

MICROCONTRÔLEUR PIC12F675 :

Dans la documentation qui suit, il est fait référence au datasheet contenant la totalité de la documentation liée au PIC12F675 ; Ce fichier est « **DS41190E.PDF** », il est fourni par Microchip.

GÉNÉRALITÉS :

Le PIC12F675 est un microcontrôleur, une unité de traitement de l'information (microprocesseur) à laquelle on a ajouté des périphériques internes (composants électroniques tels que les transistors, résistances etc etc...) permettant de dialoguer avec l'environnement extérieur sans nécessiter l'ajout d'autres composants externes.

JEU D'INSTRUCTIONS :

Les PICs sont des composants utilisant une technologie RISC (Reduced Instructions Set Computer), ce qui signifie que ce sont des composants exploitant un jeu d'instructions réduit. (Plus le nombre d'instructions est réduit, plus rapide en est le décodage ainsi que l'exécution du programme).

Le PIC12F675 dispose d'un jeu de 35 instructions au total.

La combinaison de ces instructions forment le programme (ou firmware) qui va piloter (contrôler, diriger...) le PIC.

Le jeu d'instructions est l'ensemble des mots qui permettent de faire exécuter le programme par le microcontrôleur.

Ex. « goto », « btfsc », « movlw » ...

(Le firmware - ou programme - va diriger le PIC : suivant l'état des entrées, des sorties, le programme commandera au PIC telle ou telle autre action...)

HORLOGE ET TEMPS DE CYCLE :

Ces instructions sont stockées dans un seul mot de programme, et s'exécutent, (sauf exceptions telle que l'instruction « goto » par exemple), en 1 cycle d'horloge.

L'horloge est le coeur du microcontrôleur, ses battements permette au PIC de cadencer les échanges de données et d'effectuer une synchronisation complète de tout l'ensemble de son système.

Les temps d'instruction (Nombre de clocks d'horloge pour décodage et exécuter une instruction) sont de l'ordre de 1 cycle.

L'horloge interne du PIC12F675 est prédivisée par 4. C'est cette base de temps qui donne la durée d'un cycle.

L'horloge interne est définie par un quartz de 4MHz ; on obtient donc une fréquence réelle de $4^{66}/4$ soit 1^{66} de cycles par seconde, et comme le PIC12F675 exécute 1 instruction par cycle, (hormis quelques instructions, comme l'instruction de saut « goto »), cela donne une puissance de l'ordre de 1 MIPS (1 Million d'Instructions Par Seconde).

En utilisant un quartz de 4Mhz, on obtient un temps de cycle d'une instruction de l'ordre de 1µs.

EXEMPLES :

« nop » : Pour exécuter l'instruction : 1 cycle d'horloge soit 1µs lors de l'utilisation du quartz interne de 4Mhz.

« movlw .12 » : Exécution 1 cycle, soit 1µs.

Exceptions : « goto label » : exécution 2 cycles soit 2µs. (d'autres instructions, mais peu nombreuses, consomment 2 cycles d'horloges pour leur décodage et leur exécution : « call label », « return »...)

NOTES SUR L'HORLOGE :

On peut adjoindre au PIC12F675 une horloge externe, dans ce cas celle-ci peut atteindre les 20Mhz, soit un temps de 200ns par cycle ! Mais l'inconvénient majeur est la condamnation des deux entrées/sorties GP4 et GP5 qui doivent alors être obligatoirement utilisées pour la connexion du quartz externe.

On peut remarquer aussi que l'horloge externe d'un PIC peut être abaissée jusqu'à l'arrêt complet sans perte de données ni de fonctionnement erratique ! Ce qui explique que l'on puisse adjoindre des quartz de toute valeurs ou bien même un système d'oscillation très simple à base de résistance et de condensateur.

UTILISATION DE L'HORLOGE INTERNE :

L'horloge interne du PIC12F675 est de 4Mhz, mais, celle-ci doit être calibrée à la mise sous tension du microcontrôleur, c'est une particularité de ce type de PIC, car tous n'ont pas besoin de calibrer leur horloge interne.

