Formation Automatique et Informatique Industrielle

Master 1 S2

Matière : Systèmes Embarqués et Systèmes Temps Réel SE-STR

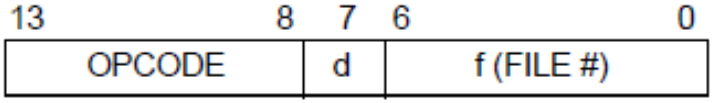
Par : ATOUI Hamza

Plan du cours –partie 2-

- Présentation du jeu d'instructions de MID-RANGE.
  1. **Formats des instructions.**
  2. **Table des instructions.**
  3. **Le registre d'état STATUS.**
  4. **L'arbre du HUFFMAN pour le jeu d'instructions.**
  5. **Le registre compteur ordinal PC.**
  6. **Groupes du jeu d'instructions.**
- Organisation de la zone CODE et la zone STACK.
- Gestion des interruptions.

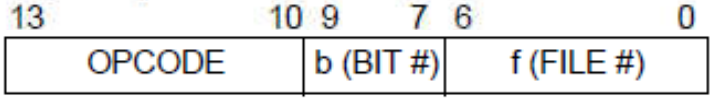
Formats des instructions

Byte-oriented file register operations



d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

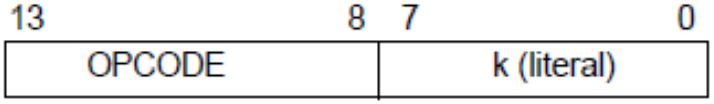
Bit-oriented file register operations



b = 3-bit bit address
f = 7-bit file register address

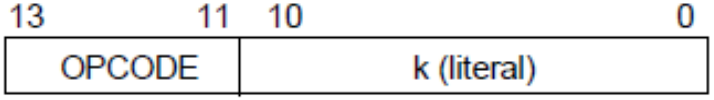
Literal and control operations

General



k = 8-bit literal (immediate) value

`CALL` and `GOTO` instructions only



k = 11-bit literal (immediate) value

Encore une autre limite!!!

Le MID-RANGE est un CPU RISC a 35 instructions.

La plus par des instructions fait un seul cycle machine.

Table des instructions

Mnemonic, Operands		Description	Cycles	14-Bit Instruction Word MSb			LSb	Status Affected
BYTE-ORIENTED FILE REGISTER OPERATIONS								
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff	
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff	
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z
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
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff	
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z
BIT-ORIENTED FILE REGISTER OPERATIONS								
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff	
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff	
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff	
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff	
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

STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

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	C
bit 7							bit 0

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 **RP1:RP0**: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h-1FFh)
10 = Bank 2 (100h-17Fh)
01 = Bank 1 (80h-FFh)
00 = Bank 0 (00h-7Fh)
Each bank is 128 bytes.

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

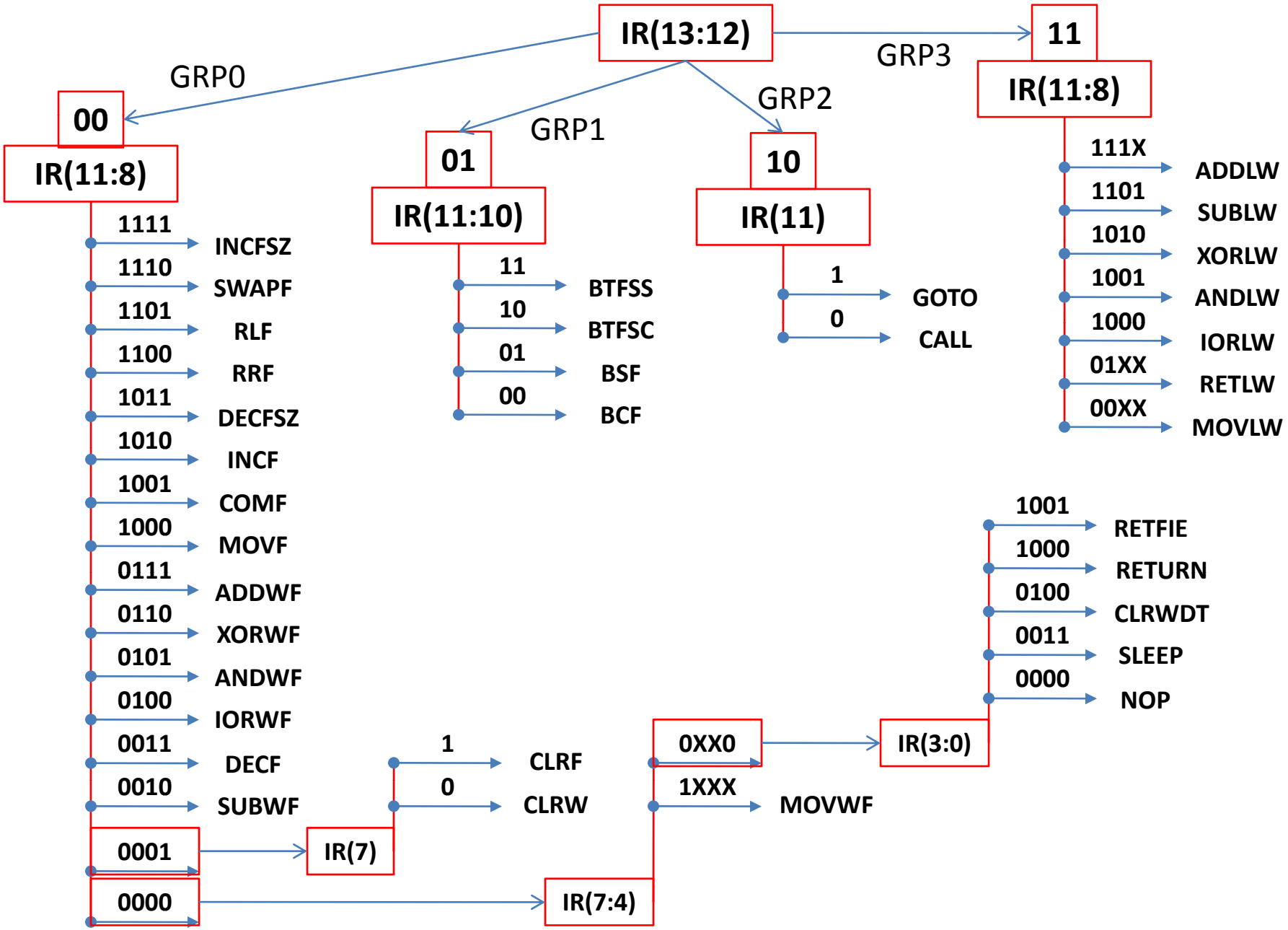
bit 2 **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit carry/$\overline{\text{borrow}}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
(for $\overline{\text{borrow}}$, the polarity is reversed)
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/$\overline{\text{borrow}}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
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: For $\overline{\text{borrow}}$, 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, or low order bit of the source register.

