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

0.95

5. Bits, Moteurs et délais

5.1. Action sur des bits

Nous avons jusqu'à présent manipulé les 8 bits du port B simultanément. Pour le port A, ce serait la même chose, mais seuls les 5 bits de poids faible sont disponibles, avec un 5e bit (RA4) un peu spécial puisqu'il est en collecteur ouvert (dans **PIC*Génial*** la résistance sur le buzzer garantit l'état 1 si le port est utilisé en entrée). Le PIC est très performant pour agir directement sur l'un des bits d'un port ou d'une variable en mémoire (mais pas sur les registres de direction), sans modifier les autres bits si leur niveau logique est correct. Les bits sont numérotés de 0 (poids faible) à 7 (poids fort)

```
Clr       Port:#BitNumber   ; Met le bit à zéro
Set       Port:#BitNumber   ; Met le bit à un
```

On peut de même lire un bit d'un port initialisé en entrée, sur lequel on a câblé un interrupteur par exemple, ou un bit dans une variable. L'instruction teste le bit (bit à 1 = BS = bit set, bit à zéro = BC = bit clear) et saute l'instruction suivante si le bit est à un ou à zéro.

```
TestSkip,BC  Port:#bNumber ; Skip si le bit testé est à zéro
TestSkip,BS  Port:#bNumber ; Skip si le bit testé est à un
```

Par exemple, si on veut copier l'état du poussoir du module d'expérimentation (câblé sur RA4) sur la diode de poids fort RB7, on initialisera au moins RA4 en entrée et RB7 en sortie avant de tester et activer les bons bits. Il faut naturellement définir clairement ce que l'on veut: Si le poussoir est pressé, la diode doit être allumée. L'interrupteur pressé veut dire que le circuit est fermé, donc on a l'état zéro sur l'entrée RA4. Diode allumée veut dire qu'il faut mettre un zéro sur la sortie RB7. Le programme s'écrit donc:

Program **PicgT3** Lit et copie un interrupteur

```
.proc 16F84
```

Variables **Néant**

Constant **Port A**

```
bInter = 4        ; Interrupteur en RA4
DirA   = 2'11000            ; RA4 RA3 en entrée
```

Constant **Port B**

```
bLed   = 0        ; Led en RB0
DirB   = 2'00000000         ; Tout en sortie
```

Program **Début du programme**

```
Debut:Move        #DirA,W
      Move        W,TrisA
      Move        #DirB,W
      Move        W,TrisB
Boucle:
      TestSkip,BS  PortA:#bInter  ; Si 0 (bit clear), on
      Clr         PortB:#bLed
      TestSkip,BC  PortA:#bInter  ; dans l'autre cas, on
      Set         PortB:#bLed
;...
      Jump        Boucle
.End
```

Si on voulait afficher un motif spécial sur les 8 diodes du port B quand le commutateur est en haut, on pourrait commencer par écrire un état 1 (lampe éteinte) sur le port, et écrire le motif conditionnellement. Evidemment, lorsque l'interrupteur est à un, chaque fois qu'on passe dans la boucle, on éteint les diodes pour 3 microsecondes. Si l'on ne fait rien d'autre dans la boucle comme ci-dessous, on se trouve avec 50% d'intensité seulement.

```
  Motif   = 2'10101100      ; 0 = allumé
Boucle:
        Move      #2'11111111,W
        Move      W,PortB
        Move      #Motif,W
        TestSkip,BS  PortA:#bInter ; Si 0 (bit clear), on allume,
        Move      W,PortB          ; Exécuté seulement si BC (bit clear)
;...
        Jump      Boucle
```

Si l'on veut compter les actions du poussoir, on attend dans une boucle que la sortie passe à un, puis dans une autre boucle qu'elle passe à zéro. On appelle alors la routine qui, par exemple, incrémente et affiche. Mais attention, il y a des rebonds pendant quelques millisecondes. Une boucle d'attente de 3ms au moins doit être insérée dans les boucles qui testent l'état du poussoir.