L'avantage réside dans le fait que les 2 entrées/sorties GP4 et GP5 sont alors libres et peuvent être utilisées comme on le désire.

Notes avancées sur la calibration de l'horloge interne du PIC :

(Page 54 du datasheet).

Noter que ces remarques importent peu lors de l'exploitation d'un PIC avec un quartz externe, elles ne sont à connaître que pour l'utilisation du PIC avec son horloge interne.

La calibration de l'horloge s'effectue par l'écriture d'une valeur précise (et différente selon les modèles de PIC12F675) dans un registre dédié (Ce registre est nommé « OSCCAL »).

La valeur à copier dans le registre de calibration est stocké dans la dernière adresse de la mémoire programme du PIC.

Cette valeur qui est très importante, n'est pas effacée par les programmeurs de PICs (PICSTART, PICKIT1...), elle ne peut-être que lue par le programmeur et par le firmware du PIC.

Une façon de régler ce problème de calibration est d'effectuer un saut à l'adresse où se trouve la valeur de l'octet de calibration et de copier cette valeur dans le registre adéquat, et ce, en début de programme pour ne plus s'en préoccuper par la suite lors de l'élaboration du programme.

LES DIFFÉRENTES FAMILLES DES PICs :

La famille des PIC est divisée selon plusieurs critères : microcontrôleurs 8 bits, 16 bits, 32bits... ou bien encore, selon 3 familles : La famille « Base-Line », qui utilise des mots d'instructions de 12 bits, la famille « Mid-Range », qui utilise des mots de 14 bits, et la famille « High-End », qui utilise des mots de 16 bits.

Le PIC12F675 fait partie de la famille des « Mid-Range », celle qui utilise des mots de 14 bits.

Un mot d'instruction d'une largeur de 14 bits correspond au nombre de bits qui sont nécessaires au PIC pour coder une instruction de programme.

IDENTIFICATION D'UN PIC :

Ex. PIC12F675 I/P

Les 2 premiers chiffres indiquent la famille du PIC : « 12 », « 16 », « 18 » ...

Suit une lettre :

« C » indique que la mémoire programme est une EPROM.

« CR » pour indiquer une mémoire de type ROM.

« F » pour indiquer une mémoire de type FLASH.

Les versions « C » peuvent être effacés (suivant le cas) aux ultraviolet.

Les versions « CR » ne sont pas effaçables.

Un composant qu'on ne peut reprogrammer est aussi appelé O.T.P. pour One Time Programming.

Les versions « F » sont reprogrammables électriquement, via un programmeur de PIC (PICSTART, PICKIT...)

Suit les chiffres composants la désignation du PIC :

« 675 », « 84 », « 628 » etc etc...

Pour finir, on peut trouver en fin de référence un suffixe « -XX » dans lequel XX représente la fréquence d'horloge maximale que le PIC peut utiliser.

Ex.

- 04 pour un quartz de 4MHz.

- 10 pour un quartz de 10MHz.

- 20 pour un quartz de 20MHz.

D'autres informations peuvent être imprimées à la suite de la référence du PIC, concernant les dates de fabrications, les numéros de lots...

Une information importante est la lecture de la lettre « P » ou bien des lettres « PA » qui informent que le pic utilisé est fabriqué en format « PDIP » - format circuit intégré au pas de 2.54mm entre pattes du composant (pins).

De même, les lettres suivantes, « SN » ou bien « OA », désignent un PIC fabriqué en format réduit (Boîtier SOIC), au pas de 1.27mm entre pins.

ORGANISATION DE LA MÉMOIRE DU PIC12F675 :

(Page 7 du datasheet).

La mémoire du PIC12F675 est divisée en 3 parties : la mémoire programme (endroit où se situ le programme ou firmware du PIC), la mémoire ram (endroit où se trouve les données : contrôles de boucles, sauvegarde des registres etc... Données perdues une fois l'alimentation du PIC coupée) et enfin la mémoire eeprom : mémoire où se trouvent aussi des données mais qui ne sont pas perdues après une nouvelle mise sous tension !

