CH-1092 Belmont, Switzerland
info@didel.com, www.didel.com

PIC*Génial* - Microcontrôleur PIC 16F84/16F870

Introduction à la programmation avec le PIC*Génial* - suite

4. Arithmétique, tables et musique

4.1. Nombres

Les opérations sur des nombres ne sont pas faciles, même en 8 bits. Si l'application demande des calculs compliqués que l'on ne peut pas approximer avec des tables, le choix d'un PIC est inadéquat.

4.2. Opérations logiques

Le ET logique (instruction AND) est utile pour masquer, c'est-à-dire mettre à zéro, quelques bits dans un mot. Par exemple, pour transformer les code Ascii d'un chiffre entre 0 et 9 dans son équivalant binaire, il suffit de masquer les 4 bits de bits fort:

```
And     #2'00001111,W
```

On aurait aussi pu écrire

```
And  # 2'1111,W  ou  And  #16'F,W  ou  And  #15,W
```

mais les défauts de ces notations sont évidente.

Si c'est une variable qui doit être masquée, il y a 2 cas à distinguer. Si la variable ne doit pas être modifiée, on écrira:

```
Move    Var,W                    ou    Move    #Mask,W
And     #Mask,W                        And     Var,W
```

Si la variable doit être mise à jour, on a aussi deux écritures possibles:

```
Move    Var,W                    ou    Move    #Mask,W
And     #Mask,W                        And     W,Var
Move    W,Var
```

Le OR (ou logique) permet de forcer à un les bits en face du masque. Dans le programme Picg0 de la page Pour éteindre les LEDs après avoir lu le port C dans le programme PicgT (page 8), on peut insérer l'instruction

```
Or      #2'11111100,W
```

Le XOR inverse les bits correspondant au masque. Si l'on a par exemple deux LEDs en RB7 et RB2 que l'on veut les faire clignoter sans toucher les autres bits du port, on écrira:

```
Move    PortB,W
Xor     #(2**bLed1+2**bLed2),W    ; ** est le signe de l'exposant
Move    W,PortB
```

Le XOR a deux autres applications. Pour inverser les bits d'une variable, on a l'instruction NOT, mais il n'y a pas d'instruction NOT W. On écrit

```
Xor     #-1,W    ; équivalent à    Xor     #2'11111111,W
```

La comparaison d'égalité se fait de préférence avec un XOR: si les deux opérandes sont identiques, le ou exclusif donne des zéros partout. Par exemple, pour savoir si une variable est égale à la valeur Max, on écrit:

```
Move    #Max,W
Xor     Var,W
Skip,NE
Jump    Egal
.. on continue ici si différent
```

4.3. Incrémentation/décrémentation

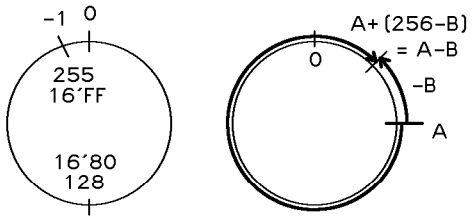
Les instructions Inc et Dec n'agissent que sur des variables. Inc W est remplaçé par Add #1,W et Dec W par Add #-1,W. On verra qu'il ne faut pas écrire Sub W,#1,W, et encore moins Sub #1,W, qui était acceptée par les assembleurs CALM du siècle dernier.

Le passage par zéro est détacté avec un Skip,EQ ou Skip,NE. Les instructions DecSkip,EQ et IncSkip,EQ n´existent pas avec la condition NE.

4.4. Nombres négatifs

On a utilisé -1 comme raccourci pour 2´11111111 = 16´FF. Le processeur ne connaît pas les nombres négatifs et positifs. Il applique toujours la même rêgle sur les nombres binaires qu´on lui donne. C´est nous qui donnons à 2´11111111 la valeur 255 ou la valeur -1, selon l´application. L´assembleur fait un peu le pont entre nos habitudes et les contraintes du processeur.

