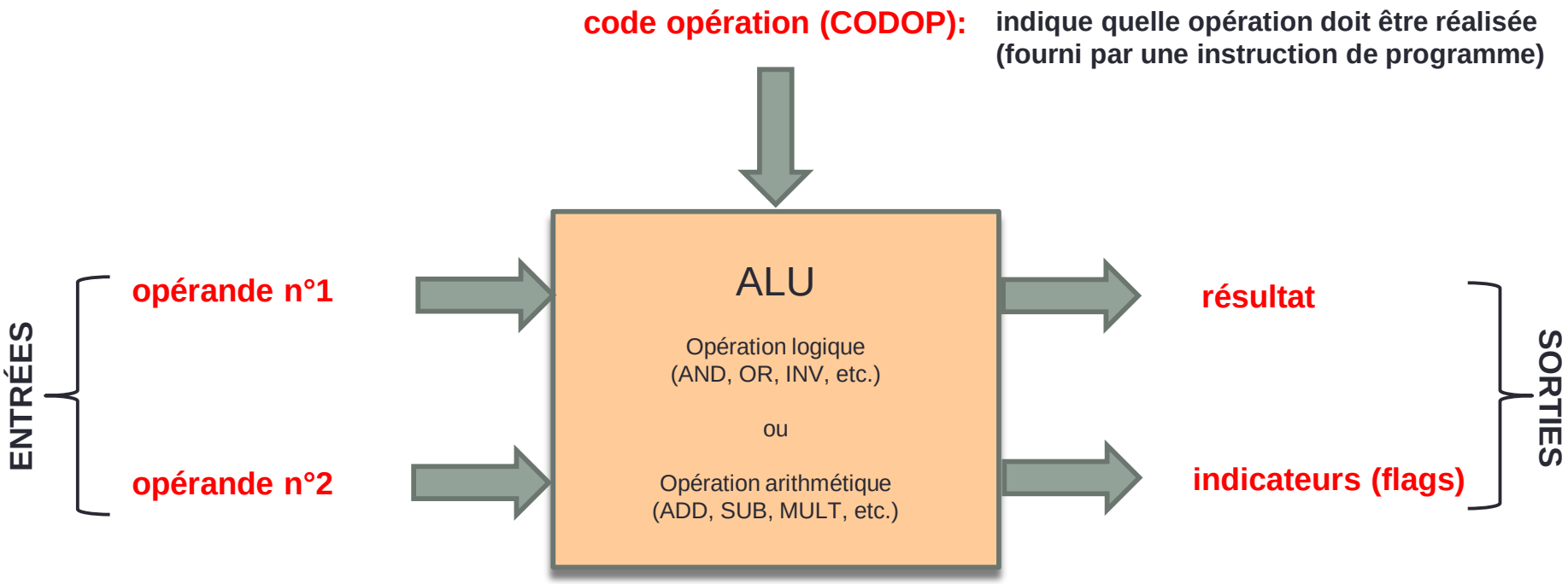
Architecture des microcontrôleurs

l'unité arithmétique et logique (ALU)