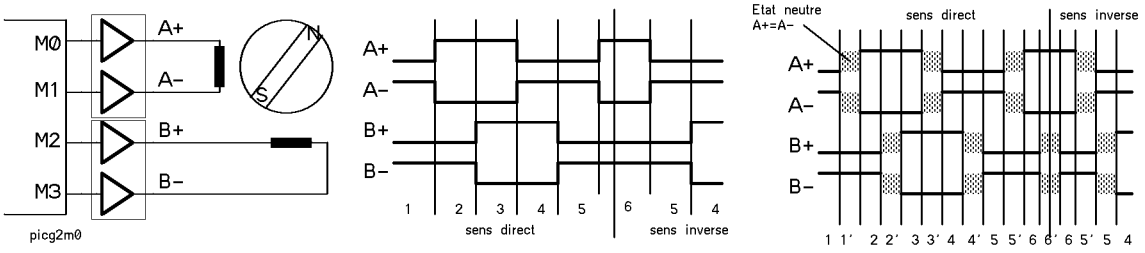
5.2. ET logique entre bits

Une condition d'exécution peut dépendre de deux bits dans des ports ou variables différents. Le AND est avantageux car il n'utilise que deux instructions.

```
; SkipAND   A,B     Skip next instruction if A.AND.B = 1
          TestSkip,BS   Reg1:#Bit1
          TestSkip,BC   Reg2:#Bit2
          (Jump  Nand)

; SkipOR    A,B     Skip next instruction if A.OR.B = 1
          TestSkip,BC   Reg1:#Bit1
          Jump      APC+2
          TestSkip,BS   Reg2:#Bit2
          (Jump     Nor)

; SkipXOR   A,B     Skip next instruction if A.XOR.B = 1
          TestSkip,BC   Reg1:#Bit1
          TestSkip,BC   Reg2:#Bit2
          Jump      APC+2
          Jump      APC+4
          TestSkip,BS   Reg1:#Bit1
          TestSkip,BS   Reg2:#Bit2
          (Jump     XNor)
```

5.3. Moteur pas-à-pas

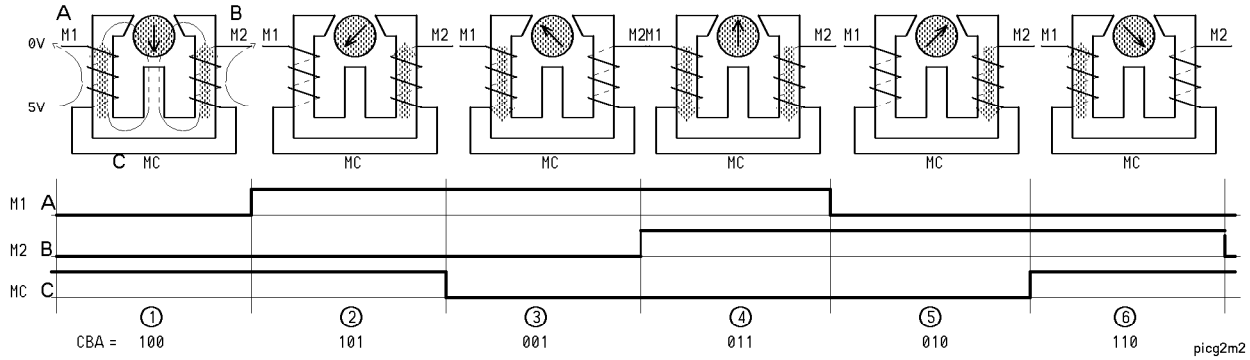
Les moteurs pas-à-pas ont 4 phases avec 2 bobines, ou 6 phases avec 2 bobines. Rappelons ici seulement le principe des moteurs à 4 phases, commandés en pas ou demi-pas. Par rapport aux moteurs à 6 phases, il suffit de changer les tables. Ces moteurs sont en général commandés par une tension de 12 Volts ou plus. Ils nécessitent donc une interface de commande appropriée.



picg2m0

Fig. 1 Moteur 4 phases en pas et demi-pas

Une documentation détaillée sous www.didel.com/picg/dopiswi.pdf décrit différentes techniques de commande du moteur switec, qui est un moteur à 6 phases, comme le montre le diagramme ci-dessous. Les deux bobines recoivent les signaux A-C et B-C.