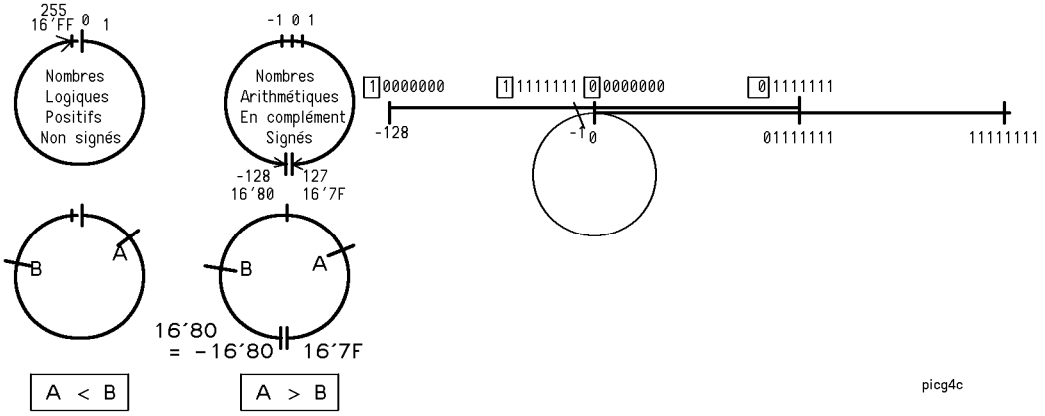
Les nombres 8 bits sont assez naturellement représentés sur un cercle, et l´on voit bien que -1 est équivalent à 255. Soustraire N revient à ajouter 256-N, identique à -N puisque 256 est égal à zéro, modulo la capacité de nos registres 8 bits.



picg4b

Fig. 1 Soustraction et addition du complément

Dans cette représentation, les nombres positifs vont de 0 à 127 (=16´7F). Len nombres négatifs de 16´80 à 16´FF, ils sont donc caractérisés par le bit de poids fort (le bit de signe) égal à 1. Il faut donc bien savoir si l´application travaille avec des nombres positifs, ou des nombres signés (positifs et négatifs selon le bit de signe). Pour les dépassements de capacité que nous ne voulons pas analyser ici, cela change tout. L´assembleur accepte les nombres négatifs jusqu´à -256, ce qui est une anomalie; il devrait s´arrêter à -128 comme dans la figure 2



picg4c

Fig. 2 Cercles des nombre et comparaison

Pour plus de détails, consultez www.didel.com/picg/doc/DocArith.pdf

4.5. Addition et soustraction

L´addition se fait entre le registre W et une variable ou valeur immédiate

```
Add     #Val,W   ; Ajoute à W la valeur Val
Add     Var,W    ; Ajoute à W la variable Var, résultat dans W
Add     W,Var    ; Ajoute à W la variable Var, résultat dans Var
```

Le Carry est mis à un s´il y a dépassement de capacité (le résultat dépasse 256=16´FF).

La soustration est très inhabituelle pour ceux qui connaissent d´autre processeurs; elle additionne le complément et permute les opérandes.

```
Sub     W,#Val,W; #Val - W --> W
   ; Prend la valeur Val, soustrait W, résultat dans W
Sub     W,Var,W ; Var - W --> W
   ; Prend le contenu de la variable Var, soustrait W, résultat dans W
Sub     W,Var   ; Var - W --> Var
```

```
; Prend le contenu de la variable Var, soustrait W, résultat dans  Var
```

Il faut comprendre que la soustraction (comme le And, Or, Add) est une opération à trois opérandes, mais que le 2e opérande est usuellement identifié avec la destination. Tous les processurs, avec la notation CALM et Motorola, exécutent et notent SUB B,A pour A-B --> A. Des processeurs RISC permettent d'avoir la destination dans un troisième registre, donc on écrirait SUB B,A,C pour A-B --> C. Le PIC a trois instruction qui exécutent #Val - W --> W , Var - W --> W et Var - W --> Var.

De plus la soustraction ajoute le complément, la valeur du Carry en est inversée

```
    20      est calculé  20
- 10                  +256
                       - 10
= 10                  = 10   Carry set
```

Avec le PIC, on préfère souvent ajouter une valeur immédiate négative plutôt que d'utiliser l'instruction Sub.

```
Sub     W,#34,Wremplacé par    Add     #-34,W
```

La valeur finale du Carry est la même dans les deux cas.