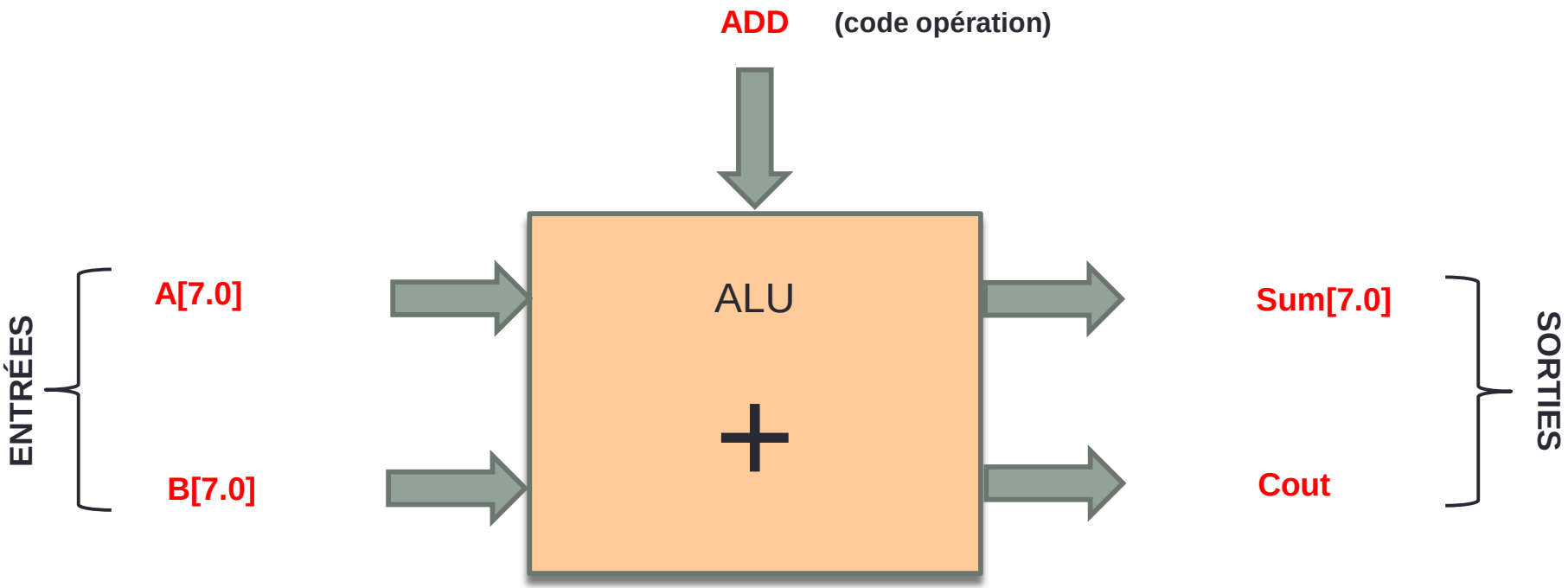
Sommaire

1. Notions générales

- L'unité arithmétique et logique (ALU) est un circuit capable d'appliquer une opération logique sur un ou deux opérandes.



Exemple: addition



Critères de performance

- Nombre de bits de l'ALU: largeur maximale des mots binaires traités en une opération

  Remarque: il est toujours possible de travailler avec des mots plus larges, mais plusieurs opérations successives seront nécessaires.

  Par exemple: l'addition de mots de 16 bits avec une ALU 8 bits est possible, mais nécessitera plusieurs opérations successives.

- Temps de traitement d'une opération

  Par exemple: 16 MIPS (Mega Instructions Per Second) au maximum sur un PIC18F

- Nombre d'ALU physiquement présentes

  Possibilité ou non de réaliser plusieurs opérations en parallèle
  (processeurs mono-cœurs / processeurs multi-cœurs)

- Diversité des opérations disponibles

  Par exemple: sur un PIC18F, ≈ 70 opérations
  opérations arithmétiques: Add, Substract, Multiply, Increment, Decrement, Negate, …
  opérations logiques: AND, OR, XOR, Complement, …

Programme

- Programme = suite d'instructions effectuées **séquentiellement** par le processeur

  En langage assembleur, 1 instruction = 1 commande directement compréhensible par le processeur

  Remarque: un programme écrit dans un langage « haut niveau » (exemple: en C) devra être « traduit » en langage « bas niveau » (assembleur) ↔ **COMPILATION**

- La liste des instructions est stockée dans la mémoire programme

  Après avoir exécuté une instruction, le processeur passe à la suivante, stockée juste après dans la mémoire (sauf en cas de branchement: CALL routine, GOTO label, etc.)

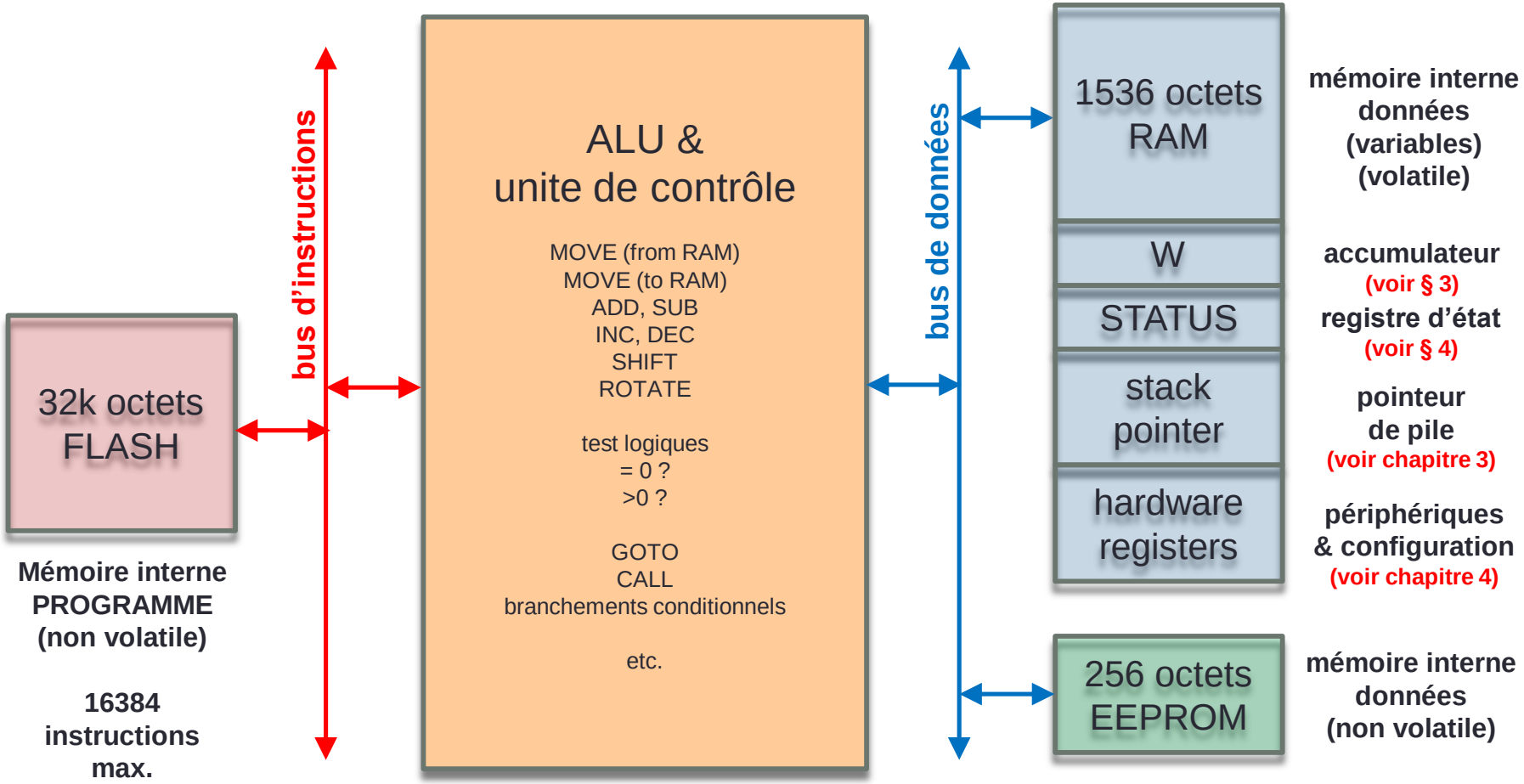
- Une instruction doit fournir plusieurs indications au processeur
  - que faut-il faire ?
  - avec quelles données ?
  - où placer le résultat ?
  - etc.