picg2m2

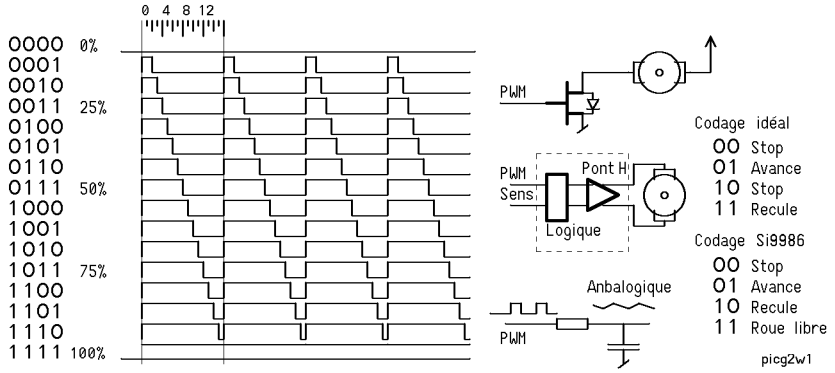
Fig. 2 Séquences des moteurs switec et smoovy

Les moteurs synchrones comme les smoovy, répondent à la même séquence, mais ils utilisent 3 bobines au lieu de deux, et les bobines sont excitées par A-$\overline{A}$ B-$\overline{B}$ et C-$\overline{C}$. Le programme PicgT6.asm dans le répertoire Picgenial est un exemple de commande du moteur switec. Une documentation plus complète est disponible sous www.didel.com/picg/DopiSwi.pdf.

5.4. PWM

Un signal en PWM (Pulse Width Modulation) donne des impulsions variables entre 0 et 100%, avec un certain nombre d´états intermédiaires selon le codage. Un problème est que, si le codage est 8 bits par exemple, 0 doit correspondre à 0 % (pas de signal) et 255 = 16´FF correspond à 100% (signal continu). Avec un compteur simple, 255 génère facilement 255/256, ce qui n´est pas un signal continu. Dans le codage, il faut décider quel code transformer (0 peut signifier 100%, 1 signifie 0%, 2 signifie 2/256, .. 255 signifie 255/256. Il est en général inutile de prévoir 8 bits pour le codage du PWM. 4 ou 6 bits sont largement suffisants dans la plupart des cas, et permettent d´augmenter la fréquence. Avec 4 bits, le codage 0000 = 0%, 0001 = 2/16 .. 1110 = 15/16 1111 = 16/16 = 100% élimine le pourcentage 1/16, qui correspond dans la pratique à un signal trop faible, ou à une excitation de moteur qui ne lui permet pas de démarrer.

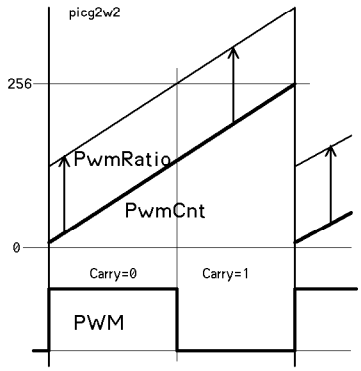
Un signal PWM est le plus souvent utilisé pour commander un moteur; l´inertie du moteur intègre les impulsions. Une fréquence de quelques centaines de Hz est suffisante, mais la carcasse du moteur vibre comme un haut-parleur. Une fréquence supérieure à 20 kHz évite que ce bruit soit audible. Un signal PWM est aussi utilisé pour générer un signal analogique par intégration dans un réseau RC.



picg2w1 *Fig. 3 Principe du PWM*

La commande en PWM d´un moteur continu se fait avec un transistor si le sens est toujours le même. Un pont en H permet une commande bidirectionelle. Le codage en entrée du pont peut nécessiter un transcodage avec quelques instructions.

La routine la plus efficace pour générer du PWM s´appuie sur l´astuce suivante: Un compteur de temps PwmCnt est incrémenté à un multiple de la fréquence PWM (par une boucle d´attente, ou un timer comme vu plus loin). La variable PwmRatio proportionnelle au PWM est ajoutée à ce compteur. S´il y a Carry, la sortie en PWM est activée, ou inverséement. Pour obtenir le 100%, une addition supplémentaire est nécessaire. Le listage suivant donne la routine PWM correspondante. Si AddPWM = 1, on a 256 pas. Si comme si-dessous AddPWM = 2**5, on a 8 pas codés 0 = 0%, 00100000 = 2/8, ... 111 = 100%.