4.6. Dépassement de capacité

Dans une addition, le résultat peut dépasser 256. Le bit Carry est alors activé. Pour comparer deux nombres, on peut additionner le complément. A - B = A + 256-B. Si A est supérieur ou égal à B (A=B+C avec C ⩾ 0), on a A - B = B+C + 256-B = C + 256; donc le Carry =1. Si A est inférieur à B, le carry vaut zéro. Donc, pour comparer une valeur et une variable, on écrit:

```
; Var inférieur à Valeur?
Move    #-Valeur,W
Add     W,Var
Skip,CC
Jump    VarLowerSameValeur
; Suite si Var est plus grand que la Valeur
```

4.7. Comparaisons

La comparaison est source d'erreur, car il faut bien savoir dans quel ordre sont les opérandes comparés, et tenir compte des anomalies du PIC. Les notations CALM et Motorola utilisent pour les nombres positifs de 0 à 255 les symboles HI pour Higher (>), HS pour Higher-Same (⩾), LS pour Lower-Same (⩽) et LO pour Lower(<). L'assembleur acepte aussi ces abbréviations dans les pseudo-instructions conditionelles (.HI. etc, chapitre 9). Il est bon de s'en souvenir, même si le PIC n'a pas d'instructions directes pour nous faciliter ces comparaisons, à part le EQ et NE déjà vus. A noter que les comparaisons sont différentes pour les nombres négatifs, que nous ne considérons pas ici.

Les groupes d'instruction suivantes sont utilisées pour les comparaisons:

```
; SkipHI Comp #Val,W  skip if (W) higher than Val (modify W)
        Sub     W,#Val,W
        Skip,CC
        Jump    TaskifLS
        ... continue if HI

; SkipHS (CC or EQ) Comp #Val,W and skip if higher or same
        Sub     W,#Val,W
        Skip,CS
        Jump    APC+3
        Skip,EQ
        Jump    TaskifLO
        ... continue if HS

; SkipLO (CS or NE) Comp #Val,W and skip if lower
        Sub     W,#Val,W
        Skip,EQ
        Skip,CS
        Jump    TaskifHS
        ... continue if LO

; SkipLS Comp #Val,W and skip if lower or same
        Sub     W,#Val,W
        Skip,CS
```

```
Jump     TaskifHI
... continue if LS
```

Dans les instructions du SkipHS, Jump APC+3 signifie que l'on saute en 2 positions plus loin (APC est le compteur d'adresse de l'assembleur, que l'on peut initialiser par un .Loc).

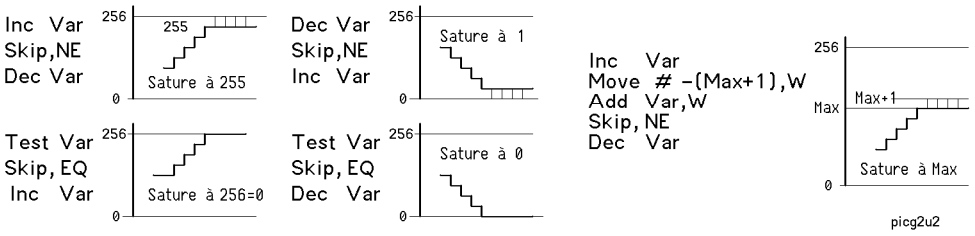
4.8. Saturation

Le concept d'opérations avec saturation est très important. Une variable doit souvent rester entre deux valeurs, une vitesse n'a pas de sens si elle est supérieure à 7 par exemple, une durée ne peut pas devenir négative, etc. Pour saturer, on compare, et mets la valeur maximum

```
Sature à #Min
Move     #Min,W
Sub      W,Var
Skip,CS
Clr      Var
Add      W,Var
```

```
Sature à #Max
Move     #Max,W
Sub      W,Var
Skip,CC
Clr      Var
Add      W,Var
```

Après une incrémentatio ou décrémentation, la figure 3 montre quelques cas d'utilisation astucieuse d'instructions du PIC.



picg2u2

Fig. 3 Exemples d'opérations avec saturation

4.9. Accès indirect en mémoire

Le registre FSR (adresse 4) permet de pointer une variable, donc de gérer des tableaux en mémoire. Le pointeur peut être déplacé avec des Inc, Dec, Add, etc., puisque FSR est un registre comme les autres. De même pour le registre pointé. On accède le contenu du registre pointé par FSR avec les instructions

```
Move     W,{FSR}
Move     {FSR},W     (modifie Z)
```

Ainsi donc, il est équivalent d'écrire

```
Move     Variable,W
```

ou -->

```
Move     #Variable,W
Move     W,FSR
Move     {FSR},W
```

Le registre FSR et un registre 8 bits, et permet de pointer dans les banques 0 et 1 des PICs. Il ne peut naturellement pas pointer dans la mémoire programme. Pour le 16F87x, il y a moyen de lire et écrire cette mémoire programme. Les registres AddrH et AddrL sont le pointeur en mémoire programme, et permettent une action sur des tables plus flexibles que la solution de la prochaine section (voir www.didel.com/picg/picg87x/doc/DocEe870)..pdf

