fi necesar ca pinul RB7, care este conectat direct la butonul K2, să fie setat ca pin de intrare. Acest lucru se realizează cu ajutorul registrului TRIS și anume se va folosi instrucțiunea:

```
_TRISB7 = 1;
```

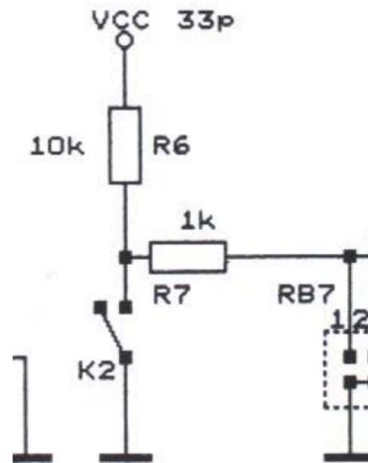


Figura 3. Schema electrică. Butonul K2 și pinul RB7

#### 5. Polaritatea

Al doilea aspect al unei întreruperi externe este polaritatea întreruperii. Pentru o întrerupere declanșată de front, polaritatea poate fi de front crescător sau pozitiv (0→1) sau de front descrescător sau negativ (1→0).

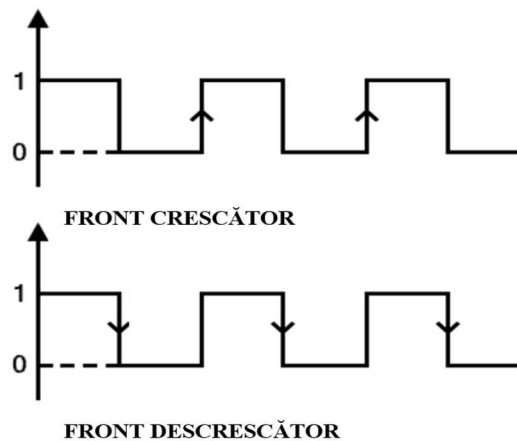


Figura 4. Polaritate

Pentru a înțelege mai bine, vom prezenta lucrul cu un senzor de obstacol, care returnează valoarea 1 logic atâta timp cât nu are nici un obstacol în față și 0 logic atunci când apare un obstacol.

În funcție de scenariul de funcționare al senzorului, Se disting trei cazuri:

**a. ISR declanșată de frontul negativ**

Se setează polaritatea pe frontul negativ în cazul în care utilizăm senzorul de obstacol pe o mașină, în vederea frânării acesteia la apariția unui obstacol în față. Ieșirea senzorului va trece în valoarea 0 atunci când va detecta obstacolul și va declanșa întreruperea.

**b. ISR declanșată de frontul pozitiv**

Se setează polaritatea pe frontul pozitiv pentru cazul în care utilizăm senzorul de obstacol ca măsură de protecție antifurt al unui exponat de muzeu. Ieșirea senzorului va trece în valoarea 1 atunci când va detecta obstacolul și va declanșa întreruperea.

**c. ISR declanșată de ambele fronturi**

Se setează polaritatea de ambele fronturi în cazul în care folosim senzorul pentru un sistem de asistență la parcare a mașinii. Senzorul activează o alarmă când va detecta un obiect în apropierea mașinii și oprește alarma pe măsură ce ne îndepărtăm de obiectul respectiv.

### c) Foaia de catalog – Registrele controlerului de întreruperi

Pe capsula microcontrolerului găsim întreruperile externe - INT0, INT1, INT2 etc. La microchip aceasta se numește INT0 și o găsim pe pinul RB7. Mai multe detalii despre întreruperi, găsim la capitolul *Interrupt Controller* în foaia de catalog.

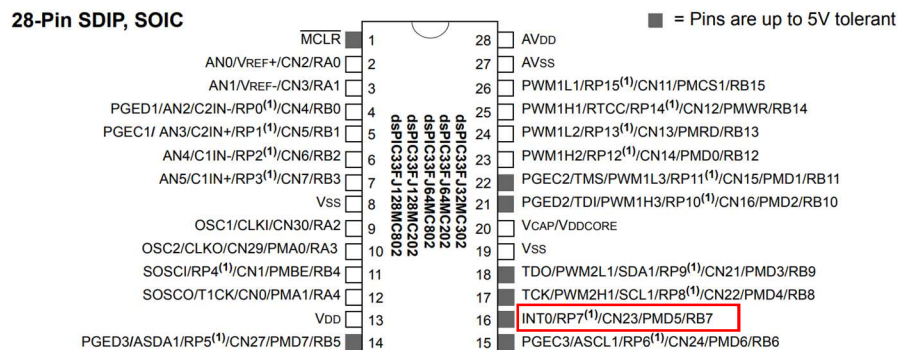


Figura 5. Capsula microcontrolerului

Cu o simplă căutare prin text după *INT0*, la capitolul dedicat întreruperilor, o găsim pe aceasta în tabelul vectorilor de întreruperi și observăm că se află în partea de început a tabelului, ceea ce înseamnă că ar avea o prioritate mare. După care ajungem să vedem registrele în care avem biții de control.