picg2w2 *Fig. 4 Principe de la routine PWM*

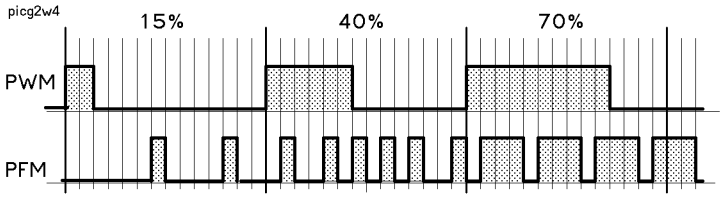
```
....
PWM 3 bits, 12 kHz at 100 µs Update

Variables Variables
PwmRatio: .Blk.16 1
PwmCnt:   .Blk.16 1
AddPWM    = 2´00100000 ; 8 values PwmRatio   xxx00

....
; Initialisation
Clr         PwmRatio
....
```

```
Module PWM Pulse Width calculation. Task repeated e.g. every 100 µs
PWM:Move        #AddPWM,W
    Add         W,PwmCnt
    Skip,NE
    Add         W,PwmCnt
    Move        PwmCnt,W
    Add         PwmRatio,W
    Skip,CC
    MotOn       ; Macro that sets the motor control bit
    Skip,CS
    MotOff
```

0.95

5.5. PWM et PFM

Le PFM (Pulse Frequency Modulation est peu connu, tout aussi facile à générer que le PWM, et génère des signaux de fréquence varaible plus élevée. Pour plus d´information on consultera la documentation sous www.didel.com/picg/DopiSwi.pdf.



picg2w4

Fig. 5 Différence entre PWM et PFM

1

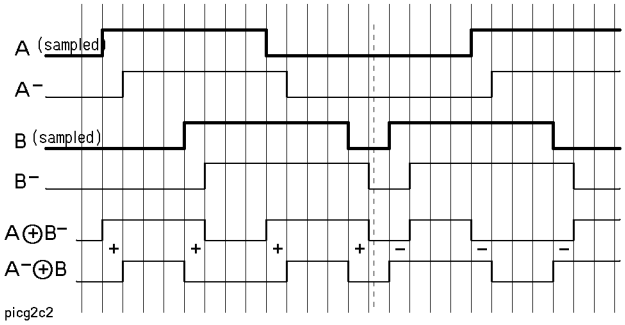
Certains PICs ont une ou deux paires d´entrées associées à des compteurs évitant le temps perdu par l´exécution d´une routine de détection de sens et de comptage des pas, et surtout la nécessité d´appeler cette routine sans cesse pour éviter de perdre des pas. Utiliser des interruptions n´apporte rien, car la routine d´interruption prends plus de temps que l´appel synchrone de la routine.

5.6. Encodeur

Le programme pour un encodeur de souris est donné ci-dessous. L´échantillonnage doit être suffisament fréquent pour ne pas perdre des pas. Le PIC peut avoir de la peine à suivre un encodeur monté directement sur l´axe d´un moteur continu.

0.92

```
Program Picgc2.asm Rotary sense detection
R.Sommer (Logitech Inc) algorithm on a PIC
```



picg2c2

```
; The set of instructions to be executed synchronously or
; (about every 1ms if less than 1kHz step frequency)
; lasts 18 µs  with 8-bit up/down counters (2 channel
.Proc 16F84
DebVar= 16´20
 PortA
; Bits 0 Y2 Y1 X2 X1
 PortB affichage
.Loc  DebVar
OldPort:        .Blk.16     1
Temp: .Blk.16   1
Cnt1: .Blk.16   1           ; Compteur 8 bit codeur
Cnt2: .Blk.16   1           ; Compteur 8 bit codeur
```

```
.Loc  0             ; Programme de test et routine
Begin:
      Move          #2´01111,W    ; inputs
      Move          W,TrisA
      Clr           W             ; PortB out (affichage)
      Move          W,TrisB
      Move          PortA,W
      Move          W,OldPort
      Clr           Cnt1
      Clr           Cnt2
Loop:; Affichage de test sur le PortB Cnt1 (ou Cnt2, un à
      Move          Cnt1,W
      Move          W,PortB
      Jump          Loop

Routine Enco Update Cnt1 and Cnt2 according to the encoders
Enco: RLC           OldPort,W
      And           #2´1010,W
      Move          W,Temp
      RRC           OldPort,W
      And           #2´0101,W
      Or            W,Temp
      Move          PortA,W
      And           #2´1111,W
      Move          W,OldPort
      Xor           W,Temp
      TestSkip,BC   Temp:#0
      Dec           Cnt1
      TestSkip,BC   Temp:#1
      Inc           Cnt1
      TestSkip,BC   Temp:#2
      Dec           Cnt2
      TestSkip,BC   Temp:#3
      Inc           Cnt2
      Ret
.End
```

5.6.1. Impulsions brèves

L'impulsions la plus brève que l'on peut génèrer avec un PIC à 4 MHz est de 1 µs.

```
Pulse:   OutbitOn
         OutbitOff
```

Des Nops peuvent être ajoutés entre ces deux instructions pour augmenter la durée de l'impulsion. Si dans une application on a besoin d'impulsions de 1 à 20 microsecondes, on a avantage à définir une macro permettant d'écrire par exemple *Pulse 15* pour une impulsion de 15 µs. Cette macro s'écrit:

Fichier Picgp2

```
.Macro Pulse
       OutbitOn
.If    %1.GT.20
.Error "Parameter too large"
.Endif
.If    %1.EQ.1
.Endif
.If    %1.EQ.2
       Nop
.Endif
.If    %1.EQ.3
       Nop
       Nop
.Endif
.If    %1.EQ.4
       Nop
       Nop
       Nop
.Endif
.If    %1.GT.5
       OutbitOn
       Call        Delay%1
.Endif
       OutbitOff
.EndMacro
```

```
Delay20:        Nop
Delay19:        Nop
Delay18:        Nop
Delay17:        Nop
Delay16:        Nop
Delay15:        Nop
Delay14:        Nop
Delay13:        Nop
Delay12:        Nop
Delay11:        Nop
Delay10:        Nop
Delay9:         Nop
Delay8:         Nop
Delay7:         Nop
Delay6:         Nop
Delay5:         Ret
```

Par exemple: Pulse 3 génère

```
OutbitOn
Nop
Nop
OutbitOff
```

On remarque que pour des délais supérieurs à 3 NOP, on a avantage à appeler une routine. Pour plus de 6 NOP, ou pourrait introduire une boucle, qui utiliserait une variable et obligerait de distinguer 3 cas. C'est ce qu'il faut faire si le retard est un paramètre dans une variable, copiée dans W. La durée de l'impulsion est alors de 5 µs au moins.

```
      Move       Delay,W
      OutbitOn
      Move       W,C1
L$:   DecSkip,EQ C1
      Jump       W
      OutbitOff
```

```
Delay = 1 --> 4 µs
Delay = 2 --> 7 µs
Delay = N --> 2 + (N-1)x3 + 2
Delay = 0  --> 769 µs
```

S'il faut des durées courtes, le passage par une table est une solution. Ci-dessous, la durée de l'impulsion codée dans W est de 6 à 10 µs, mais il n'y a pas de vérification de validité: cela rajouterait plusieurs instructions.

```
      OutBitOn
      Sub        #6,W
      Add        W,PCL
      Jump       F6$
      Jump       F7$
      Jump       F8$
      Jump       F9$
      Jump       F10$
F10$: Nop
F9$:  Nop
F8$:  Nop
F7$:  Nop
F6$:  OutbitOff
```

Ces exemples n'épuisent pas la variété de solutions que l'on peut imaginer pour générer des timings précis dans une application donnée.

5.7. Comptages et délais longs

Compter des évènements ou compter un délai élémentaire pour obtenir un délai plus long revient au même. Une variable compteur permet de compter par 256. Deux variables permettent de travailler en 16 bits. L'exemple ci-dessous concerne un compteur 24 bits traité en routine, ou inséré dans le corps du programme. L'exécution dure 5 à 9 µs pour la routine, et 3 à 5 µs dans l'autre cas.

```
    ; Routine
IncCnt24:       IncSkip,EQ   CntLow
      Ret
       IncSkip,EQ   CntMiddle
      Ret
       Inc         CntHigh
      Ret
```

```
    ; Instructions ou macro
      IncSkip,EQ CntLow
      Jump       F$
      IncSkip,EQ CntMiddle
      Jump       F$
      Inc        CntHigh
F$:
```

S'il faut compter par 100 ou 1000, il faut savoir si le codage est important (on veut afficher l'heure par exemple) ou s'il suffit de diviser. Les deux cas de division par 100, en binaire et en BCD, sont donnés ci-dessous. Le comptage BCD est une routine. Si on veut appliquer cette routine à différentes variables décimales, il faut considérer CntBCD comme une variable locale, et transférer la variable globale dans CntBCD via le registre W avant d'appeler la routine.

```
; Incrémentation binaire
      Inc         Cnt100
      Move        #100,W
      Xor         Cnt100,W     ; Egalité?
      Skip,NE
      Clr         Cnt100
      ...suite
ou
      Inc         Cnt100
      Move        #100,W
      Xor         Cnt100,W
      Skip,EQ
      Jump        F$
      Clr         Cnt100
      .... Ce qu'il faut faire quand on arrive à 100
  F$:...suite
```

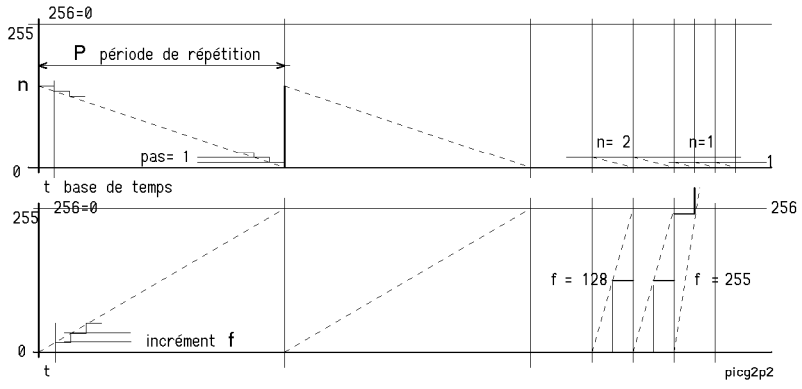
```
; Routine d'incréemtation BCD (0 à 99=16'10011001)
IncBCD:
      Inc         CntBCD
      TestSkip,BC  CntBCD:#1
      TestSkip,BS  CntBCD:#3
      Ret
      Move        #6,W
      Add         W,CntBCD
      TestSkip,BC  CntBCD:#4+1
      TestSkip,BS  CntBCD:#4+3
      Ret
      Move        #(2**4)*6,W  ; Another way of handlin
      Add         W,CntBCD
      Ret
```

5.8. Actions répétitives

Une action répétitive dépend d'un paramètre donnant la période, ou la fréquence. Les trois groupes d'applications sont la génération de sons (qu'un PIC à 4MHz a de la peine à bien faire), la commande d'un moteur pas-a-pas, et le clignotement de lampes. Si l'opération a un paramètre variable, cela limite les performances pour deux raisons: il est un peu plus long de manipuler un paramètre variable, et surtout il faut déterminer sa valeur dans un autre module du programme.

Une durée dépendant d'un paramètre est liée à une base de temps t définie
- par des instructions calibrées dans le cas de durées minimales
- par une boucle d'attente
- par le timer surveillé par programmation
- par une interruption du timer ou d'un signal extérieur

Pour obtenir une période de répétition P avec une précision 8 bits, on peut soit charger un décompteur C1 avec une valeur n proportionelle à la période, et décompter à chaque cycle, soit ajouter à la variable C1 une valeur f proportionnelle à la fréquence de répétition.



picg2p2 *Fig. 6 Délai basé sur une période ou une fréquence.*

Les modules de programme correspondants s'écrivent

```
; Initialisation
      Move        #n,W
      Move        W,C1
...
; Boucle principale, période fixe n
 L$:  .... Attente selon la base de temps t
      DecSkip,EQ  C1
      Jump        L$
      Move        #n,W
      Move        W,C1
      .... Action répétée avec la période P
      Jump        L$
```

```
; Initialisation
      Clr         C1
...
; Boucle principale, fréquence fixe f
 L$:  .... Attente selon la base de temps t
      Move        #f,W
      Add         W,C1
      Skip,CS
      Jump        L$
      Clr         C1
      .... Action répétée avec la période P
      Jump        L$
```

Ces deux approches ont une précision qui dépend de la valeur du paramètre. Avec la période, la précision relative est mauvaise si n est petit. Des valeurs différentes de n donnent des durées différentes. Avec la fréquence, la période est peu précise si f est grand. Le nombre de périodes possibles est plus faible, car à partir de f=16, des valeurs successives peuvent donner la même période. Par exemple 16 et 17, 20 et 21, ..., 29 à 32, ..., 128 à 255. Pour F=128, le Carry se produit à la 2e addition: 128+128=256=Carry+0. Avec 155 aussi: 255+255=Carry+254. En supprimant l'instruction Clr C1, les durées deviennent plus précises, mais en moyenne seulement (par exemple, si f=17, on a une période valant succesivement 15 et 16 fois t, donc 15,5 en moyenne).