- Une instruction est codée sur « un certain nombre » de bits

  Par exemple: sur un PIC18F, 2 octets (16 bits) pour la majorité des instructions

2. ALU et registres(1) du PIC18F

Exemple du PIC18F45K20



(1) un registre = une case mémoire RAM

3. Un registre particulier: l'accumulateur

Accumulateur (ou Working Register, noté W ou WREG):

registre particulier qui contient (dans la plupart des cas) un opérande avant d'exécuter l'instruction, et qui peut contenir le résultat de l'opération

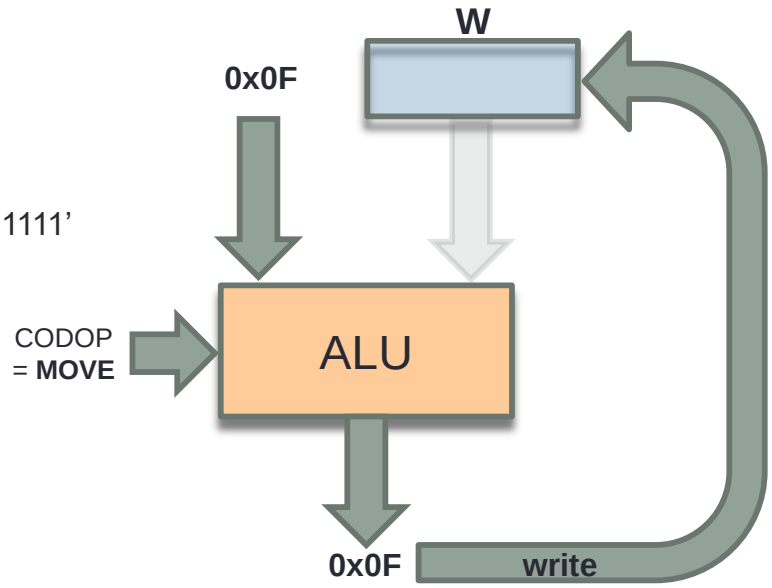
Exemple 1: **MOVLW 0x0F**

MOVE

literal

WREG = destination de l'opération

valeur
notations équivalentes: 0x0F
h'0F'
b'00001111'
d'15'



remarque:
instruction **littérale**:
l'instruction fournit directement la valeur de l'opérande

On parle également d'**adressage immédiat**
(abus de langage car l'instruction ne fournit pas une adresse mais une valeur !)