4.10. Table de conversion

Les PICs ont une architecture de Harvard, avec une mémoire programme séparée de la mémoire des variables. Le 16F84/16F870 a des instructions de 14 bits, il n'y a pas de modes d'adressage pour accéder la mémoire programme. C'est pourtant essentiel de pouvoir mettre une table de constantes dans la mémoire programme. L'accès à une table de valeurs 8 bits en mémoire programme se fait astucieusement en alignant en mémoire des instructions "RetMove #n,W". Cette instruction est un Ret (retour de routine, revient à l'instruction suivant le Call). La routine appelée commence par un saut calculé par l'instruction "Add W,PCL", qui ajoute W aux 8 bits de poids faibles du compteur d'adresse. Le compteur d'adresse PC a ses 8 bits de poids faibles dans le registre PCL et 2 bits de sélection de page dans PCLatH que l'on ne peut ignorer que pour les petits programmes en page zéro. La table ne doit pas traverser une frontière de page de 256 positions (si on dépasse l'accès se fait au début du bloc de 256).

Par exemple, pour sauter à différentes parties de programmes selon une valeur 0, 1, 2, 3 dans un registre Data, on peut écrire simplement:

```
Move    Data,W
; éventuellement masquer ou saturer si on n´est pas sûr que Data<4
Add     W,PCL
Jump    Do0
Jump    Do1
Jump    Do2
Jump    Do3
```

L´instruction RetMove #n,W, de même que le Add W,PCL, est très astucieuse et performante. Par exemple, pour jouer des notes, une table est préparée avec les valeurs des durées des périodes correspondant aux notes de la gamme, mais contrairement aux processeurs comme le HC11 qui ont un registre (IX) qui peut pointer en mémoire programme, une table doit être faite avec des RetMove. L´instruction Add W,PCL fait sauter sur le W-ième élément de la table, qui est l´instruction Ret avec chargement de la valeur qui nous intéresse.

Le programme suivant utilise une table pour lire la durée d´une demi-période à partir du numéro logique de la note (do=0, ré=1, etc). La gamme est logarithmique et a 6 tons ou 12 demi-tons. Il y a un demi-ton seulement entre le mi et le fa, et entre le si et le do. A noter que la durée des notes n´est pas la même pour toutes les notes: les aiguës sont plus courtes que les basses, car on exécute un nombre de périodes (DurNotes) qui est toujours le même. Le nombre de périodes correct devrait être défini dans une autre table.

```
Program PicgT5  Gamme
.Proc   16F870
.Ref    16F870
.Loc    DebVar

Variables Variables
C1:     .16     1       ; Décompteur fixant la duré
C2:     .16     1       ; Décompteur durée de la n
Note:   .16     1       ; Note 0 à 8

Constant PortC
DirC    = 0     ; Tout en sortie

Constant Fixe la durée des notes
DurNote = 250   ;

Program Début
.Loc    0
Debut:  Move    #DirC,W
        Move    W,TrisC
Gamme:
        Clr     Note
Not$:
        Move    #DurNote,W
        Move    W,C2
 N$:
        Move    Note,W
  ; W contient le numéro logique, valeur max=8
        Call    ConvNote
  ; W contient maintenant la période trouvée dans la table
; On joue la note pendant 0,5 sec (selon DurNote)
        Not     PortC
        Move    W,C1
; On joue une demi-période
  A$:   DecSkip,EQ  C1
        Jump    A$
        DecSkip,EQ  C2
        Jump    N$
        Inc     Note
        Move    Note,W
        Sub     W,#NbNotes,W
        Jump,NE Not$
; Un silence pour marquer la fin, C1 et C2 sont nuls
 S$:    DecSkip,EQ  C1
        Jump    S$
        DecSkip,EQ  C2
        Jump    S$
        Jump    Gamme

ConvNote:
        Add     W,PCL   ; On ajoute aux poids faible
        RetMove #250,W  ; do
        RetMove #223,W  ; ré  (=250 * (2^-1/6))
        RetMove #198,W  ; mi
        RetMove #187,W  ; fa
        RetMove #166,W  ; sol
        RetMove #148,W  ; la
        RetMove #132,W  ; si
        RetMove #125,W  ; do
NbNotes = 8
.End
```

L´assembleur permet de définir des macroinstructions qui allègent la notation des tables. Si on déclare

```
.Macro  dd
        RetMove #%1,W
.Endmacro
```

l´assembleur, quand il voit "dd 250", crée les instructions de la macro et remplace %1 par le premier paramètre, ici 250. L´instruction "RetMove #250,W" est donc générée.