La figure 2-1 de la page 7 montre l'organisation interne d'un PIC12F675.

LA MÉMOIRE PROGRAMME :

La mémoire programme est constituée de 1k mots (1024 instructions possible) de 14 bits.

C'est dans cette zone que se trouve le programme de contrôle (ou firmware) du PIC.

La largeur du bus programme est de 14 bits, ce qui explique qu'il faille 2 octets (2 bytes en anglais), pour coder une instruction de 14 bits.

Lorsqu'on lit le programme d'un PIC12F675 vierge (non programmé), par un programmeur (PICSTART, PICKIT...), on ne lit que des suites de mots égaux à 0x3FFF (Le suffixe 0x indique que la notation qui suit est effectuée en hexadécimale) ; La valeur 0x3FFF correspond à la valeur binaire B'111111111111', soit 14 bits. (Le suffixe B'xxx' indique que le contenu entre quote est défini en binaire).

Notes :

les vecteurs d'interruptions, l'adresse de l'octet de configuration se trouvent dans l'espace de la mémoire programme.

Dans cet espace mémoire, on stockera aussi les tableaux de données constantes.

LA MÉMOIRE RAM :

(Page 8 du datasheet).

La mémoire RAM (Random Access Memory) est celle où l'on stocke toutes les variables du programme.

Noter que toutes les données qui y sont stockées sont perdues lors d'une coupure de l'alimentation du PIC.

La mémoire RAM est organisée en 2 banques : (banque #0 - bank 0 - et banque #1 - bank 1 -)

Ces deux banques de ram sont aussi divisées en deux parties distinctes.

La première partie comprend les registres spéciaux (Special function register). Ces registres spéciaux permettent de contrôler le microcontrôleur.

Ex. Registre 16 bits de comptage du Timer 1 appelés « TMR1L » et « TMR1H ».

Ces emplacements mémoires ne peuvent pas être utilisés autrement que pour gérer les registres « TMR1L » et « TMR1H » !

La seconde partie comprend les registres à utilisation générale : (General Purpose Registers).

Ces emplacements mémoires peuvent être librement utilisés par le programme de contrôle (firmware) du PIC.

Ce sont les adresses des variables globales, locales, des buffers de données etc...

Dans la banque #0 (bank 0), on dispose de 64 octets de libre à partir de l'adresse 0x20 jusqu'à l'adresse 0x5F.

Dans la banque #1 (bank 1), on remarque d'autres registres spéciaux (« TRISIO », « OSCCAL » etc...) ainsi qu'un espace d'octets libre ; mais attention ! Ces octets libres sont ceux déjà définis dans la banque #0 ! (L'indication « **accesses 20h-5Fh** » indique qu'accéder à ces 64 octets depuis la banque #1 donne en fait accès aux cases mémoires déjà définies en banque #0).

Notes :

On peut remarquer que certains registres (« STATUS », « FSR », etc...) sont définis à la fois dans l'espace mémoire de la banque #0 et de la banque #1 ; Cela signifie qu'ils sont accessibles quelle que soit la banque sur laquelle le programme du PIC est positionné.

La banque #0 commence à l'adresse mémoire 0x00 et finit à l'adresse 0x7F, tandis que la banque #1 commence à l'adresse 0x80 et se termine à l'adresse 0xFF.

Il faut aussi prendre en compte l'espace mémoire non alloué en fin de banque #0 et #1 (Adresses [0x60 à 0x7F] pour la banque #0 et [0xE0 à 0xFF] pour la banque #1) ; L'accès à cet espace non alloué en lecture renverra une valeur 0x00 pour le contenu de chaque adresse et l'écriture d'une valeur sur ces emplacements ne sera pas effectuée.