Arbre du HUFFMAN pour le jeu d'instruction du MID-RANGE



Désassemblage par l’arbre du HUFFMAN

- **Utiliser l’arbre du HUFFMAN et le tableau des instructions pour désassembler le code machine suivant d’un CPU MID-RANGE.**

ADDR	CODE-MACHINE HEX
0000	2805
0001	3FFF
0002	3FFF
0003	3FFF
0004	0009
0005	3050
0006	0084
0007	1383
0008	0180
0009	0F84
000A	2808
000B	280B

The Program Counter (PC) is 13 bits wide. The low byte comes from the PCL register which is a readable and writable register. The upper bits (PC<12:8>) are not readable, but are indirectly writable through the PCLATH register. On any Reset, the upper bits of the PC will be cleared. Figure 2-5 shows the two situations for the loading of the PC. The upper example in the figure shows how the PC is loaded on a write to PCL (PCLATH<4:0> → PCH). The lower example in the figure shows how the PC is loaded during a CALL or GOTO instruction (PCLATH<4:3> → PCH).

Le registre compteur ordinal PC

LOADING OF PC IN DIFFERENT SITUATIONS

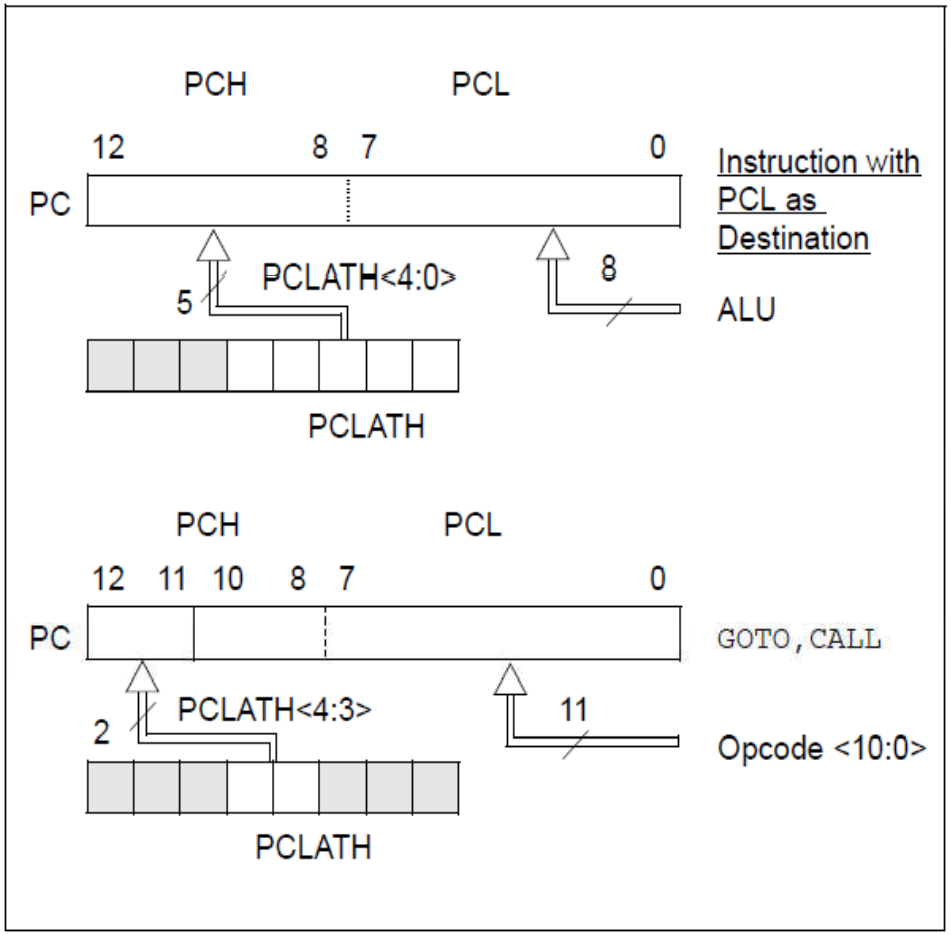


FIGURE 2-5

Le registre compteur ordinal PC

- Le registre PCL au moment de lecture:

Adresse	mnémonique
101h	MOVF PCL, W
102h	MOVWF X

- Quelle est la valeur chargée dans X ???
- Le registre PCL au moment d'écriture:

Adresse	mnémonique
100h	MOVLW 10h
101h	MOVWF PCLATH
102h	MOVLW 03h
103h	MOVWF PCL

- Qu'est ce qui se passe après l'exécution de ce code ?????

Groupes de jeu d'instructions

- D'après le tableau de diaporama -4-, le jeu l'instructions de MID-RANGE se divise en 3 groupes sont:
  - **Instructions orientées FILE.**
  - **Instructions orientées BIT.**
  - **Instructions de contrôle et manipulation des valeurs immédiates.**
- Mais, selon la fonction, on a 5 groupes sont:
  - **Instructions de transfert** : **MOVLW, MOVF et MOVWF.**
  - **Instructions orientées BIT** : **BCF, BSF, BTFSC et BTFSS.**
  - **Instructions arithmétiques** : **ADDLW, ADDWF, SUBLW, SUBWF, INCF, DECF, INCFSZ, DECFSZ, CLRF et CLRW.**
  - **Instructions logiques** : **ANDLW, ANDWF, IORLW, IORWF, XORLW, XORWF, COMF, RLF, RRF et SWAPF.**
  - **Instructions de Contrôle** : **GOTO, CALL, RETURN, RETLW, RETFIE, NOP, SLEEP et CLRWDT.**

Quizz

- Dans la littérature, on trouve d'autres instructions de décalage et de rotation:
- Décalage logique/arithmétique à gauche (fig1).
- Décalage logique à droite (fig2).
- Décalage arithmétique à droite (fig3).
- Rotation à gauche sans carry (fig4).
- Rotation à droite sans carry (fig5).
- **Réaliser les opérations suivantes par le jeu d'instructions du MID-RANGE.**

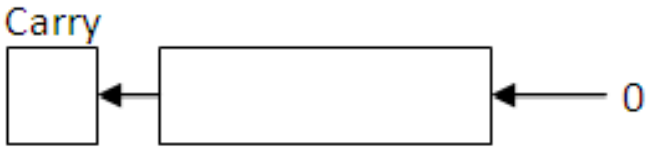


Fig1

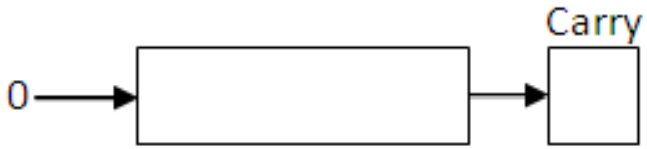


Fig2

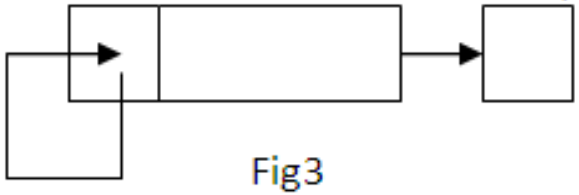


Fig3

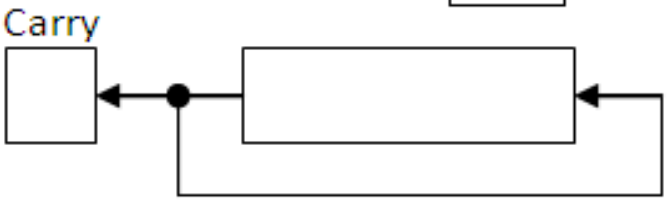


Fig4

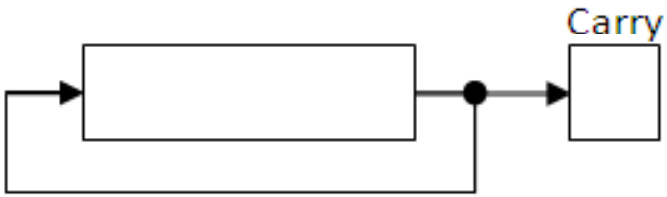


Fig5

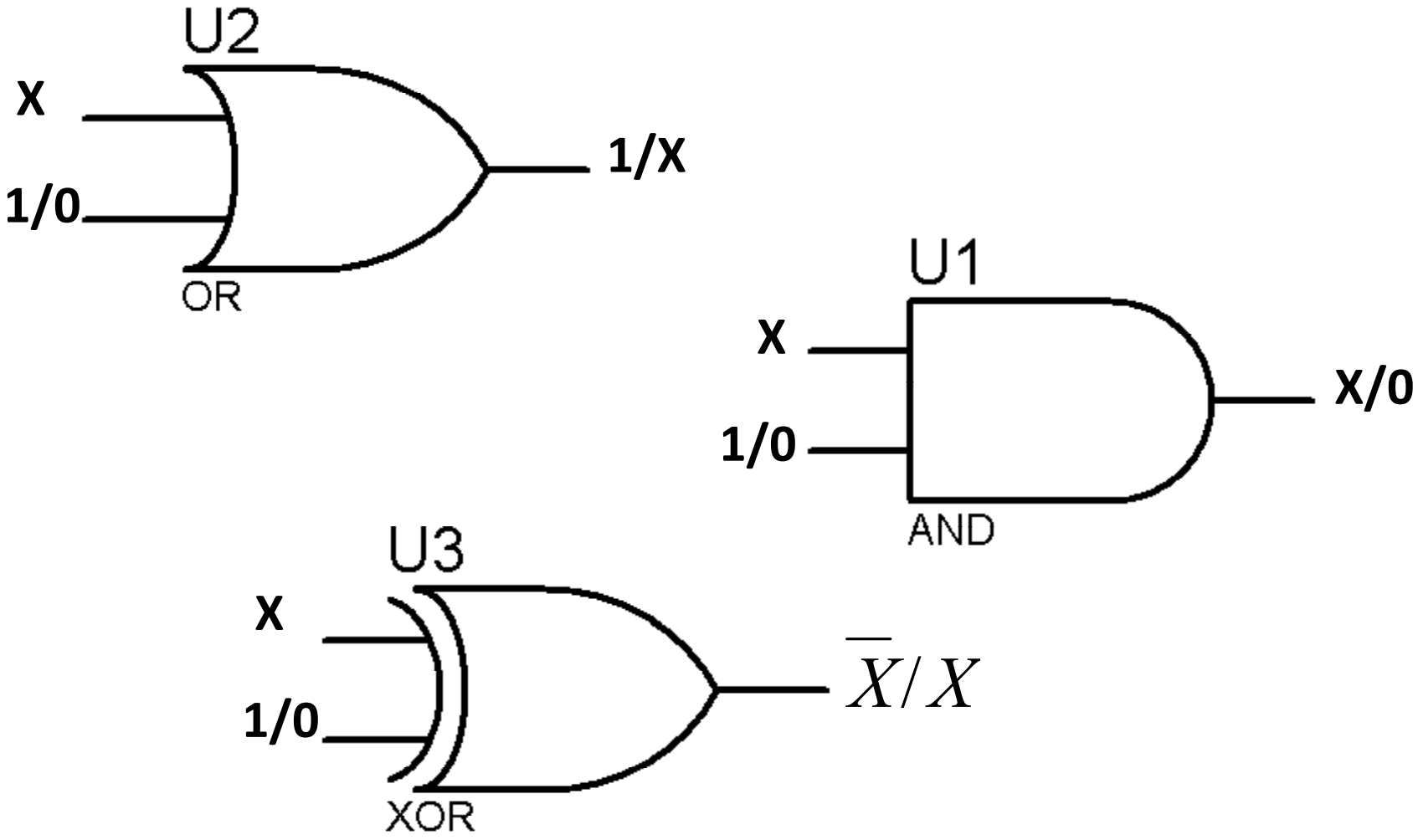
Quizz

- Écrire un programme qui fait le saut vers l'adresse code **0x1700** par :
  - Un saut statique.
  - Un saut dynamique (programmable).

Astuces

- Comment forcer un ou plusieurs bits à 1 ?
- Comment Masquer un ou plusieurs bits à 0 ?
- Comment Inverser (basculer) un ou plusieurs bits ?

Astuces

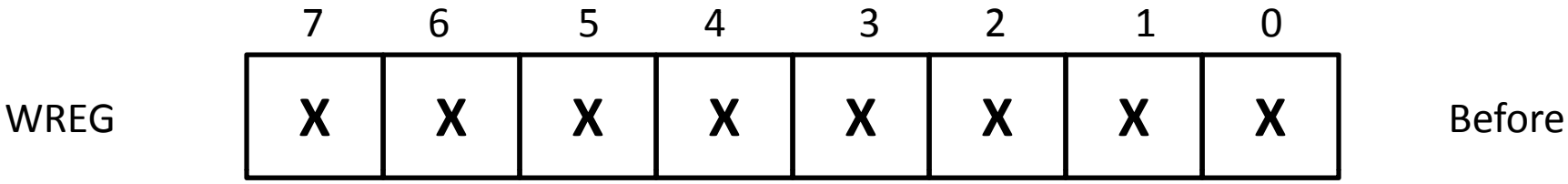


Astuces

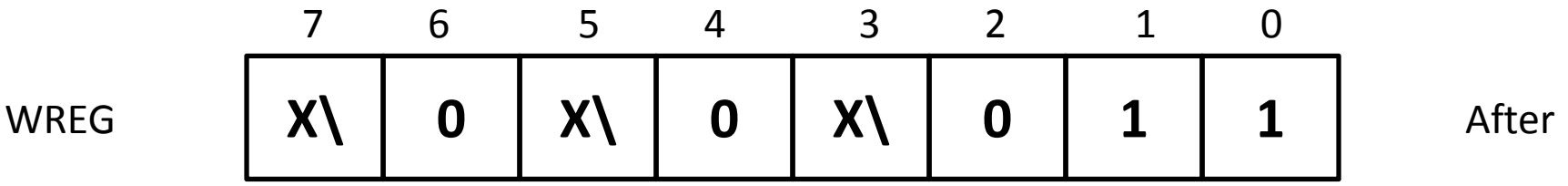
- Pour forcer un bit ou plusieurs à 1, on fait un OR avec un 1.
- Pour Masquer un bit ou plusieurs à 0, on fait un AND avec un 0.
- Pour inverser un bit ou plusieurs, on fait un XOR avec un 1.

Astuces

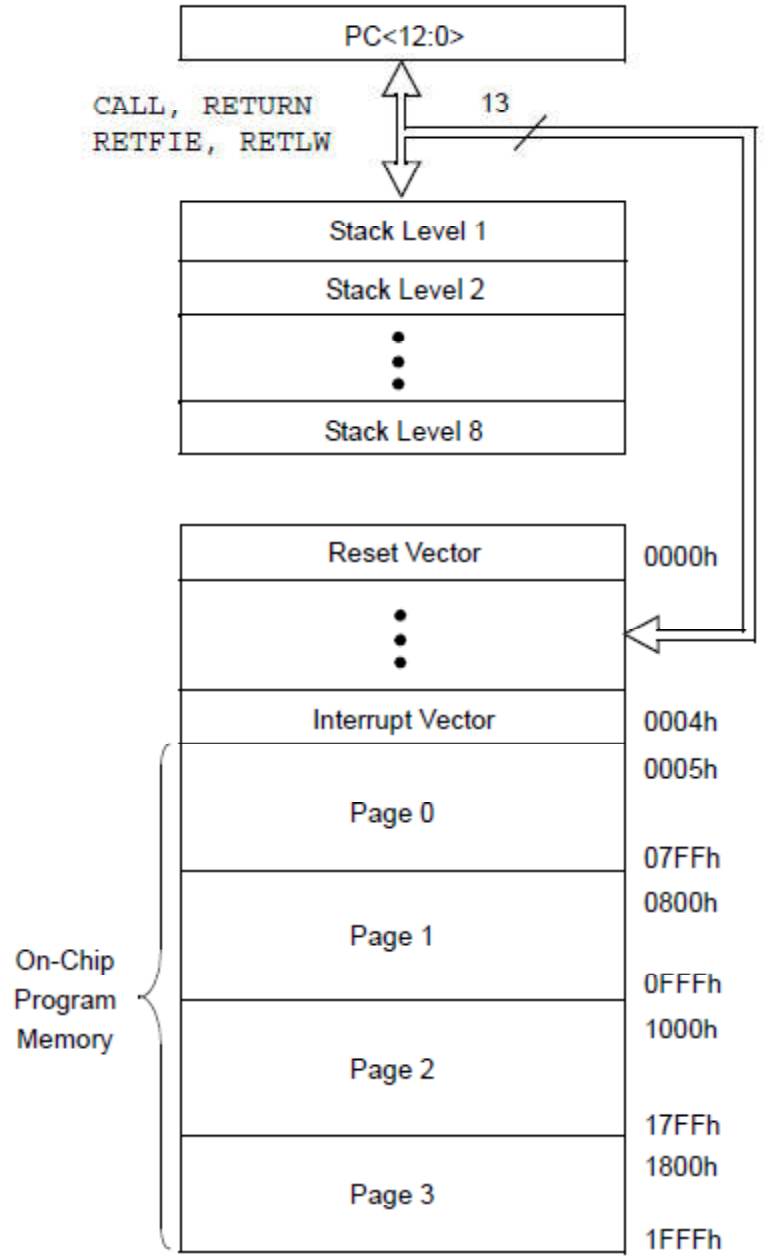
- Exemple: Forcer les bits [b0, b1] à 1, masquer les bits [b2, b4, b6] à 0, et inverser les bits [b3, b5, b7] de registre WREG.



Comment faire ???



Organisation de la zone CODE,STACK



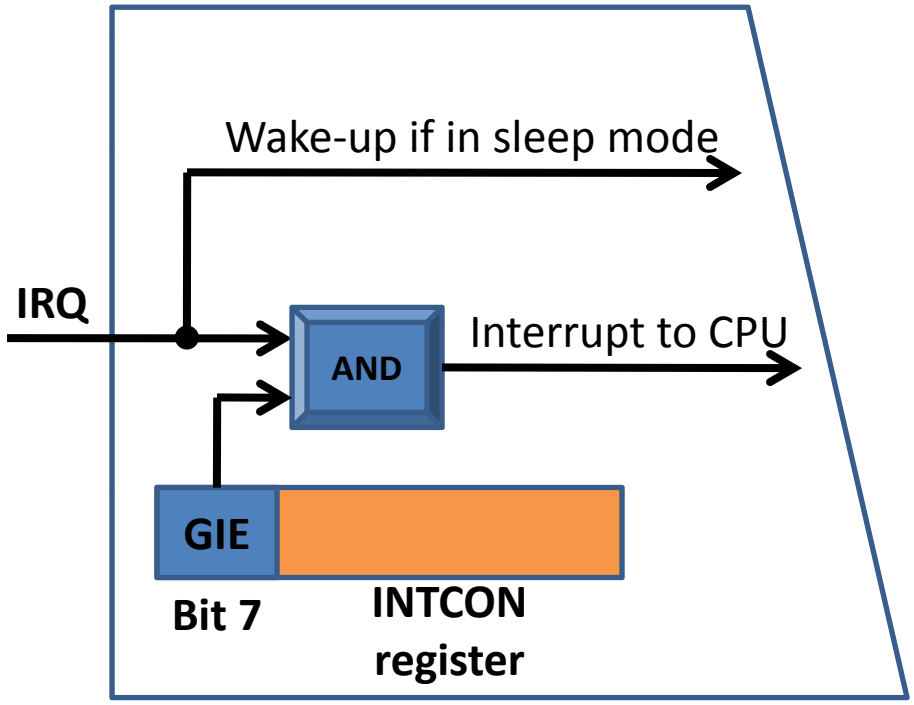
- La zone CODE est une mémoire de taille 8k*14bits (les 14 bits présente la taille d'une instruction).
- La zone STACK (Pile) est une mémoire de type LIFO (Last In First Out) de 8 niveaux de 13bits chacun (13 bits présente la taille du PC).
- Le CPU utilise cette zone pour Empiler/Dépiler les adresses de retour au moment d'appel d'un sous-programme ou un traitement d'une interruption.
- La zone STACK du MID-RANGE reste toujours inaccessible par l'utilisateur, donc on n'a pas des instructions pour manipuler comme le µP 8086 chez Intel (PUSH/POP)

Organisation de la zone CODE

- MICROCHIP appelle un emplacement mémoire dans la zone CODE par le nom « vecteur », un vecteur prend une instruction (op-code & opérandes).
- Les deux vecteurs essentiels dans cette zone sont:
  - Le vecteur RESET: est le premier vecteur d'adresse 0x0000 (au moment de démarrage/reset le CPU branche vers cette adresse).
  - Le vecteur ISR: est le vecteur d'interruption d'adresse 0x0004 (au moment d'une interruption le CPU branche vers cette adresse).

Gestion des interruptions

- Le MID-RANGE a une seule entrée de demande d'interruption masquée par le bit 7 (GIE **G**OLOBAL **I**NTERRUPT **E**NABLE) de registre INTCON comme indique la figure suivante :
- La gestion de la ligne IRQ se fait par:
  - Si GIE est 1 & IRQ est 1 ➔
  - Le CPU termine d'exécution de l'instruction en cours.
  - GIE ← 0 (pour ne pas gêner le traitement de l'interruption en cours).
  - Sauvegarde l'adresse de retour dans le TOS.
  - Saut vers l'adresse 0x0004 (ISR)



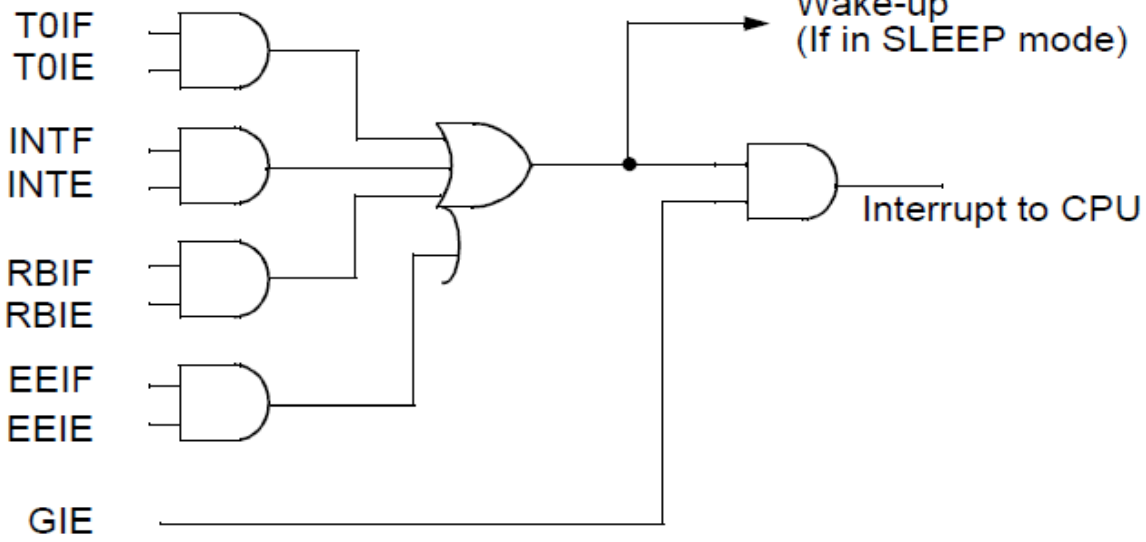
La plus par des µC sont entourés par plusieurs périphériques capables de générer des demandes d'interruption, et le MID-RANGE capable de gérer une seule demande, donc comment faire pour gérer plusieurs demandes d'interruption???

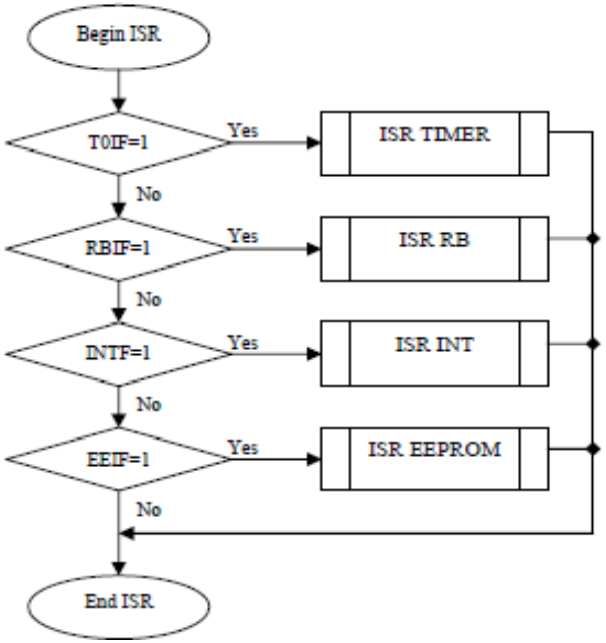
Solution

- La solution proposée par MICROCHIP est d'attribuer pour chaque demande d'interruption un bit « **autorisateur** » et un autre bit « **flag** ».
- **Le bit autorisateur**: on utilise pour activer/désactiver la demande d'interruption par ce périphérique.
- **Le bit flag**: on utilise pour indiquer est-ce que on a une demande d'interruption ou non? Comme indique la figure suivante:

Solution

- Après le saut vers le ISR d'adresse 0x0004, on fait le test de chaque flag ; si la cas, on exécute le ISR correspondant.
- Le premier flag à testé présente la demande de la plus haute priorité.
- Le dernier flag à testé présente la demande de la plus faible priorité.
- On appelle cette méthode de gestion par le nom SOFTWARE POLLING.

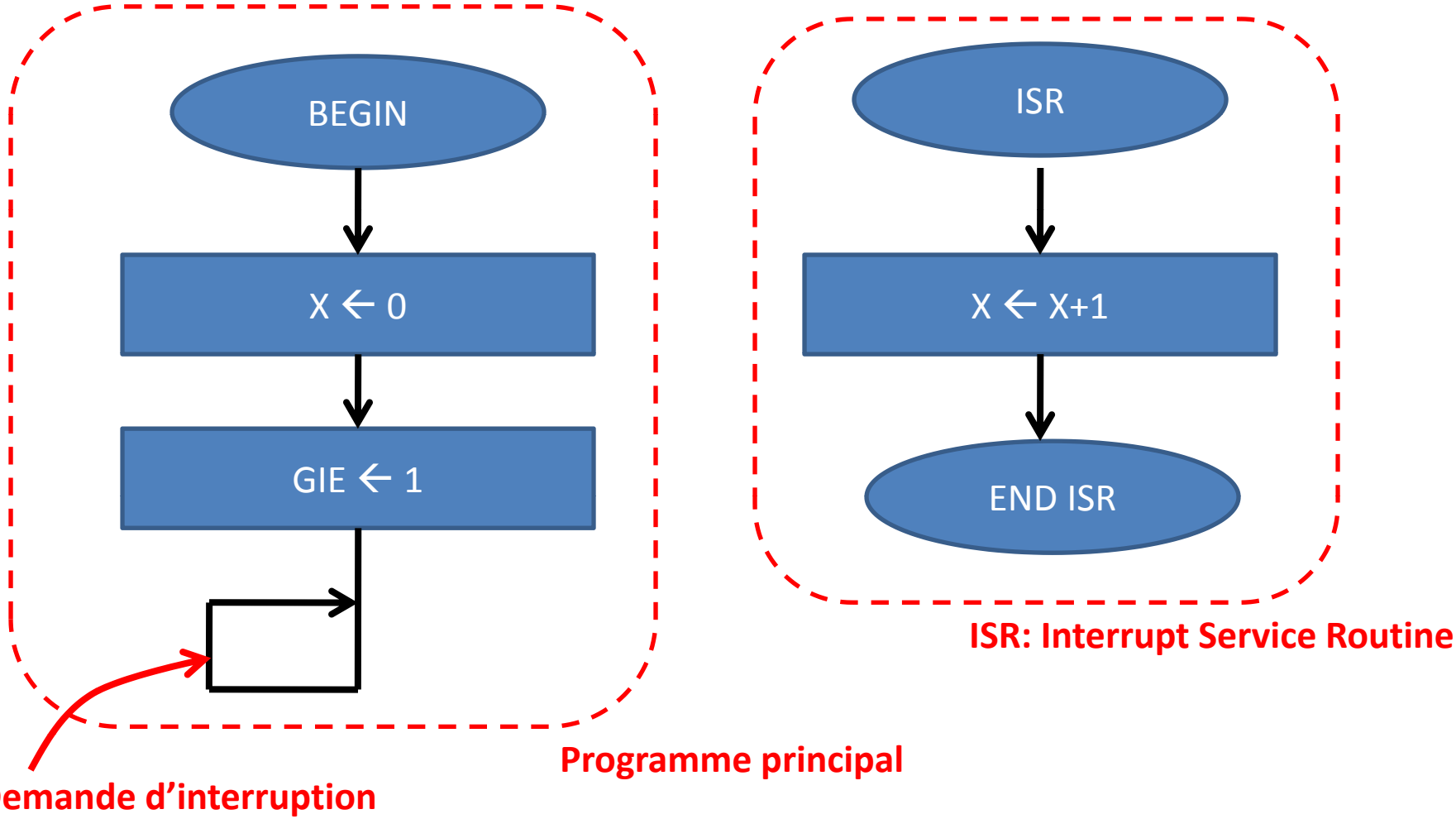




Exemple de gestion d'une interruption par le SIMULATOR du MIDRANGE

- Le code suivant ne fait rien (boucler infiniment dans la même adresse [il plante]), mais à chaque demande d'interruption le ISR incrémente la case mémoire d'offset 0x0C par rapport à la BANK0.

