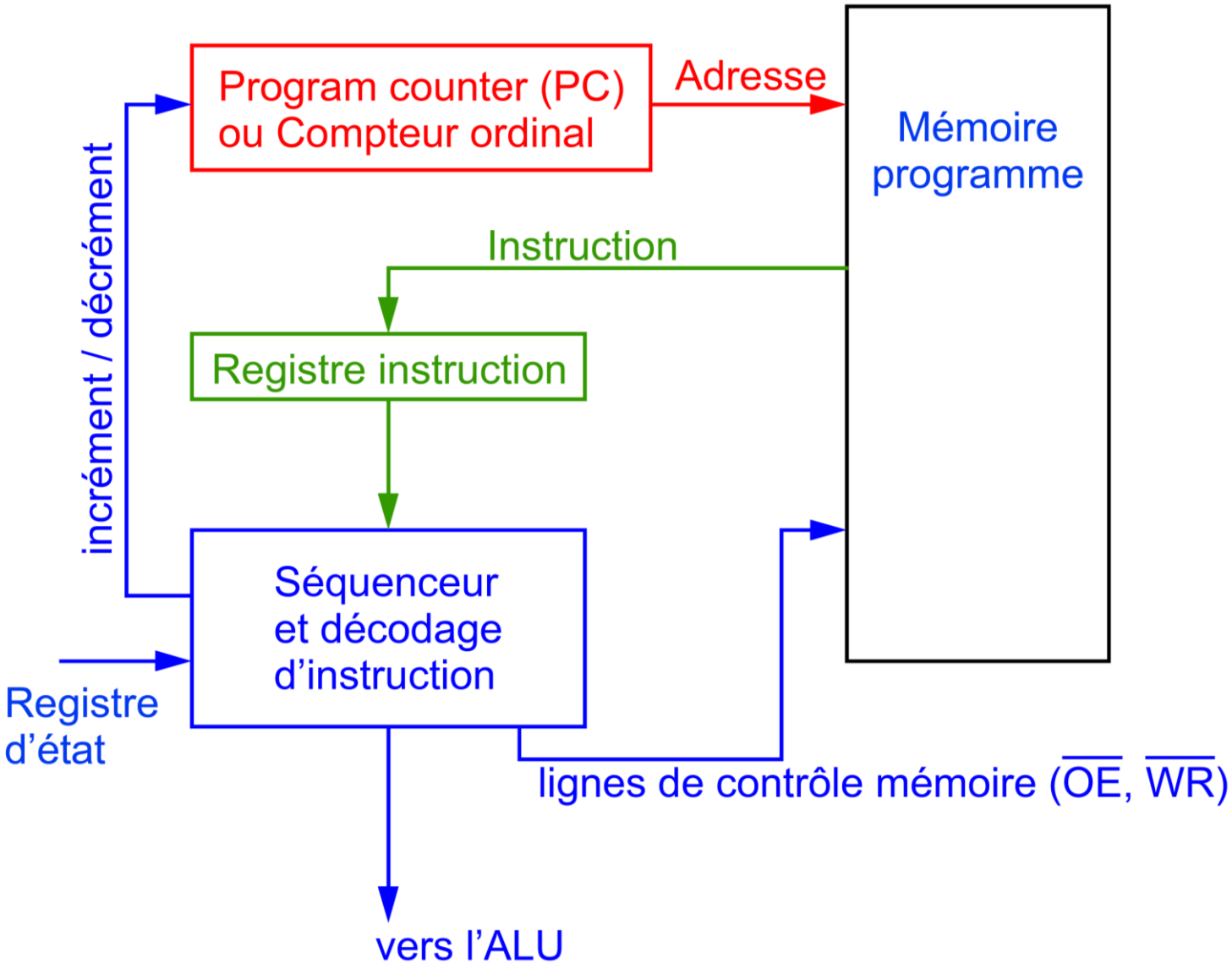
CHAPITRE EN7

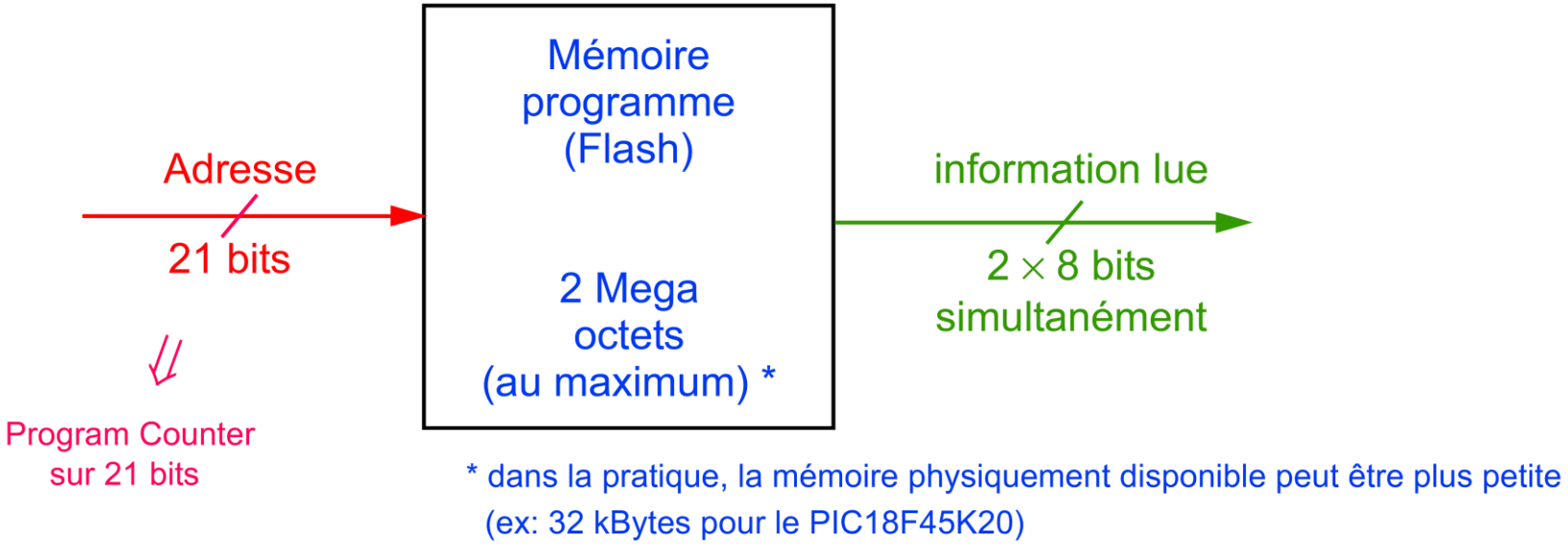
Architecture élémentaire des ordinateurs: l'unité de contrôle