On peut déjà noter que l'accès, la sélection de l'une ou l'autre de ces banques, s'effectue par un bit particulier du registre « STATUS », intitulé RP0 ; Suivant l'état de ce bit, le firmware contrôlera l'accès à la banque #0 (RP0 = « 0 ») ou bien la banque #1 (RP0 = « 1 »).

Notes avancées sur une ram particulière : la pile.

(Page 7 du datasheet).

La pile est une partie de la mémoire ram particulière : elle n'est accessible que de façon interne par le microcontrôleur et non pas accessible directement par le firmware.

La pile permet de stocker les adresses de retour lors des appels aux sous-programmes.

Le niveau de la pile (nombre d'adresses de retour pouvant être stocké avant dépassement des possibilités de stockage appelé aussi overflow de la pile) est de 8.

Aucune indication du niveau d'entassement de la pile n'est fournie (entassement peut s'intituler empilement), c'est à la charge du programmeur qu'il incombe de respecter les 8 niveaux de pile pour éviter un overflow et un fonctionnement erratique du programme.

La pile ne permet de stocker que les adresses de retours des sous-programmes : Il est impossible d'exécuter des instructions de type « PUSH » ou bien « POP » pour stocker des données dans cet espace mémoire réservé !

Ex. d'un appel à un sous-programme appelant lui-aussi un autre sous-programme : 2 niveaux de pile utilisés sur les 8 disponibles.

```

_START:
    call    Mon_Sous_Programme      ; Premier niveau de pile utilisé.
    ...
    goto   _START

; Appel de sous-programme imbriqué :
Mon_Sous_Programme:
    call   Mon_Deuxieme_Sous_Programme ; Deuxième niveau de pile utilisé.
    ...
    return

Mon_Deuxieme_Sous_Programme:
    ...
    return
    
```

LA MÉMOIRE EEPROM :

(Page 47 du datasheet).

La mémoire EEPROM (Electrical Erasable Programmable Read Only Memory), est constituée de 128 octets.

Ces 128 octets peuvent être lus et écrits par le firmware du PIC durant son exécution ou bien lors de la création de votre programme, en stockant des données sur 8 bits à l'adresse 0x2100.

Ces données sont conservées après coupure de l'alimentation du PIC.

Des routines de lecture et d'écriture spécifiques doivent être employées pour accéder à ces données (durant l'exécution du programme pilotant le PIC) pour limiter au maximum les risques d'erreurs.

L'accès aux données de l'eprom, que ce soit en lecture ou bien en écriture est réglementé par des registres spécifiques.

Le temps de lecture des données de l'eprom est plus rapide que le temps d'écriture. Ce comportement est similaire à celui des eeproms séries 24Cxx.

ORGANISATION DES INSTRUCTIONS DE CONTRÔLE DU PIC :

(Page 69 du datasheet).

A partir de cette page se trouvent les explications nécessaires pour la compréhension de toutes les instructions de contrôle du PIC. Dans l'encadré grisé, on trouve 2 instructions obsolètes à ne plus utiliser : « OPTION » et « TRISIO ».

Il existe 3 types d'instructions : les instructions orientées *opération sur les octets* « Byte-oriented operations », les instructions orientées sur les bits « Bit-oriented operations » et les instructions à usage général « Literal and control operations ».

LES INSTRUCTIONS ORIENTÉES OCTET :

Ce sont des instructions qui manipulent les données sous forme d'octets.

Elles sont codées de la manière suivante :

- **Bits n°0 à 6** : Ces 7 bits permettent de définir l'adresse de l'opérande (File). Ces bits permettent l'accès aux adresses de la ram suivante : 0x00 à 0x7F. En conjugaison avec le bit « RP0 » du registre « STATUS », on peut accéder aux adresses 0x80 à 0xFF de la ram. (« STATUS.RP0 » permettant de sélectionner l'une ou l'autre des banques mémoire disponible).
- **Bit n° 7** : 1 bit de destination (intitulé « d ») pour indiquer si le résultat obtenu doit être conservé dans le registre de travail (registre « W » - Working register ou registre de travail soit l'accumulateur) ou placé dans l'opérande (« F » pour « File »).
- **Bits n°13 à 8** : 6 bits pour réaliser le codage de l'instruction. (Ainsi le nombre maximal d'instructions possible est de : B'111111' soit 63 instructions, en réalité il n'existe que 35 instructions utilisées).