#### ● Bitul de polaritate INT0EP

Observăm INT0EP căruia îi este dedicat bitul 0 al registrului INTCON2 și este disponibil R/W (read/write), cu valoarea inițială 0.

**REGISTER 7-4: INTCON2: INTERRUPT CONTROL REGISTER 2**

R/W-0	R-0	U-0	U-0	U-0	U-0	U-0	U-0
ALTIVT	DISI	—	—	—	—	—	—
bit 15							bit 8
U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0
—	—	—	—	—	INT2EP	INT1EP	INT0EP
bit 7							bit 0

Figura 6. INT0EP: External Interrupt 0 Edge Detect Polarity Select bit

În program putem indica INT0EP în modul următor:

`_INT0EP = 1 //1 pentru front negativ (cazul a.ISR declanșată de frontul negativ)`

`_INT0EP = 0 //0 pentru front pozitiv (cazul b.ISR declanșată de frontul pozitiv)`

### ● Bitul de *flag* INT0IF

Dacă căutăm mai departe prin text, găsim bitul INT0IF, la registrul IFS0 – *Interrupt flag status*. Acesta reprezintă *flagul* de întrerupere, care trebuie setat pe 0.

**REGISTER 7-5: IFS0: INTERRUPT FLAG STATUS REGISTER 0**

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	DMA1IF	AD1IF	U1TXIF	U1RXIF	SPI1IF	SPI1EIF	T3IF
bit 15							bit 8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
T2IF	OC2IF	IC2IF	DMA0IF	T1IF	OC1IF	IC1IF	INT0IF
bit 7							bit 0

*Figura 7. INT0IF: External Interrupt 0 Flag Status bit*

În program îl inițializăm ca:

`_INT0IF = 0;`

La finalul rutinei de tratare a întreruperii, acestui *flag* îi vom reatribui valoarea 0, pentru achitarea întreruperii:

`_INT0IF = 0;`

### ● Bitul *Enable* INT0IE



**REGISTER 7-10: IEC0: INTERRUPT ENABLE CONTROL REGISTER 0**

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	DMA1IE	AD1IE	U1TXIE	U1RXIE	SPI1IE	SPI1EIE	T3IE
bit 15							bit 8
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
T2IE	OC2IE	IC2IE	DMA0IE	T1IE	OC1IE	IC1IE	INT0IE
bit 7							bit 0

*Figura 8. INT0IE: External Interrupt 0 Flag Status bit*

În program indicăm că vrem să permitem cereri de întreruperi pentru INT0 astfel:

```
_INT0IE = 1; //1 = Interrupt request enabled
```

● **Bitul de prioritate INT0IP**

**REGISTER 7-15: IPC0: INTERRUPT PRIORITY CONTROL REGISTER 0**

U-0	R/W-1	R/W-0	R/W-0	U-0	R/W-1	R/W-0	R/W-0
—	T1IP<2:0>			—	OC1IP<2:0>		
bit 15							bit 8

U-0	R/W-1	R/W-0	R/W-0	U-0	R/W-1	R/W-0	R/W-0
—	IC1IP<2:0>			—	INT0IP<2:0>		
bit 7							bit 0

*Figura 9. INT0IP<2:0>: External Interrupt 0 Priority bits*

În program putem indica nivelul de prioritate dorit, astfel:

```
_INT0IP = 7; // Interrupt is priority 7 (highest priority interrupt)
```

## d) Programul exemplu

Observăm rutina de tratare a întreruperii, care se apelează automat la apariția unei întreruperi, și funcția *main*.

```
#include "p33Fxxxx.h"
void __attribute__((interrupt, no_auto_psv)) _INT0Interrupt(void)
{
```

```

        _RB15 = ~_RB15;
        _INT0IF = 0; // Resetam flagul corespunzator intreruperii
                    // INT0 pentru a nu se reapela rutina de
intrerupere
    }
    int main(void)
    {
        TRISB = 0x0000;
        _TRISB7 = 1;
        PORTB = 0xF000; //Stingem toate LED-urile

        _INT0IF = 0;      //Resetem flagul coresp. intreruperii INT0
        _INT0IE = 1;      //Enable INT0
        _INT0EP = 1;      //Polaritatea INT0

        while(1)
        {
        }
    }

```

### e) Exerciții

1. Să se conecteze senzorul de obstacol la plăcuța de laborator. Ieșirea senzorului (Vout) va fi conectată la pinul RB7, pinul GND va fi conectat la masa plăcii și pinul Vin va fi conectat la 3.3v. Să se verifice modul în care comută LED-ul în funcție de apariția/dispariția unui obiect din fața senzorului.
2. Să se modifice programul inițial astfel încât LED-ul să se aprindă la detectarea unui obiect și să se stingă la dispariția obiectului din raza sa de acțiune.