SOMMAIRE

L'UNITÉ DE CONTRÔLE – Architecture simplifiée

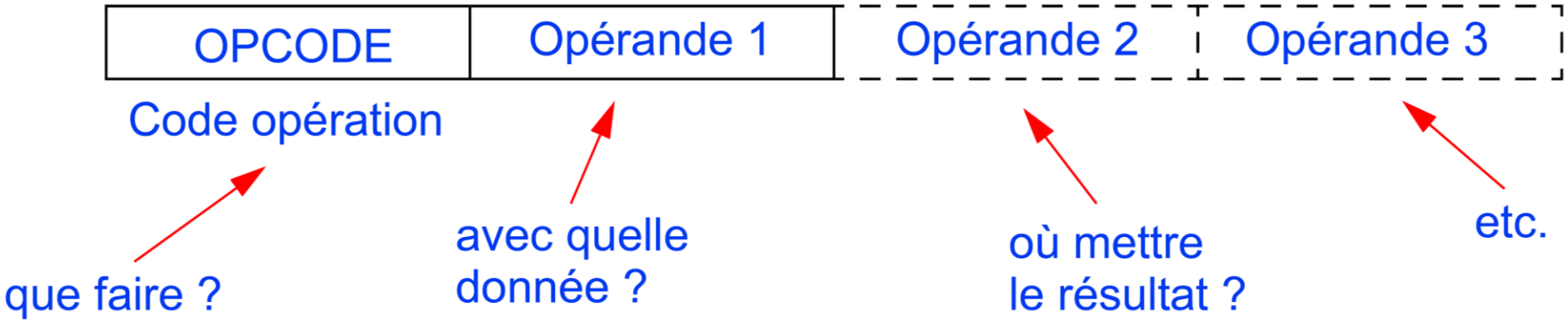


L'UNITÉ DE CONTRÔLE – Mémoire programme (cas du PIC18)



- Une instruction = 2 octets ** ⇒ 1M instructions théoriquement adressables
  - (16 k instructions possibles pour le PIC18F45K20)
  - ** excepté 4 instructions codées sur 4 octets (CALL, GOTO, MOVFF, LFSR)
- Une instruction = 2 "cases" (registres) de la mémoire Flash
  - ⇒ l'octet de poids faible est toujours à une adresse paire
  - ⇒ entre 2 instructions successives dans la mémoire programme, le PC s'incrémente de 2

L'UNITÉ DE CONTRÔLE – Structure d'une instruction



- Pour les PIC18, une instruction occupe (généralement) 2 octets (soit un mot de 16 bits) dans la mémoire programme.
- Exemple:

Byte-oriented file register operations

15 10	9	8	7 0
OPCODE	d	a	f (FILE #)

d = 0 for result destination to be WREG register
d = 1 for result destination to be file register (f)
a = 0 to force Access Bank
a = 1 for BSR to select bank
f = 8-bit file register address

Example Instruction

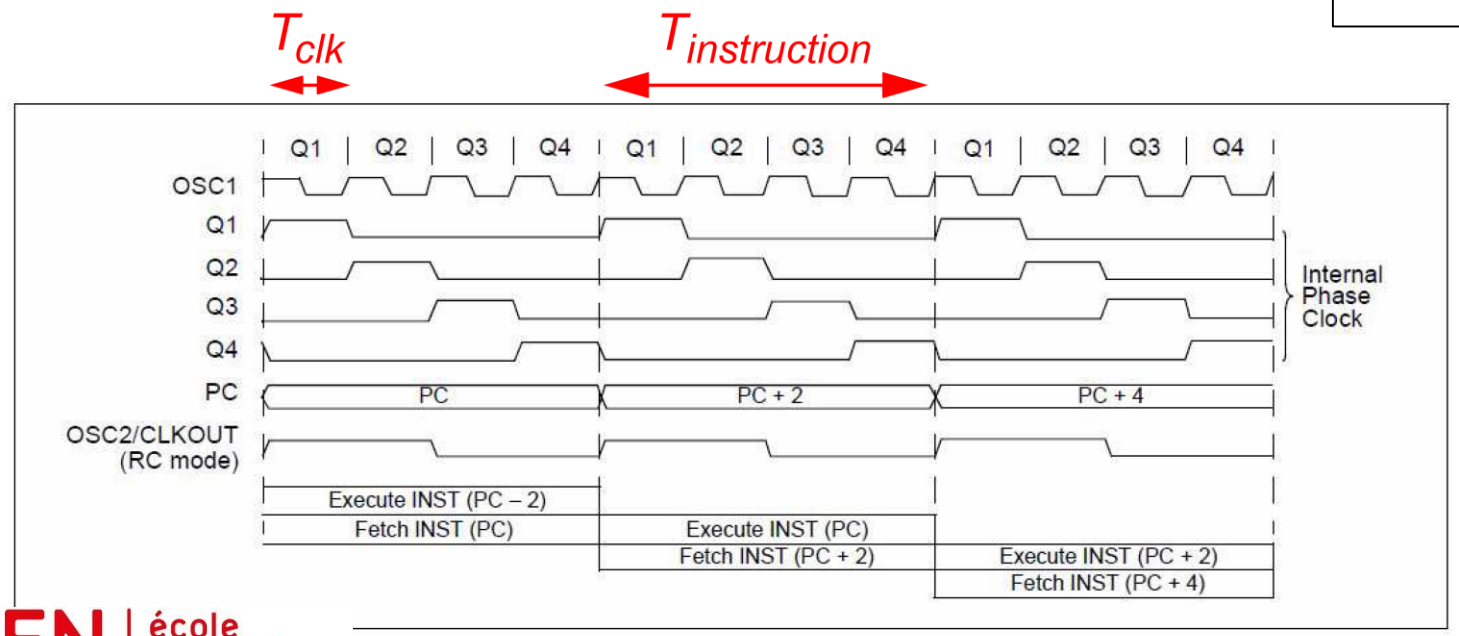
```
ADDWF MYREG, W, B
```

cf. datasheet PIC18F45K20 page 317

L'UNITÉ DE CONTRÔLE – Déroulement d'une instruction

- Une instruction = 4 cycles machine
  - 1 - Lecture de l'instruction (Fetch): le mot d'instruction est placé dans le registre d'instruction (IR) et décodé
  - 2 - Chargement de la donnée en mémoire (le cas échéant)
  - 3 - Exécution de l'opération (Execute)
  - 4 - Ecriture du résultat en mémoire