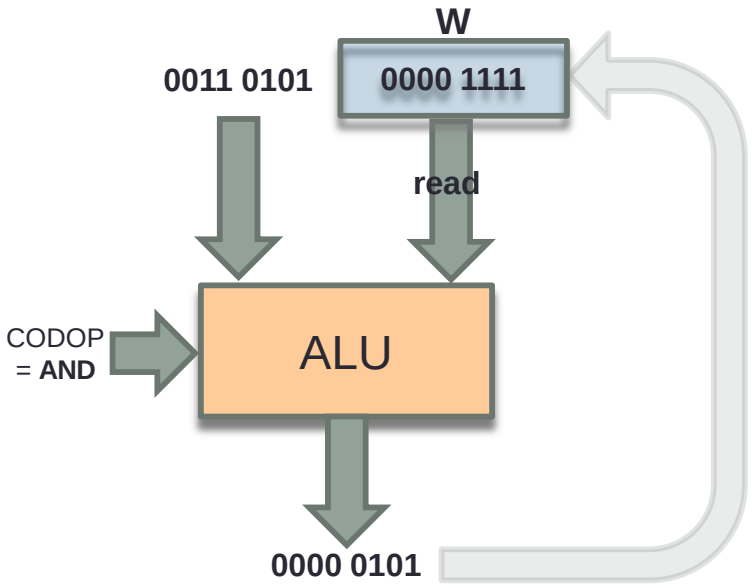
résultat:

Exemple 2: **ANDLW b'00110101'**

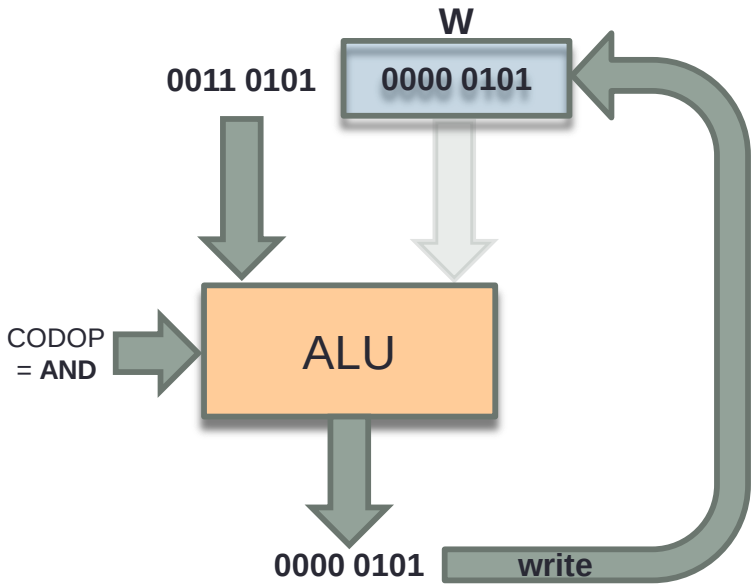
AND literal

WREG = source et destination de l'opération

D'abord:



Puis:

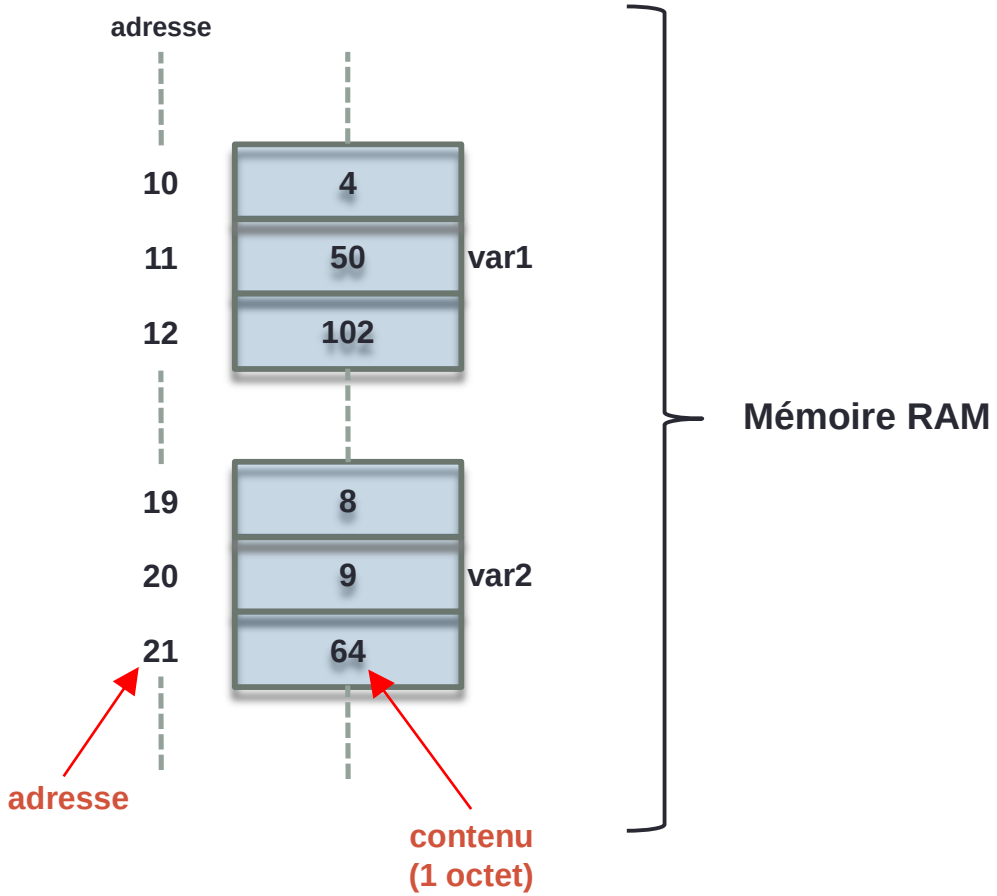


résultat:

