Département Génie Électrique – INSA de Lyon

Travaux pratiques
Micro-contrôleur PIC

Les interruptions

MODULE IF3

– Olivier Bernard – Nicolas Ducros – Thomas Grenier –
Dominique Tournier

INSA de Lyon - Département Génie Électrique - 4ème année

Table des matières

1. "Bouton poussoir intelligent" avec interruptions

Les objectifs de cette séance de travaux pratiques sont :

— de savoir mettre en oeuvre des mécanismes d'interruptions et de les appliquer sur un PIC16F,

— d'être capable de concevoir un programme simple basé sur des interruptions.

1 Pré-requis

Pour cela, il faudra être familiarisé avec le code assembleur et la documentation technique des PIC16, puis de connaitre les outils de développement de programmes en assembleur. Même si l'implémentation est spécifique à ce micro-contrôleur, les démarches sont transposables à des programmes écrit en C (seule la syntaxe sera différente) ainsi qu'aux autres de micro-contrôleurs.

Un micro-contrôleur 16F887 du fabricant Microchip est utilisé, associé à un kit de développement composé d'une interface PICKIT3 et d'une platine de prototypage rapide PICKIT2 (Figure 1).

FIGURE 1: *Outils de développement PICkit3 Debug Express*

Cours pré requis :

— Cours de microcontrôleur famille PIC16, (3GE IF2)

— Connaître la structure et les principes de fonctionnement des micro-contrôleurs (registres, jeu d'instructions, mémoires, E/S, interruptions ...).

— TD microcontrôleur PIC16 de 3GE,

— Savoir concevoir et écrire un programme assembleur.

— TP microcontrôleur PIC16 de 3GE,

— Savoir développer une application PIC en utilisant le logiciel MPLAB et le kit ICD2/PICkit3.

Cours liés :

— cours sur les interruptions IF3,
— principes fondamentaux des interruptions et de leur mise en place.

2 Présentation du matériel

a. Logiciel MPLAB de Microchip

Il s'agit de l'environnement informatique de développement pour PIC fourni par le constructeur Microchip. Il est gratuitement téléchargeable [1]. Le logiciel MPLAB permet :
— d'éditer le code assembleur (reconnaissance des instructions),
— de compiler le code (le lier à des librairies si besoin),
— de simuler le comportement d'un PIC : ceci permet de tester le bon fonctionnement d'un programme par simulation (débogage),
— de piloter des outils de développement supplémentaires comme MPLAB ICD 2, PICKITx,
et bien d'autres fonctionnalités encore.

b. Kit MPLAB PICKIT 3

Le module PICkit3, "In Circuit Debugger (ICD)", permet de programmer et de déboguer un micro-contrôleur PIC via le logiciel MPLAB IDE ou tout autre logiciel compatible avec l'ICD [2]. Ce kit est composé :
— d'un module PICkit3 (Figure 2) contenant l'électronique de communication,
— d'un câble USB permettant la communication entre le bloc rectangulaire et le PC (Figure 3),
le module PICkit3 servant d'interface entre le PC et la cible (Figure 4).

FIGURE 2: *PICkit3 "In-Circuit-Debugger"*

C'est à ce kit de deboguage/programmation que l'on vient connecter la "cible" (plateforme de prototypage) qui est décrite dans le paragraphe suivant.

c. Plateforme de prototypage

La plateforme de prototypage (Figure 5) est basée sur un kit fournit par MICROCHIP. C'est sur cette carte que se trouve le PIC **16F887**, sachant que cette plateforme est compatible avec d'autres références de PIC selon le beoin. On trouve sur cette carte, dont le schéma électrique est donné sur les Figures 6 et 7, les éléments suivant :

1. http ://www.microchip.com
2. PIKLAB, suite GPUTILS ...

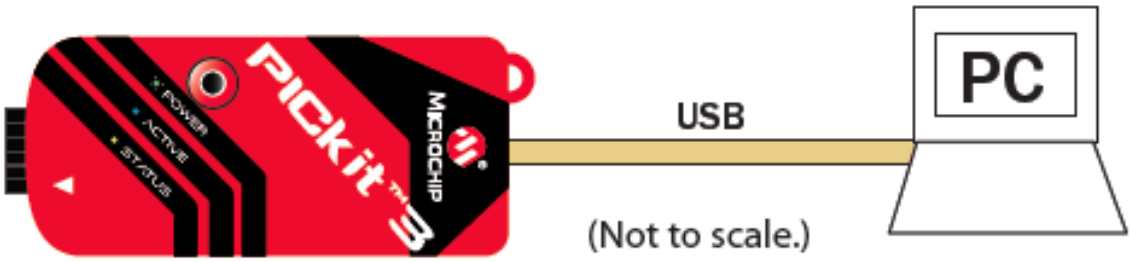


FIGURE 3: *Liaison entre le module PICkit3 et le PC*

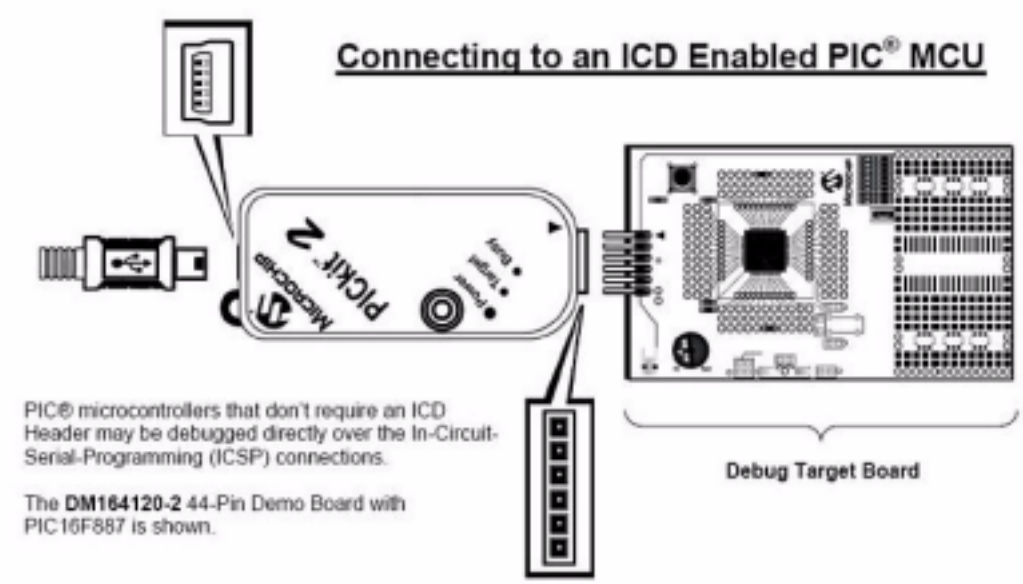


FIGURE 4: *Liaisons entre Target Board ↔ PICkit3 ↔ PC*

— un emplacement pour le soudage d'un micro-contrôleur PIC-44PIN FAMILY ,
— 1 bouton poussoir,
— 1 potentiomètre délivrant une tension variable sur une des entrées du microcontrôleur,
— 8 LEDS.

FIGURE 5: *Plateforme de prototypage*

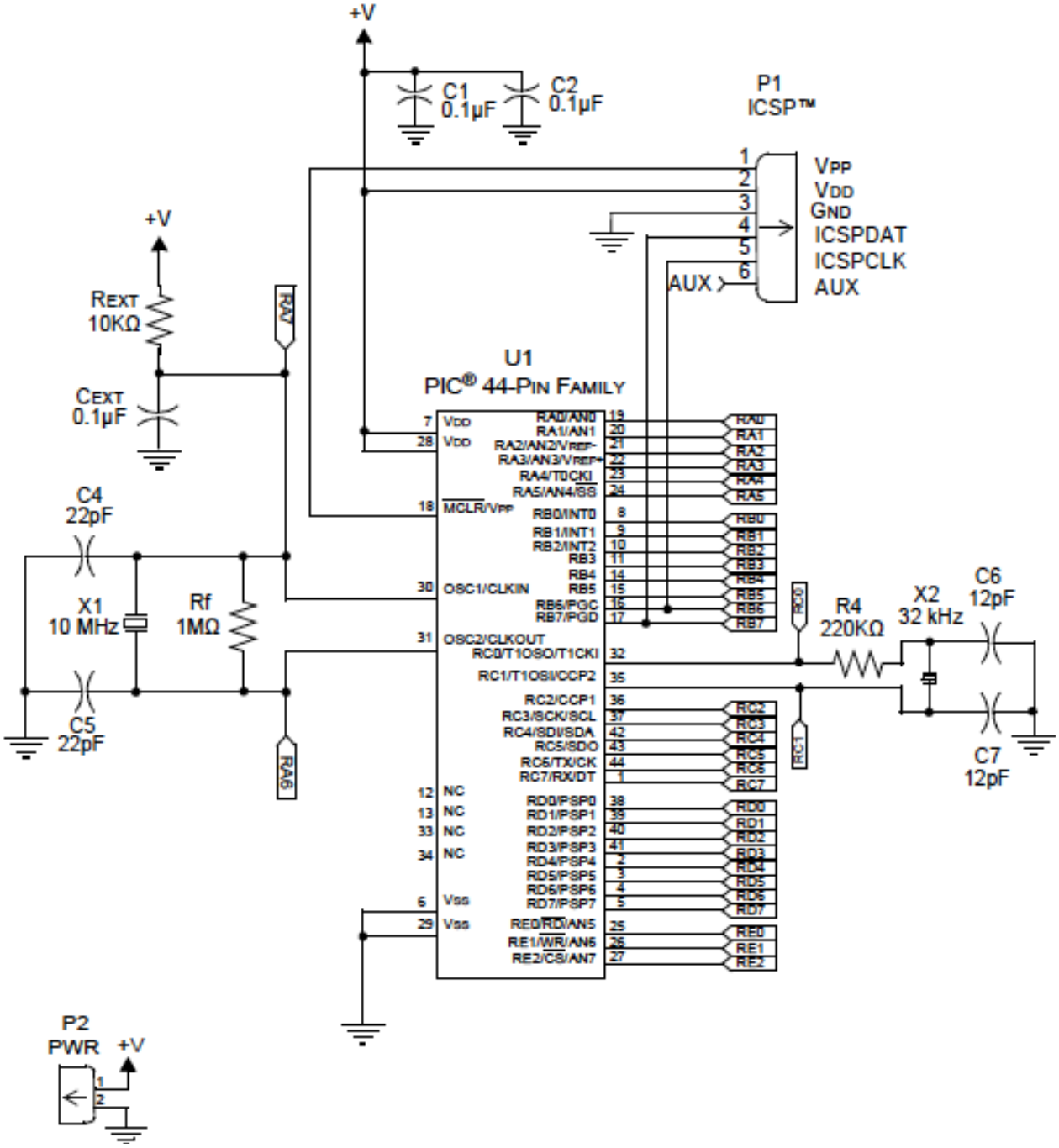


FIGURE 6: *Schéma électrique de la carte de prototypage (PIC-44PIN FAMILY (1/2))*

Quatre boutons poussoirs ont été rajoutés sur la zone dite de développement (Figure 8). Vous trouverez les informations sur les liaisons entre le μC et les périphériques utiles pour la séance de travaux pratiques dans le Tableau 1.

L'alimentation de la carte de prototypage se fait par un adaptateur externe 200V/5V. L'ensemble de ces outils est disponible sur chaque PC de la salle de TP, les logiciels, sujets, et support de cours étant également accessible sur la plateforme pédagogique MOODLE [3].

3. http ://moodle2.insa-lyon.fr

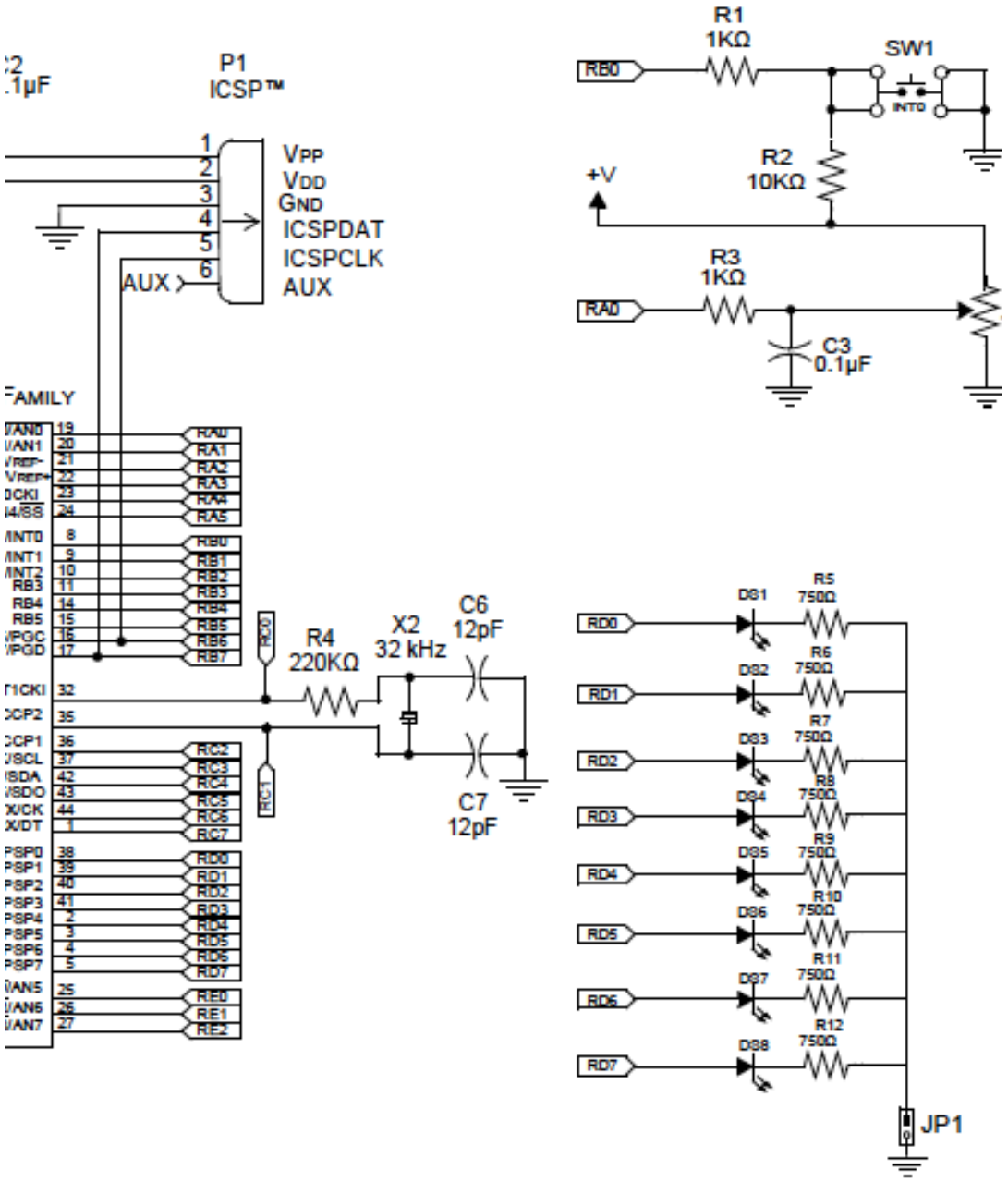


FIGURE 7: *Schéma électrique de la carte de prototypage (2/2)*

Il y a trois parties à la séance. La première partie (30 minutes) vous permettra de reprendre en main l'application et les outils de développement et de critiquer la réalisation initiale (vu en 3GE). Ensuite, il s'agira de mettre en oeuvre une interruption. Ce travail sera précédé d'un rappel sur les éléments techniques du micro-contrôleur. La dernière partie sera consacrée à la conception (et à la mise en oeuvre) d'un programme basé sur deux sources d'interruptions.