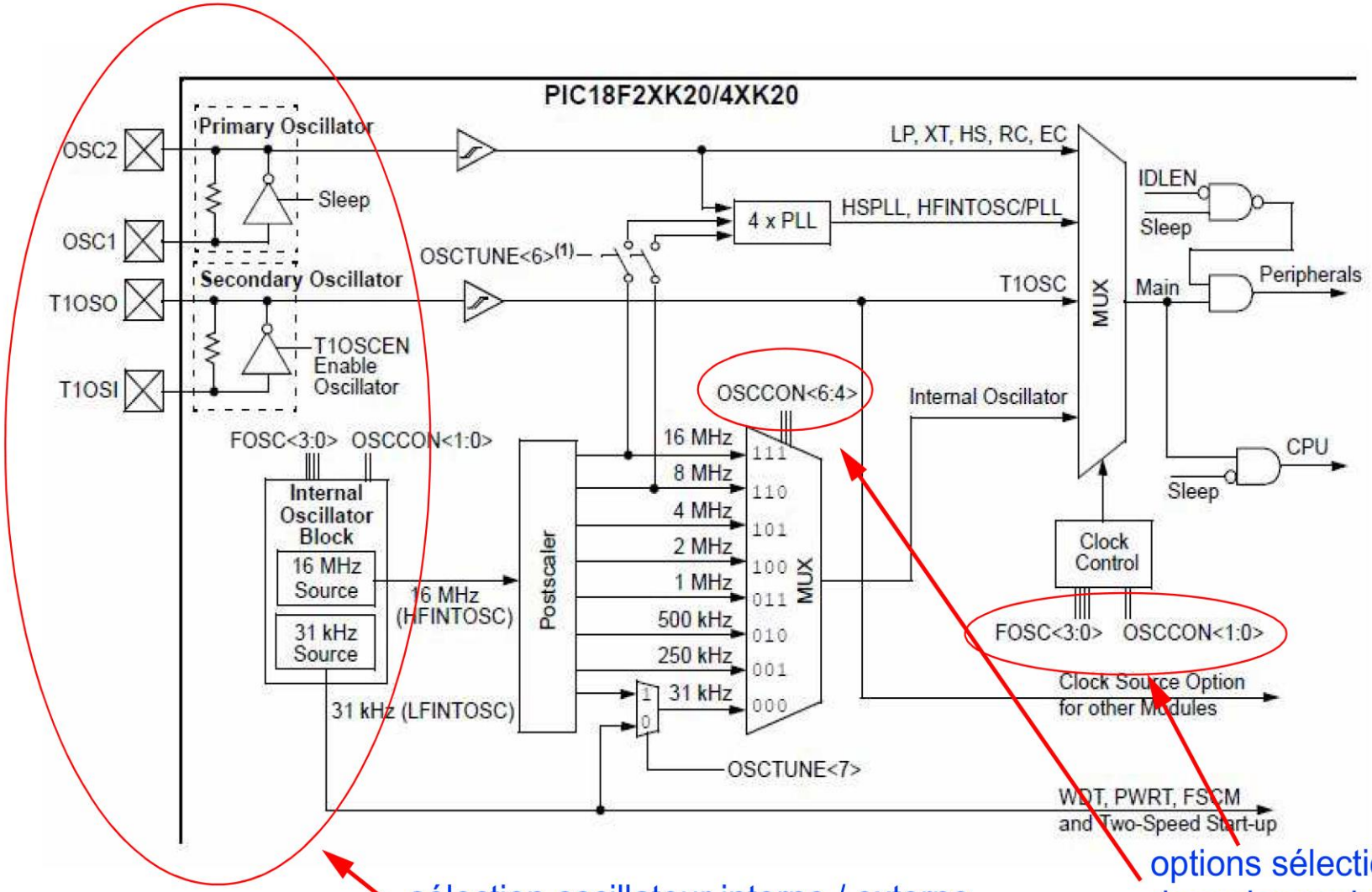
temps d'exécution : $t = \frac{4}{f_{CLK}}$



Program Counter : contient l'adresse de l'instruction qui va être executée au cycle suivant

cf. datasheet PIC18F45K20 page 69

L'UNITÉ DE CONTRÔLE – Exemple du PIC18F45K20: choix de f_{CPU}



sélection oscillateur interne / externe dans les "bits de configuration"

options sélectionnées dans des registres spécifiques

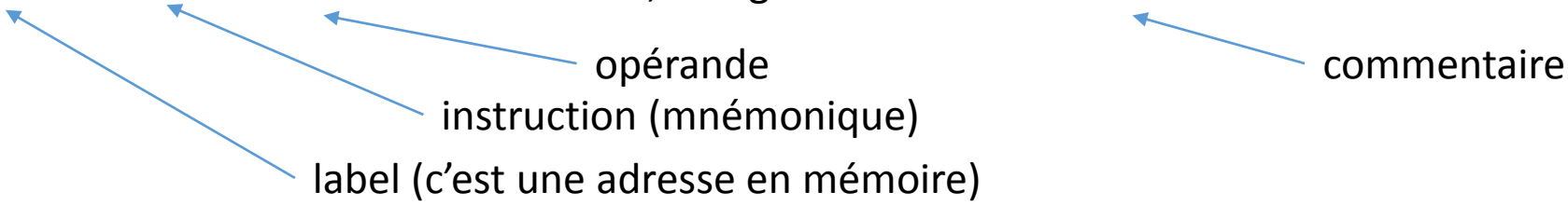
LES RESSOURCES, LE JEU D'INSTRUCTIONS, ASSEMBLEUR – Les ressources (exemple du PIC18F45K20)

- Du point de vue utilisateur, le calculateur est :
  - Un ensemble de registres :
    - Accumulateur (W) sur 8 bits
    - Compteur ordinal (PC) sur 21 bits
    - Une pile (Stack) sur 21 bits et un pointeur de pile (Stack Pointer) sur 5 bits
    - Registre d'état (STATUS)
    - Registres de gestion des périphériques (Special Function Registers)
  - Une mémoire centrale interne (microcontrôleur) et/ou externe (microprocesseur):
    - 1536 octets de RAM interne
    - 256 octets d'EEPROM interne
    - 32 Koctets de FLASH interne
  - Un jeu d'instructions, des modes d'adressage
  - Un ensemble de périphériques internes et/ou externes

- Exemple du PIC18: 75 instructions en 4 groupes pr
  - **Byte-oriented operations**: pour manipuler (lire / écrire) un octet
    ex: ADDWF, MOVWF, INCF, ...
  - **Bit-oriented operations**: pour manipuler (modifier / tester) un bit d'un octet
    ex: BSF, BTFSC, ...
  - **Literal operations**: lorsque l'instruction donne directement la valeur d'un octet
    ex: MOVLW, ADDLW, ...
  - **Control operations**: pour gérer par exemple le déroulement du programme
    ex: CALL, GOTO, NOP, ...