LES INSTRUCTIONS ORIENTÉES BITS :

Ce sont des instructions destinées à manipuler directement des bits d'un registre particulier.

Elles sont codées de la manière suivante :

- **Bits n°0 à 6** : 7 bits pour définir l'adresse de l'opérande.
- **Bits n°7 à 9** : 3 bits pour indiquer le numéro du bit à manipuler (la valeur 0 correspondant au bit n°0, et ainsi de suite jusqu'à la valeur 7 qui indique l'utilisation du bit n°7).
- **Bits n°10 à 13** : 4 bits pour différencier les instructions existantes.

LES INSTRUCTIONS À USAGE GÉNÉRAL :

Ce sont les instructions qui manipulent des données qui sont codées dans l'instruction directement.

Elles sont codées de la manière suivante :

- **Bits n°0 à 7** : Codage de la valeur immédiate sur 8 bits (Ce qui permet d'exploiter une donnée sur la plage des valeurs de 0 à 255).
- **Bits n°8 à 13** : Codage de l'instruction sur 6 bits.

LES SAUTS INCONDITIONNELS ET APPELS DE SOUS-ROUTINES :

Les dernières instructions permettent de contrôler de façon arbitraire le déroulement du programme.

Elles sont codées de la manière suivante :

- **Bits n°0 à 10** : Adresse de la destination du saut codée sur 11 bits.
- **Bits n°11 à 13** : Codage de l'instruction sur 3 bits

L'adresse de saut codée sur 11 bits permet au programme de sauter à tout emplacement de la mémoire programme compris entre les adresses B'0000000000' et B'1111111111' soit 0x0000 et 0x07FF.

Les adresses valides pour le PIC12F675 appartiennent à l'espace mémoire : 0x0000 à 0x03FF.

Il est à noter que suivant le type de PIC, la mémoire programme peut exister au delà de la limite de l'adresse 0x07FF.

De la façon dont l'instruction est codée, on s'aperçoit qu'il est impossible d'accéder à l'espace supérieur directement par une instruction « goto » ou bien « call » ! La solution apportée, pour l'accès à cet espace supérieur, est d'utiliser la combinaison du registre « PCLATH » avec l'adresse effective.

(Ce qui n'est pas nécessaire pour le PIC12F675).

APERÇU DES INSTRUCTIONS DU PIC12F675 :

(Page 70 du datasheet).

Une instruction permet de faire exécuter une action au microcontrôleur.

Le fichier source assembleur contient la somme des instructions qui, une fois traduites en code machine par un logiciel d'assemblage, seront programmés dans la mémoire du PIC et le dirigerons.

Le logiciel d'assemblage à exploiter est « MPLAB », édité et fournit gratuitement par Microchip (www.microchip.com).

Il permet :

- L'édition complète de fichiers sources assembleur ;
- L'assemblage de ces fichiers : traduction du fichier en code machine compréhensible par le microcontrôleur ;
- La programmation des microcontrôleurs : lecture/écriture du code machine dans la mémoire du microcontrôleur.

A la page 70 du datasheet se trouve le tableau récapitulatif de toutes les instructions de contrôle du PIC.

PREMIÈRE COLONNE :

La première colonne indique le mnémonique et les opérandes pour chaque opération.

Les mnémoniques sont des mots réservés (A n'utiliser que pour cet usage) compris et interprétés par le programme d'assemblage.