TABLE 1: *Éléments du projet*

bouton poussoir	***Port E/S PIC***	localisation
BP0	RB0	Bas-Gauche
BP1	RB1	Haut-Gauche
BP2	RB2	Haut-Droite
BP3	RB3	Bas-Droite

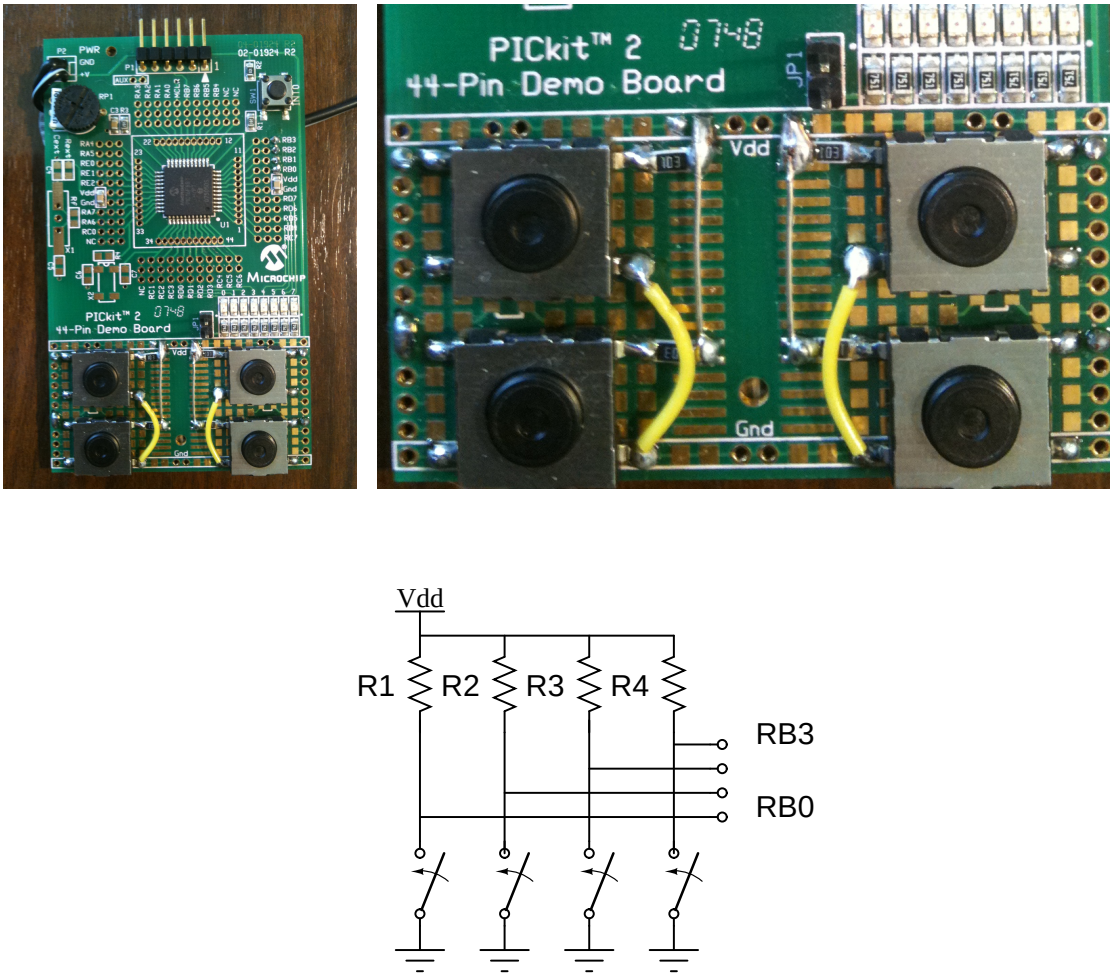


FIGURE 8: *Vue de la carte de TP (gauche) et zoom sur la zone avec les boutons poussoirs (droite) et schéma électrique (bas)*

REMARQUE : La carte de prototypage ne comporte ni quartz ni circuit RC permettant de générer des signaux d'horloge de façon externe. Des emplacement sont prévus à cet effet sur la carte mais ne sont pas utilisés. Les différentes sources d'horloges externes sont représentées sur les schémas électriques des figures 6 et 7. Le micro-contrôleur utilisé comporte une horloge interne et permet de se "passer" de composants additionnels. Une lecture attentive du code assembleur proposé et de la documentation technique vous renseignera sur la configuration de l'horloge interne.

2. Partie expérimentale : "Bouton poussoir intelligent"

Vérifier que vous disposez de l'ensemble du matériel nécessaire (PICkit3, cordon USB, protoboard) avant de commencer le TP.

1 Présentation de l'application : "Bouton-poussoir intelligent"

Pour illustrer l'intérêt des interruptions, nous réutilisons l'application bouton-poussoir intelligent vue en TP micro-contrôleur 3GE. Le but est de réaliser la commande d'un éclairage à partir d'un simple bouton poussoir :

— En appuyant une première fois sur le bouton poussoir, on allume l'éclairage.

— Une seconde pression l'éteindra. Cependant, si on ne pense pas à éteindre, l'extinction sera faite automatiquement par une minuterie.

La Figure 9 présente l'organigramme, naïf et sans interruptions, de l'application (le code assembleur est présent sur les PC et est également fourni en Annexe 1.

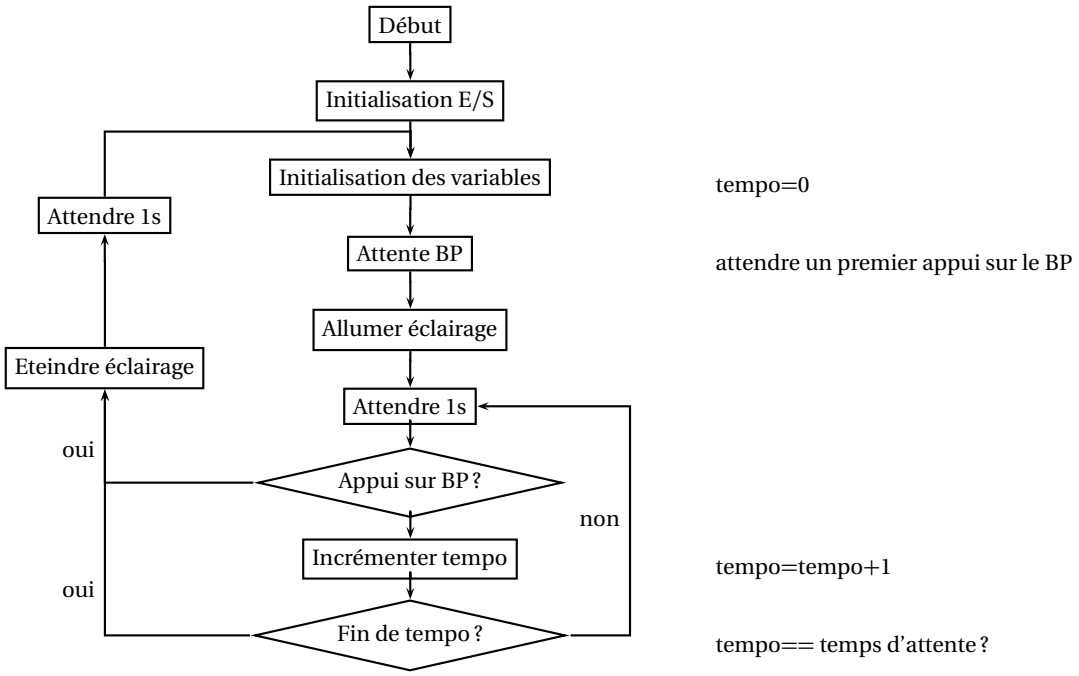


FIGURE 9: *Organigramme de l'application Bouton-poussoir intelligent sans interruption.*

2 Étude de la solution naïve

Le but de cette première série de manipulations est de déterminer les inconvénients du programme décrit par l'organigramme précédent et de se "rafraichir la mémoire" vis-à-vis de l'utilisation du logiciel MPLAB. Les différentes étapes nécessaires sont rappelées ci-dessous.

1. Récupération des fichiers sources
   - Télécharger à partir de moodle 2 les sources du TP.
   - Dé-archiver l'archive dans un répertoire de votre espace personnel.

2. Brancher le PicKit3
3. Lancement du projet MPLAB IDE
   - Ouvrir le répertoire créé suite au dé-archivage
   - Double cliquer sur le fichier **TPuC_1.mcw** se trouve dans le dossier

   → MPLAB se lance et charge les fichiers nécessaires au fonctionnement de l'application.

4. Construction du projet (Build)
   → But de la construction
   - La construction d'un projet va créer le programme complet de l'application en langage machine.
   - Quand un projet contient plusieurs fichiers et qu'il existe des liens entre ces fichiers, il est nécessaire de construire (« Build ») le projet.
   - La construction consiste à compiler tous les fichiers puis à résoudre les liens (« link ») entre les différents fichiers et librairies utilisés.
   - S'il n'y a aucune erreur (pour la compilation et l'éditeur de liens) le langage machine de l'application est généré.
     → Comment construire un projet
   - clic droit sur le nom du projet dans la fenêtre de gestion projet puis « Build All » (ou Make),
   - par le menu : Project → Build All (ou Make),
   - par les raccourcis : ctrl+F10 pour Build All (F10 pour Make).
     → Construire votre projet.

5. Programmation, Exécution avec débogage sur les maquettes. Si votre "*build*" s'est passé sans erreur, les projets sont paramétrés pour envoyer le code machine dans le micro-contrôleur et lancer l'exécution du programme (*Running* apparait en bas à gauche). Si le kit est correctement configuré et connecté, on obtient le message de la figure 10 dans la fenêtre "Output" (sinon vérifier l'alimentation des cartes, les connexions puis effectuer de nouveau la procédure). Le message de la Figure 11 peut apparaître, dans ce cas cochez "Don't show me this again" et cliquez sur "OK"

6. Programmer le PIC : au besoin, pour envoyer le code machine dans le PIC, procéder ainsi :
   - dans le menu "Degugger" → Program

   → Si la programmation s'est correctement déroulée, on obtient le message de la figure 12 dans la fenêtre "Output".

   Le programme est maintenant chargé dans le PIC mais le PC reste le superviseur de l'exécution.
7. Exécution
   - Pour exécuter le programme : Débugger → Run (F9)

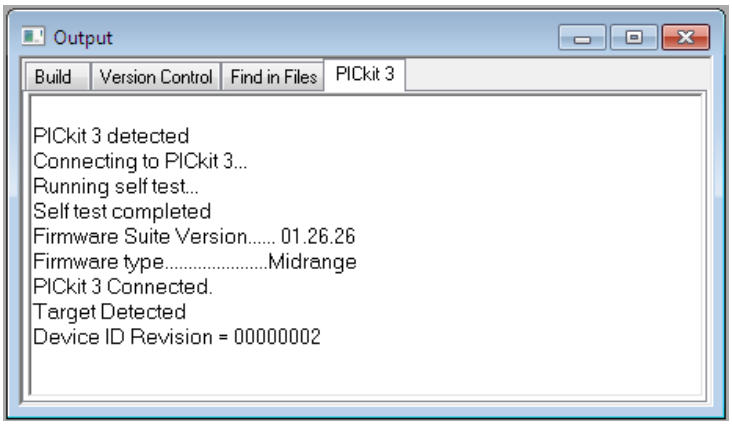


FIGURE 10: *Message indiquant une connexion réussie avec le debugger.*

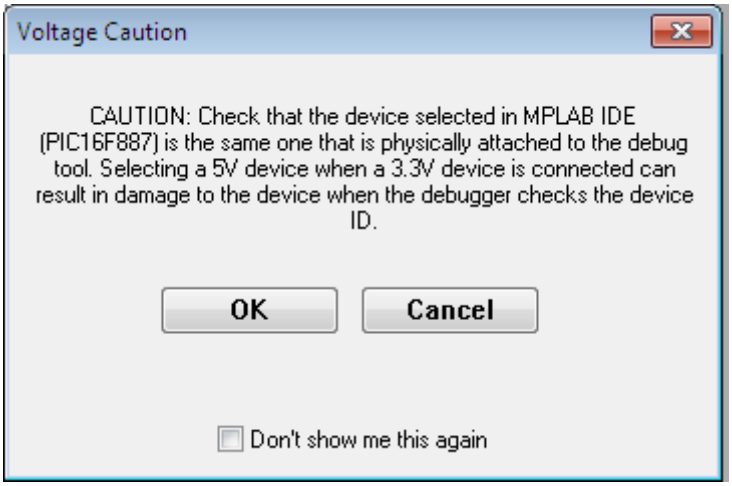


FIGURE 11: *Message d'avertissement de compatibilité d'alimentation du microcontrôleur et de debbuger.*

- Pour arrêter l'exécution du programme : Debugger → Halt (F5)
- Vous pouvez désormais visualiser les registres internes du PIC, les variables de votre programme, exécuter en "pas-à-pas" votre programme comme cela est illustré sur la Figure 13

3 Questions

1. *Un peu d'électronique... Donner L'état logiques appliqués aux entrées du PIC lorsque les boutons poussoir sont relâchés (au repos).*

2. *Critiquer le code assembleur fourni (respect de l'organigramme, lisibilité (utilisation de fonctions, commentaires, astuces tordues), utilisation des ressources, ?).*

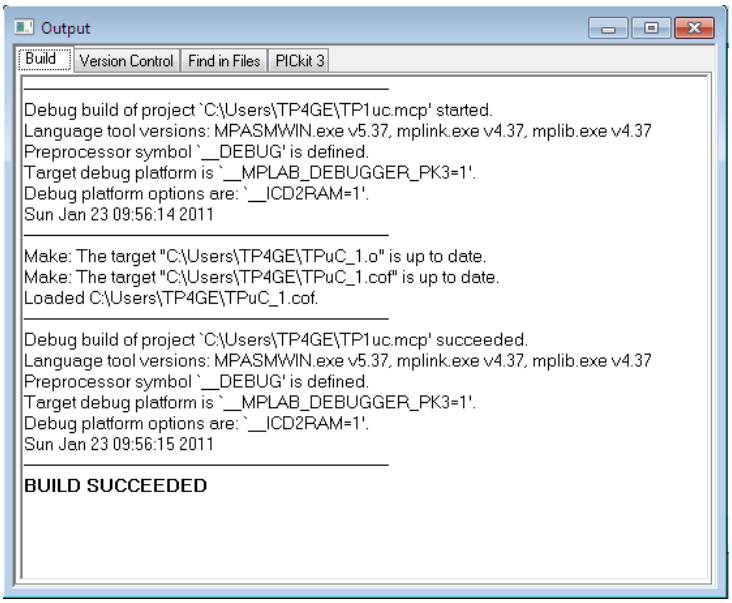


FIGURE 12: *Message d'information de construction correcte du projet.*

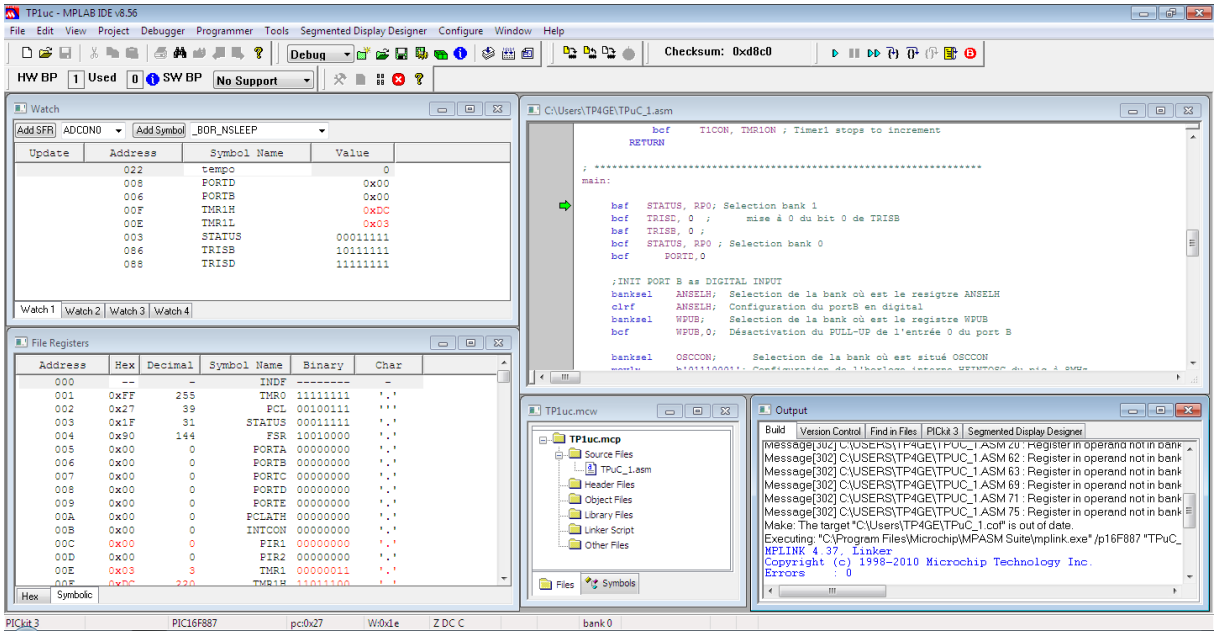


FIGURE 13: *Vue de l'environnement de développement MPLAB avec différentes fenêtres utiles au débogage d'un projet (code source, SFR, watch ...).*

3. *Identifier 2 dysfonctionnements majeurs de l'application.*

4. *Proposer une solution pour améliorer la gestion du bouton poussoir. Modifier l'organigramme.*

→ **Vous devez désormais pouvoir justifier de la naïveté de l'organigramme proposé, critiquer la qualité du code fourni et enfin proposer des solutions plus adaptées.**

3. Mise en oeuvre d'une solution avec interruption sur TIMER1

Pour améliorer le comportement du programme, la mise en oeuvre d'une interruption sur le TIMER1 est proposée. Aussi, la détection d'une transition d'états du bouton poussoir en RB0 est utilisée.

