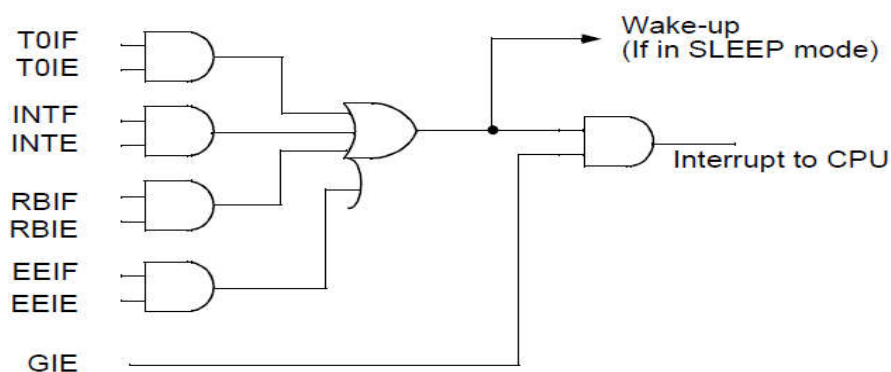


LAB 6

Le but du LAB : comment gérer plusieurs demandes interruptions générées par les périphériques de notre μC le PIC16F84A.

Un petit rappel sur les sources d'interruptions de PIC16F84A : notre μC gère 4 sources d'interruptions, dont, deux externes (changement d'état d'une des pattes RB4 :RB7 et la détection d'un franc sur la patte RB0/INT), et deux internes (fin d'écriture dans l'EEPROM et débordement de registre TMR0 [passage de la valeur 255 vers 0]).

Toutes ces interruptions sont conditionnés par des bits d'activation {RBIE, INTE, EEIE, TOIE} et indiqués par des bits flags {RBIF, INTF, EEIF, TOIF} avec l'ensemble masqué ou démasqué par le bit GIE comme indique la figure suivante :



Comme on a dit dans le cours ; le μ P MID-RANGE réserve le vecteur d'adresse 0x0004 pour gérer toute demande d'interruption, dont la priorité sera fixée par le cahier des charges.

La méthode de gestion des interruptions sous MID-RANGE prend le nom SOFTWARE POLLING (par scrutation). Cette dernière fait le test du flag le plus prioritaire jusqu'au dernier flag le moins prioritaire ; si le cas, on exécute le traitement correspondant.

Selon la nature du programme interrompu, le code de traitement d'interruption (ISR) fait deux actions de grand intérêt, le SAVE-CONTEXT et le LOAD-CONTEXT, dont le rôle de ces actions est la conservation de bon fonctionnement du programme interrompu après retour vers lui. Si le code interrompu est un plantage (boucle sur la même adresse) le SAVE/LOAD-CONTEXT devient inutile et dans le cas contraire il faut ajouter à l'ISR comme on a vu dans le cours.

=====

Maintenant, on revient à notre LAB :

Il demande de gérer deux interruptions, une externe par RB0/INT et l'autre interne par TIMERO, dont l'interruption de la plus haute priorité est RB0/INT.

Au moment de présence d'un **niveau bas** sur la patte RB0, la logique interne de la patte RB0/INT détecte la **présence d'un franc descendant** et déclenche un traitement fait l'inversion de l'état d'une LED relié à la patte RA0.

Et pour l'autre source, c'est de compter des impulsions par le module TIMERO à travers la patte RA4 sur franc descendant; si le nombre d'impulsions arrive à 10, la logique interne de module génère une demande d'interruption, dont le traitement de cette interruption fait l'inversion de l'état d'une LED relié à la patte RB4.

D'après le LAB5, on a pris comment compter un certain nombre d'impulsions par le module TIMERO afin de positionner son flag T0IF, donc générer une demande d'interruption.

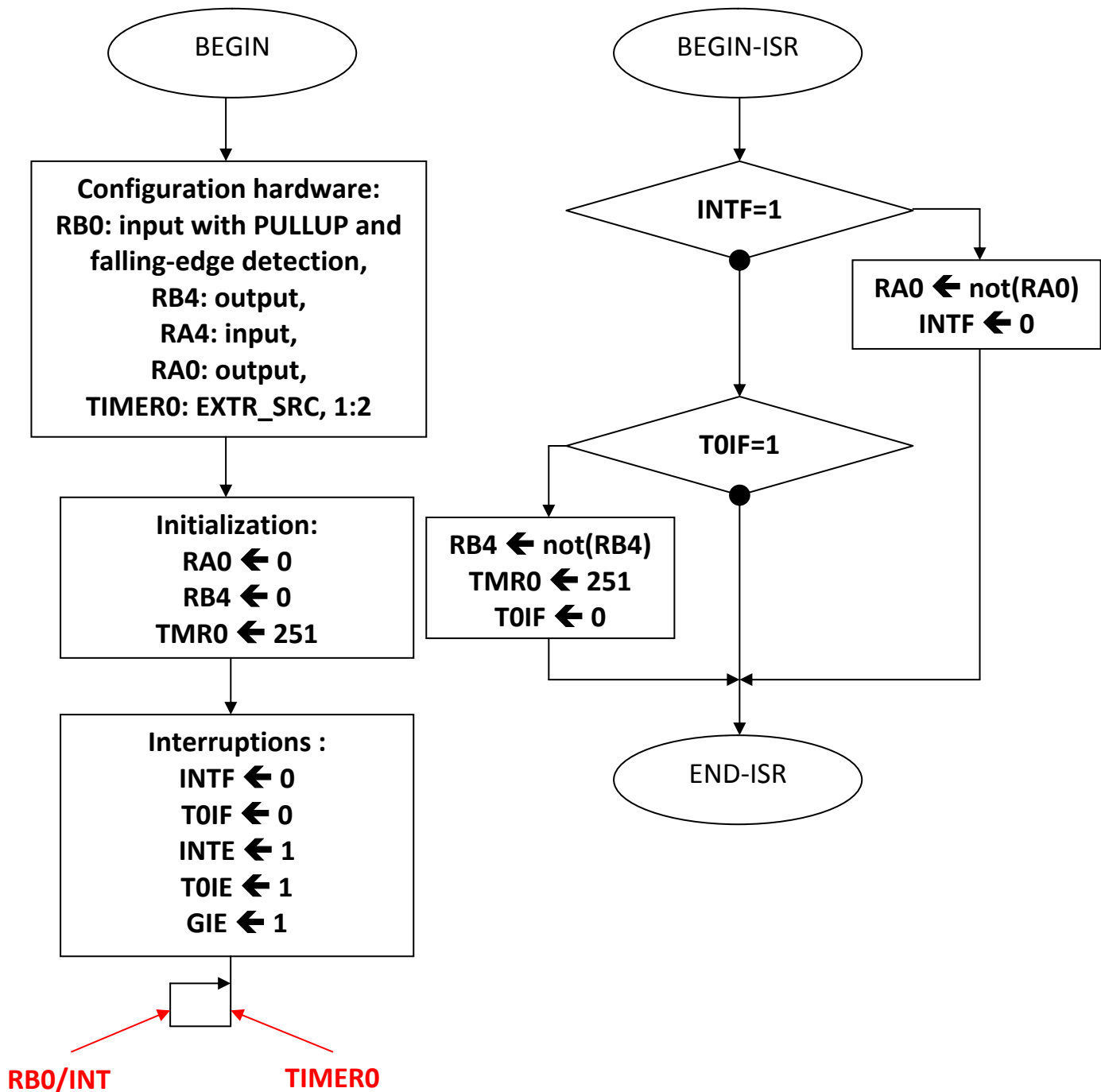
$$N_{\text{impulsion}} = (256 - \text{TMRO_INIT}) * \text{PRESCALER} \rightarrow$$
$$\text{TMRO_INIT} = 256 - (N / \text{PRESCALER}).$$

D'après le nombre d'impulsions à compter, on peut utiliser soit un pré-diviseur 1 :1 ou 1 :2

Si 1 :1 \rightarrow $\text{TMRO_INT} = 256 - 10 = 246.$

Si 1 :2 \rightarrow $\text{TMRO_INT} = 256 - 5 = 251.$

Organigramme:



N.B: d'après l'organigramme du programme principal, on distingue que ce n'est pas la peine d'ajouter le **SAVE/LOAD-CONTEXT** par ce que la partie du code à interrompre est un plantage.