Comme exercice, on peut écrire le programme qui joue une mélodie. Une nouvelle table contient la mélodie:

```
GetNote:  ; Lit la note à jouer
          Move      PtNote,W
 Tpa:     Add       W,PCL
          dd        do
          dd        re
          dd        do
          dd        sol
          ...
```

On a naturellement déclaré do = 0, re = 1, etc. et défini une variable PtNote (pointeur à la note jouée). Après avoir joué la note, on incrémente le pointeur, décide si le morceau est fini (le plus simple est un -1 comme dernière "note"). On s'arrête (instruction "Fin: Jump Fin") ou on recommence après un silence. Pour faire mieux, il faudrait une table pour les durées associées à chaque note (notre solution simple joue plus vite les notes aiguës que les basses), définir des noires, blanches, silences, etc..

Quand on entre dans une table avec un paramètre qui peut déborder la longueur de la table, il y a deux façons d'agir après comparaison:
1) On signale une erreur et on arrête l'exécution
2) On plafonne le paramètre pour rendre la dernière valeur de la table.

Ce 2e cas, saturation à une valeur maximale, se programme comme suit:

```
Table:
          Add       #-Max,W
          Skip,CC             ; Skip si W > Max
          Clr       W
          Add       #Max,W
 Tpa:     Add       W,PCL
          ... suite
```

4.11. Vérification de dépassement

Les pseudos de l'assembleur sont très utiles pour vérifier qu'une table ne passe pas une frontiére de page. Si Tpa: est une étiquette mise en face de Add W,PCL, on mets à la fin de la table les 3 lignes suivantes:

```
          .If       (APC/256) .NE. (Tpa/256)
 Aie, ce module traverse la page
          .Endif
```

APC/256 est la page en fin de table. Tpa/256 la page quand on prépare la table. S'il n'y a pas égalité, la ligne suivante est interprétée, et comme il n'y a pas de ; le message d'erreur apparaîtra. Il faut alors déplacer la table ou des routines jusqu'à ce que l'assemblage soit correct.

4.12. Tables dans des grands programmes

Si on apelle une table dans une page qui n'est pas la page zéro, le processeur, au moment où il calcule le Add W,PCL, prends dans PClatH les adresses de poids fort. Elles sont initialisées à zéro quand on fait un reset, et c'est pour cela que l'on n'a pas de problèmes en page zéro. Mais si on va chercher une table en page 1, puis en page zéro, il faudra chaque fois réinitialiser PClatH. Voir \wepdfd;doc/DocPage.pdf pour plus de détails.

Pour initialiser PClath, on utilise les deux instructions

```
          Move      #APC/256,W
          Move      W,PCLatH
```

Ces instructions modifient W, on ne peut donc pas les mettre au début de la routine Table, si le paramètre est dans W, à moins de sauver temporairement W:

```
          Table:    Move      W,SavW
                    Move      #APC/256,W
 Tpa:               Move      W,PCLatH
                    Move      SavW,W
                    Add       W,PCL
          ...
```

Etant donné que le Call n'utilise pas PClatH avec le 16F84/16F870, on peut aussi l'initialiser avant de préparer W pour l'appel de la table, mais .

4.13. Musique

Générer une fréquence de 10 kHz est facile, puisque la période est de 100 microsecondes. Mais si l´on veut une note un quart de ton avant, la base de temps est de 102,6 μs. Une oreille de musicien sera sensible à l´arrondi vers 102 ou 103. Un autre problème est que la sortie du PIC générera un signal rectangulaire, et non pas une sinusoïde ou des harmoniques plaisant à l´oreille. Le PIC n´est pas un DSP; il ne faut pas trop lui demander, mais chercher à faire le plus possible en utilisant au mieux les instructions et l´architecture à disposition.

Le LA normal étant à 440 Hz, on calcule les périodes suivantes pour les notes de la gamme (les 12 demi-tons sont sur une échelle logarithmique avec comme rapport entre chaque demi-ton la racine 12eme de 2). Il y a un demi-ton seulement entre Mi-Fa, et Si-Do.