Les interruptions, en bref : Rappelons tout d'abord que l'effet d'une instruction exécutée est dépendant de l'état du microprocesseur et que l'état d'un microprocesseur est donné par les valeurs de ses registres. Enfin, un micro-contrôleur contient un microprocesseur.

Lors d'une interruption, le microprocesseur termine l'instruction en cours et sauve l'adresse de l'instruction suivante, généralement dans la pile système. Il exécute ensuite la routine d'interruption (ISR) qui commence par sauvegarder l'état du micro-contrôleur, en se focalisant sur les registres qui vont être modifiés par la routine d'interruption, puis continue par le traitement de l'interruption. Une fois le traitement de l'interruption effectué, la routine se termine par plusieurs instructions de retour d'interruption, qui restaurent l'état sauvé et font repartir le processeur de l'endroit où il avait été interrompu.

5. *Quand la routine d'interruption va-t-elle être exécutée ?*

6. *Lors d'une interruption, quelles sont les actions effectuées par le micro-contrôleur garantissant un comportement déterministe du programme ?*

7. *Quelles sont les instructions à écrire dans une fonction d'interruption pour garantir le bon fonctionnement du programme ?*

8. *Dans le cas du PIC utilisé, détailler les actions effectuées par le micro-contrôleur et les instructions à écrire dans la fonction d'interruption (cf. Annexes).*

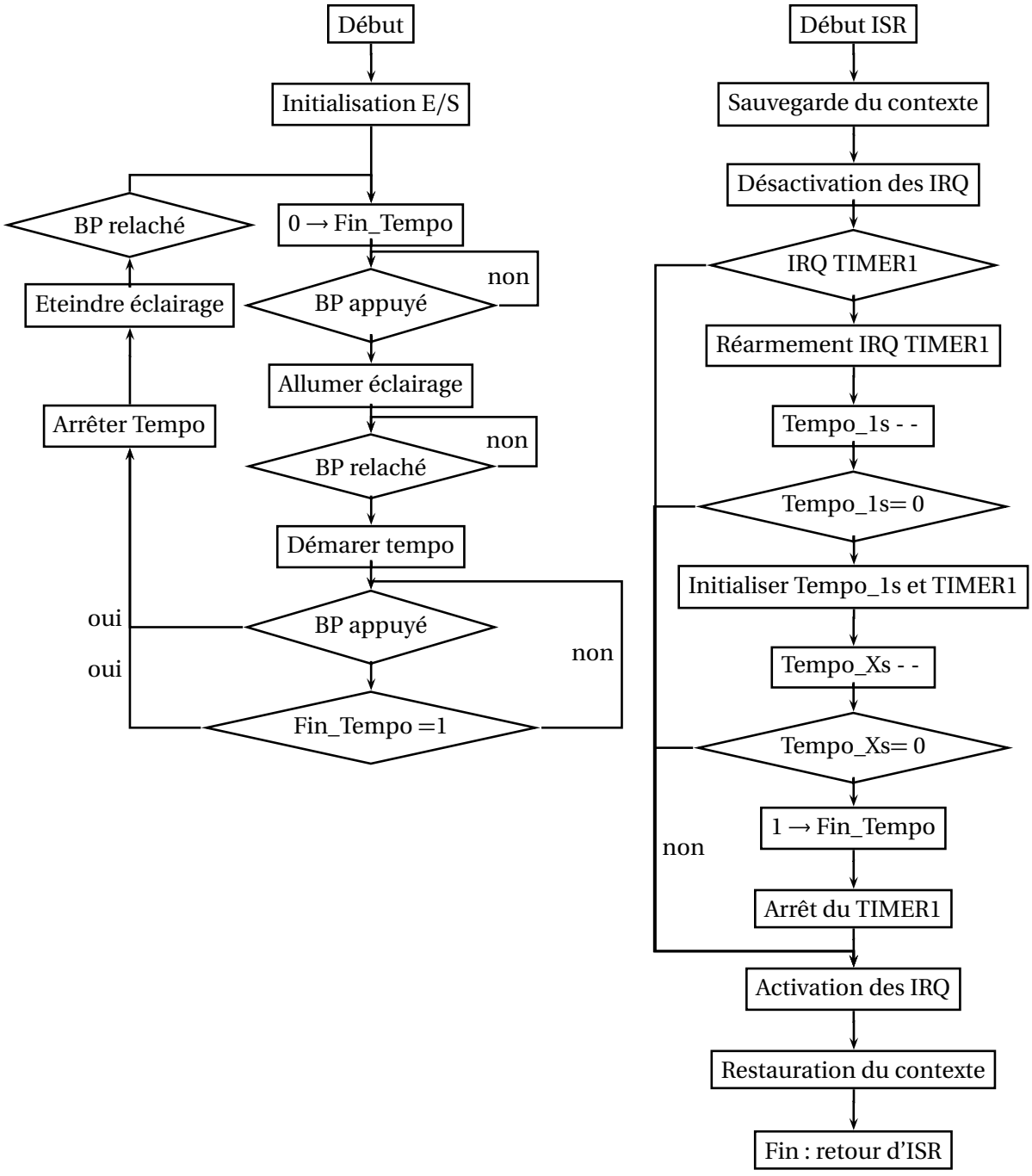


FIGURE 14: *Organigramme du programme principal (à gauche) et de la routine d'interruption (à droite).*

9. *Quel est le rôle de la variable Fin_Tempo ? Quel nom désigne ce genre de variable ? Où doit-elle être stockée ?*

10. *Estimer le temps d'exécution dans le pire des cas de la fonction d'interruption. Quelle est la différence entre cette fonction d'interruption et la fonction Tempo_Ws de la partie précédente ?*

Modification du code

- Ouvrir le workspace : TPuC_1IRQ.mcw
- Repérer la fonction d'interruption.

11. *Quel est le rôle du code org 0x0004 placé avant cette fonction ?*

Compléter le code de cette fonction : *INDICE*
— Au début de la fonction : sauvegarde du contexte (cf. Annexes)
— La fin de la fonction : restauration du contexte, Réactivation des interruptions et le retour de fonction d'interruption.

12. *Vérifier le bon fonctionnement de votre programme.*

Conclusions

13. *Quels sont les avantages et les inconvénients de la gestion par interruption, du points de vue de :*
— *la lisibilité de l'organigramme ?*
— *la lisibilité du programme ?*
— *la complexité de programmation ?*
— *l'utilisation des ressources micro-contrôleur ?*

4. Programmation avec deux interruptions

En gardant le même fonctionnement du programme, on souhaite maintenant déclencher une interruption lorsqu'un front se produit sur RB0. L'interruption sur le TIMER1 est inchangée. Avec le PIC utilisé, il est possible de déclencher une interruption sur l'un des fronts (montant ou descendant) de RB0.

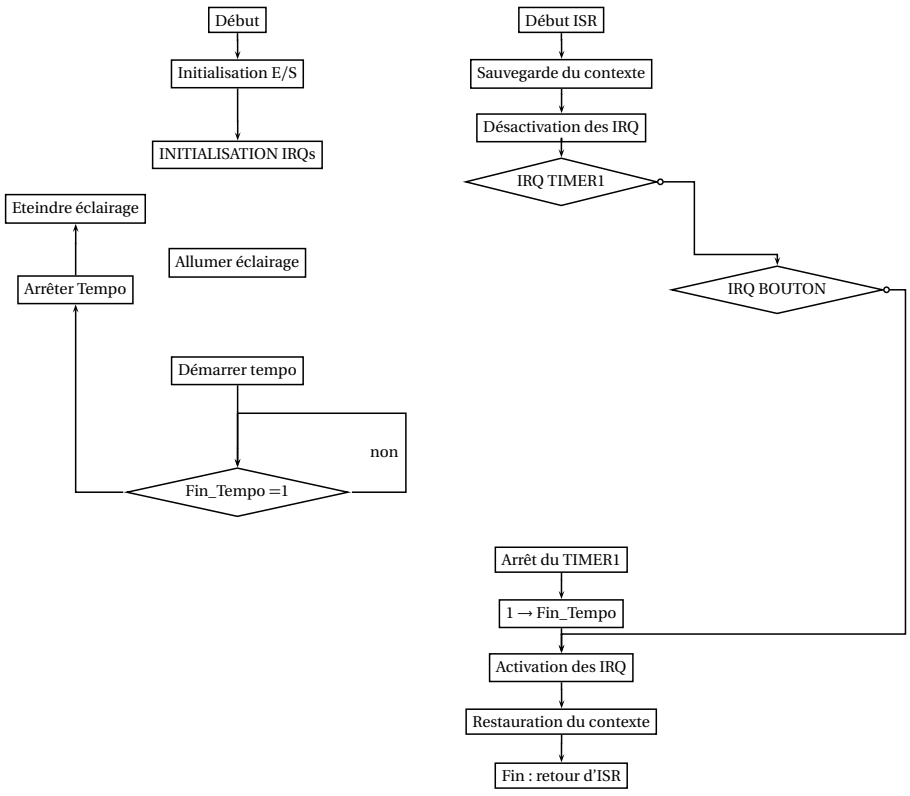


FIGURE 15: *Organigramme du programme principal (gauche) et de la routine d'interruption (droite)* ***à complèter.***

1 Questions

1. ***Organigramme*** *: Modifier les organigrammes précédents afin de gérer les deux sources d'interruptions.*

Prendre en compte les remarques suivantes :

- la même fonction d'interruption (ISR) est appelée pour les deux sources interruptions (TIMER1 et RB0),
- la fonction d'interruption ne mettra à jour qu'une seule variable (sémaphore),
- l'interruption sur le front de RB0 n'a pas besoin d'être arrêtée comme celle du TIMER1.

Paramétrage du déclenchement d'une interruption sur un front de RB0

2. *A partir des documents en annexe, donner les bits des registres à modifier ainsi que la valeur de ces bits pour obtenir le déclenchement d'une interruption sur les fronts montants de RB0. Donner aussi les banques dans lesquelles ces registres sont présents.*

Aide : lire la documentation du PORTB dans les annexes

3. *Lorsque le déclenchement d'une interruption sur un front de RB0 est correctement paramétré, quel bit de quel registre est mis à jour lors de l'apparition d'un front sur RB0 ?*

4. *Dans la fonction d'interruption, que faudra t'il faire de ce bit ? (attention deux choses à faire)*

2 Modification du code

Procéder par étapes !

Mise au point *Pour la mise au point, il est recommandé d'utiliser le simulateur (MPLAB SIM) puis le déboggeur (MPLAB PICkit3). Veillez à bien configurer l'accès aux banques des registres.*

3 Questions

1. *Faites les modifications à partir du projet précédent pour respecter le nouveau cahier des charges.*

Vous pourrez judicieusement faire les modifications suivantes :
- ajouter de la fonction de paramétrage de l'interruption,
- modifier la fonction d'interruption (en regard à votre organigramme).
- modifier le programme principal (en regard à votre organigramme).

2. *Si les deux interruptions se produisent en même temps, quelle est l'interruption traitée en premier ?*

3. *Quand est traitée la deuxième ? Proposer deux solutions pour pouvoir traiter la deuxième interruption.*

4. *Que se passe-t-il si une interruption se déclenche pendant l'exécution de la fonction de gestion d'interruption (récursivité des appels) ?*

5. *Comment valider la fiabilité du programme ?*

4 Conclusions

1. *Dans le cadre de cette application quelle solution adopter ?*

5. Pour aller plus loin

On souhaite réaliser un compteur d'impulsions. Pendant 10 secondes, on compte les impulsions sur RB0. Après les 10 secondes, le nombre d'impulsions est affiché en binaire sur les LEDs.

Proposer une solution pour cette application (organigramme(s) et code assembleur).

6. Codes assembleur des deux premiers projets

1 Programme "bouton poussoir intelligent" sans interruption

Code 1: TPuC_1.asm

```
#include <p16F887.inc>     ; processor specific variable definitions
__CONFIG _CONFIG1, _INTOSC & _DEBUG_ON  & _LVP_OFF & _FCMEN_OFF
         & _IESO_OFF & _BOR_OFF & _CPD_OFF & _CP_OFF & _MCLRE_ON
                  & _PWRTE_OFF & _WDT_OFF
__CONFIG _CONFIG2, _WRT_OFF & _BOR21V
;***** VARIABLE DEFINITIONS
Tempo_Ws_value  equ 0x0020
Tempo_Ws_mem    equ 0x0021
tempo           equ 0x0022
;**********************************************************************
    ORG     0x0000           ; processor reset vector
    goto    main             ; go to beginning of program
    ORG     0x0010
; attendre W secondes
Tempo_Ws:
    movwf   Tempo_Ws_value   ; dans W : le temps en seconde à attendre
    ; INIT compteur
    bsf     STATUS, RP0      ; Bank1
    clrf    PIE1             ; Disable peripheral interrupts
    bcf     STATUS, RP0      ; Bank0
    clrf    PIR1             ; Clear peripheral interrupts Flags
    movlw   0x00             ; Internal Clock source with 1:1 prescaler
    movwf   T1CON            ; Timer1 is stopped and T1 osc is disabled
    movlw   0x7b
    movwf   TMR1H            ;
    movlw   0x80
    movwf   TMR1L            ;
    bsf     T1CON, TMR1ON    ; Timer1 starts to increment

; 8 MHz / 4 = 2MHz  ===> Fosc/4
; prescale de 1 : 2MHZ  soit 2 000 000 incrémentations pour 1 s
; nombre de remplissage du compteur 16 bits : 2000000/65536 = 30,51 .... (>30)
; pour 30 : 30 * 65536 = 1966080 incrémentations réalisées
; il manque 2 000 000 - 1966080 = 33920 incr?mentations
; donc charger le compteur avec 65536 - 33920 = 31616 => 0x7B80
; il faut donc partir de cette valeur puis boucler 30 fois (1+30 = 31 it)
Tempo_Ws_B1:
    movlw   0x1E
    movwf   Tempo_Ws_mem
Tempo_Ws_B2:
Tempo_Ws_OVFL_WAIT:
    btfss PIR1, TMR1IF
    goto Tempo_Ws_OVFL_WAIT
; Timer has overflowed
    bcf PIR1, TMR1IF
; 9 *
    decfsz  Tempo_Ws_mem, f  ; s
```

```
48      goto    Tempo_Ws_B2      ; Tempo_Wms_Value != 0
49  ; et W *
50      decfsz  Tempo_Ws_value, f; s
51      goto    Tempo_Ws_B1      ; Tempo_Ws_Value != 0
52
53      bcf     T1CON, TMR1ON    ; Timer1 stops to increment
54      RETURN
55
56  ; *****************************************************************
57  main:
58
59      bsf   STATUS, RP0        ; Selection bank 1
60      bcf   TRISD, 0           ; mise à 0 du bit 0 de TRISB
61      bsf   TRISB, 0
62      bcf   STATUS, RP0        ; Selection bank 0
63      bcf   PORTD,0
64      ;INIT PORT B as DIGITAL INPUT
65      banksel ANSELH           ; Selection de la bank du resigtre ANSELH
66      clrf    ANSELH           ; Configuration du portB en digital
67      banksel WPUB             ; Selection de la bank où est le registre WPUB
68      bcf     WPUB,0           ; Désactivation du PULL-UP de l'entrée 0 du port B
69      banksel OSCCON           ; Selection de la bank où est situé OSCCON
70      movlw   b'01110001'      ; Configuration de l'horloge interne
71      movwf   OSCCON           ; HFINTOSC du pic à 8MHz
72      bcf     STATUS, RP0      ; Selection bank 0
73
74  main_prg:
75      ; init tempo
76      clrf tempo;
77      ; touche ?
78      btfsc PORTB, 0
79      goto main_prg
80      ; allumage
81      bsf   PORTD,0
82
83  main_attente:
84      ; attendre 1s
85      movlw   1
86      call    Tempo_Ws
87      ; touche ?
88      btfss   PORTB, 0
89      goto    main_eteindre
90      ; incrementer tempo : tempo = tempo + 1
91      incf    tempo
92
93      ; fin tempo ?
94      movf    tempo,w
95      sublw   .10
96      btfss   STATUS, Z
97      goto    main_attente ; Z=0
98
```

```
99  main_eteindre:
100     ; eteindre LED
101     bcf     PORTD, 0     ; LED eteinte
102     ; attendre 1s
103     movlw   .1
104     call    Tempo_Ws
105     bcf PORTD,0
106     ; retour au debut
107     goto    main_prg
108     END                  ; directive 'end of program'
```
Fin de code

Notes

2 Programme "bouton poussoir intelligent" AVEC une interruption (à compléter)

Code 2: TPuC_1IRQ.asm

```
; ATTENTION CE FICHIER EST A COMPLETER
;  (Sauvegarde et restauration du Context; Retour d'interruption)

#include <p16F887.inc>     ; definitions specifiques des variables
                            ; du processeurs
__CONFIG _CONFIG1, _INTOSC & _DEBUG_ON  & _LVP_OFF & _FCMEN_OFF
        & _IESO_OFF  & _BOR_OFF & _CPD_OFF & _CP_OFF & _MCLRE_ON
        & _PWRTE_OFF & _WDT_OFF
__CONFIG _CONFIG2, _WRT_OFF & _BOR21V

; Valeur tempo extinction automatique
#define TEMPO   .8         ; temporisation fixée à 8s

;***** VARIABLE DEFINITIONS
Tempo_1s        equ 0x0020 ; unité de temps ISR
Tempo_Xs        equ 0x0021 ; Tempo globale ISR
Fin_Tempo       equ 0x0022 ; Sémaphore

v_Tempo_10ms_1  equ 0x0023 ; pour tempo pour anti rebond
v_Tempo_10ms_2  equ 0x0024 ; pour tempo pour anti rebond

W_TEMP          equ 0x0025 ; Sauvegarde W pendant ISR
STATUS_TEMP     equ 0x0026 ; Sauvegarde STATUS pendant ISR

;*********************************************************************
    ORG     0x0000         ; vecteur reset
    goto    Main           ; aller au début du programme

;*********************************************************************
; Fontion de gestion de l'interruption
    ORG     0x0004         ; vecteur d'interruption
Routine_ISR:
    ; Sauvegarde du contexte


    ; blocage de nouvelles IRQ
    clrf    STATUS         ; bank 0
    bcf     INTCON, GIE

    ; detection IRQ
    btfsc   PIR1, TMR1IF   ; dépassement de TIMER1 ?
    goto    T1_INT         ; oui

    ; IRQ non geree
    goto    END_ISR

T1_INT:
    ; remise ? 0 du flag d'IRQ du TIMER1
    bcf     PIR1, TMR1IF
```

```
50
51      ; compteur de generation d'unité (seconde)
52      decfsz  Tempo_1s,f      ; 1 seconde ecoulée ?
53      goto    END_ISR         ; non, rien à faire
54
55      ; 1 seconde ecoul?e : re initialiser le TIMER1
56      movlw   .30
57      movwf   Tempo_1s
58      movlw   0x7B
59      movwf   TMR1H
60      movlw   0x80
61      movwf   TMR1L
62
63      ; decrementation compteur de seconde
64      decfsz  Tempo_Xs,f      ; décrémenter et test si fin tempo ?
65      goto    END_ISR         ; non
66      ; oui :
67      movlw   1               ; mettre le semaphore Fin_Tempo à 1
68      movwf   Fin_Tempo
69      bcf     T1CON, TMR1ON   ; Timer1 stop
70
71      goto    END_ISR
72
73          ; fin de routine d'interruption
74  END_ISR:
75          ; Restauration du contexte
76
77
78
79          ; retour d'interruption ET reactivation des IRQ
80
81  ; ******************************************************************
82  ; Lancer le TIMER 1 avec interruption
83  Start_TIMER1:
84      ; INIT IRQ
85      bsf     STATUS, RP0     ; Bank1
86      bcf     STATUS, RP1     ; Bank1
87      bcf     PIE1, TMR1IE    ; Disable TIMER1 interrupt
88      bcf     STATUS, RP0     ; Bank0
89      clrf    PIR1            ; Clear peripheral interrupts Flags
90
91      ; INIT des compteurs
92      movlw   TEMPO
93      movwf   Tempo_Xs        ; nb de secondes ? attendre
94      movlw   .30             ; 30 d?passements de TIMER1
95      movwf   Tempo_1s
96
97      ; Initialisation du Timer1
98      movlw   0x00            ; Internal Clock source avec 1:1 prescaler
99      movwf   T1CON           ; Timer1 is stopped and T1 osc is disabled
100     movlw   0x7B            ; valeur de départ (cf explication plus bas)
```

```
101     movwf   TMR1H
102     movlw   0x80
103     movwf   TMR1L
104
105     ; Activation IRQ TIMER1
106     bsf     STATUS, RP0    ; Bank1
107     bsf     PIE1, TMR1IE   ; activate TIMER1 interrupt
108
109     bcf     STATUS, RP0    ; Bank0
110     bsf     INTCON, PEIE   ; activate perif interrupt
111
112     bsf     T1CON, TMR1ON ; Timer1 starts to increment
113 ; 8 MHz / 4 = 2MHz  ===> Fosc/4
114 ; prescale de 1 : 2MHZ  soit 2 000 000 incrémentations pour 1 s
115 ; nombre de remplissage du compteur 16 bits : 2000000/65536 = 30,51 .... (>30)
116 ; pour 30 : 30 * 65536 = 1966080 incrémentations réalisées
117 ; il manque 2 000 000 - 1966080 = 33920 incr?mentations
118 ; donc charger le compteur avec 65536 - 33920 = 31616 => 0x7B80
119 ; il faut donc partir de cette valeur puis boucler 30 fois (1+30 = 31 it)
120     RETURN
121 ; ******************************************************************
122 ; Fonction d'arret du Timer1
123 Stop_TIMER1:
124     bcf     STATUS, RP0    ; Bank0
125     bcf     STATUS, RP1    ; Bank0
126     bcf     T1CON, TMR1ON  ; Timer1 stop to increment
127     RETURN
128 ; ******************************************************************
129 ; Fonction d'attente anti rebonds
130 Tempo_10ms:
131     movlw   .196
132     movwf   v_Tempo_10ms_1
133 Tempo_10ms_b1:
134     movlw   0xFF
135     movwf   v_Tempo_10ms_2
136 Tempo_10ms_b2:
137     decfsz  v_Tempo_10ms_2,f
138     goto    Tempo_10ms_b2
139
140     decfsz  v_Tempo_10ms_1,f
141     goto    Tempo_10ms_b1
142     RETURN
143 ; ******************************************************************
144 ; Fonction d'attente que RB0 soit enfoncé
145 RB0_Enfonce:
146     btfsc   PORTB, 0
147     goto    RB0_Enfonce
148     call    Tempo_10ms
149     RETURN
150 ; ******************************************************************
151 ; Fonction d'attente que RB0 soit relaché
```

```
152 RB0_Relache:
153     btfss   PORTB, 0
154     goto    RB0_Relache
155     call    Tempo_10ms
156     RETURN
157 ; *****************************************************************
158 ; *****************************************************************
159 ; PROGRAMME PRINCIPAL
160 Main:
161     ; INIT DES REGISTRES D'IRQ
162      clrf   INTCON          ; blocage
163      bsf    INTCON, GIE     ; activation des interruptions globales
164 
165      bsf    STATUS, RP0     ; Selection bank 1
166      clrf   PIE1
167      bcf   STATUS, RP0      ; Selection bank 0
168 
169     ; INIT DES E/S
170      bsf    STATUS, RP0     ; Selection bank 1
171      bcf    TRISD, 0        ; mise ? 0 du bit 0 de TRISB
172      bsf    TRISB, 0
173      bcf    STATUS, RP0     ; Selection bank 0
174      bcf    PORTD, 0
175 
176      ;INIT PORT B as DIGITAL INPUT
177      banksel    ANSELH      ; Selection de la bank où est situé ANSELH
178      clrf       ANSELH      ; Configuration du portB en digital
179      banksel    WPUB        ; Selection de la bank où est situé WPUB
180      bcf        WPUB,0      ; Désactivation du PULL-UP de l'entrée 0 du port B
181 
182      banksel    OSCCON      ; Selection de la bank où est situé OSCCON
183      movlw      b'01110001'; Configuration de l'horloge interne
184      movwf      OSCCON      ; HFINTOSC du pic à 8MHz
185      bcf        STATUS, RP0; Selection bank 0
186 
187 Main_Boucle:
188     ; INIT des variables
189     clrf    Fin_Tempo
190     ; Attendre BP enfoncé
191     call    RB0_Enfonce
192 
193     ; Allumer la led
194     bsf     PORTD, 0
195 
196     ; Attendre BP relaché
197     call    RB0_Relache
198 
199     ; Démarrage du timer
200     call    Start_TIMER1
201 
202 Main_Attente:
```

```
203     ; RB0 enfoncé ?
204     btfss   PORTB, 0
205     goto    Main_Eteindre
206
207     ; Fin tempo ?
208     btfss   Fin_Tempo,0
209     goto    Main_Attente   ; Fin_tempo = 0 => pas fini
210
211 Main_Eteindre:
212     ; Arreter TIMER1 (cas de l'appuie sur le BP)
213     call    Stop_TIMER1
214
215     ; Eteindre la led
216     bcf     PORTD, 0       ; LED eteinte
217
218     ; Attendre BP relaché
219     call    RB0_Relache
220
221     ; retour au debut
222     goto    Main_Boucle
223
224     END                    ; directive 'end of program'
```

Fin de code

Notes

Département Génie Électrique – INSA de Lyon

Travaux pratiques
Micro-contrôleur PIC

Annexes techniques

MODULE IF3

– Olivier Bernard – Nicolas Ducros – Thomas Grenier –
Dominique Tournier

INSA de Lyon - Département Génie Électrique - 4ème année

A. Annexes

1 Extraits de la documentation technique Microchip du PIC 16F887

TABLE 2: *Descriptif des extraits de la documentation technique*

Descriptif	Page(s)
REGISTRES, PLAN MEMOIRE ET REGISTRES SPECIFIQUES	27-36
DESCRIPTION DES PORT B & D : schéma et registres	47-52 ;57-58
TIMER1 : schéma, fonctionnement et registres de configuration	76-80
INTERRUPTION : description et mise en oeuvre	216-219
JEU D'INSTRUCTIONS	226

FIGURE 2-6: PIC16F886/PIC16F887 SPECIAL FUNCTION REGISTERS

	File Address		File Address		File Address		File Address
Indirect addr. (1)	00h	Indirect addr. (1)	80h	Indirect addr. (1)	100h	Indirect addr. (1)	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h	WDTCON	105h	SRCON	185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	CM1CON0	107h	BAUDCTL	187h
PORTD(2)	08h	TRISD(2)	88h	CM2CON0	108h	ANSEL	188h
PORTE	09h	TRISE	89h	CM2CON1	109h	ANSELH	189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDAT	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2(1)	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved	18Eh
TMR1H	0Fh	OSCCON	8Fh	EEADRH	10Fh	Reserved	18Fh
T1CON	10h	OSCTUNE	90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h	WPUB	95h		115h		195h
CCPR1H	16h	IOCB	96h	General Purpose Registers	116h	General Purpose Registers	196h
CCP1CON	17h	VRCON	97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h	16 Bytes	119h	16 Bytes	199h
RCREG	1Ah	SPBRGH	9Ah		11Ah		19Ah
CCPR2L	1Bh	PWM1CON	9Bh		11Bh		19Bh
CCPR2H	1Ch	ECCPAS	9Ch		11Ch		19Ch
CCP2CON	1Dh	PSTRCON	9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Registers 96 Bytes	20h – 6Fh	General Purpose Registers 80 Bytes	A0h – EFh	General Purpose Registers 80 Bytes	120h – 16Fh	General Purpose Registers 80 Bytes	1A0h – 1EFh
	70h – 7Fh	accesses 70h-7Fh	F0h – FFh	accesses 70h-7Fh	170h – 17Fh	accesses 70h-7Fh	1F0h – 1FFh
Bank 0		Bank 1		Bank 2		Bank 3	

Unimplemented data memory locations, read as '0'.

Note 1: Not a physical register.

2: PIC16F887 only.

TABLE 2-1: PIC16F882/883/884/886/887 SPECIAL FUNCTION REGISTERS SUMMARY BANK 0

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 0											
00h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	37,213
01h	TMR0	Timer0 Module Register								xxxx xxxx	73,213
02h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	37,213
03h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	29,213
04h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	37,213
05h	PORTA(3)	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxxx xxxx	39,213
06h	PORTB(3)	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	48,213
07h	PORTC(3)	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	53,213
08h	PORTD(3,4)	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	57,213
09h	PORTE(3)	—	—	—	—	RE3	RE2(4)	RE1(4)	RE0(4)	---- xxxx	59,213
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of Program Counter					---0 0000	37,213
0Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF(1)	0000 000x	31,213
0Ch	PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	-000 0000	34,213
0Dh	PIR2	OSFIF	C2IF	C1IF	EEIF	BCLIF	ULPWUIF	—	CCP2IF	0000 00-0	35,213
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	76,213
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	76,213
10h	T1CON	T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	0000 0000	79,213
11h	TMR2	Timer2 Module Register								0000 0000	81,213
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	82,213
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	179,213
14h	SSPCON(2)	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	177,213
15h	CCPR1L	Capture/Compare/PWM Register 1 Low Byte (LSB)								xxxx xxxx	126,213
16h	CCPR1H	Capture/Compare/PWM Register 1 High Byte (MSB)								xxxx xxxx	126,213
17h	CCP1CON	P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	0000 0000	124,213
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	159,213
19h	TXREG	EUSART Transmit Data Register								0000 0000	151,213
1Ah	RCREG	EUSART Receive Data Register								0000 0000	156,213
1Bh	CCPR2L	Capture/Compare/PWM Register 2 Low Byte (LSB)								xxxx xxxx	126,213
1Ch	CCPR2H	Capture/Compare/PWM Register 2 High Byte (MSB)								xxxx xxxx	126,214
1Dh	CCP2CON	—	—	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	125,214
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	99,214
1Fh	ADCON0	ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	ADON	0000 0000	104,214

Legend: – = Unimplemented locations read as '0', u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented

Note 1: $\overline{MCLR}$ and WDT Reset do not affect the previous value data latch. The RBIF bit will be cleared upon Reset but will set again if the mismatch exists.

2: When SSPCON register bits SSPM<3:0> = 1001, any reads or writes to the SSPADD SFR address are accessed through the SSPMSK register. See Registers • and 13-4 for more detail.

3: Port pins with analog functions controlled by the ANSEL and ANSELH registers will read '0' immediately after a Reset even though the data latches are either undefined (POR) or unchanged (other Resets).

4: PIC16F884/PIC16F887 only.

TABLE 2-2: PIC16F882/883/884/886/887 SPECIAL FUNCTION REGISTERS SUMMARY BANK 1

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 1											
80h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	37,213
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	30,214
82h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	37,213
83h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	29,213
84h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	37,213
85h	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111	39,214
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	48,214
87h	TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0	1111 1111	53,214
88h	TRISD(3)	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0	1111 1111	57,214
89h	TRISE	—	—	—	—	TRISE3	TRISE2(3)	TRISE1(3)	TRISE0(3)	---- 1111	59,214
8Ah	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	37,213
8Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF(1)	0000 000x	31,213
8Ch	PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	-000 0000	32,214
8Dh	PIE2	OSFIE	C2IE	C1IE	EEIE	BCLIE	ULPWUIE	—	CCP2IE	0000 00-0	33,214
8Eh	PCON	—	—	ULPWUE	SBOREN	—	—	$\overline{POR}$	$\overline{BOR}$	--01 --qq	36,214
8Fh	OSCCON	—	IRCF2	IRCF1	IRCF0	OSTS	HTS	LTS	SCS	-110 q000	62,214
90h	OSCTUNE	—	—	—	TUN4	TUN3	TUN2	TUN1	TUN0	---0 0000	66,214
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	177,214
92h	PR2	Timer2 Period Register								1111 1111	81,214
93h	SSPADD(2)	Synchronous Serial Port (I^2C mode) Address Register								0000 0000	185,214
93h	SSPMSK(2)	MSK7	MSK6	MSK5	MSK4	MSK3	MSK2	MSK1	MSK0	1111 1111	204,214
94h	SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	185,214
95h	WPUB	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0	1111 1111	49,214
96h	IOCB	IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0	0000 0000	49,214
97h	VRCON	VREN	VROE	VRR	VRSS	VR3	VR2	VR1	VR0	0000 0000	97,214
98h	TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	0000 0010	158,214
99h	SPBRG	BRG7	BRG6	BRG5	BRG4	BRG3	BRG2	BRG1	BRG0	0000 0000	161,214
9Ah	SPBRGH	BRG15	BRG14	BRG13	BRG12	BRG11	BRG10	BRG9	BRG8	0000 0000	161,214
9Bh	PWM1CON	PRSEN	PDC6	PDC5	PDC4	PDC3	PDC2	PDC1	PDC0	0000 0000	144,214
9Ch	ECCPAS	ECCPASE	ECCPAS2	ECCPAS1	ECCPAS0	PSSAC1	PSSAC0	PSSBD1	PSSBD0	0000 0000	141,214
9Dh	PSTRCON	—	—	—	STRSYNC	STRD	STRC	STRB	STRA	---0 0001	145,214
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	99,214
9Fh	ADCON1	ADFM	—	VCFG1	VCFG0	—	—	—	—	0-00 ----	105,214

Legend: – = Unimplemented locations read as '0', u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented

Note 1: $\overline{MCLR}$ and WDT Reset do not affect the previous value data latch. The RBIF bit will be cleared upon Reset but will set again if the mismatch exists.

2: Accessible only when SSPCON register bits SSPM<3:0> = 1001.

3: PIC16F884/PIC16F887 only.

TABLE 2-3: PIC16F882/883/884/886/887 SPECIAL FUNCTION REGISTERS SUMMARY BANK 2

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 2											
100h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	37,213
101h	TMR0	Timer0 Module Register								xxxx xxxx	73,213
102h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	37,213
103h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	29,213
104h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	37,213
105h	WDTCON	—	—	—	WDTPS3	WDTPS2	WDTPS1	WDTPS0	SWDTEN	---0 1000	221,214
106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	48,213
107h	CM1CON0	C1ON	C1OUT	C1OE	C1POL	—	C1R	C1CH1	C1CH0	0000 -000	88,214
108h	CM2CON0	C2ON	C2OUT	C2OE	C2POL	—	C2R	C2CH1	C2CH0	0000 -000	89,214
109h	CM2CON1	MC1OUT	MC2OUT	C1RSEL	C2RSEL	—	—	T1GSS	C2SYNC	0000 --10	91,215
10Ah	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	37,213
10Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF(1)	0000 000x	31,213
10Ch	EEDAT	EEDAT7	EEDAT6	EEDAT5	EEDAT4	EEDAT3	EEDAT2	EEDAT1	EEDAT0	0000 0000	112,215
10Dh	EEADR	EEADR7	EEADR6	EEADR5	EEADR4	EEADR3	EEADR2	EEADR1	EEADR0	0000 0000	112,215
10Eh	EEDATH	—	—	EEDATH5	EEDATH4	EEDATH3	EEDATH2	EEDATH1	EEDATH0	--00 0000	112,215
10Fh	EEADRH	—	—	—	EEADRH4(2)	EEADRH3	EEADRH2	EEADRH1	EEADRH0	---- 0000	112,215

Legend: – = Unimplemented locations read as '0', u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented
Note 1: $\overline{MCLR}$ and WDT Reset does not affect the previous value data latch. The RBIF bit will be cleared upon Reset but will set again if the mismatch exists.
2: PIC16F886/PIC16F887 only.

TABLE 2-4: PIC16F882/883/884/886/887 SPECIAL FUNCTION REGISTERS SUMMARY BANK 3

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 3											
180h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	37,213
181h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	30,214
182h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	37,213
183h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	29,213
184h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	37,213
185h	SRCON	SR1	SR0	C1SEN	C2REN	PULSS	PULSR	—	FVREN	0000 00-0	93,215
186h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	48,214
187h	BAUDCTL	ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN	01-0 0-00	160,215
188h	ANSEL	ANS7(2)	ANS6(2)	ANS5(2)	ANS4	ANS3	ANS2	ANS1	ANS0	1111 1111	40,215
189h	ANSELH	—	—	ANS13	ANS12	ANS11	ANS10	ANS9	ANS8	--11 1111	99,215
18Ah	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	37,213
18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF(1)	0000 000x	31,213
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	113,215
18Dh	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	111,215

Legend: – = Unimplemented locations read as '0', u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented
Note 1: $\overline{MCLR}$ and WDT Reset does not affect the previous value data latch. The RBIF bit will be cleared upon Reset but will set again if the mismatch exists.
2: PIC16F884/PIC16F887 only.

2.2.2.1 STATUS Register

The STATUS register, shown in Register 2-1, contains:

- the arithmetic status of the ALU
- the Reset status
- the bank select bits for data memory (GPR and SFR)

The STATUS register can be the destination for any instruction, like any other register. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to the device logic. Furthermore, the $\overline{TO}$ and $\overline{PD}$ bits are not writable. Therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, `CLRF STATUS`, will clear the upper three bits and set the Z bit. This leaves the STATUS register as `'000u u1uu'` (where `u` = unchanged).

It is recommended, therefore, that only `BCF`, `BSF`, `SWAPF` and `MOVWF` instructions are used to alter the STATUS register, because these instructions do not affect any Status bits. For other instructions not affecting any Status bits, see **Section 15.0 "Instruction Set Summary"**

Note 1: The C and DC bits operate as a $\overline{Borrow}$ and Digit Borrow out bit, respectively, in subtraction.

REGISTER 2-1: STATUS: STATUS REGISTER

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC(1)	C(1)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **IRP:** Register Bank Select bit (used for indirect addressing)
`1` = Bank 2, 3 (100h-1FFh)
`0` = Bank 0, 1 (00h-FFh)

bit 6-5 **RP<1:0>:** Register Bank Select bits (used for direct addressing)
`00` = Bank 0 (00h-7Fh)
`01` = Bank 1 (80h-FFh)
`10` = Bank 2 (100h-17Fh)
`11` = Bank 3 (180h-1FFh)

bit 4 **$\overline{TO}$:** Time-out bit
`1` = After power-up, `CLRWDT` instruction or `SLEEP` instruction
`0` = A WDT time-out occurred

bit 3 **$\overline{PD}$:** Power-down bit
`1` = After power-up or by the `CLRWDT` instruction
`0` = By execution of the `SLEEP` instruction

bit 2 **Z:** Zero bit
`1` = The result of an arithmetic or logic operation is zero
`0` = The result of an arithmetic or logic operation is not zero

bit 1 **DC:** Digit Carry/$\overline{Borrow}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)(1)
`1` = A carry-out from the 4th low-order bit of the result occurred
`0` = No carry-out from the 4th low-order bit of the result

bit 0 **C:** Carry/$\overline{Borrow}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)(1)
`1` = A carry-out from the Most Significant bit of the result occurred
`0` = No carry-out from the Most Significant bit of the result occurred

Note 1: For $\overline{Borrow}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high-order or low-order bit of the source register.

2.2.2.2 OPTION Register

The OPTION register, shown in Register 2-2, is a readable and writable register, which contains various control bits to configure:

- Timer0/WDT prescaler
- External INT interrupt
- Timer0
- Weak pull-ups on PORTB

Note:	To achieve a 1:1 prescaler assignment for Timer0, assign the prescaler to the WDT by setting PSA bit of the OPTION register to '1'. See **Section 6.3 "Timer1 Prescaler"**.

REGISTER 2-2: OPTION_REG: OPTION REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual PORT latch values

bit 6 **INTEDG:** Interrupt Edge Select bit
1 = Interrupt on rising edge of INT pin
0 = Interrupt on falling edge of INT pin

bit 5 **T0CS:** Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (FOSC/4)

bit 4 **T0SE:** Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA:** Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS<2:0>:** Prescaler Rate Select bits

Bit Value	Timer0 Rate	WDT Rate
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

2.2.2.3 INTCON Register

The INTCON register, shown in Register 2-3, is a readable and writable register, which contains the various enable and flag bits for TMR0 register overflow, PORTB change and external INT pin interrupts.

Note:	Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the Global Enable bit, GIE of the INTCON register. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt.

REGISTER 2-3: INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	T0IE	INTE	RBIE(1,3)	T0IF(2)	INTF	RBIF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **PEIE:** Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **T0IE:** Timer0 Overflow Interrupt Enable bit
1 = Enables the Timer0 interrupt
0 = Disables the Timer0 interrupt

bit 4 **INTE:** INT External Interrupt Enable bit
1 = Enables the INT external interrupt
0 = Disables the INT external interrupt

bit 3 **RBIE:** PORTB Change Interrupt Enable bit(1,3)
1 = Enables the PORTB change interrupt
0 = Disables the PORTB change interrupt

bit 2 **T0IF:** Timer0 Overflow Interrupt Flag bit(2)
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF:** INT External Interrupt Flag bit
1 = The INT external interrupt occurred (must be cleared in software)
0 = The INT external interrupt did not occur

bit 0 **RBIF:** PORTB Change Interrupt Flag bit
1 = When at least one of the PORTB general purpose I/O pins changed state (must be cleared in software)
0 = None of the PORTB general purpose I/O pins have changed state

Note 1: IOCB register must also be enabled.
2: T0IF bit is set when Timer0 rolls over. Timer0 is unchanged on Reset and should be initialized before clearing T0IF bit.
3: Includes ULPWU interrupt.

2.2.2.4 PIE1 Register

The PIE1 register contains the interrupt enable bits, as shown in Register 2-4.

Note: Bit PEIE of the INTCON register must be set to enable any peripheral interrupt.

REGISTER 2-4: PIE1: PERIPHERAL INTERRUPT ENABLE REGISTER 1

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'

bit 6 **ADIE:** A/D Converter (ADC) Interrupt Enable bit
1 = Enables the ADC interrupt
0 = Disables the ADC interrupt

bit 5 **RCIE:** EUSART Receive Interrupt Enable bit
1 = Enables the EUSART receive interrupt
0 = Disables the EUSART receive interrupt

bit 4 **TXIE:** EUSART Transmit Interrupt Enable bit
1 = Enables the EUSART transmit interrupt
0 = Disables the EUSART transmit interrupt

bit 3 **SSPIE:** Master Synchronous Serial Port (MSSP) Interrupt Enable bit
1 = Enables the MSSP interrupt
0 = Disables the MSSP interrupt

bit 2 **CCP1IE:** CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE:** Timer2 to PR2 Match Interrupt Enable bit
1 = Enables the Timer2 to PR2 match interrupt
0 = Disables the Timer2 to PR2 match interrupt

bit 0 **TMR1IE:** Timer1 Overflow Interrupt Enable bit
1 = Enables the Timer1 overflow interrupt
0 = Disables the Timer1 overflow interrupt

2.2.2.6 PIR1 Register

The PIR1 register contains the interrupt flag bits, as shown in Register 2-6.

Note:	Interrupt flag bits are set when an interrupt condition occurs, regardless of the state of its corresponding enable bit or the Global Enable bit, GIE of the INTCON register. User software should ensure the appropriate interrupt flag bits are clear prior to enabling an interrupt.

REGISTER 2-6: PIR1: PERIPHERAL INTERRUPT REQUEST REGISTER 1

U-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **Unimplemented:** Read as '0'

bit 6 **ADIF:** A/D Converter Interrupt Flag bit
1 = A/D conversion complete (must be cleared in software)
0 = A/D conversion has not completed or has not been started

bit 5 **RCIF:** EUSART Receive Interrupt Flag bit
1 = The EUSART receive buffer is full (cleared by reading RCREG)
0 = The EUSART receive buffer is not full

bit 4 **TXIF:** EUSART Transmit Interrupt Flag bit
1 = The EUSART transmit buffer is empty (cleared by writing to TXREG)
0 = The EUSART transmit buffer is full

bit 3 **SSPIF:** Master Synchronous Serial Port (MSSP) Interrupt Flag bit
1 = The MSSP interrupt condition has occurred, and must be cleared in software before returning from the Interrupt Service Routine. The conditions that will set this bit are:
SPI
A transmission/reception has taken place
I^2C Slave/Master
A transmission/reception has taken place
I^2C Master
The initiated Start condition was completed by the MSSP module
The initiated Stop condition was completed by the MSSP module
The initiated restart condition was completed by the MSSP module
The initiated Acknowledge condition was completed by the MSSP module
A Start condition occurred while the MSSP module was idle (Multi-master system)
A Stop condition occurred while the MSSP module was idle (Multi-master system)
0 = No MSSP interrupt condition has occurred

bit 2 **CCP1IF:** CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode

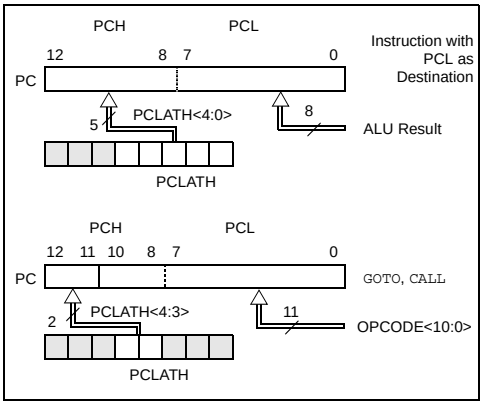
bit 1 **TMR2IF:** Timer2 to PR2 Interrupt Flag bit
1 = A Timer2 to PR2 match occurred (must be cleared in software)
0 = No Timer2 to PR2 match occurred

bit 0 **TMR1IF:** Timer1 Overflow Interrupt Flag bit
1 = The TMR1 register overflowed (must be cleared in software)
0 = The TMR1 register did not overflow

2.3 PCL and PCLATH

The Program Counter (PC) is 13 bits wide. The low byte comes from the PCL register, which is a readable and writable register. The high byte (PC<12:8>) is not directly readable or writable and comes from PCLATH. On any Reset, the PC is cleared. Figure 2-7 shows the two situations for the loading of the PC. The upper example in Figure 2-7 shows how the PC is loaded on a write to PCL (PCLATH<4:0> → PCH). The lower example in Figure 2-7 shows how the PC is loaded during a CALL or GOTO instruction (PCLATH<4:3> → PCH).

FIGURE 2-7: LOADING OF PC IN DIFFERENT SITUATIONS



2.3.1 MODIFYING PCL

Executing any instruction with the PCL register as the destination simultaneously causes the Program Counter PC<12:8> bits (PCH) to be replaced by the contents of the PCLATH register. This allows the entire contents of the program counter to be changed by writing the desired upper 5 bits to the PCLATH register. When the lower 8 bits are written to the PCL register, all 13 bits of the program counter will change to the values contained in the PCLATH register and those being written to the PCL register.

A computed GOTO is accomplished by adding an offset to the program counter (ADDWF PCL). Care should be exercised when jumping into a look-up table or program branch table (computed GOTO) by modifying the PCL register. Assuming that PCLATH is set to the table start address, if the table length is greater than 255 instructions or if the lower 8 bits of the memory address rolls over from 0xFF to 0x00 in the middle of the table, then PCLATH must be incremented for each address rollover that occurs between the table beginning and the target location within the table.

For more information refer to Application Note AN556, "*Implementing a Table Read*" (DS00556).

2.3.2 STACK

The PIC16F882/883/884/886/887 devices have an 8-level x 13-bit wide hardware stack (see Figures 2-2 and 2-3). The stack space is not part of either program or data space and the Stack Pointer is not readable or writable. The PC is PUSHed onto the stack when a CALL instruction is executed or an interrupt causes a branch. The stack is POPed in the event of a RETURN, RETLW or a RETFIE instruction execution. PCLATH is not affected by a PUSH or POP operation.

The stack operates as a circular buffer. This means that after the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push. The tenth push overwrites the second push (and so on).

Note 1:	There are no Status bits to indicate stack overflow or stack underflow conditions.
2:	There are no instructions/mnemonics called PUSH or POP. These are actions that occur from the execution of the CALL, RETURN, RETLW and RETFIE instructions or the vectoring to an interrupt address.

2.4 Indirect Addressing, INDF and FSR Registers

The INDF register is not a physical register. Addressing the INDF register will cause indirect addressing.

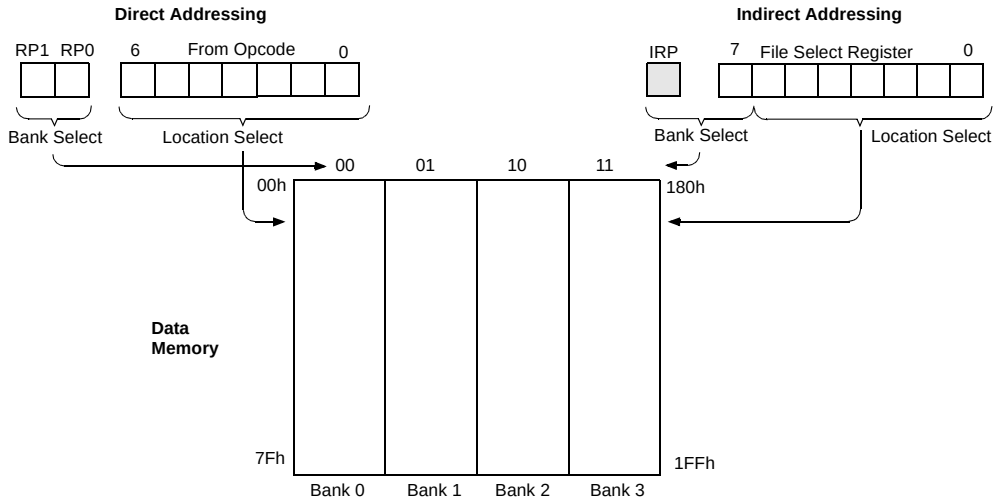
Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses data pointed to by the File Select Register (FSR). Reading INDF itself indirectly will produce 00h. Writing to the INDF register indirectly results in a no operation (although Status bits may be affected). An effective 9-bit address is obtained by concatenating the 8-bit FSR and the IRP bit of the STATUS register, as shown in Figure 2-8.

A simple program to clear RAM location 20h-2Fh using indirect addressing is shown in Example 2-1.

EXAMPLE 2-1: INDIRECT ADDRESSING

```
        MOVLW   0x20    ;initialize pointer
        MOVWF   FSR     ;to RAM
NEXT    CLRF    INDF    ;clear INDF register
        INCF    FSR     ;inc pointer
        BTFSS   FSR,4   ;all done?
        GOTO    NEXT    ;no clear next
CONTINUE                ;yes continue
```

FIGURE 2-8: DIRECT/INDIRECT ADDRESSING PIC16F882/883/884/886/887



Note: For memory map detail, see Figures 2-2 and 2-3.

3.3 PORTB and TRISB Registers

PORTB is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISB (Register 3-6). Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISB bit (= 0) will make the corresponding PORTB pin an output (i.e., enable the output driver and put the contents of the output latch on the selected pin). Example 3-3 shows how to initialize PORTB.

Reading the PORTB register (Register 3-5) reads the status of the pins, whereas writing to it will write to the PORT latch. All write operations are read-modify-write operations. Therefore, a write to a port implies that the port pins are read, this value is modified and then written to the PORT data latch.

The TRISB register (Register 3-6) controls the PORTB pin output drivers, even when they are being used as analog inputs. The user should ensure the bits in the TRISB register are maintained set when using them as analog inputs. I/O pins configured as analog input always read '0'. Example 3-3 shows how to initialize PORTB.

EXAMPLE 3-3: INITIALIZING PORTB

```
BANKSEL PORTB       ;
CLRF    PORTB       ;Init PORTB
BANKSEL TRISB       ;
MOVLW   B'11110000' ;Set RB<7:4> as inputs
                    ;and RB<3:0> as outputs
MOVWF   TRISB       ;
```

Note:	The ANSELH register must be initialized to configure an analog channel as a digital input. Pins configured as analog inputs will read '0'.

3.4 Additional PORTB Pin Functions

PORTB pins RB<7:0> on the device family device have an interrupt-on-change option and a weak pull-up option. The following three sections describe these PORTB pin functions.

Every PORTB pin on this device family has an interrupt-on-change option and a weak pull-up option.

3.4.1 ANSELH REGISTER

The ANSELH register (Register 3-4) is used to configure the Input mode of an I/O pin to analog. Setting the appropriate ANSELH bit high will cause all digital reads on the pin to be read as '0' and allow analog functions on the pin to operate correctly.

The state of the ANSELH bits has no affect on digital output functions. A pin with TRIS clear and ANSELH set will still operate as a digital output, but the Input mode will be analog. This can cause unexpected behavior when executing read-modify-write instructions on the affected port.

3.4.2 WEAK PULL-UPS

Each of the PORTB pins has an individually configurable internal weak pull-up. Control bits WPUB<7:0> enable or disable each pull-up (see Register 3-7). Each weak pull-up is automatically turned off when the port pin is configured as an output. All pull-ups are disabled on a Power-on Reset by the $\overline{RBPU}$ bit of the OPTION register.

3.4.3 INTERRUPT-ON-CHANGE

All of the PORTB pins are individually configurable as an interrupt-on-change pin. Control bits IOCB<7:0> enable or disable the interrupt function for each pin. Refer to Register 3-8. The interrupt-on-change feature is disabled on a Power-on Reset.

For enabled interrupt-on-change pins, the present value is compared with the old value latched on the last read of PORTB to determine which bits have changed or mismatched the old value. The 'mismatch' outputs of the last read are OR'd together to set the PORTB Change Interrupt flag bit (RBIF) in the INTCON register.

This interrupt can wake the device from Sleep. The user, in the Interrupt Service Routine, clears the interrupt by:

a) Any read or write of PORTB. This will end the mismatch condition.
b) Clear the flag bit RBIF.

A mismatch condition will continue to set flag bit RBIF. Reading or writing PORTB will end the mismatch condition and allow flag bit RBIF to be cleared. The latch holding the last read value is not affected by a $\overline{MCLR}$ nor Brown-out Reset. After these Resets, the RBIF flag will continue to be set if a mismatch is present.

Note:	If a change on the I/O pin should occur when the read operation is being executed (start of the Q2 cycle), then the RBIF interrupt flag may not get set. Furthermore, since a read or write on a port affects all bits of that port, care must be taken when using multiple pins in Interrupt-on-Change mode. Changes on one pin may not be seen while servicing changes on another pin.

REGISTER 3-4: ANSELH: ANALOG SELECT HIGH REGISTER

U-0	U-0	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
—	—	ANS13	ANS12	ANS11	ANS10	ANS9	ANS8
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-6 **Unimplemented:** Read as '0'

bit 5-0 **ANS<13:8>**: Analog Select bits
Analog select between analog or digital function on pins AN<13:8>, respectively.
1 = Analog input. Pin is assigned as analog input**(1)**.
0 = Digital I/O. Pin is assigned to port or special function.

Note 1: Setting a pin to an analog input automatically disables the digital input circuitry, weak pull-ups, and interrupt-on-change if available. The corresponding TRIS bit must be set to Input mode in order to allow external control of the voltage on the pin.

REGISTER 3-5: PORTB: PORTB REGISTER

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **RB<7:0>**: PORTB I/O Pin bit
1 = Port pin is > V_{IH}
0 = Port pin is < V_{IL}

REGISTER 3-6: TRISB: PORTB TRI-STATE REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **TRISB<7:0>:** PORTB Tri-State Control bit
1 = PORTB pin configured as an input (tri-stated)
0 = PORTB pin configured as an output

REGISTER 3-7: WPUB: WEAK PULL-UP PORTB REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **WPUB<7:0>**: Weak Pull-up Register bit
1 = Pull-up enabled
0 = Pull-up disabled

Note 1: Global $\overline{\text{RBPU}}$ bit of the OPTION register must be cleared for individual pull-ups to be enabled.
2: The weak pull-up device is automatically disabled if the pin is in configured as an output.

REGISTER 3-8: IOCB: INTERRUPT-ON-CHANGE PORTB REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **IOCB<7:0>:** Interrupt-on-Change PORTB Control bit
1 = Interrupt-on-change enabled
0 = Interrupt-on-change disabled

3.4.4 PIN DESCRIPTIONS AND DIAGRAMS

Each PORTB pin is multiplexed with other functions. The pins and their combined functions are briefly described here. For specific information about individual functions such as the SSP, I^2C or interrupts, refer to the appropriate section in this data sheet.

3.4.4.1 RB0/AN12/INT

Figure 3-9 shows the diagram for this pin. This pin is configurable to function as one of the following:

- a general purpose I/O
- an analog input for the ADC
- an external edge triggered interrupt

3.4.4.2 RB1/AN10/P1C(1)/C12IN3-

Figure 3-9 shows the diagram for this pin. This pin is configurable to function as one of the following:

- a general purpose I/O
- an analog input for the ADC
- a PWM output(1)
- an analog input to Comparator C1 or C2

Note 1: P1C is available on PIC16F882/883/886 only.

3.4.4.3 RB2/AN8/P1B(1)

Figure 3-9 shows the diagram for this pin. This pin is configurable to function as one of the following:

- a general purpose I/O
- an analog input for the ADC
- a PWM output(1)

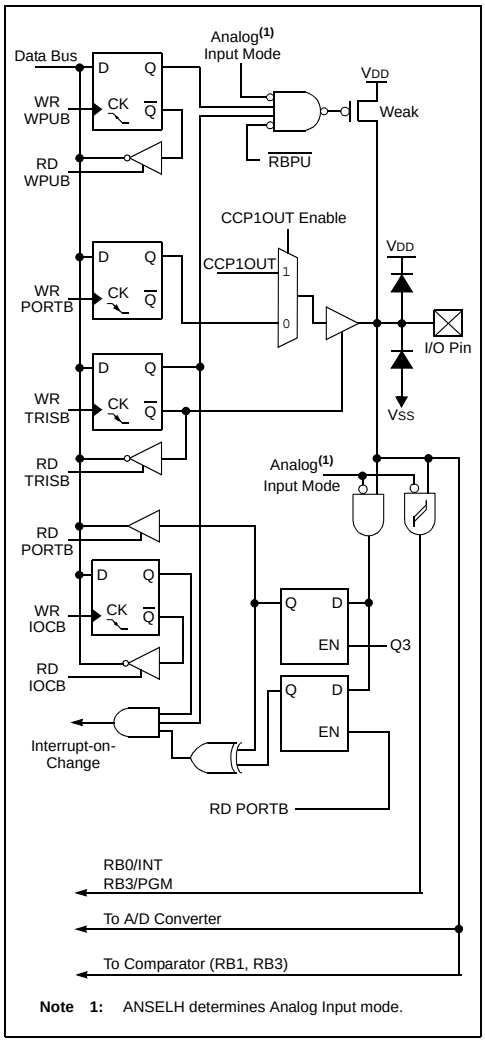
Note 1: P1B is available on PIC16F882/883/886 only.

3.4.4.4 RB3/AN9/PGM/C12IN2-

Figure 3-9 shows the diagram for this pin. This pin is configurable to function as one of the following:

- a general purpose I/O
- an analog input for the ADC
- Low-voltage In-Circuit Serial Programming enable pin
- an analog input to Comparator C1 or C2

FIGURE 3-9: BLOCK DIAGRAM OF RB<3:0>



Note 1: ANSELH determines Analog Input mode.

3.4.4.5 RB4/AN11/P1D(1)

Figure 3-10 shows the diagram for this pin. This pin is configurable to function as one of the following:

- a general purpose I/O
- an analog input for the ADC
- a PWM output(1)

Note 1: P1D is available on PIC16F882/883/886 only.

3.4.4.6 RB5/AN13/$\overline{T1G}$

Figure 3-10 shows the diagram for this pin. This pin is configurable to function as one of the following:

- a general purpose I/O
- an analog input for the ADC
- a Timer1 gate input

3.4.4.7 RB6/ICSPCLK

Figure 3-10 shows the diagram for this pin. This pin is configurable to function as one of the following:

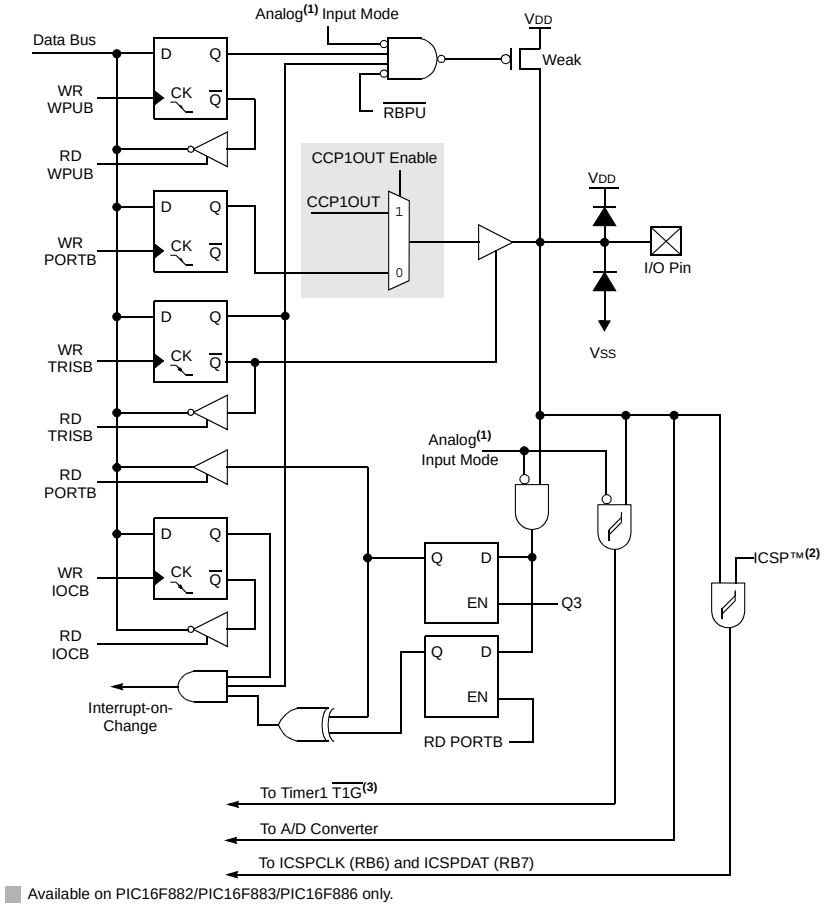
- a general purpose I/O
- In-Circuit Serial Programming clock

3.4.4.8 RB7/ICSPDAT

Figure 3-10 shows the diagram for this pin. This pin is configurable to function as one of the following:

- a general purpose I/O
- In-Circuit Serial Programming data

FIGURE 3-10: BLOCK DIAGRAM OF RB<7:4>



Note 1: ANSELH determines Analog Input mode.
2: Applies to RB<7:6> pins only).
3: Applies to RB5 pin only.

TABLE 3-2: SUMMARY OF REGISTERS ASSOCIATED WITH PORTB

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
ANSELH	—	—	ANS13	ANS12	ANS11	ANS10	ANS9	ANS8	--11 1111	--11 1111
CCP1CON	P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	0000 0000	0000 0000
CM2CON1	MC1OUT	MC2OUT	C1RSEL	C2RSEL	—	—	T1GSS	C2SYNC	0000 --10	0000 --10
IOCB	IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0	0000 0000	0000 0000
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000x
OPTION_REG	$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	1111 1111
WPUB	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged, – = unimplemented read as '0'. Shaded cells are not used by PORTB.

3.6 PORTD and TRISD Registers

PORTD(1) is a 8-bit wide, bidirectional port. The corresponding data direction register is TRISD (Register 3-12). Setting a TRISD bit (= 1) will make the corresponding PORTD pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISD bit (= 0) will make the corresponding PORTD pin an output (i.e., enable the output driver and put the contents of the output latch on the selected pin). Example 3-5 shows how to initialize PORTD.

Reading the PORTD register (Register 3-11) reads the status of the pins, whereas writing to it will write to the PORT latch. All write operations are read-modify-write operations. Therefore, a write to a port implies that the port pins are read, this value is modified and then written to the PORT data latch.

Note 1: PORTD is available on PIC16F884/887 only.

The TRISD register (Register 3-12) controls the PORTD pin output drivers, even when they are being used as analog inputs. The user should ensure the bits in the TRISD register are maintained set when using them as analog inputs. I/O pins configured as analog input always read '0'.

EXAMPLE 3-5: INITIALIZING PORTD

```
BANKSEL PORTD          ;
CLRF    PORTD          ;Init PORTD
BANKSEL TRISD          ;
MOVLW   B'00001100'    ;Set RD<3:2> as inputs
MOVWF   TRISD          ;and set RD<7:4,1:0>
                       ;as outputs
```

REGISTER 3-11: PORTD: PORTD REGISTER

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-0 **RD<7:0>**: PORTD General Purpose I/O Pin bit
1 = Port pin is > V_{IH}
0 = Port pin is < V_{IL}

REGISTER 3-12: TRISD: PORTD TRI-STATE REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

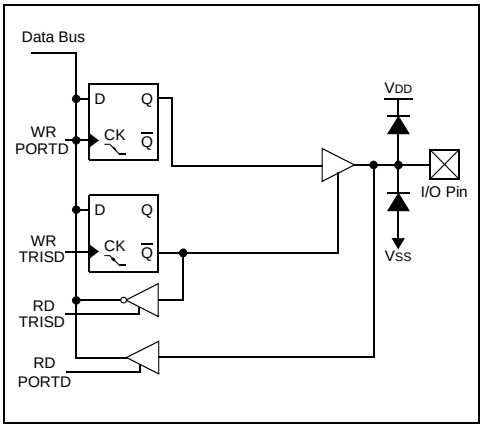
bit 7-0 **TRISD<7:0>:** PORTD Tri-State Control bit
1 = PORTD pin configured as an input (tri-stated)
0 = PORTD pin configured as an output

3.6.1 RD<4:0>

Figure 3-19 shows the diagram for these pins. These pins are configured to function as general purpose I/O's.

Note:	RD<4:0> is available on PIC16F884/887 only.

FIGURE 3-19: BLOCK DIAGRAM OF RD<4:0>



3.6.2 RD5/P1B(1)

Figure 3-20 shows the diagram for this pin. This pin is configurable to function as one of the following:

- a general purpose I/O
- a PWM output

Note 1:	RD5/P1B is available on PIC16F884/887 only. See RB2/AN8/P1B for this function on PIC16F882/883/886.

3.6.3 RD6/P1C(1)

Figure 3-20 shows the diagram for this pin. This pin is configurable to function as one of the following:

- a general purpose I/O
- a PWM output

Note 1:	RD6/P1C is available on PIC16F884/887 only. See RB1/AN10/P1C/C12IN3- for this function on PIC16F882/883/886.

3.6.4 RD7/P1D(1)

Figure 3-20 shows the diagram for this pin. This pin is configurable to function as one of the following:

- a general purpose I/O
- a PWM output

Note 1:	RD7/P1D is available on PIC16F884/887 only. See RB4/AN11/P1D for this function on PIC16F882/883/886.

FIGURE 3-20: BLOCK DIAGRAM OF RD<7:5>

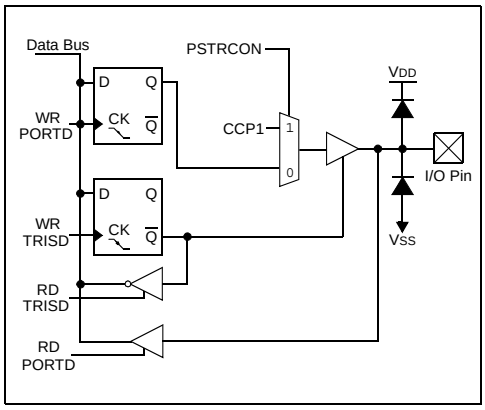


TABLE 3-4: SUMMARY OF REGISTERS ASSOCIATED WITH PORTD

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
PSTRCON	—	—	—	STRSYNC	STRD	STRC	STRB	STRA	---0 0001	---0 0001
TRISD	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged, – = unimplemented locations read as '0'. Shaded cells are not used by PORTD.

6.0 TIMER1 MODULE WITH GATE CONTROL

The Timer1 module is a 16-bit timer/counter with the following features:

- 16-bit timer/counter register pair (TMR1H:TMR1L)
- Programmable internal or external clock source
- 3-bit prescaler
- Optional LP oscillator
- Synchronous or asynchronous operation
- Timer1 gate (count enable) via comparator or $\overline{T1G}$ pin
- Interrupt on overflow
- Wake-up on overflow (external clock, Asynchronous mode only)
- Time base for the Capture/Compare function
- Special Event Trigger (with ECCP)
- Comparator output synchronization to Timer1 clock

Figure 6-1 is a block diagram of the Timer1 module.

6.1 Timer1 Operation

The Timer1 module is a 16-bit incrementing counter which is accessed through the TMR1H:TMR1L register pair. Writes to TMR1H or TMR1L directly update the counter.

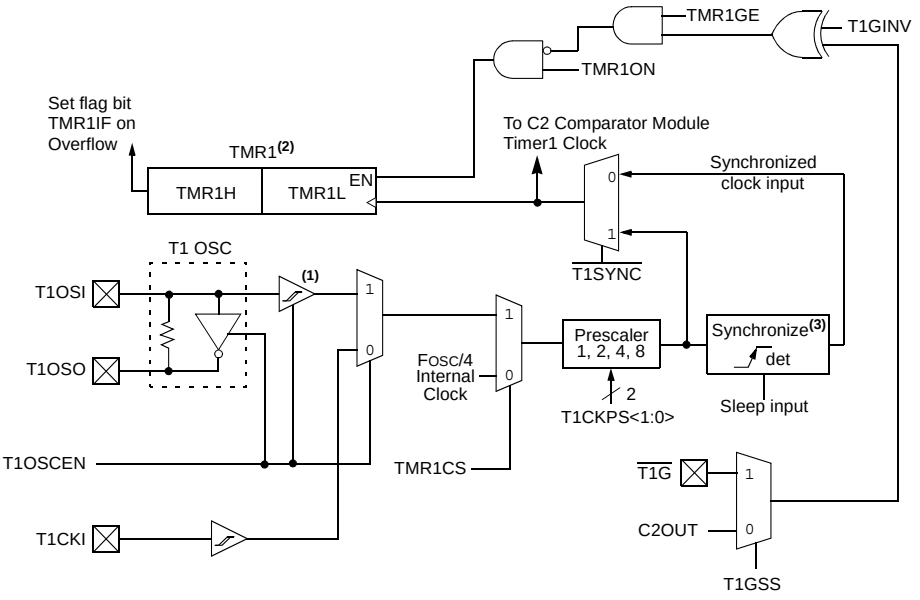
When used with an internal clock source, the module is a timer. When used with an external clock source, the module can be used as either a timer or counter.

6.2 Clock Source Selection

The TMR1CS bit of the T1CON register is used to select the clock source. When TMR1CS = 0, the clock source is FOSC/4. When TMR1CS = 1, the clock source is supplied externally.

Clock Source	TMR1CS
FOSC/4	0
T1CKI pin	1

FIGURE 6-1: TIMER1 BLOCK DIAGRAM



Note 1: ST Buffer is low power type when using LP oscillator, or high speed type when using T1CKI.
2: Timer1 register increments on rising edge.
3: Synchronize does not operate while in Sleep.

6.2.1 INTERNAL CLOCK SOURCE

When the internal clock source is selected the TMR1H:TMR1L register pair will increment on multiples of FOSC as determined by the Timer1 prescaler.

6.2.2 EXTERNAL CLOCK SOURCE

When the external clock source is selected, the Timer1 module may work as a timer or a counter.

When counting, Timer1 is incremented on the rising edge of the external clock input T1CKI. In addition, the Counter mode clock can be synchronized to the microcontroller system clock or run asynchronously.

If an external clock oscillator is needed (and the microcontroller is using the INTOSC without CLKOUT), Timer1 can use the LP oscillator as a clock source.

Note:	In Counter mode, a falling edge must be registered by the counter prior to the first incrementing rising edge.

6.3 Timer1 Prescaler

Timer1 has four prescaler options allowing 1, 2, 4 or 8 divisions of the clock input. The T1CKPS bits of the T1CON register control the prescale counter. The prescale counter is not directly readable or writable; however, the prescaler counter is cleared upon a write to TMR1H or TMR1L.

6.4 Timer1 Oscillator

A low-power 32.768 kHz crystal oscillator is built-in between pins T1OSI (input) and T1OSO (amplifier output). The oscillator is enabled by setting the T1OSCEN control bit of the T1CON register. The oscillator will continue to run during Sleep.

The Timer1 oscillator is identical to the LP oscillator. The user must provide a software time delay to ensure proper oscillator start-up.

TRISC0 and TRISC1 bits are set when the Timer1 oscillator is enabled. RC0 and RC1 bits read as '0' and TRISC0 and TRISC1 bits read as '1'.

Note:	The oscillator requires a start-up and stabilization time before use. Thus, T1OSCEN should be set and a suitable delay observed prior to enabling Timer1.

6.5 Timer1 Operation in Asynchronous Counter Mode

If control bit $\overline{\text{T1SYNC}}$ of the T1CON register is set, the external clock input is not synchronized. The timer continues to increment asynchronous to the internal phase clocks. The timer will continue to run during Sleep and can generate an interrupt on overflow, which will wake-up the processor. However, special precautions in software are needed to read/write the timer (see **Section 6.5.1 "Reading and Writing Timer1 in Asynchronous Counter Mode"**).

Note:	When switching from synchronous to asynchronous operation, it is possible to skip an increment. When switching from asynchronous to synchronous operation, it is possible to produce a single spurious increment.

6.5.1 READING AND WRITING TIMER1 IN ASYNCHRONOUS COUNTER MODE

Reading TMR1H or TMR1L while the timer is running from an external asynchronous clock will ensure a valid read (taken care of in hardware). However, the user should keep in mind that reading the 16-bit timer in two 8-bit values itself, poses certain problems, since the timer may overflow between the reads.

For writes, it is recommended that the user simply stop the timer and write the desired values. A write contention may occur by writing to the timer registers, while the register is incrementing. This may produce an unpredictable value in the TMR1H:TTMR1L register pair.

6.6 Timer1 Gate

Timer1 gate source is software configurable to be the $\overline{\text{T1G}}$ pin or the output of Comparator C2. This allows the device to directly time external events using $\overline{\text{T1G}}$ or analog events using Comparator C2. See the CM2CON1 register (Register 8-3) for selecting the Timer1 gate source. This feature can simplify the software for a Delta-Sigma A/D converter and many other applications. For more information on Delta-Sigma A/D converters, see the Microchip web site (www.microchip.com).

Note:	TMR1GE bit of the T1CON register must be set to use either $\overline{\text{T1G}}$ or C2OUT as the Timer1 gate source. See Register 8-3 for more information on selecting the Timer1 gate source.

Timer1 gate can be inverted using the T1GINV bit of the T1CON register, whether it originates from the $\overline{\text{T1G}}$ pin or Comparator C2 output. This configures Timer1 to measure either the active-high or active-low time between events.

6.7 Timer1 Interrupt

The Timer1 register pair (TMR1H:TMR1L) increments to FFFFh and rolls over to 0000h. When Timer1 rolls over, the Timer1 interrupt flag bit of the PIR1 register is set. To enable the interrupt on rollover, you must set these bits:

- Timer1 interrupt enable bit of the PIE1 register
- PEIE bit of the INTCON register
- GIE bit of the INTCON register

The interrupt is cleared by clearing the TMR1IF bit in the Interrupt Service Routine.

Note:	The TMR1H:TTMR1L register pair and the TMR1IF bit should be cleared before enabling interrupts.

6.8 Timer1 Operation During Sleep

Timer1 can only operate during Sleep when setup in Asynchronous Counter mode. In this mode, an external crystal or clock source can be used to increment the counter. To set up the timer to wake the device:

- TMR1ON bit of the T1CON register must be set
- TMR1IE bit of the PIE1 register must be set
- PEIE bit of the INTCON register must be set

The device will wake-up on an overflow and execute the next instruction. If the GIE bit of the INTCON register is set, the device will call the Interrupt Service Routine (0004h).

6.9 ECCP Capture/Compare Time Base

The ECCP module uses the TMR1H:TMR1L register pair as the time base when operating in Capture or Compare mode.

In Capture mode, the value in the TMR1H:TMR1L register pair is copied into the CCPRxH:CCPRxL register pair on a configured event.

In Compare mode, an event is triggered when the value CCPRxH:CCPRxL register pair matches the value in the TMR1H:TMR1L register pair. This event can be a Special Event Trigger.

See **Section 11.0 "Capture/Compare/PWM Modules (CCP1 and CCP2)"** for more information.

6.10 ECCP Special Event Trigger

If an ECCP is configured to trigger a special event, the trigger will clear the TMR1H:TMR1L register pair. This special event does not cause a Timer1 interrupt. The ECCP module may still be configured to generate a ECCP interrupt.

In this mode of operation, the CCPRxH:CCPRxL register pair effectively becomes the period register for Timer1.

Timer1 should be synchronized to the FOSC to utilize the Special Event Trigger. Asynchronous operation of Timer1 can cause a Special Event Trigger to be missed.

In the event that a write to TMR1H or TMR1L coincides with a Special Event Trigger from the ECCP, the write will take precedence.

For more information, see **Section 11.0 "Capture/Compare/PWM Modules (CCP1 and CCP2)"**.

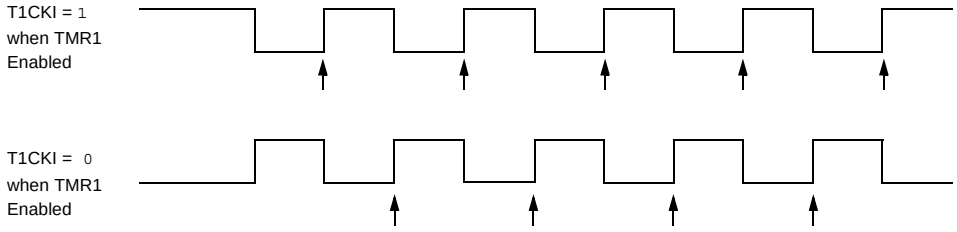
6.11 Comparator Synchronization

The same clock used to increment Timer1 can also be used to synchronize the comparator output. This feature is enabled in the Comparator module.

When using the comparator for Timer1 gate, the comparator output should be synchronized to Timer1. This ensures Timer1 does not miss an increment if the comparator changes.

For more information, see **Section 8.0 "Comparator Module"**.

FIGURE 6-2: TIMER1 INCREMENTING EDGE



Note 1: Arrows indicate counter increments.

2: In Counter mode, a falling edge must be registered by the counter prior to the first incrementing rising edge of the clock.

6.12 Timer1 Control Register

The Timer1 Control register (T1CON), shown in Register 6-1, is used to control Timer1 and select the various features of the Timer1 module.

REGISTER 6-1: T1CON: TIMER1 CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
T1GINV(1)	TMR1GE(2)	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **T1GINV:** Timer1 Gate Invert bit(1)
1 = Timer1 gate is active-high (Timer1 counts when gate is high)
0 = Timer1 gate is active-low (Timer1 counts when gate is low)

bit 6 **TMR1GE:** Timer1 Gate Enable bit(2)
If TMR1ON = 0:
This bit is ignored
If TMR1ON = 1:
1 = Timer1 is on if Timer1 gate is not active
0 = Timer1 is on

bit 5-4 **T1CKPS<1:0>:** Timer1 Input Clock Prescale Select bits
11 = 1:8 Prescale Value
10 = 1:4 Prescale Value
01 = 1:2 Prescale Value
00 = 1:1 Prescale Value

bit 3 **T1OSCEN:** LP Oscillator Enable Control bit
1 = LP oscillator is enabled for Timer1 clock
0 = LP oscillator is off

bit 2 **$\overline{\text{T1SYNC}}$:** Timer1 External Clock Input Synchronization Control bit
TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock

bit 1 **TMR1CS:** Timer1 Clock Source Select bit
1 = External clock from T1CKI pin (on the rising edge)
0 = Internal clock (FOSC/4)

bit 0 **TMR1ON:** Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

Note 1: T1GINV bit inverts the Timer1 gate logic, regardless of source.
2: TMR1GE bit must be set to use either $\overline{\text{T1G}}$ pin or C2OUT, as selected by the T1GSS bit of the CM2CON1 register, as a Timer1 gate source.

TABLE 6-1: SUMMARY OF REGISTERS ASSOCIATED WITH TIMER1

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
CM2CON1	MC1OUT	MC2OUT	C1RSEL	C2RSEL	—	—	T1GSS	C2SYNC	0000 --10	0000 --10
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000x
PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	-000 0000	-000 0000
PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	-000 0000	-000 0000
TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
T1CON	T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	0000 0000	uuuu uuuu

Legend: x = unknown, u = unchanged, – = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.

14.3 Interrupts

The PIC16F882/883/884/886/887 has multiple interrupt sources:

- External Interrupt RB0/INT
- Timer0 Overflow Interrupt
- PORTB Change Interrupts
- 2 Comparator Interrupts
- A/D Interrupt
- Timer1 Overflow Interrupt
- Timer2 Match Interrupt
- EEPROM Data Write Interrupt
- Fail-Safe Clock Monitor Interrupt
- Enhanced CCP Interrupt
- EUSART Receive and Transmit Interrupts
- Ultra Low-Power Wake-up Interrupt
- MSSP Interrupt

The Interrupt Control register (INTCON) and Peripheral Interrupt Request Register 1 (PIR1) record individual interrupt requests in flag bits. The INTCON register also has individual and global interrupt enable bits.

A Global Interrupt Enable bit, GIE (INTCON<7>), enables (if set) all unmasked interrupts, or disables (if cleared) all interrupts. Individual interrupts can be disabled through their corresponding enable bits in the INTCON, PIE1 and PIE2 registers, respectively. GIE is cleared on Reset.

The Return from Interrupt instruction, `RETFIE`, exits the interrupt routine, as well as sets the GIE bit, which re-enables unmasked interrupts.

The following interrupt flags are contained in the INTCON register:

- INT Pin Interrupt
- PORTB Change Interrupts
- Timer0 Overflow Interrupt

The peripheral interrupt flags are contained in the PIR1 and PIR2 registers. The corresponding interrupt enable bits are contained in PIE1 and PIE2 registers.

The following interrupt flags are contained in the PIR1 register:

- A/D Interrupt
- EUSART Receive and Transmit Interrupts
- Timer1 Overflow Interrupt
- Synchronous Serial Port (SSP) Interrupt
- Enhanced CCP1 Interrupt
- Timer1 Overflow Interrupt
- Timer2 Match Interrupt

The following interrupt flags are contained in the PIR2 register:

- Fail-Safe Clock Monitor Interrupt
- 2 Comparator Interrupts
- EEPROM Data Write Interrupt
- Ultra Low-Power Wake-up Interrupt
- CCP2 Interrupt

When an interrupt is serviced:

- The GIE is cleared to disable any further interrupt.
- The return address is pushed onto the stack.
- The PC is loaded with 0004h.

For external interrupt events, such as the INT pin, PORTB change interrupts, the interrupt latency will be three or four instruction cycles. The exact latency depends upon when the interrupt event occurs (see Figure 14-8). The latency is the same for one or two-cycle instructions. Once in the Interrupt Service Routine, the source(s) of the interrupt can be determined by polling the interrupt flag bits. The interrupt flag bit(s) must be cleared in software before re-enabling interrupts to avoid multiple interrupt requests.

Note 1:	Individual interrupt flag bits are set, regardless of the status of their corresponding mask bit or the GIE bit.
2:	When an instruction that clears the GIE bit is executed, any interrupts that were pending for execution in the next cycle are ignored. The interrupts, which were ignored, are still pending to be serviced when the GIE bit is set again.

For additional information on Timer1, Timer2, comparators, A/D, data EEPROM, EUSART, MSSP or Enhanced CCP modules, refer to the respective peripheral section.

14.3.1 RB0/INT INTERRUPT

External interrupt on RB0/INT pin is edge-triggered; either rising if the INTEDG bit (OPTION_REG<6>) is set, or falling, if the INTEDG bit is clear. When a valid edge appears on the RB0/INT pin, the INTF bit (INTCON<1>) is set. This interrupt can be disabled by clearing the INTE control bit (INTCON<4>). The INTF bit must be cleared in software in the Interrupt Service Routine before re-enabling this interrupt. The RB0/INT interrupt can wake-up the processor from Sleep, if the INTE bit was set prior to going into Sleep. The status of the GIE bit decides whether or not the processor branches to the interrupt vector following wake-up (0004h). See **Section 14.6 “Power-Down Mode (Sleep)”** for details on Sleep and Figure 14-10 for timing of wake-up from Sleep through RB0/INT interrupt.

14.3.2 TIMER0 INTERRUPT

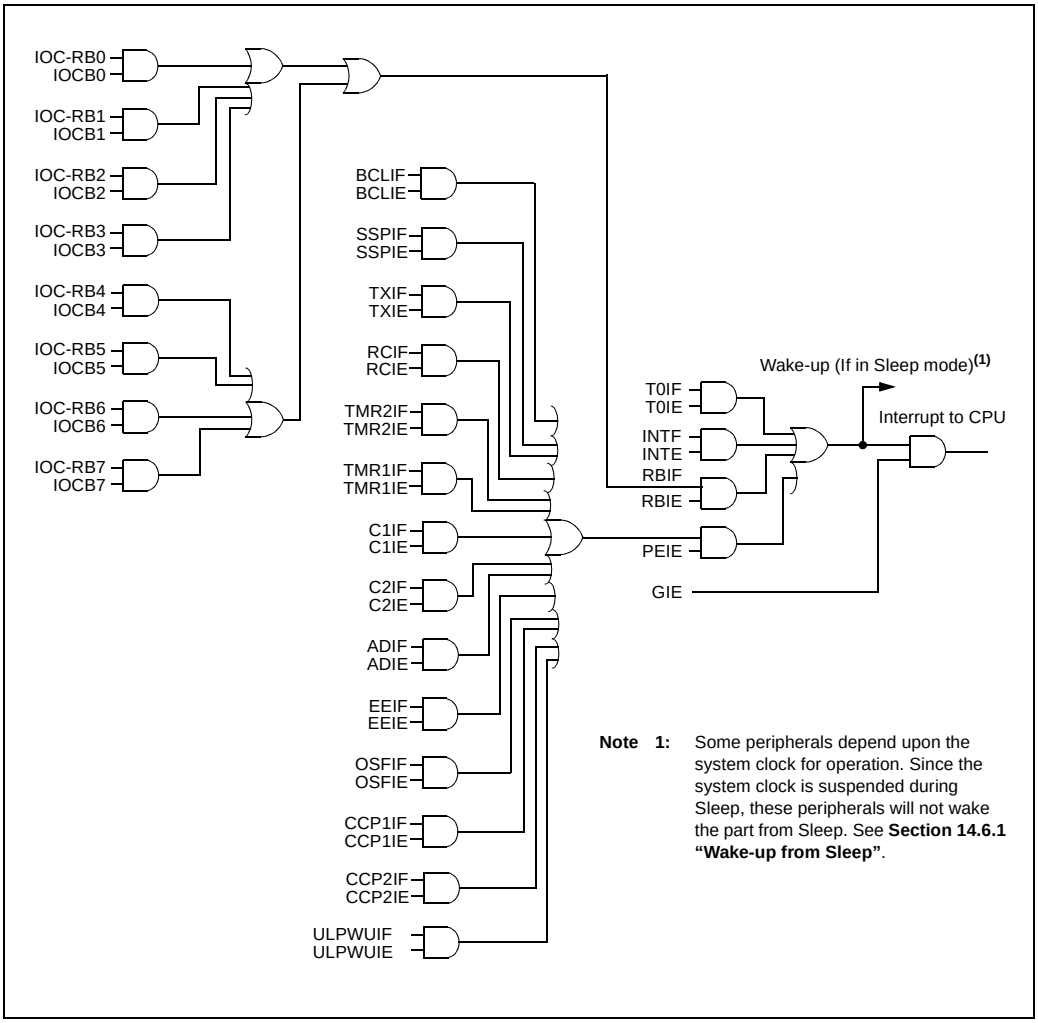
An overflow (FFh → 00h) in the TMR0 register will set the T0IF (INTCON<2>) bit. The interrupt can be enabled/disabled by setting/clearing T0IE (INTCON<5>) bit. See **Section 5.0 "Timer0 Module"** for operation of the Timer0 module.

14.3.3 PORTB INTERRUPT

An input change on PORTB change sets the RBIF (INTCON<0>) bit. The interrupt can be enabled/disabled by setting/clearing the RBIE (INTCON<3>) bit. Plus, individual pins can be configured through the IOCB register.

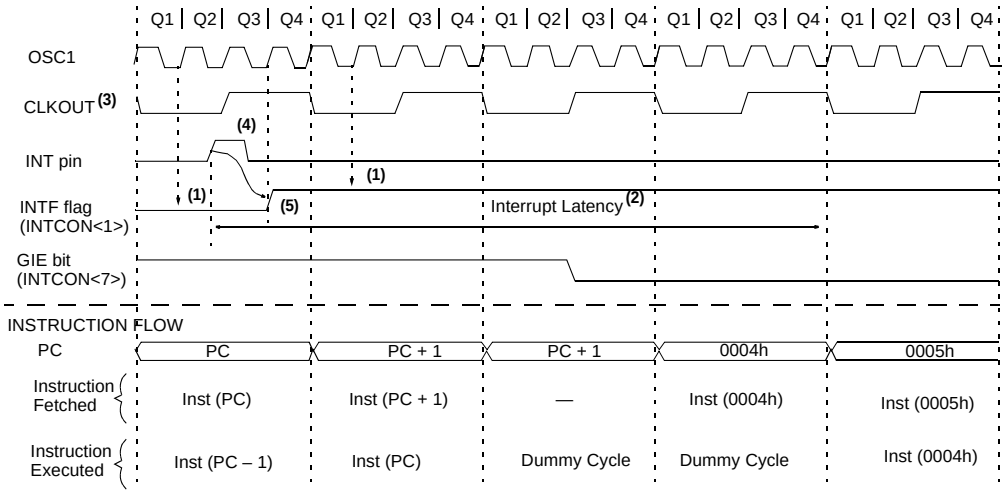
Note:	If a change on the I/O pin should occur when the read operation is being executed (start of the Q2 cycle), then the RBIF interrupt flag may not get set. See **Section 3.4.3 "Interrupt-on-Change"** for more information.

FIGURE 14-7: INTERRUPT LOGIC



Note 1: Some peripherals depend upon the system clock for operation. Since the system clock is suspended during Sleep, these peripherals will not wake the part from Sleep. See **Section 14.6.1 "Wake-up from Sleep"**.

FIGURE 14-8: INT PIN INTERRUPT TIMING



Note 1: INTF flag is sampled here (every Q1).

2: Asynchronous interrupt latency = 3-4 TCY. Synchronous latency = 3 TCY, where TCY = instruction cycle time. Latency is the same whether Inst (PC) is a single cycle or a 2-cycle instruction.

3: CLKOUT is available only in INTOSC and RC Oscillator modes.

4: For minimum width of INT pulse, refer to AC specifications in **Section 17.0 "Electrical Specifications"**.

5: INTF is enabled to be set any time during the Q4-Q1 cycles.

TABLE 14-6: SUMMARY OF INTERRUPT REGISTERS

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000x
PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	-000 0000	-000 0000
PIE2	OSFIE	C2IE	C1IE	EEIE	BCLIE	ULPWUIE	—	CCP2IE	0000 00-0	0000 00-0
PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	-000 0000	-000 0000
PIR2	OSFIF	C2IF	C1IF	EEIF	BCLIF	ULPWUIF	—	CCP2IF	0000 00-0	0000 00-0

Legend: x = unknown, u = unchanged, – = unimplemented read as '0', q = value depends upon condition.
Shaded cells are not used by the Interrupt module.

14.4 Context Saving During Interrupts

During an interrupt, only the return PC value is saved on the stack. Typically, users may wish to save key registers during an interrupt (e.g., W and STATUS registers). This must be implemented in software.

Since the upper 16 bytes of all GPR banks are common in the PIC16F882/883/884/886/887 (see Figures 2-2 and 2-3), temporary holding registers, W_TEMP and STATUS_TEMP, should be placed in here. These 16 locations do not require banking and therefore, make it easier to context save and restore. The same code shown in Example 14-1 can be used to:

- Store the W register
- Store the STATUS register
- Execute the ISR code
- Restore the Status (and Bank Select Bit register)
- Restore the W register

Note:	The PIC16F882/883/884/886/887 normally does not require saving the PCLATH. However, if computed GOTO's are used in the ISR and the main code, the PCLATH must be saved and restored in the ISR.

EXAMPLE 14-1: SAVING STATUS AND W REGISTERS IN RAM

```
MOVWF  W_TEMP           ;Copy W to TEMP register
SWAPF  STATUS,W         ;Swap status to be saved into W
                        ;Swaps are used because they do not affect the status bits
MOVWF  STATUS_TEMP      ;Save status to bank zero STATUS_TEMP register
:
:(ISR)                  ;Insert user code here
:
SWAPF  STATUS_TEMP,W    ;Swap STATUS_TEMP register into W
                        ;(sets bank to original state)
MOVWF  STATUS           ;Move W into STATUS register
SWAPF  W_TEMP,F         ;Swap W_TEMP
SWAPF  W_TEMP,W         ;Swap W_TEMP into W
```

TABLE 15-2: PIC16F883/884/886/887 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C, DC, Z	**1, 2**
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	**1, 2**
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	**2**
CLRW	–	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	**1, 2**
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	**1, 2**
DECFSZ	f, d	Decrement f, Skip if 0	1**(2)**	00	1011	dfff	ffff		**1, 2, 3**
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	**1, 2**
INCFSZ	f, d	Increment f, Skip if 0	1**(2)**	00	1111	dfff	ffff		**1, 2, 3**
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	**1, 2**
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	**1, 2**
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	–	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	**1, 2**
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	**1, 2**
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C, DC, Z	**1, 2**
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		**1, 2**
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	**1, 2**
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		**1, 2**
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		**1, 2**
BTFSC	f, b	Bit Test f, Skip if Clear	1 **(2)**	01	10bb	bfff	ffff		**3**
BTFSS	f, b	Bit Test f, Skip if Set	1 **(2)**	01	11bb	bfff	ffff		**3**
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C, DC, Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call Subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	–	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$, $\overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	–	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	–	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	–	Go into Standby mode	1	00	0000	0110	0011	$\overline{TO}$, $\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C, DC, Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., MOVF GPIO, 1), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If the Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