Exemple 3: opération « complexe »
→ faire la somme de deux variables var1 et var2, stocker le résultat dans var1

s'écrit en en langage C: **var1 = var1 + var2;**

// var1 et var2 ayant au préalable été déclarées
// comme des variables de type uint8_t (8-bit non-signé)



1ère instruction:

MOVF var2, 0

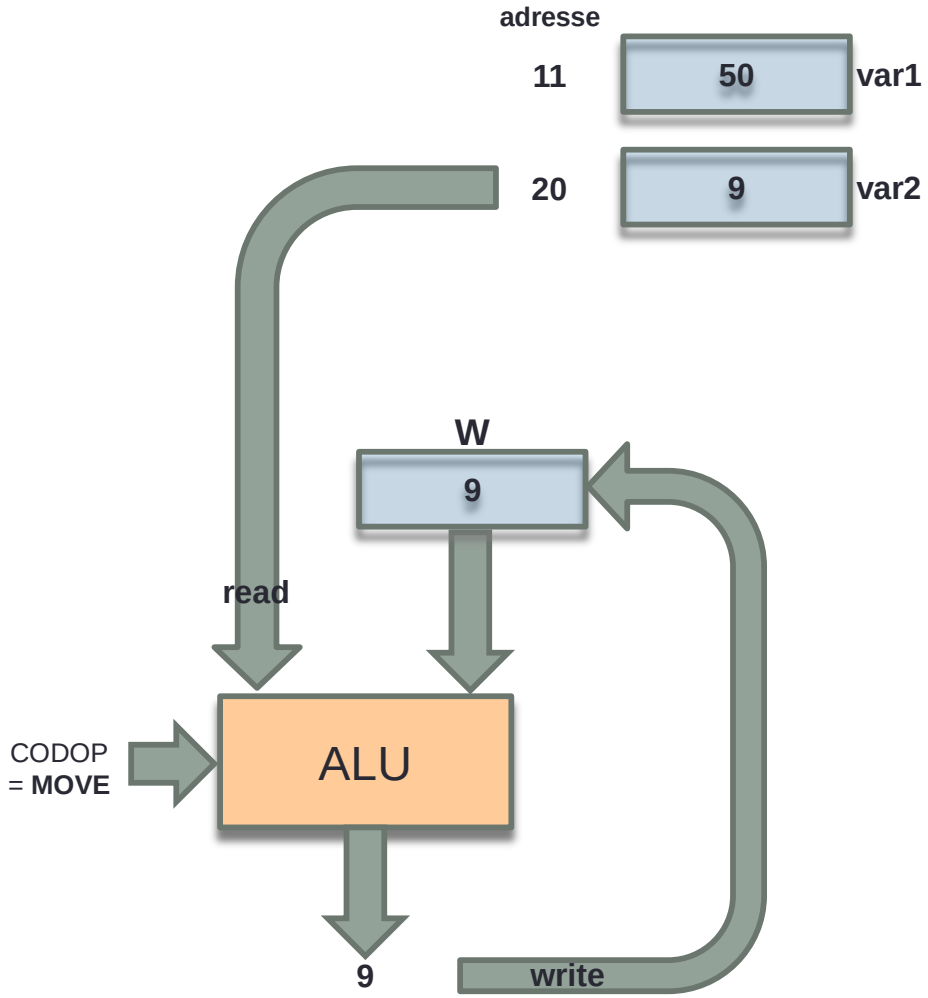
- MOVE
- « File register »
- adresse du registre à accéder
- WREG = destination de l'opération

adressage immédiat
on fournit l'adresse du registre à accéder

On peut donner le « nom » de la variable:
le compilateur le remplacera par son adresse

(le microcontrôleur ne « comprend » pas le mot « var2 », mais il « sait » quoi faire si on lui fournit une adresse).

Ici, on aurait pu remplacer **var2** par **20** dans l'instruction.
Le résultat aurait été le même.



résultat: l'accumulateur est chargé avec le contenu de var2

2ème instruction:

ADDWF var1, 1

ADD

1er opérande: W

2ème opérande: un registre

adresse du registre à accéder (1)