	Note	Facteur	Octave 2 Fréq	Période	Octave 3 Fréq	Période
0	Do	1	136	500	272	250
1	Ré	1.122	152	444	305	222
2	Mi	1.259	171	398	342	199
3	Fa	1.335	181	374	363	187
4	Sol	1.498	203	334	407	167
5	La	1.682	220	298	440	149
6	Si	1.888	256	264	513	132
7	(Do)	2	272	250	544	125

Jouer des notes, et encore plus des morceaux avec accords, en ayant encore d´autres tâches qui s´exécutent simultanément, est difficile et plein de limitations. Un premier problème est de coder les notes et leur durée, en tenant compte que le nombre de périodes à jouer est inversément proportionel à la période. Mesurer le temps avec un compteur de temps (et une autre base de temps) est possible, mais ralentit et il faut éviter de couper une note. La transition d´une note à l´autre ne doit pas entraîner de hiatus perceptible.

Pour jouer une mélodie, il faut définir une table de périodes de notes, une table de durées correspondantes (cela pourrait être la même table parcourue en sens inverse) et une table pour les notes du morçeau de musique. Le programme PicgM1 joue une fois la gamme. Il n´y a pas de code prévu pour un silence. On pourrait imaginer que la note "Silence" est comparée avant de jouer la note et appelle une boucle d´attente silencieuse.

Program **PicgT5b.asm** Joue la gamme

```
.Proc     16F870
.Ref      16F870
```

Constant **PortC**

```
bHp       = 1       ; Haut-parleur sur RC1
DirC      = 2´11111101
```

Variables **Variables**

```
.Loc      DebVar
C1:       .16       1          ; Décompteur fixant la base
C2:       .16       1          ; Décompteur longueur morc
PtNotes:  .16       1
Note:     .16       1          ; Note 0 à 8
PerNote:  .16       1
DurNote:  .16       1
SavePortC:  .16     1
```

Constant **Fixe la durée des notes** (varie selon période)

Program **Début**

```
.Loc      0
Debut:    Move      #DirC,W    ; Tout en sortie
          Move      W,TrisC
          Clr       PtNotes
          Clr       SavePortC
          Move      #LMorceau,W
          Move      W,C2
PlayNote:
          Move      PtNotes,W
          Call      TaMorceau
          Move      W,Note     ; 0 à 8
          Call      TaDurees
          Move      W,DurNote
          Move      Note,W
          Call      TaNotes    ; Période
          Move      W,PerNote
; On joue la note pendant 0,5 sec (en répétant DurNote f
RepNote:
          Move      SavePortC,W
          Xor       #2**bHp,W
          Move      W,SavePortC
          Move      W,PortC
          Move      PerNote,W
          Move      W,C1
; On joue une demi-période
 DP$:     Call      BaseTemps
          DecSkip,EQ  C1
          Jump      DP$
          DecSkip,EQ  DurNote
          Jump      RepNote
; On passe à la note suivante
          Inc       PtNotes
          DecSkip,EQ  C2
          Jump      PlayNote
          Jump      APC
  ; On s´arrête ici (JUMP Debut recommencerait)
```

Routine **BaseTemps** Attente environ 2ms (LA) / 148 période = 13 microsecondes

```
; Si oscillateur 4MHz, 4 µs Call/Ret 2 µs préparation  rs
BaseTemps:
;         Nop
;         Nop
          Nop
          Nop
          Nop
          Ret
.macro    dd        ; data pour tables
          RetMove   #%1,W
.endmacro
```

Table **TaDurees** Durées des notes

```
TaDurees:
          Add       W,PCL
          dd        125
          dd        132
          dd        148
          dd        166
          dd        187
          dd        198
          dd        223
          dd        250
```

Table **TaNotes** Périodes des notes

```
TaNotes:
          Add       W,PCL      ; On ajoute aux poids faible
          dd        250        ; do
          dd        223        ; ré  (=250 * (2^-1/6))
          dd        198        ; mi
          dd        187        ; fa
          dd        166        ; sol
          dd        148        ; la
          dd        132        ; si
          dd        125        ; do2
```

Constant **Valeur des notes**

```
do        = 0
re        = 1
mi        = 2
fa        = 3
sol       = 4
la        = 5
si        = 6
do2       = 7
```

Table **TaMorceau** Le morceau de musique choisi

```
TaMorceau:
          Add       W,PCL
          dd        do
          dd        re
          dd        mi
          dd        fa
          dd        sol
          dd        la
          dd        si
          dd        do2
LMorceau =  APC-TaMorceau-1
.Align    8
.16       "P","i","c","g","t","5","b"
.Fill.16  FinProg+1-APC,-1
.Loc      16´2007
.16       16´3F39 ; Config
.End
```

JDN 30 novembre 1903