Une définition de mnémonique fournie par Wikipedia (fr.wikipedia.org) :

« *Mnémoniques en langage assembleur :*

En langage assembleur, un opcode, composé de chiffres, indique une opération à effectuer. Programmer en langage binaire est un exercice difficile, car il faut saisir un nombre pour chaque opération à effectuer et le programmeur doit consulter un tableau de correspondance s'il ne se souvient pas de l'opcode. Cette recherche prend du temps et une saisie mal faite introduit un bogue. En conséquence, un ensemble de mnémoniques fut créé. Chaque opcode est représenté par un mot composé de 1 à 5 lettres. Par exemple, il suffit de saisir « add » plutôt que l'opcode qui correspond à l'addition. Le rappel de ces mots est nettement plus facile que celui des opcodes.

Ce type de mnémonique diffère des autres, car il ne facilite pas le rappel des nombres, il élimine le besoin de s'en souvenir, tout simplement. »

La composition d'une instruction est la suivante :

- Etiquette ou label (non obligatoire).
- Un ou plusieurs espaces, une ou plusieurs tabulations.
- Le mnémonique (en majuscules ou minuscules indifféremment).
- Un ou plusieurs espaces, une ou plusieurs tabulations.
- Opérande ou valeur.
- Virgule éventuelle de séparation.
- Bit de destination W (Accumulateur) ou F (File : une variable) ou bien encore un numéro de bit de 0 à 7 si nécessaire
- Un ou plusieurs espaces, une ou plusieurs tabulations.
- Un point-virgule. (facultatif si pas de commentaire)
- Commentaires. (facultatif)

Le mnémonique ne peut pas se trouver en première colonne.

Tout ce qui suit le point-virgule est ignoré de l'assembleur (La zone de commentaire est toujours définie par un « ; » avant le début du commentaire).

La première colonne est réservée pour les étiquettes ou labels.

On a la possibilité d'insérer un ou plusieurs espaces/tabulations de chaque côté de la virgule. (Il est conseillé de séparer les mnémoniques des opérandes par des tabulations plutôt que d'utiliser des espaces ; Ainsi, s'il y a changement d'éditeur de texte pour créer les fichiers assembleurs, la mise en colonnes des instructions sera similaire).

La coloration syntaxique de l'éditeur de MPLAB permet de repérer très rapidement les différentes fonctions du code (commentaires, labels, mnémoniques, opérandes...)

```

; Exemple :
; Cette ligne commence par un « ; », c'est donc une ligne de commentaires.
_Ceci_Est_Un_Label:           ; Label ou étiquette.
    movf   O_Variable, W    ; Cette ligne est un commentaire
    btfsc  STATUS, Z
    goto   _Ceci_Est_Un_Label
    
```

SECONDE COLONNE :

Cette colonne du tableau donne un descriptif succinct de l'instruction.

TROISIÈME COLONNE :

Cette colonne donne le nombre de cycles nécessaires pour exécuter chaque instruction.

Le cycle machine est le temps (en unité de seconde) qui est nécessaire au microcontrôleur pour lire le code machine de l'instruction, le décoder et l'exécuter ; Ce temps est directement lié à l'horloge (quartz) qui est fourni au PIC.

Ex. Pour le quartz interne de 4Mhz, le temps de cycle est de 1µs. Lié à la formule :

$$T = \left(\frac{1}{\frac{4000000}{4}} \right)$$

La quasi-totalité des instructions nécessitent un seul cycle d'horloge pour être décodée puis exécutée. Les exceptions sont les instructions de sauts qui nécessitent 2 cycles, ainsi que les opérations de tests avec saut, lorsque le résultat du test engendre un saut (Instructions notées « 1(2) »).

QUATRIÈME COLONNE :

La 4ème colonne contient l'opcode : la traduction binaire effectuée par le logiciel MPLAB du mnémonique.

Définition de l'opcode fournie par Wikipedia (fr.wikipedia.org) :

« Chaque instruction est caractérisée par un numéro appelé opcode ou code opération. Ainsi, une instruction est simplement un groupement de bits -- différentes combinaisons correspondent à différentes commandes à la machine. La traduction la plus lisible du langage machine est appelé langage assembleur, qui est une traduction de chaque groupe de bits de l'instruction. Par exemple, pour les ordinateurs d'architecture x86, l'opcode 0x6A correspond à l'instruction push et l'opcode 0x74 à je (jump if equal). »

Cette colonne est fournie plus à titre d'informations que d'utilisation.

En effet, il s'avère très difficile de programmer le microcontrôleur directement en opcodes (langage machine pur), d'autant plus que MPLAB effectue lui-même la traduction des fichiers codes sources en langage machine.

CINQUIÈME COLONNE :

Cette colonne fournit les indicateurs d'états (Les bits « C » - carry - « Z » zéro ... du registre « STATUS ») une fois l'instruction effectuée ; Les indicateurs sont affectés *après* l'exécution de l'instruction.

Les indicateurs d'état :

(Page 11 du datasheet).

Ces indicateurs sont primordiaux pour le déroulement du programme qui contrôle le PIC.

Tous les indicateurs sont des bits du registre STATUS.

Attention :

- Toutes les instructions ne modifient pas l'état des indicateurs d'état !
- Certaines instructions modifient l'état de « Z », mais pas l'état de « C » et vice-versa !

L'indicateur d'état « Z » (Appelé aussi « flag Z » pour drapeau Zéro) :

C'est l'indicateur « Zéro », il fonctionne de la manière suivante :

Si le résultat d'une opération pour laquelle il est affecté, donne un résultat nul (égal à 0), l'indicateur zéro passe à « 1 » ; A l'inverse, si le résultat d'une opération est non nul (différent de 0), alors le flag « Z » passe à « 0 ».

Conclusion :

- Si STATUS.Z = 1 alors le résultat de l'opération précédemment effectuée est nul.
- Si STATUS.Z = 0 alors le résultat de l'opération précédemment effectuée est différent de 0.

Ex.

```

clrfsf    O_Nulle           ; La variable O_Nulle <-- 0

movlw    .10
movwf    O_Non_Nulle       ; La variable O_Non_Nulle <-- 10

movf     O_Nulle, W        ; O_Nulle = 0 ?
btfss   STATUS, Z         ; STATUS.Z = 1 si O_Nulle égal 0
goto    _Valeurs_Non_Nulles ; Non, saut...

movf     O_Non_Nulle, W    ;
btfss   STATUS, Z         ; STATUS.Z = (O_Nulle == 0)
goto    _Valeurs_Non_Nulles ; STATUS.Z = 0 soit O_Nulle != 0
_Valeurs_Nulles:
...

_Valeurs_Non_Nulles:
...                               ; Saut ici puisque (O_Nulle == 0) & (O_Non_Nulle !=0)
    
```

L'indicateur d'état « C » (Appelée aussi « Carry flag ») :

C'est l'indicateur pour Carry (report). Si le résultat d'une opération entraîne un débordement (overflow en anglais), le bit C sera positionné à l'état « 1 ».

Ce bit fonctionne comme un 9ème bit sur l'opération.

Ex.

On effectue l'opération suivante :

```

    B'11111101'           ; 0xFD soit .253 en décimal.
  + B'00000011'           ; 0x03 soit .3 en décimal.
  =====
    
```

Résultat : B'1 0000000' ; 0x100 soit .256, codé sur 9 bits.

Les registres du microcontrôleur sont des registres de 8 bits, le résultat obtenu de cette opération sera de « 0 » dans l'accumulateur, l'overflow (dépassement de résultat) sera stocké dans le bit « C » du registre « STATUS ».

STATUS.C sera égal à « 1 ».

Le résultat final sera donc de 256.

Comme le résultat de l'opération 8 bits est nul dans l'accumulateur, on trouve aussi le bit STATUS.Z fixé à « 1 ».

SIXIÈME ET DERNIÈRE COLONNE :

Ce sont les notes de bas de page.

La note 1 fait allusion à la méthode « lecture/modification/écriture » propre aux ports d'entrées/sortie.

(Voir les fichiers sources pour le détail de cette note).

DOCUMENT PRELIMINAIRE - (C) ELECTROME