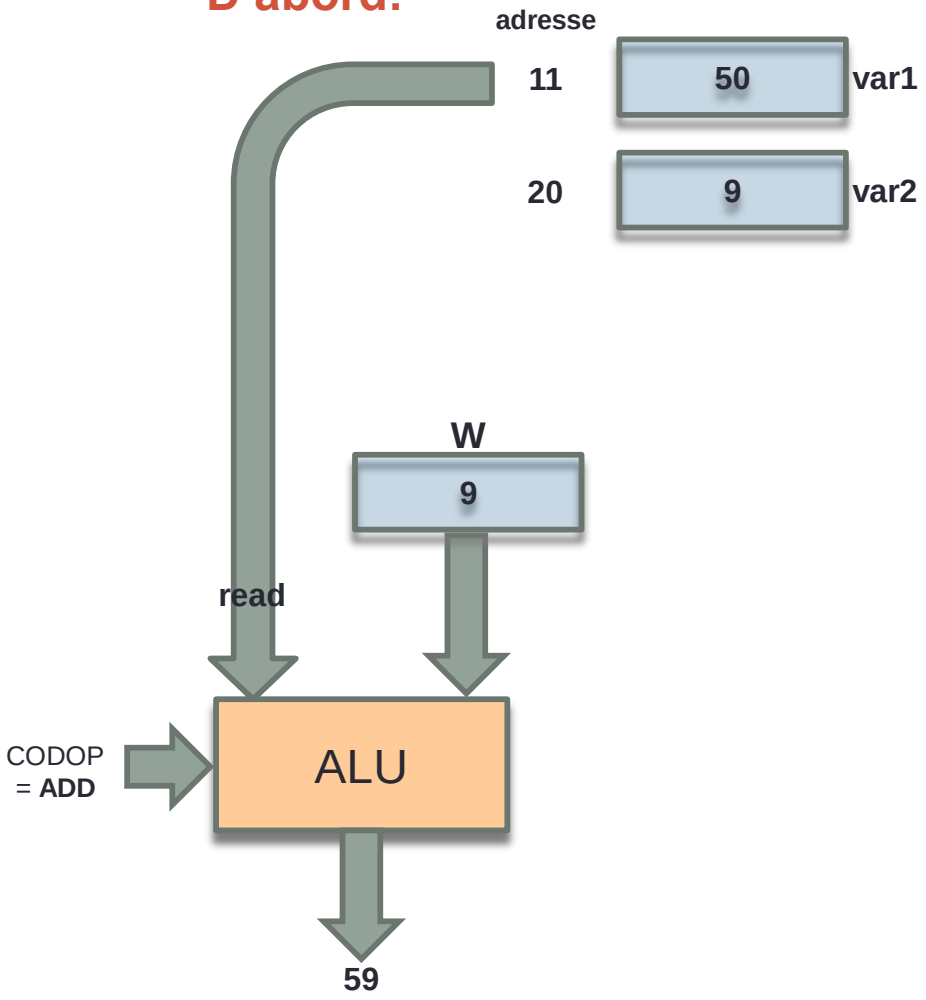
registre lu = destination de l'opération

adressage immédiat
on fournit l'adresse du registre à accéder

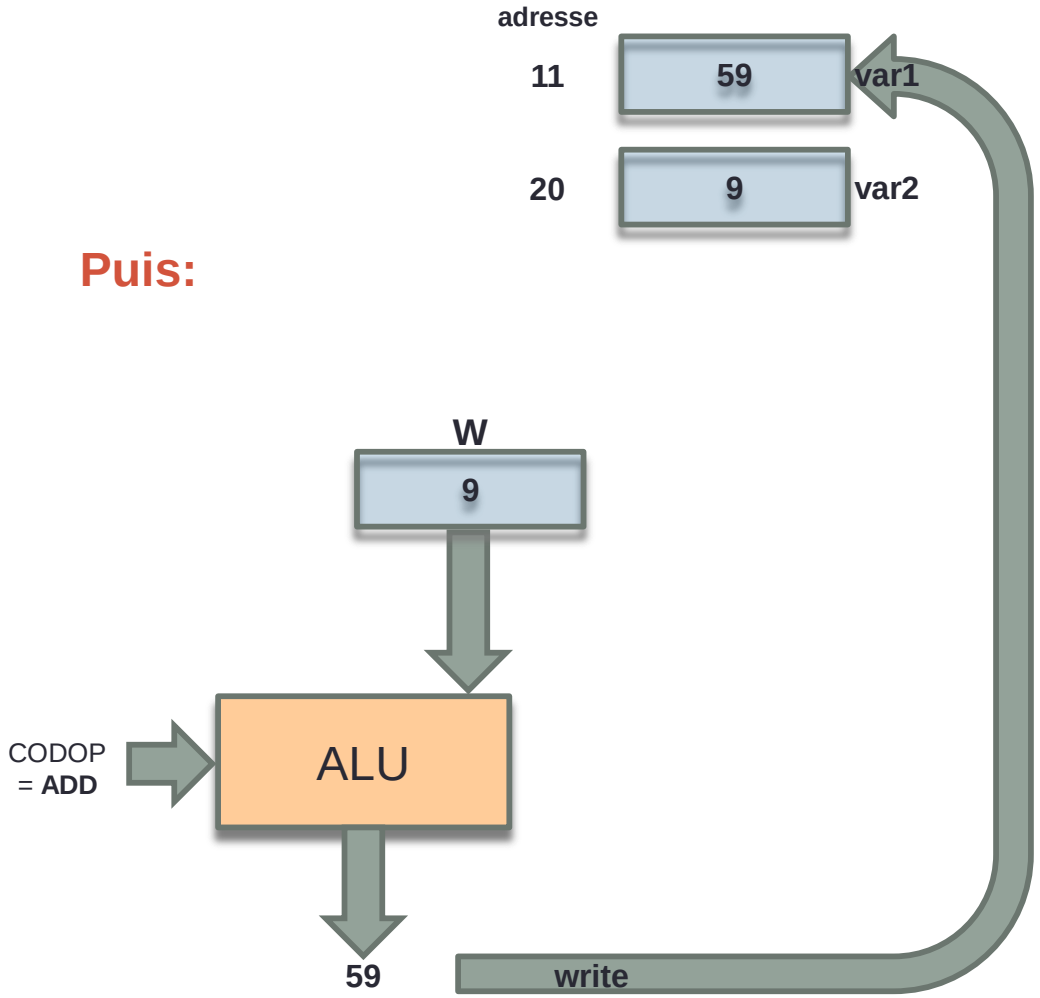
.