Exemple de gestion d'une interruption par le SIMULATOR du MIDRANGE



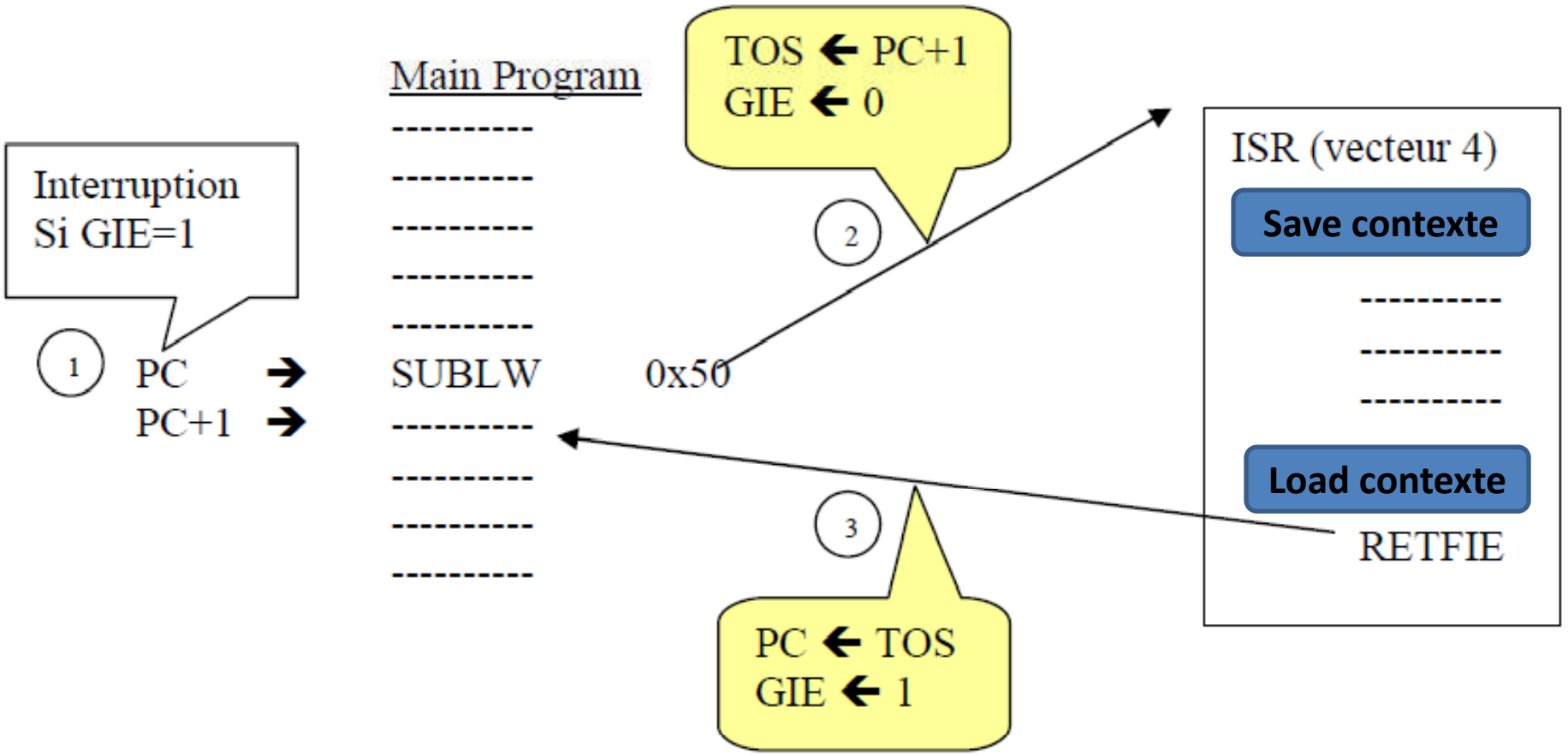
Exemple de gestion d'une interruption par le SIMULATOR du MIDRANGE

```
1  ; programme fait l'incrementation de la case memoire "x" chaque interruption
2  ; sous MID-RANGE
3      ORG 0x0000
4      ; saut vers le programme principal
5      goto main
6
7      ORG 0x0004
8      ; incremente la case memoire "x"
9      INCF 0x0C,F
10     ; retour vers le programme principal
11     RETFIE
12
13 main
14     ; selection bank0
15     BCF STATUS,RP1
16     BCF STATUS,RP0
17
18     ; Init x par zero
19     CLRF 0x0C
20
21     ; activation des interruption par la mise à 1 de GIE
22     BSF INTCON,GIE
23
24     ; boucle infini (CPU plante)
25     GOTO $
```

ISR

MAIN

Gestion avancée des interruptions



Gestion avancée des interruptions

- SAVE CONTEXTE : on fait le sauvegarde de WREG et le STATUS dans la RAM.
- LOAD CONTEXTE : on fait le chargement de WREG et le STATUS à partir de la RAM.
- le but de SAVE/LOAD CONTEXTE est de ne pas perturber le bon fonctionnement du programme interrompu.

Gestion avancée des interruptions

- SAVE CONTEXTE:

```
MOVWF W_REG
SWAPF W_REG,F
MOVF STATUS,W
MOVWF STATUS_REG
```

- LOAD CONTEXTE:

```
MOVF STATUS_REG,W
MOVWF STATUS
SWAPF W_REG,W
```

Détail de la gestion d'une interruption