LES RESSOURCES, LE JEU D'INSTRUCTIONS, ASSEMBLEUR – Langage assembleur (notions)

- Les symboles
  - nom_du_symbole EQU valeur_du symbole Exemple: tempo EQU 0xF0
- Les variables (contenu dynamique, stocké en RAM)
  - UDATA (ou UDATA_ACS) ; permet de réserver des adresses en RAM pour des variables
  - var1 RES 1 ; variable sur 1 octet
  - var2 RES 2 ; variable sur 2 octets
- Les constantes (contenu statique, stocké en ROM)
  - CODE (ou [label] CODE [ROM_address])
  - tableau DB 0x11, 0xAF, 0x34, 0xE2 ; définit un ensemble de valeurs écrites dans la mémoire Flash
- Représentation des valeurs (par exemple, le nombre 10)
  - D'10' ; valeur décimale 0x0A ou H'0A' ; valeur hexadécimale B'00001010' ; valeur binaire
- une ligne d'assembleur
  - toto MOVLW 0xE5 ; charge l'accumulateur à la valeur 229



LES MODES D'ADRESSAGE – *Vue globale pour le PIC18*

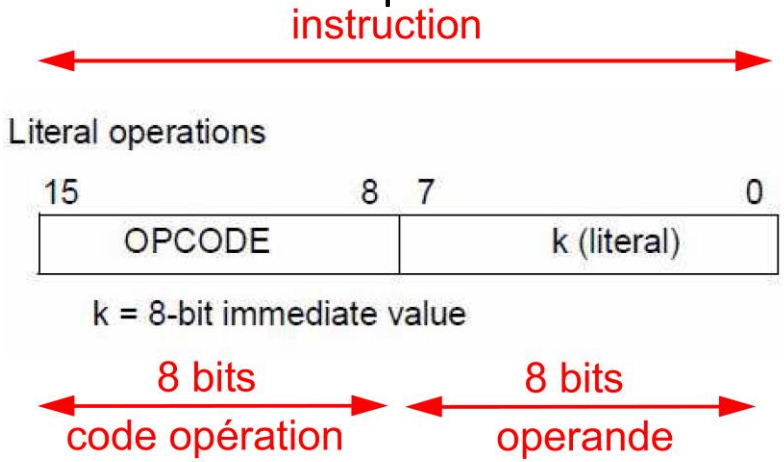
- 4 modes d'adressage
  - **Inherent** : l'instruction ne nécessite pas d'opérande
  - **littéral (ou immédiat)** : l'instruction contient la valeur de l'opérande
  - **Direct** : l'instruction contient l'adresse de l'opérande
  - **Indirect** : l'instruction indique où se trouve l'adresse de l'opérande

LES MODES D'ADRESSAGE – *Inherent*

- Adressage inherent:
  - L'instruction ne nécessite pas d'opérande
  - Exemple:
    - NOP ; No Operation
    - RESET ; Software device RESET
    - SLEEP ; Go into standby mode

LES MODES D'ADRESSAGE – *Littéral*

- Adressage littéral (ou immédiat) :
  - L'instruction contient la valeur de l'opérande



  - Exemple:

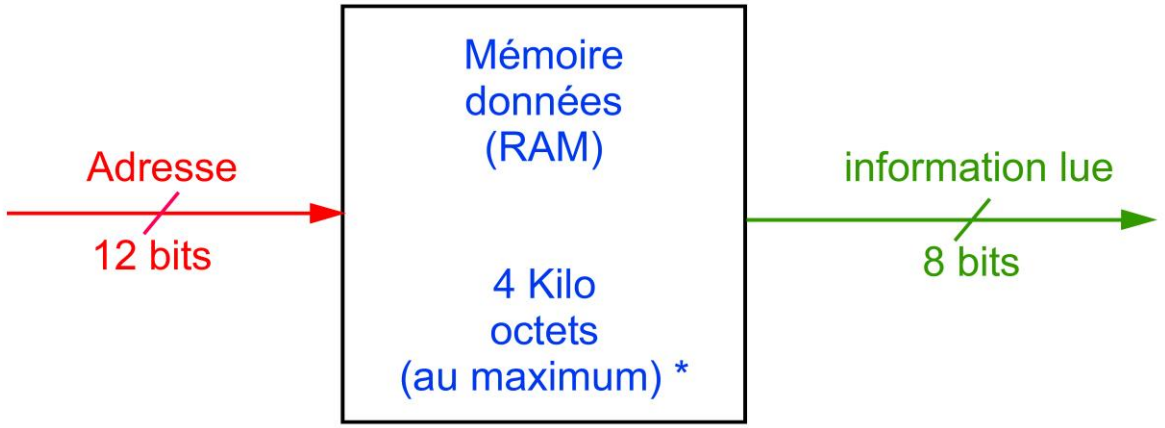
adresse	contenu	instruction		
E102	0E F1	MOVLW	0xF1	; chargement de W à F1h
E104	0F 55	ADDLW	0x55	; addition de W et 55h
E106	0B 0F	ANDLW	0x0F	; AND entre W et 0Fh

PC +2

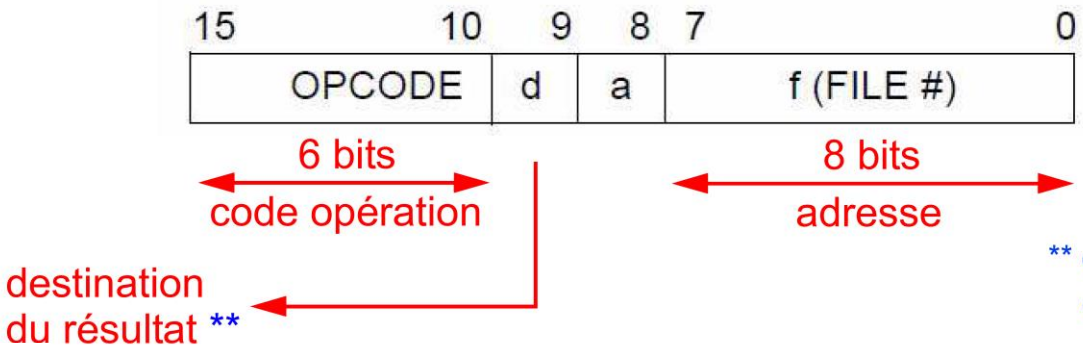
PC +2

LES MODES D'ADRESSAGE – Direct

- Adressage direct :
  - La donnée sur laquelle on veut agir est stockée à l'adresse qui suit le code opération



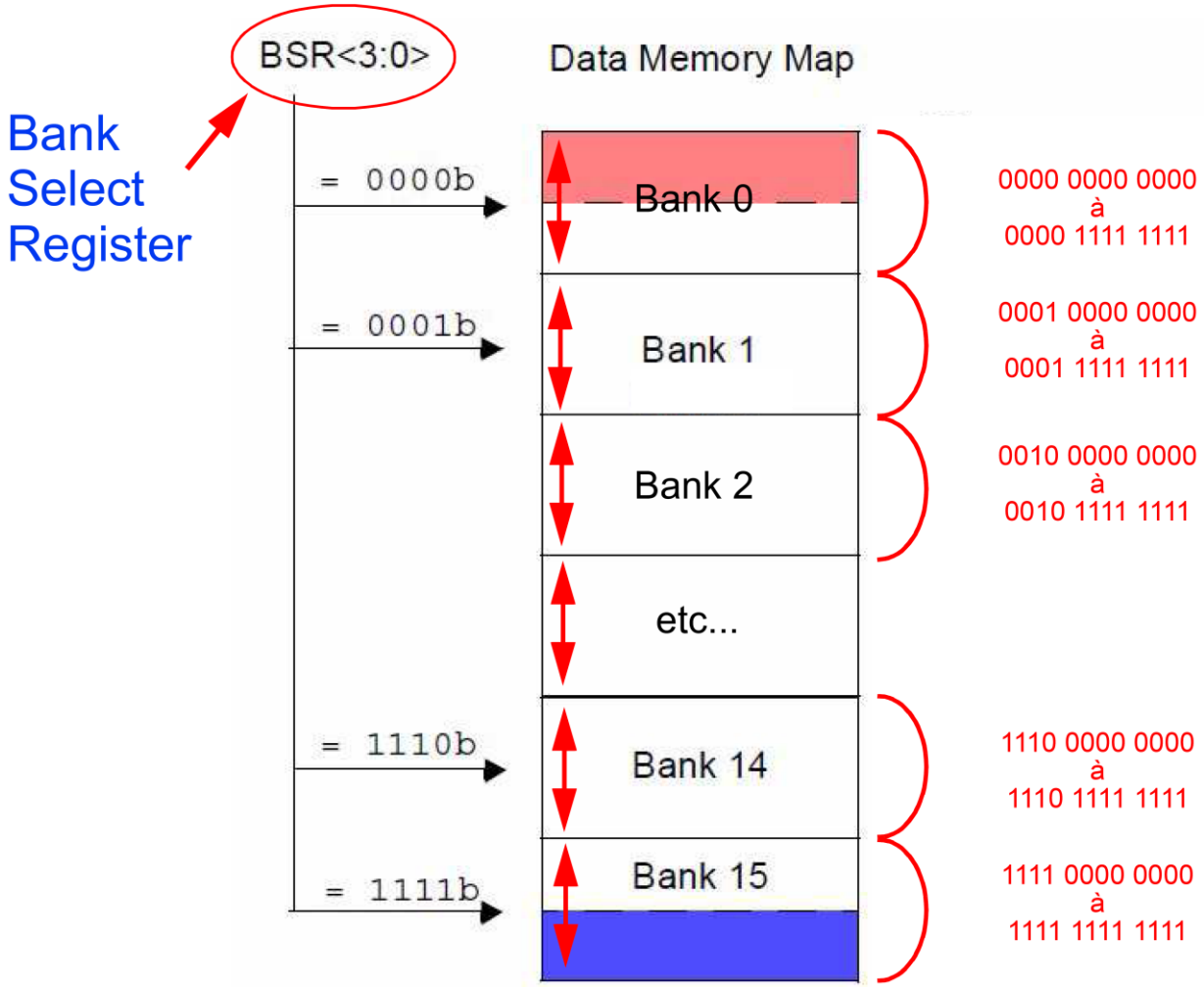
* dans la pratique, la mémoire physiquement disponible peut être plus petite
(ex: 1536 Bytes pour le PIC18F45K20)



** d = 0 ⇒ résultat dans W
d = 1 ⇒ résultat dans le registre spécifié dans l'instruction

LES MODES D'ADRESSAGE – Direct

L'instruction indique les 8 bits de poids faibles de l'adresse (sur 12 !)
⇒ les 4 bits de poids forts ("banque") sont définis ailleurs !



LES MODES D'ADRESSAGE – Direct

L'adresse complète est obtenue en concaténant les 4 bits contenus dans le regitre BSR (poids forts) et les 8 bits contenus dans l'instruction (poids faibles).

Exemple:

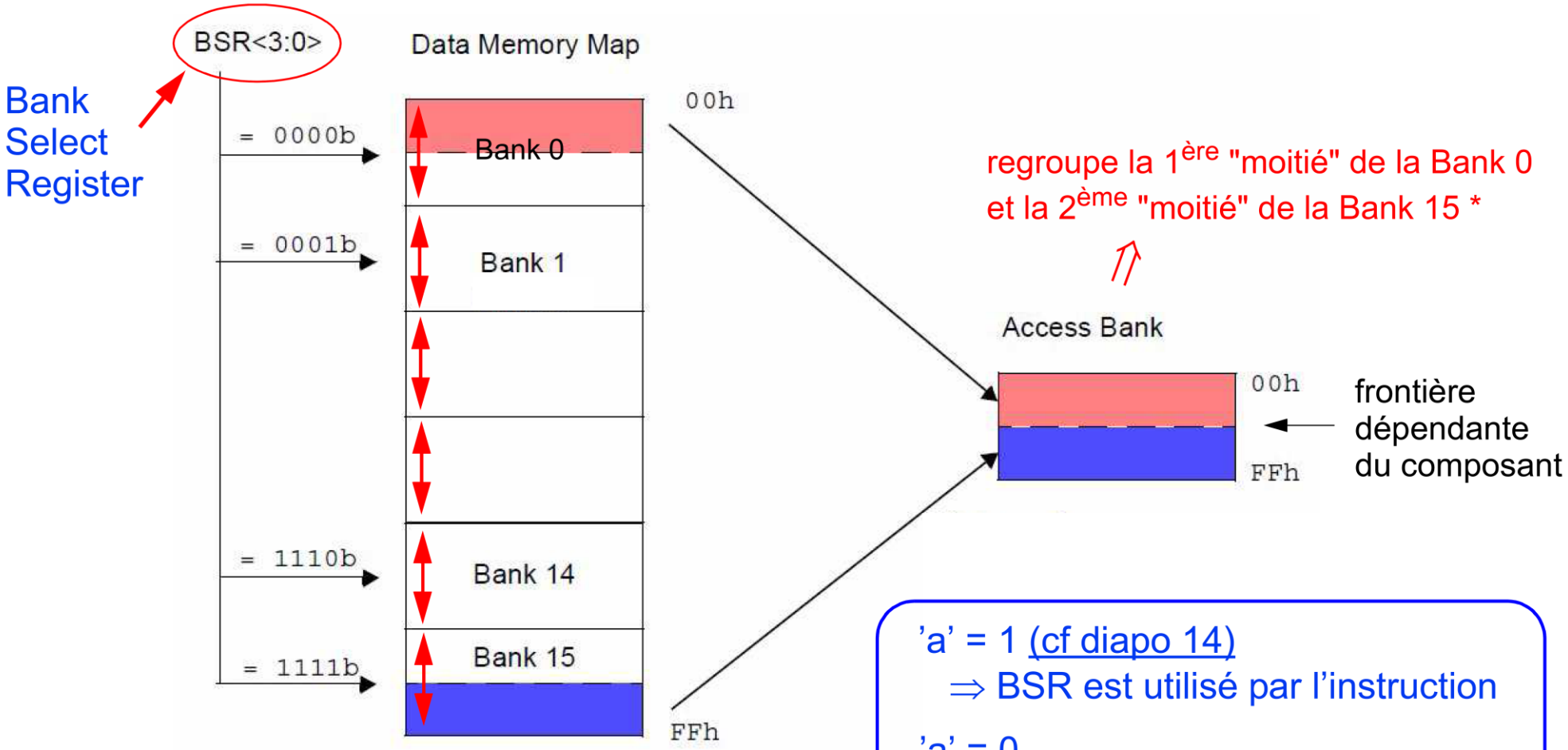
```
            UDATA
my_var      RES 1          ; la variable my_var est placée par exemple à l'adresse 0 00h
            ……
            MOVLB  0x00    ; le registre BSR est chargé à la valeur 00h
            ……
            MOVF   my_var, 1   ; l'accumulateur est chargé avec la donnée stockée à l'adresse 0 00h
            ……
            MOVLB  0x02    ; le registre BSR est chargé à la valeur 02h
            ……
            MOVF   my_var, 1   ; l'accumulateur est chargé avec la donnée stockée à l'adresse 2 00h
                               ; MAUVAISE ADRESSE !
```

instructions identiques

résultats différents !

⇒ BSR (si on l'utilise) doit être initialisé AVANT d'accéder à une donnée

LES MODES D'ADRESSAGE – Direct + "utilisation de l'Access Bank"



'a' = 1 (cf diapo 14)
⇒ BSR est utilisé par l'instruction

'a' = 0
⇒ BSR est ignoré et l'instruction utilise l'"Access Bank"

* **PIC18F45K20**: **96** premiers octets de la Bank 0 + **160** derniers octets de la Bank 15

LES MODES D'ADRESSAGE – Direct + "utilisation de l'Access Bank"

RAM pour "usage général"
(stockage de variables,
de résultats de calculs, etc.)

contient les
"Special Function Registers"

(registres utilisés pour
les périphériques et la
configuration du micro-contrôleur)

(ex: ports d'E/S, horloge CPU, etc.)

LES MODES D'ADRESSAGE – Direct

Exemple: accès à une même variable par deux routines distinctes

```
UDATA_ACS ;pour déclarer des variables dans l'Access Bank«
Var1      RES      1
Var2      RES      1
```

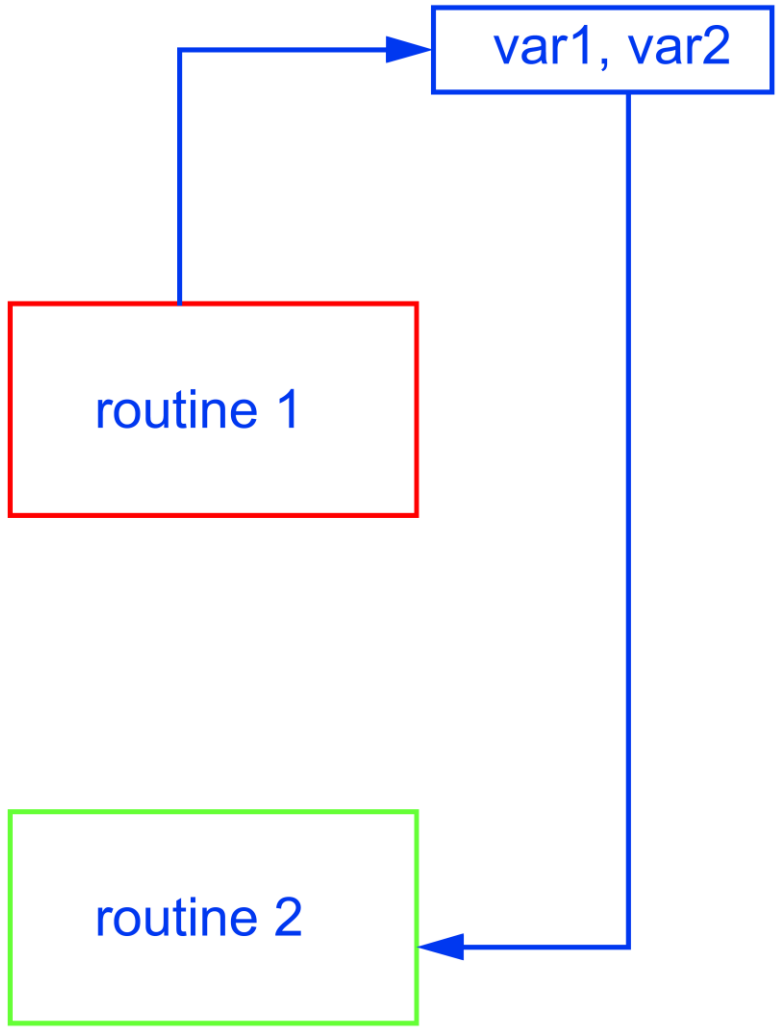
Access Bank

```
MOVWF   var1, 0     ; stockage mémoire par routine 1
……
……
MOVWF var2, 0
……
……
……
```

destination = W

Access Bank

```
MOVF      var1 ,0, 0 ; récupération du calcul précédent
……                   ; par routine 2
……
MOVF      var2 ,0, 0
```



LES MODES D'ADRESSAGE – *Indirect*

- Adressage indirect :
  - L'adresse de la donnée sur laquelle on veut agir n'est pas fixée dans l'instruction : elle se trouve dans un registre "File Select Register" (**pointeur**).
  - ⇒ utile pour manipuler des tableaux de données

    - Pour PIC18: 3 pointeurs disponibles
      - FSR0H & FSR0L
      - FSR1H & FSR1L
      - FSR2H & FSR2L

      contiennent chacun une adresse complète (12 bits) (⇒ inutile de préciser la Bank)
    - Pour accéder à une donnée dont l'adresse est stockée dans **FSRx**: utiliser **INDFx**

    - Exemples:
      - MOVF INDF0, 0 ; lit la donnée dont l'adresse est contenue dans FSR0H:FSR0L
      - MOVWF INDF1, 0 ; écrit une donnée à l'adresse contenue dans FSR1H:FSR1L

LES MODES D'ADRESSAGE – *Indirect*

Exemple: mise à 0 (CLEAR) de la RAM de l'adresse 20h à 2Fh

```
        CLRF    FSR0H, 0
        MOVLW   0x20
        MOVWF   FSR0L, 0
Boucle
        CLRF    INDF0           ; met à 0 (Clear) le contenu du registre (File)
        INCF    FSR0L, 1
        BTFSS   FSR0L, 4, 0     ; tester le bit (Bit Test) n°4 du registre ( File) FSR0L
                                ; et sauter (Skip) l'instruction suivante s'il vaut 1 (Set)
        GOTO    boucle
Suite
        ……
```

⇒

FSR0H	FSR0L
00h	20h

LES MODES D'ADRESSAGE – *Indirect*

Sur les PIC18, il est possible de modifier la valeur stockée dans FSRx
en même temps que l'on y accède.
⇒ au lieu d'utiliser INDFx, utiliser POSTDECx (post-decrement)
ou POSTINCx (post-increment)
ou PREINCx (pre-increment)
ou PLUSWx (ajoute à FSR la valeur dans W,
sans modifier FSR ni W,
⇒ ***adressage indirect indexé***)

Autre solution pour l'exemple précédent:

```
        ……
  Boucle
        CLRF POSTINC0
        BTFSS FSR0L, 4, 0
        GOTO boucle
        ……
```

LES MODES D'ADRESSAGE – *Indirect*

Adressage indirect avec FSRx et INDFx, etc.
⇒ utile pour travailler sur des tableaux de données stockées en RAM.
Pour travailler avec des tableaux de données stockées en ROM (mémoire Flash):
⇒ utiliser la méthode "COMPUTED GOTO" (cf page 68 datasheet PIC18F45K20).

```
        ……
        MOVLW 0x01
        MOVWF PCLATH   ——→ charge PCLATH en prévision de ce qui suit
        MOVF  offset, W ——→ charge dans W l'offset du tableau qui sera lu (= n° de ligne de tableau)
        CALL  table
        ……
ORG     0x0100 ————————→ écrit les lignes suivantes dans la Flash à partir de l'adresse 0x0100
Table
        ADDWF PCL ————→ modifie la valeur du Program Counter en y ajoutant le contenu de W
        RETLW 0xnn ———→ sort de la routine avec la valeur 0xnn dans W
        RETLW 0xnn
        RETLW 0xnn
        ……
```

LES ROUTINES – *Mécanisme des routines*

- Routine (sous-programme) :
  - interrompt la ***séquence normale*** du programme
  - occasionne un ***branchement*** vers une adresse définie

    CALL ma_routine

    ⇒ il ne faut pas exécuter l'instruction qui suit immédiatement
    (dont l'adresse est dans **P**rogram **C**ounter)
    (il faudra le faire plus tard...)

    ⇒ il faut remplacer l'adresse qui était dans le PC avec celle de "ma_routine"

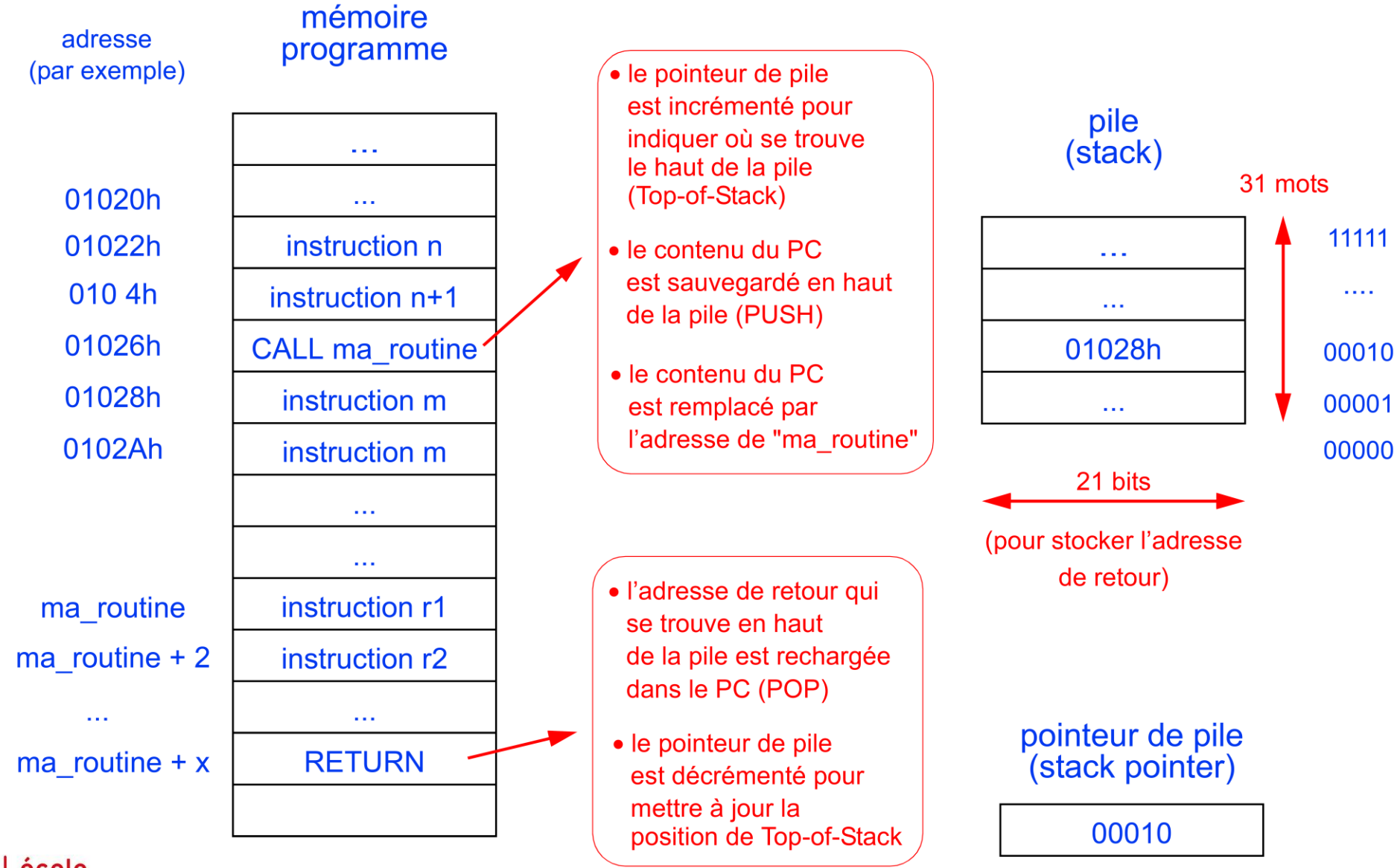
    ⇒ l'adresse qui était dans le PC doit être sauvegardée car elle indique
    où reprendre le programme qui a été interrompu
    cette adresse est sauvegardée dans la pile (stack)

  - ***retourne*** ensuite au programme principal

    RETURN

    ⇒ l'adresse de retour sauvegardée en haut de la pile est rechargée dans le PC

LES ROUTINES – Mécanisme des routines



LES ROUTINES – Sauvegarde du contexte

Sur les PIC18, il est possible de sauvegarder automatiquement 3 registres importants lors de l'exécution d'une routine:

W, STATUS et BSR (Bank Select Register)

Utiliser les instructions:

CALL ma_routine, FAST

RETURN, FAST

LES INTERRUPTIONS – Définition

Interruption : évènement extérieur asynchrone, matériel ou logiciel, qui interrompe la séquence normale du programme et occasionne un branchement vers une adresse pré-définie.

Dans le cas du PIC18, cette adresse pré-définie vaut 00008h (priorité haute) ou 00018h (priorité basse).

Les interruptions peuvent être masquées (i.e. non activées) ou non.

Sources possibles d'interruptions:

- External interrupt from the INT, INT1, and INT2 pins
- Change on RB7:RB4 pins
- TMR0 Overflow
- TMR1 Overflow
- TMR2 Overflow
- TMR3 Overflow
- USART Interrupts
  - Receive buffer full
  - Transmit buffer empty
- SSP Interrupt
- SSP I^2C bus collision interrupt
- A/D conversion complete
- CCP interrupt
- LVD Interrupt
- Parallel Slave Port
- CAN interrupts

etc.....

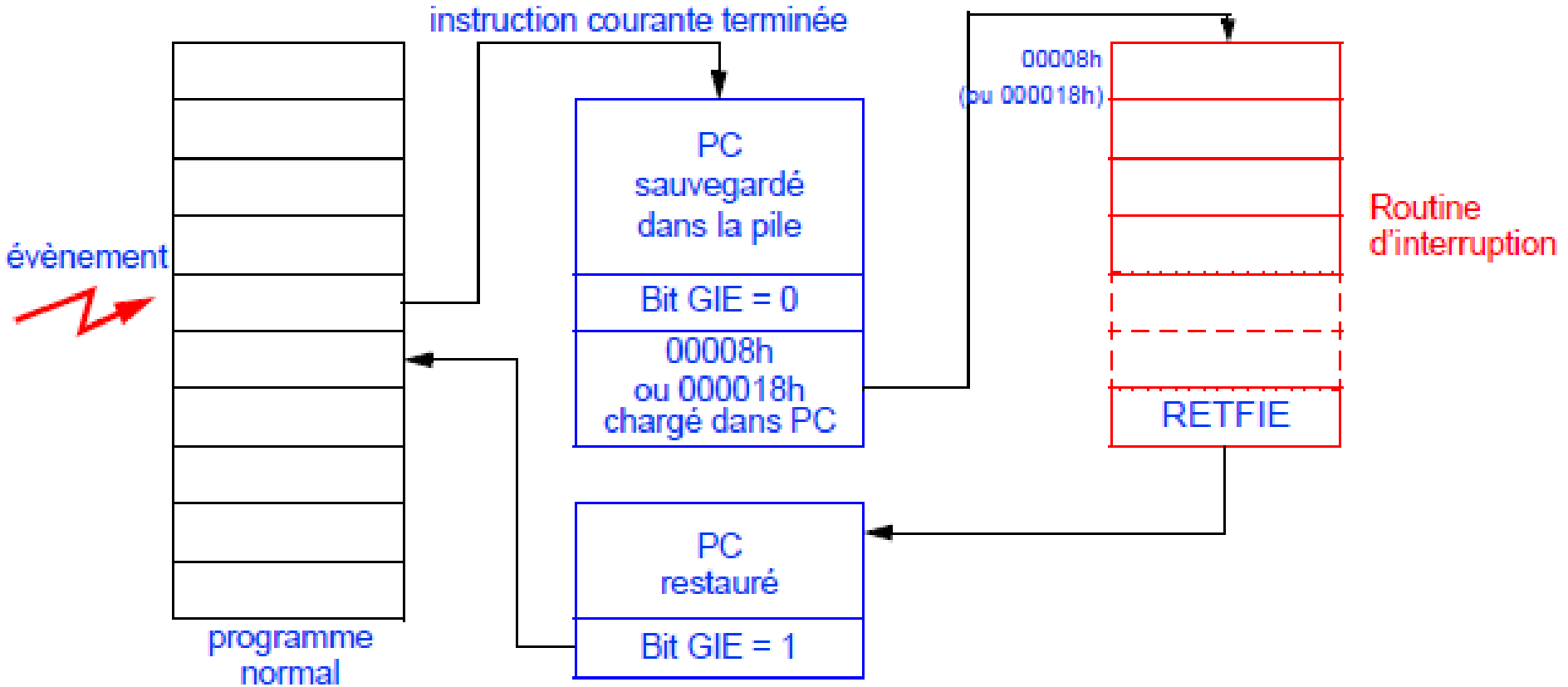
LES INTERRUPTIONS – Traitement

si masquée: non traitée mais mémorisée pour traitement ultérieur (FLAG correspondant= 1)

c'est le cas s'il y a conflit (gestion par niveau de priorité)

ou si on traite actuellement une autre interruption

si non masquée: traitement *après la* ***fin normale de l'instruction en cours***



LES INTERRUPTIONS – Traitement

- Si il existe plusieurs sources d'interruption, il est possible de tester au début de la routine d'interruption les flags d'interruption afin de déterminer lesquels sont activés.
  - L'événement ayant provoqué l'interruption peut ainsi être déterminé.
  - En cas d'interruptions simultanées, la gestion des priorités se fait de façon logicielle, selon l'ordre dans lequel sont testés les flags.
- Généralement, les flags d'interruption doivent être effacés dans la routine d'interruption, avant l'instruction RETFIE, afin d'éviter le bouclage infini de la requête d'interruption.
- Lors d'une interruption de priorité haute, W, STATUS et BSR sont sauvegardés dans des registres dédiés.
  - Pour les restaurer en sortant de la routine d'interruption, utiliser l'instruction **RETFIE 1** ou **RETFIE FAST**.