(1) Ici, on aurait pu remplacer **var1** par **11**.
Le résultat aurait été le même.

D'abord:



résultat: l'ALU calcule W + var1



remarque: W n'est pas modifié

Avec l'instruction **ADDWF var1, 0**
le résultat aurait été copié dans W.

Il aurait ensuite fallu copier le contenu
de W dans var1
(instruction **MOVWF var1**).

Indique la destination
0 ⇒ résultat dans W
1 ⇒ résultat dans registre de départ

résultat: le résultat de l'addition est copié dans var1

4. Le registre d'état

Registre d'état (ou Status Register, noté STATUS):

registre particulier qui contient l'ensemble des indicateurs (flags) mis à jour par l'instruction la plus récente (ces indicateurs peuvent entre autres être utilisés pour faire des branchements conditionnels)

par exemple:

REGISTER 5-2: STATUS: STATUS REGISTER

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	N	OV	Z	DC(1)	C(1)
bit 7							bit 0

Indique que le résultat de la dernière opération est **négatif** (MSB de l'ALU = 1)

Indique un dépassement (**overflow**) de dynamique pour les calculs arithmétiques signés (complément à 2) (< -128 ou > +127)

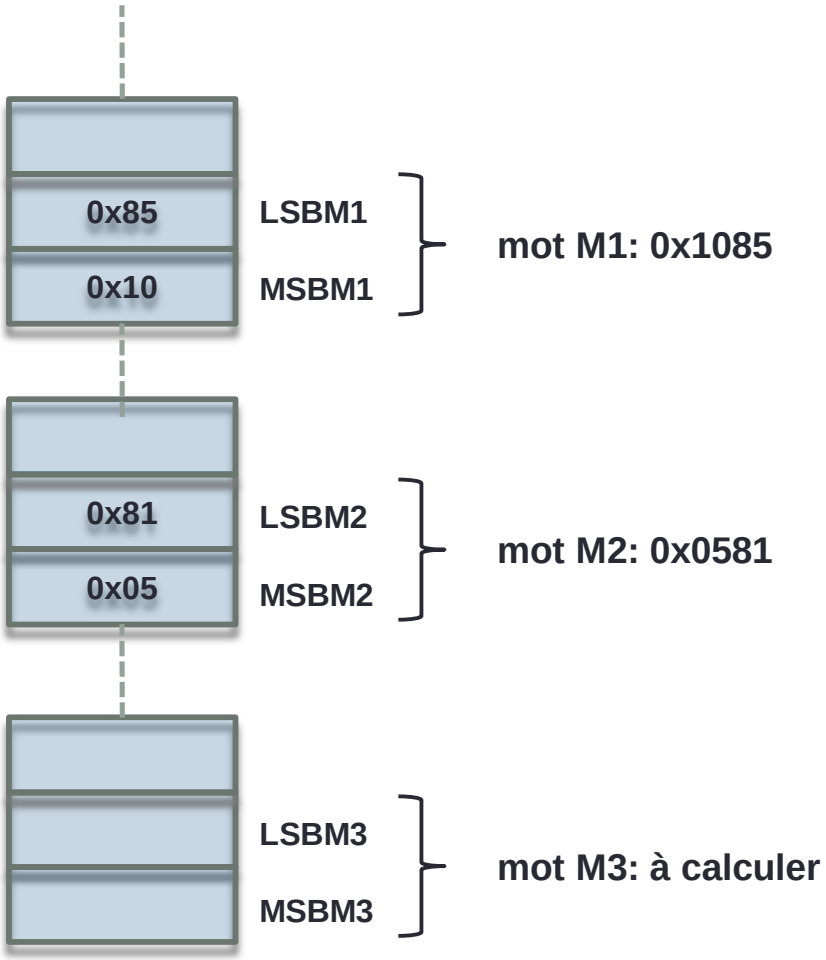
Indique que le résultat de la dernière opération est égal à **zéro**

Indique la valeur de la retenue du 4ème LSB (**digit carry**) de la dernière opération (utilisé pour le codage **B**inary **C**oded **D**ecimal)

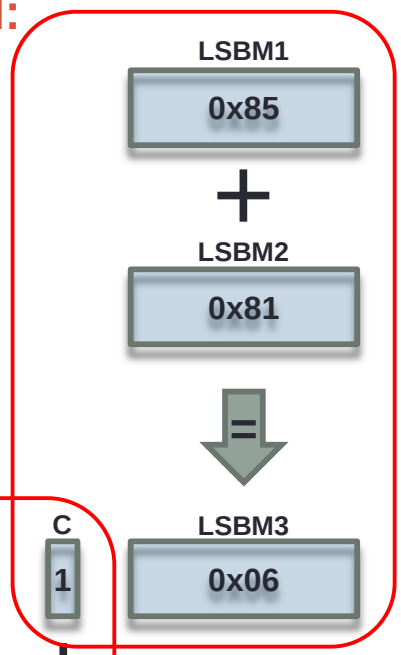
Indique la valeur de la retenue (**carry**) de la dernière opération

La retenue (carry) permet de réaliser des opérations sur des mots de plus de 8 bits.

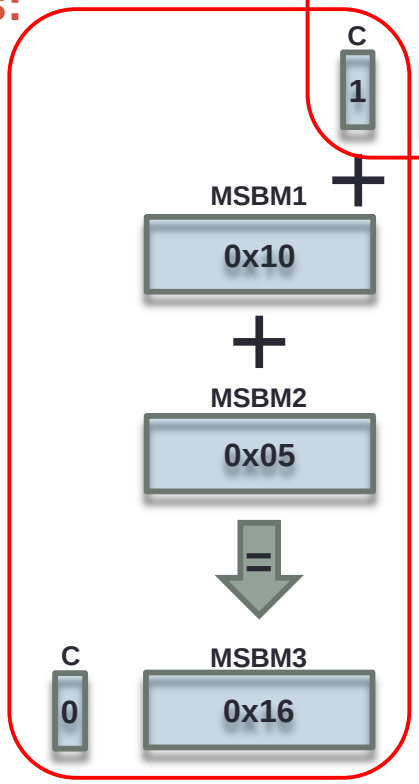
Exemple 4: addition de 2 mots de 16 bits (M1 et M2), résultat dans M3



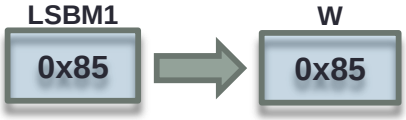
D'abord:



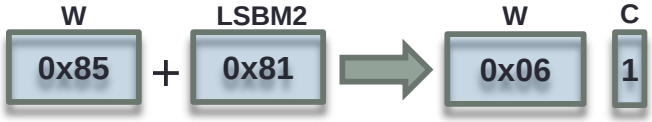
Puis:



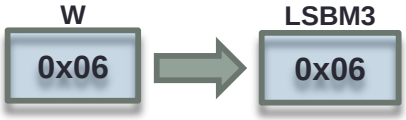
MOVF LSBM1, 0



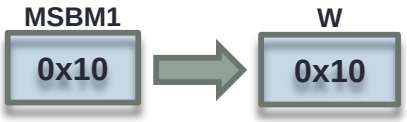
ADDWF LSBM2, 0



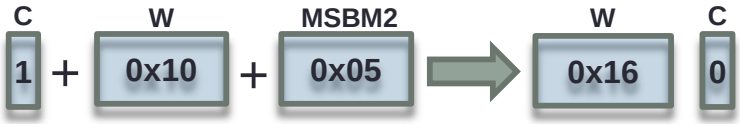
MOVWF LSBM3



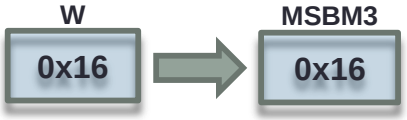
MOVF MSBM1, 0



ADDWFC MSBM2, 0



MOVWF MSBM3



Quelques remarques concernant l'exemple précédent:

❶ Il existe 2 instructions proches mais différentes pour l'addition: « ADD » et « ADD with Carry ».

❷ La « Carry » est utilisée et modifiée par certaines instructions.
(Elle est lue avant, et modifiée après.)

❸ L'opération s'écrit en une ligne en C.

```
M3 = M2 + M1;          // M1, M2 et M3 étant préalablement déclarées comme des variables
                       // de type uint16_t
```

Elle nécessite en réalité 6 instructions élémentaires.

Exemple 5: décalage d'un mot de 16 bits (M1) d'un bit vers la droite

instruction utilisée: **RRCF** registre, 1

(**R**otate **R**ight through **C**arry)

programme pour cet exemple: